Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Отчёт

по лабораторной работе №7

по курсу «Л и О А в ИЗ»

на тему «Обход графа в глубину»

Выполнили студенты группы 24ВВВ4:

Кошелев Р.Д.

Кондратьев С.В.

Приняли к.т.н., доцент:

Юрова О.В.

к.э.н., доцент:

Акифьев И.В.

Пенза 2025

Цель работы: изучение и практическая реализация алгоритма обхода графа в глубину (Depth-First Search, DFS) для графа, представленного матрицей и списками 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**

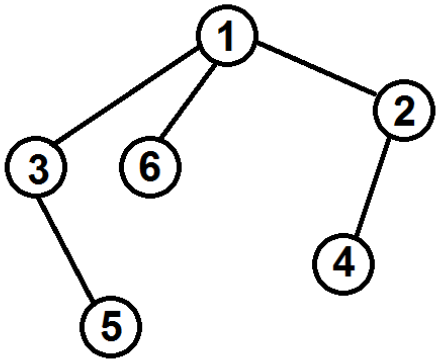
2.6. {

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).

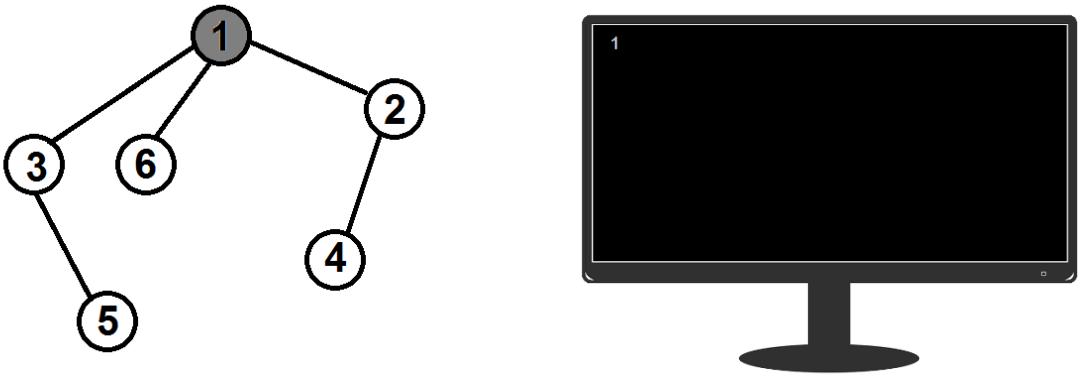
Например, пусть дан граф (рисунок 1), заданный в виде матрицы смежности:



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

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.



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

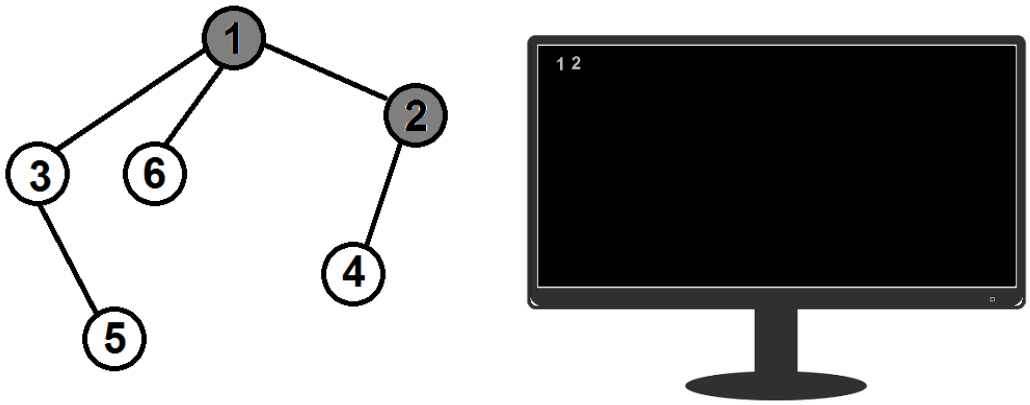
Рисунок 2 – Вызов DFS(1)

