Šiaulių universitetas

Matematikos ir informatikos fakultetas

Vidmantas Malakauskas

Informatikos specialybės V kurso neakivaizdinio skyriaus studentas

KOMBINATORIKOS ELEMENTŲ GENERAVIMO ALGORITMŲ SUDĖTINGUMO TYRIMAS

THE RESEARCH OF COMPLEXITY OF COMBINATIONS THEORY ALGORITHMS

BAKALAURO DARBAS

Darbo vadovas:

Doc. dr. V. Sirius

Recenzentas:

Dr. K. Žilinskas

Šiauliai, 2010

„Tvirtinu, jog darbe pateikta medžiaga nėra plagijuota ir paruošta naudojant literatūros sąraše pateiktus informacinius šaltinius bei savo tyrimų duomenis“

Darbo autorius:

Darbo tikslai ir uždaviniai

Tikslas – ištirti kombinatorikos elementų (kėlinių, derinių, poaibių ir Grėjaus kodų) generavimo algoritmų sudėtingumą.

Uždaviniai:

1. Išanalizuoti žemiau išvardintus algoritmus:
   a) kėlinių generavimas leksikografine, antileksikografine, minimalaus pokyčio tvarka;
   b) derinių generavimas leksikografine, minimalaus pokyčio tvarka;
   c) poaibio generavimo algoritmai: poaibių generavimas leksikografine tvarka bei Grėjaus kodų generavimas.
2. Sudaryti minėtų algoritmų blokines schemas.
3. Sukurti programą realizuojančią minėtus algoritmus.
4. Ištirti algoritmų sudėtingumą.

Darbo vadovas:

TURINYS

ĮVADAS ... 6

1. TEORINĖ DALIS ... 7

1.1 Poaibių generavimo algoritmai ... 7

1.1.1 Poaibių generavimo leksikografine didėjimo tvarka algoritmas ... 9

1.1.2 Grėjaus kodo generavimo algoritmai ... 10

1.1.3 Pirmasis Grėjaus kodo generavimo algoritmas ... 10

1.1.4 Antrasis Grėjaus kodo generavimo algoritmas ... 11

1.1.4 Trečiasis Grėjaus kodų generavimo algoritmas ... 13

1.2 Derinių generavimo algoritmai ... 15

1.2.1 Derinių generavimas leksikografine didėjimo tvarka ... 15

1.2.2 Derinių generavimas minimalaus pokyčio tvarka ... 16

1.3 Kėlinių generavimo algoritmai ... 21

1.3.1 Kėlinių generavimas leksikografine tvarka ... 21

1.3.2 Kėlinių generavimas antileksikografine tvarka ... 22

1.3.3 Kėlinių generavimas minimalaus pokyčio tvarka. ... 23

1.4 Algoritmų sudėtingumas ... 25

2 PROJEKTINĖ DALIS ... 31

2.1 Projekto vykdymo planas ... 31

2.2 Įrankių ir priemonių analizė ... 31

2.3 Algoritmų sudėtingumo tyrimas ... 32

2.3.1 Kėlinių generavimo algoritmų sudėtingumo tyrimas ... 33

2.3.2 Derinių generavimo algoritmų sudėtingumo tyrimas 36

2.3.3 Poaibių (Grėjaus kodų) generavimo algoritmų sudėtingumo tyrimas 39

3 DARBO EIGOS APRAŠYMAS 44

3.1 Darbų eigos grafas 44

3.2 Problemų sprendimas 45

IŠVADOS 47

Literatūros ir informacinių šaltinių sąrašas 48

Anotacija (Summary) 49

Priedai 50

1 priedas. Poaibių generavimo leksikografine didėjimo tvarka algoritmo blokinė schema 50

2 priedas. Pirmojo Grėjaus kodų generavimo algoritmo blokinė schema 51

3 priedas. Antrojo Grėjaus kodų generavimo algoritmo blokinė schema 52

4 priedas. Trečiojo Grėjaus kodų generavimo algoritmo blokinė schema 53

5 priedas. Derinių generavimo leksikografine tvarka algoritmo blokinė schema 54

6 priedas. Derinių generavimo minimalaus pokyčio tvarka algoritmo blokinė schema 55

7 priedas. Kėlinių generavimo leksikografine tvarka algoritmo blokinė schema 56

8 priedas. Kėlinių generavimo antileksikografine tvarka algoritmo blokinė schema 57

9 priedas. Kėlinių generavimo minimalaus pokyčio tvarka algoritmo blokinė schema 58

10.1 priedas. Algoritmo „Rezultatai“ skirto derinių generavimo leksikografine tvarka bei visiems kėlinių algoritmams blokinė schema 59

10.2 priedas. Algoritmo „Rezultatai“ skirto likusiems algoritmams blokinė schema 60

11 priedas. Programos blokinė schema 61

12 priedas. Programos naudojimo instrukcija 62

ĮVADAS

Darbo tikslas – ištirti kombinatorikos elementų (kėlinių, derinių, poaibių ir Grėjaus kodų) generavimo algoritmų sudėtingumą.

Uždaviniai šiam tikslui pasiekti:

1. Išanalizuoti žemiau išvardintus algoritmus:
    a) kėlinių generavimas leksikografine, antileksikografine, minimalaus pokyčio tvarka;
    b) derinių generavimas leksikografine, minimalaus pokyčio tvarka;
    c) poaibio generavimo algoritmai: poaibių generavimas leksikografine tvarka bei Grėjaus kodų generavimas.
2. Sudaryti minėtų algoritmų blokines schemas.
3. Sukurti programą realizuojančią minėtus algoritmus.
4. Ištirti algoritmų sudėtingumą.

1. TEORINĖ DALIS

1.1 Poaibių generavimo algoritmai

Aibė – objektų, kuriems būdingas tam tikras požymis, visuma. Aibės paprastai žymimos didžiosiomis lotyniškomis raidėmis, pavyzdžiui, *A, B, C, D* ir pan. Matematikoje dažniausiai naudojamų aibių simbolika yra nusistovėjusi. Pavyzdžiui:

N – natūraliųjų skaičių aibė,

Z – sveikųjų skaičių aibė,

Q – racionaliųjų skaičių aibė,

R – realiųjų skaičių aibė,

C – kompleksinių skaičių aibė,

$\varnothing$ - tuščioji aibė.

Paprastai aibės nusakomos tokiais būdais:

- Išvardinimo,
- Aprašymo,
- Grafiniu.

Išvardinimo būdu aibė aprašoma išvardinant visus jos elementus. Pavyzdžiui, $A = \{a_1, a_2, a_3\}$, $X = \{x_1, x_2, ..., x_n\}$.

Poaibis. Aibė A yra aibės B poaibis (žymime $A \subseteq B$), jei kiekvienas aibės A elementas yra aibės B elementas. Aibės visų galimų poaibių aibė žymima $P(A)$. $P(A) = \{X \mid X \subseteq A\}$. Pavyzdžiui, jei $A = \{1, 2, 3\}$, tai:

$P(A) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Jei $|A| = n$, tai $|P(A)| = 2^n$.

Visus poaibius galima nusakyti dvejetainiais kodais. Pavyzdžiui:

1 lentelė.

Kodas	Poaibis
000	$\varnothing$
001	$\{a_3\}$
010	$\{a_2\}$
011	$\{a_2,a_3\}$
100	$\{a_1\}$
101	$\{a_1,a_3\}$
110	$\{a_1, a_2\}$
111	$\{a_1, a_2, a_3\}$

Bendra poaibių generavimo algoritmų struktūra.

Poaibių, kaip ir kitų kombinatorinių objektų, generavimo algoritmo struktūra yra tokia:

```
begin
    generuoti pradinį poaibį;
    while „generavimo sąlyga“ do
        begin
            nagrinėti (spausdinti) poaibį;
            generuoti poaibį, gretimą išnagrinėtam (atspausdintam) poaibiui;
        end;
end;
```

Priklausomai nuo pasirinktos poaibių generavimo tvarkos šiame algoritme turi būti konkretizuoti sakiniai: „generuoti pradinį poaibį“, „generavimo sąlyga“, „poaibis, gretimas išnagrinėtam poaibiui“ [1].

1.1.1 Poaibių generavimo leksikografine didėjimo tvarka algoritmas

Tarkime, kad n – aibės elementų skaičius, $a[1,...,n]$ – aibės A elementai. Turime sugeneruoti aibės A poaibius leksikografine tvarka.

Kiekvieną aibės A poaibį galima nusakyti dvejetainiu kodu $b_1, b_2, \ldots, b_n$, $b_i \in \{0,1\}$. Jei kodo elementas $b_i = 1$, tai elementas a_i priklauso poaibiui. Tuo būdu poaibį nusakys masyvas $b[0...n]$.

Į kodą galima žiūrėti kaip į vektorių. Vektorius x yra leksikografiškai didesnis už 0 ($x > 0$), jei jo pirmoji nenulinė komponentė yra teigiama.

Vektorius $x > y$, jei $x - y > 0$.

Iš vektorių leksikografinės tvarkos išplaukia, kad, jei į kodą žiūrėsime, kaip į dvejetainį skaičių, tai kodai sudarys visus n skilčių dvejetainius skaičius, išdėstytus didėjimo tvarka. Pavyzdžiui, jei $n = 4$, tai kodai išdėstyti leksikografine didėjimo tvarka atrodys taip: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 [1].

Algoritmą sudaro trys pagrindiniai žingsniai:

1. Pradinio kodo generavimas. $b_i = 0$; $i = \overline{0,n}$. Sugeneruojama tuščia aibė, kurią atitinka dvejetainis kodas sudarytas iš 0.

2. Gretimo kodo generavimas. Gretimą poaibį nusako kodas – dvejetainis skaičius, vienetu didesnis nei išnagrinėtasis. Masyve b ieškome pirmo iš dešinės elemento, kuris lygus nuliui, padidiname jo reikšmę vienetu, o visų kairiau esančių elementų reikšmę prilyginame nuliui. Atspausdiname gautą poaibį. Jei tenkinama generavimo sąlyga, tęsiamas kodų (poaibių) generavimas.

3. Generavimas baigiamas, kai sugeneruojamas poaibis iš visų elementų. Dvejetainio kodo visi elementai turi reikšmę $b_i = 1$.

Algoritmo blokinė schema pateikiama 1 priede.

1.1.2 Grėjaus kodo generavimo algoritmai

Grėjaus kodai – poaibių aibė, kurioje bet kokie du gretimi poaibiai skiriasi tik vienu elementu. Grėjaus sekos naudojamos įvairiose srityse: kodavimo teorijoje, lygiagretiesiems kompiuteriams konstruoti, tinklų inžinerijoje. Literatūroje minimi trys Grėjaus kodo generavimo algoritmai.

1.1.3 Pirmasis Grėjaus kodo generavimo algoritmas

Šis algoritmas remiasi aukščiau išnagrinėtu poaibių generavimo leksikografine tvarka algoritmu. Sugeneruoti dvejetainiai skaičiai perskaičiuojami į Grėjaus kodus. Dvejetainių skaičių poaibis $b_1, b_2, ..., b_n$ perskaičiuojamas į Grėjaus kodą $c_1, c_2, ..., c_n$ pagal formulę:

$c_1 = b_1$;

$c_i = b_{i-1} \oplus b_i,\ i = \overline{2,n}$

Simbolis $\oplus$ žymi XOR funkciją (griežtoji disjunkcija).

Dvejetainio ir Grėjaus kodo pirmosios iš kairės skilties reikšmė yra vienoda. Antrosios iš dešinės Grėjaus kodo skilties reikšmė apskaičiuojama atlikus griežtos disjunkcijos loginę operaciją (XOR) tarp dvejetainio kodo pirmosios ir antrosios skilties reikšmių. Trečiosios Grėjaus kodo skilties reikšmė apskaičiuojama atlikus XOR operaciją tarp dvejetainio kodo antrosios ir trečiosios skilties reikšmių. Skaičiavimai tokia pačia tvarka atliekami tol, kol apskaičiuojama paskutiniosios Grėjaus kodo skilties reikšmė.

Aukščiau pateiktas poaibių generavimo leksikografine didėjimo tvarka algoritmas tiks Grėjaus kodų generavimui, jei prieš spausdindami rezultatus pagal formulę apskaičiuosime masyvo c[1..n] elementus [1].

Jei n = 3, perskaičiavimą iš dvejetainio kodo į Grėjaus kodą iliustruoja žemiau pateikta 2 lentelė.

2 lentelė.

Dvejetainis kodas	Grėjaus kodas
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Pirmojo Grėjaus kodo generavimo algoritmo blokinė schema pateikta 2 priede.

1.1.4 Antrasis Grėjaus kodo generavimo algoritmas

Jei aibė turi vieną elementą, tai Grėjaus kodai bus 0 ir 1. Turėdami Grėjaus kodus aibei, turinčiai (n-1) elementų, generuosime Grėjaus kodus aibei, turinčiai n elementų tokiu būdu:

1) prie aibės, turinčios (n-1) elementą, Grėjaus kodų iš dešinės prirašome „0“;

2) po to prie aibės iš (n-1) elementų Grėjaus kodų, rašomų atvirkščia tvarka, dešinės prirašome „1“.

Tokiu būdu gauname:

Jei n = 1, tai Grėjaus kodai yra 0, 1;

Jei n = 2, tai Grėjaus kodai yra 00, 10, 11, 01;

Jei n = 3, tai Grėjaus kodai yra 000, 100, 110, 010, 011, 111, 101, 001 ir t. t.

Pabraukti simboliai yra Grėjaus kodai aibei iš (n-1) elemento. Šis metodas yra paprastas, bet neatitinka bendros poaibių generavimo schemos. Naudosime realizaciją pateiktą Lipskij knygoje [2].

Funkcija $Q(i)$, $i \in N$ – tai toks didžiausias dvejeto laipsnis, kad $2^{Q(i)}$ yra i daliklis, t. y. i mod $2^{Q(i)} = 0$. Pavyzdžiui, $Q(3) = 0$; $Q(4) = 2$; $Q(5) = 0$; $Q(6) = 1$; $Q(8) = 3$; $Q(12) = 2$.

Funkcijos Q(i) savybė. $Q(2^k + m) = Q(2^k - m)$, kai $0 \leq m \leq 2^k - 1$.

Pavyzdžiui, kai $k = 2$, tai

$Q(4 + 0) = Q(4 - 0) = 2$,

$Q(4 + 1) = Q(4 - 1) = 0$,

$Q(4 + 2) = Q(4 - 2) = 1$,

$Q(4 + 3) = Q(4 - 3) = 0$.

Remdamiesi funkcija $Q(i)$ sudarysime Grėjaus kodų generavimo algoritmą.

Įėjimo duomenys:

n – aibės elementų skaičius;

$a[1.. n]$ – aibės A elementai.

Antrojo Grėjaus algoritmo blokinė schema su komentarais pateikta 3 priede.

3 lentelė. Grėjaus kodų generavimas, kai $n = 3$

i	*b1*	*b2*	*b3*	$p = 1 + Q(i)$
0	0	0	0	
1	1	0	0	1
2	1	1	0	2
3	0	1	0	1
4	0	1	1	3
5	1	1	1	1
6	1	0	1	2
7	0	0	1	1
8				4 ($p > 3$)

Generuojant Grėjaus kodus aibei iš n elementų, kai $i \leq 2^{n-1} - 1$, generuojami Grėjaus kodai iš (n-1) elementų. Šiems kodams pozicija b_n yra lygi nuliui. Kai $i = 2^{n-1} - 1$, b_n įgauna reikšmę 1 ir toliau, didinant i reikšmę, atvirkščia tvarka generuojami Grėjaus kodai aibei iš (n-1) elemento.

1.1.4 Trečiasis Grėjaus kodų generavimo algoritmas

Kaip matyti iš antrojo Grėjaus kodų generavimo algoritmo (žr. 1.1.4 skyrelį), Grėjaus kodus galima generuoti iš pradinio kodo, žinant seką $T_n = t_1, t_2, ..., t_i, ..., t_{2n-1}$. Šios sekos elementas t_i žymi skiltį, kurią reikia invertuoti, kad iš (i – 1)-ojo kodo gautume i-ąjį kodą.

Pavyzdžiui, kai n = 3, seka T_3 = 1,2,1,3,1,2,1.

Vadinasi, pakanka mokėti efektyviai generuoti seką T_n, kad galėtume generuoti Grėjaus kodus. Seką T_n galima efektyviai generuoti naudojant steką.

Pradžioje stekas užpildomas elementais: n, n – 1, n – 2, ..., 3, 2, 1 (viršuje yra elementas 1). Algoritmas ima viršutinį steko elementą i ir patalpina jį į seką T_n; po to į steką įrašomi elementai i – 1, i – 2, ..., 1. Būtent tokiu būdu žemiau pateiktas algoritmas generuoja Grėjaus kodus [1]. $\overline{g_i}$ reiškia – g_i.

1 paveikslas. a) Trečiasis Grėjaus algoritmas [3]

S – tuščias stekas

for j = n + 1 **to** 1 **by** -1 $\begin{cases} g_j \leftarrow \\ 0 \end{cases}$

while i < n + 1 $\begin{cases} \text{spausdinti } (g_n g_{n-1} \ldots g_1) \\ i \Leftarrow S \\ g_i \leftarrow \overline{g_i} \end{cases}$

Skaičiuojant pagal šį algoritmą, steko turinys (viršutinis elementas yra steko viršūnėje) kis taip:

1							
2	2	1		1			
3	3	3	3	2	2	1	0

, o i reikšmės sudarys T_3 seką 1,2,1,3,1,2,1.

Pateiktą algoritmą galima patobulinti. Kiekvieną kartą, kai į steką patalpinamas elementas $i > 0$, mes apriori žinome, kad virš jo bus patalpinti elementai $j - 1, ..., 1$. Protingas šios informacijos panaudojimas įgalina vietoj steko naudoti masyvą $\tau = (\tau_n, \tau_{n-1}, ..., \tau_1, \tau_0)$ tokiu būdu. Elementas τ_0 yra viršutinis steko elementas ir kiekvienam $j > 0$ τ_j yra elementas, steke esantis tuoj po elemento j (žemiau elemento j), jei j yra steke. Jei elemento j steke nėra, tai elemento τ_j reikšmė negali turėti jokios įtakos skaičiavimams ir todėl τ_j priskiriame reikšmę $j + 1$, kadangi mes žinome, kad kai kitą kartą elementas $j + 1$ bus patalpintas į steką, elementu, esančiu virš jo, bus elementas j. Dar daugiau, kadangi elementai $i - 1, i - 2, ..., 1$ bus talpinami į steką pašalinus elementą i , mums reikia tiktai priskirti $\tau_{i-1} = \tau_i$, laikant, kad visiems j, $1 \leq j \leq i - 2$, reikšmė τ_j buvo priskirta, kai j buvo pašalintas iš steko (primename, tada $\tau_j = j + 1$). Pagaliau, kai $i \neq 1$, elementai į steką patalpinami operacijos $\tau_0 = 1$ dėka.

Dabar gausime tokį algoritmą.

2 paveikslas. b) Trečiasis Grėjaus algoritmas [3]

$$\textbf{for } j = 0 \textbf{ to } n + 1 \begin{cases} g_j \leftarrow 0 \\ \tau_i \leftarrow j + 1 \end{cases}$$

$$i \leftarrow 0$$

$$\textbf{while } i < n + 1 \begin{cases} \text{spausdinti } (g_n g_{n-1} \ldots g_1) \\ i \leftarrow \tau_0 \\ g_i \leftarrow \overline{g_i} \\ \tau_{i-1} \leftarrow \tau_i \\ \tau_i \leftarrow i + 1 \end{cases}$$

Kai $n = 3$, masyvo τ turinys skaičiavimo eigoje kis taip:

4 lentelė. Seka T_3

τ_4	5	5	5	5	5	5	5	5
τ_3	4	4	4	4	4	4	4	4
τ_2	3	3	3	3	4	4	4	4
τ_1	2	2	3	2	2	2	4	2
τ_0	1	2	1	3	1	2	1	4

Pateiktąjį algoritmą galima šiek tiek patobulinti. Pastebėsim, jei operatorių $\tau_0 = 1$ patalpintume po operatoriaus $g[i] = 1 - g[i]$, tai operatorių if $i \neq 1$ then $\tau[0] = 1$ galima išmesti. Jei $i = 1$, tai operatorius $\tau[i-1] = \tau[i]$ tuoj pat ištaisys neteisingą $\tau[0]$ reikšmę [1]. Galutinis Grėjaus kodų generavimo algoritmas pavaizduotas 4 priede.

1.2 Derinių generavimo algoritmai

1.2.1 Derinių generavimas leksikografine didėjimo tvarka

Aptarsime derinių generavimo leksikografine didėjimo tvarka algoritmą. Leksikografinė vektorių tvarka aptarta skyrelyje apie poaibių generavimą leksikografine tvarka. Jei į derinį žiūrėsime, kaip į skaičių, tai deriniams atitinkantys skaičiai bus išdėstyti didėjimo tvarka.

Žemiau surašyti C_5^3 deriniai leksikografine didėjimo tvarka:

123, 124, 125, 134, 135, 145, 234, 235, 245, 345.

Derinius C_n^k nuosekliai talpinsime masyve $c[1..k]$ ir šio masyvo $c_1, c_2, \ldots, c_k$ elementai apibrėš derinį:

for $m = 1$ to k do $c[m] = m$;

Paties mažiausio leksikografiškai didesnio derinio generavimas. Tarkime $c_1, c_2 \ldots c_m, \ldots, c_k$ – turimas derinys. Leksikografine tvarka pats didžiausias derinys: $n - k + 1, \ldots, n - 1, n$. Vadinasi, $c_m \leq n - k + m$, $m = \overline{1,k}$. Patį mažiausią leksikografiškai didesnį derinį apskaičiuosime taip:

1) masyve $c[1..k]$ rasime pirmą iš dešinės elementą c_m $1 \leq m \leq k$, tenkinantį sąlygą $c_m < n - k + 1$;

2) šį elementą padidinsime vienetu t. y. $c_m = c_m + 1$;

3) likusius elementus užpildysime iš eilės einančiais natūraliaisiais skaičiais t. y.

for $i = m + 1$ to k do $c_i = c_{i-1} + 1$;

Generavimo pabaigos sąlyga. Jei pirmasis iš dešinės elementas, tenkinantis sąlygą $c_m < n - k + m$, yra c_0, tai reiškia, kad nagrinėjame derinį: 0, $n - k + 1$, ..., n. Vadinasi jau visi deriniai sugeneruoti [1].

