LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

VARBŪTISKI ALGORITMI IDENTITĀŠU PĀRBAUDEI

BAKALAURA DARBS

Autors: **Dmitrijs Gavrilovs**
Stud. apl. dg05018
Darba vadītājs: Rūsiņš Mārtiņš Freivalds, Dr. habil. math., LZA akadēmiķis

RĪGA 2009

Anotācija

Bakalaura darbā tiek apskatīti dažādi varbūtiski un determinēti algoritmi, lai pārbaudītu identitāti. Algoritmi ir aprakstīti, izanalizēti un noprogrammeti Delphi 7 vidē. Determinētu un varbūtisku algoritmu ātrumi ir salīdzināti. Ir doti testu datu rezultāti un ir izdarīti secinājumi. Darbs ir interesants ar to, ka tika veiktā analīze svarīgiem, no praktiskā viedokļa, algoritmiem.

Annotation

The work examines different randomness and deterministic algorithms for identity testing. The algorithms are described, analysed and developed using Delphi 7. Working speed of randomness and deterministic algorithms are evaluated. The test results are given and conclusions are made. The work is interesting because there has been made an analysis of important practical algorithms.

Atslēgvārdi

Algoritms

Varbūtisks algoritms

Identitāte

Saturs

Ievads.

Ir pazīstams teiciens : „Cilvēces vēsture – ir vēsture par tās kariem.” Bet tāpat cilvēces vēsture tā ir vēsture par tās atklājumiem, izmēģinājumiem un uzdevumu risinājumiem.
No rakstības sākumiem un metālu apstrādāšanas līdz kosmisko kuģu palaišanas kosmosā. Visā pasaules vēsture cilvēks ne tikai iznīcināja, bet arī cēla un veidoja. Bet veidošanas procesu pavada arī ar lietojamo uzdevumu risināšanu. Cilvēces attīstības gaitā izrēķināšanas tilpumi un sarežģītība strauji auga. Un parādījās tādi uzdevumi, kurus ar parastiem determinētiem algoritmiem bija ļoti sarežģīti atrisināt. Tā radās vajadzība pēc varbūtiskiem algoritmiem.

Parastiem lietotājiem varbūtiskie algoritmi liekas ļoti aizdomīgi.
Liekas, ka rezultātu ieguvums ir ne stipri ātrāks, bet rezultāts ir pārāk neprecīzs. Bet īstenībā tas nav tā. Es savā bakalaura darbā gribu parādīt tieši to un pat vairāk, ka varbūtiskie algoritmi, kaut kādu uzdevumu klasēm ir labāki par determinētiem algoritmiem.
Es izvēlējos atsevišķu uzdevumu klase – identitātes pārbaudi.
Es gatavojos parādīt varbūtisko algoritmu pārākumu par determinētiem algoritmiem, risinot dažus šīs klases uzdevumus un salīdzinot dabūto varbūtisko algoritmu rezultātus ar rezultātiem, kuri dabūti ar determinētiem algoritmiem.

Es izvēlējos šī uzdevuma klasi, tāpēc ka materiālu par šiem algoritmiem tīklā ir pārāk maz, tēma ir samērā jauna un varbūt izmantoti dažādi objekti matemātikā.

Pirmā darba daļa tiek doti algoritmu terminu definīcija : kas ir algoritms, varbūtisks algoritms, kuriem mērķiem tie ir paredzēti un kur ir to atšķirības. Tāpat tiek aprakstīts tas, kas bija izdarīts pirms mana bakalaura darba.

Otrā daļā tiek realizēti dažādi algoritmi, doti to analīze un testu rezultāti.

Trešā daļā tiek izdarīti secinājumi.

Algoritms, varbūtisks algoritms, varbūtisks algoritms lai pārbaudītu identitāti

Lai par kaut ko runātu, tas ir jādefinē.
Kas tas ir tie algoritmi ? Varbūtiskie algoritmi.
Kāda ir viņu atšķirība un kuriem mērķiem tie domāti

Mūsdienu pasaulē, cilvēce sastopas ar lielu skaitu dažādiem matemātiskiem uzdevumiem. Viena no uzdevumu klasēm ir divu objektu identitātes pārbaude. Piemēram ir dots uzdevums A* B = C, kur A,B,C ir kaut kādi naturāli skaitļi. Vajag pārbaudīt A * B = C identitāti. Kā to var izdarīt ? Parasts atrasinājuma algoritms ir tāds : sareizināt A * B un salīdzināt rezultātu ar C. Tas lieliski strādā ar skaitļiem, kuri nepārsniedz robežu 10^4 , bet ja mums ir skaitļi, kuriem kārta ir apmēram 10^{80} (skaitļi kuru var salīdzināt ar atomu skaitļi pasaulē) tad parādās problēmas par mūsdienu datoram – izrēķināšanas process ilgs pārāk ilgi. Ja mums ir $10^{100..000}$ skaitļu kārtība, tad mēs riskējam nekad nesagaidīt atbilde pat līdz pasaules eksistēšanas galam. Ko darīt ? Var izmantot varbūtisko algoritmu, lai pārbaudītu identitāti. Ja atbilde uz uzdevumu TRUE / FALSE būs pareiza ar varbūtība > ½, bet algoritma darbības laiks būs ievērojami zemāks, tad ir acīm redzams, ka varbūtisko algoritmu, lai pārbaudītu identitāti ,izmantošana ir pareiza.

Kaut dotais piemērs nav praktisks uzdevums(diez vai kam kaut kad būs vajadzība reizināt tādus lielus skaitļus), vien alga tas labi parāda iespējamo situāciju.

Šī bakalaura darba mērķis ir parādīt tādu varbūtisku algoritmu eksistenci, kuri darbojas labāk nekā determinētie algoritmi līdzīgā situācijā. Es parādīšu varbūtisko algoritmu piemērus dažādiem objektiem matemātikā, un ar programmas testu palīdzību parādīšu, ka tie ir labāki nekā determinēti algoritmi.

Vispirms vajag aprakstīt to, kas ir izdarīts šajā virzienā līdz manam bakalaura darbam, un tā arī definēt terminu algoritms, varbūtisks algoritms, kas ir šo algoritmu atšķirība un kādam mērķim tie domāti.

Algoritma terminu definēšana

Algoritms senajā traktēšanas sistēmā ir noteikta instrukciju virkne, kura apraksta izpildītāja darbību kartību, lai sasniegtu uzdevuma risināšanas rezultātu par galīgo laiku. Paralēles datora darbības attīstības gaitā vārdu „virkne" sāk mainīt ar vairāk vispārīgo vārdu „kārtība". Tas ir saistīts, ar to ka kaut kādas darbības algoritmos vajag izpildīt tikai vienu aiz otra, bet kaut kāds citi var būt neatkarīgi. Bieži izpildītajā lomā darbojas kaut kāds mehānisms(dators, šujmašīna, darbagalds), bet algoritma termins nav obligāti piederēt pie datorprogrammas, tā piemēram, precīzi uzrakstīta ēdiena recepte, kuru izgatavo, ari ir algoritms, šajā gadījumā izpildītājs ir cilvēks

Definēt terminu algoritms var dažādi.(Nav kopīgu, viennozīmīgu uzskatu) Kā šo terminu definēja daži slaveni autori.

Algoritms – tas ir nobeigta likumu virkne, kura norada operāciju kartību, lai risinātu konkrētu uzdevumu kopu un tai ir piecas svarīgas īpašības : galīgums, noteiktība, ievads, izvads, efektivitāte> (*D. E. Knuts*)

Algoritms – tā ir jebkura izrēķināšanu sistēma, kura tiek izpildita pēc stingri definētiem noteikumiem, kura pēc kaut kādu soļu skaita noteikti noved pie dotā uzdevuma atrisināšanas (*A. Kolmogorovs*)

Algoritms – tas ir noteikts priekšraksts, kurš definē izrēķināšanas procesu, kurš iet no variētiem datiem līdz meklētam rezultātam. (*A. Markovs*)

Algoritms ir noteikts priekšraksts par operācijas sistēmu izpildi noteiktā kartībā, kura risina visus noteikta tipa uzdevumus> (*Философский словарь / Под ред. М. М. Розенталя*)

Algoritms – stingri determinēta darbību virkne, kura apraksta objektu pārmaiņu no sākuma līdz beigu stāvoklim, un ir uzrakstīta ar komandām, kuras ir saprotamas izpildītājiem. (*Николай Дмитриевич Угринович, учебник «Информатика и информ. технологии»*)

Algoritms – darbību virkne, kura izdarīta, lai dabūtu noteikto rezultātu, izmantojot galīgo soļu skaitu.

Algoritms, tas ir saprotams un precīzs priekšraksts izpildītājam izdarīt galīgo soļu skaitu, lai risinātu doto uzdevumu.

Algoritms, tā ir darbību virkne, kura noved pie uzdevuma atrisināšanas, vai paskaidro, kāpēc šo uzdevumu nevar risināt

Es definēju algoritmu tā : Tiek nodefinēta objektu beigu un sākumu stāvokļi pēc tam ir nodefinēta izpildītājs vai izpildītāju grupa. Var veidot darbību virkni, lai pārietu no sākuma stāvokļa uz beigu stāvokli.(Pie nosacījuma, ka pāreja ir iespējama) – Šī darbības virkne ir algoritms šo uzdevumu risināšanai. Šī darbības virkne ir galīga. Katra darbība virknē ir saprotama izpildītājām, un ir viennozīmīga. Tādu algoritmu sauc par parastu.

Algoritmu forma

Algoritms var tikt pierakstīts vārdiem vai attēlots shematiski. Parasti sākumā (idejas līmenī) algoritms tiek aprakstīts vārdiem, bet, tuvojoties realizācijai, tas iegūst daudz formālāku apveidu un formulējumu valodā, kas saprotama izpildītājam (piemēram, mašīnkods). Kā vienu no algoritma pieraksta veidiem var minēt blokshēmas. Vēl viens pieraksta veids ir pseidokods, kas nav atkarīgs no konkrētās programmēšanas valodas.

Parasta algoritmu piemēri

Parasts algoritms vieglam uzdevumam.

Uzdevums

Vienādojums vispārīgā veidā.

A + X = B (Kur, A,B э N{1..10})

Algoritms, lai risinātu uzdevumu

Atrast vērtību izteiksmi B - A

Algoritms, lai risinātu grūtu uzdevumu

uzdevums

Izgatavot ābolu pīrāgu

Izpildītājs – ir mājsaimniece.

Algoritms

- ✓ Atrast 5 ābolus un sagriezt to.
- ✓ Izgatavot parastu mīklu pīrāgam.
- ✓ Pievienot divas karotes cukura
- ✓ Ielikt krāsnī uz 10 minūtēm
- ✓ Izņemt izcepto pīrāgu.

Kas ir varbūtisks algoritms ?

Varbūtisks algoritms – tas ir tāds algoritms, kurš dod uzdevuma atrisinājumu ar kaut kādu varbūtību. (Varbūtība ir atkarīga no uzdevuma un algoritma), tāpēc varbūtisks algoritms var būt vieglāk/ātrāk neka parasts algoritms, kurš ir paredzēts tāda paša uzdevuma atrisināšanai. Patiešām tiešas atkarības nav. Viss ir atkarīgs no algoritma realizācijas.

Varbūtisko algoritmu saknes noved pie Monte Carlo metodēm, kuras tiek izmantotas skaitļu analīzei, statistiskā fizikā un simulācijā. Sarežģītības teorijā varbūtiskās Tjuringa mašīnas saprašana bija piedāvāta de Leeuwet'am 1955. gadā [1] un bija vairāk sīkāk izstrādāta Rabin 1963. gadā [2] un Gill 1977. gadā [3] darbos. Pašie agrakie varbūtisko algoritmu piemēri bija parādīti Berlekamp 1970. gadā [4] , Rabin 1976 [5] gadā, un Solovay and Strassen 1977. gadā [6] darbos. Rabin 1976. gadā [5] izmantojot izrēķināšanas ģeometrijas uzdevumu piemēros un skaitļu teoriju noteikti parādā, ka varbūtības izmantošana ir labs instruments. Piemēram, tanī pašā laikā Solovay un Strassen 1977. gadā [6] parādīja varbūtisko algoritmu, lai pārbadītu skaitļu vienkāršību (Vai skaitlis ir pirmskaitlis vai nav)

Pēc tam tika izstrādāta, iespaidīga kopa ar tehnikām, lai sastādītu un analizētu varbūtiskos algoritmus. Var vērsties pie Karp 1991. gadā [7], Maffioli 1985. gadā [8] un Welsh 1983. gadā [9], kuru darbi ir veltīti varbūtisko algoritmu izpētei.

Kopā ar „parastām" darbībām varbūtiskā algoritmā ir atļauta „monētas mešana", tas ir nejaušu vienmērīgi sadalītu bitu izmantošana. Varbūtības algoritmu rezultāts nav noteikti zināms. Var tikai runāt par tāda vai cita rezultātā varbūtību. Algoritms tiek pieņemts par labu, ja kļūdas varbūtībā nav pārāk lielas. Tāds izplūdis apraksts, patiešām, prasa noteiktību. No praktiskā viedokļa ir interesanti tikai tie algoritmi, kuri kļūdās ļoti reti. Daudzos gadījumos, salīdzinot viegli samazināt varbūtības algoritma kļūdu. Lai samazinātu kļūdu varbūtību, algoritms tiek palaists vairākas reizes un tiek izvēlēta tāda atbilde, kura parādās vairāk par visām. Kļūdas

varbūtība tiek diezgan ātri novērsta. Piemēram, ja algoritma kļūdu varbūtība ir vienāda ar 0,4 (tiešām daudz ?), tad pie to darbu 30 reižu atkārtojumiem tā samazināsies apmēram divas reizes. Izpildot algoritmu tūkstoš reižu, mēs dabūjam kļūdu varbūtību vienādu ar 10^{-9}; divu tūkstošu atkārtojumu dos 10^{-18} , kas ir pilnīgi pietiekoši no praktiskā skatu viedokļa. Tāda procedūra tiek saukta par varbūtības pastiprināšanu.

Kāpēc vajadzīgi varbūtiskie algoritmi (Kāpēc pietrūkst parastie algoritmi) ?

Parastā gadījumā, bieži ir, ka parasti algoritmi atrisina uzdevumu ļoti ilgi(Sevišķi ilgi), un lai būtu ātrāks un vieglāks algoritms, izmanto varbūtiskos algoritmus šo uzdevuma atrisināšanai. Bieži varbūtisks algoritms prasa pārāk mazāk izrēķināšanas nekā parasts(determinets) algoritms šo uzdevumu atrisināšanai. Šad tad notiek, ka ar parastiem algoritmiem uzdevumu nevar atrisināt, bet var mēģināt risināt ar varbūtiskiem algoritmiem.

Eksistē dažādu uzdevumu klases, kurus pieņemts risināt ar varbūtiskiem algoritmiem. Pats pazīstamākais un svarīgākais, no praktiskā viedokļa, efektīvo varbūtisko algoritmu piemērs ir skaitļa pārbaude vai ir pirmskaitlis vai nav ? Atcerēsimies, ka naturālskaitlis tiek saukts par vienkāršu, ja tas dalās bez atlīkuma tikai ar vieninieku un pats ar sevi.
Piemēram, 3,5,7,13 – pirmskaitļi, bet 9, 14 – nav. Ja skaitļi ir nelieli, tad šo pārbaudi ir viegli izdarīt, pārbaudot dalīšanu az visiem skaitļiem, kuri ir mazāki par doto skaitļi. Bet ko darīt, ja jāpārbauda, piemēram, sekojošu skaitļi
170141183460469931731687976785678712430987654683265886543097532134 1 ? Pie tam atbildi var atrast dažās sekundēs, izmantojot personālo datoru. (šo skaitļu ievads no tastūras aizņem ilgāku laiku) Skaitļis ir salikts, kaut arī algoritms neko nerunā, par to, kādi ir to dalītāji. (Izmantojot netiešas pirmskaitļa pazīmes) Pārbaudes algoritms (vai skaitlis ir pirmskaitlis vai nav) ir iekārtots tā ka, atbilde „skaitlis nav pirmskaitlis" algoritms dod pārliecību (Kļūdas varbūtība ir 0). Bet ja algoritma atbilde - „skaitlis ir pirmskaitlis", tad eksistē daža kļūdas varbūtība, kuru var samazināt līdz pietiekami mazai, izmantojot iepriekš uzrakstīto varbūtības pastiprinājumu. (Šādu veidu algoritmus sauc par Las Vegas algoritmiem, pretēji Monte Karlo algoritmiem, kuri pieļauj kļūdas abos atbilžu variantus. Šāda terminoloģija nepiedomā tehnoloģijas atšķirību spēļu namos vecā un jaunā pasaulē. Šī terminoloģija ieviesās un iedzīvojās pavisam nejauši.)

Pierādījums, ka jebkuru uzdevumu, kuru var atrisināt ar parastiem algoritmiem, var ari risināt ar varbūtiskiem algoritmiem

Lai **Q** – galīgā kopa ar iespējamiem ieejas datiem (Q1,Q2,Q3…n, kur Q ir kaut kāda datu struktūra)
A – galīgā kopa ar instrukcijām (operāciju kopa, kuru var saprast izpildītājs, faktiski tā ir kopa ar dažādam funkcijām ar dažādiem mainīgiem)
Z – galīgā kopa ar rezultātu datiem(Visas iespējamās variantu funkcijas A pielietošana pie datiem Q, A1(Q1), A1(Q2), A2(A1(Q1)) u.t.t)

Tādā veidā mēs varam uzdot kaut kādu algoritmu, izvēloties elementus no kopas Q,A Piemēram, algoritms 1 : A3(A2(A1(Q1))) : Darbība A1 pie datiem Q1, pēc tam darbība A2 uz iegūtu rezultātu u.t.t Jebkuram uzrakstītam algoritmam mēs varam izvēlēties nejaušus elementus no kopas A. Šos elementus saglabāsim jaunā kopā A(prim).

Izveidosim jaunu algoritmu, izvēloties elementus no kopas Q un A(prim)
Jaunais algoritms risinās tādu pašu uzdevumu un būs varbūtisks. Ja mēs izvēlēsimies pilnīgi nejauši (un pie tam neveiksmīgi), tad algoritma precizitāte būs apmēram vienāda ar 0. Šajā gadījumā nejaušam vērotajam var likties, ka algoritms risina pavisam citu uzdevumu. Ja mēs izvēlēsimies elementus no kopas A gudrāk, tad algoritma precizitātē tuvosies 1.

Varbūtības algoritmu piemēri.

Varbūtības algoritms vienkāršam uzdevumam
Uzdevums.

Vienādojums vispārīgā veidā.

A + X = B (Kur, A,B э N{1..10})

Algoritms atrisinājumam

Dot atbilde “42”

Algoritms grūtam uzdevumam.

