Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №10  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Поиск расстояний во взвешенном графе»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Поиск расстояний во взвешенном графе

**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++.
3. **\***Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа G. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

**Задание 2**

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

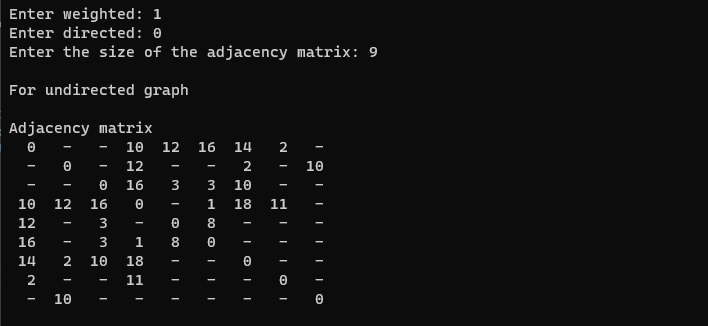
**Задание 3\***

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже).  В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

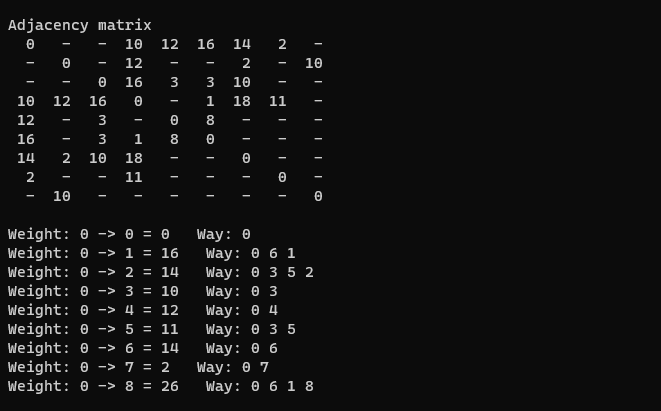
**Ход работы**

**Задание 1**

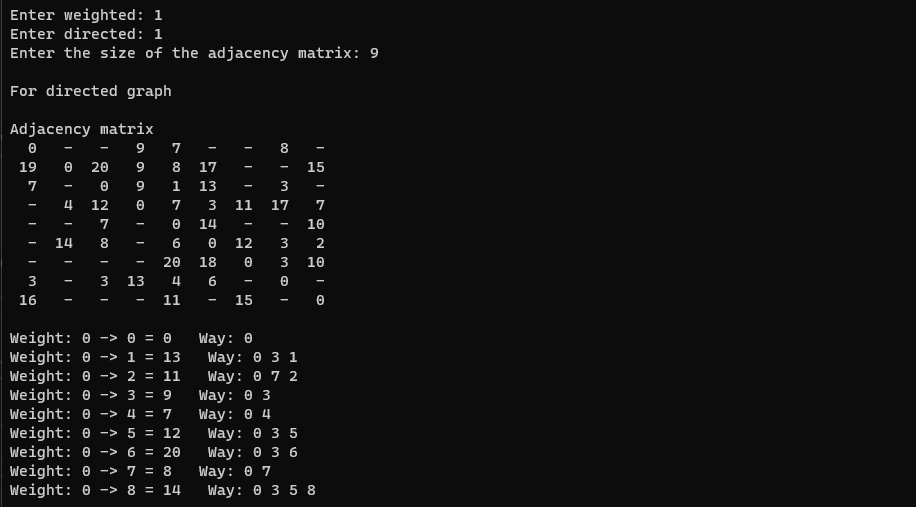
1. Сгенерировал (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа G. Вывел матрицу на экран.



1. Для сгенерированного графа осуществил процедуру поиска расстояний.

