Kauno technologijos universitetas

Informatikos fakultetas

Trečio laboratorinio darbo ataskaita

3LD individuali užduotis

Aistis Jakutonis

Studentas

Dalius Makackas
Andrius Kriščiūnas
Tadas Kraujalis
Vidmantas Rimavičius

Dėstytojai

Kaunas, 2025

Turinys

1. Pirmos užduoties dalys

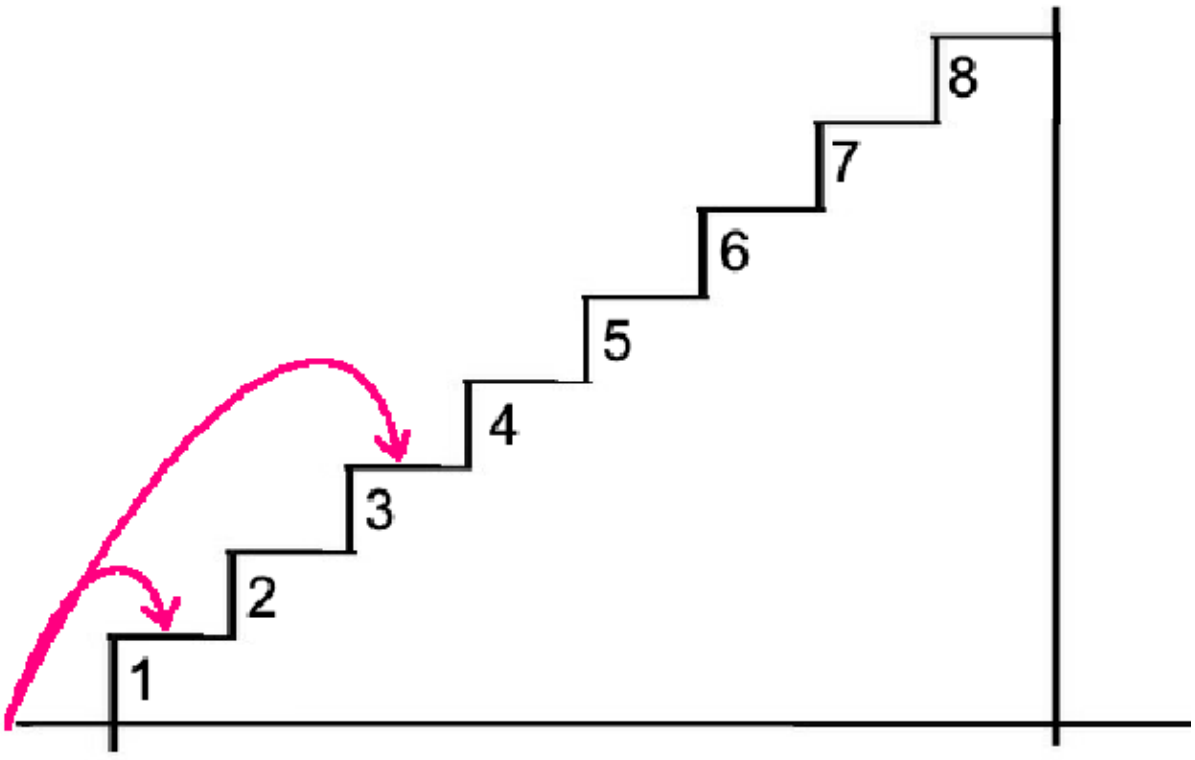
1.1. Pateikti uždavinio sprendimo algoritmą

> Turime *n* skersinių kopėčias, priešais kurias stovi žmogus. Vienu žingsniu jis gali arba užlipti ant sekančio skersinio arba praleisdamas du skersinius užšokti ant trečio skersinio (nuo laiptų apačios gali atsidurti ant pirmo arba trečio skersinio). Kiek yra būdų pasiekti paskutinį skersinį?

1 pav. Užduotis

Šioje užduotyje reikia pirmiausia atkreipti dėmesį į vieno žingsnio variantus:

- Žmogus gali užlipti ant sekančio skersinio (tai reiškia, kad jei sakysime, jog žmogus stovi ant nulinio skersinio tai, kai įvykdys šį žingsnį jis jau stovės ant pirmojo skersinio).
- Žmogus gali peršokti du skersinius (tai reiškia, kad jei vėl sakysime, jog žmogus stovi ant nulinio skersinio tai, kai jis atliks peršokimo per du skersinius veiksmą jis jau stovės ant trečiojo skersinio).

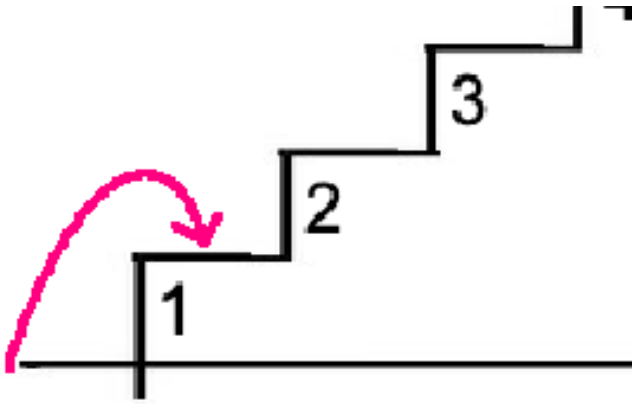


2 pav. Užduoties paaiškinimas

Toliau sprendžiant šį uždavinį sakysime, kad $n - skersinių\ skaičius$. Taip pat galimų būdų pasiekti paskutinį skersinį įsivardinsime kaip: $S(n)$.

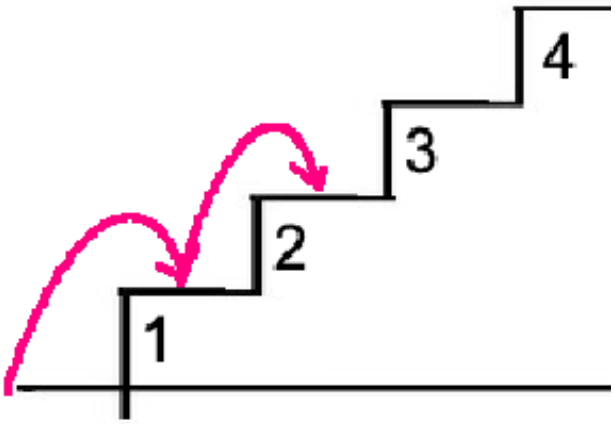
Taigi pereikime prie pavyzdžių:

$$S(0) = 1$$
$$S(1) = 1$$



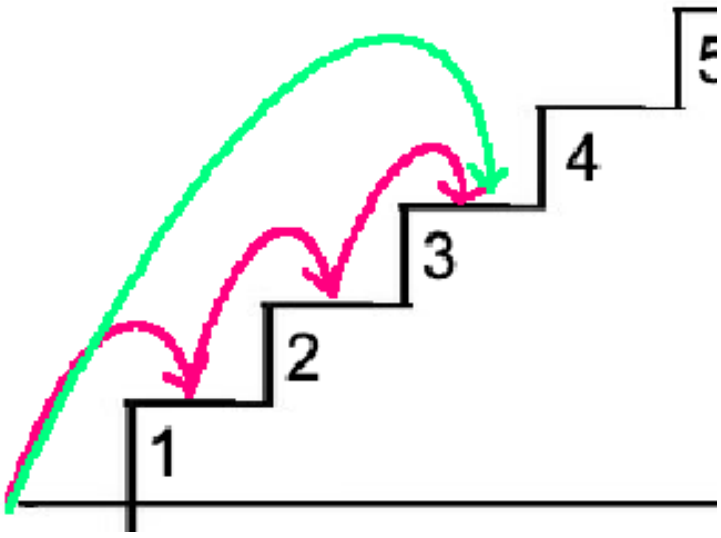
3 pav. Užduoties pavyzdys

$$S(2) = 1$$



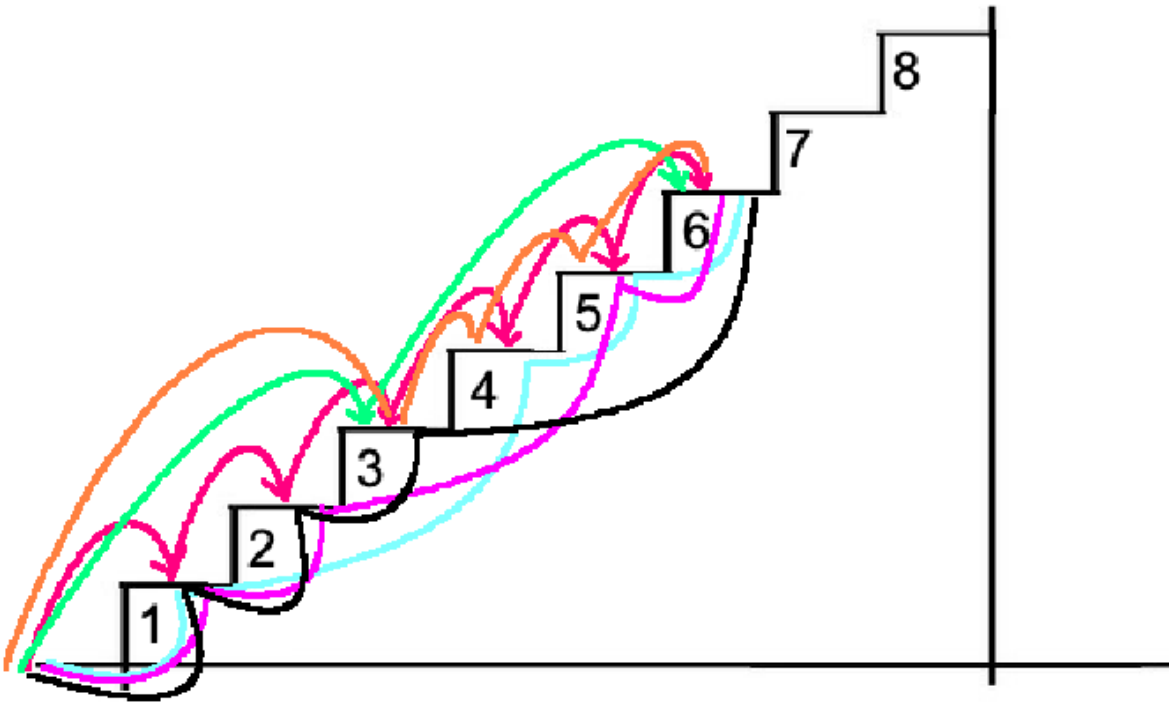
4 pav. Užduoties pavyzdys

$$S(3) = 2$$



5 pav. Užduoties pavyzdys

$$S(6) = 6$$



6 pav. Užduoties pavyzdys

Galime pastebėti, kad tai yra gana panašu į fibonači skaičių seką. Tačiau čia yra kiek kito seka, kurią galime išreikšti štai tokia lygybe:

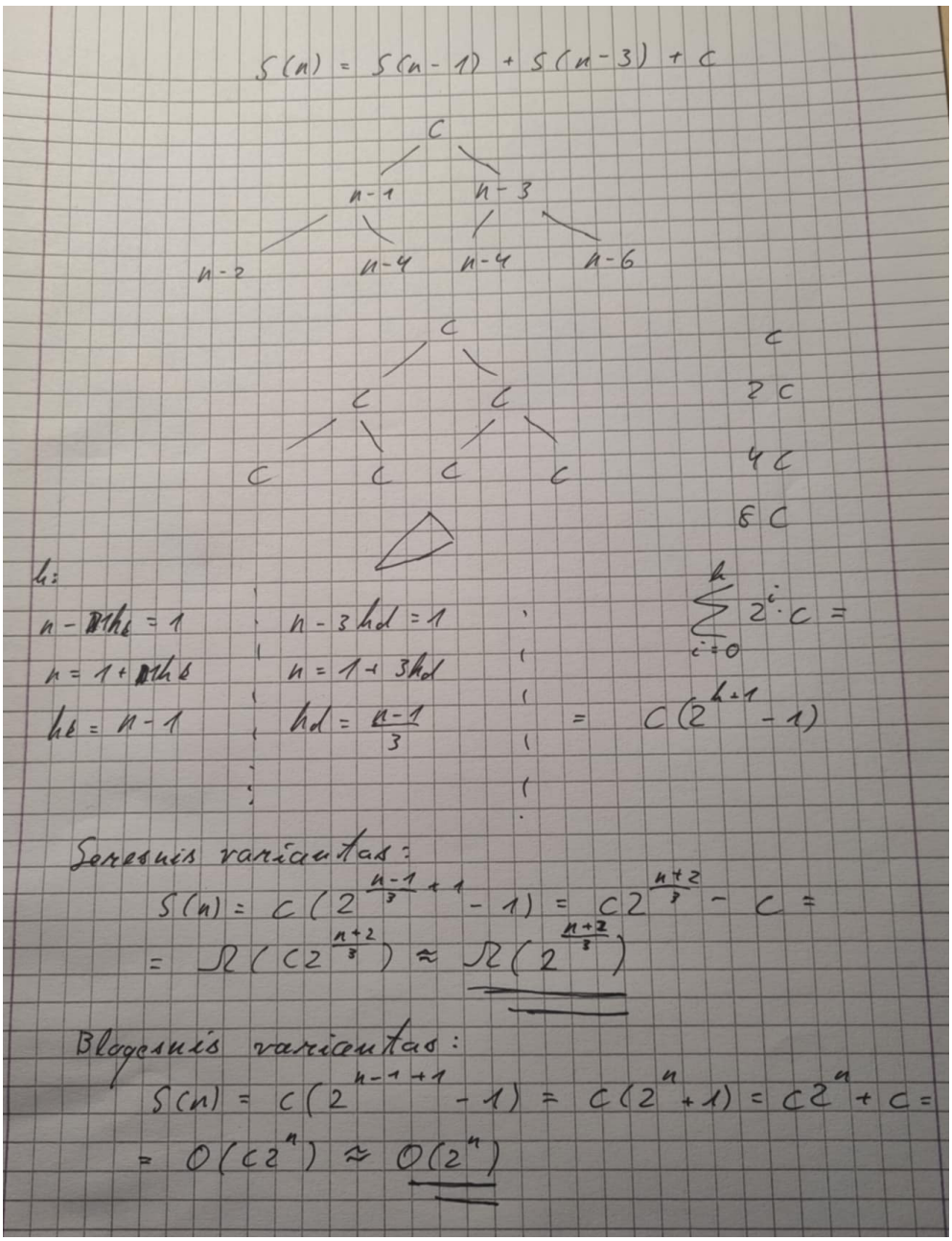
$$S(n) = S(n-1) + S(n-3)$$

Lygybės gavimas: jeigu ieškome, kaip žmogus gali patekti į n-tąjį skersinį, tai atsižvelgiame į tuos du variantus, kuriais tas žmogus gali vaikščioti tarp skersinių. Taigi, norint pasiekti n-tąjį skersinį yra du variantai, tai į jį ateiti iš prieš tai buvusio skersinio $(n-1)$ arba užšokti praleidus du skersinius $(n-3)$. Todėl visi galimi būdai pasiekti n-tąjį skersinį yra šių dviejų variantų suma, iš ko ir išsiveda anksčiau gauta lygybė.

Iš gautos lygybės seka:

$$S(0) = 1$$
$$S(1) = 1$$
$$S(2) = 1$$

$$S(3) = S(2) + S(0) = 1 + 1 = 2$$
$$S(4) = S(3) + S(1) = 2 + 1 = 3$$
$$S(5) = S(4) + S(2) = 3 + 1 = 4$$
$$S(6) = S(5) + S(3) = 4 + 2 = 6$$
$$\dots$$



7 pav. Sąryšio sprendimas

1.2. Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus

Sudaryta rekursinė procedūra:

```
1  static int SkaiciuotiBudus1(int n)
2      {
3          if (n < 0)
4          {
5              return 0;
6          }
7
8          if (n == 0)
9          {
10             return 1;
11         }
```

```
12
13          return SkaiciuotiBudus1(n - 1) + SkaiciuotiBudus1(n - 3);
14      }
```

1.3. Atlikti procedūros eksperimentinius tyrimus

1.3.1. Laiko priklausomybė nuo skersinių skaičiaus

1 lentelė Laiko priklausomybės nuo skersinių skaičiaus lentelė

Stairs, element count	Time, mikroseconds
0	533
1	601
2	696
4	710
8	750
16	783
22	1372
28	3293
32	12466

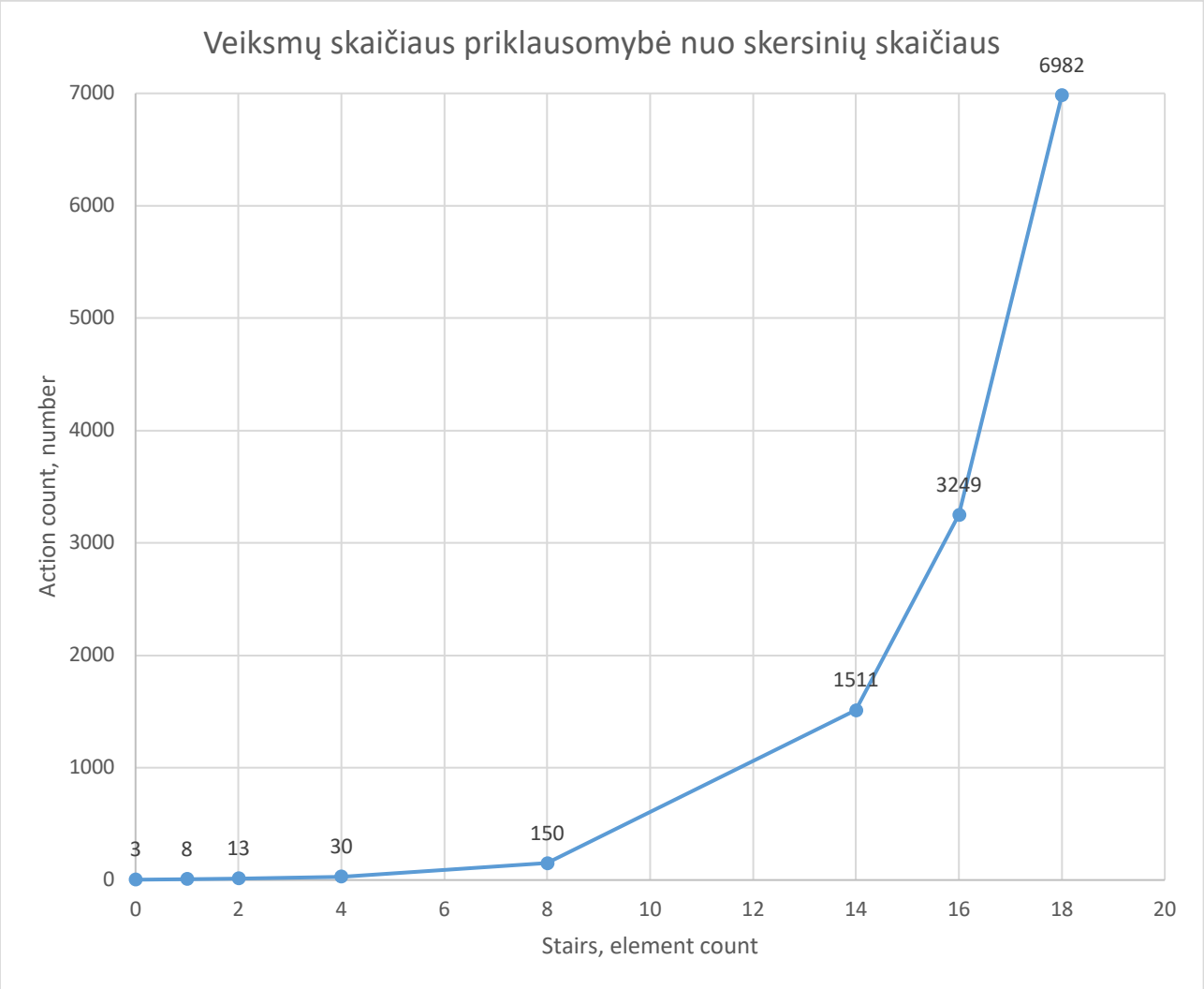
1 grafikas Laiko priklausomybės nuo skersinių skaičiaus grafikas

1.3.2. Veiksmų skaičiaus priklausomybė nuo skersinių skaičiaus

2 lentelė Veiksmų skaičiaus priklausomybės nuo skersinių skaičiaus lentelė

Stairs, element count	Action count, number
0	3
1	8
2	13
4	30
8	150
14	1511
16	3249
18	6982

2 grafikas Veiksmų skaičiaus priklausomybės nuo skersinių skaičiaus grafikas



1.4. Pritaikius dinaminio programavimo metodologiją rekursinei lygybei realizuoti procedūrą bei atlikti eksperimentinius našumo tyrimus

