

Proiectarea algoritmilor

Raport Tehnic

-Tema de casă-

**Studentă: Rîcea Florentina-Alexandra ricea.florentina.d7i@student.ucv.ro**

Universitatea din Craiova

Facultatea de Automatică, Calculatoare si Electronică

Specializarea: Calculatoare și Tehnologia Informației cu predare în limba română

Anul: I

Grupa: CR1.3A

**Enunțul problemei:**

Să presupunem că sunteți agent de investiții ¸si că dispuneți de un capital C. Sunt disponibile pentru investiție n oferte dintre care trebuie să alegeți. Pentru fiecare ofertă i sunt disponibile ai acțiuni de valoare vi fiecare, ce vă pot aduce un profit estimat pi pentru fiecare acțiune achiziționată. Se cere să determinați investiția care vă poate aduce profitul estimat maxim. Se vor implementa doi algoritmi diferiți.

**Algoritmi:**

**Algoritm 1(Programare dinamica)**

**Dynamic\_Solution(C,actiuni\_val,profit,n)**

1.for i=0,n do

2. for c=0,C do

3. if i=0 or c=0 then

4. k[i][c] <= 0

5. else if actiuni\_val[i-1] <= c then

6. K[i][c] <= maxx(profit[i-1]+K[i-1][c-actiuni\_val[i-1],K[i-1][c])

7. else

8. K[i][c] <= K[i-1][c]

9.return K[n][C]

Codul folosește programare dinamică pentru a evita calculele repetate și a reduce timpul de execuție. Întrucât avem de a face cu o problemă de optimizare, care implică găsirea valorii maxime.

Funcția Dynamic\_Solution primește următoarele argumente:  
- C: capitalul disponibil pentru investiție  
- actiuni\_val: o listă cu valorile fiecărei acțiuni disponibile pentru investiție  
- profit: o listă cu profitul corespunzător fiecărei acțiuni achiziționate  
- n: numărul total de acțiuni disponibile pentru investiție

Codul pornește cu o matrice bidimensională K cu dimensiunea (n + 1) x (C + 1), unde fiecare element K[i][c] reprezintă profitul maxim ce poate fi obținut cu primele i obiecte și cu un rucsac de capacitate c. Deoarece matricea are n + 1 linii și C + 1 coloane, indecșii i și c pornesc de la 0.  
 Algoritmul utilizează două bucle for pentru a parcurge toate valorile posibile ale K. Prima buclă for parcurge toate valorile de la 0 până la n, în timp ce a doua buclă for parcurge toate valorile de la 0 până la C.  
 Dacă i sau c este 0, atunci nu se poate face nicio investiție, așadar profitul maxim este 0, iar matricea K[i][c] este inițializată cu 0.  
 Dacă valoarea unei acțiuni, reprezentată de actiuni\_val[i-1], este mai mică sau egală cu capacitatea rucsacului c, atunci există două opțiuni: (1) să cumpărați acțiunea, iar profitul maxim va fi dat de suma profitului obținut de la acea acțiune, profit[i-1], și profitul maxim obținut anterior, K[i-1][c-actiuni\_val[i-1]], sau (2) să nu cumpărați acea acțiune, iar profitul maxim rămâne profitul obținut anterior, K[i-1][c]. Deci, profitul maxim pentru i și c va fi dat de maximul dintre cele două opțiuni, adică K[i][c] <= maxx(profit[i-1]+K[i-1][c-actiuni\_val[i-1],K[i-1][c]).  
 În caz contrar, nu este posibilă achiziționarea acelei acțiuni, iar profitul maxim rămâne profitul obținut anterior, K[i-1][c].  
 Complexitatea în timp a algoritmului este de O(nC), deoarece trebuie să parcurgem toate obiectele (n) și să calculăm valoarea maximă pentru fiecare. Complexitatea spațială este de O(nC), deoarece avem nevoie de un tabel bidimensional K pentru a stoca valorile intermediare.

**Exemplu:**

Capitalul disponibil: C = 10  
Numărul de oferte disponibile: n = 4  
Valoarea acțiunilor pentru fiecare ofertă: actiuni\_val = [6, 3, 4, 2]  
Profitul estimat pentru fiecare acțiune: profit = [30, 14, 16, 9]  
 Atunci, putem crea un tabel cu dimensiunea (n+1) x (C+1), numit K, pentru a stoca valorile maxime ale profitului posibil. Primul rând și prima coloană a tabelului vor fi inițializate cu 0, deoarece nu avem oferte disponibile sau nu avem capital disponibil. Apoi, vom completa tabelul începând cu celula de pe coloana 1, rândul 1 și vom avansa spre dreapta și în jos. Fiecare celulă va conține profitul maxim posibil pentru capitalul respectiv și numărul de oferte.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 30 | 30 | 30 | 30 |
| 2 | 0 | 0 | 0 | 14 | 14 | 14 | 30 | 30 | 30 | 44 | 44 |
| 3 | 0 | 0 | 0 | 14 | 16 | 16 | 30 | 30 | 46 | 46 | 46 |
| 4 | 0 | 0 | 9 | 14 | 16 | 23 | 30 | 33 | 46 | 48 | 48 |

Profitul maxim posibil este 48, pe care îl putem obține investind în ofertele 1, 2 și 4.

