МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №7

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Обход графа в глубину"

Выполнили студенты группы 24ВВВ3:

Пяткин Р. С.

Гусаров Е. Е.

Принял:

к.т.н., доцент, Юрова О. В.

к.т.н., Деев М. В.

Пенза 2025

Цель

Изучение обхода графа в глубину.

Лабораторное задание

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

3.* Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

Задание 2*

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

Теория

Обход графа – одна из наиболее распространенных операций с графами. Задачей обхода является прохождение всех вершин в графе.

Одним из способов обхода графов является поиск в глубину. Идея такого обхода состоит в том, чтобы начав обход из какой-либо вершины всегда переходить по первой встречающейся в процессе обхода связи в следующую вершину, пока существует такая возможность. Как только в процессе обхода исчерпаются возможности прохода, необходимо вернуться на один шаг назад и найти следующий вариант продвижения. Таким образом, итерационно выполняя описанные операции, будут пройдены все доступные для прохождения вершины. Чтобы не заходить повторно в уже пройденные вершины, необходимо их пометить как пройденные.

Реализация

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОГ

1.1. для всех i положим NUM[i] = False пометим как "не посещенную";

1.2. ПОКА существует "новая" вершина v

1.3. ВЫПОЛНЯТЬ DFS (v).

Алгоритм DFS(v):

2.1. пометить v как "посещенную" NUM[v] = True;

2.2. вывести на экран v;

2.3. ДЛЯ i = 1 ДО size_G ВЫПОЛНЯТЬ

2.4. ЕСЛИ G(v,i) = = 1И NUM[i] = = False

2.5. ТО

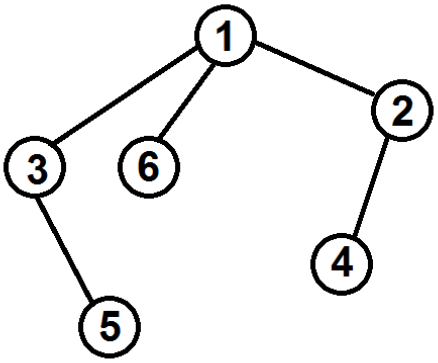
2.6. {

2.7. DFS(i);

2.8. }

Реализация состоит из подготовительной части, в которой все вершины помечаются как не помеченные (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3). И непосредственно процедуры обхода, которая помечает текущую (т.е. ту, в которой на текущей итерации находится алгоритм) вершину как посещенную (п. 2.1). Затем выводит номер текущей вершины на экран (п.2.2) и в цикле просматривает **v**-ю строку матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.4), то для этой вершины вызывается процедура обхода (п.2.7).

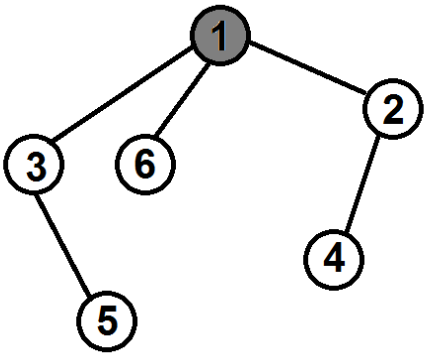
Например, пусть дан граф (рисунок 1), заданный в виде матрицы смежности:



$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Рисунок 1 – Граф

Тогда, если мы начнем обход из первой вершины, то на шаге 2.1 она будет помечена как посещенная (NUM[1] = True), на экран будет выведена единица.



$$NUM = \{True \quad False \quad False \quad False \quad False \quad False\}$$

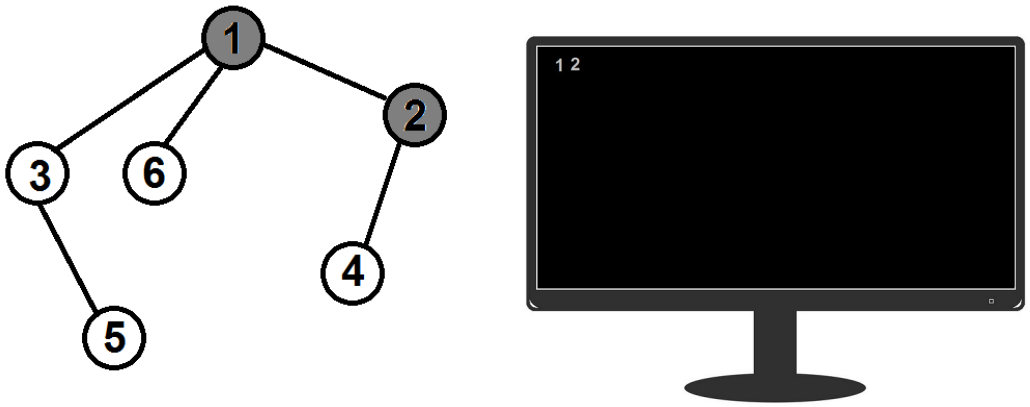
Рисунок 2 – Вызов DFS(1)

При просмотре 1-й строки матрицы смежности

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

будет найдена смежная вершина с индексом 2 (G(1,2) = =1), которая не посещена (NUM[2] = = False) и будет вызвана процедура обхода уже для нее - DFS(2).