Realizuota procedūra:

```
1  static int SkaiciuotiBudus2(int n)
2      {
3          if (n < 0)
4          {
5              return 0;
6          }
7
8          int[] budai = new int[n + 1];
9          budai[0] = 1;
10
11         for (int i = 1; i <= n; i++)
12         {
13             budai[i] += budai[i - 1];
14
15             if (i >= 3)
16             {
17                 budai[i] += budai[i - 3];
18             }
19         }
20
21         return budai[n];
22     }
```

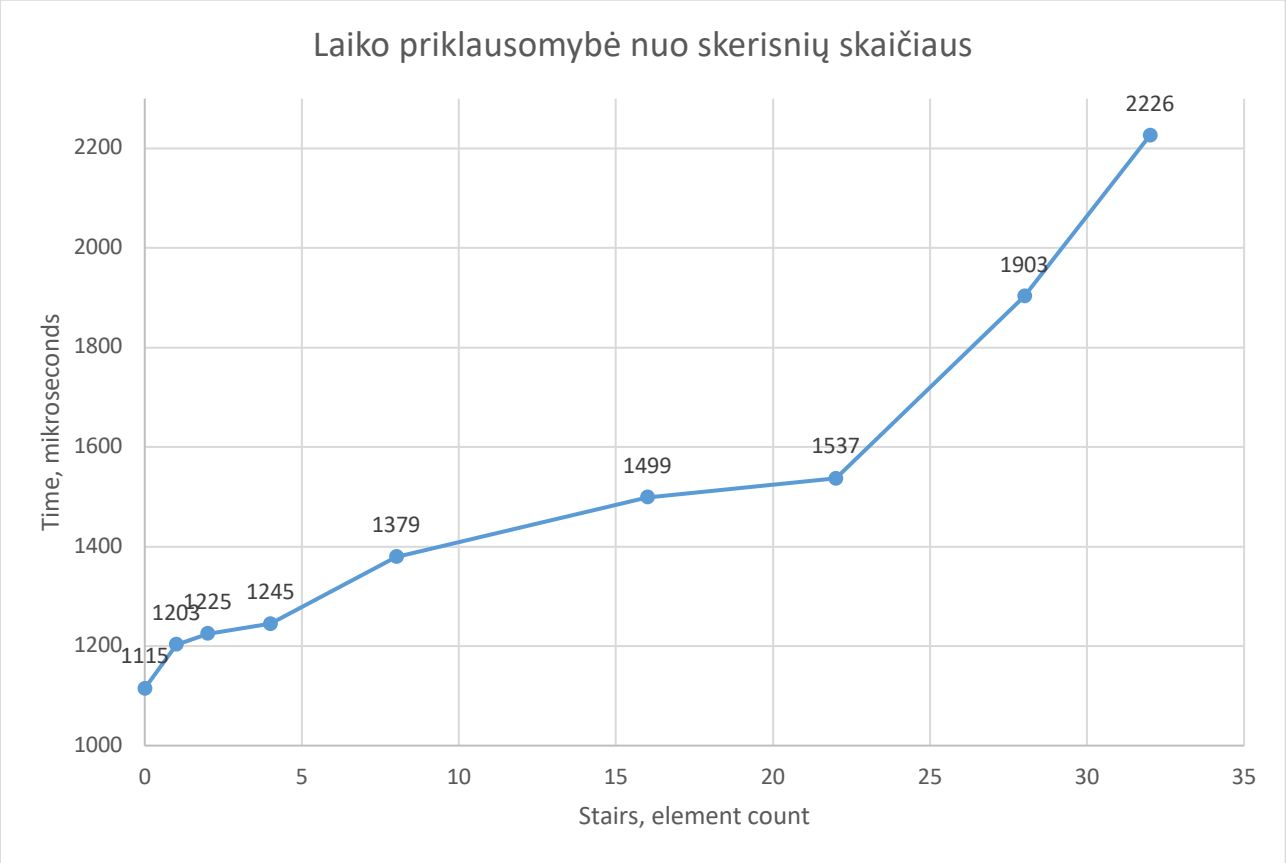
1.5. Atlikti procedūros eksperimentinius tyrimus:

1.5.1. Laiko priklausomybė nuo skersinių skaičiaus

3 lentelė Laiko priklausomybės nuo skersinių skaičiaus lentelė

Stairs, element count	Time, mikroseconds
0	1115
1	1203
2	1225
4	1245
8	1379
16	1499
22	1537
28	1903
32	2226

3 grafikas Laiko priklausomybės nuo skersinių skaičiaus grafikas

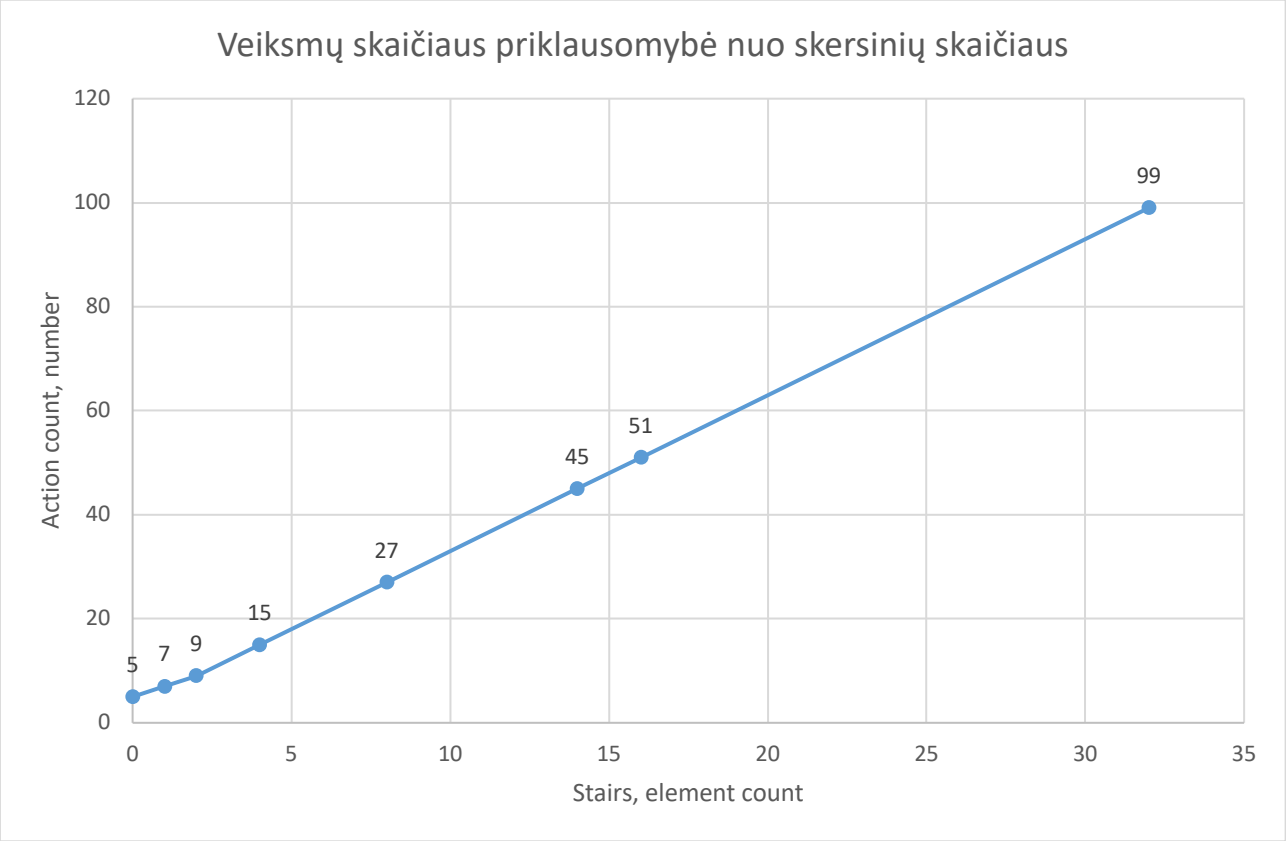


1.5.2. Veiksmų skaičiaus priklausomybė nuo skersinių skaičiaus

4 lentelė Veiksmų skaičiaus priklausomybės nuo skersinių skaičiaus lentelė

Stairs, element count	Action count, number
0	5
1	7
2	9
4	15
8	27
14	45
16	51
32	99

4 grafikas Veiksmų skaičiaus priklausomybės nuo skersinių skaičiaus grafikas



1.6. Atlikti realizuotų programinių kodų analizę

Programinio kodo analizė:

5 lentelė Rekursinės procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	`static int SkaiciuotiBudus1(int n)`		
2	`{`		
3	`if (n < 0)`	c_1	$1;\ X_1\{0,1\}$
4	`{`		
5	`return 0;`	c_2	X_1
6	`}`		
7			
8	`if (n == 0)`	c_3	$1 - X_1; X_2\{0,1\}$
9	`{`		
10	`return 1;`	c_4	$(1 - X_1)X_2$
11	`}`		
12			
13	`return SkaiciuotiBudus1(n - 1) +`	c_5	$(1 - X_1)(1 - X_2)S(n)$
14	`SkaiciuotiBudus1(n - 3);`		$S(n) = S(n-1) + S(n-3)$
15	`}`		

$$S_{SkaiciuotiBudus1}(n) = c_1 + c_2X_1 + c_3(1 - X_1) + c_4(1 - X_1)X_2 + c_5(1 - X_1)(1 - X_2) * \big(S(n-1) + S(n-3)\big)$$

Geriausias atvejis, kai: $X_1 = 1$

$$S_{SkaiciuotiBudus1}(n) = c_1 + c_2 = \Omega(C)$$

Blogiausias atvejis, kai: $X_1 = 0; X_2 = 0$

$$S_{SkaiciuotiBudus1}(n) = S(n-1) + S(n-3) + C = O(2^n)$$

130

131 6 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	`static int SkaiciuotiBudus2(int n)`		
2	`{`		
3	`if (n < 0)`	c_1	$1; X_1\{0,1\}$
4	`{`		
5	`return 0;`	c_2	X_1
6	`}`		
7			
8	`int[] budai = new int[n + 1];`	c_3	$1 - X_1$
9	`budai[0] = 1;`	c_4	$1 - X_1$
10			
11	`for (int i = 1; i <= n; i++)`	c_5	$(1 - X_1)(n - 1 + 2)$
12	`{`		
13	`budai[i] += budai[i - 1];`	c_6	$(1 - X_1)n$
14			
15	`if (i >= 3)`	c_7	$(1 - X_1)n; X_2\{0,1\}$
16	`{`		
17	`budai[i] += budai[i - 3];`	c_8	$(1 - X_1)X_2 n$
18	`}`		
19	`}`		
20			
21	`return budai[n];`	c_9	$1 - X_1$
22	`}`		

$$S_{SkaiciuotiBudus2}(n) = c_1 + c_2X_1 + (1 - X_1)c_3 + (1 - X_1)c_4 + (1 - X_1)(n + 1)c_5 + (1 - X_1)nc_6 + (1 - X_1)nc_7 + (1 - X_1)X_2nc_8 + (1 - X_1)c_9$$