**Algoritm 2 (Recursiv)**

**Recursion\_Solution(C,action\_val,profit,n)**

1. If n=0 or C=0 then
2. Return 0;
3. If action\_val[n-1]> C then
4. Return Recursion\_Solution(C, action\_val,profit,n-1)
5. Else
6. Return maxx(profit[n-1] + Recursion\_Solution(C-action\_val[n-1],action\_val,profit,n-1), Recursion\_Solution(C,action\_val,profit,n-1))

Codul prezentat este o implementare a algoritmului de tip recursiv pentru rezolvarea problemei de alegere a investiției care maximizează profitul, având la dispoziție un anumit capital și o serie de oferte cu acțiuni de valoare și profit estimat.  
 Acest algoritm utilizează recursivitatea pentru a explora toate posibilitățile de alegere a ofertelor și de a determina cea mai bună alegere. Practic, el începe prin a verifica cazurile de bază, adică când numărul de oferte sau capitalul disponibil este 0. În aceste cazuri, nu există nicio alegere și algoritmul returnează 0.  
 În caz contrar, algoritmul verifică dacă valoarea acțiunilor celei mai recente oferte este mai mare decât capitalul disponibil. Dacă acesta este cazul, ofertele anterioare sunt considerate pentru a determina cea mai bună alegere, fără a lua în considerare oferta curentă.  
 În cazul în care valoarea acțiunilor ofertei curente este mai mică sau egală cu capitalul disponibil, se calculează două valori: una pentru cazul în care oferta curentă este inclusă în portofoliu și alta pentru cazul în care nu este inclusă. În ambele cazuri, se apelează recursiv funcția pentru ofertele anterioare și se adaugă sau nu oferta curentă în portofoliu.  
 Complexitatea în timp a acestui algoritm este O(2^n), deoarece pentru fiecare ofertă există două posibilități: fie este inclusă în portofoliu, fie nu este inclusă. În plus, deoarece fiecare apel recursiv generează un nou nivel de stivă, complexitatea spațială este de asemenea O(n), unde n este numărul de oferte.  
 În general, algoritmul recursiv este mai puțin eficient decât cel dinamic prezentat anterior, deoarece necesită o cantitate mai mare de resurse și timp pentru a explora toate posibilitățile. Cu toate acestea, poate fi util în cazurile în care numărul de oferte este relativ mic și eficiența algoritmului nu este o problemă critică.

**Exemplu:**

Capitalul disponibil: C = 50  
Numărul de oferte disponibile: n = 3  
Valoarea acțiunilor pentru fiecare ofertă: actiuni\_val = [10, 20, 30]  
Profitul estimat pentru fiecare acțiune: profit = [30, 100, 120]

Vom aplica algoritmul recursiv pentru a găsi investiția care ne poate aduce profitul estimat maxim. Pentru aceasta, vom aplica funcția Recursion\_Solution(C, action\_val, profit, n) cu valorile corespunzătoare:  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 3)

Pasul 1:

Deoarece n ≠ 0 și C ≠ 0, vom trece la linia 3 a codului, unde se verifică dacă valoarea acțiunilor ofertei curente este mai mică sau egală cu capitalul disponibil C. În cazul acestui exemplu, 10 < 50, deci trecem la linia 6 a codului și aplicăm funcția maxx între două apeluri recursive:  
maxx(profit[2] + Recursion\_Solution(50-30, [10, 20, 30], [30, 100, 120], 2), Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 2))

Pasul 2:

Primul apel recursive din maxx se referă la adăugarea acțiunilor ofertei curente (Ofertă 3) la portofoliul nostru, iar al doilea apel recursive se referă la excluderea ofertei curente din portofoliu. Evaluând cele două apeluri recursive, obținem:  
Recursion\_Solution(20, [10, 20], [30, 100], 2) = 100  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 2) = 130  
 Deoarece 130 > 100, vom alege a doua opțiune, adică vom exclude ofertă 3 din portofoliu și vom evalua portofoliul format din primele două oferte. Deci, apelul recursiv se transformă în:  
 Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 2)

Pasul 3:

Trecem la linia 3 a codului, unde verificăm dacă valoarea acțiunilor ofertei curente este mai mică sau egală cu capitalul disponibil C. În cazul acestui exemplu, 20 < 50, deci trecem la linia 6 a codului și aplicăm funcția maxx între două apeluri recursive:  
maxx(profit[1] + Recursion\_Solution(50-20, [10, 20, 30], [30, 100, 120], 1), Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 1))  
  
 Pasul 4:

Primul apel recursive din maxx se referă la adăugarea acțiunilor ofertei curente (Ofertă 2) la portofoliul nostru, iar al doilea apel recursive se referă la excluderea ofertei curente din portofoliu. Evaluând cele două apeluri recursive, obținem:  
Recursion\_Solution(30, [10, 20], [30, 100], 1) = 100  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 1) = 0  
 Deoarece 100 > 0, vom alege a doua opțiune, adică vom exclude ofertă 2 din portofoliu și vom evalua portofoliul format din prima ofertă. Deci, apelul recursiv se transformă în:  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 1)

Pasul 5:

Trecem la linia 3 a codului, unde verificăm dacă valoarea acțiunilor ofertei curente este mai mică sau egală cu capitalul disponibil C. În cazul acestui exemplu, 10 < 50, deci trecem la linia 6 a codului și aplicăm funcția maxx între două apeluri recursive:  
maxx(profit[0] + Recursion\_Solution(50-10, [10, 20, 30], [30, 100, 120], 0), Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 0))

Pasul 6:

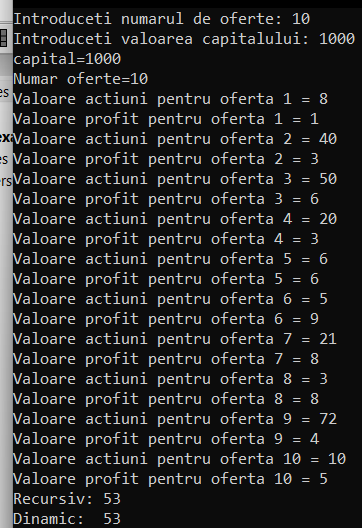
Primul apel recursive din maxx se referă la adăugarea acțiunilor ofertei curente (Ofertă 1) la portofoliul nostru, iar al doilea apel recursive se referă la excluderea ofertei curente din portofoliu. Evaluând cele două apeluri recursive, obținem:  
Recursion\_Solution(40, [20, 30], [100, 120], 0) = 120  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 0) = 0  
 Deoarece 120 > 0, vom alege a doua opțiune, adică vom exclude ofertă 1 din portofoliu și vom evalua portofoliul gol. Deci, apelul recursiv se transformă în:  
Recursion\_Solution(50, [10, 20, 30], [30, 100, 120], 0)

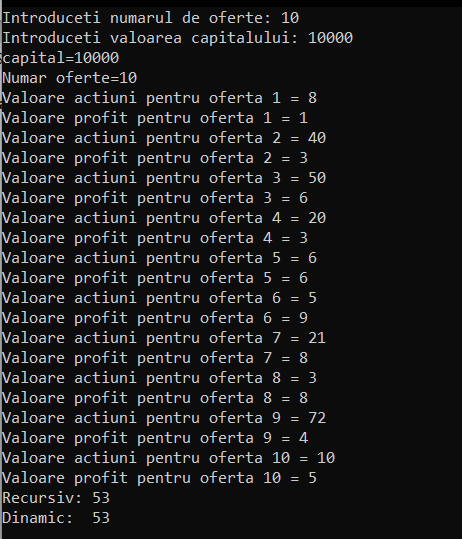
Pasul 7:

Deoarece n=0, vom ajunge la linia 2 a codului, unde se va returna 0. Deci, algoritmul recursiv ne-a indicat că investiția care ne poate aduce profitul estimat maxim este de 120.

Astfel, conform acestui exemplu, vom achiziționa acțiunile ofertei 2 (cu o valoare de 20), apoi pe cele ale ofertei 3 (cu o valoare de 30), iar portofoliul va avea un profit estimat de 120.

**Date experimentale:**





Observam ca prin ambele metode obținem aceleași rezultate.

**Rezultate si concluzii:**

Intre cei doi algoritmi exista diferențe semnificative de complexitate. In cazul primului algoritm complexitatea timpului este foarte mica, dar ocupa foarte multa memorie pentru generarea matricei. Al doilea algoritm care rezolva problema prin recursivitate este ineficient din punct de vedere al timpului, deoarece abordează toate cazurile posibile, dar din punct de vedere al spațiului este mai folositor.

In timpul testării algoritmilor am observat ca daca avem un număr foarte mare de oferte metoda recursiva calculează intr-un timp foarte lung soluția. Al doilea algoritm nu funcționează in cazul in care produsul dintre numărul de oferte si capital este mare deoarece nu putem genera o matrice atât de mare.

Aceasta problema permite multiple abordări de rezolvare, însă eu am ales sa aleg un algoritm eficient si unul mai puțin optimizat(brute force) pentru a semnaliza diferențele dintre acestea.

Timpi de execuție:

|  |  |  |  |
| --- | --- | --- | --- |
| Număr oferte | Capital | Timp alg\_1 | Timp alg\_2 |
| 10 | 100 | 3.853 | 11.143 |
| 20 | 500 | 3.588 | 3.480 |
| 30 | 1000 | 5.326 | 13.107 |
| 40 | 1500 | 3.161 | 15.209 |

Observam ca timpii de execuție nu depind doar de numărul ofertelor si de capital. Aceștia depind si de valorile profiturilor si acțiunilor.