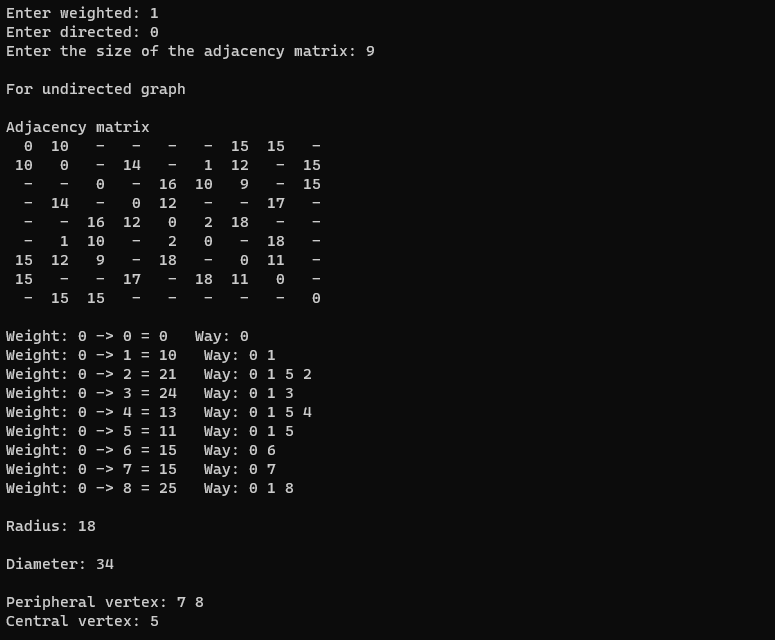


1. **\***Сгенерировал (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа G. Вывел матрицу на экран и осуществил процедуру поиска расстояний.



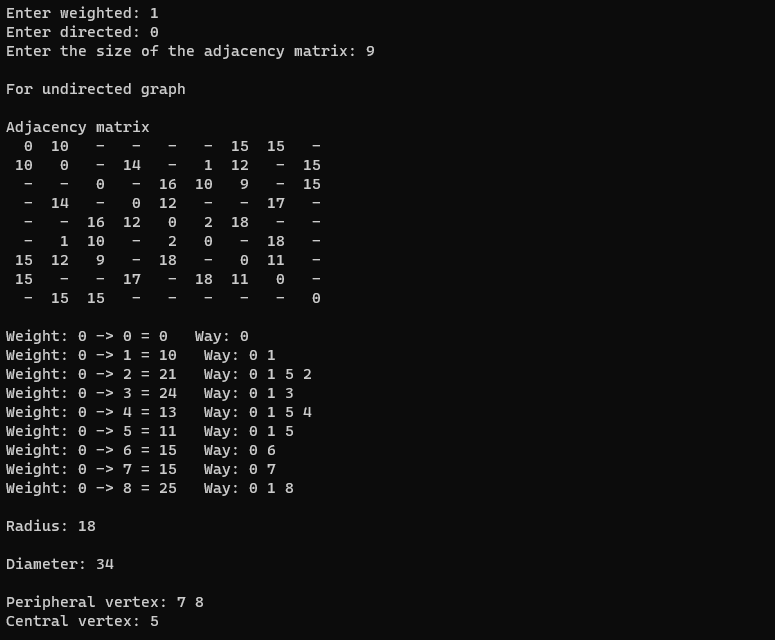
**Задание 2**

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определил радиус и диаметр.





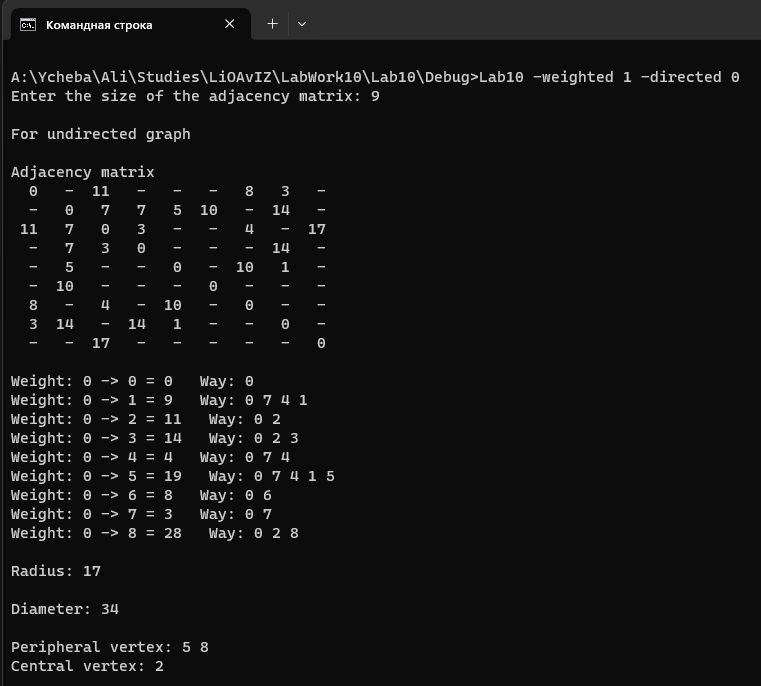
1. Определил подмножества периферийных и центральных вершин.