Geriausias atvejis, kai: $X_1 = 1$

$$S_{SkaiciuotiBudus2}(n) = c_1 + c_2 = \Omega(C)$$

Blogiausias atvejis, kai: $X_1 = 0; X_2 = 1$

$$S_{SkaiciuotiBudus2}(n) = (n + 1) + 3n + C = O(3n) = O(n)$$

132

133 Pagal apskaičiuotus asimptotinius įverčius ir gautus eksperimentinius rezultatus matyti, kad gauti
134 rezultatai atitinką numatytuosius. Taigi, iš to galima teigti, kad viskas buvo atlikta teisingai.
135

2. Antros užduoties dalys

2.1. Pateikti uždavinio sprendimo algoritmą

Tarkime turimeS_n grupių žmonių laukiančių persikelti į kitą pusę. Stovintys m keltai prieplaukoje gali perkelti keleivius, tačiau kiekvienas jų gali pervežti skirtingą kiekį žmonių. Kaip sutalpinti į keltus žmonių grupes, kad pervežamas keleivių kiekis kiekviename kelte būtų kuo didesnis. Žmonių grupių skaidyti negalima.

8 pav. Uždavinys

Pagrinde šią užduotį traktuosime taip: yra kelio žmonių grupės, kurių negalima išskaidyti ir jos nori persikelti per upę. Taip pat yra keltai, į kuriuos telpa kažkiek žmonių. Taigi, turime keltus ir jų talpą bei žmonių grupes su skirtingų kiekių žmonių. Ašinis šio uždavinio bus tai, kad keltai atplaukia vienas po kito ir į juos iš eilės yra laipinamos grupės, kurios telpa į tą keltą. Taip kartojama kol nebelieka keltų arba nebelieka žmonių grupių, kurios nori persikelti per upę.

Uždavinio sprendimas:
Tai optimizavimo uždavinys – norime maksimaliai išnaudoti kiekvieno kelto talpą. Sprendimas – simuliuoti keltų atplaukimą ir kiekvienam jų priskirti kuo daugiau iš eilės stovinčių grupių, kol jų bendra suma neviršys kelto talpos.

Uždavinio rekursinis sąryšis:

$$dp(i,j) = \max\{dp(k,j-1) + sum(k+1,i) \mid sum(k+1,i) \leq capacity[j]\}$$

Čia:
$dp(i,j) - tai\ maksimalus$ žmonių $skaičius, kuriuos\ galime\ perkelti, kai$:
n grupių žmonių, kurių skaičiai yra masyve: $groups[i]$ – žmonių kiekis i-tojoje grupėje
m keltų, kurių talpos yra masyve: $capacity[j]$ – j-to kelto maksimali talpa

$i - pirmos$ žmonių $grupės$
$j - pirmi\ keltai$
$k - paskutinės\ grupės\ indeksas, kuri\ buvo$ $įtraukta$ į $ankstesnį\ keltą$
$sum(k+1,i)$ – tai žmonių skaičius nuo grupės k+1 iki i, kurias norime įdėti į dabartinį keltą

Uždavinio sąryšio paaiškinimas:
Žiūrime į i pirmų žmonių grupių. Norime paskutiniam keltui j priskirti kiek galima daugiau grupių, kurios telpa į jo talpą. Tada reikia pažiūrėti kiek maksimaliai buvome perplukdę iki k grupės su $j-1$ keltais. Prie viso to reikia pridėti, kiek žmonių galima įkelti į naują keltą. Jei suma yra mažesnė arba lygi kelto talpai, tai yra galimas sprendimas – išsaugome maksimumą.

Rekursinio sąryšio įrodymas:
Sakykime, kad turime tokias grupes:

$$groups = [2,3,5,2,4,3]$$

Ir turime tokias keltų talpas:

$$capacity = [5,6,7]$$

Pradiniai atvejai:

$$dp(0,0) = 0\ (nėra\ grupių, nėra\ keltų)$$
$$dp(i,0) = 0\ (negalima\ perkelti\ nieko\ be\ keltų)$$

$$dp(0, j) = 0\ (nėra\ grupių, nėra\ ką\ perkelti)$$

Pavyzdžio sprendimas:
(Laikysime, kad masyvo indeksas prasideda ne nuo 0, o nuo 1)

$$dp(2,1) = 0 + 2 + 3 = 5$$

$$dp(3, 2) = dp(2, 1) + 5 = 5 + 5 = 10$$

$$dp(5, 3) = dp(3, 2) + 2 + 4 = 10 + 6 = 16$$

Dabar pabandysime išspręsti šią užduotį šiek tiek sunkesniu variantu: kai keltai neatplaukia vienas po kito ir žmonių grupės nėra laipinamos viena po kitos, o tiesiogiai bandoma iš karto rasti pačius optimaliausius sprendimus kaip perkelti kuo daugiau žmonių grupių per upę.

Pasunkinto uždavinio pavyzdys:
Sakykime, kad turime tokias grupes:

$$groups = [2, 3, 5, 2, 4, 3]$$

Ir turime tokias keltų talpas:

$$capacity = [5, 6, 7]$$

Tada šį uždavinį reikėtų išspręsti taip:
Į keltą su talpa 5 reikėtų įlaipinti grupę, kurios žmonių kiekis yra 5.
Tada į keltą su talpa 6 reikėtų įlaipinti grupes, kurių žmonių kiekiai yra 2 ir 4.

Atlikus šiuos veiksmus jau turėsime likusius tokius variantus:

$$groups = [2, 3, 3]$$
$$capacity = [7]$$

Tuomet į likusį keltą galime įlaipinti grupes, kurių žmonių kiekiai yra 3 ir 3.

Taigi, kai sulaipinome taip grupes į keltus, rezultate matosi, kad pavyksta šiuo atveju maksimaliai perkelti 17 žmonių.

2.2. Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus

Sudaryta rekursinė procedūra:

```
1  static int Solve(int i, int j)
2      {
3          if (i >= groups.Length || j >= capacity.Length)
4          {
5              return 0;
6          }
7
8          if (dp[i, j] != -1)
9          {
10             return dp[i, j];
11         }
12
13         int maxPeople = 0;
14         int sum = 0;
15
```

```
        for (int k = i; k < groups.Length; k++)
        {
            sum += groups[k];

            if (sum > capacity[j])
            {
                break;
            }

            int result = Solve(k + 1, j + 1);

            maxPeople = Math.Max(maxPeople, sum + result);
        }

        dp[i, j] = maxPeople;

        return maxPeople;
    }
```

Pasunkintos užduoties sudaryta rekursinė procedūra:

```
static int Dp(int usedMask, int[] capacities)
    {
        string key = usedMask + "|" + string.Join(",", capacities);
        if (memo.ContainsKey(key))
        {
            return memo[key];
        }

        int maxValue = 0;

        for (int i = 0; i < groups.Length; i++)
        {
            if ((usedMask & (1 << i)) != 0)
            {
                continue;
            }

            for (int j = 0; j < capacities.Length; j++)
            {
                if (groups[i] <= capacities[j])
                {
                    int[] newCaps = new int[capacities.Length];
                    Array.Copy(capacities, newCaps, capacities.Length);
                    newCaps[j] -= groups[i];

                    int newMask = usedMask | (1 << i);
                    int value = groups[i] + Dp(newMask, newCaps);

                    if (value > maxValue)
                    {
                        maxValue = value;
                    }
                }
            }
        }

        memo[key] = maxValue;
```

```
38
39          return maxValue;
40      }
```

2.3. Atlikti procedūros eksperimentinius tyrimus

2.3.1. Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus

7 lentelė Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Time, mikroseconds
5	10	2	20	583
10	10	3	25	791
20	10	5	30	1792
50	10	6	40	2833
100	10	10	50	17833
200	10	15	60	32583
500	10	20	70	112834
1000	10	30	100	398542