Uzdevums

Izgatavot ābolu pīrāgu

Izpildītājs – jebkurš cilvēks

Algoritms

- ✓ Atrast 1 ābolu.
- ✓ Iejaukt kaut ko nejaušā veidā
- ✓ Ielikt krāsnī uz 2 minūtēm

Algoritms lai pārbaudīt vai ir skaitlis pirmskaitlis vai nav

[10]

Ievads : skaitlis n .

Solis 1. Pārbaudam n pārību. Ja $n = 2$, tad atbilde "n — pirmskaitlis", Ja n— pāra skaitlis un ir lielaks par 2, tad atbilde "n — salikts skaitlis", pretējā gadījumā jāiet pie soli 2.

Solis 2. Pārbaudam, vai var izņemt no n veselo sakni k -j pakapē pie $k = 2, \ldots, \lfloor \log_2 n \rfloor$. Ja var, tad atbilde "n — salikts skaitlis", pretējā gadījumā jāiet pie Soli 3.

Solis 3. Rakstam $n - 1$ veidā $2^k \cdot l$, kur $k > 0$, un l — nepāra skaitlis.

Solis 4. Izvēlēsimies nejaušo a starp skaitliem no 1 līdz n .

Solis 5. Izrēķinām $a^l,\ a^{2l},\ \ldots,\ a^{n-1}$ pēc moduli n .

Pārbaude 1. Ja $a^{n-1} \not\equiv 1 \pmod{n}$, tad atbilde "n — salikts skaitlis".

Pārbaude 2. Ja var atrast tādu j , kuram $a^{2^j l} \not\equiv \pm 1 \pmod{n}$, un $a^{2^{j+1} l} \equiv 1 \pmod{n}$, tad atbilde "$n$ — salikts skaitlis".

Pretējā gadījumā atbilde "n — pirmskaitlis".

Algoritmu analīze.

Lemma Ja n — pirmskaitlis, tad aprakstīts agrāk algoritms vienmer (ar varbūtību 1) dot atbildi "n — pirmskaitlis".

Ja n — salikts skaitlis, tad atbilde "n — pirmskaitlis" būs ar varbūtību $\geqslant 1/2$.

Lemma pierādījumu var paņemt no [10]

Algoritmu formālas pazīmes

Dažas algoritmu definēšanas tiešā vai netiešā formā satur kopīgas prasības.

1. Izpildījums (Galīgums). Pie pareiziem dotiem izejošiem datiem algoritmam jāpabeidz darbs un jādot rezultāta izmantojot galīgu soļu skaitu. No otras puses varbūtisks algoritms var nedot rezultātu nekad, bet to varbūtība ir vienāda ar 0.
2. Determinisms (Noteiktība) Katrā laika momentā nākošais darba solis tiek viennozīmīgi nodefinēts ar sistēmas stāvokli. Tādā veidā algoritms izdot vienu un to pašu rezultātu (atbilde) priekš vieniem un tiem pašiem izejošiem datiem. Mūsdienu traktēšanas sistēmā vienam algoritmam pie dažādas realizācijas jābūt vienam izomorfiskam grafam. No otras puses eksistē varbūtiski algoritmi, kura nākošais darba solis ir atkarīgs no tekošās sistēmas stāvokļa un nejauša skaitļa, kurš tiek noģenerēts. Bet, kad nejauša skaitļa ģenerēšanas metodes ir iekļauta izejošā datu sarakstā, varbūtisks algoritms kļuvis parasta algoritma veids.
3. Ievads. Algoritmam vajadzīgi dažādi ievada dati.
4. Rezultāts(Izvads) – algoritms beidzas ar noteiktiem rezultātiem.
5. Saprotamība. Algoritms izpildītājam vajag iekļaut tikai tās komandas, kuras izpildītājam pieejamas un kuras ir iekļautas izpildītāja sistēmu komandā.
6. Masveidīgs. Algoritmam jābūt tādam, lai to būtu iespējams izlietot pie dažādiem izejošiem datiem.

Kā noteikt, kurš algoritms labāks – parasts vai varbūtisks ?

Katram uzdevumam to nosaka subjektīvi. Var tikai pateikt, ka eksistē uzdevumu klase, kuriem varbūtisks algoritms principā nav pieņemams.
Šie ir uzdevumu, kuriem vajag 100% pareizības – kur kļūda var novest, piemēram, pie cilvēku nāves, tāpēc to nepieļauj.

Tā arī ir uzdevumu klase, kuru ar parastiem algoritmiem nevar atrisināt, to var mēģināt darīt ar varbūtiskiem algoritmiem.
Tāpēc abu algoritmu veidi eksistē un tiek lietoti.

Vai var visus uzdevumus(kurus var atrisināt) risināt tikai ar parastiem algoritmiem ?

Viss izskatās ļoti loģiski. Kāpēc jādabū varbūtisko atbilde, kad var visu risināt ar parastiem algoritmiem. Bet īstenībā eksistē ļoti eleganti varbūtiskie algoritmi, kuri risina šo uzdevumu daudz ātrāk, darot mazāk izrēķināšanas.
Pie tam šo algoritmu vieglāk realizēt. Parādīt šādu algoritmu eksistenci dažiem zināmiem objektiem matemātikā, ir šo darbu mērķis.

Kā pamatuzdevums ir izvēlēts identitāšu pārbaude. Kāpēc šis uzdevums, bet ne cits ? Pēc autora subjektīva uzskata vairāk elegantus algoritmus var atrast tieši šeit.

Tas, kas ir jau izdarīts par šo tēmu(Varbūtiski algoritmi identitāšu pārbaudei) pirms mana bakalaura darba

Algoritms

Neskatoties uz to, ka mūsdienu algoritmu formāla definēšana tika izveidota 30 – 50 gados, XX gadsimtā Tjuringa, Posta, Čerča (Čerča – Tjuringa tezes), N.Vinea, A.A.Markova darbos, paša algoritma ideja bija zināma jau senajā Grieķijā.

Daudzi zinātnieki izmantoja algoritmu, ka pierasta formu, lai risinātu uzdevumus.

Viens no pašiem pirmajiem un visslavenākiem piemēriem ir zinātnieka Abu Abdullah Muhammed Ibn Musa aļ-Horesmi darbs. Apmēram 825. gadā viņš uzrakstīja sacerējumu, kurā pirmo reizi deva aprakstu desmitnieku pozicionālas izrēķināšanas sistēmai, kura tika izgudrota Indijā. Tulkotājs, kura vārdu līdz mums nenonāca, deva tai nosaukumu „Algoritmi de numero Indorum" („Algoritmi par Indijas rēķināšanu"). Arābu valodā grāmata saucas „Kitab aļ-džebr vaļ-mukabala" („ Grāmata par saskaitīšanu un atņemšanu ").

Tāpat gribas dot references par E.D. Knuta darbiem [11], kurš aprakstīja daudz meklēšanas un kārtošanas algoritmus dažādām struktūrām. Viņa idejas veiksmīgi tiek izmantotas arī šodien.

Varbūtisks algoritms

Neskatoties uz to, ka varbūtības algoritmu definēšana tika izveidota nesen, mūsdienās eksistē milzīga kopa ar dažādiem varbūtiskiem algoritmiem, kuri tiek izmantoti dažādos pieļietojamos uzdevumos.

Balstoties uz Karp 1991. gadā [7] darba var izdalīt galvenos uzdevumus, kuriem risinājuma metode ar varbūtisko algoritmu palīdzību ir acīm redzama.

Varbūtisks algoritms lai pārbaudītu identitāti

Speciāla uzdevumu klase, kura ir konkrēta uzdevumu piemērs, kurus var risināt ar varbūtiskiem algoritmiem.

Atsevišķs darbs, kurš ir veltīts dotai tēmai, internetā vai pieejamā zinātniskā literatūrā, es neatradu.

Bet eksistē nedaudz darbu, kuros ir aprakstīti līdzīgi algoritmi. Algoritmi ir aprakstīti, izanalizēti, testēti un daudz kur lietoti, lai risinātu tos uzdevumus, kuros tie ir paredzēti. Vispirms ir jāsaprot, algoritmu motīvus izveidošana. Tie, manuprāt, ir acīm redzami. Izveidot algoritmu, kurš strādās ātrāk, vienkāršāk u.t.t. Kaut kādam objektam eksistējošas atrisinājuma metodes, ar determinētu algoritmu palīdzību darbojas pārāk ilgi vai arī parasti ir iespējams izmantot varbūtisko algoritmu, kurš strādās ātrāk un tā precizitāte būs vismaz lielāka par ½. (Bet praktiskiem uzdevumiem būtu labi, ja precizitāte tuvotos 1)

Freivalds, R 1979. gadā [12] darbā bija aprakstīts līdzīgs varbūtisks algoritms, lai pārbaudītu matrices reizinājuma identitāti. (Pārbaudit identitāti A * B = C, kur A,B,C ir matrices veida n * n) Ideja ir tāda, ka var noģenerēt stabiņu *x*, atrast ABx (Q`) un atrast Cx (W`) tālāk salīdzināt Q` un W`. Algoritms dara mazāk izrēķināšanas operācijas, tādēļ ir vienkāršāk un ātrāk, bet to precizitāte ir vienāda apmēram 99,9%. Acīm redzams, ka šis algoritms ir labāks nekā tā determinēts analogs(Sareizināt matricas A un B un pa elementiem salīdzināt ar C). Arī var atsaucies uz Freivalds, R 1977. gadā [13] darbu. Tāpat Blum M. (http://www.cs.cmu.edu/~mblum/research/) savos darbos, kuri veltīti programmas vertifikācījām ir iekļauti līdzīgi varbūtiski algoritmi.

Tajā skaitā arī par šiem darbiem Blum 1995. gadā saņema Tjuringa prēmiju. [„Darbi par izrēķināšanas sarežģītības teorijas pamatiem, un tās pielietošana pie kriptogrāfijas un programmas vertifikācijām ”]. Viņa izstrādāta teorija kopā ar algoritmiem tiek veiksmīgi lietota mūsdienās.

Šodien, intensīvas internēta attīstības laikā, tīklā ir viegli atrast jau izveidotus algoritmus gandrīz jebkuriem mērķiem.

Sava darbā laikā, es arī izmantoju dažus līdzīgus algoritmus. Viens no uzdevumiem bija pārbaudīt programmas veselumu pirms tā darba sākuma. Programma pārbauda savu izpildošo (.exe) failu ar failu, kurš glabājas serverī. Failu salīdzināšana tīklā ar determinēto algoritmu palīdzību (pa bitiem) darbojas pārāk ilgi, tāpēc bija jāizmanto varbūtisko algoritmu, lai pārbaudītu identitāti. Algoritma ideja ir tāda, ka jāsalīdzina ne paši faili, bet funkcijas no tiem. (Piemēram, jaucējfunkciju md5) Šī metode nav manis izdomāta, es paņēmu algoritma ideju no viena internēta foruma. [14]

Tāpat man bija uzdevums, kuru nebija iespējams izrisināt, neizmantojot varbūtisku algoritmu. Tā bija datubāzes atjaunošanas uzdevums, pie tam kā visas nepieciešamas informācijās pietrūka.

Viena vidēja izmēra firma Rigā, kura nodarbojās ar vairumu un mazumtirdzniecības tirgošanos, janvārī 2007. gadā pazaudēja visu informāciju vairāk nekā par vienu gadu.

Atjaunošanas grupas rīcībā bija stingras atskaites blankas (pavadzīmes) par šo periodu un kases grāmatas ieraksti (z - atskaites) Ir saprotams, ka visu līdzekļu ieņēmumi un 80% pārdošanas (Tiešam tāda attiecība bija starp pārdošanu vairumā un mazumā šajā firma) var atjaunot no pavadzīmēm, kuras satur visu nepieciešamo informāciju. (Preces kodu, daudzumu un cenu). Bet kas var izdarīt ar pārdošanu mazumā ? Kases lentas kopijas nesatur kodus. Preces nosaukumi, kuri ir uz čeka ir saīsināti, visai tuvināti. Cenas uz preces mainās. Bez tam kases lenta, atšķirībā

no pavadzīmēm, kuras bez grūtībām var saskanēt un ciparot, ir ļoti neērta atjaunošanas mērķiem. Tiek pieņemts lēmums pieskaņot preces nejauši, izejot no nodefinētiem kritērijiem. Ir zināmi atlikumi uz doto momentu, vidējais uzcenojums un tekošās preces cena. Uzdevums bijā vēlāk parādīt mēneša realizāciju (kura ir tikai 20% mazuma) un, protams, dabūt kopsavilkumu atlikumu.

Ir grūti novērtēt šo algoritmu precizitāti. Preces pieprasījums ir proporcionāls tā atlikumam tikai pie ideālās piegādes darba. Bet datubāze pēc atjaunošanas neizsauc ne mazāko neuzticību visu līmeņu kontrolējušiem orgāniem un ekspertiem

Es savā bakalaura darbā parādīšu no dažādiem avotiem varbūtisko algoritmu identitāšu pārbaudei dažādu objektu matemātikā piemērus. Es noprogrammēšu šos algoritmus. Es izmantošu jau izdarīto uzdevumu analīzes un pacentīšos parādīt, ka varbūtiskie algoritmi šo uzdevumu klases atrisināšanai ir labāki nekā determinētie. Tāpat es pacentīšos izveidot savu personīgo algoritmu.

Uzdevumu reķināšana

Tādēļ, lai redzami nodemonstrētu varbūtiskos algoritmu pārākumu par parastajiem, šo uzdevumu klases risināšanai ir vajadzīgi izvēlēties dažādus objektus.
Varbūtisko algoritmu izvēle ir subjektīva, galvenais ir ,lai tas stipri nezaudētu precizitāti un uzvarētu ātrumā.

Es izvēlējos tādus uzdevumus.

1) Matricas nevienādība.
2) Virknes salīdzināšana.
3) Nulles polinoms.

Ka salīdzina algoritmus (Varbūtisks un parasts algoritms)

Ir dots uzdevums pārbaudīt identitāti. Es parādīšu varbūtiskos algoritmus šo uzdevumu risināšanā daudzos objektos piemērā. Lai būtu ar ko salīdzināt uzdevumu priekš tiem objektiem, tā ir jārisina ar parastiem algoritmiem. Lai nebūtu tikai teorētiski paskaidrojumi (Algoritmu darbības ātrumi ir apmērām zināmi), es tos realizēšu izmantojot Delphi valodu Delphi 7 vidē.

Sastādīšu testu piemērus un analizēšu algoritmu darbības rezultātus. Katra algoritma darba rezultātus tiek saglabāti norādītā izejošā failā.

Par pamatiem kritērijiem ņemšu tādus kā precizitāti, ātrumu un subjektīvo kritēriju „Realizācijas sarežģītība".

Pēc izdarītiem rezultātu analīzēm var izdarīt secinājumus.
Programēšanas valodas un vides izvēle nav tik kritiska šajā gadījumā, ir vēlams tikai, lai visi algoritmi tiek realizēta vienā programēšanas valodā.
Es izvelējos Delphi 7 tāpēc ka šī vide man patīk vairāk par citām.
Ideju un saprotamu realizāciju, piemēram, izmantojot pseido-kodu, katram algoritmam, es arī iekļāvu savā darbā.

Par cik parasta algoritma darbības precizitāte ir vienāda ar 100%, problēmas ar varbūtiska algoritma precizitātes novērtēšanas nebūs. Algoritmi strādās ar vienādi dotiem testiem.

Kā es novērtēšu, kurš algoritms ir labāks ?

Es novērtēšu testa rezultātus. Acīm redzams, ka varbūtiska algoritma precizitāte būs mazāka par A% [0..100%], nekā parastam algoritmam.
Bet ir iespējams, ka citi faktori izrādīsies labāki, un varēs izdarīt secinājumu, ka šinī gadījumā varbūtisko algoritmu izmantošana ir pilnīgi pieejama un no tas secina, ka tas ir labs algoritms.

Matricu sareizināšana.

Pieņemsim ,ka jāpārbauda matricas vienādība A * B = C, kurā matricas A,B,C ir kvadrātiski izmērā n * n. Lai pārbaudītu tieši ar parastu matricas A,B sareizināšanu un pēc tam salīdzinot to ar matricas C.

Determinets algoritms

Ir nepieciešams izpildīt $O(n^3)$ operācijas.
Atgadinam ka matricas reizina sekojoši

$$AB = \begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix} \times \begin{pmatrix} -1 & 2 \\ -2 & 3 \end{pmatrix} = \begin{pmatrix} 2 \cdot (-1) + 3 \cdot (-2) & 2 \cdot 2 + 3 \cdot 3 \\ 5 \cdot (-1) + 7 \cdot (-2) & 5 \cdot 2 + 7 \cdot 3 \end{pmatrix} = \begin{pmatrix} -8 & 13 \\ -19 & 31 \end{pmatrix}$$

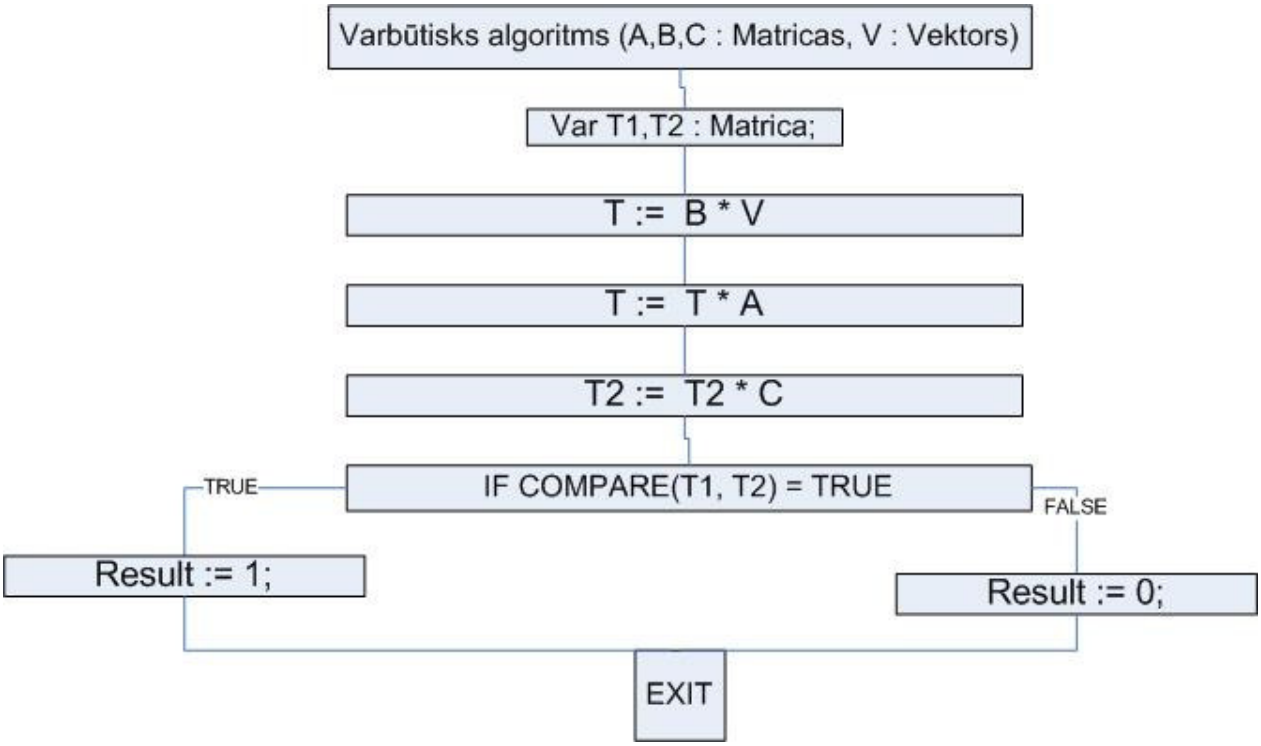
Varbūtisks algoritms
[11]

Izvēlēsimies vektoru $Y=(y_1, ..., y_n)$, kuram koordinātes ir vienādas ar 0 vai 1, pie tam 0 vai 1 parādīšanās varbūtība ir stipri vienāda. Salīdzināsim A * (B * Y) un C * Y, ja vienādības nav, tad sakuma vienādība nav izpildīta. Ja $A \cdot (B \cdot Y) = C \cdot Y$, tad varbūtība ,ka $A \cdot B <> C$ *ir* mazāka par ½. Tādā gadījumā mēs varam pārbaudīt vienādību ar nepieciešamo varbūtību.

