VILNIAUS PEDAGOGINIS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

VALDAS AMBRAZIŪNAS

DIDELIŲ DUOMENŲ SEKŲ ANALIZĖS PROBLEMOS

Magistro baigiamasis darbas

Darbo vadovas
Habilituotas daktaras
Profesorius
Gintautas Dzemyda

Vilnius, 2004

Turinys

1 Įvadas

Tai, ką sunku padaryti, reikia
daryti su didžiausiu atkaklumu.
(Konfucijus)

Darbo tikslas:

- Palyginti asociacinių taisyklių sudarymo algoritmus, išanalizuoti jų tinkamumą praktinių uždavinių sprendimui.

Iškelti uždaviniai:

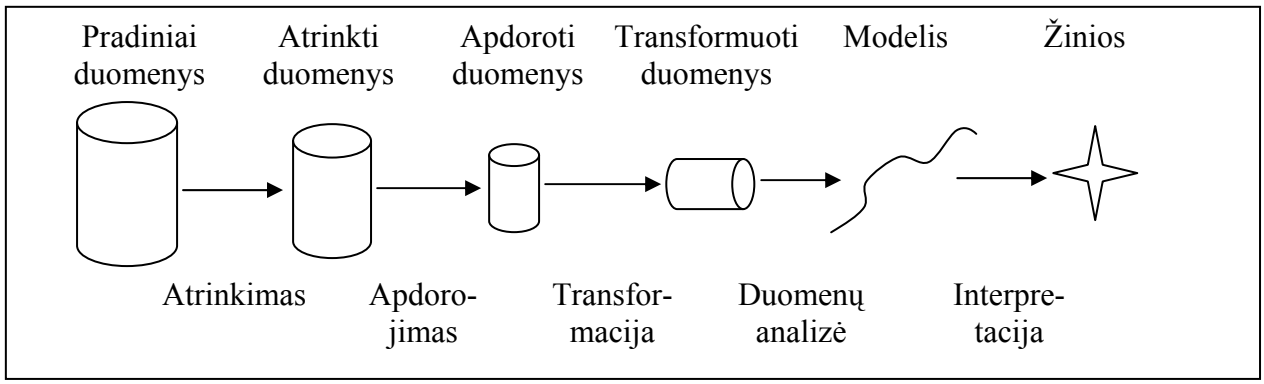
1. Susipažinti su didelių duomenų sekų analize, jos komercine nauda
2. Identifikuoti kylančias asociacinių taisyklių radimo problemas
3. Išanalizuoti žinomus asociacinių taisyklių radimo algoritmus
4. Sukurti techninę bazę algoritmų analizei ir palyginimui
5. Sukurti testinių duomenų aibes
6. Atlikti algoritmų palyginimą, panaudojant sukurtas duomenų aibes
7. Atlikti algoritmų palyginimą, panaudojant realius duomenis (telefono abonentų pokalbių įrašai)
8. Atlikti gautų rezultatų analizę ir pateikti išvadas

Visame pasaulyje duomenų kiekis, kaupiamas duomenų saugyklose, sparčiai didėja. Tuo pačiu vartotojai nori gauti vis sudėtingesnės informacijos iš šių duomenų. Pardavimų vadovo jau nebetenkina paprastas klientų sąrašas, bet jis nori gauti detalią informaciją apie klientų pirkinius, bei ateities prognozes. Paprastos struktūrinės užklausos į duomenų bazes nebegali patenkinti šio vis didėjančio poreikio, todėl pasitelkiama duomenų analizė.

Šiuo metu duomenų analizė yra panašiame etape, kuriame buvo paprastų duomenų užklausų sistemos 1970-taisiais. Per ateinančius dešimt metų, duomenų analizės dėka, prognozuojama padaryti didelius kokybinius šuolius informacinėse verslo palaikymo sistemose. Nors duomenų analizė šiuo metu yra dar „kūdikystės“ stadijoje, per praėjusį dešimtmetį buvo sukurta daug duomenų analizės algoritmų, taikomųjų programų, algoritminių siūlymų.

Duomenų analizėje yra naudojami skirtingi metodai, priklausantys nuo sprendžiamos užduoties. Tai gali būti prognoziniai metodai (klasifikavimas, regresija, laiko eilučių analizė, tiesinė prognozė) arba analitiniai metodai (klasterizavimas, apibendrinimas, asociacinių taisyklių radimas, sekų analizė). Visi duomenų analizės metodai sukuria modelį, kuris geriausiai atvaizduoja analizuojamus duomenis pasirinktu pjūviu.

Pastaruoju metu pradedamas plačiai naudoti „Žinių radimo duomenų bazėse“ terminas. Tai terminas apibūdinantis daugiapakopį procesą, kurio viena iš sudedamųjų dalių yra duomenų analizė:



Ateityje duomenų analizės tobulėjimas leis sukurti naujas duomenų sistemas, galinčias ne tik kaupti, bet ir analizuoti duomenis; galinčias sukurti modelius, kurie remtųsi turimais duomenimis. Šios sistemos galės vykdyti DMQL (Data Mining Query Language) užklausas, kurios skirtingai nuo standartinių SQL (Structured Query Language) užklausų, galės būti naudojamos su hierarchinėmis duomenų struktūromis ir grąžinamas rezultatas bus ne duomenų poaibis, o išanalizuotas modelis.

Šio darbo teorinėje dalyje yra nagrinėjamas asociacinių taisyklių radimo duomenų analizės metodas, bei analizuojami šio metodo algoritmai.

Praktinėje dalyje yra pateikiamas algoritmų sukūrimas, duomenų paruošimas, algoritmų testavimas.

2 Teorinė dalis

2.1 Ryšių tarp duomenų elementų (asociacinės) taisyklės

Vieno produkto pirkimas, kai yra perkamas kitas produktas, nusako asociacinę taisyklę. Asociacinės taisyklės dažnai naudojamos mažmeninės prekybos įmonėse, kad palengvinti įvairių efektyvių reklamos kampanijų organizavimą, padėti racionaliai išnaudoti prekybinį plotą. Nors asociacinės taisyklės daugiausia efekto duoda mažmeninėje prekyboje, tačiau jos gali būti panaudotos ir kitose verslo srityse, pavyzdžiui telekomunikacijų tinklo efektyvaus elementų apkrovimo planavime, bei gedimų prevencijoje. Asociacinės taisyklės nusako ryšius tarp duomenų elementų. Šie ryšiai nėra funkcinės priklausomybės, ar bet kokio tipo koreliaciniai priežastiniai ryšiai. Asociacinės taisyklės nusako bendrą duomenų elementų naudojimą.

Pavyzdžiui:

Maisto mažmeninės prekybos įmonė užregistravo savaitės pardavimų informaciją ir turi duomenis apie kiekvieno pirkėjo pirktas prekes. Ši turima informacija leidžia nustatyti, kokios prekės buvo perkamos kartu. Tarkime 100% atveju, perkant sviestą, buvo perkama duona. Nors sviestas buvo perkamas tik 50% sandorių. Tad įmonė gali priimti sprendimą, kad būtų patogiau vartotojui sviesto ir duonos lentynas patalpinti netoli vienas kito prekybinėje salėje. Kadangi žinoma, jog 100% pirkėjų paėmę sviestą eis link duonos, tai šiame kelyje galima patalpinti kitas norimas parduoti prekes.

Asociacinės taisyklės yra randamos duomenų bazėje, kurią sudaro įrašai apie transakcijas. Viena transakcija turi duomenis apie susijusius duomenų elementus, tarkime apie vieno pirkėjo nupirktas prekes: (Sviestas, Duona, Sūris). T.y. viena transakcija gali būti pirkimo čekis, atmetus iš jo prekių kainas bei nupirktus kiekius.

Duomenų bazės su transakcijomis pavyzdys:

Transakcija	Elementai
T1	Duona, Saldainiai, Sviestas
T2	Duona, Sviestas
T3	Duona, Pienas, Sviestas
T4	Alus, Duona
T5	Alus, Pienas

Šiame pavyzdyje yra pateiktos penkios transakcijos sudarytos iš penkių galimų elementų (Duona, Saldainiai, Sviestas, Alus, Pienas).

Elemento palaikymas nusako procentinę dalį transakcijų, kuriose yra šis elementas. Žemiau esančioje lentelėje yra pateiktos visos galimos elementų kombinacijos ir jų palaikymai bazėje:

Aibe	Palaikymas
Sviestas	60
Saldainiai	20
Saldainiai,Sviestas	20
Pienas	40
Pienas,Sviestas	20
Pienas,Saldainiai	0
Pienas,Saldainiai,Sviestas	0
Duona	80
Duona,Sviestas	60
Duona,Saldainiai	20
Duona,Saldainiai,Sviestas	20
Duona,Pienas	20
Duona,Pienas,Sviestas	20
Duona,Pienas,Saldainiai	0
Duona,Pienas,Saldainiai,Sviestas	0
Alus	40
Alus,Sviestas	0
Alus,Saldainiai	0
Alus,Saldainiai,Sviestas	0
Alus,Pienas	20
Alus,Pienas,Sviestas	0
Alus,Pienas,Saldainiai	0
Alus,Pienas,Saldainiai,Sviestas	0
Alus,Duona	20
Alus,Duona,Sviestas	0
Alus,Duona,Saldainiai	0
Alus,Duona,Saldainiai,Sviestas	0
Alus,Duona,Pienas	0
Alus,Duona,Pienas,Sviestas	0
Alus,Duona,Pienas,Saldainiai	0
Alus,Duona,Pienas,Saldainiai,Sviestas	0

Didėjant elementų skaičiui, galimų elementų kombinacijų kiekis didėja eksponentiškai. Pateiktame pavyzdyje su 5 elementais yra 31 galima elementų kombinacija. Bendru atveju, pažymėjus elementų skaičių m, galimų elementų kombinacijų kiekis yra 2^m-1. Ši, didelio galimų elementų kombinacijų kiekio, problema yra viena iš problemų, kurias turi išspręsti Asociacinių taisyklių radimo algoritmai.

Formalus asociacinių taisyklių aprašymas yra toks:

Turima elementų aibė $I = (I_1, I_2, ..., I_m)$ ir transakcijų duomenų bazė $D = (t_1, t_2, ..., t_n)$, sudaryta iš transakcijų $t_i = (I_{i1}, I_{i2}, \ldots, I_{ik})$, kur $I_{ij} \in I$. Asociacinė taisyklė yra implikacija $X \Rightarrow Y$, kur X ir Y yra elementų kombinacijos $X \subset I$ ir $Y \subset I$, bei $X \cap Y = \varnothing$.

Asociacinės taisyklės $X \Rightarrow Y$ palaikymas (s) yra procentinė duomenų bazės transakcijų dalis, kuriose yra $X \cup Y$.

Asociacinės taisyklės $X \Rightarrow Y$ stiprumas (α) yra duomenų bazės transakcijų, kuriose yra $X \cup Y$, kiekio santykis su duomenų bazės transakcijų, kuriose yra X, kiekiu. T.y. $\alpha = s_{X \Rightarrow Y} / s_X$.

Ieškant asociacinių taisyklių duomenų bazėse, dažniausiai domina ne visi ryšiai, o tik tie, kurie yra svarbūs. Svarbumą nusako du parametrai – palaikymas ir stiprumas.

Lentelėje pateiktos asociacinės taisyklės $X \Rightarrow Y$, kurių stiprumas didesnis už 0 (naudojami duomenys iš aukščiau pateiktos transakcijų duomenų bazės):

Aibe X	Aibe Y	Palaik.	Stiprum
Duona	Alus	20	25
Alus	Duona	20	50
Pienas	Alus	20	50
Alus	Pienas	20	50
Sviestas	Duona,Pienas	20	33.33333
Pienas	Duona,Sviestas	20	50
Pienas,Sviestas	Duona	20	100
Duona	Pienas,Sviestas	20	25
Duona,Sviestas	Pienas	20	33.33333
Duona,Pienas	Sviestas	20	100
Pienas	Duona	20	50
Duona	Pienas	20	25
Sviestas	Duona,Saldainiai	20	33.33333
Saldainiai	Duona,Sviestas	20	100
Saldainiai,Sviestas	Duona	20	100
Duona	Saldainiai,Sviestas	20	25
Duona,Sviestas	Saldainiai	20	33.33333
Duona,Saldainiai	Sviestas	20	100
Saldainiai	Duona	20	100
Duona	Saldainiai	20	25
Sviestas	Duona	60	100
Duona	Sviestas	60	75
Sviestas	Pienas	20	33.33333
Pienas	Sviestas	20	50
Sviestas	Saldainiai	20	33.33333
Saldainiai	Sviestas	20	100

Duomenys parodo, kad 100% perkančiųjų saldainius, perka ir duona, tačiau, palyginus su bendru įrašu kiekiu, tokių transakcijų yra tik 20%. Nusakant svarbumą dviem parametrais s = 30% ir α = 50% (t.y. domina tik asociacinės taisyklės, kurių palaikymas >= 30%, bei stiprumas >= 50%), lieka tik dvi taisyklės:

Sviestas $\Rightarrow$ Duona, s = 60%, α = 100%

Duona $\Rightarrow$ Sviestas, s = 60%, α = 75%

Turint elementų aibę $I = (I_1, I_2, ..., I_m)$ ir transakcijų duomenų bazę $D = (t_1, t_2, ..., t_n)$, sudarytą iš transakcijų $t_i = (I_{i1}, I_{i2}, \ldots, I_{ik})$, kur $I_{ij} \in I$, asociacinių taisyklių radimo algoritmų uždavinys yra rasti implikacijas $X \Rightarrow Y$ su užduotu minimaliu palaikymu ir stiprumu.

Asociacinių taisyklių radimo algoritmų efektyvumas apsprendžiamas dviem dydžiais:

- duomenų bazės nuskaitymo kartų skaičiumi (nes realios transakcijų duomenų bazės yra labai didelės apimties);
- generuojamu elementų kombinacijų kiekiu (nes, kaip pateikta aukščiau, galimų elementų kombinacijų kiekis didėja eksponentiškai, didėjant elementų skaičiui);

2.2 Didelės elementų aibės

Asociacinių taisyklių radimo uždavinį galima padalinti į dvi dalis:

1. Rasti dideles elementų aibes.
2. Suformuoti asociacines taisykles iš didelių elementų aibių.

Didelių elementų aibių aibė $L = (l_1, l_2, ...,l_n)$ yra sudaryta iš elementų aibių l_i, kurių palaikymas yra didesnis už užduotąjį s.

Kai rastos visos didelės elementų aibės, tai visų, užduotą minimalų palaikymą tenkinančių, asociacinių taisyklių $X \Rightarrow Y$ jungtinės aibės $X \cup Y$ turi būti tarp šių didelių elementų aibių. Bet kuris didelės elementų aibės poaibis yra didelė elementų aibė.

Didelių elementų aibių radimas yra labai daug resursų reikalaujantis uždavinys. Paprasčiausias būdas yra sugeneruoti visas galimas elementų aibes ir patikrinti jas duomenų bazėje. Tačiau galimų elementų aibių skaičius eksponentiškai priklauso nuo elementų skaičiaus ir turint m elementų, galima sugeneruoti $2^m - 1$ skirtingų elementų aibių. Kaip parodyta pavyzdyje aukščiau, kai elementų skaičius m = 5, tai galimų aibių skaičius lygus 31. Tačiau kai elementų skaičius m = 30, tai galimų aibių skaičius išauga iki 1073741823. Dauguma asociacinių taisyklių radimo algoritmų stengiasi sumažinti nagrinėtinų aibių skaičių. Šios nagrinėtinos aibės sudaro aibių-kandidatų aibę (C).

Kai visos didelės elementų aibės yra rastos, tada atliekamas asociacinių taisyklių radimas. Taisyklių radimo algoritmas yra tiesinis:

Įvesties duomenys:

D // Transakcijų duomenų bazė

I // Elementai

L // Didelės elementų aibės

s // Palaikymai

α // Stiprumas

Išvesties duomenys:

R // Asociacinės taisyklės

Algoritmas ARGen:

$R = \varnothing$;

kiekvienam $l \in L$ **vykdyti**

kiekvienam $x \subset l$, išskyrus $x = \varnothing$ **vykdyti**

jei $s_l / s_x \geq \alpha$ **tai**

$R = R \cup (x \Rightarrow (1 - x))$;

Iš aukščiau pavyzdžiuose pateiktos transakcijų duomenų bazės, naudojant palaikymą s = 30%, yra rastos didelės elementų aibės L = ((Alus), (Duona), (Pienas), (Sviestas), (Duona, Sviestas)). Paskutinioji aibė (Duona, Sviestas) turi du netuščius poaibius (Duona) ir (Sviestas). Panaudoję pirmąjį (x = (Duona)), gauname:

$s_{(Duona, Sviestas)} / s_{(Duona)} = 60 / 80 = 75\%$,

asociacinės taisyklės Duona $\Rightarrow$ Sviestas stiprumas yra 75%.

Atitinkamai panaudoję antrąjį poaibį (x = (Sviestas)), gauname:

$s_{(Duona, Sviestas)} / s_{(Sviestas)} = 60 / 60 = 100\%$,

asociacinės taisyklės Sviestas $\Rightarrow$ Duona stiprumas yra 100%.

Toliau nagrinėjamų asociacinių taisyklių radimo algoritmų pirminis uždavinys yra efektyviai rasti dideles elementų aibes L.