На следующем вызове на шаге 2.1 вершина 2 будет помечена как посещенная (NUM[2] = True), на экран будет выведена двойка.



$$NUM = \{True \quad True \quad False \quad False \quad False \quad False\}$$

Рисунок 3 – Вызов DFS(2)

И алгоритм перейдет к просмотру второй строки матрицы смежности. Первая смежная с вершиной 2 - вершина с индексом 1(G(2,1) = =1),

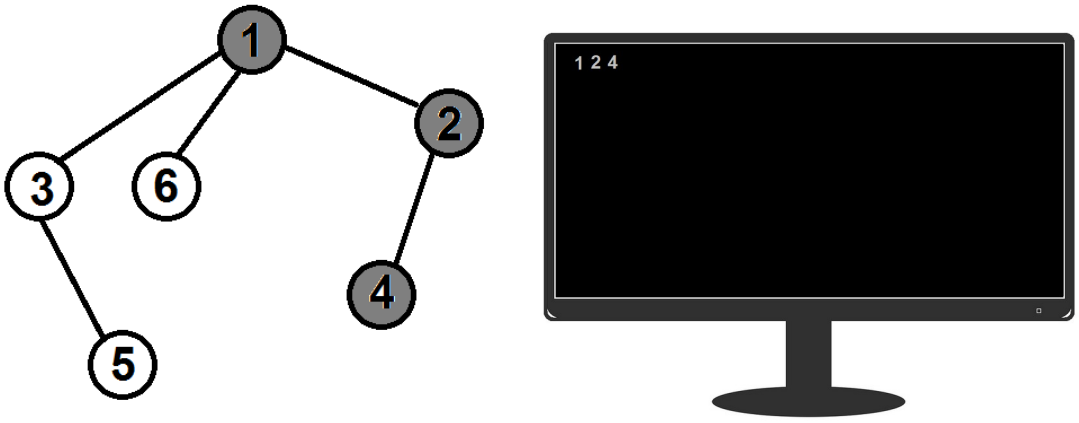
$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

которая к настоящему моменту уже посещена (NUM[1] = = True) и процедура обхода для нее вызвана не будет. Цикл 2.3 продолжит просмотр матрицы смежности.

Следующая найденная вершина, смежная со второй, будет иметь индекс 4 (G(2,4) = =1), она не посещена (NUM[4] = = False) и для нее будет вызвана процедура обхода - DFS(4).

$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Вершина 4 будет помечена как посещенная (NUM[4] = True), на экран будет выведена четверка.



$$NUM = \{True \quad True \quad False \quad True \quad False \quad False\}$$

Рисунок 4 – Вызов DFS(4)

При просмотре 4-й строки матрицы будет найдена вершина 2, но она уже посещена (NUM[2] = True), поэтому процедура обхода вызвана не будет.

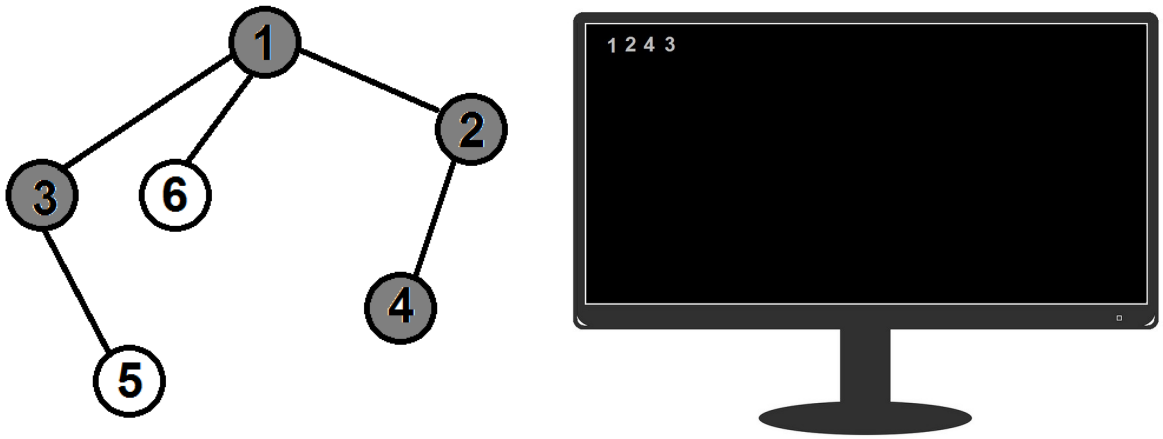
$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Цикл 2.3 завершится и для текущего вызова DFS(4) процедура закончит свою работу, вернувшись к точке вызова, т.е. к моменту просмотра циклом 2.3 строки с индексом 2 для вызова DFS(2).

