****

**Kauno technologijos universitetas**

Informatikos fakultetas

**Trečio laboratorinio darbo ataskaita**

3LD individuali užduotis

|  |  |
| --- | --- |
|  |  |
| **Aistis Jakutonis**  Studentas | (parašas) (data) |
|  |  |
| **Dalius Makackas**  **Andrius Kriščiūnas**  **Tadas Kraujalis**  **Vidmantas Rimavičius**  Dėstytojai | (parašas) (data) |
|  |  |

**Kaunas, 2025**

Turinys

[1. Pirmos užduoties dalys 3](#_Toc197201653)

[1.1. Pateikti uždavinio sprendimo algoritmą 3](#_Toc197201654)

[1.2. Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus 6](#_Toc197201655)

[1.3. Atlikti procedūros eksperimentinius tyrimus 7](#_Toc197201656)

[1.3.1. Laiko priklausomybė nuo skersinių skaičiaus 7](#_Toc197201657)

[1.3.2. Veiksmų skaičiaus priklausomybė nuo skersinių skaičiaus 8](#_Toc197201658)

[1.4. Pritaikius dinaminio programavimo metodologiją rekursinei lygybei realizuoti procedūrą bei atlikti eksperimentinius našumo tyrimus 8](#_Toc197201659)

[1.5. Atlikti procedūros eksperimentinius tyrimus: 9](#_Toc197201660)

[1.5.1. Laiko priklausomybė nuo skersinių skaičiaus 9](#_Toc197201661)

[1.5.2. Veiksmų skaičiaus priklausomybė nuo skersinių skaičiaus 10](#_Toc197201662)

[1.6. Atlikti realizuotų programinių kodų analizę 11](#_Toc197201663)

[2. Antros užduoties dalys 13](#_Toc197201664)

[2.1. Pateikti uždavinio sprendimo algoritmą 13](#_Toc197201665)

[2.2. Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus 14](#_Toc197201666)

[2.3. Atlikti procedūros eksperimentinius tyrimus 16](#_Toc197201667)

[2.3.1. Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus 16](#_Toc197201668)

[2.3.2. Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus 17](#_Toc197201669)

[2.4. Pritaikius dinaminio programavimo metodologiją rekursinei lygybei realizuoti procedūrą bei atlikti eksperimentinius našumo tyrimus 19](#_Toc197201670)

[2.5. Atlikti procedūros eksperimentinius tyrimus 21](#_Toc197201671)

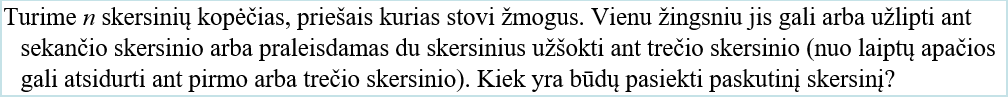
[2.5.1. Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus 21](#_Toc197201672)

[2.5.2. Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus 23](#_Toc197201673)

[2.6. Atlikti realizuotų programinių kodų žmonių grupių ir keltų skaičiaus 25](#_Toc197201674)

# Pirmos užduoties dalys

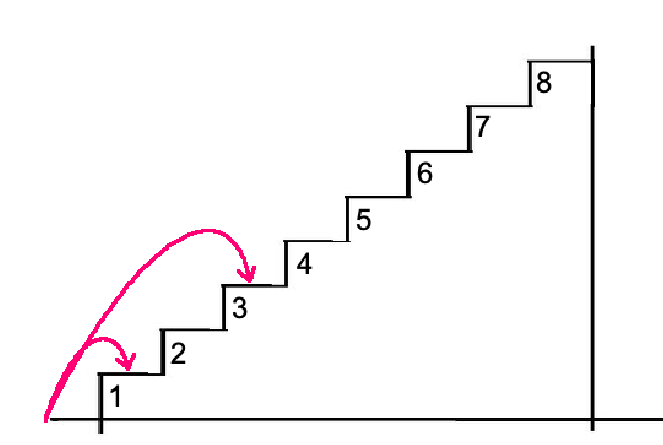
## Pateikti uždavinio sprendimo algoritmą



1 pav. Užduotis

Šioje užduotyje reikia pirmiausia atkreipti dėmesį į vieno žingsnio variantus:

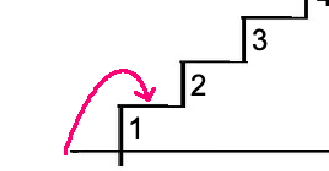
* Žmogus gali užlipti ant sekančio skersinio (tai reiškia, kad jei sakysime, jog žmogus stovi ant nulinio skersinio tai, kai įvykdys šį žingsnį jis jau stovės ant pirmojo skersinio).
* Žmogus gali peršokti du skersinius (tai reiškia, kad jei vėl sakysime, jog žmogus stovi ant nulinio skersinio tai, kai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



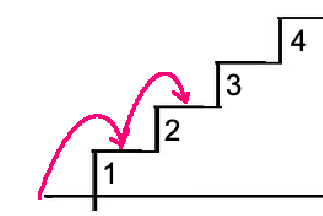
2 pav. Užduoties paaiškinimas

Toliau sprendžiant šį uždavinį sakysime, kad . Taip pat galimų būdų pasiekti paskutinį skersinį įsivardinsime kaip: .

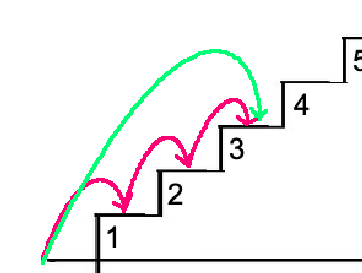
Taigi pereikime prie pavyzdžių:



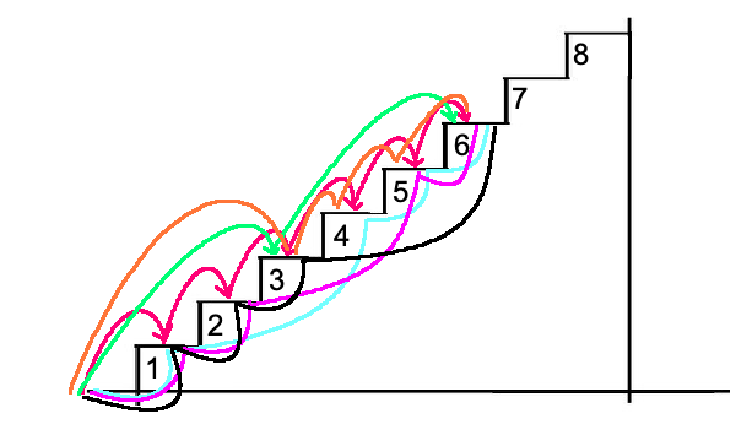
3 pav. Užduoties pavyzdys



4 pav. Užduoties pavyzdys



5 pav. Užduoties pavyzdys

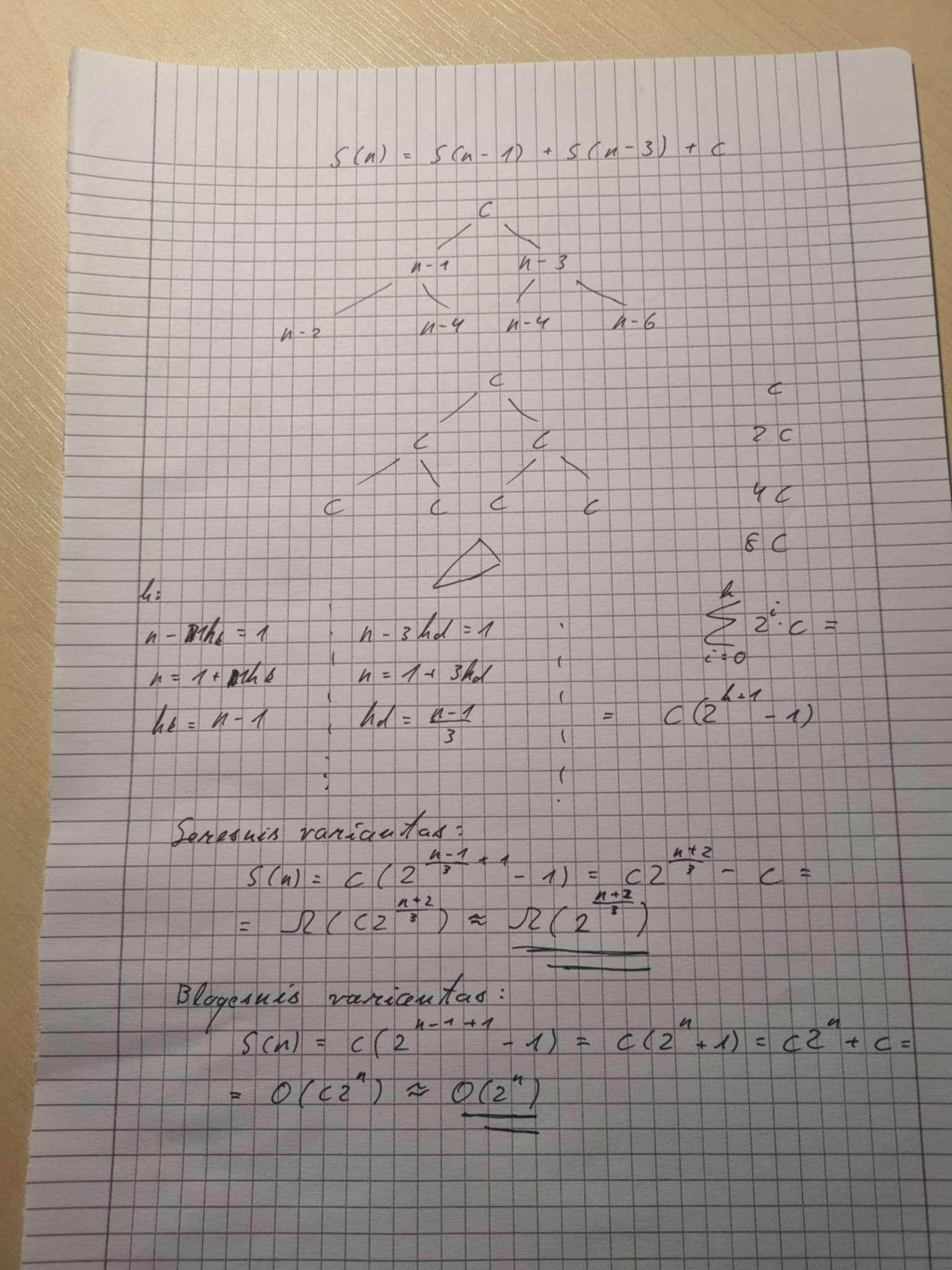


6 pav. Užduoties pavyzdys

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

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 arba užšokti praleidus du skersinius . Todėl visi galimi būdai pasiekti n-tąjį skersinį yra šių dviejų variantų suma, iš ko ir išsiveda anksčiau gauta lygybė.

Iš gautos lygybės seka:



7 pav. Sąryšio sprendimas

## Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus

Sudaryta rekursinė procedūra:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **static** **int** **SkaiciuotiBudus1**(**int** n)  {  **if** (n < **0**)  {  **return** **0**;  }  **if** (n == **0**)  {  **return** **1**;  }  **return** **SkaiciuotiBudus1**(n - **1**) + SkaiciuotiBudus1(n - **3**);  } | |

## Atlikti procedūros eksperimentinius tyrimus

### 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

### 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

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

Realizuota procedūra:

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

## Atlikti procedūros eksperimentinius tyrimus:

### 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

### 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

## Atlikti realizuotų programinių kodų analizę

Programinio kodo analizė:

5 lentelė Rekursinės procedūros kodo analizės lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **static** **int** **SkaiciuotiBudus1**(**int** n)  {  **if** (n < **0**)  {  **return** **0**;  }  **if** (n == **0**)  {  **return** **1**;  }  **return** **SkaiciuotiBudus1**(n - **1**) + SkaiciuotiBudus1(n - **3**);  } | c1  c2  c3  c4  c5 |  |
| Geriausias atvejis, kai:  Blogiausias atvejis, kai: | | | |

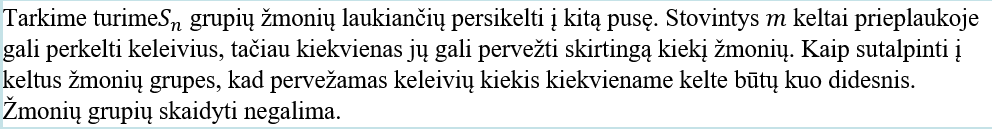
6 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **static** **int** **SkaiciuotiBudus2**(**int** n)  {  **if** (n < **0**)  {  **return** **0**;  }  **int**[] budai = **new** **int**[n + **1**];  budai[**0**] = **1**;  **for** (**int** i = **1**; i <= n; i++)  {  budai[i] += budai[i - **1**];  **if** (i >= **3**)  {  budai[i] += budai[i - **3**];  }  }  **return** budai[n];  } | c1  c2  c3  c4  c5  c6  c7  c8  c9 |  |
| Geriausias atvejis, kai:  Blogiausias atvejis, kai: | | | |

