**Bratasanu Anda Iulia – grupa 201**

1. **Programare dinamica (problema nr. 9)**

* **Enuntul problemei**Se consideră un rucsac având capacitatea G și n obiecte pentru care se cunosc greutățile și câștigurile asociate. Știind că orice obiect fie este încărcat complet în rucsac, fie nu este încărcat deloc, să se determine o modalitatea de încărcare a rucsacului astfel încât câștigul obținut să fie maxim.
* **Exemplul numeric**   
  Pentru greutatea G = 50 si numarul de obiecte n = 5 cu urmatoarele proprietati,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. obiect | 0 | 1 | 2 | 3 | 4 |
| Greutatea (g) | 10 | 20 | 40 | 30 | 5 |
| Castigul (c) | 10 | 10 | 15 | 2 | 20 |

=> Se va afisa castigul maxim de 40 (au fost alese obiectele cu nr. 0, 1 si 4).

* **Detalii despre program**Presupunem ca inaltimea, latimea si diametrul obiectelor nu au importanta pentru problema, in ghiozdan incapand obiectul/obiectele pana la capacitatea maxima a rucsacului (greutatea este singurul factor pe care il luam in considerare cand ne referim la ce obiecte pot sa intre in ghiozdan).  
    
  Mentionari: *Daca* – numarul de obiecte n = 0, programul returneaza valorea 0 (nu exista castig)  
   – castigul unui obiect este inexistent (0), raspunsul final nu este afectat.  
    
   – greutatea unui obiect este 0, castigul obiectului onsider va fi adaugat la castigul maxim

Citire si afisare:– se *citeste* n (nr de obiecte) si G (greutatea maxima) din fisierulproblema1.txt.in (pe prima linie)  
 – se aloca cu ajutorul operatorului “new” matricea a[n+1][G+1] pe care o vom folosi pentru a stoca selectia de obiecte care urmeaza a fi verificate si poate si modificate, daca castigul se modifica–sealoca memoria vectorilor g[n+1] si c[G+1] cu care citim proprietatile obiectelor (greutate si castig) de pe a doua, onsider, a treia linie din fisierulproblema1.txt.in (precum in exemplul din tabelul de mai sus)

–se *afiseaza*castigul maxim in consola/terminal.Algoritm pentru calcularea castigului maxim:  
*Elementele de prima linie a[0][cap] primesc valorea 0 (cap ia valori de la (0,G)).  
Folosim “cap” pentru a stoca capacitatea curenta folosita (prin adunarea si scaderea succesiva a obiectelor care intra si ies din selectie.*  
 – Algoritmul parcurge si verifica obiectele de pe o linie cu alta, luand in considerare doar ultimul rand inaintea celui pe care este onside (astfel memoria si timpul sunt optimizate in onsider cu metoda greddy, care ar compara constant valorile verificate cu cele din mutiplele etape anterioare).  
  
 – Conditia / Modul de selectie a obiectelor dupa valoare este:  
  
 a[i][cap] = max(a[i-1][cap]   
 , pentru I = (1,n) si cap = (0,G)  
 a[i-1][cap-g[i]]+p[i]   
   
  
 – Selectia se face intre varianta de deasupra (solutia anterioara) sau pe cea cu obiectul current. Astfel, la finalul parcurgerii, pe ultima pozitie se afla castigul maxim;

* **Codul sursa in limbajul c++**

#include <iostream>

#include <fstream>

using namespace std;

ifstream f("problema1.in.txt");

// n  = numarul de obiecte inregistrate

// G  = greutatea maxima a rucsacului

// (g[i], c[i]) = greutatea, castigul obiectului

int main()

{

    int n = 0, i = 0, j = 0, G = 0;

    f >> n; // Introducem din fisier valoarea lui nf

    f >> G; // Introducem din fisier valoarea lui G

    int linii = n + 1, coloane = G + 1;

    int\*\* a = new int\* [linii];

    for (int i = 0; i < linii; ++i)  // Alocam matricea a[n + 1][G + 1]

        a[i] = new int[coloane];

    int\* g = new int[linii]; int\* c = new int[coloane];  // Alocam memoria vectorilor g[n+1] si c[G+1]

    for (i = 1; i <= n; i++) { // Citire date din fisier, prin vector (greutatiile obiectelor)

        f >> g[i];

    }

    for (i = 1; i <= n; i++) { // analog castigurile obiectelor

        f >> c[i];

    }

    // pe linia 0, capacitatea e 0  pentru ca nu avem obiecte inca

    for (int cap = 0; cap <= G; ++cap) { // cap = capacitatea folosita pana la pasul  onside

        a[0][cap] = 0;