Kā ieejas dati algoritms saņem matricas A,B,C un varbūtību ar kuru ir nepieciešams mums pārliecināties par vienādību, Uz izeju saņemam True vai False (Patiess, vai aplams),atbilstīgi, ja vienādība izpildīta ar uzdotu varbūtību vai ne,

Par tik, par cik visas operācijas datorā tiek izpildītas ar kaut kādu absolūtu kļūdu, algoritms pieņem mazo skaitļi Epsilon, kurš raksturo pieņemamo atšķirību lielumu, pie kura visi skaitļi tomēr tiek pieņemti kā vienādi. Ja vienādības laba puse atšķiras no kreisās puses ar skaitli, kurš ir mazāks neka Epsilon, tad skaitļi tiek pieņemti kā vienādi.

Funckcija **COMPARE** – salidzina divus matricas
Result – globalais mainigais



2.1. att. **Varbūtiska algoritma Blokshēma, kura izveidotā programmā VISIO**

Lemma

Algoritmam vb_2 ir darba laiks vienāds ar $\mathbf{O(n^2)}$ un kļūdas varbūtību ne lielāku par ½. Pie tam algoritms var kļūdīties tikai, ja ir $\mathbf{A \times B \neq C}$

Pierādījums

Pieņemsim, ka $\mathbf{A} \times \mathbf{B} \neq \mathbf{C}$

Algoritms, kļūdās, ja nejaušam vektoram, kuru tas izvēlas

ir $(\mathbf{A} \times \mathbf{B} - \mathbf{C}) \times x = 0$

Apskatīsim nulles virkni $d = (d_1,...,d_n)$ matricu $\mathbf{A} \times \mathbf{B} - \mathbf{C}$ (tāda ir, par cik $\mathbf{A} \times \mathbf{B} \neq \mathbf{C}$)

$d_1 \neq 0$

Šis virknes sareizināšana ar x =

$$\sum_{i=1}^{n} d_i x_i$$

Šis reizinājums ir vienāds ar 0, tad un tikai tad, kad $x_1 = h(x_2,...,x_n)$, kur

$$h(x_2, \ldots, x_n) = -\frac{\sum_{i=2}^{n} d_i x_i}{d_1}$$

Atzīmēsim, ka, **h** var pieņemt vērtību, kura atšķiras no **0** un **1**.

Ar varbūtību vismaz **½** $\mathbf{x}_1 \neq \mathbf{h}(\mathbf{x}_2,...,\mathbf{x}_n)$,

Matricu salīdzināšanas realizācija

Testu datu ģenerēšana

- Ciklā no 4 līdz 220 tiek noģenerētas kvadrātu matricas A un B. (Matricas 4x4,5x5 u.t.t)
- Matrica A un Matrica B tiek aizpildītas ar nejaušiem skaitļiem no 1 līdz 10000
- Ja n (Matricas veids ir n x n) dalās ar 2 tad Matrica C tiek ģenerēta ka matricas A un B reizinājums, tā lai nevienādība (A * B = C) tika izpildīta, ja n nedalās ar 2 ,tad matrica C tiek ģenerēta nejauši.
- Tiek noģenerēts nejaušs vektors V (n x 1) sastāvošs no 1 vai 0.

(Agrāk uzrakstītas operācijas tiks izpildītas 100 reizes. (100 testi))

Algoritmu laiku un precizitātes izskaitļošana

Determinēts algoritms

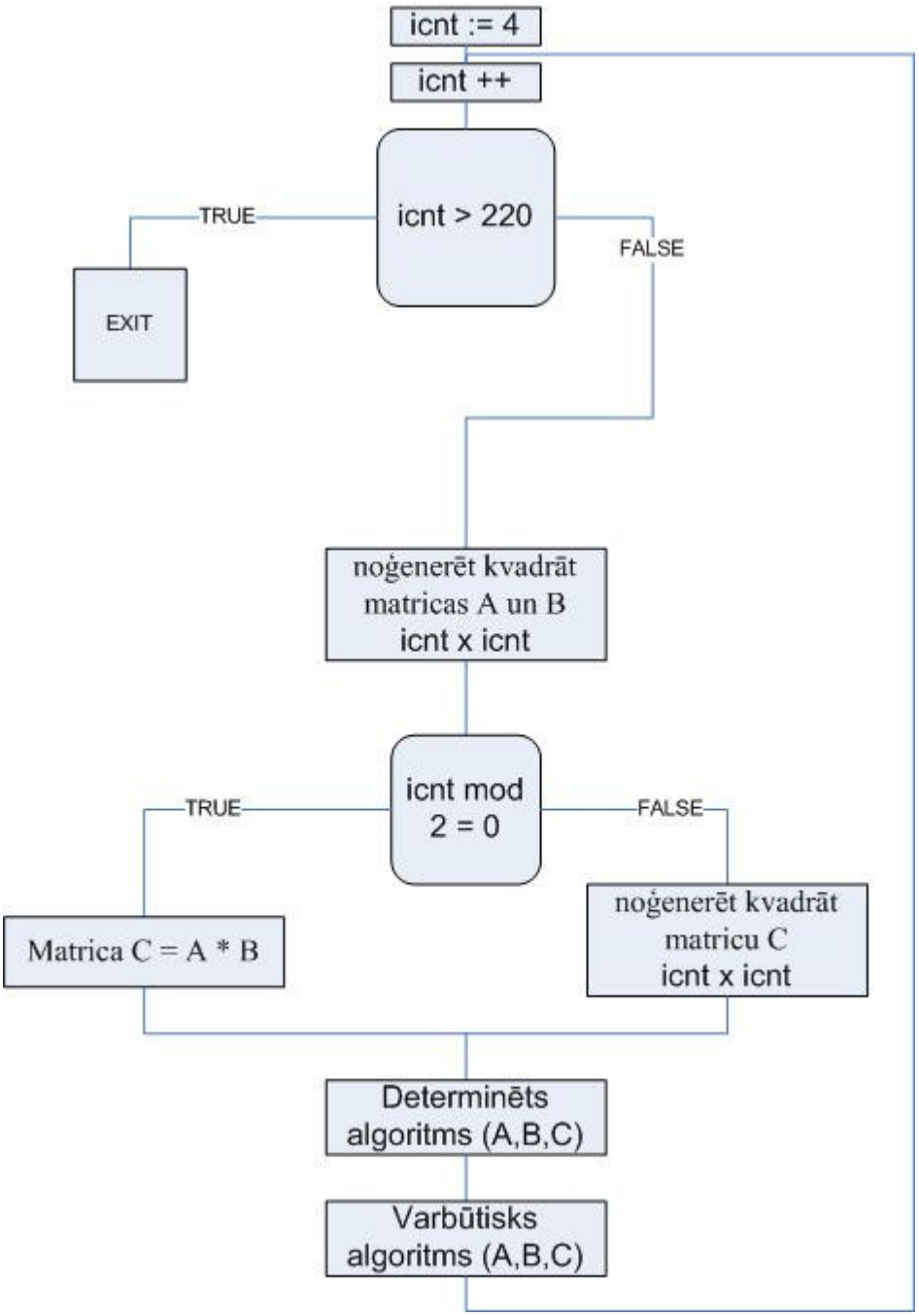
- Tiek noņemts esošais laiks : StartTick
- IF A * B = C
- THEN answer := 1 ELSE answer := 0;
- Tiek noņemts esošais laiks : EndTick
- Ierakstīt failā Output1.txt EndTick – StartTick
- Ierakstīt failā Output3.txt answer

Varbūtisks algoritms.

- Tiek noņemts esošais laiks : StartTick
- IF A * (B * V) = C * V
- THEN answer := 1 ELSE answer := 0;
- Tiek noņemts esošais laiks : EndTick
- Ierakstīt failā Output2.txt EndTick – StartTick
- Ierakstīt failā Output4.txt answer

Analīzes

- Output1.txt,Output2.txt,Otput3.txt,Output4.txt ielādēts uz Excel.
- Procentu izrēķināšana, uz kuru Output4.txt atšķiras no Output2.txt (Determinēts algoritms vienmēr strādā pareizi.)
- Parametru izskaitļošana : determinēta algoritma vidējais ātrums, varbūtiska algoritma vidējais ātrums, % varbūtiskā algoritma pareizība.
- Uzskates grafiku uzbūve.



2.2. att. **Matricas programma blokshēma, kura izveidotā programmā VISIO**

Programmas kods

```
for icnt := 4 to 220
 do begin
  //Random
   TRandMatrixA := CreateSquareMatrix(icnt); //Create Square Matrix where n = i
```

```
    TRandMatrixB := CreateSquareMatrix(icnt);
    FillMatrixAtRandom(TRandMatrixA,NumRand);
    FillMatrixAtRandom(TRandMatrixB,NumRand);
    TRandVector := CreateMatrix(icnt,1); //Create random Vector [n x 1]
    FillMatrixAtRandom(TRandVector,2); //Fill with 0..1

   if icnt mod 2 = 0 then  //Each Second C Matrix Apply
   begin
     TRandMatrixC := MultipleMatrixOnMatrix(TRandMatrixA,TRandMatrixB);
   end
   else
   begin //create Random Matric C no A * B
     TRandMatrixC := CreateSquareMatrix(icnt);
     FillMatrix(TRandMatrixC,NumRand);
   end;

  //Deterministic Algorithm
  StartTick := GetTickCount;
If  CompareMatrix(MultipleMatrixOnMatrix(TRandMatrixA,
TRandMatrixB),TRandMatrixC,icnt,icnt) then
  begin
   answer := 1;
  end
  else
   answer := 0;
  endTick := GetTickCount;
  writeln(DetFail,PChar(MSecToTime(StartTick - EndTick)));
  writeln(DetFailAtb,intToStr(answer));

  //Randomness Algorithm
  StartTick := GetTickCount;
  if CompareMatrix(MultipleMatrixOnMatrix(TRandMatrixA,MultipleMatrixOnMatrix(
```

```
    TRandMatrixB,TRandVector)),
MultipleMatrixOnMatrix(TRandMatrixC,TRandVector),icnt,1) then
  begin
   answer := 1;
  end
  else
   answer := 0;

  endTick := GetTickCount;
  writeln(vrbFail,PChar(MSecToTime(StartTick - EndTick)));
  writeln(vrbFailAtb,intToStr(answer));

 end; //End Generation Cicle
```

100 testu pa vienādu principu, rezultāti.

Determinēta algoritma kopējais darbības laiks
(Sekundes, Miļi sekundes)
124.840

Determinēta algoritma videjais darbības laiks
(Sekundes, Miļi sekundes)
0.575

Varbūtīska algoritma kopējais darbības laiks
(Sekundes, Miļi sekundes)
3.015

Varbūtīska algoritma videjais darbības laiks
(Sekundes, Miļi sekundes)
0.014

Varbūtiska algoritma precizitāte 100 vienādos testos ir 100%

Varbūtiska algoritma ātrums vidēji ir 98.36 reizes lielāks nekā determinētam algoritmam.

[*// ISNULL(a,b) ir ja mainīgam a nav vertības, (musu gadijumā < 1 ms) tad tai vertībai // tiek pieškirta b*

SUM(**for icnt** := 4 to 220

ISNULL(Detirminēta algoritma atrums [**icnt**],0.001) / ISNULL(Varbūtiska algoritma atrums [**icnt**],0.001)

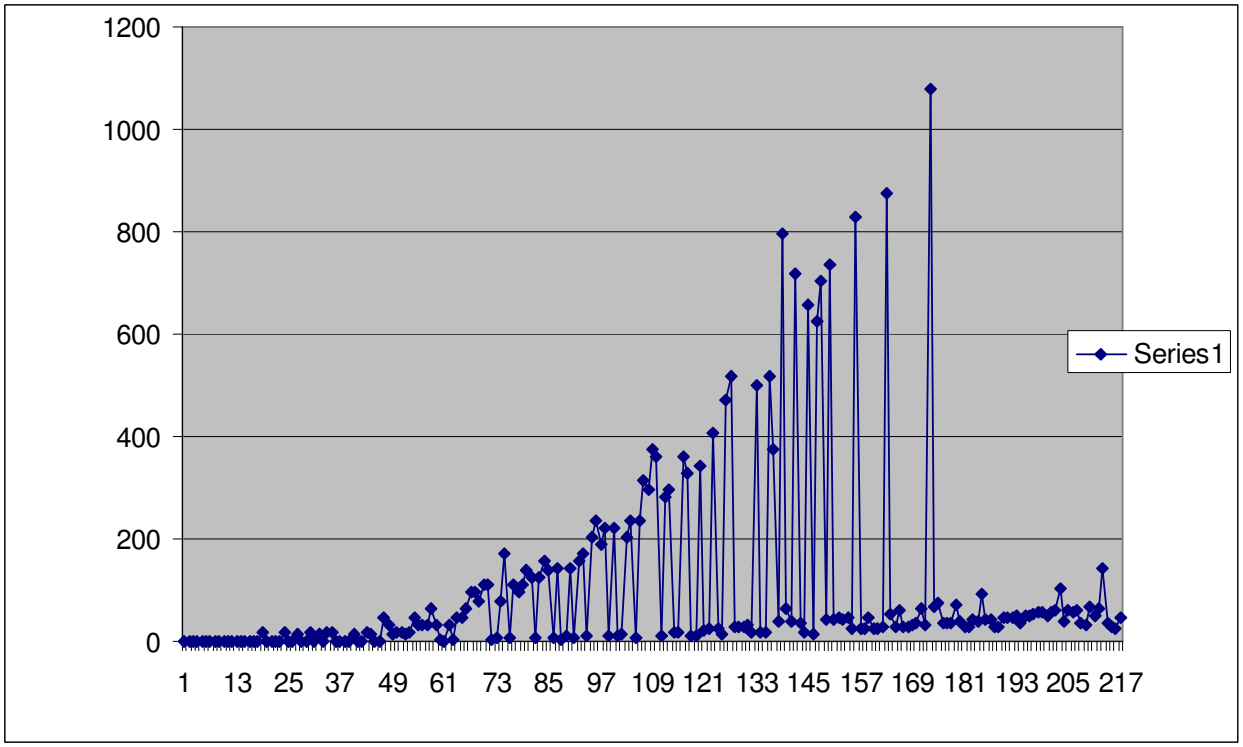
]

=> 21344.6727 / 217 = **98.36**

Grafiks

Pa X asi matricas n × n izmēri

Pa Y asi ir parādīts cik reizes varbūtisks algoritms strādā ātrāk neka determinēts.



2.3. att. **Matricas programmas rezultatu grafiks, kurš ir izveidots programmā Excel**

Rezultātu analīze

Tā kā bija paredzēts teorijā. Uz maza izmēra matricām algoritma darbības ātrums ir apmēram vienāds. Paaugstinot matriču izmērus, algoritma darbības ātrums palielinās.

Tuvojoties pie maksimālām X vērtībām (220), ātruma atšķirība vidēji palielinās daudz reizes. Grafikā precīzi redzama šī tendence.

Atsevišķus grafika lēcienus var izskaidrot ar to, ka visas matricas (A,B un dažreiz C) tika ģenerētas nejauši. Kāda no matricēm varbūt stipri sarežģītāka nekā citas. (pārsvarā ir lieli skaitļi šajā gadijumā varbūtisks algoritms stipri uzvares), bet cita var būt vienkāršāka par citām.(Piemēram, viens no elementiem ir 0. Varbūtība ir 1/10000 * n) Tas var stipri ietekmēt galīgos rezultātus. Tapāt jāatceras, ka katra matricas nevienadība ir aplama.

Tāpat operācijas sistēmas Windows procesu apstrādes dēļ, tomēr tā ir stipri neticami. (Izveidojusies kļūdas apstrādes situācija, cits process pieprasīja vairāk atmiņas un procesora laikā u.t.t)

Virknes salīdzināšana

Uzdevuma apraksts.

Ir dotas divas bitu virknes **A** un **B,** kuras ir nepieciešamas salīdzināt ar sakritību, izmantojot pēc iespējas mazāk informācijas(piemēram, vajadzīgi salīdzināt failus datortiklā)

Determinēts algoritms

Blok shēmas es neparādīšu, tāpēc ka, algoritms ir ļoti viegls.

Salīdzināšanas algoritms.

- Paņemt maksimālo iespējamo pirmos bitos skaitli no pirmās un otrās virknes
- Salīdzināt uz sakritību.
- Ja sakrīt tad ar tādu pašu piegājienu paņemt nākošos bitus no abām virknēm un atkārtot salīdzināšanu. Ja nesakrīt, tad dot atbildi „Nē". Ja virknēs vienādā pozīcijā ir virknes beigu simbols un salīdzināšana ir pozitīva, tad dod atbildi „jā".

Programēšanas paņēmienu izmantošana, lai uzlabotu determinēta algoritma ātruma darbību.

1. Nolasīt no faila ir labāk ne vienu baitu, bet uzreiz daudz, kaut kad, apmēram, 4 kilobaitos. (4096 Delphi) vinčesteram ka pirmais, tā otrais – viens apgrieziens.

2. Arī salīdzināt ir labāk ne pa batiem, bet ar masīviem, piemēram, ar API funcijas strncmp palīdzību.

Varbūtisks algoritms

Galvenā ideja : salīdzināt ne pašas virknes, bet funkcijas no tām.
Salīdzināsim **a mod p** un **b mod p**. Šādai salīdzināšanai pietiek nodot $\mathbf{2log_{(2)}P}$ bitus.