Detali blokinė algoritmo schema pateikta 5 priede.

1.2.2 Derinių generavimas minimalaus pokyčio tvarka

Minimalaus pokyčio tvarka – tai tokia derinių generavimo seka, kai bet koks šios sekos derinys yra gaunamas iš prieš jį esančio derinio, pakeitus jame vieną elementą kitu.

Pavyzdžiui, C_5^3 deriniai surašyti minimalaus pokyčio tvarka bus tokie:

123, 134, 234, 124, 145, 245, 345, 135, 235, 125.

Kiekvienas šios sekos derinys, išskyrus pirmąjį, yra gaunamas iš prieš jį stovinčio derinio, pakeitus vieną elementą kitu.

Ši derinių seka yra užciklinta. Jei paskutiniame derinyje elementą 5 pakeisime elementu 3, tai gausime pirmąjį šio sekos derinį.

Kiekvieną derinį galima užkoduoti n-skilčiu dvejetainiu skaičiumi $g_1, g_2 \ldots g_i \ldots g_n$: jei $g_i = 1$, tai elementas i priklauso deriniui; jei $g_i = 0$, tai elementas i nepriklauso deriniui. Tuo būdu deriniai C_n^k bus užkoduoti n-skilčiais dvejetainiais skaičiais, kiekvienas iš kurių turi k vienetukų ir $(n - k)$ nulių.

Minimalaus pokyčių derinių sekos kodai bus dvejetainiai skaičiai, turintys k vienetukų, ir bet kokie du gretimi skaičiai skirsis dviem skiltimis: vienoje nulis keičiamas vienetu, o kitoje vienetas keičiamas nuliu.

Derinių C_5^3 sekos kodai bus:

5 lentelė. C_5^3 sekos dvejetainiai kodai

5	4	3	2	1		5	4	3	2	1	
0	0	1	1	1	{1,2,3},	1	1	0	1	0	{2,4,5},
0	1	1	0	1	{1,3,4},	1	1	1	0	0	{3,4,5},

0	1	1	1	0	{2,3,4},	1	0	1	0	1	{1,3,5},
0	1	0	1	1	{1,2,4},	1	0	1	1	0	{2,3,5},
1	1	0	0	1	{1,4,5},	1	0	0	1	1	{1,2,5}.

Tokių kodų generavimas yra glaudžiai susijęs su 1.1.3 skyrelyje išnagrinėtu Grėjaus kodų generavimu.

Tarkime $G(n)$ – Grėjaus kodai, o $G(n,k)$, $0 \le k \le n$ – seka Grėjaus kodų, turinčių lygiai k vienetukų. Tada $G(n,k)$ seka sutampa su minimalaus pokyčio derinių seka taip, kaip parodyta literatūroje [3].

Žemiau pateiktoje lentelėje išdėstyti Grėjaus kodai, kai $n = 5$.

6 lentelė. Grėjaus kodai, kai $n = 5$.

Nr.	5	4	3	2	1	Nr.	5	4	3	2	1	Nr.	5	4	3	2	1
0	0	0	0	0	0	11	0	1	1	1	0	22	1	1	1	0	1
1	0	0	0	0	1	12	0	1	0	1	0	23	1	1	1	0	0
2	0	0	0	1	1	13	0	1	0	1	1	24	1	0	1	0	0
3	0	0	0	1	0	14	0	1	0	0	1	25	1	0	1	0	1
4	0	0	1	1	0	15	0	1	0	0	0	26	1	0	1	1	1
5	0	0	1	1	1	16	1	1	0	0	0	27	1	0	1	1	0
6	0	0	1	0	1	17	1	1	0	0	1	28	1	0	0	1	0
7	0	0	1	0	0	18	1	1	0	1	1	29	1	0	0	1	1
8	0	1	1	0	0	19	1	1	0	1	0	30	1	0	0	0	1
9	0	1	1	0	1	20	1	1	1	1	0	31	1	0	0	0	0
10	0	1	1	1	1	21	1	1	1	1	1						

Kodai turintys tris vienetukus, – pabraukti. Išrašę šiuos kodus, gausime minimalaus pokyčio derinio C_5^3 sekos kodus.

Norėdami efektyviai generuoti $G(n,k)$, $0 \le k \le n$, kodus, pasinaudokime $G(n,k)$ rekursyviniu apibrėžimu:

$$G(n,k) = \begin{pmatrix} 0, G(n-1,k) \\ 1, G(n-1,k-1)^R \end{pmatrix},$$

ir

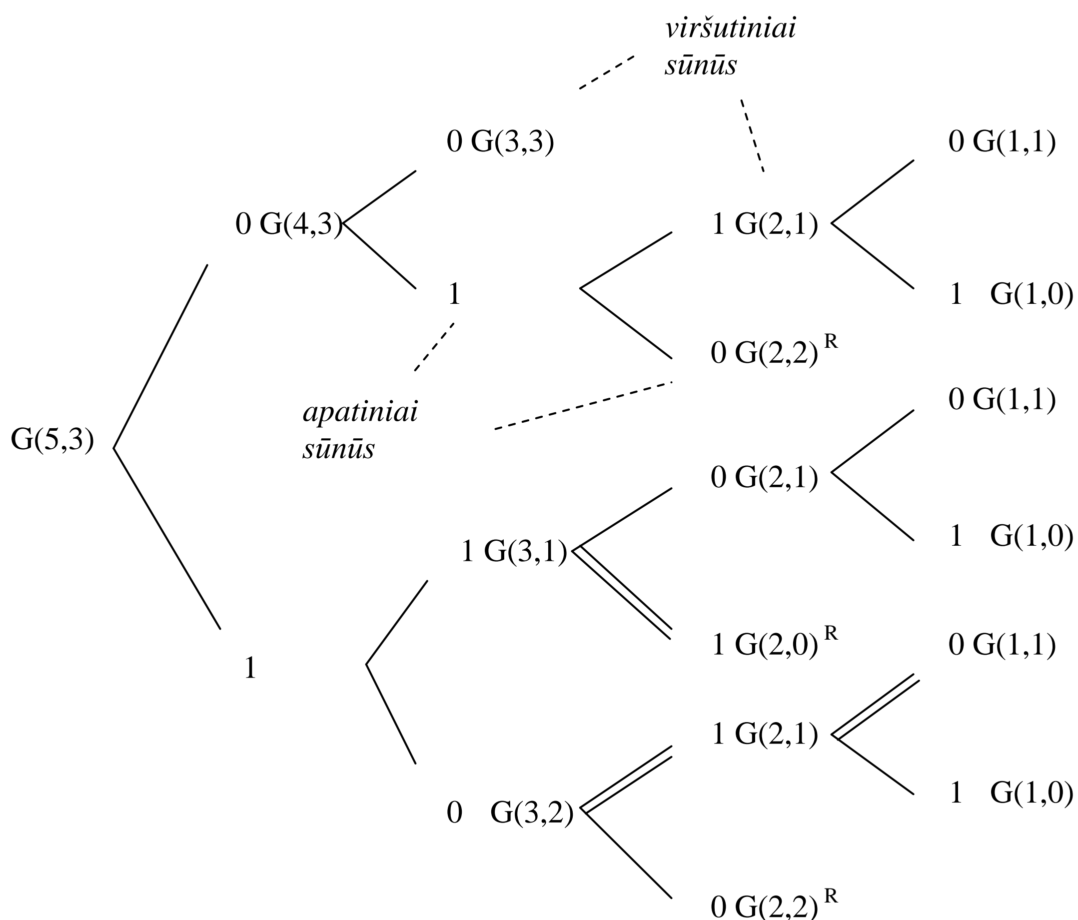
$G(n,0) = 0^n$,

$G(n,n) = 1^n$,

čia $G(n-1,k-1)^R$ žymi kodus, surašytus atvirkščia tvarka, o 0^n reiškia, kad n skilčių užpildome nuliais, o 1^n – kad n skilčių užpildome vienetais.

Pastaroji rekurentinė formulė generuoja C_n^k pradedant (1, 2, ..., k) ir baigiant (1, 2, ..., k –1, n).

3 paveikslas. $C_5^3 = G(5,3)$ generavimo medis



Remiantis šiuo medžiu gausime lentelėje nr. 4 surašytą $G(5,3)$. 3 pav. pavaizduoto binarinio medžio viršūnės yra arba 1 $G(m,0)^R$ arba 0 $G(m,m)$ (žr. [3]).

Kodų $G(n,k)$ generavimo uždavinys transformuojasi į binarinio medžio kabančių viršūnių peržiūrą: pradedant viršutine ir baigiant apatine.

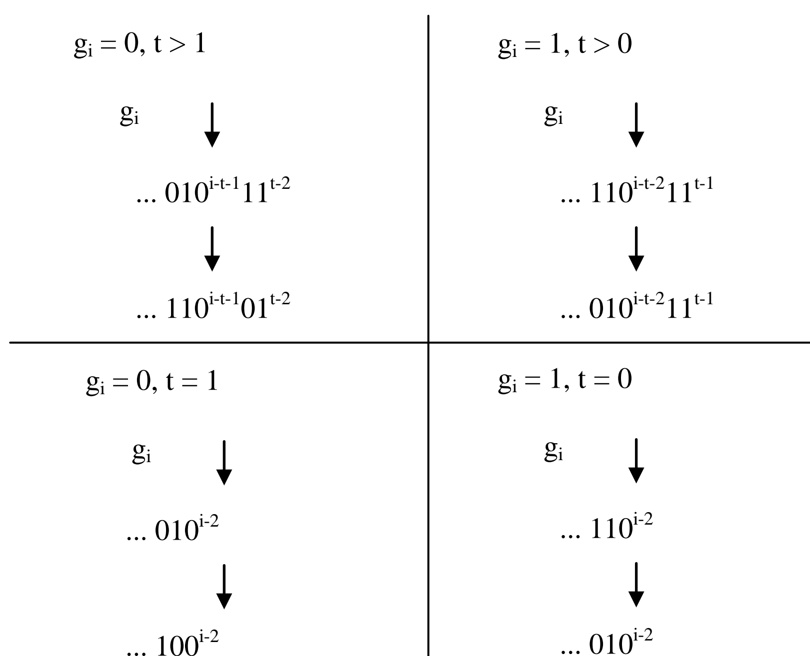
Norint pereiti nuo vienos kabančios viršūnės į gretimą kabančią viršūnę, nagrinėjame kelią iš pirmos kabančios viršūnės medžio šaknies link. Tuo keliu link šaknies keliaujame viršūnėmis, kurios yra apatiniai sūnūs (žr. 3 pav.). Pasiekę pirmąją viršūnę, kuri yra viršutinis sūnus, pereiname pas jo brolį (apatinį sūnų) ir iš jo, eidami viršutiniais sūnumis, einame į kabančią viršūnę.

Pavyzdžiui (žr. 3 pav.), išnagrinėję kabančią viršūnę 1 $G(2,0)^{R}$, pirmiausia pasieksime viršutinį sūnų 1 $G(3,1)$ ir pereisime pas jo brolį (apatinį sūnų) 0 $G(3,2)^{R}$. Ir iš jo viršutiniais sūnumis 1 $G(2,1)$, 0 $G(1,1)$ pasieksime viršūnei 1 $G(2,0)^{R}$ gretimą kabančią viršūnę 0 $G(1,1)$. (3 pav. šį kelią vaizduoja dviguba linija).

Nagrinėdami kabančią binarinio medžio viršūnę (g_n, g_{n-1}, ..., g_1), mes tuo pačiu žinome kelią iš šaknies į kabančią viršūnę. Norėdami pereiti prie gretimos kabančios viršūnės, turime saugoti informaciją apie viršūnę g_i $G(i-1, t)$, į kurią turime sugrįžti (viršūnės g_i $G(i-1, t)^{R}$ visada yra apatiniai sūnus). Šios viršūnės bus saugomos steke. Toliau pateiktame C_n^k generavimo algoritme stekas bus organizuojamas taip pat, kaip generuojant Grėjaus kodus (žr. trečiąjį Grėjaus kodų algoritmą). Parametras t rodo skaičių vienetukų tarp skilčių (g_{i-1}, g_{i-2}, ..., g_1) ir gali būti nesunkiai apskaičiuojamas. Todėl steke saugoma tik i reikšmė.

Atidžiai išnagrinėjus binarinį medį, galima pastebėti, kad nagrinėjamo kodo, kai iš jo grįžtama į viršūnę g_i $G(i-1, t)$, perskaičiavimas į gretimą kodą reikalauja nagrinėti žemiau pateiktus keturis atvejus, priklausomai nuo g_i ir t reikšmės:

4 paveikslas. Binarinio medžio nagrinėjimo atvejai.



Šie pertvarkymai turi įtakos vienetukų skaičiui tarp skilčių (g_{i-1}, g_{i-2}, ..., g_1): t reikšmę turime sumažinti 1, jei $g_i = 0$, ir padidinti 1, jei $g_i = 1$.

Pereinant prie gretimo kodo yra invertuojamos dvi skiltys:

1) g_i ,

ir

2) g_{t-1}, jei $g_i = 0$ ir $t > 1$,

g_{i-1}, jei $g_i = 0$ ir $t = 1$,

g_t, jei $g_i = 1$ ir $t > 0$,

g_{i-1}, jei $g_i = 1$ ir $t = 0$.

Atlikus šiuos veiksmus, į steką reikia įrašyti papildomą informaciją: tarpines viršūnes, esančias tarp nagrinėjamos viršūnės ir nagrinėjamos kabančios viršūnės. Kadangi tas priklauso nuo viršūnės steko viršuje, tai t reikšmė gali vėl pasikeisti.

Jei $t = 0$ arba $t = i - 1$, tai reiškia, kad viršūnė, kurioje mes esame, yra kabanti viršūnė: g_i $G(i - 1, i - 1)$ arba g_i $G(\mathrm{i} - 1, 0)^R$ ir jokių tarpinių viršūnių saugoti nereikia. Šiuo atveju t visada lygus $t + 1$.

Kita vertus, jei $t \neq 0$ ir $t \neq i - 1$, tai į steką turime įtraukti viršūnes $i - 1$, $i - 2$, ..., $t + 1$, išskyrus tą atvejį, kai $g_{i-2} = ... = g_1 = 0$. Šiuo atveju į steką įtraukiame tik $i - 1$.

Parametro t reikšmė turi būti perskaičiuojama. Kadangi tarp (g_{i-1}, g_{i-2}, ..., g_{t+1}) vienetukų gali būti tik $g_{\mathrm{i}-1}$, tai $t = t - g_{\mathrm{i}-1}$. Be to, jei šiuo atveju t tampa lygus 0, tai $g_{i-2} = ... = g_1 = 0$.

Priede 6 pateiktoje C_n^k generavimo minimalaus pokyčio tvarka algoritmo blokinėje schemoje stekas organizuojamas masyvu τ, o jo viršutinis elementas $\tau[1]$ (žr. trečiąjį Grėjaus kodų algoritmą). Kadangi elementas $\tau[i\text{-}1] = \tau[1]$, tai $\tau[1] = t + 1$ekvivalentu, kad į steką įtraukiame $i - 1$, i – 2, ..., $i + 1$ [1].

1.3 Kėlinių generavimo algoritmai

1.3.1 Kėlinių generavimas leksikografine tvarka

Algoritmą sudarysime laikydamiesi kombinatorinių objektų generavimo struktūros (1.1 skyrelis). Panagrinėkime pavyzdį, kai $n = 8$.

Pradinio kodo generavimas. Kėlinį talpinsime masyve $p[0...n]$, kurio elementai p_1, $p_2, ..., p_n$ ir nusako kėlinį. Pradinis kėlinys yra 1, 2, 3, 4, …, n. Jei $n = 8$, tai kėlinį sudarys elementai 1, 2, 3, 4, 5, 6, 7, 8.

Paties mažiausio, leksikografiškai didesnio kėlinio nei nagrinėjamas, apskaičiavimas. Tarkime, p_1, $p_2, ..., p_n$ – nagrinėjamas kėlinys. Pavyzdžiui, 2, 1, 4, 8, 7, 6, 5, 3. Jei į kėlinį žiūrėsime kaip į skaičių, tai reikia rasti patį mažiausią skaičių, didesnį nei nagrinėjamas. Tam tikslui reikia:

1) rasti pirmą iš dešinės porą (p_i, p_{i+1}) tokią, kad $p_i < p_{i+1}$,

2) rasti $p_j = \min\limits_{i+1 \le z \le n} (p_z \mid p_z > p_i)$, t.y. tarp elementų $p_{i+1}, p_{i+2}, ..., p_n$ rasti patį mažiausią elementą, didesnį už p_i,

3) elementus p_i ir p_j sukeisti vietomis,

4) elementus $p_{i+1}, p_{i+2}, ..., p_n$ („uodegą“) surašyti atvirkščia tvarka („apversti“).

Pavyzdžiui, pirmoji iš dešinės pora, tenkinanti 1) punkto reikalavimą yra (4,8), o elementas $p_j = 5$. Tada, sukeitę 4 su 5, gausime 2, 1, 5, 8, 7, 6, 4, 3. „Apvertę“ uodegą gausime patį mažiausią leksikografiškai didesnį kėlinį nei duotas: 2, 1, 4, 8, 7, 6, 5, 3.

Generavimo pabaigos sąlyga. Pats didžiausias kėlinys yra 0, n, n – 1, …, 3, 2, 1 (masyvo $p[0…n]$ elementas $p[0]$ yra pagalbinis, ir jo reikšmė lygi nuliui). Tada pirma iš dešinės pora (p_i, p_{i+1}), tenkinanti sąlygą $p_i < p_{i+1}$ yra (0, n). Kitaip tariant, jei $i = 0$, tai generavimo pabaiga. Kėlinių generavimo algoritmas pateiktas 7 priede [1].

1.3.2 Kėlinių generavimas antileksikografine tvarka

Apibrėžimas. Vektorius $x = (x_1, x_2, ..., x_n)$ yra antileksikografiškai mažesnis už vektorių $y = (y_1, y_2, ..., y_n)$ (žymime $x < y$), jei egzistuoja toks $k \le n$, kad $x_k > y_k$, o $x_i = y_i$, $i = \overline{k+1, n}$. Pastebėsime, jei vietoje skaičių 1, 2, …, n paimtume raides $a, b, c, ..., z$, išdėstytas natūralia, pavyzdžiui, abėcėlės tvarka: $a < b < c < ... < z$, tai leksikografinė tvarka apibrėš n ilgio žodžių išsidėstymą žodyne įprasta tvarka. Tuo tarpu antileksikografinė tvarka apibrėžia žodžių tvarką, kai tiek pozicijų eiliškumas sekoje, tiek ir aibės elementų išsidėstymas žodyne yra atvirkščias .

Pavyzdžiui, užrašykime aibės $X = \{1, 2, 3\}$ kėlinius leksikografine ir antileksikografine tvarka.

Leksikografine tvarka:

1, 2, 3; 1, 3, 2; 2, 1, 3; 2, 3, 1; 3, 1, 2; 3, 2, 1.

Antileksikografine tvarka:

1, 2, 3; 2, 1, 3; 1, 3, 2; 3, 1, 2; 2, 3, 1; 3, 2, 1.

Iš leksikografinės tvarkos kėlinio 1, 3, 2 gausime antileksikografinės tvarkos kėlinį, jei 1-tą keisime 3-tu, 3-tą keisime 1-tu (imame aibės X elementus atvirkščia tvarka) ir kėlinio pozicijas surašome atvirkščia tvarka. Gausime 3, 1, 2, o po to 2, 1, 3.

Kėlinių generavimas antileksikografine didėjimo tvarka yra labai panašus į kėlinių generavimą leksikografine didėjimo tvarka algoritmą. Kėlinių P_n generavimui įveskime masyvą $p[1..n + 1]$, kurio elementai $p_1, p_2, ..., p_n$ apibrėžia nagrinėjamąjį kėlinį.

Pradinio kėlinio generavimas. for $i = 1$ to $n + 1$ do $p[i] = i$.

Paties mažiausio antileksikografiškai didesnio kėlinio apskaičiavimas. Tam tikslui reikia:

1) rasti pirmą iš kairės porą (p_{i-1}, p_i), tokią, kad $p_{i-1} < p_i$,

2) rasti $p_j = \max\limits_{i \le z \le i-1} (p_z \mid p_z < p_i)$, t.y. tarp elementų $p_1, p_2, ..., p_{i-1}$ rasti patį didžiausią elementą, mažesnį nei p_i ,

3) elementus p_i ir p_j sukeisti vietomis,

4) elementus $p_{1,} p_{2,...,} p_{i-1}$ surašyti atvirkščia tvarka.

Generavimo pabaigos sąlyga. Paskutinysis antileksikografinės tvarkos kėlinys yra n, n – 1, ..., 3, 2, 1, o p_{n+1}, t.y. masyvo p[1..n + 1] elementai yra n, n – 1, ..., 3, 2, 1, p_{n+1} . Aišku, kad šiuo atveju pirmoji iš dešinės pora (p_{i-1}, p_i), tenkinanti 1)-ojo punkto reikalavimą: $p_{i-1} < p_i$, yra (1, n + 1). Vadinasi, jei $i = n + 1$, tai generavimo pabaigos sąlyga [1]. Algoritmo blokinė schema pateikta 8 priede.

