Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе № 10

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “ Поиск расстояний во взвешенном графе”

Выполнили студенты группы 22ВВС1:

Казаров И.И.

Эрмедов 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 взвешенного 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++.

**Задание 2**

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

**Задание №10**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <queue>

#include <clocale>

#include <limits.h>

#define MAX\_NODES 400

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

}

}

}

}

int minDistance(int dist[], int sptSet[], int V) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (sptSet[v] == 0 && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

return min\_index;

}

void dijkstra(int graph[MAX\_NODES][MAX\_NODES], int src, int V, int\* dist) {

int\* sptSet = (int\*)malloc(V \* sizeof(int));

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

dist[i] = INT\_MAX;

sptSet[i] = 0;

}

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet, V);

sptSet[u] = 1;

for (int v = 0; v < V; v++) {

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

free(sptSet);

}

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;

}

}

}

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

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

}

}

int findEccentricity(int graph[MAX\_NODES][MAX\_NODES], int n, int vertex) {

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

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

std::queue<int> q;

q.push(vertex);

visited[vertex] = 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;

}

}

}

int eccentricity = 0;

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

if (distances[i] > eccentricity) {

eccentricity = distances[i];

}

}

return eccentricity;

}

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

int minEccentricity = INT\_MAX;

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

int eccentricity = findEccentricity(graph, n, i);

if (eccentricity < minEccentricity) {

minEccentricity = eccentricity;

}

}

return minEccentricity;

}

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

int maxEccentricity = 0;

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

int eccentricity = findEccentricity(graph, n, i);

if (eccentricity > maxEccentricity) {

maxEccentricity = eccentricity;

}

}

return maxEccentricity;

}

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

printf("Центральные вершины:\n");

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

int eccentricity = findEccentricity(graph, n, i);

if (eccentricity == radius) {

printf("%d\n", i);

}

}

}

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

printf("Периферийные вершины:\n");

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

int eccentricity = findEccentricity(graph, n, i);

if (eccentricity == diameter) {

printf("%d\n", i);

}

}

}

bool isIsolatedGraph(int graph[MAX\_NODES][MAX\_NODES], int n) {

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

bool hasConnection = false;

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

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

hasConnection = true;

break;

}

}

if (!hasConnection) {

return true; // Найдена изолированная вершина

}

}

return false; // Граф не изолированный

}

int main() {

setlocale(LC\_ALL, "Russian");

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

int startNode = 0;

int radius;

int diameter;

int n; // Количество узлов в графе

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

scanf\_s("%d", &n);

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

generateRandomGraph(graph, n);

if (!isIsolatedGraph(graph, n)) {

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");

}

printf("Введите начальную вершину для поиска расстояний: ");

scanf\_s("%d", &startNode);

bfs(graph, n, startNode);

radius = findRadius(graph, n);

diameter = findDiameter(graph, n);

printf("Радиус графа: %d\n", radius);

printf("Диаметр графа: %d\n", diameter);

findCentralVertices(graph, n, radius);

findPeripheralVertices(graph, n, diameter);

}

else {

printf("Граф изолированный. Информация недоступна.\n");

}

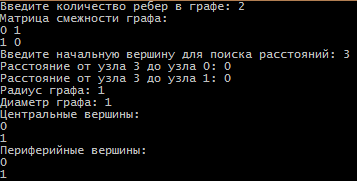
return 0;

}

**Результаты работы программ**



**Рисунок 1 - Результат работы программы**

****

**Рисунок 2 - Результат работы программы**

**Вывод**

В результате выполнения лабораторной работы были успешно реализованы алгоритмы построения матрицы смежности, усвоены и практически проверены навыки в алгоритме реализации поиска расстояний во взвешенном графе, а также реализован поиск диаметра и радиуса графов, изучили и нашли подмножества периферийных и центральных вершин.