Matematiska izziņa

Asimptotisks pirmskaitļu sadalīšanas likums.

Definēšana

Pirmskaitļu sadalīšanas funkcija (*Prime distribution function*)

π(n) tiek definēts, ka pirmskaitļu daudzums, kuri nepārsniedz **n**.

Asimptotisks pirmskaitļu sadalīšanas likums.

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1.$$

Algorimts

- Lai **| a | = | b | = n, N = $n^2 \log_{(2)} n^2$.**
- Nejauši izvēlēsimies pirmskaitļi **p** no intervāla [**2**..**N**]
- Izdot «**jā**», ja **a mod p = b mod p**
- Izdot «**nē**»

Lemma

Algoritms vb_1 ir varbūtisks algoritms ar vienpusēju kļūdu ar kļūdas varbūtību **O(1/n)** un prasa nosūtīt **O(log n)** bitus.

Pierādījums

Nodoto bitu skaits.

$$2\log_2 p \le 2\log_2 N = 2\log_2(n^2 \log_2 n^2) = O(\log_2 n).$$

Algoritms kļūdas tikai, ja **a** ≠ **b**, bet izvēlētam pirmskaitlim **p** izpildīta vienādība **(a - b) ≡ 0 mod p**.

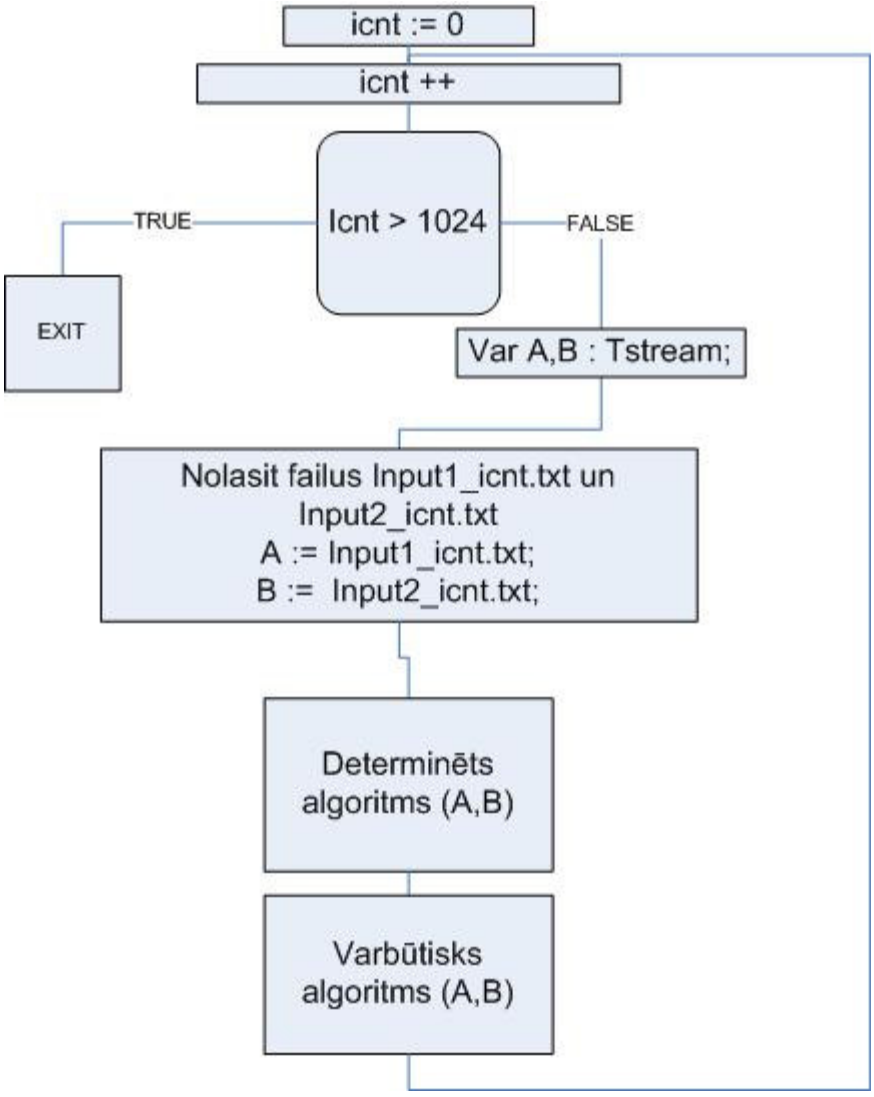
Tādu pirmskaitļu **p** skaits ir

$$|\{p \in \mathbb{P} : (a-b) \equiv 0 \mod p\}| \le n.$$

No tā seko, ka kļūdas varbūtība ir

$$O\left(\frac{n}{N/\ln N}\right) = O\left(\frac{n(\log n^2 + \log\log n^2)}{n^2 \log n^2}\right) = O\left(\frac{1}{n}\right)$$

Virknes salīdzināšana realizācija



2.4. att. **Virknes programmas blokshēma, kura ir izveidota programmā VISIO**

Testu datu ģenerēšana

- Ciklā no 1 līdz 1024 tiek nolasītas izveidotas faili (faili ir Unicode)
- Izveidoti 2048([1..1024] * 2) faili ar indeksiem 1 un 2 , lai salīdzināt. Katrs otrais pāris atšķiras ar vienu bitu patvaļīgā vietā.

Algoritmu laiku un precizitātes izskaitļošana

Determinēts algoritms

- Tiek izveidotas failu straumes (Stream). Tie pārveidoti kā bitu straumes.
- Tiek noņemts esošais laiks : StartTick.
- Bitu straumes ir salīdzinātas atmiņā.
- Ierakstīt failā Output1.txt EndTick – StartTick.
- Ierakstīt failā Output3.txt answer.

Varbūtisks algoritms.

- Tiek noņemts esošais laiks : StartTick.
- Tiek ģenerēts P.
- Abu bitu straume (Stream) tik paveidota ka skaitlis un ar to tiek izdarīta operācija mod.
- Tiek salīdzināti rezultāti.
- Ierakstīt failā Output2.txt EndTick – StartTick.
- Ierakstīt failā Output4.txt answer.

Analīzes

- Output1.txt,Output2.txt,Otput3.txt,Output4.txt ielādēts uz Excel.
- Procentu izrēķināšana, uz kuru Output4.txt atšķiras no Output2.txt (Determinēts algoritms vienmēr strādā pareizi.)
- Parametru izskaitļošana : determinēta algoritma vidējais ātrums, varbūtiska algoritma vidējais ātrums, % varbūtiskā algoritma pareizība.
- Uzskates grafiku uzbūve.

Programmas kods

```
for icnt := 1 to 1024 //1 MG
  do
  begin //Read Randomly Generated Files
    File1 := TFileStream.Create(Pchar('input1_' + intToStr(icnt) + '.txt'),fmOpenRead);
```

```
    File2 := TFileStream.Create(Pchar('input2_' + intToStr(icnt) + '.txt'),fmOpenRead);
    f1 := ConvertToBinary(file1,EncodingUnicode);
    f2 := ConvertToBinary(file2,EncodingUnicode);

    StartTick := GetTickCount;
    if CompareBuffer(f1,f2)
    then
    begin
     answer := 1;
    end
    else
     answer := 0;
    endTick := GetTickCount;
    writeln(DetFail,PChar(MSecToTime(StartTick - EndTick)));
    writeln(DetFailAtb,intToStr(answer));
    StartTick := GetTickCount;
    P := GetP(f1,f2);
    if ((ConvertToNumber(f1) mod p) = (ConvertToNumber(f2) mod p))
    then
    begin
     answer := 1;
    end
    else
     answer := 0;

    endTick := GetTickCount;
    writeln(VrbFail,PChar(MSecToTime(StartTick - EndTick)));
    writeln(VrbFailAtb,intToStr(answer));
end;
```

Testu rezultāti.

Determinēta algoritma kopējais darbības laiks

(Sekundes, Miļi sekundes)

3.07

Determinēta algoritma videjais darbības laiks

(Sekundes, Miļi sekundes)

0.002

Varbūtīska algoritma kopējais darbības laiks

(Sekundes, Miļi sekundes)

0.80

Varbūtīska algoritma videjais darbības laiks

(Sekundes, Miļi sekundes)

< 1 ms

Varbūtiska algoritma precizitāte ir 100%

Varbūtiska algoritma ātrums vidēji 4.06 reizes lielāks nekā determinētam algoritmam.

[*// ISNULL(a,b) ir ja mainīgam a nav vertības, (musu gadijumā < 1 ms) tad tai vertībai*
// tiek pieškirta b

SUM(**for icnt** := 1 to 1024

ISNULL(Detirminēta algoritma atrums [**icnt**],0.001) / ISNULL(Varbūtiska algoritma atrums [**icnt**],0.001)

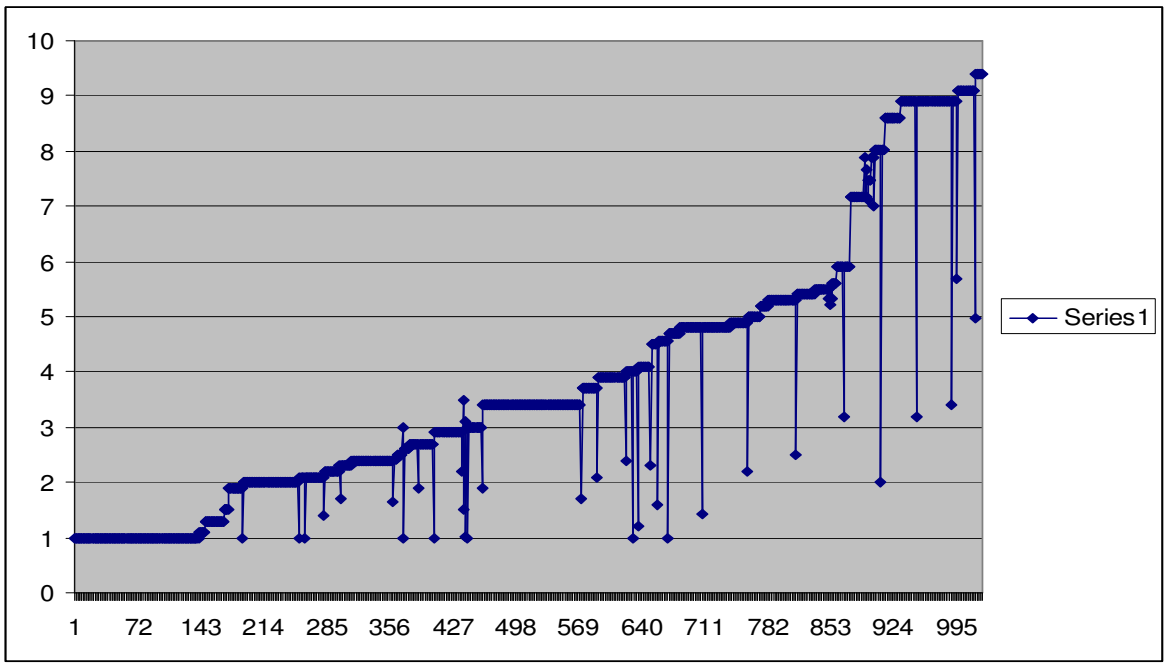
]

=> 4153.38 / 1023 = **4.06**

Grafiks

Pa X asi faila izmēri baitos.

Pa Y asi ir parādīts cik reizes varbūtisks algoritms strādā ātrāk neka determinēts.



2.5. att. **Virknes programmas rezultatu grafiks, kurš ir izveidots programmā Excel**

Rezultātu analīze

Ir atšķirība starp teoriju un praksi. To izskaidro tā, ka determinētu algoritmu izmantošanai tiek lietoti ļoti labas un optimizētas funkcijas, kuru izveidoja paša Borland kompānija, tās(funkcijas) ir stipri ātrākas nekā uzrakstīta „ar roku" pabitu failu salīdzināšana.

Punkti, kuri „izkrīt" no grafikas var paskaidrot ar to, ka determinēts algoritms atrada atšķirību tuvāk pie faila sākuma un pabeidza darbu ātrāk.

Vidējā ātruma atšķirība 4.06 tiešām ir ļoti liels pārsvars, tā kā pie maziem failiem algoritmu ātruma atšķirību nav iespējams noteikt koeficents ir 1. (mazo failu daudzums ir mazs.) Pie tam ari vidējo ātrumu ietekmē izkrītoši punkti.

Var paredzēt, ka ar failu izmēru palielināšanos darbības ātruma atšķirība arī palielināsies.

Pati programma strādā apmēram 32 minūtes. Bet pati buffera salīdzināšana atmiņā(determinēts algoritms) vai lielu skaitļu aritmētika (varbūtisks algoritms) ieņem < 1% no kopēja laika.

Polinomu salīdzināšana ar 0

Uzdevuma apraksts.

Ir dots polioms $f(x_1,...,x_n)$, un mēs gribam pārbaudīt nevienādību f = 0 (f ir vienads ar 0 pie jebkurām mainīgo vērtībām. Tāda pārbaude var būt vajadzīga, piemēram, ja mums ir matrica, kuru elementi ir polinomi un mums interesē vai ir tas determinants vienāds ar 0 vai nav. Izrēķināt determinantu šādā situacijā ir grūti

Determinēts algoritms

Determinēts algoritms ir ļoti vienkāršs. Tas ir tāds pats, kā pārbaudīt, izmantojot pildspalvu un papīru. Ņemam pirmo (ja tam koeficients pie burta nav nulle) polinoma locekļi (no kreisās puses) un salīdzinām ar visiem pārējiem locekļiem. Ja kādam loceklim no labās puses ir tāds pats burts un pakāpe (piemēram $4a^2$, $54a^2$), tad salīdzinām zīmes, pie vienādas zīmes pieliekam koeficientus pie burta, pie atšķirīgiem - atņemam un piešķiram loceklim jaunu zīmi, ja otram loceklim ir cita zīme un koeficients pie burta ir lielāks.

Polinoms veidā

([Exists[1]] [Sign[1]] int[1] string[1] ^ power[1] [Exists[2]] [Sign[2]] int[2] string[2] ^ power[2] .... [Exists[n]] [Sign[n]] int[n] string[n] ^ power[n])

n – numurs.

[**IsEmpty[n]**] = (**1** vai **0**) **ir aizsvitrots vai nav. 1 – nav 0 - ir**

[**Sign[n]**] - (1 vai 0) + vai – 1 = Plus 0 = minus

int[n] - koeficients pie burta (no 1 līdz 1000)

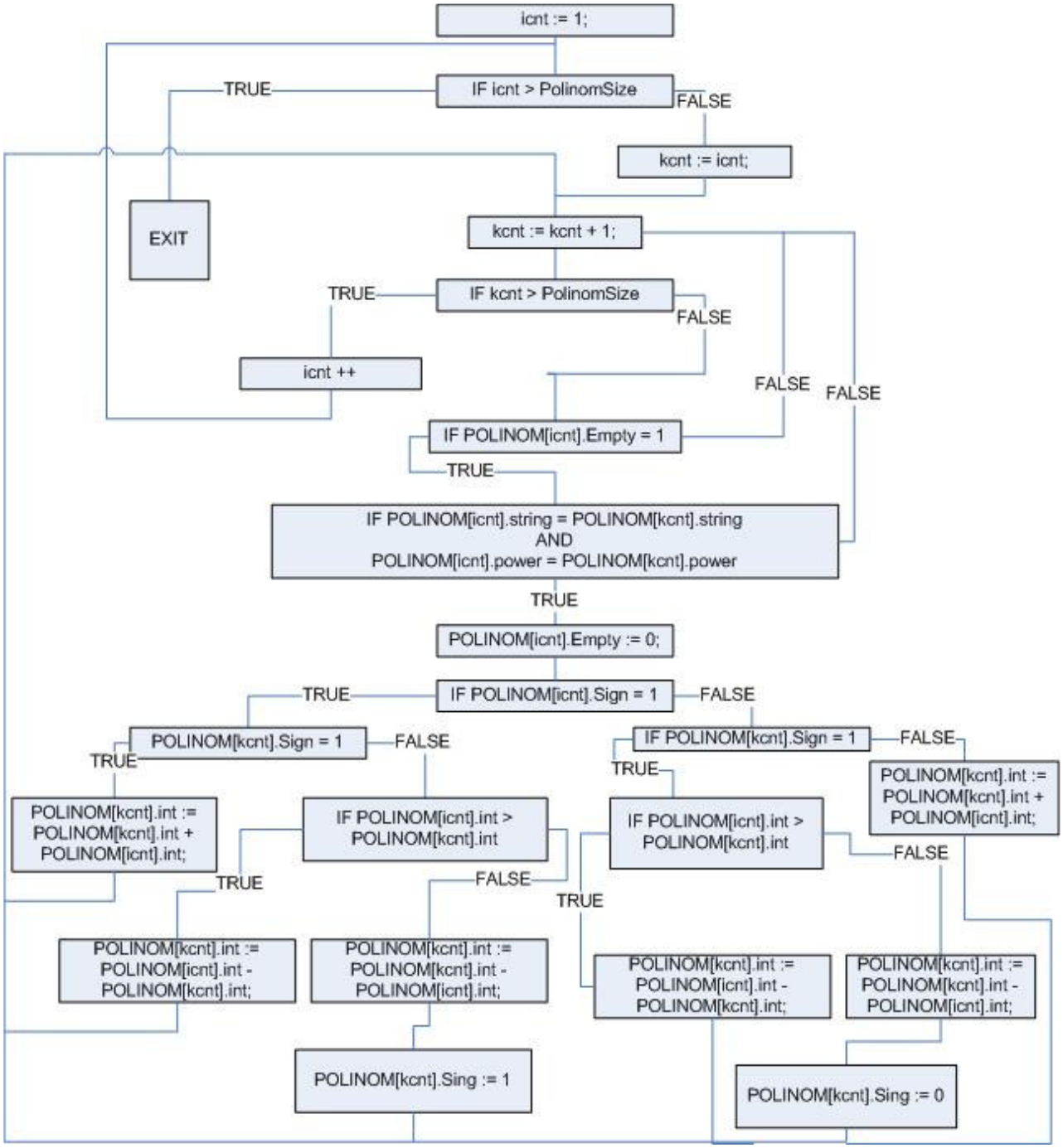
string[n] - burta koeficients [1,2,3,4,5] -> [a,b,c,x,y]

power[n] – burtas pakapes koeficients (maksimali 10)

```
 + 355c^10 [1]  - 355c^10 [1]  + 264a^2 [1]  - 264a^2 [1]  + 125b^10 [1]  - 125b
^10 [1]  - 666a^8 [1]  + 666a^8 [1]
```

2.6. att. **Nulles polinoma strukturas izvada piemers.**

Determīneta algoritma blokshēma.



2.7. att. **Polinoma programma blokshēma, ir izveidotā programmā VISIO**

Varbūtisks algoritms

- Izvēlēt nejaušus skatļus (izvēletā robežā), katram polinoma burtam. Ielikt polinomā un izrēķināt to vērtību.
- Salīdzināt rezultātu ar 0.
- Dot atbildi.