1.3.3 Kėlinių generavimas minimalaus pokyčio tvarka.

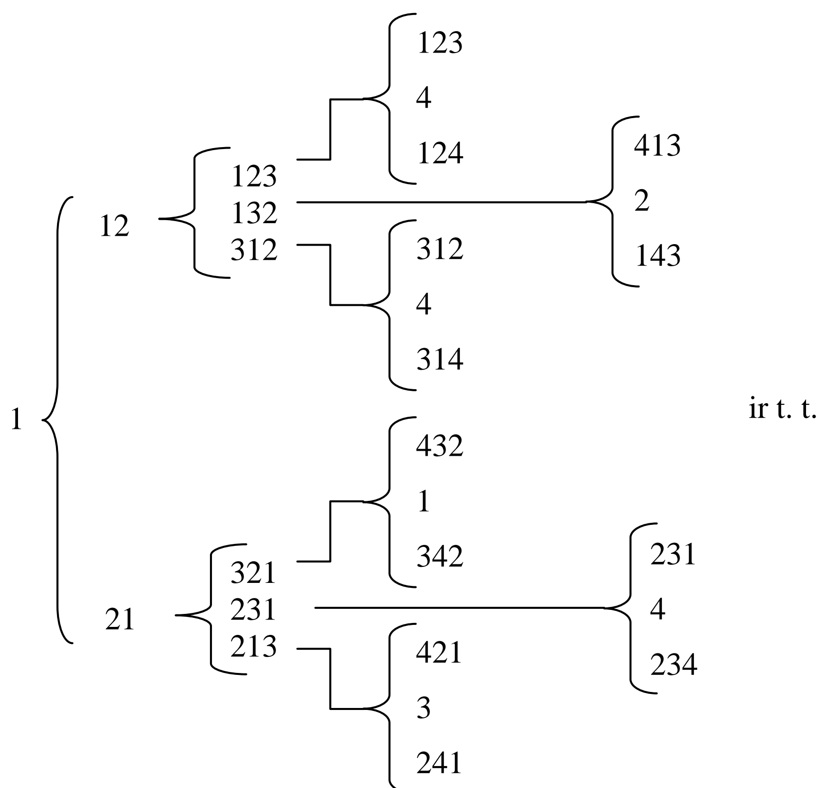
Kėlinių minimalaus pokyčio tvarka – tai tokia kėlinių generavimo seka, kai bet kokie du gretimi kėliniai skiriasi tik dviejų elementų padėtimi. Pavyzdžiui, kai n = 3, gautume seką: 1, 2, 3; 1, 3, 3; 3, 1, 2; 3, 2, 1; 2, 3, 1; 2, 1, 3.

Kai n = 1, tai egzistuoja vienintelis kėlinys: 1. Tarkime, kad turime P_{n-1} kėlinius, tenkinančius minimalaus pokyčio tvarką. Tada kėlinius P_n generuosime taip.

Sunumeruokime visus P_{n-1} kėlinius numeriais nuo 1 iki (n – 1)!. Tada, jei kėlinio iš n – 1 elemento numeris yra nelyginis skaičius, tai prie šio kėlinio prirašykime elementą n iš dešinės ir nuosekliai atlikime visus galimus to elemento postūmius į kairę. Jei kėlinio iš n – 1 elemento nuris yra lyginis, tai prie jo prirašykime elementą n iš kairės ir nuosekliai atlikime visus galimus to elemento postūmius į dešinę [1].

Atlikę šiuos veiksmus, gausime kėlinių P_n minimalaus pokyčio seką (žr. 5 pav.).

5 paveikslas. Kėlinių generavimas minimalaus pokyčio tvarka



Aptartas kėlinių minimalaus pokyčio sekos generavimo metodas neatitinka aukščiau pateiktos kombinatorinių objektų generavimo algoritmo schemos: skaičiavimo eigoje reikia saugoti visus P_{n-1} kėlinius. Literatūroje [2] pateikta šio metodo realizacija, atitinkanti kombinatorinių objektų generavimo algoritmo schemą. Aptarkime šią realizaciją.

Įveskime tris masyvus: $p[0..n + 1]$, $a[1..n]$ ir $d[1..n]$.

Masyvo p elementai p_1, $p_2, \ldots, p_n$ apibrėš nagrinėjamą kėlinį. Elementai p_0 ir p_{n+1} yra pagalbiniai elementai.

Masyvo a elementai nusako atvirkštinį kėlinį: a_k nurodo vietą (adresą), kurioje masyve p yra elementas k. Kitaip tariant, $p_{a_k} = k$. Masyvo d elementai apibrėžiami taip:

$d_i = $ - 1, jei elementas i kėlinyje yra „stumiamas“ į kairę (keičiamas su kaimynu iš kairės),

+ 1, jei elementas i kėlinyje yra

Kėlinyje elementas *i* „stumiamas“ iki jis pasiekia elementą, didesnį už jį patį. Tada yra keičiama šio elemento judėjimo kryptis ir bandoma „stumti“ (jei galima) elementą $i - 1$. Kadangi turime masyvą *a*, tai šis elementas lengvai randamas masyve *p*.

Tam, kad nutrauktume elemento *n* judėjimą, į masyvą *p* įvesti du papildomi elementai p_0 ir p_{n+1} , kurių reikšmės yra lygios $n + 1$.

Elemento d_1 reikšmė yra lygi 0, nes *p* elemento, lygaus vienetui, judinti nereikia. Tai sekos generavimo pabaigos sąlyga [1]. Algoritmo blokinė schema pateikta 9 priede.

1.4 Algoritmų sudėtingumas

Algoritmas – tai tikslus veiksmų sąrašas, nurodantis, kokius veiksmus ir kokia tvarka reikia atlikti norint gauti reikiamą rezultatą.

Algoritmams būdingos tokios bendrosios savybės:

Diskretumas. Algoritmas skaidomas į tiksliai aprašytus vykdymo žingsnius (etapus).

Baigtumas. Algoritmas turi pabaigą.

Universalumas. Algoritmas turi tikti bet kokiam numatyto tipo pradinių duomenų rinkiniui.

Rezultatyvumas. Algoritmas visada turi pateikti konkretų rezultatą (jei jis egzistuoja) arba paaiškinimą, kodėl jis negautas.

Aiškumas. Algoritmas turi būti pateikiamas taip, kad jį visi vienareikšmiškai suprastų.

Plačiausiai paplitę du grafiniai algoritmų vaizdavimo būdai:

1. Algoritmų schemos.

2. Struktūrogramos [4].

Algoritmai naudojami įvairių problemų sprendimui:

- Žmogaus genomo projekto tikslas nustatyti 100 000 genų esančių DNR, kurią sudaro 3 bilijonai cheminių jungčių, išsaugant visą informaciją duomenų bazėje ir sukuriant įrankius duomenų analizei. Šiai užduočiai realizuoti reikalingi efektyvūs algoritmai, kurių dėka būtų sutaupomas tiek kompiuterių tiek žmonių laikas ir pinigai.

- Interneto dėka žmonės visame pasaulyje paprastai ir greitai keičiasi informacija. Greitai keistis informacija leidžia efektyvus trumpiausio kelio paieškos algoritmas.
- Paplitus elektroninei komercijai svarbu užtikrinti duomenų saugumą. Viešojo rakto kriptografija, elektroniniai parašai paremti efektyviais algoritmai ir skaičių teorija.
- Verslui yra svarbu tinkamai paskirstyti resursus. Pvz., svarbu parinkti efektyvų maršrutą prekių pristatymui, organizuoti darbuotojų darbą, interneto ryšio tiekėjui svarbu tinkamai išdėstyti tinklo mazgus ir t.t.[5]

Informatikoje nagrinėjami du klasikiniai sudėtingumo matai: **laikas**, kuriuo nusakomas algoritmo elementarių etapų (t. y. žingsnių) skaičius ir **atmintis**, kuriuo nusakoma algoritmui reikalinga atmintis. Tie abu matai yra pradinių duomenų asimptotinės funkcijos. Paaiškinsime tai tiesinio laiko pavyzdžiu:

Algoritmo sudėtingumas yra tiesinis laiko atžvilgiu, jeigu bet kuriems ilgio n pradiniams duomenims x elementariųjų skaičiavimo etapų skaičius mažesnis už n, kai n – pakankamai didelis[1]. [6]

Algoritmų sudėtingumo asimptotinis žymėjimas.

Kai domimės vykdymo laiko funkcijos augimo pobūdžiu (tiesinis, kvadratinis ir pan.), tai sakome, kad tyrinėjame asimptotinį algoritmų efektyvumą. Mus domina, kaip algoritmų vykdymo laikas didėja, lyginant su uždavinio dydžiu, riboje, kai uždavinio dydis neribotai auga (artėja į begalybę).

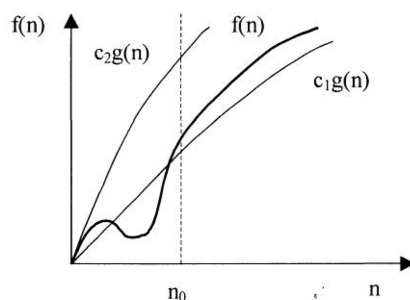
Θ žymėjimas

$\Theta(g(n)) = \{f(n)$: egzistuoja konstantos c_1, c_2 ir n_0, kad $0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall n \ge n_0\}$

Funkcija $f(n)$ priklauso $\Theta(g(n))$, jei egzistuoja teigiamos konstantos c_1 ir c_2, kad ji gali būti įterpta tarp $c_1 g(n)$ ir $c_2 g(n)$ pakankamai dideliems n. Θ sudėtingumo funkcija pavaizduota paveiksle 6.

[1] Tai tik pavyzdys, o ne apibrėžimas

6 paveikslas. Grafinis Θ žymėjimo pavyzdys



Pvz., galime sakyti, kad rūšiavimo įterpimu vykdymo laiko funkcija blogiausiuoju atveju $T(n) = \Theta(n^2)$.

Sakome, kad g(n) yra asimptotiškai griežta funkcijos f(n) riba.

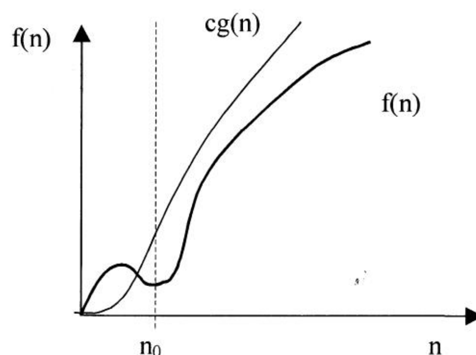
***O* žymėjimas**

Θ asimptotiškai apriboja funkciją tarp viršutinės ir apatinės ribos. Kai turime tik viršutinę asimptotinę ribą, naudojame *O* žymėjimą.

$O(g(n)) = \{f(n)$: egzistuoja konstantos c ir n_0, kad $0 \leq f(n) \leq cg(n)\ \forall n \geq n_0\}$

O sudėtingumo funkcija pavaizduota paveiksle 7.

7 paveikslas. Grafinis *O* žymėjimo pavyzdys



Iš f(n) = Θ (g(n)) išplaukia f(n) = *O*(g(n)). Θ – priklausomybė yra stipresnė, negu *O* – priklausomybė.

Pvz., $an^2 + bn = \Theta(n^2); an^2 + bn = O(n^2)$,

Tačiau $an + bn \neq \Theta(n^2), an + bn = O(n^2)$.

Literatūroje kartais pasitaiko, kad O – žymėjimu žymimos asimptotiškai griežtos ribos.

O sudėtingumas nurodo, kiek daugiausia laiko gali užimti algoritmo vykdymas (blogiausias vykdymo atvejis) [5]. Žemiau lentelėje pateikti dažnai pasitaikantys žymėjimai.

7 lentelė. Dažnai pasitaikantys O sudėtingumo žymėjimai. [8],[9]

Žymėjimas	**Sudėtingumas**	**Klasė**	**Pavyzdžiai**
O(1)	konstantinis	Polinominė (P)	Nustatyti ar skaičius yra lyginis ar ne lyginis
$O(\log n)$	logaritminis		Elemento radimas surūšiuotame masyve dvejetainės paieškos būdu
$O([\log n]^c)$	polilogaritminis		
$O(n)$	tiesinis		Dviejų n-ilgio skaičių sudėtis
$O(n \cdot \log n)$	supratiesinis		Heapsort rūšiavimo algoritmas
$O(n^2)$	kvadratinis		Įterpimo, išrinkimo algoritmai
$O(n^c)$	polinominis, kartais geometrinis		Paieška kd-tree
$O(c^n)$	eksponentinis	Eksponentinė (NP)	Keliaujančio pirklio uždavinio sprendimas naudojant dinaminį programavimą
$O(n!)$	faktorialinis		Keliaujančio pirklio uždavinio sprendimas naudojant brutalią paiešką.
$O(n^n)$			

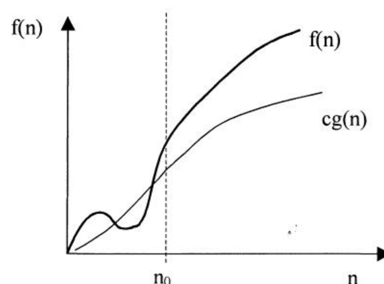
Ω žymėjimas

Kaip O žymėjimas nurodo viršutinę funkcijos asimptotinę ribą, taip Ω žymi apatinę asimptotinę ribą.

$\Omega(g(n)) = \{f(n)$: egzistuoja konstantos c ir n_0, kad $0 \leq cg(n) \leq f(n)\ \forall n \geq n_0\}$.

Ω sudėtingumo funkcija pavaizduota paveiksle 8. [5]

8 paveikslas. Grafinis Ω žymėjimo pavyzdys



Algoritmų sudėtingumo klasės:

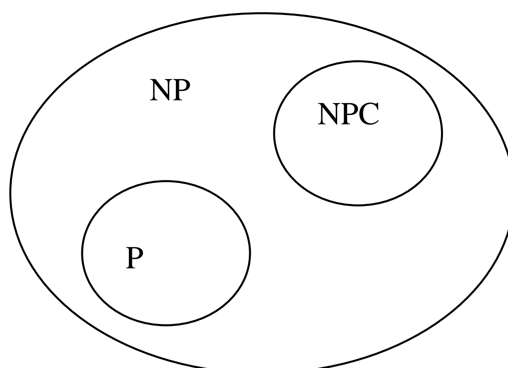
- **P**. Uždaviniai gali būti išspręsti per polinominį laiką („P“ reiškia polinominis)
- **NP**. Santrumpa reiškia „nedeterminuotas polinominis laikas“. Uždavinys priklauso NP klasei, jei per priimtiną laiką (polinominį) galima patikrinti, ar sprendinys yra teisingas.[7]

„NP-pilnoji“ (NP-complete) uždavinių klasė. Pagrindinė priežastimi kodėl informatikos teoretikai tiki, kad $P \neq NP$, tai NP-pilnosios uždavinių klasės egzistavimas. Klasė pasižymi tuo, kad iki šiol nerastas polinominio laiko algoritmas nei vienam jos uždaviniui. Tačiau įrodyta, kad jei bent vienas uždavinys iš NP pasirodytu išsprendžiamas polinominiu laiku, tada kiekvienas iš tos klasės turėtų polinominį sprendinį.

Nežiūrint daug įdėto darbo, nė vienas iš NP-complete klasės uždavinių iki šiol nėra polinominiu laiku išspręstas. Kaip tik iš to ir kyla informatikos teoretikų įsitikinimas, kad polinominiai algoritmai šiems uždaviniams neegzistuoja.

„NP-complete“ (NPC) kalbos yra sunkiausios iš visų NP kalbų. Informatikos teoretikai mano, kad $P \cup NPC = \emptyset$. Bendras, P, NP, NPC klasių išsidėstymas tada atrodytų taip, kaip parodyta toliau sekančiame paveikslėlyje.

9 paveikslas. P, NP, NPC klasių išsidėstymas



NP-pilnosios klasės uždavinių pavyzdžiai:

1. Hamiltono ciklo (kelio) radimas grafe.
2. Grafo dengimo uždavinys. Bekrypčiame grafe reikia rasti minimalų skaičių viršūnių, tokių, kad visos grafo briaunos būtų tomis išrinktosioms viršūnėms gretimos.
3. Poaibio sumos uždavinys. Duota natūrinių skaičių aibė ir skaičius $1 \in N$ (natūrinių skaičių aibės). Reikia patikrinti, ar galima surasti tokį aibės poaibį, kurio suma būtų 1.
4. Keliaujančio prekeivio uždavinys. Grafe su svoriais reikia rasti trumpiausią ciklą, praeinantį per visas viršūnes po vieną kartą.

2 PROJEKTINĖ DALIS

2.1 Projekto vykdymo planas

Darbo vykdymo laikotarpis nuo 2010 m. sausio 1 d. iki 2010 m. gegužės 1 d. Numatoma darbo apimtis 320 valandų (8 kreditai).

Darbai	Trukmė	Laikotarpis
Konsultacija su vadovu	8 h	2010 01 01 – 2010 05 01
Teorinės medžiagos analizė	120 h	2010 01 01 – 2010 03 15
Įrankių ir priemonių analizė	10 h	2010 01 01 – 2010 04 15
Programavimas	162 h	2010 02 01 – 2010 04 15
Algoritmų sudėtingumo tyrimas	10 h	2010 04 15 – 2010 04 20
Aprašo rašymas	10 h	2010 01 15 – 2010 05 01

2.2 Įrankių ir priemonių analizė

Tyrimo tikslams sukurta programa realizuojanti teorinėje dalyje aprašytus algoritmus. Gauti rezultatai apdoroti ir algoritmų sudėtingumo analizė atlikta MS Excel programa.

Programa parašyta „Java“ kalba „Neatbeans“ programavimo aplinkoje. Pasirinkimą sąlygojo šios priežastys:

1. Daug išsamios nemokamai platinamos medžiagos, kaip programuoti „Java“ kalba. Pvz., programavimo kalbos kūrėjų svetainėje www.java.com pateikiama išsami mokymosi medžiaga. Taip pat gerai dokumentuota „Netbeans“ programavimo aplinka. „Netbeans“ svetainėje pateikiami ne tik aprašymai, bet ir filmuota medžiaga, pavyzdžiai.
2. „Netbeans“ yra atvirojo kodo nemoka programavimo aplinka. Ji lenkia „Eclipse“ programavimo aplinką gausia dokumentacija bei teigiamais vartotojų atsiliepimais.

3. „Netbeans“ pagalba patogu kurti grafinę vartotojo sąsają. Nereikia pačiam aprašyti visų grafinių elementų. Kodas sugeneruojamas automatiškai. Belieka suteikti programos grafiniams elementams funkcionalumą.
4. „Java“ virtuali mašina skirta „Windows“, „Linux“, „Solaris“, „OS X“ operacinėms sistemoms. Kartą sukompiliuota programa gali būti naudojama skirtingose operacinėse sistemose.
5. „Java“ virtuali mašina automatiškai sunaikina nebereikalingus objektus (nebenaudojamus programos) taip atlaisvindama atmintį. Programuotojui nereikia tuo rūpintis.

Programos pavadinimas „Algoritmai.jar“. Programos schema pateikta 11 priede. Kiekvienas algoritmas realizuojamas kaip atskiras objektas. Kiekvieno algoritmo grafinė sąsaja taip pat realizuojama kaip atskiras objektas. Objektiniu atžvilgiu programą sudaro 21 klasė (objektas). Be pagrindinės programos klasės yra: klasė realizuojanti algoritmų pasirinkimą, vykdymo laiko fiksavimui ir apskaičiavimui bei 9 klasės algoritmų realizavimui, kurios turi po vieną pagalbinę vidinę klasę. Vykdant programą sukuriama po vieną kiekvienos klasės objektą. Realiai objektų sukuriama daugiau nei čia aprašyta, nes „Java“ kalba savaime naudoja daug objektų. Pavyzdžiui, duomenų tipas „String“ realizuojamas kaip objektas.

Programa tikrina ar įvesti duomenys į duomenų laukelius (pvz., aibės elementų skaičius) ir ar jie yra skaičiai, ar sutampa įvestas aibės elementų skaičius su realiai įvestu elementų kiekiu. Taip pat tikrinama, ar galima įrašyti sugeneruotus poaibius į failą. Klaidos atveju programa parodo informacinį pranešimą. Vartotojas pasirenka, kur išsaugoti failą, į kurį bus įrašomi sugeneruoti poaibiai. Programos teisingumas buvo patikrintas lyginant gautus rezultatus su literatūroje [1] pateiktais duomenimis. Programos naudojimo instrukcija pateikta 12 priede.

MS Excel programa pasirinkta gautų duomenų analizei, nes ja galima atlikti regresinę analizę bei nupiešti gautų funkcijų grafikus. Regresinė analizė – yra būdas iš turimų duomenų išvesti funkciją.

2.3 Algoritmų sudėtingumo tyrimas

Tyrimui naudotas nešiojamasis kompiuteris su šiais techniniais parametrais:

1. Procesorius – Intel Core2 T7200 2 GHz;

2. Operatyvioji atmintis (RAM) – 1,5 GB;
3. Kietasis diskas – 120 GB;
4. Operacinė sistema – Windows XP SP3.

Tyrimo metu buvo išjungtos visos nereikalingos programos, tame tarpe ir antivirusinė programa, kompiuteris atjungtas nuo tinklo.

