UNIVERZITET U NIŠU
PRIRODNO-MATEMATIČKI FAKULTET

Selver H. Pepić

MATRIČNA IZRAČUNAVANJA U PHP/MySQL OKRUŽENJU

- doktorska disertacija -

Niš, maj 2012. god.

UNIVERZITET U NIŠU
PRIRODNO-MATEMATIČKI FAKULTET

Selver H. Pepić

MATRIČNA IZRAČUNAVANJA U PHP/MySQL OKRUŽENJU

- doktorska disertacija -

Mentor:
Prof. dr Milan B. Tasić

Niš, maj 2012. god.

Velika mi je čast i posebno zadovoljstvo da se zahvalim mentoru prof. dr Milanu Tasiću, vanrednom profesoru PMF-a u Nišu, na nesebičnom zalaganju, podršci, brojnim korisnim savetima i primedbama koji su vešto usmeravali moj naučni rad i uticali na kvalitet ove disertacije.

Veliku zahvalnost dugujem svim članovima Komisije, prof. dr Predragu Stanimiroviću, redovnom profesoru PMF-a u Nišu, prof. dr Miroslavu Ćiriću, redovnom profesoru PMF-a u Nišu, dr Predragu Krtolici, docentu PMF-a u Nišu i prof. dr Aleksandru Cvetkoviću, vanrednom profesoru Mašinskog fakulteta u Beogradu, koji su svojim korisnim sugestijama i stručnim savetima doprineli izradi ove disertacije.

Najtoplije se zahvaljujem svojoj porodici na strpljenju, razumevanju, podršci i ohrabrenjima koja su me podsticala da istrajem u radu.

Sadržaj

Predgovor

Doktorska disertacija pripada oblasti web programiranja i baza podataka, odnosno podoblasti koja se bavi upravljanjem, skladištenjem i pretraživanjem sistema skladištenja na Internetu. Po *MSC2010* klasifikuje se u *68P15 Database theory, 68N15 Programming languages, 68P20 Information storage and retrieval kao i 15A09 Matrix inversion, generalized inverses. CPU* vreme izvršavanja je veliki problem koji se javlja u matričnoj algebri i algoritmima sa rekurzijom kao i problemima karakterističnim za tehnike dinamičkog programiranja. Osnovni cilj disertacije je skraćenje *CPU* vremena pri matričnim izračunavanjima, implementacijom klijent/server modela zasnovanog na *MatrixDBMS* sistemu skladištenja. *MatrixDBMS* model omogućava skladištenje različitih tipova matrica i operacija nad njima, koje je, umesto dugotrajnog ponovnog izračunavanja, potrebno samo pronaći u bazi i pročitati. Efekat ovog modela je pokazan kroz primenu na triangulaciju konveksnih poligona i obradu slike. Klijent/server aplikacija koja je bazirana na *MatrixDBMS* sistemu je implementirana u *PHP/MySQL* okruženju, otvorenog je koda, zadovoljava stroge uslove troslojne web arhitekture i prilagođena je radu na različitim platformama i pod različitim operativnim sistemima.

Rezultati istraživanja se mogu svrstati u tri celine. Prvu celinu čine rezultati koji se odnose na matričnu algebru, osnovne matrične operacije kao i izračunavanje generalisanih inverza. U okviru ove celine je razmatran i odnos između generalisanog *LM* inverza i težinskog *MP* inverza. Drugu celinu čine rezultati koji predstavljaju primenu *MatrixDBMS* sistema skladištenja na triangulaciju konveksnih poligona. Predstavili smo nekoliko algoritama za izračunavanje triangulacije konveksnog poligona u korelaciji sa *MatrixDBMS* modelom. Treću celinu čine rezultati koji se odnose na primenu pomenutog modela na obradu slike.

Rad je predstavljen u šest glava koje su podeljene na poglavlja a ona opet na odeljke, tj. sekcije. U odeljcima su iznete definicije, teoreme, formule i algoritmi. Pored

toga u radu je data lista oznaka, spiskovi algoritama, tabela i slika kao i dodatak u kome su dati izvodi iz procedura i funkcija aplikacije.

Prva glava sadrži osvrt na razvoj ideja koje su dovele do realizacije ove doktorske disertacije. Iznete su osnovne informacije i dosad otkrivene činjenice na koje se baziramo u daljem radu. Predstavljeni su i problem, predmet, ciljevi, hipoteze i tehnike istraživanja.

U drugoj glavi koju čine pet poglavlja dati su osnovni pojmovi, definicije i teoreme vezane za matričnu algebru, osnovne matrične operacije i generalisane inverze konstantnih matrica (*Moore-Penroseov* inverz i težinski *MP* inverz). Dat je i najefikasniji način za predstavljanje matrica na aplikacionom sloju (*MatrixAL* model) kao i univerzalni sistem skladištenja matrica na sloju baze podataka (*MySQL MatrixDBMS*), troslojne web arhitekture. U ovoj glavi je data i teorema o ekvivalentnosti LM i Težinskog MP inverza.

U trećoj glavi su predstavljeni primeri primene MatrixDBMS modela na triangulaciju konveksnih poligona i obradu slike. Predstavljen je do sad nekorišćen pristup triangulacije konveksnih poligona u korelaciji sa *MySQL* sistemom skladištenja. Data je mogućnost uklanjanja duplikata, korišćenjem jedinstvenog ključa i servisa relacionog *DBMS*. Takođe su razmatrani načini skladištenja slike, u pogledu brzine transformacije i memorijskih kapaciteta koje zauzima.

Četvrta glava sadrži nove pristupe vezane za matrična izračnavanja. Opisana je realizacija univerzalnog *Cassandra NoSQL MatrixDBMS* sistema skladištenja. Testiranje *CPU* vremena komunikacije sa bazom podataka dato je kroz dva dijametralno suprotna pristupa, relacioni *MySQL DBMS* i *Cassandra NoSQL* sistem. Predstavljeni su i neobjavljeni rezultati vezani za paralelni pristup matričnih izračunavanja primenom na triangulaciju konveksnih poligona. Na kraju svake oblasti su dati rezultati testiranja.

Dizajn i implementacija klijent/server modela su dati u petoj glavi. Predstavljene su različite mogućnosti aplikacije, i to: kombinacija različitih matričnih operacija, izračunavanje generalisanih inverza, triangulacije konveksnih poligona i obrade slike. Opisani su različiti alati i tehnologije po nivoima, kao i komunikacija i transfer među slojevima troslojne web arhitekture. Poseban akcenat je dat na funkcionalnost *MatrixDBMS* i *MatrixAL* modela u sistemu troslojne web arhitekture.

Zaključna razmatranja kao i budući pravci razvoja klijent/server modela, baziranog na *MatrixDBMS* sistemu skladištenja, dati su u poslednjoj glavi, dok se kodovi nalaze u dodatku na kraju rada.

Disertacija je prvenstveno nastala na osnovu publikovanih naučnih radova u časopisima [27,45,46,58,59], ali sadrži i rezultate koji se prvi put pojavljuju i do sada nisu

objavljeni. Predloženi klijent/server model, sa *MatrixDBMS* sistemom skladištenja, je moguće primeniti u različitim situacijama slične problematike.

Svi implementacioni kodovi su otvoreni i mogu se preuzeti sa internet adrese:

http://rnserver.pmf.ni.ac.rs/matrixdbms

Glava 1

1. Uvod

Matrice je u matematiku uveo engleski matematičar A.Cauley (1821-1895) u svom radu iz 1857. god. Matrice predstavljaju sisteme brojeva koje možemo tretirati skoro kao i brojeve, pa one u izvesnom smislu uopštavaju brojeve. Probleme u primenjenoj nauci, matematici i tehnici, kad god je moguće, treba posmatrati kroz prizmu matrica, jer je za njih poznat dobro razvijen matematički aparat. Postoji veliki broj algoritama za programiranje igara koji koriste matrične operacije.

Postoji mnogo primena matrica kako u matematici tako i u drugim naukama. Matrice se pominju u teoriji igara, ekonomiji, *data mining*-u i *text mining*-u. U početnoj fazi razvoja enkripcije su korišćene matrice, ali su zbog linearne prirode matrica kriptovane šifre relativno lako "razbijane". U kompjuterskoj grafici koriste se matrice za predstavljanje objekata kao i za transformaciju i predstavljanje trodimenzionalnih objekata na dvodimenzionalnoj podlozi. U hemiji se matrice koriste u razne svrhe, posebno u oblasti kvantne teorije i spektroskopije. Srećemo ih i pri rešavanje *Roothaan*-ovih jednačina koje omogućavaju preklapanje orbite molekula u *Hartree–Fock*-ovoj metodi. Grevilov metod se koristi kao merilo za izračunavanje pseudo inverza. Zbog svoje dominacije ovaj metod se obimno primenjuje u mnogim matematičkim oblastima kao što su statističko zaključivanje, teorija filtriranja, teorija linearne procene, optimizacija, analitička dinamika itd. [23] i [61]. Primenjuje se i u direktnom pristupu za izračunavanje gradijenta pseudo – inverza predstavljenog u [29]. Takođe, ovaj metod je široku primenu našao i u bazama podataka kao i u izračunavanjima vezanim za neuronske mreže [21]. Sekvencijalno određivanje MP inverza dinamičkim programiranjem je primenjeno na dijagnostikovanje i klasifikaciju elektromiografskih signala.

Predmet ovog istraživanja će biti skraćenje *CPU* vremena izvršavanja matričnih izračunavanja analizom, modeliranjem i dizajnom objektno-orijentisanog klijent/server modela u *PHP/MySQL* okruženju sa *MatrixDBMS* sistemom skladištenja. *MatrixDBMS*

sistem skladištenja predstavlja univerzalni, fleksibilni i dinamički model prilagodljiv skladištenju različitih tipova matrica. Dinamičnost sistema se ogleda u mogućnosti nadogradnje u bilo kojoj fazi, čak i nakon realizacije. Glavni napor će, dakle, biti usmeren na uključivanje univerzalnog *MatrixDBM* sistema za upravljanje bazom podataka u proces matričnih izračunavanja, kroz primenu na osnovna matrična izračunavanja (množenje, sabiranje, oduzimanje, determinanta, inverz itd.), izračunavanja generalisanih inverza (*MP* inverz i težinski *MP* inverz), kao i primenu matričnih izračunavanja na triangulaciju konveksnih poligona i obradu slike.

Veliki broj međunarodnih i domaćih naučnih i stručnih časopisa, konferencija, udžbenika i internet resursa govore u prilog aktuelnosti teme doktorske disertacije. U radu su analizirani dostupni algoritami u [38,39,40,41,43,51,52,62,66] vezani za matričnu algebru (osnovna matrična izračunavanja i generalisane inverze), algoritmi u radovima [15,20,25] vezani za triangulaciju konveksnih poligona kao i radovi [3,9,28] vezani za novi pristup matričnih izračunavanja, *Cassandra NoSQL* sistem skladištenja.

Pretraga u velikim bazama podataka ili ponovno izračunavanje u algoritmima sa rekurzijama i problemima karakterističnim za tehnike dinamičkog programiranja su neka od pitanja koja će biti u fokusu. Kako bi podržali širok spektar zadataka i implementirali različite algoritme, opisali smo tehnike skladištenja matrica kao i tehnike za identifikovanje traženih vrednosti.

Web aplikacija je instalirana na serveru i nezavisna je od platforme, pa robusne i skupe aplikacije, koje je potrebno instalirati na sopstvenim računarima i koje zauzimaju memorijske resurse, postaju prošlost. Uz to, aplikacija je besplatna i dostupna korisniku u bilo koje vreme i sa bilo kog mesta. Aplikacija je implementirana alatima *OSS* (softver otvorenog koda), koji je postao predmet interesovanja u akademskoj javnosti sredinom 90-tih. *OSS* je skrenuo pažnju na sebe prevashodno svojim ekonomskim prednostima [30]. Postoji mnogo uspešnih projekata vezanih za *OSS* zajednicu, kao što su *Mozilla* browser, *Linux* operativni sistem, *Apache* web server, *PHP* [34] i *Perl* programski jezici, kao i *MySQL* baza podataka i *Apache Cassandra NoSQL* sistem skladištenja [9]. Na osnovu rezultata obrade koji se skladište u bazi moguće je kreirati *XML* i *PDF* fajlove koji služe za razmenu podataka sa drugim aplikacijama. *PHP* i *MySQL* su jedni od najpopularnijih *OSS* sistema zbog svojih brzih performansi, visoke pouzdanosti i lake upotrebe. Karakteristike i aktuelnost *PHP*-a, *MySQL* –a i *Cassandra* modela bile su od presudnog značaja za izbor tehnologije za implementaciju aplikacije. Korišćenje programskog jezika *PHP* prevazilazi ograničenja koja se susreću u potprocedurama slične konotacije pisanim u *Fortran*-u i *C*-u [22,38]. *SQL* je

standardni jezik koji služi za analizu i sprovođenje upita na podacima u relacionoj bazi podataka [11]. Na nesreću, *SQL* nema ugrađene metode za sprovođenje operacija nad vektorima i matricama. Interesantan kod za sprovođenje operacija nad matricama može se naći u radu [42]. Problemi koji se javljaju u relacionim bazama podataka, kakav je *MySQL*, a tiče se brzine unosa i pretrage podataka, rešen je *Cassandra NoSQL* sistemom skladištenja. Ovaj pristup je izuzetno aktuelan sa stanovišta dizajna. Prvobitna namena mu je bila skladištenje podataka u prijemnom poštanskom sandučetu *Facebook*-a [28] koga karakteriše visoka propusna moć. U pitanju je pisanje milijarde pisama dnevno kao i formiranje skale sa korisnicima. *Cassandra* ne podržava relacione modele podatak, već pruža klijentima jednostavan model podataka koji podržava dinamičke kontrole nad rasporedom i formatom podataka. *Cassandra* sistem je dizajniran da radi na jeftinim hardverima velike propusne moći, ali ne na uštrb efikasnosti pri čitanju. *Cassandru* kao bazu podataka koriste neke od najpoznatijih korporacija na web-u kao što su *Amazon web shop, Facebook, Twitter, Cisco, Rackspace, Digg, Cloudkick, Reddit* i drugi.

Cilj istraživanja je teorijsko (naučno) i praktično unapređenje postojećih metoda i algoritama za uspešna i efikasna matrična izračunavanja implementacijom klijent/server aplikacije na bazi *MatrixDBMS* sistema skladištenja. Ovaj zadatak je složen i multidisciplinaran jer zadire u oblasti programiranja, matematike, softverskog inženjeringa, modeliranja, internet tehnologija, računarske grafike i obrade slika. Metodologija obuhvata čitav životni ciklus od planiranja i definisanja korisničkih zahteva do implementacije sistema. Model troslojne web arhitekture (klijentski sloj, srednji sloj i sloj baze podataka), sa *MatrixDBMS* sistemom skladištenja i *MatrixAL* sistemom reprezentacije na njenim slojevima, je bio osnova za razvoj aplikacije. Brzina i fleksibilnost su glavne prednosti predstavljenog *MatrixDBMS* sistema skladištenja. Predloženi *DB* i *LA* mehanizmi omogućavaju univerzalnu podršku sprovođenju operacija na ulaznim matricama, efikasno skladištenje matrica i pronalaženje podataka. *DB* mehanizam čuva čitav niz informacija koje se odnose na ulaznu matricu. Svi korisnički upiti se arhiviraju u sistemu i na taj način se izbegava ponavljanje istog zahteva. Relacija između ulazne matrice *IM* i skupa rezultujućih matrica *OM* se takođe čuva. Vreme koje se utroši na komunikaciju sa bazom je zanemarljivo u odnosu na ponovno izračunavanje. Detaljnije, implementacija na aplikacionom sloju se sastoji u pronalaženju najefikasnijeg sistema za predstavljanje matrice (*MatrixAL*) analizom jednodimenzionalnog niza (*1D_niz*), dvodimenzionalnog niza (*2z_niy*) i objekta klase (*object*). Implementacija na sloju baze podataka je podrazumevala kreiranje najefikasnijeg sistema matrične prezentacije (sistem za skladištenje u troslojnoj web arhitekturi). To je postignuto korišćenjem usluga

DBMS (*MatrixDBMS*) i to skladištenje matrice u jednom redu tabele u bazi podataka (*R format*) i skladištenje matrice u onoliko redova tabele koliko je i redova matrice (*mR format*). Kombinacijom predstavljenih rešenja može se zaobići složeni sistem za sprovođenje operacija nad matricama, uključiti sistem za skladištenje i na taj način skratiti *CPU* vreme izvršavanja. Polazeći od navedenog cilja, potrebno je realizovati sledeće zadatke istraživanja:

a) Ispitati kakav uticaj ima *PHP/MySQL* razvojno okruženje na brzinu sprovođenja matričnih izračunavanja.
b) Ispitati koje su prednosti web klijent/server aplikacija za matrična izračunavanja, zasnovanih na softveru otvorenog koda, u odnosu na desktop aplikacije.
c) Utvrditi ekvivalentnost težinskog *MP* i *LM* inverza.
d) Utvrditi u kojoj formi je potrebno predstaviti matrice na sloju aplikacione logike (*MatrixAL model*) u cilju što bržeg odziva pri sprovođenju operacije nad njima.
e) Ispitati u kojoj formi je potrebno predstaviti matricu u bazi podataka (*MatrixDBM* sistem skladištenja) u cilju što bržeg odziva sistema.
f) Uporediti različite načine komunikacije među nivoima troslojne web arhitekture u cilju poboljšanja performansi, lakoće korišćenja i fleksibilnosti algoritama za matrična izračunavanja, kroz primenu na triangulaciju konveksnih poligona i obradu slike.
g) Uporediti vremena izvršavanja matričnih operacija korišćenjem *MySQL* metoda baze podataka i procedura na aplikacionom sloju, troslojne web arhitekture.
h) Ispitati koji od *MySQL* tipova podataka je najefikasniji za skladištenje matrica u bazi podataka.
i) Utvrditi razlike u brzini unosa i pretrage podataka, u cilju kreiranja jedinstvenog *MatrixDBMS*, kod dva potpuno različita pristupa skladištenja podataka: relacione baze podataka (*MySQL*) i *NoSQL* sistemi skladištenja (*Cassandra*).
j) Ispitati efekat primene *MatrixDBM* sistema skladištenja na triangulaciju konveksnih poligona.
k) Utvrditi prednosti i nedostatke različitih sistema za skladištenje (baza podataka i tekstualna datoteka) pri matričnim izračunavanjima kroz primenu na triangulaciju konveksnih poligona.
l) Utvrditi najefikasniji način transformacije slike u pogodan oblik za skladištenje (matrica) u pogledu brzine i veličine podataka koju tako dobijena transformacija zauzima.

Na osnovu predmeta istraživanja, postavljenog cilja i zadataka istraživanja, potrebno je utvrditi da li praćenjem formalne procedure i optimalnim alociranjem resursa klijent/server aplikacija, bazirana na *MatrixDBMS* sistemu skladištenja, omogućava skraćene *CPU* vremena izvršavanja matričnih izračunavanja.

Da bi uspešno realizovali ciljeve istraživanja, u doktorskoj disertaciji "*Matrična izračunavanja u PHP/MySQL okruženju*" će biti korišćene posebne i pojedinačne naučne metode i postupci: analiza i sinteza, metoda apstrakcije i konkretizacije, metoda generalizacije, metoda specijalizacije i komparacije, metoda komparacije kao i induktivno i deduktivno zaključivanje.

Pored navedenih metoda istraživanja koristili smo i sledeće opšte naučne metode koje proizilaze iz specifičnosti postavljenog predmeta i cilja istraživanja a to su: modeliranje *MatrixDBMS* sistema skladištenja, implementacija klijent/server aplikacije, testiranje i analiza dobijenih rezultata.

Metodom komparacije izvršeno je poređenje teorijskih osnova i praktičnih rešenja obuhvaćenih klijent/server modelom. Metode analize i sinteze, kao i metode apstrakcije i konkretizacije, su korišćene pri istraživanju strukture postojećih sistema za matrična izračunavanja i algoritama za triangulaciju konveksnih poligona. Metodom apstrakcije i konkretizacije izvršena je obrada podataka i prikupljanje informacija o problematici vezanoj za matrična izračunavanja, triangulaciju konveksnih poligona i obradu slike sa ciljem dobijanja efikasnijeg *CPU* vremena kao i jedinstvenog modela za različite tipove matrica. U procesu upoznavanja sa nivoom rezultata, koji su postignuti istraživanjem u oblasti koja se bavi matričnim izračunavanjima korišćene su analitičko-sintetičke metode. Nakon postavke klijent/server modela, verifikovana je primenjivost postavljenog modela, obavljena je analiza dobijenih rezultata, a metodom sinteze je donet zaključak o problemima koji su bili predmet istraživanja. Izvori informacija bili su radovi objavljeni u vodećim inostranim naučnim časopisima. Utvrđivanje istinitosti polazne pretpostavke vršeno je metodama dedukcije i indukcije kao i primenom statističkih i matematičkih metoda. Metoda dedukcije i indukcije je korišćena u toku eksperimentalnog istraživanja i formiranja zaključaka o dobijenim rezultatima i opštoj primenjivosti klijent/server modela na širok spektar modela sa sličnom problematikom. Na osnovu ograničenog broja sagledanih postojećih vrsta i tipova obrađenih matrica metodom indukcije došlo se do konkretnih zaključaka. Takođe, metodu indukcije smo koristili pri komparaciji težinskog *MP* inverza i generalisanog *LM* inverza i kod problema triangulacije konveksnog poligona. Matematičkim metodama problem je predstavljen

matematičkim iskazima i dat je pristup njihovom rešavanju. Statistička metoda je korišćena u fazi pripremnih aktivnosti na oblikovanju i brojčanoj valorizaciji rezultata.

Na osnovu neposredne implementacije stečenih saznanja možemo govoriti o stepenu naučnog otkrića i naučnog objašnjenja rezultata istraživanja, čime direktno ostvarujemo i društveni značaj rada, a time i kompatibilnosti sa svetskim standardima i trendovima u oblastima koje predstavljaju predmet istraživanja, a sve u cilju postizanja što boljeg *CPU* vremena izvršavanja matričnih operacija (*MatrixDBMS*). Naučni doprinosi se ogleda u rezultatima istraživanja publikovanim u inostranim naučnim časopisima [27,46,58,59].

Doktorska disertacija donosi novine u odnosu na postojeće stanje i otvara prostor za dalja istraživanja kada su u pitanju skladištenja matrica, na koje se svode mnogi problemi. Rezultati doktorske disertacije (razvijeni klijent/server model i prateći softver) mogu imati široku praktičnu primenu u aplikacijama slične konotacije jer je predstavljeni *MatrixDBMS* model dinamičan i moguće ga je modifikovati i prilagoditi na bilo kom nivou realizacije, čak i nakon same realizacije. Model i softversko rešenje su razvijeni korišćenjem najnovijih dostignuća iz oblasti web softvera, ali su pre svega dizajnirani imajući u vidu praktičnu primenu i realne zahteve. Priroda problema i postavljeni cilj istraživanja uticali su na to da ovo istraživanje ima empirijski i eksperimentalni karakter.

Glava 2

2. Matrična algebra i sistemi skladištenja

U ovoj oblasti su predstavljena izračunavanja osnovnih matričnih operacija (sabiranje, oduzimanje, množenje, množenje skalarom, determinanta, inverz itd.) i generalisanih inverza (*MP* inverz i težinski *MP* inverz), baziranih na sistemima skladištenja. Data je i teorema [59] i dokaz o ekvivalentnosti težinskog [65] i *LM* inverza [62]. Rezultati pomenute teoreme su korišćeni u klijent/server aplikaciji koja je implementirana po kriterijumima troslojne web arhitekture. Inkorporacija sistema skladištenja matrica pri sprovođenju operacija matrične algebre je predstavljena u drugom odeljku, dok je sistem reprezentacije matrica dat u trećem odeljku ove glave.

2.1. Matrična algebra

Osnovni pojmovi i neka od svojstava matrične algebre su dati u ovoj sekciji.

2.1.1. Osnovne matrične operacije

Neka je

$$D = \{(i,j); i = 1 \ldots . m; j = 1 \ldots n\}.$$

Realna matrica tipa $m \times n$ je funkcija

$$A: D \to R$$

čije su vrednosti $A(i,j) = a_{ij}$, a koja se simbolički zapisuje u obliku

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

Element a_{ij} dolazi na mesto (i,j) u matrici A. Brojevi $a_{m1}, a_{m2}, \dots, a_{1mn}$ čine m-ti red matrice, dok brojevi $a_{1n}, a_{2n}, \dots, a_{nm}$ čine n-tu kolonu matrice A.

Transponovana matrica matrice $A = (a_{ij})$ tipa $m \times n$ je matrica $B = (b_{ij})$ tipa $n \times m$ za koju je $b_{ji} = a_{ij}$ $(i \in \{1 \dots . m\}, j \in \{1 \dots . n\})$ (*Funkcija A.1.7*). Transponovanu matricu obeležavamo sa A^T.

Sabiranje matrica $A = (a_{ij})$ i $B = (b_{ij})$ tipa $m \times n$ daje matricu $C = (c_{ij})$ tipa $m \times n$ sa elementima $c_{ij} = a_{ij} + b_{ij}, i \in \{1 \dots . m\}, j \in \{1 \dots . n\}$. (*Funkcija A.1.2*).

Proizvod broja λ s matricom $A = (a_{ij})$ tipa $m \times n$ je matrica $\lambda A = (\lambda a_{ij})$. (*Funkcija A.1.4*).

Oduzimanje matrica $A - B$ možemo shvatiti kao skraćeni zapis $A + (-1) \cdot B$ (*Funkcija A.1.1*).

Proizvod AB matrica A i B definiše se ako su matrice saglasne tj. ako je broj kolona matrice A jednak broju redova matrice B. (*Funkcija A.1.3*). Broj redova matrice AB jednak je broju redova matrice A, a broj kolona jednak je broju kolona matrice B. Neka je matrica A tipa $m \times n$, a matrica B tipa $n \times p$. Proizvod matrica A i B je matrica C tipa $m \times p$, čiji su elementi

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=0}^{n} a_{ik}b_{kj},$$

tj. na (i,j) −tom mestu je umnožak i −tog reda matrice A i $j - te$ kolone matrice B. Važnu ulogu u množenju matrica imaju nulta i jedinična matrica (*Funkcija A.2.1*). Pod uslovom da je za proizvoljno datu matricu A množenje jediničnom matricom I, nekog reda, i nultom matricom O, nekog reda, dobro definisano, važi $A \cdot I = I \cdot A = A$ (tj. jedinična matrica deluje kao neutralni element za operaciju množenja matrica) i $A \cdot O = O \cdot A = O$. Formalno možemo uvesti i stepenovanje matrica $A \cdot A = A^2, A \cdot A \cdot A = A^3$ itd.

Determinanta je realna funkcija koja kvadratnoj matrici pridružuje realan broj (koga zovemo determinanta matrice). Determinantu matrice $A = (i,j)$ označavamo sa $|a_{ij}|$ (*Funkcija A.1.5*).

Ako je A regularna kvadratna matrica, tj. ako je $detA = 0$, tada postoji jedinstvena matrica X takva da je $AX = XA = I$, gde je I jedinična matrica. U tom slučaju X je inverzna matrica matrice A i označava se sa A^{-1} (*Funkcija A.1.6*). Ukoliko je A singularna matrica (ili pravougaona matrica), tada matrica X sa pomenutim osobinama ne postoji. U tim slučajevima, korisno je odrediti neku vrstu ”inverza” matrice A, tj. matrice koja će zadržati što je moguće više svojstava inverzne matrice. To je dovelo do pojma uopštenog inverza matrice

A. Pod uopštenim inverzom matrice *A* podrazumeva se matrica *X* koja je u izvesnom smislu pridružena matrici *A* tako da važi:

- uopšteni inverz postoji za klasu matrica koja je šira od klase regularnih matrica (u nekim slučajevima za proizvoljnu matricu *A*),
- ima neka svojstva običnog inverza i
- svodi se na obični inverz kada je *A* nesingularna kvadratna matrica.

2.1.2. Generalisani MP inverz

Generalisani (uopšteni) inverzi matrica predstavljaju uopštenje pojma običnog matričnog inverza. Potreba za jednom takvom teorijom javila se u vezi tzv. nekorektno postavljenih linearnih problema.

Ideja o generalisanim inverzima je implicitno sadržana još u radovima C. F. Gaussa iz 1809., i to u vezi sa uvođenjem principa metoda najmanjih kvadrata kod nekonzistentnih sistema. Nakon toga je I. Fredholm, 1903. godine definisao pseudoinverz linearnog integralnog operatora koji nije invertibilan u običnom smislu, a kojim se rešavaju integralne jednačine u slučajevima kada inverzni operator ne postoji. Pokazalo se da tako definisan uopšteni inverzni operator nije jedinstven. W. A. Hurwitz je 1912. godine, koristeći pojam pseudo-rezolvente, opisao čitavu klasu takvih operatora. Generalisani inverzi diferencijalnih operatora implicitno su sadržani u Hilbertovom razmatranju generalisane Greenove funkcije 1904. godine a kasnije su ih proučavali i drugi autori, npr. W. T. Reid 1931., itd.

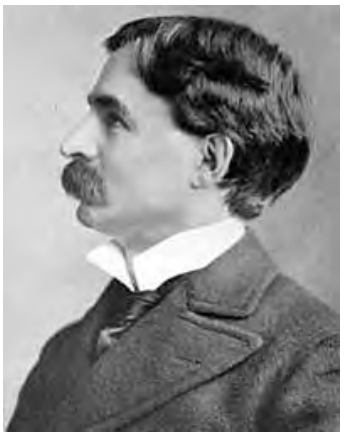
Teorijski pristup i metode za izračunavanje generalisanih inverza razvijali su se veoma brzo u poslednjih 50 godina. Publikovan je veliki broj naučnih radova i nekoliko monografija, npr. Ben-Izrael i Grevile [1], Rao i Mitra [50]Wang kao i Wei i Qiao [67].

Poznat je veći broj klasa generalisanih inverza (*Moore-Penroseov inverz*, *Drazinov inverz*, *grupni inverz*, *težinski Moore-Penroseov inverz*, *LM inverz*, *{i, j, k} inverzi*, *Bott-Duffinov inverz* itd...).

Generalisani inverzi imaju značajnu ulogu u mnogim disciplinama: linearna algebra, teorija operatora, teorija semigrupa, matematička statistika (regresija), izračunavanje polarne dekompozicije, teorija električnih kola, teorija automatskog upravljanja, filtriranje signala, diferentne jednačine, prepoznavanje slike itd. Napomenimo da je mnogo detaljnije razmatranje klasičnih primena generalisanih inverza dato u monografiji [1]. U mnogim naukama veoma često postoji potreba određivanja direktne funkcionalne zavisnosti između dve ili više veličina. Oblast matematičke statistike u kojoj generalisani inverzi imaju primenu,

a koja se bavi utvrđivanjem i opisivanjem ovih zavisnosti, naziva se *regresija*. Takođe, generalisane inverze susrećemo u teoriji automatskog upravljanja (projektovanje sistema sa povratnom vezom).

E. H. Moore [31] je 1920. prvi definisao i proučio jedinstveni generalisani inverz proizvoljne matrice, nazvavši ga ”uopštena recipročnost matrice”.



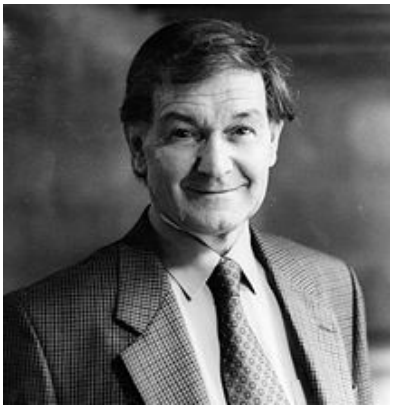
Slika 2.1.1 Eliakim Hastings Moore.

Moguće je da je do ovih rezultata Moore došao još 1906. godine, mada su prvi rezultati objavljeni tek 1920. godine. Međutim njegov rad malo je bio poznat široj javnosti, verovatno zbog specifičnosti terminologije i oznaka. Na primer, ovako je izgledala jedna teorema iz tog rada:

Teorema 2.1.1. $\mathfrak{U}^{C}\mathfrak{B}^{1\,II}\mathfrak{B}^{2\;\;II}\kappa^{12}.)$.

$$\exists|\lambda^{21\,type\,\mathfrak{M}_{\kappa}*\overline{\mathfrak{M}_{\kappa}}}\ni\cdot S^{2}\kappa^{12}\,\lambda^{21}=\delta^{11}_{\mathfrak{M}_{\kappa}}\cdot S^{1}\lambda^{21}\,\kappa^{12}=\delta^{22}_{\mathfrak{M}_{\kappa}}.$$

Više o Mooreovom rezultatima može se pronaći u [1]. Tek 1955. godine rad R. Penrosea [43] pobudio je pravi interes za izučavanje ove problematike. Penrose je dokazao da je Mooreov inverz zapravo rešenje sistema matričnih jednačina i zbog toga se ovaj inverz danas naziva Moore-Penroseov inverz.



Slika 2.1.2 Roger Penrose.

Penrose je, takođe, ukazao na ulogu ovog generalisanog inverza u rešavanju sistema linearnih jednačina.

Poznat je veći broj ekvivalentnih definicija Moore-Penroseovog inverza. R. Penrose je 1955. godine dokazao teoremu [43].

Teorema 2.1.2. (Penrose) *Za datu matricu* $A \in C^{m,n}$ *postoji jedinstvena matrica* $X \in C^{n,m}$ *koja ispunjava jednačine*

$$\begin{array}{ll}(1)\ AXA = A & (2)\ XAX = X \\ (3)\ (AX)^* = AX & (4)(XA)^* = XA\end{array} \tag{2.1}$$

Penrose je matricu X označio sa $A^\dagger$ i nazvao je *generalisani inverz matrice A*. On postoji za singularne i pravougaone matrice, a u slučaju regularne matrice je $A^\dagger = A^{-1}$. Za matricu $A^\dagger$ koristi se naziv Moore-Penroseov inverz matrice A.

Teorema 2.1.3. (Moore) *Za datu matricu* $A \in C^{m,n}$ *postoji jedinstvena matrica* $X \in C^{n,m}$ *tako da za pogodno izabrane matrice Y i Z važi:*

$$AXA = A, X = YA^* = A^*Z. \tag{2.2}$$

Osim toga je

$$XAX = (AX)^* = AX\ , (XA)^* = XA. \tag{2.3}$$

Definicija 2.1.1. (Funkcionalna definicija generalisanog inverza) *Za datu matricu* $A \in C^{m,n}$ *definišimo linearnu transformaciju* $\tilde{A}^\dagger : C^m \to C^n$ *relacijom* $\tilde{A}^+ x = 0$ *ako* $x \in R(A)^\perp$ *i* $\tilde{A}^+ x = \left(A_{|R(A^*)}\right)^{-1} x$ *ako* $x \in R(A)$. *Matrica linearne transformacije* $\tilde{A}^\dagger$ *označava se sa* $A^\dagger$ *i naziva se generalisani inverz za A.*

Definicija 2.1.2. (Mooreova definicija) *Generalisani inverz za* $A \in C^{m\times n}$ *je jedinstvena matrica* $A^\dagger$ *takva da je*

$$(i)\ AA^\dagger = P_{R(A)}\ (ii)\ A^\dagger A = P_{R(A^\dagger)}.$$

Definicija 2.1.3. (Penroseova definicija). *Za* $A \in C^{m\times n}$ *generalisani inverz je jedinstvena matrica* $A^\dagger \in C^{m\times n}$ *koja ispunjava jednačine (1), (2), (3)* i *(4).*

Teorema 2.1.4. *Funkcionalna, Mooreova i Penroseova definicija generalisanog inverza su ekvivalentne.*

Poznate su različite metode za izračunavanje *MP* inverza matrica. Uglavnom se ti metodi zasnivaju na *Cayley–Hamiltonovoj* teoremi potpune rang faktorizacije i singularno – vrednosne dekompozicije. Grevilov metod pregrađivanja uveden u [18] je jedan od najefikasnijih metoda za izračunavanje MP inverza.

Sivakumar je u [54] koristio Grevilovu formula i ukazao je da zadovoljava četiri principa. Ovo pruža dokaz o ispravnosti Grevilovih metoda. Grevilov algoritam se koristi u različitim varijantama izračunavanja i verifikovan je preko različitih direktnih metoda za izračunavanja pseudo inverza. Izračunavanje $A(s)^{\dagger}$ za racionalne matrice (*Funkcija A.2.13*), je dat sledećim Algoritmom.

Algoritam 2.1.1. Izračunavanje $\boldsymbol{A(s)^{\dagger}}$ za racionalne matrice.

1: $A_1 = a_1$

2: **if** $a_1 = 0$ **then** $A_1^{\dagger} = a_1^*$ **else** $A_1^{\dagger} = \frac{1}{a_1^* a_1} a_1^*$

3:**for** $i = 2\ to\ n$ **do**

4: $d_i = {A_{i-1}}^{\dagger} a_i$

5: $c_i = a_i - A_{i-1} d_i = (I - A_{i-1} {A_{i-1}}^{\dagger}) a_i$

6: **if** $c_i = 0$ **then** $b_i = \frac{1}{1 + d_i^* d_i} \left(\left(A_{i-1}^{\dagger} \right)^* d_i \right)$ **else** $b_i = \frac{1}{c_i^* c_i} c_i$

7: $A_i^{\dagger} = \begin{bmatrix} A_{i-1}^{\dagger} - d_i b_i^* \\ b_i^* \end{bmatrix}$

8: **end for**

9: **return** $A^{\dagger}$

Na osnovu rezultata u [29] imamo da će Grevilov algoritam dati čak 8 puta bolje rezultate od *SVD*, kad su u pitanju kvadratne matrice sa svim elementima različitim od nule. Takođe, kod pravougaonih slučajeva imamo sličan doprinos u brzini.

Grevilovi rekurzivni principi se koriste za generisanje podskupova spoljnih inverza proširenih na racionalne matrice sa jednom promenjivom kao i na polinomijalne matrice. Algoritam za izračunavanje *MP* inverza racionalnih i/ili polinomijalnih matrica sa jednom promenljivom dat je u [57].

2.1.3. Težinski MP inverz

Postoji mnogo nastavaka metode pregrađivanja. Wang u radu [66], primenjuje generalisani Grevilov metod za izračunavanje težinskog MP inverza i ti rezultati su dobijeni korišćenjem do tada nepoznatog pristupa.

Neka je $A \in C^{m \times n}$ i neka su $M \in C^{m \times m}$ i $N \in C^{n \times n}$ Hermitske, pozitivno definisane matrice. Težinski skalarni proizvod u prostorima C^m i C^n, respektivno, možemo definisati na sledeći način:

$$(x, y)_M = y^* M x,\ (x, y)_N = y^* N x.$$

Ovako definisani skalarni proizvodi indukuju norme $\|x\|_M$ i $\|x\|_N$ na uobičajen način. Podsetimo se da konjugovano transponovana matrica A^* zadovoljava $(Ax, y) = (x, A^*y)$ za svako $x \in C^{m\times 1}$ i $y \in C^{n\times 1}$. Na isti način možemo defnisati *težinsku konjugovano transponovanu matricu* $A^{\#}$ koja zadovoljava $(Ax, y)_M = (x, A^{\#}y)_N$.

Teorema 2.1.5. *Neka je* $A \in C^{m\times n}$, *skup kompleksnih matrica i* $C_r^{m\times n} = \{X \in C^{m\times n}: rank(X) = r\}$ *i neka su M i N hermitske, pozitivno definisane matrice reda m i n, respektivno. Tada je* $x = Xb$ *minimalno (u odnosu na normu* $\| \ \|_N$*) srednje kvadratno rešenje (u odnosu na normu* $\| \ \|_M$*) nekonzistentnog sistema linearnih jednačina* $Ax = b$ *za svako* $b \notin R(A)$ *ako i samo ako X zadovoljava sledeće matrične jednačine*

$$\begin{array}{l}(1)\ AXA = A \quad (3M)\ (MAX)^* = MAX \\ (2)\ XAX = X \quad (4N)\ (NXA)^* = NXA\end{array} \tag{2.4}$$

gde $*$ označava konjugovanu i transponovanu vrednost. Štaviše, sistem matričnih jednačina (1) ima jedinstveno rešenje.

Matrica X koja zadovoljava (1) naziva se *težinski Moore-Penroseov inverz* i označava sa $X = A_{MN}^{\dagger}$. Težinski Moore-Penroseov inverz $A_{MN}^{\dagger}$ je uopštenje Moore-Penroseovog inverza $A^{\dagger}$. Ako je $M = I_m$, $N = I_n$, onda je $A_{MN}^{\dagger} = A^{\dagger}$, tj. težinski Moore-Penroseov inverz se svodi na Moore-Penroseovog inverz.

Teorema 2.1.6. (Wang i Chen [65]) *Neka* $A \in C^{m\times n}$ *i* A_k *je podmatrica matrice A koja se sastoji od k prvih kolona. Za* $k = 2, \dots, n$ *matrica* A_k *je predstavljena sa*

$$A_k = [A_{k-1}|a_k] \tag{2.5}$$

i neka je data submatrica matrice N $N_k \in C^{k\times k}$

$$N_k = \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix}. \tag{2.6}$$

Neka su matrice X_{k-1} *i* X_k *definisane kao*

$$X_{k-1} = (A_{k-1})_{MN_{k-1}}^{\dagger} X_k = (A_k)_{MN_k}^{\dagger}, \tag{2.7}$$

a vektori d_k *i* c_k *definisani kao*

$$d_k = X_{k-1}a_k \tag{2.8}$$

$$c_k = a_k - A_{k-1}d_k = (I - A_{k-1}X_{k-1})a_k, \tag{2.9}$$

onda je

$$X_k = \begin{bmatrix} X_{k-1} - (d_k + (I - X_{k-1}A_{k-1})N_{k-1}^{-1}l_k)b_k^* \\ b_k^* \end{bmatrix}, \tag{2.10}$$

gde je

$$b_k^* = \begin{cases} (c_k^* M c_k)^{-1} c_k^* M & c_k \neq 0 \\ \delta_k^{-1}(d_k^* N_{k-1} - l_k^*)X_{k-1} & c_k = 0 \end{cases} \tag{2.11}$$

i

$$\delta_k = n_{kk} + d_k^* N_{k-1} d_k - l_k^* d_k - d_k^* l_k - l_k^*(I - X_{k-1}A_{k-1})N_{k-1}^{-1}l_k. \tag{2.12}$$

Algoritam 2.1.2. Izračunavanje težinskog MP inverza $\boldsymbol{A_{MN}(s)^\dagger}$ iz [65].

Require: Neka je $A \in C^{m\times n}$ i M i N pozitivno definisane matrice dimanzija $m \times m$ i $n \times n$, respektivno.

1: $A_1 = a_1$
2: **if** $a_1 = 0$ **then**
3: $\quad X_1^\dagger = \dfrac{1}{a_1^* M a_1} a_1^* M$
4: **else**
5: $\quad X_1 = 0$
6: **end if**
7: **for** $k = 2\ to\ n$ **do**
8: $\quad d_k = X_{k-1}a_k$
9: $\quad c_k = a_k - \hat{A}_{k-1}d_k$
10: $\quad$ **if** $c_k \neq 0$ **then**
11: $\quad\quad b_k^* = (c_k^* M c_k)^{-1} c_k^* M$, **go to** Step 16
12: $\quad$ **else**
13: $\quad\quad \delta_k = n_{kk} + d_k^* N_{k-1} d_k - (d_k^* l_k + l_k^* d_k) - l_k^*\left(I - X_{k-1}\hat{A}_{k-1}\right)N_{k-1}^{-1}l_k$
14: $\quad\quad b_k^* = \delta_k^{-1}(d_k^* N_{k-1} - l_k^* X_{k-1})$
15: $\quad$ **end if**
16: $\quad X_k = \begin{bmatrix} X_{k-1} - \left(d_k + \left(I - X_{k-1}\hat{A}_{k-1}\right)N_{k-1}^{-1}l_i\right)b_k^* \\ b_k^* \end{bmatrix}$
17: **end for**
18: **return** $A_{MN}^\dagger = X_n$

Za izračunavanje Algoritma 2.1.2., iz [65], neophodan je i pomoćni Agoritam 2.1.3 (*Funkcija A.2.12*).

Algoritam 2.1.3. Izračunavanje $\boldsymbol{N_i^{-1}}$ za racionalne matrice.

1: $N_1^{-1} = n_{11}^{-1}$
2: **for** $k = 2\ to\ n$ **do**
3: $g_{kk} = \left(n_{kk} - l_k^* N_{k-1}^{-1} k\right)^{-1}$
4: $f_{ii} = -g_{kk} N_{k-1}^{-1} k$
5: $E_{k-1} = N_{k-1}^{-1} + g_{kk}^{-1} f_k f_k^*$
6: $N_k^{-1} = \begin{bmatrix} E_{k-1} & f_k \\ f_k^* & g_{kk} \end{bmatrix}$
7: **end for**
8: **return** $N^{-1} = N_n^{-1}$

Wangov metod pregrađivanja za izračunavanje težinskog *MP* inverza, je proširen na skup racionalnih i polinomijalnih matrica sa jednom promenljivom u [57]. Takođe, efikasan algoritam za izračunavanje težinskog *MP* inverza, prikladan polinomijalnom izračunavanju, gde su samo nekoliko koeficijenata različiti od nule, je predstavljen u [32].

2.1.4. LM inverz

Definicija generalisanog *LM* inverza data u [61] bazira se na korišćenju linearne jednačine $Ax = b$, gde je A matrica $m \times n$, b je m - vector i x je n- vector.

Matrica $A_{LM}^{\dagger}$ je takva da je vektor x jedinstveno dat jednačinom $x = A_{LM}^{\dagger} b$ minimizujući forme sledeća dva vektora (uslovi (3) i (4) iz [61])

$$G = \left\| L^{1/2}(Bx - b) \right\|^2 = \left\| L^{1/2}(Bx - b) \right\|_L^2 \text{ i}$$

$$H = \left\| M^{1/2} \right\|^2 = \|x\|_M^2,$$

gde su L i M pozitivno definisane matrice dimenzija $m \times m$ i $n \times n$, respektivno. Rekurzivne formule za određivanje generalisanog *LM* inverza $A_{LM}^{\dagger}$ bilo koje matrice A su uvedene u [61,64] i one su ovde svrsishodno prepravljene.

Udwadia i Kalaba su dali jednostavan i konstruktivan dokaz Grevilovih formula u [64]. U [13] su Fan i Kalaba odredili *MP* inverz matrica koristeći dinamičko programiranje i Belmanov princip optimizacije. Udwadia i Kalaba su razvili rekurzivne relacije za različite tipove generalisanih inverza [61,64].

Teorema 2.1.7. (Udwadia i Phohomsiri [61,62]) *Generalisani LM inverz bilo koje date matrice* $B = [A|a] \in C^{m \times n}$ *je determinisan korišćenjem sledećih rekurzivnih relacija*

$$B_{LM}^{\dagger} = [A|a]_{LM}^{\dagger} = \begin{cases} \begin{bmatrix} A_{LM\_}^{\dagger} - A_{LM\_}^{\dagger} a d_L^{\dagger} - p d_L^{\dagger} \\ d_L^{\dagger} \end{bmatrix} & d = \left(I - A A_{LM\_}^{\dagger}\right) a \neq 0 \\ \begin{bmatrix} A_{LM\_}^{\dagger} - A_{LM\_}^{\dagger} a h - p h \\ h \end{bmatrix} & d = \left(I - A A_{LM\_}^{\dagger}\right) a = 0 \end{cases} \tag{2.13}$$

gde je A matrica dimenzija $m \times (n-1)$*, a je vektor kolona sa* m *elemenata,*

$$d_L^{\dagger} = \frac{d^T L}{d^T L d}\ h = \frac{q^T M U}{q^T M q}\ U = \begin{bmatrix} A_{LM\_}^{\dagger} \\ 0_{1\times 0} \end{bmatrix}\ q = \begin{bmatrix} v + p \\ -1 \end{bmatrix}\ v = A_{LM\_}^{\dagger} a \text{ i } p = \left(I - A_{LM\_}^{\dagger} A\right) M_{\_}^{-1} \tilde{m}.$$

Napominjemo da je L simetrična pozitivno definisana matrica dimenzija $m \times m$, a M je data kao:

$$M = \begin{bmatrix} M_{\_} & \tilde{m} \\ \tilde{m}^T & \overline{m} \end{bmatrix}, \tag{2.14}$$

gde je M simetrična pozitivno definisana matrica dimenzija $n \times n$, $M_{\_}$ je simetrična pozitivno definisana matrica dimenzija $(n-1) \times (n-1)$, $\tilde{m}$ je vektor kolona sa $n-1$ elemenata, a $\overline{m}$ je skalar. U ostatku rada ćemo uvesti sledeću oznaku $B = A_k$, $A = A_{k-1}$, $a = a_k$. Štaviše, jasno je da sledeća notacija neposredno sledi iz Algoritma 2.1.2:

$$B_{LM}^{\dagger} = X_k,\ A^{\dagger} = X_{k-1}.$$

Takođe ćemo koristiti sledeću notaciju za *M*

$$M = \begin{bmatrix} M_{k-1} & \tilde{m}_k \\ \tilde{m}_k^T & m_{kk} \end{bmatrix}. \tag{2.15}$$

Konačno, vektor d kojem odgovara prvih k kolona matrice *A* biće označen sa d_k.