Lemma :

Lai $f(x_1 ,\ldots,x_n)$ - nenulles polinoms d ($d > 0$) pakapē uz lauka F. Tas nozīme, kā $\in$ $(a_1,\ldots,a_n) \in F^n$: $f(a_1 ,\ldots,a_n)\neq\mathbf{o}$. Lai arī $S \subseteq F$ un $|S|\geq d$. (galīga kopa, kurā ir vissmaz d elementus). Tad ir vissmaz $(|S|-d)^n$ punktus $(a_1,\ldots,a_n) \in S^n$, kuros $f(a_1 ,\ldots,a_n)\neq\mathbf{o}$.

Pierādījums :

Indukcija pēc n. Pie n=1 tas ir acīm redzams, tāpēc ka, nenulles polinomam d pakapē no viena mainīga nevar but vairāk par d saknem. izvēlesim tadu punktu $(a_1,\ldots,a_n)$, kur $f(a_1,\ldots,a_n)\neq\mathbf{o}$ un apskatīšim divus polinomus :

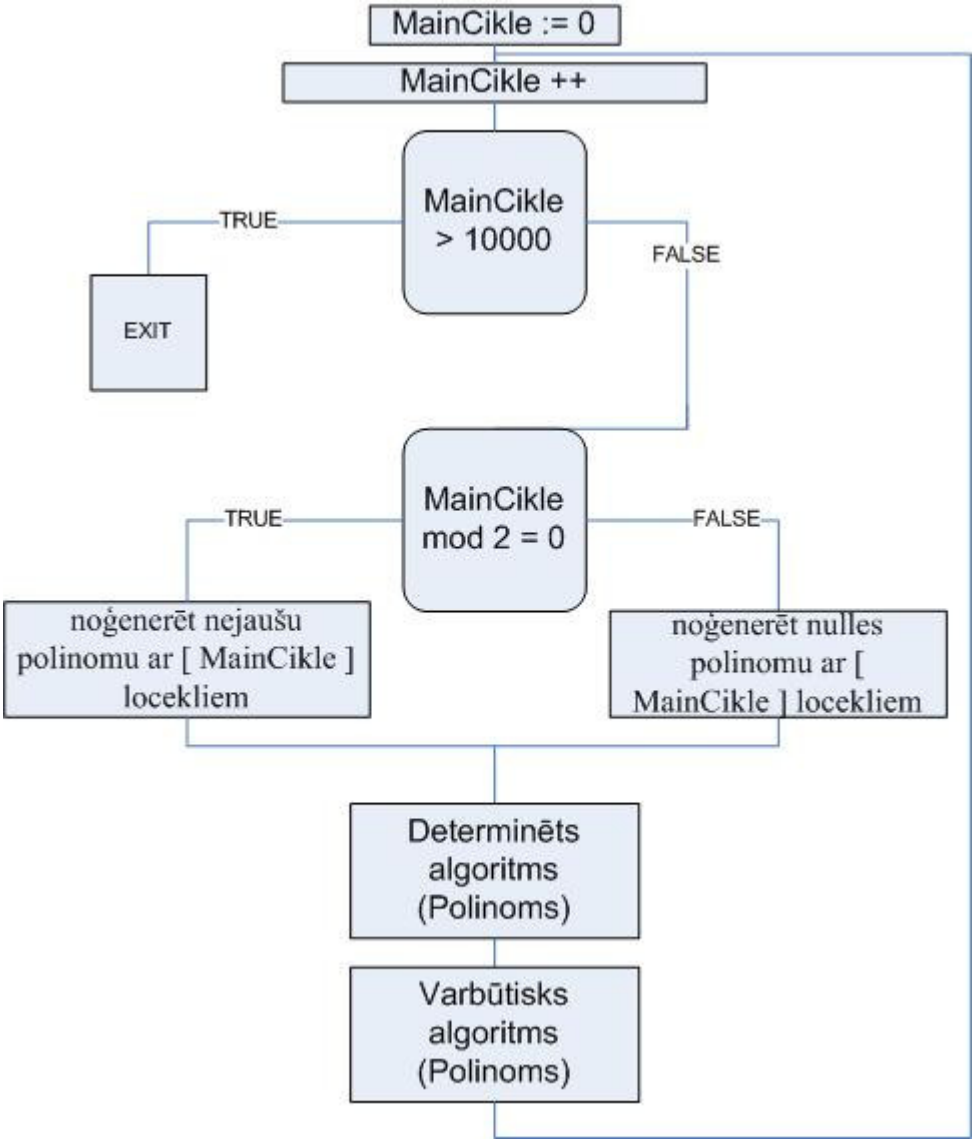
$f_1(x_1,\ldots, x_{n-1}) = f(x_1 ,\ldots ,x_{n-1},a_n)$

$f_2(x_n) = f(a_1 ,\ldots ,a_{n-1},x_n)$

Abiem šiem polinomiem pakape ne vairāk ka d un abi ir nenulles polinomi. f_1 ir atkarīgs no n - 1 mainīga un f_1 polinomam ir (pēc indukcijas pieņēmumu) vissmaz $(|S|-d)^{n-1}$ nenulles punkti uz S^{n-1} . f_2 - polinoms no viena mainīga un polinomam ir vissmaz $(|S|-d)$ nenulles punkti. Tāda veidā katram no $(|S|-d)$ a_n vertībam, kur polinoms $f_2\neq\mathbf{o}$ mums ir $(|S|-d)^{n-1}$ punkti, kur $f_1\neq\mathbf{o}$. Tātad pavisam nenulli punkti polinomam f vissmaz $(|S|-d)x(|S|-d)^{n-1}= (|S|-d)^n$.

Tātad, nejauši izvēletam $x_1, \ldots, x_n$ no galīgas kopas $S \subseteq F(|S| \geq d)$, varbūtība $f(x_1, \ldots, x_n) \neq \mathbf{o} \geq (|S|-d)^n / |S|^n = (1-d/|S|^n)$

Nulles polinoma realizācija



2.8. att. **Nulles polinoma programmas blokshēma, kura ir izveidota programmā VISIO**

Testu datu ģenerēšana

- Ciklā no 1 līdz 10000 jāģenere polinomus, ar locekļu skaitu = cikla mainīgam.
- Katrs otrais polinoms ir nulles polinoms.

Algoritmu laiku un precizitātes izskaitļošana

Determinēts algoritms

- Tiek noņemts esošais laiks : StartTick
- Tiek izmantots aprakstīts algoritms. [*2.7. att.*]
- Tiek noņemts esošais laiks : EndTick
- Ierakstīt failā Output1.txt EndTick – StartTick
- Ierakstīt failā Output3.txt answer

Varbūtisks algoritms.

- Tiek noņemts esošais laiks : StartTick
- Tiek izmantots aprakstīts algoritms.
- Tiek noņemts esošais laiks : EndTick
- Ierakstīt failā Output2.txt EndTick – StartTick
- Ierakstīt failā Output4.txt answer

Analīzes

- Output1.txt,Output2.txt,Otput3.txt,Output4.txt ielādēts uz Excel.
- Procentu izrēķināšana, uz kuru Output4.txt atšķiras no Output2.txt (Determinēts algoritms vienmēr strādā pareizi.)
- Parametru izskaitļošana : determinēta algoritma vidējais ātrums, varbūtiska algoritma vidējais ātrums, % varbūtiskā algoritma pareizība.
- Uzskates grafiku uzbūve.

Programmas kods

```
for MainCycle := 1 to 10000
do
begin
  PolinomSize := PolinomSize + 1;
 // Generate Polynom.
   if PolinomSize mod 2 <> 0  //IS NOT NULL
```

```
 then
 begin
  for genpol := 1 to PolinomSize
  do
  begin
   Wpower := random(10) + 1; //random 0 <= < X   10
   Wkoef := random(5) + 1;  //a..x 5
   TSign := random(2); //0 or 1 - or +
   TletNum := Random(1001) + 1;
   PolinomElem.PSign := TSign;
   PolinomElem.PKoef := Wkoef;
   PolinomElem.PLetNum := TletNum;
   PolinomElem.PPower := Wpower;
   PolinomElem.isempty := 1;
   PolinomStruct[genpol] := PolinomElem;
   PolinomStruct_DETERMINET[genpol] := PolinomElem;
   PolinomStruct_RANDOMNESS[genpol] := PolinomElem;
  end;
end
else  //GENERATE NULL POLINOM
begin
   genpol := 1;
   while(genpol <= PolinomSize)
   do
   begin
     Wpower := random(10) + 1; //random 0 <= < X   10
     Wkoef := random(5) + 1;  //a..x 5
     TSign := random(2); //0 or 1 - or +
     TletNum := Random(1001) + 1;
     PolinomElem.PSign := TSign;
     PolinomElem.PKoef := Wkoef;
     PolinomElem.PLetNum := TletNum;
```

```
      PolinomElem.PPower := Wpower;
      PolinomElem.isempty := 1;
      PolinomStruct[genpol] := PolinomElem;


    PolinomStruct_DETERMINET[genpol] := PolinomElem;
    PolinomStruct_RANDOMNESS[genpol] := PolinomElem;
    //Symetric monom
    PolinomStruct_DETERMINET[genpol+1] := PolinomElem;
    PolinomStruct_RANDOMNESS[genpol+1] := PolinomElem;
    //Change sign of the monom
    PolinomStruct_DETERMINET[genpol+1].PSign :=
AnotherSign(PolinomStruct_DETERMINET[genpol+1].PSign);
    PolinomStruct_RANDOMNESS[genpol+1].PSign :=
AnotherSign(PolinomStruct_RANDOMNESS[genpol+1].PSign);

    genpol := genpol + 2; //for +1 we have another;
    end;
  end;

  Summa := 0;

  //Deterministic Algorithm
  StartTick := GetTickCount;

  for icnt := 1 to PolinomSize
  do
  begin
    if PolinomStruct_DETERMINET[icnt].isempty = 1 //exists
    then
    begin
      for jcnt := icnt + 1 to PolinomSize
      do
```

```
        begin
         if PolinomStruct_DETERMINET[icnt].isempty = 1 //exists -> check
         then
         begin
          if (PolinomStruct_DETERMINET[icnt].PKoef =
PolinomStruct_DETERMINET[jcnt].PKoef) AND
             (PolinomStruct_DETERMINET[icnt].PPower =
PolinomStruct_DETERMINET[jcnt].PPower)
            then //SUMM
            begin
              PolinomStruct_DETERMINET[icnt].isempty := 0;
              if PolinomStruct_DETERMINET[icnt].PSign = 1 // +
              then
              begin
               if PolinomStruct_DETERMINET[jcnt].PSign = 1 // + +
               then
                 PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum + PolinomStruct_DETERMINET[icnt].PLetNum
                else  //Second polinom is -
                if PolinomStruct_DETERMINET[icnt].PLetNum <
PolinomStruct_DETERMINET[jcnt].PLetNum //second polinom >
                then
                begin
                  PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum - PolinomStruct_DETERMINET[icnt].PLetNum
                end
                else //second polinom <
                begin
                  PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[icnt].PLetNum - PolinomStruct_DETERMINET[jcnt].PLetNum;
                  PolinomStruct_DETERMINET[jcnt].PSign := 1; //become +
                end;
```

```
            end // if PolinomStruct_DETERMINET[icnt].PSign
            else //sign = 0 = -
            begin
              if PolinomStruct_DETERMINET[jcnt].PSign = 0 // + +
             then
              PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum + PolinomStruct_DETERMINET[icnt].PLetNum
              else  //Second polinom is +
              if PolinomStruct_DETERMINET[icnt].PLetNum <
PolinomStruct_DETERMINET[jcnt].PLetNum //second polinom >
              then
              begin
               PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum - PolinomStruct_DETERMINET[icnt].PLetNum
              end
              else //second polinom <
              begin
               PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[icnt].PLetNum - PolinomStruct_DETERMINET[jcnt].PLetNum;
               PolinomStruct_DETERMINET[jcnt].PSign := 0; //become +
              end;
             end;
           end;
         end; //End Chek is empty or not second polinom
       end; //All polinom througth
    end //end check existence
    else
    begin
     //nothing get next()
    end;
  end; //end cicle
```

```
  for jcnt := 1 to PolinomSize
  do
  begin
   if (PolinomStruct_DETERMINET[jcnt].isempty <> 0) AND
(PolinomStruct_DETERMINET[jcnt].PLetNum <> 0)
   then
   begin
    result := 0 //Is not NULL
   end
   else
   begin
    result := 1 //IS NULL
   end;
  END;

  endTick := GetTickCount;
  writeln(DetFail,PChar(MSecToTime(StartTick - EndTick)));
  writeln(DetFailAtb,intToStr(result));

  //Randomness Algorithm

  Summa := 0;
  StartTick := GetTickCount;
  //GENERATE
  abcxy[1] := (random(10) + 1)/Round(10)+1; //Lemma Var pieradit, ka var atrast saknes start
tiem skaitliem
  abcxy[2] := (random(10) + 1)/Round(10)+1; //ja Ir lieli skaitli ~40 tad 40 ^ 10 pakape nepareizi
strada.
  abcxy[3] := (random(10) + 1)/Round(10)+1;
  abcxy[4] := (random(10) + 1)/Round(10)+1;
  abcxy[5] := (random(10) + 1)/Round(10)+1;
```

```
  for icnt := 1 to PolinomSize
  do
  begin
   if PolinomStruct_RANDOMNESS[icnt].PSign = 1
   then
   begin
     Summa := Summa +
     ROUND(PolinomStruct_RANDOMNESS[icnt].PLetNum *
Power(abcxy[PolinomStruct_RANDOMNESS[icnt].PKoef],PolinomStruct_RANDOMNESS[icn
t].PPower))
   end
   else
   begin
     Summa := Summa -
     ROUND(PolinomStruct_RANDOMNESS[icnt].PLetNum *
Power(abcxy[PolinomStruct_RANDOMNESS[icnt].PKoef],PolinomStruct_RANDOMNESS[icn
t].PPower));
   end;
  end;

  if Summa <> 0 then
  begin
   result := 0; //Is not NULL
  end
   else
   result := 1; //IS NULL

  endTick := GetTickCount;
  writeln(VrbFail,PChar(MSecToTime(StartTick - EndTick)));
  writeln(VrbFailAtb,intToStr(result));
end;  //END MAIN CYCLE
```

Testu rezultāti.

Determinēta algoritma kopējais darbības laiks
(Sekundes, Miļi sekundes)
528.759

Determinēta algoritma videjais darbības laiks
(Sekundes, Miļi sekundes)
0.05288

Varbūtīska algoritma kopējais darbības laiks
(Sekundes, Miļi sekundes)
17.212

Varbūtīska algoritma videjais darbības laiks
(Sekundes, Miļi sekundes)
0.001

Varbūtiska algoritma precizitāte ir 100%

Varbūtiska algoritma ātrums vidēji 48.01 reizes lielāks nekā determinētam algoritmam.

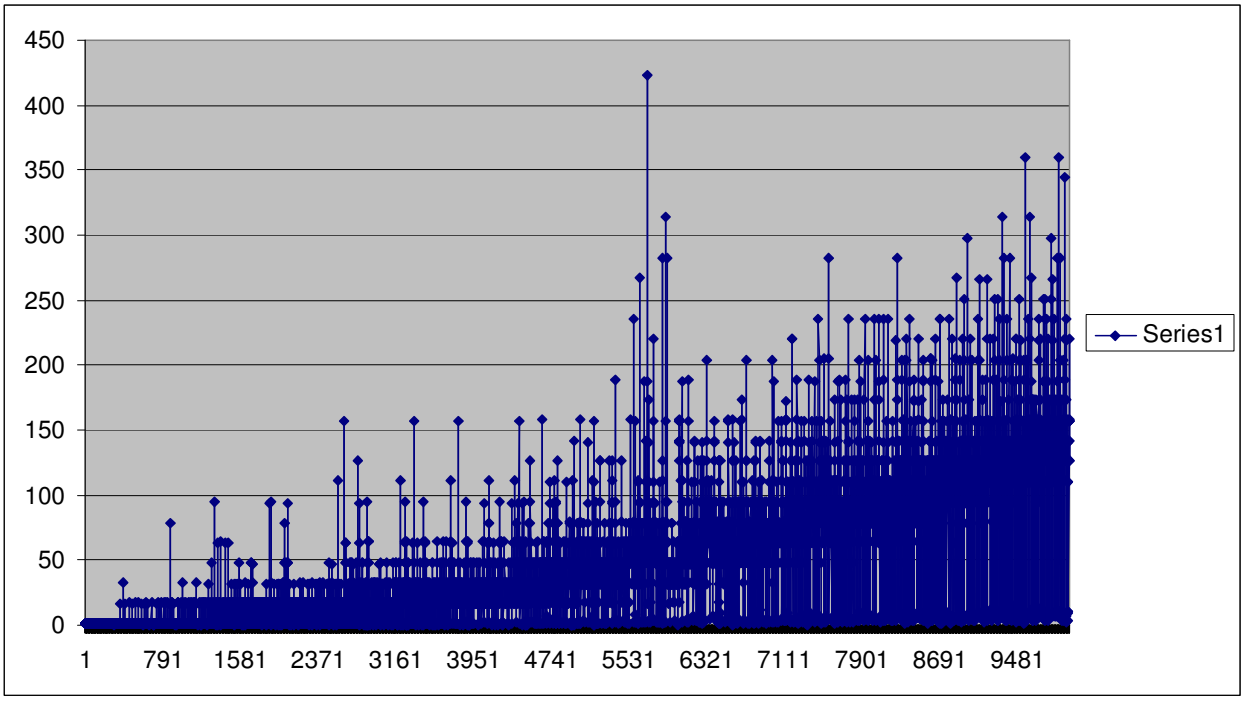
[*// ISNULL(a,b) ir ja mainīgam a nav vertības, (musu gadijumā < 1 ms) tad tai vertībai*
// tiek pieškirta b

SUM(**for icnt** := 1 to 10000
ISNULL(Detirminēta algoritma atrums [**icnt**],0.001) / ISNULL(Varbūtiska algoritma atrums [**icnt**],0.001)
]
=> 480131.67 / 10000 = **48.01**

Grafiks

Pa X asi polinoma locekļu daudzums.

Pa Y asi ir parādīts cik reizes varbūtisks algoritms strādā ātrāk neka determinēts.



2.9. att. **Nulles polinomu programmas rezultatu grafiks, kurš ir izveidots programmā Excel**

Rezultātu analīze

Ar locekļu daudzumu pieaugumu polinomā, varbūtiskā algoritma ātruma pārākums palielinās. Atsevišķus punktu lēcienus grafikā var paskaidrot ar to, ka kaut kādam noģenerētam polinomam matemātiskās operācijās bija daudz vienkāršāk. (Mazi skaitļi, maza pakāpe). Vai bija noģenerēts polinoms ar tieši dažādiem locekļiem, tādā veidā determinēts algoritms izdarīja daudz vairāk gājienus nekā parasti. (Var arī būt, ka pirmais un otrais variants vienlaicīgi.) Pie nelieliem polinomiem ātruma atšķirības nepamanāmas.

