Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Обход графа в глубину**»

Выполнили:

студенты группы 23ВВВ4

Брагин А.М.

Зарубин Я.

Герасимов К.

Приняли:

Деев М.В.

Юрова О.В.

Пенза 2024

**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:

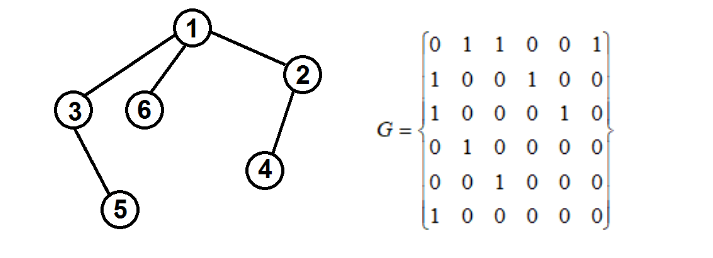
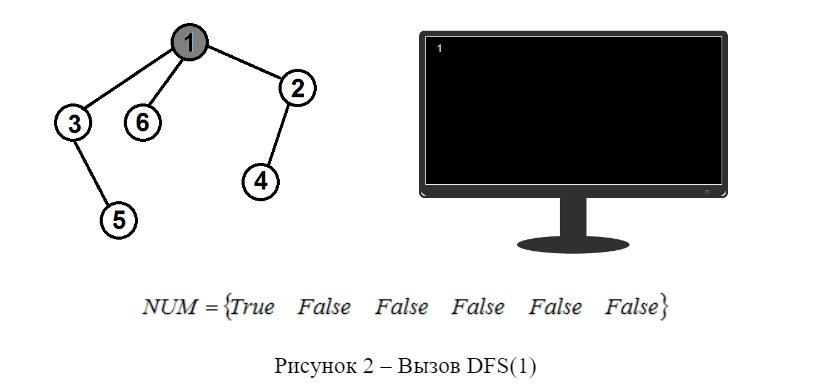
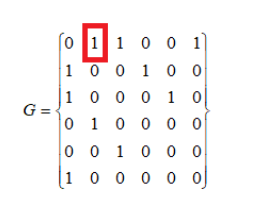


Рисунок 1 – Граф

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.

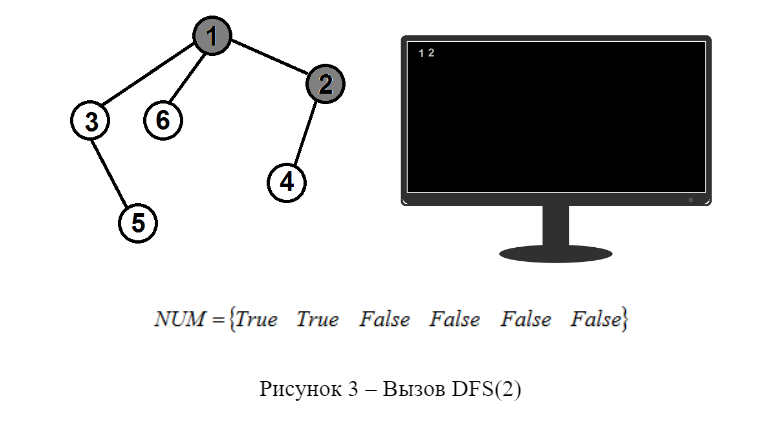


При просмотре 1-й строки матрицы смежности

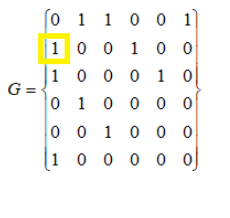


будет найдена смежная вершина с индексом 2 (G(1,2) = =1), которая не посещена (NUM[2] = = False) и будет вызвана процедура обхода уже для нее -  DFS(2).

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

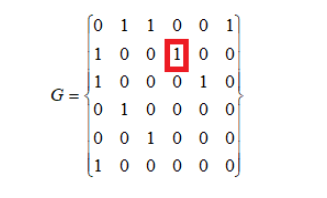


И алгоритм перейдет к просмотру второй строки матрицы смежности. Первая смежная с вершиной 2 - вершина с индексом 1(G(2,1) = =1),



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.

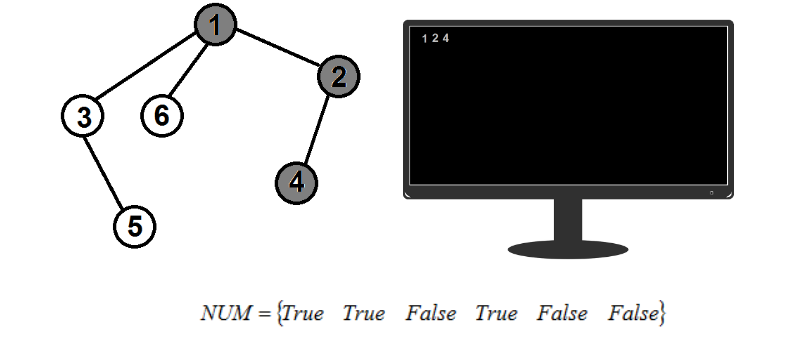
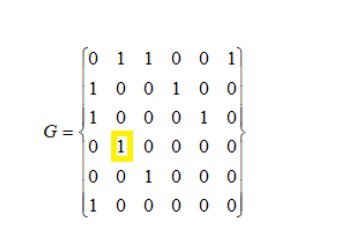


Рисунок 4 – Вызов DFS(4)

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.