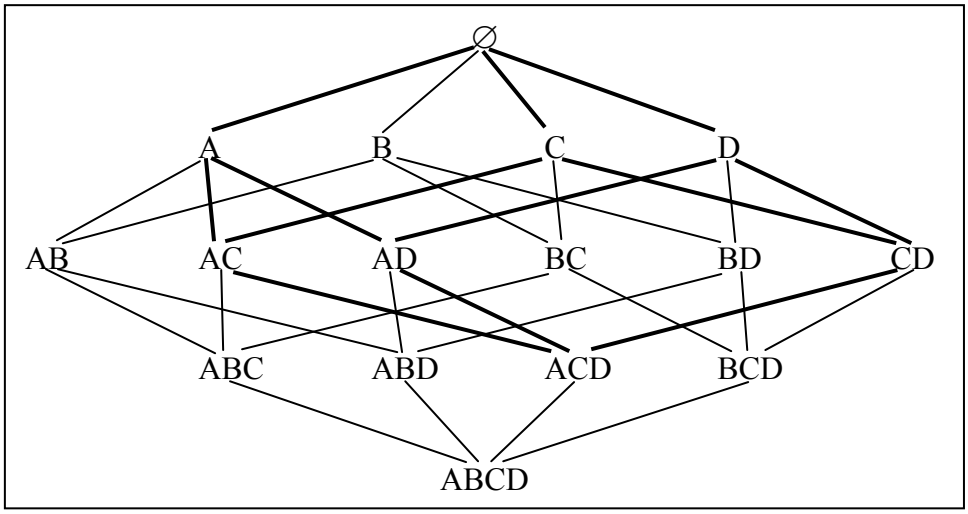
2.3 Apriorinis algoritmas

Apriorinis yra labiausiai paplitęs asociacinių taisyklių radimo algoritmas, naudojamas daugelyje komercinių produktų. Jis naudojasi viena taisykle:

Bet kuris didelės elementų aibės poaibis turi būti didelis.

Todėl didelė elementų aibė yra uždara žemyn, nes jei ji tenkina reikalavimus, tai šiuos reikalavimus tenkins ir bet kuris aibės poaibis. Jei kažkuri elementų aibė nėra didelė, tai ji nėra nagrinėtina, kaip stambesnės elementų aibės poaibis.

Tarkime turime 4 elementus (A, B, C, D). Šių elementų sudaromų poaibių grafas yra:



Linijos grafe parodo ryšį tarp aibių ir jas sudarančių poaibių. Aibė ACD yra sudaryta iš poaibių AC, AD, CD, A, C, D. Ir jei ACD yra didelė, tai visi šie poaibiai yra dideli, o jei nors vienas iš poaibių nėra didelis, tai ACD nėra didelė.

Apriorinis algoritmas generuoja tam tikros eilės (elementų skaičių aibėje) kandidatus ir tikrina transakcijų duomenų bazėje ar šie kandidatai yra dideli. Tik tie kandidatai, kurie buvo dideli, yra naudojami sekančios eilės kandidatų generavime. Tai yra L_i yra naudojama generuojant C_{i+1}. Generuojant i+1 eilės kandidatus, yra apjungiamos aibės iš L_i.

Pavyzdžiui:

Eilė	Kandidatai	Didelės elementų aibės
1	(Alus), (Duona), (Saldainiai), (Pienas), (Sviestas)	(Alus), (Duona), (Pienas), (Sviestas)
2	(Alus, Duona), (Alus, Pienas), (Alus, Sviestas), (Duona, Pienas), (Duona, Sviestas), (Pienas, Sviestas)	(Duona, Sviestas)
3	∅	

Trečios eilės kandidatų nėra, nes yra tik viena antros eilės didelė elementų aibė.

Aukštesnės nei pirmos eilės kandidatų generavimui yra naudojamas algoritmas Apriori-Gen. Pirmos eilės kandidatais naudojamos vienaelementės aibės. Algoritmas Apriori-Gen apjungia i-1 eilės dideles elementų aibes, kurios skiriasi tik vienu elementu:

Įvesties duomenys:

L_{i-1} // i-1 eilės didelės elementų aibės

Išvesties duomenys:

C_i // i eilės kandidatai

Algoritmas Apriori-Gen:

$C_i = \varnothing$;

kiekvienam $I \in L_{i-1}$ **vykdyti**

kiekvienam $J \in L_{i-1}$, išskyrus J = I **vykdyti**

jei i-2 elementai tarp I ir J yra lygūs **tai**

$Ci = Ci \cup (I \cup J)$;

Algoritmas Apriori naudojasi algoritmu Apriori-Gen kandidatų generavimui, bei atlieka kandidatų pasikartojimų transakcijų duomenų bazėje suskaičiavimą:

Įvesties duomenys:

D // Transakcijų duomenų bazė

I // Elementai

s // Reikalaujamas palaikymas

Išvesties duomenys:

L // Didelės elementų aibės

Algoritmas Apriori:

$L = \varnothing$;

k = 0; // k nurodo eilės numerį

$C_1 = I$;

kartoti

k = k + 1;

$L_k = \varnothing$;

kiekvienam $I_i \in C_k$ **vykdyti**

$c_i = 0$; // elementų aibių skaitikliai

kiekvienam $t_j \in D$ **vykdyti**

kiekvienam $I_i \in C_k$ **vykdyti**

jei $I_i \in t_j$ **tai**

$c_i = c_i + 1$;

kiekvienam $I_i \in C_k$ **vykdyti**

jei $c_i \geq (s * |D|)$ **tai**

$L_k = L_k \cup I_i$;

$L = L \cup L_k$;

C_{k+1} = Apriori-Gen (L_k);

kol $C_{k+1} = \varnothing$;

Algoritmas Apriori intensyviai naudojasi transakcijų duomenų baze, todėl laikoma, kad duomenų bazė yra operatyvinėje atmintyje. Maksimalus duomenų bazės peržiūrėjimų skaičius yra vienu didesnis nei didžiausios rastos didelės elementų aibės eilė. Šis didelis duomenų bazės peržiūrėjimo kartų kiekis yra algoritmo trūkumas.

2.4 Mėginio algoritmas

Kai transakcijų duomenų bazės yra didelės, galima naudoti mėginio paėmimo metodus. Algoritmas, paimantis mėginį potencialių didelių elementų aibių radimui, leidžia sumažinti duomenų bazės praėjimų skaičių iki dviejų arba net vieno – geriausiu atveju. Pradžioje yra paimamas duomenų bazės mėginys, telpantis į operatyvinę atmintį, ir yra panaudojamas Apriorinis algoritmas didelių elementų aibių radimui šiame mėginyje. Rastosios elementų aibės yra vadinamos potencialiai didelėmis elementų aibėmis PL. Jos yra naudojamos, atliekant pilną transakcijų duomenų bazės peržiūrą, kaip kandidatai. Papildomi kandidatai yra pridedami atliekant negatyvios ribos funkciją $BD^{-}()$. Bendra kandidatų aibė, atliekant pilną transakcijų duomenų bazės peržiūrą, yra $C = BD^{-}(PL) \cup PL$. Negatyvios ribos funkcija yra algoritmo Apriori-Gen apibendrinimas bet kurios eilės elementų aibėms. $BD^{-}(PL)$ grąžina aibes, kurios nėra PL, bet visi tos aibės poaibiai yra PL.

Pavyzdžiui:

Turimi keturi elementai (A, B, C, D) ir atlikus mėginio analizę yra rastos potencialiai didelės elementų aibės PL = (A, C, D, CD). Tada kandidatai pirmam pilnam transakcijų duomenų bazės praėjimui yra $C = BD^{-}(PL) \cup PL$ = (B, AC, AD) $\cup$ (A, C, D, CD). Kandidatas AC buvo pridėtas todėl, kad abu poaibiai A ir C yra PL. Kandidatas ACD nėra pridėtas todėl, kad nei AC, nei AD nėra PL. B pridėtas, nes B negalima išskaidyti į netuščius poaibius.

Suformavus kandidatus, yra atliekama pirma pilna transakcijų duomenų bazės peržiūra. Jei peržiūros metu visos rastos didelės elementų aibės L priklauso PL, tai laikoma, jog visos galimos didelės elementų aibės yra rastos ir antra peržiūra nėra reikalinga. Jei tarp L buvo tokių aibių, kurios nepriklauso L, bet priklauso $BD^{-}(PL)$, tai antra peržiūra yra reikalinga, nes dar potencialiai yra kitų nepatikrintų kandidatų. Turimos keturios elementų aibių grupės:

1. Elementų aibės, kurios žinomai yra didelės
2. Elementų aibės, kurios žinomai yra mažos
3. Elementų aibės, kurios yra $BD^{-}(PL)$ buferyje
4. Kitos dar neištirtos elementų aibės

Negatyvios ribos funkcija $BD^{-}(PL)$ veikia kaip elementų aibių buferis tarp žinomai didelių elementų aibių ir kitų dar neištirtų elementų aibių. $BD^{-}(PL)$ nusako mažiausią poaibį elementų aibių, kurios gali būti didelės. Jei pirmos pilnos transakcijų duomenų bazės peržiūros metu nebuvo rastos didelės elementų aibės buferyje $BD^{-}(PL)$, tai tikrai didelių elementų aibių nėra tarp kitų dar neištirtų elementų aibių.

Antros pilnos transakcijų duomenų bazės peržiūros metu tikrinami kiti kandidatai, kurių parinkimas turi užtikrinti, jog visos didelės elementų aibės bus rastos. Tam yra atliekamas pakartotinis negatyvios ribos funkcijos $BD^{-}()$ panaudojimas jau rastoms didelėms elementų aibėms, kol nebesugeneruojama naujų elementų aibių. Šio proceso metu sugeneruojama daug kandidatų, tačiau tai užtikrina, kad antros pilnos transakcijų duomenų bazės peržiūros metu visos didelės elementų aibės bus rastos.

Mėginio paėmimo algoritmas Sampling:

Įvesties duomenys:

D // Transakcijų duomenų bazė

I // Elementai

s // Reikalaujamas palaikymas

Išvesties duomenys:

L // Didelės elementų aibės

Algoritmas Sampling:

D_s = D bazės mėginys;

PL = Apriori (I, D_s, smalls);

$C = BD^{-}(PL) \cup PL$;

$L = \varnothing$;

kiekvienam $I_i \in C$ **vykdyti**

$c_i = 0$; // elementų aibių skaitikliai

kiekvienam $t_j \in D$ **vykdyti** // pirmas bazės praėjimas

kiekvienam $I_i \in C$ **vykdyti**

jei $I_i \in t_j$ **tai**

$c_i = c_i + 1$;

kiekvienam $I_i \in C$ **vykdyti**

jei $c_i \geq (s * |D|)$ **tai**

$L = L \cup I_i$;

$ML = (x \mid x \in BD^{-}(PL) \wedge x \in L)$; // Buferyje buvusios aibės

jei $ML \neq \varnothing$ **tai**

$C = L$;

kartoti

$C = BD^{-}(C) \cup C$;

kol C nebebus papildyta naujomis aibėmis;

kiekvienam $I_i \in C$ **vykdyti**

$c_i = 0$;

kiekvienam $t_j \in D$ **vykdyti** // antras bazės praėjimas

kiekvienam $I_i \in C$ **vykdyti**

jei $I_i \in t_j$ **tai**

$c_i = c_i + 1$;

kiekvienam $I_i \in C$ **vykdyti**

jei $c_i \geq (s * |D|)$ **tai**

$L = L \cup I_i$;

Algoritmas mėginio analizei naudoja Apriorinį algoritmą su palaikymo reikšme smalls. Čia smalls gali būti naudojama bet kokia mažesnė reikšmė už s. Palaikymo sumažinimas leidžia mėginyje rasti daugiau didelių elementų aibių, kurios bus naudojamos atliekant pilną transakcijų duomenų bazės peržiūrą, tad didesnė tikimybė, kad visos didelės elementų aibės bus rastos pirmos peržiūros metu.

Pavyzdžiui:

Duomenų bazės mėginį sudaro dvi transakcijos D_s = ((Duona, Saldainiai, Sviestas), (Duona, Sviestas)). Reikalaujamas palaikymas s = 40%. Mėginio analizei naudojamas palaikymas smalls = 20%, t.y. aibė bus didelė, jei ji pasikartos ne mažiau kaip 0,2 x 2 transakcijose (bent vienoje iš dviejų mėginio transakcijų). Atlikus Apriorinį algoritmą su D_s ir smalls gaunamos potencialiai didelės elementų aibės:

PL = ((Duona), (Saldainiai), (Sviestas), (Duona, Saldainiai), (Duona, Sviestas), (Saldainiai, Sviestas), (Duona, Saldainiai, Sviestas))

Įvykdžius negatyvios ribos funkciją gaunama:

$BD^{-}(PL)$ = ((Alus), (Pienas))

Tad kandidatų aibė pirmai pilnai transakcijų duomenų bazės peržiūrai yra:

C = ((Duona), (Saldainiai), (Sviestas), (Duona, Saldainiai), (Duona, Sviestas), (Saldainiai, Sviestas), (Duona, Saldainiai, Sviestas), (Alus), (Pienas))

Tada atliekama peržiūra su s = 40%, kuri grąžina tokias dideles elementų aibes:

L = ((Duona), (Sviestas), (Duona, Saldainiai), (Alus), (Pienas))

Kadangi ML = ((Alus), (Pienas)), tai reikalinga antra bazės peržiūra. Kandidatai jai yra:

$BD^{-}(C)$ = ((Alus, Duona), (Alus, Pienas), (Alus, Sviestas), (Duona, Pienas), (Pienas, Sviestas));

Kartojant negatyvios ribos funkciją gaunami nauji kandidatai:

$BD^{-}(C)$ = ((Alus, Duona, Pienas), (Alus, Duona, Sviestas), (Duona, Pienas, Sviestas), (Alus, Pienas, Sviestas));

Dar kartą kartojant negatyvios ribos funkciją gaunami nauji kandidatai:

$BD^{-}(C)$ = ((Alus, Duona, Pienas, Sviestas));

Tolesnis kartojimas naujų kandidatų nesugeneruoja. Kandidatų aibė antrai pilnai transakcijų duomenų bazės peržiūrai yra:

C = ((Duona), (Sviestas), (Duona, Saldainiai), (Alus), (Pienas), (Alus, Duona), (Alus, Pienas), (Alus, Sviestas), (Duona, Pienas), (Pienas, Sviestas), (Alus, Duona, Pienas), (Alus, Duona, Sviestas), (Duona, Pienas, Sviestas), (Alus, Pienas, Sviestas), (Alus, Duona, Pienas, Sviestas))

2.5 Padalinimo algoritmas

Didelė transakcijų duomenų bazė D gali būti padalinama į p dalių D^1, D^2, ..., D^p. Toks padalinimas leidžia didelių elementų aibių radimą padaryti efektyvesniu, nei naudojant pilną bazę, nes:

- Duomenų bazės dalis gali būti patalpinama operatyvinėje atmintyje.
- Kandidatų, kurie ieškomi duomenų bazės dalyje, aibė potencialiai yra mažesnė, nei kandidatų aibė, naudojama paieškoje pilnoje bazėje.
- Galima panaudoti paralelinius ar paskirstytuosius algoritmus ir skirtingas duomenų bazės dalis gali analizuoti skirtingi kompiuteriai.

Padalinimo algoritmas remiasi taisykle:

Bet kuri didelė elementų aibė visoje bazėje turi būti didelė bent vienoje iš dalių.

Padalinimo panaudojimas leidžia sumažinti duomenų bazės praėjimų skaičių iki dviejų. Pirmo praėjimo metu dalys paeiliui užkraunamos į operatyvinę atmintį ir yra atliekama didelių elementų aibių paieška. L^i pažymi dideles elementų aibes, rastas nagrinėjant D^i transakcijų

duomenų bazės dalį. Didelės elementų aibės gali būti randamos panaudojant Apriorinį algoritmą. Pirmo praėjimo rezultate suformuojama kandidatų aibė $C = L^1 \cup L^2 \cup ... \cup L^p$. Antrojo pilno transakcijų duomenų bazės praėjimo metu tikrinami sugeneruoti kandidatai.

Padalinimo algoritmas Partition:

Įvesties duomenys:

p // Duomenų bazės dalių skaičius

$D = (D^1, D^2, ..., D^p)$ // Transakcijų duomenų bazė

I // Elementai

s // Reikalaujamas palaikymas

Išvesties duomenys:

L // Didelės elementų aibės

Algoritmas Partition:

$C = \varnothing$;

kiekvienam $i \in [1 ... p]$ **vykdyti**

L^i = Apriori (I, D^i, s);

$C = C \cup L^i$;

$L = \varnothing$;

kiekvienam $I_i \in C$ **vykdyti**

$c_i = 0$; // elementų aibių skaitikliai

kiekvienam $t_j \in D$ **vykdyti** // pirmas bazės praėjimas

kiekvienam $I_i \in C$ **vykdyti**

jei $I_i \in t_j$ **tai**

$c_i = c_i + 1$;

kiekvienam $I_i \in C$ **vykdyti**

jei $c_i \geq (s * |D|)$ **tai**

$L = L \cup I_i$;

Pavyzdžiui:

Turimos dvi duomenų bazės dalys:

D^1 = ((Duona, Saldainiai, Sviestas), (Duona, Sviestas))

D^2 = ((Duona, Pienas, Sviestas), (Alus, Duona), (Alus, Pienas))

Šiuo atveju p = 2. Reikalaujamas palaikymas s = 40%, tad pirmoje dalyje aibei pakanka būti vienoje transakcijoje, o antroje dalyje aibe privalo būti ne mažiau kaip dvejose transakcijose. Apriorinio algoritmo pritaikymas duomenų bazės dalims duoda:

L^1 = ((Duona), (Saldainiai), (Sviestas), (Duona, Saldainiai), (Duona, Sviestas), (Saldainiai, Sviestas), (Duona, Saldainiai, Sviestas))

L^2 = ((Duona), (Alus), (Pienas))

Bendras kandidatų sąrašas antram praėjimui yra:

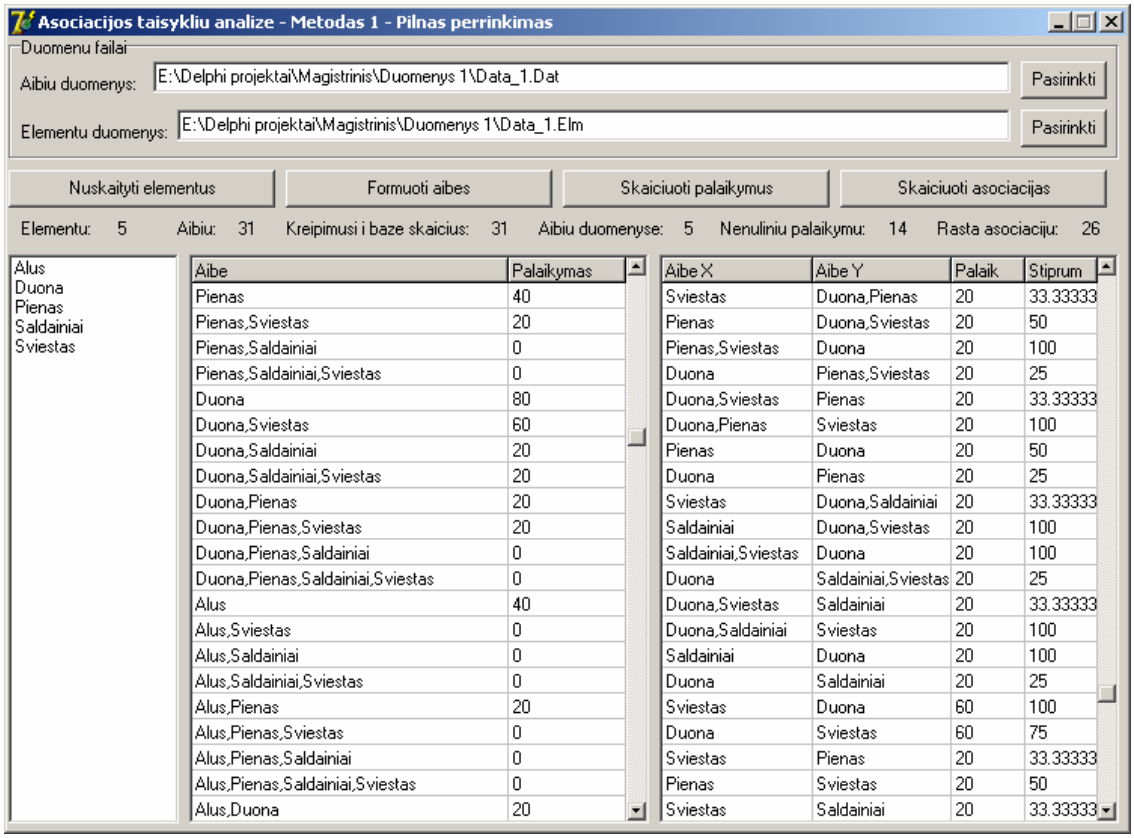
L^1 = ((Duona), (Saldainiai), (Sviestas), (Duona, Saldainiai), (Duona, Sviestas), (Saldainiai, Sviestas), (Duona, Saldainiai, Sviestas), (Alus), (Pienas))

Bazės padalinimo algoritmo veikimas priklauso nuo duomenų pasiskirstymo transakcijų duomenų bazėje. Jei duomenys pasiskirstę tolygiai, tai kiekvienos dalies analizė duos didelių elementų aibių sąrašą artimą galutiniam ir antrame praėjime naudojamas kandidatų sąrašas bus minimalus. Jei duomenys pasiskirstę netolygiai, tai potencialiai galimas didelis neteisingų kandidatų skaičius.

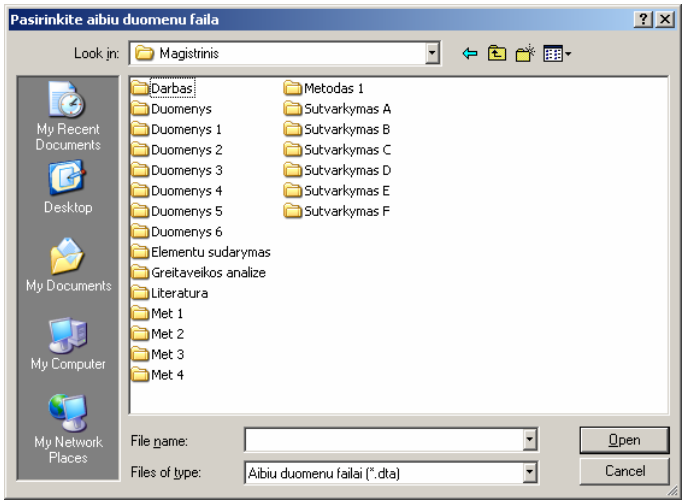
3 Praktinė dalis

3.1 Asociacinių taisyklių radimas pilno perrinkimo būdu

Siekiant geriau išsiaiškinti asociacinių taisyklių radimo metodiką, buvo nuspręsta pradžioje sukurti asociacinių taisyklių radimo programą su grafiniu interfeisu. Ši programa nėra skirta algoritmų greitaveikos patikrinimui. Ji skirta vaizdžiai pateikti analizuojamus duomenis ir tarpinius rezultatus. Programos kūrimui buvo pasirinkta Delphi programavimo kalba. Programos išeities tekstas pateikiamas priede 1. Programa turi patogų vartotojo interfeisą:



Vartotojo patogumui programoje naudojami failų išrinkimo dialogai:



Programoje realizuotas asociacinių taisyklių radimo procesas išskaidytas į 4 etapus:

1. Nuskaitoma elementų aibė
2. Suformuojamos visos įmanomos netuščios ir nepasikartojančios elementų aibės
3. Atliekamas aibių palaikymų suskaičiavimas
4. Randamos asociacinės taisyklės, bei suskaičiuojami jų stiprumai

Šiuos etapus vartotojas valdo atitinkamų mygtukų paspaudimais.

Programa duomenis nuskaito į dinaminę duomenų struktūrą. Šiam tikslui buvo sukurti sekantys duomenų tipai:

- Aibe – tiesinis dinaminis sąrašas, kurio nariai laiko informaciją apie aibei priklausančius elementus
- Palaik – tiesinis dinaminis sąrašas, kurio kiekvienas narys turi nuoroda į aibę, bei suskaičiuotą palaikymo vertę
- Asoc - tiesinis dinaminis sąrašas, kurio kiekvienas narys turi po dvi nuorodas į aibes, bei tų aibių suskaičiuotas palaikymo vertes.

Pradžioje nuskaitomas elementų sąrašas (procedūra Button3Click), kuris patalpinamas į elementų vardų masyvą ElemVard, bei pateikiamas vartotojui kairioje interfeiso lango dalyje.

Antrame etape generuojamos aibės iš nuskaitytųjų elementų (procedūra Button4Click). Aibės yra generuojamos rekurentiškai, įtraukiant ar išmetant po vieną elementą (procedūra ElmGo). Visos sugeneruotos aibės apjungiamos į sąrašą Sp, bei išvedamos vartotojui centrinėje interfeiso lango dalyje.

Trečio etapo metu skaičiuojami palaikymai (procedūra Button5Click). Tam imama kiekviena sugeneruotoji aibė ir skaičiuojamas jos pasikartojimų kiekis duomenų bazėje. Rezultatai vartotojui pateikiami centrinėje interfeiso lango dalyje.

Ketvirtame etape ieškoma asociacinių taisyklių (procedūra Button6Click). Tam formuojama nauja struktūra Sc. Taisyklės randamos skaidant aibes, turinčias daugiau nei vieną elementą, bei turinčias nenulinį palaikymą. Sugeneruotos asociacinės taisyklės pateikiamos vartotojui dešinėje interfeiso lango dalyje.

Šios programos kūrimo metu buvo išsiaiškintas darbas su duomenimis ir įgyta patirtis, kuri buvo panaudota kuriant greitaveikius asociacinių taisyklių radimo algoritmus.

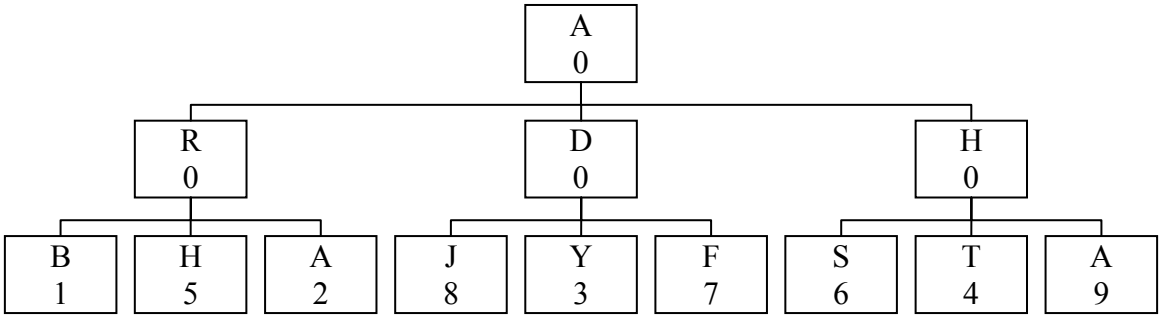
3.2 Asociacinių taisyklių radimo algoritmų bendra dalis

Siekiant užtikrinti asociacinių taisyklių radimo algoritmų greitaveiką, buvo nuspręsta programines algoritmų realizacijas kurti C++ kalba (kompiliatorius Borland C++ 5.5.1 for Win32), kaip komandinės eilutės aplikacijas be grafinio vartotojo interfeiso. Tai leidžia maksimaliai išnaudoti kompiuterio resursus, bei atlikti automatizuotą algoritmų greitaveikos tyrimą. Taip pat buvo nuspręsta, kad skirtingų algoritmų programos turi būti su vienodais duomenų interfeisais.

Išnagrinėjus teorinėje dalyje pateiktus algoritmus buvo pastebėta, kad algoritmai turi daug bendrų dalių, bei naudoja tokias pačias duomenų struktūras, todėl buvo nuspręsta bendrąsias algoritmų dalis iškelti į atskirą modulį. Šis modulis turi talpinti savyje visus naudojamus objektus.

Bendrosios dalies išeities tekstas pateiktas priede 2.

Elementų saugojimui ir darbui su jais, greitaveikos sumetimais, pasirinkta naudoti elementų skaitinius identifikatorius. Šių identifikatorių greitam nustatymui (tai reikalinga atliekant transakcijų duomenų bazės nuskaitymą) sukurtas objektas RaidSar. Objekte realizuota medžio struktūra:



Tad medyje ieškant elemento pavadinimo ARA, gaunamas identifikatorius 2, o pavadinimo AHA – 9. Taip išvengiama didelio kiekio tikrinimų, palyginus su masyvo struktūra.

Elementų vardų medis padarytas dinaminis, tad atitinkamos šakos sukuriamos tik jei yra elementų, turinčių tokius simbolius savo pavadinimuose. Atminties taupymo sumetimais, simbolių, galimų naudoti elementų varduose, aibė apribota iki didžiųjų lotyniškų raidžių, mažųjų lotyniškų raidžių, arabiškų skaitmenų ir pabraukimo simbolio „_“.

Greitam elemento vardo radimui, kai turimas elemento identifikatorius, sukurtas dinaminis elementų vardų masyvo objektas NumSar, kur elemento identifikatorius sutampa su masyvo indeksu.

Šiuos du objektus RaidSar ir NumSar į vieną visumą apjungia objektas Elementai. Šiame objekte papildomai realizuoti metodai naujo elemento įdėjimui, elemento pašalinimui, elemento pavadinimo radimui žinant identifikatorių, elemento identifikatoriaus radimui žinant

elemento pavadinimą, pilno elementų sąrašo nusiskaitymui iš užduoto failo, bei elementų pasikartojimo suskaičiavimui užduotoje transakcijų duomenų bazėje.

Elementų aibės saugojimui atmintyje sukurtas objektas Aibe. Kiekvienas elementas Aibe objekte užima 1 bitą. Binarinis elementų saugojimas įgalina greitai atlikinėti veiksmus su aibėmis – aibių sudėtis, atimtis, daugyba. Šie metodai realizuoti objekte. Papildomai dar sukurti metodai elemento buvimo aibėje patikrinimui, aibių palyginimui, išvedimui į nurodytą failą.

Objektas Palaikymas apibendrina Aibe objektą papildydamas jį atminties ląstele suskaičiuoto palaikymo saugojimui, bei suriša aibes į tiesinį dinaminį sąrašą.

Visų aukščiau paminėtų duomenų struktūrų apibendrinimui sukurtas objektas PalaikymuSk. Šis objektas tarnauja ir kaip elementų aibių palaikymo, bei asociacinių taisyklių radimo ir stiprumo skaičiavimo metodų konteineris. Objekte sukurti metodai Aibių prijungimui, pašalinimui, radimui, aibės palaikymo radimui, mažų aibių pašalinimui, palaikymų skaičiavimui, kito objekto duomenų perėmimui, algoritmas ARGen asociacinių taisyklių radimui.

Mėginio ir Padalinimo algoritmuose naudojamas duomenų bazės dalies patalpinimas operatyvinėje atmintyje, todėl sukurtas objektas VidBaze realizuojantis tiesinį sąrašą, kuriame saugomi transakcijų duomenų bazės elementai. Objektas turi duomenų bloko nusiskaitymo iš užduoto failo metodą.

Sukurtieji objektai yra skirti tik šiam darbui, todėl visi objektų elementai ir metodai yra viešojo tipo. Norint sukurtuosius objektus panaudoti kitiems komerciniams tikslams, rekomenduotina vidinius objektų elementus padaryti privačiais ir palikti atvirus tik interfeisus.

3.3 Algoritmų programinės realizacijos

Turint bendrus visiems algoritmams duomenų objektus, sekančiame etape buvo sukurtos keturių algoritmų programinės realizacijos. Kadangi buvo iškeltas reikalavimas vienodiems interfeisams, tai visos programos priima tokią komandinę eilutę:

<vardas> <baze> <elementai> <rezultatai> <palaikymas> <stiprumas>

<vardas> programos vykdomojo failo pavadinimas

<baze> transakcijų duomenų bazės failo pavadinimas

<elementai> elementų sąrašo failo pavadinimas

<rezultatai> išvedamų rezultatų failo pavadinimas

<palaikymas> pareikalaujamas minimalus palaikymas

<stiprumas> pareikalaujamas minimalus asociacinės taisyklės stiprumas

3.3.1 Pilno perrinkimo algoritmas

Šio algoritmo programos išeities tekstas pateiktas priede 3.

Programoje realizuotas elementų sąrašo nuskaitymas, visų įmanomų elementų aibių suformavimas, transakcijų duomenų bazės skaitymas ir palaikymų skaičiavimas, asociatyvinių taisyklių radimas. Programa nuskaitinėja transakcijų duomenų bazę tik vieną kartą.

Elementų aibių formavimui panaudota rekurentinė funkcija Formuoti.

3.3.2 Apriorinis algoritmas

Šio algoritmo programos išeities tekstas pateiktas priede 4.

Programa nuskaito elementų sąrašą ir suformuoja kandidatus-aibes po vieną elementą. Toliau vykdomas ciklas, kol yra nors vienas kandidatas. Cikle atliekamas transakcijų duomenų bazės praėjimas ir palaikymų suskaičiavimas. Panaudojant Apriori-Gen algoritmą atliekamas naujų kandidatų radimas. Pabaigus ciklą atliekamas asociatyvinių taisyklių radimas.

3.3.3 Mėginio algoritmas

Šio algoritmo programos išeities tekstas pateiktas priede 5.

Programa nuskaito elementų sąrašą ir patalpina dalį transakcijų duomenų bazės į operatyvinę atmintį. Tada su šia dalimi atlieka Apriorinį algoritmą. Gautos aibės ir aibės iš

negatyvios ribos funkcijos naudojamos kaip kandidatai pilnam transakcijų duomenų bazės skanavimui. Jei šio skanavimo metu buvo gautos aibės, priklausiusios negatyvios ribos funkcijos rezultatui, tai atliekamas kandidatų sąrašo išplėtimas ir dar vienas transakcijų duomenų bazės praėjimas. Pabaigoje vykdomas asociatyvinių taisyklių radimas.

3.3.4 Padalinimo algoritmas

Šio algoritmo programos išeities tekstas pateiktas priede 6.

Programa nuskaito elementų sąrašą ir vykdo ciklą. Ciklo metu patalpina eilinę duomenų bazės dalį į atmintį ir atlieka Apriorinį algoritmą. Ciklas baigiamas, kai nuskaitomas paskutinis transakcijų duomenų bazės blokas. Visos sukauptosios elementų aibės naudojamos kaip kandidatai antram transakcijų duomenų bazės praėjimui. Pabaigoje vykdomas asociatyvinių taisyklių radimas.

3.3.5 Elementų sąrašo sudarymas

Visų algoritmų programinės realizacijos naudoja elementų sąrašo failą, todėl duomenų paruošimo fazėje yra reikalingas tokio failo sudarymas. Šią funkciją atliekančios programos išeities tekstas pateiktas priede 7.

Programa skaito visus transakcijų duomenų bazės įrašus ir formuoja pavadinimų medį. Kai pabaigiamas skaitymas, programa įrašo elementų sąrašą į failą, išrūšiuodama elementus abėcėlės tvarka.

3.4 Testinės duomenų aibės

Algoritmų tyrimui reikia testinių transakcijų duomenų bazių. Tam buvo pasirinkta naudoti devynias duomenų aibes:

1. Maža duomenų aibė iš 5 elementų ir 5 įrašų. Aibė pateikta priede 8. Ši aibė naudojama pavyzdžiuose teorinėje algoritmų analizėje.
2. Praplėstas mažos duomenų aibės variantas. Aibę sudaro 11 elementų ir 23 įrašai. Aibė pateikta priede 9.
3. Mažas realių skambučių poaibis. Šią aibę sudaro 21 elementas ir 40 įrašų. Aibė pateikta priede 10.

4. Realių skambučių vienos paros rajonų kodų duomenys. Aibę sudaro 218 elementų ir 42913 įrašų. Aibės dalis pateikta priede 11.
5. Realių skambučių kilmės kodų vienos paros duomenys. Aibę sudaro 65 elementai ir 45069 įrašai. Aibės dalis pateikta priede 12.
6. Realių skambučių paskirties kodų vienos paros duomenys. Aibę sudaro 228 elementai ir 20479 įrašai. Aibės dalis pateikta priede 13.
7. Realių skambučių vienerių metų rajonų kodų duomenys. Aibę sudaro 503 elementai ir 15205852 įrašai. Aibės struktūra identiška 4 variantui.
8. Realių skambučių kilmės kodų vienerių metų duomenys. Aibę sudaro 106 elementai ir 16169534 įrašai. Aibės struktūra identiška 5 variantui.
9. Realių skambučių paskirties kodų vienerių metų duomenys. Aibę sudaro 502 elementai ir 7608725 įrašai. Aibės struktūra identiška 6 variantui.