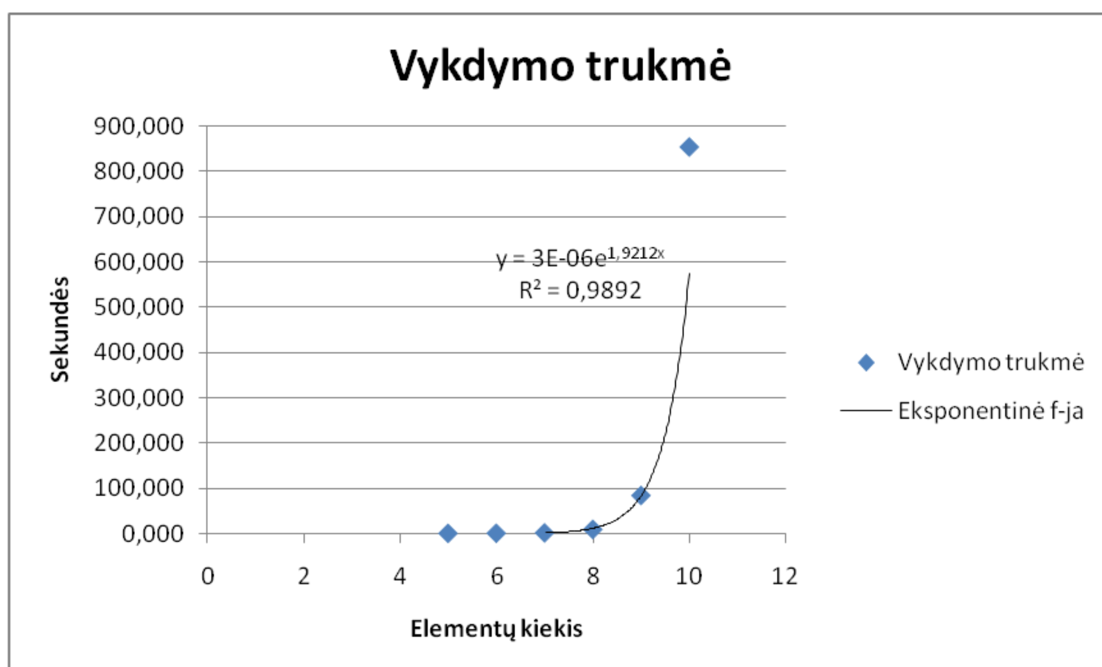
Aibės elementų skaičius tyrime didintas tol, kol algoritmų vykdymas tilpo į priimtiną laiką (10 – 15 min.) arba gautų duomenų pakako išvesti laiko funkcijai. Algoritmai su tuo pačiu pradinių duomenų kiekiu vykdyti po tris kartus. Apskaičiuotas visų trijų bandymų algoritmo vykdymo laiko vidurkis, kuriuo remiantis išvestos funkcijos.

2.3.1 Kėlinių generavimo algoritmų sudėtingumo tyrimas

Kėlinių generavimo leksikografine didėjimo tvarka algoritmo tyrimas

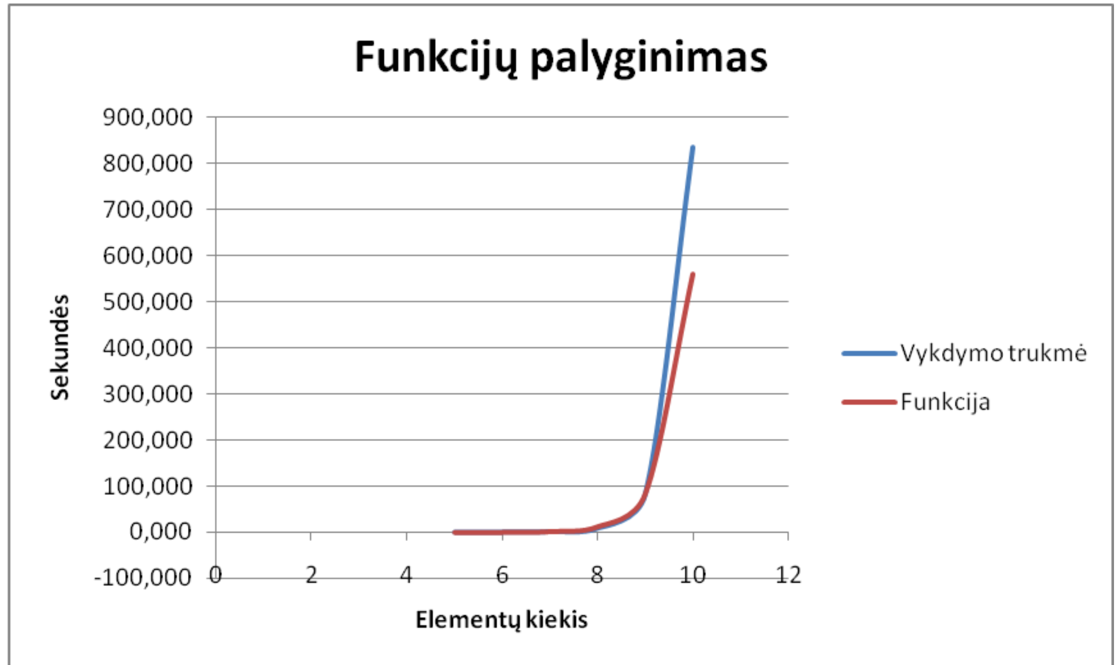
Algoritmas vykdytas su aibe elementų nuo 5 iki 10. Su 10 elementų algoritmas vykdytas 14 min. 23 s. Įdomumo dėlei buvo bandoma paleisti kėlinių generavimą iš 11 elementų. Po 1.5 h laukimo generavimas nutrauktas. Paveiksle nr. 10 atvaizduoti gauti tyrimo duomenys bei išvesta funkcija. Kuo R^2 reikšmė artėja prie 1, tuo tikslesnė gautoji funkcija.

10 paveikslas. Kėlinių generavimo leksikografine didėjimo tvarka grafikas



Gauta funkcija $y = 0{,}000003e^{1{,}9212x}$. Iš funkcijos išraiškos matome, kad algoritmo sudėtingumas yra eksponentinis t.y. $\Omega(c^n)$. Kintamasis x reiškia aibės elementų kiekį. Žemiau pateikiamas realaus algoritmo vykdymo laiko ir apskaičiuotosios funkcijų palyginimas.

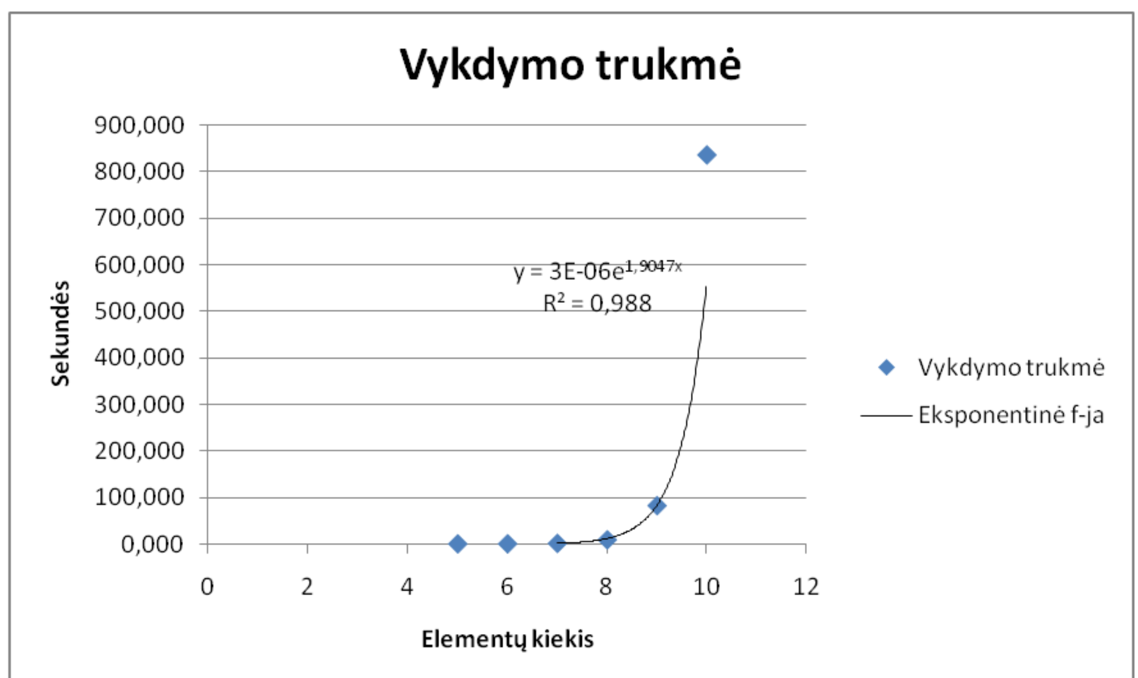
11 paveikslas. Kėlinių generavimo leksikografine didėjimo tvarka funkcijų palyginimas



Kėlinių generavimo antileksikografine didėjimo tvarka algoritmo tyrimas

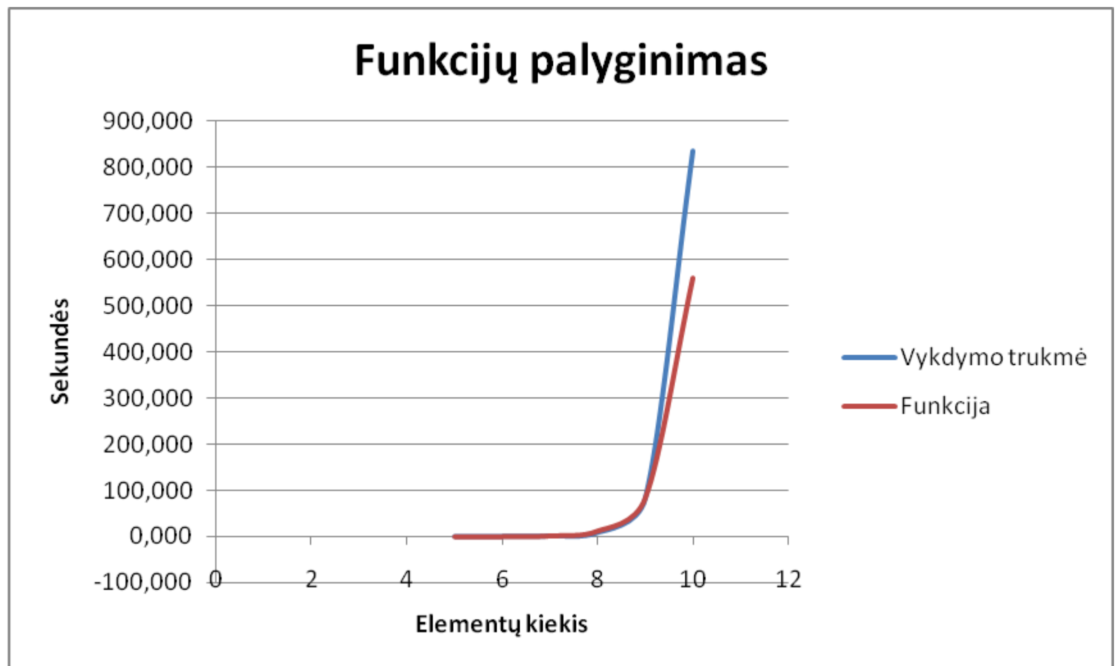
Algoritmas taip pat vykdytas su aibe elementų nuo 5 iki 10. Tyrimo rezultatai labai panašūs į aukščiau aprašytus. Žemiau pateiktas gautų rezultatų grafikas.

12 paveikslas. Kėlinių generavimo antileksikografine didėjimo tvarka grafikas



Gauta funkcija $y = 0{,}000003e^{1{,}9047x}$. Darome išvadą, kad algoritmo sudėtingumas eksponentinis $\Omega(c^n)$. 13 paveiksle pateiktas gautų funkcijų palyginimas.

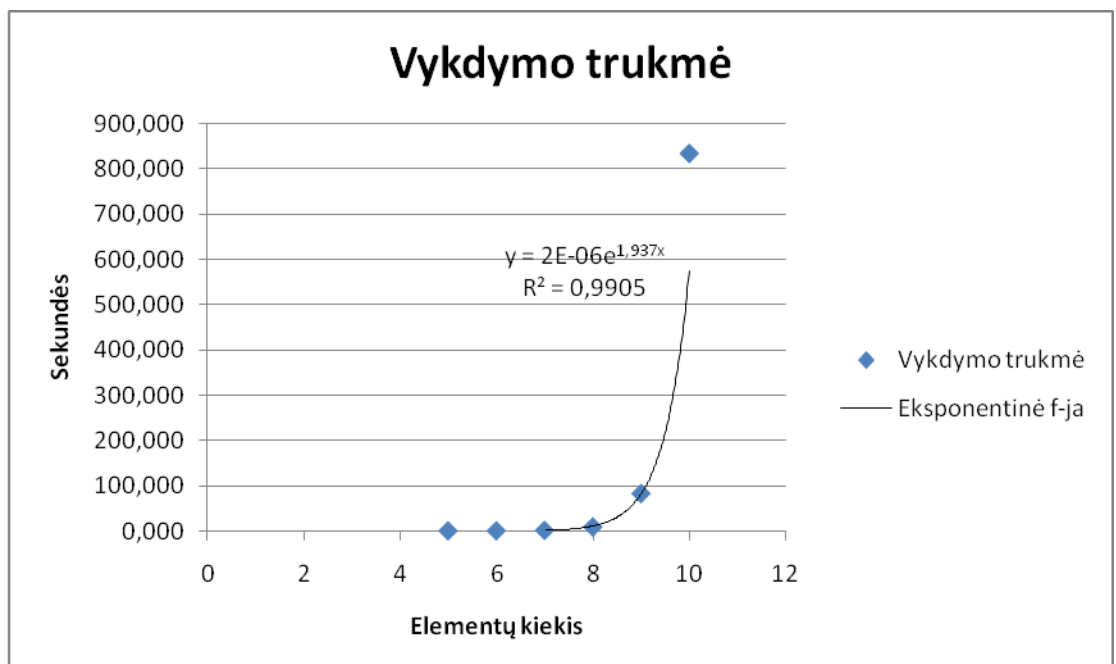
13 paveikslas. Kėlinių generavimo antileksikografine didėjimo tvarka funkcijų palyginimas



Kėlinių generavimo minimalaus pokyčio didėjimo tvarka algoritmo tyrimas

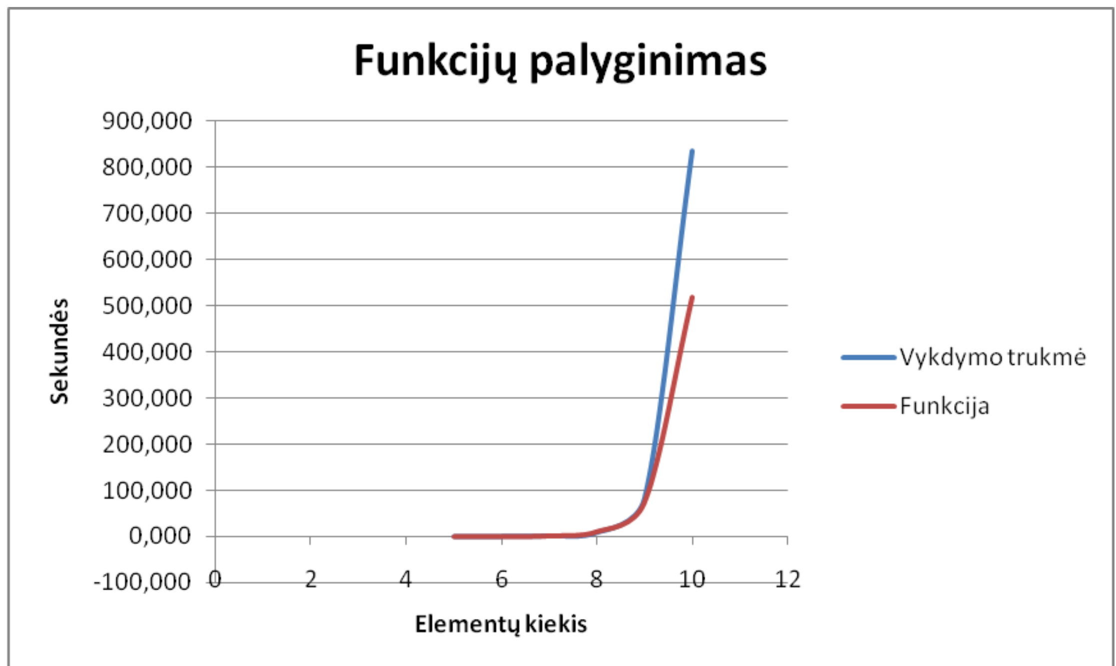
Algoritmas vykdytas su aibe elementų nuo 5 iki 10. Paveiksle 14 pateikiamas gautų rezultatų grafikas.

14 paveikslas. Kėlinių generavimo minimalaus pokyčio didėjimo tvarka grafikas



Gauta f-ja $y = 0{,}000002e^{1{,}937x}$. Išvada – eksponentinis sudėtingumas $\Omega(c^n)$. Paveiksle 15 pateikiamas gautų funkcijų palyginimas.

15 paveikslas. Kėlinių generavimo minimalaus pokyčio didėjimo tvarka funkcijų palyginimas



Visų trijų kėlinių generavimo algoritmų tyrimo rezultatai panašūs. Visų jų sudėtingumo klasė ta pati $\Omega(c^n)$.

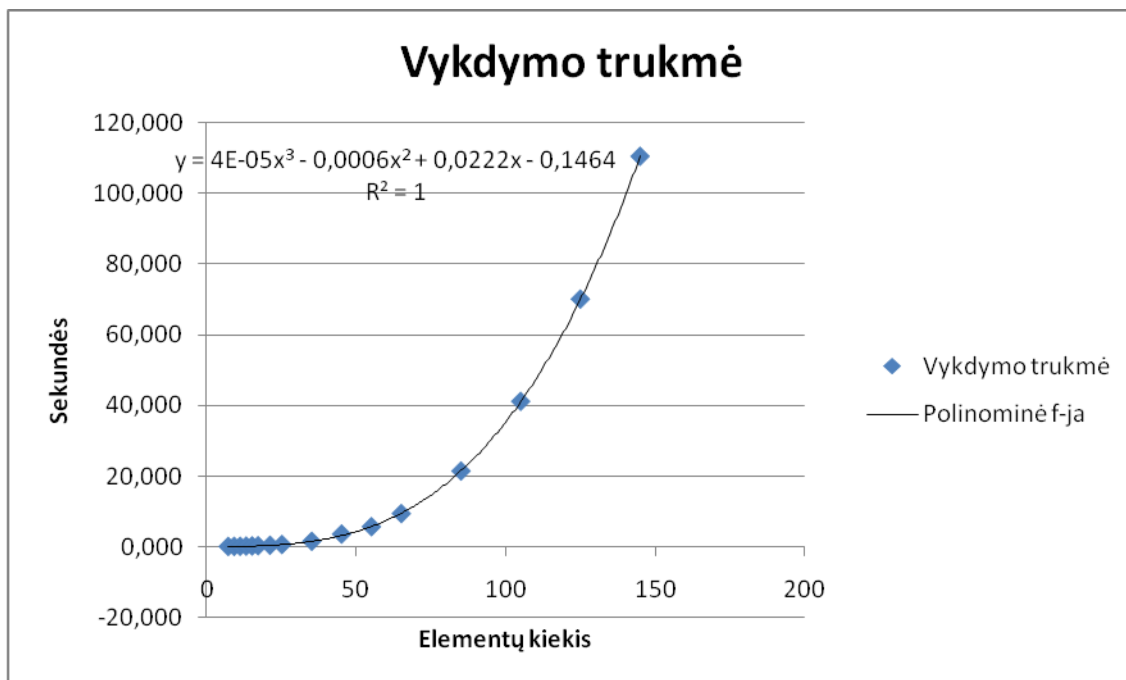
2.3.2 Derinių generavimo algoritmų sudėtingumo tyrimas

Tyrime panaudota aibė elementų nuo 7 iki 145. Tyrimas atliktas pagal aibės elementų kiekį (n, kai C_n^k). Tiriant abu derinių generavimo algoritmus elementų skaičius (k) derinyje parinktas 3.

Derinių generavimo leksikografine didėjimo tvarka algoritmo tyrimas

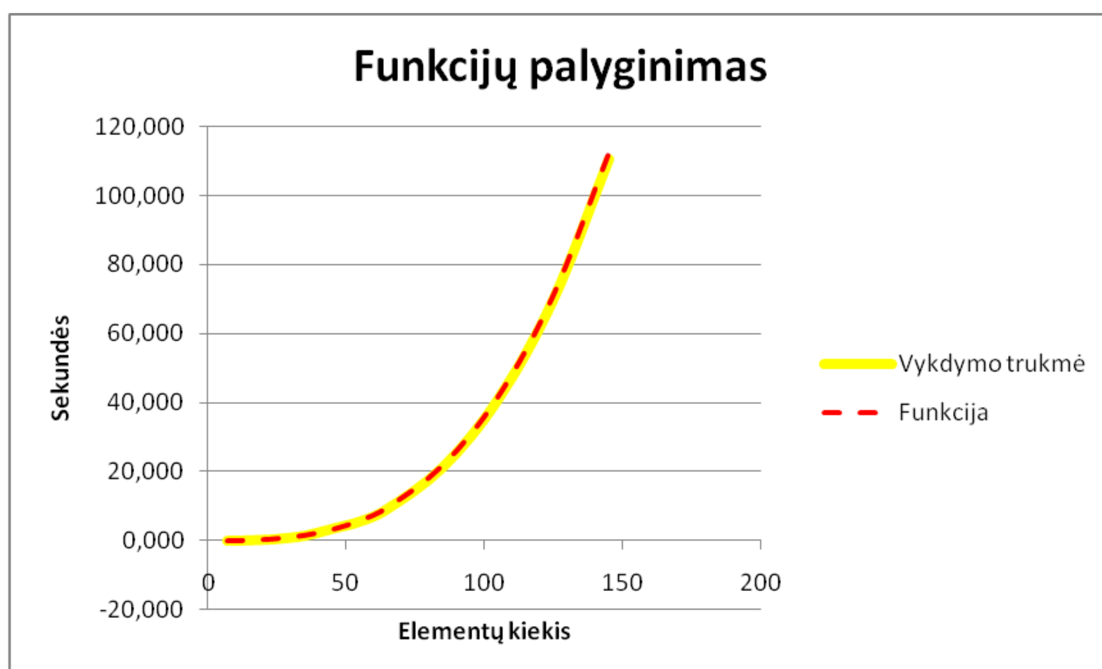
Žemiau pateiktame grafike atvaizduoti tyrimo rezultatai.

