*Problema rafinata a rucsacului*

*5 Iunie 2022*

***Student: Cocei Janina Constantina***

***Specializarea: Calculatoare Romana***

***Anul I***

***Grupa 1.1 B***

***Student: COCEI JANINA CONSTANTINA***

***Specializarea: Calculatoare Romana***

***Anul I***

***Grupa 1.1 B***

Cuprins

[I. Enuntul Problemei 3](#_Toc105335020)

[II. Algoritmii 3](#_Toc105335021)

[III. Date experimentale 5](#_Toc105335022)

[i. Proiectarea aplicatiei experimentale 6](#_Toc105335023)

[a) Structura de nivel inalt a aplicatiei 6](#_Toc105335024)

[b) Descrierea Multimii Datelor de intrare 6](#_Toc105335025)

[c) Descrierea Multimii Datelor de iesire 6](#_Toc105335026)

[d) Lista modulelor aplicatiei 7](#_Toc105335027)

[ii. Functiile aplicatiei 7](#_Toc105335028)

[ Organizarea datelor de iesire 8](#_Toc105335029)

[ Testarea datelor de iesire 9](#_Toc105335030)

[ Timpii de executie pentru date mici si mari 9](#_Toc105335031)

[ Observatii si concluzii 11](#_Toc105335032)

[Referinte 12](#_Toc105335033)

# Enuntul Problemei

Sa presupunem ca sunteti agent de investitii si ca dispuneti de un capital C. Sunt disponibile pentru investitii n oferte dintre care trebuie sa alegeti. Pentru fiecare oferta i sunt disponibile ai actiuni de valoare vi fiecare, ce va pot aduce un profit estimat pi pentru fiecare actiune achizitionata. Se cere sa determinati investitia care va poate aduce profitul estimat maxim. Se vor implementa doi algoritmi diferiti.

# Algoritmii

* ***Algoritmul 1( DINAMIC)***

*Dynamic\_programming\_capital\_discrete(oferte, no\_oferte, capital\_capacity)*

pentru actiuni\_iterator = 0, no\_oferte executa

{

pentru weight\_iterator = 0, capital\_capacity executa

{

daca actiuni\_iterator == 0 || weight\_iterator == 0

set\_matrix\_value(matrix, actiuni\_iterator, weight\_iterator, 0);

altfel daca (oferte[actiuni\_iterator - 1].pret \* oferte[actiuni\_iterator - 1].nr\_actiuni <= weight\_iterator)

{

int carry = get\_max(oferte[actiuni\_iterator - 1].profit \* oferte[actiuni\_iterator - 1].nr\_actiuni + get\_matrix\_value(matrix, actiuni\_iterator - 1, weight\_iterator - oferte[actiuni\_iterator - 1].pret \* oferte[actiuni\_iterator - 1].nr\_actiuni), get\_matrix\_value(matrix, actiuni\_iterator - 1, weight\_iterator));

set\_matrix\_value(matrix, actiuni\_iterator, weight\_iterator, carry);

}

altfel

set\_matrix\_value(matrix, actiuni\_iterator, weight\_iterator, get\_matrix\_value(matrix, actiuni\_iterator - 1, weight\_iterator));

}

}

capital\_value = get\_matrix\_value(matrix, matrix.no\_rows - 1, matrix.no\_cols - 1);

Problema ne propune sa alegem dintr-o lista de actiuni la care le cunoastem valoarea si profitul , actiunile care obtin suma valorilor cea mai mare, astfel incat sa ne folosim de tot capitalul. Functia de mai sus preia lista de actiuni, cu caracteristicile date, numarul de actiuni si capitalul. Functia trebuie sa genereze toate posibilitatile intr-un timp cat mai scurt, astfel incat la final, dupa parcurgerea tuturor obiectelor, obtinem rezultatul cerut.