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

# Antros užduoties dalys

## Pateikti uždavinio sprendimo algoritmą



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:

Čia:

grupių žmonių, kurių skaičiai yra masyve: – žmonių kiekis i-tojoje grupėje

keltų, kurių talpos yra masyve: – j-to kelto maksimali talpa

– tai žmonių skaičius nuo grupės k+1 iki i, kurias norime įdėti į dabartinį keltą

Uždavinio sąryšio paaiškinimas:

Žiūrime į pirmų žmonių grupių. Norime paskutiniam keltui priskirti kiek galima daugiau grupių, kurios telpa į jo talpą. Tada reikia pažiūrėti kiek maksimaliai buvome perplukdę iki grupės su 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:

Ir turime tokias keltų talpas:

Pradiniai atvejai:

Pavyzdžio sprendimas:

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

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:

Ir turime tokias keltų talpas:

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:

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.

## Sudaryti rekursinę procedūrą ir atlikti eksperimentinius našumo tyrimus

Sudaryta rekursinė procedūra:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | **static** **int** **Solve**(**int** i, **int** j)  {  **if** (i >= groups.Length || j >= capacity.Length)  {  **return** **0**;  }  **if** (dp[i, j] != -**1**)  {  **return** dp[i, j];  }  **int** maxPeople = **0**;  **int** sum = **0**;  **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:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | **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;  **return** maxValue;  } | |

## Atlikti procedūros eksperimentinius tyrimus

### Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus

7 lentelė Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus lentelė

5 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

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 |

6 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

### Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus

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 |

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

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

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

Realizuota procedūra:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | **static** **int** **SolveDP**()  {  **int** n = groups.Length;  **int** m = capacity.Length;  dp[**0**, **0**] = **0**;  **for** (**int** j = **0**; j < m; j++)  {  **for** (**int** i = **0**; i <= n; i++)  {  **if** (dp[i, j] == -**1**)  {  **continue**;  }  **int** sum = **0**;  **for** (**int** k = i; k < n; k++)  {  sum += groups[k];  **if** (sum > capacity[j])  {  **break**;  }  dp[k + **1**, j + **1**] = Math.Max(dp[k + **1**, j + **1**], dp[i, j] + sum);  }  }  }  **int** max = **0**;  **for** (**int** i = **0**; i <= n; i++)  {  max = Math.Max(max, dp[i, m]);  }  **return** max;  } | |

Pasunkintos užduoties realizuota procedūra:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | **static** **int** **IterativeDP**()  {  **var** dp = **new** Dictionary<**string**, **int**>();  **var** queue = **new** Queue<(**int** usedMask, **int**[] caps)>();  **int**[] startCaps = (**int**[])ferries.Clone();  **string** startKey = "0|" + **string**.Join(",", startCaps);  dp[startKey] = **0**;  queue.Enqueue((**0**, startCaps));  **int** maxPeople = **0**;  **while** (queue.Count > **0**)  {  **var** (usedMask, caps) = queue.Dequeue();  **string** stateKey = usedMask + "|" + **string**.Join(",", caps);  **int** currentValue = dp[stateKey];  maxPeople = Math.Max(maxPeople, currentValue);  **for** (**int** i = **0**; i < groups.Length; i++)  {  **if** ((usedMask & (**1** << i)) != **0**)  **continue**;  **for** (**int** j = **0**; j < caps.Length; j++)  {  **if** (groups[i] > caps[j])  **continue**;  **int**[] newCaps = (**int**[])caps.Clone();  newCaps[j] -= groups[i];  **int** newMask = usedMask | (**1** << i);  **string** newKey = newMask + "|" + **string**.Join(",", newCaps);  **int** newValue = currentValue + groups[i];  **if** (!dp.ContainsKey(newKey) || newValue > dp[newKey])  {  dp[newKey] = newValue;  queue.Enqueue((newMask, newCaps));  }  }  }  }  **return** maxPeople;  } | |