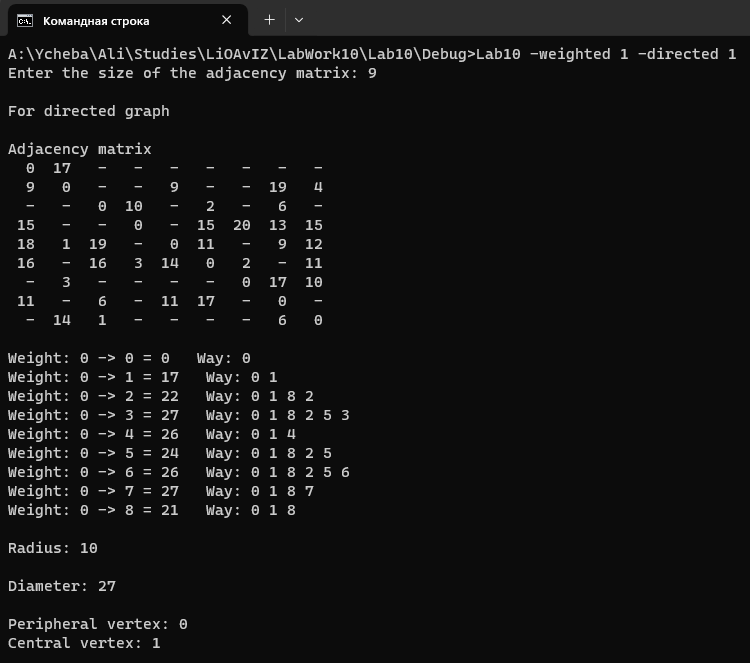




**Задание 3\***

1. Модернизировал программу так, чтобы получить возможность запуска программы с параметрами командной.





**Листинг**

* Header.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <queue>

#include <iostream>

#include <string>

using namespace std;

#define MAX 2147483647

void fillMatrix(int\*\* matrix, int size, int directed, int weighted);

int\* fillArreyDistance(int\* arreyDistance, int size);

void dijkstraAlgorithm(int\* arreyDistance, int\* predok, int\*\* matrix, int size, int vertex);

int getDiameter(int\*\* matrix, int size);

int getRadius(int\*\* matrix, int size);

int\* getEccentricityArrey(int\*\* matrix, int size);

void printMatrix(int\*\* matrix, int size);

void printArr(int\* arreyDistance, int size);

void printWay(int\* arr, int size, int vertex);

void printShortestPath(int\* arreyDistance, int size, int\* ancestor, int vertex);

void printDiameter(int\*\* matrix, int size);

void printRadius(int\*\* matrix, int size);

void printPeripheralVertex(int\*\* matrix, int size);

void printCentralVertex(int\*\* matrix, int size);

pair<int, int> getWeightedAndDirected(int argc, char\* argv[]);

* Lab10.cpp

#include "Header.h"

int main(int argc, char\* argv[])

{

srand(time(NULL));

int sizeMatrix, start = 0, end = 0;

int\*\* matrix;

int vertex = 0;

int weighted = 1;

int directed = 1;

pair<int, int> weightedAndDirected = getWeightedAndDirected(argc, argv);

weighted = weightedAndDirected.first;

directed = weightedAndDirected.second;

printf("Enter the size of the adjacency matrix: ");

scanf("%d", &sizeMatrix);

int\* ancestor = (int\*)calloc(sizeMatrix, sizeMatrix \* sizeof(int));

int\* arreyDistance = (int\*)calloc(sizeMatrix, sizeMatrix \* sizeof(int));

arreyDistance = fillArreyDistance(arreyDistance, sizeMatrix);

matrix = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

for (int i = 0; i < sizeMatrix; ++i)