Problema se foloseste de memorizare pentru a salva valorile fiecarei oferte pentru fiecare valoare pana la cea a capitalului. Astfel avem o matrice de dimensiuni (w+1)\*(n+1), unde w reprezinta valoarea capitalului iar n retine numarul de actiuni. Matricea este creata cu tipul de data struct a\_matrix unde retine numarul de coloane, numarul de linii si un vector de dimensiunea (w+1)\*(n+1). Pentru accesarea elementelor se folosesc functiile set\_matrix\_value() si get\_matrix\_value().

Algoritmul functioneaza astfel:

* Daca valoarea actiunii la care ne aflam este mai mare sau egala cu valoarea capitalului la care ne aflam atunci valoarea curenta este maximul dintre valoarea de pe oferta curenta si oferta trecuta, si valoarea ofertei trecute si valoarea curenta minus valoarea ofertei curente. Adica pentru fiecare oferta verifica daca valoarea ofertei curente + valoarea pana la oferta trecuta cu valoarea ramasa libera este mai mare decat valoarea pana la oferta trecuta cu valoarea curenta.
* In caz contrar, se pune valoarea pana la oferta curenta de pe valoarea curenta. Astfel la final se obtine pentru toate ofertele valorile maxime adunate pentru fiecare capacitate de valoare. Se afiseaza ultima valoare pentru toate obiectele, adica M[n-1][w-1], unde M este matricea generala.

Complexitatea acestui program este liniara, de ordin O (n\*w), la fel si memoria.

* ***Algoritmul 2 (GREEDY)***

void greedy\_capital\_discrete(oferte, no\_oferte, capital\_capacity){

int iterator;

int capital\_current\_capacity;

float capital\_value;

capital\_current\_capacity = capital\_capacity;

capital\_value = 0.0f;

pentru iterator = 0, no\_oferte executa {

float pret\_unitate = value\_per\_unit(oferte[iterator]);

daca pret\_unitate <= capital\_current\_capacity

{

int unitati\_cumparate = capital\_current\_capacity/pret\_unitate;

capital\_current\_capacity -= unitati\_cumparate/pret\_unitate;

capital\_value += unitati\_cumparate\*oferte[iterator].profit;

}

}

printf("\nProfitul maxim: %f", capital\_value);

}

# Date experimentale

Ca date de intrare, problema are nevoie de doi parametrii, aceia fiind capitalul disponibil si numarul de oferte, care urmeaza a fi cumparate.

In prima etapa a crearii programului, datele de intrare apartineau intervalului (1, 60), si executarea programului dureaza peste 50 de secunde cu oricare dintre algoritmi. Dupa ce am limitat numarul de pasi scrisi la 1000, am putut mari intervalul pana la 100, ın acest caz executia durand maxim 50-60 secunde, ın cel mai rau caz, iar fisierul de output are doar 60 Kb. Am creat un generator simplu care va genera un numar din intervalul (1, 10000) folosind, ın limbajul C functia rand().

# Proiectarea aplicatiei experimentale

# Structura de nivel inalt a aplicatiei

Aplicatia contine urmatoarele fisiere ın limbajul C: main.c, capital.c, capital.h, greedy.c, greedy.h conectate astfel:

# Descrierea Multimii Datelor de intrare

Datele de intrare pentru algoritm vor fi doua numere, care vor simboliza capitalul de care dispune agentul de vanzari si numarul de actiuni/oferte.

# Descrierea Multimii Datelor de iesire

Datele de iesire ale problemei cuprind in primul rand un mesaj initial, care va arata capitalul de care depinde investitorul. Urmeaza un alt mesaj text cu numarul de oferte disponibile, apoi matricea si pe ultimul rand un mesaj cu profitul maxim.

# Lista modulelor aplicatiei

In C, modulele sunt:

* main.c -modulul principal
* capital.c -modulul ce cuprinde majoritatea corpurilor functiilor utilizate
* greedy.c -modulul ın care se afla corpul functiei, care va calcula profitul maxim
* capitalt.h - modulul ce contine headerele functiilor din capital.c
* greedy.h modulul ce contine headerele functiei din greedy.c

# Functiile aplicatiei

* main.c

Pentru ambii algoritmi, acest modul contine doar deschiderea fisierelor unde se vor salva datele, apeluri de functii, initializari de structuri necesare/variabile si metoda prin care este cronometrat timpul de executie al algoritmului.