5 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

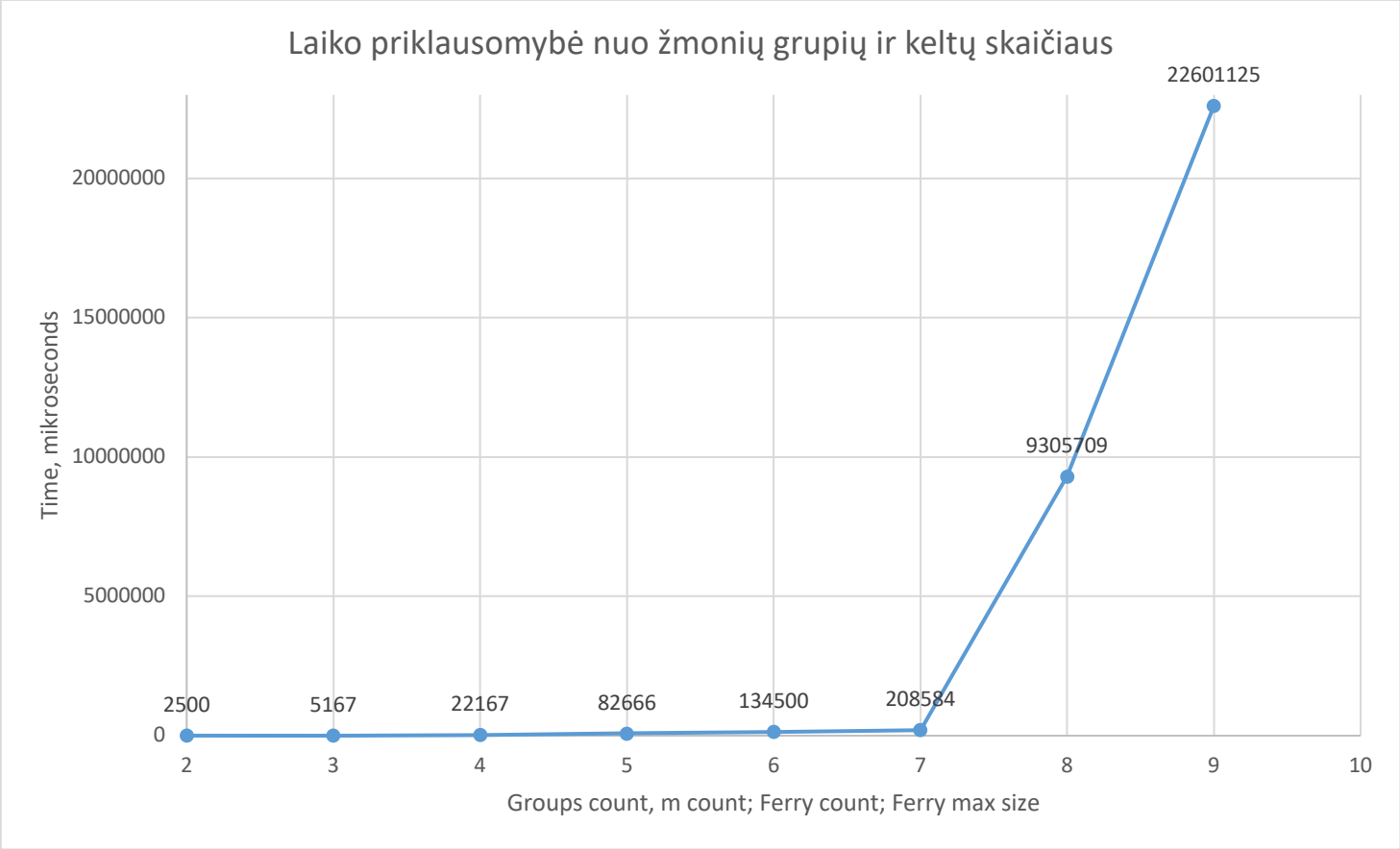
227 8 lentelė Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Time, mikroseconds
2	4	1	5	2500
3	4	1	5	5167
4	4	2	6	22167
5	4	2	6	82666
6	4	2	7	134500
7	4	2	7	208584
8	4	3	8	9305709
9	4	3	8	22601125

228

6 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

229



230 **2.3.2. Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus**

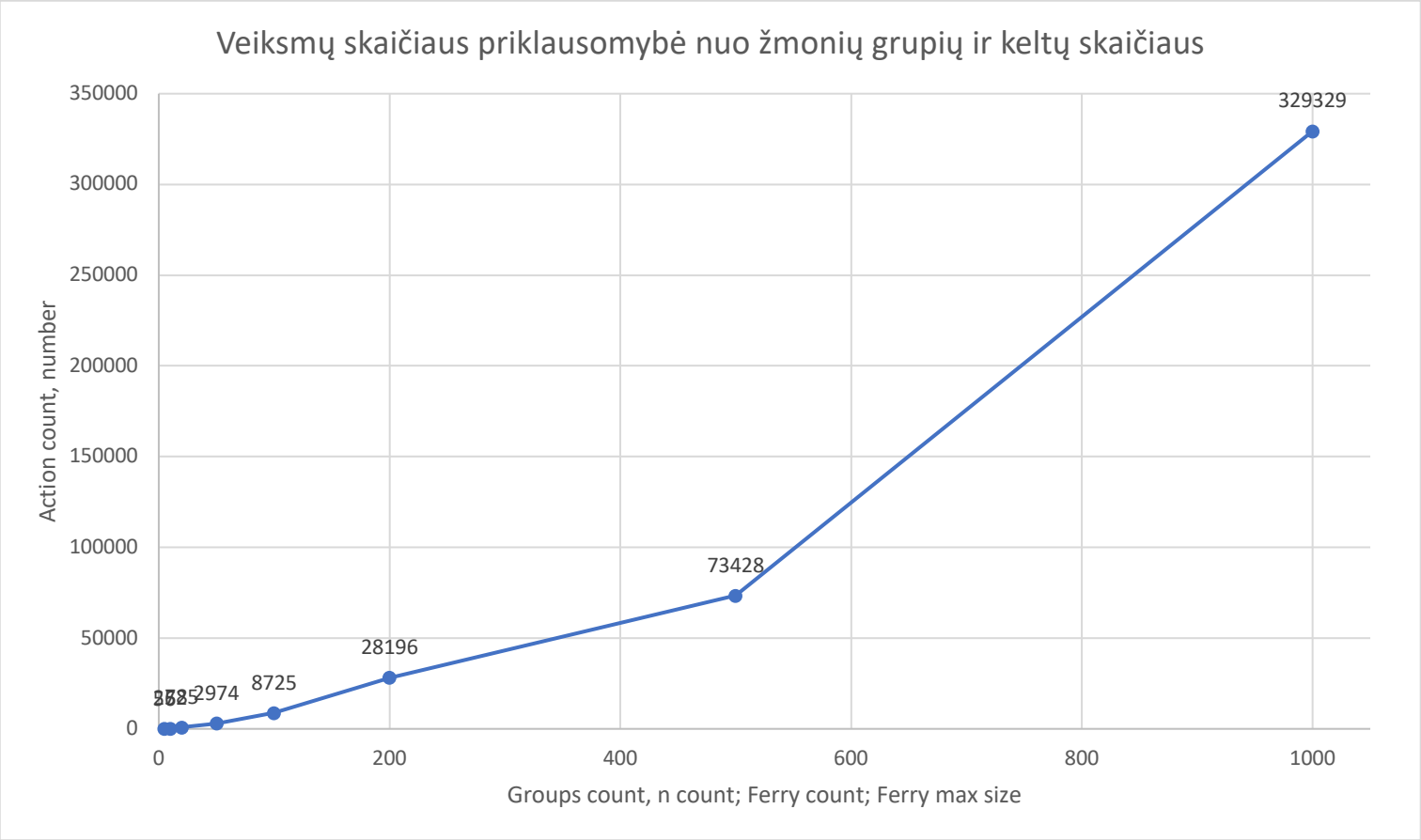
231 9 lentelė Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Action count, number
5	10	2	20	56
10	10	3	25	252

20	10	5	30	785
50	10	6	40	2974
100	10	10	50	8725
200	10	15	60	28196
500	10	20	70	73428
1000	10	30	100	329329

232

7 grafikas Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

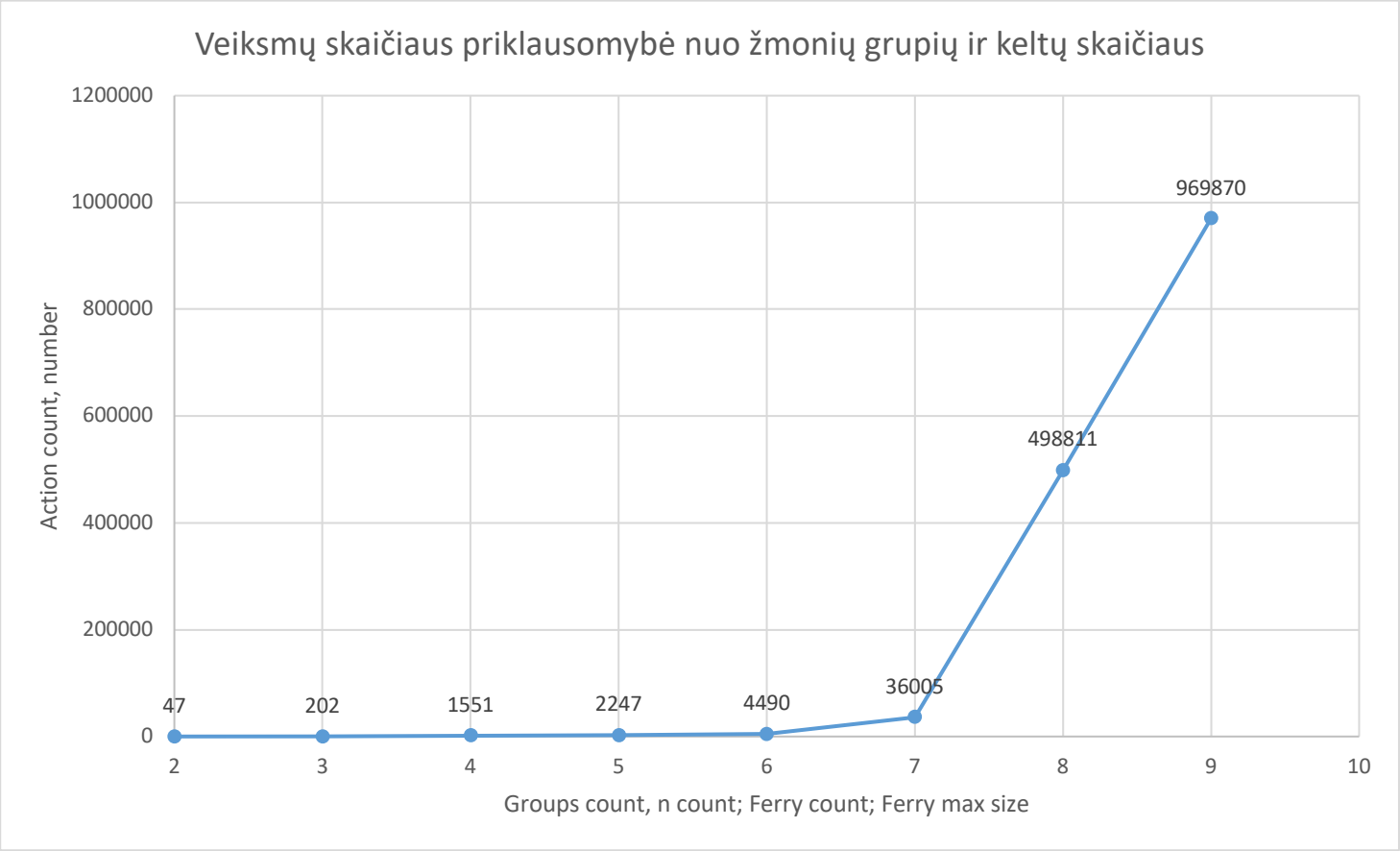


233

234 10 lentelė Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Action count, number
2	4	1	5	47
3	4	1	5	202
4	4	2	6	1551
5	4	2	6	2247
6	4	2	7	4490
7	4	2	7	36005
8	4	3	8	498811
9	4	3	8	969870

