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

1

1	Duota: $x_1, x_2, \ldots, x_m$, ir $y_1, y_2, \ldots, y_n$. $F(m,n) = \begin{cases} m, & jei\ n = 0; \\ n, & jei\ m = 0, n > 0; \\ \min\{1 + F(m-1,n), 1 + F(m,n-1), D(m,n) + F(m-1,n-1)\} & kitais\ atvejais. \end{cases}$ kur $D(i,j) = \begin{cases} 1 & jei\ x_i = y_j; \\ 0 & kitais\ atvejais. \end{cases}$

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

```
static int F1(int m, int n)
{
    count++;
    if (n == 0) return m;

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

    else return Math.Min(1 + F1( m: m-1,n), Math.Min(1 + F1(m, n: n-1), D1( i: m,  j: n) + F1( m: m-1, n: n-1)));
}
1 usage
static int D1(int i, int j)
{
    if (i == j) return 1;
    else return 0;
}
```

Kaina	Kiekis
c1	1
c2	1
Tr(m-1, n) + Tr(m, n-1)+Td()+Tr(m-1, n-1)	1
c3	1
c4	1

Algoritmo sudėtingumas:

Geriausiu atveju: kai n = 0 arba (m = 0 ir n > 0), tada $Tr(m,n) = \Omega(1)$

Blogiausiu atveju: $\underline{Tr(m,n) = Tr(m-1,n) + Tr(m,n-1) + Tr(m-1,n-1) + C}$

Jeigu laikysime kad m=n, tai galime sakyti, kad algoritmo sudėtingumas yra: $O(3^n)$