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),

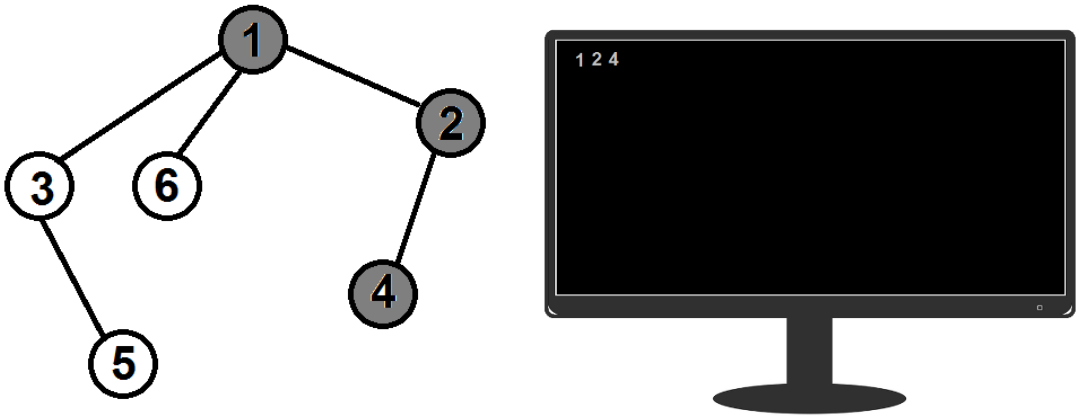
$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

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.

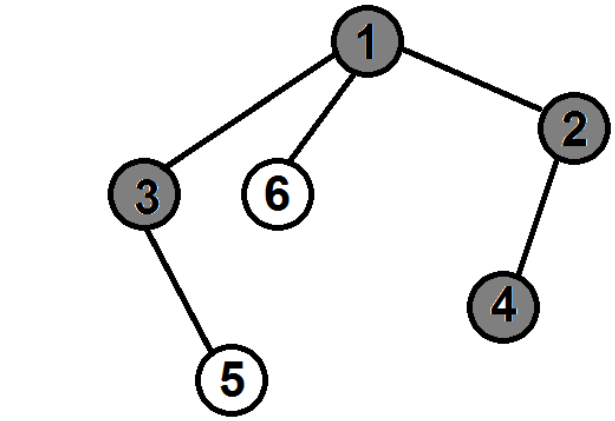
$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

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

Задание 1

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

1. 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.