Definicija *LM* inverza i rekurzivnog algoritma Grevilovog tipa (za matrice kojima se dodaje vektor kolona) su dati u [61,62]. Rekurzivne relacije su dokazane direktnom verifikacijom četiri uslova za generalisani *LM* inverz. Odvojene relacije u situaciji kad imamo pravougaone matrice koje se proširuju kolona vektorom i red vektorom su date u [61]. Jedan od dokaza za određivanje generalisanog *MP M* inverza matrica preko direktne verifikacije četiri svojstva *MP M* inverza je dat u [63].

Nije teško pokazati da su uslovi koji karakterišu generalisani *LM* inverz ekvivalentni onima koji karakterišu težinski *MP* inverz. Štaviše, norme minimizacije matrice (3) i (4) korištene u radu [61] takođe karakterišu težinski *MP* inverz. Dakle, težinski *MP* inverz i generalisani LM inverz su ekvivalentni.

U skladu sa Teoremom 2.1.7. predstavićemo sledeći Algoritam (*Funkcija A.2.15*).

Algoritam 2.1.4. Izračunavanje generalisanog LM inverza $\boldsymbol{A_{LM}}^{\dagger}$ iz [61].

Require: Neka je $A \in C^{m\times n}$ i L i M pozitivno definisane matrice dimanzija $m \times m$ i $n \times n$, respektivno.

1: $A_1 = a_1$
2: **if** $a_1 = 0$ **then**
3: $X_1 = \dfrac{1}{a_1^* L a_1} a_1^* L$
4: **else**
5: $X_1 = 0$
6: **end if**
7: **for** $k = 2\ to\ n$ **do**
8: $d_k = (I - A_{k-1}X_{k-1})a_k$
9: $p = (I - X_{k-1}A_{k-1})M_{\_}^{-1}\tilde{m}$
10: **if** $d_k \neq 0$ **then**
11: $b_k^* = (c_k^* L c_k)^{-1} c_k^* L$, **go to** $Step\ 17$
12: **else**
13: $q = \begin{bmatrix} X_{k-1}a_k + p \\ -1 \end{bmatrix}$
14: $U = \begin{bmatrix} X_{k-1} \\ 0_{1\times 0} \end{bmatrix}$
15: $b_k^* = \dfrac{q^*}{q^{*T} M_k q} M_k U$
16: **end if**
17: $X_k = \begin{bmatrix} X_{k-1} - X_{k-1}a_k b_k^* - p b_k^* \\ b_k^* \end{bmatrix}$
18: **end for**
19: **return** $A_{LM}^{\dagger} = X_n$

U ovom radu smo upoređivali odgovarajuće algoritme za težinski *MP* inverz i generalisani *LM* inverz. Testiranja su potvrdila da Algoritmi 2.1.2. i 2.1.4. daju iste rezultate. Realno je predvideti ekvivalentnost algoritama za izračunavanje težinskog *MP* inverza, predstavljenog u [65], i generalisanog *LM* inverza, predstavljenog u ovom radu i baziranog na rezultatima [61,62]. Verifikacija ovog predviđanja je jedan od rezultata ovog rada.

2.1.5. Ekvivalentnost Težinskog i LM inverza

Teorema 2.1.8. [59] Težinski MP inverz [65] je identičan *LM* inverzu predstavljenom u [61].

Dokaz. Kako bi obezbedili nedvosmislenost tokom dokaza pretpostavili smo da oznaku za Wangov algoritam predstavlja simbol W u eksponentu. Slično tome oznaka U se odnosi na Algoritam Udwadije. Pošto smo pretpostavili da su *LM inverz* i težinski *MP inverz* identični, zaključujemo da su matrice M i N, iz Algoritma 2.1.2. koji se odnosi na težinski MP inverz, upravo matrice L i M iz Algoritma 2.1.4 koji se odnosi na *LM* inverz, te ih stoga nije neophodno označavati odgovarajućim eksponentom.

Teoremu smo dokazali verifikacijom ekvivalentnosti izlaza odgovarajućih algoritamskih koraka pomenutih Algoritama. Do dokaza smo došli matematičkom indukcijom.

Za slučaj $k = 1$ dokaz ekvivalentnosti relacije iz koraka 3 u oba algoritma je trivijalan. Pretpostavimo da iskaz važi za prvih $k - 1$ kolona, npr.

$$X_{k-1}^{U} = X_{k-1}^{W} = (A_{k-1})_{MN_{k-1}}^{\dagger} = A_{LM\_}^{\dagger}. \tag{2.16}$$

Wang koristi matricu X_ku formi

$$X_k^W = \begin{bmatrix} X_{k-1}^W - (d_k^W + (I - X_{k-1}^W A_{k-1}) N_{k-1}^{-1} l_k)(b_k^*)^W \\ (b_k^*)^W \end{bmatrix}, \tag{2.17}$$

dok Udwadia posmatra dva slučaja kao u (2.13). Ako ih označimo sa

$$(b_k^*)^U = \begin{cases} d_L^\dagger & d_k^U \neq 0 \\ h & d_k^U = 0 \end{cases} \tag{2.18}$$

Onda jednačine u (2.13) postaju

$$X_k^U = \begin{bmatrix} X_{k-1}^U - X_{k-1}^U a_k (b_k^*)^U \\ (b_k^*)^U \end{bmatrix}. \tag{2.19}$$

Pokažimo da je izlaz koraka 9 iz Algoritma 2.1.2 isti kao izlaz koraka 8 iz Algoritma 2.1.4.

$$c_k^W \equiv a_k - A_{k-1} d_k^W = a_k - A_{k-1} X_{k-1}^W a_k = (I - A_{k-1} X_{k-1}^W) a_k \equiv d_k^U.$$

Sada ćemo pokazati da koraci 11 oba algoritma daju ekvivalentne izlaze. Već smo prethodno naveli da važi $c_k^W = d_k^U$, tako da ćemo u slučaju $c_k^W \neq 0$ imati

$$(b_k^*)^W = ((c_k^*)^W M c_k^W)^{-1} (c_k^*)^W M = ((d_k^*)^U L d_k^U)^{-1} (d_k^*)^U L = (b_k^*)^U.$$

Na sličan način ćemo pokazati da su koraci 14 i 15 iz Algoritma 2.1.2 i Algoritma 2.1.4, respektivno, ekvivalentni. Za slučaj $c_k^W = 0$ počinjemo sa izrazom iz koraka 15 Algoritma 2.1.4.

$$(b_k^*)^U = \frac{q^* M_k U}{q^* M_k q}.$$

Na osnovu koraka 9 iz Algoritma 2.1.4 i induktivne hipoteze imamo

$$p = (I - X_{k-1}^U A_{k-1}) N_{k-1}^{-1} l_k = (I - X_{k-1}^W A_{k-1}) N_{k-1}^{-1} l_k$$

tako da dobijamo

$$\begin{aligned}
q^T M_k U &= [(X_{k-1}^U a_k + p)^* | -1] \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix} \begin{bmatrix} X_{k-1}^U \\ 0 \end{bmatrix} \\
&= [(X_{k-1}^U a_k + p)^* N_{k-1} - l_k^* | (X_{k-1}^U a_k + p)^* l_k - n_{kk}] \begin{bmatrix} X_{k-1}^U \\ 0 \end{bmatrix} \\
&= (X_{k-1}^U a_k + p)^* N_{k-1} X_{k-1}^U - l_k^* X_{k-1}^U \{since\ X_{k-1}^U a_k = X_{k-1}^W a_k = d_k^W\} \\
&= (d_k^*)^W N_{k-1} X_{k-1}^W - l_k^* X_{k-1}^W + p^* N_{k-1} X_{k-1}^W \\
&= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + \left((I - X_{k-1}^W A_{k-1}) N_{k-1}^{-1} l_k\right)^* N_{k-1} X_{k-1}^W \\
&= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + (l_k^* N_{k-1}^{-1} (I - X_{k-1}^W A_{k-1})^*) N_{k-1} X_{k-1}^W \\
&= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + l_k^* X_{k-1}^W - l_k^* N_{k-1}^{-1} (X_{k-1}^W A_{k-1})^* N_{k-1} X_{k-1}^W
\end{aligned}$$

Kako je N_{k-1} pozitivno definisano, primenom jednakosti (4N) na prethodni izraz dobijamo

$$(X_{k-1}^W A_{k-1})^* N_{k-1} = (N_{k-1} X_{k-1}^W A_{k-1})^* = N_{k-1} X_{k-1}^W A_{k-1},$$

i zatim

$$\begin{aligned}
q^T M_k U &= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + l_k^* X_{k-1}^W - l_k^* N_{k-1}^{-1} N_{k-1} X_{k-1}^W A_{k-1} X_{k-1}^W \\
&= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + l_k^* (X_{k-1}^W - X_{k-1}^W A_{k-1} X_{k-1}^W) \\
&= ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W + l_k^* (X_{k-1}^W - X_{k-1}^W) = ((d_k^*)^W N_{k-1} - l_k^*) X_{k-1}^W
\end{aligned}$$

Imamo da je i

$$\begin{aligned}
q^T M_k q &= [(X_{k-1}^U a_k + p)^* | -1] \begin{bmatrix} N_{k-1} & l_k \\ l_k^* & n_{kk} \end{bmatrix} \begin{bmatrix} X_{k-1}^U a_k + p \\ -1 \end{bmatrix}. \\
&= [(X_{k-1}^U a_k + p)^* N_{k-1} - l_k^* | (X_{k-1}^U a_k + p)^* l_k - n_{kk}] \begin{bmatrix} X_{k-1}^U a_k + p \\ -1 \end{bmatrix} \\
&= ((X_{k-1}^U a_k + p)^* N_{k-1} - l_k^*)(X_{k-1}^U a_k + p) - (X_{k-1}^U a_k + p)^* l_k \\
&+ n_{kk} (since \{X_{k-1}^U a_k = X_{k-1}^W a_k = d_k^W\}) \\
&= (d_k^W + p)^* N_{k-1} (d_k^W + p) - l_k^* (d_k^W + p) - (d_k^W + p)^* l_k + n_{kk} \\
&= (d_k^W)^* N_{k-1} d_k^W - l_k^* d_k^W - (d_k^*)^W l_k + n_{kk} - l_k^* p + p^* N_{k-1} d_k^W \\
&+ (d_k^*)^W N_{k-1} p + p^* N_{k-1} p - p^* l_k
\end{aligned}$$

Osim toga imamo i da je $p^*N_{k-1}d_k^W + (d_k^*)^W N_{k-1}p + p^*N_{k-1}p - p^*l_k = 0$. Prvo ćemo pokazati da je $p^*N_{k-1}d_k^W$, kao što sledi

$$\begin{aligned} p^*N_{k-1}d_k^W &= \left((I - X_{k-1}^W A_{k-1})N_{k-1}^{-1}l_k\right)^* N_{k-1}X_{k-1}^W a_k \\ &= l_k^* N_{k-1}^{-1}(I - X_{k-1}^W A_{k-1})^* N_{k-1}X_{k-1}^W a_k \\ &= l_k^* N_{k-1}^{-1}(N_{k-1} - (X_{k-1}^W A_{k-1})^* N_{k-1})X_{k-1}^W a_k \\ &= l_k^* N_{k-1}^{-1}(N_{k-1} - N_{k-1}X_{k-1}^W A_{k-1})X_{k-1}^W a_k \\ &= l_k^* N_{k-1}^{-1}N_{k-1}(I - X_{k-1}^W A_{k-1})X_{k-1}^W a_k = l_k^*(I - X_{k-1}^W A_{k-1})X_{k-1}^W a_k \\ &= l_k^*(X_{k-1}^W - X_{k-1}^W A_{k-1}X_{k-1}^W)a_k = l_k^*(X_{k-1}^W - X_{k-1}^W)a_k = 0 \end{aligned}$$

Takođe, iz prethodne jednakosti imamo $(d_k^*)^W N_{k-1}p = (p^*N_{k-1}d_k^W)^* = 0$. Konačno, poslednji termin u sumi na levoj strani izraza, $p^*N_{k-1}p - p^*l_k$, je takođe nula

$$\begin{aligned} p^*N_{k-1}p - p^*l_k &= p^*(N_{k-1}(I - X_{k-1}^W A_{k-1})N_{k-1}^{-1}l_k - l_k) = p^*(-N_{k-1}X_{k-1}^W A_{k-1}N_{k-1}^{-1}l_k) \\ &= p^*(-(X_{k-1}^W A_{k-1})^* l_k) = \left((I - X_{k-1}^W A_{k-1})N_{k-1}^{-1}l_k\right)^* (-A_{k-1}^*(X_{k-1}^*)^W l_k) \\ &= -l_k^* N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W l_k + l_k^* N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W A_{k-1}^*(X_{k-1}^*)^W l_k \\ &= l_k^* N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W l_k + l_k^* N_{k-1}^{-1}A_{k-1}^* + l_k^* N_{k-1}^{-1}A_{k-1}^*(X_{k-1}^*)^W = 0 \end{aligned}$$

Nastavljajući sa transformacijom izraza $q^T M_k q$ imamo

$$q^T M_k q = (d_k^*)^W N_{k-1}d_k^W - l_k^* d_k^W - (d_k^*)^W l_k + n_{kk} - l_k^*(I - X_{k-1}^W A_{k-1})N_{k-1}^{-1}l_k = \delta_k.$$

Sada imamo argumenata da dokaz privedemo kraju. Na osnovu jednakosti

$$(b_k^*)^U = \frac{q^* M_k U}{q^* M_k q}.$$

Korak 15 iz Algoritma 2.1.4 daje rezultat $(b_k^*)^U = \delta_k^{-1}(d_k^* N_{k-1} - l_k^*)X_{k-1}^W$, koji je identičan rezultatu $(b_k^*)^W$, izveden iz koraka 14 Algoritma 2.1.2. Na osnovu koraka 16 i 17 Algoritama 2.1.2 i Algoritma 2.1.4, respektivno, imamo da su generalisani *LM* inverz i težinski *MP* inverz prvih k kolona matrice A identični, što je bila i polazna hipoteza.

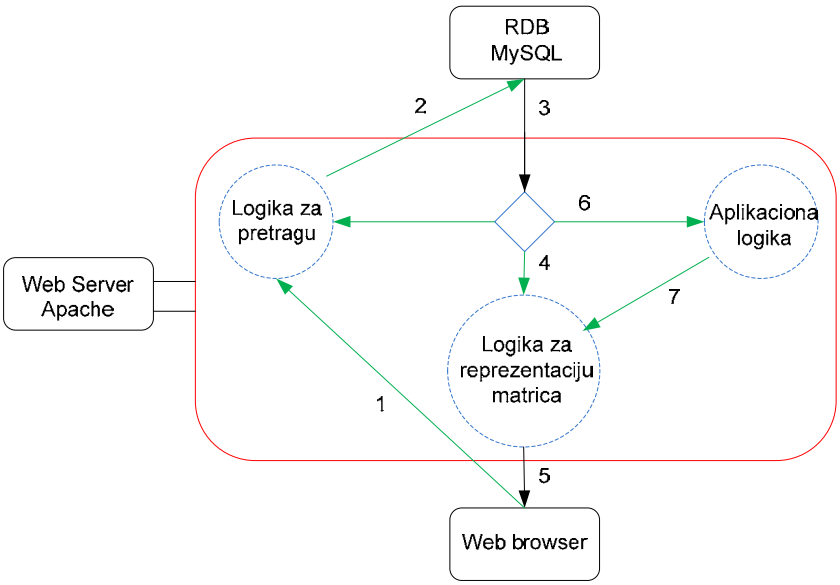
$$X_k^U = \begin{bmatrix} X_{k-1}^U - (X_{k-1}^U a_k + p)(b_k^*)^U \\ (b_k^*)^U \end{bmatrix} = \begin{bmatrix} X_{k-1}^W - (d_k^W + (I - X_{k-1}^W A_{k-1})N_{k-1}^{-1}l_k)(b_k^*)^W \\ (b_k^*)^W \end{bmatrix} = X_k^W.$$

Konačno za $k = n$, neposredno sledi da je $A_{LM}^\dagger = A_{MN}^\dagger$, što znači da su rezultati oba algoritma identični. Ekvivalentnost pomenutih algoritama je potvrđena i testiranjem klijent/server aplikacije implementirane u *PHP/MySQL* okruženju [59]. Rezultati testiranja su dati u Tabeli 2.5.1.

2.2. Sistemi skladištenja matrica (*storage systems*)

MatrixDBMS je osnova na kojoj se bazira naša klijent/server aplikacija. U cilju implementacije ovakvog sistema za matrična izračunavanja testirali smo različite metodologije za skladištenje matrica. Razvili smo algoritam za skladištenje svake ulazne matrice i odgovarajućih rezultata obrade uz sistem baze podataka.

Sledeća slika ilustruje našu ideju. Upoređujemo ulazne matrice, unete u web forme, sa matricama koje su uskladištene u tabeli baze podataka. Ukoliko ove matrice postoje u bazi podataka, potrebno ih je samo prikazati pa je logika izračunavanja određena koracima (1,2,3,4,5). U slučaju da matrice unete u web formu ne postoje u bazi podataka, logika izračunavanja prati korake (1,2,3,6,7,5) kao što je prikazano na slici 2.2.1. Drugim rečima, ako matrica i rezultati obrade postoje u bazi potrebno ih je samo pročitati. Na ovaj način se složeni matematički proračuni svode na pretragu baze podataka, što je zanemarljivo u odnosu na ponovno izračunavanje.



Slika 2.2.1 Mašina za izračunavanje na srednjem sloju.

Eksperimantalno smo uporedili uticaj različitih matričnih reprezentacija i sistema skladštenja na izračunavanje težinskog *MP* inverza korišćenjem klijent/server arhitekture i *SQL*-a, uzimajući u obzir izvodljivost, lakoću upotrebe fleksibilnost i skalabilnost.

U nastavku rada smo predstavili *MatrixDBMS* za matrična izračunavanja na osnovu ideje korišćenu u [32] gde se skladištenje matrica u bazi podataka koristi u pseudo inverznim izračunavanjima. Takođe, smo razmatrali rezultate iz [22] gde je predstavljena paralelna dinamička biblioteka matrice. Model *MatrixDBMS* sistema skladištenja je razvijen analizom

osnovnih matričnih operacija (sabiranje, oduzimanje, množenje dve matrice, izračunavanje determinanti itd.), i izračunavanjem generalisanih inverza (*MP* inverz i težinski *MP* inverz). Efekti uvođenja ovog modela, na *CPU* vreme izvršavanja matričnih operacija, je potvrđen i na primerima triangulacije konveksnih poligona i obrade slike. Sve matrične operacije razmatrane su kako za guste (dense) tako i retko posednute (sparse) matrice, sa mogućnošću proširenja na dijagonalne, trougaone i simetrične proređene matrice. Ograničenja potprograma napisanih u *FORTRANA* i *C* –u iz [22] i [37] prevaziđene su uz pomoć *PHP*. Superiornost *PHP*-a u odnosu na pomenute programske jezike ogleda se u web objektno orijentisanom pristupu i dobroj komunikaciji sa *HTML*-om, *MySQL*-om, *XML*-om, *PDF* i *Cassandra modelom*.

Predstavljena klijent/server aplikacija (vidi glavu pet) za matrična izračunavanja je otvorenog koda. Prednosti koje nudi ovakav pristup problemu u odnosu na rad u desktop okruženju su sledeće:

- korisnik ne opterećuje radnu memoriju i procesor računara glomaznim i sporim aplikacijama za matrična izračunavanja,
- nije potreban poseban program za pristup aplikaciji koja je na serveru osim operativnog sistema i web browsera koji mogu biti besplatni i
- aplikaciji je moguće pristupiti sa bilo kog mesta i u bilo koje vreme.

Matrice i njihove reprezentacije skladištenja predstavljene u radu će biti klasifikovane u nekoliko grupa.

Test matrice i matrice sa slučajno generisanim elementima su uskladištene u bazi podataka u formatu u obliku redova, *R format*, gde svi elementi jedne matrice čine jedan vektor - string koji sadrži elemente matrice. Pored *R format*-a, takođe smo testirali brzinu pretraživanja matrice kada je ova uskladištena u bazi u *mR format*-u. Ovaj format karakteriše svojstvo da su matrice uskladištene u bazi podataka tako da je broj zapisa u tabeli jednak broju redova matrice. Uporedili smo brzinu pretraživanja već uskladištenih matrica, koristeći reprezentaciju matrica u oba formata, *R format* i *mR format*. Na osnovu rezultata testiranja [58] odlučili smo se za skladištenje matrice u vidu *R formata*. Ovaj sistem skladištenja koristimo isključivo kako bi uštedeli memorijski prostor u bazi podataka i kako bi poboljšali brzinu pretrage.

Retko posednute nestrukturirane matrice su matrice sa najvećim brojem elemenata jednakim nuli i rasporedom nenultih unosa koji se ne poklapaju ni sa jednim od specifičnih modela matrica. Postoji puno različitih šema skladištenja koje smanjuju memorijski prostor i opremu za izračunavanje, skladištenjem i izvođenjem aritmetičkih operacija samo sa nenultim

elementima. Najjednostavnija stuktura retko posednute matrice za skladištenje je *koordinativni format (COO)*[6], gde je matrica uskladištena u vidu tri vektora. Umesto skladištenja celokupne matrice skladište se koordinate koje predstavljaju redove i kolone, sa odgovarajućim nenultim vrednostima. Na ovaj način štedimo memorijske resurse. Prvi vektor je vektor koji skladišti indeks reda za nenulte elemente. Drugi vektor u *COO format*-u je vektor koji skladišti indekse kolona odgovarajućih nenultih elemenata. Treći vektor čine nenulti elementi matrice.

Primer 2.2.1. U ovom primeru *matrix_A* je test matrica iz [70], *matrix_B* je simetrična pozitivno definisana i *matrix_C* je retko posednuta nestrukturirana matrica.

$$A=\begin{bmatrix} 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\ 10 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\ 9 & 9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\ 8 & 8 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\ 7 & 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 \\ 6 & 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & -1 \end{bmatrix}$$

$$B=\begin{bmatrix} 282 & -11 & -206 & -39 & 84 & 94 \\ -11 & 241 & -80 & 129 & 121 & -86 \\ -206 & -80 & 306 & 4 & -113 & 2 \\ -39 & 129 & 4 & 394 & -19 & -219 \\ 84 & 121 & -113 & -19 & 119 & 15 \\ 94 & -86 & 2 & -219 & 15 & 184 \end{bmatrix}$$

$$C=\begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

SQL iskaz `CREATE TABLE` za kreiranje strukture podataka i tabela u *MySQL* bazi podataka sastoji od sledećih delova:

- ime tabele (*matrices_in* i *matrices_out*) za sprovođenje fundamentalnih operacija nad maricama i izračunavanje generalisanih inverza,
- lista polja koja sadrži atributivna imena, tipove i modifikatore i
- lista ključeva.

Struktura tabela baze podataka za ulazne matrice i rezultate obrade je data sledećim kodom:

```
CREATE TABLE matrices_in (
  id_i n int (11) NOT NULL auto_increment,
  elements_ in longtext NOT NULL,
  dimension t inytext NOT NULL,
  test tinytext NOT NULL,
  sparse tinytext NOT NULL,
  PRIMARY KEY (id_ in)) TYPE=InoDB;

CREATE TABLE matrices_out (
  Id_out int (11) NOT NULL auto_increment,
  element s_out long text NOT NULL,
  operation tinytext NOT NULL,
  matrix_I int (11) NOT NULL default '0',
  matrix_II int (11) NOT NULL default '0',
  matrix_III int (11) NOT NULL default '0',
  r tinyint (4) NOT NULL default '0',s tinyint (4) NOT NULL default '0',
  p tinyint (4) NOT NULL default '0',q tinyint (4) NOT NULL default '0',
  PRIMARY KEY ( id_out )) TYPE=InoDB;
```

Polja korišćena u tabelama baze podataka su:

- *id_in* ili *id_out*: identifikacioni broj definisan kao auto_increment,
- *elements_in* ili *elements_out*: string koji sadrži elemente matrice odvojene zarezima čija se dužina kreće u intervalu od $[0 - 2^{32}]$,
- *dimension*: string u formi $m \times n$ koji predstavlja dimenzije matrice,
- *test:* ime test matrice iz [70] ako je u pitanju test matrica,
- *sparse*: oznaka za retko posednutu matricu koja ima jednu od vrednosti 1, 2, 3 ili vrednost 0 ukoliko je gusta matrica.
- *operation:* definiše operacije nad matricama unosa,
- *matrix_I*, *matrix_II*, *matrix_III*: sadrži indekse matrica iz tabele *matrices_in* ili sadrži podrazumevanu vrednost 0, u cilju pronalaženja rešenja za odabranu operaciju i unete matrice i
- *r*, *s*, *p*, *q*: koeficijenti koji se koriste pri izračunavanju.

Tabela *matrices_in* iz baze podataka sadrži tri matrice iz Primera 2.2.1.

Tabela 2.2.1 Izgled tabele *matrix_in.*

id_i	elements_in	dimension	test	sparse
1	11,10,9,...,1,1,1,0,-1	11x10	A_10_1	0
2	282,-11,-206,...,2,-219,15,184	6x6	‘ ‘	0
3	0,0,0,1,2,2,4,4,9	10x10	’ ’	1
4	0,1,2,1,0,3,1,5,9	10x10	’ ’	2
5	1,2,3,2,1,4,4,8,2	10x10	’ ’	3

Matrica $A_{11\times10}$ iz [70] koja predstavlja test matricu je smeštena u prvom redu tabele. Reprezentacija matrice B, iz primera 2.2.1, je u drugom redu. U trećem, četvrtom i petom redu je predstavljena reprezentacija retko posednute matrice C, iz istog primera.

Tabela 2.2.2 Sadržaj tabele *matrices_out.*

id_out	elements_in	operation	matrix_I	matrix_II	matrix_III	s	p	q	r
1	1,-1,...,-0.25,-0.417	A^-1	1	0	0	0	0	0	0
2	1974,-77,...,1288	rA+sB	2	2	0	3	4	0	0
3	0,0,0,0,1,2,2,2,4,9	A*B	3	3	0	0	0	0	0
4	0,1,2,3,1,0,1,2,1,9	A*B	4	4	0	0	0	0	0
5	4,6,3,12,4,1,2,3,8,4	A*B	5	5	0	0	0	0	0
6	0.755,-0.156,... ,-0.2	AW	18	14	17	0	0	0	0

Pošto smo se na osnovu rezultata testiranja [58] odlučili da *MatrixDBMS* sadrži matrice u *R* formatu, da bi bio kompletan sistem bilo je potrebno testirati i *MySQL* tip podataka. Postoje tri ključne grupe *MySQL* tipova podataka: numerički tipovi, datum i vreme, i string (ili karakter) tip. Upoređivali smo različite podtipove podataka string *MySQL* tipa podataka: *blob, varchar i tekst*. Blob je string tip podataka koji se koristi za binarne podatke. Nasuprot blob tipu podataka text tip podataka nije osetljiv na mala i velika slova. Text je podtip podataka string *MySQL* tipa koji služi za skladištenje teksta. Varchar podtip funkcioniše kao char. U varchar (*M*) *M* označava dužinu polja, maksimalna dužina varchar tipa je 255 znakova [4]. Tabela 2.2.3 pokazuje tri nabrojana *MySQL* podtipa podataka string tipa i njihove moguće vrednosti. *M* je skraćenica za maksimalan broj prikazanih karaktera, i koristi se opciono.

Tabela 2.2.3 *MySQL* tipovi podataka.

MySQL tipovi podataka		
tipovi podataka	**veličina**	**opis**
varchar(M)	Up to M bytes	Dužina mora biti <= 255.
blob	Up to 64KB	blob je case-sensitive za sortitanje i upoređivanje;
text	Up to 64KB	text je case-insensitive.
longblob	Up to 4GB	longblob je case-sensitive za sortitanje i upoređivanje
longtext	Up to 4GB	longtext je case-insensitive.

Analiza brzine izvršavanja unosa i pretrage podataka kada imamo različite *MySQL* tipove podataka data je u [46]. Na osnovu te analize imamo prednosti blob tipa u odnosu na ostale *MySQL* tipove podataka.

2.3. Reprezentacija matrica na aplikacionom sloju

U cilju kreiranja najefikasnijeg *MatrixAL* modela na aplikacionom sloju troslojne web arhitekture, pri implementaciji klijent/server aplikacije, testirali smo tri različita načina matrične reprezentacije na aplikacionom sloju:

- jednodimenzionalni nizovi (*1D_niz*),
- dvodimenzionalni nizovi (*2D_niz*) i
- instanca klase - objekt (*objekat).*

2.3.1. Reprezentacija matrica u vidu 1-dimenzionalnog niza

Za razliku od promenljivih, koje čuvaju pojedinačne vrednosti, nizovi se koriste za čuvanje skupa ili sekvence vrednosti. Svaki niz se sastoji od dva dela, indeksa i elementa. Indeks niza se koristi za identifikaciju i pristup elementu.

PHP podržava dve vrsta nizova [34]:

- nizovi sa numeričkim indeksiranjem (vectors/ indexed) i
- asocijativni nizovi (na nekim platformama poznati su pod nazivom “*mape*”).

Takođe, nizovi mogu biti jednodimenzionalni ili multidimenzionalni. Nizovi se mogu koristiti na mnogo načina za brzo i efikasno čuvanje i organizovanje podataka. Niz u *PHP*-u je zapravo uređena mapa. Mapa vezuje vrednosti ključeva. Ovaj tip je optimizovan za nekoliko različitih rešenja i može se tretirati kao niz, lista (vektor), tabela, rečnik, kolekcija, stek, red itd. Elementi niza mogu da budu drugi nizovi, stabla i višedimenzionalni nizovi. Niz ne mora biti jednostavna lista ključeva i vrednosti; svaki element niza može da sadrži kao vrednost i drugi niz. Jednodimenzionalni numerički niz skladišti svaki element niza sa numeričkim indeksom. Za pristup podacima u jednodimenzionalnom nizu potrebno je imati podatke o nizu: ime i indeks.

2.3.2. Reprezentacija matrica u vidu 2-dimenzionalnog niza

U dvodimenzionalnom nizu svaki element je zapravo novi niz. Svaki ovakav niz ima dva elementa: indekse redova i kolona [34].

Primer 2.3.1. Matrična reprezentacija dvodimenzionalnog niza iz primera 2.2.1 je

```
$array_2D=array(array(11,10,9,8,7,6,5,4,3,2),
                array(10,10,9,8,7,6,5,4,3,2),
                ...,
                array(1,1,1,1,1,1,1,1,1-1));
```

Ovakvi nizovi, korisni su za rad sa podacima koji imaju matričnu ili tabelarnu formu. Jedna od olakšica koju je moguće koristiti prilikom rada sa višedimenzionalnim nizovima je `foreach()` petlja. Ova petlja sama vodi računa o veličini i pozicijama članova niza prilikom iteracije, pa je tako dobra za neke jednostavnije operacije. Sa druge strane, ovakav pristup smanjuje kontrolu nad samim članovima niza. Ova rešenja su praktična kada imamo fiksan broj članova podniza (u ovom slučaju to su kolone naše tabele). Međutim, ukoliko želimo da rukujemo nekim dinamičkim podacima, čija količina nije unapred poznata, ova rešenja nisu dobra.

Izbor alata kao što je *PHP* za rešavanje problema predstavlja veliki izazov u dizajniranju, implementiranju, testiranju i održavanju softvera. Reprezentacija matrica, na aplikacionom sloju, u vidu jednodimenzionalnog (*1D_niz*) i dvodimenzionalnog niza (*2D_niz*) pripada tehnikama proceduralnog programiranja.

2.3.3. Reprezentacija matrica sistemom objekta

Postoji veliki broj razloga zašto koristiti objektno-orijentisano programiranje. Neki od njih su:

- kod se lakše održava,
- kod sam sebe objašnjava i
- veća kontrola nad kodom.

Sisteme sa ugrađenom *OO* tehnologijom karakterišu visoka brzina izvršavanja i ograničenja memorije. Osnovna razlika pristupa objektno-orijentisanog programiranja (*OOP*) u odnosu na proceduralno programiranje (*PP*) je da su podaci i kod integrisani u jednu celinu - objekat. Objektno orijentisano aplikacije su obično podeljene na veći broj objekata koji međusobno komuniciraju. Svaki objekat se obično odnosi na jedan problem, koji je samostalan i ima gomilu svojstava i metoda.

Zbog popularnosti *OO* programiranja, ceo objektni model je potpuno redizajniran u *PHP5* [34], na taj način što mu je dodat veliki broj karakteristika.

Klase predstavljaju opštu kategoriju u objektno-orijentisanom programiranju, koja opisuje grupu posebnih elemenata koji se nazivaju objekti i nalaze se unutar grupe. Klasa je opisni element, i u programiranju služi za definisanje skupa atributa ili skupa akcija (funkcija) koje su dostupne drugim delovima programa, a karakteristični su za sve članove jedne klase. Definisanje klasa u objektno-orijentisanom programiranju je vrlo slično definisanju tipova podataka u strukturnom programiranju.

Objekat predstavlja i instancu klase. Za kreiranje objekata koji pripadaju klasi koristi se ključna reč `new`. Kao i druge vrednosti u promenjivima, objekti mogu da se prenose, npr. kao parametri u funkcijama. U našem slučaju smo kao parametre funkcija prenosili objekte, odnosno instance klase koji predstavljaju matrice. Objekti nam pomažu da razdvojimo različite logičke komponente aplikacije. Možemo odvojiti one delove koda koji pišu *HTML* kod (prezentaciona logika) od delova koje izvode manipulaciju podacima (logika obrade). Sve ovo doprinosi jasnijem dizajnu, boljoj modularnosti i većoj fleksibilnosti.

Centralno mesto u našoj aplikaciji pripada klasi Matrix koja sadrži promenljive i metode za kreiranje matrice koja sadrži elemente u obliku `2D_niz` i broj redova i kolona, kao i metode za sprovođenje fundamentalnih operacija nad matricama i izračunavanje generalisanih inverza, kao i triangulacije konveksnih poligona i digitalne obrade slike.

Nakon definisanja klase Matrix, moguće je napraviti bilo koji broj objekata te klase. Posle imena klase sledi deklaracija niza promenljivih. Deklaracija promenljivih počinje sa posebnom ugrađenom formom `var`, a zatim sledi konvencionalno `$ime_promenljive`. Promenljive mogu imati početnu zadatu konstantnu vrednost.

Svu neophodnu inicijalizaciju vrši konstruktor klase. On predstavlja funkciju koja determiniše šta će se desiti svaki put kad se kreira instanca klase, objekat. Konstruktor klase je jednostavno matoda koji ima isto ime kao i klasa ili joj ime počinje sa _. Konstruktor se u *PHP*-u pojavio u verziji 4.2. U našem slučaju konstruktor klase `Matrix` je `Matrix()`.
Objekti mogu da sadrže podatke, kao što su elementi matrice, broj redova i kolona, a klasa definiše koja će svojstva instance klase imati. Klasu definišu skup svojstava i ponašanja. Podaci koji se dodele instanci klase svojstveni su samo za nju. Metod je funkcija koja je član klase a implementira se tako što se definiše unutar klase. Nakon toga se metod može pozvati unutar klase pomoću operatora −>, odnosno poziv izgleda kao objekat−>metoda. Da bi manipulisali objektom, koji predstavlja instancu klase, moramo ga prvo kreirati a zatim

komunicirati sa njim putem poruka. Svojstva objekta postoje dogod postoji i objekat, za razliku od ostalih nosioca vrednosti promenljive: funkcija, promenljiva i sesija.

Klase i objekte smo koristili pri implementaciji matričnih operacija, a rezultati testiranja i superiornost klase u odnosu na skup funkcija su predstavljeni u sekciji sa testiranjem.

Primer 2.3.2. Inicijalizacija objekta vrši se na sledeći način

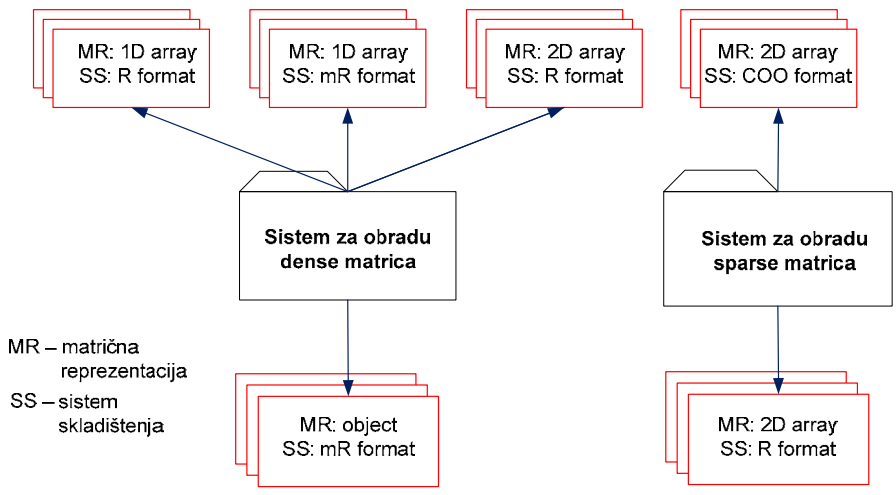
```
$matrica=new matrix($array_2D,$m,$n);.
```

`$array_2D` je dvodimenzionalni niz, iz Primera 2.2.1., koji predstavlja elemente matrice, `$m` predstavlja broj redova, a `$n` predstavlja broj kolona.

Na osnovu sprovedenog testiranja, čiji su rezultati prikazani u četvrtoj oblasti utvrdili smo da *MatrixAL* model na aplikacionom sloju treba da funkcioniše na bazi objekt formata, koji sadrži elemente matrice, sa tačno utvrđenim pozicijama i broj redova i kolona.

2.4. Korelacije sistema skladištenja i sistema reprezentacije matrica

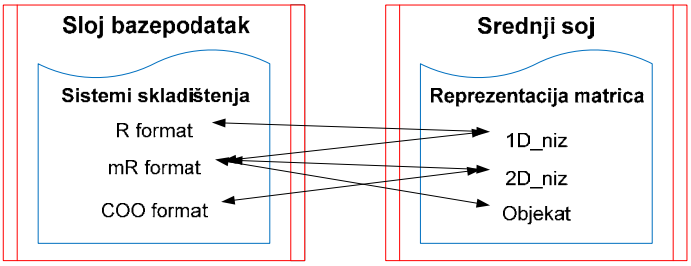
Kako bi utvrdili najefikasniji način funksionisanja *MatrixDBMS* i *MatrixAL* sistema upoređivali smo različite metode za izračunavanje $A^{\dagger}_{MN}$, $A^{\dagger}$ i fundamentalnih matričnih operacija u zavisnosti od relacije sistema za skladištenje, na sloju baze podataka, i matričnih reprezenrtacija, na aplikacionom sloju. Testirali smo 3 formata skladištenja matrica u bazi podataka, *R format*, *mR format* i *COO format* (samo za retko posednute matrice).



Slika 2.4.1 Sistemi za izračunavanja nad gustim i retko posednutim maricama.

Reprezentacije matrica na aplikacionom sloju date su u sledećim formama, `1D_niz, 2D_niz i objekat`. Testiranje smo obavili na gustim i retko posednutim matricama. Za guste matrice koristimo sisteme za izračunavanja, kao što je ilustrovano na slici 2.4.1 a), dok za retko posednute matrice koristimo sisteme izračunavanja, kao što je prikazano na slici 2.4.1 b).

Da bi ubrzali pretragu uskladištenih matrica i izračunavanje pseudo inverza $A^{\dagger}_{MN}$ i $A^{\dagger}$ pokazujemo korelaciju između MatrixDBMS i MatrixAL modela za skladištenje na slici 2.4.2.



Slika 2.4.2 Relacije između sistema za skladištenje i matričnih reprezentacija.

Implementacija proizvoljne operacije nad matricama u *Cassandra* bazi i sa *OOP* pristupom data je sledećim algoritmom.

Algoritam 2.4.1. OOP + Cassandra pristup izračunavanju težinskog MP inverza

Zahtev: Unos matrice

1: *upload fajla*
2: $getConection()$: *Funkcija A.3.1*
3: $\$exist\_in = getColumnMatrixIn()$, *Funkcija A.3.3*
4: **if** $\$exist\_in == TRUE$ **then**
5: $\$exist\_out = getColumnMatrixOut()$, *Funkcija A.3.5*
6: **if** $\$exist\_out == TRUE$ **then**
7: **go to** Step 16:
8: **else**
9: **go to** Step 14:
10: **end if**
11: **else**
12: $InsertColumnMatrixIn()$, *Funkcija A.3.2* **go to** Step 14:
13: **end if**
14: $\$matrix = new\ Matrix(\ );\ \$matrix \rightarrow WeightedInverse(\ )$
15: $InsertColumnMatrixOut()$, *Funkcija A.3.4*
16: **return** $displayColumnMatrixOut()$, *Funkcija A.3.6*

Na osnovu rezultata testiranja predstavljenih u narednom odeljku korisnički definisane funkcije za obe vrste, guste i retko posednute matrice, su implementirane koristeći instance klasa tj. objekte. *UDF* za retko posednute matrice su adaptirane *COO formatu* skladištenja, nasuprot korisnički definisanim funkcijama za guste matrice, koje su adaptirane *R format.* Sledeći algoritam sadrži korake za izračunavanje težinskog MP inverza koristeći *OO* pristup i *Cassandra DBMS*.

2.5. Matrična algebra u MySQL okruženju

U ovom odeljku je predstavljena implementacija operacija matrične algebre korišćenjem procedura *MySQL* baze podataka, bez korišćenja aplikacione logike. *MySQL* procedure se razlikuju od korisnički definisanih funkcija (*UDF*) po tome što se oslanjaju samo na sloj baze podataka u troslojnoj web arhitekturi. Zbog rezultata prikazanih u Tabeli 2.5.8., koji ukazuju na slabije *CPU* vreme izvršavanja matričnih operacija implementiranih korišćenjem *MySQL* procedura u odnosu na *UDF*, nismo implementirali složenije matrične operacije kao ni Algoritame 2.1.2 i Algoritme 2.1.4.

2.5.1. Osnovne matrične operacije i MySQL

Za implementaciju i testiranje *MySQL* procedura za matrična izračunavanja koriste se dve tabele `matrix(ime,i,j,val)` i `matrix_inf(ime,broj_vrsta,broj_kolona)`. U tabeli `matrix` nalaze se samo ona polja koja su različita od nule, dok tabela `matrix_temp` privremeno čuva podatke o matrici. Operacije sabiranja, oduzimanja i množenja matrica A i B su implementirane pomoću kursora složenosti, $O(a+b)$, $O(a+b)$, $O(a*b)$, gde a i b predstavljaju broj nenultih elemenata u ovim matricama što je optimalno za retko posednute matrice (u odnosu na $O(n^2)$ i $O(n^3)$), dok kod gustih matrica dolazi do gubitka u vremenu kad je u pitanju množenje.

Strukture tabela `` `matrix` `` i `` `matrix_temp` `` su data sledećim kodom:

```
CREATE TABLE `matrix_temp`(
  `name` varchar(10) NOT NULL,
  `row` int(11) DEFAULT NULL,
  `column` int(11) DEFAULT NULL,
  PRIMARY KEY (`matrix_name`),
  UNIQUE KEY `matrix_name_UNIQUE`(`matrix_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `matrix`(
  `matrix_name` varchar(10) NOT NULL,
  `row_ind` int(11) NOT NULL,
  `col_ind` int(11) NOT NULL,
  `value` double DEFAULT NULL,
  PRIMARY KEY (`matrix_name`,`row_ind`,`col_ind`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Polja korišćena u tabelama baze podataka su:

- *name*: ime privremene matrice,
- *row*: indeks reda privremene matrice,
- *column*: indeks kolone privremene matrice,
- *matrix_name*: ime matrice,
- *row_ind*: indeks reda matrice,
- *col_ind*: indeks kolone matrice i
- *value*: vrednost elementa (i,j).

Funkcija 2.5.1 *MySQL* procedura `` `matrix_multiplication()` `` služi za množenje matrica koje zadovoljavaju uslov da je broj kolona prve jednak broju redova druge. Množenje matrica je implementirano tako što se prolazi kroz sve nenulte elemente a_{ik} matrice A, a zatim se od svih nenultih elemenata matrice B posmatraju samo oni oblika b_{kj}. Ukoliko u tabeli '`matrix`' postoji red `(c,i,j,val)`, vrednost val transformišemo u vrednost $val + a\_ik * b\_kj$, inače kreiramo odgovarajući red. Složenost je $O(a * b)$, gde su a, b brojevi nenultih elemenata u matricama A i B, što je za retke matrice bolje od standardnog $O(n^3)$ algoritma.

```
delimiter $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `matrix_multiplication`(
    IN A varchar(10),
    IN B varchar(10),
    IN C varchar(10),
    OUT message varchar(200))
    MODIFIES SQL DATA
BEGIN
-- deklaracija promenljivih
DECLARE cursor_end, a_end, b_end BOOLEAN DEFAULT false;
DECLARE xa, ya, xb, yb INT;
DECLARE ma, mb, na, nb INT;
DECLARE val_a, val_b DOUBLE;
-- deklaracija kursora
DECLARE cur_A CURSOR FOR
    SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
    FROM matrix m
    WHERE m.matrix_name = A
    ORDER BY m.row_ind, m.col_ind;
```

```
-- deklaracija hendlera
DECLARE continue HANDLER FOR SQLSTATE '02000' SET cursor_end = true;
SELECT m.row INTO ma FROM matrix_temp m WHERE m.name = A;
SELECT m.column INTO na FROM matrix_temp m WHERE m.name = A;
SELECT m.row INTO mb FROM matrix_temp m WHERE m.name = B;
SELECT m.column INTO nb FROM matrix_temp m WHERE m.name = B;
--mnozenje moguce ako je na = mb
IF (na = mb) THEN
    INSERT INTO matrix_temp(name,row,column)VALUES(C,ma,nb);
    OPEN cur_A;
    FETCH cur_A INTO xa, ya, val_a;
    WHILE (NOT a_end) DO
    BEGIN
       DECLARE cur_B CURSOR FOR
       SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
       FROM matrix m
       WHERE m.matrix_name = B
       ORDER BY m.row_ind, m.col_ind;
       SET b_end = false;
       OPEN cur_B;
       FETCH cur_B INTO xb, yb, val_b;
       WHILE (NOT b_end) DO
          IF (ya = xb) THEN
              IF (EXISTS(SELECT * FROM matrix WHERE (matrix_name = C AND
                    row_ind = xa AND col_ind = yb))) THEN
                    UPDATE matrix
                  SET value = value + val_a * val_b
                  WHERE (matrix_name = C AND row_ind = xa AND col_ind = yb);
              ELSE
                  INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
              VALUES(C,xa,yb,val_a*val_b);
              END IF;
          END IF;
          FETCH cur_B INTO xb, yb, val_b;
          SET b_end = cursor_end;
          SET cursor_end = false;
      END WHILE;
      CLOSE cur_B;
      FETCH cur_A INTO xa, ya, val_a;
      SET a_end = cursor_end;
      SET cursor_end = false;
   END;
   END WHILE;
   CLOSE cur_A;
   SET message = 'Mnozenje uspesno';
ELSE
   SET message = 'Matrice se ne mogu mnoziti';
END IF;
SELECT message;
END$$
```

Funkcija 2.5.2 Procedura `` `matrix_scalar_mult()` `` se koristi za množenje matrica skalarom. To je standardno kopiranje vrednosti pomnožene koeficijentom s. U pitanju je složenost O (broj nenultih elemanata).

```
delimiter $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `matrix_scalar_mult`(
   IN A varchar(10),
   IN s DOUBLE,
   IN B varchar(10),
   OUT message varchar(200))
   MODIFIES SQL DATA
BEGIN
-- deklaracija promenljivih
DECLARE cursor_end BOOLEAN DEFAULT false;
DECLARE x, y INT;
DECLARE m, n INT;
DECLARE val DOUBLE;
-- deklaracija kursora
DECLARE cur_A CURSOR FOR
   SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
   FROM matrix m
   WHERE m.matrix_name = A;
-- deklaracija hendlera
DECLARE continue HANDLER FOR SQLSTATE '02000' SET cursor_end = true;
SELECT m.row INTO m FROM matrix_temp m WHERE m.name = A;
SELECT m.column INTO n FROM matrix_temp m WHERE m.name = A;
INSERT INTO matrix_temp (name,row,column)VALUES(B,m,n);
IF (s <> 0) THEN
  OPEN cur_A;
  FETCH cur_A INTO x, y, val;
  WHILE (NOT cursor_end) DO
    INSERT INTO matrix (matrix_name,row_ind,col_ind,value)
    VALUES(B,x,y,s*val);
    FETCH cur_A INTO x, y, val;
  END WHILE;
  CLOSE cur_A;
END IF;
SET message = 'Mnozenje skalarom uspesno';
SELECT message;
END$$
```

Funkcija 2.5.3 Procedura `` `matrix_addition()` `` se koristi za sabiranje matrica u *MySQL* okruženju, koje su istih dimenzija. Optimizovano za retko posednute matrice, radi u složenosti $O(a+b)$ + složenost formiranja kursora sa sortiranim vrednostima (po vrstama i kolonama), gde a i b predstavljaju brojeve nenultih elemenata matrica A i B. Umesto da koristimo dve for

petlje (složenost n^2) prolazimo paralelno kroz dva sortirana niza i matrici C dodajemo nenulte elemente.