Tāpat grafiku ietekmē realizācijas specifika. Pie nenulles polinoma determinēts algoritms veic n – const (maksimāla const = 2/n) soļus. Pie nulles(katrs otrais) polinoma 2/n soļus, tāpēc ka noģenerēta nulles polinoma pirmais loceklis ģenerēs nejauši, bet otrais ir tās kopijas ar pretējo zīmi, trešais atkal nejaušs, bet ceturtais ir tā kopija ar pretējo zīmi u.t.t

Secinajumi

Varbūtiska algoritma novērtējums apriori ir subjektīvs.
Piemēram, algoritms darbojas ar precizitāti 99,8% - tā ir ļoti laba precizitāte, bet ja kļūda var novest pie smagām sekām, piemēram , algoritmu lietotāja vai kādas cilvēku grupas nāve, tad izmantot šo algoritmu negribas. No otras puses, ja algoritma precizitāte ir tikai 40% un uzdevums, piemēram, nodot mājas darbu universitātē, tad tāda precizitāte ir pilnīgi pieejama.

Jebkuram varbūtiskam / determinētam algoritms jābūt tādam kritērijam kā precizitāte (tā kā to varētu novērtēt) tāpat tādam kritērijam kā darbības laiks, ja kāda no šiem kritērijiem nav – tad tas ir murgi, bet ne algoritms.

Secinājumos jāsalīdzina varbūtiskus un determinētus algoritmus, izmantojot galvenokārt šos kritērijus. Dabīgi jāsalīdzina algoritmus, kuri paredzēti vienai uzdevuma izpildīšanai.

Kas mums no tā iznāca. .

Pirmais algoritms : Matricu vienādību pārbaude.

No praktiskā viedokļa uzdevums ir ļoti interesants, daudz kur zinātnē tas ir pieprasīts. Teorijā varbūtiska algoritma ātrums ir $O(n)^3$ / $O(n)^2$ reizes lielāks. (Faktiski ieguvām paātrinājumu, tādēļ ka salīdzinājām vienkāršākus objektus) Bet varbūtiska algoritma precizitāte ir lielāka par 50% un tikai tad, ja A * B <> C, pretējā gadījumā vienmēr ir 100%.

Ar matriču izmēru palielināšanu varbūtiska algoritma ātrums attiecībā pret determinēto paaugstinās. Praksē, biežāk matricas nevienādība būs patiesa, bet pat, ja izmanto varbūtisko algoritmu „sliktos" apstākļos (visas nevienādības ir aplamas), precizitātes zaudējumam jāsedzas ar ātruma palielinājumu. Tapāt arī var uzdot pašu negodīgāko testu varbūtiskam algoritmam : daudz mazu matriču nevienādības, starp kurām neviena nevienādība nav patiesa. Tad atšķirības ātrumā būs minimālas, bet varbūt atšķirības precizitātē determinēta algoritma labā. Bet tādu situāciju praksē grūti iedomāties. Zinātnieks tādā situācijā ,bez datoru palīdzības, var secināt, ka nevienādības nav patiesas.

Testi bija izpildīti 220 matriču nevienādībām, pie tam katra otrā nevienādība bija aplama. Maksimālais matrices izmērs ir 220x220, kam jāpietiek jebkuriem praktiskiem uzdevumiem.

Šajos testos varbūtisks algoritms parādīja precizitāti 100%. Vidējais ātrums ir 98.36 reizes lielāks, pie tam, ka maza izmēra matricām, ātruma atšķirības nav noteiktas !

Ņemot vēra iepriekš teikto, var viennozīmīgi teikt, ka varbūtisks algoritms identitāšu pārbaudei šajā gadījumā ir labāks, nekā determinēts algoritms.

Virknes salīdzināšana

Tās ir vēl vairāk pieprasīts no praktiskā viedokļa uzdevumus.
Failu salīdzināšana, signālu, atmiņas saturu : antivīrusu programmas darbība, brendmauru, ugunsmūru, programmas aizsardzība no vīrusu iekļūšanas, meklēšana un vēl citas pieprasītas lietas.

Teorijā varbūtiska algoritma ātrums ir O(n) / O($\log_2$ n) reizes lielāks. (Faktiski ieguvām paātrinājumu, tādēļ ka salīdzinājām vienkāršākus objektus) Bet varbūtiska algoritma precizitāte ir O(1/n). Tās nozīme, ka varbūtiskā algoritma precizitāte ir pietiekami laba. Ņemot vērā, ka praksē ir nepieciešams veikt miljonu dažādus salīdzinājumus, tad var zaudēt precizitāti, lai paaugstinātu ātrumu. Piemēram, filtrējam ar ugunsmūri ieejošās paketes. Nekas nav briesmīgs, ja nobloķējam 1-2 paketes varbūtiska algoritma kļūdas dēļ. Tāpat, ja ielaižam ap desmit „uzbrūkošas paketes", arī nebūs nekas traģisks, īpaši ņemot vērā, ka ar lielu varbūtību kļūdas nebūs izdarītas pēc kārtas. Maza kļūda nav briesmīga, bet no citas puses bieži palielināt ātrumu ir kritiski vajadzīgs. Pats neveiksmīgākais tests varbūtiskam algoritmam ir tas : izveidot daudz liela izmēra failu, kuri atšķiras tikai pirmos bitos. Tad vidēji determinēta algoritma ātrums būs līdzīgs varbūtiska algoritma ātrumam, bet determinēta algoritma precizitāte būs lielāka. Bet no praktiskā viedokļa tādu situāciju ir grūti iedomāties. Tādi faili(Liela izmēra un atšķiras pirmos bitos) salīdzināšanai reti parādās. (Ļoti maza varbūtība) Ja tikai nesalīdzināt lielu kopu ar vispār dažādiem failiem (Tieši dažādiem failiem varbūtība, ka tie atšķiras pirmos bitos ir liela). Bet kam tas ir vajadzīgs ?

Pie failu, mazāku par vienu MG (1024 bait) testēšanas, varbūtisks algoritms parādīja precizitāti 100%, bet ātrums ir 4.06 reizes lielāks nekā determinētam algoritmam. Pie failu izmēru pieauguma ātruma atšķirība palielinās.

Ņemot vēra iepriekš teikto, var viennozīmīgi teikt, ka varbūtisks algoritms identitāšu pārbaudei šajā gadījumā ir labāks, nekā determinēts algoritms.

Nulles polinoms

Algoritms ātrāk (Jāatceras par to, ka nulles polinoma ģenerācijas specifikas dēļ determinēts algoritms strādā ar šiem polinomiem ātrāk nekā ar nenulles polinomiem, ja izmainītu ģenerācijas specifiku (sajauktu polinomu locekļus ar vietām) vai iekļaut determinētā algoritmā polinoma kārtošanu, tad determinēta algoritma vidējais darbības laiks palielinās, kas runā varbūtiska algoritma labā.) strādā nekā determinētais algoritms, un reti kļūdās. Biežāk praksē to izmantot būs izdevīgi. Ar polinomu izmēru pieaugumu (locekļu skaits palielinās) Es izveidoju testu izmantojot 10000 polinomos. (No viena locekļa līdz 10000).

Ja sarežģītu polinoma locekļu struktūru (no kombinācijas no viena burta [4a^3; 5b^7 u.t.t] līdz kombinācijai no dažādiem burtiem [4a^1c^2x^5]), tad varbūtiskam algoritmam jāveic vairāk matemātiskas operācijas, bet determinētam jāveic lielāku soļu skaitu un tam būs nepieciešama vairāk sarežģītāka loģika. (saprast, ka abx = xba, xbd <> abd u.t.t) Pār cik ar polinoma locekļu skaita pieaugumu varbūtisks algoritms uzvar, var secināt, ka pēc polinoma struktūras izmaiņām rezultāts neizmainīsies.

Vienīgā problēma varbūtiskam algoritmam ir milzīgie pakāpes vai burtu koeficienti, piemēram, [50a^80 +
800000678678678678678678634512341525325435345345345346 43b^120].
Determinētam algoritmam nebūs problēmas ar šo uzdevumu, bet varbūtiskam algoritmam būs nopietnas problēmas ar matemātiskām operācijām.
Pie tādiem polinomiem (ja noģenerē tādus testu datus, kuri sastāv pilnībā no šādiem polinomiem) varbūtisks algoritms zaudēs. Bet no praktiskā viedokļa tāds uzdevums ir neticams. Tādus testu parametrus, kurus es devu, jāpietiek jebkuram praktiskam uzdevumam.

Ņemot vēra iepriekš teikto, var viennozīmīgi teikt, ka varbūtisks algoritms identitāšu pārbaudei šajā gadījumā ir labāks, nekā determinēts algoritms.

Algoritmu atzīmes

Algoritms	*Elegance*	*Precizitāte*	*Ātrums*	*Vienkaršība / Prasības pret resursiem*
1.Determinēts Matricas	*4*	*10+*	*5*	*9*
2. Determinēts Virknes	*7*	*10+*	*7*	*9+ (var izmantot jau gatavos funkcijas)*
3. Determinēts Polinoms	*5*	*10+*	*5*	*2− (grūti realizēt, jāveido strukturu lai glābt, daudz funkcijas šo strukturu apstradei)*
4.Varbūtisks Matricas	*10+*	*10*	*10+*	*10*
5.Varbūtisks Virknes	*9+*	*10*	*9*	*4*
6.Varbūtisks Polinoms	*8*	*10*	*9+*	*10*

Es uzskatu, ka mana bakalaura darba mērķis ir sasniegts : Es paspēju parādīt ar piemēriem, kā varbūtiski algoritmi identitāšu pārbaudei ir labāki nekā determinēti algoritmi. Ja speciāli nedot tādus testu datus, lai parādītu pretējo, bet dot normālus no praktiska viedokļa testu datus, tad tā ir.

Varbūtiski algoritmi ir labāki nekā determinēti algoritmi !

Izmantotā literatūra un avoti

1. **De Leeuw, K., Moore, E. F., Shannon**, C.E., un Shapiro, N. Compitability by probabilistic machines. In Automata Studies, C.E. Shannon un J.McCarthy, red. Princeton University Press, Princeton, NJ, 1955. 183-212 lpp.
2. **Rabin, M. O.** Probabilistic automata, Inf. Control 6, 1963 230-245. lpp.
3. **Gill, J**. Computational complexity of probabilistic Turing machines. SIAM J. Comput 6, 4(Dec.), 1977. 675-695. lpp.
4. **Berlekamp, E. R.** Factoring polynomials over large finite fields. Math. Comput. 24, 1970. 713-735. lpp.
5. **Rabin, M. O**. Probabilistic algorithms. In algorithms and Complexity, Recent Results and New Directions, J.F Traub, Ed., Academic Press, New York, 1976. 21-39. lpp.
6. **Solovay, R. un Strassen, V. A** *fast Monte-Carlo test for primality.* SIAM J. Comput. 6, 1 (March), 1977. 84-85 lpp. Arī SIAM J. Comput 7, 1 (Feb.), 1978. 118. lpp.
7. **Karp, R. M.** *An introduction to randomized algorithms*. Discrete Appl. Math. 34, 1991. 165-201.
8. **Maffioli, F., Speranza, M., and Vercellis, C.** *Randomized algorithms, in Combinatorial Optimization - Annotated Bibliographies.* M. O'hEigertaigh, J.K Lenstra, and A.H.G. Rinooy Kan, Eds., Wiley, New Youk, 1985. 89-105. lpp.
9. **Welsh, D. J. A**. *Randomised algorithms.* Discrete. Appl. Math. 5, 1983. 133-145. lpp.
10. **А.Х. Шень, М.Н**. **Вялый** *Классические и квантовые вычисления* Интернет-библиотека МЦНМО. nodaļa Вероятностные алгоритмы и класс BPP. Проверка простоты числа.
11. **Knuth. D.E** *The Art of Computer Programming.*
12. **Freivalds, R**., *Fast probabilistic algorithms*, Proc. Of Mathematical Foundations of Computer Sciense, Springer-Verlag Lecture Notes in Computer Science, (1979), 57-69. lpp.
13. **Freivalds, R.** 1977. Probabilistic machines can use less running time. In Information Processing 77, Proceedings of IFIP Congress 77, B. Gilchrist, Ed., (Aug.), North-Holland, Amsterdam, 839-842.
14. **http://forum.sources.ru**
15. **ALON, N. AND SPENCER, J**. *The Probabilistic Method.* Wiley, New York. 1992.
16. **М. Вялый** *Случайность как вычислительный ресурс* журнал "Компьютерра" №10 от 18 марта 2002 год

17. ***http://alglib.sources.ru/***

18. ***http://www.delphikingdom.com/***

Pielikumi

1. pielikums

Programmas output failu paraugi(fragmenti)

(Visiem algoritmiem pēc strukturas izskatas lidzīgi)

Output1.txt

Output2.txt

Output3.txt

Output4.txt

Programmas kods (Matricas) (Kompilators Delphi 7)

{ Project1 – programmas galvenais unit.

MatrixOp – unit lai nodorboties ar matricam tiek paņemts no www.delphikingdom.com dažadas funkcijas tiek izmainitas. Un izveidotas jaunas funkcijas (piem. CompareMatrix) }

```
program Project1;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Windows,
  MatixOp in 'MatixOp.pas';
{
function MSecToTime(mSec: Int64): string;
var
  dt : TDateTime;
begin
   dt := mSec / MSecsPerSec / SecsPerDay;
   Result := Format('%d days, %s', [Trunc(dt), FormatDateTime('hh:nn:ss.z', Frac(dt))]) ;
end;
}
function MSecToTime(mSec: Int64): string;
var
  dt : TDateTime;
begin
   dt := mSec / MSecsPerSec / SecsPerDay;
   Result := Format('%d* %s', [Trunc(dt), FormatDateTime('ss.zzz', Frac(dt))]) ;
end;

var
```

```
  TRandMatrixA  : MatrixPtr;
  TRandMatrixB  : MatrixPtr;
  TRandMatrixC  : MatrixPtr;
  TRandVector : MatrixPtr;
  icnt,jcnt : integer;
  numRand : integer;
  startTick, endTick : Int64;
  DetFail : TextFile;
  VrbFail : TextFile;
  DetFailAtb : TextFile;
  VrbFailAtb : TextFile;

  Answer : integer;

begin
  numRand := 10000;
  //Creating matrix Cicle
   AssignFile(DetFail , 'Output1.txt');
   ReWrite(DetFail);
   AssignFile(VrbFail, 'Output2.txt');
   ReWrite(VrbFail);
   AssignFile(DetFailAtb, 'Output3.txt');
   ReWrite(DetFailAtb);
   AssignFile(VrbFailAtb, 'Output4.txt');
   ReWrite(VrbFailAtb);

   // Close the file

  for icnt := 4 to 220
  do
  begin
    //Random
    TRandMatrixA := CreateSquareMatrix(icnt); //Create Square Matrix where n = i
```

```
 TRandMatrixB := CreateSquareMatrix(icnt);

  FillMatrixAtRandom(TRandMatrixA,NumRand);
  FillMatrixAtRandom(TRandMatrixB,NumRand);

 TRandVector := CreateMatrix(icnt,1);
 FillMatrixAtRandom(TRandVector,2);

  if icnt mod 2 = 0 //Each Second C Matrix Apply
  then
  begin
    TRandMatrixC := MultipleMatrixOnMatrix(TRandMatrixA,TRandMatrixB);
  end
  else
  begin
    TRandMatrixC := CreateSquareMatrix(icnt);
    FillMatrix(TRandMatrixC,NumRand);
  end;

 //Deterministic Algorithm
 StartTick := GetTickCount;
 if
CompareMatrix(MultipleMatrixOnMatrix(TRandMatrixA,TRandMatrixB),TRandMatrixC,ic
nt,icnt) then
 begin
  answer := 1;
 end
 else
  answer := 0;

 endTick := GetTickCount;
 writeln(DetFail,PChar(MSecToTime(StartTick - EndTick)));
 writeln(DetFailAtb,intToStr(answer));
```

```
   //Randomness Algorithm
   StartTick := GetTickCount;
   if CompareMatrix(MultipleMatrixOnMatrix(TRandMatrixA,MultipleMatrixOnMatrix(
      TRandMatrixB,TRandVector)),
MultipleMatrixOnMatrix(TRandMatrixC,TRandVector),icnt,1)
      then
   begin
    answer := 1;
   end
   else
    answer := 0;

   endTick := GetTickCount;
   writeln(vrbFail,PChar(MSecToTime(StartTick - EndTick)));
   writeln(vrbFailAtb,intToStr(answer));


  end; //End Generation Cicle

  //TRandMatrix := CreateSquareMatrix(NumRand);
//   DisplayMatrix(TRandMatrixC,7,0);
  CloseFile(DetFail);
  CloseFile(VrbFail);
  CloseFile(DetFailAtb);
  CloseFile(VrbFailAtb);
  readln;
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

```
// http://www.delphikingdom.com
Unit MatixOp;

interface

type
  MatrixPtr = ^MatrixRec;
  MatrixRec = record
    MatrixRow   : byte;
    MatrixCol   : byte;
    MatrixArray : pointer;
  end;
  MatrixElement = real;

(* Функция возвращает целочисленную степень *)
function IntPower(X,n : integer) : integer;

(* Функция создает квадратную матрицу *)
function  CreateSquareMatrix(Size : byte) : MatrixPtr;

(* Функция создает прямоугольную матрицу *)
function  CreateMatrix(Row,Col : byte) : MatrixPtr;

(* Функция дублирует матрицу *)
function  CloneMatrix(MPtr : MatrixPtr) : MatrixPtr;

(* Функция удаляет матрицу и возвращает TRUE в случае удачи *)
function  DeleteMatrix(var MPtr : MatrixPtr) : boolean;

(* Функция заполняет матрицу указанным числом *)
function  FillMatrix(MPtr : MatrixPtr;Value : MatrixElement) : boolean;

(* Функция заполняет матрицу указанным числом *)
```

```
function  FillMatrixAtRandom(MPtr : MatrixPtr;Value : MatrixElement) : boolean;


(* Функция удаляет матрицу MPtr1 и присваивает ей значение MPtr2 *)
function  AssignMatrix(var MPtr1 : MatrixPtr;MPtr2 : MatrixPtr) : MatrixPtr;

(* Функция отображает матрицу на консоль *)
function  DisplayMatrix(MPtr : MatrixPtr;_Int,_Frac : byte) : boolean;

(* Функция возвращает TRUE, если матрица 1x1 *)
function  IsSingleMatrix(MPtr : MatrixPtr) : boolean;

(* Функция возвращает TRUE, если матрица квадратная *)
function  IsSquareMatrix(MPtr : MatrixPtr) : boolean;

