Федеральное государственное бюджетное образовательное учреждение высшего образования «Пензенский Государственный университет»

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №10

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему «Поиск расстояний во взвешенном графе»

Выполнили: ст. гр. 21ВВ1

Сурков А.А

Хорошильцев Е.А

Кривенков И.В

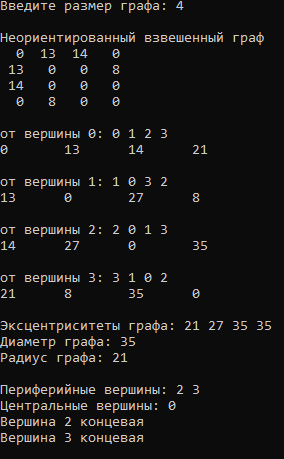
Приняли: Юрова О.В.

Акифьев И.В.

2022

**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++.



**Вывод:**

Выполняя данную лабораторную работу, мы научились реализовывать и применять алгоритма поиска расстояний в взвешенном графе.

**Листинг**

**Задание 1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

#include <queue>

using namespace std;

void adj(int\*\* G, int n) {

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

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

if (i == j) {

G[i][j] = 0;

}

if (i < j) {

if (rand() % 100 > 50) {

G[i][j] = rand() % 15;

}

else {

G[i][j] = 0;

}

G[j][i] = G[i][j];

}

}

}

}

void print\_adj(int\*\* G, int n) {

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

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

printf("%3d ", G[i][j]);

}

printf("\n");

}

printf("\n");

}

void BFSD(int s, int\*\* G, int\* dist, int n) {

queue <int> q;

q.push(s);

dist[s] = 0;

while (!q.empty()) {

s = q.front();

q.pop();

printf("%d ", s);

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

if ((G[s][i] != 0) && (dist[i] == 1000)) {

q.push(i);

dist[i] = dist[s] + G[s][i];

}

}

}

printf("\n");

}

int main() {

int\*\* G; // указатель на указатель на строку элементов

int\* ex; // эксцентриситет

int\* dist, \*step;

int i, j, n, max = 0, min = 0;

setlocale(LC\_ALL, "rus");

srand(time(NULL));

printf("Введите размер графа: ");

scanf("%d", &n);

printf("\n");

G = (int\*\*)malloc(n \* sizeof(int\*));

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

G[i] = (int\*)malloc(n \* sizeof(int));

}

dist = (int\*)malloc(n \* sizeof(int));

ex = (int\*)malloc(n \* sizeof(int));

step = (int\*)malloc(n \* sizeof(int));

printf("Неориентированный взвешенный граф \n");

adj(G, n);

print\_adj(G, n);

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

max = -1;

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

dist[j] = 1000;

}

printf("от вершины %d: ", i);

BFSD(i, G, dist, n);

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

{

if (dist[j] != 1000) {

printf("%d\t", dist[j]);

if (dist[j] > max)

max = dist[j];

}

else

printf("нет прохода\t");

}

ex[i] = max;

printf("\n\n");

}

printf("Эксцентриситеты графа: ");

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

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

}

min = ex[0];

max = ex[0];

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

if (ex[i] > max)

max = ex[i];

if (ex[i] < min)

min = ex[i];

}

printf("\nДиаметр графа: %d\t", max);

printf("\n");

printf("Радиус графа: %d\n\n", min);

printf("Периферийные вершины: ");

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

if (ex[i] == max) {

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

}

}

printf("\nЦентральные вершины: ");

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

if (ex[i] == min) {

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

}

}

printf("\n");

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

step[i] = 0;

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

if (G[i][j] != 0)

step[i]++;

}

switch (step[i]) {

case 0: printf("Вершина %d изолированая\n", i);

break;

case 1:printf("Вершина %d концевая\n", i );

break;

default:

if (step[i] == n-1)

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

}

}

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

free(G[i]);

}

free(G);

free(ex);

free(dist);

free(step);

getchar();

getchar();

}