16 paveikslas. Derinių generavimo leksikografine didėjimo tvarka grafikas



Gauta f-ja $y = 0{,}00004x^3 - 0{,}0006x^2 + 0{,}0222x - 0{,}1464$ (x – elementų kiekis). Darome išvada, kad algoritmas priklauso polinominei sudėtingumo klasei. Jo sudėtingumas $O(n^3)$. Funkcijos tikslumas lygus 1, tai aiškiai galime matyti paveiksle 17.

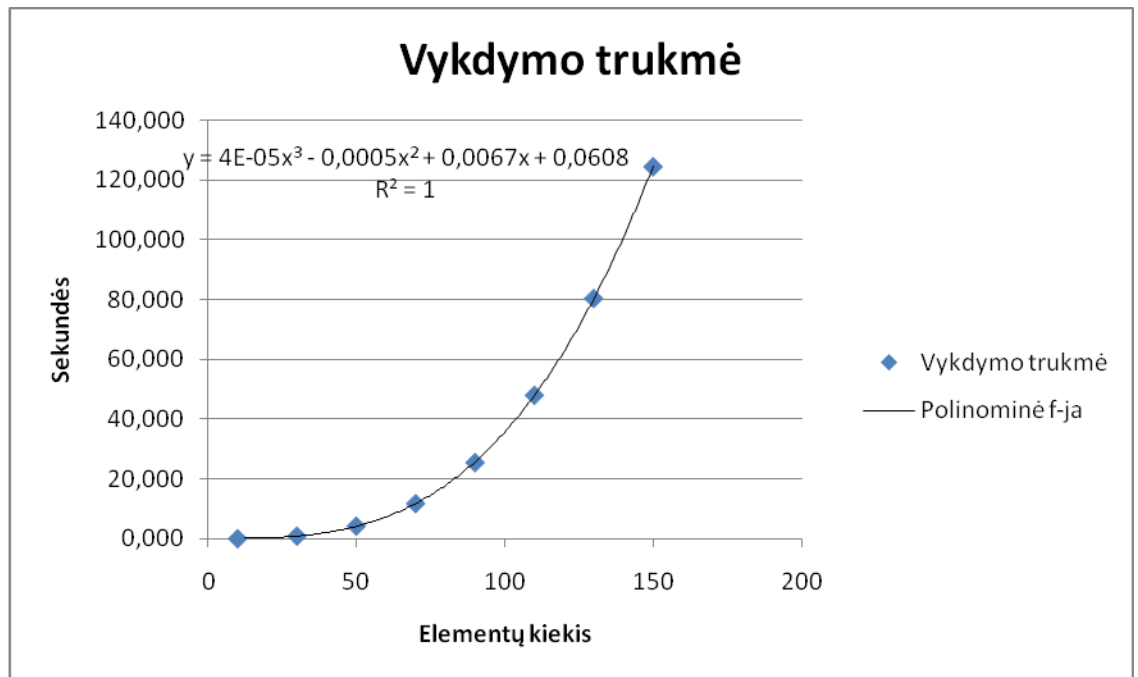
17 paveikslas. Derinių generavimo leksikografine didėjimo tvarka funkcijų palyginimas



Derinių minimalaus pokyčio tvarka algoritmo tyrimas

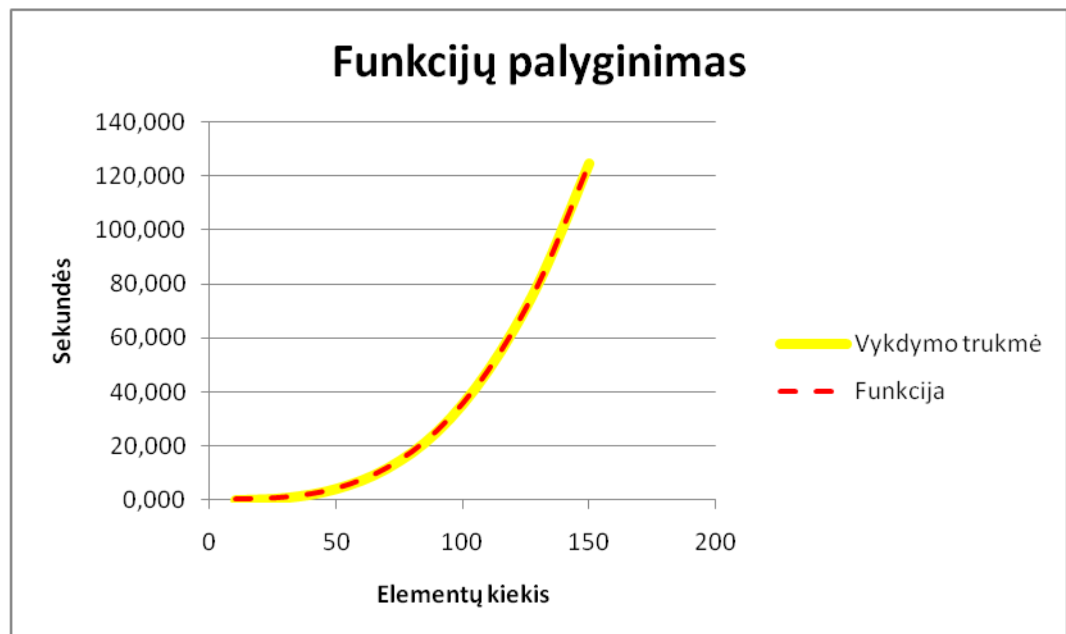
Paveiksle 18 atvaizduoti tyrimo rezultatai.

18 paveikslas. Derinių generavimo minimalaus pokyčio didėjimo tvarka grafikas



Gautoji funkcija $y = 0{,}00004x^3 - 0{,}0005x^2 + 0{,}0067x + 0{,}0608$. Išvada – sudėtingumas $O(n^3)$.

19 paveikslas. Derinių generavimo minimalaus pokyčio didėjimo tvarka funkcijų palyginimas



Abu derinių generavimo algoritmai priklauso tai pačiai sudėtingumo klasei $O(n^3)$.

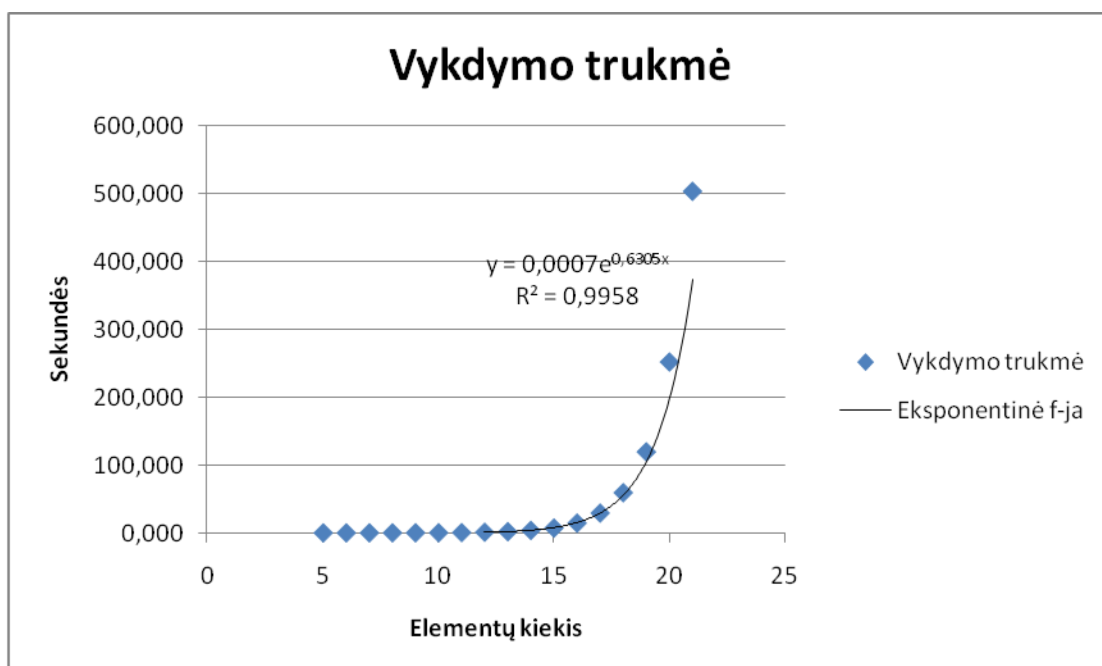
2.3.3 Poaibių (Grėjaus kodų) generavimo algoritmų sudėtingumo tyrimas

Poaibiai generuoti iš 5 – 21 aibės elemento. Pastebėta, kad padidinus aibės elementų skaičių vienetu algoritmo vykdymo laikas dvigubėja.

Poaibių generavimo leksikografine didėjimo tvarka algoritmo tyrimas

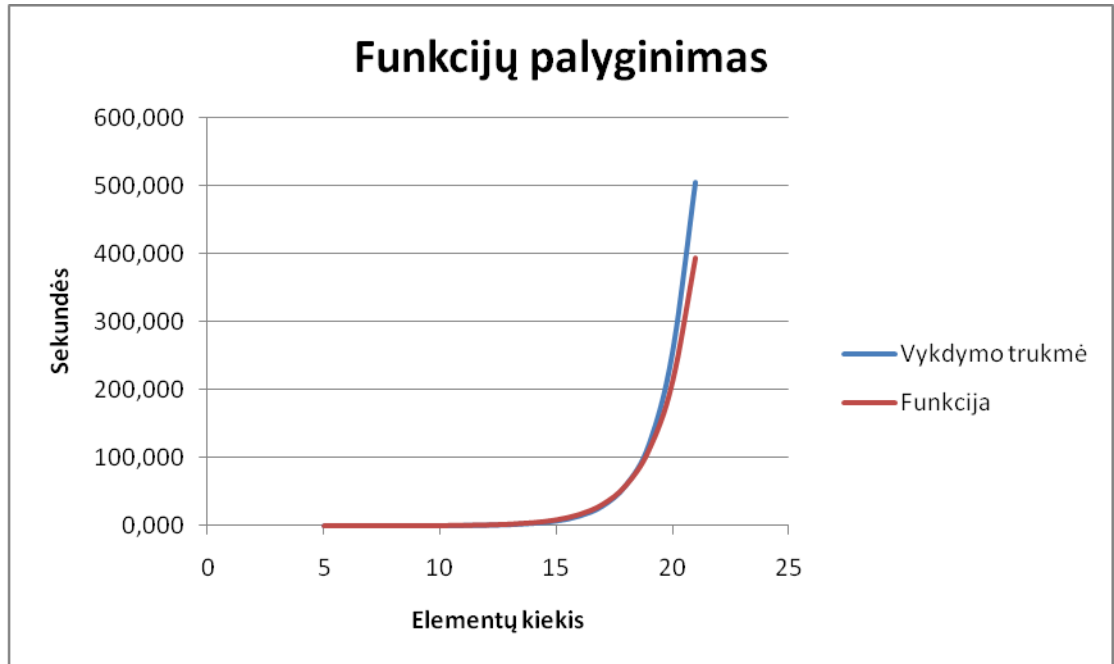
Tyrimo rezultatai.

20 paveikslas. Poaibių generavimo leksikografine didėjimo tvarka grafikas



Gauta eksponentinė funkcija $y = 0{,}0007e^{0{,}6305x}$. Išvada – algoritmo sudėtingumas $\Omega(c^n)$. Algoritmo vykdymo laiko funkcijų palyginimas pateiktas paveiksle 21.

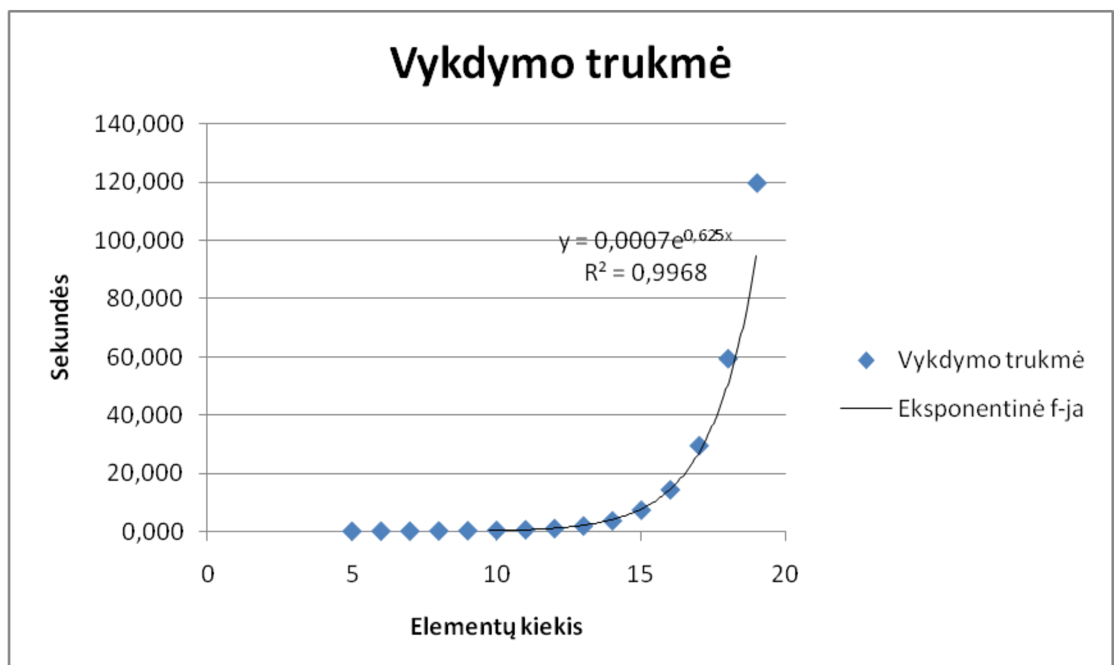
21 paveikslas. Poaibių generavimo leksikografine didėjimo tvarka funkcijų palyginimas



Pirmojo Grėjaus kodų generavimo algoritmo sudėtingumo tyrimas

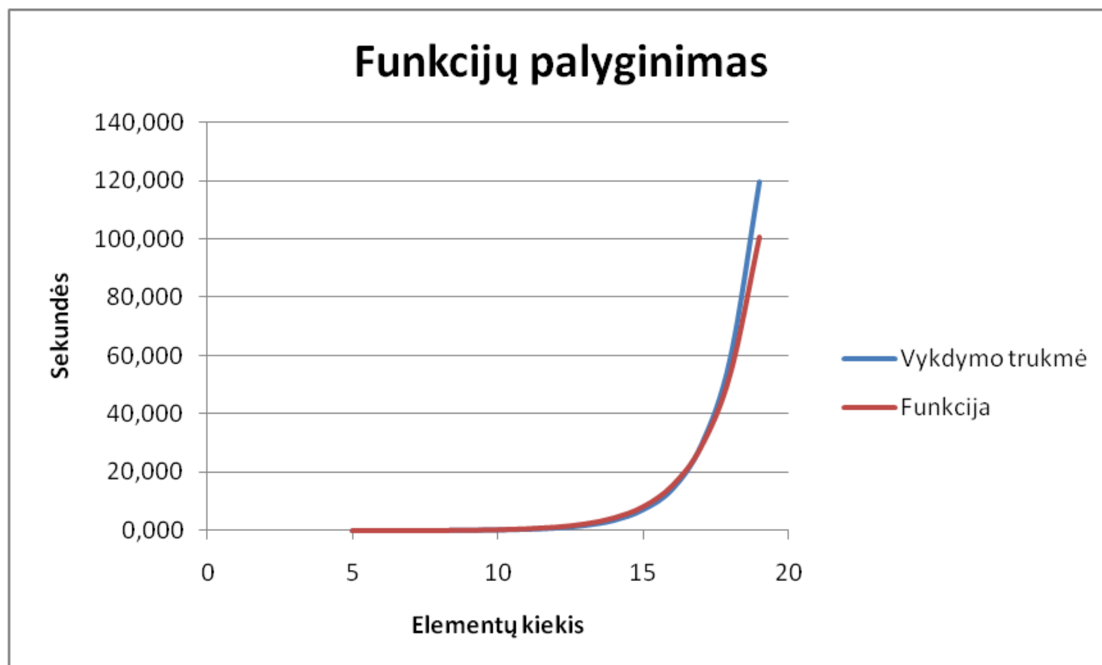
Kadangi šis algoritmas yra paremtas poaibių generavimo leksikografine didėjo tvarka algoritmu, galime daryti prielaidą, kad ir sudėtingumas tas pats. Žemiau pateikti tyrimo rezultatai.

22 paveikslas. Pirmojo Grėjaus kodo generavimo algoritmo grafikas



Gauta funkcija y = 0,0007e^{0,625x}. Aiškiai matome, kad algoritmo sudėtingumas $\Omega(c^n)$. Žemiau pateiktas funkcijų palyginimas.

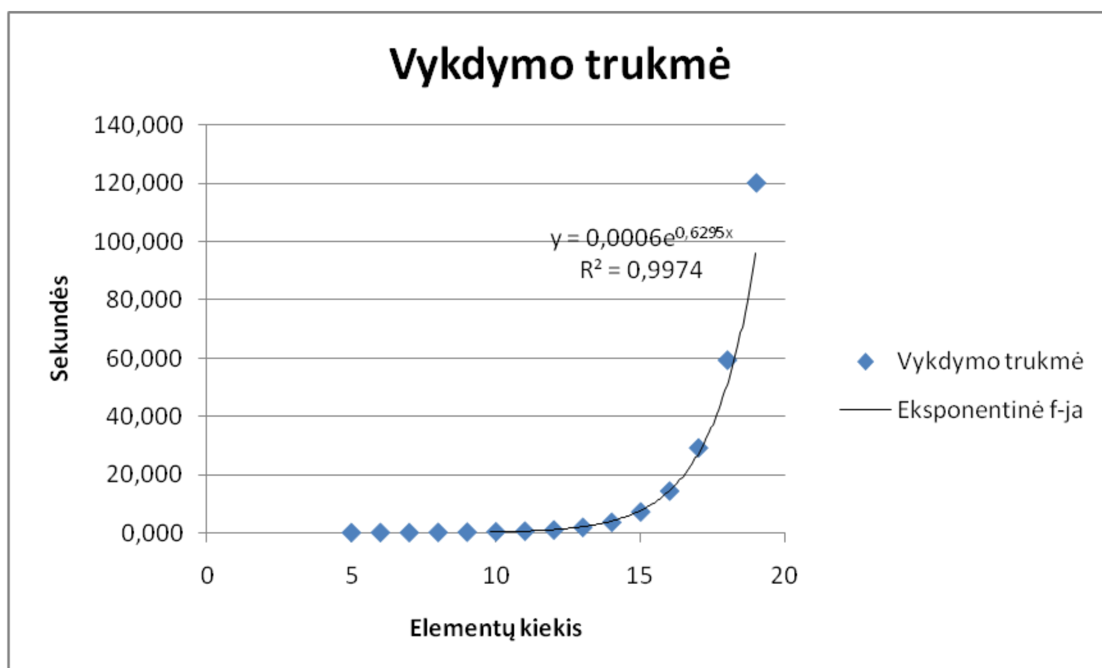
23 paveikslas. Pirmojo Grėjaus kodo generavimo algoritmo funkcijų palyginimas



Antrojo Grėjaus kodų generavimo algoritmo sudėtingumo tyrimas

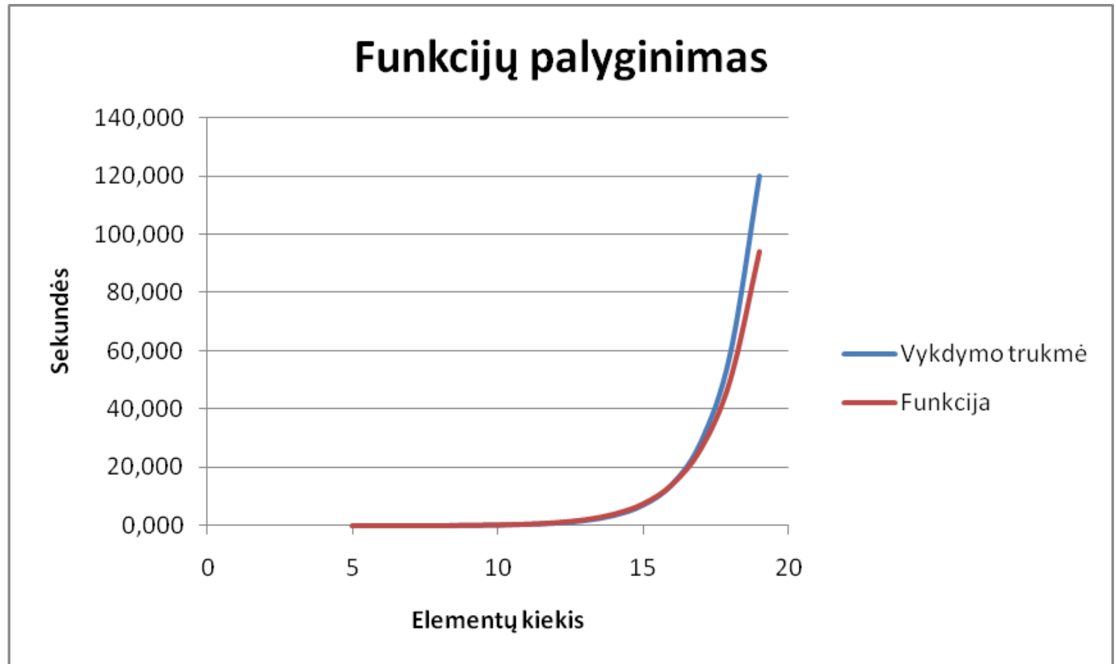
24 paveiksle atvaizduoti tyrimo duomenys.

