**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**RAPORT**

Lucrarea de laborator nr.3

*la Analiza și Proiectarea Algoritmilor*

A efectuat:

st. gr. TI-216 Roșca Dorin

A verificat : Lisnic Inga

Chişinău - 2022

**Lucrare de laborator nr 3**

**Tema:** Algoritmi greedy

**Scopul lucrării**:

1. Studierea tehnicii greedy.

2. Analiza şi implementarea algoritmilor greedy.

**Rezumat succint la tema lucrării de laborator:**

Tehnica greedy Algoritmii greedy (greedy = lacom) sunt în general simpli şi sunt folosiţi la probleme de optimizare, cum ar fi: să se găsească cea mai bună ordine de executare a unor lucrări pe calculator, să se găsească cel mai scurt drum într-un graf etc. În cele mai multe situaţii de acest fel avem:

• o mulţime de candidaţi (lucrări de executat, vârfuri ale grafului etc);

• o funcţie care verifică dacă o anumită mulţime de candidaţi constituie o soluţie posibilă, nu neapărat optimă, a problemei;

• o funcţie care verifică dacă o mulţime de candidaţi este fezabilă, adică dacă este posibil să completăm această mulţime astfel încât să obţinem o soluţie posibilă, nu neapărat optimă, a problemei;

• o funcţie de selecţie care indică la orice moment care este cel mai promiţător dintre candidaţii încă nefolosiţi;

• o funcţie obiectiv care dă valoarea unei soluţii (timpul necesar executării tuturor lucrărilor într-o anumită ordine, lungimea drumului pe care l-am găsit etc); aceasta este funcţia pe care urmărim să o optimizăm (minimizăm/maximizăm). Pentru a rezolva problema de optimizare, se caută o soluţie posibilă care să optimizeze valoarea funcţiei obiectiv. Un algoritm greedy construieşte soluţia pas cu pas. Iniţial, mulţimea candidaţilor selectaţi este vidă. La fiecare pas, se adaugă acestei mulţimi cel mai promiţător candidat, conform funcţiei de selecţie.

Dacă, după o astfel de adăugare, mulţimea de candidaţi selectaţi nu mai este fezabilă, se elimină ultimul candidat adăugat; acesta nu va mai fi niciodată considerat. Dacă, după adăugare, mulţimea de candidaţi selectaţi este fezabilă, ultimul candidat adăugat va rămâne de acum încolo în ea. De fiecare dată când se lărgeşte mulţimea candidaţilor selectaţi, se verifică dacă această mulţime nu constituie o soluţie posibilă a problemei. Dacă algoritmul greedy funcţionează corect, prima soluţie găsită va fi totodată o soluţie optimă a problemei.

**Codul programului in C**

**Algoritmul Kruskal:**

**#include** <iostream>

**using** **namespace** std;

**typedef** **struct**

{

    int a, b, costul;// *Vectorul este format din a-varful1, b-varful2 si costul de pe muchia lor*

} muchie;

int viz[30], n, m, c;

muchie varf[30], aux;

// *cititm datele problemei sub forma: varf1 -> varf2 – cost*

void **citire**()

{

    int i;

    cout << "Introduceti numarul de varfuri: " << endl;

    cin >> n;

    cout << "Introduceti numarul de muchii: " << endl;

    cin >> m;

**for** (i = 1; i **<=** m; i++)

    {

        cout << i <<". "; cin >> varf[i]**.**a >> varf[i]**.**b >> varf[i]**.**costul;

    }

}

// *Sortam muchiile in functie de costul lor- Partea „Gready” din algoritm*

void **sortare**()

{

    int k = 0, i;

**while** (k **==** 0)

    {

        k = 1;

**for** (i = 1; i **<=** m - 1; i++)

        {

**if** (varf[i]**.**costul **>** varf[i + 1]**.**costul)

            {

                aux = varf[i + 1];

                varf[i + 1] = varf[i];

                varf[i] = aux;

                k = 0;

            }

        }

    }

}

void **kruskal**()

{

    int i, j, k;

    i = 1;

**for** (k = 1; k **<=** n - 1; k++)

    {// *verificam sa nu vizitam varfuri deja vizitate*

**while** (viz[varf[i]**.**a] **==** viz[varf[i]**.**b] **&&** viz[varf[i]**.**a] **!=** 0)

            i++;

// *calculam costul total al arborelui de cost minim*

        c += varf[i]**.**costul;