(* Функция возвращает количество строк матрицы *)
function  GetMatrixRow(MPtr : MatrixPtr) : byte;

(* Функция возвращает количество столбцов матрицы *)
function  GetMatrixCol(MPtr : MatrixPtr) : byte;

(* Процедура устанавливает элемент матрицы *)
procedure SetMatrixElement(MPtr : MatrixPtr;Row,Col : byte;Value : MatrixElement);

(* Функция возвращает элемент матрицы *)
function  GetMatrixElement(MPtr : MatrixPtr;Row,Col : byte) : MatrixElement;

(* Функция исключает векторы из матрицы *)
function  ExcludeVectorFromMatrix(MPtr : MatrixPtr;Row,Col : byte) : MatrixPtr;

(* Функция заменяет строку(столбец) матрицы вектором *)
function  SetVectorIntoMatrix(MPtr,VPtr : MatrixPtr;_Pos : byte) : MatrixPtr;
```

```
(* Функция возвращает детерминант матрицы *)
function  DetMatrix(MPtr : MatrixPtr) : MatrixElement;

(* Функция возвращает детерминант треугольной матрицы *)
function  DetTriangularMatrix(MPtr : MatrixPtr) : MatrixElement;

(* Функция возвращает алгебраическое дополнение элемента матрицы *)
function  AppendixElement(MPtr : MatrixPtr;Row,Col : byte) : MatrixElement;

(* Функция создает матрицу алгебраических дополнений элементов матрицы *)
function  CreateAppendixMatrix(MPtr : MatrixPtr) : MatrixPtr;

(* Функция транспонирует матрицу *)
function TransponeMatrix(MPtr : MatrixPtr) : MatrixPtr;

(* Функция возвращает обратную матрицу *)
function ReverseMatrix(MPtr : MatrixPtr) : MatrixPtr;

(* Функция умножает матрицу на число *)
function MultipleMatrixOnNumber(MPtr : MatrixPtr;Number : MatrixElement) : MatrixPtr;

(* Функция умножает матрицу на матрицу *)
function MultipleMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;

(* Функция суммирует две матрицы *)
function AddMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;

(* Функция вычитает из первой матрицы вторую *)
function SubMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;

(* Функция решает систему методом Гаусса и возвращает LU-матрицы *)
(* Результат функции - вектор-столбец решений                    *)
```

```
function GausseMethodMatrix(MPtr,VPtr : MatrixPtr;var LPtr,UPtr,BPtr : MatrixPtr) :
MatrixPtr;

function CompareMatrix(MPtr1,MPtr2 : MatrixPtr; row,col : integer) : boolean;

implementation

function CompareMatrix(MPtr1,MPtr2 : MatrixPtr; row,col : integer) : boolean;
var
  i,j : integer;
begin
  result := True;
  for i:= 1 to row do
  begin
    for j:= 1 to col do
    begin
      IF GetMatrixElement(MPtr1,i,j) <> GetMatrixElement(MPtr2,i,j)
      then
      Result := False;
    end;
  end;

end;

function IntPower(X,n : integer) : integer;
var
  Res,i : integer;
begin
  if n < 1 then IntPower:= 0
  else begin
    Res:= X;
    for i:=1 to n-1 do Res:= Res*X;
    IntPower:= Res;
```

```
  end;
end;


function CreateSquareMatrix(Size : byte) : MatrixPtr;
var
  TempPtr : MatrixPtr;
begin
  TempPtr:= nil;
  GetMem(TempPtr,SizeOf(MatrixRec));
  if TempPtr = nil then begin
    CreateSquareMatrix:= nil;
    Exit;
  end;
  with TempPtr^ do begin
    MatrixRow:= Size;
    MatrixCol:= Size;
    MatrixArray:= nil;
    GetMem(MatrixArray,Size*Size*SizeOf(MatrixElement));
    if MatrixArray = nil then begin
      FreeMem(TempPtr,SizeOf(MatrixRec));
      CreateSquareMatrix:= nil;
      Exit;
    end;
  end;
  FillMatrix(TempPtr,0);
  CreateSquareMatrix:= TempPtr;
end;


function CreateMatrix(Row,Col : byte) : MatrixPtr;
var
  TempPtr : MatrixPtr;
```

```
begin
  TempPtr:= nil;
  GetMem(TempPtr,SizeOf(MatrixRec));
  if TempPtr = nil then begin
    CreateMatrix:= nil;
    Exit;
  end;
  with TempPtr^ do begin
    MatrixRow:= Row;
    MatrixCol:= Col;
    MatrixArray:= nil;
    GetMem(MatrixArray,Row*Col*SizeOf(MatrixElement));
    if MatrixArray = nil then begin
      FreeMem(TempPtr,SizeOf(MatrixRec));
      CreateMatrix:= nil;
      Exit;
    end;
  end;
  FillMatrix(TempPtr,0);
  CreateMatrix:= TempPtr;
end;


function DeleteMatrix(var MPtr : MatrixPtr) : boolean;
begin
  if MPtr = nil then DeleteMatrix:= FALSE
  else with MPtr^ do begin
    if MatrixArray <> nil then
      FreeMem(MatrixArray,MatrixRow*MatrixCol*SizeOf(MatrixElement));
    FreeMem(MPtr,SizeOf(MatrixRec));
    MPtr:= nil;
    DeleteMatrix:= TRUE;
  end;
```

```
end;


function CloneMatrix(MPtr : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j     : byte;
begin
  if MPtr = nil then CloneMatrix:= nil
  else with MPtr^ do begin
    TempPtr:= CreateMatrix(MPtr^.MatrixRow,MPtr^.MatrixCol);
    if TempPtr <> nil then begin
      for i:= 1 to MatrixRow do
        for j:= 1 to MatrixCol do
          SetMatrixElement(TempPtr,i,j,GetMatrixElement(MPtr,i,j));
      CloneMatrix:= TempPtr;
    end else CloneMatrix:= nil;
  end;
end;


function FillMatrix(MPtr : MatrixPtr;Value : MatrixElement) : boolean;
var
  i,j : byte;
begin
  if MPtr = nil then FillMatrix:= FALSE
  else with MPtr^ do begin
    for i:= 1 to MatrixRow do
      for j:= 1 to MatrixCol do
        SetMatrixElement(MPtr,i,j,Value);
    FillMatrix:= TRUE;
  end;
end;
```

```
function FillMatrixAtRandom(MPtr : MatrixPtr;Value : MatrixElement) : boolean;
var
  icnt,jcnt : byte;
  GenMaxValue : Integer;
  InsertValue : MatrixElement;
begin
  Randomize;
  GenMaxValue := Round(Value);
  if MPtr = nil then FillMatrixAtRandom:= FALSE
  else with MPtr^ do begin
    for icnt:= 1 to MatrixRow do
    begin
      for jcnt:= 1 to MatrixCol do
      begin
        InsertValue := random(GenMaxValue);
        SetMatrixElement(MPtr,icnt,jcnt,InsertValue);
      end;
    end;
    FillMatrixAtRandom:= TRUE;
  end;
end;


function AssignMatrix(var MPtr1 : MatrixPtr;MPtr2 : MatrixPtr) : MatrixPtr;
begin
  DeleteMatrix(MPtr1);
  MPtr1:= MPtr2;
  AssignMatrix:= MPtr1;
end;
```

```
function DisplayMatrix(MPtr : MatrixPtr;_Int,_Frac : byte) : boolean;
var
  i,j : byte;
begin
  if MPtr = nil then DisplayMatrix:= FALSE
  else with MPtr^ do begin
    for i:= 1 to MatrixRow do begin
      for j:= 1 to MatrixCol do
        write(GetMatrixElement(MPtr,i,j) : _Int : _Frac);
      writeln;
    end;
    DisplayMatrix:= TRUE;
  end;
end;


function IsSingleMatrix(MPtr : MatrixPtr) : boolean;
begin
  if MPtr <> nil then with MPtr^ do begin
    if (MatrixRow = 1) and (MatrixCol = 1) then
      IsSingleMatrix:= TRUE
    else IsSingleMatrix:= FALSE;
  end else IsSingleMatrix:= FALSE;
end;


function IsSquareMatrix(MPtr : MatrixPtr) : boolean;
begin
  if MPtr <> nil then with MPtr^ do begin
    if MatrixRow = MatrixCol then
      IsSquareMatrix:= TRUE
    else IsSquareMatrix:= FALSE;
  end else IsSquareMatrix:= FALSE;
```

```
end;

function GetMatrixRow(MPtr : MatrixPtr) : byte;
begin
  if MPtr <> nil then GetMatrixRow:= MPtr^.MatrixRow
  else GetMatrixRow:= 0;
end;

function GetMatrixCol(MPtr : MatrixPtr) : byte;
begin
  if MPtr <> nil then GetMatrixCol:= MPtr^.MatrixCol
  else GetMatrixCol:= 0;
end;

procedure SetMatrixElement(MPtr : MatrixPtr;Row,Col : byte;Value : MatrixElement);
var
  TempPtr : ^MatrixElement;
begin
  if MPtr <> nil then
    if (Row <> 0) or (Col <> 0) then with MPtr^ do begin
      pointer(TempPtr):= pointer(MatrixArray);
      Inc(TempPtr,MatrixRow*(Col-1)+Row-1);
      TempPtr^:= Value;
    end;
end;


function GetMatrixElement(MPtr : MatrixPtr;Row,Col : byte) : MatrixElement;
var
  TempPtr : ^MatrixElement;
begin
  if MPtr <> nil then begin
    if (Row <> 0) and (Col <> 0) then with MPtr^ do begin
```

```
    pointer(TempPtr):= pointer(MatrixArray);
    Inc(TempPtr,MatrixRow*(Col-1)+Row-1);
    GetMatrixElement:= TempPtr^;
   end else GetMatrixElement:= 0;
  end else GetMatrixElement:= 0;
end;


function ExcludeVectorFromMatrix(MPtr : MatrixPtr;Row,Col : byte) : MatrixPtr;
var
  NewPtr : MatrixPtr;
  NewRow, NewCol : byte;
  i,j : byte;
  DiffRow, DiffCol : byte;
begin
  if MPtr <> nil then with MPtr^ do begin

    if Row = 0 then NewRow:= MatrixRow
    else NewRow:= MatrixRow-1;
    if Col = 0 then NewCol:= MatrixCol
    else NewCol:= MatrixCol-1;

    NewPtr:= CreateMatrix(NewRow, NewCol);
    if (NewPtr = nil) or (NewPtr^.MatrixArray = nil) then begin
      ExcludeVectorFromMatrix:= nil;
      Exit;
    end;

    DiffRow:= 0;
    DiffCol:= 0;
    for i:= 1 to MatrixRow do begin
      if i = Row then DiffRow:= 1
      else  for j:= 1 to MatrixCol do if j = Col then DiffCol:= 1
```

```
      else SetMatrixElement(NewPtr,i-DiffRow,j-DiffCol,
        GetMatrixElement(MPtr,i,j));
     DiffCol:= 0;
   end;

   ExcludeVectorFromMatrix:= NewPtr;
 end else ExcludeVectorFromMatrix:= nil;
end;


function SetVectorIntoMatrix(MPtr,VPtr : MatrixPtr;_Pos : byte) : MatrixPtr;
var
 TempPtr : MatrixPtr;
 i       : byte;
begin
 if (MPtr <> nil) and (VPtr <> nil) then begin
   TempPtr:= CloneMatrix(MPtr);
   if TempPtr = nil then begin
     SetVectorIntoMatrix:= nil;
     Exit;
   end;
   if VPtr^.MatrixRow = 1 then begin
     for i:= 1 to TempPtr^.MatrixCol do
       SetMatrixElement(TempPtr,_Pos,i,GetMatrixElement(VPtr,1,i));
   end else begin
     for i:= 1 to TempPtr^.MatrixRow do
       SetMatrixElement(TempPtr,i,_Pos,GetMatrixElement(VPtr,i,1));
   end;
   SetVectorIntoMatrix:= TempPtr;
 end else SetVectorIntoMatrix:= nil;
end;
```

```
function DetMatrix(MPtr : MatrixPtr) : MatrixElement;
var
  TempPtr : MatrixPtr;
  i,j     : byte;
  Sum     : MatrixElement;
begin
  if IsSquareMatrix(MPtr) then begin
    if not IsSingleMatrix(MPtr) then begin
      TempPtr:= nil;
      Sum:= 0;
      for j:= 1 to GetMatrixCol(MPtr) do begin
        AssignMatrix(TempPtr,ExcludeVectorFromMatrix(MPtr,1,j));
        Sum:= Sum+IntPower(-1,j+1)*GetMatrixElement(MPtr,1,j)*DetMatrix(TempPtr);
      end;
      DeleteMatrix(TempPtr);
      DetMatrix:= Sum;
    end else DetMatrix:= GetMatrixElement(MPtr,1,1);
  end else DetMatrix:= 0;
end;


function DetTriangularMatrix(MPtr : MatrixPtr) : MatrixElement;
var
  i       : byte;
  Sum     : MatrixElement;
begin
  if IsSquareMatrix(MPtr) then begin
    Sum:= 1;
    for i:= 1 to MPtr^.MatrixRow do
      Sum:= Sum*GetMatrixElement(MPtr,i,i);
    DetTriangularMatrix:= Sum;
  end else DetTriangularMatrix:= 0;
end;
```

```
function AppendixElement(MPtr : MatrixPtr;Row,Col : byte) : MatrixElement;
var
  TempPtr : MatrixPtr;
begin
  if IsSquareMatrix(MPtr) then begin
    TempPtr:= ExcludeVectorFromMatrix(MPtr,Row,Col);
    if TempPtr = nil then begin
      AppendixElement:= 0;
      Exit;
    end;
    AppendixElement:= IntPower(-1,Row+Col)*DetMatrix(TempPtr);
    DeleteMatrix(TempPtr);
  end else AppendixElement:= 0;
end;


function CreateAppendixMatrix(MPtr : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j     : byte;
begin
  if (MPtr <> nil) or (MPtr^.MatrixArray <> nil) or
     (not IsSquareMatrix(MPtr)) then with MPtr^ do begin
    TempPtr:= CreateMatrix(MatrixCol,MatrixRow);
    for i:= 1 to MatrixRow do
      for j:= 1 to MatrixCol do
        SetMatrixElement(TempPtr,i,j,AppendixElement(MPtr,i,j));
    CreateAppendixMatrix:= TempPtr;
  end else CreateAppendixMatrix:= nil;
end;
```

```
function TransponeMatrix(MPtr : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j     : byte;
begin
  if (MPtr <> nil) or (MPtr^.MatrixArray <> nil) then with MPtr^ do begin
    TempPtr:= CreateMatrix(MatrixCol,MatrixRow);
    for i:= 1 to MatrixRow do
      for j:= 1 to MatrixCol do
        SetMatrixElement(TempPtr,j,i,GetMatrixElement(MPtr,i,j));
    TransponeMatrix:= TempPtr;
  end else TransponeMatrix:= nil;
end;


function ReverseMatrix(MPtr : MatrixPtr) : MatrixPtr;
var
  TempPtr     : MatrixPtr;
  Determinant : MatrixElement;
begin
  if MPtr <> nil then begin
    TempPtr:= nil;
    AssignMatrix(TempPtr,CreateAppendixMatrix(MPtr));
    AssignMatrix(TempPtr,TransponeMatrix(TempPtr));
    Determinant:= DetMatrix(MPtr);
    if (TempPtr = nil) or (Determinant = 0) then begin
      DeleteMatrix(TempPtr);
      ReverseMatrix:= nil;
      Exit;
    end;
    AssignMatrix(TempPtr,MultipleMatrixOnNumber(TempPtr,1/Determinant));
```

```
    ReverseMatrix:= TempPtr;
  end else ReverseMatrix:= nil;
end;



function MultipleMatrixOnNumber(MPtr : MatrixPtr;Number : MatrixElement) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j     : byte;
begin
  if MPtr <> nil then with MPtr^ do begin
    TempPtr:= CreateMatrix(MatrixRow,MatrixCol);
    if TempPtr = nil then begin
      MultipleMatrixOnNumber:= nil;
      Exit;
    end;
    for i:= 1 to MatrixRow do
      for j:= 1 to MatrixCol do
        SetMatrixElement(TempPtr,i,j,GetMatrixElement(MPtr,i,j)*Number);
    MultipleMatrixOnNumber:= TempPtr;
  end else MultipleMatrixOnNumber:= nil;
end;



function MultipleMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j,k   : byte;
begin
  if (MPtr1 <>  nil) and (MPtr2 <> nil) then begin
    TempPtr:= CreateMatrix(MPtr1^.MatrixRow,MPtr2^.MatrixCol);
    if TempPtr = nil then begin
```

```
    MultipleMatrixOnMatrix:= nil;
    Exit;
   end;
   for i:= 1 to TempPtr^.MatrixRow do
    for j:= 1 to TempPtr^.MatrixCol do
     for k:= 1 to MPtr1^.MatrixCol do
      SetMatrixElement(TempPtr,i,j,GetMatrixElement(TempPtr,i,j)+
       GetMatrixElement(MPtr1,i,k)*GetMatrixElement(MPtr2,k,j));
   MultipleMatrixOnMatrix:= TempPtr;
  end else MultipleMatrixOnMatrix:= nil;
end;



function AddMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j,k   : byte;
begin
  if (MPtr1 <>  nil) and (MPtr2 <> nil) then begin
   TempPtr:= CreateMatrix(MPtr1^.MatrixRow,MPtr2^.MatrixCol);
   if TempPtr = nil then begin
    AddMatrixOnMatrix:= nil;
    Exit;
   end;
   for i:= 1 to TempPtr^.MatrixRow do
    for j:= 1 to TempPtr^.MatrixCol do
     SetMatrixElement(TempPtr,i,j,GetMatrixElement(Mptr1,i,j)+
      GetMatrixElement(MPtr2,i,j));
   AddMatrixOnMatrix:= TempPtr;
  end else AddMatrixOnMatrix:= nil;
end;
```

```
function SubMatrixOnMatrix(MPtr1,MPtr2 : MatrixPtr) : MatrixPtr;
var
  TempPtr : MatrixPtr;
  i,j,k   : byte;
begin
  if (MPtr1 <>  nil) and (MPtr2 <> nil) then begin
    TempPtr:= CreateMatrix(MPtr1^.MatrixRow,MPtr2^.MatrixCol);
    if TempPtr = nil then begin
      SubMatrixOnMatrix:= nil;
      Exit;
    end;
    for i:= 1 to TempPtr^.MatrixRow do
      for j:= 1 to TempPtr^.MatrixCol do
        SetMatrixElement(TempPtr,i,j,GetMatrixElement(MPtr1,i,j)-
          GetMatrixElement(MPtr2,i,j));
    SubMatrixOnMatrix:= TempPtr;
  end else SubMatrixOnMatrix:= nil;
end;



function GausseMethodMatrix(MPtr,VPtr : MatrixPtr;var LPtr,UPtr,BPtr : MatrixPtr) :
MatrixPtr;
var
  TempPtr  : MatrixPtr;
  TempVPtr : MatrixPtr;
  TempLPtr : MatrixPtr;
  TempUPtr : MatrixPtr;
  XSum     : MatrixElement;
  i,j,k    : byte;
begin
  if (MPtr <> nil) and (VPtr <> nil) then begin
```

