Universitatea de Stat din Chisinau, Facultatea de Matematica si Informatica

LUCRARE DE LABORATOR NR.1

La disciplina: Matematica Discreta Algebra Geometrica

Tema: Parcurgerea Grafurilor

Efectuat de: Lupascu Iulii

Grupa: IA2402 (subgrupa I)

Coordonator: Niculita Angela

**Parcurgerea în adâncime (DFS)**

Parcurgerea în adâncime reprezintă explorarea “naturală” a unui graf neorientat. Este foarte asemănătoare cu modul în care un turist vizitează un oraș în care sunt obiective turistice (vârfurile grafului) și căi de acces între obiective (muchiile). Vizitarea orașului va avea loc din aproape în aproape: se pleacă de la un obiectiv de pornire, se continuă cu un obiectiv învecinat cu acesta, apoi unul învecinat cu al doilea, etc.

Parcurgerea în adâncime se face astfel:

* Se începe cu un vârf inițial x, care este în acest moment **vârf curent**.
* Vârful x se vizitează. Se determină primul său vecin nevizitat y al lui x, care devine vârf curent.
* Apoi se vizitează primul vecin nevizitat al lui y, şi aşa mai departe, mergând în adâncime, până când ajungem la un vârf care nu mai are vecini nevizitați. Când ajungem într-un astfel de vârf, ne întoarcem la “părintele” acestuia – vârful din care am ajuns în acesta.
* Dacă acest vârf mai are vecini nevizitați, alegem următorul vecin nevizitat al său și continuam parcurgerea în același mod.
* Dacă nici acest vârf nu mai are vecini nevizitați, revenim în vârful său părinte și continuăm în același mod, până când toate vârfurile accesibile din vârful de start sunt vizitate.

**Observație**

Dacă graful nu este conex, nu se vor vizita toate vârfurile.

**Codul**

```#include <iostream>

using namespace std;

class Graf {

public:

int nrnoduri;

int matrice[50][50];

bool vizitat[50];

Graf(int n) {

    nrnoduri=n;

    for(int i=0;i<nrnoduri;i++) {

        for(int j=0; j<nrnoduri; j++) {

            matrice[i][j]=0;

        }

    }

}

void adaugaMuchie(int v, int u) {

    matrice[u][v]=1;

    matrice[v][u]=1;

}

void dfs(int first) {

    for(int i=0;i<nrnoduri;i++) {

        vizitat[i]=false;

    }

    dfsfunc(first);

}

void afiseazaMatricea() {|

    cout << "Matricea de adiacenta este: \n";

    for (int i = 0; i < nrnoduri; i++) {

        for (int j = 0; j < nrnoduri; j++) {

            cout << matrice[i][j] << " ";

        }

    cout << endl;

}

}

private:

void dfsfunc(int nod) {

    vizitat[nod]=true;

    cout << nod << " ";

    for(int i=0;i<nrnoduri; i++) {

        if(matrice[nod][i]==1 && !vizitat[i]) {

            dfsfunc(i);

        }

    }

}

};

int main() {

    int n, a, b;

    cout << "Introdu numarul de noduri: " << endl;

    cin >> n;

    Graf Graf(n);

    for(int i=0; i<n;i++) {

        cout << "Pentru " << i+1 << "-a muchie: ";

        cin >> a >> b;

        Graf.adaugaMuchie(a, b);

    }

Graf.afiseazaMatricea();

    cout << "dfs:";

    Graf.dfs(0);

    return 0;

}```

Изображение выглядит как линия

Контент, сгенерированный ИИ, может содержать ошибки. Pentru oricare graf, programul incepe cu cel mai mic varf(cu indexul ‘0’). Si deja apoi parcurge toate varfurile, prioritate avand cele cu indecele mai mic.

In public se defineste clasa “Graf”, in care se afla functiile de formare a grafului, de formare a muchiilor, deformare si afisare a matricei de adiacenta, si de atribuire tuturor varfurilor starea “false”.

In private se afla functia de compilare a programului, care parcurge toate varfurile si le afiseaza.

La inceput se introduce numarul maxim de muchii, apoi manual se introduce varfurile adiacente pentru fiecare muchie, care este formata.

Si apoi se afiseaza rezultatul, si anume matricea de adiacenta si sirul varfurilor in ordinea in care au fost parcurse.

Pentru graful de mai sus se va introduce si se va afisa urmatorul:

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.