// *afisam cele 2 varfuri*

        cout << varf[i]**.**a << " " << varf[i]**.**b << "\n";

// *daca cele 2 varfuri nu sunt vizitate atunci le vizitam si le salvam o valoare din cele 2 //in „viz”*

**if** (viz[varf[i]**.**a] + viz[varf[i]**.**b] **==** 0)

            viz[varf[i]**.**a] = viz[varf[i]**.**b] = varf[i]**.**a;

// *daca 1 din cele 2 valori este 0 o completam cu valoarea celeilalte*

**else**

// *daca ambele valori fac parte deja dintr-un arbore secundar, modificam toate valorile*

// *lui cu valorile celuilalt; practic legam cei 2 arbori*

**if** (viz[varf[i]**.**a] \* viz[varf[i]**.**b] **==** 0)

                viz[varf[i]**.**a] = viz[varf[i]**.**b] = viz[varf[i]**.**a] + viz[varf[i]**.**b];

**else**

            {

**for** (j = 1; j **<=** n; j++)

**if** (viz[j] **==** viz[varf[i]**.**a] **&&** j **!=** varf[i]**.**a)

                        viz[j] = viz[varf[i]**.**b];

                viz[varf[i]**.**a] = viz[varf[i]**.**b];

            }

        i++;

    }

    cout <<"costul minim este:" << c;

}

int **main**()

{

**citire**();

**sortare**();

**kruskal**();

}

**Algoritmul Prim:**

**#include** <iostream>

**using** **namespace** std;

int nr\_vf, nr\_eg;

**typedef** **struct**

{

    int beg, end;

    double cost;

} muchie;

muchie \*M;

int \*APM, \*za;

void **init**()

{

    int i;

    cout << "Introduceti nr. virfurilor: ";

    cin >> nr\_vf;

    cout << "Introduceti nr. arcelor: ";

    cin >> nr\_eg;

    APM = **new** int[nr\_vf - 1];

    M = **new** muchie[nr\_eg];

    cout << "\nIntroduceti arcul si costul:\n";

**for** (int i = 0; i **<** nr\_eg; ++i)

    {

        cin >> M[i]**.**beg >> M[i]**.**end >> M[i]**.**cost;

        M[i]**.**beg--;

        M[i]**.**end--;

    }

    za = **new** int[nr\_vf];

**for** (i = 0; i **<** nr\_vf; i++)

        za[i] = 0;

}

int **cut\_min**()

{

    int rm;

    double q = 1.E15;

**for** (int i = 0; i **<** nr\_eg; i++)

**if** (za[M[i]**.**beg] ^ za[M[i]**.**end])

**if** (M[i]**.**cost **<** q)

            {

                rm = i;

                q = M[i]**.**cost;

            }

**return** rm;

}

void **apm\_Prim**(int start)

{

    za[start] = 1;

**for** (int i = 0; i **<** nr\_vf - 1; i++)

    {

        int rm = **cut\_min**();

        APM[i] = rm;

**if** (za[M[rm]**.**beg])

            za[M[rm]**.**end] = 1;

**else**

            za[M[rm]**.**beg] = 1;

    }

}

int **main**()

{

    double cost\_apm = 0;

    int i;

**init**();

    cout << "\nIntroduceti varful de plecare: ";

    cin >> i;

**apm\_Prim**(i - 1);

    cout << "\n-------------------------------\n Rezultatele:\n";

**for** (i = 0; i **<** nr\_vf - 1; i++)

    {

        int rm = APM[i];

        cout << (M[rm]**.**beg + 1) << " - " << (M[rm]**.**end + 1) << "\tCost: " << M[rm]**.**cost << endl;

        cost\_apm += M[rm]**.**cost;

    }

    cout << "\nCostul minim = " << cost\_apm << '\n';

**delete**[] APM;

**delete**[] M;

**delete**[] za;

}

**Concluzie**

Efectuînd lucrarea dată am studiat algoritmii greedy.In special algoritmul Kruskal si Prim.Prin intermediul acestora ,am aflat costul minim al unui al unui graf: se observa :

1.Algoritmul Kruskal sorteaza la inceput costul fiecarei muchii iar apoi,in baza tabloului obtinut se creeaza arborele partial in ordine crescatoare a costului,se omit muchiil,varful carora se repeta.

2.Algoritmul Prim permite alegerea unui punct de plecareal grafului si in baza acestuia se creeaza arborele partial