В вызове DFS(2) цикл 2.3 продолжит просмотр строки 2 в матрице смежности, и, пройдя её до конца завершится. Вместе с этим завершится и вызов процедуры DFS(2), вернувшись к точке вызова - просмотру циклом 2.3 строки с индексом 1 для вызова DFS(1).

При просмотре строки 1 циклом 2.3 в матрице смежности будет найдена следующая не посещенная, смежная с 1-й, вершина с индексом 3 (G(1,2) = =1 и NUM[3] = = False) и для нее будет вызвана DFS(3).

Вершина 3 будет помечена как посещенная (NUM[3] = True), на экран будет выведена тройка.



$$NUM = \{True \quad True \quad True \quad True \quad False \quad False\}$$

Рисунок 4 – Вызов DFS(3)

Работа алгоритма будет продолжаться до тех пор, пока будут оставаться не посещенные вершины, т.е. для которых NUM[i] = = False.

В конце работы алгоритма все вершины будут посещены. А на экран будут выведены номера вершин в порядке их посещения алгоритмом.

$$NUM = \{True \quad True \quad True \quad True \quad True \quad True\}$$

Рисунок 5 – Результат работы обхода

Результаты программ

```
Введите кол-во вершин в матрице: 8
   0   1   2   3   4   5   6   7
0| 0   1   1   1   0   0   1   1
1| 1   0   1   0   0   1   0   0
2| 1   1   0   0   0   1   0   1
3| 1   0   0   0   1   1   1   0
4| 0   0   0   1   0   0   0   0
5| 0   1   1   1   0   0   0   0
6| 1   0   0   1   0   0   0   0
7| 1   0   1   0   0   0   0   0

Вершина с которой начинать обход: 1
Результат: 10253467
```

Рис. 1

Вывод:

В ходе выполнения лабораторной работы были разработаны программа для выполнения заданий Лабораторной работы №7. В процессе выполнения работы был изучен способ обхода графа в глубину.

Листинг

Файл lab_7_1.cpp

```
#include <iostream>
#include <ctime>
#include <random>

using namespace std;

int printMatrix(int** m, int n){
cout << " ";
for(int i = 0; i < n; i++){
cout << i << " ";
}
cout << endl;
for(int i = 0; i < n; i++){
cout << i << "| ";
for(int j = 0; j < n; j++){
cout << m[i][j] << " ";
}
cout << endl;
}
cout << endl;
return 0;
}

int** createMatrix(int n){
int** m = new int*[n];
for (int i = 0; i < n; i++) {
m[i] = new int[n];
}

for(int i = 0; i < n; i++){
for(int j = i; j < n; j++){
m[i][j] = m[j][i] = (i == j ? 0 : rand() % 2);
}
}
return m;
}
void deleteM(int** m, int n, int* v) {
for (int i = 0; i < n; ++i) {
delete[] m[i];
}
```

```
delete[] m;
delete[] v;
}
void DFS(int** G, int numG, int* visited, int s){
visited[s] = 1;
cout<<s;
for(int i = 0; i < numG; i++){
if(G[s][i] == 1 && visited[i] == 0){
DFS(G, numG, visited, i);
}
}
}

int main(){
srand(time(0));
int numG, current = 0;
cout << "Введите кол-во вершин в матрице: ";
cin >> numG;
int* visited = new int[numG];

for(int i = 0; i < numG; i++){
visited[i] = 0;
}

int** G = createMatrix(numG);
printMatrix(G, numG);

cout << "Вершина с которой начинать обход: ";
cin >> current;
cout << "Результат: ";
DFS(G, numG, visited, current);
cout << endl;
deleteM(G, numG, visited);
return 0;
}
```

