Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr. 3

*la Matematica Discretă*

Tema: ALGORITMUL DE CĂUTARE ÎN LĂRGIME

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

A verificat: lect. asist. Popovici Nadejda

Chişinău 2022

1. **SCOPUL LUCRĂRII:**

⦁ Studierea algoritmului de căutare în lărgime;

⦁ Elaborarea programului de căutare în lărgime.

**2.NOTE DE CURS**

**Algoritmul de căutare în lărgime**

Parcurgerea grafului în lărgime, ca şi parcurgerea în adâncime, va garanta vizitarea fiecărui vârf al grafului exact o singură dată, însă principiul va fi altul. După vizitarea vârfului iniţial, de la care va începe căutarea în lărgime, vor fi vizitate toate vârfurile adiacente cu vârful dat, apoi toate vârfurile adiacente cu aceste ultime vârfuri ş.a.m.d. până vor fi vizitate toate vârfurile grafului. Evident, este necesar ca graful să fie conex. Această modalitate de parcurgere a grafului (în lărgime sau postordine), care mai este adesea numită parcurgere în ordine orizontală, realizează parcurgerea vârfurilor de la stânga la dreapta, nivel după nivel.

Vom nota că procedura parcurgerii grafului în lărgime permite să realizăm arborele de căutare şi în acelaşi timp să construim acest arbore. Cu alte cuvinte, se va rezolva problema determinării unei rezolvări sub forma vectorului (*a1, a2,*...) de lungime necunoscută, dacă este cunoscut că există o rezolvare finită a problemei.

Algoritmul pentru cazul general este analogic cu cel pentru un graf în formă de arbore cu o mică modificare care constă în aceea că fiecare vârf vizitat va fi marcat pentru a exclude ciclarea algoritmului.

**3. SARCINA DE BAZĂ**

1. Elaboraţi procedura care va realiza algoritmul de parcurgere a grafului în lărgime;
2. Folosind procedurile din lucrările precedente, elaboraţi programul care va permite:

* introducerea grafului în calculator;
* parcurgerea grafului în lărgime;
* extragerea datelor la display şi printer.

**4. CODUL PROGRAMULUI**

#include <iostream>

#include <list>

#include <algorithm>

using namespace std;

class Graph

{

private:

int V;

list<int> \*adj;

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 BFS(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::BFS(int s)

{

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

bool \*visited = new bool[V];

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

{

visited[i] = false;

}

list<int> queue;

visited[s] = true;

cout << s << " ";

queue.push\_back(s);

while (!queue.empty())

{

s = queue.front();

queue.pop\_front();

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

{

if (!visited[\*i])

{

cout << \*i << " ";

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

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.BFS(root);

break;

}

else

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

}

return 0;

}

**5. EXECUTIA CODULUI**



**6. CONCLUZII:**

* Parcurgerea grafului în lărgime, ca şi parcurgerea în adâncime, va garanta vizitarea fiecărui vârf al grafului exact o singură dată, însă principiul se diferă prin faptul că se vizitează vârfurile la același nivel, și apoi copiii lor și tot așa până se ating vârfurile fără muchii care duc la vârfuri nevizitate.
* Una dintre cele mai efective metode de parcurgere este căutarea printr-un ciclu care folosește un rând pentru a parcurge vârfurile în ordinea necesară.