24 paveikslas. Antrojo Grėjaus kodo generavimo algoritmo grafikas



Gauta funkcija $y = 0{,}0006e^{0{,}6295x}$. Eksponentinis sudėtingumas $\Omega(c^n)$.

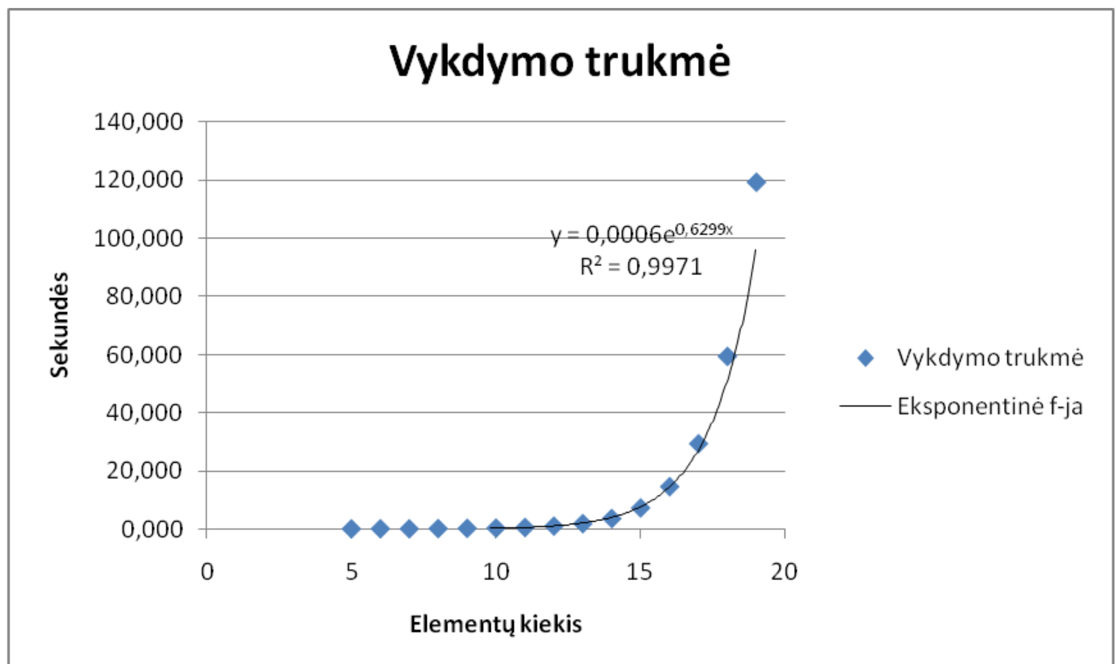
25 paveikslas. Antrojo Grėjaus kodo generavimo algoritmo funkcijų palyginimas



Trečiojo Grėjaus kodų generavimo algoritmo sudėtingumo tyrimas

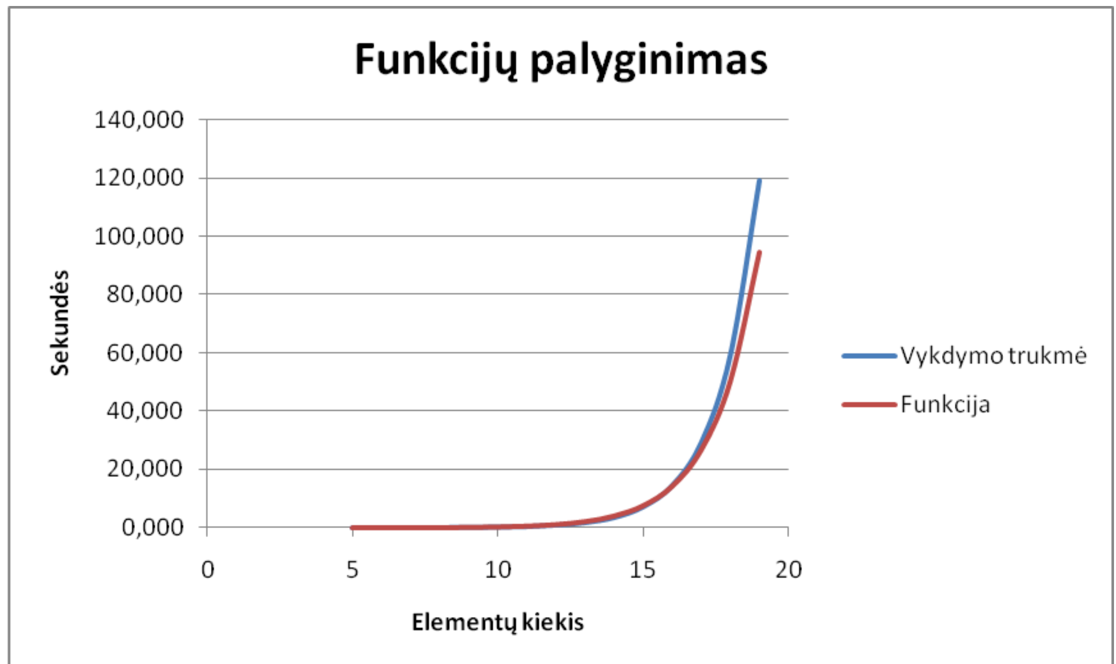
Tyrimo rezultatai atvaizduoti paveiksle 26.

26 paveikslas. Trečiojo Grėjaus kodo generavimo algoritmo grafikas



Gautoji funkcija y = 0,0006e^{0,6299x}. Algoritmo sudėtingumas $\Omega(c^n)$.

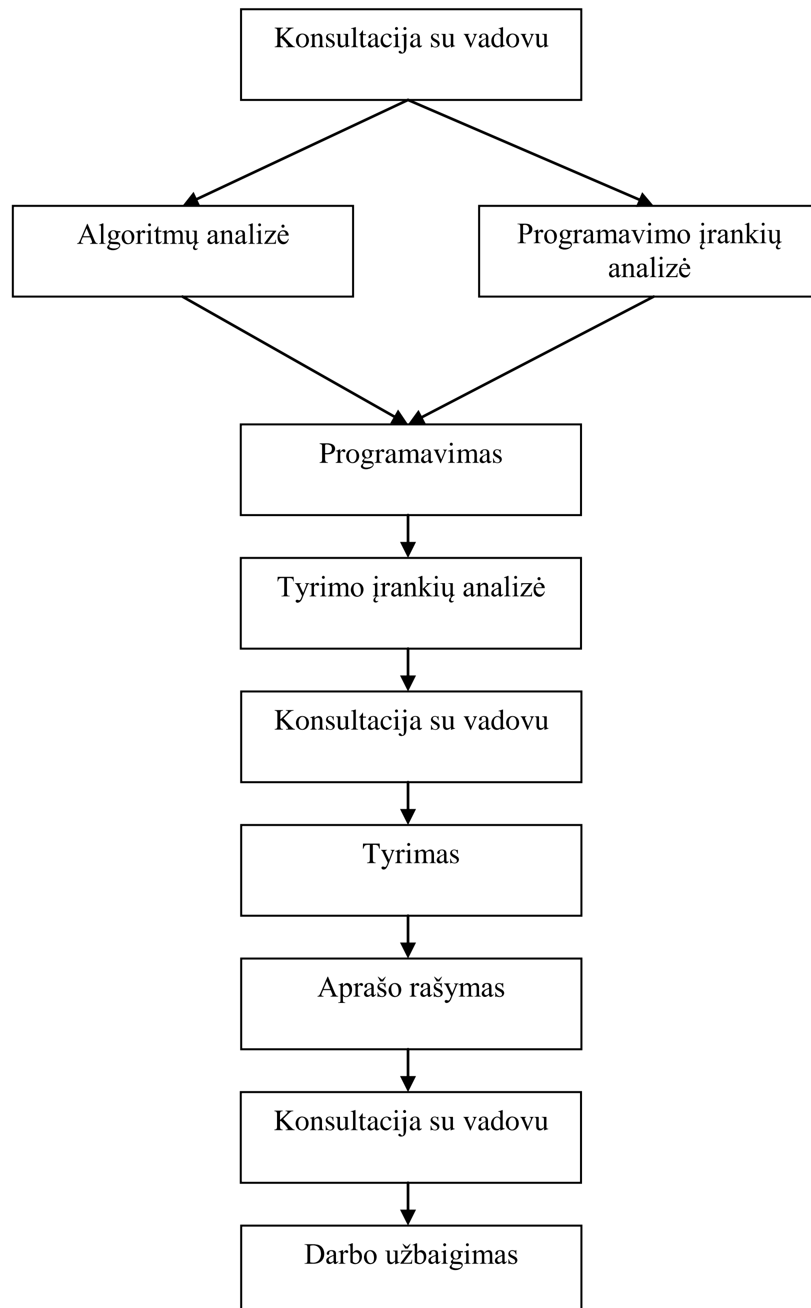
27 paveikslas. Trečiojo Grėjaus kodo generavimo algoritmo funkcijų palyginimas



Poaibių generavimo leksikografine didėjimo tvarka bei visų trijų Grėjaus kodų generavimo algoritmų sudėtingumas vienodas $\Omega(c^n)$.

3 DARBO EIGOS APRAŠYMAS

3.1 Darbų eigos grafas



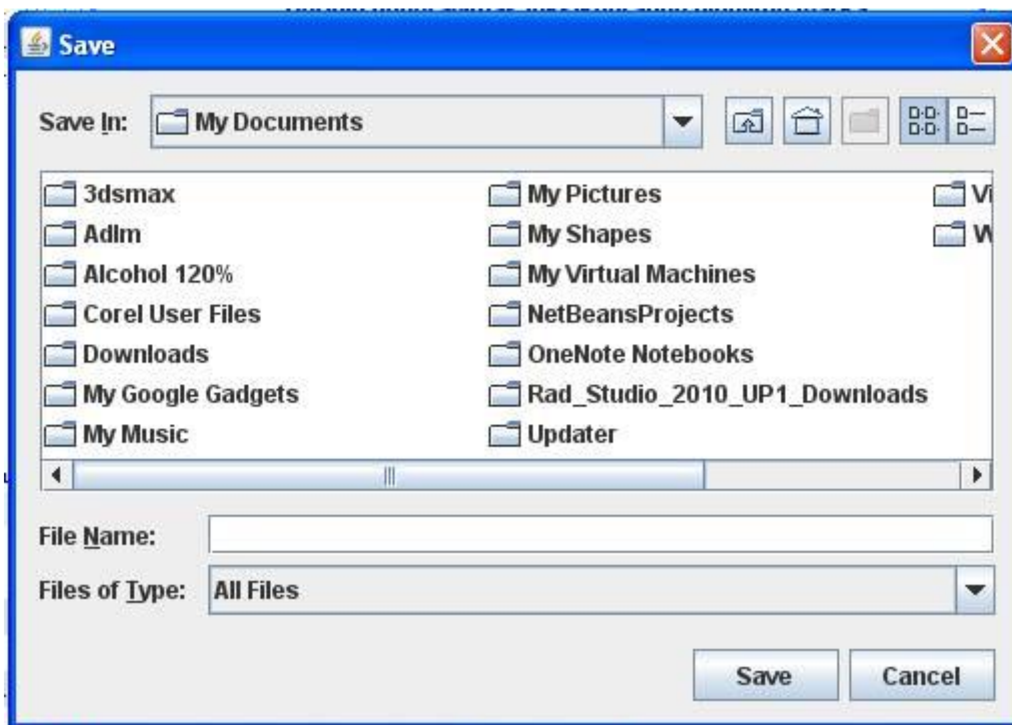
3.2 Problemų sprendimas

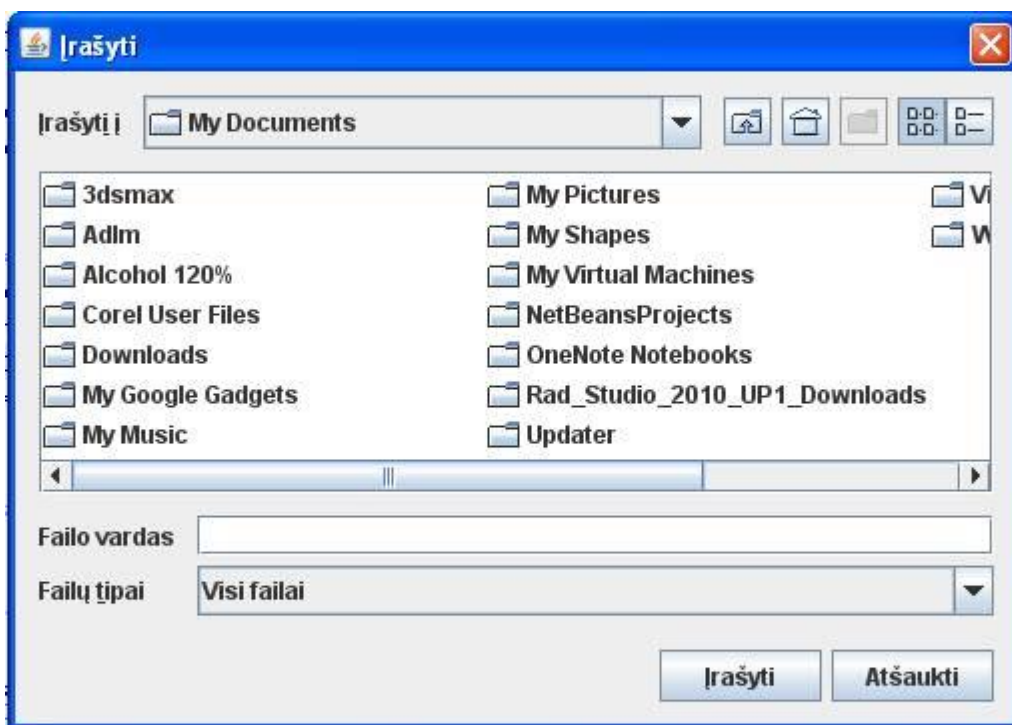
Testuojant programą pastebėta, kad jos negalima uždaryti, kol nebaigtas generavimo algoritmo vykdymas. Programą buvo galima nutraukti pasinaudojant operacinės sistemos galimybėmis. Nustatyta, kad problema kilo dėl to, kad ta pati gija realizavo grafinę vartotojo sąsają bei vykdė generavimo algoritmą. Problema išspręsta generavimo algoritmo metodą patalpinus į atskirą klasę, kuri realizuojama kaip atskira gija. Generavimo algoritmą vykdo naujai sukurta gija. Dėl to pagrindinė gija lieka laisva ir gali vykdyti vartotojo sąsajos komandas tame tarpe nutraukti generavimo algoritmo vykdymą. Žemiau pateiktas klasės, kurios objektą realizuoja naujai sukurta gija, pavyzdys:

```
private class generate extends Thread {
   public void run() {
      //generavimo algoritmas
            }
}
```

Sugeneruotus poaibius programa įrašydavo į automatiškai sukurtą tekstinį failiuką ta pačiame aplanke, kuriame patalpintas vykdomasis programos failas. Testuojant programą nuspręsta suteikti galimybę vartotojui pačiam pasirinkti, kur ir kokiu pavadinimu išsaugoti rezultatų failą. Paaiškėjo, kad „Java“ vartojo sąsaja nėra išversta į lietuvių kalbą. Tad failų pasirinkimo dialogo langas „kreipėsi“ į vartotoją anglų kalba. Problema išspręsta pačiam lokalizavus failų pasirinkimo lango grafinę sąsają klasės „UIManager“ pagalba. Lokalizuojant naudotasi „Enciklopediniu kompiuterijos žodynu „ [10]. 28 paveiksle pavaizduotas failų pasirinkimo langas prieš ir po lokalizavimo.

28 paveikslas. Failų pasirinkimo dialogo langas





Visos problemos sėkmingai išspręstos.

IŠVADOS

1. Kėlinių generavimo leksikografine, antileksikografine, minimalaus pokyčio tvarka algoritmų sudėtingumas eksponentinis $\Omega(c^n)$.
2. Derinių generavimo leksikografine ir minimalaus pokyčio tvarka algoritmų sudėtingumas polinominis $O(n^3)$.
3. Poaibių ir Grėjaus kodų generavimo algoritmų sudėtingumas eksponentinis $\Omega(c^n)$.

Literatūros ir informacinių šaltinių sąrašas

1. Plukas K., Mačikėnas E., Jarašiūnienė B., Mikuckienė I. (2008). *Taikomoji diskrečioji matematika*. Kaunas: Technologija.
2. Липский В. (1988). *Комбинаторика для программистов.* Москва: Мир.
3. Reingold E. M., Nievergelt J., Deo N. (1977). *Combinatorial algorithms: theory and practise.* EnglewoodCiffs, New Jersey: Prentice-Hall, Inc.
4. Adomavičius J. Informatika2.(2001). *Algoritmai ir jų diegimas*. Kaunas: Technologija.
5. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. (2001). *Introduction to Algorithms, Second Edition*. London: The MIT Press.
6. Lassaigne R., Rougemont M. (1999). *Logika ir algoritmų sudėtingumas*. [Lietuviškojo leidimo pratarmė ir priedas „Kai kurie papildomi logikos rezultatai“, Norgėla S.] Vilnius: Žara.
7. http://www.ics.uci.edu/~eppstein/161/960312.html [žr. 2010-04-05].
8. http://lt.wikipedia.org/wiki/Algoritmų_sudėtingumas [žr. 2010-04-05].
9. http://en.wikipedia.org/wiki/Big_O_notation [žr. 2010-04-05].
10. Dagienė V., Grigas G., Jevsikova T.(2009). *Enciklopedinis kompiuterijos žodynas*. Prieiga per internetą: http://www.likit.lt/term/enc.html [žr. 2010-04-10]

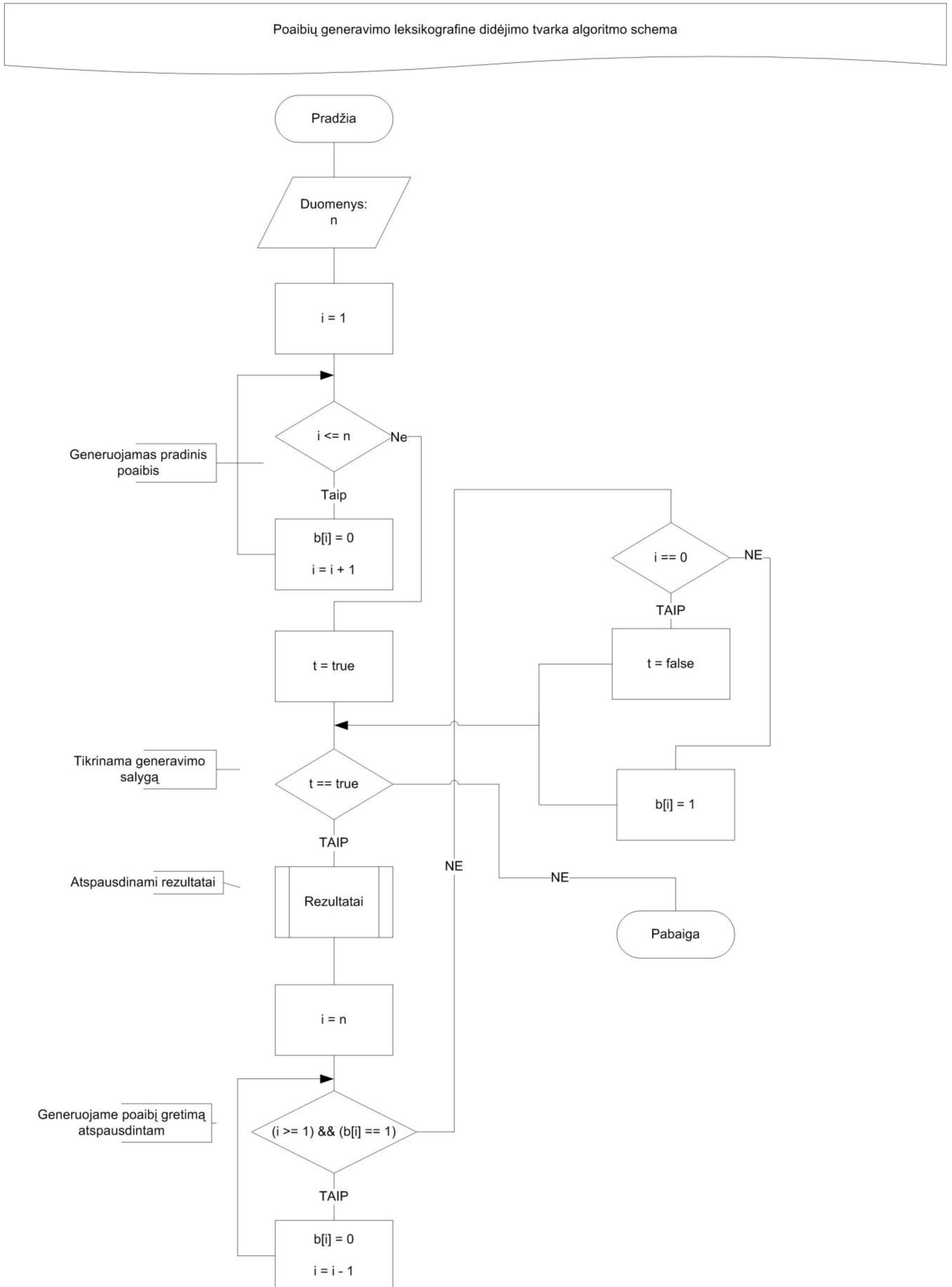
Anotacija (Summary)

Darbe tiriamas kėlinių, derinių, poaibių ir Grėjaus kodų generavimo algoritmų sudėtingumas. Atliekama algoritmų analizė. Tyrimo tikslams sukurta programa realizuojanti minėtus algoritmus.

