

МИНИСТЕРСТВО 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:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. 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.

Пояснительный текст к программам

Эта программа реализует алгоритм обхода графа в глубину (DFS).

Пользователь задаёт количество вершин графа, после чего программа автоматически генерирует симметричную матрицу смежности, где:

- 0 означает отсутствие ребра между вершинами
- 1 означает наличие ребра

Главная диагональ матрицы заполняется нулями (исключаются петли).

После отображения матрицы пользователь выбирает стартовую вершину, и программа выполняет обход в глубину, выводя порядок посещения вершин. По завершении вся выделенная память освобождается.

Псевдокод:

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

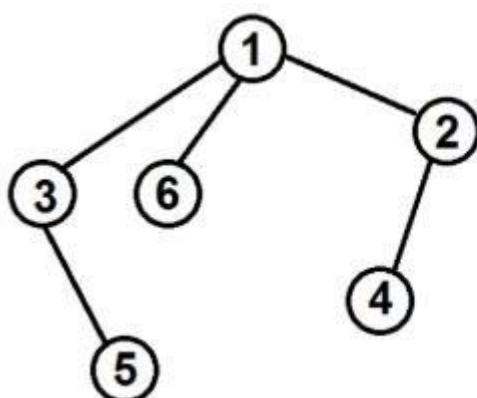
2.5. **ТО**

2.6. {

2.7. DFS(i);

2.8. }

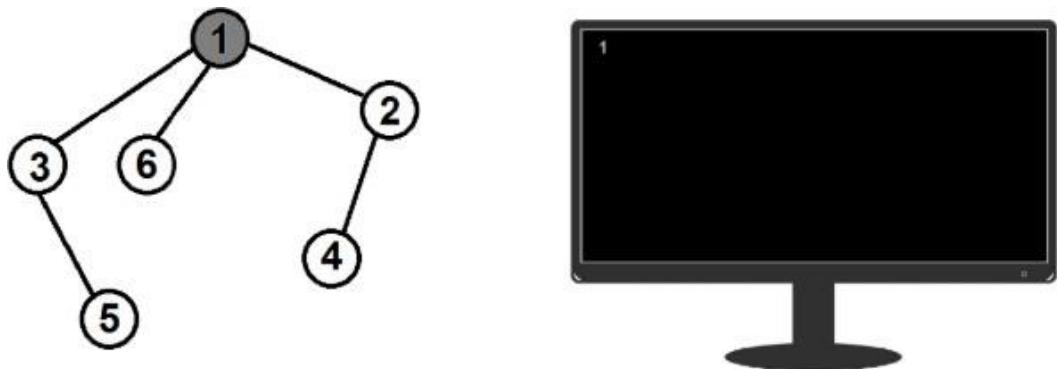
Например, пусть дан граф (рисунок 1), заданный в виде матрицы смежности:



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

Рисунок 1 - Граф

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



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

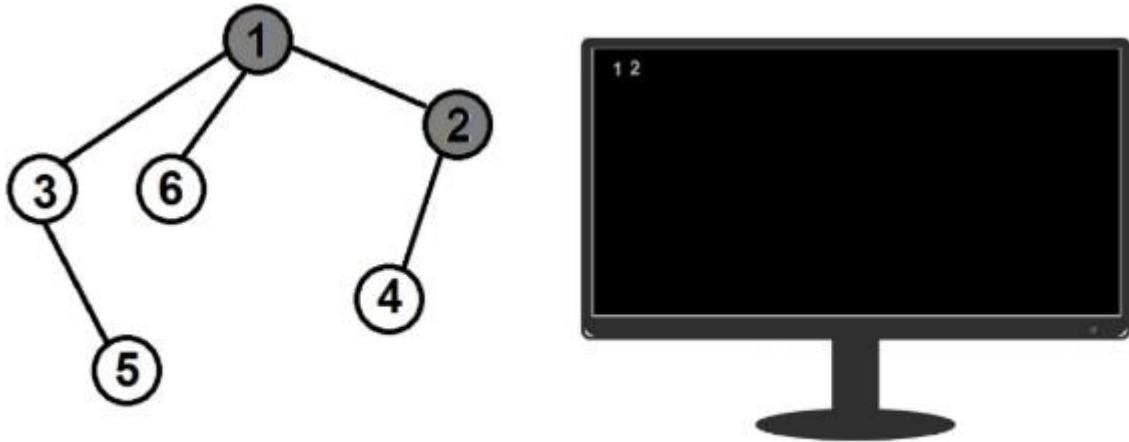
Рисунок 2 – Вызов DFS(1)

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 ($\text{NUM}[2] == \text{False}$) и будет вызвана процедура обхода уже для нее - DFS(2).