Результат работы программы

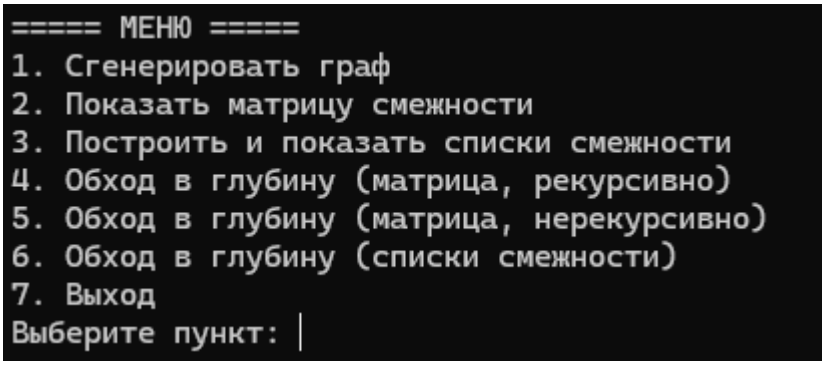


Рисунок 1 – Меню программы

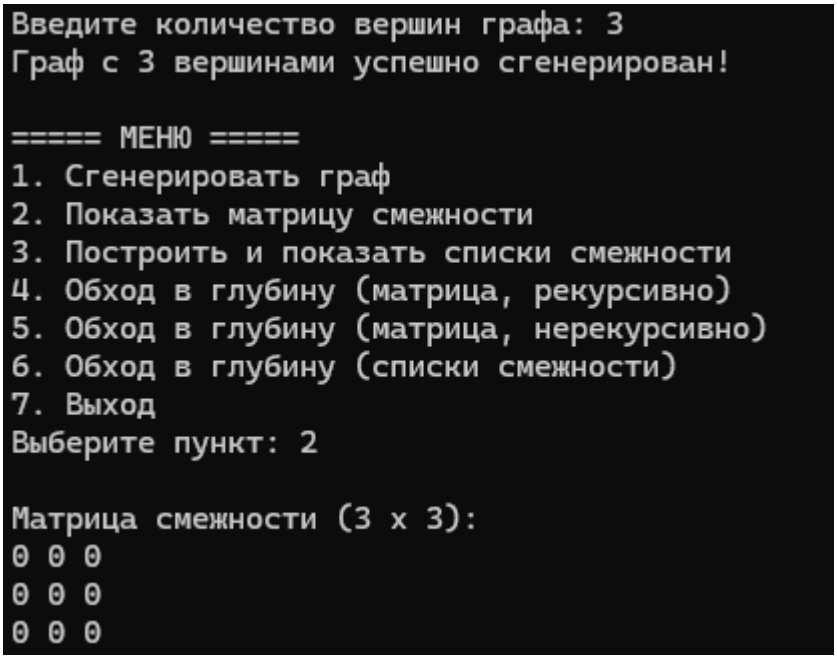


Рисунок 2 – Генерация и вывод матрицы смежности

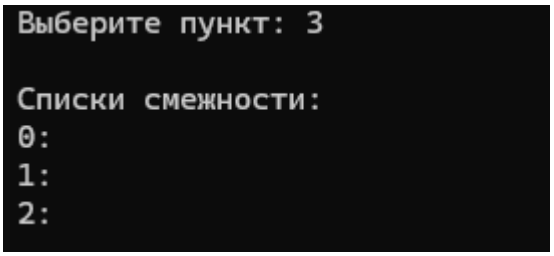


Рисунок 3 – Построение и вывод списка смежности

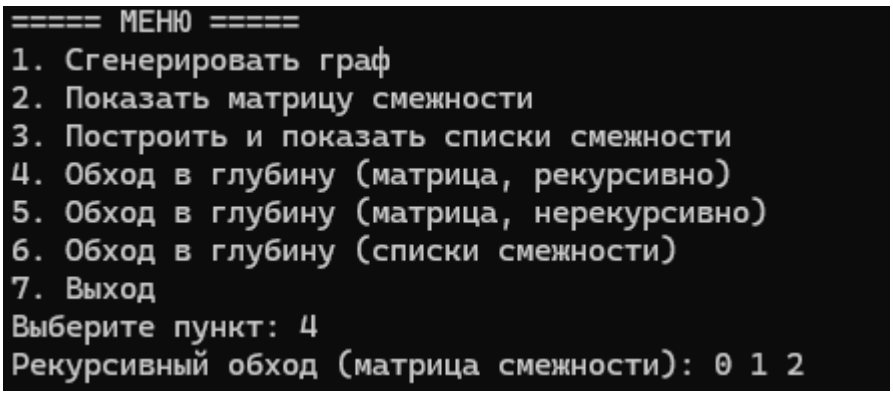


Рисунок 4 – Рекурсивный обход матрицы в глубину

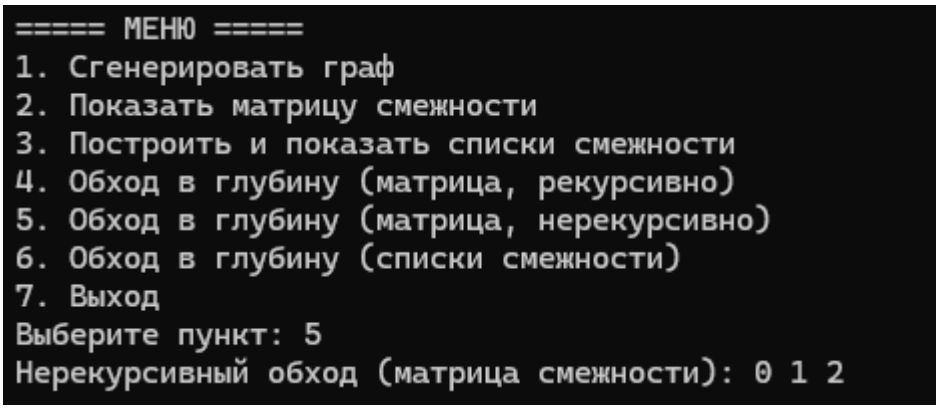


Рисунок 5 – Нерекурсивный обход матрицы в глубину

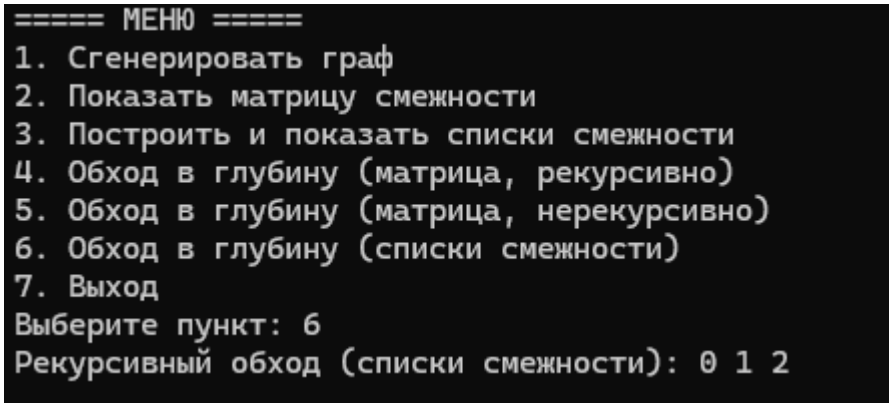


Рисунок 6 – Обход в глубину по списку смежности

Вывод: в результате выполнения работы была успешно разработана программа, реализующая алгоритм обхода графа в глубину. Алгоритм обхода в глубину демонстрирует правильную работу, помечая вершины как посещенные и обеспечивая корректный порядок обхода в глубину с обработкой всех компонент связности графа.