Pentru algoritmul dinamic ın main.c sunt apelate functiile generator\_oferte(), print\_oferte(), dynamic\_programming\_capital\_discrete().

Pentru algoritmul greedy ın main.c este apelata functia greedy\_capital\_discrete().

* capital.h si greedy.h

Aici se afla prototipurile functiilor folosite. Pentru algoritmul dinamic, tot aici (capital.h) se va gasi declaratia structurii *oferta*, ce va fi o stiva in care se vor stoca id-urile, pretul, profitul, dar ¸si numarul ofertelor.Dar si declaratia structurii *a\_matrix* , ce va fi o stiva in care se vor stoca liniile, coloanele si matricea.

* greedy.c

In acest modul se gasesc corpul functiilor care sorteaza si calculeaza profitul maxim prin metoda greedy.

* capital.c

Aici se gasesc corpurile majoritatii functiilor folosite in programe.

Functia *generator\_oferte()* primeste ca parametrii *structura oferta* si o variabila de tip int, no\_oferte, care reprezinta numarul de oferte. Dupa cum sugereaza si numele, aceasta functie genereaza ofertele precum si id-ul,numarul de actiuni, pretul si profitul acestora.

Functia *print\_oferte()* afiseaza in consola ofertele impreuna cu id-ul,numarul de actiuni, pretul si profitul acestora.

Functia *set\_matrix\_value()* are ca parametrii *structura a\_matrix* si trei variabile de tip integer*, row\_index, column\_index, element\_value.* Aceasta functie seteaza valorile din matrice.

Functia *print\_matrix()* primeste ca parametru structura *a\_matrix*. Dupa cum sugereaza si numele, aceasta afiseaza matricea in consola.

Functia *get\_max()* calculeaza maximul dintre doua valori intr-un mod recursiv.

Functia *value\_per\_unit()* este de tip float, aceasta calculeaza valoarea pe unitate a unei oferte.

Functia *dynamic\_programming\_capital\_discrete()* primeste ca parametrii structura oferte, si doua elemente de tip intreg, *no\_oferte* care reprezinta numarul de oferte si *capital\_capacity*, care reprezinta valoarea capitalului.Aceasta functie se foloseste de memorizare pentru a salva valorile fiecarei oferte pentru fiecare valoare pana la cea a capitalului. Astfel avem o matrice de dimensiuni (w+1)\*(n+1), unde w reprezinta valoarea capitalului iar n retine numarul de actiuni. Matricea este creata cu tipul de data struct a\_matrix unde retine numarul de coloane, numarul de linii si un vector de dimensiunea (w+1)\*(n+1). Pentru accesarea elementelor se folosesc functiile *set\_matrix\_value()* si *get\_matrix\_value().*

* *greedy.c*

Aici este intalnit cel de-al doilea algoritm, implementarea prin metoda greedy.

Functia *sort\_objects\_by\_value\_per\_unit()* sorteaza crescator ofertele.

Functia *greedy\_capital\_discrete()* primeste ca parametrii structura oferte si doua elemente de tip intreg, no\_oferte si capital\_capacity. Aceasta functie calculeaza prin metoda greedy profitul maxim.

