Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.2

*la Matematica Discretă*

Tema: ALGORITMUL DE CĂUTARE ÎN ADÂNCIME

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Popovici Nadejda

Chişinău 2022

1. **SCOPUL LUCRĂRII:**

* Studierea algoritmilor de căutare în graf şi a diferitor forme de păstrare şi prelucrare a datelor.
* Elaborarea procedurii de căutare în adâncime.

**2. NOTE DE CURS**

Operaţiile de bază cu firele de aşteptare:

* Formarea unui FA vid;
* Verificare dacă FA nu este vid;
* Alegerea primului element cu eliminarea lui din FA;
* Introducerea unei valori noi în calitate de ultim element al FA.

Operaţiile de bază cu stivele sunt următoarele:

* Formarea unei stive vide;
* Verificare la vid;
* Alegerea elementului din topul stivei cu sau fără eliminare;
* Introducerea unui element nou în topul stivei.

Operaţiile de bază pentru arbori vor fi:

* Formarea unui arbore trivial;
* Alegerea sau înlocuirea rădăcinii arborelui;
* Alegerea sau înlocuirea listei rădăcinilor subarborilor;
* Operaţiile de bază care sunt valabile pentru liste.

**Căutare în adâncime**

La căutarea în adâncime (parcurgerea unui graf în sens direct, în preordine) vârfurile grafului vor fi vizitate în conformitate cu următoarea procedură recursivă:

*mai întâi va fi vizitată rădăcina arborelui q, apoi, dacă rădăcina arborelui nu este frunză - pentru fiecare fiu p al rădăcinii q ne vom adresa recursiv procedurii de parcurgere în adâncime pentru a vizita vârfurile tuturor subarborilor cu rădăcina p ordonate ca fii ai lui q.*

**3. SARCINA DE BAZĂ**

1. Elaboraţi procedura căutării în adâncime într-un graf arbitrar;
2. Elaboraţi un program cu următoarele posibilităţi:

* introducerea grafului în calculator,
* parcurgerea grafului în adâncime,
* vizualizarea rezultatelor la display şi imprimantă.

**4. CODUL PROGRAMULUI**

#include <iostream>

#include <list>

#include <algorithm>

using namespace std;

class Graph

{

private:

int V;

list<int> \*adj;

void DFS\_helper(int s, bool \*visited);

bool isPresent(int n, int list\_i)

{

return (std::find(adj[list\_i].begin(), adj[list\_i].end(), n) != adj[list\_i].end());

};

public:

Graph(int v);

bool isOverSize(int n)

{

return (n >= V ? true : false);

};

void addEdge(int v, int w);

void introduce();

void print();

void DFS(int s);

};

Graph::Graph(int v)

{

V = v;

adj = new list<int>[v];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w);

}

void Graph::introduce()

{

for (auto i = 0; i < V; i++)

{

int data = 0;

cout << "Ce varfuri se unesc cu varful " << i << "? (-1 pentru a continua)" << endl;

while (data != -1)

{

while (true)

{

cin >> data;

if (data < -1 || data > V - 1)

{

cout << "Valoare invalida, introduceti din nou" << endl;

}

else

break;

}

if (data != -1 && !isPresent(data, i))

addEdge(i, data);

else if (isPresent(data, i))

{

cout << "Elementul dat este deja prezent, introduceti din nou." << endl;

}

}

}

}

void Graph::print()

{

cout << "Graful obtinut in forma de lista de adiacenta:" << endl;

for (int i = 0; i < V; i++)

{

cout << i << " -> ";

for (auto const &j : adj[i])

{

cout << j << ' ';

}

cout << endl;

}

}

void Graph::DFS\_helper(int s, bool \*visited)

{

cout << s << " ";

visited[s] = true;

for (auto i = adj[s].begin(); i != adj[s].end(); i++)

{

if (!visited[\*i])

{

DFS\_helper(\*i, visited);

}

}

}

void Graph::DFS(int s)

{

cout << "Efectuarea cautarii in adancime: " << endl;

bool \*visited = new bool[V];

for (int i = 0; i < V; i++)

{

visited[i] = false;

}

DFS\_helper(s, visited);

}

int main()

{

int size;

cout << "Cate varfuri va avea graful?" << endl;

while (true)

{

cin >> size;

if (size > 0)

break;

else

cout << "Valoare invalida, introduceti din nou." << endl;

}

Graph g(size);

g.introduce();

g.print();

int root;

cout << "Care va fi elementul de la care se va incepe cautarea?" << endl;

while (true)

{

cin >> root;

if (!g.isOverSize(root) && root >= 0)

{

g.DFS(root);

break;

}

else

cout << "Valoare invalida, introduceti din nou" << endl;

}

return 0;

}

**5. EXECUTIA CODULUI**



**6. CONCLUZII:**

* Căutarea sau parcurgerea în adâncime (denumită și ca în engleză depth-first search, abreviat DFS) este un algoritm pentru parcurgerea sau căutarea într-o structură de date de tip arbore⁠(d) sau graf⁠(d). Se începe de la rădăcină⁠(d) (sau alegând un nod arbitrar ca rădăcină în cazul unui graf) și se explorează cât mai mult posibil de-a lungul fiecărei ramuri înainte de a face pași înapoi.
* Una dintre cele mai efective metode de parcurgere este cautarea printr-o funcție sau metodă recursivă a vârfurilor care încă nu au fost accesate.