

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Algoritmų sudarymas ir analizė

2 Laboratorinis darbas

Atliko:

IF 8/1 grupės stud.

Martynas Kemežys

Priėmė

doc. Dalius Makackas

KAUNAS, 2020

TURINYS

[1 Uždavinys 3](#_Toc42466427)

[**Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją**: 3](#_Toc42466428)

[**Algoritmo sudėtingumas** 4](#_Toc42466429)

[Rekurentinės lygties algoritmas realizuotas panaudojant savybę, kad galime įsiminti dalinių sprendinių vertes 4](#_Toc42466430)

[Eksperimentinis algoritmų sudėtingumo įvertinimas 5](#_Toc42466431)

[2 Uždavinys 7](#_Toc42466432)

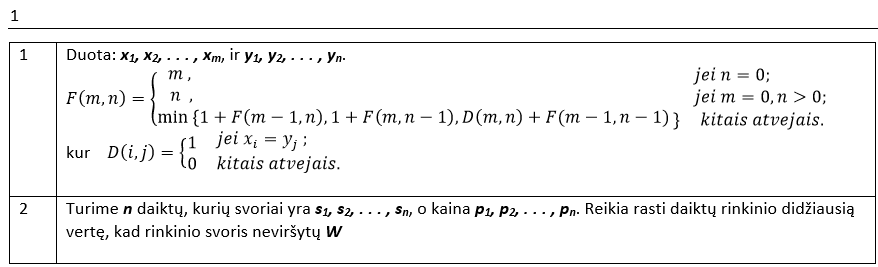
[**Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją**: 7](#_Toc42466433)

[**Algoritmo sudėtingumas** 8](#_Toc42466434)

[**Algoritmo sudėtingumas** 9](#_Toc42466435)

[Eksperimentinis algoritmų sudėtingumo įvertinimas 10](#_Toc42466436)

# 1 Uždavinys



Duotai rekurentinei formulei sudarykite du algoritmus:

• Tiesiogiai panaudojant rekursiją;

• Panaudojant dinaminio programavimo metodo savybę, kad galime įsiminti dalinius sprendinius;

Programiškai realizuokite ir eksperimentiškai įvertinkite ir palyginkite abiejų algoritmų sudėtingumą.

# Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją:

**static** **int** **F1**(**int** m, **int** n)

{

**if** (n == **0**) return **m**;

**if** (m == **0** &&n > **0**) return **n;**

**else** return Math.Min(**1** + F1(m – **1**, n), Math.Min(**1** + F1(m, n – **1**), D1(m, n) + F1(m – **1**, n – **1**)));

}

**static** **int** **D1**(**int** i, **int** j)

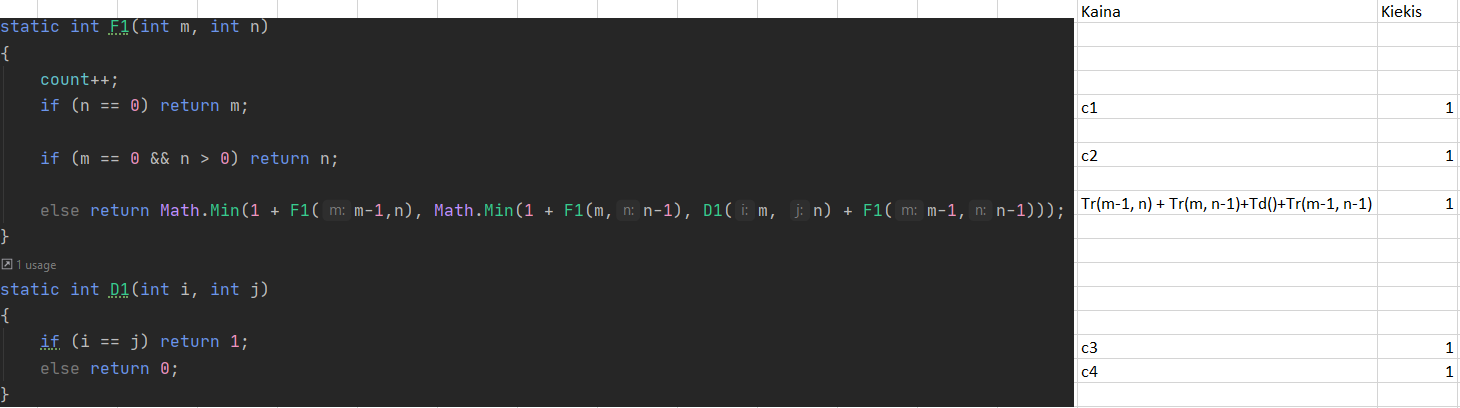
{

**if** (i == j) return **1**;

**else** return **0;**

}

# Algoritmo sudėtingumas



Algoritmo sudėtingumas:

