TEHNICA DE PROGRAMARE "GREEDY" (continuare)

1. Problema rucsacului (varianta continuă/fracţionară)

Considerăm un rucsac având capacitatea maximă G şi n obiecte $O_1, O_2, \dots, O_n$ pentru care cunoaştem greutăţile lor $g_1, g_2, \dots, g_n$ şi câştigurile $c_1, c_2, \dots, c_n$ obţinute prin încărcarea lor completă în rucsac. Ştiind faptul că orice obiect poate fi încărcat şi fracţionat (doar o parte din el), să se determine o modalitate de încărcare a rucsacului astfel încât câştigul total obţinut să fie maxim. Dacă un obiect este încărcat fracţionat, atunci vom obţine un câştig direct proporţional cu fracţiunea încărcată din el (de exemplu, dacă vom încărca doar o treime dintr-un obiect, atunci vom obţine un câştig egal cu o treime din câştigul integral asociat obiectului respectiv).

Deoarece dorim să găsim o rezolvare de tip Greedy pentru varianta fracţionară a problemei rucsacului, vom încerca să încărcăm obiectele în rucsac folosind unul dintre următoarele criterii:

a) în ordinea descrescătoare a câştigurilor integrale (cele mai valoroase obiecte ar fi primele încărcate);
b) în ordinea crescătoare a greutăţilor (cele mai mici obiecte ar fi primele încărcate, deci am încărca un număr mare de obiecte în rucsac);
c) în ordinea descrescătoare a greutăţilor.

Analizând cele 3 criterii propuse mai sus, putem găsi uşor contraexemple care să dovedească faptul că nu vom obţine o soluţii optime. De exemplu, criteriul c) ar putea fi corect doar presupunând faptul că, întotdeauna, un obiect cu greutate mare are asociat şi un câştig mare, ceea ce, evident, nu este adevărat! În cazul criteriului a), considerând $G = 10$ kg şi 3 obiecte având câştigurile $(100, 90, 80)$ RON şi greutăţile $(10, 5, 5)$ kg, vom încărca în rucsac primul obiect (deoarece are cel mai mare câştig integral) şi nu vom mai putea încărca niciun alt obiect, deci câştigul obţinut va fi de 100 RON. Totuşi, câştigul maxim de 170 RON se obţine încărcând în rucsac ultimele două obiecte! În mod asemănător (de exemplu, modificând câştigurilor obiectelor anterior menţionate în $(100, 9, 8)$ RON) se poate găsi un contraexemplu care să arate faptul că nici criteriul b) nu permite obţinerea unei soluţii optime în orice caz.

Se poate observa faptul că primele două criterii nu conduc întotdeauna la soluţia optimă deoarece ele iau în considerare fie doar câştigurile obiectelor, fie doar greutăţile lor, deci criteriul corect de selecţie trebuie să le ia în considerare pe ambele. Intuitiv, pentru a obţine un câştig maxim, trebuie să încărcăm mai întâi în rucsac obiectele care sunt cele mai "eficiente", adică au un câştig mare şi o greutate mică. Această "eficienţă" se poate cuantifica prin intermediul *câştigului unitar* al unui obiect, adică prin raportul $u_i = c_i / g_i$.

Algoritmul Greedy pentru rezolvarea variantei fracţionare a problemei rucsacului este următorul:

- sortăm obiectele în ordinea descrescătoare a câştigurilor unitare;
- pentru fiecare obiect testăm dacă încape integral în spaţiul liber din rucsac, iar în caz afirmativ îl încărcăm complet în rucsac, altfel calculăm fracţiunea din el pe care

trebuie să o încărcăm astfel încât să umplem complet rucsacul (după încărcarea oricărui obiect, actualizăm spațiul liber din rucsac și câștigul total);
- algoritmul se termină fie când am încărcat toate obiectele în rucsac (în cazul în care $g_1 + g_2 + \cdots + g_n \leq G$), fie când nu mai există spațiu liber în rucsac.

