Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 7

по дисциплине: "Логика и ОА в ИЗ"

## на тему: "Обход графа в глубину"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза, 2023

**Цель.**

### Изучить принцип обхода графов в глубину с помощью рекурсивной и не рекурсивной функции, с помощью матрицы смежности и списка смежности.

**Общие сведения.**

### Обход графа – одна из наиболее распространенных операций с графами. Задачей обхода является прохождение всех вершин в графе. Обходы применяются для поиска информации, хранящейся в узлах графа, нахождения связей между вершинами или группами вершин и т.д.

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.

Таким образом, можно предложить следующую рекурсивную реализацию алгоритма обхода в глубину.

**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) = = 1**И** 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:



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.

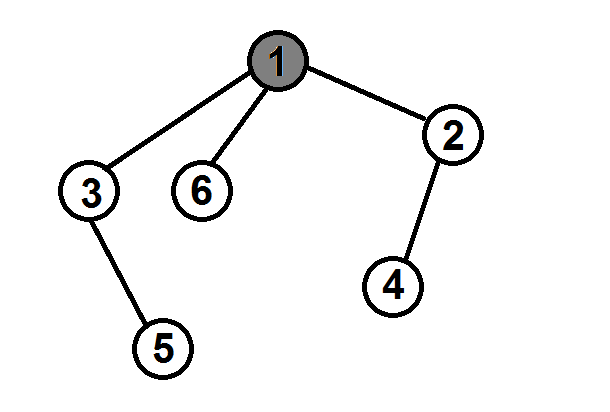
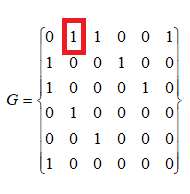
 



Рисунок 2 – Вызов DFS(1)

При просмотре 1-й строки матрицы смежности



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.

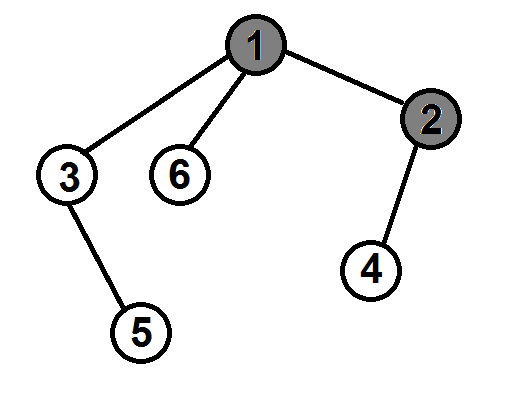
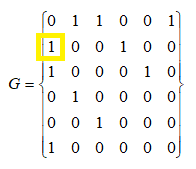




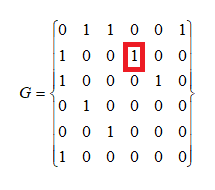
Рисунок 3 – Вызов DFS(2)

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



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).



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

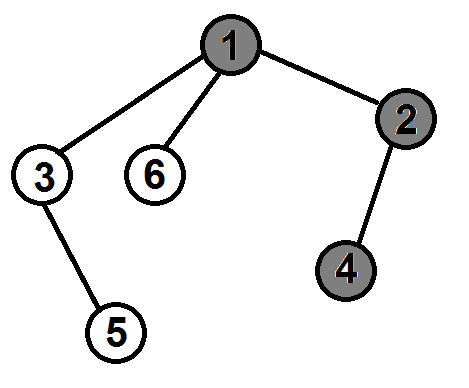
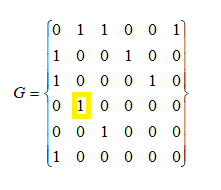




Рисунок 4 – Вызов DFS(4)

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



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.

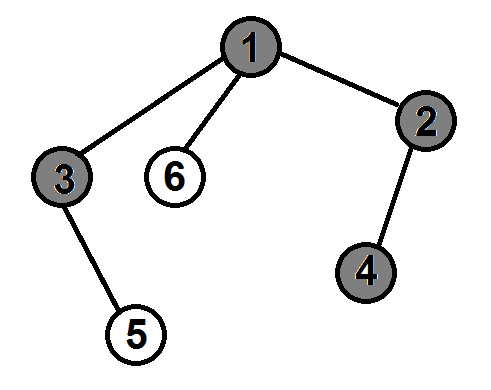




Рисунок 4 – Вызов DFS(3)

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.