```
delimiter $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `matrix_addition`(
   IN A varchar(10),IN B varchar(10),IN C varchar(10),
   OUT message varchar(200))
   MODIFIES SQL DATA
BEGIN
--deklaracija promenljivih
DECLARE cursor_end, a_end, b_end BOOLEAN DEFAULT false;
DECLARE xa, ya, xb, yb INT;
DECLARE ma, mb, na, nb INT;
DECLARE val_a, val_b DOUBLE;
DECLARE next_fetch INT; -- 0 za A, 1 za B, 2 za oba
-- deklaracija kursora
DECLARE cur_A CURSOR FOR
   SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
   FROM matrix m
   WHERE m.matrix_name = A
   ORDER BY m.row_ind, m.col_ind;
   DECLARE cur_B CURSOR FOR
     SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
     FROM matrix m
     WHERE m.matrix_name = B
     ORDER BY m.row_ind, m.col_ind;
-- deklaracija hendlera
DECLARE continue HANDLER FOR SQLSTATE '02000' SET cursor_end = true;
SELECT m.row INTO ma FROM matrix_temp m WHERE m.name = A;
SELECT m.column INTO na FROM matrix_temp m WHERE m.name = A;
SELECT m.row INTO mb FROM matrix_temp m WHERE m.name = B;
SELECT m.column INTO nb FROM matrix_temp m WHERE m.name = B;
-- sabiranje moguce ako su matrice istih dimenzija
IF (ma = mb AND na = nb) THEN
    INSERT INTO matrix_temp(name,row,column)
    VALUES(C,ma,na);
    OPEN cur_A;
    OPEN cur_B;
    FETCH cur_A INTO xa, ya, val_a;
    FETCH cur_B INTO xb, yb, val_b;
    WHILE ((NOT a_end) OR (NOT b_end)) DO
       IF (xa = xb AND ya = yb) THEN
          IF (val_a + val_b <> 0) THEN
             INSERT INTO matrix (matrix_name,row_ind,col_ind,value)
              VALUES(C,xa,ya,val_a+val_b);
          END IF;
          SET next_fetch = 2;
      END IF;
      IF((xa<xb)OR(xa=xb AND ya<yb))THEN
          INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
VALUES(C,xa,ya,val_a);
```

```
          SET next_fetch = 0;
      END IF;
      IF((xb < xa) OR (xb = xa AND yb < ya)) THEN
          INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
          VALUES(C,xb,yb,val_b);
          SET next_fetch = 1;
        END IF;
        IF(next_fetch = 0 OR next_fetch = 2) THEN
          FETCH cur_A INTO xa, ya, val_a;
          IF(cursor_end) THEN
             SET xa = na + 1; SET ya = ma + 1;
             SET a_end = true; SET cursor_end = false;
          END IF;
        END IF;
        IF(next_fetch = 1 OR next_fetch = 2) THEN
          FETCH cur_B INTO xb, yb, val_b;
          IF(cursor_end) THEN
             SET xb = nb + 1; SET yb = mb + 1;
             SET b_end = true; SET cursor_end = false;
          END IF;
        END IF;
     END WHILE;
     CLOSE cur_A;
     CLOSE cur_B;
     SET message = 'Sabiranje uspesno';
   ELSE
     SET message = 'Dimenzije matrice se ne slazu';
   END IF;
   SELECT message;
END$$
```

Funkcija 2.5.4 Procedura za oduzimanje matrica istih dimenzija u *MySQL*-u.

```
delimiter $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `matrix_substraction`(
   IN A varchar(10),
   IN B varchar(10),
   IN C varchar(10),
   OUT message varchar(200))
   MODIFIES SQL DATA
BEGIN
-- deklaracija promenljivih
DECLARE cursor_end, a_end, b_end BOOLEAN DEFAULT false;
DECLARE xa, ya, xb, yb INT;
DECLARE ma, mb, na, nb INT;
DECLARE val_a, val_b DOUBLE;
DECLARE next_fetch INT; -- 0 za A, 1 za B, 2 za oba
-- deklaracija kursora
DECLARE cur_A CURSOR FOR
SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
   FROM matrix m
   WHERE m.matrix_name = A
```

```
   ORDER BY m.row_ind, m.col_ind;
   DECLARE cur_B CURSOR FOR
   SELECT m.row_ind AS x, m.col_ind AS y, m.value AS val
   FROM matrix m
   WHERE m.matrix_name = B
   ORDER BY m.row_ind, m.col_ind;
-- deklaracija hendlera
DECLARE continue HANDLER FOR SQLSTATE '02000' SET cursor_end = true;
SELECT m.row INTO ma FROM matrix_temp m WHERE m.name = A;
SELECT m.column INTO na FROM matrix_temp m WHERE m.name = A;
SELECT m.row INTO mb FROM matrix_temp m WHERE m.name = B;
SELECT m.column INTO nb FROM matrix_temp m WHERE m.name = B;
-- sabiranje moguce ako su matrice istih dimenzija
IF (ma = mb AND na = nb) THEN
   INSERT INTO
   matrix_temp(name,row,column)VALUES(C,ma,na);
   OPEN cur_A;OPEN cur_B;
   FETCH cur_A INTO xa, ya, val_a;
   FETCH cur_B INTO xb, yb, val_b;
   WHILE ((NOT a_end) OR (NOT b_end)) DO
     IF (xa = xb AND ya = yb) THEN
        IF (val_a - val_b <> 0) THEN
           INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
       VALUES (C, xa, ya, val_a - val_b);
        END IF;
        SET next_fetch = 2;
     END IF;
     IF ((xa < xb) OR (xa = xb AND ya < yb)) THEN
        INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
VALUES (C, xa, ya, val_a);
        SET next_fetch = 0;
     END IF;
     IF ((xb < xa) OR (xb = xa AND yb < ya)) THEN
         INSERT INTO matrix (matrix_name, row_ind, col_ind, value)
VALUES (C, xb, yb, -val_b);
         SET next_fetch = 1;
     END IF;
     IF(next_fetch = 0 OR next_fetch = 2) THEN
        FETCH cur_A INTO xa, ya, val_a;
        IF (cursor_end) THEN
           SET xa = na + 1; SET ya = ma + 1;
           SET a_end = true; SET cursor_end = false;
        END IF;
     END IF;
     IF (next_fetch = 1 OR next_fetch = 2) THEN
        FETCH cur_B INTO xb, yb, val_b;
        IF (cursor_end) THEN
           SET xb = nb + 1; SET yb = mb + 1;
           SET b_end = true; SET cursor_end = false;
        END IF;
      END IF;
```

```
   END WHILE;
   CLOSE cur_A;
   CLOSE cur_B;
   SET message = 'Oduzimanje uspesno';
   ELSE
     SET message = 'Dimenzije matrice se ne slazu';
   END IF;
SELECT message;
END$$
```

Funkcija 2.5.5 Procedura za transponovanje matrica u *MySQL*-u. Prikupljanje svih vrednosti iz tabele '`matrix`', zamena mesta indeksa i update tebele `'matrix_temp'`. Složenost je O (broj nenultih elemenata).

```
delimiter $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `matrix_transpose`(
   IN A varchar(10),OUT message varchar(200))
   MODIFIES SQL DATA
BEGIN
-- deklaracija promenljivih
DECLARE cursor_end BOOLEAN DEFAULT false;
DECLARE x, y INT;DECLARE m, n INT;
-- deklaracija kursora
DECLARE cur_A CURSOR FOR
SELECT m.row_ind AS x, m.col_ind AS y
FROM matrix m
WHERE m.matrix_name = A;
-- deklaracija hendlera
DECLARE continue HANDLER FOR SQLSTATE '02000' SET cursor_end=true;
SELECT m.row INTO m FROM matrix_temp m WHERE m.name = A;
SELECT m.column INTO n FROM matrix_temp m WHERE m.name = A;
IF (m <> n) THEN
  UPDATE matrix_temp
  SET row = m, column = n
  WHERE name = A;
END IF;
OPEN cur_A;
FETCH cur_A INTO x, y;
WHILE (NOT cursor_end) DO
  UPDATE matrix
  SET row_ind = y, col_ind = x
  WHERE (matrix_name = A AND row_ind = x AND col_ind = y);
  FETCH cur_A INTO x, y;
END WHILE;
CLOSE cur_A;
sSET message = 'Transponovanje uspesno';
ELECT message;
END$$
```

Primer 2.5.1. Množenje matrica B i B iz Primera 2.2.1. Izgled tabele `matrix` sa elementima matrice B je dat na slici 2.5.1.

matrix_name	row_ind	col_ind	value
matrix	0	0	282
matrix	0	1	-11
matrix	0	2	-206
matrix	0	3	-39
matrix	0	4	84
matrix	0	5	-4
matrix	1	0	-11
matrix	1	1	241
matrix	1	2	-80
matrix	1	3	129
matrix	1	4	121
matrix	1	5	-86

Slika 2.5.1 Fragment matrice B uskladištene u tabeli matrix.

Za aktiviranje uskladištene *MySQL* procedure se koristi `CALL` funkcija. Poziv *MySQL* procedure `matrix_multiplication()` se vrši na sledeći način:

```
CALL matrix_multiplication();
```

Rezultat dobijen primenom `matrix_multiplication()` procedure se smešta u tabelu `matrix`, koja je data na slici 2.5.4.

matrix_name	row_ind	col_ind	value
matrix_mul	0	0	139494
matrix_mul	0	1	7776
matrix_mul	0	2	-129708
matrix_mul	0	3	-50789
matrix_mul	0	4	57782
matrix_mul	0	5	54139
matrix_mul	1	0	7776
matrix_mul	1	1	103280
matrix_mul	1	2	-54823

Slika 2.5.2 Fragment matrice koja predstavlja rezultat množenja dve matrice.

2.5.2. Sistemi linearnih jednačina i MySQL

MySQL okruženje omogućava direktno rešavanje sistema linearnih jednačina nakon skladištenja implementiranih procedura [78]. Broj nepoznatih u jednačinama može biti varijabilan. Jasno je da broj jednačina mora biti jednak broju promenljivih koje figurišu u jednačini. Prednosti i mane ovog pristupa u odnosu na *UDF* u proceduralnom programiranju su date u poslednjem poglavlju. Za rešavanje sistema linearnih jednačina korišćenjem *MySQL* okruženja neophodno je kreirati dve tabele, jednu za problem i drugu za rešenje [78].

Tabela koja sadrži koeficijente uz nepoznate i slobodne članove.

```
CREATE TABLE coefficient_matrix(
  column integer NOT NULL DEFAULT '0',
  row integer NOT NULL DEFAULT '0',
  coefficient double DEFAULT NULL,
  PRIMARY KEY(col,row));
```

Tabela za skladištenje rezultata obrade.