Файл lab_7_2.cpp

```
#include <iostream>
#include <ctime>
#include <random>

using namespace std;

class Node{
public:
int data;
Node* next;

Node(int value):data(value),next(nullptr){}
```

```
};

class LinkedList{
private:
Node* head;

public:
LinkedList():head(nullptr){}
~LinkedList(){ clear(); }

void push_front(int value) {
Node* newNode = new Node(value);
newNode->next = head;
head = newNode;
}
Node* getHead() const{
return head;
}
void clear() {
Node* current = head;
while (current != nullptr) {
Node* next = current->next;
delete current;
current = next;
}
head = nullptr;
}
bool empty() const {
return head == nullptr;
}
};

class Graph{
int numVertices;
LinkedList* adjLists;
public:
Graph(int V);
void addEdge(int src, int dest);
void printGraph();
~Graph();

int getNumVertices() const { return numVertices; }
Node* getAdjListHead(int vertex) const { return adjLists[vertex].getHead(); }
};

Graph::Graph(int vertices){
numVertices = vertices;
adjLists = new LinkedList[vertices];
}
Graph::~Graph(){
delete[] adjLists;
```

```
}

void Graph::addEdge(int src, int dest){
adjLists[src].push_front(dest);
}
void Graph::printGraph()
{
for (int i = 0; i < numVertices; i++)
{
cout << "Вершина " << i << ":";
Node* current = adjLists[i].getHead();
while (current != nullptr)
{
cout << " -> " << current->data;
current = current->next;
}
cout << endl;
}
cout << endl;
}
void DFS(const Graph& graph, int* visited, int s){
visited[s] = 1;
cout << s << " ";
Node* current = graph.getAdjListHead(s);
while (current != nullptr) {
int neighbor = current->data;
if (visited[neighbor] == 0) {
DFS(graph, visited, neighbor);
}
current = current->next;
}
}
void runDFS(const Graph& graph, int startVertex) {
int numVertices = graph.getNumVertices();
if (startVertex < 0 || startVertex >= numVertices) {
cout << "Ошибка: начальная вершина " << startVertex << " не существует!" << endl;
return;
}
int* visited = new int[numVertices]();
cout << "DFS обход начиная с вершины " << startVertex << ": ";
DFS(graph, visited, startVertex);
cout << endl;
delete[] visited;
}

Graph createGraph(int n) {
Graph graph(n);
srand(time(0));
cout << "Создание случайного графа с " << n << " вершинами..." << endl;
for (int i = 0; i < n; i++) {
for (int j = i + 1; j < n; j++) {
```

```
if (rand() % 2 == 1) {
graph.addEdge(i, j);
graph.addEdge(j, i);
}
}
}
return graph;
}

int main(){
int n;
cout << "Введите количество вершин в графе: ";
cin >> n;
if (n <= 0) {
cout << "Ошибка: количество вершин должно быть положительным!" << endl;
return 1;
}
Graph g = createGraph(n);
cout << "\nСтруктура графа:" << endl;
g.printGraph();
int startVertex;
cout << "Введите начальную вершину (0-" << n-1 << "): ";
cin >> startVertex;
runDFS(g, startVertex);

return 0;
}
```

Файл lab_7_3.cpp

```
#include <iostream>
#include <ctime>
#include <random>
#include <stack>

using namespace std;

int printMatrix(int** m, int n){
cout << " ";
for(int i = 0; i < n; i++){
cout << i << " ";
}
cout << endl;
for(int i = 0; i < n; i++){
cout << i << "| ";
for(int j = 0; j < n; j++){
cout << m[i][j] << " ";
}
cout << endl;
}
cout << endl;
```

```
return 0;
}

int** createMatrix(int n){
int** m = new int*[n];
for (int i = 0; i < n; i++) {
m[i] = new int[n];
}

for(int i = 0; i < n; i++){
for(int j = i; j < n; j++){
m[i][j] = m[j][i] = (i == j ? 0 : rand() % 2);
}
}
return m;
}

void deleteM(int** m, int n, int* v) {
for (int i = 0; i < n; ++i) {
delete[] m[i];
}
delete[] m;
delete[] v;
}

void DFS(int** G, int numG, int* visited, int start){
stack<int> st;
st.push(start);
visited[start] = 1;
cout << start;
while (!st.empty()) {
int current = st.top();
bool found = false;
for (int i = 0; i < numG; i++) {
if (G[current][i] == 1 && visited[i] == 0) {
visited[i] = 1;
cout << i;
st.push(i);
found = true;
break;
}
}
if (!found) {
st.pop();
}
}
}

int main(){
srand(time(0));
int numG, current = 0;
```

```
cout << "Введите кол-во вершин в матрице: ";
cin >> numG;
int* visited = new int[numG];

for(int i = 0; i < numG; i++){
visited[i] = 0;
}

int** G = createMatrix(numG);
printMatrix(G, numG);

cout << "Вершина с которой начинать обход: ";
cin >> current;
cout << "Результат: ";
DFS(G, numG, visited, current);
cout << endl;
deleteM(G, numG, visited);
return 0;
}
```