8 grafikas Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas



2.4. Pritaikius dinaminio programavimo metodologiją rekursinei lygybei realizuoti procedūrą bei atlikti eksperimentinius našumo tyrimus

Realizuota procedūra:

```
1  static int SolveDP()
2      {
3          int n = groups.Length;
4          int m = capacity.Length;
5          dp[0, 0] = 0;
6
7          for (int j = 0; j < m; j++)
8          {
9              for (int i = 0; i <= n; i++)
10             {
11                 if (dp[i, j] == -1)
12                 {
13                     continue;
14                 }
15
16                 int sum = 0;
17
18                 for (int k = i; k < n; k++)
19                 {
20                     sum += groups[k];
```

```
21
22                      if (sum > capacity[j])
23                      {
24                          break;
25                      }
26
27                      dp[k + 1, j + 1] = Math.Max(dp[k + 1, j + 1], dp[i, j] + sum);
28                  }
29              }
30          }
31
32          int max = 0;
33
34          for (int i = 0; i <= n; i++)
35          {
36              max = Math.Max(max, dp[i, m]);
37          }
38
39          return max;
40      }
```

241

242 Pasunkintos užduoties realizuota procedūra:

243

```
1  static int IterativeDP()
2      {
3          var dp = new Dictionary<string, int>();
4          var queue = new Queue<(int usedMask, int[] caps)>();
5
6          int[] startCaps = (int[])ferries.Clone();
7          string startKey = "0|" + string.Join(",", startCaps);
8          dp[startKey] = 0;
9          queue.Enqueue((0, startCaps));
10
11         int maxPeople = 0;
12
13         while (queue.Count > 0)
14         {
15             var (usedMask, caps) = queue.Dequeue();
16             string stateKey = usedMask + "|" + string.Join(",", caps);
17             int currentValue = dp[stateKey];
18
19             maxPeople = Math.Max(maxPeople, currentValue);
20
21             for (int i = 0; i < groups.Length; i++)
22             {
23                 if ((usedMask & (1 << i)) != 0)
24                     continue;
25
26                 for (int j = 0; j < caps.Length; j++)
27                 {
28                     if (groups[i] > caps[j])
29                         continue;
30
31                     int[] newCaps = (int[])caps.Clone();
32                     newCaps[j] -= groups[i];
33                     int newMask = usedMask | (1 << i);
34                     string newKey = newMask + "|" + string.Join(",", newCaps);
35                     int newValue = currentValue + groups[i];
```

```
36
37                      if (!dp.ContainsKey(newKey) || newValue > dp[newKey])
38                      {
39                          dp[newKey] = newValue;
40                          queue.Enqueue((newMask, newCaps));
41                      }
42                  }
43              }
44          }
45
46          return maxPeople;
47      }
```

244

245 ## 2.5. Atlikti procedūros eksperimentinius tyrimus

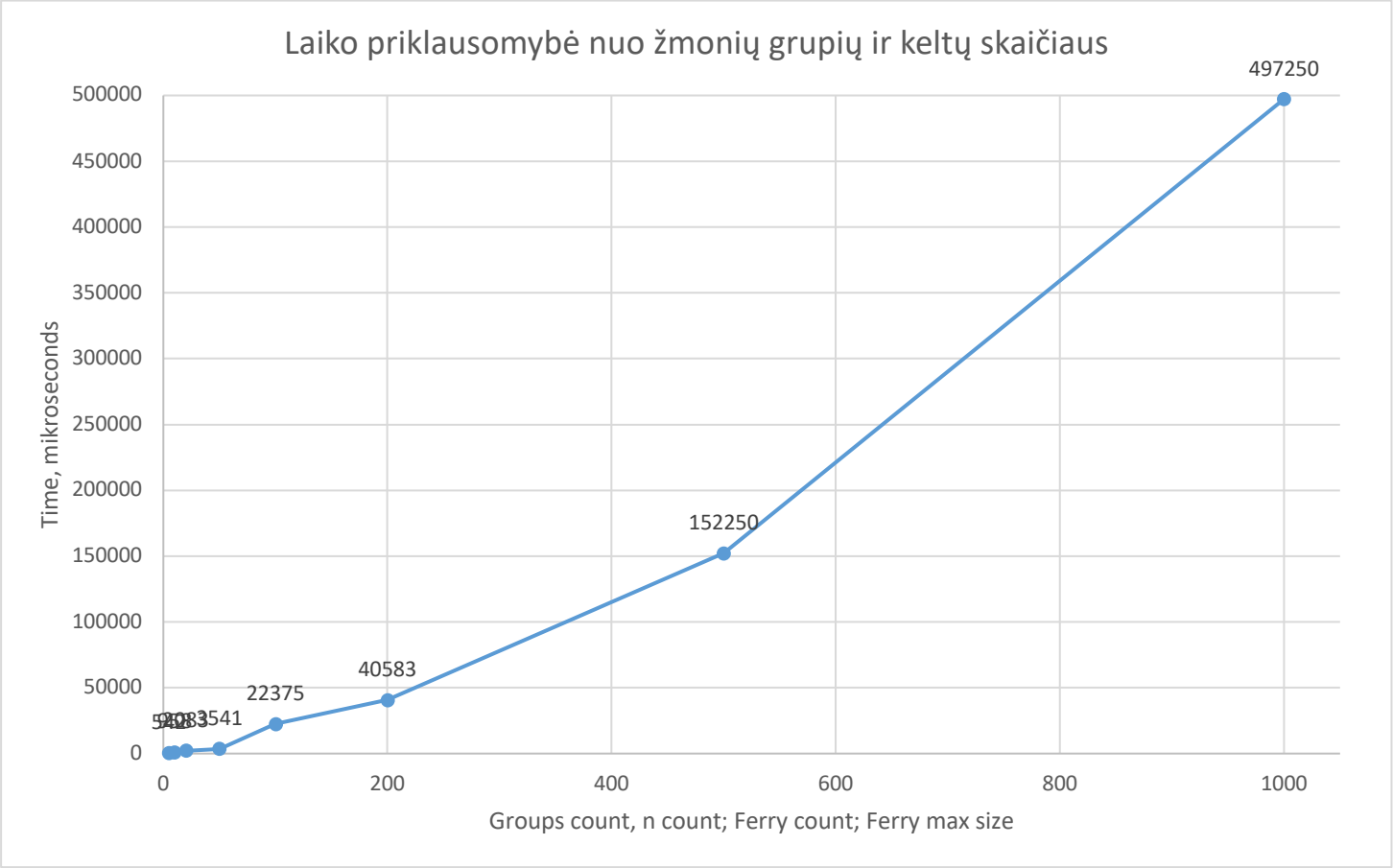
246 ### 2.5.1. Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus

247 11 lentelė Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Time, mikroseconds
5	10	2	20	542
10	10	3	25	958
20	10	5	30	2083
50	10	6	40	3541
100	10	10	50	22375
200	10	15	60	40583
500	10	20	70	152250
1000	10	30	100	497250

248

9 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas



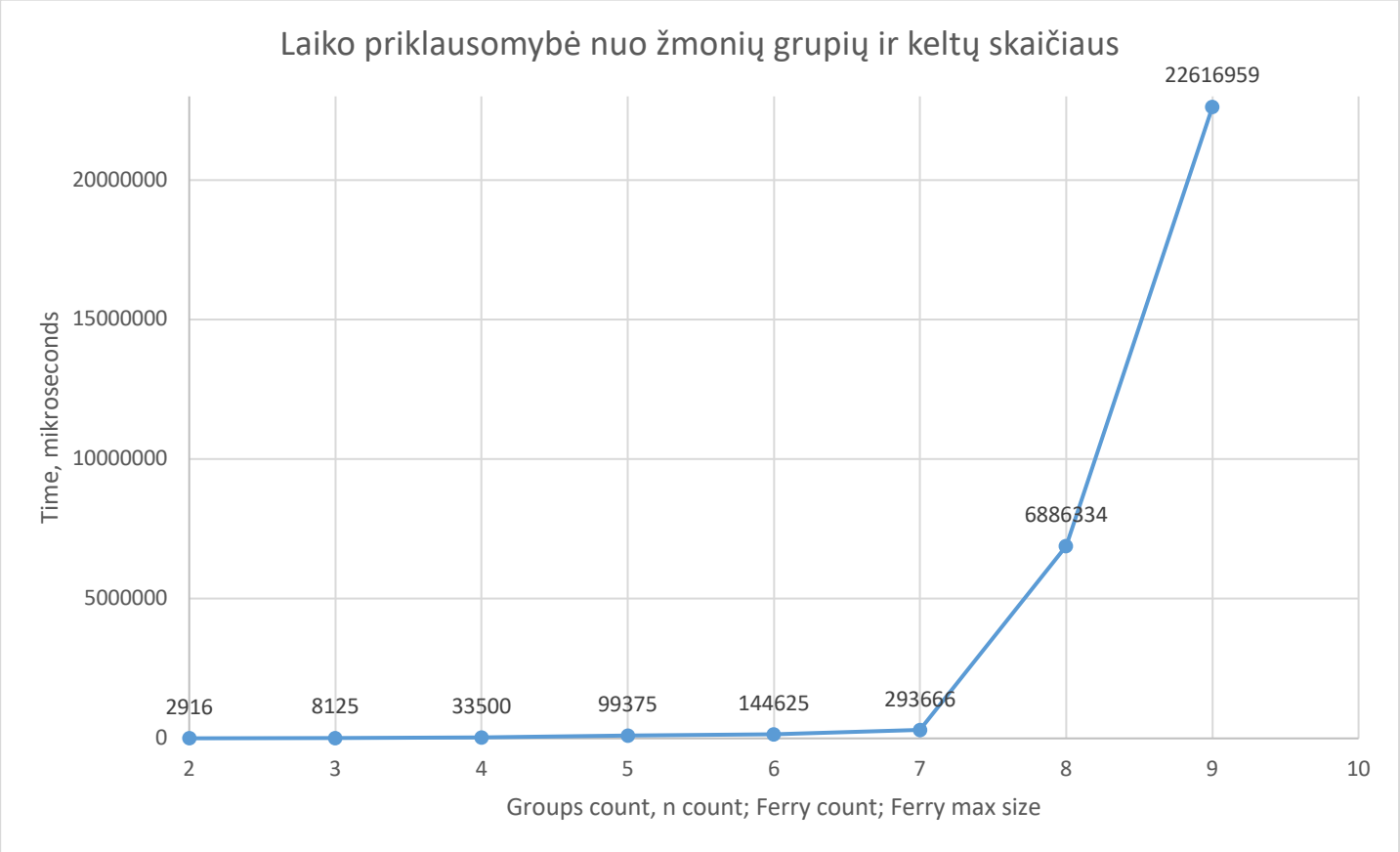
249

250 12 lentelė Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Time, mikroseconds
2	4	1	5	2916
3	4	1	5	8125
4	4	2	6	33500
5	4	2	6	99375
6	4	2	7	144625
7	4	2	7	293666
8	4	3	8	6886334
9	4	3	8	22616959

