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 22ВВВ3:

Шнайдер К. С.

Тельнов И. В.

Городничев М. И.

Приняли:

Юрова О. В.

Акифьев И. В.

Пенза 2023

**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.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <malloc.h>

#include <stdbool.h>

#include <stack>

using namespace std;

struct Node

{

int data;

Node\* next;

};

void print\_adjacency\_list(Node\*\* adjacencyList, int num\_vertices)

{

Node\* current;

for (int i = 0; i < num\_vertices; i++)

{

printf("Вершина %d примыкает к: ", i);

current = adjacencyList[i];

while (current != NULL)

{

printf("%d ", (current->data));

current = current->next;

}

printf("\n");

}

}

// Функция для преобразования матрицы смежности в список смежности с обратным порядком соседних вершин

Node\*\* adjacency\_matrix\_to\_adjacency\_list(int\*\* matrix, int num\_vertices)

{

Node\*\* adjacencyList = new Node \* [num\_vertices];

for (int i = 0; i < num\_vertices; i++)

{

adjacencyList[i] = nullptr; // Инициализируем список для вершины i

for (int j = num\_vertices - 1; j >= 0; j--) // Обходим в обратном порядке для добавления соседних вершин в обратном порядке

{

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

{

Node\* newNode = new Node;

newNode->data = j;

newNode->next = adjacencyList[i];

adjacencyList[i] = newNode;

}

}

}

print\_adjacency\_list(adjacencyList, num\_vertices);

return adjacencyList;

}

void free\_adjacency\_list(Node\*\* adjacencyList, int num\_vertices)

{

for (int i = 0; i < num\_vertices; i++)

{

Node\* current = adjacencyList[i];

while (current != NULL)

{

Node\* next = current->next;

free(current);

current = next;

}

}

free(adjacencyList);

}

// Функция поиска в глубину на основе матрицы смежности

void DFS(int vertex, int num\_vertices, bool\* viseted, int\*\* graph, int& current\_index)

{

viseted[vertex] = true;

printf("%d ", vertex);

for (int i = 0; i < num\_vertices; i++)

{

if (graph[vertex][i] == 1 && viseted[i] == false)

{

DFS(i, num\_vertices, viseted, graph, current\_index);

}

}

}

// Функция для обхода графа в глубину (DFS) на основе списков смежности

void DFS\_list(Node\*\* adjacencyList, int vertex, bool\* visited, int\* result, int currentIndex)

{

visited[vertex] = true; // Помечаем вершину как посещенную

result[currentIndex++] = vertex;

// Перебираем соседние вершины, представленные в списке смежности

Node\* current = adjacencyList[vertex];

while (current != nullptr) {

int neighbor = current->data;

if (!visited[neighbor])

{

DFS\_list(adjacencyList, neighbor, visited, result, currentIndex); // Рекурсивно обходим соседнюю вершину

}

current = current->next;

}

}

// Функция для нерекурсивного обхода графа в глубину (DFS)

int\* nonRecursiveDFS(int\*\* graph, int num\_vertices, int startVertex)

{

bool\* visited = new bool[num\_vertices];

for (int i = 0; i < num\_vertices; i++)

{

visited[i] = false;

}

int\* result = new int[num\_vertices];

int currentIndex = 0;

stack<int> vertexStack;

vertexStack.push(startVertex);

while (!vertexStack.empty())

{

int currentVertex = vertexStack.top();

vertexStack.pop();

if (!visited[currentVertex])

{

visited[currentVertex] = true;

result[currentIndex++] = currentVertex;

for (int i = num\_vertices - 1; i >= 0; i--)

{

if (graph[currentVertex][i] == 1 && !visited[i])

{

vertexStack.push(i);

}

}

}

}

delete[] visited;

return result;

}

// Основная функция

int main()

{

setlocale(LC\_ALL, "RUS");

int num\_vertices;

// Ввод количества вершин графа

printf("Введите количество вершин графа(Пожалуйста введите целое число): ");

if (scanf("%d", &num\_vertices) != 1)

{

printf("Ошибка");

return 0;

}

// Расширение памяти для матрицы смежности

int\*\* graph = (int\*\*)malloc(num\_vertices \* sizeof(int\*));

for (int i = 0; i < num\_vertices; i++)

{

graph[i] = (int\*)malloc(num\_vertices \* sizeof(int));

}

bool\* viseted = new bool[num\_vertices];

srand(time(NULL));

// Генерируем случайную матрицу смежности

for (int i = 0; i < num\_vertices; i++)

{

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

{

if (i == j)

{

graph[i][j] = 0; // Заполняем обратную диагональ 0

}

else

{

graph[i][j] = graph[j][i] = rand() % 2; // Случайным образом присваиваем 0 или 1

}

}

}

// Распечатываем матрицу смежности

printf("\nМатрица смежности:\n");

for (int i = 0; i < num\_vertices; i++)