Geriausiu atveju: kai n = 0 arba (m = 0 ir n > 0), tada

Blogiausiu atveju:

Jeigu laikysime kad m=n, tai galime sakyti, kad algoritmo sudėtingumas yra:

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

**static** **int** **F2**(**int** m, **int** n)

{

**for** (**int** mm = 0; mm <= m; mm++)  
 **for** (**int** nn = 0; nn <= n; nn++)  
 {

**if** (nn == **0**) cache[mm, nn] = **m**;

**else if** (mm == **0** &&nn > **0**) cache[mm, nn] = **n;**

**else** Math.Min(**1** + cache[mm – **1**, nn], Math.Min(**1** + cache[mm, nn - **1**], D2[mm, nn] + cache[mm – **1**, nn – **1**]));

}

return cache[m, n];

}

**static** **int** **D2**(**int** i, **int** j)

{

**for** (**int** ii = 0; ii <= i; ii++)  
 **for** (**int** jj = 0; jj <= j; jj++)  
 {

**if** (ii == jj) dcache[ii, jj] = **1**;

**else** dcache[ii, jj] = **0**;

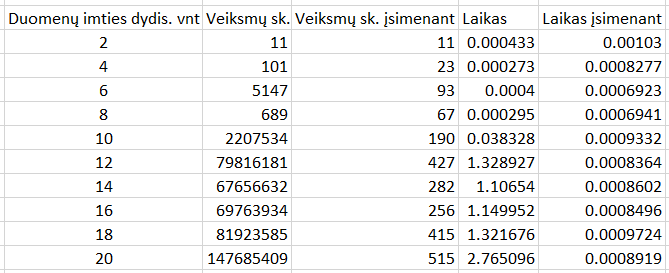
}

return dcache[i, j];

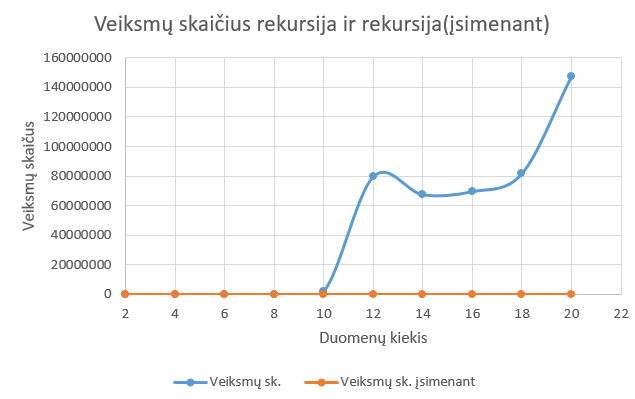
}

N\*m

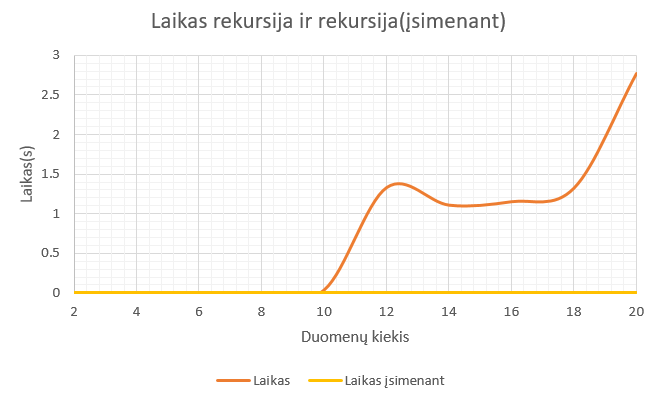
## Eksperimentinis algoritmų sudėtingumo įvertinimas



**Veiksmų skaičius:**

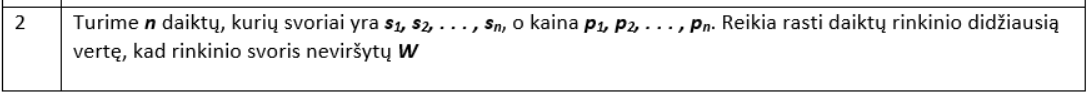


**Laikas:**



Aiškiai matome, kad algoritmas įsimenant veikia žymiai geriau nei neįsimenant veikiantis algoritmas.

# 2 Uždavinys



Pagal pateiktą žodinį uždavinio aprašymą, naudojantis dinaminio programavimo metodu, sudaryti uždavinio sprendimo algoritmą ir įvertinti jo sudėtingumą.