De exemplu, să considerăm un rucsac în care putem să încărcăm maxim $G = 53$ kg și $n = 7$ obiecte, având greutățile $g = (10, 5, 18, 20, 8, 40, 20)$ kg și câștigurile integrale $c = (30, 40, 36, 10, 16, 30, 20)$ RON. Câștigurile unitare ale celor 7 obiecte sunt $u = \left(\frac{30}{10}, \frac{40}{5}, \frac{36}{18}, \frac{10}{20}, \frac{16}{8}, \frac{30}{40}, \frac{20}{20}\right) = (3, 8, 2, 0.5, 2, 0.75, 1)$, deci prin sortarea descrescătoare a obiectelor în funcție de câștigul unitar vom obține următoarea ordine a lor: $O_2, O_1, O_3, O_5, O_7, O_6, O_4$. Prin aplicarea algoritmului Greedy prezentat anterior asupra acestor date de intrare, vom obține următoarele rezultate:

Obiectul curent	Fracțiunea încărcată din obiectul curent	Spațiul liber în rucsac	Câștigul total
—	—	53	0
O_2: $c_2 = 40, g_2 = 5 \leq 53$	1	53 – 5 = 48	0 + 40 = 40
O_1: $c_1 = 30, g_1 = 10 \leq 48$	1	48 – 10 = 38	40 + 30 = 70
O_3: $c_3 = 36, g_3 = 18 \leq 38$	1	38 – 18 = 20	70 + 36 = 106
O_5: $c_5 = 16, g_5 = 8 \leq 20$	1	20 – 8 = 12	106 + 16 = 122
O_7: $c_7 = 20, g_7 = 20 > 12$	12/20 = 0.6	0	122 + 0.6*20 = 134

În concluzie, pentru a obține un câștig maxim de 134 RON, trebuie să încărcăm integral în rucsac obiectele O_2, O_1, O_3, O_5 și o fracțiune de $0.6 = \frac{3}{5}$ din obiectul O_7.

În continuare, vom prezenta implementarea în limbajul C++ a algoritmului Greedy pentru rezolvarea variantei fracționare a problemei rucsacului:

```
#include<iostream>
#include<iomanip>
#include<fstream>
#include<algorithm>

using namespace std;

//structura Obiect permite memorarea informațiilor despre un obiect
//ID-ul unui obiect este numărul său de ordine inițial
struct Obiect
{
    float castig, greutate;
    int ID;
};

//funcție comparator utilizată pentru a sorta un tablou cu elemente
//de tip Obiect în ordinea descrescătoare a câștigurilor unitare
```

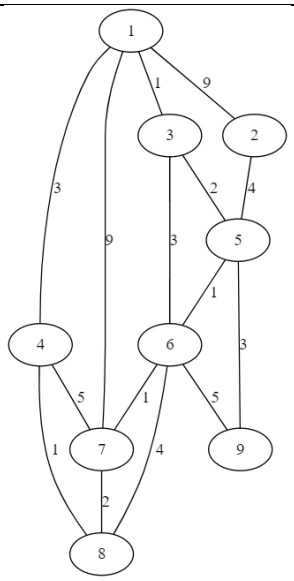
```
bool cmpObiecte(Obiect a, Obiect b)
{
    return a.castig/a.greutate > b.castig/b.greutate;
}

int main()
{
    //n = numărul de obiecte
    int i, n;
    //G = capacitatea maximă a rucsacului
    //total = câștigul total (maxim)
    //p = fracțiunea care se va încărca din ultimul obiect care
    //poate fi selectat pentru a umple complet rucsacul
    float G, total, p;
    //ob = obiectele date
    Obiect *ob;

    //datele de intrare se vor citi din fișierul text "obiecte.txt"
    //care conține pe prima linie numărul n de obiecte, pe
    //următoarele n linii sunt informațiile despre cele n obiecte
    //sub forma "greutate câștig", iar pe ultima linie se găsește
    //capacitatea maximă G a rucsacului
    ifstream fin("obiecte.txt");
    fin >> n;
    ob = new Obiect[n];
    for(i = 0; i < n; i++)
    {
        fin >> ob[i].greutate >> ob[i].castig;
        ob[i].ID = i+1;
    }
    fin >> G;
    fin.close();

    //sortam obiectele descrescător în funcție de câștigul unitar
    sort(ob, ob + n, cmpObiecte);

    //inițializăm câștigul total (maxim) cu 0
    total = 0;
    for (i = 0; i < n; i++)
        //dacă obiectul curent ob[i] încape complet în rucsac,
        //îl încărcăm, după care actualizăm câștigul total și
        //spațiul liber din rucsac
        if (ob[i].greutate <= G)
        {
            cout << "Obiectul " << ob[i].ID << " -> 100%" << endl;
            total = total + ob[i].castig;
            G = G - ob[i].greutate;
        }
        //dacă obiectul curent ob[i] nu încape complet în rucsac,
```

```
            //calculăm fracțiunea din el pe care trebuie să o încărcăm
            //pentru a umple complet rucsacul, actualizăm câștigul
            //total și algoritmul se termină
            else
            {
                p = G / ob[i].greutate;
                cout << "Obiectul " << ob[i].ID << " -> "
                     << setprecision(2) << p*100 << "%" << endl;
                total = total + p*ob[i].castig;
                break;
            }

        //afișăm câștigul total (maxim) pe care l-am obținut
        cout << "Castig total: " << fixed << setprecision(2)
             << total << endl;

        delete []ob;

        return 0;
    }
```