{

matrix[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

}

if(directed == 0)

printf("\nFor undirected graph\n");

else

printf("\nFor directed graph\n");

printf("\nAdjacency matrix\n");

fillMatrix(matrix, sizeMatrix, directed, weighted);

printMatrix(matrix, sizeMatrix);

dijkstraAlgorithm(arreyDistance, ancestor, matrix, sizeMatrix, vertex);

printShortestPath(arreyDistance, sizeMatrix, ancestor, vertex);

printRadius(matrix, sizeMatrix);

printDiameter(matrix, sizeMatrix);

printPeripheralVertex(matrix, sizeMatrix);

printCentralVertex(matrix, sizeMatrix);

printf("\n");

for (int i = 0; i < sizeMatrix; i++)

free(matrix[i]);

free(matrix);

free(arreyDistance);

free(ancestor);

return 0;

}

* DijkstraAlgorithm.cpp

#include "Header.h"

void dijkstraAlgorithm(int\* arreyDistance, int\* ancestor, int\*\* matrix, int size, int vertex)

{

std::priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> queue;

arreyDistance[vertex] = 0;

ancestor[vertex] = vertex;

queue.push({ 0, vertex });

while (!queue.empty())

{

pair<int, int> c = queue.top();

queue.pop();

vertex = c.second;

int weight = c.first;

if (weight > arreyDistance[vertex])

continue;

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

{

if (matrix[vertex][i] != MAX)

{

int v = i;

int len = matrix[vertex][i];

if (arreyDistance[v] > arreyDistance[vertex] + len) {

ancestor[v] = vertex;

arreyDistance[v] = arreyDistance[vertex] + len;

queue.push({ arreyDistance[v], v });

}

}

}

}

}

* fillArreyDistance.cpp

#include "Header.h"

int\* fillArreyDistance(int\* arreyDistance, int size)

{

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

{

arreyDistance[i] = MAX;

}

return arreyDistance;

}

* fillMatrix.cpp

#include "Header.h"

void fillMatrix(int\*\* matrix, int size, int directed, int weighted)

{

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

{

for (int j = 0; j < size; j++)

{

if (i == j)

{

matrix[i][j] = 0;

continue;

}

if (rand() % 2 == 1)

{

if (weighted == 1)

matrix[i][j] = rand() % 20 + 1;

else

matrix[i][j] = 1;

if (directed == 0)

matrix[j][i] = matrix[i][j];

}

else

{

matrix[i][j] = MAX;

if (directed == 0)

matrix[j][i] = matrix[i][j];

}

}

}

}

* getDiameter.cpp

#include "Header.h"

int getDiameter(int\*\* matrix, int size)

{

int max = 0;

int\* arreyEccentricity = getEccentricityArrey(matrix, size);

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

{

if (max < arreyEccentricity[i])

max = arreyEccentricity[i];

}

free(arreyEccentricity);

return max;

}

* getEccentricityArrey.cpp

#include "Header.h"

int\* getEccentricityArrey(int\*\* matrix, int size)

{

int\* arreyEccentricity = (int\*)calloc(size, size \* sizeof(int));

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

{

int\* arreyDistance = (int\*)calloc(size, size \* sizeof(int));

arreyDistance = fillArreyDistance(arreyDistance, size);

int\* ancestor = (int\*)calloc(size, size \* sizeof(int));

dijkstraAlgorithm(arreyDistance, ancestor, matrix, size, i);

for (int j = 0; j < size; j++)

{

if (arreyEccentricity[i] < arreyDistance[j])

arreyEccentricity[i] = arreyDistance[j];

}

free(arreyDistance);

free(ancestor);

}

return arreyEccentricity;

}

* getRadius.cpp

#include "Header.h"

int getRadius(int\*\* matrix, int size)

{

int min = MAX;

int\* arreyEccentricity = getEccentricityArrey(matrix, size);

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

{

if (min > arreyEccentricity[i])

min = arreyEccentricity[i];

}

free(arreyEccentricity);

return min;

}

* getWeightedAndDirected.cpp

#include "Header.h"

pair<int, int> getWeightedAndDirected(int argc, char\* argv[])

{

int weighted = 1;

int directed = 1;

if (argc > 1)

{

for (int i = 0; i < argc; ++i)

{

if (strcmp(argv[i], "-weighted") == 0)

weighted = std::stoi(argv[i + 1]);

if (strcmp(argv[i], "-directed") == 0)

directed = std::stoi(argv[i + 1]);

}

}

else

{

printf("Enter weighted: ");

scanf("%d", &weighted);

getchar();

printf("Enter directed: ");

scanf("%d", &directed);

getchar();