Algoritmo sudarymas turi apimti tokius žingsnius:

• pradinio uždavinio suskaidymas į mažesnius uždavinius (rekurentinės formulės užrašymas),

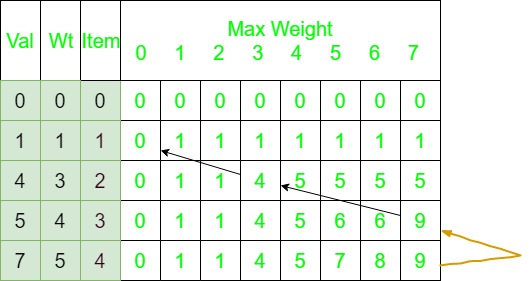
• suskaidytų uždavinių sprendimo iliustravimas. Iliustravimui galima naudoti grafines priemones,

• programos pseaudo kodo sudarymas su paaiškinimais.

Programiškai realizuokite ir eksperimentiškai įvertinkite sudaryto algoritmo sudėtingumą

Algoritmui įgyvendinti naudojau „ Knapsack bottom up“ metodiką

<https://www.youtube.com/watch?v=nLmhmB6NzcM>



# Rekurentinės lygties algoritmas realizuotas panaudojant rekursiją:

Rekursyvi uždavinio imlplementacija

**public static** **int** **Kp**(**int** n, **int** W, **int[]** s, **int[]** p)

{

**if** (n == **0** ||W== **0**) return **0**;

**else** **if** (s[n - 1] > W) return **Kp(**n - 1, W, s, p**)**;

**else** return Math.Max(**Kp**(n – 1, W, s, p), **p**[n – 1] + **Kp**(n – 1, W – s[n – 1], s, p));

}

# Algoritmo sudėtingumas

Sudėtingumas , nes išbandomi visi variantai.

# Rekurentinės lygties algoritmas realizuotas panaudojant dinaminį programavimą

Dinaminė uždavinio implementacija

**public static** **int** **KpBottomUp**(**int** k, **int** W, **int[]** s, **int[]** p)

{

**int** n = s.**Length**;

**int[,]** arr = **new int**[n + 1, W + 1];

**for** (**int** i = 0; i < n; i++){

**for** (**int** j = 0; j < W; j++){

arr[i, j] = 0;

}

}

**for** (**int** i = 1; i <= n; i++){

**for** (**int** j = 0; j <= W; j++){

**int** weight = s[i -1];

**if** (weight <= j){

arr[i, j] = **Math.Max**(p[i – 1] + arr[i – 1, j – weight], arr[i – 1, j]);

} **else** {

arr[i, j] = arr[i – 1, j];

}

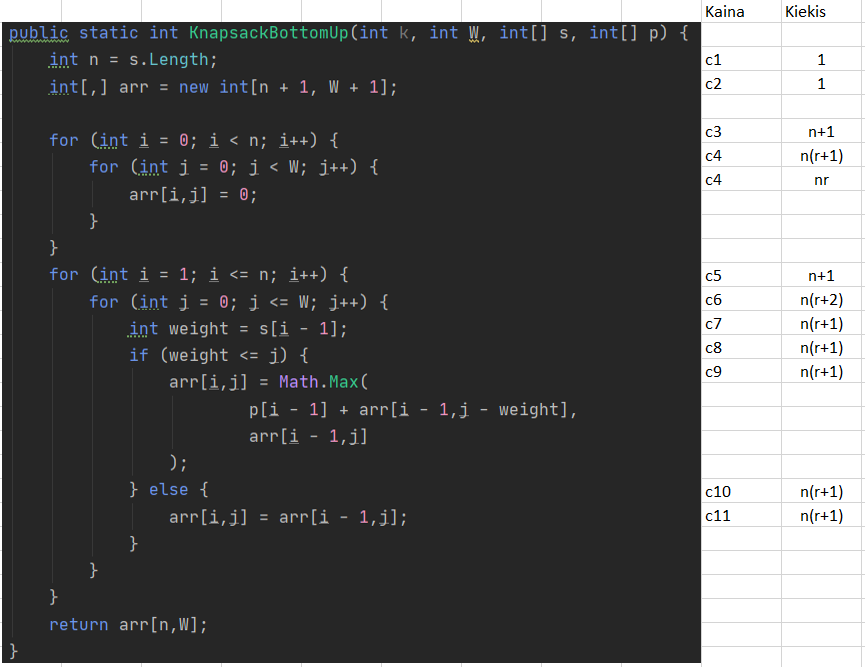
}

}

**return** arr[n, W];

}

# Algoritmo sudėtingumas



Sudėtingumas

## 

## Eksperimentinis algoritmų sudėtingumo įvertinimas