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];
}
```

Eksperimentinis algoritmų sudėtingumo įvertinimas

Duomenų imties dydis. vnt	Veiksmų sk.	Veiksmų sk. įsimenant	Laikas	Laikas įsimenant
2	11	11	0.000433	0.00103
4	101	23	0.000273	0.0008277
6	5147	93	0.0004	0.0006923
8	689	67	0.000295	0.0006941
10	2207534	190	0.038328	0.0009332
12	79816181	427	1.328927	0.0008364
14	67656632	282	1.10654	0.0008602
16	69763934	256	1.149952	0.0008496
18	81923585	415	1.321676	0.0009724
20	147685409	515	2.765096	0.0008919

Veiksmų skaičius:

Laikas:

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

2 Uždavinys

2	Turime n daiktų, kurių svoriai yra $s_1, s_2, \ldots, s_n$, o kaina $p_1, p_2, \ldots, p_n$. Reikia rasti daiktų rinkinio didžiausią vertę, kad rinkinio svoris neviršytų W

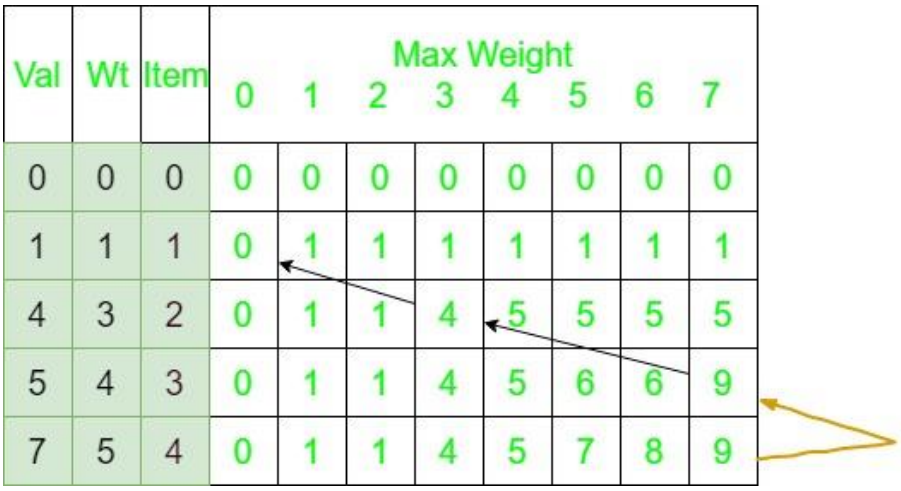
Pagal pateiktą žodinį uždavinio aprašymą, naudojantis dinaminio programavimo metodu, sudaryti uždavinio sprendimo algoritmą ir įvertinti jo sudėtingumą.

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



Val	Wt	Item	Max Weight 0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1
4	3	2	0	1	1	4	5	5	5	5
5	4	3	0	1	1	4	5	6	6	9
7	5	4	0	1	1	4	5	7	8	9

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

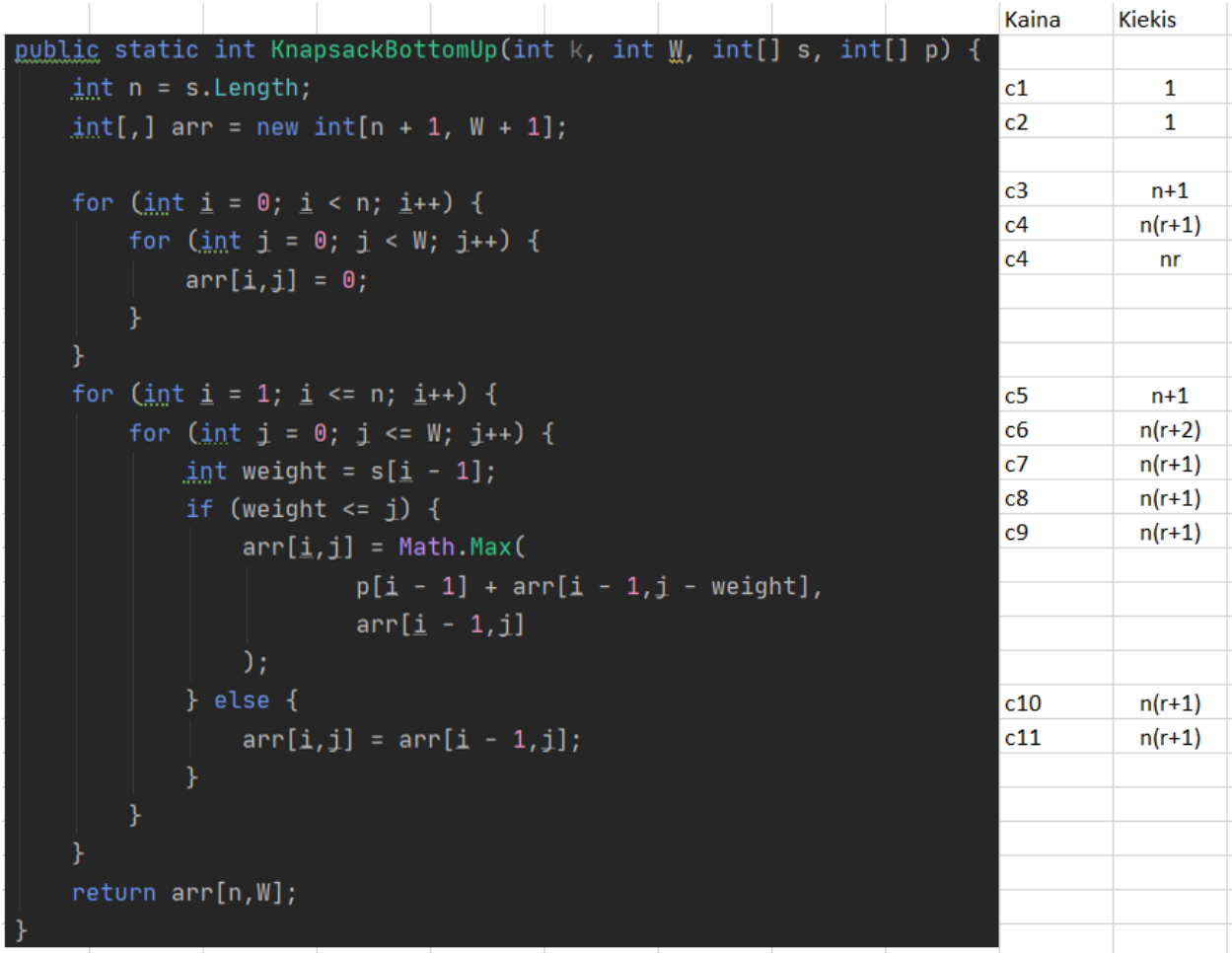
Sudėtingumas $O(2^n)$, nes išbandomi visi variantai.

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



```
public static int KnapsackBottomUp(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];
}
```

Kaina	Kiekis
c1	1
c2	1
c3	n+1
c4	n(r+1)
c4	nr
c5	n+1
c6	n(r+2)
c7	n(r+1)
c8	n(r+1)
c9	n(r+1)
c10	n(r+1)
c11	n(r+1)

Sudėtingumas $\underline{O(n * W)}$

Eksperimentinis algoritmų sudėtingumo įvertinimas

Duomenų kiekis	Laikas rekursyvi	Laikas dinaminė
10	0.0009871	0.0009851
20	0.0002043	0.0005172
30	0.0013665	0.0005349
40	0.0092069	0.0006208
50	0.170379	0.0005326
60	0.5195163	0.0005008
70	2.8626945	0.0005506
80	6.7576112	0.0005039
90	21.8003958	0.0006727
100	64.2523255	0.0007304