    }

    // incepem popularea matricii si verificarea obiectelor

    for (i = 1; i <= n; ++i) {

        for (int cap = 0; cap <= G; ++cap) {  // parcurgem matricea pe coloane

            a[i][cap] = a[i - 1][cap]; // preluam castigul de la pasul anterior

            if (cap - g[i] >= 0) { // verificam daca exista loc pentru obiect in raport cu capacitatea curenta

                int castigmaxim = a[i - 1][cap - g[i]] + c[i]; // calculam castigul adaugand obiectul i

                a[i][cap] = max(a[i][cap], castigmaxim); // memoram varianta cu profit mai mare (profitul de dinainte sau combinatia cu obiectul i)

            }

        }

    }

    cout << "Castigul maxim este: " << a[n][G] << endl;

    return 0;

}

* **Incadrarea solutiei in tehnica de programare dinamica***Initial, am incercat sa construiesc un algoritm care calculeaza “profitul” fiecarui obiect (castig / greutate), insa, acesta presupunea metoda greddy.*  
  Asadar, pentru incadrarea problemei in tehnica programarii dinamice am folosit urmatoarele proceduri:  
    – identificarea structurii utilizate in caracterizarea solutiei optime   
   – alocarea dinamica a memoriei  
   – tinem cont de exceptii, verificam si preferam a folosi o metoda care sa nu afecteze algoritmul de baza.
* **Complexitatea solutiei**– Complexitate temporala : O(nG)

– Complexitate sptiala : O(nG)

1. **Metoda Greddy (problema nr. 10)**

**• Enuntul problemei**Se onsider o listă formată din n proiecte pe care trebuie să le execute o echipă de programatori din Departamentul Dev. Pentru fiecare proiect se cunoaște termenul său limită, precum și bonusul pe care îl va obține echipa dacă proiectul este finalizat la timp în caz contrar, bonusul va fi 0). Durata de executare a oricărui proiect se onsider egală cu o zi. Ajutați-vă echipa scriind un program care să determine o modalitate de planificare a executării proiectelor astfel încât bonusul obținut să fie maxim!

• **Exemplul numeric**  
Pentru numarul de proiecte n = 5 si urmatoarele proprietati ale acestora,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nr. proiect | 0 | 1 | 2 | 3 | 4 |
| Termenul limita | 3 | 1 | 2 | 2 | 3 |
| Bonusurile | 5 | 10 | 10 | 1 | 10 |

=> Se va afisa bonusul maxim de 30.

• **Detalii despre program**  
Algoritmul incepe cautarea termenilor limita de la cel mai mare, pana la valorea 1 (care reprezinta prima zi), apoi, selecteaza pentru fiecare zi, proiectul cu bonusul maxim din ziua respectiva sau una mai intarziata.  
Dupa ce bonusul unui proiect a fost adaugat la suma, variabila termenului limita este schimbata in -1, a.i. algoritmul sa nu il selecteze de mai multe ori.  
Citirea datelor se face din fisier.  
  
• **Codul sursa in limbajul c++**

#include <iostream>

#include <fstream>

using namespace std;

ifstream f("problema2.in.txt");

int main()

{

    int n, t[101], b[101], i, poz, max\_t = 0, min\_t, max\_b, nr\_pr, sum = 0;

    f >> n;

    for (i = 0; i < n; i++) {

        f >> t[i];

        if (t[i] > max\_t) max\_t = t[i]; // calculam maximul de t

    }

    for (i = 0; i < n; i++) f >> b[i];

    for (i = max\_t; i > 0; i--) { // incepem cautarea de la ultimul termen limita la prima zi

        max\_b = 0;  // ii dam maximului valorea 0 inainte sa parcurga proiectele pe zile

        for (int j = 0; j < n; j++) {

            if (i >= t[j]) { // conditia de selectie

                if (b[j] > max\_b) { //

                    max\_b = b[j];

                    poz = j; //salvam pozitia proiectului pe care l-am selectat a fi cel mai profitabil

                }

            }

        }

        if (max\_b) { // daca valorea exista (a fost schimbata)

            sum+= max\_b;

            t[poz]=-1; // atribuim valorea -1 proiectui pe care l-am adaugat la suma

        }

    }

    cout << sum;

    return 0;

}

• **Incadrarea solutiei in metoda Greddy**Programul se incadreaza in metoda Greddy, deoarece, la fiecare iteratie face alegerea optima dintre cele posibile (termenul limita >= termenul limita maxim - numarul iteratiei) avand in vedere bonusul maxim pe care il poate obtine; selecteaza cel mai bun element.   
  
• **Complexitatea solutiei**  
Programul are complexitatea:  
- O(n) \*max\_t (temporal)   
- O(n) (spatial)