```
CREATE TABLE result_eq(
  row_eq integer NOT NULL DEFAULT '0',
  solve double DEFAULT NULL,
  PRIMARY KEY (row));
```

Polja korišćena u tabelama baze podataka su:

- *row*: sadrži indeks reda matrice koja je obrazovana od koeficijenata uz nepoznate i slobodnih članova,
- *column*: sadrži indeks kolone matrice koja je obrazovana od koeficijenata uz nepoznate i slobodnih članova,
- *coefficient*: koeficijenti,
- *row_eq*: redni broj nepoznatih i
- *solve*: rešenje linearne jednačine.

Primer 2.5.2. Rešavanje sistema linearnih jednačina sa tri nepoznate korišćenjem *MySQL* procedure [78].

$$2x + 2y + 4z = 18$$

$$x + 3y - 2z = 1$$

$$3x + y + 3z = 14$$

Izgled dela tabele `coefficient_matrix` u *MySQL* bazi podataka je kao na slici 2.5.3.

col	row	coefficient
0	0	2
0	1	1
0	2	3
1	0	2
1	1	3
1	2	1
2	0	4
2	1	-2

Slika 2.5.3 Koeficijenti i slobodni članovi sistema linearnih jednačina.

Rezultat dobijen primenom procedure [78] se smešta u tabelu `result_eq`, koja je data na slici 2.5.4.

row	coefficient
0	0.999999999999997
1	2
2	3

Slika 2.5.4 Rešenje sistema linearnih jednačina u tabeli result_eq u MySQL bazi podataka.

2.6. Rezultati testiranja i primeri

Kao što smo već naveli, aplikacija je implementirana poštujući uslove troslojne web arhitekture.

Testiranje je obavljeno na serveru sa sledećim karakteristikama: OS *Windows edition*: *Windows Vista(MT) Ultimatum*, Procesor: *Intel (R) Pentium Dual T3200@2000GHz*, Memorija: *RAM 2940 MB*, Tip sistema: 32-bitni Operativni sistem, Free softver: *PHP 5.2.5, MySQL 5.0.45, PhpMyAdmin 2.11.2.1, Apach Cassandra, Thrift 1.0.6.* Testiranje je urađeno na lokalnoj mašini i u bežičnoj mreži. Pristupili smo web serveru preko mreže u infrastructure modu sa *acces point*-om.

Primer 2.5.1. Posmatrajmo test matricu iz [70], kad je *a=1*,

$$A=\begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 3 & 4 & 6 & 2 \\ 1 & 3 & 4 & 6 & 2 & 2 & 3 & 4 & 5 & 3 \\ 2 & 3 & 4 & 5 & 3 & 3 & 4 & 5 & 6 & 4 \\ 3 & 4 & 5 & 6 & 4 & 4 & 5 & 6 & 7 & 6 \\ 4 & 5 & 6 & 7 & 6 & 6 & 6 & 7 & 7 & 8 \\ 6 & 6 & 7 & 7 & 8 & 1 & 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 1 & 3 & 4 & 6 & 2 & 1 & 2 \\ 4 & 6 & 2 & 2 & 3 & 4 & 5 & 3 & 1 & 3 \\ 4 & 5 & 3 & 3 & 4 & 5 & 6 & 4 & 2 & 3 \\ 5 & 6 & 4 & 4 & 5 & 6 & 7 & 6 & 3 & 4 \\ 6 & 7 & 6 & 6 & 6 & 7 & 7 & 8 & 4 & 5 \end{bmatrix}$$

i slučajno generisane pozitivno definisane matrice, odgovarajućih dimenzija, *L* i *M*

$$M(N)=\begin{bmatrix} 280 & -5 & -133 & -27 & -12 & -93 & -133 & -42 & 84 & 52 \\ -5 & 216 & -93 & -23 & 141 & -2 & -108 & -48 & 21 & 165 \\ -133 & -93 & 336 & 1 & 0 & 0 & 9 & -5 & 91 & 81 \\ -27 & -23 & 1 & 260 & -62 & -42 & -40 & 6 & 85 & -8 \\ -12 & 141 & 0 & -62 & 278 & -63 & -19 & -135 & 9 & 99 \\ -93 & -2 & 0 & -42 & -63 & 238 & 68 & -27 & -80 & -34 \\ -133 & -108 & 9 & -40 & -19 & 68 & 290 & 12 & -233 & -244 \\ -42 & -48 & -5 & 6 & -135 & -27 & 12 & 209 & -15 & -87 \\ 84 & 21 & 91 & 85 & 9 & -80 & -233 & -15 & 332 & 145 \\ 52 & 165 & 81 & -8 & 99 & -34 & -244 & -87 & 145 & 402 \end{bmatrix}$$

$$L(M)=\begin{bmatrix} 452 & 91 & -186 & -97 & -161 & 68 & 28 & 16 & 151 & 41 & -65 \\ 91 & 413 & -74 & -119 & -317 & -41 & -12 & 67 & 180 & -53 & 54 \\ -186 & -74 & 497 & -136 & 78 & -208 & -175 & -120 & 9 & -99 & 6 \\ -97 & -119 & -136 & 371 & 157 & 154 & 129 & 29 & -102 & -16 & -96 \\ -161 & -317 & 78 & 157 & 444 & 28 & -39 & -201 & -165 & 3 & 43 \\ 68 & -41 & -208 & 154 & 28 & 509 & 52 & 55 & 90 & 179 & -81 \\ 28 & -12 & -175 & 129 & -39 & 52 & 454 & -38 & -157 & 145 & 22 \\ 16 & 67 & -120 & 29 & -201 & 55 & -38 & 408 & -9 & -16 & -100 \\ 151 & 180 & 9 & -102 & -165 & 90 & -157 & -9 & 257 & -32 & 14 \\ 41 & -53 & -99 & -16 & 3 & 179 & 145 & -16 & -32 & 376 & -15 \\ -65 & 54 & 6 & -96 & 43 & -81 & 22 & -100 & 14 & -15 & 339 \end{bmatrix}$$

Generalisani LM inverz $A_{LM}^{\dagger}$ iz [61,62] i Težinski MP inverz $A_{MN}^{\dagger}$ iz [65] su identični [59].

$$A^{\dagger}_{LM(MN)} = \begin{bmatrix} 0.755 & -0.156 & -0.197 & 0.823 & 0.143 & 0.033 & 0.383 & -0.33 & 0.213 & -0.802 & 0.67 \\ -0.542 & -0.078 & 1.683 & -0.544 & -0.27 & 0.003 & -0.432 & 0.673 & 0.075 & 0.087 & -0.347 \\ 0.346 & -0.194 & -0.881 & 0.751 & -0.187 & 0.002 & 0.32 & -0.149 & 0.711 & -1.749 & 1.003 \\ -0.049 & 0.558 & -0.724 & 0.145 & 0.065 & 0.007 & 0.128 & -0.258 & -0.245 & 0.481 & -0.141 \\ -0.454 & -0.013 & 1.181 & -0.79 & 0.188 & 0.109 & -0.3 & 0.057 & -0.563 & 1.498 & -0.907 \\ -1.188 & -0.468 & 2.87 & -0.203 & -0.792 & -0.107 & 0.233 & 0.19 & 1.258 & -2.561 & 1.083 \\ 0.701 & 0.364 & -2.009 & 0.492 & 0.282 & 0.009 & 0.395 & -0.407 & -0.481 & 0.786 & -0.192 \\ 0.472 & 0.156 & -0.533 & -0.675 & 0.507 & -0.02 & -0.62 & -0.008 & -0.822 & 2.147 & -0.848 \\ -0.415 & -0.481 & 1.824 & -0.22 & -0.377 & 0.002 & 0.013 & 0.247 & 0.615 & -1.143 & 0.257 \\ 0.328 & 0.156 & -1.388 & 0.469 & 0.362 & -0.02 & 0.028 & -0.008 & -0.47 & 0.526 & -0.2 \end{bmatrix}$$

U slučaju da su matrice M i N date kao jedinične matrice odgovarajućih dimenzija tada se Težinski MP inverz $A^{\dagger}_{MN}$ transformiše u MP inverz $A^{\dagger}$.

$$A^{\dagger} = \begin{bmatrix} 0.294 & -0.169 & -1.511 & 1.415 & -0.391 & 0.041 & -0.04 & -0.26 & 0.454 & -0.264 & 0.23 \\ -0.067 & -0.074 & 1.227 & -1.064 & 0.23 & 0.001 & -0.192 & 0.649 & -0.103 & -0.191 & -0.102 \\ 0.227 & -0.179 & -0.692 & 0.705 & -0.214 & -0.009 & -0.254 & -0.238 & 1.514 & -1.319 & 0.449 \\ -0.165 & 0.547 & -0.657 & 0.376 & -0.116 & 0.015 & 0.179 & -0.196 & -0.456 & 0.531 & -0.104 \\ -0.297 & -0.01 & 1.035 & -0.972 & 0.359 & 0.108 & 0.238 & 0.044 & -1.065 & 0.938 & -0.366 \\ -0.008 & -0.474 & 1.663 & -1.319 & 0.352 & -0.103 & -0.428 & 0.224 & 1.83 & -1.844 & 0.414 \\ 0.062 & 0.368 & -1.359 & 1.081 & -0.33 & 0.006 & 0.426 & -0.434 & -0.442 & 0.712 & -0.156 \\ -0.081 & 0.158 & 0.029 & -0.144 & -0.034 & -0.021 & 0.087 & -0.019 & -1.499 & 1.448 & -0.138 \\ 0.195 & -0.484 & 1.198 & -0.791 & 0.211 & 0.005 & -0.19 & 0.268 & 0.765 & -0.906 & 0.049 \\ -0.081 & 0.158 & -0.971 & 0.856 & -0.034 & -0.021 & 0.087 & -0.019 & -0.499 & 0.448 & -0.138 \end{bmatrix}$$

Primer 2.5.2. Posmatrajmo sledeću test matricu sa jednom promenljivom, razmatranu u [70].

$$A = \begin{bmatrix} x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 & 0 \\ x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 & 0 \\ x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 & 0 \\ x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 & 0 \\ x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 & 0 \\ x^{11} & x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x & 1 \\ x^{12} & x^{11} & x^{10} & x^9 & x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x \end{bmatrix}$$

I neka su *L(M) i M(N)* jedinične matrice odgovarajućih dimenzija. Oba predstavljena algoritma daju sledeći MP inverz.

$$A^{\dagger}=\begin{bmatrix}
\frac{x}{x^2+1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{x}{x^2+1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & \frac{x}{x^2+1} & \frac{x}{x^2+1}
\end{bmatrix}$$

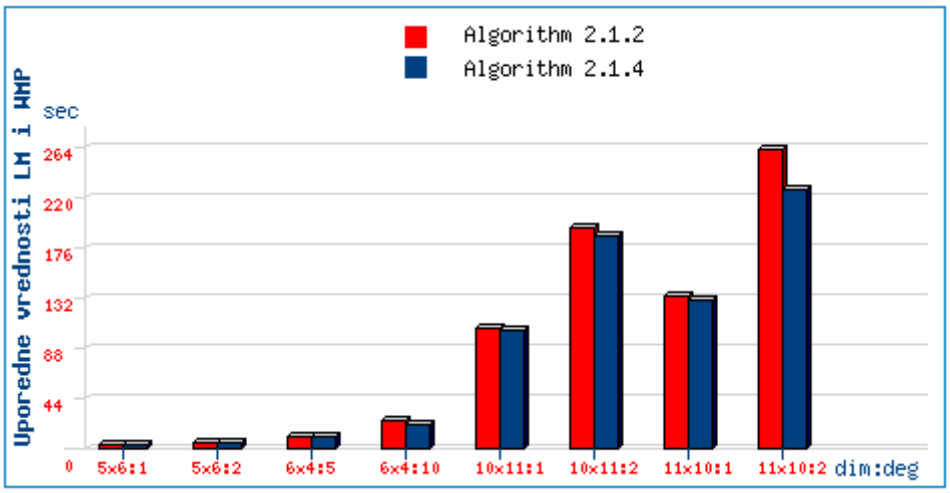
Primer 2.5.3. CPU vremena (u sekundama) neophodna za izračunavanje generalisanog *LM* i težinskog *MP* inverza [59] (predstavljenih u vidu Algoritma 2.1.4 i Algoritma 2.1.2, respektivno) su upoređivana u sledećoj tabeli. Predstavljeni rezultati su dobijeni testiranjem slučajno generisanih test matrica i pozitivno definisanih matrica $L_{m\times m}$ (odnosno $M_{m\times m}$) i $M_{n\times n}$(odnosno $N_{n\times n}$).

Tabela 2.5.1 Uporedne vrednosti Generalisanog LM inverza i Težinskog MP inverza.

mxn	**stepen**	**Algoritam 2.1.2**	**Algoritam 2.1.4**
5x6	1	2.265	2.235
5x6	2	4.079	4.079
6x4	5	9.969	9.625
6x4	10	23.328	21.11
10x11	1	104.109	103.484
10x11	2	192.734	186.297
11x10	1	133.469	130.125
11x10	2	261.357	227.047

Grafički prikaz je dat u nastavku. Svi grafikoni su implementirani u *PHP*-u. Detaljnije o tome u drugom odeljku treće glave.

Grafik 2.5.1 Uporedne vrednosti Generalisanog *LM* inverza i težinskog MP inverza.



Na osnovu rezultata testiranja iz [59] može se videti da su rezultati koje daje Algoritam 2.1.2 približno jednaki onima koje daje Algoritam 2.1.4 za sve slučajeve testiranja, pa je teorema 2.1.9 i eksperimentalno potvrđena.

Primer 2.5.4. U sledećoj tabeli upoređujemo vreme pretraživanja kada su matrice uskladištene u bazi podataka.

Tabela 2.5.2 CPU vreme za pretrage matrica dimenzija 70x70 u R i mR formatu.

broj matrica	R format	mR format
50	0.102	0.647
100	0.166	0.660
500	0.791	14.079
1000	1.901	28.650

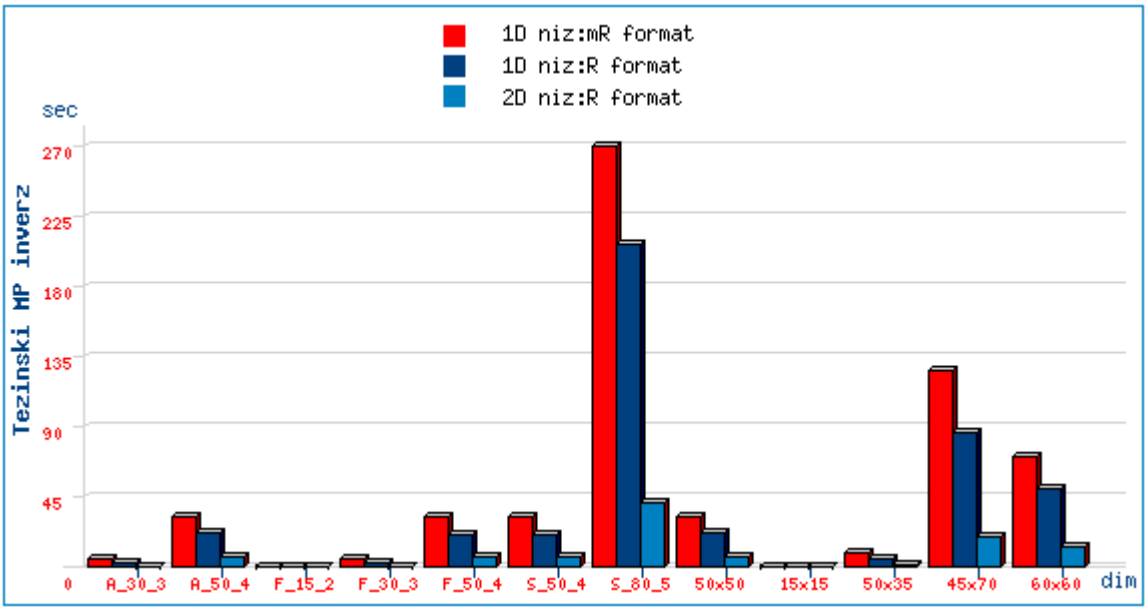
Ovo testiranje izvršeno je sa gustim matricama sa slučajno generalisanim elementima. *CPU* vreme pretrage je mnogo kraće kada je matrica uskladištena u bazi podataka u *R formatu*. Treba napomenuti da je pretraživanje test matrica iz [70] zasnovano na pretrazi po jedinstvenom imenu i ne zavisi od broja matrica uskladištenih u bazi podataka.

Primer 2.5.5. U sledećoj tabeli su predstavljena CPU vremena potrebna za izračunavanje pseudo inverza. Matrice su, na sloju aplikacione logike, predstavljene na dva načina: kao *1D_niz* sa aplikacionom logikom koja je adaptirana *mR formatu* i *R formatu*, ili kao *2D_niz* sa aplikacionom logikom adaptiranom *R formatu* sistema skladištenja.

Tabela 2.5.3 Težinski MP inverz $A^{\dagger}_{MN}$.

m × n	1D_niz: mR format	1D_niz: R format	2D_niz: R format
A_30_3	4.485	2.440	0.897
A_50_4	32.834	21.270	6.199
F_15_2	0.666	0.342	0.166
F_30_3	4.379	2.347	0.880
F_50_4	32.564	20.981	6.077
S_50_4	32.597	20.874	6.070
S_80_5	269.282	206.007	40.75
50x50	32.423	21.294	6.203
15x15	0.632	0.471	0.160
50x35	9.79	5.587	1.997
45x70	125.18	86.210	19.920
60x60	70.260	49.964	12.627

Grafik 2.5.2 Izračunavanje vremena za $A^{\dagger}_{MN}$ kada matrice ne postoje u bazi podataka.



Rezultati testiranja pokazuju da je najbolje vreme procesiranja postignuto u slučaju kada su matrice predstavljene u formi *2D_niza* [58].

Primer 2.5.6. Rezultati dobijeni prilikom izračunavanja pseudo inverza, u slučaju kada su, i kad nisu, ulazne matrice uskladištene u bazi podataka, su predstavljeni u sledećoj tabeli. Matrice su predstavljene u *2D_niz* matričnoj reprezentaciji, a sistem skladištenja je u *R formatu* [58].

Tabela 2.5.4 CPU vreme izračunavanja $\boldsymbol{A}_{\boldsymbol{MN}}^{\dagger}$ kada su matrice uskladištene u bazi i kad to nije slučaj.

m×n	**2D_niz:R format (uskladištena)**	**2D_niz:R format (nije uskladištena)**
20x20	0.043	0.309
30x30	0.051	2.105
45x45	0.062	4.665
50x50	0.070	6.729
60x60	0.075	12.675
70x70	0.079	26.614
80x80	0.096	43.809

Vreme potrebno za pretragu rešenja iz baze podataka je zanemarivo u odnosu na vreme potrebno za ponovno izračunavanje, naročito ako matrica ima veće dimenzije. U tabeli *matrices_in* u trenutku testiranja bilo je 10030 zapisa.

Primer 2.5.7. U sledećoj tabeli su predstavljeni rezultati dobijeni upoređivanjem vremena unosa podataka u *MySQL* bazu podataka kod koje figurišu različiti tipovi podataka [46].

Tabela 2.5.5 Unos podataka u DB sa različitim MySQL tipovima podataka.

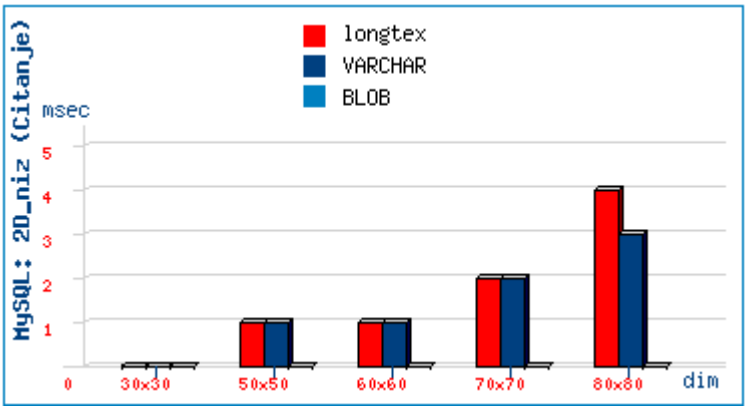
MySQL 2D_niz (R format): UNOS			
mxn	**longtext**	**BLOB**	**VARCHAR**
30x30	0.151	0.078	0.106
50x50	0.162	0.079	0.117
60x60	0.156	0.075	0.119
70x70	0.162	0.079	0.122
80x80	0.165	0.068	0.140

Primer 2.5.8. Tabela 2.5.6 sadrži rezultate dobijene upoređivanjem vremena pretrage podataka koji su uskladišteni u *MySQL* bazu podataka kod koje figurišu različiti tipovi podataka.

Tabela 2.5.6 Čitanje podataka iz različitih MySQL tipova podataka.

MySQL 2D_niz (R format): Čitanje			
mxn	**longtext**	**BLOB**	**VARCHAR**
30x30	0.0089	0.0010	0.0085
50x50	0.0104	0.0017	0.0087
60x60	0.0145	0.0024	0.0104
70x70	0.0293	0.0025	0.0204
80x80	0.0444	0.0026	0.0324

Grafik 2.5.3 Čitanje podataka iz DB sa različitim tipovima podataka.



Najbolje *CPU* vreme izvršavanja imamo kada je tabela za unos i pretragu podataka, u *MySQL* sistemu, *blob* tipa podataka [46].

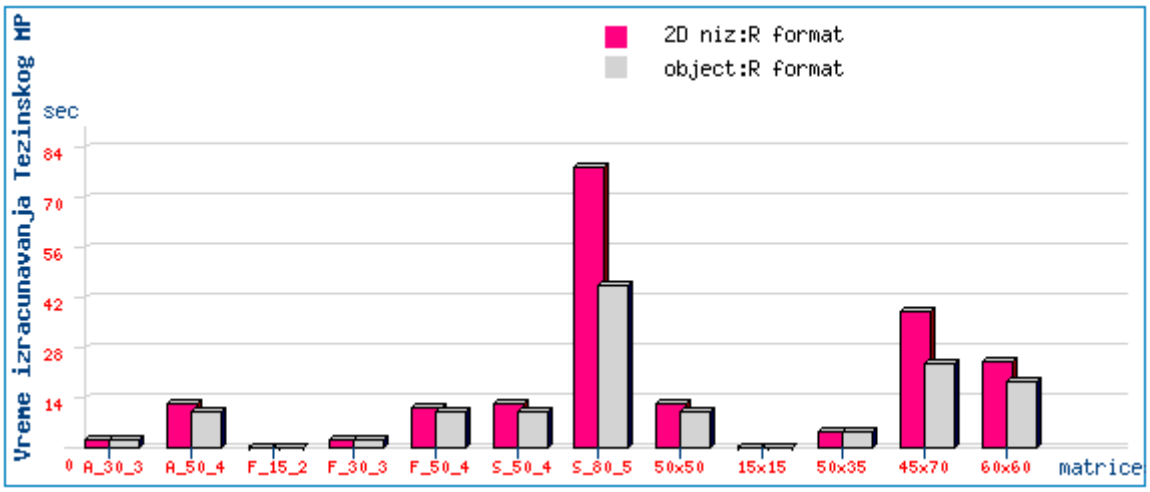
Primer 2.5.9. Uporedne vrednosti dobijene prilikom izračunavanja težinskog *MP* inverza kad je matrica na sloju aplikacije predstavljena u formi *2D_niz* i *objekta*, a uskladištena je u vidu *R formata*.

Tabela 2.5.7 Reprezentacije matrica u vidu 2D_niza (PP pristup) i objekta (OO pristup).

mxn	2D_niz	OOP
A_30_3	0.897	0.790
A_50_4	6.199	5.140
F_15_2	0.166	0.120
F_30_3	0.880	0.750
F_50_4	6.077	5.100
S_50_4	6.070	5.058
S_80_5	40.75	30.459
50x50	6.203	5.870
15x15	0.160	0.116
50x35	1.997	1.602
45x70	19.920	17.549
60x60	12.627	10.722

U našoj aplikaciji *CPU* vreme izvršavanja, za *AL* proračune, je nešto kraće kada imamo *OOP* pristup. Predstavljeni primer pokazuje da na ovaj način dobijamo pozitivne rezultate u odnosu na slučaj sa proceduralnim programiranjem, jer smo samo jednom napravili instancu *klase Matrix* – objekat, koga karakterišu elementi matrice i broj redova i kolona, i čitamo sve potrebne informacije bez dodatnih izračunavanja.

Grafik 2.5.4 Uporedne vrednosti reprezentacije matrica u vidu 2D niza i objekta**.**



Primer 2.5.10. Izračunavanje *CPU* vremena potrebnog za izvršavanje fundamentalnih matričnih operacija korišćenjem *MySQL* procedura (*Funkcija A.4.1 - Funkcija A.4.4*) [45].

Tabela 2.5.8 CPU vreme za *1D_niz* i *2D_niz* u *R formatu* i *MySQL* procesiranje.

m×n		**operacija**	**MySQL obrada**	**1D_niz:R format**	**2D_niz:R format**
3x3	3x3	množenje	0.0493	0.0468	0.0043
5x5	5x5	množenje	0.0650	0.0481	0.0042
10x10	10x10	množenje	0.1218	0.0643	0.0062
20x50	20x50	množenje	1.5318	0.1531	0.0778
45x45	45x45	množenje	3.4856	0.5618	0.1369
80x80	80x80	množenje	7.3275	0.9325	0.7270
80x70	80x70	množenje	7.3787	0.9675	0.5600
81x81	81x81	množenje	9.3250	1.2831	0.7470
3x3	3x3	sabiranje	0.0462	0.0425	0.0040
5x5	5x5	sabiranje	0.0643	0.0431	0.0043
10x10	10x10	sabiranje	0.0968	0.0431	0.0048
50x50	50x50	sabiranje	1.3875	0.1400	0.0326
60x60	60x60	sabiranje	2.0900	0.2012	0.0480
70x70	70x70	sabiranje	2.7681	0.2481	0.0705
80x80	80x80	sabiranje	3.6875	0.3287	0.0919
3x3	3x3	oduzimanje	0.0437	0.0475	0.0041
5x5	5x5	oduzimanje	0.0662	0.0512	0.0044
10x10	10x10	oduzimanje	0.1018	0.0350	0.0050
50x50	50x50	oduzimanje	1.4706	0.1431	0.0319
60x60	60x60	oduzimanje	2.0356	0.1875	0.0733
70x70	70x70	oduzimanje	2.7681	0.02512	0.0415
81x81	81x81	oduzimanje	3.5956	0.3150	0.1196

Rezultati pokazuju da bolje vreme dobijamo u slučaju kad se matrične operacije odvijaju na srednjem sloju nego korišćenjem samo usluga sloja baze podataka, tj. *MySQL* procedura.

Primer 2.5.11. U sledećoj tabeli su upoređivane dobijene vrednosti *CPU* vremena kad retko posednute matrice (sa određenim brojem nenultih elemenata) propuštamo kroz sistem za obradu retko posednutih i sistem za obradu gustih matrica.

Tabela 2.5.9 Retko posednute matrice u sparse i dense sistemu obrade.

bro nenultih elemenata	**sparse sistem obrade**	**dense sistem obrade**	**ugao**
1	0.026	78.127	Gornji levi
2	0.028	77.112	Gornji levi
5	0.054	78.422	Gornji levi
10	0.091	77.153	Gornji levi
20	0.122	77.890	Gornji levi
30	0.151	77.442	Gornji levi
40	0.181	78.223	Gornji levi
50	0.230	79.007	Gornji levi
80	0.254	78.122	Gornji levi
1	10.113	78.127	Donji desni
2	10.134	77.112	Donji desni
5	10.141	78.422	Donji desni
10	10.170	77.153	Donji desni
20	10. 251	77.890	Donji desni
30	10. 325	77.442	Donji desni
40	10. 581	78.223	Donji desni
50	10. 430	79.007	Donji desni
80	10. 508	78.122	Donji desni
80_i	12.986	77.590	Jedinična matrica

Rezultati predstavljeni u tabeli ukazuju da sistem za obradu retko posednute matrica daje bolje rezultate nego sistem konstruisan za guste matrice kad kroz njega propustimo retko posednutu matricu, jer se obrađuju samo nenulti elementi i skraćuje vreme obrade.

Primer 2.5.12. U sledećoj tabeli su prikazane uporedne vrednosti *CPU* vremena za kvadratne matrice posle sprovedenih operacija inverz i generalisani inverz.

Tabela 2.5.10 Inverz i generalisani inverz kvadratne matrice.

mxn	inverz matrice	generalisani MP inverz
10	0.145 sec.	0.060 sec.
20	0.346 sec.	0.112 sec.
30	0.739 sec.	0.436 sec.
45	1.050 sec.	0.711 sec.
50	3.072 sec.	1.115 sec.
80	10.870 sec.	6.048 sec.

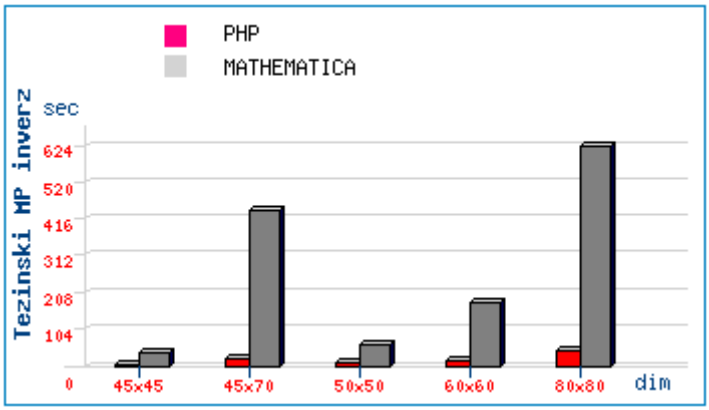
Rezultati dobijeni testiranjem pokazuju da je za izračunavanje inverza kvadratne regularne matrice bolje koristiti sistem za izračunavanje proširenih - generalisanih inverza.

Primer 2.5.13. Uporedili smo *PHP* i *MATHEMATICA* implementacije Algoritma za težinski *MP* inverz. U tabeli su predstavljeni rezultati dobijeni na skupu slučajno generisanih test matrica i matrica $A_{m\times n}$ iz [70], kao i slučajno generisanih simetričnih pozitivno definisanih matrica $M_{m\times m}$ i $N_{n\times n}$ [45].

Tabela 2.5.11 *PHP* i *MATHEMATICA* vreme izvršavanja.

m×n	PHP	MATHEMATICA
15x15	0.166	0.234
20x20	0.277	0.889
30x30	0.880	5.444
45x45	4.665	35.771
45x70	19.920	439.127
50 x50	6.023	62.822
60x60	12.627	179.058
80x80	40.750	621.477

Grafik 2.5.5 *PHP* i *MATHEMATICA* vreme izvršavanja.



Na osnovu rezultata iz prethodne tabele, vidimo da *PHP* daje bolje rezultate u poređenju sa implementacijom u *MATHEMATICA*. Ovo je naročito evidentno za matrice velikih dimenzija.

Primer 2.5.14. Brzina izvršavanja matričnih operacija: množenje, sabiranje, oduzimanje, transponovanje, množenje matrice skalarom, izdvajanje i – te kolone matrice, generisanje submatrice i slično je veoma bitna pri izračunavanju težinskog *MP* inverza. U sledećoj tabeli smo dali *CPU* vremena izračunavanja pojedinih izraza, koji figurišu u težinskom inverzu, u *PHP*-u i *MATHEMATICA* [45].

Tabela 2.5.12 CPU vreme izvršavanja matematičkih izraza.

m×n	**matematički izraz**	**PHP**	**MATHEMATICA**
60x60	A^{-1}	0.416	0.515
70x70	A^{-1}	0.690	0.969
80x80	A^{-1}	1.081	1.825
80x80	$X_1 = (a_1^* M_1 a_1)^{-1} a_1^* M$	0.007	0.234
80x80	$X_1 = (A^* MA)^{-1} A^* M$	2.285	25.709

Na primer, pretposlednji izraz u tabeli, koji konfiguriše u Algoritmu 2.1.2, kad je implementiran u *PHP*-u daje 33 puta bolje rezultate u odnosu na implementaciju u *MATHEMATICA*. Za velike matrice izračunavanje težinskog *MP* inverza, zahteva sprovođenje velikog broja iteracija, pa se zato javljaju velike razlike u vremenu izračunavanja. Svrha navedenih vremena je da daju predstavu o tome da u određenim okolnostima uvek možemo dramatično poboljšati *CPU* vreme izvršavanja, pri dugotrajnom izračunavanju, korišćenjem *PHP* koda u odnosu na kvalitetne *QPS's* kakva je *MATHEMATICA*.

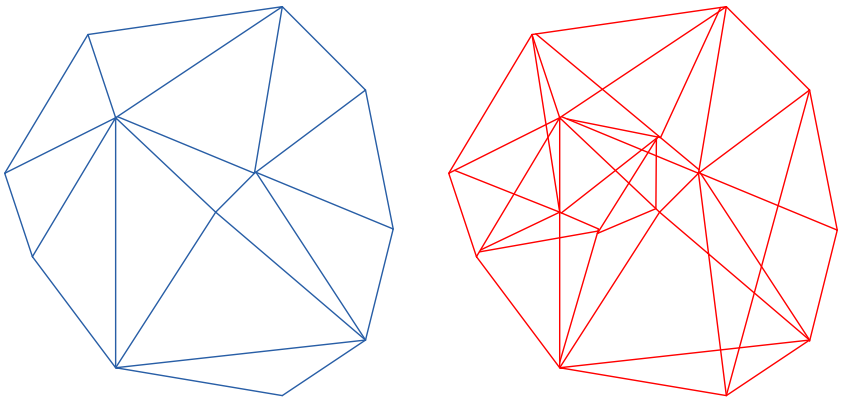
Glava 3

3. Primena MatrixDBMS sistema

U praksi postoji mnogo primera koje možemo svesti na model matrice,a mi smo u radu obradili dva: triangulaciju konveksnih poligona i obradu slike. Razlog transformacije problema na model matrice je činjenica da za matrice postoji dobro razvijen matematički aparat.

3.1. Triangulacija konveksnih poligona

Triangulacija poligona je istorijski veoma star problem koji je doveo do otkrića *Catalan*-ovih brojeva. Kod triangulacije se zapravo razmatra broj načina, koji ćemo označiti sa $Td(n)$, na koji je moguća maksimalna dekompozicija konveksnog n - tougla na $n-2$ trouglova, pa otuda ime triangulacija. Da bismo ga triangulirali, potrebno je povući $n-3$ dijagonala koje se ne smeju seći.



Slika 3.1.1 Slučajevi kad se dijagonale ne seku i kad se seku.

Triangulacija poligona je veoma bitna stvar u kompjuterskoj grafici, ona omogućava da se iz skupa tačaka dobije prikaz trodimenzionalnih objekata. Triangulacija, takođe, omogućava mehanizam za “glačanje” trodimenzionalnih figura, što je jako bitno za brzinu, kvalitet i rezoluciju prikaza objekata u računarskoj grafici. Triangulacija konveksnih poligona

je aktuelan problem koji se javlja u dvodimenzionalnoj računarskoj geometriji. Ovo je klasični problem koji je do sada rešen na nekoliko načina [15,20,26]. Triangulacija poligona ima mnogo primena u računarskoj grafici i, generalno, koristi se u pretprocesnoj fazi broja netrivijalnih operacija prostog poligona [15,49]. Triangulacija poligona ima široku primenu pri modelovanju 3D objekata. Triangulacija je našla primenu pri detekciji sudara objekata. Ona omogućava fizičke simulacije, izračunavanje osećaja dodira, sprečava prolazak jednog objekta kroz drugi. U urbanizmu se koristi prilikom određivanja položaja važnih objekata, a u izgradnji puteva kod detekcije prepreka. U geodeziji se koristi pri premeru, utvrđivanju geodetske mreže, modelovanju terena, kartografskoj obradi i izradi tačnih karata u okviru jedinstvenog geodetskog sistema kao i prilikom utvrđivanja trase tunela.

3.1.1. Dinamičko programiranje

Triangulacija spada u grupu poznatih primera koji su karakteristični za tehniku dinamičkog programiranja, pored lančanog množenja matrica, pronalaženje najdužeg zajedničkog podniza dva niza, problem trgovačkog putnika itd.

Pod dinamičkim programiranjem se podrazumeva tehnika u kojoj se ubrzanje računanja postiže memorisanjem određenih međurezultata tako da kasnije ne moraju ponovo da se izračunavaju, već se samo koriste za nova izračunavanja. Dinamičko programiranje je naziv popularne tehnike u programiranju kojom drastično možemo smanjiti složenost algoritma: od eksponencionalne do polinomijalne. Dakle, kod dinamičkog programiranja rešavanje jednog problema se svodi na rešavanje podproblema. Za ovakve probleme se kaže da imaju optimalnu strukturu. Vrši se particija glavnog problema na nezavisne podprobleme. Zatim nastupa rekurzivno rešavanje podproblema, kako bi se njihovim spajanjem dobilo rešenje polaznog problema. Još jedna bitna karakteristika dinamičkog programiranja jeste da se svaki podproblem rešava najviše jednom, čime se izbegava ponovno računanje numeričkih karakteristika istog stanja.

3.1.2. Algoritmi za triangulaciju konveksnih poligona

Očigledno je da je algoritam koji su dali Hurtado i Noy u [20], rekurzivan i nalaženje partikularne triangulacije se svodi na prolazak preko odgovarajućeg stabla izraza.

Koristićemo pomenuti algoritam za poređenje sa Algoritmima koje smo razvili. To zahteva prirodan broj n i triangulacije za n-tougao. Svaka triangulacija je opisana kao struktura koja sadrži $2n-3$ para temena predstavljenih dijagonala n-tougla (dijagonala se ovde odnosi na unutrašnje dijagonale i spoljašnje ivice).

Broj novih triangulacija $(n+1)$-ugla jednak je stepenu čvora n, tj. broju dijagonala koje se u tom čvoru sustiču (*Funkcija B.1.9*).

Algoritam 3.1.1. Hurtadov algoritam za triangulaciju konveksnih poligona.

Zahtev: Pozitivan ceo broj n.

1: Proverava se struktura koja sadrži $2n-3$ parova temena, pretragom parova $(i_k, n), i_k \in \{1,2,\dots,n-1\};\ 2 \le k \le n-1$, odnosno dijagonala koje se sustiču u teme n. Pozicije ovih indeksa i_k u okviru strukture koja opisuje triangulacije treba čuvati u nizu.

2: Za svaki i_k treba izvršiti transformaciju $(i_l, n) \rightarrow (i_l, n+1), i_l < i_k;\ 0 \le l \le n-3$

3: Unos novih parova $(i_k, n+1), (n, n+1)$.

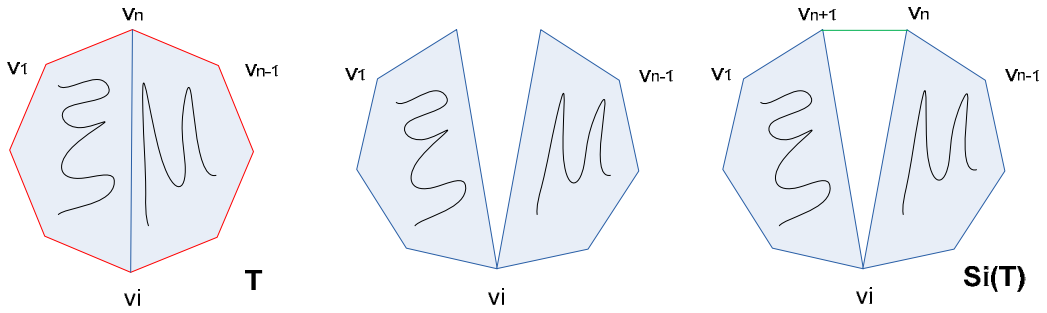
4: Uzimamo sledeće i_k, ako postoji i idemo na korak 2.

5: Nastaviti prethodni postupak sa sledećom n-tougaonom triangulaciom (tj. struktura sa $2n$ -3 parova temena) ako postoje. U suprotnom postupak je potrebno zaustaviti.

Konveksan n-tougaoni poligon P je opisan listingom temena $v_1, \dots, v_n$. Unutrašnja dijagonala koja povezuje temena v_i i v_j je obeležena sa $\delta_{p,q}$. Po dogovoru i stranice poligona se smatraju dijagonalama (ali se pridev interna ne koristi) pa tako $\delta_{i,i+1}$ predstavlja ivicu $v_i v_{i+1}$. Dve dijagonale se ne seku kada nemaju zajedničkih tačaka. Podela unutrašnjosti poligona P na trouglove tako da se unutrašnje dijagonale ne seku naziva se triangulacija poligona. Podela se uvek vrši sa $n-3$ unutrašnje dijagonale. Skup triangulacija poligona P je obeležen sa $\mathcal{T}(n)$. Svi konveksni n-tougaoni poligoni imaju isti graf triangulacija $G_T(n)$. Elementi $\mathcal{T}(n)$ čine skup triangulacija konveksnog poligona. U tom cilju je dogovoreno postojanje poligona sa 0,1 i 2 temena. Ovo je samo formalnost i u daljem tekstu će biti reči o poligonima sa $n > 3$. Svaki čvor T koji pripada $\mathcal{T}(n)$ će imati jednog roditelja, koji pripada $\mathcal{T}(n-1)$, i određen broj sinova koji pripadaju $\mathcal{T}(n+1)$. Formalno, neka $\mathrm{T} \in \mathcal{T}(n)$ bude takav da $\delta_{i,n} \in T$. Njegovi sinovi $S^i(T)$ kao elementi skupa $\mathrm{T}(n+1)$ definisani su kao

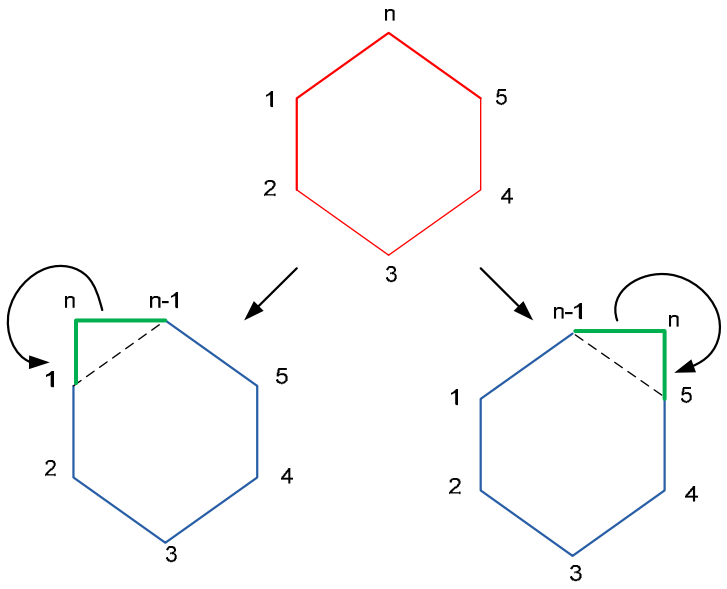
$$S^i(T) = \{\delta_{p,q} | p, q \neq n, \delta_{p,q} \in T\} \cup \{\delta_{p,n+1} | 1 \le p \le i, \delta_{p,n} \in T\} \cup \{\delta_{p,n} | i \le p \le n, \delta_{p,n} \in T\} \cup \{\delta_{p,n+1}\}.$$

Ova operacija se lako može shvatiti preko slike 3.1.2 na kojoj je pokazano cepanje konveksnog n-tougaonog poligona. Ivice otvora su mu dijagonale $\delta_{i,n}$ koje se završavaju u temenu v_i. Staro teme v_n se deli na dva temena, teme v_nčiji je sused v_{n-1} i teme v_{n+1} čiji je sused v_1. Na kraju se dodaje ivica $\delta_{i,n}$.



Slika 3.1.2. Konstrukcija sina $\boldsymbol{S^i(T)}$ triangulacije $\boldsymbol{T}$.

Broj sinova triangulacije $T \in \mathcal{T}(n)$ zapravo predstavlja stepen v_n u T. T će imati najmanje dva sina $S^1(T)$ i $S^{n-1}(T)$, kao što je pokazano na slici 3.1.3.

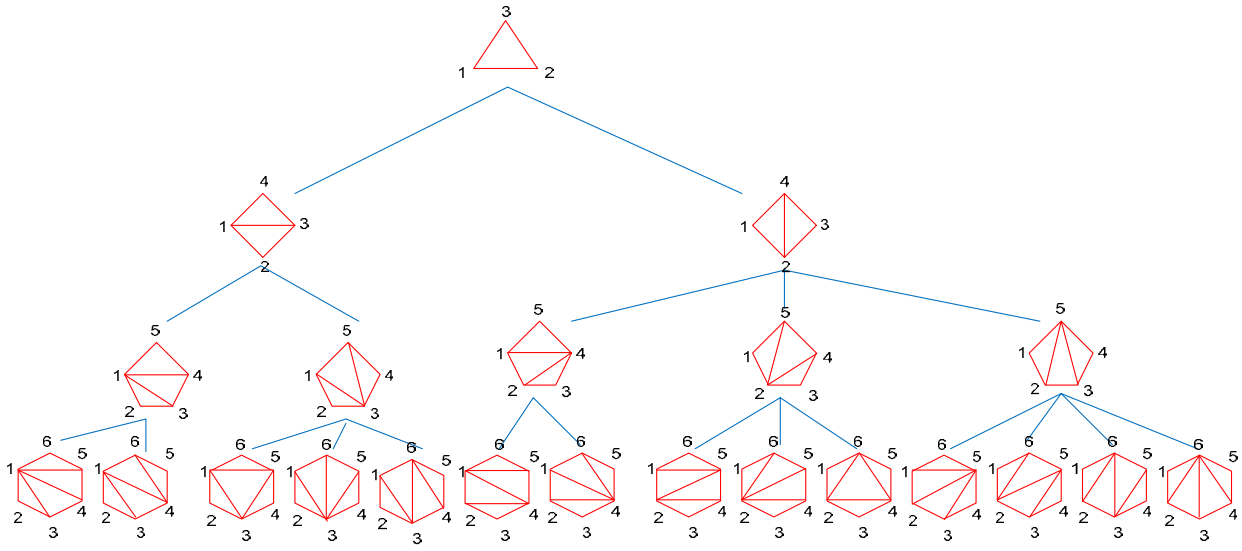


Slika 3.1.3 Sinovi $\boldsymbol{S^1(T)}$ i $\boldsymbol{S^{n-1}(T)}$.

Ako $T \in \mathcal{T}(n)$ tada sin $\tilde{T} \in \mathcal{T}(n-1)$ i pišemo $\tilde{T} = father(T)$ ili jednostavnije $\tilde{T} = f(T)$. Otac $f(T)$ je dobijen od T, cepanjem ivice $v_{n-1}v_n$ (i zadržavanje oznake v_{n-1}), što predstavlja uobičajenu operaciju u teoriji grafova. Ovo takođe pokazuje jedinstvenost oca, tj. da različite triangulacije ne mogu imati zajedničkog sina. Konačno, definišemo binarnu relaciju $T(n)$ tako što se T_1 vezuje sa T_2 ako i samo ako oni imaju istog oca $f(T_1) = f(T_2)$ (možemo reći i da su T_1 i T_2 braća). Po Hurtadu, pri triangulaciji konveksnih poligona sa

$n+1$ temena baziramo se na poligonu sa n temena pri čemu vodimo računa o levoj i desnoj grani. Kod desne grane poslednja grana sa temenima $(n-1\ i\ n)$ se razlaže na dve podgrane i to: $[n-1\ i\ n]$ i $[n\ i\ n+1]$. Kod leve grane poslednja grana sa temenima $(n\ i\ 1)$ se razlaže na dve podgrane: $[n\ i\ n+1]$ i $[n+1\ i\ 1]$.

Indukcijom dobijamo beskonačno stablo koje kao čvorove sadrži sve triangulacije. Čvorovi na nivou n su elementi $\mathcal{T}(n)$, koji su bili čvorovi $G_T(n)$, što pokazuje slika 3.1.4.



Slika 3.1.4 Tri nivoa u stablu triangulacija – Hurtadova i Noyeva hijerarhija.

Dakle, triangulacija je problem koji se sastoji u pronalasku broja mogućih deoba poligona na trouglove pomoću njegovih dijagonala bez preklapanja u tim deobama. Triangulacija n-tougaonog poligona zahteva podelu na trouglove sa $(n-3)$-unutrašnje dijagonale koje se ne ukrštaju. Prema tome, pri izračunavanju svih triangulacija n-tougaonog poligona mora biti razmotren i oblik poligona. Ovo čini problem izračunavanja veoma teškim. Problem smo redukovali tako što smo se ograničili na proračune vezane za konveksne poligone. Za konveksne poligone sve dijagonale su unutrašnje dijagonale. U ovom slučaju broj triangulacija konveksnog n-tougaonog poligona je nezavistan od oblika i može biti jedinstveno okarakterisan brojem temena n.

Broj triangulacija konveksnog n-tougaonog poligona jednak je $(n-2)$ –ugom Katalanovom broju.

$$\delta(n) = C_{n-2} = \frac{1}{n-1}\binom{2n-4}{n-2}$$

Ovaj broj rapidno raste. Naš cilj je uključivanje baze podataka u rešavanju ovog problema i eliminisanju ponovnih izračunavanja.

Problem triangulacija je do sada rešavan na više načina. Jedno rešenje iz [25] koristi bezkontekstnu gramatiku sa produkcijama:

$$S \rightarrow aSS,\ S \rightarrow b \tag{3.1}$$

gde su a i b terminalni, a S neterminalni simbol.

Triangulacija poligona zasniva se na sledećim principima:

a) Neterminal S predstavlja orijentisani topološki segment koji se naziva potencijalnim.
b) Niz aSS odgovara orijentisanom topološkom trouglu sa jednom realnom stranom a i dve potencijalne strane S i S.
c) Smena $S \rightarrow aSS$ znači zamenu potencijalnih segmenata S trouglom aSS koji se sastoji od realne strane a sa definisanom orijentacijom i potencijalnim stranama S koje su stranice poligona koji se na taj način dobija.
d) Smena $S \rightarrow b$ predstavlja zamenu potencijalne strane S realnom stranom b.

Primenom prve produkciju $n-2$ puta u (3.1) (ako se napravi reč sa $n-2$ simbola a u njemu), a onda primenom druge produkcije iz (3.1) odgovarajući broj puta dobija se jedna triangulacija poligona sa n strana.

Broj svih mogućih triangulacija n-tougla $f(n)$ je dat rekurentnom formulom

$$f(2) = f(3) = 1,$$
$$f(n) = \sum_{i=2}^{n-1} f(i)f(n-i+1).$$

Zatim se vrši zamena prvog pravila gramatike u (3.1) i koriste sledeća pravila za generisanje aritmetičkog izraza u inverznoj poljskoj notaciji.

$$S \rightarrow SS + \tag{3.2}$$

$$S \rightarrow b \tag{3.3}$$

Pošto produkcija $S \rightarrow aSS$ predstavlja konstrukciju trougla aSS sa realnom orijentisanom stranom a i dve potencijalne trane S, može se smatrati da produkcija $S \rightarrow SS+$ predstavlja konstrukciju trougla dobijenog zamenom realne strane a trougla aSS realnom stranom $+$, zadržavajući orijentaciju. Na ovaj način se zamenjuje terminal a znakom $+$ koji se može smatrati aritmetičkim operatorom sa dva S kao operandima.

Generisane triangulacije n-tougla se čuvaju u nizu koji sadrži C_{n-2} matrica sa $(n-2) \times 3$ reda. Naš algoritam zapravo generiše sve moguće matrice sa $(n-1) \times 3$ koje odgovaraju svim triangulacijama $(n+1)$-ugla (poželjno bez duplikata), pošto sadrži parove

dijagonala. Da bi prepoznali potencijalne ivice u takvoj matrici cirkularno posmatramo redove matrica tražeći susedne elemente. Izuzetak je ivica $(1,2)$, koja je uvek prava. Takođe, potencijalne su one ivice u formi $(n,1)$ ili $(1,n)$. Pored toga, moramo da generišemo triangulaciju $(n+1)$ – ugla. Ovaj korak je sličan odgovarajućem iz algoritma [20]. Kada smo iscrpeli sve mogućnosti, za sve matrice u nizu, dobijamo sve triangulacije $(n+1)$ – ugla. U nardnom delu ćemo detaljnije izneti korake algoritma.

Algoritam 3.1.2. Triangulacija poligona sa duplikatima.

Zahtev: Pozitivan ceo broj n.

1: Za prvu matricu u nizu izračunavanje triangulacija se vrši na sledeći način:

2: U redu i $(1 \leq i \leq n\ -2\ )$
treba naći dva elementa, koja zadovoljavaju cirkularnu orijentaciju, koja zadovoljavaju sledeće forme $(k,k+1),(k+1,k),(1,n)\ ili\ (n,1)$.

3: a) Kad je pronađen par elemenata u formi $(k,k+1),(k+1,k)$ prepišemo odgovarajuću matricu, kojoj par pripada, i dodamo joj novi red koji počinje ovim elementima, odnosno parom, i dodamo novi element $k+1$, a zatim inkrementiramo svaki elemenat koji je veći ili jednak novom elementu.

b) Ako je pronađen par u formi $(1,n)$ ili $(n,1)$ novo dodati element je $n+1$ i nema više potrebe za promenama.

c) Ako u tekućem redu nema više parova koji zadovoljavaju uslove a) ili b) idemo na korak 4, u suprotnom ponavljamo korak 3.

4: Uvećava se i i ako je zadovoljen uslov $(i \leq n\ -2\ )$ idemo na korak 2, u suprotnom idemo na korak 5.

5: Uzimamo sledeću matricu, ako postoji i idemo na korak 2.

Primer 3.1.1. Pretpostavimo da počinjemo sa trouglom. On ima jedinstvenu, trivijalnu triangulaciju.

$$1 \quad 2 \quad 3$$

Kriterijum iz koraka 2, Algoritma 3.1.2, zadovoljava par $(2,3)$, tako da počinjemo da kreiramo matricu predstavljanu triangulacijom, prepisivanjem matrice sa prethodnog nivoa i dodavanjem novog reda koji počinje parom $(2,3)$. Dalje, dodajemo treći čvor novom redu koji ima vrednost 3.

$$\begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 3 \end{matrix}$$

Sada ćemo inkrementirati svaki elemenat u matrici koji je veći ili jednak novoformiranim čvorom $k+1$ (osim novododatog), i dobijamo

$$\begin{matrix} 1 & 2 & 4 \\ 2 & 4 & 3 \end{matrix}$$

Jedna triangulacija je završena. Proverićemo da li postoji još parova koji zadovoljavaju uslove iz koraka 3, algoritma. Lako je zaključiti da uslove zadovoljava i par $(3,1)$. Dakle, kreiramo još jednu matricu prepisivanjem matrice koja je predstavljena trouglom i dodajemo novi red koji počinje sa $(3,1)$. Dodajemo treći elemenat koji ima vrednost 4, poštujući uslove, i nije potrebno inkrementirati elemente. Dobijamo

$$\begin{matrix} 1 & 2 & 3 \\ 3 & 1 & 4 \end{matrix}$$

Pošto ne možemo naći još parova koji zadovoljavaju uslove iz Algoritma 3.1.2, posao je završen. Imamo dve različite triangulacije, opisane sa dve 2×3 matrice. Na ovaj način ćemo dobiti duplikate, koji se i očekuju. Neke od dobijenih matrica imaju permutovane redove, ali predstavljaju jednu teistu triangulaciju.

Primer 3.1.2. Ako nastavimo sa matričnim predstavljanjem, od triangulacija četvorougla dobijamo triangulacije petougla tako što ćemo obrađivati obe dobijene matrice po uslovima koje diktira Algoritam. Svaki pronađeni par generiše jednu 3×3 matricu koja predstavlja triangulaciju petougla. Korišćenjem procedura opisanih u Algoritmu 3.1.2 dobićemo 6 matrica (po tri od prve i druge matrice koje predstavljaju triangulaciju četvorougla), kao što sledi.

1	2	3	→	1 2 4	2 4 1	4 3 5	1 2 5	2 5 3	5 3 4	1 2 4	2 5 2	5 4 3
3	1	4	→	1 4 2	2 1 4	4 5 3	1 3 5	2 1 3	3 5 4	1 3 4	2 1 1	3 4 5

Pošto je broj triangulacija određenog n-tougla jednak katalanovom broju C_{n-2}, imamo jednu suvišnu matricu. Zaista, prve matrice, u oba reda, su sa permutovanim drugim i trećim redom, ali predstavljaju istu triangulaciju. Ako nastavimo sa triangulacijama za šestougao imamo još težu situaciju, ne samo da imamo 6 duplikata $C_4 = 14$, nego imamo i duplikate sa permutovanim redovima koji nisu obavezno poslednja dva u matrici. Jedno od mogućih rešenja za eliminaciju duplikata je leksikografsko sortiranje matrica po redovima. Drugi pristup je da se uspostavi jedinstvena karakterizacija triangulacije. Postfiks izrazi iz [26] jasno karakterišu određene triangulacije. Iz dokaza Leme 2.2 iz [26] imamo efikasnu proceduru za dobijanje odgovarajućeg postfiks izraza. Pretpostavimo da imamo jednu triangulaciju sa

orijentisanim ivicama i dijagonalama. Svaku dijagonalu i ivicu $(1,2)$ označićemo karakterom $'+'$, a ostale ivice ćemo označiti karakterom $'b'$. Posmatrajmo dve ivice koje definišu jedan trougao sa stranicom $(1,2)$. Dolaznu ivicu ćemo označiti sa $'L'$, a odlaznu sa $'R'$. Uopšte, potreban izraz je u formi $L + R$. Ako je L ili R označeno karakterom $+$ biće u zagradama i algoritam će se rekurzivno primenjivati na njih. Kada rekurzija bude završena tada će novoformirani izraz dobiti poljsku notaciju, koja je očigledno element skupa dozvoljenih postfiks izraza P_n.

Algoritam 3.1.3. Uklanjanje duplikata iz Algoritma 3.1.2.

Zahtev: Pozitivan ceo broj n.

1: Posmatrajmo skup dijagonala poligona plus ivicu $(1,2)$.
Dijagonle su predstavljene parovima u obliku (p, q), gde je $|p - q| > 1$, osim parova u formi $(1, n)$ ili $(n, 1)$. Svakom od njih odgovara karakter $+$ kao i ivici $(1,2)$. Ostalim ivicama odgovara karakter b. Ako matrica $T = (t_{i,j}), i = 1, \dots, n-2, j = 1,2,3,$ predstavlja pojedinačnu triangulaciju n – tougla skup svih linija u triangulaciji (spoljašnje ivice poligona i dijagonale) je

$$E = \{(t_{i,1}, t_{i,2}), (t_{i,2}, t_{i,3}), (t_{i,3}, t_{i,1}) | i + 1, .. n - 2\}. \quad (3.4)$$

Skup dijagonala sa spoljašnjom ivicom $(1,2)$ je

$$E_+ = \{(p, q) || p - q| > 1 \wedge |p - q| \neq n - 1, (p, q) \in E\} \cup \{1,2\}. \quad (3.5)$$

Ostale spoljne ivice čine skup

$$E_b = \{(p, q) || p - q| = 1 \vee |p - q| = n - 1, (p, q) \in E\} \setminus \{1,2\}. \quad (3.6)$$

2: Počnimo sa ivicom $(w_1, w_2) = (1,2)$ kao radnom ivicom. Posmatrajmo dolaznu ivicu radnoj ivici, $(1,2)$, koja je leva ivica i označimo je parom $(w_i, w_1) \in E$ i odlaznu ivicu radne ivice, koja je desna ivica i koju ćemo označiti sa $(w_2, w_0) \in E$.

3: Kreirajmo string u formi $L + R$ gde je L (odnosno R) rezultat rekurzivnih poziva sa (w_i, w_1) odnosno (w_2, w_0) kao radnih ivica. To znači da ako je znak na levoj (desnoj) strani $'+'$ napravićemo rekurzivni poziv za trougao sa ovom ivicom kao osnovom vodeći računa o orijentaciji i rezultat rekurzivnog poziva će biti u zagradama. Osim toga, postavićemo karakter $'b'$ na levoj (desnoj) strani od znaka $'+'$ koji je glavni element (i odgovara ivici $(1,2)$).

4: Transformacija novoformiranih postfiks izraza da bi se dobili jedinstveni izrazi koji odgovaraju pojedinačnim triangulacijama. Moguće suvišne triangulacije koje donosi Algoritam 2.2.1., koje su predstavljene matricama sa permutovanim redovima, će rezultirati jedinstvenim postfiks izrazima.

Primer 3.1.3. Posmatrajmo dve matrice iz primera 3.1.2. koje predstavljaju duplikat. One predstavljaju jedinstvenu triangulaciju.

1	2	4	1	2	5	1	2	5
2	4	3	2	5	3	2	5	4
4	1	5	5	3	4	4	2	3

1	2	4	1	2	3	1	2	3
4	1	5	3	1	5	3	1	4
2	4	3	5	3	4	4	1	5

Korišćenjem procedure opi

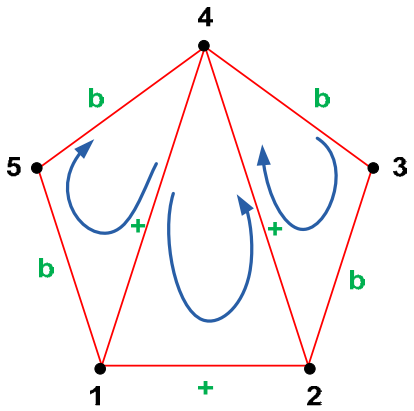
sane u Algoritmu 3.1.3. od prve matrice dobijamo

$$E = \{(1,2),(2,4),(4,1),(4,3),(3,2),(1,5),(5,4)\} \quad (3.7)$$

$$E_+ = \{(1,2),(2,4),(4,1)\} \quad (3.8)$$

$$E_b = \{(4,3),(3,2),(1,5),(5,4)\} \quad (3.9)$$

Dodeljivanje karaktera $+$ i b na linije triangulacije petougla je ilustrovano na slici 3.1.5.



Slika 3.1.5 Dodeljivanje b i $+$.

Prateći dalje korake u Algoritmu 3.1.3 dobijamo da je postfiks odgovarajući izraz ove matrice predstavljen izrazom

$$bb + bb + +$$

Ako primenimo istovetan algoritam na sledeću matricu dobićemo iste skupove $E, E_+ \; i \; E_b$ koji daju isti izraza

$$bb + bb + +$$

Dakle, ne treba ići tako daleko da se nađu postfiks izrazi koji odgovaraju matricama. Dovoljno je naći njihove skupove ivica. Upoređivanjem njihovih skupova ivica lako je zaključiti da jedna od matrica opisuje duplu triangulaciju. Predstavićemo Algoritam 3.1.4, uglađenu verziju Algoritma 3.1.3.

Algoritam 3.1.4. Uglađena verzija Algoritma 3.1.3.

Zahtev: Pozitivan ceo broj n.

1: Za svaku generisanu matricu potrebno je naći skup ivica

$$E = \{(t_{i,1}, t_{i,2}), (t_{i,2}, t_{i,3}), (t_{i,3}, t_{i,1}) | i + 1, .. n - 2\}.$$

Za potrebe programiranja neophodno je dati skup leksikografski urediti. Takođe, mi uvek znamo tačan broj parova, on je $2n - 3$.

2: Proverimo da li je element skupa već sačuvan. Leksikografsko uređenje u kombinaciji sa binarnom pretragom će biti zadovoljavajuće efikasno. Ako je tako odbacićemo ishodišnu matricu.

Složenost sistema za skladištenje će biti od izuzetne važnosti zbog velikog broja triangulacija za veliko n. Za svaku matricu je potrebno $(n-2) \times 3 = 3n - 6$ bajtova za skladištenje, a broj matrica rapidno raste sa povećanjem broja n. Broj dijagonala je $n - 3$, tako da će nam biti potrebno $2n - 6$ parova, što je za trećinu manje nego li u prethodnom slučaju (*Funkcija B.1.6*).

Algoritam 3.1.5. Algoritam sa skladištenjem dijagonala.

Zahtev: Pozitivan ceo broj n.

1: Za svaku triangulaciju n - tougla skladišti se kolekcija skupa dijagonala $E_d(n) = \{E_d^1(n), \dots, E_d^{C_{n-2}}(n)\}$, gde svaki skup $E_d^i(n)$ sadrže dijagonale koje opisuju pojedinačne triangulacije. Za skup $E_d(3) = \emptyset$. Takođe se generiše skup spoljnih ivica n – trougla $E_0(n) = \{(i, (i+1)\ mod\ n) | i = 1, \dots, n\}$. Da bi lakše upoređivali skupove treba ih leksikografski urediti po parovima. Treba napomenuti i to da orijentacija trougla koja je bila obavezujuća u prethodnim algoritmima ovde nije bitna, tako da bi sami parovi bili uređeni (npr. manji članovi kao prvi, a veći kao naredni). Ono što nam je u daljem radu zaista potrebno jesu skupovi $E_b(n) = E_0(n) \setminus \{1,2\}$.

2: Označimo sa $T_d(n)$ skup različitih elemenata u $E_d(n)$. Za pravljenje svih triangulacija (kolekcije skupova $E_d(n+1)$) $(n+1)$-ugla od svakog skupa $T_d^i(n)$ pravimo dodatnih $(n-1)$ skupova $E_d^{i_k}(n+1), k = 1, \dots, n-1$ dijagonala $(n+1)$-ugla uključujući jedan član iz skupa $E_b(n)$. Dakle,

$$E_d^{i_k}(n+1) = T_d^i(n) \cup \{E_b(n)[k]\}, i \in \{1, \dots, C_{n-2}\}, k = 1, \dots, n-1.$$

Generalno, imamo

$$E_d^i(n+1) = T_d^{\lfloor i/n \rfloor + 1}(n) \cup \{E_b(n)[\langle i, n \rangle]\}, i = 1, \dots, (n-1)C_{n-2},$$

gde $\lfloor \cdot \rfloor$ označava nad funkciju i

$$\langle i,n\rangle = \begin{cases} i \bmod (n-1) & i \bmod (n-1) \neq 0 \\ n-1 & i \bmod (n-1) = 0 \end{cases}$$

3: Slično prethodnim algoritmima, ako je uključen par $(k, k+1)$ za jedan će biti uvećan svaki član para (iz skupa $E_d^i(n+1)$) koji je veći ili jednak novoformiranom $k+1$, isključujući njega. Za parove u formi $(1,n)$ nije potrebno vršiti inkrementaciju.

Na ovaj način ćemo imati isti broj duplikata kao u slučaju Algoritma 3.1.2. Broj duplikata koji se generiše Algoritmom 3.1.5 je jednak

$$(n-1)sizeT_d(n) - sizeT_d(n+1) = (n-1)C_{n-2} - C_{n-1} = \frac{(n-2)(n-3)}{n}C_{n-2}, \quad (3.10)$$

gde operator $size(\cdot)$ broj elemenata u skupu. Eliminacija duplikata će biti izvršena binarnom pretragom pošto su svi skupove pri kreiranju predstavljeni leksikografski uređenim nizovima.

Primer 3.1.4. Počnimo skupom $T_d(3) = \emptyset$ i $E_b(3) = E_0(3) \setminus \{1,2\} = \{(2,3),(3,1)\}$. Na osnovu Algoritma 3.1.5 dobijamo kolekciju skupova $E_d(4) = \{E_d^1(4), E_d^2(4)\}$ kome odgovaraju triangulacije četvorougla:

$E_d^1(4) = T_d(3) \cup \{E_b(3)[1]\} = \{(2,3)\} \rightarrow \{(2,4)\}$ i
$E_d^2(4) = T_d(3) \cup \{E_b(3)[2]\} = \{(3,1)\}$.

Pošto se skup $E_b(3)$ sastoji samo od dva elementa imaćemo dva skupa $E_d^i(4)$. Elementi skupa $E_d^1(4)$ će biti inkrementirani shodno koraku (3) Algoritma 3.1.5. Sada ćemo kreirati triangulacije petougla na osnovu triangulacija četvorougla $E_d^i(4)$. Prvo generišemo skup $E_b(4) = \{(2,3),(3,4),(4,1)\}$. Na osnovu Algoritma 3.1.5. Lako je pokazati

$$E_d(5) = \{E_d^i(5), i = 1, \dots 2*3\} = \{E_d^i(5), i = 1, \dots C_2*3\}.$$

Dakle, pošto je $C_3 = 5$ jedna triangulacija će biti suvišna.

$$E_d^1(5) = T_d^1(4) \cup \{E_b(4)[1]\} = \{(2,4),(2,3)\} \rightarrow \{(2,45),(2,4)\}$$
$$E_d^2(5) = T_d^1(4) \cup \{E_b(4)[2]\} = \{(2,4),(3,4)\} \rightarrow \{(2,5),(3,5)\}$$
$$E_d^3(5) = T_d^1(4) \cup \{E_b(4)[3]\} = \{(2,4),(4,1)\}$$
$$E_d^4(5) = T_d^2(4) \cup \{E_b(4)[1]\} = \{(3,1),(2,3)\} \rightarrow \{(4,1),(2,4)\} \equiv E_d^3(5)$$
$$E_d^5(5) = T_d^2(4) \cup \{E_b(4)[2]\} = \{(3,1),(3,4)\} \rightarrow \{(3,1),(3,5)\}$$
$$E_d^6(5) = T_d^2(4) \cup \{E_b(4)[3]\} = \{(3,1),(4,1)\}$$

Izračunavanje duplih triangulacija je zamoran posao koji troši mnogo vremena za izračunavanje. Štaviše, broj duplikata umnogome premašuje broj stvarno potrebnih triangulacija, naročito za veliko n. To stvara velike problem sistemu za skladištenje. Idealan slučaj bi bio kada se ne bi generalisali nepotrebni viškovi triangulacija.

3.1.3. Detalji aplikacione logike

U radu su implementirane i predstavljene tri različite strategije nazvane: *AL* (Application Logic), *ALD* (Application Logic + Database) i *ALDF* (Application Logic + Database + Files). *AL* pristup predstavlja korišćenje samo usluga aplikacionog sloja (Application Logic) u vidu nizova čiji su elementi stringovi koji predstavljaju triangulacije. Ovaj pristup je brz ali ima nekoliko nedostataka. Osobine ovog pristupa su označene sa *AL1 – AL3*.

AL 1: Glavni nedostatak ovog pristupa za generisanje triangulacija je što triangulacije nisu sačuvane za buduću obradu.

AL 2: Predstavljeni algoritmi generišu duplikate koje treba ukloniti. Značajna cena uklanjanja duplikata biće plaćena pri izračunavanju (zagušenje operativne memorije biće takođe problem). Ugrađena *PHP* funkcija $array\_unique()$ uzima niz sa triangulacijama i uklanja duplikate. Funkcija $array\_unique()$ prvo sortira triangulacije predstavljene stringovima, čuva prvi ključ čija je vrednost različita od ostalih i ignoriše sve ostale ključeve.

AL 3: Broj triangulacija $n-$tougla rapidno raste. *AL* pristup koristi nizove za skladištenje triangulacija, što predstavlja razlog za relativno brzo zagušenje operativne memorije. Maksimalna veličina niza u *PHP*-u je uslovljena direktivom $memory\_limit$ u $php.ini$ konfiguracionom fajlu. Ova direktiva ograničava maksimalnu memoriju koju naš *PHP* skript može da koristi. Zbog toga, moramo koristiti sekundarnu memoriju koja podrazumeva neophodnost *ALD* podrške. Ovaj pristup podrazumeva podršku od strane *DBMS* sistema aplikacionoj logici. Umesto skladištenja svake triangulacije ${E_d}^i(n+1)$, odmah nakon generisanja, generišemo blokove podataka (triangulacija) i onda ih skladištimo u bazu. Na ovaj način smanjujemo vreme skladištenja. Istaćićemo osobine *ALD* pristupa, koje smo označili sa *ALD 1 – ALD 4*.

ALD1: Rešava problem iscrpljenosti resursa operativne memorije

ALD 2: Štaviše, u *ALD* podrška, duplikati će biti uklonjeni na taj način što ćemo polje za unos triangulacija (skupove dijagonala u Algoritmu 3.1.5) definisati kao primarni ključ. Ako je u bazi uneta neka triangulacija, primarni ključ onemogućava unos iste.

ALD 3: Takođe, korektnost triangulacije je moguće utvrditi, veoma brzo, jednostavnim upitom nad podacima u bazi podataka.

ALD 4: Mana ovog pristupa je u tome što usled frekventne komunikacije između slojeve u troslojnoj web arhitekturi imamo velike gubitke u vremenu.

Gubici u vremenu su donekle otklonjeni korišćenjem prelaznog rešenja na putu od aplikacione logike ka bazi podataka, i obrnuto, u vidu tekstualne datoteke. To je tzv. *ALDF*

pristup i on nadomešćuje nedostatke *ALD* pristupa. Da bi izbegli permanentnu pretragu, unos i skladištenje podataka koji su predstavljeni triangulacijama, odlučili smo se za podršku u vidu tekstualne datoteke. Odlučili smo se za prelazno rešenje zbog mogućnosti paralelnog transfera podataka iz fajla u bazu podataka i obrnuto. Na ovaj način ćemo, donekle, otkloniti problem u brzini izvršavanja operacija koji se javlja kod *ALD* sistema. Duplikati se ne mogu obrisati korišćenjem fajla ali se mogu paralelno preneti u bazu podataka gde se eliminišu preko jedinstvenog ključa polja u koje treba uneti triangulacije. Još preciznije, *DBMS* sistem se koristi za efikasnu transformaciju skupa $E_d(n+1)$ u skup $T_d(n+1)$.

Najbolje rezultate dobijamo kombinacijom *ALD* sistema i tekstualne datoteke, tj. *ALDF* podrška. *ALDF* podrška podrazumeva da $E_d(n)$ ostaje u txt fajlu u toku procesa generisanja skupa $E_d(n+1)$. Sve triangulacije skupa $E_d(n+1)$ se skladište u nizu do maksimalno dozvoljene vrednosti indeksa, koga diktira mašina (processor i RAM memorija). Za n-tougao $n < 15$ ceo skup $E_d(n+1)$ u vidu bloka podataka (triangulacija) se smešta u txt fajl posle uklanjanja duplikata, na aplikacionom sloju, korišćenjem *PHP*-ove funkcije `array_unique()`. Zatim se podaci iz txt fajla paralelno učitavaju u bazu podataka jednostavnom *SQL* funkcijom `LOAD DATA LOCAL INFILE`. Za n-tougao $n > 15$, gde bi imali da broj elemenata u nizu premašuje kapacitete memorije i dovodi do zagušenja memorije, primenjujemo sistem podele podataka na blokove. Testiranjem smo utvrdili da je maksimalan dozvoljeni broj triangulacija koje možemo smestiti u niz, a da ne dođe do preopterećenja memorije, ≈ 5 miliona. Shodno tome, niz punimo triangulacijama skupa $E_d(n+1)$ do 5 miliona, zatim primenjujemo funkciju `array_unique()` koja očisti duplikate u tom bloku kojeg zatim smeštamo u fajl. Podatke iz niza brišemo, rasterećujemo memoriju, i ponovo generišemo blok od 5 miliona triangulacija $E_d(n+1)$ iz kojeg brišemo duplikate i smeštamo ih u txt fajl. Proces nastavljamo do poslednje triangulacije skupa $E_d(n+1)$, koju generišemo na osnovu $E_d(n)$. Pošto u txt fajlu imamo čiste blokove, ali ne i ukupan fajl, moramo očistiti preostale duplikate, kojih ima znatno manje nego u slučaju da smo sve triangulacije $E_d(n+1)$ smeštali direktno u fajl. Čišćenje obavljamo korišćenjem usluga baze podataka i jedinstvenog ključa kojim smo definisali polje za smeštanje triangulacija. Još preciznije, DBMS se koristi za transformaciju skupa $E_d(n+1)$ u skup $T_d(n+1)$. Na kraju se sadržaj fajla nadograđuje podacima iz baze podataka. On sadrži skup $T_d(n+1)$ i spreman je da pruži usluge pri sledećim izračunavanjima.

Dakle, najbolje rezultate dobijamo kombinacijom aplikacione logike, baze podataka i fajla. Korišćenjem ovakvog pristupa izbegavamo ponovna izračunavanja usled kojih imamo

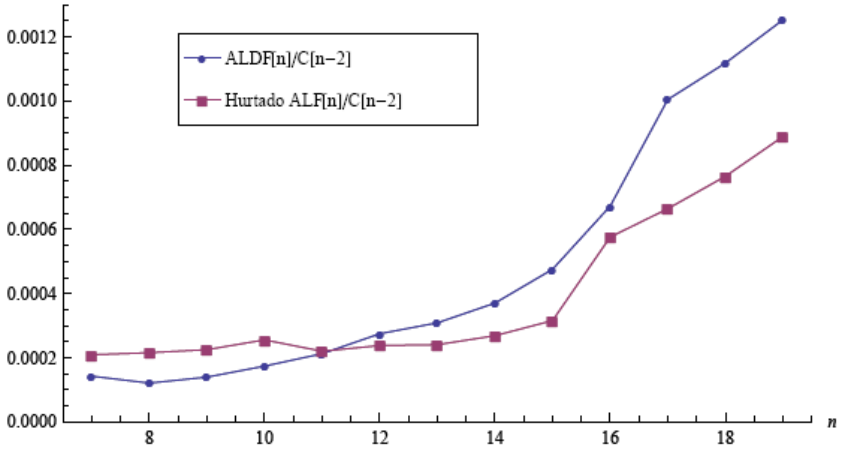
gubitke u vremenu. Takođe, podaci se trajno čuvaju i korisnik može jednostavnim upitom nad tabelom u bazi podataka da utvrdi ispravnost svojih podataka.

Moguće je podatke koji su skladišteni u bazi kombinovati sa koordinatama i manipulisati njima korišćenjem usluga *GUI (Grafic User Interface).* Na kraju proračuna, ili tokom njega, vrši se paralelni transfer sadržaja tekst fajla, sa izračunatim triangulacijama, u bazu podataka korišćenjem jednostavne *SQL* funkcije `LOAD DATA LOCAL INFILE`.

3.1.4. Interpolacija podataka

Iskoristili smo standardnu funkciju iz programskog paketa *MATHEMATICA*, `Interpolation()` za interpolaciju rezultata predstavljenih u kolonama *ALDF* i Hurtado ALF-u Tabeli 3.1.2 ili 3.1.3. Za više informacija o paketu pogledajte [69].

Korišćenjem standardne funkcije `fit()` nalazimo odgovarajuće najmanje kvadrate koji odgovaraju listi vrednosti za količnik$ALDF\ [n]/C_{n-2}$ i $HurtadoALF\ [n]/C_{n-2}$. Ovi količnici predstavljaju CPU-vreme koje je potrebno za generisanje partikularnih triangulacija za dati n-tougao. Grafički prikaz za $n = 7, \ldots, 19$ dat je na slici 3.1.6



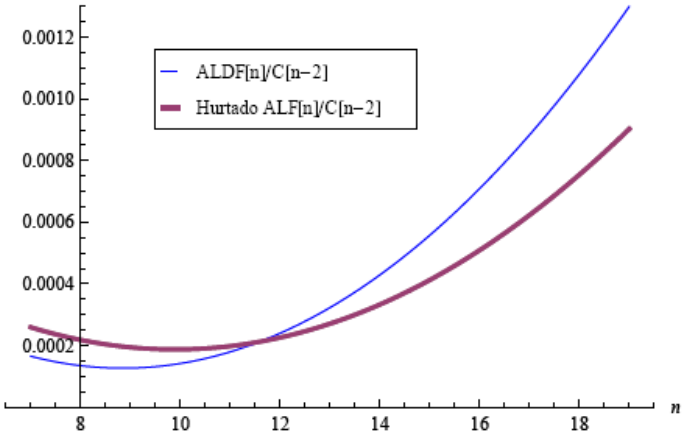
Slika 3.1.6 Vrednosti za $\boldsymbol{ALDF\ [n]/C_{n-2}}$ i $\boldsymbol{HurtadoALF\ [n]/C_{n-2}}$.

Kvadrat odgovarajućih sekvenci količnika $ALDF\ [n]/C_{n-2}$ i $HurtadoALF\ [n]/C_{n-2}$ su jednaki

$0.0000113862x^2 - 0.000201402x + 0.00101689$ i

$8.622342551125387 \times 10^{-6}x^2 - 0.000170513x + 0.00103012$, respektivno.

Grafička ilustracija ovih kvadratnih aproksimacija je data na sledećoj slici.



Slika 3.1.7 Kvadrati koji odgovaraju $\boldsymbol{ALDF}\ [\boldsymbol{n}]/\boldsymbol{C_{n-2}}$ i $\boldsymbol{HurtadoALF}\ [\boldsymbol{n}]/\boldsymbol{C_{n-2}}$.

Ova činjenica ukazuje na to da *CPU* vreme za pojedine triangulacije neznatno raste. Takođe, enormno *CPU* vreme potrebno za generisanje svih triangulacija je proporcionalno njihovom povećanju.

3.1.5. Složenost algoritma

Razmotrimo kompleksnost za generisanje jedne triangulacije $E_d^{i_k}(n+1), 1 \le i_k \le (n-1)C_{n-2}$, po Algoritmu 3.1.5.

Korak 1: Za generisanje skupa $E_d(n)$ potrebna nam je petlja složenosti $n-1$.

Korak 2: U procesu izrade skupa $E_d^{i_k}(n+1)$ potrebno je povezati skup $E_b(n)[k]$ sa skupom $T_d^i(n)$. Označićemo sa I kompleksnost ovih operacija.

Korak 3: Broj elemenata skupa $E_d^{i_k}(n+1)$ je $(n+1)-3 = n-2$. Dakle, kompleksnost ovog koraka je $2(n-2)$.

Kompleksnost generisanja $E_d^{i_k}(n+1)$, je $n-2+I$. Kompleksnost generisanja svih triangulacija skupa $E_d^{i_k}(n+1), k = 1, \dots, n-1$ je $(n-2+I)(n-1)$. Zbirno, od ukupnog broja elemenata u skupu $T_d(n)$ koji iznosi C_{n-2} zaključujemo da je kompleksnost generisanih $E_d(n+1)$ po Algoritmu 3.1.5.

$$Ef(n+1) = n-1+(n-2+I)(n-1)C_{n-2}.$$

Vreme neophodno za paralelni transfer podataka iz tekstualnog fajla u bazu podataka je $S(n+1)$ (kompleksnost transformacije $E_d(n+1)$ u $T_d(n+1)$). Zato je ukupna kompleksnost našeg Algoritma 3.1.5 $Ef_0(n+1) = Ef(n+1) + S(n+1)$.

Sada ćemo posmatrati kompleksnost Hurtadovog Algoritma.

Korak 1: Za proveru $2n-3$ parova potrebno je napraviti mnogo upoređivanja.

Korak 2: Za svako i_k imamo l transformacija. Maksimalna vrednost l je $n-3$.

Korak 3: Imamo dva unosa parova kompleksnosti $2I$.

Korak 4: Maksimalna vrednost k je $n-1$ i potreban nam je toliki broj ponavljanja koraka 2 i koraka 3. Ukupna efikasnost Algoritma 3.1.1 je

$$Ef_H(n+1) = [2n-3+(n-2+2I)(n-1)]C_{n-2}.$$

Razlika broja operacija kod ova dva algoritma je

$$Ef_H(n+1) - Ef(n+1) = (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1),$$

što objašnjava zašto naš algoritam radi brže nego Hurtadov u nekim situacijama. Naravno, kod našeg algoritma pri skladištenju podataka, kad se uklanjaju duplikati, imamo gubitke u vremenu. Pošto Hurtadov algoritam ne proizvodi duplikate razlika u vremenu izvršavanja se smanjuje

$$Ef_H(n+1) - Ef(n+1) = (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1) - S(n+1)$$

Veoma je teško utvrditi vreme neophodno za obavljanje $S(n+1)$ operacija za uklanjanje duplikata i skladištenje (to zavisi od DBMS). Ali za malo n imamo mali broj duplikata tako da je cena eliminacije duplikata umerena i naš Algoritam daje bolje rezultate. Za velike vrednosti n broj duplikata preovlađuje stvaran broj triangulacija te stoga performanse našeg Algoritma slabe.

3.1.6. Sistem za skladištenje triangulacija

Naša baza podataka pored tabela za ulazne i izlazne matrice sadrži i jednu tabelu pod nazivom `triangulations` sa sledećim poljima:

- *id:* identifikacioni broj, definisan kao ključ i autoincrement polje,
- *diagonals:* stringovi koji se sastoje od parova dijagonala odvojenih zarezom ($T_d(n)$), (definisano kao primarni ključ zbog nemogućnosti unosa duplih vrednosti) i
- *n*: broj uglova poligona.

Struktura *SQL* koda za generisanje tabele `triangulations` je data u sledećoj formi