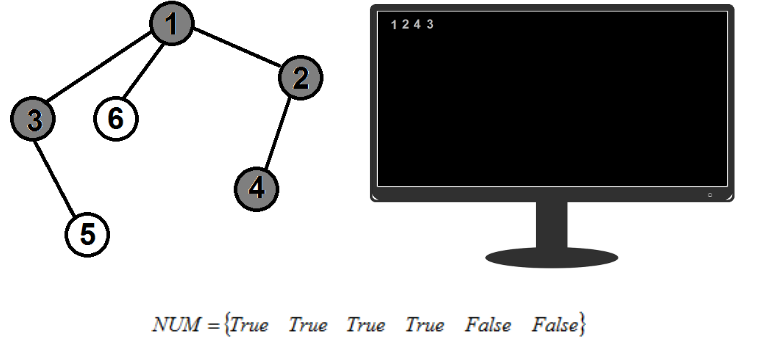


Рисунок 4 – Вызов DFS(3)

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.

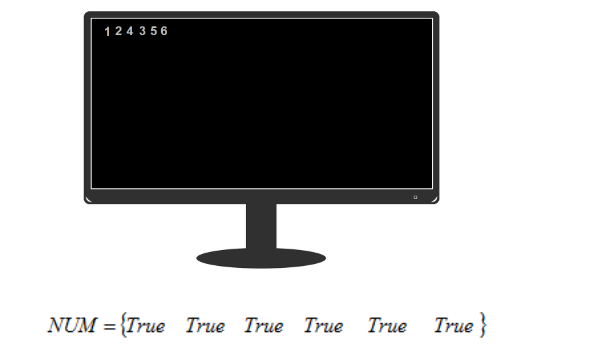


Рисунок 5 – Результат работы обхода

**Практическая часть:**

**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.

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <locale.h> // Добавили библиотеку для установки локали

#define N 5 // Размер графа

// Генерация матрицы смежности

void generateAdjacencyMatrix(int graph[N][N]) {

srand(time(NULL));

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

for (int j = i; j < N; j++) {

if (i == j) {

graph[i][j] = 0;

}

else {

int edge = rand() % 2;

graph[i][j] = edge;

graph[j][i] = edge;

}

}

}

}

// Печать матрицы смежности

void printMatrix(int graph[N][N]) {

printf("Матрица смежности:\n");

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

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

printf("%d ", graph[i][j]);

}

printf("\n");

}

}

// Рекурсивный обход в глубину по матрице смежности

void DFS\_Matrix\_Recursive(int v, bool visited[], int graph[N][N]) {

visited[v] = true;

printf("%d ", v);

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

if (graph[v][i] && !visited[i]) {

DFS\_Matrix\_Recursive(i, visited, graph);

}

}

}

// Нерекурсивный обход в глубину по матрице смежности (с использованием стека)

void DFS\_Matrix\_NonRecursive(int graph[N][N], int start) {

bool visited[N] = { false };

int stack[N];

int top = -1;

stack[++top] = start;

while (top >= 0) {

int v = stack[top--];

if (!visited[v]) {

visited[v] = true;

printf("%d ", v);

}

for (int i = N - 1; i >= 0; i--) { // Добавляем в стек вершины в обратном порядке

if (graph[v][i] && !visited[i]) {

stack[++top] = i;

}

}

}

}

// Узел для списка смежности

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

// Создание нового узла

Node\* createNode(int v) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Добавление ребра в список смежности

void addEdge(Node\* adjList[], int src, int dest) {

Node\* newNode = createNode(dest);

newNode->next = adjList[src];

adjList[src] = newNode;

newNode = createNode(src);

newNode->next = adjList[dest];

adjList[dest] = newNode;

}

// Печать списка смежности

void printAdjList(Node\* adjList[]) {

printf("Список смежности:\n");

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

printf("%d: ", i);

Node\* temp = adjList[i];

while (temp) {

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("NULL\n");

}

}

// Рекурсивный обход в глубину по спискам смежности

void DFS\_List(int v, bool visited[], Node\* adjList[]) {

visited[v] = true;

printf("%d ", v);

Node\* temp = adjList[v];

while (temp) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

DFS\_List(adjVertex, visited, adjList);

}

temp = temp->next;

}

}

// Освобождение памяти для списка смежности

void freeGraph(Node\* adjList[]) {

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

Node\* temp = adjList[i];

while (temp) {

Node\* next = temp->next;

free(temp);

temp = next;

}

}

}

int main() {

setlocale(LC\_ALL, ""); // Установили локаль для поддержки русского языка

int graph[N][N];

generateAdjacencyMatrix(graph);

printMatrix(graph);

// Рекурсивный обход в глубину по матрице смежности

bool visited[N] = { false };

printf("Рекурсивный обход в глубину (матрица смежности): ");

DFS\_Matrix\_Recursive(0, visited, graph);

printf("\n");

// Нерекурсивный обход в глубину по матрице смежности

printf("Нерекурсивный обход в глубину (матрица смежности): ");

DFS\_Matrix\_NonRecursive(graph, 0);

printf("\n");

// Создание списка смежности

Node\* adjList[N] = { NULL };

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

for (int j = i + 1; j < N; j++) {

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

addEdge(adjList, i, j);

}

}

}

printAdjList(adjList);

// Обход в глубину по спискам смежности

for (int i = 0; i < N; i++) visited[i] = false; // Сброс массива visited

printf("Рекурсивный обход в глубину (список смежности): ");

DFS\_List(0, visited, adjList);

printf("\n");

// Освобождение памяти

freeGraph(adjList);

return 0;

}