МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №9  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Поиск расстояний в графе»

Выполнили:

Студенты группы 23ВВВ2

Лисов Е.А.

Кочегин В.В.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Реализация алгоритма поиска расстояний в графе. Закрепление этих навыков посредством выполнения задания.

**Задания**

### Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

### Задание 2\*

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

**Ход работы**

**Задание 1.**

**1** Реализовали код генерации и вывода матрицы смежности неориентированного графа

vector<vector<int>> generateAdjacencyM(int n) {

vector<vector<int>> matrix(n, vector<int>(n, 0));

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

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

if (rand() % 2 == 1) {

matrix[i][j] = 1;

matrix[j][i] = 1;

}

}

}

return matrix;

}

void printMatrix(const vector<vector<int>>& matrix) {

cout << "Матрица смежности:" << endl;

for (const auto& row : matrix) {

for (int val : row) {

cout << val << " ";

}

cout << endl;

}

}

**2** Для сгенерированного графа осуществили процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди использовали класс **queue** из стандартной библиотеки С++

vector<int> bfsDistance(const vector<vector<int>>& G, int start) {

int n = G.size();

vector<int> DIST(n, -1);

queue<int> Q;

Q.push(start);

DIST[start] = 0;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

cout << v+1 << " ";

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

if (G[v][i] == 1 && DIST[i] == -1) {

Q.push(i);

DIST[i] = DIST[v] + 1;

}

}

}

return DIST;

}

**3\*** Реализовали процедуру поиска расстояний для графа, представленного списками смежности

void printAdjList(const vector<vector<int>>& adjList, int N) {

printf("Список смежности:\n");

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

printf("Вершина %d: ", i+1);

for (int j : adjList[i]) {

printf("%d ", j+1);

}

printf("\n");

}

}

void BFS(const vector<vector<int>>& adjList, int N, int v, vector<int>& DIST) {

DIST.assign(N, -1);

queue<int> Q;

Q.push(v);

DIST[v] = 0;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

for (int neighbor : adjList[v]) {

if (DIST[neighbor] == -1) {

Q.push(neighbor);

DIST[neighbor] = DIST[v] + 1;

}

}

}

}

**Задание 2\*.**

1. Реализовали процедуру поиска расстояний на основе обхода в глубину.

void DFS(const vector<vector<int>>& G, vector<int>& DIST, int v, int start, vector<bool>& visited) {

if (DIST[v] == -1 || start < DIST[v]) {

DIST[v] = start;

}

visited[v] = true;

cout << v + 1 << " ";

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

if (G[v][i] == 1 && !visited[i]) {

DFS(G, DIST, i, start + 1, visited);

}

}

visited[v] = false;

}

1. Реализовали процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.

void DFS(const vector<vector<int>>& adjList, vector<int>& DIST, int v, int currentDist, vector<bool>& visited) {

if (DIST[v] == -1 || currentDist < DIST[v]) {

DIST[v] = currentDist;

}

visited[v] = true;

for (int next : adjList[v]) {

if (!visited[next]) {

DFS(adjList, DIST, next, currentDist + 1, visited);

}

}

visited[v] = false;

1. Оценили время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

vector<int> bfsDistance(const vector<vector<int>>& G, int start) {

int n = G.size();

vector<int> DIST(n, -1);

queue<int> Q;

Q.push(start);

DIST[start] = 0;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

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

if (G[v][i] == 1 && DIST[i] == -1) {

Q.push(i);

DIST[i] = DIST[v] + 1;

}

}

}

return DIST;