251

10 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas



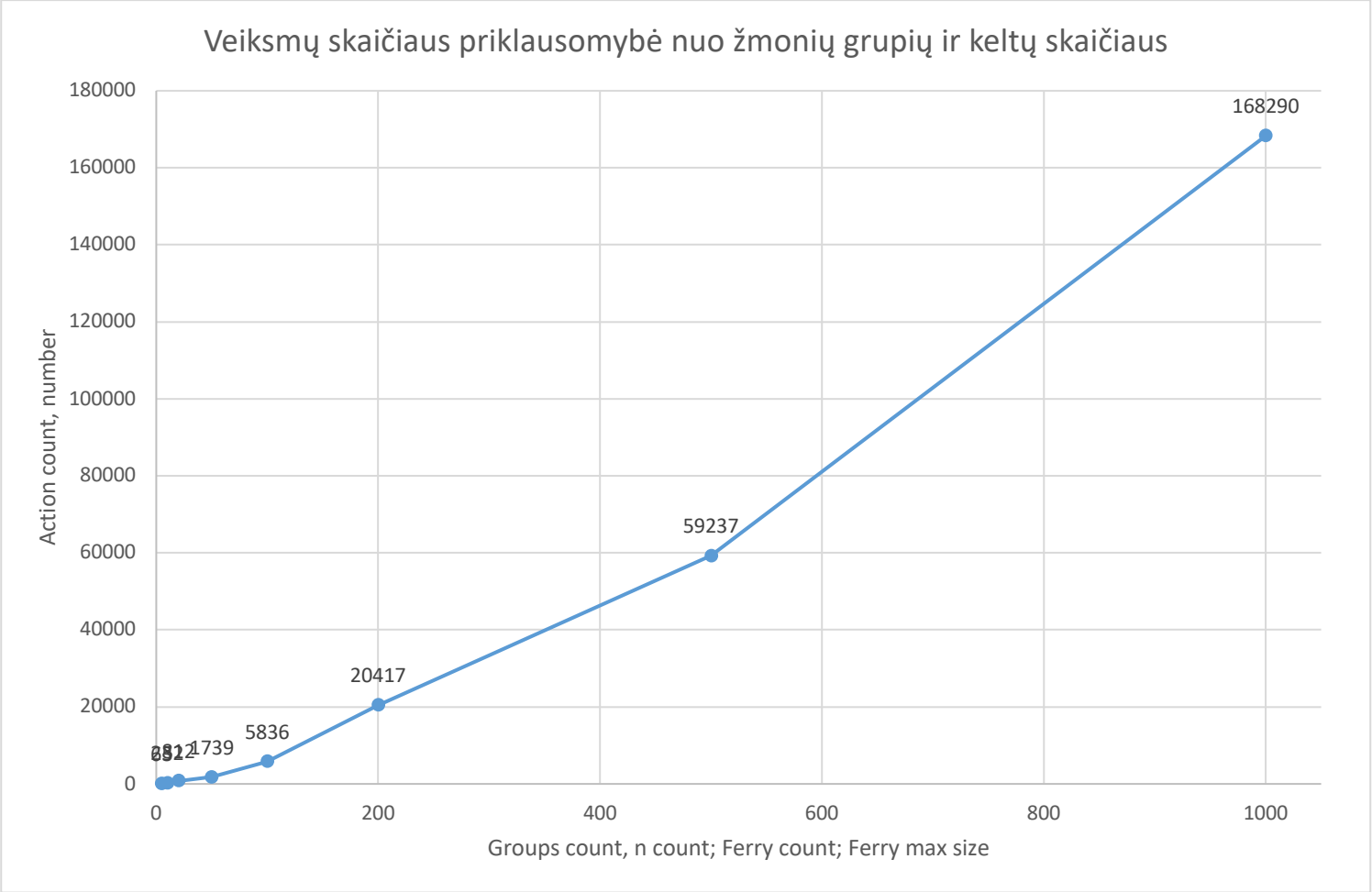
252

253 **2.5.2. Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus**

254 13 lentelė Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Action count, number
5	10	2	20	63
10	10	3	25	252
20	10	5	30	812
50	10	6	40	1739
100	10	10	50	5836
200	10	15	60	20417
500	10	20	70	59237
1000	10	30	100	168290

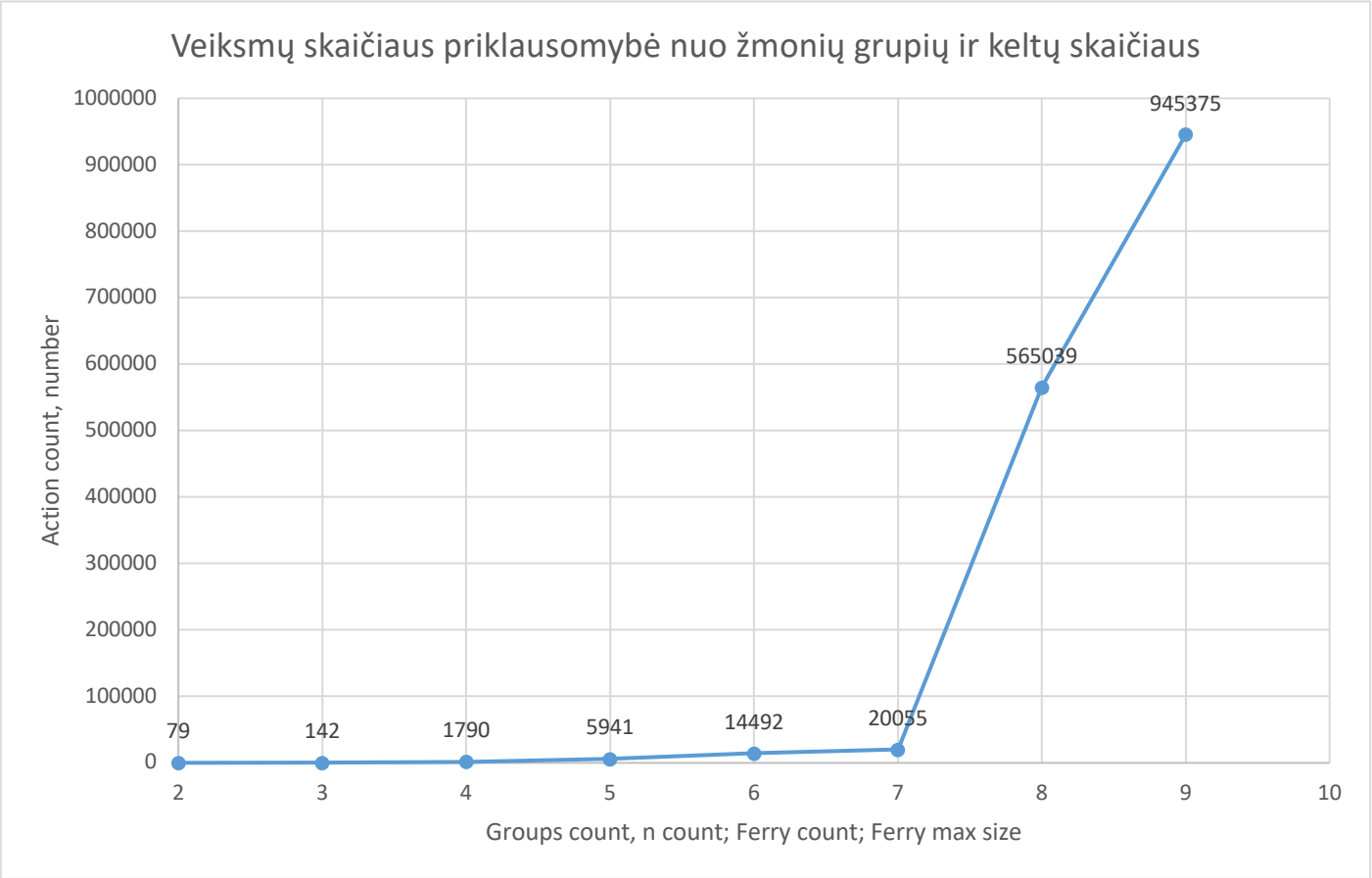
11 grafikas Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas



14 lentelė Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

Groups count, n count	Group max size, number	Ferry count, m count	Ferry max size, number	Action count, number
2	4	1	5	79
3	4	1	5	142
4	4	2	6	1790
5	4	2	6	5941
6	4	2	7	14492
7	4	2	7	20055
8	4	3	8	565039
9	4	3	8	945375

12 grafikas Veiksmų skaičiaus priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas



2.6. Atlikti realizuotų programinių kodų žmonių grupių ir keltų skaičiaus

Programinio kodo analizė:

15 lentelė Rekursinės procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	`static int Solve(int i, int j)`		
2	`    {`		
3	`        if (i >= groups.Length ||`	c_1	$1; X_1\{0,1\}$
4	`            j >= capacity.Length)`		
5	`        {`		
6	`            return 0;`	c_2	X_1
7	`        }`		
8			
9	`        if (dp[i, j] != -1)`	c_3	$1; X_2\{0,1\}$
10	`        {`		
11	`            return dp[i, j];`	c_4	X_2
12	`        }`		
13			
14	`        int maxPeople = 0;`	c_5	1
15	`        int sum = 0;`	c_5	1
16			
17	`        for (int k = i; k < groups.Length; k++)`	c_6	$n-i+1$
18	`        {`		
19	`            sum += groups[k];`	c_7	$n-i$
20			

21	`if (sum > capacity[j])`	c_8	$n-i;\ X_3\{0,1\}$
22	`{`		
23	`break;`	c_9	$(n-i)X_3$
24	`}`		
25			
26	`int result = Solve(k + 1, j + 1);`	T(k+1,j+1)	$(n-i)T(k+1,j+1)$
27			
28	`maxPeople = Math.Max(maxPeople,`	c_{10}	$n-i$
29	`sum + result);`		
30	`}`		
31			
32	`dp[i, j] = maxPeople;`	c_{11}	1
33			
34	`return maxPeople;`	c_{12}	1
35	`}`		

$$S_{Solve}(i,j) = c_1 + c_2X_1 + c_3 + c_4X_2 + 2c_5 + c_6(n-i+1) + c_7(n-i) + c_8(n-i) + c_9(n-i)X_3 + (n-i)T(k+1,j+1) + c_{10}(n-i) + c_{11} + c_{12}$$