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



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

Рисунок 3 – Вызов DFS(2)

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

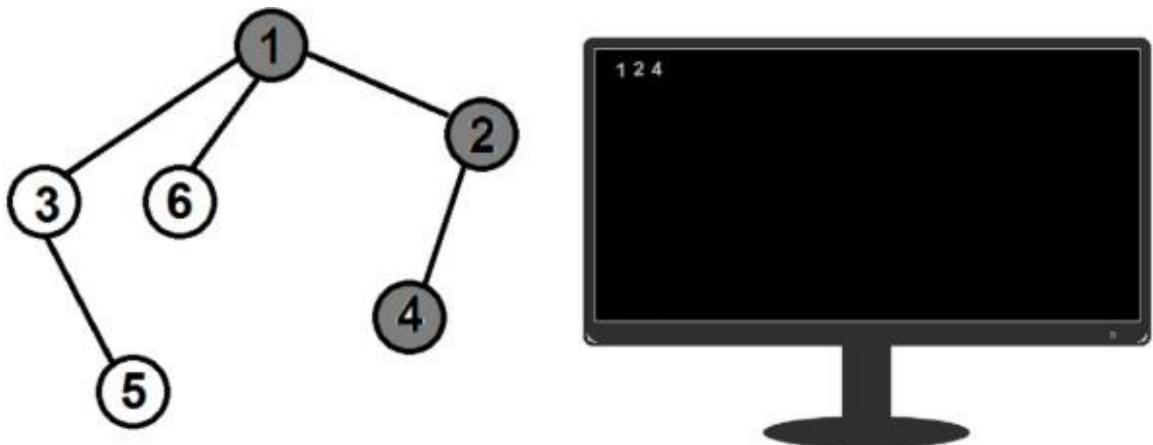
$$G = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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).

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

Вершина 4 будет помечена как посещенная ($\text{NUM}[4] = \text{True}$), на экран будет выведена четверка.



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

Рисунок 4 – Вызов DFS(4)

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

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

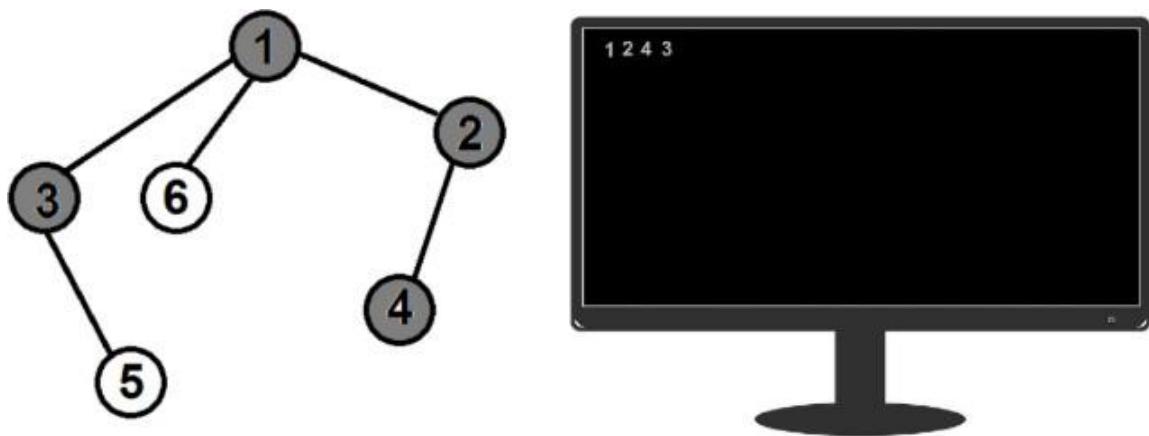
Цикл 2.3 завершится и для текущего вызова $\text{DFS}(4)$ процедура закончит свою работу, вернувшись к точке вызова, т.е. к моменту просмотра циклом 2.3 строки с индексом 2 для вызова $\text{DFS}(2)$.

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

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 $\text{NUM}[3] == \text{False}$) и для нее будет вызвана $\text{DFS}(3)$.

Вершина 3 будет помечена как посещенная ($\text{NUM}[3] = \text{True}$), на экран будет выведена тройка.



$$\text{NUM} = \{\text{True}, \text{True}, \text{True}, \text{True}, \text{False}, \text{False}\}$$

Рисунок 4 – Вызов $\text{DFS}(3)$

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

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



$$\text{NUM} = \{\text{True}, \text{True}, \text{True}, \text{True}, \text{True}, \text{True}\}$$

Рисунок 5 – Результат работы обхода

Результаты программ

Рисунок 6 - Результат работы laba7.cpp

```
Введите количество вершин графа: 5
```

```
Матрица смежности:
```

```
0 0 0 0 0
0 0 0 1 0
0 0 0 0 1
1 1 1 0 0
1 0 1 0 0
```

```
Списки смежности:
```

```
0:
1: 3
2: 4
3: 0 1 2
4: 0 2
```

```
Введите вершину для начала обхода: 2
```

```
Путь (обычный DFS):
```

```
2
4
0
```

```
Путь (DFS со стеком):
```

```
2
4
0
```

```
Путь (DFS по спискам смежности):
```

```
2
4
0
```

Вывод: в ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №7 – обход графа в глубину.