3.5 Realių duomenų aibių paruošimas

Turima informacija apie 50000 telefono abonentų metų skambučius formate (pateiktas vienas įrašas):

```
8|1|356|2004.05.20|01:16:59|01111011111111111111111111111111|45|E034|??|
Global|852650199|68|00|890155555|0|-1|-1|0|384|01|128|00|55555|??|??|0|
0|1|0|1|0|FF|??|??|0|0|??|0|0|0|0|0|0|00000000000000000000|??|0|0|00|0|
0|00000000000000000000|55555|??|??|0|3|0|25|??
```

Šiuose įrašuose yra daug perteklinės informacijos, todėl sukurta programa įrašų supaprastinimui (programos išeities tekstas pateiktas priede 23). Programa iš įrašų išskiria skambinančiojo ir paskirties abonentų numerius, tad iš aukščiau pateikto įrašo suformuojamas naujas įrašas:

852650199,890155555

Tokie atskirų abonentų įrašai panaudojami trijų tipų duomenų aibių sukūrimui:

1. Skambučių rajonų kodų duomenys, nusakantys skambučių trauką tarp skirtingų rajonų. Programos, atliekančios šį pertvarkymą išeities tekstas pateiktas priede 24.
2. Skambučių kilmės kodų į vieną fizinį numerį sąrašas, nusakantis rajonų kodus, skambinusius į tam tikrą numerį. Programos, atliekančios šį pertvarkymą išeities tekstas pateiktas priede 25.
3. Skambučių paskirties kodų iš vieno fizinio numerio sąrašas, nusakantis rajonų kodus, į kuriuos buvo skambinto iš vieno tam tikro numerio. Programos, atliekančios šį pertvarkymą išeities tekstas pateiktas priede 26.

Šių tipų aibės buvo naudojamos tiek algoritmų greitaveikos tyrime, tiek ir pačių duomenų analizėje.

3.6 Algoritmų veikimo tyrimas

Tiriant algoritmus buvo panaudotas automatizuotas programų paleidimas ir registruojami paleidimo, bei programos darbo pabaigos laikai. Šiam tikslui sukurtas komandinis failas:

```
@Echo Off
  if %1.==. Goto Klaida
  Echo Paleidimo laikas : %Time% > Data_1_Met_%1.Run
  "..\Met %1\Met_%1.Exe" Data_1.Dat Data_1.Elm Data_1_Met_%1.Rez 0.3 0.5
  Echo Pabaigos laikas :  %Time% >> Data_1_Met_%1.Run
  Goto Pabaiga

:Klaida
  Echo Klaida !
  Echo Nenurodytas metodas.

:Pabaiga
```

Tokie komandiniai failai (jie buvo sukurti kiekvienai duomenų aibei atskirai) atlikinėjo nurodytų programų iškvietimą, bei laiko registravimą.

Pilno perrinkimo algoritmas buvo naudojamas tik su pirmomis trejomis duomenų aibėmis, nes didėjant elementų kiekiui, eksponentiškai didėjo šio algoritmo pareikalaujamas operatyviosios atminties kiekis. Testuojant su trečiąja duomenų aibe, turinčia 21 elementą, programa pareikalavo ~900 Mb darbinės atminties.

3.6.1 Teisingumas

Visi algoritmai gavo teisingus rezultatus, t.y. visi algoritmai rado visas dideles elementų aibes. Gautieji algoritmų rezultatai yra identiški visose devyniose duomenų aibėse. Algoritmų sugeneruotos didelės elementų aibės su palaikymais, bei rastos asociatyvinės taisyklės su jų stiprumo įverčiais pateiktos prieduose:

Duomenų aibė	Pareikalautieji parametrai		Priedo numeris
	Palaikymas	Stiprumas	
1	30 %	50 %	14
2	10 %	50 %	15
3	15 %	50 %	16

Duomenų aibė	Pareikalautieji parametrai		Priedo numeris
	Palaikymas	Stiprumas	
4	0,5 %	20 %	17
5	0,5 %	5 %	18
6	0,5 %	5 %	19
7	0,5 %	20 %	20
8	0,5 %	5 %	21
9	0,5 %	5 %	22

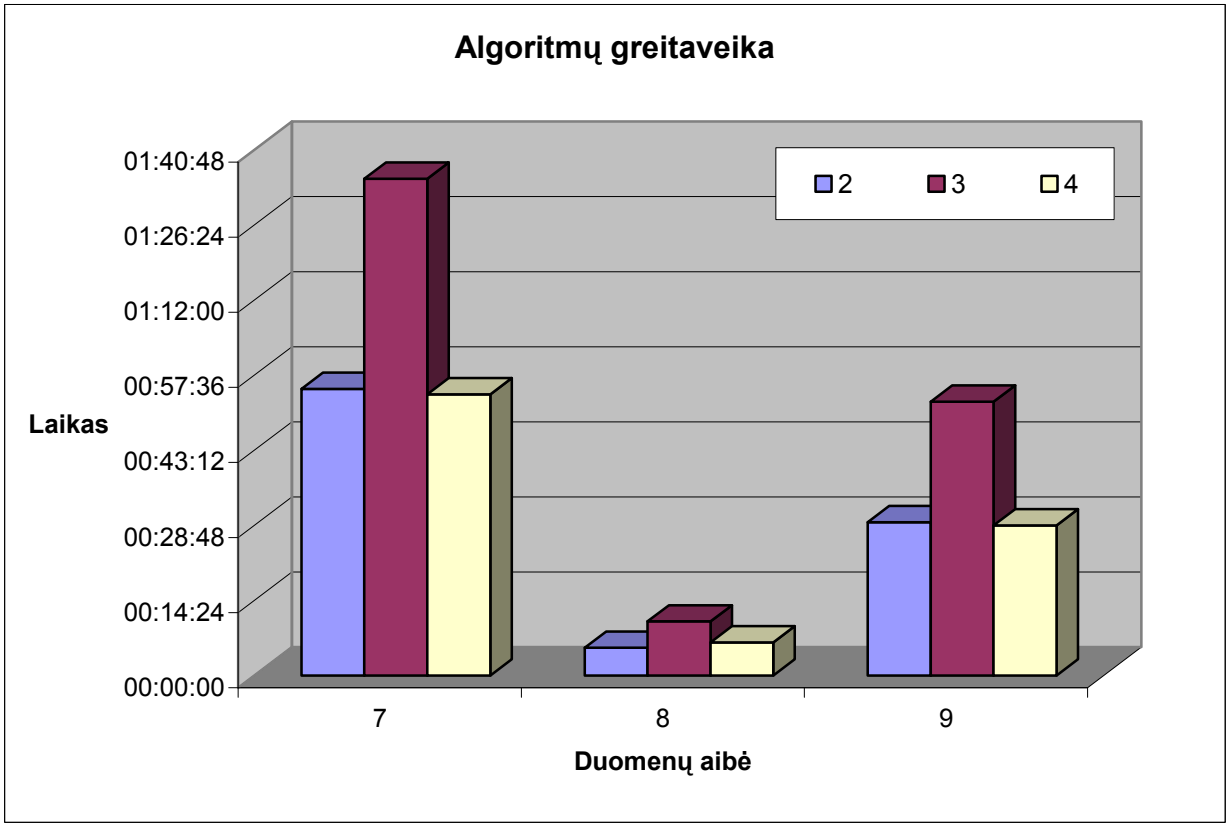
3.6.2 Greitaveika

Užregistruotieji algoritmų veikimo laiko įrašai yra pateikti priede 27. Apibendrinta panaudoto mašininio laiko lentelė atrodo taip:

Duomenys	Algoritmas			
	1 Pilno perrinkimo	2 Apriorinis	3 Mėginio	4 Padalinimo
1	00:00:00	00:00:00	00:00:00	00:00:00
2	00:00:00	00:00:00	00:00:00	00:00:00
3	00:00:09	00:00:00	00:00:00	00:00:00
4		00:00:05	00:00:15	00:00:06
5		00:00:01	00:00:02	00:00:01
6		00:00:03	00:00:06	00:00:03
7		00:54:58	01:35:17	00:53:56
8		00:05:22	00:10:24	00:06:23
9		00:29:24	00:52:31	00:28:47

Kaip matoma, pilno perrinkimo algoritmas, net su sąlyginai maža duomenų aibe 3, veikė 9 sek. Taip įvyko dėl operatyviosios atminties trūkumo ir intensyvaus diskinio „swap“ naudojimo. Tačiau reikia neužmiršti, kad pilno perrinkimo algoritmas visada atlieka tik vieną pilną transakcijų duomenų bazės nuskaitymą, tad jis būtų naudotinas tais atvejais, kai turimas didelis įrašų skaičius, tačiau šiuos įrašus sudarančių elementų kiekis yra nedidelis.

Panagrinėkime detaliau algoritmų laikus su 7, 8 ir 9 duomenų aibėmis:



Šioje histogramoje skaitmenimis 2, 3 ir 4 atitinkamai pažymėti Apriorinis, Mėginio ir Padalinimo algoritmai. Matome, kad Mėginio algoritmas visais atvejais buvo prasčiausias. Tai įvyko todėl, kad atliekant mėginio analizę yra naudojamas mažesnis palaikymo lygis ir yra sugeneruojamas didelis kandidatų kiekis pilnam transakcijų duomenų bazės skanavimui.

Padalinimo algoritmas buvo pranašesnis už Apriorinį, su 7 ir 9 duomenų aibėmis, tačiau su 8 duomenų aibe atsiliko. Tai įvyko todėl, kad 8 duomenų aibėje nėra antros eilės didelių elementų aibių ir Aprioriniam algoritmui nereikėjo antrojo praėjimo. Šis Padalinimo algoritmo pranašumas, matomas 7 ir 9 variantuose, turėtų dar labiau išryškėti, jei būtų sudarinėjamos aukštesnės eilės kandidatų aibės.

3.7 Realių duomenų analizė

Gautieji rezultatai, panaudojant realias telefoninių skambučių duomenų aibes parodo, kad Asociatyvinių ryšių analizė gali būti praktiškai pritaikoma versle.

7 duomenų aibės analizė atskleidė miestų kodus, tarp kurių abonentų vyksta masiškiausi skambučiai. Tai yra vertinga informacija naujam telekomunikacijų rinkos dalyviui. Jei jis norės plėtoti savo veiklą vietovėje, aptarnaujamoje tarpmiestiniu kodu [Kodas 1], tai jam tikslinga turėti tiesioginį sujungimą su kodą [Kodas 2] aptarnaujančia stotimi, nes 78% skambučių, kuriuose dalyvauja [Kodas 1], vyksta su [Kodas 2]. Operatorius šiuo atveju galės

pasiūlyti pigesnius tarifus į minėtąjį kodą, kas turėtų pritraukti tokiomis kryptimis aktyviai skambinančius klientus.

9 duomenų aibės analizė parodė, kad yra asociatyviniai ryšiai tarp kodų, kuriuos renka abonentai. Net 54.4% abonentų, kurie skambina kodais [Kodas 3] ir [Kodas 4], aktyviai skambina ir kodu [Kodas 5]. Tai vėlgi naudinga informacija, kuria galima remtis atliekant telekomunikacijų tinklo dimensijavimą ir optimizavimą.

4 Išvados

Šiame darbe buvo atlikta trijų asociatyvinių algoritmų teorinė analizė, sukurtos keturių algoritmų programinės realizacijos, atlikti jų teisingumo, bei greitaveikos tyrimai. Papildomai patyrinėta asociatyvinių ryšių radimo pritaikomumas realių verslo uždavinių sprendimui.

Algoritmų teisingumo tyrimas parodė, kad:

1. Visi algoritmai rado visas dideles elementų aibes.
2. Gautieji algoritmų rezultatai yra identiški visose devyniose duomenų aibėse.

Algoritmų greitaveikos tyrimas nustatė, kad:

1. Pilno perrinkimo algoritmas yra iš viso nenaudotinas, kai elementų kiekis >20, tačiau jis būtų naudotinas tais atvejais, kai turimas didelis įrašų skaičius, tačiau šiuos įrašus sudarančių elementų kiekis yra nedidelis.
2. Mėginio algoritmas buvo mažiausiai našus, nes atliekant mėginio analizę yra naudojamas mažesnis palaikymo lygis ir yra sugeneruojamas didelis kandidatų kiekis pilnam transakcijų duomenų bazės skanavimui.
3. Padalinimo algoritmas yra pranašiausias kai sudarinėjamos aukštesnės eilės kandidatų aibės.

Asociatyvinių ryšių radimas realiuose verslo duomenyse parodė, kad:

1. Asociatyvinių ryšių analizė gali būti praktiškai pritaikoma versle.
2. Remiantis šios analizės rezultatais galima suformuoti tikslinės klientų grupės pritraukimo kampaniją.

5 Literatūros sąrašas

1. David Hand, Heikki Mannila, Padhraic Smyth. Principles of Data Mining. – Massachusetts: Massachusetts Institute of Technology, 2001.
2. Jiawei Han, Micheline Kamber. Data Mining: Concepts and Techniques. - Morgan Kaufmann Publishers, 2000.
3. Kurt Thearling. An Introduction to Data Mining. - www.thearling.com, dmintro.pdf, 2004.
4. Margaret H. Dunham. Data Mining Introductory and Advanced Topics. – New Jersey: Pearson Education, Inc., 2003.
5. Mehmed Kantardzic. Data Mining: Concepts, Models, Methods and Algorithms. - John Wiley & Sons, 2003.
6. Mohammed J. Zaki, Ching-Tien Ho. Large-Scale Parallel Data Mining. – Berlin: Springer, 2000.
7. Paul J. Lucas. The C++ Programmer‘s Handbook. – New Jersey: AT&T Bell Laboratories, 1992.
8. R. Tidikis, J. S. Pečkaitis, S. Šedbaras. Magistrų baigiamųjų darbų rengimas ir gynimas. – Vilnius: Lietuvos teisės universitetas, 2003.

6 Santrauka

Šio darbo tikslas yra palyginti asociacinių taisyklių sudarymo algoritmus ir išanalizuoti jų tinkamumą praktinių uždavinių sprendimui. Šio tikslo pasiekimui, darbe yra teoriškai išnagrinėti trys didelių elementų aibių radimo algoritmai:

1. Apriorinis algoritmas, kuris remiasi taisykle, kad bet kuris didelės elementų aibės poaibis turi būti didelis. Šis algoritmas intensyviai naudojasi transakcijų duomenų baze – maksimalus duomenų bazės peržiūrėjimų skaičius yra vienu didesnis nei didžiausios rastos didelės elementų aibės eilė.
2. Mėginio algoritmas, paimantis duomenų bazės dalį (mėginį) potencialių didelių elementų aibių radimui. Šis algoritmas leidžia sumažinti duomenų bazės praėjimų skaičių iki dviejų arba net vieno, geriausiu atveju, kai visos didelės elementų aibės nustatomos iš mėginio.
3. Padalinimo algoritmas, kuris analizuoja duomenų bazę dalimis ir remiasi taisykle, kad bet kuri didelė elementų aibė visoje bazėje turi būti didelė bent vienoje iš dalių. Padalinimo panaudojimas leidžia sumažinti duomenų bazės praėjimų skaičių iki dviejų.

Darbo praktinėje dalyje atliekamas algoritmų palyginimas. Tam sukurtos keturių algoritmų (trejų išnagrinėtų teorinėje dalyje ir pilno perrinkimo algoritmo) programinės realizacijos C++ kalba. Siekiant užtikrinti asociacinių taisyklių radimo algoritmų greitaveiką, programos yra komandinės eilutės aplikacijos be grafinio vartotojo interfeiso. Algoritmų palyginimas atliktas tiriant algoritmų darbą su devyniomis testinių duomenų aibėmis, kurių apimtis kinta nuo mažiausios duomenų aibės, iš 5 elementų ir 5 įrašų, iki didžiausios – 503 elementai ir 15205852 įrašai. Algoritmai lyginami dviem aspektais: algoritmų veikimo teisingumas ir algoritmų greitaveika.

Ištyrus algoritmų veikimo teisingumą buvo nustatyta, kad visi algoritmai gavo teisingus rezultatus, t.y. visi algoritmai rado visas dideles elementų aibes.

Algoritmų greitaveikos tyrimas parodė, kad pilno perrinkimo algoritmas yra iš viso nenaudotinas, kai elementų kiekis >20; mėginio algoritmas buvo mažiausiai našus, nes atliekant mėginio analizę yra naudojamas mažesnis palaikymo lygis ir yra sugeneruojamas didelis kandidatų kiekis; padalinimo algoritmas yra pranašiausias kai sudarinėjamos aukštesnės eilės kandidatų aibės.

Algoritmų tyrimas su realiais duomenimis parodė, kad asociatyvinių ryšių analizė gali būti praktiškai pritaikoma versle ir gali padėti suformuoti tikslines klientų grupes.

7 Summary

The main goal of these thesis is to compare association rules finding algorithms and to indicate the usability of finding association rules in business area. In order to achieve this goal, the theoretical analysis of three algorithms is done:

1. The Apriori algorithm – the most well known association rule algorithm – based on the property: "Any subset of a large itemset must be large". This algorithm assumes that the database is memory-resident. The maximum number of database scans is one more than the cardinality of the largest large itemset.
2. The Sampling algorithm deals with the database sample prior the full database scan. The database sample is drawn such that it can be memory-resident. The Sampling algorithm reduces the number of database scans to one in the best case and two in the worst case.
3. The Partitioning algorithm divides database into partitions and bases on the property: "A large itemset must be large in at least one of the partitions". This algorithm reduces the number of database scans to two and divides the database into partitions such that each partition can be placed into main memory.

There are created programs for all three algorithms plus the program for the full set of itemsets algorithm. Programs are created in C++ language. In order to achieve topmost performance, the GUI is missed.