```
CREATE TABLE `triangulations` (
    `id` int(11) NOT NULL auto_increment,
    `diagonals` varchar(255) NOT NULL,
    PRIMARY KEY  (`triangulation`),
    `n` int(3) NOT NULL,
    KEY `id` (`id`))
    ENGINE=InoDB DEFAULT CHARSET=latin1 MAX_ROWS=4294967295;
```

Primer 3.1.5. Tabele za petougao, sa uskladištenim skupovima dijagonala (na osnovu Algoritma 3.1.5) dat je u sledećoj tabeli.

Tabela 3.1.1 Tabela sa smeštenim triangulacijama.

Id	dijagonale	n
1	2,4,2,5	5
2	2,5,3,5	5
3	2,4,4,1	5
4	3,1,3,5	5
5	3,1,4,1	5

Maksimalan veličina podataka koja može biti smeštena u tabelu je determinisana operativnim sistemom kao i tehnologijom vezanom za *MySQL* (mašinom za skladištenje) [68]. U našem slučaju je imamo *InnoDB* mašinu za skladištenje podataka koja je ograničena na 2^{32}-1 zapisa u tabeli i definisana je sa `MAX ROWS = 4294967295` redova. Sistem *InnoDB* prevazilazi problem maksimalne veličine tabele korišćenjem sistema nekoliko fajlova [68].

3.1.7. Test primeri i rezultati testiranja

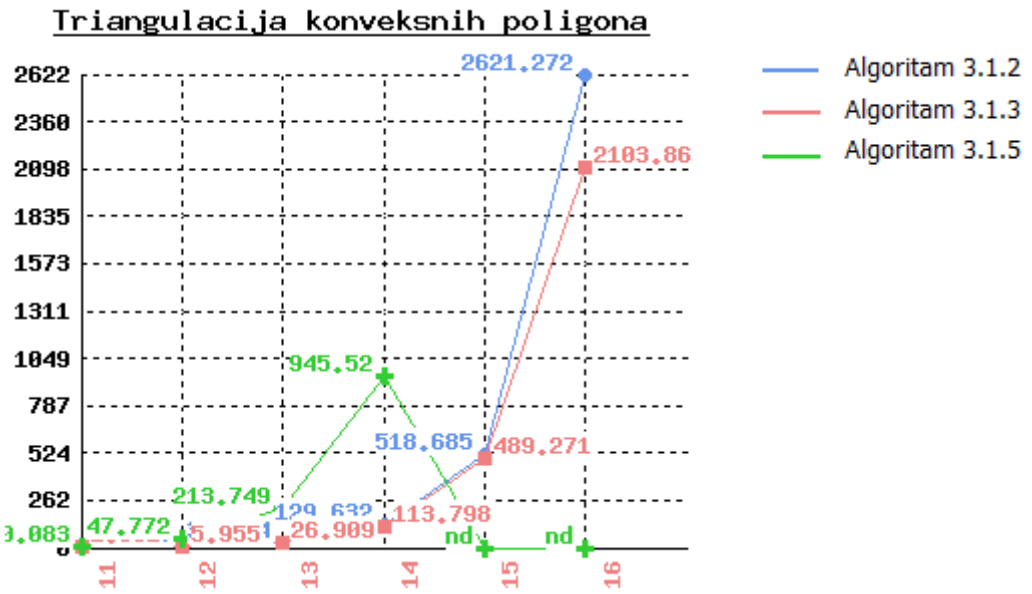
Glavni problem na koji smo naišli pri implementaciji jeste kapacitet memorije, koji je veoma brzo popunjen, zbog rapidnog rasta broja triangulacija za određeni n – tougao.

CPU vremena (u sekundama) za navedene algoritme, u slučaju kad su implementirani korišćenjem usluga aplikacionog sloja bez baze podataka, data su u sledećoj tabeli.

Tabela 3.1.2 Algoritmi za triangulaciju konveksnih poligona

n	Algoritam 3.1.2	Algoritam 3.1.3	Algoritam 3.1.5
7	0.010	0.021	0.007
8	0.041	0.096	0.019
9	0.100	0.420	0.090
10	0.367	1.864	0.336
11	1.523	10.083	1.301
12	7.055	47.772	5.955
13	31.094	213.749	26.909
14	129.632	945.520	113.798
15	518.685	*	489.271
16	2621.272	*	2103.860

Grafik 3.1.1 Algoritmi za triangulaciju konveksnih poligona.



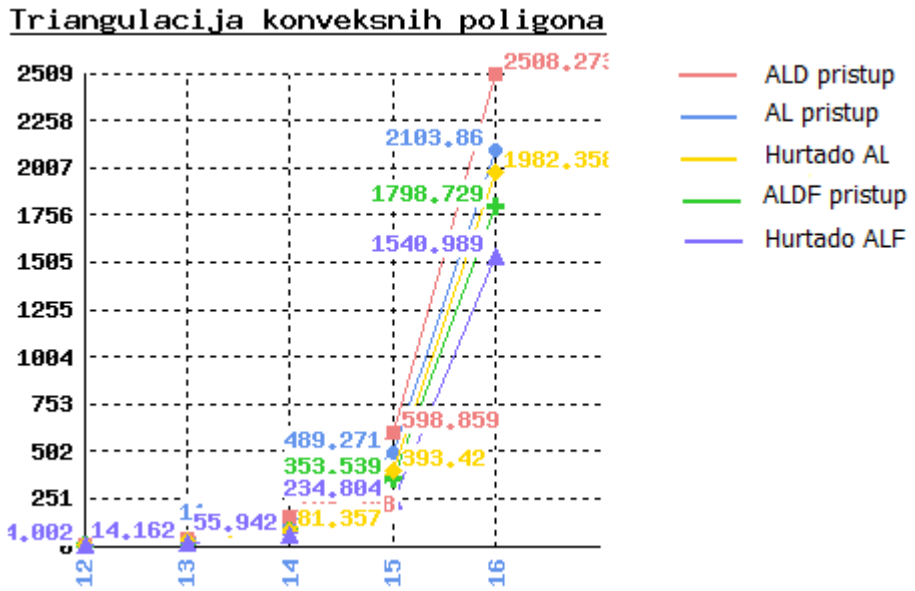
Napominjemo da * označava maksimalnu vrednost za n za koju je moguće obaviti izračunavanje triangulacija. Posle navedenih vrednosti za n dolazi do preopterećenja memorije i dobijamo sledeću poruku *"Allowed memory size of 174483046 bytes exhausted"*.

Problem koji se ovde javlja će biti eliminisan korišćenjem baze podataka. U sledećoj tabeli je dato upoređivanje različitih pristupa Algoritma 3.1.5, kao i Hurtadov Algoritam.

Tabela 3.1.3 Hurtado vs. ALDF pristup.

n	AL podrška	ALD podrška	ALDF podrška	Hurtado AL pristup	Hurtado ALF pristup
7	0.007	0.086	0.006	0.007	0.0088
8	0.019	0.189	0.016	0.038	0.0285
9	0.090	0.421	0.060	0.122	0.0967
10	0.336	1.168	0.249	0.427	0.366
11	1.301	3.562	1.035	1.978	1.072
12	5.955	10.276	4.620	5.626	4.002
13	26.909	40.634	18.166	25.532	14.162
14	113.798	153.248	77.090	81.357	55.942
15	489.271	598.859	353.539	393.420	234.804
16	2103.860	2508.273	1798.729	1982.358	1540.989
17	*	10717.120	9741.606	*	6443.415
18	*	44122.383	39564.870	*	27069.430
19	*	Very long time	162452.920	*	115315.772

Grafik 3.1.2 Hurtado vs. ALDF pristup triangulaciji konveksnih poligona.



Iz tabele se jasno može videti da najbolje rezultate daje sistem podrške *ALDF*, kad su u pitanju algoritmi koje smo implementirali, čak bolje i od rezultata Hurtadovog algoritma za $n <= 11$, a mnogo ne zaostaje za njim ni za $n >= 11$ [27]. Ovo su malo začuđujući rezultati koje smo pokušali da razjasnimo u sekciji 3.1.5. To je najbolji način za uštedu u vremenu i memoriji. Unos i čitanje podataka iz baze je nezavistan od broja zapisa koji se nalaze u tabeli. Tako imamo:

- za unos fajla sa 427 redova korišćenjem `LOAD DATA LOCAL INFILE` imamo vreme od 0.066 sekundi i
- selektovanje 1000 zapisa iz tabele baze podataka daje vreme od 0.000617 sekundi.

Aplikacija omogućava prikaz sto slučajno odabranih triangulacija za slučaj velikog broja uglova poligona. CPU vremena su prikazana u sledećoj tabeli.

Tabela 3.1.4 Prikaz slučajno odabranih triangulacija za $n > 20$.

n	CPU vremena
20	0.242
25	0.329
30	0.427
35	0.631
40	0.0878
45	1.226
50	1.399

Jasno je da na ovaj način sa malim skupom triangulacija smanjujemo vreme obrade i povećavamo broj uglova za koji je moguće veoma brzo izračunati i prikazati triangulacije.

3.2. Obrada slike

Sa naglim razvojem elektronike i računarske tehnike, a posebno mikroprocesora i integrisanih digitalnih procesora signala, digitalna obrada slike počela je da prodire i u mnoge druge oblasti, gde ranije nije mogla biti korišćena zbog visoke cene. Tako se, na primer, digitalna obrada slike intenzivno koristi u medicini za obradu i popravku kvaliteta rendgenskih i ultrazvučnih slika, za trodimenzionalnu rekonstrukciju organa u tomografiji i nuklearnoj magnetskoj rezonansi itd. Važnu primenu digitalna obrada slike je našla u fizici, astronomiji, biologiji, kriminalistici, metalurgiji, geologiji, pa čak i u arheologiji i rekonstrukciji umetničkih dela. U poslednjih desetak godina digitalna obrada slike je našla primene i u filmu, televiziji, proizvodnji muzičkih spotova i reklama, robotici, kontroli proizvoda u industriji, sistemima za automatsko upravljanje itd.

PHP karakteriše fleksibilnost koja se ogleda u širokom spektru platformi i aplikacija na kojima može da radi. Može se sigurno i efikasno povezati na bazu podataka i podatke prikazati u grafičkom formatu. *PHP* sadrži niz funkcija koje omogućavaju otvaranje, manipulaciju i prikaz grafike u Web browser-u. Da bi koristili usluge biblioteke *GD* pod Windowsom potrebno je ukloniti komentare sa linije

```
extenstion=php_gd2.dll
```

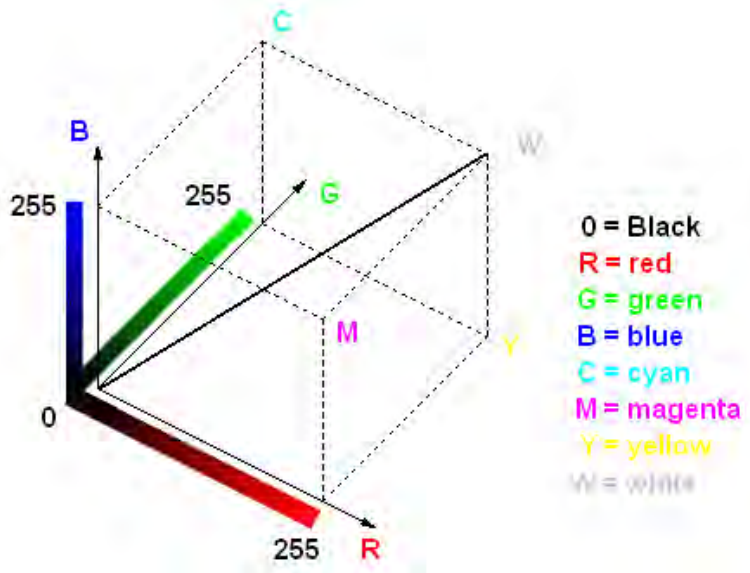
u php.ini konfiguracionom fajlu [68].

Digitalna obrada slike predstavlja skup metoda za obradu slike pomoću računara. Slike koje se obrađuju mogu imati različito poreklo. Osim sa kamere, kao najčešćeg ulaza u sistem za obradu slike, slike mogu poticati sa satelita i drugih letelica, uređaja za medicinsku dijagnostiku, telekomunikacionih uređaja za prenos, memorijskih medijuma za skladištenje, radara, sonara i drugih uređaja. U najvećem broju slučajeva ulazna slika u sistem za obradu slike potiče iz vidljivog dela spektra, mada u nekim slučajevima predstavlja dvodimenzionalni signal iz nevidljivog dela elektromagnetskog spektra. Zbog toga se u širem smislu, termin digitalna obrada slike koristi i za obradu bilo kakvih dvodimenzionalnih podataka. Izlazni signal iz sistema za digitalnu obradu slike je najčešće nova slika koja se prikazuje na video monitoru. Osim toga, izlazni podaci mogu biti prikazani i u obliku fotografije ili filma, a u nekim primenama i u vidu numeričkih podataka.

Uvođenjem elementarnih tehnika digitalne obrade slike znatno je skraćen proces prenosa slike i popravljen kvalitet. Mada je u kasnijem periodu došlo do znatnog napretka u tehnikama prenosa i rekonstrukcije slike, tek je sa usavršavanjem računarske opreme i

početkom istraživanja svemira došlo do naglog razvoja digitalne obrade slike. Naime, 1964. god. digitalni računar je prvi put iskorišćen za popravku kvaliteta slika Meseca dobijenih sa svemirske sonde "Ranger 7" [14]. Posle toga, digitalna obrada slike je postala nezaobilazan deo obrade podataka sa raznih satelita i sondi za istraživanje svemira. Postoji mnogo metoda koje se mogu primeniti na slici: poboljšanje slike, restauracija, kompresija, rekonstrukcija iz projekcija, histogram, filtriranje itd.

RGB prostor boja je definisan pomoću tri aditivna primara: crvene, zelene i plave boje. Svaka boja nastaje zbrajanjem pojedinih komponenata te tri boje. RGB model je predstavljen pomoću kocke, gde crvena boja predstavlja x-osu, zelena boja y-osu, a plava boja z-osu. Slika 3.2.1 prikazuje model *RGB* prostora boja. Taj se prostor boja najčešće koristi u računarima. Svaka boja najčešće je predstavljena sa 8 bitova, odnosno vrednostima od 0 do 255 (256 vrednosti). To daje ukupno 256^3 = 16777216 mogućih boja [14]. U različitim računarskim programima za obradu slike najčešće se navodi podatak o 16 miliona boja. Najčešće se taj prostor boja normira na vrednostima od 0 do 1. *RGB* prostor boja je jednostavan za računar, ali nije prikladan za čoveka. Crvena, zelena i plava komponenta međusobno su korelisane tako da je čoveku vrlo teško da izborom tih komponenata definiše željenu boju u *RGB* prostoru boja. Stoga se najčešće koriste neki drugi prostori boja kao što su npr. *HSV (HSI, HSB) ili HSL* prostor boja.



Slika 3.2.1 Model RGB prostora boja.

3.2.1. Generisanje grafike

PHP ima sjajnu grafičku biblioteku *GD* koja može da kreira slike. *GD* je program za crtanje grafike koji radi sa poznatim formatima, kao što su *PNG, JPEG, GIF*... Pomoću *PHP*–a je moguće kreirati stubičaste dijagrame profesionalnog izgleda (*Funkcija B.4.1, Funkcija B.4.2*), koji se koriste kod različitih izveštaja. Ovi dijagrami mogu da daju dodatnu snagu analizi rada preko Weba, pošto oni koriste različite tehnike kompresije. Format *JPEG* koristi kompresiju sa gubicima. To znači da se neki podaci originalne slike gube za vreme kompresije, pa je ona pogodna za slike kao što su fotografije gde ima dosta nežnih senki i ne mnogo preciznih detalja. Taj se format koristi u slučaju kada mali gubitak u kvalitetu neće posmatraču biti važan. Format *PNG* kompresuje bez gubitaka. Najbolje radi sa slikama koje sadrže linije, kakav je slučaj sa slikama koje predstavljaju moguće triangulacije odabranog n-tougla. Kad se slika dekompresuje sadrži sve originalne informacije.

Koraci koje je potrebno sprovesti u *PHP* - u da bi napravili sliku su sledeći:

1. Napraviti platno za sliku na kome će *PHP* raditi - to je rezervisani deo memorije na serveru na kome će skript da “crta” pre nego ih prikaže u Web čitaču ili na disk kao sliku.
2. Crtanje slike na platnu.
3. Slanje slike Web browser-u.
4. Čišćenje zauzete memorije “bacajući platno”.

Kreiranje platna za crtanje se vrši pozivom funkcije

```
int ImageCreate(int x_size, y_size).
```

Argumenti funkcije `ImageCreate()` su širina i visina slike, respektivno. Funkcija će vratiti identifikator prazne slike u memoriji, tako da će sve sledeće funkcije sa slikama da se obraćaju slici u memoriji preko ovog identifikatora.

Platno za crtanje, linije i oblici kojima se crta imaju određenu boju koju kreiramo funkcijom

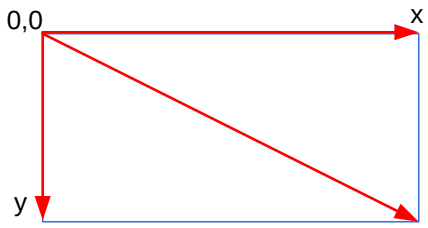
```
int ImageColorAllocate(int image, int red, int green, int blue).
```

Iscrtavanje teksta se vrši funkcijom

```
int ImageString(int image, int font, int x, int y, string s, int color).
```

Prvi argument je slika u koju se tekst ubacuje, drugi je font, treći i četvrti su koordinate gornjeg levog ugla gde počinje ispisivanje stringa, peti je sam string, a šesti boja

koja se koristi u tekstu. Za precizan opis tačke na slici koristi se koordinatni sistem. Kod *PHP* – a sve koordinate idu iz gornjeg levog ugla slike kao što se može videti sa slike 3.2.2.



Slika 3.2.2 Koordinatni sistem.

Za crtanje linija koristimo funkciju

`int ImageLine().`

Argumenti su joj identifikator platna za sliku, x i y koordinate početka i kraja linije, kao i boja za crtanje. Nacrtanu liniju, na platnu, treba sačuvati na disk ili poslati Web browseru. Funkcije koje nam to omogućavaju su

`int ImageJPEG()` i `int ImagePNG(int image,[string file name,int quality])`,

sa argumentima: identifikator platna, naziv slike (opciono) i kvalitet slike (opciono).

PHP funkcija za crtanje kruga, elipse ili kružnog luka je `ImageArc()`, a za crtanje pravougaonika koristi se funkcija

`int ImageRectangle(int image,intx1,int y1, int x2, int y2, int color).`

Argumenti su joj slika, koordinate donjeg levog i gornjeg desnog temena i boja. Crtanje figura koje nemaju četiri ugla može biti malo komplikovanije. Funkcija koju koristimo za crtanje poligona je

`int ImagePolygon(int image, int array points, int num_points, int color).`

Argumenti su identifikator platna, argument u obliku niza sa svim vrednostima koordinata koje treba da budu nacrtane, broj temena i boja.

PHP omogućava crtanje stubičastih dijagrama koji služe za poređenje sličnih rezultata iz različitih kategorija ili oblasti. Oni omogućavaju da lakše prikažemo razlike podataka koji se porede. Dijagrami su, generalno, medij koji pomaže da se podaci vizuelno predstavljaju i to je odličan način za pomoć u shvatanju značenja podataka. Za crtanje stubičastih dijagrama koristimo linije, poligone kao i funkcije

`ImageTTFText()` i `ImageTTFBBox()`.

3.2.2. Kreiranje PDF dokumenata

Uprkos napretku koji je u poslednjoj deceniji postignut u formatiranju Web dokumenata, ipak ponekad *HTML* nije dovoljno dobar. Iako standard *CSS (Cascading Style Sheet)* omogućava da, do zadivljujućeg nivoa, podesimo izgled elektronskih dokumenata, to nije garancija za dobijanje strana koje izgledaju savršeno. To u prvom redu zavisi od pretraživača da podrži sva *CSS* pravila koja su ugrađena u dokument, što je očigledno u slučaju ako pretraživač u potpunosti podržava *CSS*. To se odnosi, na primer, na mogućnost pretraživača da tekstualne elemente prikazuju istim fontom, sa istim razmakom između redova, karaktera i slično. Pored toga, važno je da li korisnik može da se u potpunosti povinuje *CSS* pravilima koja su zadata, a da ih ne isključuje u svom pretraživaču. Iako ovo može da bude dovoljno dobro za većinu Web dokumenta, ipak to nije dovoljno za sve dokumente.

Jezik *PDF* ima mnogo više mogućnosti od *HTML*-a i *CSS*-a, a *PDF* pokazivači garantuju da će dokument biti prikazan u originalnom obliku, onako kako je dokument i kreiran. To je i razlog zbog koga se *PDF* dokumenti koriste u situacijama kada se ne mogu dopustiti odstupanja u njihovom izgledu, bez obzira na računar na kome se to štampa ili prikazuje. Ovakvi dokumenti potpuno isto izgledaju bez obzira da li je *PDF* pokazivač na *Macintosh* računaru ili je u pitanju *Linux OS.*

Ipak sa korišćenjem *PDF*-a treba biti obazriv jer su takvi dokumenti mnogo veći nego li *HTML* dokumenti, komplikovaniji su od njih te ih je stoga mnogo teže menjati. Generisanje *PDF* dokumenata nije ni besplatno. *PDFlib* biblioteka se koristi za dinamičko generisanje dokumenta, besplatna je za upotrebu koja nije komercijalna. Da bi koristili usluge *PDFlib* biblioteke pod *Windowsom* potrebno je ukloniti komentare sa linije *php.ini* konfiguracijskog fajla

```
extenstion=php_pdf.dll.
```

U radu smo koristili *PDFlib* biblioteku za prikaz matrica i triangulacija konveksnih poligona iz tabele baze podataka. Korisnik može da prikaže rezultate obrade, koji se nalaze u tabeli, u pdf formatu i snimi ih na svoj računar.

3.2.3. Sistemi skladištenja slike

MatrixDBMS model ima mogućnost skladištenja slika u bilo kom formatu. Takođe, u njemu se skladište i rezultati obrade dobijeni primenom neke od metoda za uređivanje slike.

Slika 3.2.3. Test primeri digitalne obrade slike.

SQL iskaz `CREATE TABLE` za kreiranje strukture podataka i tabela za skladištenje slike i rezultata obrade u *MySQL* bazi podataka dat je kao:

```
# Table structure for table `image_in `
      CREATE TABLE image_in(
      id_in int (11) NOT NULL auto_increment,
      elements_in longtext NOT NULL,
      dimension inytext NOT NULL,
      type tinytext NOT NULL,
      PRIMARY KEY(id_in)
      )TYPE=InoDB

# Table structure for table `image_out `
      CREATE TABLE image_out(
      id_out int (11) NOT NULL auto_increment,
      element s_out long text NOT NULL,
      operation tinytext NOT NULL,
      original tinyint (4) NOT NULL default '0',
      PRIMARY KEY(id_out)
      )TYPE=InoDB;
```

Transformaciju slike u format koji je moguće skladištiti u bazu podataka moguće je odraditi na više načina, a mi smo koristili i upoređivali dva: *RGB* model (*Funkcija B.3.1*, *Funkcija B.3.2*) i transformacija `base_64()` funkcijom (*Funkcija B.3.3*, *Funkcija B.3.4*).

Slika Babun je transformisana `base_64()` funkcijom, i imamo sledeći kod:

```
/9j/4AAQSkZJRgABAgEAyADIAAD/wAARCAIAAgADAREAAhEBAxEB/9sAhAADAgIDAgIDA
wIDAwMDBAUIBQUEBAUKBwgGCAwLDQ0MCwwMDg8UEQ4PEw8...3Fd7jy7HmfFqOurbyyGz
tIPDL2xWLTexpbVpm14N0KOWaMyKWb72M8VhKdnZmz1jc9SSGO0tVCDGO49ayjrqcVWT5
jy3x14mFjI6Zfk43euOldtKL6GcdUeMa3rEt3ORO SCxAUr2/wA8V2Rhylp82xNodpJdTJ
9nU4Ug5Lf/AF6UtNh2urPY9P8AD3hVLa1XKKZAcjcAee9eHiq7va572Cw91zH/2Q==
```

RGB model slike Lena je predstavljen sledećim fragmentom koda, a komponente *R*, *G* i *B* su povezane u string i tako čine *R format*.

```
21,22,27,23,24,29,24,25,30,25,26,31,24,25,30,22,23,28,21,22,27,21,22,
27,21,22,27,19,20,25,25,26,31,22,23,28,17,20,25,21,24,29,19,22,27,16,
21,25,24,24,32,21,21,29,23,23,31,23,24,29,24,25,30,23,24,29,20,21,25,
24,25,29,24,25,30,23,24,29,22,…236,230,225,231,231,224,231,225,218,226,
234,228,238,241,237,251,241,239,253,241,239,252,237,235,248,237,235,
248,235,233,246…
```

Tabela 3.2.1. sadrži RGB modele slika različitih dimenzija.

Tabela 3.2.1 Tabela za unos slike.

id_in	elements_in	type	dimension
1	21,22,27,...,233,246	RGB	300x372
2	225,138,121,...,75,81	RGB	512x512
3	130,130,104,...,124,122	RGB	50x50

Rezultate obrađenih slika sadrži Tabela 3.2.2.

Tabela 3.2.2 Tabela sa rezultatima obrađene slike.

id_in	elements_in	operation	original
1	19,22,27,...,12,11,16	FILTER	1
2	161,160,...,103,106,112	GRAY	2
2	0.06,0.131,...,0.14,0.29	HISTOGRAM	2

Sistem *InnoDB* i *MySQL* omogućavaju smeštanje slike maksimalne veličine 4GB [68].

MySQL MatrixDBMS predstavlja samo delimično ostvarenje našeg glavnog cilja. Cilj će u potpunosti biti ostvaren *Cassandra MatrixDBMS* modelom podataka koji je dat u glavi četiri.

3.2.4. Rezultati testiranja i primeri

U sledeće dve tabele smo prikazali *CPU* vremena za dva različita slučaja transformacije slike u string (matricu) i skladištenja u bazu podataka.

Tabela 3.2.3 Vreme potrebno za transformaciju slike.

Take time to transform image			
dimenzije slike	**model**	**RGB model**	**base_64 kodiranje**
50x50	RGB	0.0572	0.0013
350x372	RGB	3.5893	0.0021
512x512	RGB	7.6160	0.0026
1200x800	grey	21.1110	0.0068

U tabeli su prikazana CPU vremena od momenta lociranja stringa uskladištenog u bazi do momenta kreiranja izvorišne slike.

Tabela 3.2.4 Vreme potrebno za kreiranje slike.

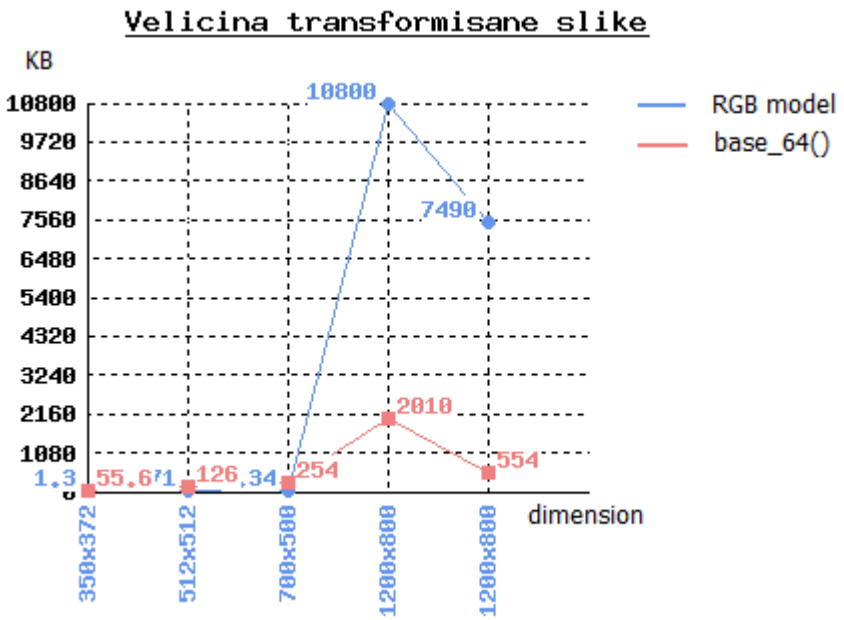
Take time to create image			
dimenzije slike	**model**	**RGB model**	**base_64 kodiranje**
50x50	RGB	0.0973	0.0013
350x372	RGB	5.5295	0.06771
512x512	RGB	10.6700	0.1154
700x500	RGB	15.7097	0.1998
1200x800	RGB	31.8750	0.3615
1200x800	grey	31.4026	0.3359

Veličine stringova, dobijenih transformacijom na jedan od dva navedena načina obrade, su prikazani u Tabeli 3.2.5.

Tabela 3.2.5 Memorijski kapaciteti koje zauzima transformisana slika.

Size of the created image			
dimenzije slike	**model**	**RGB model**	**base_64 kodiranje**
50x50	RGB	24.3 KB	2.26 KB
350x372	RGB	1.30 MB	55.6 KB
512x512	RGB	2.71 MB	126 KB
700x500	RGB	3.34 MB	254 KB
1200x800	RGB	10.8 MB	2.01 MB
1200x800	grey	7.49 MB	554 KB

Grafik 3.2.1 Kapaciteti memorisane slike.



Svi dobijeni rezultati, predstavljeni u gore navedenim tabelama, nedvosmisleno ukazuju na prednosti `base_64()`modela reprezentacije slike u odnosu na transformaciju *RGB modelom.* Te prednosti su očigledne i u pogledu brzine transformacije i u pogledu prostora koji zauzima transformisana slika. Međutim, napominjemo da je računarska grafika prilagođena RGB modelu i da je sve obrade slike potrebno obaviti transformacijom ovim modelom.

Glava 4

4. Novi pristup matričnim izračunavanjima

4.1. NoSQL Cassandra univerzalni MatrixDBMS model

Postoji mnogo razloga zbog kojih su relacione baze podataka postale toliko popularane, naročito tokom poslednje četiri decenije. Veoma važna je *SQL* deklarativna sintaksa, koja je bogata mnogobrojnim funkcijama i jednostavna za korišćenje. *SQL* je jednostavan za upotrebu, osnovna sintaksa može se brzo naučiti. Programeri početnici mogu veoma lako postati vešti u radu sa *MySQL*-om. U industriji koja je prepuna promena, zasnovana na kratkim rokovima, lakoća upotrebe može biti veoma važna. Osim jednostavne sintakse *MySQL* ima veoma robustan alat koji uključuju dobar grafički interfejs za gledanje i rad sa bazom podataka.

U poslednje vreme, nauka i web kompanije su uključeni u istraživanje vezano za baze podataka što je dovelo do pojave velikog izbora alternativnih baza podataka. To dovodi do pojave novog pristupa poznatog kao *NoSQL* baze podataka [28] (ključ-vrednost sistemi skladištenja, dokument baze podataka, kolona orijentisane baza podataka) .

Cassandra model podataka je dobio ime po liku Kasandra iz grčke mitologije. Kasandra je bila ćerka kralja i kraljice Troje, Prijama i Hekube [9]. Kasandra je bila toliko lepa da joj je bog Apolon dao sposobnost da vidi budućnost. Kada je odbila njegovu ljubav, on je prokleo tako da je ona i dalje bila u stanju da tačno predvidi sve što će se desiti, ali da joj niko ne poveruje. Kasandra je predvidela uništenje grada Troje, ali je bila nemoćna da spreči to uništenje.

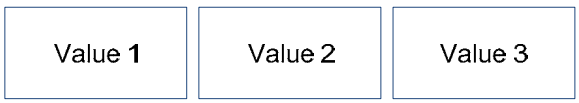
Apache Cassandra je *open source*, distribuiran, decentralizovan, elastičan u pogledu proširenja, izuzetno dostupan, otporan na greške, dosledan, kolona-orijentisan model baze podataka. To je *NoSQL* rešenje, koje je u početku razvijeno od strane *Facebook*-a za pokretanje sopstvenog prijemnog poštanskog sandučeta sve do kraja 2010. Jeff

Hammerbacher, koji je tada vodio *Facebook* tim je opisao *Cassandru* kao *Big Table* model podataka koji radi na *Amazon Dinamo*. *Apache Cassandra* je napravljena kao hibrid između *Google-BigTable* [3] i *Amazon Dinamo* sistema [8]. *Cassandra* predstavlja repliku modela kreiranog u Dinamo stilu sa neuspehom svedenim na minimum, sa dodatkom moćnog *familija kolona (column family)* modela podataka [9]. Velike razlike između *MySQL* i *Cassandra,* kad je reč o pisanju/čitanju (R/W), će biti pomenuti u sekciji sa rezultatima testiranja.

Cassandra je projekat *Apache Softvare Foundation* koji je dizajniran da procesira velike količine podataka, dok pruža visoko dostupne usluge bez greške.

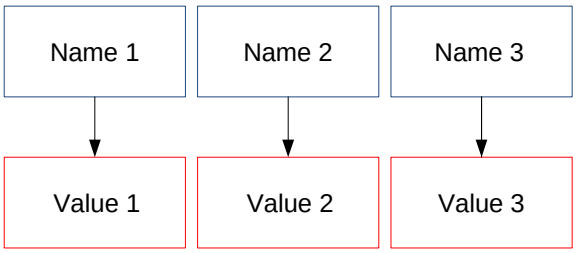
Postoji više razlika između *MySQL* relacionog sistema skladištenja i *Cassandra NoSQL* modela podataka. U relacionoj bazi podataka imamo bazu koja odgovara jednoj aplikaciji. Baza podataka sadrži tabele, koje imaju imena i sadrže jednu ili više kolona, koje, takođe, imaju imena. Kada se unose podaci u tabelu navede se vrednosti za svaku već definisanu kolonu, dok za slučaj da ne postoji vrednost za određenu kolonu, koristi se vrednost *null*. Svaki novi unos dodaje red tabeli, koji kasnije možemo pročitati korišćenjem jedinstvenog identifikatora reda (primarni ključ), ili korišćenjem *SQL* upita pomoću kojih se može pristupiti podacima koji se nalaze u određenom redu. Ako je potrebno ažurirati vrednosti u tabeli, moguće je ažurirati sve redove ili samo neke od njih, u zavisnosti od filtera koje se primenjuje. U relacionim bazama podataka imena kolona mogu biti data samo u vidu stringa, dok u *Cassandra* modelu ne postoji to ograničenje. I ključevi koji se odnose na red i imena kolona mogu da budu u vidu stringa ili da budu celi brojevi, ili pak bilo kakav niz bajta.

Najjednostavniji sistem skladištenja podataka može biti niz ili lista, kao što je pokazano na slici 4.1.



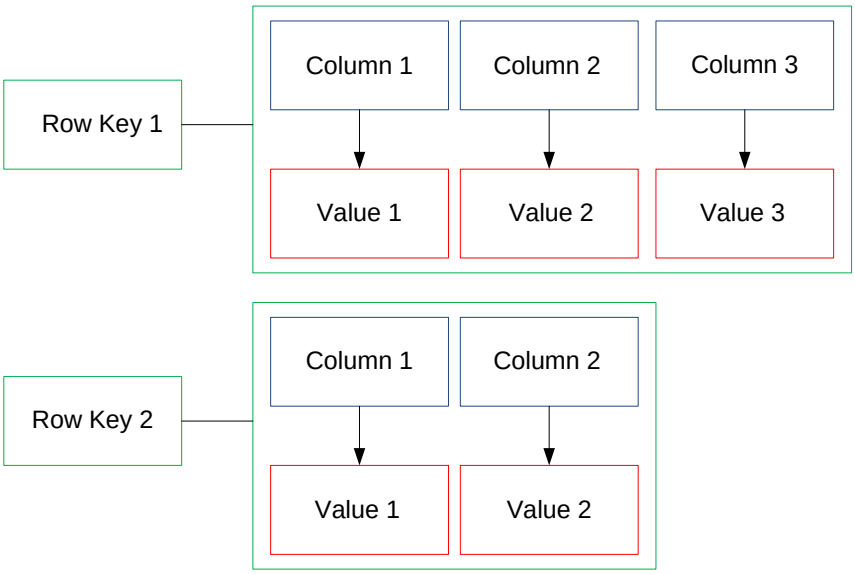
Slika 4.1 Lista vrednosti.

Postoje sistemi koji imaju drugu dimenziju u nizu, odnosno listi imena koja odgovaraju vrednostima. Svaka ćelija dobija ime i na taj način se formira struktura mape, kao na slici 4.2.



Slika 4.2 Mapa parova ime/vrednost.

Cassandra zahteva da se definiše spoljašnji kontejner, koji se zove *keyspace*, koji sadrži familiju kolona (*column family*). *Keyspace* je u suštini samo logički prostor za čuvanje familija kolona i nekih svojstava vezanih za konfiguraciju. U *Cassandr*a modelu postoje redovi sa jedinstvenim ključem, koji sadrži parove kolona/vrednost. Organizacija familije kolona je data na sledećoj slici.



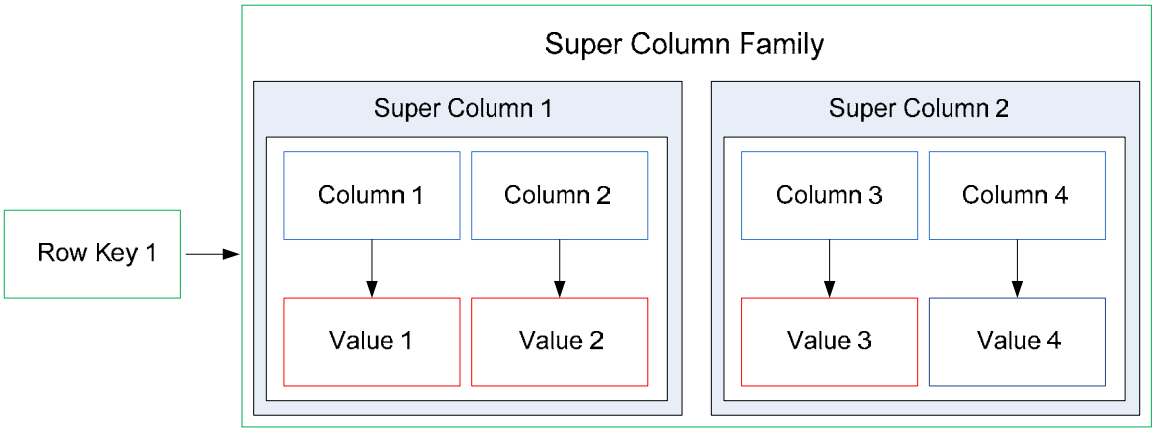
Slika 4.3 Familija kolona.

Familija kolona kod *Cassandre* odgovara tabelama u relacionim bazama podataka i predstavlja kontejner za kolekciju podataka. Svaki red sadrži uređene kolone. Familija kolona je kontejner za prikupljanje uređenih redova, od kojih je svaki po sebi uređena kolekcija podataka. U relacionom svetu, fizički se kreiraju baze podataka, odredi se ime baze podataka (*keyspace*), imena tabela (slično familiji kolona), a zatim da se definišu imena kolona koje će se nalaziti u svakoj tabeli. *Cassandra* je otvorena šema kod koje se definišu familije kolona, ali ne i same kolone. *Cassandra* omogućava dodavanje bilo koje kolone bilo kojoj familiji kolona u bilo koje vreme, u zavisnosti od potreba korisnika. Umesto čuvanja *null* za one

vrednosti koje nam nisu potrebne, što bi bespotrebno zauzimalo prostor, neće se uopšte skladištiti kolone za taj red. Takođe, familija kolona ima dva atributa: ime i komparator. Vrednost komparatora ukazuje kako će kolone biti sortirane kada se vrate korišćenjem upita bajt, *UTF8*, ili drugih tipova uređivanja.

U relacionim bazama podataka korisnik zna gde se tabele nalaze na disku, dok se familije kolona pojedinačno čuvaju u posebnim datotekama na disku, samo je važno da se sačuvaju veze kolone koje su definisane u istoj familiji kolona.

U *Cassandra* modelu tabele mogu da sadrže kolone, ili se mogu definisati kao familije super kolona.



Slika 4.4 Familija super kolona.

Familija super kolona je posebna vrsta kolone. Obe vrste kolona predstavljaju par ime/vrednost, ali regularna kolona skladišti niz vrednosti bajta, a vrednost super kolone je mapa podkolona (koji skladište niz vrednosti bajta). Treba imati na umu da one skladište samo mape kolona, ne mogu se definisati super kolone koje skladište mapu drugih super kolona. Dakle, super kolona ide samo jedan nivo u dubinu, ali zato je moguće imati neograničen broj kolona.

Osnovna struktura super kolone je njeno ime, koja je niz bajtova (kao i kod redovnih koloni), i u njoj su uskladištene kolone. Kolone u super kolona strukturi sadrže mapu čiji ključevi su imena kolona i čije vrednosti su kolone. Svaka familija kolona se čuva na disku u svom posebnom fajlu. Dakle, da bi se optimizovale performanse, važno je da kolone, na koje se odnosi upit, budu zajedno u istoj familiji kolona kako bi super kolona bila korisna za nju.

Može se reći da je *Cassandr*a prsten kome se podaci dodeljuju u vidu čvorova. *Cassandra API* klijent/ server aplikacije, sastoji se od sledeće tri jednostavne metode:

- *insert (table; key; rowMutation)*,

- *get (table; key; columnName) i*
- *delete (table; key; columnName).*

Argument *columnName* u *get* i *delete* operaciji identifikuje familiju kolona ili familiju super kolona.

Korisnik određuje kako će imena kolona biti sortirana kada se rezultat vraća klijentu. Za sortiranje kolona koristi se jedna od sledećih opcija: *ASCII, bajt, LexicalUUID, tip celog broja, dugi, TimeUUID ili UTF8.*

Ascii tip direktno upoređuje bajtove. Ulaz može biti analiziran kao US-ASCII. *US-ASCII* kodiranje znakova je mehanizam zasnovan na leksičkom nalogu engleskog alfabeta. On definiše 128 znakova, od kojih 94 mogu biti štampani.

Tip *bajt* je podrazumevana opcija, i sortiranje se obavlja direktnim upoređivanjem bajtova, preskakanjem koraka validacije bajtova. Ovo je podrazumevani tip iz razloga što on obezbeđuje ispravno sortiranje za većinu tipova podataka (*ASCII i UTF-8 uključeni*).

LexicalUUID predstavlja 16-bajtni (128-bitni) univerzalni jedinstveni identifikator (*UUID*) koji vrši leksikografsko poređenje (po vrednosti bajta).

Dugi tip vrši sortiranje po 8 bajtnom (64 bita) dugom numeričkom tipu.

Tip celog broja je uveden od 0.7 verzije, brži je nego dugi tip celih brojeva i dozvoljava manje ili više bita od 64-bitnog sortiranja kod dugog tipa.

UTF-8 enkoder karaktera može izgledati kao dobar podrazumevani tip zato što je pogodan za programere koji su navikli da koriste *XML* ili druge mehanizme za razmenu podataka koji zahteva uobičajeno kodiranje. U *Cassandra* modelu bi trebalo koristiti *UTF8* tip samo za proveru validnosti podataka.

Moguće je kreirati sopstveni mehanizam za sortiranje kolone. Kao i mnoge stvari u *Cassandra* modelu i ovo je dodatak koji se ostvaruje proširenjem

`org.apache.cassandra.db.marshal.`

Postoji nekoliko razlika između *NoSQL Cassandra* modela i relacionih *DBMS*- a:

- *Upiti. SQL* je standardni jezik zasnovan na upitima koji se koriste u relacionim bazama podataka. *Cassandra* ne podržava upite. Ona ima *API* kojem se pristupa preko *RPC* mehanizma, *Thrift.*
- *Povezivanje po referenci. Cassandra* nema koncept povezivanja po referenci te nema ni koncept pridruživanja. U relacionoj bazi podataka je moguće odredite strane ključeve u tabeli koji će da upućuju na primarni ključ zapisa u drugoj tabeli, dok kod *Cassandre* to nije slučaj.

- *Sekundarni indeksi.* Ako treba pronaći jedinstveni ID za neku promenljivu, u relacionoj bazi podataka je moguće koristite upit *“SELECT imeID FROM Tabela WHERE name = 'ime'“*. Ovaj upit je moguće koristiti ako se zna ime koje se traži, ali ne i jedinstveni ID. Kada se zada ovakav upit relaciona baza podataka će izvršiti skeniranje cele tabele i pretražiti kolone svih zapisa. Ovo može da postane vrlo sporo kada tabela ima veliki broj zapisa i raste veoma brzo, kao što je slučaj kod triangulacije konveksnih poligona. Rešenje za ovo je da se stvori indeks imena kolone, koja predstavlja kopiju podataka i koju relaciona baza podataka može da pregleda veoma brzo. Da bi se postigla ista stvar u *Cassandri*, potrebno je kreirati drugu familiju kolona koja deluje kao eksplicitan sekundarni indeks.
- *Upiti za ažuriranje podataka. Cassandra* model ne podržava koncept ažuriranja podataka, što znači da ne postoji mogućnost da klijent zada upit, ali se isti efekat lako i jednostavno postiže kada se na postojeći ključ reda ubaci podatak. Pri unosu podataka za već postojeći ključ *Cassandra* će zameniti vrednosti za bilo koje odgovarajuće kolone, a ako upit sadrži dodatne kolone koje ne postoje za dati ključ reda te kolone će biti umetnute.
- *Duple vrednosti ključeva.* U *SQL* bazi podataka je moguće umetnuti više od jednog reda sa identičnim vrednostima, ako nije definisan jedinstveni primarni ključ na jednu od kolona. Ovo nije moguće u *Cassandra* modelu. Pri unosu novog zapisa, sa ključem koji već postoji u familiji kolona, sve vrednosti za postojeće kolone će biti zamenjene, a sve kolone koje nisu postojale će biti dodate datom redu.
- *Osnovne osobine unosa podataka.* Unos podataka je veoma brz u *Casandra* modelu jer je dizajniran da ne zahteva čitanje diska. *MemTabele* i *SSTabele* čuvaju obavljanje operacije pisanja u *Casandra* modelu, koje usporavaju mnoge baze podataka.

Konekcija PHP i *Cassandra API*-ja se vrši preko *API-ja Thrift*. Programske biblioteke *PHP, Java, Ruby, Python C # ...* su dostupni za interakciju sa *Cassandrom*. Jednostavno rečeno, *Apache Thrift* je binarni komunikacioni protokol. *Thrift* je interfejs programskog jezika koji se koristi za definisanje i stvaranje usluga za veliki broj jezika. Koristi se kao daljinski poziv procedure *(RPC)* okvir i razvijen je na *Facebook*-u za "razvoj skalabilne usluge međusobnog povezivanja jezika". On kombinuje softver sa mašinom za generisanje koda za izgradnju servisa koje efikasno rade između *C #, C +, Erlang, Go, Haskell, Java, Ocaml, Perl, PHP, Python, Ruby, and Smalltalk.* Iako razvijena na *Facebook*-u, *Cassandra* je sada projekat otvorenog koda u *Apache Softvare Foundation.*

4.1.1. Modeliranje jedinstvenog MatrixDBMS

Na osnovu rezultata testiranja u [46] odlučili smo se za realizaciju *MatrixDBMS* sistema skladištenja u *Cassandra* modelu podataka. *MatrixDBMS* sistem sadrži jednu ulaznu familiju kolona za sve vrste matrica, i jednu izlaznu familiju kolona za rezultate obrade. Predstavljene familije kolona odgovaraju tabelama u *MySQL*-u. Pored ulazne i izlazne familije kolona *Cassandra keyspace* sadrži i jednu familiju super kolona za triangulacije konveksnih poligona. Model je dat sledećim kodom.