Geriausias atvejis, kai: $X_1 = 1$

$$S_{Solve}(i,j) = c_1 + c_2 = \Omega(C)$$

Blogiausias atvejis, kai: $X_1 = 0; X_2 = 0; X_3 = 0$

$$S_{Solve}(i,j) = c_1 + c_3 + 2c_5 + (n-i)c_6 + c_6 + c_7(n-i) + c_8(n-i) + (n-i)T(k+1,j+1) + c_{10}(n-i) + c_{11} + c_{12} = C + (n-i)(c_6 + c_7 + c_8 + c_{10}) + (n-i)T(k+1,j+1) = O(\ \ )$$

263

264 16 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	`static int SolveDP()`		
2	`{`		
3	`int n = groups.Length;`	c_1	1
4	`int m = capacity.Length;`	c_1	1
5	`dp[0, 0] = 0;`	c_2	1
6			
7	`for (int j = 0; j < m; j++)`	c_3	$m-0+1 = m+1$
8	`{`		
9	`for (int i = 0; i <= n; i++)`	c_4	$m(n+2)$
10	`{`		
11	`if (dp[i, j] == -1)`	c_5	$m(n+1); X_1\{0,1\}$
12	`{`		
13	`continue;`	c_6	$m(n+1)X_1$
14	`}`		
15			
16	`int sum = 0;`	c_7	$m(n+1)$
17			
18	`for (int k = i; k < n; k++)`	c_8	$m(n+1)(n-i+1)$
19	`{`		
20	`sum += groups[k];`	c_9	$m(n+1)(n-i)$
21			
22	`if (sum > capacity[j])`	c_{10}	$m(n+1)(n-i);\ X_2\{0,1\}$
23	`{`		
24	`break;`	c_{11}	$m(n+1)(n-i)X_2$
25	`}`		
26			

27	dp[k + 1, j + 1] =	c_{12}	$m(n+1)(n-i)$
28	Math.Max(		
29	dp[k + 1, j + 1],		
30	dp[i, j] + sum		
31	);		
32	}		
33	}		
34	}		
35			
36	int max = 0;	c_{13}	1
37			
38	for (int i = 0; i <= n; i++)	c_{14}	$n+2$
39	{		
40	max = Math.Max(max, dp[i, m]);	c_{15}	$n+1$
41	}		
42			
43	return max;	c_{16}	1
44	}		

$$S_{SolveDP}(\ ) = 2c_1 + c_2 + (m+1)c_3 + \big(m(n+2)\big)c_4 + \big(m(n+1)\big)c_5 + (m(n+1)X_1)c_6 + \big(m(n+1)\big)c_7 + \big(m(n+1)(n-i+1)\big)c_8 + \big(m(n+1)(n-i)\big)c_9 + \big(m(n+1)(n-i)\big)c_{10} + (m(n+1)(n-i)X_2)c_{11} + \big(m(n+1)(n-i)\big)c_{12} + c_{13} + (n+2)c_{14} + (n+1)c_{15} + c_{16}$$

Geriausias atvejis, kai: $X_1 = 1$

$$S_{SolveDP}(\ ) = 2c_1 + c_2 + (m+1)c_3 + \big(m(n+2)\big)c_4 + \big(m(n+1)\big)c_5 + \big(m(n+1)\big)c_6 + c_{13} + (n+2)c_{14} + (n+1)c_{15} + c_{16}$$
$$= 2c_1 + c_2 + c_3 + c_{13} + 2c_{14} + c_{15} + c_{16} + mc_3 + mnc_4 + 2mc_4 + mnc_5 + mc_5 + mnc_6 + mc_6 + nc_{14} + nc_{15}$$
$$= C + mn(c_4 + c_5 + c_6) + m(c_3 + 2c_4 + c_5 + c_6) + n(c_{14} + c_{15}) = C + mn + m + n = \Omega(mn)$$

Blogiausias atvejis, kai: $X_1 = 0; X_2 = 0$

$$S_{SolveDP}(\ ) = C + mc_3 + mnc_4 + 2mc_4 + mnc_5 + mc_5 + mnc_7 + mc_7 + \big((mn+m)(n+1)\big)c_8 + \big(mn(n+1)\big)c_9 + \big(mn(n+1)\big)c_{10} + \big(mn(n+1)\big)c_{12} + nc_{14} + nc_{15}$$
$$= C + mn(c_4 + c_5 + c_7 + 2c_8 + c_9 + c_{10} + c_{12}) + mn^2(c_8 + c_9 + c_{10} + c_{12}) + m(c_3 + 2c_4 + c_5 + c_7 + c_8) + n(c_{14} + c_{15}) = C + mn + mn^2 + m + n = O(mn^2)$$

Gauti asimptotiniai sudėtingumai bei našumo testavimai parodo panašius rezultatus. Taigi galima teigti, kad buvo apskaičiuota teisingai.

Pasunkintos užduoties programinio kodo analizė:

17 lentelė Rekursinės procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	static int Dp(int usedMask, int[] capacities)		
2	{		
3	string key = usedMask + "|"	c_1	1
4	+ string.Join(",", capacities);		
5	if (memo.ContainsKey(key))	c_2	$1; X_1\{0,1\}$
6	{		
7	return memo[key];	c_3	X_1
8	}		
9			
10	int maxValue = 0;	c_4	1
11			
12	for (int i = 0; i < groups.Length; i++)	c_5	$groups.Length + 1$

13	{		
14	if ((usedMask & (1 << i)) != 0)	c_6	$groups.Length; X_2\{0,1\}$
15	{		
16	continue;	c_7	$(groups.Length)X_2$
17	}		
18			
19	for (int j = 0; j <		
20	capacities.Length; j++)	c_8	$(groups.Length)(capacities.Length + 1)$
21	{		
22	if (groups[i] <= capacities[j])		
23	{	c_9	$(groups.Length)(capacities.Length); X_2\{0,1\}$
24	int[] newCaps		
25	= new	c_{10}	$(groups.Length) * (capacities.Length) * X_2$
26	int[capacities.Length];		
27	Array.Copy(		
28	capacities,	c_{11}	$(groups.Length) * (capacities.Length) * X_2$
29	newCaps,		
30	capacities.Length);		
31	newCaps[j] -= groups[i];		
32		c_{12}	$(groups.Length) * (capacities.Length) * X_2$
33	int newMask		
34	= usedMask | (1 << i);	c_{13}	$(groups.Length) * (capacities.Length) * X_2$
35	int value = groups[i]		
36	+ Dp(newMask, newCaps);	T(M,C)	$(groups.Length) * (capacities.Length) * X_2 + T(M,C)$
37			
38	if (value > maxValue)		
39	{	c_{14}	$(groups.Length) * (capacities.Length) * X_2; X_3\{0,1\}$
40	maxValue = value;		
41	}	c_{15}	$(groups.Length) * (capacities.Length) * X_2 * X_3$
42	}		
43	}		
44	}		
45			
46	memo[key] = maxValue;		
47	return maxValue;	c_{16}	1
48	}	c_{17}	1

271

272 18 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

Nr.	Kodas	Laikas	Kartai
1	static int IterativeDP()		
2	{		
3	var dp = new Dictionary<string, int>();	c_1	1
4	var queue = new Queue<(	c_2	1
5	int usedMask, int[] caps)>();		
6			
7	int[] startCaps =	c_3	1
8	(int[])ferries.Clone();		
9	string startKey = "0|"	c_4	1
10	+ string.Join(",", startCaps);		
11	dp[startKey] = 0;	c_5	1
12	queue.Enqueue((0, startCaps));	c_6	1
13			
14	int maxPeople = 0;	c_7	1
15			
16	while (queue.Count > 0)	c_8	$queue.Count + 1$
17	{		

18	var (usedMask, caps)	c_9	$queue.Count$
19	= queue.Dequeue();		
20	string stateKey = usedMask + "|"	c_9	$queue.Count$
21	+ string.Join(",", caps);		
22	int currentValue = dp[stateKey];	c_{10}	$queue.Count$
23			
24	maxPeople = Math.Max(	c_{11}	$queue.Count$
25	maxPeople, currentValue);		
26			
27	for (int i = 0; i < groups.Length;	c_{12}	$queue.Count * (groups.Length + 1)$
28	i++)		
29	{		
30	if ((usedMask & (1 << i)) != 0)	c_{13}	$queue.Count * groups.Length; X_1\{0,1\}$
31	{		
32	continue;	c_{14}	$queue.Count * groups.Length * X_1$
33	}		
34			
35	for (int j = 0; j <	c_{15}	$queue.Count * groups.Length * (caps.Length + 1)$
36	caps.Length; j++)		
37	{		
38	if (groups[i] > caps[j])	c_{16}	$queue.Count * groups.Length * caps.Length; X_2\{0,1\}$
39	{		
40	continue;	c_{17}	$queue.Count * groups.Length * caps.Length * X_2$
41	}		
42			
43	int[] newCaps	c_{18}	$queue.Count * groups.Length * caps.Length$
44	= (int[])caps.Clone();		
45	newCaps[j] -= groups[i];	c_{18}	$queue.Count * groups.Length * caps.Length$
46	int newMask = usedMask | (1	c_{18}	$queue.Count * groups.Length * caps.Length$
47	<< i);		
48	string newKey = newMask +	c_{18}	$queue.Count * groups.Length * caps.Length$
49	"|"		
50	+ string.Join(",",		$queue.Count * groups.Length * caps.Length$
51	newCaps);		
52	int newValue = currentValue	c_{19}	$queue.Count * groups.Length * caps.Length$
53	+ groups[i];		
54			
55	if (!dp.ContainsKey(newKey)	c_{20}	$queue.Count * groups.Length * caps.Length; X_3\{0,1\}$
56	|| newValue >		
57	dp[newKey])		
58	{		
59	dp[newKey] = newValue;	c_{21}	$queue.Count * groups.Length * caps.Length * X_3$
60	queue.Enqueue(	c_{21}	
61	(newMask,		$queue.Count * groups.Length * caps.Length * X_3$
62	newCaps));		
63	}		
64	}		
65	}		
66	}		
67			
68	return maxPeople;	c_{22}	1
69	}		

273