Министерство науки и высшего образования 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 22ВВП2: Бормотов А.А.  
Кочетков А.М  
Кузьмин Д.В

Приняли

Акифьев И.В. Юрова О.В.

Пенза 2023

# 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 нерекурсивной.

# Выполнение задания 2

Для нерекурсивной реализации мы использовали обход графа в глубину с использованием стека.

Преимущество нерекурсивного обхода в глубину с использованием стека заключается в том, что он более эффективен с точки зрения использования памяти и не ограничен глубиной стека вызовов. Этот метод также легко адаптируется для обхода графов с большим количеством вершин.

# Листинг

#include <iostream> #include <ctime> #include <cstdlib> #include <stack>

using namespace std; struct Node {

int data; Node\* next;

};

//функция для печати списка смежности

void printAdjacencyList(Node\*\* adjacencyList, int size) { for (int i = 0; i < size; i++) {

cout << "Вершина " << i << " смежна с: "; Node\* current = adjacencyList[i];

while (current != nullptr) {

cout << current->data << " "; current = current->next;

}

cout << endl;

}

}

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

Node\*\* adjacencyMatrixToAdjacencyList(int\*\* matrix, int size) { Node\*\* adjacencyList = new Node\*[size];

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

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

вершины i

for (int j = size - 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;

}

}

}

printAdjacencyList(adjacencyList, size);

return adjacencyList;

}

// Функция для освобождения памяти, занятой для списков смежности void freeAdjacencyList(Node\*\* adjacencyList, int size) {

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

Node\* current = adjacencyList[i]; while (current != nullptr) {

Node\* next = current->next; delete current;

current = next;

}

}

delete[] adjacencyList;

}

// Функция для выделения памяти под двумерный массив int\*\* createMatrix(int size) {

int\*\* matrix = new int\*[size]; for (int i = 0; i < size; i++) {

matrix[i] = new int[size];

}

return matrix;

}

// Функция для освобождения памяти, выделенной под матрицу void deleteMatrix(int\*\* matrix, int size) {

for (int i = 0; i < size; i++) { delete[] matrix[i];

}

delete[] matrix;

}

// Функция для заполнения матрицы случайными значениями void fillMatrixRandom(int\*\* matrix, int size) {

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

for (int j = i; j < size; j++) { if (i == j) {

matrix[i][j] = 0;

}

else {

}

}

}

}

int randomValue = rand() % 2; matrix[i][j] = randomValue;

matrix[j][i] = randomValue; // Матрица симметрична

// Функция для вывода матрицы с элементами, разделенными запятой и пробелом

void printMatrix(int\*\* matrix, int size) { for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) { cout << matrix[i][j]; cout << ", ";

}

cout << endl;

}

}

// Функция для обхода графа в глубину (DFS)

void depthFirstSearch(int\*\* matrix, int size, int vertex, bool\* visited, int\* result, int& currentIndex) {

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

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

if (matrix[vertex][i] == 1 && !visited[i]) { depthFirstSearch(matrix, size, i, visited, result,

currentIndex);

}

}

}

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

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

}

current = current->next;

}

}

// Функция для нерекурсивного обхода графа в глубину (DFS) int\* nonRecursiveDFS(int\*\* matrix, int size, int startVertex) { bool\* visited = new bool[size];

for (int i = 0; i < size; i++) { visited[i] = false;

}

int\* result = new int[size]; 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 = size - 1; i >= 0; i--) {

if (matrix[currentVertex][i] == 1 && !visited[i]) { vertexStack.push(i);

}

}

}

}

delete[] visited; return result;

}

int main() { setlocale(LC\_ALL, "Rus"); srand(time(0));

int size;

cout << "\nВведите размер матрицы смежности: "; cin >> size;

int\*\* adjacencyMatrix = createMatrix(size); fillMatrixRandom(adjacencyMatrix, size); printMatrix(adjacencyMatrix, size);

bool\* visited = new bool[size]; for (int i = 0; i < size; i++) {

visited[i] = false; // Изначально все вершины помечаем как не

посещенные

}

// Преобразуем матрицу смежности в список смежности и выводим его cout << "\nСписок смежности:\n";

Node\*\* adjacencyList = adjacencyMatrixToAdjacencyList(adjacencyMatrix, size);

int startVertex;

cout << "Введите начальную вершину для обхода: "; cin >> startVertex;

int\* dfsResult = new int[size]; int currentIndex = 0;

depthFirstSearch(adjacencyMatrix, size, startVertex, visited, dfsResult, currentIndex);

cout << "Результат обхода в глубину матрицы смежности начиная с вершины " << startVertex << ": ";

for (int i = 0; i < currentIndex; i++) { cout << dfsResult[i] << " ";

}

cout << "\n";

int\* dfsResult1 = nonRecursiveDFS(adjacencyMatrix, size, startVertex);

cout << "Результат нерекурсивного обхода в глубину матрицы смежности начиная с вершины " << startVertex << ": ";

for (int i = 0; i < currentIndex; i++) { cout << dfsResult1[i] << " ";

}

cout << "\n";

visited = new bool[size];

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

visited[i] = false; // Изначально все вершины помечаем как не

посещенные

}

int\* dfsResult2 = new int[size]; int currentIndex2 = 0;

DFS(adjacencyList, startVertex, visited, dfsResult2, currentIndex2);

cout << "Результат обхода в глубину матрицы смежности начиная с вершины " << startVertex << ": ";

for (int i = 0; i < currentIndex2; i++) { cout << dfsResult2[i] << " ";

}

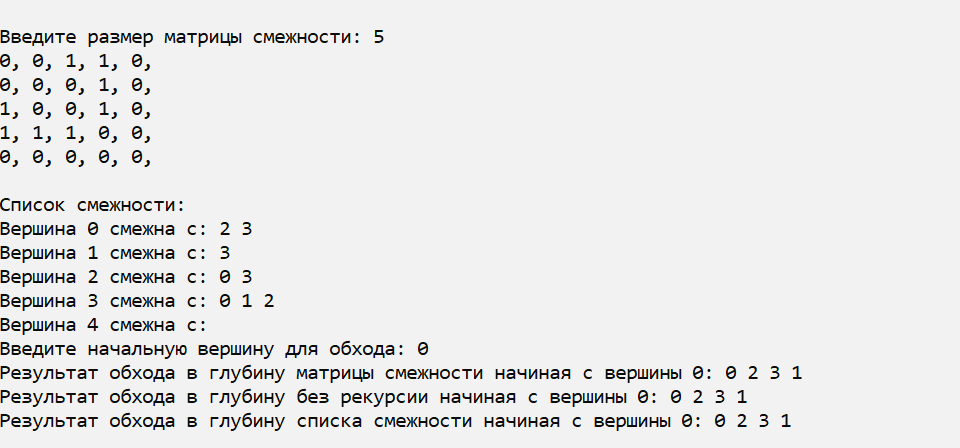
cout << "\n";

deleteMatrix(adjacencyMatrix, size); delete[] visited;

return 0;

}

# Результаты работы программы



**Вывод**

В ходе выполнения данной работы был изучен обход графа в глубину, как с использованием рекурсии, так и без нее. Мы изучили обходить графы в глубину на различных способах представления графов, включая матричную форму и списки смежности.