```
$servers = array(
array('host' => '127.0.0.1',
      'port' => 9160,
      'use-framed-transport' => true,
      'send-timeout-ms' => 1000,
      'receive-timeout-ms' => 1000));
$cassandra = Cassandra::createInstance($servers);
$cassandra2 = Cassandra::getInstance();
$cassandra->createKeyspace('MatricesComputation');
$cassandra->useKeyspace('MatricesComputation6');
$cassandra->setMaxCallRetries(5);
$cassandra->createStandardColumnFamily(
   'MatricesComputation',
   'matricesIn',
   array(array('name' => 'elements_in',
               'type' => Cassandra::TYPE_UTF8,
               'index-type' => Cassandra::INDEX_KEYS,
               'index-name' => 'elementsIdx'),
         array('name' => 'dimension',
               'type' => Cassandra::TYPE_UTF8,
               'index-type' => Cassandra::INDEX_KEYS,
               'index-name' => 'dimIdx'),
        array('name' => 'type',
              'type' => Cassandra::TYPE_UTF8)));
$cassandra->createStandardColumnFamily(
   'MatricesComputation6',
   'matricesOut',
    array(array('name' => 'elements_out',
                'type' => Cassandra::TYPE_UTF8,),
          array('name' => 'matrix1',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'mat1Idx'),
          array('name' => 'matrix2',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'mat2Idx'),
          array('name' => 'matrix3',
```

```
                'type' => Cassandra::TYPE_INTEGER,
                'index-type'=> Cassandra::INDEX_KEYS,
                'index-name' => 'mat3Idx'),
          array('name' => 'operation',
                'type' => Cassandra::TYPE_UTF8,
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'opIdx'),
          array('name' => 'r',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'rIdx'),
          array('name' => 's',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type'=> Cassandra::INDEX_KEYS,
                'index-name' => 'sIdx'),
          array('name' => 'p',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type'=> Cassandra::INDEX_KEYS,
                'index-name' => 'pIdx'),
          array('name' => 'q',
                'type' => Cassandra::TYPE_INTEGER,
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'qIdx')));
$cassandra->createSuperColumnFamily(
    'MatrixComputation',
    'triangulation',
    array(array('name' => 'triangulation',
                'type' => Cassandra::TYPE_UTF8),
                'index-type' => Cassandra::INDEX_KEYS,
                'index-name' => 'nameT'
         array('name' => 'id',
               'type' => Cassandra::TYPE_INTEGER)),
Cassandra::TYPE_UTF8,
Cassandra::TYPE_UTF8,
Cassandra::TYPE_UTF8,
'Capitals supercolumn test',
1000,1000,0.5);
```

Dakle, M*atrixDBMS* model sadrži *keyspace MatrixComputation* i familiju kolona *MatrixIn i MatrixOut* u okviru *keyspace*-a, sa definisanim sekundarnim indeksima `dimidx` i `opidx` za polja `dimension` i `operation`. Svaki zapis sadrži elemente i dimenziju matrice. Izgled familije kolona *MatrixIn* dat je na sledećoj slici.

Key	Value	
3	dimension	116x150
	elements_in	178,185,77,161,149,66,137,...
	type	RGB
2	dimension	6x6
	elements_in	282,11,206 ...,2,219,15,184
1	dimension	11x10
	elements_in	11,10,9,...,1,1,1,0,1
	type	A_11_10

Slika 4.1.1 Ulazne matrice u MatrixDBMS modelu.

Na sledećoj slici je prikazana familija super kolona *MatrixTriangl, MatrixDBMS* sistema skladištenja, koji se koristi za smeštanje triangulacija konveksnog poligona. U polje *Value* unosimo sve triangulacije za n.

Key	Value	
2	triangulation_6 Delete	
	...	...
	1	2,4,2,5,2,6
	2	2,5,2,6,3,5
	3	2,4,2,6,4,6
	13	3,1,4,1,4,6
	14	3,1,4,1,5,1
1	triangulation_5 Delete	
	1	2,4,2,5
	2	2,5,3,5
	3	2,4,4,1
	4	3,1,3,5
	5	3,1,4,1

Slika 4.1.2 Super kolone u MatrixDBMS modelu.

Problem na koji smo naišli pri implementaciji *MatrixDBMS*, u *Cassandra* modelu podataka, je bio unos ogromne matrice koja predstavlja sliku. Tada bi dobili obaveštenje *"Error while getting key> Key length of – is longer than maximum of 65535"*. Problem smo rešili definisanjem sekundarnih indeksa i unosom matrice u polje koje nije definisano kao ključ. Na sledećoj slici je prikazana kolona familija koja sadrži rezultate obrađenih matricama.

Key	Value	
3	elements_out	0.755,-0.156,...,0.526,-0.2
	matrix1	18
	matrix2	14
	matrix3	17
	operation	AMN
2	elements_out	1974,-77,...,105,1288
	matrix1	2
	matrix2	2
	operation	rA+sB
	r	3
	s	4
1	elements_out	1,-1,0,...,-0.25,-0.417
	matrix1	1
	operation	A-^1
4	elements_out	140,150,157,168,175,17,66,161,172,...
	matrix1	3
	operation	FILTER

Slika 4.1.3 Izlazne matrice u MatrixDBMS modelu.

Na prethodnim slikama su prikazane familije kolona ulaznih i izlaznih matrica, kao i familija super kolona sa triangulacijom konveksnih poligona. Zapisi 1 i 2 u familiji kolona *MatrixIn* sadrže matrice *A* i *B* iz Primera 2.2.1, dok zapis 4 u familiji kolona *MatrixOut* predstavlja matricu koja reprezentuje sliku iz familije kolona *MatrixIn*, zapis 6, i rezultat je sprovođenja filtriranja ulazne slike.

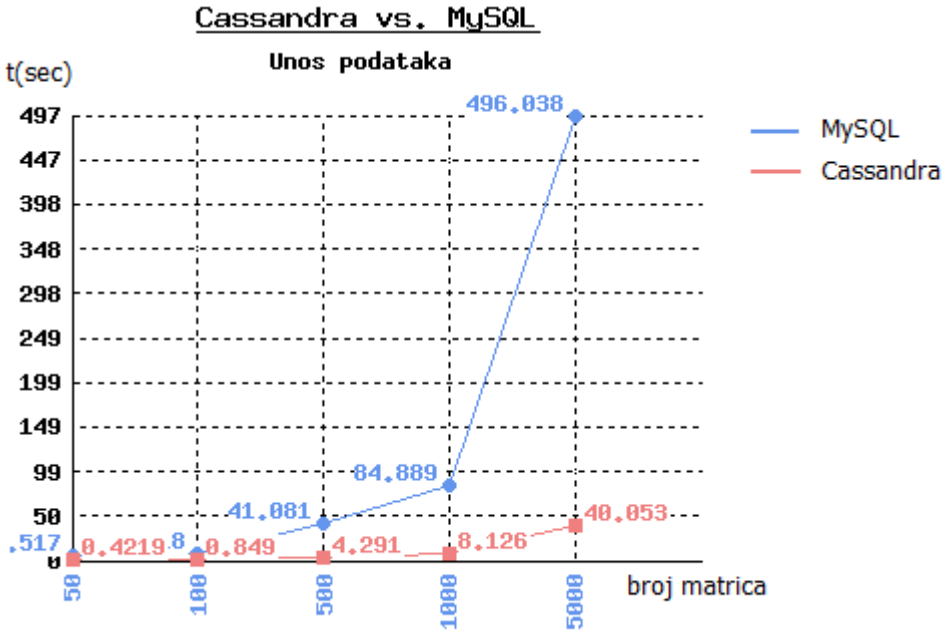
4.1.2. Rezultati testiranja

Primer 4.1.1. U sledećoj tabeli su prikazani dobijeni razultati uporednih vremena za unos matrice u dva potpuno različita sistema za skladištenje podataka, *MySQL* baza podataka i ključ-vrednost sistem skladištenja, *Apache Cassandra* [46].

Tabela 4.1.1 Unos podataka u MySQL i Cassandra DB.

UPIS podataka			
dimenzije	**NOR**	**DB MySQL**	**CASSANDRA**
80x80	5000	496.038	40.053
80x80	1000	84.889	8.126
80x80	500	41.081	4.291
80x80	100	8.018	0.849
80x80	50	4.517	0.4219

Grafik 4.1.1 MySQL vs Cassandra: Unos podataka.



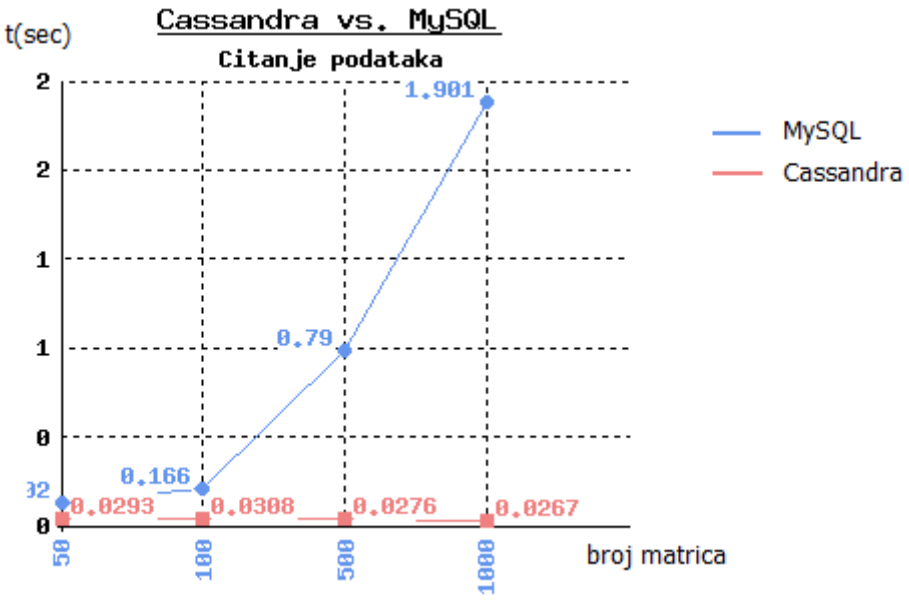
CPU izvršna vremena, predstavljena u tabeli 4.1.1, pokazuju mnogo bolje rezultate u slučaju ključ-vrednost sistema skladištenja, kakva je *Cassandr*a.

Primer 4.1.2 Uporedne vrednosti pretrage matrica kada su uskladištene u dva različita sistema skladištenja, *MySQL* i *NoSQL* baze podataka, sa dati u sledećoj tabeli [46].

Tabela 4.1.2 Pretraga podataka u MySQL i Cassandra DB.

Pretraga uskladištenih matrica (Čitanje podataka)		
broj matrica	**R format MySQL**	**R format CASSANDRA**
50	0.102	0.0293
100	0.166	0.0308
500	0.790	0.0276
1000	1.901	0.0267

Grafik 4.1.2 MySQL vs Cassandra: Pretraga podataka.



Vreme za pretraživanje uskladištenih matrica je mnogo kraće kada je matrica uskladištena u ključ-vrednost sistemu skladištenja, tj. *Cassandra* modelu podataka.

4.2. Paralelno programiranje

Kada treba rešiti mnogo zadataka koji se ponavljaju, a moguće ih je podeliti na više nezavisnih delova, istovremenim multiprocesiranjem (pokretanjem više procesa) moguće je povećati brzinu obrade.

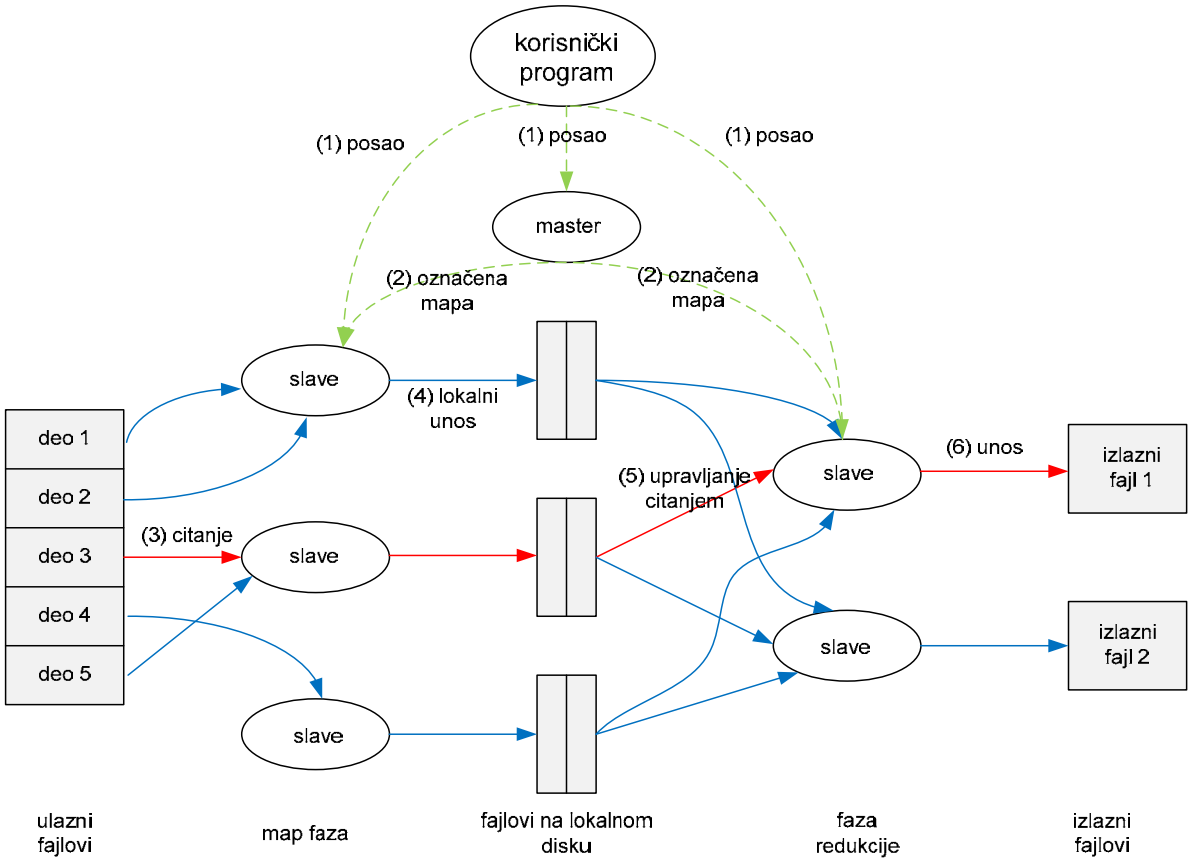
Prvi korak u paralelnom programiranju je identifikovanje skupa zadataka koji mogu da rade istovremeno, odnosno identifikovanje zadataka koji se mogu istovremeno obrađivati. U praksi se javljaju problemi koje je moguće brže rešiti primenom tehnike paralelnog programiranja kao što je slučaj sa triangulacijom konveksnih poligona.

Ako je ista obrada neophodna za svaki element niza, a elementi niza nisu međusobno povezani, postoji idealna prilika za sprovođenje tehnike paralelnog programiranja. U okviru

paralelnog programiranja je česta primena tehnike gospodar/rob (*master/slave*). Gospodar radi sledeće: deli niz prema broju raspoloživih robova, svakom robu šalje podniz i dobija rezultate obrade. Rob dobija niz od mastera, vrši obradu podniza i vraća rezultate gospodaru.

Koncept paralelnog programiranja se sreće kod *MapReduce* modela. On je razvijen u okviru Googla kao mehanizam za obradu velike količine nestrukturiranih podataka. Podaci koji se obrađuju su toliko veliki da se moraju distribuirati preko nekoliko hiljada mašina kako bi se obradili u razumnom roku. Ova distribucija podrazumeva paralelne proračune jer se isti proračuni, ali sa različitim podacima, izvode na svakom procesoru.

Karta koju napiše korisnik *MapReduce* biblioteke proizvodi niz posrednih parova ključ/vrednost. U ovom modelu se vrši distribucija na više modela automatskim particionisanjem ulaznih podataka u skup od n delova. Ulazni podaci se na ovaj način obrađuju na različitim mašinama. Sledeća slika ilustruje protok podataka na *MapReduce* modelu. Kad korisnik poziva *MapReduce* funkcije počinje da se obavlja sledeći redosled operacija koje su numerisane na slici.



Slika 4.1.4 Proces paralelnog programiranja.

1. *MapReduce* biblioteka u korisničkom programu deli ulazni fajl na M delova. Nakon ovoga se pokreću kopije programa.

2. Jedan od delova programa je poseban i on je „gospodar”. Ostali delovi su “robovi” kojima je dodeljen posao od strane gospodara. Gospodar prati proces, traži robove koji nisu zauzeti i dodeljuje im zadatke.
3. Rob kojem je dodeljen zadatak čita sadržaj odgovarajućeg ulaznog podatka. To dovodi do formiranja para ključ/vrednost od ulaznih podataka, i prosleđuje svaki par na korisnički definisanu `map()` funkciju.
4. Povremeno se uskladišteni parovi ključ/vrednost dele na regione. Lokacije ovih uskladištenih parova na lokalnom disku se prosleđuju nazad gospodaru koji ih prosleđuje robovima.
5. Ako su podaci preveliki da stanu u memoriju koristi se spoljna memorija.
6. Kreiranje izlaznih fajlova.

U *PHP*-u je moguće koristiti usluge paralelnog programiranja na više načina: *fork, curl, multithread,* pokretanje u pozadini itd. Sva navedena podešavanja rade pod *Unix* operativnim sistemom.

Unix podržava proces koji omogućava `fork()` funkciju procesu koji je predstavljen kao roditelj i ima mogućnost da se deli na više delova koji predstavljaju njegovu decu. Pre korišćenja *PHP* funkcija za kontrolu procesa potrebno je u konfiguracijskom fajlu podesiti configure - enable - pcntl zajedno sa svim ostalim konfiguracionim opcijama. Potrebno je instalirati i dodatne biblioteke. Pozivom funkcije `pcntl_fork()` sve promenjive i objekti se kopiraju sa procesa roditelj na proces dete i predstavljaju nove kopije koje pripadaju novom procesu. Izmena u procesu deteta ne utiče na vrednost roditelja, ili bilo kog drugog odvojenog, procesa. Proces roditelj će imati vrednost označenu sa `$pid`, a proces dete je neće imati. Proces roditelj mora da čeka dok se dete proces ne završi, što omogućava funkcija `pcntl_vaitpid()`. Poziv funkcije `exit()`, u okviru matičnog procesa, onemogućava pokretanje nekog koda namenjenog detetu. Parametar `$status` prosleđuje `pcntl_vaitpad()` skladištu povratnu vrednost procesa dete.

Za korišćenje *cURL* funkcija je potrebno koristiti usluge *HTTP* ili *FTP* protokola. *Multithread* ili pokretanje više niti je proces koji podrazumeva paralelno pokretanje više poslova. Za funkcionisanje *cURL* funkcija, na serveru, potrebna je instalirati *PHP 5.0* verziju. Dakle, ovaj pristup prvo kreira veze ka serverima sa kojih želimo da dobijemo informacije.

Primena principa paralelnog programiranja data je na primeru triangulacije konveksnih poligona. Primena paralelnog izvršavanja skripti se vrši tako što fajlove sa triangulacijama delimo na više manjih, a zatim nad njima sprovodimo dekompoziciju. Na

ovaj način skraćujemo vreme obrade. Zapravo, suština ovog načina programiranja ogleda se u pokretanju više skripti paralelno koje "rade" u pozadini. U radu smo koristili pristup za paralelnu obradu pokretanjem skripta sa komandne linije. Na ovaj način se skripte paralelno procesiraju u pozadini. Ova tehnika radi jedino kad skript pokrećemo sa komandne linije pod *Unix/Linux* operativnim sistemom. Funkcija `exec()` služi za izvršavanje komandi koje joj se zadaju (*Funkcija B.2.1*, *Funkcija B.2.2*). Kad *PHP* naiđe na komandu čita argument i izvršava ga. Moguće je izvršiti preusmeravanje izlaza komande da radi u pozadini tako što se na kraju komande unese ampersand *&*., tako da umesto

```
exec("ls -al");
```

unosimo

```
exec("ls -al > delovi.txt &");.
```

Na ovaj način će skript stvoriti novi proces, koji se izvršava u pozadini, i zatim odmah preći na novi zadatak. Kad se *PHP* skript pokreću sa komandne linije, uvek će podržati superglobalne promenljive `$arg`, koja predstavlja niz argumenata koji se koriste kad je skript pokrenut. Ako imamo skript `triangl.php`, pokrećemo ga sa komandne linije na sledeći način

```
php triangl.php arg1 arg2 arg3,
```

tada je promenjiva $arg predstavljena nizom

```
array(4){
        [0]=>string(11) "triangl.php"
        [1]=>string(4) "arg1"
        [2]=>string(4) "arg2"
        [3]=>string(4) "arg3"}.
```

Kombinujući prethodne slučajeve moguće je kreirati skript koji će sam sebe pozivati više puta. U slučaju triangulacije konveksnih poligona imamo više desetina hiljada, čak i miliona triangulacija koje su smeštene u bazi podataka i koje treba obraditi. Umesto da koristimo jedan skript koji obrađuje sve podatke od prvog do poslednjeg, korišćenjem prethodne logike, možemo taj proces podeliti u više njih koji obrađuju pojedinačne delove podataka smeštenih u tabeli. Na isti način možemo da obrađujemo i podatke smeštene u fajlu tako što fajl podelimo na više delova i obrađujemo ih pojedinačno. Rezultate obrade, takođe, smeštamo u fajl. Pošto pri triangulaciji dolazi do pojave duplikata moramo koristiti filtriranje u vidu baze podataka i jedinstvenog ključa kako bi dobili "čistu" strukturu. Dakle, prvo ćemo izvršiti podelu fajla, na onoliko delova koliko ćemo istovremeno pokrenuti skripti, jednostavnim unosom putanje fajla, mesta gde želimo da smestimo delove i broja linija po fajlu.

```
require_once("podelafajla.php");
    $s = new spleetfiles;
$s->Setsource("./files/1.txt");
$s->Setpath("./files/");
$s->Setlines(5879);
    $s->configure("logs.txt", "logs/", 2000);*/
$s→run();
```

Nakon deljenja fajla sledi pozadinsko pokretanje istih i primena odgovarajuće procedure.

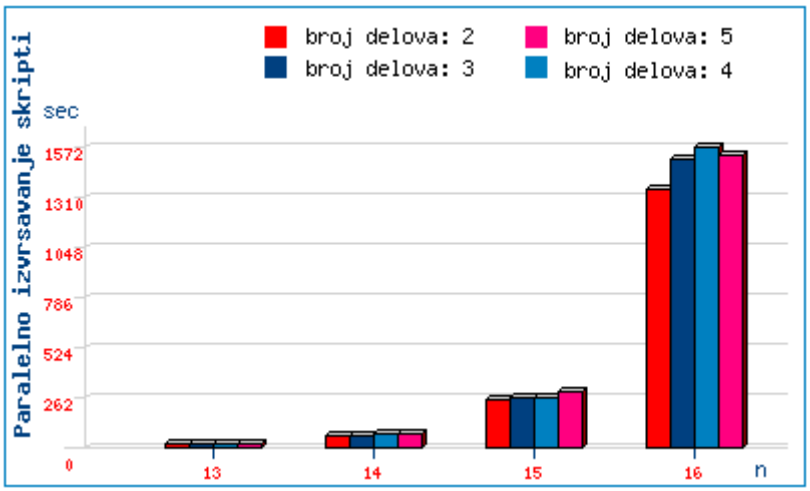
```
function parallel($arg){
if ($argv[1]) {
    $n=14;
 load_file_data($argv[1]);}
else {
    for ($i = 1; $i <= 10; $i += 1)
 exec("php script.php $i > $i.log &");}}
```

U cilju skraćivanja vremena izračunavanja, za velike vrednosti n, delili smo ulazni fajl na više delova koje smo unosili u bazu radi čišćenja duplikata. U sledećoj tabeli su prikazani rezultati testiranja iz kojih je moguće videti da najbolje rezultate dobijamo kad je fajl podeljen na dva dela.

Tabela 4.2.1 Izvršavanje skripti po delovima.

delovi	n=13	n=14	n=15	n=16
2	12.122	55.388	246.758	1342.577
3	12.568	55.463	259.880	1499.280
4	12.865	63.737	260.563	1530.782
5	14.643	64.237	287.844	1571.957

Grafik 4.2.1 Izvršavanje skripti po delovima.



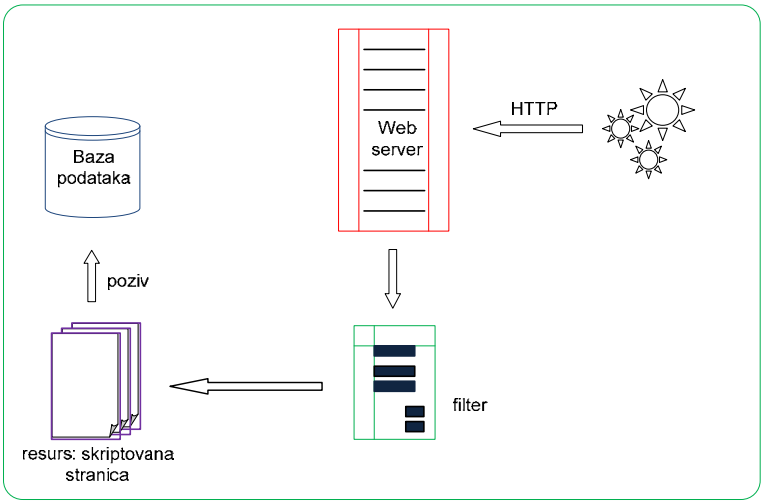
Glava 5

5. Dizajn i implementacija aplikacije

5.1. Troslojna web arhitektura

Klijentski sloj, u troslojnoj web arhitekturi, pruža interfejs korisniku aplikacije. Realizuje se pomoću web čitača (browser), koji sa srednjim slojem komunicira preko *HTTP* protokola. Browser je u stanju da prihvati korisnički input i da od njega kreira odgovarajući *HTTP* upit, koji zatim prosleđuje srednjem sloju. Tamo ga dočekuje web server, koji zahtev obrađuje i vraća odgovor klijentu. Klijent/server aplikacija kreira stranicu na osnovu podataka iz baze podataka. Stranicu može da kreira neki modul web servera, koji u opštem slučaju možemo nazvati filter stranice. U ovom slučaju se unutar stranice nalazi programski kod koji modul tj. filter treba da izvrši. Sam modul može biti implementiran na dva načina [17]:

- *CGI (Common Gateway Interface) i*
- *SAPI (Server API, odnosno Server Aplication Programming Interface)*



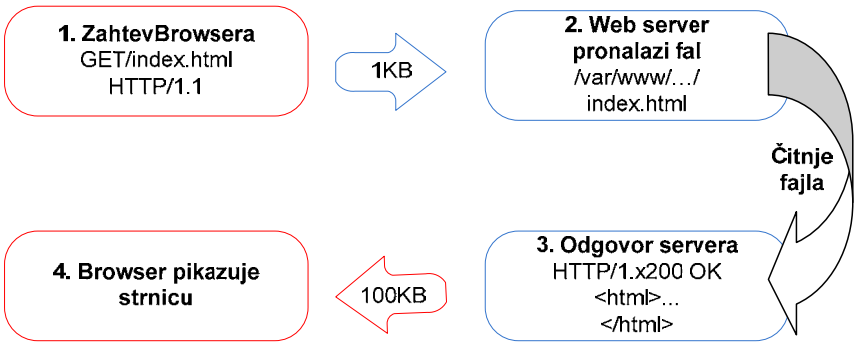
Slika 5.1.1 Arhitektura dinamičkog web sajta.

Slika 5.1.1. pokazuje arhitekturu web aplikacije čiji se sadržaj kreira u toku rada. Kod *CGI* implementacije filter se nalazi na serveru kao zasebna aplikacija koju web server poziva kroz rutine operativnog sistema i prosleđuje mu parametre, a zatim prihvata sve izlaze od istog, obrađuje ih i prosleđuje klijentu. *SAPI* arhitektura označava da je filter ugrađen u sam web server, gde on može biti sastavni deo sistema ili eksterna biblioteka funkcija koja se poziva u toku rada.

5.1.1. Protokol za prenos hiperteksta HTTP

Vezu između čitača Weba i Web servera obezbeđuju protokoli i mreža koji povezuju klijentski i srednji sloj aplikacije.

Sve slojeve u troslojnoj arhitekturi povezuje protokol *HTTP*. *HTTP* protokol vrši transfer zahteva za resursima, koje šalju Web čitači, i odgovore na te zahteve, koje šalju Web serveri. *HTTP* odgovor prenosi zahtevani resurs (*HTML* dokument, sliku ili izlazne podatke nekog programa) natrag do čitača weba kao “koristan tovar”. Jednostavan model zahtev–odgovor prikazan je na slici 5.1.2.



Slika 5.1.2 Sistem zahtev odgovor.

Naša aplikacija radi s bazama podataka i ima potrebu za čuvanjem stanja. Za razliku od PHP programskog jezika koji čuva stanje, *HTTP* to ne čini. Drugim rečima, svaka interakcija između čitača weba i web servera je nezavisna od bilo koje druge interakcije.

Posmatrajući iz perspektive mreže, *HTTP* predstavlja protokol aplikacionog sloja koji se nalazi neposredno iznad seta *TCP/IP* mrežnih protokola.

5.1.2. XML proširivi znakovni jezik

Istorija *XML*-a počinje 60-ih godina prošlog veka, kada je u *IBM*-u konstruisan prvi višenamenski jezik za serializaciju podataka [16]. Ovaj jezik zvao se *GML (Generalized Markup Language).*

XML predstavlja jedan od čestih pojmova upotrebljavanih u kontekstu današnjeg programiranja. *To* je svojevrsan jezik namenjen za komunikaciju na Internetu. Takođe, *XML* je proširivi jezik zasnovan na oznakama, tagovima, koji pruža tekstualni format zapisivanja podataka nezavisan od računarske platforme. *XML* je osnovni jezik pomoću kojeg su oblikovani ostali standardni jezici koji pružaju mrežne usluge.

Za razliku od *HTML*-a gde oznake određuju način prikazivanja podataka, *XML* oznake nose informaciju o značenju podataka. Pored toga on predstavlja i tehnologiju za upravljanje, prikazivanje i organizovanje podataka.

S obzirom da se *XML* fajl može koristiti za mnoge stvari, a ne samo za web, postavljena su pravila kako jedan *XML* fajl mora da izgleda, a sve to da bi mogao ispravno da se koristi. Program koji učitava podatke iz *XML* fajla mora da razume šta znači koji podatak i da ga ispravno tretira. Sam *XML*-ov procesor se naziva parser, a podrazumeva se da ovaj *XML*-ov procesor svoj posao obavlja za neku drugu konkretnu aplikaciju. Uz veliku podršku *CSS*-a i *XSL(T)*-a, *XML* pruža neverovatne mogućnosti kreiranja dinamičkih web stranica [19]. *XML* se može povezati sa bazom podataka, koristi se za selektovanje podataka iz jedne ili više tabela, za smeštanje *XML*-a u bazu kao i za kreiranje višeslojnih aplikacija.

Kao i u ostalim jezicima i u *PHP*-u je sve više dobrih mehanizama za rukovanje *XML*-om *(DOM XML, XMLParser...).* Jedan od takvih mehanizama koristi klasu *SimpleXML*. Učitavnje fajla u *XML* objekat je jednostavno.

```
$xml=simplexml_load_file('proba.xml');.
```

Za pisanje u *XML*-u, nije potrebna nikakva posebna tehnologija. U sledećem primeru ćemo pokazati kako se vrši transformacija niza koji predstavlja elemente matrice u *XML* dokument i obratno.

Primer 5.1.1. Posmatrajmo matricu $A_{11\times10}$ [70] za slučaj $a = 1$. Za dati primer niz predstavljen u *2D_niz* formi je dat kao:

```
$array=array(array(1,2,3,4,1),
             array(1,3,4,6,2),
             array(2,3,4,5,3),
             array(3,4,5,6,4),
```

```
        array(4,5,6,7,6),
        array(6,6,7,7,8);
```

Primer 5.1.2. Pozivom funkcije *array2xml*() dobijamo *XML* dokument.

```
$myData = new multidi_array2xml("$Array");
print_r($myData->array2xml($Array));
```

Primenom odgovarajućih funkcija dobijamo izgled xml fajla koji sledi.

```
<?xml version="1.0" encoding="utf-8"?>
<matrix>
<row0>
   <column0>1</column0><column1>2</column1><column2>3</column2>
   <column3>4</column3><column4>1</column4></row0>
<row1>
   <column0>1</column0><column1>3</column1><column2>4</column2>
   <column3>6</column3><column4>2</column4></row1>
<row2>
   <column0>2</column0><column1>3</column1><column2>4</column2>
   <column3>5</column3><column4>3</column4></row2>
<row3>
   <column0>3</column0><column1>4</column1><column2>5</column2>
   <column3>6</column3><column4>4</column4></row3>
<row4>
   <column0>4</column0><column1>5</column1><column2>6</column2>
   <column3>7</column3><column4>6</column4></row4>
<row5>
   <column0>6</column0><column1>6</column1><column2>7</column2>
   <column3>7</column3><column4>8</column4></row5></matrix>
```

Primer 5.1.3. Pozivom procedure *createArray()* vršimo transformaciju podataka iz *XML* dokumenta u niz.