Nine test data sets are created to compare the algorithms. Six of them contains real life data from telecommunications business area. Datasets varies from the smallest one of 5 elements and 5 records, to the biggest one of 503 elements and 15205852 records. In order to manipulate the data, five more supplementary programs were created.

The verification of algorithms discovers that all analyzed algorithms are good. All of them passed all nine tests, and provided the identical results.

The performance tests ranked the algorithm with the full set of itemsets as not usable with a data where number of elements exceeds 20. The Sampling algorithm was less efficient than the Apriori and the Partitioning. It is all because of the reduced support and the big quantity of the potential candidates. The Partitioning algorithm has the best performance with a data where the cardinality of the largest large itemset is big.

The result analysis of the real life data revealed, that data mining and association rules are usable in business. There are showed how finding of Association rules may help to indicate target customers groups.

8 Priedai

8.1 Priedas 1. Pilno perrinkimo algoritmas su GVS Delphi kalba

```
unit Rysiai;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Grids;

type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    GroupBox1: TGroupBox;
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    OpenDialog2: TOpenDialog;
    Button3: TButton;
    ListBox1: TListBox;
    Label3: TLabel;
    Label4: TLabel;
    StringGrid1: TStringGrid;
    Button4: TButton;
    Label5: TLabel;
    Label6: TLabel;
    Button5: TButton;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Button6: TButton;
    Label11: TLabel;
    Label12: TLabel;
    StringGrid2: TStringGrid;
    Label13: TLabel;
    Label14: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

const
  MaxElem = 20;
  ElemVardIlgis = 20;
  StVardasDte = '..\Duomenys\test.dte';
  StVardasDta = '..\Duomenys\test.dta';

type
  Elem = LongWord;
  Vardas = String [ElemVardIlgis];
  PAibe = ^Aibe;
  Aibe =
    Record
      E : Elem;
      K : PAibe;
    End;
  PPalaik = ^Palaik;
  Palaik =
    Record
      A : PAibe;
      K : PPalaik;
      S : Elem;
    End;
  PAsoc = ^Asoc;
  Asoc =
    Record
      A : PAibe;
      B : PAibe;
      Ap : Elem;
      Bp : Elem;
      K : PAsoc;
    End;

var
  ElemVard : Array [1..MaxElem] Of Vardas;
  ElementaiNuskaityti : Boolean;
  AibesSuformuotos : Boolean;
  PalaikymaiSuskaiciuoti : Boolean;
  ElemKiek : Word;
  Sp : PPalaik;
  Sc : PAsoc;
  BazejeIrasu : Elem;

function ElemId (S : Vardas) : Elem;
var
  A : Elem;
begin
  A := 1;
  While (A < ElemKiek) And (ElemVard [A] <> S) Do
    Inc (A);
  ElemId := A;
end;

procedure AbTrinti (Var A : PAibe);
var
  A1 : PAibe;
begin
  While A <> Nil Do
    Begin
      A1 := A;
      A := A^.K;
      Dispose(A1);
```

```
    End;
end;

procedure AbKopij (S : PAibe; Var D : PAibe);
var
  A : PAibe;
begin
  AbTrinti (D);
  While S <> Nil Do
    Begin
      New (A);
      A^.E := S^.E;
      A^.K := D;
      D := A;
      S := S^.K;
    End;
end;

function AbRask (S : PAibe; E : Elem) : Boolean;
var
  Rado : Boolean;
begin
  Rado := False;
  While (Not Rado) And (S <> Nil) Do
    Begin
      Rado := (S^.E = E);
      S := S^.K;
    End;
  AbRask := Rado;
end;

procedure AbInc (Var S : PAibe; E : Elem);
var
  A : PAibe;
begin
  If Not AbRask (S, E) Then
    Begin
      New (A);
      A^.E := E;
      A^.K := S;
      S := A;
    End;
end;

procedure AbDec (Var S : PAibe; E : Elem);
var
  Rado : Boolean;
  D : ^PAibe;
  A : PAibe;
begin
  D := @S;
  Rado := (D^^.E = E);
  While (Not Rado) And (D^ <> Nil) Do
    Begin
      D := @(D^^.K);
      Rado := (D^^.E = E);
    End;
  If Rado Then
    Begin
      A := D^;
      D^ := A^.K;
      Dispose(A);
    End;
end;
```

```
Function AbList (A : PAibe) : String;

  Var
    V : String;

  Begin
    V := ElemVard [A^.E];
    A := A^.K;
    While A <> Nil Do
      Begin
        V := V + ',' + ElemVard [A^.E];
        A := A^.K;
      End;
    AbList := V;
  End;

procedure SpTrinti (Var S : PPalaik);
var
  S1 : PPalaik;
begin
  While S <> Nil Do
    Begin
      AbTrinti(S^.A);
      S1 := S;
      S := S^.K;
      Dispose(S1);
    End;
end;

procedure AsTrinti (Var S : PAsoc);
var
  S1 : PAsoc;
begin
  While S <> Nil Do
    Begin
      AbTrinti(S^.A);
      AbTrinti(S^.B);
      S1 := S;
      S := S^.K;
      Dispose(S1);
    End;
end;

Procedure SpAdd (Var S : PPalaik; A : PAibe);

  Var
    S1 : PPalaik;

  Begin
    New (S1);
    S1^.K := S;
    S := S1;
    S^.A := Nil;
    AbKopij (A, S^.A);
    S^.S := 0;
  End;

Procedure SpVesti;

  Var
    A : Elem;
    S : PPalaik;
```

```
  Begin
    Form1.StringGrid1.RowCount := 5000;
    A := 1;
    Form1.StringGrid1.Cells [0, 0] := 'Aibe';
    Form1.StringGrid1.Cells [1, 0] := 'Palaikymas';
    S := Sp;
    While (S <> Nil) And (A < Form1.StringGrid1.RowCount) Do
      Begin
        Form1.StringGrid1.Cells [0, A] := AbList (S^.A);
        If PalaikymaiSuskaiciuoti Then
          Form1.StringGrid1.Cells [1, A] := FloatToStr (S^.S *
            100 / BazejeIrasu)
        Else
          Form1.StringGrid1.Cells [1, A] := '';
        S := S^.K;
        Inc (A);
      End;
    Form1.StringGrid1.RowCount := A;
  End;

Function AbTuri (A : PAibe; B : PAibe) : Boolean;

  Begin
    If B = Nil Then
      AbTuri := True
    Else
      AbTuri := AbTuri (A, B^.K) And AbRask (A, B^.E);
  End;

Function AbLygu (A : PAibe; B : PAibe) : Boolean;

  Begin
    AbLygu := AbTuri (A, B) And AbTuri (B, A);
  End;

Function PlRasti (A : PAibe) : PPalaik;

  Var
    T1 : PPalaik;
    Rado : Boolean;

  Begin
    Rado := False;
    T1 := Sp;
    While (Not Rado) And (T1 <> Nil) Do
      Begin
        Rado := AbLygu (A, T1^.A);
        If Not Rado Then
          T1 := T1^.K;
      End;
    PlRasti := T1;
  End;

Procedure AsVesk;

  Var
    A : Elem;
    S : PAsoc;

  Begin
    Form1.StringGrid2.RowCount := 5000;
    A := 1;
    Form1.StringGrid2.Cells [0, 0] := 'Aibe X';
    Form1.StringGrid2.Cells [1, 0] := 'Aibe Y';
```

```
    Form1.StringGrid2.Cells [2, 0] := 'Palaik';
    Form1.StringGrid2.Cells [3, 0] := 'Stiprum';
    S := Sc;
    While (S <> Nil) And (A < Form1.StringGrid2.RowCount) Do
      Begin
        Form1.StringGrid2.Cells [0, A] := AbList (S^.A);
        Form1.StringGrid2.Cells [1, A] := AbList (S^.B);
        Form1.StringGrid2.Cells [2, A] := FloatToStr (S^.Bp *
          100 / BazejeIrasu);
        Form1.StringGrid2.Cells [3, A] := FloatToStr (S^.Bp * 100 / S^.Ap);
        S := S^.K;
        Inc (A);
      End;
    Form1.StringGrid2.RowCount := A;
  End;

procedure TForm1.Button1Click(Sender: TObject);
begin
  If OpenDialog1.Execute Then
    Edit1.Text := OpenDialog1.FileName;
  AibesSuformuotos := False;
  PalaikymaiSuskaiciuoti := False;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  If OpenDialog2.Execute Then
    Edit2.Text := OpenDialog2.FileName;
  ElementaiNuskaityti := False;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  A : Elem;
begin
  For A := 1 To MaxElem Do
    ElemVard [A] := '';
  ElementaiNuskaityti := False;
  AibesSuformuotos := False;
  PalaikymaiSuskaiciuoti := False;
  Sp := Nil;
  Sc := Nil;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  F : TextFile;
  A : Elem;
begin
  ElemKiek := 0;
  ListBox1.Clear;
  ElementaiNuskaityti := False;
  AibesSuformuotos := False;
  PalaikymaiSuskaiciuoti := False;
  If Edit2.Text = '' Then
    Edit2.Text := StVardasDte;
  AssignFile (F, Edit2.Text);
  {$I-}
  Reset (F);
  {$I+}
  If IOResult = 0 Then
    Begin
      While Not Eof (F) Do
        Begin
```

```
          Inc (ElemKiek);
          ReadLn (F, ElemVard [ElemKiek]);
        End;
      CloseFile (F);
      If ElemKiek > 0 Then
        Begin
          ElementaiNuskaityti := True;
          For A := 1 To ElemKiek Do
            ListBox1.Items.Append(ElemVard [A]);
        End;
    End;
  Label4.Caption := IntToStr (ElemKiek);
end;

procedure TForm1.Button4Click(Sender: TObject);

  Var
    Ak : Elem;
    Ab : PAibe;

  Procedure Itraukti;

    Begin
      If Ab <> Nil Then
        Begin
          Inc (Ak);
          SpAdd (Sp, Ab);
        End;
    End;

  Procedure ElmGo (E : Elem);

    Begin
      AbInc (Ab, E);
      If (E < ElemKiek) Then
        ElmGo (E + 1)
      Else
        Itraukti;
      AbDec (Ab, E);
      If (E < ElemKiek) Then
        ElmGo (E + 1)
      Else
        Itraukti;
    End;

  Begin
    Ak := 0;
    SpTrinti (Sp);
    AibesSuformuotos := False;
    PalaikymaiSuskaiciuoti := False;
    If ElementaiNuskaityti Then
      Begin
        Ab := Nil;
        ElmGo (1);
        SpVesti;
        AibesSuformuotos := True;
      End;
    Label6.Caption := IntToStr (Ak);
  End;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SpTrinti (Sp);
  AsTrinti (Sc);
```

```
end;

Procedure TForm1.Button5Click(Sender: TObject);

  Var
    KrKiek : Elem;
    F : TextFile;
    Rks : Boolean;
    Nne : Elem;

  Procedure Skanuok (S : PPalaik);

    Var
      A : String;
      A1 : PAibe;
      R : Boolean;

    Begin
      Inc (KrKiek);
      Reset (F);
      S^.S := 0;
      While Not Eof (F) Do
        Begin
          ReadLn (F, A);
          If Rks Then
            Inc (BazejeIrasu);
          A1 := S^.A;
          R := True;
          While R And (A1 <> Nil) Do
            Begin
              R := R And (Pos (ElemVard [A1^.E], A) > 0);
              A1 := A1^.K;
            End;
          If R Then
            Inc (S^.S);
        End;
      If S^.S > 0 Then
        Inc (Nne);
      If Rks Then
        Rks := False;
    End;

  Procedure SkanuotiBaze;

    Var
      S : PPalaik;

    Begin
      S := Sp;
      Rks := True;
      While S <> Nil Do
        Begin
          Skanuok (S);
          S := S^.K;
        End;
    End;

  Begin
    KrKiek := 0;
    BazejeIrasu := 0;
    Nne := 0;
    PalaikymaiSuskaiciuoti := False;
    If AibesSuformuotos Then
      Begin
```

```
        If Edit1.Text = '' Then
          Edit1.Text := StVardasDta;
        AssignFile (F, Edit1.Text);
        {$I-}
        Reset (F);
        {$I+}
        If IOResult = 0 Then
          Begin
            SkanuotiBaze;
            CloseFile (F);
            PalaikymaiSuskaiciuoti := True;
            SpVesti;
          End;
      End;
    Label8.Caption := IntToStr (KrKiek);
    Label10.Caption := IntToStr (BazejeIrasu);
    Label14.Caption := IntToStr (Nne);
  End;

procedure TForm1.Button6Click(Sender: TObject);

  Var
    A1 : PPalaik;
    Ras : Elem;

  Procedure Itrauk2 (Var A1 : PAibe; Var A2 : PAibe; E : Elem);

    Var
      T1 : PAibe;

    Procedure Itrauk3;

      Var
        C1 : PAsoc;
        C2 : PPalaik;

      Begin
        If (A1 <> Nil) And (A2 <> Nil) Then
          Begin
            C2 := PlRasti (A1);
            New (C1);
            C1^.K := Sc;
            Sc := C1;
            Sc^.Bp := E;
            Sc^.Ap := C2^.S;
            Sc^.A := Nil;
            Sc^.B := Nil;
            AbKopij (A1, Sc^.A);
            AbKopij (A2, Sc^.B);
            Inc (Ras);
          End;
      End;

    Procedure ElmGo (T : PAibe);

      Begin
        AbInc (A1, T^.E);
        AbDec (A2, T^.E);
        If (T^.K <> Nil) Then
          ElmGo (T^.K)
        Else
          Itrauk3;
        AbDec (A1, T^.E);
        AbInc (A2, T^.E);
```

```
        If (T^.K <> Nil) Then
          ElmGo (T^.K)
        Else
          Itrauk3;
      End;

    Begin
      T1 := Nil;
      AbKopij (A2, T1);
      ElmGo (T1);
      AbTrinti (T1);
    End;

  Procedure Itraukti (A : PPalaik);

    Var
      T1 : PAibe;
      T2 : PAibe;

    Begin
      T1 := Nil;
      T2 := Nil;
      AbKopij (A^.A, T2);
      Itrauk2 (T1, T2, A^.S);
      AbTrinti (T1);
      AbTrinti (T2);
    End;

begin
  If PalaikymaiSuskaiciuoti Then
    Begin
      A1 := Sp;
      AsTrinti (Sc);
      Ras := 0;
      While A1 <> Nil Do
        Begin
          If A1^.A <> Nil Then
            If A1^.A^.K <> Nil Then
              If A1^.S > 0 Then
                Itraukti (A1);
          A1 := A1^.K;
        End;
      Label12.Caption := IntToStr (Ras);
      AsVesk;
    End;
end;

End.
```