Приложение А

Листинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <locale.h>

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int adjList[MAX_VERTICES][MAX_VERTICES];
int listSize[MAX_VERTICES];
int visited[MAX_VERTICES];
int n = 0;
int generated = 0;

// Проверка: положительное целое число
int is_positive_integer(const char* str) {
    if (str == NULL || *str == '\0')
        return 0;
    for (int i = 0; str[i] != '\0'; i++)
        if (!isdigit(str[i])) return 0;
    int num = atoi(str);
    return num > 0;
}
```

```
// Безопасный ввод количества вершин
int get_vertex_count() {
    char input[100];
    int num;
    while (1) {
        printf("\nВведите количество вершин графа: ");
        if (fgets(input, sizeof(input), stdin) == NULL) {
            printf("Ошибка чтения ввода.\n");
            continue;
        }
        for (int i = 0; input[i] != '\0'; i++) {
            if (input[i] == '\n') {
                input[i] = '\0';
                break;
            }
        }
        if (!is_positive_integer(input)) {
            printf("Ошибка: введите положительное целое число!\n");
            continue;
        }
        num = atoi(input);
        if (num > MAX_VERTICES) {
            printf("Ошибка: максимум %d вершин!\n", MAX_VERTICES);
            continue;
        }
        break;
```

```
    }
    return num;
}

// Ввод стартовой вершины
int get_start_vertex() {
    char input[100];
    int start;
    while (1) {
        printf("Введите номер начальной вершины (0–%d): ", n - 1);
        if (fgets(input, sizeof(input), stdin) == NULL) {
            printf("Ошибка ввода.\n");
            continue;
        }
        for (int i = 0; input[i] != '\0'; i++)
            if (input[i] == '\n') { input[i] = '\0'; break; }

        if (!is_positive_integer(input)) {
            printf("Ошибка: введите неотрицательное целое число!\n");
            continue;
        }
        start = atoi(input);
        if (start < 0 || start >= n) {
            printf("Ошибка: вершина вне диапазона!\n");
            continue;
        }
        break;
```

```
    }
    return start;
}

// Генерация неориентированного графа (петли разрешены)
void generateAdjacencyMatrix(int n) {
    srand(time(NULL));

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int edge = rand() % 2;
            adjMatrix[i][j] = edge;
            adjMatrix[j][i] = edge;
        }
    }
}

// Вывод матрицы смежности
void printAdjacencyMatrix() {
    printf("\nМатрица смежности (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%d ", adjMatrix[i][j]);
        printf("\n");
    }
}
```

```
// Построение списков смежности
void buildAdjacencyLists() {
    for (int i = 0; i < n; i++) {
        listSize[i] = 0;
        for (int j = 0; j < n; j++) {
            if (adjMatrix[i][j] == 1) {
                adjList[i][listSize[i]++] = j;
            }
        }
    }
}

// Вывод списков смежности
void printAdjacencyLists() {
    printf("\nСписки смежности:\n");
    for (int i = 0; i < n; i++) {
        printf("%d: ", i);
        for (int j = 0; j < listSize[i]; j++)
            printf("%d ", adjList[i][j]);
        printf("\n");
    }
}

// Рекурсивный DFS по матрице
void DFS_matrix_recursive(int v) {
    visited[v] = 1;
    printf("%d ", v);
```

```
    for (int i = 0; i < n; i++) {
        if (adjMatrix[v][i] == 1 && !visited[i])
            DFS_matrix_recursive(i);
    }
}

// Нерекурсивный DFS по матрице
void DFS_matrix_iterative(int start) {
    int stack[MAX_VERTICES];
    int top = -1;

    visited[start] = 1;
    stack[++top] = start;

    while (top >= 0) {
        int v = stack[top--];
        printf("%d ", v);

        for (int i = n - 1; i >= 0; i--) {
            if (adjMatrix[v][i] == 1 && !visited[i]) {
                visited[i] = 1;
                stack[++top] = i;
            }
        }
    }
}
```

