Knapsack

Bianca-Mihaela Stan, grupa 232

March 2021

1 Knapsack - 2p

Fie un șir de numere $S = s_1, s_2, \ldots, s_n$ și un număr natural K, cu $K \geq s_i$ pentru orice i între 1 și n.
a) Să se scrie un algoritm pseudo-polinomial care găsește suma maximă, dar care să fie $\leq$K, ce poate fi formată din elemente din S (numere întregi, pozitive, luate cel mult o singură dată).(1p)

```
#include <iostream>
#include <fstream>
using namespace std;

ifstream f ("a.in");
ofstream g ("a.out");
const int nr_maxim_elemente = 100;
const int k_maxim = 100;


int v[nr_maxim_elemente+5];
int dp[nr_maxim_elemente+5][k_maxim+5];

/*
dp[i][j] = suma maxima care poate fi formata din primele i elemente, astfel incat suma <= j
dp[i][j] = max(dp[i-1][j], dp[i-1][j-v[i]]+v[i]), daca v[i]<=j
           dp[i-1][j], daca v[i]>j
           0, daca i==0 sau j==0
*/

int main()
{
    int n, k;
    f>>n;
    f>>k;
    for(int i=0; i<n; i++)
    {
        f>>v[i];
    }

    for(int i=0; i<=n; i++)
    {
        for(int j=0; j<=k; j++)
        {
            if(i==0 || j==0)
```

```
            {
                dp[i][j]=0;
            }
            else
            {
                if(v[i]<=j)
                {
                    dp[i][j]=max(dp[i-1][j], dp[i-1][j-v[i]] + v[i]);
                }
                else
                {
                    dp[i][j]=dp[i-1][j];
                }
            }
        }
    }

    cout<<dp[n][k];
    return 0;
}
```

Complexitate de timp: O(n*k).

b) Să se găsească un algoritm aproximativ care calculează o sumă cel puțin pe jumătate de mare ca cea optimă dar rulează în timp O(n) și complexitate spațiu O(1). Mai exact: aveți voie să parcurgeți fiecare element din S cel mult o singură dată, respectiv aveți memorie alocată doar pentru 3 variabile de tip int (dintre care una este K) + o variabila de tip ifstream (1p)

Problema prezentata este un caz particular la Knapsack problem: raportul valoare/greutate este pentru fiecare obiect = 1.

Asadar, aplicam algoritmul 1/2 aproximativ de la Knapsack problem cu mentiunea ca elementele nu mai trebuie sortate dupa raportul valoare/greutate, pentru ca e acelasi pentru fiecare element.
Deci algoritmul este:

```
#include <iostream>
#include <fstream>
#include <algorithm>
using namespace std;

ifstream f ("a.in");
ofstream g ("a.out");

int main()
{
    // x - variabila de tip ifstream
    int x;
    // Cele 3 variabile de tip int sunt:
    // k - dat in problema
    // elem_max - care va retine elementul maxim din sirul dat
    // sum - suma calculata prin algoritmul tip greedy
    int k, elem_max=0, sum=0;
    f>>k;
    while(f>>x)
    {
```

```
            if(x<=k)
            {
                sum+=x;
                k-=x;
            }
            elem_max=max(elem_max, x);
        }

        cout<<max(elem_max, sum);
        return 0;
}
```

Evident, complexitatea de timp a algoritmlului prezentat este O(n) pentru ca doar citim numerele.

Demonstratie ca algoritmul prezentat este un algoritm 1/2 aproximativ:

Are sens implementarea?

- sum $\leq$ k pentru ca asa am pus conditia
- elementul maxim < k pentru ca k > s_i oroicare ar fi i

Justificarea factorului de aproximare:

Notam:

- $OPT_{1/0}$ = rezultatul optim pentru problema noastra
- OPT_G = rezultatul optim pentru problema noastra, doar ca avem voie sa "taiem" numerele
- ALG = valoarea optima obtinuta de algoritmul propus de mine
- O_j = primul numar care nu este ales de algoritmul propus de mine
- O_p = valoarea celui mai mare numar din lista
- I = un set de date pentru problema noastra

Evident, daca putem sa taiem numerele, putem sa facem o suma cel putin egala cu cea obtinuta fara taiere.

$$OPT_{1/0}(I) \leq OPT_G(I) \qquad (1)$$

In algoritmul in care avem voie sa taiem obiectele, putem adauga si o parte din acest obiect O_j.

$$OPT_G(I) < \sum_{i=1}^{j} O_i \qquad (2)$$

Din (1) si (2) rezulta:

$$OPT_{1/0}(I) \leq \sum_{i=1}^{j} O_i$$

$$OPT_{1/0}(I) \leq \sum_{i=1}^{j-1} O_i + O_j \qquad (3)$$

Stim ca:

$$O_j \leq O_p$$

Inlocuind in (3):

$$OPT_{1/0}(I) \leq \sum_{i=1}^{j-1} O_i + O_j \leq \sum_{i=1}^{j-1} O_i + O_p$$

Stim ca:

$$\sum_{i=1}^{j-1} O_i \leq ALG(I)$$

$$O_p \leq ALG(I)$$

Deci:

$$OPT_{1/0}(I) \leq ALG(I) + ALG(I)$$

$$OPT_{1/0}(I) \leq 2 * ALG(I)$$

Si, evident:

$$ALG(I) \leq OPT_{1/0}(I)$$

pentru ca daca algoritmul nostru gaseste o valoare mai buna, inseamna ca optimul nu era optim.

Deci:

$$ALG(I) \leq OPT_{1/0}(I) \leq 2 * ALG(I)$$

Deci avem un algoritm 1/2-aproximativ.