8.2 Priedas 2. Algoritmų bendra dalis C++ kalba

```
// Bendra metodu dalis - Versija 2
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

#define Byte  unsigned char
#define Word  unsigned short
#define DWord unsigned long
#define Bool  unsigned char

const Byte MaxVardoIlgis = 20;
const DWord MaxElementuKiek = 600;

const Byte SmS = ('9' - '0') + 2;
const Byte SmR = ('9' - '0') + ('Z' - 'A') + 3;
const Byte SmK = ('9' - '0') + ('Z' - 'A') + ('z' - 'a') + 4;

const DWord BitLent [32] = {0x00000001, 0x00000002, 0x00000004, 0x00000008,
                            0x00000010, 0x00000020, 0x00000040, 0x00000080,
                            0x00000100, 0x00000200, 0x00000400, 0x00000800,
                            0x00001000, 0x00002000, 0x00004000, 0x00008000,
                            0x00010000, 0x00020000, 0x00040000, 0x00080000,
                            0x00100000, 0x00200000, 0x00400000, 0x00800000,
                            0x01000000, 0x02000000, 0x04000000, 0x08000000,
                            0x10000000, 0x20000000, 0x40000000, 0x80000000};

Byte Simb (char A)
  {
    return (A == '_') ? 0 : ((A >= '0') && (A <= '9')) ? A - '0' + 1 :
      ((A >= 'A') && (A <= 'Z')) ? A - 'A' + SmS :
      ((A >= 'a') && (A <= 'z')) ? A - 'a' + SmR : 0;
  }

char Sima (Byte A)
  {
    return (A == 0) ? '_' : (A < SmS) ? A - 1 + '0' :
      (A < SmR) ? A - SmS + 'A' : A - SmR + 'a';
  }

struct RaidSar
  {
    DWord N;
    RaidSar *Sek [SmK];
    RaidSar ();
    ~RaidSar ();
  };

RaidSar::RaidSar ()
  {
    N = 0;
    for (int A = 0; A < SmK; A ++)
      Sek [A] = NULL;
  }

RaidSar::~RaidSar ()
  {
    for (int A = 0; A < SmK; A ++)
      if (Sek [A])
        delete Sek [A];
  }
```

```
struct NumSar
  {
    char V [MaxVardoIlgis];
    DWord K;
    NumSar ();
  };

NumSar::NumSar ()
  {
    V [0] = 0;
    K = 0;
  }

struct VidBazeE
  {
    VidBazeE *S;
    char Skyr;
    char V [MaxVardoIlgis];
    VidBazeE ();
  };

VidBazeE::VidBazeE ()
  {
    S = NULL;
  }

struct VidBaze
  {
    VidBazeE *V;
    VidBaze ();
    ~VidBaze ();
    void Trinti ();
    DWord Skaityti (char *, DWord, DWord);
  };

VidBaze::VidBaze ()
  {
    V = NULL;
  }

VidBaze::~VidBaze ()
  {
    if (V)
      Trinti ();
  }

void VidBaze::Trinti ()
  {
    VidBazeE *S;
    while (V)
      {
        S = V;
        V = S -> S;
        S -> S = NULL;
        delete S;
      }
  }

DWord VidBaze::Skaityti (char *Fv, DWord Nuo, DWord Kiek)
  {
    FILE *F = fopen (Fv, "rt");
    fseek (F, Nuo, 0);
    if (V)
```

```
      Trinti ();
    unsigned char C = fgetc (F);
    VidBazeE *B;
    VidBazeE **G = &V;
    DWord Num = 0;
    while ((! feof (F)) && (Num < Kiek))
      {
        Num ++;
        while ((! feof (F)) && (C != 10))
          {
            int Poz = 0;
            B = new VidBazeE;
            (*G) = B;
            G = & (B -> S);
            while ((! feof (F)) && (C != 10) && (C != ','))
              {
                B -> V [Poz] = C;
                Poz ++;
                C = fgetc (F);
              }
            B -> V [Poz] = 0;
            B -> Skyr = C;
            if ((! feof (F)) && (C == ','))
              C = fgetc (F);
          }
        if (Num < Kiek)
          while ((! feof (F)) && ((C == 10) || (C == ',')))
            C = fgetc (F);
      }
    DWord Ats = (feof (F)) ? 0xFFFFFFFF : ftell (F);
    fclose (F);
    return Ats;
  }

struct Elementai
  {
    RaidSar *R;
    NumSar *N [MaxElementuKiek];
    DWord Kiek;
    Elementai ();
    ~Elementai ();
    void Ideti (char *, DWord);
    void Ideti2 (RaidSar **, char *, DWord);
    DWord RastiV (char *);
    char *RastiN (DWord);
    void Nusiskaityti (char *);
    DWord Skaic (char *);
    DWord SkaicV (VidBaze *);
    void Nulinti ();
  };

Elementai::Elementai ()
  {
    R = NULL;
    for (int A = 0; A < MaxElementuKiek; A ++)
      N [A] = NULL;
    Kiek = 0;
  }

Elementai::~Elementai ()
  {
    if (R)
      delete R;
    for (int A = 0; A < MaxElementuKiek; A ++)
```

```
      if (N [A])
        delete N [A];
  }

void Elementai::Ideti (char *Vr, DWord Nr)
  {
    Ideti2 (&R, Vr, Nr);
    if (! (N [Nr]))
      N [Nr] = new NumSar;
    Byte Poz = 0;
    while (Vr [Poz])
      {
        N [Nr] -> V [Poz] = Vr [Poz];
        Poz ++;
      }
    N [Nr] -> V [Poz] = 0;
  }

void Elementai::Ideti2 (RaidSar **E, char *c, DWord Nr)
  {
    if (! (*E))
      (*E) = new RaidSar;
    if (! *c)
      (*E) -> N = Nr;
    else
      Ideti2 (& ((*E) -> Sek [Simb (*c)]), (c + 1), Nr);
  }

DWord Elementai::RastiV (char *c)
  {
    RaidSar *A = R;
    while ((*c) && (A))
      A = A -> Sek [Simb (*(c++))];
    return (A) ? A -> N : 0;
  }

char *Elementai::RastiN (DWord Nr)
  {
    return (N [Nr]) ? N [Nr] -> V : NULL;
  }

void Elementai::Nusiskaityti (char *Fv)
  {
    FILE *F = fopen (Fv, "rt");
    unsigned char C = fgetc (F);
    char Zodis [MaxVardoIlgis];
    DWord Num = 1;
    while (! feof (F))
      {
        int Poz = 0;
        while ((! feof (F)) && (C != 10))
          {
            Zodis [Poz] = C;
            Poz ++;
            C = fgetc (F);
          }
        Zodis [Poz] = 0;
        Ideti (Zodis, Num);
        Num ++;
        while ((! feof (F)) && (C == 10))
          C = fgetc (F);
      }
    Kiek = Num - 1;
    fclose (F);
```

```
  }

DWord Elementai::Skaic (char *Fv)
  {
    FILE *F = fopen (Fv, "rt");
    unsigned char C = fgetc (F);
    char Zodis [MaxVardoIlgis];
    DWord Num = 0;
    while (! feof (F))
      {
        Num ++;
        while ((! feof (F)) && (C != 10))
          {
            int Poz = 0;
            while ((! feof (F)) && (C != 10) && (C != ','))
              {
                Zodis [Poz] = C;
                Poz ++;
                C = fgetc (F);
              }
            Zodis [Poz] = 0;
            N [RastiV (Zodis)] -> K ++;
            if ((! feof (F)) && (C == ','))
              C = fgetc (F);
          }
        while ((! feof (F)) && ((C == 10) || (C == ',')))
          C = fgetc (F);
      }
    fclose (F);
    return Num;
  }

DWord Elementai::SkaicV (VidBaze *F)
  {
    VidBazeE *G = F -> V;
    DWord Num = 0;
    while (G)
      {
        Num ++;
        char Sk = 0;
        while (G && (Sk != 10))
          {
            N [RastiV (G -> V)] -> K ++;
            Sk = G -> Skyr;
            if (Sk == ',')
              G = G -> S;
          }
        if (G)
          G = G -> S;
      }
    return Num;
  }

void Elementai::Nulinti ()
  {
    for (DWord A = 0; A < Kiek; A ++)
      N [A + 1] -> K = 0;
  }

struct Aibe
  {
    DWord D;
    DWord *E;
    Aibe (Elementai *);
```

```
    ~Aibe ();
    void Valyt ();
    void Ideti (DWord);
    void Ideti (Aibe *);
    void Isimti (DWord);
    void Isimti (Aibe *);
    void Kart (Aibe *);
    Bool ArYra (DWord);
    Bool ArYra (Aibe *);
    Bool Lygu (Aibe *);
    DWord Nariu ();
    Bool Tuscia ();
    Bool Papildo (Aibe *, Elementai *);
    void Uzpildyti (Elementai *);
    void Print (FILE *, Elementai *);
  };

Aibe::Aibe (Elementai *B)
  {
    D = ((B -> Kiek) - 1) / 32 + 1;
    E = new DWord [D];
    for (DWord A = 0; A < D; A ++)
      E [A] = 0;
  }

Aibe::~Aibe ()
  {
    delete E;
  }

void Aibe::Valyt ()
  {
    for (DWord A = 0; A < D; A ++)
      E [A] = 0;
  }

void Aibe::Ideti (DWord N)
  {
    DWord A1 = (N - 1) / 32;
    DWord A2 = N - A1 * 32 - 1;
    E [A1] |= BitLent [A2];
  }

void Aibe::Ideti (Aibe *A)
  {
    for (DWord B = 0; B < D; B ++)
      E [B] |= (A -> E [B]);
  }

void Aibe::Isimti (DWord N)
  {
    DWord A1 = (N - 1) / 32;
    DWord A2 = N - A1 * 32 - 1;
    E [A1] &= ~(BitLent [A2]);
  }

void Aibe::Isimti (Aibe *A)
  {
    for (DWord B = 0; B < D; B ++)
      E [B] &= ~(A -> E [B]);
  }

void Aibe::Kart (Aibe *A)
  {
```

```
    for (DWord B = 0; B < D; B ++)
      E [B] &= (A -> E [B]);
  }

Bool Aibe::ArYra (DWord N)
  {
    DWord A1 = (N - 1) / 32;
    DWord A2 = N - A1 * 32 - 1;
    return (E [A1] & BitLent [A2]) ? 1 : 0;
  }

Bool Aibe::ArYra (Aibe *A)
  {
    Bool Rado = 1;
    for (DWord B = 0; (B < D) && (Rado); B ++)
      if (A -> E [B])
        Rado = Rado && ((E [B] & (A -> E [B])) == (A -> E [B]));
    return Rado;
  }

Bool Aibe::Lygu (Aibe *A)
  {
    Bool Rado = 1;
    for (DWord B = 0; (B < D) && (Rado); B ++)
      Rado = Rado && (E [B] == (A -> E [B]));
    return Rado;
  }

DWord Aibe::Nariu ()
  {
    DWord Nar = 0;
    for (DWord B = 0; B < D; B ++)
      for (Byte C = 0; C < 32; C ++)
        if (E [B] & BitLent [C])
          Nar ++;
    return Nar;
  }

Bool Aibe::Tuscia ()
  {
    Bool Rado = 1;
    for (DWord B = 0; (B < D) && (Rado); B ++)
      Rado = Rado && (E [B] == 0);
    return Rado;
  }

Bool Aibe::Papildo (Aibe *A, Elementai *El)
  {
    Aibe *B1 = new Aibe (El);
    Aibe *B2 = new Aibe (El);
    Aibe *B3 = new Aibe (El);
    B1 -> Ideti (this);
    B1 -> Kart (A);
    B2 -> Ideti (this);
    B2 -> Isimti (B1);
    B3 -> Ideti (A);
    B3 -> Isimti (B1);
    Bool Rez = ((B2 -> Nariu () == 1) && (B3 -> Nariu () == 1));
    delete B1;
    delete B2;
    delete B3;
    return Rez;
  }
```

```
void Aibe::Uzpildyti (Elementai *El)
  {
    for (DWord A = 0; A < El -> Kiek; A ++)
      Ideti (A + 1);
  }

void Aibe::Print (FILE *F, Elementai *El)
  {
    fprintf (F, "(");
    Bool Jb = 0;
    for (DWord B = 0; B < El -> Kiek; B ++)
      if (ArYra (B + 1))
        {
          if (Jb)
            fprintf (F, ",");
          fprintf (F, El -> RastiN (B + 1));
          Jb = 1;
        }
    fprintf (F, ")");
  }

struct Palaikymas
  {
    Aibe *A;
    Palaikymas *S;
    DWord K;
    Bool Jau;
    Palaikymas ();
    ~Palaikymas ();
    void Print (FILE *, DWord, Elementai *);
  };

Palaikymas::Palaikymas ()
  {
    A = NULL;
    S = NULL;
    K = 0;
    Jau = 0;
  }

Palaikymas::~Palaikymas ()
  {
    if (A)
      delete A;
    if (S)
      delete S;
  }

void Palaikymas::Print (FILE *F, DWord N, Elementai *E)
  {
    if (A)
      A -> Print (F, E);
    fprintf (F, " - %10.6lf%%\n", ((double) (K)) * 100 / ((double) (N)));
  }

struct PalaikymuSk
  {
    Palaikymas *P;
    Elementai *E;
    PalaikymuSk (Elementai *);
    ~PalaikymuSk ();
    Bool ArYra (Aibe *);
    DWord Rask (Aibe *);
    Bool Prijungti (Aibe *);
```

```
    void PrijungtiB (Aibe *);
    void PrijungtiB (DWord);
    void Atjungti (Aibe *);
    void Print (FILE *, DWord);
    void MeskMazus (DWord, double);
    DWord Skaic (char *);
    DWord SkaicV (VidBaze *);
    void Isikelti (PalaikymuSk *);
    void ARGen (FILE *, double);
    void ARGenVidinis (Aibe *, Aibe *, DWord, FILE *, double, Palaikymas *);
    void ARGenVedimas (FILE *, Aibe *, Aibe *, double, Palaikymas *);
  };

PalaikymuSk::PalaikymuSk (Elementai *El)
  {
    P = NULL;
    E = El;
  }

PalaikymuSk::~PalaikymuSk ()
  {
    if (P)
      delete P;
  }

Bool PalaikymuSk::ArYra (Aibe *A)
  {
    Bool Rado = 0;
    Palaikymas *Pl = P;
    while (Pl && (! Rado))
      {
        Rado = (A -> Lygu (Pl -> A));
        Pl = Pl -> S;
      }
    return Rado;
  }

DWord PalaikymuSk::Rask (Aibe *A)
  {
    DWord Rado = 0;
    Palaikymas *Pl = P;
    while (Pl && (! Rado))
      {
        if (A -> Lygu (Pl -> A))
          Rado = Pl -> K;
        Pl = Pl -> S;
      }
    return Rado;
  }

Bool PalaikymuSk::Prijungti (Aibe *A)
  {
    if (! ArYra (A))
      {
        Palaikymas *V = new Palaikymas;
        V -> S = P;
        P = V;
        V -> A = new Aibe (E);
        V -> A -> Ideti (A);
        return 1;
      }
    else
      return 0;
  }
```

```
void PalaikymuSk::PrijungtiB (Aibe *A)
  {
    Palaikymas *V = new Palaikymas;
    V -> S = P;
    P = V;
    V -> A = new Aibe (E);
    V -> A -> Ideti (A);
  }

void PalaikymuSk::PrijungtiB (DWord N)
  {
    Palaikymas *V = new Palaikymas;
    V -> S = P;
    P = V;
    V -> A = new Aibe (E);
    V -> A -> Ideti (N);
    V -> K = E -> N [N] -> K;
    V -> Jau = 1;
  }

void PalaikymuSk::Atjungti (Aibe *A)
  {
    Bool Rado = 0;
    Palaikymas **Pl = &P;
    while ((*Pl) && (! Rado))
      {
        Rado = A -> Lygu ((*Pl) -> A);
        if (Rado)
          {
            Palaikymas *V = (*Pl);
            (*Pl) = V -> S;
            V -> S = NULL;
            delete V;
          }
        else
          Pl = &((*Pl) -> S);
      }
  }

void PalaikymuSk::Print (FILE *F, DWord N)
  {
    Palaikymas *V = P;
    while (V)
      {
        V -> Print (F, N, E);
        V = V -> S;
      }
  }

void PalaikymuSk::MeskMazus (DWord N, double s)
  {
    Palaikymas **Pl = &P;
    while (*Pl)
      {
        if ((((double) ((*Pl) -> K)) / ((double) N)) < s)
          {
            Palaikymas *V = (*Pl);
            (*Pl) = V -> S;
            V -> S = NULL;
            delete V;
          }
        else
          Pl = &((*Pl) -> S);
```

```
      }
  }

DWord PalaikymuSk::Skaic (char *Fv)
  {
    FILE *F = fopen (Fv, "rt");
    unsigned char C = fgetc (F);
    char Zodis [MaxVardoIlgis];
    DWord Num = 0;
    Palaikymas *Pl;
    Aibe *A = new Aibe (E);
    while (! feof (F))
      {
        Num ++;
        A -> Valyt ();
        while ((! feof (F)) && (C != 10))
          {
            int Poz = 0;
            while ((! feof (F)) && (C != 10) && (C != ','))
              {
                Zodis [Poz] = C;
                Poz ++;
                C = fgetc (F);
              }
            Zodis [Poz] = 0;
            A -> Ideti (E -> RastiV (Zodis));
            if ((! feof (F)) && (C == ','))
              C = fgetc (F);
          }
        Pl = P;
        while (Pl)
          {
            if (! (Pl -> Jau))
              if (A -> ArYra (Pl -> A))
                (Pl -> K) ++;
            Pl = Pl -> S;
          }
        while ((! feof (F)) && ((C == 10) || (C == ',')))
          C = fgetc (F);
      }
    delete A;
    Pl = P;
    while (Pl)
      {
        if (! (Pl -> Jau))
          Pl -> Jau = 1;
        Pl = Pl -> S;
      }
    fclose (F);
    return Num;
  }

DWord PalaikymuSk::SkaicV (VidBaze *F)
  {
    VidBazeE *G = F -> V;
    DWord Num = 0;
    Palaikymas *Pl;
    Aibe *A = new Aibe (E);
    while (G)
      {
        Num ++;
        A -> Valyt ();
        char Sk = 0;
        while (G && (Sk != 10))
```

```
          {
            A -> Ideti (E -> RastiV (G -> V));
            Sk = G -> Skyr;
            if (Sk == ',')
              G = G -> S;
          }
        Pl = P;
        while (Pl)
          {
            if (! (Pl -> Jau))
              if (A -> ArYra (Pl -> A))
                (Pl -> K) ++;
            Pl = Pl -> S;
          }
        if (G)
          G = G -> S;
      }
    delete A;
    Pl = P;
    while (Pl)
      {
        if (! (Pl -> Jau))
          Pl -> Jau = 1;
        Pl = Pl -> S;
      }
    return Num;
  }

void PalaikymuSk::Isikelti (PalaikymuSk *S)
  {
    while (S -> P)
      {
        Palaikymas *Pl = S -> P;
        S -> P = S -> P -> S;
        Pl -> S = P;
        P = Pl;
      }
  }

void PalaikymuSk::ARGen (FILE *F, double Alfa)
  {
    Palaikymas *Pl = P;
    Aibe *l = new Aibe (E);
    Aibe *x = new Aibe (E);
    while (Pl)
      {
        if ((Pl -> A -> Nariu ()) > 1)
          {
            l -> Valyt ();
            x -> Valyt ();
            l -> Ideti (Pl -> A);
            ARGenVidinis (l, x, E -> Kiek, F, Alfa, Pl);
          }
        Pl = Pl -> S;
      }
    delete l;
    delete x;
  }

void PalaikymuSk::ARGenVidinis (Aibe *A, Aibe *B, DWord En, FILE *F,
                                double Alfa, Palaikymas *Pl)
  {
    if (En)
      {
```

```
        while (En && (! (A -> ArYra (En))))
          En --;
        if (En)
          {
            A -> Isimti (En);
            B -> Ideti (En);
            ARGenVidinis (A, B, En - 1, F, Alfa, Pl);
            A -> Ideti (En);
            B -> Isimti (En);
            ARGenVidinis (A, B, En - 1, F, Alfa, Pl);
          }
      }
    if (! En)
      if (! ((A -> Tuscia ()) || (B -> Tuscia ())))
        ARGenVedimas (F, B, A, Alfa, Pl);
  }

void PalaikymuSk::ARGenVedimas (FILE *F, Aibe *A, Aibe *B,
                                double Alfa, Palaikymas *Pl)
  {
    double Al = ((double) (Pl -> K)) / ((double) (Rask (A)));
    if (Al >= Alfa)
      {
        A -> Print (F, E);
        fprintf (F, " => ");
        B -> Print (F, E);
        fprintf (F, " - %10.6lf%%\n", Al * 100);
      }
  }
```