```
$xml_data = "xml fajl iz prethodnog primera";
//Kreiranje instance klase
$xmlObj    = new XmlToArray($xml_data);
//Kreiranje niza
$arrayData = $xmlObj->createArray();
//Prikaz niza
print_r($arrayData);
Array([matrix]=>
      Array([row0=>Array([0]=>Array(
      [column0]=>1[column1]=>2[column2]=>3[column3]=>4[column4]=>1))
      [row1]=>Array([0]=>Array(
      [column0]=>1[column1]=>3[column2]=>4[column3]=>6[column4]=>2))
      [row2]=>Array([0]=>Array(
      [column0]=>2[column1]=>3[column2]=>4[column3]=>5[column4]=>3))
      [row3]=>Array([0]=>Array(
      [column0]=>3[column1]=>4[column2]=>5[column3]=>6[column4]=>4))
      [row4]=>Array([0]=>Array(
      [column0]=>4[column1]=>5[column2]=>6[column3]=>7[column4]=>6))
      [row5]=>Array([0]=>Array(
      [column0]=>6[column1]=>6[column2]=>7[column3]=>7[column4]=>8)))
```

5.1.3. Klijentski sloj

5.1.3.1. Web browser kao klijent

Web čitač (*browser*) je samostalna aplikacija pokrenuta na klijenskoj mašini, tj. računaru krajnjeg korisnika sistema. To je program koji se koristi kao interfejs između krajnjeg korisnika i sistema. Browser mora da ima dve osnovne sposobnosti:

- da komunicira sa web serverom preko *HTTP* protokola i
- da vrši rendering (raspoređivanje i iscrtavanje) sadržaja na ekranu.

Komunikacija web browsera sa serverom se obavlja tako što korisnik u aplikaciji unese adresu (*URL - Uniform Resource Locator*) sadržaja koji želi da vidi. Noviji web čitači imaju mogućnosti da prikažu *DHTML stranice (Dynamic HTML),* upotrebom nekih od skript jezika *(JavaScript, VBScript) i CSS-a (Cascading Style Sheets).* Mogućnost upotrebe ovih tehnika može znatno da poveća komfor, interaktivnost i produktivnost prilikom rada sa web aplikacijom.

Trenutno, najpopularniji web čitači su *Microsoft Internet Explorer, Mozilla, Opera, Konqueror, Netscape Navigator i Safari.* Aktuelne verzije najpoznatijih web browsera su:

- *Internet Explorer: IE 9,*
- *Firefox FF: 4.0,*
- *Google Chrome: 13.0,*
- *Safari Safari: S 5* i
- *Opera: O 11.*

5.1.3.2. JavaScript

JavaScript je programski jezik koji spada u grupu skript jezika za koje je karakteristično da se interpretiraju tokom izvršavanja. Ova grupa programskih jezika je veoma pogodna za razvoj Internet aplikacija. Osim skript jezika postoje i kompajlirani jezici kod kojih se kompajliranje i interpretiranje vrše pre izvršavanja.

Kada su u pitanju kompajlirani jezici, programer piše izvorni kod koji predstavlja tekst, a zatim ga propušta kroz kompajler koji ga prevodi u mašinski jezik, kojeg računar razume. Posle kompajliranja i interpretacije kompajlirani kod se izvršava. Program kompajliran na ovaj način je vezan za određeni operativni sistem, mada je kod veoma brz.

Tako da se kod koji je kompajliran za računare na kojima se izvršava *Windows* neće izvršavati na računarima sa *UNIX*-om. Kod skript jezika ne postoje kompajleri i nije zavisan od platforme na kojoj se izvršava i uvek ostaje u obliku izvornog koda. Skript jezici poseduju softver za izvršavanje skriptova, tzv. *scripting engine* koji interpretira kod tokom izvršavanja, i na taj način se izvršavanje usporava. Usporavanje je posledica činjenice da računar prevodi kod na mašinski jezik red po red. Takođe, program izvršava kod red po red. Sa druge strane skript kod je u većoj meri prenosiv, pošto može da se izvršava na svakom računaru koji ima mašinu za skriptovanje.

Dakle, kod pisan u *PHP*-u ili *JavaScript*-u može da se izvršava na *Windows*-u i *UNIX*-u ili *Macintosh* -u, jedino je potrebna mašina za skriptovanje. *JavaScript* smo koristili pri implementaciji sistema zaštite, kojim se korisnicima omogućava, odnosno onemogućava, unos određenih karaktera.

5.1.4. Srednji sloj

U većini troslojnih sistema baza podataka na Webu, najveći deo logike aplikacije nalazi se u srednjem sloju. Klijentski sloj prikazuje podatke korisniku i prihvata podatke koje on unosi, dok sloj baze podataka skladišti i učitava podatke. Srednji sloj obavlja većinu preostalih zadataka pomoću kojih se objedinjuju ostali slojevi: upravlja strukturom i sadržajem podataka koji se prikazuju korisniku i obrađuje podatke dobijene od korisnika pretvarajući ih u upite za učitavanje ili upisivanje u bazu podataka. Takođe, ovaj sloj omogućava upravljanje stanjem uz upotrebu *HTTP* protokola. Logika aplikacije ugrađena u srednji sloj integriše Web sa sistemom za upravljanje bazom podataka. Komponente srednjeg sloja su web server, programski jezik za pisanje web skriptova i mašina za izvršavanje skriptova.

Za pisanje skriptova koji rade u srednjem sloju koristili smo programski jezik *PHP*. Budući da je *PHP* nastao kao projekat otvorenog koda, iza koga stoji *Apache Software Foundation*, ne iznenađuje što je to najpopularniji dodatni modul za *Apache HTTP server*, sa oko 40% *Apache HTTP servera* koji podržavaju *PHP* [68]. *PHP* je izuzetno prikladan za Web aplikacije zahvaljujući alatkama za integraciju Weba i okruženja baze podataka. Izuzetna fleksibilnost skriptova ugrađenih u *HTML* stranice omogućava laku integraciju s klijentskim slojem. Podrška za integraciju sa slojem baze podataka takođe je odlična zahvaljujući više od 15 ugrađenih biblioteka [34] koje omogućavaju povezivanje sa skoro svim poznatim sistemima za upravljanje bazama podataka.

5.1.4.1. Apache Web server kao komponenta sloja aplikacije

Web server je serverski program koji se izvršava na računaru - serveru koji ima zadatak da prihvata *HTTP* zahteve klijenata i da daje odgovore na njih. Komunikacija između servera i klijenata se odvija tako što server očekuje klijentske konekcije na *TCP* portu 80, mada se može definisati i neki drugi port. Nakon konektovanja sledi razmena niza zahteva i odgovora na način definisan Protokolom za prenos hiperteksta, *HTTP Hyper-Text Transfer Protocol*. Dobijeni zahtev server predstavlja kao određeni resurs koji je njemu dostupan. To može biti neki fajl na sistemu gde se server nalazi. Sem toga, server može pokretati druge programe na računaru i rezultate njihovog rada prosleđivati kljentu. Ti programi mogu biti izvršni programi, pisani u *C*-u ili *C++*-u, ili pak interpreteri koji u toku svog rada prevode izvorni kod.

Apache web server je jedan od najuspešnijih *OpenSource* projekata, ali činjenica da je besplatan nije ono što odlučuje. Licenca za komercijalne web servere nije velika suma ako se uzme u obzir celokupan prihod koji jedna kompanija može da ima. Međutim, i najbogatije kompanije se odlučuju za *Apache*. Razlozi za kvalitet leže u samoj misiji *Apache* grupe, u kojoj se navodi da im je cilj kreiranje robustnog browsera koji implementira *HTTP* protokol na način koji se očekuje od komercijalnog servera. Dalje, navode da *Apache* mora uvek biti plaforma nad kojom organizacije i individue mogu graditi pouzdane sisteme, kako u ekspermientalne svrhe, tako i za zadatke od krucijalnog značaja. Želja razvojnog tima je da se Apache koristi širom sveta u velikim i malim kompanijama, istraživačkim institucijama, školama i od strane individualaca. Server je besplatan, jer je vođen idejom da stvari koje se tiču same infrastrukture Interneta (web serveri, mail serveri i slično) treba da su dostupne svima.

Apache postoji u verzijama za prakično sve plaforme i operativne sisteme. Navedimo samo neke [17]: *BeOS, FreeBSD, HPUX, IRIX, Linux, MacOSx, NetBSD, NetWare, OpenBSD, OS/2, QNX, Sinix, Solaris i Microsoft Windows…*

5.1.4.2. Upotreba PHP-a za Web skriptove

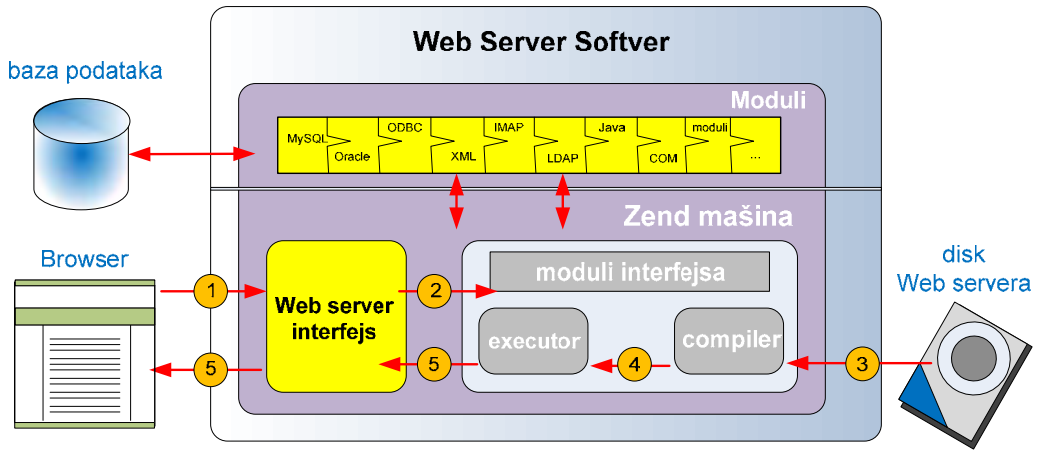
PHP je najpopularniji script jezik u 2011. god. [75]. *PHP* se nametnuo kao komponenta mnogih dinamičkih Web aplikacija srednjeg i velikog obima.

pozicija	programski jezik
1	Java
2	C
3	C++
4	C#
5	PHP
6	(Visual) Basic
7	Objective C
8	Python
9	Perl
10	Lua
11	JavaScript

Tabela 5.1.1 Kotiranje programski jezika u 2011. god.

PHP čita skript, analizira i prevodi u međuformat, a zatim se kod međuformata izvršava pomoću izvršioca *PHP Zend* mašine za izvršavanje skriptova. To znači da se svaki red u skriptu iz izvornog oblika prevodi samo jednom, čak i kad se izvršava stotinama puta. Osim toga, prevođenje omogućava optimizaciju segmenata koda.

PHP je softverski modul web servera. *PHP* softver je podeljen u dve komponente: funkcijske biblioteke (module) i *Zend mašinu* [4].



Slika 5.1.3 Struktura PHP okruženja za izvršavanje skriptova.

Kada korisnički agent uputi zahtev web serveru za *PHP* skriptom, proces se odvija u sledećih šest koraka:

1. Web server prosleđuje zahtev do interfejsa Zend mašine.
2. Interfejs Web servera poziva Zend mašinu i prosleđuje joj parametre.
3. Mašina preuzima PHP skript sa diska.
4. Prevodilac prevodi skript.
5. Izvršilac pokreće prevedeni kôd koji može da sadrži pozive funkcijskih modula. Izlazni podaci izvršioca prosleđuju se interfejsu Web servera.
6. Interfejs Web servera prosleđuje izlazne podatke Web serveru (koji, na kraju, prosleđuje izlazne podatke korisničkom agentu u obliku *HTTP* odgovora).

Glavni "konkurenti" *PHP*-a su *Perl, Microsoftov ASP (Active Server Pages), JSP (Java Server Pages) i ColdFusion firme Allaire.*

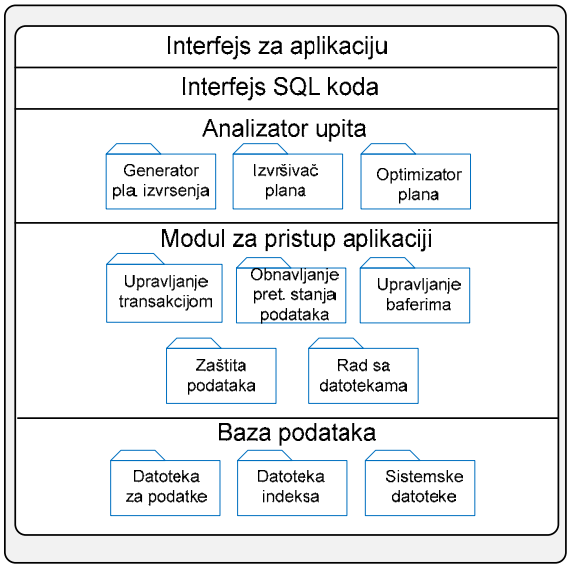
PHP softver je otvorenog koda i potpuno je besplatan. Zbog tog razloga nastojanja programerske zajednice da ga unapredi i održava nisu ugrožena komercijalnim potrebama. Izvorni kod *PHP*-a je transparenatan. Za razliku od komercijalnih proizvoda zatvorenog koda, gde je potrebno čekati da proizvođač objavi zakrpe i servisne pakete, kod *PHP-a* se to može slobodno učiniti. U statičke *HTML* datoteke može se ugraditi jedan ili više *PHP* skriptova, što značajno pojednostavljuje integraciju sa klijentskim slojem. S druge strane, to može dovesti do mešanja skriptova s prezentacijom rezultata. Primenom tehnika sa šablonima može se rešiti većina navedenih problema.

S novim poboljšanjima *Zend* mehanizma za obradu skriptova, izvršavanje je brzo i sve komponente rade unutar glavnog memorijskog prostora *PHP*-a (za razliku od drugih okruženja za izvršavanje skriptova, u kojima su komponente zasebni moduli). Empirijski dokazi svedoče da *PHP* obavlja zadatke srednje složenosti brže od drugih poznatih okruženja za skriptove. Fleksibilan je po pitanju platformi i operativnih sistema. *Apache* radi na različitim platformama i na nekoliko operativnih sistema; *PHP* radi na svemu tome, ali i na mnogim drugim kada se integriše s drugim Web serverima. *PHP* je podesan za razvoj složenih sistema. To je potpun programski jezik sa preko 50 funkcijskih biblioteka. *PHP* karakterišu visoke performance, veoma je efikasan. Jedan relativno jeftin server može da obradi više miliona zahteva dnevno. Rezultati testova koje je objavila firma *Zend Technologies* [72] dokazuju da *PHP* nadmašuje konkurente. Povezivanje s velikim brojem sistema za upravljanje bazama podataka je veoma bitna karakteristika *PHP*-a. *PHP* standardno omogućava uspostavljanje veza s više sistema za upravljanje bazama podataka. Osim s *MySQL*-om, moguće je uspostavljanje direktnih veza sa

sistemima Cassandra, PostgreSQL, mSQL, Oracle, dbm, filePro, Hyperwave, Informix, InterBase i Sybase. Postoji preko 15 biblioteka za osnovni, brz pristup sloju baze podataka. Ugrađene biblioteke služe za obavljanje velikog broja poslova koji su uobičajeni u Web aplikacijama. Budući da je projektovan za upotrebu na Webu, *PHP* sadrži veliki broj funkcija koje omogućavaju obavljanje mnogih korisnih poslova koji su potrebni u Web aplikacijama. Moguće je generisati *GIF* slike tokom rada aplikacije, kakav je slučaj sa iscrtavanjem triangulacija, uspostavljati veze s drugim uslugama u mreži, slati poruke e-poštom, raditi s kolačićima, generisati dokumente u *PDF* formatu itd. Najnoviju verziju, *PHP 5.3.6*, možete preuzeti sa adrese [75], potpuno besplatno. Sintaksa *PHP*-a slična je sintaksama drugih programskih jezika, prvenstveno jezika *C* i *Perl*.

5.1.4.3. RDBMS sistem skladištenja – MySQL

U informacionim tehnologijama, podaci su osnova od koje sve polazi. Da bi se na kvalitetan način mogla vršiti manipulacija sa velikom količinom podataka, potrebno ih je struktuirati i organizovati. Zbog toga su nastale baze podataka. Da bi se standardizovao pristup bazama podataka koristi se uniformni jezik *SQL*. Jedan od glavnih ciljeva rada je razvoj *MatrixDBMS* sistema. Pri implementaciji *MatrixDBMS* sistema smo testirali različite reprezentacije matrica u bazi, različite *MySQL* tipove podataka, kao i brzinu unosa i čitanja podataka kad imamo relacone *SQL* i *NoSQL* baze podataka. *DBMS* je skup komponenata za definisanje, izradu i korišćenje baze podataka. Slika 5.1.4 prikazuje pojednostavljenu strukturu tipičnog *DBMS*-a.



Slika 5.1.4 Struktura sloja baze podataka DBMS-a.

MySQL DBMS čini nekoliko komponenti, koji su opisani u delu koji sledi.

- *Interfejs za aplikacije*. Biblioteke za komunikaciju sa *DBMS*-om. Većina *DBMS*-ova ima jednostavan prevodilac u obliku komandne linije koji često koristi te biblioteke za prenošenje zahteva unetih sa tastature do *DBMS*-a i za prikazivanje odgovora.
- *Interpretator SQL koda.* Sintaksni analizator proverava sintaksu prosleđenih upita i prevodi ih u interni prikaz.
- *Analizator upita.* Generiše razne planove izvršavanja upita na osnovu statistike baze podataka i njenih svojstava, odabira jedan od planova i prevodi ga u akcije koje izvršava na niskom nivou.
- *Pristup podacima.* U module koji upravljaju pristupom podacima uskladištenim na disku spadaju: modul za upravljanje transakcijama, modul za upravljanje postupkom obnavljanja stanja, modul za upravljanje baferom glavne memorije, modul za zaštitu podataka i modul za upravljanje pristupom datotekama.
- *Baza podataka.* To su, zapravo, sami podaci fizički smešteni u datoteke. Podaci pored toga obuhvataju i indeksne datoteke za brz pristup, kao i statističke podatke o bazi podataka i sistemu, koji se prvenstveno koriste za generisanje i optimizovanje planova za izvršavanje upita.

Aplikacija omogućava kreiranje izveštaja u vidu *PDF* dokumenta, iscrtavanje triangulacije poligona na osnovu podataka iz baze itd. *MySQL* podržava osnovni nivo standarda *SQL-92*. Sistem za upravljanje bazama podataka skladišti podatke, pretražuje ih i upravlja njima. *MySQL* je *DBMS* namenjen upravo radu na Internetu i na *MySQL*-u je baziran veliki broj sajtova kao što su: *Yahoo!*, *MP3.com*, *Motorola*, *NASA*, *Silicon Graphics*, *Texas Instruments* i mnogi drugi. Najvažniji takmaci *MySQL*-a su *PostgreSQL*, *Microsoftov SQL Server i Oracle*. *MySQL* je nesporno brz. Na web lokaciji [76] nalazi se i stranica s rezultatima uporednih testova. Mnogi pokazuju da je *MySQL* za više redova veličine brži od konkurentnih programa. *MySQL* se može nabaviti potpuno besplatno, pod uslovima licence za otvoren izvorni kod ili za veoma nisku cenu ukoliko je za upotrebu aplikacije neophodna komercijalna licenca.

5.2. Dizajn aplikacije

Klijent/server aplikacija je implementirana prateći zahteve troslojne web arhitekture, sa univerzalnim *MatrixDBMS* modelom na sloju baze podataka i *MatrixAL* modelom na aplikacionom sloju.

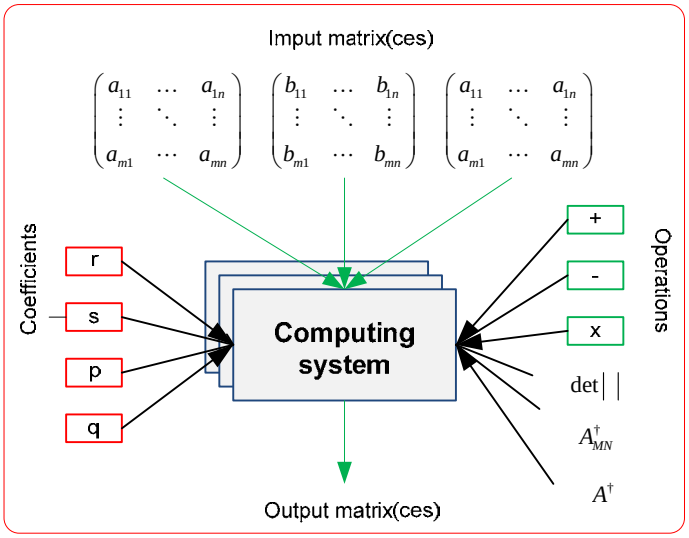
Za web dizajn možemo reći da je to korisnički orijentisana multidisciplinarna delatnost koja uključuje vizualne veštine, izbor tehnologije i sadržaja. Web dizajn predstavlja spoj tehnike (programiranje) i estetike (dizajn). Aplikacija za matrična izračunavanja je dizajnirana korišćenjem paketa *Adobe Design Premium CS3* i, u okviru njega, alata za izradu web strana *Adobe Dreamweaver CS3* i *Adobe Photoshop CS3*. To su alati koji u potpunosti odgovara profesionalnim i tehničkim zahtevnima web aplikacija. Ove aplikacije podržavaju kako hipertekst tako i multimedije (tekst, slika i video materijal).

5.2.1. Matrična izračunavanja

Kad je reč o matričnoj algebri aplikacija omogućava sledeće operacije nad matricama:

- *unarne operacije*: operacije gde se samo jedna matrica i koeficijent po slobodnom izboru koriste kako bi se dobio jedinstveni rezultat $A^{-1}, rA, A^{\dagger}$,
- *binarne operacije*: operacije koje uključuju dve matrice ($A + B, A^p \times B^q$ ...) i
- *tercijarne operacije*: operacije koje uključuju tri matrice, kao što je izračunavanje težinskog *MP* inverza $A^{\dagger}_{MN}$.

Slika 5.2.2. prikazuje kako aplikacija radi, kada su u pitanju matrične operacije.

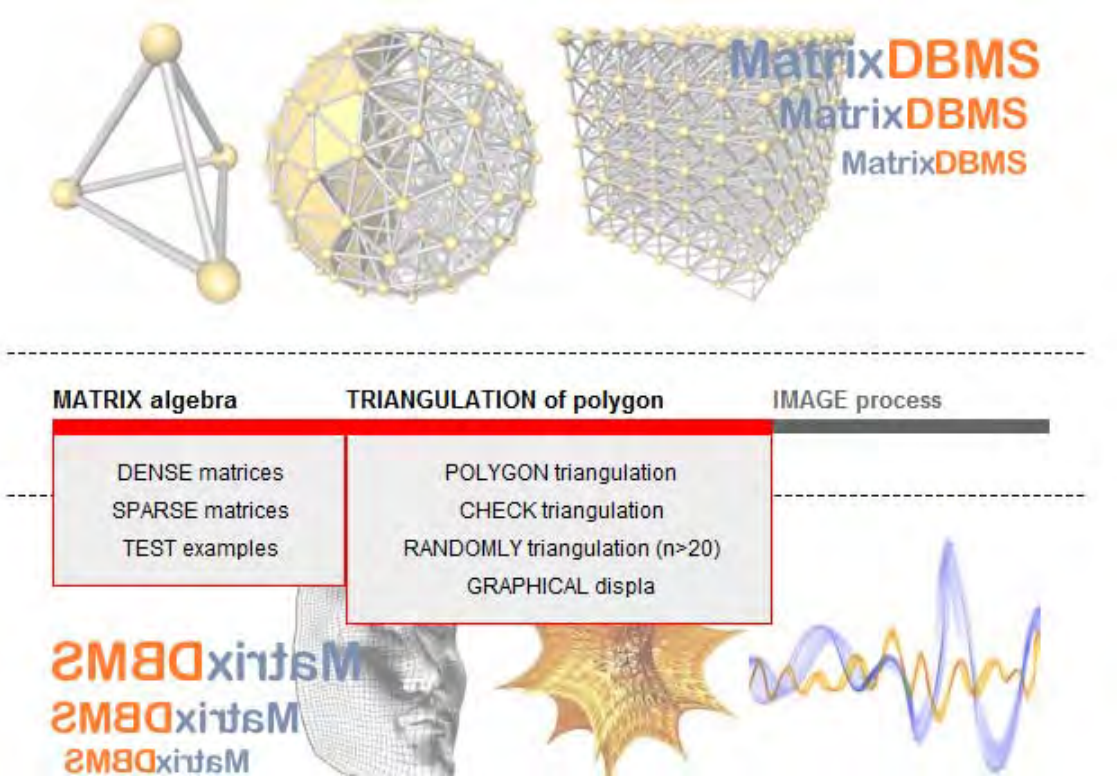


Slika 5.2.1 Sistem izračunavanja matrica.

Korisnik unosi jednu ili više matrica, određene koeficijente, ako su potrebni, i bira operaciju koja će biti sprovedena. Matrica se prvo transformiše u niz sa elementima koji su odvojeni zarezima, a potom se takav string upoređuje se matricama iz baze podataka. Ukoliko su kriterijumi pretrage uspešni, onda ponovno izračunavanje nije potrebno i rezultat će biti prikazan u odgovarajućoj web formi.

Početna strana aplikacije omogućava odabir jedne od tri obrađene oblasti:

- matrična algebra,
- triangulacija konveksih poligona i
- obrada slike.



Slika 5.2.2 Sistem izračunavanja matrica.

U nastavku slede pojedinačni delovi i kompletne web strane koje se odnose na matričnu algebru. U levom delu ekrana nalazi se navigacija, kombinacija eksterne i interne navigacije, u vidu stabla.

Slika 5.2.3 Vertikalna navigacija.

Kod matričnih izračunavanja na početku je potrebno odabrati tip matrice: dense (guste matrice) ili sparse matrice (retko posednute matrice sa malo nenultih). Takođe, korisnicima je omogućeno da pogledaju i test primere za sve operacije koje se sprovode pomoću predstavljene aplikacije.

Posle odabira tipa matrice, u navigaciji sa leve strane, vrši se odabir načina unošenja matrica:

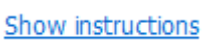
- unos matrice upload-ovanjem txt fajla koji sadrži matricu,
- unos matrice element po element i
- unos matrice u textarea polju.

Slika 5.2.4 Mogućnosti za unos matrice.

U donjem delu ekrana nalazi se vertikalna eksterna navigacija sa linkovima na stranice sa odgovorima na najčešće postavljena pitanja, pravima i obavezama korisnika, kontakt stranicu, stranicu za obradu gustih i retko posednutih matrica, respektivno.

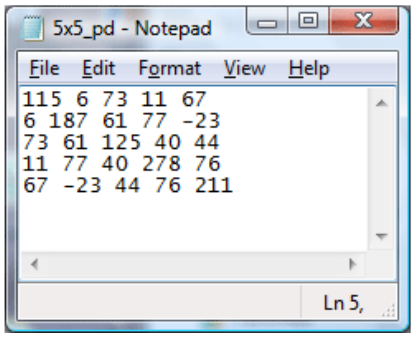
Slika 5.2.5 Horizontalna externa navigacija na dnu strane.

Instrukcije sa načinom korišćenja aplikacije moguće je videti preko sledećeg linka.



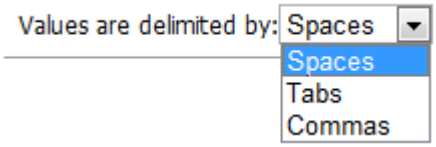
Slika 5.2.6 Pregled instrukcija za funkcionisanje aplikacije.

Na sledećoj strani prikazan je izgled txt fajla sa elementima matrice koju ubacujemo u sistem obrade preko dugmeta *browse...*



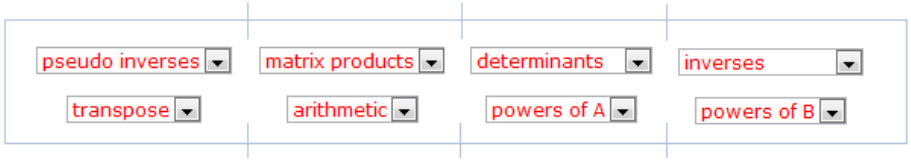
Slika 5.2.7 Izgled txt fajla.

Pri unosu fajla sa matricom, ili elementata matrice u textarea polje, potrebno je odabrati opciju kojom se navodi način na koji su elementi matrice odvojeni. Ponuđene su opcije: prazan prostor, tab taster i zarez.



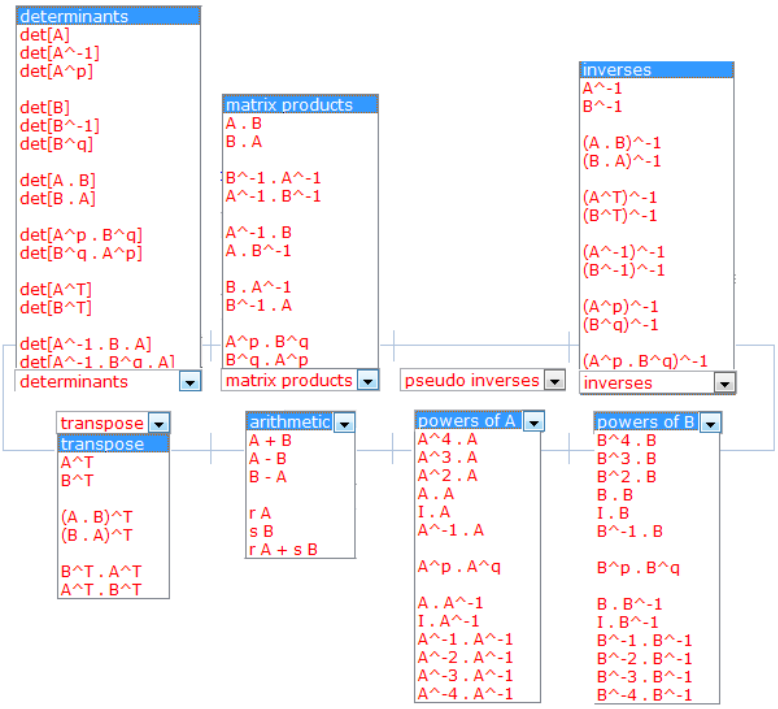
Slika 5.2.8 Pregled instrukcija za funkcionisanje aplikacije.

U donjem delu stranice nalazi se deo sa operacijama koje je moguće sprovesti nad matricama. Tu je moguće odabrati neku od osnovnih matričnih operacija (sabiranje, oduzimanje, množenje, inverz, determinanta, transponovanje itd), izračunavanje generalisanih inverza (MP inverz i težinski inverz), kao i kombinaciju predstavljenih operacija.



Slika 5.2.9 Matrične operacije.

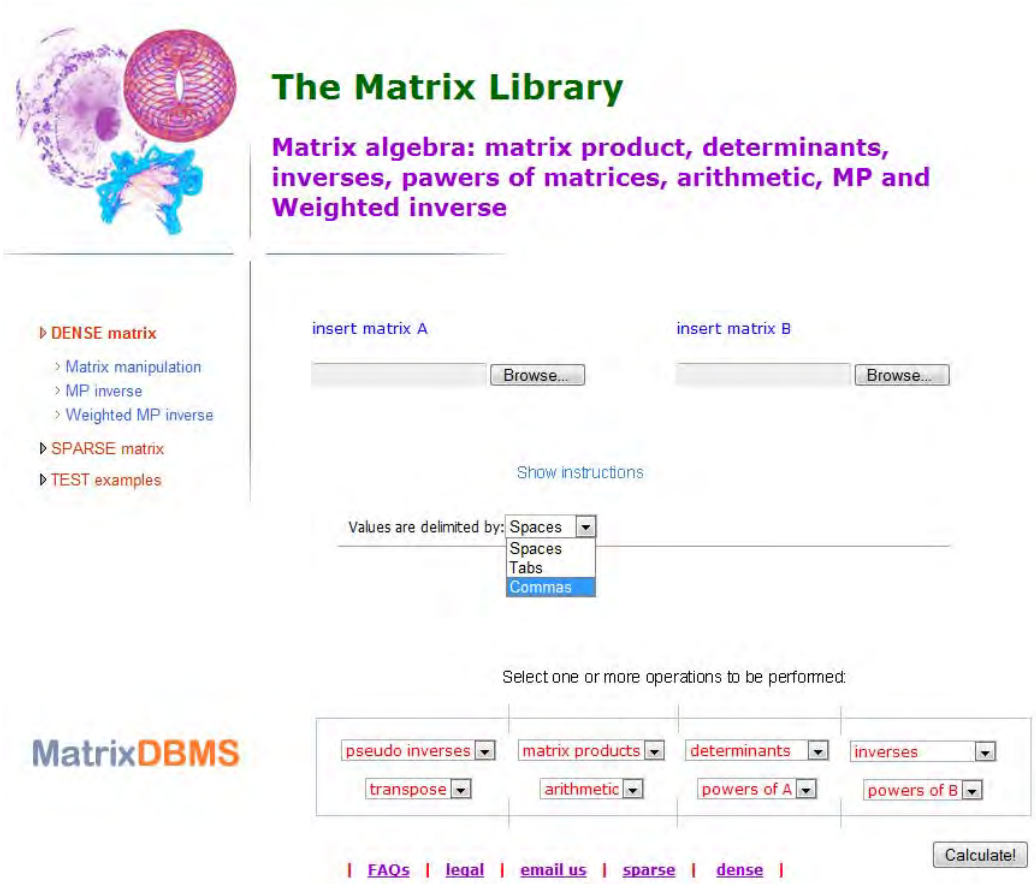
Detaljniji opis mogućih operacija, kao i međusobne kombinacije istih, predstavljene su na sledećoj slici u vidu padajuće Lista/Menu.



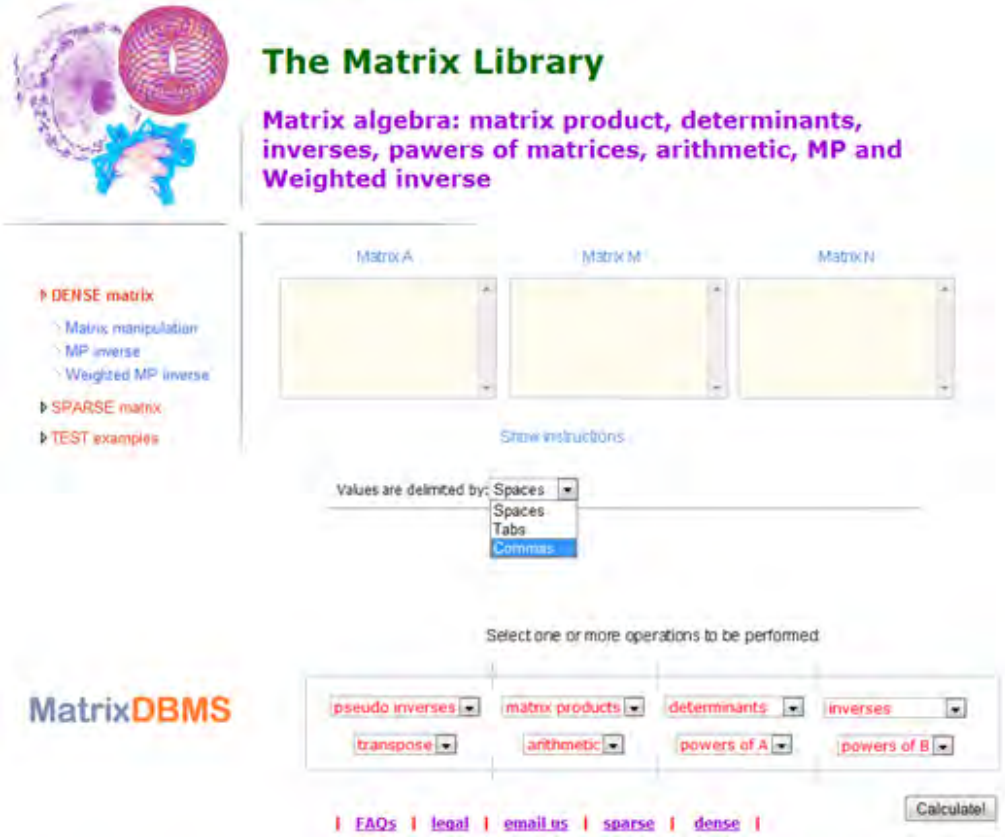
Slika 5.2.10 Moguće matrične operacije.

Na sledećim slikama su predstavljene web stranice za operacije matrične algebre. Potrebno je odabrati način unosa matrice: elemenat po elemenat, txt fajl i textarea polje, a zatim i načina obrade: unarna, binarna i tercijarna operacija.

Slika 5.2.11 Unarna operacija, elementi se unose u tekst polje.



Slika 5.2.12 Binarna operacija, matrica se unosi uploadovanjem txt fajla.



Slika 5.2.13 Tercijarna operacija, matrice se unose preko textarea polja.

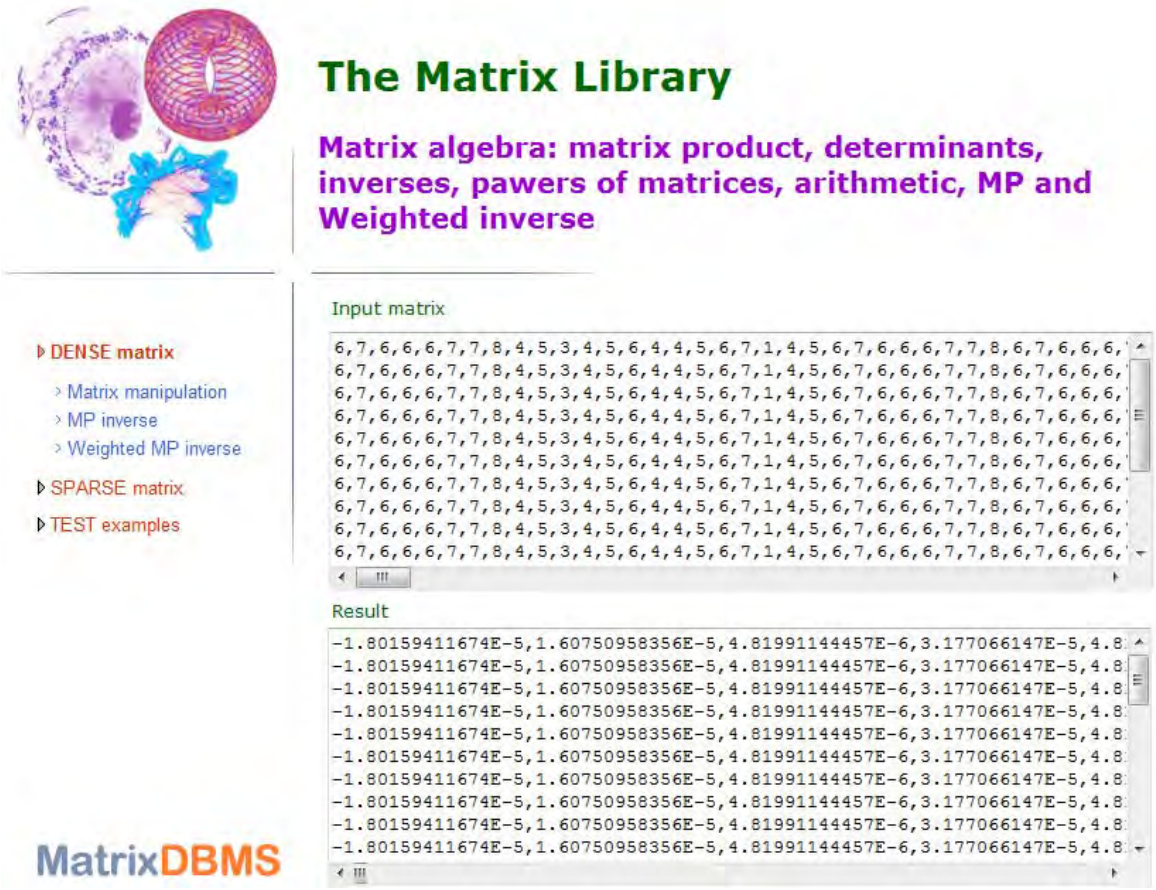
Sledeća slika pokazuje test primer za težinski MP inverz. A, M i N su slučajno odabrane simetrične pozitivno definisane matrice.

Slika 5.2.14 Test primer izračunavanja težinskog MP inverza.

Korisnicima je omogućen prikaz svih unetih marica i rešenja dobijenih nakon sprovođenja operacije. Na sledećoj slici je prikazan izgled strane kad je u pitanju unarna operacija.

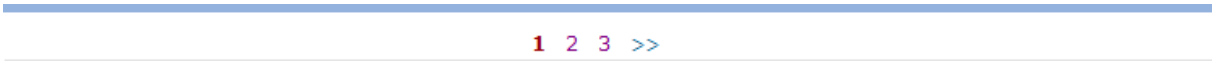
Slika 5.2.15 Prikaz svih uskladištenih matrica i rezultata obrade.

Pošto je u pitanju unarna operacija, klikom na link *view* moguće je pregledati izgled ulazne matrice i rešenje dobijeno sprovedenom operacijom.



Slika 5.2.16 Pregled unarne operacije, ulazna matrica i rešenje.

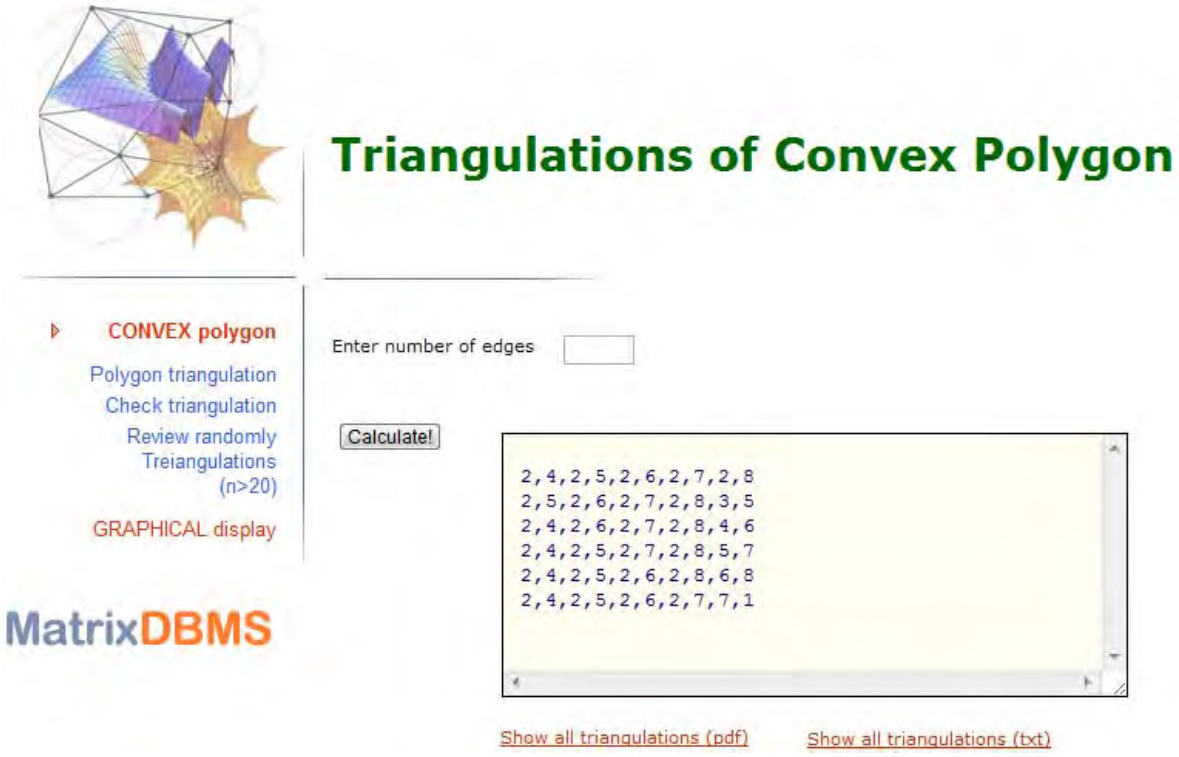
U gornjem i donjem delu stranice nalazi se navigacija kojom je omogućen prikaz po 20 stavki po strani, kako bi rasteretili stranicu i omogućili veću preglednost.



Slika 5.2.17 Navigacija za prikaz ograničenog broja triangulacija.

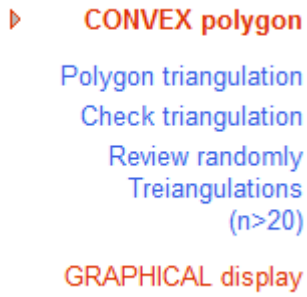
5.2.2. Triangulacija konveksnih poligona

Na sledećim slikama su prikazane stranice za triangulaciju konveksnih poligona. Sve postojeće triangulacije je moguće dobiti unosom broja ivica poligona za koji korisnik želi da dobije prikaz svih triangulacija.



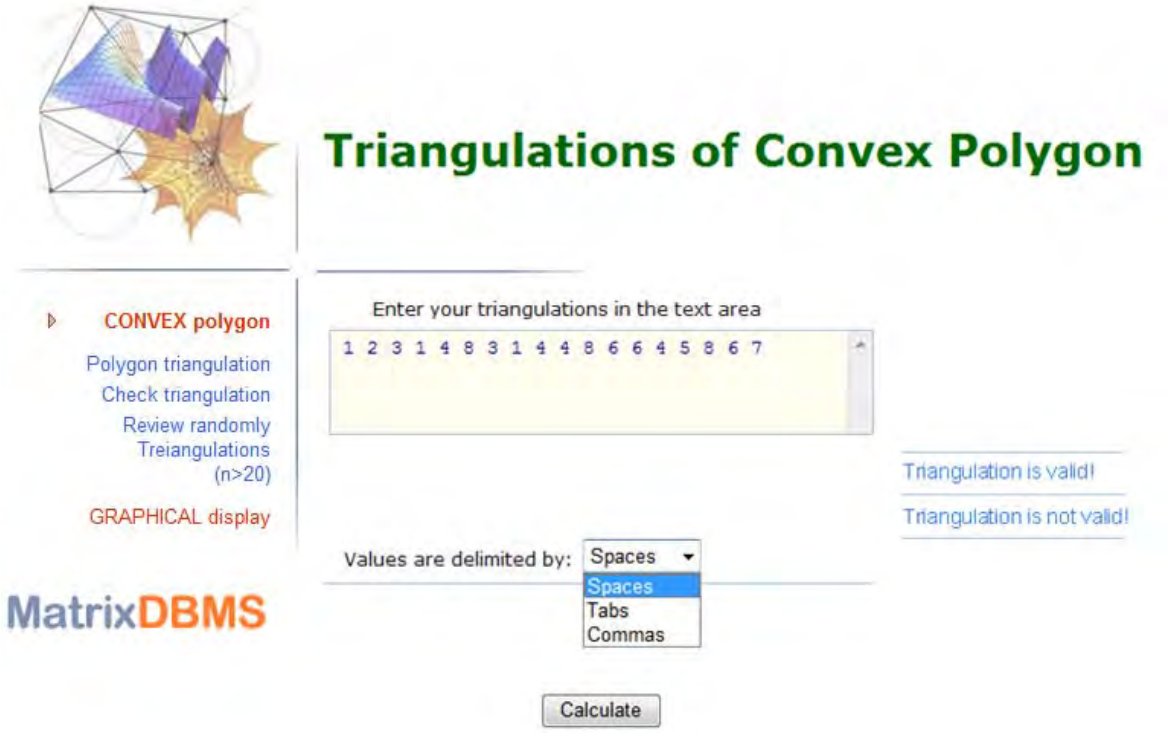
Slika 5.2.18 Izračunavanje triangulacija unosom broja ivica.

Navigacija se nalazi sa leve strane, a sve u cilju nastavka dobro poznate forme web stranica. Sastoji se od linkova kojima je moguće doći do stranica za triangulaciju poligona, proveru ispravnosti unete triangulacije, grafički prikaz kao i mogućnost prikaza i preuzimanja sa servera triangulacija za vrednost $n > 20$.



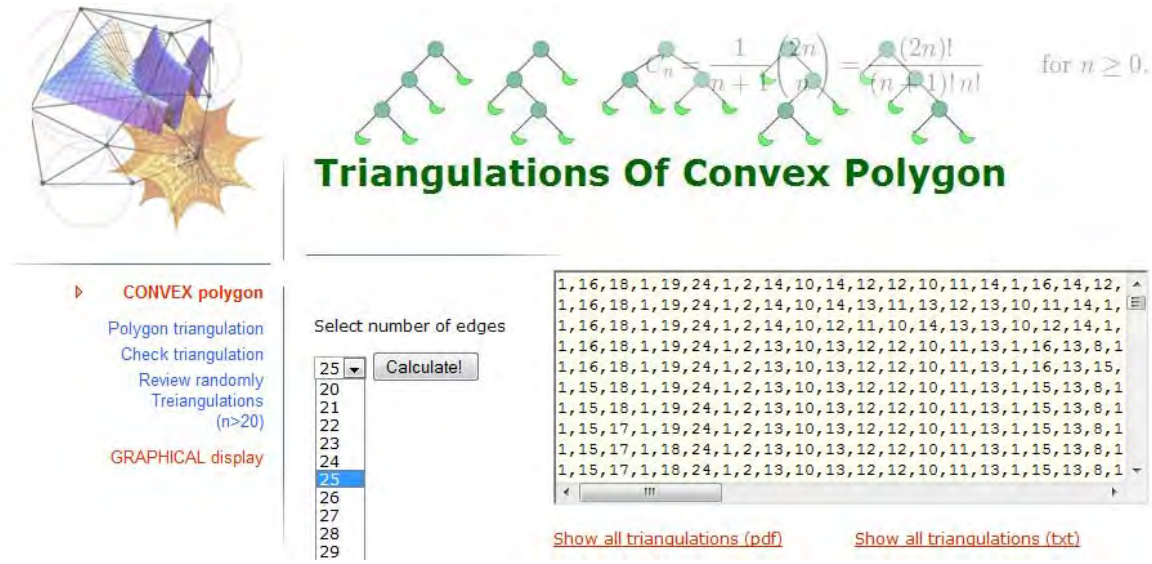
Slika 5.2.19 Vertikalna navigacija.

Aplikacija omogućava validaciju unete triangulacije. Neophodno je uneti određenu triangulaciju, odabrati način na koji su elementi triangulacije odvojeni, i klikom na dugme *Calculate* dobiti obaveštenje o ispravnosti unete triangulacije.



Slika 5.2.20 Validacija unete triangulacije.

Korisnicima je omogućen pregled slučajno odabranih triangulacija za $n > 20$, tako što se iz padajuće liste odabere broj temena poligona.



Slika 5.2.21 Prikaz slučajno odabranih triangulacija za **n** > 20.

Jedna od mogućnosti za preuzimanje dokumeta je *PDF* format.

Show all triangulations (pdf) Show all triangulations (txt)

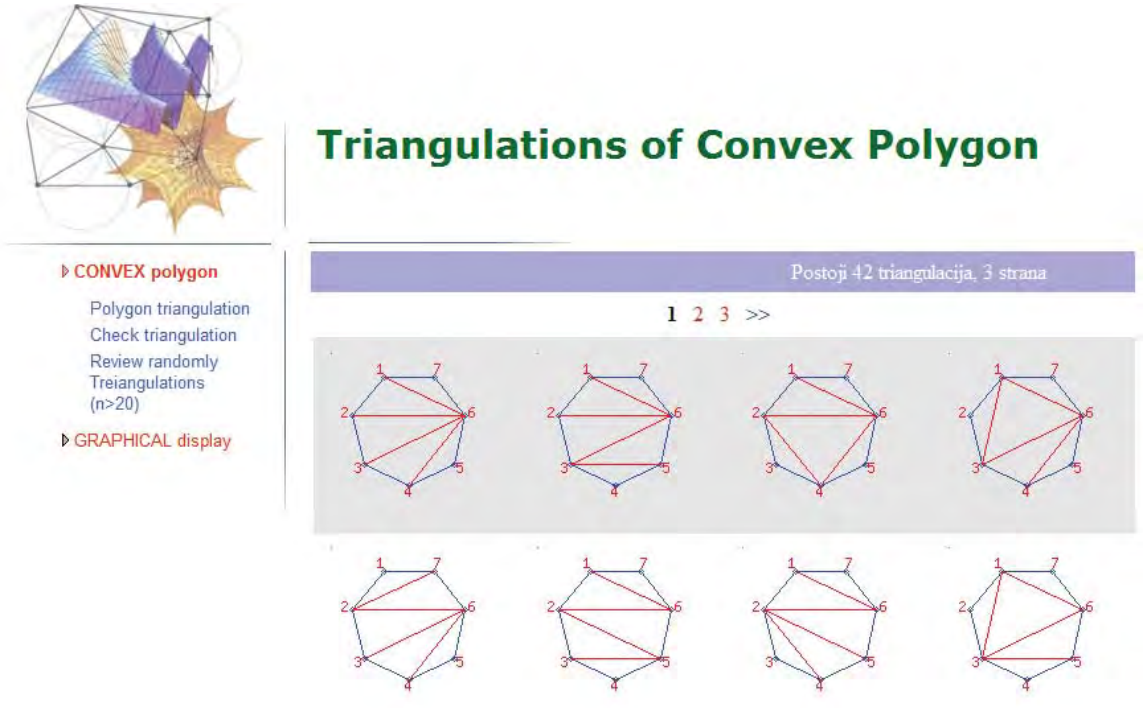
Slika 5.2.22 Odabir prikaza.

Na sledećoj slici je prikazan izgled pdf dokumenta koji sadrži podatke iz tabele baze podataka, a odnose se na triangulaciju šestougaonog poligona.

TRIANGULATION
1,3,1,4,1,5
1,3,1,4,4,6
1,3,1,5,3,5
1,3,3,5,3,6
1,3,3,6,4,6
1,4,2,4,1,5
1,4,2,4,4,6
1,5,2,4,2,5
2,4,2,5,2,6
2,4,2,6,4,6
1,5,2,5,3,5
2,5,3,5,2,6
2,6,3,5,3,6
2,6,3,6,4,6

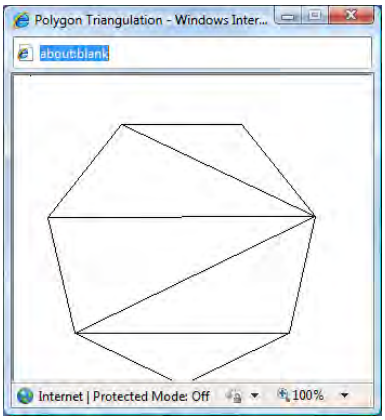
Slika 5.2.23 Prikaz rešenja u pdf formatu.

Pored *PDF* fajla, rešenje dobijeno sprovođenjem operacija nad matricama je moguće download-ovati u vidu *txt* ili *XML* fajla. Grafički prikaz triangulacija za odabrano *n* (*Funkcija B.4.3 - Funkcija B.4.5*) je pokazan na sledećoj slici.



Slika 5.2.24 Grafički prikaz triangulacija za odabrano n.

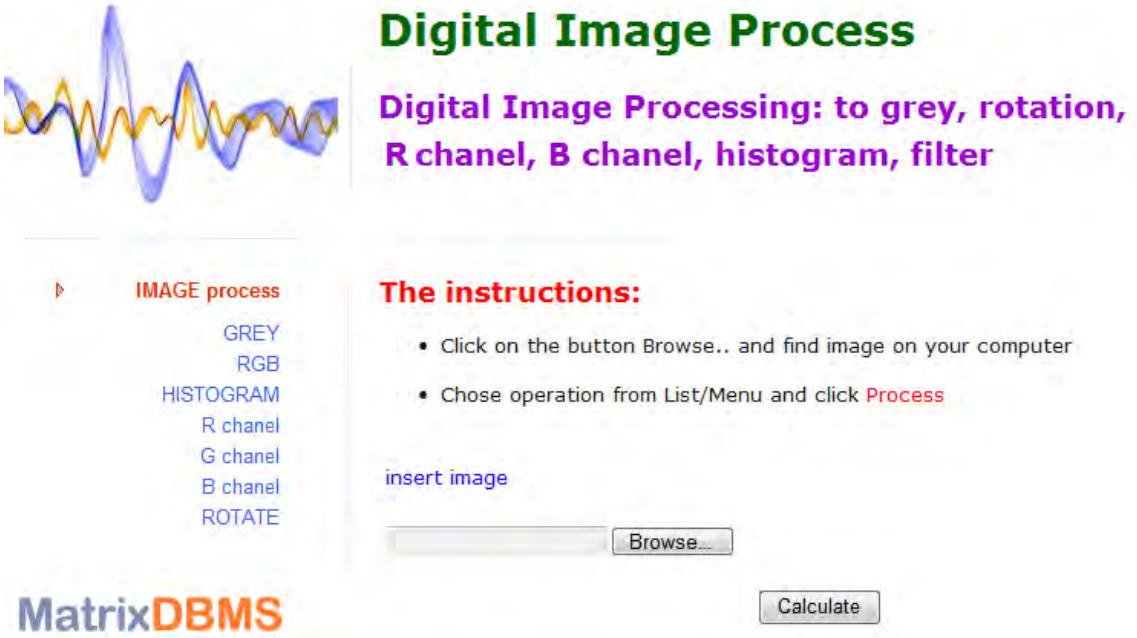
Koristeći usluge JavaScrip-a kreirali smo funkciju za uvećan prikaz odabrane triangulacije kao na slici.



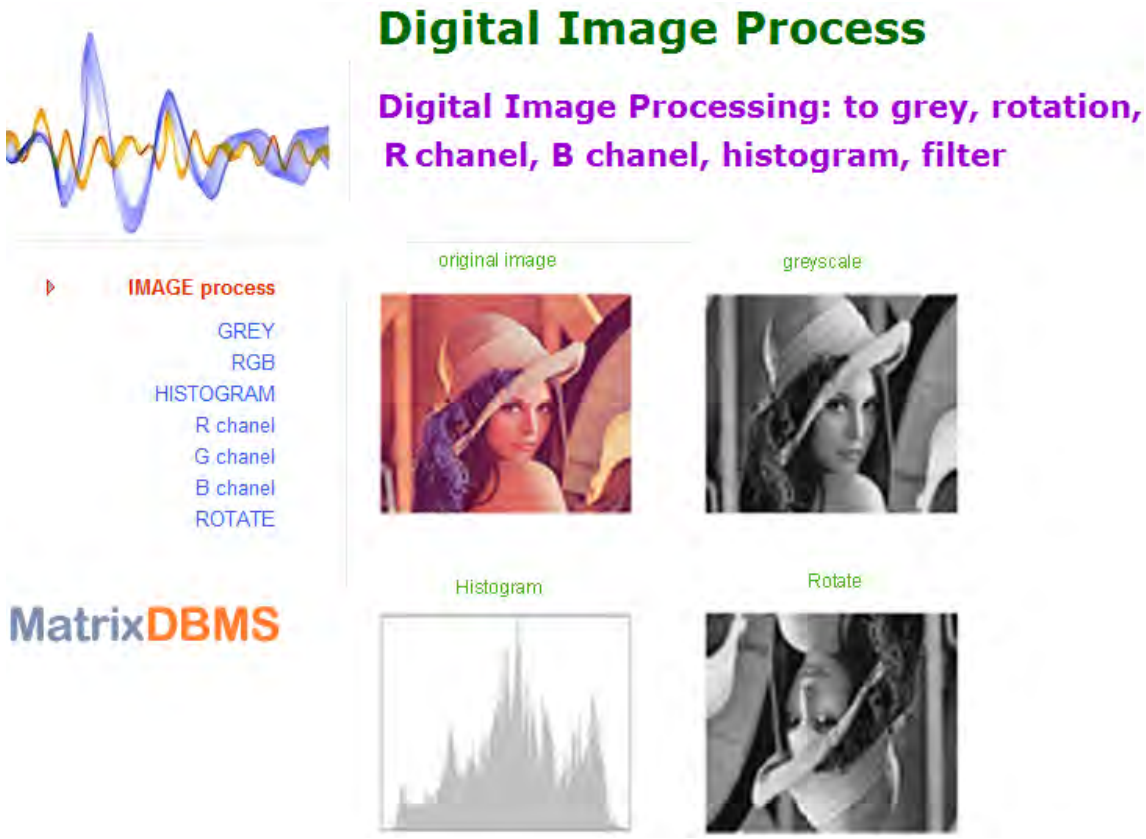
Slika 5.2.25 Uvećani prikaz triangulacije.

5.2.3. Obrada slike

Aplikacija omogućava odabir neke od metoda za obradu slike kao što su: histogram, rotacija, prebacivanje u grey model itd.

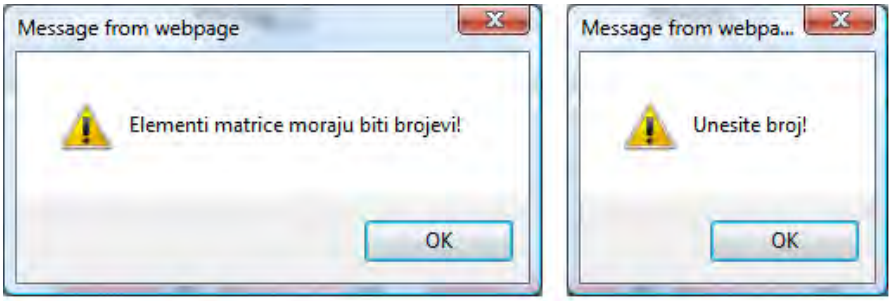


Slika 5.2.26 Početna strana za obradu slika.



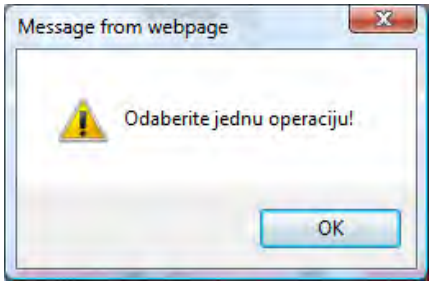
Slika 5.2.27 Neke od mogućnosti za obradu slike.

Klijent/server aplikacija onemogućava korisnicima unos karaktera koji ne zadovoljavaju uslove. Takođe, ova ograničenja su uvedena i sa namerom olakšanja upotrebe korisnicima koji nisu upućeni u rad ili su prevideli neki korak pri sprovođenju date operacije. U tu svrhu smo korišćenjem *JavaScript* jezika, na strani klijenta, onemogućili korisnicima da unose slova umesto brojeva koji predstavljaju elemente matrice ili koeficijente.



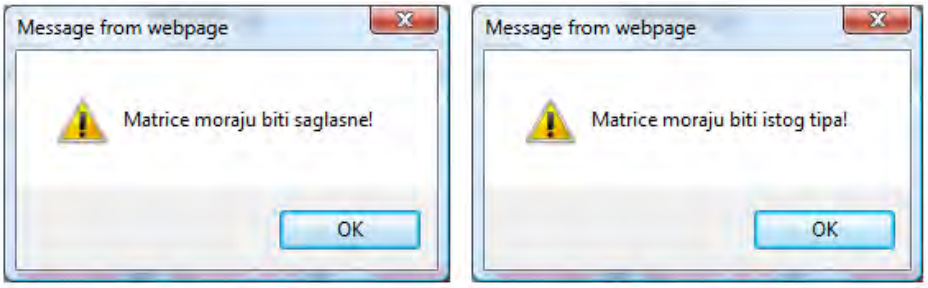
Slika 5.2.28 Zaštita kojom je onemogućen pogrešan unos elementa.

Da bi sproveo operaciju nad matricom korisnik mora da odabere jednu ili više njih.



Slika 5.2.29 Zaštita kojom se zahteva od korisnika da unese operaciju.

Potrebno je i da budu zadovoljena pravila pri sprovođenju operacije nad matricama kao npr. da matrice moraju biti istog tipa pri sabiranju i oduzimanju, odnosno saglasne pri množenju.



Slika 5.2.30 Zaštita kojom se ispunjava uslov za sprovođenje određenih matričnih operacija.

Glava 6

6. Zaključak

U disertaciji je analizirano *CPU* vreme izvršavanja matričnih izračunavanja (matrična algebra) uvođenjem klijent/server web aplikacije bazirane na *MatrixDBMS* sistemu skladištenja. Efekat analize matričnih izračunavanja je potvrđen i primenom pomenutog matričnog sistema na triangulaciju konveksnih poligona i obradu slike. Aplikacija je implementirana u *PHP/MySQL* okruženju, otvorenog je koda i dostupna sa svakog mesta u svako vreme.

Web aplikacije omogućavaju izvesne prednosti pri sprovođenju operacija nad matricama u odnosu na robusne desktop aplikacije (Tabela 2.5.11 i Tabela 2.5.12). Prilikom sprovođenja matričnih izračunavanja brži odziv sistema se dobija kad su matrice na aplikacionom sloju predstavljene korišćenjem moćnog sistema klasa i objekata (*MatrixAL model*) u odnosu na sistem nizova. Rezultati testiranja su predstavljeni u Tabeli 2.5.7. Na osnovu rezultata testiranja potvrdili smo da najbolje performanse i fleksibilnost sistema dobijamo kada su matrice predstavljanje sistemom objekata (aplikacioni sloj) i uskladištene u *R* formatu (sloj baze podataka). Eksperimentalni rezultati izneti u Tabeli 2.5.8 pokazuju da matrične operacije, koje se sprovode samo na sloju baze podataka, koristeći *MySQL* metode, daju lošije *CPU* vreme nego u slučaju *UDF (user definited functions)* na aplikacionom sloju troslojne web arhitekture. Kad je reč o tipovima podataka, u *MySQL* bazi podataka, blob tip daje najbolje CPU vreme pri unosu i pretrazi podataka. To potvrđuju rezultati u Tabelama 2.5.5 i 2.5.6. Postoje velike razlike u mogućnosti kreiranje jedinstvenog, dinamičnog i fleksibilnog *MatrixDBMS* modela kada su u pitanju relacione baze podataka (*MySQL*) i *NoSQL* sistemi skladištenja kakva je *Cassandra*. Testiranje pokazuje pozitivne strane *NoSQL* sistema skladištenja, a u prilog tome idu rezultati prikazani u Tabelama 4.1.1 i 4.1.2.

U radu smo predložili i algoritam za triangulaciju n-tougaonog poligona, na osnovu izračunatih i uskladištenih triangulacija za $(n-1)$-ugaoni poligon. Sistem se oslanja na

MatrixDBMS model. Cena skladištenja mora biti plaćena, ali će korišćenje gotovih, već izračunatih vrednosti, skratiti ukupno vreme obrade. Napominjemo i to da će za velike vrednoti *n*, zbog velikog broja triangulacija, vreme obrade biti izuzetno veliko i meriće se danima. Međutim, vreme ponovnog pregleda, već izračunatih triangulacija, je veoma kratko. Baza podataka ima značajnu ulogu pri triangulaciji poligona, jer čisti duplikate, zaobilazi kumulativan proces izračunavanja i na taj način skraćuje vreme odziva sistema. Dobili smo pozitivne rezultate do određenog broja temena i u odnosu na Hurtadov algoritam bez duplikata (Tabela 3.1.3). Rezultati, do kojih smo došli tokom istraživanja, pokazuju da duplikati, koji se javljaju pri triangulaciji konveksnih poligona, mogu biti uklonjeni na više načina ali je zbog memorijskih resursa jedino održivano rešenje korišćenje sistema baze podataka i jedinstvenog ključa. Fajl sistem brže komunicira sa aplikacionom logikom, ali ne omogućava uklanjanje duplikata, pa u pomoć pristiže *DBMS* koji nadoknađuje taj nedostatak. Iskoristili smo prednosti jednog i drugog i implementirali takav sistem. Primena *MatrixDBMS* modela na obradu slike pokazuje da, za dobijanje optimalnih rezultata, treba kombinovati više sistema. Kada su u pitanju memorijski resursi, treba koristiti transformaciju slike pomoću `base_64()` modela. Međutim, za primenu metoda obrade potrebno je koristiti *RGB* model na kojem se bazira grafika računara. Rezultati predstavljeni u Tabelama 3.2.3-3.2.5, ukazuju na prednosti u korist `base_64()`načina transformacije kad su u pitanju pretraga sistema skladištenja i memorijski resursi.

Naučni značaj predstavljenog rada ogleda se u pristupu navedenim problemima koji do sada nisu proučavani na ovakav način, dok se praktični značaj ogleda u mogućnostima primene rezultata istraživanja u vidu *MatrixDBMS* sistema skladištenja na unapređenje rešavanja problema koji se bave sličnom tematikom. Korelacija *MatrixDBMS* i *MatrixAL* modela omogućava smanjenje CPU vremena izvršavanja operacija nad matricama i odziva sistema skladištenja. U radu smo obradili nove pristupe matričnim izračunavanjima (univerzalni *DBMS* model): *Cassandra s*istema skladištenja (*Amazon, Facebook, Twiter, Google itd*) i paralelni pristup matričnim izračunavanjima kroz primer triangulacije konveksnih poligona.

U cilju daljeg proučavanja ove problematike, neophodno je identifikovati druge širokoprimenjive operacije matematične algebre koje treba implementirati i tako povećati funkcionalnost predloženog *MatrixDBMS*. Potrebno je poraditi i na razviju mehanizama za smanjenje upravljanja memorijom i povećanje različitih tipova matrica. Minimalna brzina odziva sistema skladištenja u vidu *MatrixLIBRARY* modela, koji će se zasnivati na principu mobilnog programiranja i uključivati kompresiju podataka, će biti budući pravac razvoja predstavljenog klijent/server modela.

Dodatak

A. Matrična izračunavanja

A.1. Osnovne matrične operacije

U ovom delu ćemo opisati metode klase *Matrix()* a koje se odnose na osnovne matrične operacije: sabiranje, oduzimanje, množenje, izračunavanje determinante, inverz itd.

Funkcija A.1.1 Funkcija *matrixSubstraction()* služi za oduzimanje dve matrice koje su istih dimenzija. Argument joj je instanca klase, odnosno objekat koji nosi elemente matrice i dimenzije.