{

for (int j = 0; j < num\_vertices; j++)

{

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

}

printf("\n");

}

// Инициализируем посещенный массив

for (int i = 0; i < num\_vertices; i++)

{

viseted[i] = false;

}

int vertex;

printf("Введите начальный узел для обхода DFS: ");

scanf("%d", &vertex);

printf("\nСписок смежности:\n");

Node\*\* adjacencyList = adjacency\_matrix\_to\_adjacency\_list(graph, num\_vertices);

int\* dfsResult = new int[num\_vertices];

int currentIndex = 0;

int\* dfsResult2 = new int[num\_vertices];

int currentIndex2 = 0;

printf("\nОбход поиска в глубину по матрице смежности: ");

DFS(vertex, num\_vertices, viseted, graph, currentIndex);

DFS\_list(adjacencyList, vertex, viseted, dfsResult2, currentIndex2);

printf("\n\nОбход в глубину по спискам смежности: ");

int\* dfsResult1 = nonRecursiveDFS(graph, num\_vertices, vertex);

for (int i = 0; i < num\_vertices; i++)

{

printf("%d ", dfsResult1[i]);

}

printf("\n\nРезультат нерекурсивного обхода в глубину: ");

for (int i = 0; i < num\_vertices; i++)

{

printf("%d ", dfsResult1[i]);

}

// Освобождаем память матрицы смежности

for (int i = 0; i < num\_vertices; i++)

{

free(graph[i]);

}

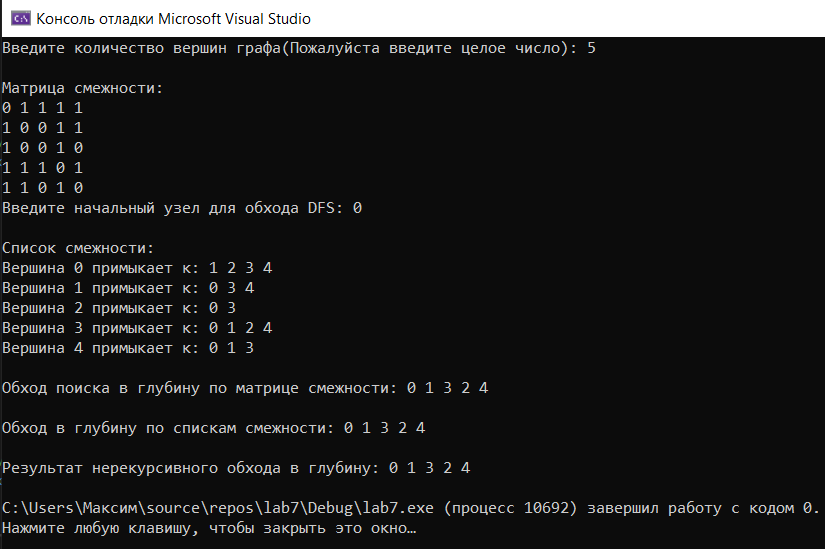
free(graph);

delete[] viseted;

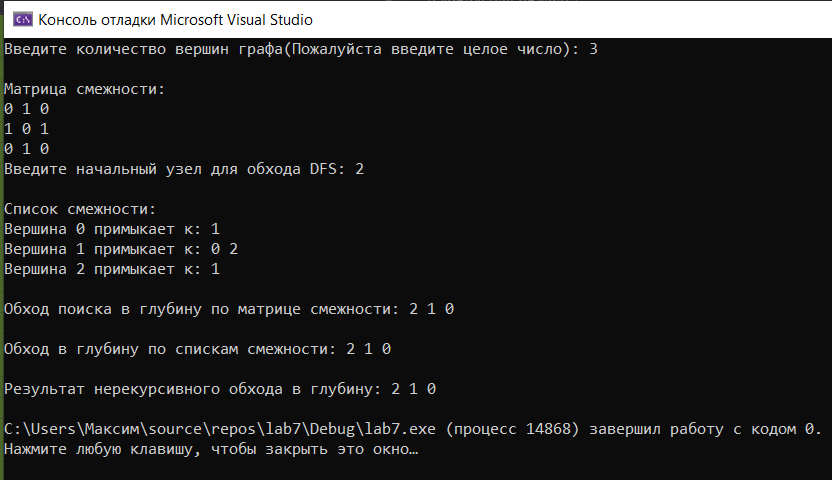
return 0;

}

**Результаты работы программ**



**Рисунок 1 - Результат работы программы**



**Рисунок 2 - Результат работы программы**

**Вывод**

В результате выполнения лабораторной работы была сгенерирована, используя генератор случайных чисел, матрица смежности для неориентированного графа G, вывели матрицу на экран. Для сгенерированного графа осуществили процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием, реализовали процедуру обхода в глубину для графа, представленного

списками смежности, для матричной формы представления графов выполнили преобразование рекурсивной реализации обхода графа к не рекурсивной.