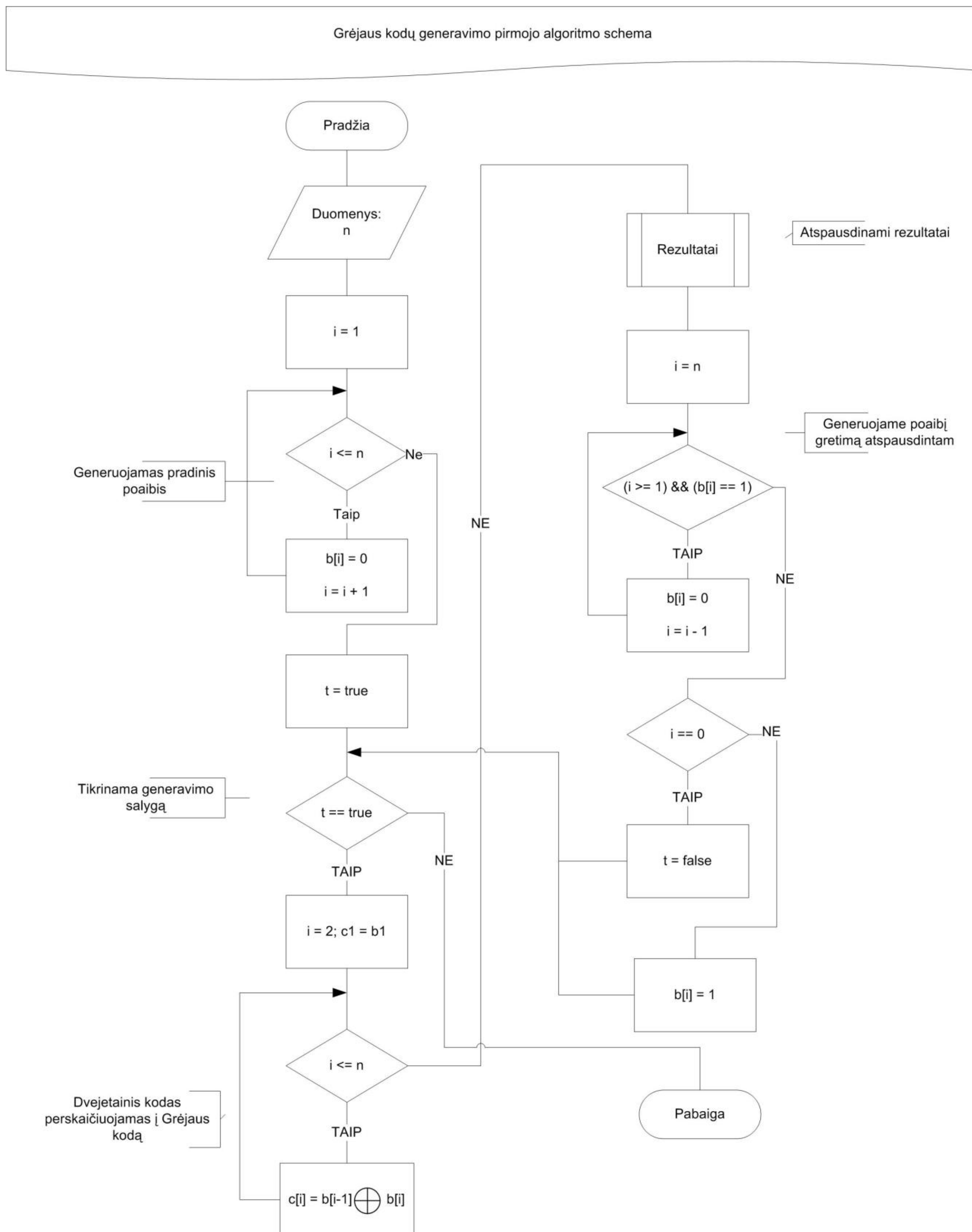
The complexity research of permutations, combinations, subsets and Gray codes generating algorithms is provided in this paper. Algorithms are analyzed and implemented in the application developed for research purposes.

Priedai

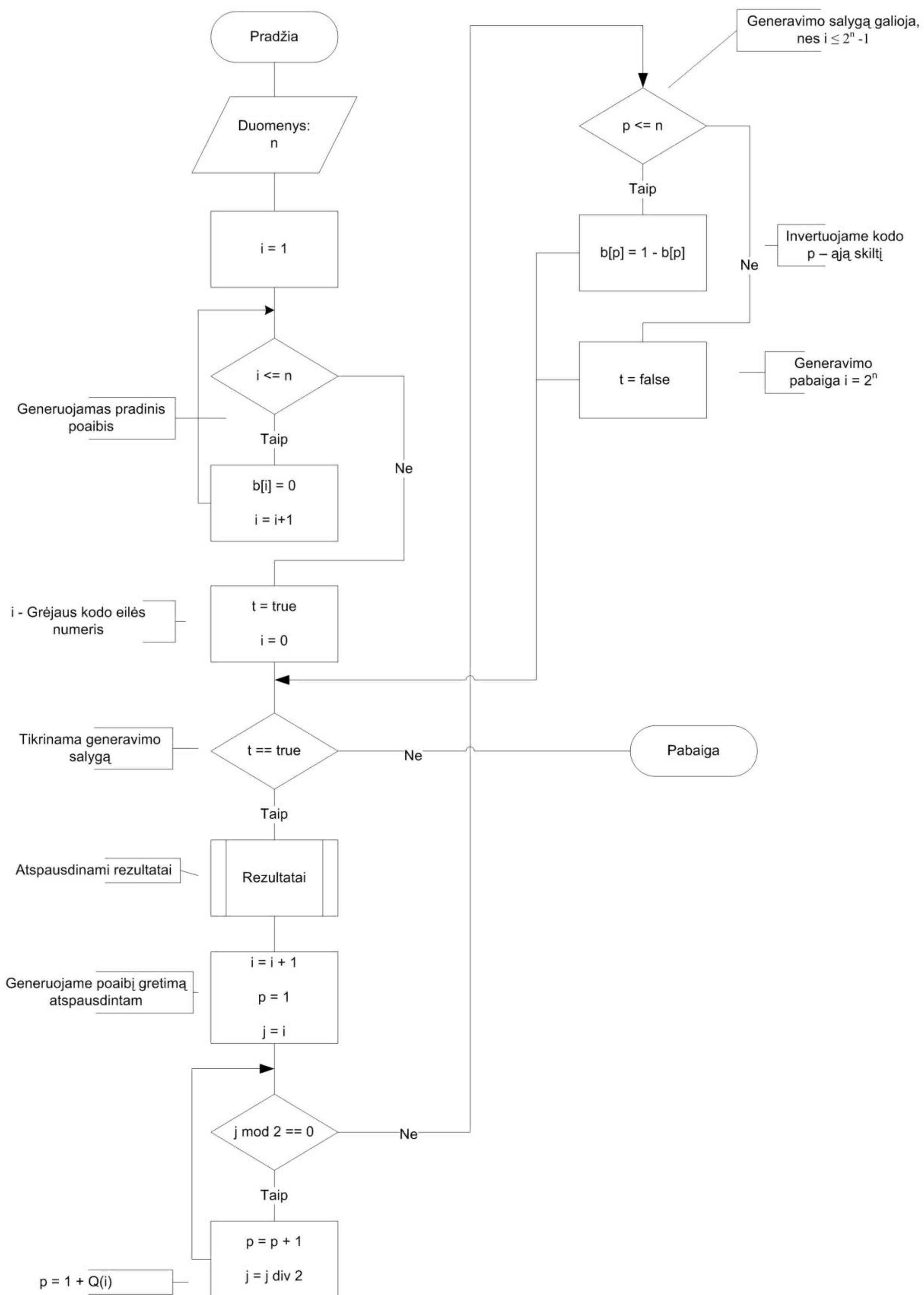
1 priedas. Poaibių generavimo leksikografine didėjimo tvarka algoritmo blokinė schema



2 priedas. Pirmojo Grėjaus kodų generavimo algoritmo blokinė schema

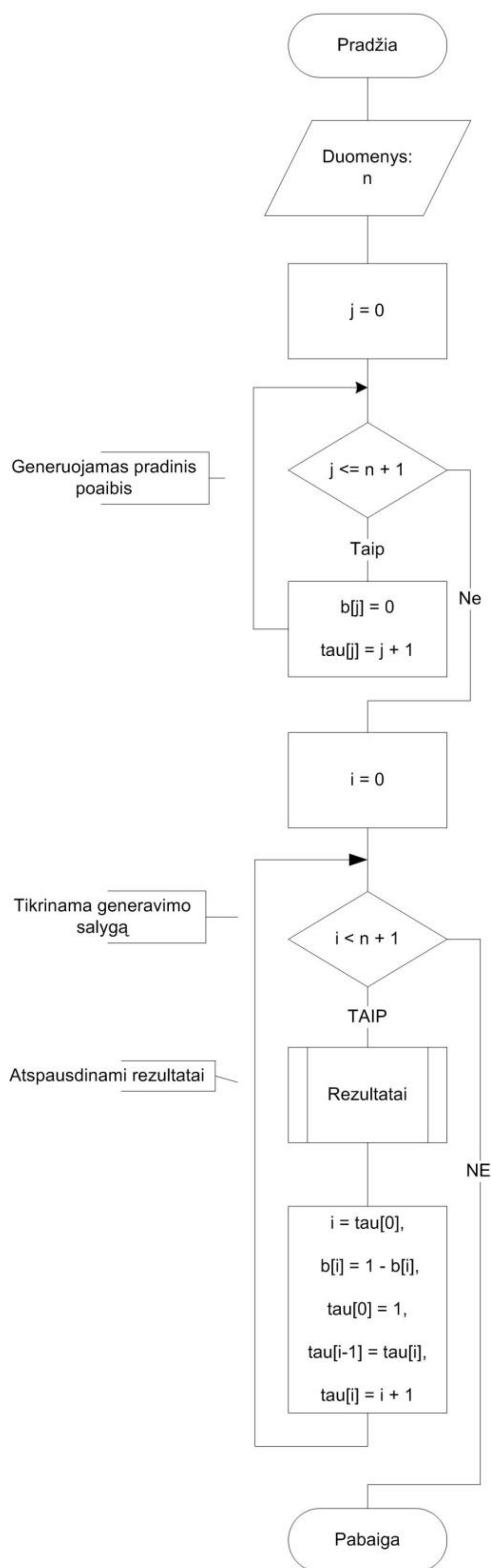


3 priedas. Antrojo Grėjaus kodų generavimo algoritmo blokinė schema

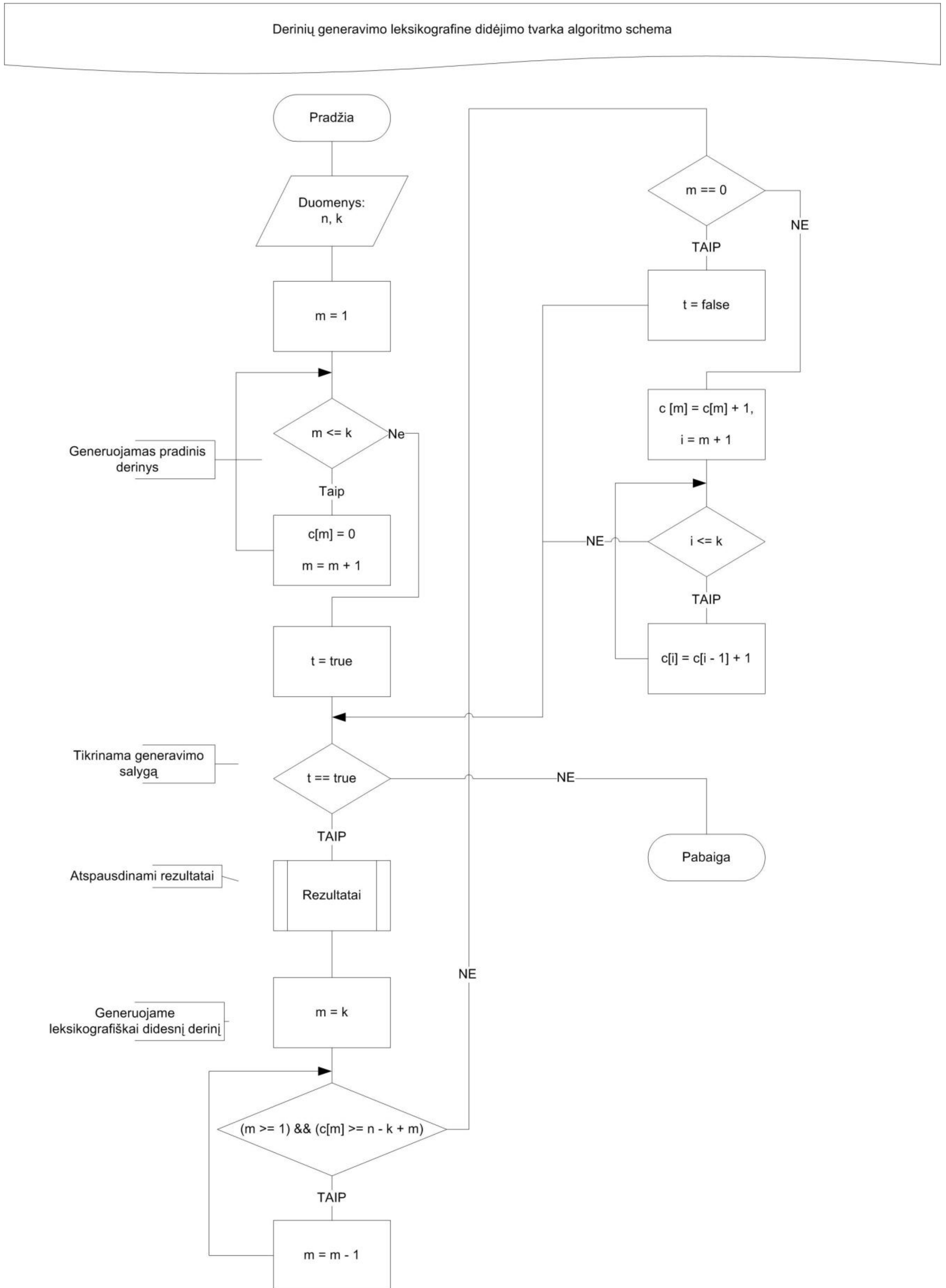


4 priedas. Trečiojo Grėjaus kodų generavimo algoritmo blokinė schema

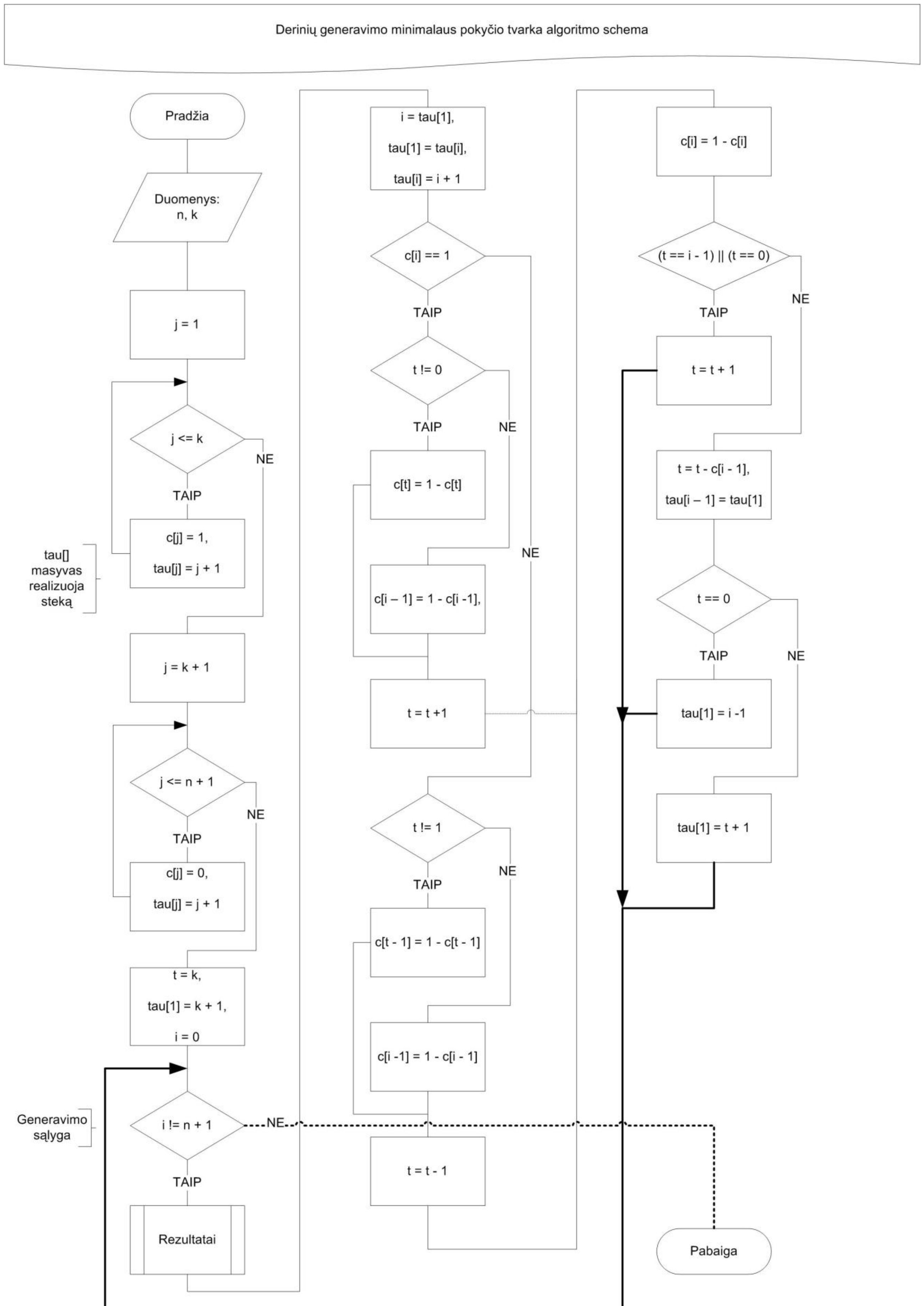
Grėjaus kodų generavimo trečiojo algoritmo schema



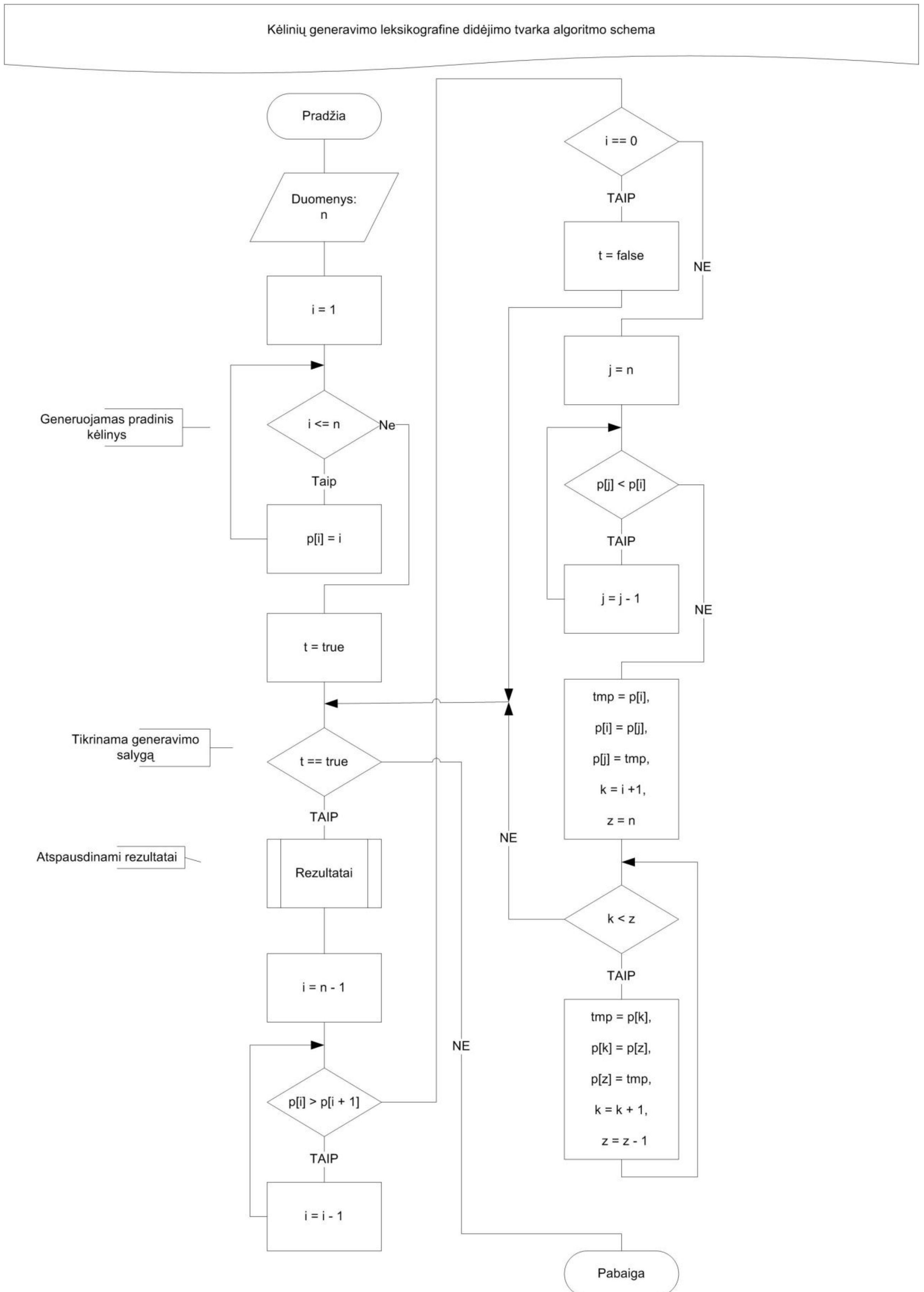
5 priedas. Derinių generavimo leksikografine tvarka algoritmo blokinė schema



