Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе № 9

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “Поиск расстояний в графе”

**Выполнил студент группы 22ВВС1:**

Пилюгин А.Э

**Приняли:**

Акифьев И. В.

Юрова О.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 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.

**Задание №9.1.2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <queue>

#include <clocale>

#define MAX\_NODES 400

// Функция для генерации случайной матрицы смежности

void generateRandomGraph(int graph[MAX\_NODES][MAX\_NODES], int n) {

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

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

if (i == j) {

graph[i][j] = 0; // Нет петель

}

else {

int random\_value = rand() % 2; // Генерируем 0 или 1

graph[i][j] = random\_value;

graph[j][i] = random\_value; // Граф неориентированный

}

}

}

}

// Функция для поиска расстояний в графе с использованием BFS

void bfs(int graph[MAX\_NODES][MAX\_NODES], int n, int startNode) {

bool visited[MAX\_NODES] = { false };

int distances[MAX\_NODES] = { 0 };

std::queue<int> q;

q.push(startNode);

visited[startNode] = true;

while (!q.empty()) {

int currentNode = q.front();

q.pop();

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

if (graph[currentNode][i] && !visited[i]) {

q.push(i);

visited[i] = true;

distances[i] = distances[currentNode] + 1;

}

}

}

// Выводим расстояния от startNode до всех остальных узлов

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

printf("Расстояние от ребра %d до ребра %d: %d\n", startNode, i, distances[i]);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL)); // Инициализируем генератор случайных чисел

int n; // Количество узлов в графе

printf("Введите количество ребер в графе: ");

scanf("%d", &n);

int graph[MAX\_NODES][MAX\_NODES] = { 0 };

generateRandomGraph(graph, n);

// Выводим матрицу смежности

printf("Матрица смежности графа:\n");

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

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

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

}

printf("\n");

}

int startNode;

printf("Введите начальный узел для поиска расстояний: ");

scanf("%d", &startNode);

clock\_t start = clock();

bfs(graph, n, startNode);

clock\_t end = clock();

double time\_spent5 = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("time adjmatrix for search in width: %f s.\n", time\_spent5);

return 0;

}

**Задание №9.1.3**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

int graph[MAX\_VERTICES][MAX\_VERTICES];

void addEdge(int from, int to) {

graph[from][to] = 1;

graph[to][from] = 1; // Учтем симметрию для неориентированного графа

}

void initializeGraph(int vertices) {

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

for (int j = 0; j < vertices; ++j) {

graph[i][j] = 0;

}

}

}

void printAdjacencyList(int vertices) {

printf("Список смежности:\n");

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

printf("%d: ", i);

for (int j = 0; j < vertices; ++j) {

if (graph[i][j] == 1) {

printf("%d ", j);

}

}

printf("\n");

}

}

void bfs(int startVertex, int vertices, int distances[]) {

bool visited[MAX\_VERTICES] = { false };

// Очередь для BFS

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

// Начинаем с начальной вершины

visited[startVertex] = true;

queue[rear++] = startVertex;

distances[startVertex] = 0;

while (front < rear) {

int currentVertex = queue[front++];

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

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

visited[i] = true;

queue[rear++] = i;

distances[i] = distances[currentVertex] + 1;

}

}

}

}

int main() {

int vertices, edges;

// Введите количество вершин и рёбер

printf("Введите количество вершин и рёбер графа: ");

scanf("%d %d", &vertices, &edges);

initializeGraph(vertices);

// Заполнение списка рёбрами автоматически

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

int from = rand() % vertices;

int to;

do {

to = rand() % vertices;

} while (from == to);

addEdge(from, to);

}

printAdjacencyList(vertices);

int startVertex;

// Введите начальную вершину

printf("Введите начальную вершину: ");

scanf("%d", &startVertex);

int distances[MAX\_VERTICES];

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

distances[i] = INT\_MAX;

}

bfs(startVertex, vertices, distances);

// Выведите расстояния от начальной вершины до всех остальных

printf("Расстояния от вершины %d до всех остальных (по обходу в ширину):\n", startVertex);

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

if (distances[i] == INT\_MAX) {

printf("До вершины %d: Недостижима\n", i);

}

else {

printf("До вершины %d: %d\n", i, distances[i]);

}

}

return 0;

}

**Задание 9.2.1.**

**#define** \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <clocale>

#include <time.h>

#define MAX\_VERTICES 400

#define INFINITY 99999

void generateRandomAdjacencyMatrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int numVertices, int numEdges) {

int i, j;

for (i = 0; i < numVertices; i++) {

for (j = 0; j < numVertices; j++) {

matrix[i][j] = 0;

}

}

int edgeCount = 0;

while (edgeCount < numEdges) {

int vertex1 = rand() % numVertices;

int vertex2 = rand() % numVertices;

if (vertex1 != vertex2 && matrix[vertex1][vertex2] == 0) {

matrix[vertex1][vertex2] = 1;

matrix[vertex2][vertex1] = 1;

edgeCount++;