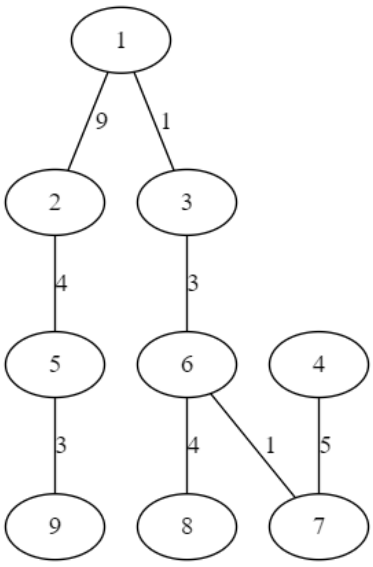
Citirea datelor de intrare are complexitatea $\mathcal{O}(n)$, sortarea are complexitatea $\mathcal{O}(n \log_2 n)$, selectarea și încărcarea obiectelor în rucsac are cel mult complexitatea $\mathcal{O}(n)$, iar afișarea câștigului maxim obținut $\mathcal{O}(1)$, deci complexitatea algoritmului este $\mathcal{O}(n \log_2 n)$.

2. Determinarea unui arbore parțial de cost minim (algoritmul lui Kruskal)

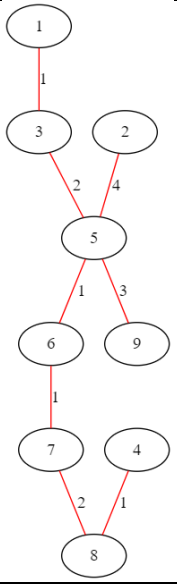
Considerăm un graf neorientat ponderat conex G (i.e., muchiile au asociate costuri). Să se determine un arbore parțial de cost minim.



Graf ponderat



Arbore parțial de cost 30



Arbore parțial de cost minim 15

Un *arbore* este un graf conex şi aciclic.

Teoremă: Orice arbore cu $n \geq 1$ vârfuri are $n - 1$ muchii.

Un *arbore parţial* este un arbore care conţine toate cele n vârfuri ale unui graf, deci are $n - 1$ muchii. Altfel zis, un arbore parţial al unui graf este un graf care aceleaşi vârfuri şi un număr minim de muchii astfel încât să rămână conex, precum şi un număr maxim de muchii astfel încât să fie aciclic. Practic, un arbore parţial de cost minim reprezintă un graf conex care se obţine eliminând un număr maxim de muchii din graful iniţial, este aciclic şi, în plus, costul său este minim (i.e., orice alt arbore parţial al grafului respectiv are un cost cel puţin egal cu al său).

Costul unui arbore parţial este suma costurilor muchiilor sale.