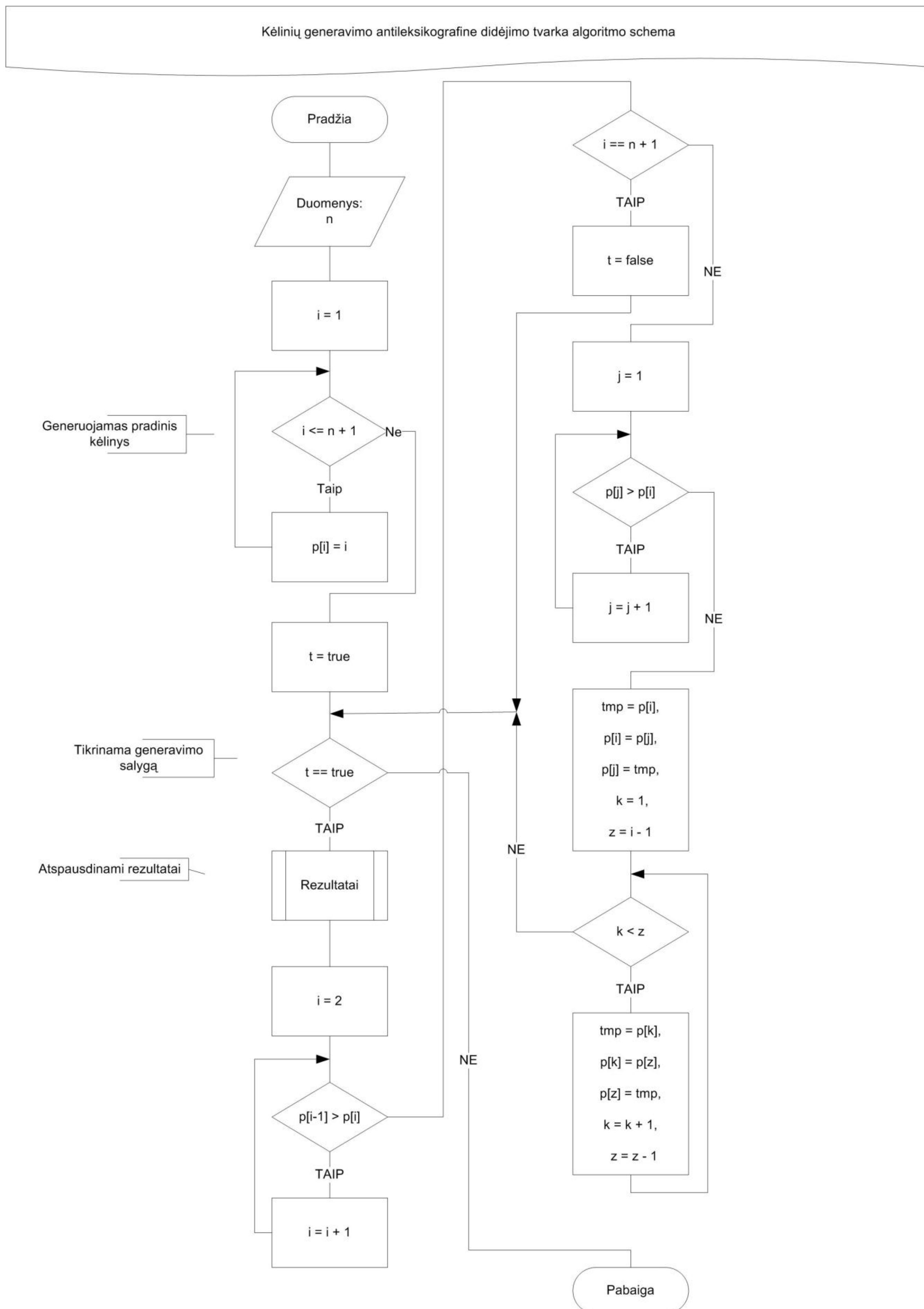
6 priedas. Derinių generavimo minimalaus pokyčio tvarka algoritmo blokinė schema



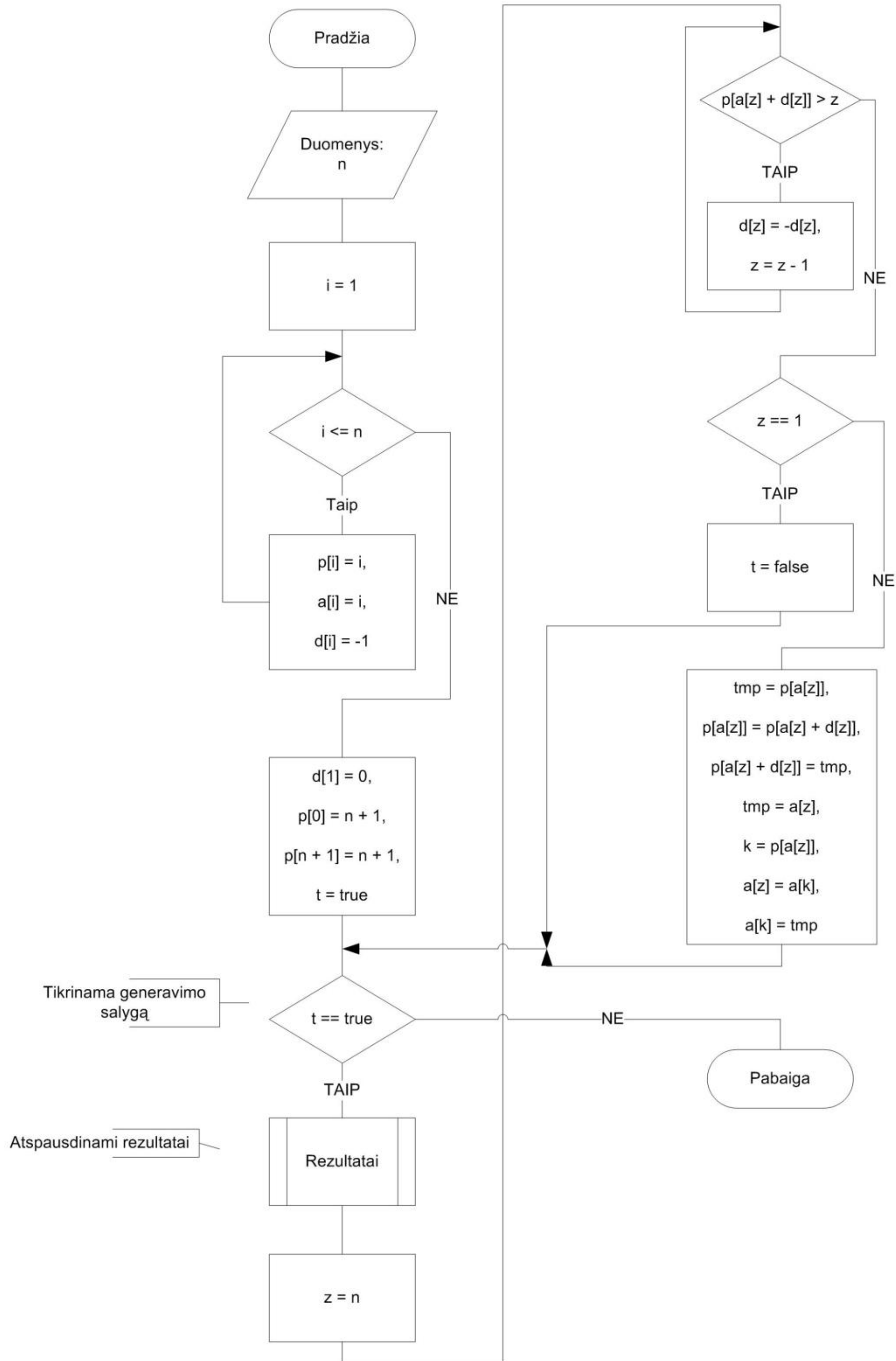
7 priedas. Kėlinių generavimo leksikografine tvarka algoritmo blokinė schema



8 priedas. Kėlinių generavimo antileksikografine tvarka algoritmo blokinė schema

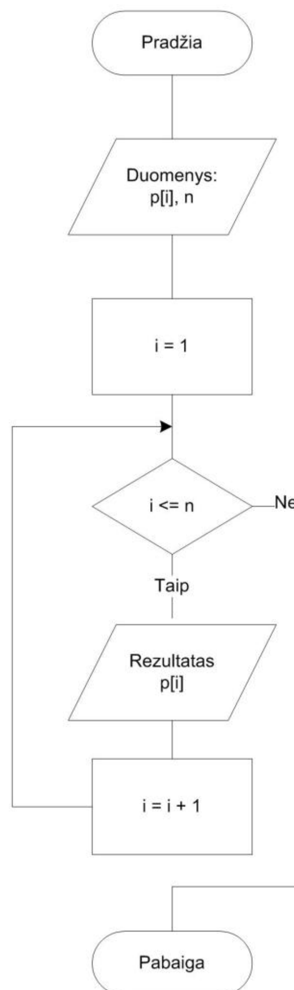


9 priedas. Kėlinių generavimo minimalaus pokyčio tvarka algoritmo blokinė schema



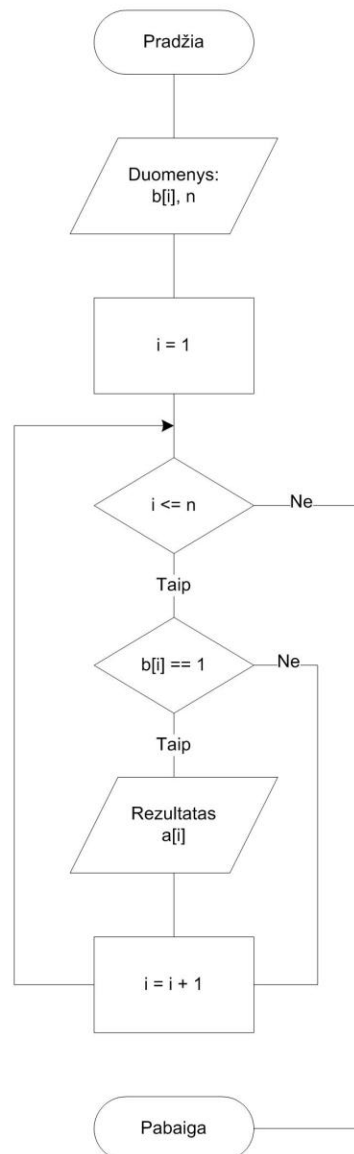
10.1 priedas. Algoritmo „Rezultatai“ skirto derinių generavimo leksikografine tvarka bei visiems kėlinių algoritmams blokinė schema

Algoritmas „Rezultatai“

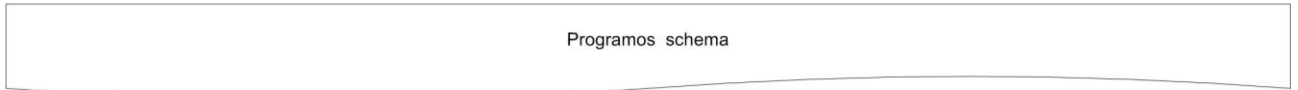


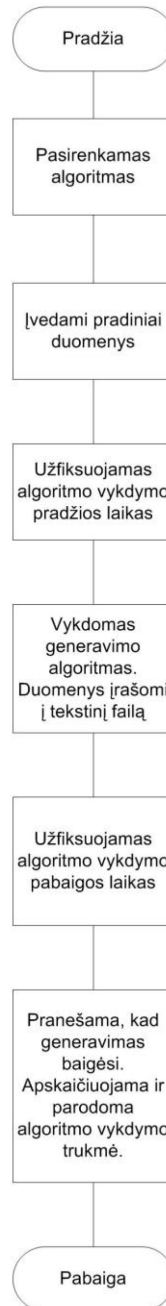
10.2 priedas. Algoritmo „Rezultatai“ skirto likusiems algoritmams blokinė schema

Algoritmas „Rezultatai“



11 priedas. Programos blokinė schema



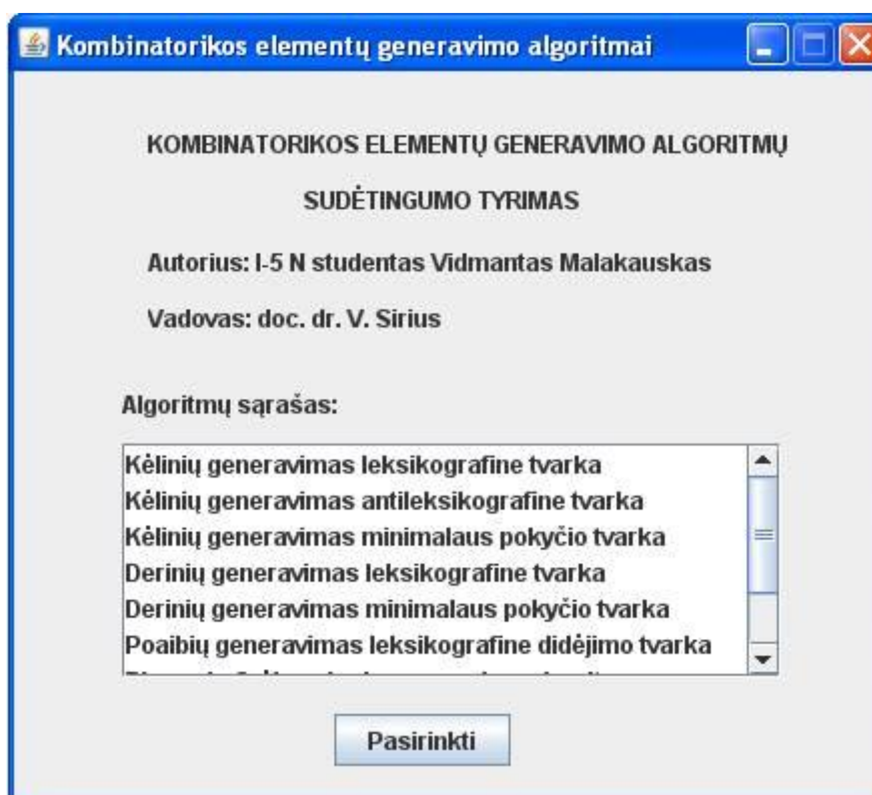


12 priedas. Programos naudojimo instrukcija

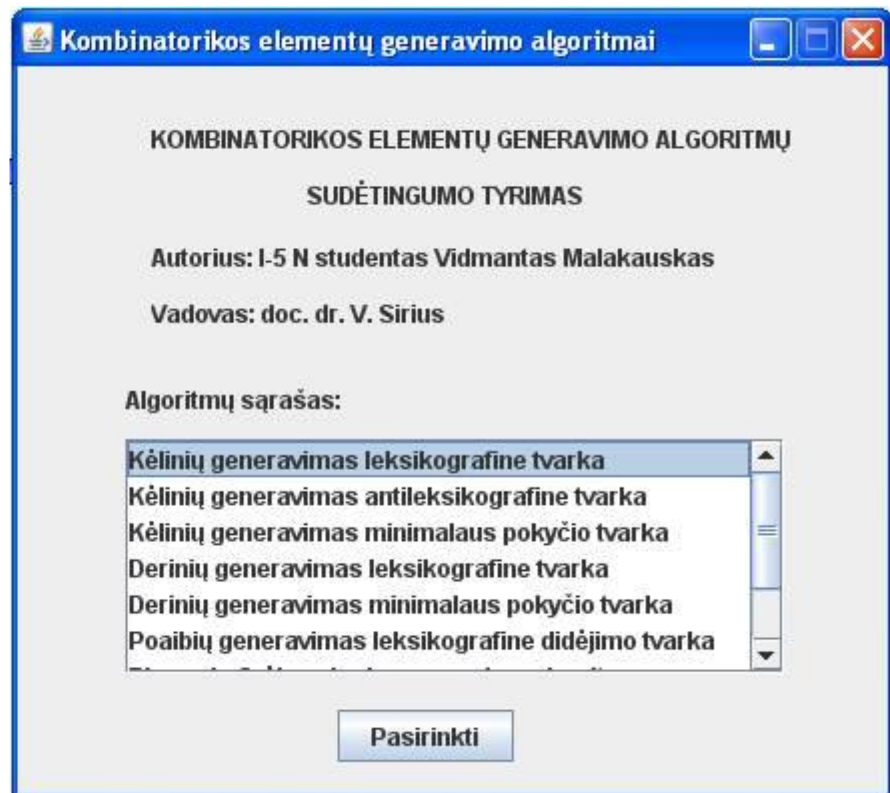
Sistemos reikalavimai. Operacinės sistemos: Windows 7/Vista/XP/2000, Linux, Apple OS X, Solaris. Priklausomai nuo naudojamos operacinės sistemos reikia įdiegti Java virtualią mašiną. Ji pasiekiama adresu http://www.java.com/en/download/manual.jsp . Maksimaliai galima generuoti poaibius iš 999 elementų.

Naudojimas:

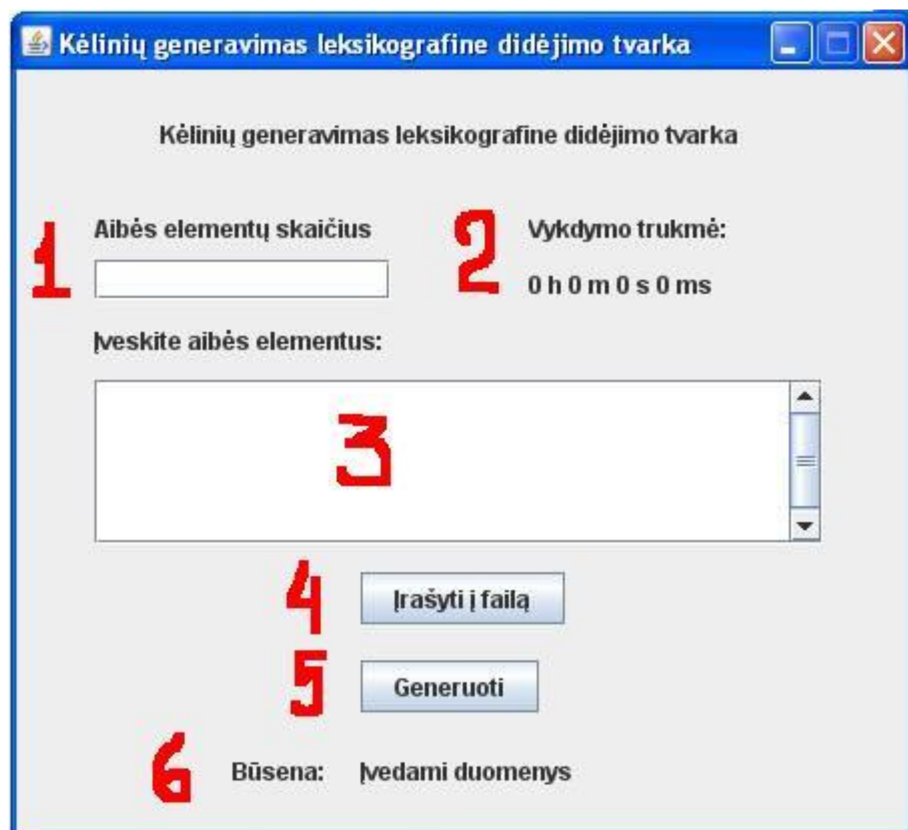
- Paleidus programą „Algoritmai.jar“ pasirodo langas, kuriame galime pasirinkti norimą algoritmą:



- Pažymėkite pageidaujamą algoritmą ir spauskite „Pasirinkti“.



- Atsidarys dar vienas langas. Laukelyje 1 įveskite pageidaujamą aibės elementų skaičių. Laukelis 2 rodo generavimo trukmę. Į laukelį 3 įveskite aibės elementus. Paspaudus mygtuką 4 pasirinksite, į kokį failą įrašyt sugeneruotus elementus. Paspaudus mygtuką 5 prasidės generavimas. Laukelis 6 rodo algoritmo vykdymo būseną. Kai vyksta generavimas rodoma „Vykdoma“, o kai baigtas – „Generavimas baigtas“.



- Sugeneruoti elementai įrašomi į tekstinį failą. Kiekviens elementų rinkinys įrašomas į atskirą eilutę.

Kėlinių generavimas leksikografine didėjimo tvarka

Kėlinių generavimas leksikografine didėjimo tvarka

Aibės elementų skaičius

3

Vykdymo trukmė:

0 h 0 m 0 s 0 ms

Įveskite aibės elementus:

1;2;3

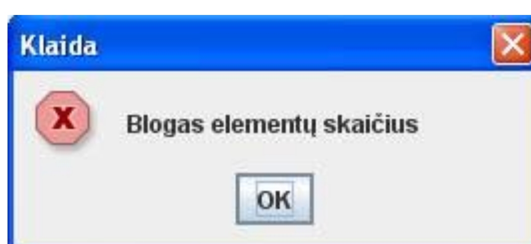
Įrašyti į failą

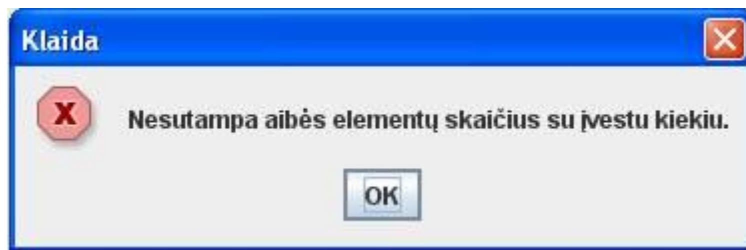
Generuoti

Būsena: Įvedami duomenys

```
1 {1,2,3}
2 {1,3,2}
3 {2,1,3}
4 {2,3,1}
5 {3,1,2}
6 {3,2,1}
7
```

- Programa tikrina, ar į duomenų laukelius yra įvedami skaičiai, ar galimas duomenų įrašymas į failą, ar sutampa aibės elementų skaičius su realiai įvestu kiekiu, ar sukurtas rezultatų failas. Klaidos atveju parodomi įspėjamieji pranešimai.





- Uždarius bet kurį programos langą, programa išjungiama.