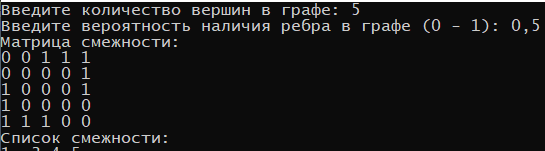




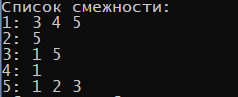
Рисунок 5 – Результат работы обхода

### **Задание 1**

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



1. Для сгенерированного графа осуществили процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

****

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



### **Задание 2\***

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



Приложение А (Листинг)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <stack>

using namespace std;

vector<bool> NUM; // Вектор для хранения информации о посещенных вершинах

void DFS\_Matrix\_NonRecursive(int start\_vertex, const vector < vector < int>>& G) {

stack<int> s;

s.push(start\_vertex);

while (!s.empty()) {

int v = s.top();

s.pop();

bool isIsolated = true;

for (int i = 1; i < G.size(); ++i) {

if (G[v][i] == 1) {

isIsolated = false;

break;

}

}

if (isIsolated) continue; // Пропустить изолированные вершины

if (!NUM[v]) {

printf("%d ", v);

NUM[v] = true;

for (int i = G.size() - 1; i >= 1; --i) {

if (G[v][i] == 1 && !NUM[i]) {

s.push(i);

}

}

}

}

}

void DFS\_Matrix(int v, const vector<vector<int>>& G) {

bool isIsolated = true;

for (int i = 1; i < G.size(); ++i) {

if (G[v][i] == 1) {

isIsolated = false;

break;

}

}

if (isIsolated) return;

NUM[v] = true;

printf("%d ", v);

for (int i = 1; i < G.size(); ++i) {

if (G[v][i] == 1 && !NUM[i]) {

DFS\_Matrix(i, G);

}

}

}

void DFS\_List(int v, const vector < vector < int>>& adjacency\_list) {

if (adjacency\_list[v].empty()) return; // Пропустить изолированные вершины

NUM[v] = true;

printf("%d ", v);

for (const int& neighbor : adjacency\_list[v]) {

if (!NUM[neighbor]) {

DFS\_List(neighbor, adjacency\_list);

}

}

}

void printList(const vector<vector<int>>& list) {

printf("Список смежности:\n");

for (int i = 1; i < list.size(); ++i) {

printf("%d: ", i);

for (const int& j : list[i]) {

printf("%d ", j);

}

printf("\n");

}

}

int swt = 0;

vector<vector<int>> generateAndPrintMatrix(int num\_vertices, double edge\_probability) {

vector<vector<int>> adjacency\_matrix(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

int cont = 0;

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < edge\_probability) {

adjacency\_matrix[i][j] = 1;

adjacency\_matrix[j][i] = 1;

cont++;

}

}

}

if (cont == 0) {

swt = 1;

}

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", adjacency\_matrix[i][j]);

}

printf("\n");

}

return adjacency\_matrix;

}

vector<vector<int>> matrixToList(const vector<vector<int>>& matrix) {

int num\_vertices = matrix.size() - 1;

vector<vector<int>> adjacency\_list(num\_vertices + 1);

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

if (matrix[i][j] == 1) {

adjacency\_list[i].push\_back(j);

}

}

}

return adjacency\_list;

}

int main() {

setlocale(LC\_ALL, "RUS");

int num\_vertices;

double edge\_probability;

printf("Введите количество вершин в графе: ");

scanf("%d", &num\_vertices);

printf("Введите вероятность наличия ребра в графе (0 - 1): ");

scanf("%lf", &edge\_probability);

srand(static\_cast<unsigned int>(time(nullptr)));

auto matrix = generateAndPrintMatrix(num\_vertices, edge\_probability);

auto adjacency\_list = matrixToList(matrix);

if (swt == 1) {

printf("\nПервая вершина - изолированная\n");

}

else {

printList(adjacency\_list);

NUM = vector<bool>(num\_vertices + 1, false);

printf("Обход в глубину для матрицы (рекурсивный): ");

for (int i = 1; i <= num\_vertices; ++i) {

if (!NUM[i]) {

DFS\_Matrix(i, matrix);

}

}

printf("\n");

NUM = vector<bool>(num\_vertices + 1, false);

printf("Обход в глубину для матрицы (не рекурсивный): ");

for (int i = 1; i <= num\_vertices; ++i) {

if (!NUM[i]) {

DFS\_Matrix\_NonRecursive(i, matrix);

}

}

printf("\n");

NUM = vector<bool>(num\_vertices + 1, false);

printf("Обход в глубину для списка: ");

for (int i = 1; i <= num\_vertices; ++i) {

if (!NUM[i]) {

DFS\_List(i, adjacency\_list);

}

}

printf("\n");

}

return 0;

}