

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Algoritmų sudarymas ir analizė

2 Laboratorinis darbas

Atliko:

IF 8/1 grupės stud.

Tomas Odinas

Priėmė

doc. Dalius Makackas

KAUNAS, 2020

TURINYS

[1. Užduotis 2](#_Toc38655443)

[2. Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją 2](#_Toc38655444)

[4. Eksperimentinis algoritmų sudėtingumo įvertinimas 3](#_Toc38655445)

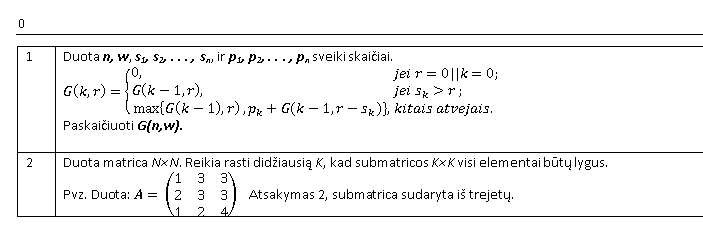
[Kai elementai saugomi operatyvioje atmintyje: 3](#_Toc38655446)

[Kai elementai saugomi diskinėje atmintyje: 4](#_Toc38655447)

[5. Išvados 5](#_Toc38655448)

[6. Priedas 6](#_Toc38655449)

# 1 Uždavinys



# Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją:

**static** **int** **Ga**(**int** k, **int** r)

{

**if** (r == **0** || k == **0**) **return** **0**;

**if** (s[k - **1**] > r) **return** Ga(k - **1**, r);

**else** **return** Math.Max(Ga(k - **1**, r), p[k - **1**] + Ga(k - **1**, r - s[k - **1**]));

}

# Algoritmo sudėtingumas:

Geriausiu atveju: kai r = arba k = 0, tada

Blogiausiu atveju:

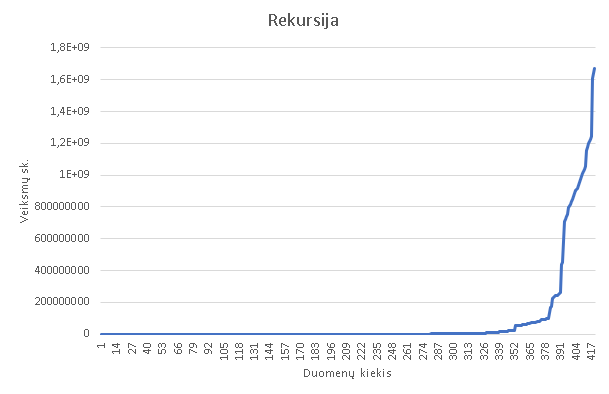
Sprendžiant medžio metodu gauname

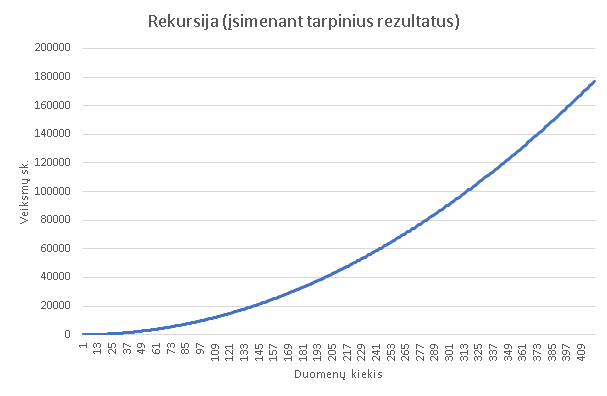
Rekurentinės lygties algoritmas realizuotas panaudojant savybę, kad galime įsiminti dalinių sprendinių vertes

static int Gc(int k, int r)  
{  
 for (int kk = 0; kk <= k; kk++)  
 for (int rr = 0; rr <= r; rr++)  
 {  
 if (rr == 0 || kk == 0) cache2[kk, rr] = 0;  
  
 else if (s[kk - 1] > rr) cache2[kk, rr] = cache2[kk - 1, rr];  
  
 else cache2[kk, rr] = Math.Max(cache2[kk - 1, rr], p[kk - 1] + cache2[kk - 1, rr - s[kk - 1]]);  
 }  
  