}

}

}

void depthFirstSearch(int matrix[MAX\_VERTICES][MAX\_VERTICES], bool visited[MAX\_VERTICES], int distance[MAX\_VERTICES], int currentVertex, int numVertices) {

visited[currentVertex] = true;

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

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

distance[i] = distance[currentVertex] + 1; // Увеличиваем расстояние на 1

depthFirstSearch(matrix, visited, distance, i, numVertices);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int numVertices;

printf("Введите количество вершин в графе: ");

scanf("%d", &numVertices);

int numEdges;

printf("Введите количество ребер в графе: ");

scanf("%d", &numEdges);

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

generateRandomAdjacencyMatrix(adjacencyMatrix, numVertices, numEdges);

printf("Матрица смежности:\n");

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

for (int j = 0; j < numVertices; j++) {

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

}

printf("\n");

}

bool visited[MAX\_VERTICES];

int distance[MAX\_VERTICES];

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

visited[i] = false;

distance[i] = INFINITY; // Исходно все расстояния равны бесконечности

}

int startVertex;

printf("Введите стартовую вершину (от 0 до %d): ", numVertices - 1);

scanf("%d", &startVertex);

clock\_t start = clock();

if (startVertex >= 0 && startVertex < numVertices) {

distance[startVertex] = 0; // Расстояние от начальной вершины до самой себя равно 0

depthFirstSearch(adjacencyMatrix, visited, distance, startVertex, numVertices);

printf("Результат обхода в глубину, начиная с вершины %d:\n", startVertex);

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

printf("Расстояние от вершины %d до вершины %d: %d\n", startVertex, i, distance[i]);

}

}

else {

printf("Некорректный ввод стартовой вершины.\n");

}

clock\_t end = clock();

double time\_spent5 = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("time adjmatrix for search in depth: %f s.\n", time\_spent5);

return 0;

}

**Задание 9.2.2.**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

int graph[MAX\_VERTICES][MAX\_VERTICES];

void addEdge(int from, int to) {

graph[from][to] = 1;

graph[to][from] = 1; // Учтем симметрию для неориентированного графа

}

void initializeGraph(int vertices) {

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

for (int j = 0; j < vertices; ++j) {

graph[i][j] = 0;

}

}

}

void printAdjacencyList(int vertices) {

printf("Список смежности:\n");

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

printf("%d: ", i);

for (int j = 0; j < vertices; ++j) {

if (graph[i][j] == 1) {

printf("%d ", j);

}

}

printf("\n");

}

}

void dfs(int currentVertex, int vertices, bool visited[], int distances[]) {

visited[currentVertex] = true;

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

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

distances[i] = distances[currentVertex] + 1;

dfs(i, vertices, visited, distances);

}

}

}

void findDistancesDFS(int startVertex, int vertices, int distances[]) {

bool visited[MAX\_VERTICES] = { false };

distances[startVertex] = 0;

dfs(startVertex, vertices, visited, distances);

}

int main() {

int vertices, edges;

// Введите количество вершин и рёбер

printf("Введите количество вершин и рёбер графа: ");

scanf("%d %d", &vertices, &edges);

initializeGraph(vertices);

// Заполнение списка рёбрами автоматически

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

int from = rand() % vertices;

int to;

do {

to = rand() % vertices;

} while (from == to);

addEdge(from, to);

}

printAdjacencyList(vertices);

int startVertex;

// Введите начальную вершину

printf("Введите начальную вершину: ");

scanf("%d", &startVertex);

int distances[MAX\_VERTICES];

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

distances[i] = INT\_MAX;

}

findDistancesDFS(startVertex, vertices, distances);

// Выведите расстояния от начальной вершины до всех остальных

printf("Расстояния от вершины %d до всех остальных (по обходу в глубину):\n", startVertex);

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

if (distances[i] == INT\_MAX) {

printf("До вершины %d: Недостижима\n", i);

}

else {

printf("До вершины %d: %d\n", i, distances[i]);

}

}

return 0;

}

**Результаты работы программ**

Обход в ширину

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 1 - Результат работы программы LR91A**

**Обход в ширину по спискам смежности**

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 2 - Результат работы программы LR91B**

Для обхода в глубинуИзображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 3 - Результат работы программы LR92C**

Для обхода в глубину по спискам смежности

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 4 - Результат работы программы LR92D**

**Задание 9.2.3**

**Оценка сложности**

**Обход в глубину:**

**10 ребер:** 

**100 ребер:** 

**200 ребер:** 

**300 ребер:** 

**400 ребер**: 

**Обход в ширину:**

**10 ребер:** 

**100 ребер:** 

**200 ребер:** 

**300 ребер:** 

**400 ребер:** 

**Итог:** Судя по полученным данным, можем сделать вывод о том, что обход в ширину является более эффективным.

**Вывод**

В результате выполнения лабораторной работы были успешно реализованы алгоритмы построения матрицы смежности, усвоены и практически проверены навыки в алгоритме реализации поиска расстояний в графе методом обхода графа в ширину и в глубину, также сравнили скорость работы программы на разных размерах матриц и выявили какой из методов быстрее.