Приложение А Листинг

Файл laba7.cpp

```
// обход в глубину, стек от руки, списки, в 1 код
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <locale>
#include <limits>
#include <iomanip>

struct StackNode {
    int value;
    StackNode* next;
};

StackNode* stackHead = nullptr;

struct Node {
    int inf;
    Node* next;
};

void pushStack(int v);
bool isEmptyStack();
int popStackC();

void dfs(int** G, int numG, int* visited, int v);
void dfsStack(int** G, int numG, int* visited, int v);
void dfsList(Node** adj, int v, int* visited);

void clearScreen();
int isInteger(const std::string& message);

int main() {
    setlocale(LC_ALL, "Rus");
    clearScreen();
    srand(time(NULL));

    int numG = isInteger("Введите количество вершин графа: ");
    while (numG <= 0) {
        std::cout << "Ошибка! Положительное число!\n";
        numG = isInteger("Введите количество вершин графа: ");
    }

    int** G = new int*[numG];
    for (int i = 0; i < numG; i++)
        G[i] = new int[numG];

    int* visited = new int[numG];

    for (int i = 0; i < numG; i++) {
        visited[i] = 0;
        for (int j = 0; j < numG; j++)
            G[i][j] = (i == j ? 0 : rand() % 2);
    }
}
```

```

std::cout << "\nМатрица смежности:\n";
for (int i = 0; i < numG; i++) {
    for (int j = 0; j < numG; j++)
        std::cout << std::setw(3) << G[i][j];
    std::cout << "\n";
}

Node** adj = new Node*[numG];
for (int i = 0; i < numG; i++)
    adj[i] = nullptr;

for (int i = 0; i < numG; i++) {
    for (int j = numG - 1; j >= 0; j--) {
        if (G[i][j] == 1) {
            Node* p = new Node{ j, adj[i] };
            adj[i] = p;
        }
    }
}
}

std::cout << "\nСписки смежности:\n";
for (int i = 0; i < numG; i++) {
    std::cout << i << ": ";
    Node* cur = adj[i];
    while (cur) {
        std::cout << cur->inf << " ";
        cur = cur->next;
    }
    std::cout << "\n";
}

int start = isInteger("\nВведите вершину для начала обхода: ");

for (int i = 0; i < numG; i++) visited[i] = 0;
std::cout << "\nПуть (обычный DFS):\n";
dfs(G, numG, visited, start);

for (int i = 0; i < numG; i++) visited[i] = 0;
stackHead = nullptr;
std::cout << "\nПуть (DFS со стеком):\n";
dfsStack(G, numG, visited, start);

for (int i = 0; i < numG; i++) visited[i] = 0;
std::cout << "\nПуть (DFS по спискам смежности):\n";
dfsList(adj, start, visited);

for (int i = 0; i < numG; i++) {
    Node* cur = adj[i];
    while (cur) {
        Node* t = cur;
        cur = cur->next;
        delete t;
    }
}
delete[] adj;
delete[] visited;

```

```

        for (int i = 0; i < numG; i++)
            delete[] G[i];
        delete[] G;

        return 0;
    }

void pushStack(int v) {
    StackNode* p = new StackNode{v, stackHead};
    stackHead = p;
}

bool isEmptyStack() {
    return stackHead == nullptr;
}

int popStack() {
    if (isEmptyStack()) return -1;
    StackNode* t = stackHead;
    int v = t->value;
    stackHead = stackHead->next;
    delete t;
    return v;
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();

            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        return value;
    }
}

void dfs(int** G, int numG, int* visited, int v) {
    visited[v] = 1;
    std::cout << " " << v << "\n";
    for (int i = 0; i < numG; i++) {
        if (G[v][i] == 1 && visited[i] == 0)
            dfs(G, numG, visited, i);
    }
}

```

```

void dfsStack(int** G, int numG, int* visited, int v) {
    pushStack(v);
    while (!isEmptyStack()) {
        int cur = popStack();
        if (!visited[cur]) {
            visited[cur] = 1;
            std::cout << " " << cur << "\n";
            for (int i = numG - 1; i >= 0; i--) {
                if (G[cur][i] == 1 && visited[i] == 0)
                    pushStack(i);
            }
        }
    }
}

void dfsList(Node** adj, int v, int* visited) {
    visited[v] = 1;
    std::cout << " " << v << "\n";
    Node* cur = adj[v];
    while (cur) {
        if (!visited[cur->inf])
            dfsList(adj, cur->inf, visited);
        cur = cur->next;
    }
}

```