 return cache2[k, r];  
}

## Eksperimentinis algoritmų sudėtingumo įvertinimas

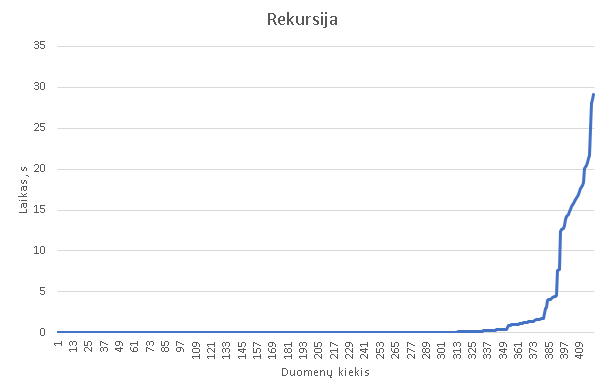
**Veiksmų skaičius:**

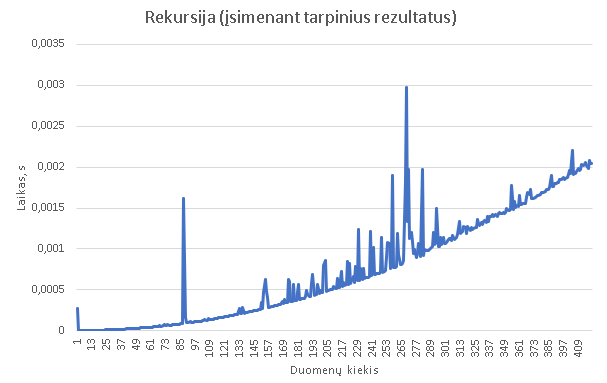




|  |  |  |
| --- | --- | --- |
| Kiekis | Rekursija (veiksmai) | Rekursija įsimenant reikšmes (veiksmai) |
| 2 | 3 | 9 |
| 4 | 5 | 25 |
| 8 | 9 | 81 |
| 16 | 17 | 289 |
| 32 | 33 | 1089 |
| 64 | 98 | 4225 |
| 128 | 2878 | 16641 |
| 256 | 451931 | 66049 |
| 420 | 1672471361 | 177241 |

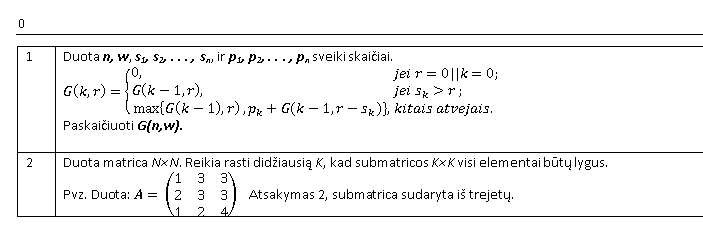
**Laikas:**





|  |  |  |
| --- | --- | --- |
| Kiekis | Rekursija (sekundės) | Rekursija įsimenant reikšmes (sekundės) |
| 2 | 0,0000024 | 0,0000024 |
| 4 | 0,0000021 | 0,0000022 |
| 8 | 0,0000022 | 0,0000029 |
| 16 | 0,0000022 | 0,0000053 |
| 32 | 0,0000029 | 0,0000138 |
| 64 | 0,0000044 | 0,0000474 |
| 128 | 0,0000552 | 0,0001899 |
| 256 | 0,0079711 | 0,0007552 |
| 420 | 29,0998506 | 0,0020483 |

# 2 Uždavinys



Optimalios struktūros paieška:

1. Nusakyti optimalią uždavinio sprendimo struktūrą:

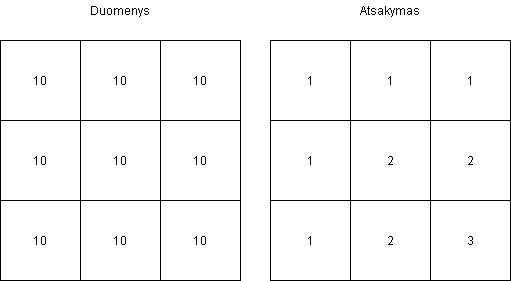
Tarkime kad elementas nurodo maksimalų submatricos , kurioje visi elementai vienodi, sudarytos iš matricos kraštinės ilgį.

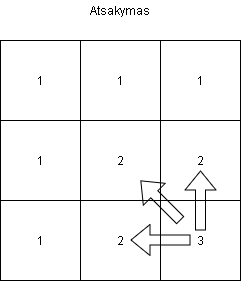
Jei visi elementai vienodi – tada

Jei bent 1 elementas skirtingas – tada , nes kiekvienas elementas sudaro dydžio matricą.

priklauso nuo, ir , kurie turi sprendinius atitinkamai į dydžio problemas, kurios persidengia. Vadinasi

Pvz:





1. Rekursiškai apibrėžti optimalų uždavinio sprendinį:
2. Apskaičiuoti optimalaus sprendinio reikšmę:

K = max(c[])

1. Rasti galutinį sprendinį:

Papildomai įsiminsime indeksus i ir j, kad butų galima nustatyti indeksus, kuriais apribota submatrica matricoje

Programos pseudokodas:

RastiK(m[])

{

// pirmos eilutes ar stulpelio elementai

// sudarys tik 1x1 kvadratus

Ats[**0**,] = Ats[,**0**] = **1**;

**for**(eilutes i **in** m)

**for**(stulpeliai j **in** m)

{

// jei 4 vienodi

**if**(m[i, j] = m[i - **1**, j] = m[i, j - **1**] = m[i - **1**, j - **1**])

Ats[i, j] = min(Ats[i - **1**, j], Ats[i, j - **1**], Ats[i - **1**, j - **1**]) + **1**;

}

**return** max(Ats[]);

}

Algoritmo sudėtingumas:

# 5. Išvados

# 6. Priedas

Nuoroda į programinio kodo saugyklą: <https://github.com/algoritmu-sudarymas-ir-analize/L2>