```
// Рекурсивный DFS по спискам смежности
void DFS_list_recursive(int v) {
    visited[v] = 1;
    printf("%d ", v);

    for (int i = 0; i < listSize[v]; i++) {
        int neighbor = adjList[v][i];
        if (!visited[neighbor])
            DFS_list_recursive(neighbor);
    }
}

// === ОБХОДЫ (ПОГ и от выбранной вершины) ===

// Полный обход (рекурсивный, матрица)
void POG_matrix_recursive() {
    for (int i = 0; i < n; i++) visited[i] = 0;
    printf("Рекурсивный обход (все компоненты, матрица): ");
    for (int v = 0; v < n; v++)
        if (!visited[v])
            DFS_matrix_recursive(v);
    printf("\n");
}

// Полный обход (нерекурсивный, матрица)
void POG_matrix_iterative() {
```

```
    for (int i = 0; i < n; i++) visited[i] = 0;
    printf("Нерекурсивный обход (все компоненты, матрица): ");
    for (int v = 0; v < n; v++)
        if (!visited[v])
            DFS_matrix_iterative(v);
    printf("\n");
}

// Обход из выбранной вершины (рекурсивный, матрица)
void DFS_matrix_recursive_from_start() {
    int start = get_start_vertex();
    for (int i = 0; i < n; i++) visited[i] = 0;

    printf("Рекурсивный DFS, начиная с вершины %d: ", start);
    DFS_matrix_recursive(start);
    printf("\n");
}

// Обход из выбранной вершины (нерекурсивный, матрица)
void DFS_matrix_iterative_from_start() {
    int start = get_start_vertex();
    for (int i = 0; i < n; i++) visited[i] = 0;

    printf("Нерекурсивный DFS, начиная с вершины %d: ", start);
    DFS_matrix_iterative(start);
    printf("\n");
}
```

```
// Обход из выбранной вершины (списки смежности)
void POG_list_recursive_from_start() {
    buildAdjacencyLists();
    int start = get_start_vertex();
    for (int i = 0; i < n; i++) visited[i] = 0;

    printf("Рекурсивный DFS (списки смежности), начиная с вершины %d: ", start);
    DFS_list_recursive(start);
    printf("\n");
}

// Меню
int menu() {
    int choice;
    printf("\n===== МЕНЮ =====\n");
    printf("1. Сгенерировать граф\n");
    printf("2. Показать матрицу смежности\n");
    printf("3. Построить и показать списки смежности\n");
    printf("4. Обход в глубину с нулевой вершины (матрица, рекурсивно)\n");
    printf("5. Обход в глубину с нулевой вершины (матрица, нерекурсивно)\n");
    printf("6. Обход из выбранной вершины (матрица, рекурсивно)\n");
    printf("7. Обход из выбранной вершины (матрица, нерекурсивно)\n");
    printf("8. Обход из выбранной вершины (списки смежности)\n");
    printf("9. Выход\n");
    printf("Выберите пункт: ");
```

```
    if (scanf("%d", &choice) != 1) {
        while (getchar() != '\n');
        return 0;
    }
    while (getchar() != '\n');
    return choice;
}

int main() {
    setlocale(LC_ALL, "rus");

    while (1) {
        int choice = menu();

        switch (choice) {
        case 1:
            n = get_vertex_count();
            generateAdjacencyMatrix(n);
            generated = 1;
            printf("Граф с %d вершинами успешно сгенерирован!\n", n);
            break;

        case 2:
            if (!generated)
                printf("Сначала сгенерируйте граф!\n");
            else
                printAdjacencyMatrix();
```

```
    break;

case 3:
    if (!generated)
        printf("Сначала сгенерируйте граф!\n");
    else {
        buildAdjacencyLists();
        printAdjacencyLists();
    }
    break;

case 4:
    if (!generated)
        printf("Сначала сгенерируйте граф!\n");
    else
        POG_matrix_recursive();
    break;

case 5:
    if (!generated)
        printf("Сначала сгенерируйте граф!\n");
    else
        POG_matrix_iterative();
    break;

case 6:
    if (!generated)
```

```
            printf("Сначала сгенерируйте граф!\n");
        else
            DFS_matrix_recursive_from_start();
        break;

    case 7:
        if (!generated)
            printf("Сначала сгенерируйте граф!\n");
        else
            DFS_matrix_iterative_from_start();
        break;

    case 8:
        if (!generated)
            printf("Сначала сгенерируйте граф!\n");
        else
            POG_list_recursive_from_start();
        break;

    case 9:
        printf("Выход из программы...\n");
        return 0;

    default:
        printf("Ошибка: выберите пункт от 1 до 9.\n");
    }
}
```

```
    return 0;
};
```