```
function matrixSubstraction($some_matrix){
  for($i=0; $i<$this->get_num_rows(); $i++){
    for($j=0; $j< $this->get_num_columns(); $j++){
      $substraction[$i][$j]= $this→numbers[$i][$j]-
      $some_matrix->numbers[$i][$j];}}
  $the_substraction = new matrix($Substraction);
  return $the_substraction ;}
```

Funkcija A.1.2 Funkcija *matrixAddition()* služi za sabiranje dve regularne matrice.

```
function matrixAddition($some_matrix){// PROMENI
  for($i=0; $i<$this->get_num_rows(); $i++) {
    for($j=0; $j< $this->get_num_columns(); $j++){
      $addition[$i][$j]= $this→numbers[$i][$j]+$some_matrix→numbers[$i][$j];}}
  $the_addition = new matrix($addition);
  return $the_addition;}
```

Funkcija A.1.3 Funkcija *matrixMultiplication()* služi za množenje dve matrice koje su saglasne, odnosno koje zadovoljavaju uslov da je broj kolona prve jednak broju redova druge.

```
function matrixMultiplication($some_matrix)
  $easier = $some_matrix->transpose();
  foreach($this->numbers as $i => $row) {
    foreach($easier->numbers as $j => $column){
      $total = 0;
      foreach($row as $k => $number){
        $total += $number * $column[$k];}
      $arrayMultiplication[$i][$j] = $total;}}
  $the_arrayMultiplication = new matrix($arrayMultiplication);
  return $the_arrayMultiplication;}
```

Funkcija A.1.4 Funkcija *scalarMultiplication()* služi za množenje matrice skalarom.

```
function scalarMultiplication($value){
  for($i = 0; $i < $this->get_num_rows(); $i++){
    for($j = 0; $j < $this->get_num_columns(); $j++){
      $arrayScalar[$i][$j] = $this->numbers[$i][$j] * $value;}}
  $the_arrayScalar = new matrix($arrayScalar);
  return $the_arrayScalar;}
```

Funkcija A.1.5 Funkcija *determinant()* služi za izračunavanje determinante kvadratne matrice.

```
function determinant($some_matrix){
  $det=0;
  $matriz=$this→zeros($someMatrix→get_num_rows(),
  $some_matrix->get_num_columns());
  if($someMatrix→get_num_rows()==2&&$ssome_matrix->get_num_columns()==2){
    $det = $some_matrix->numbers[0][0] * $some_matrix->numbers[1][1] –
    $some_matrix->numbers[0][1] * $some_matrix->numbers[1][0];}
  else{
    for($j=0;$j<$some_matrix->get_num_columns();$j++){
      $matrix = $this->matrixSubstraction($some_matrix-,0,$j);
      if(fmod($j,2)==0){
        $det += $some_matrix->numbers[0][$j]*$this->determinant($matrix);}
      else{
        $det -= $some_matrix-->numbers[0][$j]*$this->determinant($matriz);}}}
  return $det;}
```

Funkcija A.1.6 Funkcija *matrixInverse()* služi za izračunavanje inverza kvadratne matrice.

```
function matrixInverse($some_matrix){
  $matrica = array();
  for($i=0;$i<$some_matrix-->get_num_rows();$i++){
    for($j=0;$j<$some_matrix-->get_num_columns();$j++){
```

```
      if (fmod($i + $j, 2) == 0){
        $matrica[$i][$j] = $this→determinant
        (this->matrixSubstraction($some_matrix-,$i,$j));}
      else{
        $matrica[$i][$j] = -$this→determinant
        ($this->SubMatriz($some_matrix-,$i,$j));}}}
   $the_arrayInverse = new matrix($matrica);
   $r=$this->matrixDivision($the_arrayInverse,$this->determinant($someMatrix));
   return $r->matrixTranspose($r);}
```

Funkcija A.1.7 Funkcija `matrixTranspose()` služi za transponovanje matrice.

```
function matrixTranspose(){
   $ArrayT = array();
   for($i=0;$i<$this->get_num_rows();$i++){
      for($j=0;$j<$this->get_num_columns();$j++) {
        $arrayTranspose[$j][$i]=$this->numbers[$i][$j];}}
   $the_arrayTranspose = new matrix($arrayTranspose);
    return $the_tarrayTranspose}
```

A.2. Implementacija rekurzivnih formula generalisanih inverza

Funkcija A.2.1 Funkcija *makeZero()* sa argumentom broj redova matrice kreira nultu matricu.

```
function makeZero($m){
   for($i=0;$i<$m;$i++){
     $zero[0][$i]=0;}
   $the_zeroMatrix = new matrix($zero);
   return $the_zeroMatrix;}
```

Funkcija A.2.2. Funkcija *ithCol()* služi za izdvajanje i – te kolone matrice.

```
function ithCol($column){
   $arrayCol=array();
   for($i=0; $i<$this->get_num_rows(); $i++){
     for($j=0; $j<$this->get_num_columns();$j++){
       if($j+1==$column)
         $arrayCol[$i][0]= $this->numbers[$i][$j];}}
   $the_arrayCol = new matrix($arrayCol);
   return $the_arrayCol;}
```

Funkcija A.2.3 Funkcija *frstiCol()* služi za izdvajanje prvih i kolona matrice.

```
function frstiCol($columns){
  $arrayFrst=array();
  for($i=0; $i<$this->get_num_rows();$i++){
    for($j=0; $j<$this->get_num_columns();$j++){
      if($j+1<=$columns)
       $ArrayFcol[$i][$j]=$this->numbers[$i][$j];}}
  $the_arrayFrst = new matrix($arrayFrst);
  return $the_arrayFrst;}
```

Funkcija A.2.4 Funkcija *matNum()* vraća element matrice dimenzija 1×1.

```
function matNum(){
  if($this->get_num_rows()==1 & $this->get_num_columns()==1)
    $matNum = $this->numbers[0][0];
  return $matNum;}
```

Funkcija A.2.5 Funkcija *isZero()* vraća True ako su svi elementi matrice 0, odnosno False ako to nije slučaj.

```
function isZero(){
  $numZero=0;
  for($i=0; $i<$this->get_num_rows(); $i++) {
    for($j=0; $j<$this->get_num_columns(); $j++){
      if(round($this->numbers[$i][$j],3)!=0)
        $numZero++;}}
  return $numZero;}
```

Funkcija A.2.6 Funkcija *mScalar()* vraća broj koji se nalazi na određenoj poziciji na osnovu koraka 13, *Algoritma 2.1.2*.

```
function mScalar($k)
  $scalar=$this->numbers[$k-1][$k-1];
  return $scalar;}
```

Funkcija A.2.7 Funkcija *mMinus()* predstavlja implementaciju N_{k-1} iz formule 2.6, odnosno $M_{\_}$ iz formule 2.14.

```
function mMinus($k){
  $arrayMatrix=array();
  for($i=0;$i<$k-1;$i++){
    for($j=0;$j<$k-1;$j++){
      $arrayMatrix[$i][$j]=$this->numbers[$i][$j];}}
      $the_arrayMatrix= new matrix($arrayMatrix);
  return $the_arrayMatrix;}
```

Funkcija A.2.8 Funkcija *mColumn()* predstavlja implementaciju l_k iz formule 2.6, odnosno $\tilde{m}$ iz formule 2.14.

```
function mColumn($k){
    $arrayColumn=array();
    for($i=0;$i<$k-1;$i++){
       $arrayColumn[$i][0]=$this->numbers[$i][$k-1];}
       $the_arrayColumn = new matrix($arrayColumn);
    return $the_arrayColumn;}
```

Funkcija A.2.9 Funkcija *mMatrix()* predstavlja implementaciju N_k iz formule 2.6, odnosno M iz formule 2.14.

```
function mMatrx($k){
   $arrayMatrix=array();
   for($i=0;$i<=$k;$i++){
     for($j=0;$j<=$k;$j++){
       $arrayMatrix[$i][$j]=$this->numbers[$i][$j];}}
  $the_arrayMatrix = new matrix($arrayMatrix);
  return $the_arrayMatrix;}
```

Funkcija A.2.10 Funkcija *appRow()* matrici X dodaje vrsta matricu Y, kao $m + 1$ vrstu.

```
function appRow($someMatrix){
   for($i=0; $i<$someMatrix->get_num_columns(); $i++){
     $this->numbers[$this->get_num_rows()][$i]=$someMatrix2->numbers[0][$i];}
  $the_appRow = new matrix($this->numbers);
  return $the_appRow;}
```

Funkcija A.2.11 Funkcija *finaly()* kreira završnu matricu spajanjem pojedinih delova, pružajući usluge funkciji *Inverse()*.

```
function finaly($someMatrix){
   for($i=0;$i<$this->get_num_rows();$i++){
     $this->numbers[$i][$this->get_num_columns()]=$someMatrix->numbers[$i][0];}
   $the_finaly = new matrix($this->numbers);
   return $the_finaly;}
```

Funkcija A.2.12 Implementacija *Algoritma 2.1.3* je data funkcijom `Inverse()` .

```
function Inverse(){
   $nii=$this->mScalar(1);
   $N=1/$nii; $N=array(array($N));
   $NO=new matrix($N);
   for($i=2;$i<=$this->get_num_columns();$i++){
      $s=$this->mScalar($i); $nii=array(array($s));
      $niii=new matrix($nii); $li=$this->mColumn($i);
      $lit=$li->matrixTranspose();
      $giip=$lit->matrixMultiplication($NO);
```

```
    $gii1=$giip->matrixMultiplication($li);
    $gii=$niii->matrixSubstraction($gii1);
    $gii2=$gii->matNum();
    $gii=1/$gii2; $fip=$NO->matrixMultiplication($li);
    $fi1=$fip->scalarMultiplication($gii);
    $fi=$fi1->scalarMultiplication(-1);
    $fit=$fi->matrixTranspose();
    $E1=$fi->matrixMultiplication($fit);
    $E2=$E1->scalarMultiplication(1/$gii);
    $E=$NO->matrixAddition($E2);
    $ArrayData1=$E->appRow($fit);
    $giin=array(array($gii));
    $giin=new matrix($giin);
    $ArrayData2=$fi->appRow($giin);
    $NO=$ArrayData1->finaly($ArrayData2);}
  return $NO;}
```

Funkcija A.2.13 Implementacija Grevilovog algoritma, *Algoritam 2.1.1,* je data sledećom funkcijom `Grevile()`.

```
function Grevile(){
  $AA=$this->ithCol(1);
  if($AA->isZero()==0)
    $AR=$AA->matrixTranspose();
  else{
    $TA=$AA->matrixTranspose();
    $AR=$TA->matrixMultiplication($AA);
    $pom=$AR->matNum();
    $AR=$TA->scalarMultiplication(1/$pom);}
  for($i=2;$i<=$this->get_num_columns();$i++){
    $OO=$this->ithCol($i);
    $DD=$AR->matrixMultiplication($OO);
    $FC=$this->frstiCol($i-1);
    $MM=$FC->matrixMultiplication($DD);
    $CC=$OO->matrixSubstraction($MM);
    if($CC->isZero()==0){
       $TDD=$DD->matrixTranspose();
       $P=$TDD->matrixMultiplication($DD);$pom=1+$P->matNum();
       $TM=$AR->matrixTranspose(); $MM=$TM->matrixMultiplication($DD);
       $BB=$MM->scalarMultiplication(1/$pom);}
    else{
        $TTM=$CC->matrixTranspose();
        $P=$TTM->matrixMultiplication($CC);
        $pom=$P->matNum();
        $BB=$CC->scalarMultiplication(1/$pom);}
    $EE=$BB->matrixTranspose(); $QW=$DD->matrixMultiplication($EE);
    $AR=$AR->matrixSubstraction($QW);
    $AR=$AR->appRow($EE);}
  return $AR;}
```

Funkcija A.2.14 Implementacija težinskog MP inverza, *Algoritam 2.1.2.,* je data sledećom funkcijom `WeightedInverse`().

```
function WeightedInverse($matrix_m,$matrix_n){
  $aa=$this->ithCol(1);
  if($aa->isZero()==0)
      $ar=$aa->matrixTranspose();
  else{
      $ta=$aa->matrixTranspose();
      $alb=$ta->matrixMultiplication($matrix_m);
      $ali=$alb->matrixMultiplication($aa);
      $inv=$ali->Inverse();
      $ar=$inv->matrixMultiplication($alb);}
  for($i=2;$i<=$this->get_num_columns();$i++){
      $ii=$this->ithCol($i);
      $di=$ar->matrixMultiplication($ii);
      $fc=$this->frstiCol($i-1);
      $mm=$fc->matrixMultiplication($di);
      $ci=$ii->matrixSubstraction($mm);
      $nim1=$matrix_n->mMinus($i);
      $nim1g=$nim1->Inverse();
      $li=$matrixN->mColumn($i);
     if($ci->isZero()==0){
          $nii=$matrix_n->mScalar($i);
          $dit=$di->matrixTranspose();
          $dtn=$dit->matrixMultiplication($nim1);
          $dtnd=$dtn->matrixMultiplication($di);
          $dtnd=$dtnd->matNum();
          $ditl=$dit->matrixMultiplication($li);
          $lit=$li->matrixTranspose();
          $litdi=$lit->matrixMultiplication($di);
          $ditdi=$ditl->matrixSubstraction($litdi);
          $ditdi=$ditdi->matNum();
          $nim1li=$nim1g->matrixMultiplication($li);
          $litnli=$lit->matrixMultiplication($nim1li);
          $litnli=$litnli->matNum();
          $lktar=$lit->matrixMultiplication($ar);
          $arnklk=$fc->matrixMultiplication($nim1li);
          $novo=$lktar->matrixMultiplication($arnklk);
          $novo=$novo->matNum();
          $si=$nii+$dtnd-$ditdi-$litnli+$novo;
          $dtnar=$dtn->matrixMultiplication($ar);
          $dilit=$dtnar->matrixSubstraction($lktar);
          $bit=$dilit->scalarMultiplication(1/$si);}
     else{
         $cit=$ci->matrixTranspose();
         $citm=$cit->matrixMultiplication($matrix_m);
         $pom=$citm->matrixMultiplication($ci);
         $pom=$pom->matNum();
```

```
        $bit=$citm->scalarMultiplication(1/$pom);}
    $nim1li=$nim1g->matrixMultiplication($li);
    $arai=ar->matrixMultiplication($fc);
    $arnim1=$arai->matrixMultiplication($nim1li);
    $pi=$nim1li->matrixSubstraction($arnim1);
    $k1=$di->matrixMultiplication($bit);
    $ark1=$ar->matrixSubstraction($k1);
    $pib=$pi->matrixMultiplication($bit);
    $ar=$ark1->matrixSubstraction($pib);
    $ar=$ar->appRow($bit);}
  return $ar;}
```

Funkcija A.2.15 Implementacija LM inverza, *Algoritam 2.1.4.*, je data sledećom funkcijom `LMInverse`().

```
function LMinverse($matrix_m,$matrix_n){
    $aa=$this->ithCol(1);
    if($this->isZero($aa)==0)
        $ar=$this->matrixTranspose($aa);
    else{
        $ta=$this->matrixTranspose($aa);
        $alb=$ta->matrixMultiplication($matrix_m);
        $ali=$alb->matrixMultiplication($aa);
        $inv=$this->Inverse($ali);
        $ar=$inv->matrixMultiplication($alb);}
    $minusJedan=array(array(-1));
    $minusJedan=new matrix($minusJedan);
    for($i=2;$i<=$this->get_num_columns();$i++){
        $ai=$this->ithCol($i);
        $ni=$ar->matrixMultiplication($ai);
        $fc=$this->frstiCol($i-1);
        $fcni=$fc->matrixMultiplication($ni);
        $di=$ai->matrixSubstraction($fcni);
        $mm=$matrix_n->mMinus($i);
        $mmi=$this->Inverse($mm);
        $vm=$matrix_n->mColumn($i);
        $pro=$mmi->matrixMultiplication($vm);
        $pro1=$ar->matrixMultiplication($fc);
        $pro2=$this->eye($i-1,$i-1);
        $pi=$pro2->matrixMultiplication($pro);
        if($this->isZero($di)==0){
            $z=$this->makeZero($this->get_num_rows());
            $ui=$ar->appRow($z);
            $qi=$ni->matrixAddition($pi);
            $qi=$qi->appRow($minusJedan);
            $qit=$this->matrixTranspose($qi);
            $deoM=$matrix_n->mMatrica($i)
            $qitm=$qit->matrixMultiplication($deoM);
            $bi=$qitm->matrixMultiplication($qi);
```

```
            $pomb=$this->matNum($bi);
            $hipom=$qitm->scalarMultiplication(1/$pomb);
            $ei=$hipom->matrixMultiplication($ui);}
        else{
            $dnt=$this->matrixTranspose($di);
            $dlni=$dnt->matrixMultiplication($matrix_m);
            $pom=$dlni->matrixMultiplication($di);
            $pom=$this->matNum($pom);
            $ei=$dlni->scalarMultiplication(1/$pom);}
        $k1=$ni->matrixMultiplication($ei);
        $ar=$ar->matrixSubstraction($k1);
        $pd=$pi->matrixMultiplication($ei);
        $ar=$ar->matrixSubstraction($pd);
        $ar=$ar->appRow($ei);}
    return $ar;}
```

A.3. Implementacija procedura za manipulaciju nad matricama u Cassandra NoSQL modelu

Funkcija A.3.1 Uspostavljanje konekcije sa *MatrixComputation keyspace*-om u *Cassandra DBMS*.

```
function getConection(){
    $servers = array(
              array('host' => '127.0.0.1',
                    'port' => 9160,
                    'use-framed-transport' => true,
                    'send-timeout-ms' => 1000,
                    'receive-timeout-ms' => 1000));
    $cassandra = Cassandra::createInstance($servers);
    $cassandra2 = Cassandra::getInstance();
    $cassandra->useKeyspace('MatricesComputation');
    $cassandra->setMaxCallRetries(5);
    return $cassandra;}}
```

Funkcija A.3.2 Unos matrica u *Cassandra DBMS* se vrši funkcijom `ColumnMatrixIn()`.

```
function ColumnMatrixIn($cassandra,$id,$dimension,$type){
    $cassandra->cf('matricesIn')->set(
    array('id'=>$array,
          'dimension'=>$m."x".$n,
           'type'=>'RGB'),
    Cassandra::CONSISTENCY_QUORUM);}}
```

Funkcija A.3.3 Pretraga matrica u familiji kolona *MatrixIn* se vrši funkcijom `getColumnMatrixIn()`.

```
function getColumnMatrixIn($cassandra,$array,$dimension,$type){
    $search=$cassandra->cf('MatrixIn')->getWhere(array('elements_in'=>
    $array,'dimension'=>$dimension,'type'=>$type));
    return $search->getAll();}
```

Funkcija A.3.4 Unos podataka u familiju kolona *MatrixOut* se vrši narednom funkcijom.

```
function insertColumnMatrixOut($cassandra,$id,
    $array,$matrix1,$operation){
    $cassandra->cf('matricesOut')->set(
    $id,
    array('elements_out'=>$array,
          'matrix1'=>$matrix1,
          'matrix2'=>$matrix2,
          'matrix3'=>$matrix3,
          'operation'=>$operation),
    Cassandra::CONSISTENCY_QUORUM);}
```

Funkcija A.3.5 Pretraga matrica u familiji kolona *MatrixOut*, *Cassandra DBMS*, se vrši funkcijom `getColumnMatrixOut()`.

```
function getColumnMatrixOut($cassandra,$array,$matrix1,$operation){
    $search=$cassandra→cf('matricesOut')→
            getWhere(array(
            'elements'out'=>$array,
            'matrix1'=>$matrix1,
            'operation' =>$operation));
    return $search->getAll();}
```

Funkcija A.3.6 Prikaz rešenja uskladištenog u familiji kolona *MatrixOut*.

```
function displayColumnMatrixOut($cassandra,$matrix1,$operation){
    $search=$cassandra→cf('matricesOut')→
            getWhere(array('matrix1'=>$matrix1,
                           'operation'=>$operation));
            $result=array_keys($search->getAll());
    return $result[0];}
```

B. Triangulacija konveksnih poligona

B.1. Implementacija algoritama

Algoritam 3.1.5. Daje najbolje rezultate pa ga stoga i implementiramo i koristimo kao radnu verziju. Na početku ćemo opisati nekoliko pomoćnih procedura.

Funkcija B.1.1 Funkcija `numCatalan()` služi za utvrđivanje broja triangulacija.

```
function numCatalan($n){
    $catalan = factorial(2*($n-2))/(factorial($n-1)*factorial($n-2));
 return $catalan;}
```

Funkcija B.1.2 Funkcija `remove_duplicate()` uklanja duple triangulacije.

```
function remove_duplicate($array,$n){
  $num_triangulation = count($array);
  for($i=0;$i<$num_triangulation;$i++){
    $string_c = "";
    for($j=0;$j<$n-2;$j++){
      $string = $array[$i][$j].",";
      $string_c = $string_c . $string;}
    $string_c2=substr($string_c,0,strlen($string_c)-1);
    $string_complit[$i]=$string_c2;}
 $string_complit = array_unique($string_complit);
 $string_complit = array_values($string_complit);
 $num_trian = numCatalan($n);
 for($i=0;$i<$num_trian;$i++){
  $array_w = explode(",",$string_complit[$i]);
    for($j=0,$z=0;$j<$n-2;$j++){
      for($k=0;$k<3;$k++,$z++){
        $array_2D[$j][$k]=$array_w[$z];}
      $string = implode(",",$array_2D[$j]);
      $array_db[$i][$j]=$string;}}
   return $array_db;}
```

Funkcija B.1.3 Funkcija `createE`() kreira skup $E_b(n)$.

```
function createE($array_row, $E){
 for($i=0;$i<3;$i++){
  $array[$i] = $array_row[0] . "," . $array_row[1];
  $array2 = array_merge($E,$array);
  $array_row = rotate($array_row,1);}
 array_multisort($array);
 return $array;}
```

Funkcija B.1.4 Funkcija kojom se inkrementira par iz skupa $E_d^i(n+1)$, koji zadovoljava uslov iz koraka 3 *Algoritma 3.1.5.*

```
function increment2($array,$last_num){
 for($i=0;$i<2;$i++){
  if($array[$i]>=$last_num)
   $array[$i]=$array[$i]+1;}
 return $array;}
```

Funkcija B.1.5 Funkcija kojom se kreira skup $E_d(n)$.

```
function connect2($array1,$array2,$n){
 for($j=0;$j<$n-2;$j++){
  $array[$j][0] = $array1;
  $array[$j][1] = $array2[$j];
  $array[$j] = $array[$j][0] . "," . $array[$j][1];}
 return $array;}
```

Funkcija B.1.6 Implementacija *Algoritma 3.1.5* u *ALDF* formi.

```
function decomp($n,$link){
 $n_p = $n-1;
 $number_trian = numCatalan($n_p);
 $q = 1;
 $brojac = 1;
 $Eb = createEb($n-1);
 $fp = fopen($link, "r");// fajl za n-1
 $link_n = "./files1/" . $n . ".txt";
 $fp_n=fopen($link_n,"w+"); // fajl za n
 $array_out = array();
 for($i=0;$i<$number_trian;$i++){
    $string_row = fgets($fp);
    $array_1D = array(array());
    $array_1D[$i] = connect2(chop($string_row),$Eb,$n);
    for($j=0;$j<$n-2;$j++){
      $array_string = array(array()); $array_row = array(array());
      $array_row = explode(",",$array_1D[$i][$j]);
      if(abs($array_row[2*($n-3)-2]-$array_row[2*($n-3)-1])==1){
         $max = max($array_row[2*($n-3)-2],$array_row[2*($n-3)-1]);
         $increment = increment($array_row,$max,$n);
         $sortiran_string = sortiran_string($increment,$n);
         $array_out[] = $sortiran_string;
         $brojac++;}
      elseif($array_row[2*($n-3)-2]==$n-1 and $array_row[2*($n-3)-1]==1){
       $sortiran_string = sortiran_string($array_row,$n);
       $array_out[] = $sortiran_string;}}
   if($n>15){
    if(count($array_out)>=5000000){
      $array_out = array_unique($array_out);
```

```
          foreach($array_out as $key => $value){
            fwrite($fp_n,$value."\n");}
       $array_out = array();}}
   $array_out = array_unique($array_out);
   array_multisort($array_out);
   foreach($array_out as $key => $value){
      fwrite($fp_n,$value."\n");}
   return $array_out;}
```

Funkcija B.1.7 Funkcija za paralelni transfer podataka iz fajla u bazu podataka.

```
function file_mysql($i){
  $query = "TRUNCATE TABLE `temporal`";
  $result = mysql_query($query);
  $file = $i . ".txt";
  $query0 = ' LOAD DATA LOCAL INFILE
           \'C:\\\\wamp\\\\www\\\\files\\\\' . $file . '\''
           . ' INTO TABLE `temporal2` '
           . ' LINES TERMINATED BY \'\\n\''
           . ' (`TRIANGULATION`);';
  $result0 = mysql_query($query0);}
```

Funkcija B.1.8 Funkcija za transfer podataka iz tabele baze podataka u fajl.

```
function mysql_file($i){
  $file = $i . ".txt";
  $link = "./files/" . $file;
  unlink($link);
  $query1='SELECT TRIANGULATION INTO OUTFILE
         \'C:\\\\wamp\\\\www\\\\' . $file . '\''
         . ' LINES TERMINATED BY \'\\n\''
         . ' FROM `temporal2`;';
  $result1 = mysql_query($query1);}}
```

Funkcija B.1.9 Implementacija Hurtadovog *Algoritma 3.1.1*.

```
function hurtado($n, $link){
   $e = $n+1;
   $p=0;
   $fp = fopen($link, "r");// fajl za n-1
   $link_n = "./hfiles1/" . $e . ".txt";
   $fp_n=fopen($link_n,"w+");
   for($k=0;$k<numCatalan($n);$k++){
        $novi=array();
        $red = trim(fgets($fp));
        $red_a = array();
        $red_a = explode(",",$red);
        $novi = array1D_2Darray2($red_a);
        $parovi_nn = extract_n2($novi,count($novi),$n);
        for($i=0;$i<count($parovi_nn);$i++){
```

```
            $s=0;
            $red_b = array();
            $red_b=$red_a;
            $edim1 = explode(";",$parovi_nn[$i]);
            $edim2 = explode(",",$edim1[1]);
            $t = $edim2[0];
            $parovi_ns = explode(",",$parovi_nn[$i]);
            $trian_peir=array();
            for($j=0;$j<count($parovi_nn);$j++){
                 $edim1 = explode(";",$parovi_nn[$j]);
                 $edim2 = explode(",",$edim1[1]);
                 $t2 = $edim2[0];
                 if($t2<$t){
                      $red_b[2*$edim1[0]+1]=$n+1;}}
            $red_b[2*(2*$n-3)] = $t;
            $red_b[] = $n+1;
            $red_b[] = $n;
            $red_b[] = $n+1;
            $trian[$n+1][$p] = implode(",",$red_b);
            fwrite($fp_n,implode(",",$red_b)."\n");
            print_r($trian[$n+1][$p]);echo "<br>";
            $p++;}}
return $trian;}
```

B.2. Obrada slike

Funkcija B.2.1 Funkcija za transformaciju slike u niz koji sadrži *R, G i B* boje svakog piksela slike.

```
function getRFB($im){
    $x = imagesx($im);
    $y = imagesy($im);
    for($i=0;$i<$x;$i++){
        for($j=0;$j<$y;$j++){
            $rgb = imagecolorat($im, $i, $j);
            $r = ($rgb >> 16) & 0xFF;
            $g = ($rgb >> 8) & 0xFF;
            $b = $rgb & 0xFF;
            $color_tran[$i][$j][0] = $r;
            $color_tran[$i][$j][1] = $g;
            $color_tran[$i][$j][2] = $b;}}
    return $color_tran;}
```

Funkcija B.2.2 Funkcija za iscrtavanje slike na osnovu niza `$color_trian` koji sadrži podatke o *R*, *G i B* bojama pojedinačnih piksela.

```
function setRGB($x,$y,$color_tran){
    $gd = imagecreatetruecolor($x,$y);
    for($i=0;$i<$x;$i++){
        for($j=0;$j<$y;$j++){
            $color = imagecolorallocate($gd,$color_tran[$i][$j][0],
                    $color_tran[$i][$j][1],$color_tran[$i][$j][2]);
            imagesetpixel($gd,$i,$j,$color);}}
header('Content-Type: image/png');imagepng($gd);}
```

Funkcija B.2.3 Kodiranje slike korišćenjem `base_64()` procedure.

```
function base64_encode_image($filename=string,$filetype=string) {
    if($filename){
        $imgbinary = fread(fopen($filename,"r"),filesize($filename));
        return base64_encode($imgbinary);}}
```

Funkcija B.2.4 Dekodiranje slike koja je kodirana procedurom `base_64()`.

```
function base64_decode_image($string){
    $string = base64_decode($string);
    $image = imagecreatefromstring($string);
    if($image !== false){
        header('Content-Type: image/jpg');// i ovde treba stavit png
        imagejpeg($image);// i ovde treba imagejpeg umesto imagepng}}
```

B.3. Grafičke mogućnosti

B.3.1. Iscrtavanje stubičastih dijagrama

U ovom delu ćemo predstaviti procedure za crtanje grafikona uz pomoć *PHP*-a. Predstavljeni kod predstavlja korigovanu i prilagođenu verziju koda iz [77].

Funkcija B.3.1 Funkcija za kreiranje grafikona.

```
    function create_image(){
        // funkcija za iscrtavanjekoordinatnog sistema.
        $this→iscrtavanjeCoordinatnogSistema();
         // funkcija za iscrtavanje legende.
        $this->ispisivanjeNatpisa();
         // Iscrtavanje vertikalnih linija koje daju 3D utisak
        $_crr_blank = $this→iscrtavanjeVrednostiCordinata();
         // funkcija  za icrtavanje stubaca grafikona
        $this→IscrtavanjeBlokovaDijagrama();
        return $img;}
```

Funkcija B.3.2 Funkcija za kreiranje PNG slike.

```
  function createPNG($img){
     $img = $this->createImage($arrayValue, $commons);
     imagepng($img);
     ImageDestroy($img);}}
```

B.3.2. Iscrtavanje triangulacija

Funkcija B.3.3 Funkcija za utvđivanje temena pravilnog n-tougaonog poligona.

```
function GetPoints($sides){
    $d = 360/$sides;
    $y = 15;
    $x = 175;
    $points[] = $x;
    $points[] = $y;
    $sidelength = 50;
    $rotation = $d;
    for($i=1;$i<$sides;$i++){
        $x=cos(DegToRad($rotation))*$sidelength+$x;
        $points[]=$x;
        $y=sin(DegToRad($rotation))*$sidelength+$y;
        $points[]=$y;
        $rotation+= $d;}
return $points;}
```

Funkcija B.3.4 Funkcija za iscrtavanje dijagonala poligona.

```
function draw_diagonal($sides,$temp,$colorL,$coord_diagonal){
    for($i=0;$i<=$sides-3;$i++){
        imageline($temp,$coord_diagonal[4*$i],$coord_diagonal[4*$i+1],
        $coord_diagonal[4*$i+2],$coord_diagonal[4*$i+3],$colorL);}}
```

Funkcija B.3.5 Funkcija za iscrtavanje svih triangulacija za dato n.

```
function graficTriangulation ($edges){
   $query = "select * from trian_" . $edges;
   $result = mysql_query($query, $db_handle);
   while($row = mysql_fetch_row($result)){
       $temp = @imagecreate($Image_Width, $Image_Height);
       $black = imagecolorallocate($temp, 0, 0, 0);
       imagecolortransparent($temp, $black);
       $background_color = imagecolorallocate($temp , 239, 228, 239);
       $color = imagecolorallocate($temp,0,0,0);
       $colorL = imagecolorallocate($temp,0,0,0);
       $point_array = GetPoints($edges);
       $point_string = implode(",", $point_array);
       imagepolygon($temp, $point_array, count($point_array)/2, $color);
```

```
$i = $row[0];
$db_diagonal = $row[1];
$db_diagonal = str_replace(",","/,/",$db_diagonal);
$db_diagonal = "/" . $db_diagonal . "/";
$replaced = point_array_2D($edges,$point_array);
$coord_diagonal = coordinate($edges,$replaced,$db_diagonal);
draw_diagonal($edges,$temp,$colorL,$coord_diagonal);
$i = $edges . "." . $i;
imagepng($temp,'./figures/' . $i . '.png');
imagepng($temp);
imagedestroy($temp);}
```

Lista oznaka

det A	determinanta matrice A
I	inverzna matrica
A^{-1}	jedinična matrica
A^T	transponovana matrica
$m \times n$	dimenzije matrice
$adjA$	adjungovana matrica
$A^\dagger$	generalisani MP inverz
A^*	transponovana i konjugovana matrica
$\tilde{A}$	transponovana matrica
$A^\dagger_{MN}$	težinski MP inverz
$A^\dagger_{LM}$	generalisani LM inverz
$A_1 \dots A_n A_{n+1}$	poligonska linija
$A_n A_{n+1}$	ivica poligonske linije
C_{n-1}	katalanov broj
E	skup svid dijagonala u trouglu
E_+	Skup dijagonala sa spoljašnjom ivicom
E_b	ostale spoljne ivice
$\lfloor \cdot \rfloor$	nad funkcija

Spisak algoritama

Spisak tabela

Spisak slika

Literatura

[1] A. Ben-Israel, T.N.E. Greville, *Generalized Inverses. Theory and Applications, Second edition*, CMS Books in Mathematics/Ouvrages de Mathmatiques de la SMC, 15. Springer-Verlag, New York, 2003.

[2] S.L. Campbell, C.D. Meyer Jr., *Generalized Inverses of Linear Transformations*, Pitman, London, 1979.

[3] F. Chang, J. Dean et. al. *Bigtable: A distributed storage system for structured data*, In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation, Volume 7, 205–218, 2006.

[4] T. Converse, J. Park, C. Morgan, *PHP 5 and MySQL Bible*, Wiley Publishing, Inc., USA, 2004.

[5] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, P. Lohman, *Non-stop SQL/MX primitives for knowledge discovery*, In ACM KDD Conference, (1999) 425-429.

[6] T. Chong, S.D. Sharma, E. Brewer, J. Saltz, *Sparse matrix application, Parallel Process*. Lett. 5(4) (1995), pp. 671–683.

[7] B. Chazelle, *Triangulation a Simple Polygon in Linear Time*, Discrete Comput. Geom. 6 (1991), 485-524.

[8] G. DeCandia, et. al. *Dynamo: Amazons highly available key-value store* In Proceedings of 21th ACM SIGOPS symposium on Operating systems principles, 205–220, ACM, 2007.

[9] E. Hewit, *Cassandra: The definitive guide*, OReilly Media, Inc., USA, 2011.

[10] R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, 4th edition, 2003.

[11] S. Egilsson, H. Gudbjartsson, S. Sigurjnsson, *SQL query generator utilizing matrix structures*, US Patent 6,578,028, 10 June 2003.

[12] R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems, 4th ed.*, Addison-WesleyY., Reading, MA, USA, 2003

[13] Y. Fana, R. Kalaba, *Dynamic programming and pseudo-inverses*, Appl. Math. Comput. 139 (2003) 323–342.

[14] R. C. Gonzales, R. E. Woods: *Digital image processing*, Prentice Hall, 2002.

[15] M.R. Garey, D.S. Johnson, F.P. Preparata, R.E. Tarjan, *Triangulating a simple polygon*, Inform. Process. Lett. 7 (1978), 175-180.

[16] D. Guldbransen, *Special edition using XML, Second edition*. Indianapolis, Indiana 2002.

[17] J. Greenspan, B. Bulger, *MySQL/PHP Database Applications*, M&T Books: An imprint of IDG Books Worldwide, Inc., Foster City, NY, USA, 2001.

[18] T.N.E. Grevile, *Some applications of the pseudo-inverse of matrix*, SIAM Rev. 3 (1960), pp. 15–22.

[19] D. Hunter et. al, *Begining XML*, Wrox Press, Birmingham UK, 2001.

[20] F. Hurtado, M. Noy, *Graph of Triangulations of a Convex Polygon and tree of triangulations*, Comput. Geom. 13 (1999), 179-188.

[21] C. Itiki, *Dynamic programming and diagnostic classification*, J. Optim. Theory Appl. 127 (2005), pp. 579–586.

[22] E. Karaman, E. Kayhan, *A client server based model for parallel matrix computation*, SIAM Rev. 3 (1960), pp. 15–22.

[23] T. Kurmayya, K.C. Sivakumar, *Moore-Penrose inverse of a Gram matrix and its nonnegativity*, J. Optim. Theory Appl. 139 (2008), pp. 201–207.

[24] P.V. Krtolica, P.S. Stanimirović, *On Some Properties of Reverse Polish Notation*, FILOMAT 13 (1999), 157-172.

[25] P.V. Krtolica, P.S. Stanimirović, R. Stanojević, *Reverse Polish Notation in Constructing the Algorithm for Polygon Triangulation*, FILOMAT 15 (2001), 25-33.

[26] P.V. Krtolica, P.S. Stanimirović, *Reverse Polish Notation Method*, Int. J. Comput. Math. 81:3 (2004), 273-284.

[27] P.V. Krtolica, P.S. Stanimirović, M.B. Tasić, S.H. Pepić, *Triangulation of Convex Polygon with Storage Suport.* - submitted

[28] A., Lakshman, P., Malik, *Cassandra: a decentralized structured storage system*, SIGOPS Oper. Syst. Rev. 44 (2010) 35-40.

[29] J.B. Layton, *Efficient direct computation of the pseudo-inverse and its gradient*, Internat. J. Numer. Methods Engrg. 40 (1997), pp. 4211–4223.

[30] S.Y.T. Lee, H.W. Kim, S. Gupta, *Measuring open source software success*, OMEGA-Int. J. Manag. Sci 37 (2009), pp. 426–438.

[31] E.H. Moore, *On the reciprocal of the general algebraic matrix*, Bull. Amer. Math. Soc. 26 (1920), 394-395.

[32] S. Mohideen, V. Cherkassky, *On recursive calculation of the generalized inverse of a matrix*, ACM Trans. Math. Softw. 17 (1991) 130–147.

[33] MySQL Developer Zone, http://dev.mysql.com/doc/refman/5.0/en/full-table.html

[34] J. Meloni, *PHP 5*, Thomson Course Techology, Boston, MA, 2004.

[35] J. Miao, *Representations for the weighted Moore-Penrose inverse of a partitioned matrix*, J. Comput. Math., 7(1989), pp.320-323.

[36] S. Mohideen, V. Cherkassky, *On recursive calculation of the generalized inverse of a matrix*, ACM Trans. Math. Software 17 (1991), pp. 130–147.

[37] J.C. Nash, R.L.C.Wang, *Algorithm 645 Subroutines for testing programs that compute the generalized inverse of a matrix*, ACM Trans. Math. Software 12 (1986), pp. 274–277.

[38] P. Noble, *A methods for computing the generalized inverse of a matrix*, SIAM J. Numer.Anal. 3 (1966) 582-584.

[39] M.Z. Nashed, *Generalized Inverses and Applications*, Academic Press, New York,

[40] C. Ordonez, J. García, *Vector and Matrix Operations Programmed with UDFs in a RelationalDBMS*, Proceedings of the 15th ACM International Conference on Information and Knowledge Management, Arlington, VA, 2006, pp. 503–512.

[41] R. Page, P. Factor, *SQL Server Matrix Workbench*, 2008. Available at http://www.simple-talk.com/sql/t-sqlprogramming/ sql-server-matrix-workbench/.

[42] P. Phohomsiri, B. Han, *An alternative proof for the recursive formulae for computing the Moore–Penrose M-inverse of a matrix*, Appl. Math. Comput. 174 (2006) 81–97.

[43] R. Penrose, *A generalized inverse for matrices*, Proc. Cambridge Philos. Soc. 51 (1955), 406-413

[44] M.D. Petković, P.S. Stanimirović, M.B. Tasić, *Effective partitioning method for computing weighted Moore–Penrose inverse*, Comput. Math. Appl. 55 (2008), pp. 1720–1734.

[45] S.H. Pepić, *"Weighted Moore - Penrose inverse: PHP vs. MATHEMATICA"*. FACTA, Niš 25 (2011), pp. 35–45.

[46] S.H. Pepić, M.B. Tasić, *"Matrix Computations: MySQL vs. Cassandra DataBase"*. - submitted

[47] M.D. Petković, *Simboličko izračunavanje henkelovih determinanti i generalisanih inverza matrica*, doktorska disertacija, Niš 2008.

[48] F.P. Preparata, M.I. Shamos, *Computational Geometry-An Introduction*, Springer-Verlag, Berlin, (1988).

[49] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, (1987).

[50] C.R. Rao, S.K. Mitra, *Generalized Inverse of Matrices and its Applications*, John Wiley and Sons, Inc, New York, London, Sydney, Toronto, 1971.

[51] P.S. Stanimirović, M.B. Tasić, *A problem in computation of pseudoinverses*, Appl. Math. Comput. 135 (2003), pp. 443–469.

[52] P.S. Stanimirović, M.B. Tasić, *Partitioning method for rational and polynomial matrices*, Appl. Math. Comput. 155 (2004), pp. 137–163.

[53] P.S. Stanimirović, M.B. Tasić, *Computing generalized inverses using LU factorization of matrix product*, Int. J. Comput. Math. 85 (2008), pp. 1865–1878.

[54] K.C. Sivakumar, *Proof by verification of the Greville/Udwadia/Kalaba formula for the Moore–Penrose inverse of a matrix*, J. Optim. Theor. Appl. 131 (2006) 307–311.

[55] M.B. Tasić, P.S. Stanimirović, *Symbolic and recursive computation of different types of generalized inverses*, Appl. Math. Comput. 199 (2008) 349–367.

[56] M.B. Tasić, P.S. Stanimirović, *Differentiation of generalized inverses for rational and polynomial matrices*, Appl. Math. Comput. 216 (2010), pp. 2092–2106.

[57] M.B. Tasić, P.S. Stanimirović, M.D. Petković, *Symbolic computation of weighted Moore-Penrose inverse using partitioning method*, Appl. Math. Comput. 155 (2004), pp. 137–163.

[58] M.B, Tasić, P.S, Stanimirović, S.H. Pepić, *Computation of generalized inverses using Php/MySql environment.* International Journal of Computer Mathematics, Vol. 88, No.11, (2011), 2429-2446.

[59] M.B. Tasić, P.S. Stanimirović, S.H. Pepić, *About the generalized LM-inverse and the weighted Moore–Penrose inverse*, Appl. Math. Comput. 216 (2010), pp. 114–124.

[60] F.E. Udwadia, R.E. Kalaba, *Analytical Dynamics*. A New Approach, Cambridge University Press, Cambridge, England, 1996.

[61] F.E. Udwadia, P. Phohomsiri, *Generalized LM-inverse of a matrix augmented by a column vector*, Appl. Math. Comput. 190 (2007) 999–1006.

[62] F.E. Udwadia, P. Phohomsiri, *Recursive formulas for generalized LM-inverse of a matrix*, J. Optim. Theor. Appl. 131 (2007) 1–16.

[63] F.E. Udwadia, P. Phohomsiri, *Recursive determination of the generalized Moore–Penrose M-inverse of a matrix*, J. Optim. Theor. Appl. 127 (2005) 639–663.

[64] F.E. Udwadia, R.E. Kalaba, *An alternative proof for Greville's formula*, J. Optim. Theor. Appl. 94 (1997) 23–28.

[65] G.R.Wang, Y.L.Chen, *A recursive algorithm for computing the weighted Moore-Penrose inverse* $A^{\dagger}_{MN}$, J. Comput. Math. 4 (1986), pp. 74–85.

[66] G.R. Wang, *A new proof of Greville's method for computing the weighted MP inverse*, J. Shangai Teach. Univ. Nat. Sci. Ed. 3 (1985) 32–38.

[67] G. Wang, Y. Wei, S. Qiao, *Generalized Inverses: Theory and Computations*, Science Press, 2003.

[68] H. Williams, D. Lane, *Web Database Applications with PHP & MySQL, 2nd Edition*, O'Reilly Media, Inc., Beijing, Cambridge, Famham, KÄoln, Paris, Sebastopol, Taipei, Tokyo, 2004.

[69] S. Wolfram, *The Mathematica Book*, 5th ed., Champaign: Wolfram Media, Inc., 2004.

[70] G. Zielke, *Report on test matrices for generalized inverses*, Computing 36 (1986), pp. 105–162.

[71] F. Zhang, *Matrix Theory: Basic Results and Techniques*, New York: Springer-Verlag,

[72] *Zend Technologies* http://www.zend.com

[73] http://www.w3schools.com/browsers/browsers_stats.asp

[74] http://www.w3.org

[75] http://www.php.net

[76] http://mysql.com

[77] http://www.phpclasses.com

[78] http://www.juillerat-marly-ch.net/sql/lin_eq_with_sql.pdf

ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ
НИШ

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		монографска
Тип записа, **ТЗ**:		текстуални / графички
Врста рада, **ВР**:		докторска дисертација
Аутор, **АУ**:		Селвер Х. Пепић
Ментор, **МН**:		Милан Б. Тасић
Наслов рада, **НР**:		МАТРИЧНА ИЗРАЧУНАВАЊА У PHP/MySQL ОКРУЖЕЊУ
Језик публикације, **ЈП**:		српски
Језик извода, **ЈИ**:		енглески
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		Србија
Година, **ГО**:		2012.
Издавач, **ИЗ**:		ауторски репринт
Место и адреса, **МА**:		Ниш, Вишеградска 33.
Физички опис рада, **ФО**: (поглавља/страна/цитата/табела/слика/графика/прилога)		6 поглавља/161 страна/139 цитата/28 табела/58 слика/11 графика/ 1 прилог
Научна област, **НО**:		рачунарске науке
Научна дисциплина, **НД**:		Веб програмирање, базе података
Предметна одредница/Кључне речи, **ПО**:		Матрична алгебра, MatrixDBMS модел, MySQL DB, Cassandra модел података, триангулација конвексног полигона.
УДК		517.983.2 (043.3) 517.984/.986 (043.3)
Чува се, **ЧУ**:		библиотека
Важна напомена, **ВН**:		
Извод, **ИЗ**:		Матрични модели су нашли широку примену у економији, анализа података и текста, рачунарској графици итд. Проблем који се јавља у матричним израчунавањима је време извршавања операција и поновног израчунавања. Основни циљ дисертације је да се смањи CPU време приликом извршавања матричних операција. Развијен је нови модел података назван MatrixDBMS. Такође су уведени и нови приступи матричним израчунавањима помоћу Cassandra NoSQL модел података и методом паралелног програмирања. Ефикасност уведеног Matrix/DBMS модела потврђена је применом при триангулацији конвексног полигона и обради слика. За различита матрична израчунавања развијена је клијент/сервер апликација у PHP/MySQL окружењу.
Датум прихватања теме, **ДП**:		09.05.2011.
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	
	Члан:	
	Члан:	
	Члан:	
	Члан, ментор:	

ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ
НИШ

KEY WORDS DOCUMENTATION

Accession number, **ANO**:		
Identification number, **INO**:		
Document type, **DT**:		monograph
Type of record, **TR**:		textual/graphic
Contents code, **CC**:		doctoral dissertation
Author, **AU**:		Selver H. Pepić
Mentor, **MN**:		Milan B. Tasić
Title, **TI**:		MATRIX COMPUTATIONS IN PHP/MYSQL ENVIRONMENT
Language of text, **LT**:		Serbian
Language of abstract, **LA**:		English
Country of publication, **CP**:		Serbia
Locality of publication, **LP**:		Serbia
Publication year, **PY**:		1998
Publisher, **PB**:		author's reprint
Publication place, **PP**:		Niš, Višegradska 33.
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/ appendixes)		6 chapters/161 pages/139 citation/28 tables/58 mages/11 graphs/1 appendix
Scientific field, **SF**:		Computer science
Scientific discipline, **SD**:		Web programming, database storage system
Subject/Key words, **S/KW**:		Matrix algebra, MatrixDBMS model, MySQL DB, Casandra data model, Triangulation of the convex polygon
UC		517.983.2(043.3) 517.984/.986(043.3)
Holding data, **HD**:		library
Note, **N**:		
Abstract, **AB**:		Matrix models are encountered in the economy, data mining, text mining, computer graphics etc. The problem that arises in the matrix computations is the computational time and re computation. Basic goal of the dissertation is to decrease the CPU execution time of matrix operations. A new data model called MatrixDBMS is developed. We also introduced the new approaches to matrix computations using Cassandra NoSQL data model and the principle of parallel programming. The effects of introduced MatrixDBMS model are confirmed with the examples of triangulation convex polygon and image processing. A client/ server applications for matrix computations is developed in PHP/MySQl environment.
Accepted by the Scientific Board on, **ASB**:		
Defended on, **DE**:		09.05.2011.
Defended Board, **DB**:	President:	
	Member:	
	Member:	
	Member:	
	Member, Mentor:	