8.3 Priedas 3. Pilno perrinkimo algoritmas C++ kalba

```
#include <stdio.h>
#include <math.h>
#include "Bendras.drf"

void Formuoti (PalaikymuSk *P, Aibe *A, DWord E)
  {
    if (E)
      {
        A -> Isimti (E);
        Formuoti (P, A, E - 1);
        A -> Ideti (E);
        Formuoti (P, A, E - 1);
      }
    else
      if (! (A -> Tuscia ()))
        P -> PrijungtiB (A);
  }

void main (int Ac, char *Ap [])
  {
    printf ("Asociatyviniu rysiu tyrimo 1 algoritmas\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ac != 6)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ap [0],
        "<baze> <elementai> <rezultatai> <palaikymas> <stiprumas>");
    else
      {
        Elementai *E = new Elementai;
        E -> Nusiskaityti (Ap [2]);
        PalaikymuSk *P = new PalaikymuSk (E);
        Aibe *A = new Aibe (E);
        A -> Uzpildyti (E);
        Formuoti (P, A, E -> Kiek);
        delete A;
        DWord N = P -> Skaic (Ap [1]);
        FILE *F = fopen (Ap [3], "wt");
        double s, alfa;
        s = atof (Ap [4]);
        alfa = atof (Ap [5]);
        P -> MeskMazus (N, s);
        fprintf (F, "Daznos aibes :\n");
        P -> Print (F, N);
        fprintf (F, "\n\nAsociatyvus rysiai :\n");
        P -> ARGen (F, alfa);
        fclose (F);
        delete P;
        delete E;
      }
  }
```

8.4 Priedas 4. Apriorinis algoritmas C++ kalba

```
#include <stdio.h>
#include <math.h>
#include "Bendras.drf"

void main (int Ac, char *Ap [])
  {
    printf ("Asociatyviniu rysiu tyrimo 2 algoritmas\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ac != 6)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ap [0],
        "<baze> <elementai> <rezultatai> <palaikymas> <stiprumas>");
    else
      {
        Elementai *E = new Elementai;
        E -> Nusiskaityti (Ap [2]);
        double s, alfa;
        s = atof (Ap [4]);
        alfa = atof (Ap [5]);
        PalaikymuSk *P1, *P2, *P3;
        P1 = new PalaikymuSk (E);
        P2 = new PalaikymuSk (E);
        P3 = new PalaikymuSk (E);
        DWord N;
        N = E -> Skaic (Ap [1]);
        for (DWord A1 = 1; A1 <= E -> Kiek; A1 ++)
          P2 -> PrijungtiB (A1);
        DWord Eile = 0;
        Aibe *A = new Aibe (E);
        while (P2 -> P)
          {
            if (Eile)
              N = P2 -> Skaic (Ap [1]);
            P2 -> MeskMazus (N, s);
            Palaikymas *Pl1, *Pl2;
            Pl1 = P2 -> P;
            Eile ++;
            while (Pl1)
              {
                Pl2 = Pl1 -> S;
                while (Pl2)
                  {
                    if (Pl1 -> A -> Papildo (Pl2 -> A, E))
                      {
                        A -> Valyt ();
                        A -> Ideti (Pl1 -> A);
                        A -> Ideti (Pl2 -> A);
                        if (Eile == 1)
                          P3 -> PrijungtiB (A);
                        else
                          P3 -> Prijungti (A);
                      }
                    Pl2 = Pl2 -> S;
                  }
                Pl1 = Pl1 -> S;
              }
            P1 -> Isikelti (P2);
            P2 -> Isikelti (P3);
          }
        delete A;
        delete P3;
```

```
      delete P2;
      FILE *F = fopen (Ap [3], "wt");
      fprintf (F, "Daznos aibes :\n");
      P1 -> Print (F, N);
      fprintf (F, "\n\nAsociatyvus rysiai :\n");
      P1 -> ARGen (F, alfa);
      fclose (F);
      delete P1;
      delete E;
    }
}
```

8.5 Priedas 5. Mėginio algoritmas C++ kalba

```
#include <stdio.h>
#include <math.h>
#include "Bendras.drf"

void main (int Ac, char *Ap [])
  {
    printf ("Asociatyviniu rysiu tyrimo 3 algoritmas\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ac != 6)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ap [0],
        "<baze> <elementai> <rezultatai> <palaikymas> <stiprumas>");
    else
      {
        Elementai *E = new Elementai;
        E -> Nusiskaityti (Ap [2]);
        double s, alfa;
        s = atof (Ap [4]);
        alfa = atof (Ap [5]);
        VidBaze *Vb = new VidBaze;
        Vb -> Skaityti (Ap [1], 0, 100000);
        PalaikymuSk *P1, *P2, *P3;
        P1 = new PalaikymuSk (E);
        P2 = new PalaikymuSk (E);
        P3 = new PalaikymuSk (E);
        DWord N;
        N = E -> SkaicV (Vb);
        for (DWord A1 = 1; A1 <= E -> Kiek; A1 ++)
          P2 -> PrijungtiB (A1);
        DWord Eile = 0;
        Aibe *A = new Aibe (E);
        Palaikymas *Pl1, *Pl2;
        while (P2 -> P)
          {
            if (Eile)
              N = P2 -> SkaicV (Vb);
            P2 -> MeskMazus (N, s);
            Pl1 = P2 -> P; Eile ++;
            while (Pl1)
              {
                Pl2 = Pl1 -> S;
                while (Pl2)
                  {
                    if (Pl1 -> A -> Papildo (Pl2 -> A, E))
                      {
                        A -> Valyt ();
                        A -> Ideti (Pl1 -> A);
                        A -> Ideti (Pl2 -> A);
                        if (Eile == 1)
                          P3 -> PrijungtiB (A);
                        else
                          P3 -> Prijungti (A);
                      }
                    Pl2 = Pl2 -> S;
                  }
                Pl1 = Pl1 -> S;
              }
            P1 -> Isikelti (P2);
            P2 -> Isikelti (P3);
          }
        delete Vb; E -> Nulinti ();
```

```
      N = E -> Skaic (Ap [1]);
      for (DWord A1 = 1; A1 <= E -> Kiek; A1 ++)
        P2 -> PrijungtiB (A1);
      P2 -> MeskMazus (N, s);
      Pl1 = P1 -> P;
      while (Pl1)
        {
          Pl2 = Pl1 -> S;
          while (Pl2)
            {
              if (Pl1 -> A -> Papildo (Pl2 -> A, E))
                {
                  A -> Valyt ();
                  A -> Ideti (Pl1 -> A);
                  A -> Ideti (Pl2 -> A);
                  P2 -> Prijungti (A);
                }
              Pl2 = Pl2 -> S;
            }
          Pl1 = Pl1 -> S;
        }
      P2 -> Skaic (Ap [1]);
      P2 -> MeskMazus (N, s);
      Bool BuvoNauju = 0;
      Pl1 = P2 -> P;
      while ((Pl1) && (! BuvoNauju))
        {
          BuvoNauju = (! (P1 -> ArYra (Pl1 -> A)));
          Pl1 = Pl1 -> S;
        }
      if (BuvoNauju)
        {
          printf ("Rastos naujos elementu sekos\n");
          Pl1 = P2 -> P;
          while (Pl1)
            {
              Pl2 = Pl1 -> S;
              while (Pl2)
                {
                  if (Pl1 -> A -> Papildo (Pl2 -> A, E))
                    {
                      A -> Valyt ();
                      A -> Ideti (Pl1 -> A);
                      A -> Ideti (Pl2 -> A);
                      P2 -> Prijungti (A);
                    }
                  Pl2 = Pl2 -> S;
                }
              Pl1 = Pl1 -> S;
            }
          P2 -> Skaic (Ap [1]);
          P2 -> MeskMazus (N, s);
        }
      delete A;
      FILE *F = fopen (Ap [3], "wt");
      fprintf (F, "Daznos aibes :\n");
      P2 -> Print (F, N);
      fprintf (F, "\n\nAsociatyvus rysiai :\n");
      P2 -> ARGen (F, alfa);
      fclose (F);
      delete P3; delete P2;
      delete P1; delete E;
    }
}
```

8.6 Priedas 6. Padalinimo algoritmas C++ kalba

```
#include <stdio.h>
#include <math.h>
#include "Bendras.drf"

void main (int Ac, char *Ap [])
  {
    printf ("Asociatyviniu rysiu tyrimo 4 algoritmas\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ac != 6)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ap [0],
        "<baze> <elementai> <rezultatai> <palaikymas> <stiprumas>");
    else
      {
        Elementai *E = new Elementai;
        E -> Nusiskaityti (Ap [2]);
        double s, alfa;
        s = atof (Ap [4]);
        alfa = atof (Ap [5]);
        VidBaze *Vb = new VidBaze;
        PalaikymuSk *P1, *P2, *P3, *P4;
        P1 = new PalaikymuSk (E);
        P2 = new PalaikymuSk (E);
        P3 = new PalaikymuSk (E);
        P4 = new PalaikymuSk (E);
        DWord N;
        DWord EinPoz = 0;
        Aibe *A = new Aibe (E);
        while (EinPoz != 0xFFFFFFFF)
          {
            Vb -> Trinti ();
            EinPoz = Vb -> Skaityti (Ap [1], EinPoz, 100000);
            N = E -> SkaicV (Vb);
            for (DWord A1 = 1; A1 <= E -> Kiek; A1 ++)
              P2 -> PrijungtiB (A1);
            DWord Eile = 0;
            Palaikymas *Pl1, *Pl2;
            while (P2 -> P)
              {
                if (Eile)
                  N = P2 -> SkaicV (Vb);
                P2 -> MeskMazus (N, s);
                Pl1 = P2 -> P;
                Eile ++;
                while (Pl1)
                  {
                    Pl2 = Pl1 -> S;
                    while (Pl2)
                      {
                        if (Pl1 -> A -> Papildo (Pl2 -> A, E))
                          {
                            A -> Valyt ();
                            A -> Ideti (Pl1 -> A);
                            A -> Ideti (Pl2 -> A);
                            if (Eile == 1)
                              P3 -> PrijungtiB (A);
                            else
                              P3 -> Prijungti (A);
                          }
                        Pl2 = Pl2 -> S;
                      }
```

```
                Pl1 = Pl1 -> S;
              }
            P1 -> Isikelti (P2);
            P2 -> Isikelti (P3);
          }
        Pl1 = P1 -> P;
        while (Pl1)
          {
            P4 -> Prijungti (Pl1 -> A);
            Pl2 = Pl1;
            Pl1 = Pl1 -> S;
            Pl2 -> S = NULL;
            delete Pl2;
          }
        P1 -> P = NULL;
        E -> Nulinti ();
      }
    delete Vb;
    delete A;
    N = P4 -> Skaic (Ap [1]);
    P4 -> MeskMazus (N, s);
    FILE *F = fopen (Ap [3], "wt");
    fprintf (F, "Daznos aibes :\n");
    P4 -> Print (F, N);
    fprintf (F, "\n\nAsociatyvus rysiai :\n");
    P4 -> ARGen (F, alfa);
    fclose (F);
    delete P3;
    delete P2;
    delete P1;
    delete E;
  }
}
```

8.7 Priedas 7. Elementų sąrašo sudarymas C++ kalba

```
// Elementu saraso sudarymas is pilnos bazes
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

const int SmS = ('9' - '0') + 2;
const int SmR = ('9' - '0') + ('Z' - 'A') + 3;
const int SmK = ('9' - '0') + ('Z' - 'A') + ('z' - 'a') + 4;

unsigned char Simb (unsigned char A)
  {
    return (A == '_') ? 0 : ((A >= '0') && (A <= '9')) ? A - '0' + 1 :
      ((A >= 'A') && (A <= 'Z')) ? A - 'A' + SmS :
      ((A >= 'a') && (A <= 'z')) ? A - 'a' + SmR : 0;
  }

unsigned char Sima (unsigned char A)
  {
    return (A == 0) ? '_' : (A < SmS) ? A - 1 + '0' :
      (A < SmR) ? A - SmS + 'A' : A - SmR + 'a';
  }

struct ElSar
  {
    char Yra;
    ElSar *Sek [SmK];
    ElSar ();
    ~ElSar ();
  };

ElSar::ElSar ()
  {
    Yra = 0;
    for (int A = 0; A < SmK; A ++)
      Sek [A] = NULL;
  }

ElSar::~ElSar ()
  {
    for (int A = 0; A < SmK; A ++)
      if (Sek [A])
        delete Sek [A];
  }

void Ideti (ElSar **E, unsigned char *c)
  {
    if (! (*E))
      (*E) = new ElSar;
    if (! *c)
      (*E) -> Yra = 1;
    else
      Ideti (& ((*E) -> Sek [Simb (*c)]), (c + 1));
  }

unsigned char Zodis [50];

void Isvesti (ElSar *E, FILE *Fo, int Poz)
  {
    Zodis [Poz] = 0;
    if (E -> Yra)
```

```
      fprintf (Fo, "%s\n", (char *) Zodis);
    for (int A = 0; A < SmK; A ++)
      if (E -> Sek [A])
        {
          Zodis [Poz] = Sima (A);
          Isvesti (E -> Sek [A], Fo, Poz + 1);
        }
  }

int Simbolis (unsigned char C)
  {
    return (((C >= '0') && (C <= '9')) || ((C >= 'a') &&
      (C <= 'z')) || ((C >= 'A') && (C <= 'Z')) || (C == '_'));
  }

void main (int Ak, char *Ae [])
  {
    printf ("Elementu saraso sudarymas is pilnos bazes\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ak != 3)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ae [0],
        "<duomenu baze> <elementu sarasas>");
    else
      {
        FILE *Fi = fopen (Ae [1], "rt");
        if (! Fi)
          printf ("Klaida !\nNegaliu atidaryti bazes failo\n");
        else
          {
            ElSar *E = NULL;
            unsigned char C = fgetc (Fi);
            while (! feof (Fi))
              {
                int Poz = 0;
                while ((! feof (Fi)) && Simbolis (C))
                  {
                    Zodis [Poz] = C;
                    Poz ++;
                    C = fgetc (Fi);
                  }
                Zodis [Poz] = 0;
                Ideti (&E, Zodis);
                while ((! feof (Fi)) && (! Simbolis (C)))
                  C = fgetc (Fi);
              }
            fclose (Fi);
            FILE *Fo = fopen (Ae [2], "wt");
            if (! Fo)
              printf ("Klaida !\nNegaliu sukurti elementu failo\n");
            else
              {
                Isvesti (E, Fo, 0);
                fclose (Fo);
              }
          }
      }
  }
```

8.8 Priedas 8. Testinė duomenų aibė 1

```
Duona,Saldainiai,Sviestas
Duona,Sviestas
Duona,Pienas,Sviestas
Alus,Duona
Alus,Pienas
```

8.9 Priedas 9. Testinė duomenų aibė 2

```
Duona,Saldainiai,Sviestas
Duona,Sviestas
Duona,Pienas,Sviestas
Alus,Duona
Duona,Sviestas
Alus,Pienas
Varske,Salotos
Duona,Sviestas
Duona,Sviestas,Desra
Alus,Cipsai,Riesutai
Duona,Alus,Sviestas
Alus,Riesutai
Alus,Duona,Riesutai,Sviestas
Duona,Pienas
Duona,Riesutai,Varske,Sviestas
Alus,Riesutai
Vynas,Saldainiai
Duona,Vynas,Sviestas
Duona,Sviestas,Desra
Duona,Salotos
Duona,Desra
Alus,Duona,Desra
Alus,Riesutai,Duona,Sviestas,Desra
```

8.10 Priedas 10. Testinė duomenų aibė 3

Viso duomenų aibėje yra 40 įrašų.

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.11 Priedas 11. Testinės duomenų aibės 4 dalis

Viso duomenų aibėje yra 42913 įrašų.

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.12 Priedas 12. Testinės duomenų aibės 5 dalis

Viso duomenų aibėje yra 45068 įrašai.

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.13 Priedas 13. Testinės duomenų aibės 6 dalis

Viso duomenų aibėje yra 20479 įrašai.

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.14 Priedas 14. Rezultatas 1

```
Daznos aibes :
(Duona,Sviestas) -  60.000000%
(Sviestas) -  60.000000%
(Pienas) -  40.000000%
(Duona) -  80.000000%
(Alus) -  40.000000%


Asociatyvus rysiai :
(Sviestas) => (Duona) - 100.000000%
(Duona) => (Sviestas) -  75.000000%
```

8.15 Priedas 15. Rezultatas 2

```
Daznos aibes :
(Duona,Riesutai,Sviestas) -  13.043478%
(Riesutai,Sviestas) -  13.043478%
(Desra,Duona,Sviestas) -  13.043478%
(Alus,Duona,Sviestas) -  13.043478%
(Duona,Sviestas) -  52.173913%
(Desra,Sviestas) -  13.043478%
(Alus,Sviestas) -  13.043478%
(Sviestas) -  52.173913%
(Duona,Riesutai) -  13.043478%
(Alus,Riesutai) -  21.739130%
(Riesutai) -  26.086957%
(Pienas) -  13.043478%
(Desra,Duona) -  21.739130%
(Alus,Duona) -  21.739130%
(Duona) -  73.913043%
(Desra) -  21.739130%
(Alus) -  39.130435%


Asociatyvus rysiai :
(Riesutai,Sviestas) => (Duona) - 100.000000%
(Duona,Riesutai) => (Sviestas) - 100.000000%
(Riesutai) => (Duona,Sviestas) -  50.000000%
(Riesutai) => (Sviestas) -  50.000000%
(Desra,Sviestas) => (Duona) - 100.000000%
(Desra,Duona) => (Sviestas) -  60.000000%
(Desra) => (Duona,Sviestas) -  60.000000%
(Alus,Sviestas) => (Duona) - 100.000000%
(Alus,Duona) => (Sviestas) -  60.000000%
```

```
(Sviestas) => (Duona) - 100.000000%
(Duona) => (Sviestas) -  70.588235%
(Desra) => (Sviestas) -  60.000000%
(Riesutai) => (Duona) -  50.000000%
(Riesutai) => (Alus) -  83.333333%
(Alus) => (Riesutai) -  55.555556%
(Desra) => (Duona) - 100.000000%
(Alus) => (Duona) -  55.555556%
```

8.16 Priedas 16. Rezultatas 3

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.17 Priedas 17. Rezultatas 4

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.18 Priedas 18. Rezultatas 5

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.19 Priedas 19. Rezultatas 6

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.20 Priedas 20. Rezultatas 7

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.21 Priedas 21. Rezultatas 8

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.22 Priedas 22. Rezultatas 9

Konfidenciali informacija priklausanti AB „Lietuvos telekomas“.