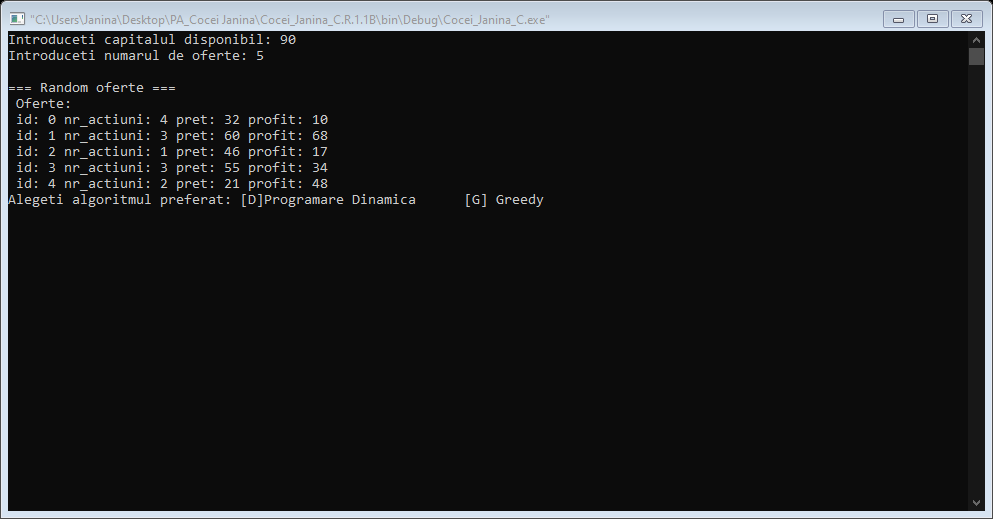
1. ***Rezultate***

Aici voi arata rezultatele testelor si anumite observatii pe baza implementarilor si ale executarii codului ın timpul testelor.

# Organizarea datelor de iesire

Pentru o simpla rulare, rezultatele sunt afisate ın consola.

Atunci cand se vor rula testele, vor fi afisate urmatoarele mesaje sugestive:



# Testarea datelor de iesire

Pentru a testa daca algoritmul calculeaza profitul corecte am folosit mai multe metode:

1. Pentru valorile mici ale capitalului si numarul de oferte am generat random valorile si profitul, dupa care am verificat manual profitul si toate s-au dovedit a fi corecte.
2. Pentru valori mai mari, doar am dat scroll prin fisier pentru a vedea ca nu exista erori ale profitului.

# Timpii de executie pentru date mici si mari

Am organizat compararea datelor timpilor de executie ın doua tabele: primul prin metoda dinamica, iar cel de-al doilea prin metoda greedy. Ambele metode sunt iterative.

Tabel comparare algoritm dinamic

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. test | Capital | Nr. oferte | Dinamic |
| 1 | 20 | 3 | 7.983 |
| 2 | 30 | 5 | 8.652 |
| 3 | 60 | 10 | 8.988 |
| 4 | 100 | 16 | 9.123 |
| 5 | 135 | 18 | 9.544 |
| 6 | 178 | 20 | 10.222 |
| 7 | 280 | 22 | 10.788 |
| 8 | 320 | 28 | 13.987 |
| 9 | 400 | 29 | 16.807 |
| 10 | 542 | 30 | 17.567 |
| Tabel comparare algoritm greedy | | | |
| Nr. test | Capital | Nr. oferte | Greedy |
| 1 | 20 | 3 | 7.886 |
| 2 | 30 | 5 | 11.455 |
| 3 | 60 | 10 | 11.634 |
| 4 | 100 | 16 | 11.988 |
| 5 | 135 | 18 | 12.439 |
| 6 | 178 | 20 | 12.984 |
| 7 | 280 | 22 | 13.345 |
| 8 | 320 | 28 | 15.654 |
| 9 | 400 | 29 | 16.952 |
| 10 | 542 | 30 | 17.654 |

# Observatii si concluzii

Observam ca algoritmii ın C, pentru amblele metode, sunt destul de rapizi, dar observam ca cu cat capitalul devine mai mare, cu atat timpul de executie creste.

Pe viitor, acest program poate fi dezvoltat, iar ın loc sa afiseze in consola ar putea sa salveze datele furnizate intr-un fisier, dar acesta este un proiect de viitor.

# Referinte

Preluat de pe https://en.wikipedia.org/wiki/Knapsack\_problem

Kernighan, D. R. (1978). *The C Programming Language. 2nd.* Prentice Hall.

Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein. (2009). *Introduction to Algorithms.* MIT Press, 3rd Edition.