}

return pair<int, int>(weighted, directed);

}

* printArr.cpp

#include "Header.h"

void printArr(int\* arreyDistance, int size)

{

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

{

if (arreyDistance[i] == MAX)

printf("- ", arreyDistance[i]);

else

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

}

}

* printCentralVertex.cpp

#include "Header.h"

void printCentralVertex(int\*\* matrix, int size)

{

int min = getRadius(matrix, size);

int\* arreyEccentricity = getEccentricityArrey(matrix, size);

printf("\nCentral vertex: ");

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

{

if (min == arreyEccentricity[i])

printf("%d ", i);

}

free(arreyEccentricity);

}

* printDiameter.cpp

#include "Header.h"

void printDiameter(int\*\* matrix, int size)

{

if (getDiameter(matrix, size) == MAX)

printf("\nDiameter: -\n");

else

printf("\nDiameter: %d\n", getDiameter(matrix, size));

}

* printMatrix.cpp

#include "Header.h"

void printMatrix(int\*\* matrix, int size)

{

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

{

for (int j = 0; j < size; j++)

{

if (matrix[i][j] == MAX)

printf("%3c ", '-');

else

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

}

printf("\n");

}

}

* printPeripheralVertex.cpp

#include "Header.h"

void printPeripheralVertex(int\*\* matrix, int size)

{

int max = getDiameter(matrix, size);

int\* arreyEccentricity = getEccentricityArrey(matrix, size);

printf("\nPeripheral vertex: ");

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

{

if (max == arreyEccentricity[i])

printf("%d ", i);

}

free(arreyEccentricity);

}

* printRadius.cpp

#include "Header.h"

void printRadius(int\*\* matrix, int size)

{

if (getRadius(matrix, size) == MAX)

printf("\nRadius: -\n");

else

printf("\nRadius: %d\n", getRadius(matrix, size));

}

* printShortestPath.cpp

#include "Header.h"

void printShortestPath(int\* arreyDistance, int size, int\* ancestor, int vertex)

{

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

{

if (arreyDistance[i] == MAX)

printf("\nWeight: %d -> %d = - ", vertex, i, arreyDistance[i]);

else

printf("\nWeight: %d -> %d = %d ", vertex, i, arreyDistance[i]);

printf("Way: ");

printWay(ancestor, size, i);

}

printf("\n");

}

* printWay.cpp

#include "Header.h"

void printWay(int\* arr, int size, int vertex)

{

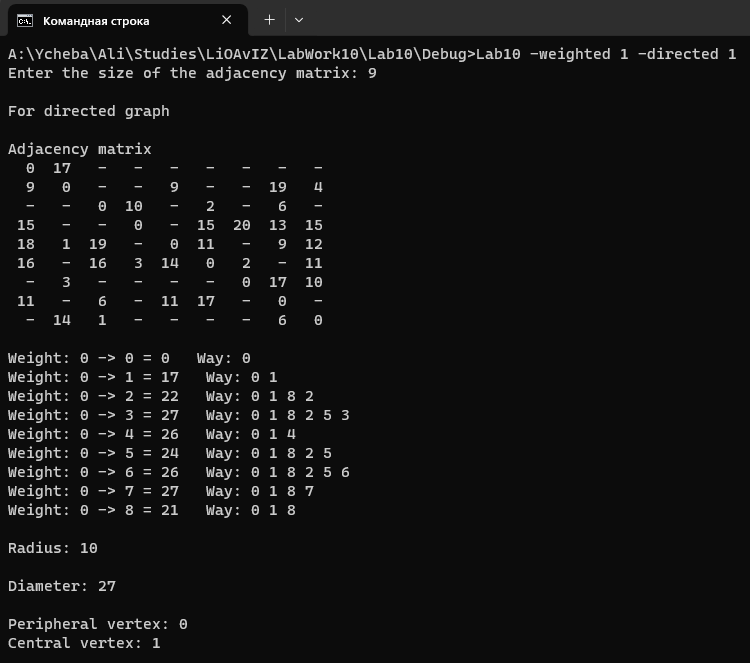
if (arr[vertex] != vertex)

printWay(arr, size, arr[vertex]);

printf("%d ", vertex);

}

**Результат работы программы**

****

**Вывод**

В результате выполнения лабораторной работы я научился выполнять поиск расстояний во взвешенном графе.