8.23 Priedas 23. Duomenų pertvarkymas į A-B sąrašą C++ kalba

```
// Pradines bazes pertvarkymas i sarasa A-B
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

unsigned char Zodis1 [50], Zodis2 [50];

int Simbolis (unsigned char C)
  {
    return ((C != '|') && (C != 10));
  }

void main (int Ak, char *Ae [])
  {
    printf ("Pradines bazes pertvarkymas i sarasa A-B\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ak != 3)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ae [0],
        "<pradine baze> <sutvarkyta baze>");
    else
      {
        FILE *Fi = fopen (Ae [1], "rt");
        if (! Fi)
          printf ("Klaida !\nNegaliu atidaryti pradines bazes failo\n");
        else
          {
            FILE *Fo = fopen (Ae [2], "wt");
            if (! Fo)
              printf ("Klaida !\nNegaliu sukurti sutvarkytos bazes failo\n");
            else
              {
                unsigned char C = fgetc (Fi);
                while (! feof (Fi))
                  {
                    for (int A = 0; A < 10; A ++)
                      {
                        while ((! feof (Fi)) && Simbolis (C))
                          C = fgetc (Fi);
                        while ((! feof (Fi)) && (! Simbolis (C)))
                          C = fgetc (Fi);
                      }
                    int Poz1 = 0, Poz2 = 0;
                    while ((! feof (Fi)) && Simbolis (C))
                      {
                        Zodis1 [Poz1 ++] = C;
                        C = fgetc (Fi);
                      }
                    Zodis1 [Poz1] = 0;
                    while ((! feof (Fi)) && (! Simbolis (C)))
                      C = fgetc (Fi);
                    for (int A = 0; A < 2; A ++)
                      {
                        while ((! feof (Fi)) && Simbolis (C))
                          C = fgetc (Fi);
                        while ((! feof (Fi)) && (! Simbolis (C)))
                          C = fgetc (Fi);
                      }
                    while ((! feof (Fi)) && Simbolis (C))
                      {
                        Zodis2 [Poz2 ++] = C;
```

```
                    C = fgetc (Fi);
                  }
                Zodis2 [Poz2] = 0;
                while ((! feof (Fi)) && (C != 10))
                  C = fgetc (Fi);
                if (! feof (Fi))
                  C = fgetc (Fi);
                if ((Poz1 == 9) && (Zodis1 [0] == '8'))
                  {
                    if ((Poz2 == 9) && (Zodis2 [0] == '8'))
                      fprintf (Fo, "%s,%s\n", Zodis1, Zodis2);
                    if ((Poz2 == 13) && (Zodis2 [2] == 'E') &&
                        (Zodis2 [4] == '8'))
                      fprintf (Fo, "%s,%s\n", Zodis1, Zodis2 + 4);
                    if ((Poz2 == 8) && (Zodis2 [0] != '8'))
                      fprintf (Fo, "%s,8%s\n", Zodis1, Zodis2);
                  }
              }
            fclose (Fo);
          }
        fclose (Fi);
      }
  }
}
```

8.24 Priedas 24. A-B sąrašo pertvarkymas į kodus C++ kalba

```
// Saraso A-B pertvarkymas i A-B kodu sarasa
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

unsigned char Zodis1 [50], Zodis2 [50];

void main (int Ak, char *Ae [])
  {
    printf ("Saraso A-B pertvarkymas i A-B kodu sarasa\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ak != 3)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ae [0],
        "<pradine baze> <sutvarkyta baze>");
    else
      {
        FILE *Fi = fopen (Ae [1], "rt");
        if (! Fi)
          printf ("Klaida !\nNegaliu atidaryti pradines bazes failo\n");
        else
          {
            FILE *Fo = fopen (Ae [2], "wt");
            if (! Fo)
              printf ("Klaida !\nNegaliu sukurti sutvarkytos bazes failo\n");
            else
              {
                unsigned char C = fgetc (Fi);
                while (! feof (Fi))
                  {
                    int Poz = 0;
                    while ((! feof (Fi)) && (C != ','))
                      {
                        Zodis1 [Poz ++] = C;
                        C = fgetc (Fi);
                      }
                    Zodis1 [4] = 0;
                    while ((! feof (Fi)) && (C == ','))
                      C = fgetc (Fi);
                    Poz = 0;
                    while ((! feof (Fi)) && (C != 10))
                      {
                        Zodis2 [Poz ++] = C;
                        C = fgetc (Fi);
                      }
                    Zodis2 [4] = 0;
                    if ((Zodis1 [1] != Zodis2 [1]) || (Zodis1 [2] !=
                         Zodis2 [2]) || (Zodis1 [3] != Zodis2 [3]))
                      fprintf (Fo, "%s,%s\n", Zodis1, Zodis2);
                    if (! feof (Fi))
                      C = fgetc (Fi);
                  }
                fclose (Fo);
              }
            fclose (Fi);
          }
      }
  }
```

8.25 Priedas 25. A-B sąrašo pertvarkymas į A pradžias

```
// Saraso A-B pertvarkymas i A sarasa
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

const int SmS = ('9' - '0') + 2;
const int SmR = ('9' - '0') + ('Z' - 'A') + 3;
const int SmK = ('9' - '0') + ('Z' - 'A') + ('z' - 'a') + 4;

unsigned char Simb (unsigned char A)
  {
    return (A == '_') ? 0 : ((A >= '0') && (A <= '9')) ? A - '0' + 1 :
      ((A >= 'A') && (A <= 'Z')) ? A - 'A' + SmS :
      ((A >= 'a') && (A <= 'z')) ? A - 'a' + SmR : 0;
  }

unsigned char Sima (unsigned char A)
  {
    return (A == 0) ? '_' : (A < SmS) ? A - 1 + '0' :
      (A < SmR) ? A - SmS + 'A' : A - SmR + 'a';
  }

struct ElSar
  {
    char Yra;
    ElSar *Sek [SmK];
    ElSar *Zod;
    ElSar ();
    ~ElSar ();
  };

ElSar::ElSar ()
  {
    Yra = 0;
    for (int A = 0; A < SmK; A ++)
      Sek [A] = NULL;
    Zod = NULL;
  }

ElSar::~ElSar ()
  {
    for (int A = 0; A < SmK; A ++)
      if (Sek [A])
        delete Sek [A];
    if (Zod)
      delete Zod;
  }

void Ideti2 (ElSar **E, unsigned char *c)
  {
    if (! (*E))
      (*E) = new ElSar;
    if (! *c)
      {
        (*E) -> Yra = 1;
      }
    else
      Ideti2 (& ((*E) -> Sek [Simb (*c)]), (c + 1));
  }
```

```
void Ideti (ElSar **E, unsigned char *c, unsigned char *d)
  {
    if (! (*E))
      (*E) = new ElSar;
    if (! *c)
      {
        (*E) -> Yra = 1;
        Ideti2 (& ((*E) -> Zod), d);
      }
    else
      Ideti (& ((*E) -> Sek [Simb (*c)]), (c + 1), d);
  }

unsigned char Zodis1 [50], Zodis2 [50];
int Buvo;

void Isvesti (ElSar *E, FILE *Fo, int Poz)
  {
    Zodis1 [Poz] = 0;
    if (E -> Yra)
      {
        if (Buvo)
          fprintf (Fo, ",");
        fprintf (Fo, "%s", (char *) Zodis1);
        Buvo = 1;
      }
    for (int A = 0; A < SmK; A ++)
      if (E -> Sek [A])
        {
          Zodis1 [Poz] = Sima (A);
          Isvesti (E -> Sek [A], Fo, Poz + 1);
        }
  }

void Isvesti2 (ElSar *E, FILE *Fo)
  {
    if (E -> Yra)
      {
        Buvo = 0;
        Isvesti (E -> Zod, Fo, 0);
        fprintf (Fo, "\n");
      }
    for (int A = 0; A < SmK; A ++)
      if (E -> Sek [A])
        Isvesti2 (E -> Sek [A], Fo);
  }


int Simbolis (unsigned char C)
  {
    return ((C != '|') && (C != 10));
  }

void main (int Ak, char *Ae [])
  {
    printf ("Saraso A-B pertvarkymas i A sarasa\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ak != 3)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ae [0],
        "<pradine baze> <sutvarkyta baze>");
    else
      {
        FILE *Fi = fopen (Ae [1], "rt");
        if (! Fi)
```

```
        printf ("Klaida !\nNegaliu atidaryti pradines bazes failo\n");
      else
        {
          FILE *Fo = fopen (Ae [2], "wt");
          if (! Fo)
            printf ("Klaida !\nNegaliu sukurti sutvarkytos bazes failo\n");
          else
            {
              ElSar *E;
              E = NULL;
              unsigned char C = fgetc (Fi);
              while (! feof (Fi))
                {
                  int Poz = 0;
                  while ((! feof (Fi)) && (C != ','))
                    {
                      Zodis1 [Poz ++] = C;
                      C = fgetc (Fi);
                    }
                  Zodis1 [4] = 0;
                  while ((! feof (Fi)) && (C == ','))
                    C = fgetc (Fi);
                  Poz = 0;
                  while ((! feof (Fi)) && (C != 10))
                    {
                      Zodis2 [Poz ++] = C;
                      C = fgetc (Fi);
                    }
                  Zodis2 [Poz] = 0;
                  Ideti (&E, Zodis2, Zodis1);
                  if (! feof (Fi))
                    C = fgetc (Fi);
                }
              Isvesti2 (E, Fo);
              fclose (Fo);
            }
          fclose (Fi);
        }
    }
}
```

8.26 Priedas 26. A-B sąrašo pertvarkymas į B pradžias

```
// Saraso A-B pertvarkymas i B sarasa
// (c) 2004 Valdas Ambraziunas

#include <stdio.h>

const int SmS = ('9' - '0') + 2;
const int SmR = ('9' - '0') + ('Z' - 'A') + 3;
const int SmK = ('9' - '0') + ('Z' - 'A') + ('z' - 'a') + 4;

unsigned char Simb (unsigned char A)
  {
    return (A == '_') ? 0 : ((A >= '0') && (A <= '9')) ? A - '0' + 1 :
      ((A >= 'A') && (A <= 'Z')) ? A - 'A' + SmS :
      ((A >= 'a') && (A <= 'z')) ? A - 'a' + SmR : 0;
  }

unsigned char Sima (unsigned char A)
  {
    return (A == 0) ? '_' : (A < SmS) ? A - 1 + '0' :
      (A < SmR) ? A - SmS + 'A' : A - SmR + 'a';
  }

struct ElSar
  {
    char Yra;
    ElSar *Sek [SmK];
    ElSar *Zod;
    ElSar ();
    ~ElSar ();
  };

ElSar::ElSar ()
  {
    Yra = 0;
    for (int A = 0; A < SmK; A ++)
      Sek [A] = NULL;
    Zod = NULL;
  }

ElSar::~ElSar ()
  {
    for (int A = 0; A < SmK; A ++)
      if (Sek [A])
        delete Sek [A];
    if (Zod)
      delete Zod;
  }

void Ideti2 (ElSar **E, unsigned char *c)
  {
    if (! (*E))
      (*E) = new ElSar;
    if (! *c)
      {
        (*E) -> Yra = 1;
      }
    else
      Ideti2 (& ((*E) -> Sek [Simb (*c)]), (c + 1));
  }
```

```
void Ideti (ElSar **E, unsigned char *c, unsigned char *d)
  {
    if (! (*E))
      (*E) = new ElSar;
    if (! *c)
      {
        (*E) -> Yra = 1;
        Ideti2 (& ((*E) -> Zod), d);
      }
    else
      Ideti (& ((*E) -> Sek [Simb (*c)]), (c + 1), d);
  }

unsigned char Zodis1 [50], Zodis2 [50];
int Buvo;

void Isvesti (ElSar *E, FILE *Fo, int Poz)
  {
    Zodis1 [Poz] = 0;
    if (E -> Yra)
      {
        if (Buvo)
          fprintf (Fo, ",");
        fprintf (Fo, "%s", (char *) Zodis1);
        Buvo = 1;
      }
    for (int A = 0; A < SmK; A ++)
      if (E -> Sek [A])
        {
          Zodis1 [Poz] = Sima (A);
          Isvesti (E -> Sek [A], Fo, Poz + 1);
        }
  }

void Isvesti2 (ElSar *E, FILE *Fo)
  {
    if (E -> Yra)
      {
        Buvo = 0;
        Isvesti (E -> Zod, Fo, 0);
        fprintf (Fo, "\n");
      }
    for (int A = 0; A < SmK; A ++)
      if (E -> Sek [A])
        Isvesti2 (E -> Sek [A], Fo);
  }


int Simbolis (unsigned char C)
  {
    return ((C != '|') && (C != 10));
  }

void main (int Ak, char *Ae [])
  {
    printf ("Saraso A-B pertvarkymas i B sarasa\n");
    printf ("(c) 2004 Valdas Ambraziunas\n\n");
    if (Ak != 3)
      printf ("Klaida !\nNaudojimas : %s %s\n", Ae [0],
        "<pradine baze> <sutvarkyta baze>");
    else
      {
        FILE *Fi = fopen (Ae [1], "rt");
        if (! Fi)
```

```
        printf ("Klaida !\nNegaliu atidaryti pradines bazes failo\n");
      else
        {
          FILE *Fo = fopen (Ae [2], "wt");
          if (! Fo)
            printf ("Klaida !\nNegaliu sukurti sutvarkytos bazes failo\n");
          else
            {
              ElSar *E;
              E = NULL;
              unsigned char C = fgetc (Fi);
              while (! feof (Fi))
                {
                  int Poz = 0;
                  while ((! feof (Fi)) && (C != ','))
                    {
                      Zodis1 [Poz ++] = C;
                      C = fgetc (Fi);
                    }
                  Zodis1 [Poz] = 0;
                  while ((! feof (Fi)) && (C == ','))
                    C = fgetc (Fi);
                  Poz = 0;
                  while ((! feof (Fi)) && (C != 10))
                    {
                      Zodis2 [Poz ++] = C;
                      C = fgetc (Fi);
                    }
                  Zodis2 [4] = 0;
                  Ideti (&E, Zodis1, Zodis2);
                  if (! feof (Fi))
                    C = fgetc (Fi);
                }
              Isvesti2 (E, Fo);
              fclose (Fo);
            }
          fclose (Fi);
        }
    }
}
```

8.27 Priedas 27. Algoritmų greitaveikos įrašai

```
File Data_1_Met_1.Run:
Paleidimo laikas :  4:03:30.84
Pabaigos laikas :   4:03:30.89
File Data_1_Met_2.Run:
Paleidimo laikas :  4:03:39.60
Pabaigos laikas :   4:03:39.65
File Data_1_Met_3.Run:
Paleidimo laikas :  4:03:48.75
Pabaigos laikas :   4:03:48.79
File Data_1_Met_4.Run:
Paleidimo laikas :  4:04:11.78
Pabaigos laikas :   4:04:11.78
File Data_2_Met_1.Run:
Paleidimo laikas :  4:03:30.89
Pabaigos laikas :   4:03:30.89
File Data_2_Met_2.Run:
Paleidimo laikas :  4:03:39.67
Pabaigos laikas :   4:03:39.67
File Data_2_Met_3.Run:
Paleidimo laikas :  4:03:48.79
Pabaigos laikas :   4:03:48.81
File Data_2_Met_4.Run:
Paleidimo laikas :  4:04:11.78
Pabaigos laikas :   4:04:11.79
File Data_3_Met_1.Run:
Paleidimo laikas :  4:03:30.90
Pabaigos laikas :   4:03:39.60
File Data_3_Met_2.Run:
Paleidimo laikas :  4:03:39.67
Pabaigos laikas :   4:03:39.68
File Data_3_Met_3.Run:
Paleidimo laikas :  4:03:48.81
Pabaigos laikas :   4:03:48.82
File Data_3_Met_4.Run:
Paleidimo laikas :  4:04:11.79
Pabaigos laikas :   4:04:11.79
File Data_4_Met_2.Run:
Paleidimo laikas :  4:03:39.68
Pabaigos laikas :   4:03:44.90
File Data_4_Met_3.Run:
Paleidimo laikas :  4:03:48.82
Pabaigos laikas :   4:04:03.96
File Data_4_Met_4.Run:
Paleidimo laikas :  4:04:11.81
Pabaigos laikas :   4:04:17.35
File Data_5_Met_2.Run:
Paleidimo laikas :  4:03:44.90
Pabaigos laikas :   4:03:45.84
File Data_5_Met_3.Run:
Paleidimo laikas :  4:04:03.96
Pabaigos laikas :   4:04:05.65
File Data_5_Met_4.Run:
Paleidimo laikas :  4:04:17.37
Pabaigos laikas :   4:04:18.35
File Data_6_Met_2.Run:
Paleidimo laikas :  4:03:45.84
Pabaigos laikas :   4:03:48.75
File Data_6_Met_3.Run:
Paleidimo laikas :  4:04:05.67
Pabaigos laikas :   4:04:11.78
```

```
File Data_6_Met_4.Run:
Paleidimo laikas :  4:04:18.35
Pabaigos laikas :   4:04:21.42
File Data_7_Met_2.Run:
Paleidimo laikas :  9:02:07.14
Pabaigos laikas :   9:57:05.79
File Data_7_Met_3.Run:
Paleidimo laikas :  6:23:55.50
Pabaigos laikas :   7:59:12.71
File Data_7_Met_4.Run:
Paleidimo laikas :  4:54:49.17
Pabaigos laikas :   5:48:45.50
File Data_8_Met_2.Run:
Paleidimo laikas :  9:57:05.79
Pabaigos laikas :  10:02:27.57
File Data_8_Met_3.Run:
Paleidimo laikas :  7:59:12.71
Pabaigos laikas :   8:09:36.25
File Data_8_Met_4.Run:
Paleidimo laikas :  5:48:45.50
Pabaigos laikas :   5:55:08.20
File Data_9_Met_2.Run:
Paleidimo laikas : 10:02:27.57
Pabaigos laikas :  10:31:51.51
File Data_9_Met_3.Run:
Paleidimo laikas :  8:09:36.25
Pabaigos laikas :   9:02:07.14
File Data_9_Met_4.Run:
Paleidimo laikas :  5:55:08.20
Pabaigos laikas :   6:23:55.50
```