}

### Результаты работы программы

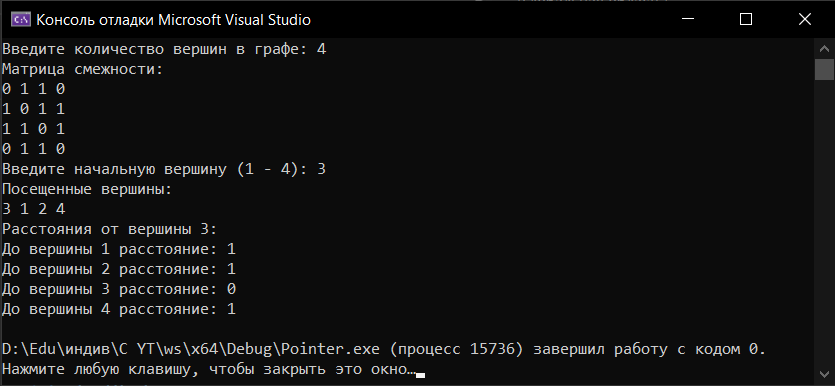


Рисунок 1 — Результаты работы программы для задания 1 без \*

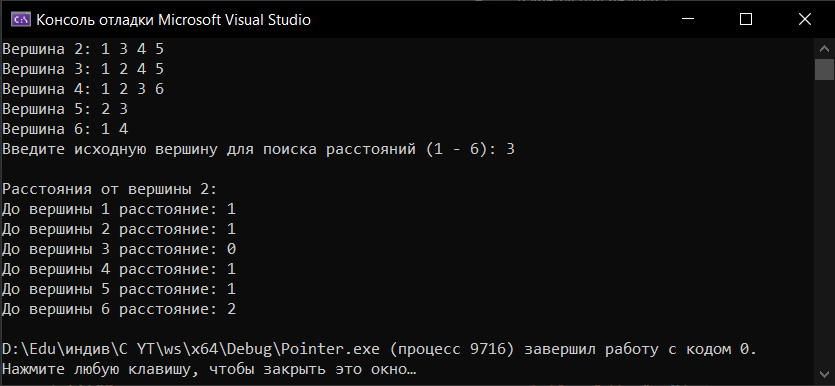
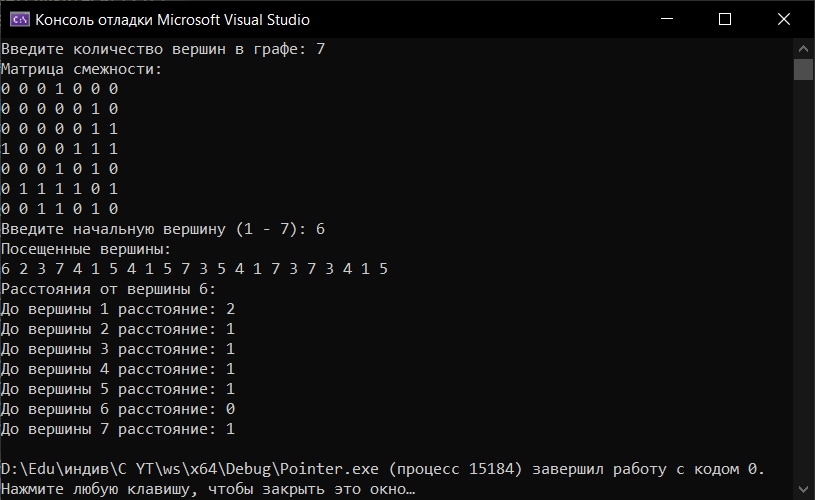
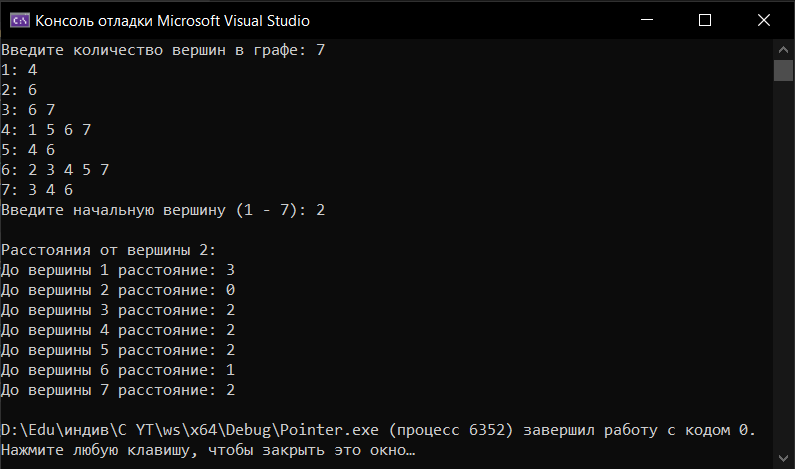
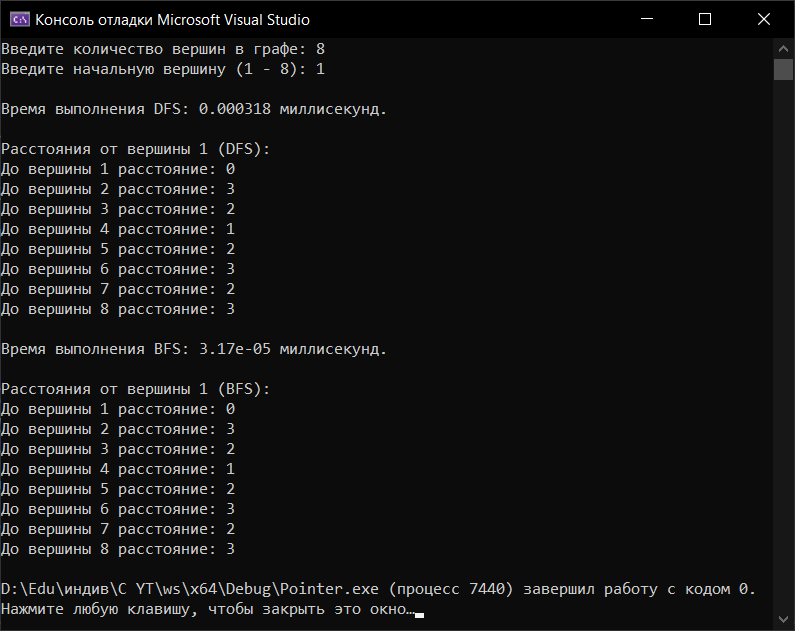


Рисунок 2 — Результаты работы программы для заданий 1 со \*

  
Рисунок 3 — Результаты работы программы для задания 2.1



**Рисунок 4 — Результаты работы программы для задания 2.2**

****

**Рисунок 5 — Результаты работы программы для задания 2.3**

### Вывод

В ходе выполнения работы были изучены принцип алгоритма поиска расстояний в графе и его реализация.