```
TempUPtr:= CloneMatrix(MPtr);
if TempUPtr = nil then begin
  GausseMethodMatrix:= nil;
  Exit;
end;
TempLPtr:= CreateMatrix(MPtr^.MatrixRow,MPtr^.MatrixCol);
if TempLPtr = nil then begin
  DeleteMatrix(TempUPtr);
  GausseMethodMatrix:= nil;
  Exit;
end;
TempVPtr:= CloneMatrix(VPtr);
if TempVPtr = nil then begin
  DeleteMatrix(TempLPtr);
  DeleteMatrix(TempUPtr);
  GausseMethodMatrix:= nil;
  Exit;
end;
TempPtr:= CreateMatrix(MPtr^.MatrixRow,1);
if TempPtr = nil then begin
  DeleteMatrix(TempVPtr);
  DeleteMatrix(TempLPtr);
  DeleteMatrix(TempUPtr);
  GausseMethodMatrix:= nil;
  Exit;
end;

for j:= 1 to MPtr^.MatrixCol-1 do begin
  SetMatrixElement(TempLPtr,j,j,1);
  for i:= j+1 to MPtr^.MatrixRow do begin
    SetMatrixElement(TempLPtr,i,j,GetMatrixElement(TempUPtr,i,j)/
      GetMatrixElement(TempUPtr,j,j));
```

```
      for k:= j to MPtr^.MatrixCol do begin
        SetMatrixElement(TempUPtr,i,k,GetMatrixElement(TempUPtr,i,k)-
          GetMatrixElement(TempLPtr,i,j)*GetMatrixElement(TempUPtr,j,k));
      end;
      SetMatrixElement(TempVPtr,i,1,GetMatrixElement(TempVPtr,i,1)-
        GetMatrixElement(TempLPtr,i,j)*GetMatrixElement(TempVPtr,j,1));
    end;
  end;

  SetMatrixElement(TempLPtr,TempLPtr^.MatrixRow,TempLPtr^.MatrixCol,1);
  SetMatrixElement(TempPtr,TempPtr^.MatrixRow,1,
    GetMatrixElement(TempVPtr,TempVPtr^.MatrixRow,1)/
    GetMatrixElement(TempUPtr,TempUPtr^.MatrixRow,TempUPtr^.MatrixCol));

  for j:= MPtr^.MatrixCol-1 downto 1 do begin
    XSum:= 0;
    for k:= j+1 to MPtr^.MatrixCol do
      XSum:= XSum+GetMatrixElement(TempUPtr,j,k)*
        GetMatrixElement(TempPtr,k,1);
    SetMatrixElement(TempPtr,j,1,(GetMatrixElement(TempVPtr,j,1)-XSum)/
      GetMatrixElement(TempUPtr,j,j));
  end;
  LPtr:= TempLPtr;
  UPtr:= TempUPtr;
  BPtr:= TempVPtr;
  GausseMethodMatrix:= TempPtr;
 end else GausseMethodMatrix:= nil;
end;

end.
```

3. pielikums

Programmas kods (Virknes salidzinašana) (Kompilators Delphi 7)

Funkcijas ConvertToBinary, CompareBuffer, ConvertToNumber – paņemtas no www.delphikingdom.com , bez izmaiņam. Vel tiek ļietota biblioteka lai nodorboties ar lieliem skaitļiem.

```
function MSecToTime(mSec: Int64): string;
var
  dt : TDateTime;
begin
   dt := mSec / MSecsPerSec / SecsPerDay;
   Result := Format('%d* %s', [Trunc(dt), FormatDateTime('ss.zzz', Frac(dt))]) ;
end;
```

Programmas kods (Nulles Polinoms) (Kompilators Delphi 7)

```
program PolyNom;

{$APPTYPE CONSOLE}

uses
  SysUtils,Math,Dialogs,
  Windows; //time evaluate

  function MSecToTime(mSec: Int64): string;
var
  dt : TDateTime;
begin
   dt := mSec / MSecsPerSec / SecsPerDay;
   Result := Format('%d* %s', [Trunc(dt), FormatDateTime('ss.zzz', Frac(dt))]) ;
end;
  function AnotherSign(a : integer) : integer;
  begin
    if a = 1
    then
      result := 0
    else
      result := 1;
  end;
type Tpolinom = record  //Single part of polynom like + (4a*a)
  PSign : integer; // + or -
  PKoef : integer;   // a
  PLetNum : integer; // 4
  PPower : integer; // ^2
  isempty : integer; //0 = 0 1 = something else
```

```
end;
var
  TPower : integer;
  TSign : integer;
  LetKoef : array[1..5] of string;
  icnt,jcnt,MainCycle : integer;
  genpol : integer;
  Wpower : integer;
  Wkoef : integer;
  Tletnum : integer;
  Summa : real;
  polinomSize : integer;
  PolinomStruct : array[1..10000] of Tpolinom; //polinom (+ 2a*a) + 6*b*b*b i.t.d
  PolinomStruct_DETERMINET : array[1..10000] of Tpolinom; //polinom (+ 2a*a) + 6*b*b*b
i.t.d
  PolinomStruct_RANDOMNESS : array[1..10000] of Tpolinom; //polinom (+ 2a*a) + 6*b*b*b
i.t.d
  abcxy : array[1..5] of real;
  PolinomElem : Tpolinom;
  Result : integer;
  Sconstant : integer;

  startTick, endTick : Int64;
  DetFail : TextFile;
  VrbFail : TextFile;
  DetFailAtb : TextFile;
  VrbFailAtb : TextFile;

begin
  randomize; //generate letter
  LetKoef[1] := 'a';
  LetKoef[2] := 'b';
  LetKoef[3] := 'c';
```

```
  LetKoef[4] := 'x';
  LetKoef[5] := 'y';


  AssignFile(DetFail , 'Output1.txt');
  ReWrite(DetFail);
  AssignFile(VrbFail, 'Output2.txt');
  ReWrite(VrbFail);
  AssignFile(DetFailAtb, 'Output3.txt');
  ReWrite(DetFailAtb);
  AssignFile(VrbFailAtb, 'Output4.txt');
  ReWrite(VrbFailAtb);

   Sconstant := 5 * 10 * 2; //2dn, where d - 10(power) n - variable count = 5
   Summa := 0;
 PolinomSize := 0;
TRY
for MainCycle := 1 to 10000
do
begin
  PolinomSize := PolinomSize + 1;
  // Generate Polynom.
    if PolinomSize mod 2 <> 0  //IS NOT NULL
    then
    begin
     for genpol := 1 to PolinomSize
     do
     begin
      Wpower := random(10) + 1; //random 0 <= < X   10
      Wkoef := random(5) + 1;  //a..x 5
      TSign := random(2); //0 or 1 - or +
      TletNum := Random(1001) + 1;
      PolinomElem.PSign := TSign;
```

```
  PolinomElem.PKoef := Wkoef;
  PolinomElem.PLetNum := TletNum;
  PolinomElem.PPower := Wpower;
  PolinomElem.isempty := 1;
  PolinomStruct[genpol] := PolinomElem;
  PolinomStruct_DETERMINET[genpol] := PolinomElem;
  PolinomStruct_RANDOMNESS[genpol] := PolinomElem;
 end;
end
else  //GENERATE NULL POLINOM
begin
  genpol := 1;
  while(genpol <= PolinomSize)
  do
  begin
    Wpower := random(10) + 1; //random 0 <= < X   10
    Wkoef := random(5) + 1;  //a..x 5
    TSign := random(2); //0 or 1 - or +
    TletNum := Random(1001) + 1;
    PolinomElem.PSign := TSign;
    PolinomElem.PKoef := Wkoef;
    PolinomElem.PLetNum := TletNum;
    PolinomElem.PPower := Wpower;
    PolinomElem.isempty := 1;
    PolinomStruct[genpol] := PolinomElem;

  PolinomStruct_DETERMINET[genpol] := PolinomElem;
  PolinomStruct_RANDOMNESS[genpol] := PolinomElem;
  //Symetric monom
  PolinomStruct_DETERMINET[genpol+1] := PolinomElem;
  PolinomStruct_RANDOMNESS[genpol+1] := PolinomElem;
  //Change sign of the monom
```

```
    PolinomStruct_DETERMINET[genpol+1].PSign :=
AnotherSign(PolinomStruct_DETERMINET[genpol+1].PSign);
    PolinomStruct_RANDOMNESS[genpol+1].PSign :=
AnotherSign(PolinomStruct_RANDOMNESS[genpol+1].PSign);



    genpol := genpol + 2; //for +1 we have another;
    end;
  end;
   // writeln('----------------------------------');
 {
  for icnt := 1 to PolinomSize do
   begin

   if  PolinomStruct_DETERMINET[jcnt].PSign = 0
   then
    write(' - ' +  intToStr(PolinomStruct_DETERMINET[icnt].PLetNum) +
LetKoef[PolinomStruct_DETERMINET[icnt].PKoef] + '^' +
intToStr(PolinomStruct_DETERMINET[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ')
   else
    write(' + ' +  intToStr(PolinomStruct_DETERMINET[icnt].PLetNum) +
LetKoef[PolinomStruct_DETERMINET[icnt].PKoef] + '^' +
intToStr(PolinomStruct_DETERMINET[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ');
  end;
  writeln('----------------------------------');
  writeln;
  writeln;
  writeln;
  writeln;
 }
  Summa := 0;
```

```
{ for jcnt := 1 to PolinomSize
  do
  begin

   if  PolinomStruct[jcnt].PSign = 0
   then
     write(' - ' +  intToStr(PolinomStruct[jcnt].PLetNum) + PolinomStruct[jcnt].PKoef + '^' +
intToStr(PolinomStruct[jcnt].Ppower) + ' [' + intToStr(PolinomStruct[jcnt].isempty) + '] ')
   else
     write(' + ' +  intToStr(PolinomStruct[jcnt].PLetNum) + PolinomStruct[jcnt].PKoef + '^' +
intToStr(PolinomStruct[jcnt].Ppower) + ' [' + intToStr(PolinomStruct[jcnt].isempty) + '] ');
  end;
  }

  //Deterministic Algorithm
  StartTick := GetTickCount;

  for icnt := 1 to PolinomSize
  do
  begin
    if PolinomStruct_DETERMINET[icnt].isempty = 1 //exists
    then
    begin
      for jcnt := icnt + 1 to PolinomSize
      do
      begin
       if PolinomStruct_DETERMINET[icnt].isempty = 1 //exists -> check
       then
       begin
         if (PolinomStruct_DETERMINET[icnt].PKoef =
PolinomStruct_DETERMINET[jcnt].PKoef) AND
```

```
            (PolinomStruct_DETERMINET[icnt].PPower =
PolinomStruct_DETERMINET[jcnt].PPower)
            then //SUMM
            begin
              PolinomStruct_DETERMINET[icnt].isempty := 0;
              if PolinomStruct_DETERMINET[icnt].PSign = 1 // +
              then
              begin
               if PolinomStruct_DETERMINET[jcnt].PSign = 1 // + +
               then
                 PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum + PolinomStruct_DETERMINET[icnt].PLetNum
                 else  //Second polinom is -
                 if PolinomStruct_DETERMINET[icnt].PLetNum <
PolinomStruct_DETERMINET[jcnt].PLetNum //second polinom >
                 then
                 begin
                   PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum - PolinomStruct_DETERMINET[icnt].PLetNum
                 end
                 else //second polinom <
                 begin
                   PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[icnt].PLetNum - PolinomStruct_DETERMINET[jcnt].PLetNum;
                   PolinomStruct_DETERMINET[jcnt].PSign := 1; //become +
                 end;

              end // if PolinomStruct_DETERMINET[icnt].PSign
              else //sign = 0 = -
              begin
                if PolinomStruct_DETERMINET[jcnt].PSign = 0 // + +
               then
```

```
                PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum + PolinomStruct_DETERMINET[icnt].PLetNum
                else  //Second polinom is +
                if PolinomStruct_DETERMINET[icnt].PLetNum <
PolinomStruct_DETERMINET[jcnt].PLetNum //second polinom >
                then
                begin
                  PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[jcnt].PLetNum - PolinomStruct_DETERMINET[icnt].PLetNum
                end
                else //second polinom <
                begin
                  PolinomStruct_DETERMINET[jcnt].PLetNum :=
PolinomStruct_DETERMINET[icnt].PLetNum - PolinomStruct_DETERMINET[jcnt].PLetNum;
                  PolinomStruct_DETERMINET[jcnt].PSign := 0; //become +
                end;
              end;
            end;
          end; //End Chek is empty or not second polinom
        end; //All polinom througth
      end //end check existence
      else
      begin
        //nothing get next()
      end;
    end; //end cicle

   //WRITELN;
   //WRITELN;
    for jcnt := 1 to PolinomSize
    do
    begin
```

```
    if (PolinomStruct_DETERMINET[jcnt].isempty <> 0) AND
(PolinomStruct_DETERMINET[jcnt].PLetNum <> 0)
    then
    begin
      result := 0 //Is not NULL
    end
    else
    begin
      result := 1 //IS NULL
    end;
   END;
 {
   if (result = 0) and (PolinomSize mod 2 = 0) //chk
   then
   begin
    for icnt := 1 to PolinomSize do
    begin

    if  PolinomStruct_DETERMINET[icnt].PSign = 0
    then
      write(' - ' +  intToStr(PolinomStruct_DETERMINET[icnt].PLetNum) +
LetKoef[PolinomStruct_DETERMINET[icnt].PKoef] + '^' +
intToStr(PolinomStruct_DETERMINET[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ')
    else
      write(' + ' +  intToStr(PolinomStruct_DETERMINET[icnt].PLetNum) +
LetKoef[PolinomStruct_DETERMINET[icnt].PKoef] + '^' +
intToStr(PolinomStruct_DETERMINET[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ');
   end;
   writeln;
   writeln;
   writeln;
```

```
  writeln;
  writeln;
  end;
 }


  endTick := GetTickCount;
  writeln(DetFail,PChar(MSecToTime(StartTick - EndTick)));
  writeln(DetFailAtb,intToStr(result));

 {
   if  PolinomStruct_DETERMINET[jcnt].PSign = 0
   then
     write(' - ' +  intToStr(PolinomStruct_DETERMINET[jcnt].PLetNum) +
PolinomStruct_DETERMINET[jcnt].PKoef + '^' +
intToStr(PolinomStruct_DETERMINET[jcnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ')
   else
     write(' + ' +  intToStr(PolinomStruct_DETERMINET[jcnt].PLetNum) +
PolinomStruct_DETERMINET[jcnt].PKoef + '^' +
intToStr(PolinomStruct_DETERMINET[jcnt].Ppower) + ' [' +
intToStr(PolinomStruct_DETERMINET[jcnt].isempty) + '] ');
  end;
 }
  //Randomness Algorithm

  Summa := 0;
  StartTick := GetTickCount;
  //GENERATE
  abcxy[1] := (random(10) + 1)/Round(10)+1; //Lemma Var pieradit, ka var atrast saknes start
tiem skaitliem
  abcxy[2] := (random(10) + 1)/Round(10)+1; //ja Ir lieli skaitli ~40 tad 40 ^ 10 pakape nepareizi
strada.
```

```
  abcxy[3] := (random(10) + 1)/Round(10)+1;
  abcxy[4] := (random(10) + 1)/Round(10)+1;
  abcxy[5] := (random(10) + 1)/Round(10)+1;

  for icnt := 1 to PolinomSize
  do
  begin
   if PolinomStruct_RANDOMNESS[icnt].PSign = 1
   then
   begin
    Summa := Summa +
    ROUND(PolinomStruct_RANDOMNESS[icnt].PLetNum *
Power(abcxy[PolinomStruct_RANDOMNESS[icnt].PKoef],PolinomStruct_RANDOMNESS[icn
t].PPower))
   end
   else
   begin
    Summa := Summa -
    ROUND(PolinomStruct_RANDOMNESS[icnt].PLetNum *
Power(abcxy[PolinomStruct_RANDOMNESS[icnt].PKoef],PolinomStruct_RANDOMNESS[icn
t].PPower));
   end;
  end;

  if Summa <> 0 then
  begin
   result := 0; //Is not NULL
  end
   else
   result := 1; //IS NULL

  endTick := GetTickCount;
  writeln(VrbFail,PChar(MSecToTime(StartTick - EndTick)));
```

```
  writeln(VrbFailAtb,intToStr(result));

 {
  if (result = 0) and (PolinomSize mod 2 = 0) //chk
  then
  begin
   for icnt := 1 to PolinomSize do
   begin

   if  PolinomStruct_RANDOMNESS[icnt].PSign = 0
   then
     write(' - ' +  intToStr(PolinomStruct_RANDOMNESS[icnt].PLetNum) +
LetKoef[PolinomStruct_RANDOMNESS[icnt].PKoef] + '^' +
intToStr(PolinomStruct_RANDOMNESS[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_RANDOMNESS[icnt].isempty) + '] ')
   else
     write(' + ' +  intToStr(PolinomStruct_RANDOMNESS[icnt].PLetNum) +
LetKoef[PolinomStruct_RANDOMNESS[icnt].PKoef] + '^' +
intToStr(PolinomStruct_RANDOMNESS[icnt].Ppower) + ' [' +
intToStr(PolinomStruct_RANDOMNESS[icnt].isempty) + '] ');
  end;
  writeln;
  writeln;
  writeln;
  writeln(FloatToStr(abcxy[1]));
  writeln(FloatToStr(abcxy[2]));
  writeln(FloatToStr(abcxy[3]));
  writeln(FloatToStr(abcxy[4]));
  writeln(FloatToStr(abcxy[5]));

  writeln;

  write(FloatToStr(Summa));
```

```
   writeln;
   writeln;
   end;
 }


  { TODO -oUser -cConsole Main : Insert code here }


end;  //END MAIN CYCLE
EXCEPT
on e:exception
do
begin
  showmessage(e.Message);
end;
end;

  CloseFile(DetFail);
  CloseFile(VrbFail);
  CloseFile(DetFailAtb);
  CloseFile(VrbFailAtb);
  readln;
end.
```

5. Pielikums

Dokumentārās lapas forma

Bakalaura darbs

_____

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto bakalaura darbu un atzīstu to par **piemērotu/nepiemērotu** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu bakalaura studiju programmas gala pārbaudījuma komisijas sēdē.

Darba vadītājs(-ja): _____
(Vadītāja paraksts)

Darbs iesniegts Datorikas fakultātē _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.
Metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____

Darbs aizstāvēts bakalaura darbu gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____, vērtējums _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)