## Atlikti procedūros eksperimentinius tyrimus

### Laiko priklausomybė nuo žmonių grupių ir keltų skaičiaus

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 |

9 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

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 |

10 grafikas Laiko priklausomybės nuo žmonių grupių ir keltų skaičiaus grafikas

### Veiksmų skaičiaus priklausomybė nuo žmonių grupių ir keltų skaičiaus

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

## Atlikti realizuotų programinių kodų žmonių grupių ir keltų skaičiaus

Programinio kodo analizė:

15 lentelė Rekursinės procedūros kodo analizės lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | **static** **int** **Solve**(**int** i, **int** j)  {  **if** (i >= groups.Length ||  j >= capacity.Length)  {  **return** **0**;  }  **if** (dp[i, j] != -**1**)  {  **return** dp[i, j];  }  **int** maxPeople = **0**;  **int** sum = **0**;  **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;  } | c1  c2  c3  c4  c5  c5  c6  c7  c8  c9  T(k+1,j+1)  c10  c11  c12 | 1  1  1  1 |
| Geriausias atvejis, kai:  Blogiausias atvejis, kai: | | | |

16 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | **static** **int** **SolveDP**()  {  **int** n = groups.Length;  **int** m = capacity.Length;  dp[**0**, **0**] = **0**;  **for** (**int** j = **0**; j < m; j++)  {  **for** (**int** i = **0**; i <= n; i++)  {  **if** (dp[i, j] == -**1**)  {  **continue**;  }  **int** sum = **0**;  **for** (**int** k = i; k < n; k++)  {  sum += groups[k];  **if** (sum > capacity[j])  {  **break**;  }  dp[k + **1**, j + **1**] =  Math.Max(  dp[k + **1**, j + **1**],  dp[i, j] + sum  );  }  }  }  **int** max = **0**;  **for** (**int** i = **0**; i <= n; i++)  {  max = Math.Max(max, dp[i, m]);  }  **return** max;  } | c1  c1  c2  c3  c4  c5  c6  c7  c8  c9  c10  c11  c12  c13  c14  c15  c16 | 1  1  1  1  1 |
| Geriausias atvejis, kai:  Blogiausias atvejis, kai: | | | |

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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | **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;  **return** maxValue;  } | c1  c2  c3  c4  c5  c6  c7  c8  c9  c10  c11  c12  c13  T(M,C)  c14  c15  c16  c17 | 1  1  1  1 |

18 lentelė Dinaminio programavimo procedūros kodo analizės lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | **static** **int** **IterativeDP**()  {  **var** dp = **new** Dictionary<**string**, **int**>();  **var** queue = **new** Queue<(  **int** usedMask, **int**[] caps)>();  **int**[] startCaps = (**int**[])ferries.Clone();  **string** startKey = "0|"  + **string**.Join(",", startCaps);  dp[startKey] = **0**;  queue.Enqueue((**0**, startCaps));  **int** maxPeople = **0**;  **while** (queue.Count > **0**)  {  **var** (usedMask, caps)  = queue.Dequeue();  **string** stateKey = usedMask + "|"  + **string**.Join(",", caps);  **int** currentValue = dp[stateKey];  maxPeople = Math.Max(  maxPeople, currentValue);  **for** (**int** i = **0**; i < groups.Length; i++)  {  **if** ((usedMask & (**1** << i)) != **0**)  {  **continue**;  }  **for** (**int** j = **0**; j < caps.Length; j++)  {  **if** (groups[i] > caps[j])  {  **continue**;  }  **int**[] newCaps  = (**int**[])caps.Clone();  newCaps[j] -= groups[i];  **int** newMask = usedMask | (**1** << i);  **string** newKey = newMask + "|"  + **string**.Join(",", newCaps);  **int** newValue = currentValue + groups[i];  **if** (!dp.ContainsKey(newKey)  || newValue > dp[newKey])  {  dp[newKey] = newValue;  queue.Enqueue(  (newMask, newCaps));  }  }  }  }  **return** maxPeople;  } | c1  c2  c3  c4  c5  c6  c7  c8  c9  c9  c10  c11  c12  c13  c14  c15  c16  c17  c18  c18  c18  c18  c19  c20  c21  c21  c22 | 1  1  1  1  1  1  1  1 |