Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра “Вычислительная техника”

Отчёт

по лабораторной работе №10
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Поиск расстояний во взвешенном графе”

Выполнил студент гр. 22ВВВ3:
Кулахметов С.И.

Приняли:
к.т.н., доцент Юрова О.В.
к.э.н., доцент Акифьев И.В.

Пенза 2023

Цель работы

Реализовать алгоритм поиска расстояний между вершинами ориентированного графа, выполнить лабораторное задание.

Лабораторное задание

Задание 1

1. Сгенерировать (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Вывести матрицу на экран.

2. Для сгенерированного графа осуществить процедуру поиска расстояний, реализованную в соответствии с приведенным в методическом указании алгоритмом. При реализации алгоритма в качестве очереди использовать класс **queue** из стандартной библиотеки C++.

3. Сгенерировать (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Вывести матрицу на экран и осуществить процедуру поиска расстояний.

Задание 2

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

2. Определить подмножества периферийных и центральных вершин.

Задание 3

Модернизировать программу так, чтобы получить возможность запуска программы с параметрами терминала. В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

Пояснительный текст к прорамме

Программа разделена на 2 файла, которые отвечают за выполение различных частей лабораторной работы: 1 файл Task2.cpp — отвечает за выполнение первого и второго заданий полностью; 2 файл Task3.cpp — отвечает за выполнение третьего задания полностью.

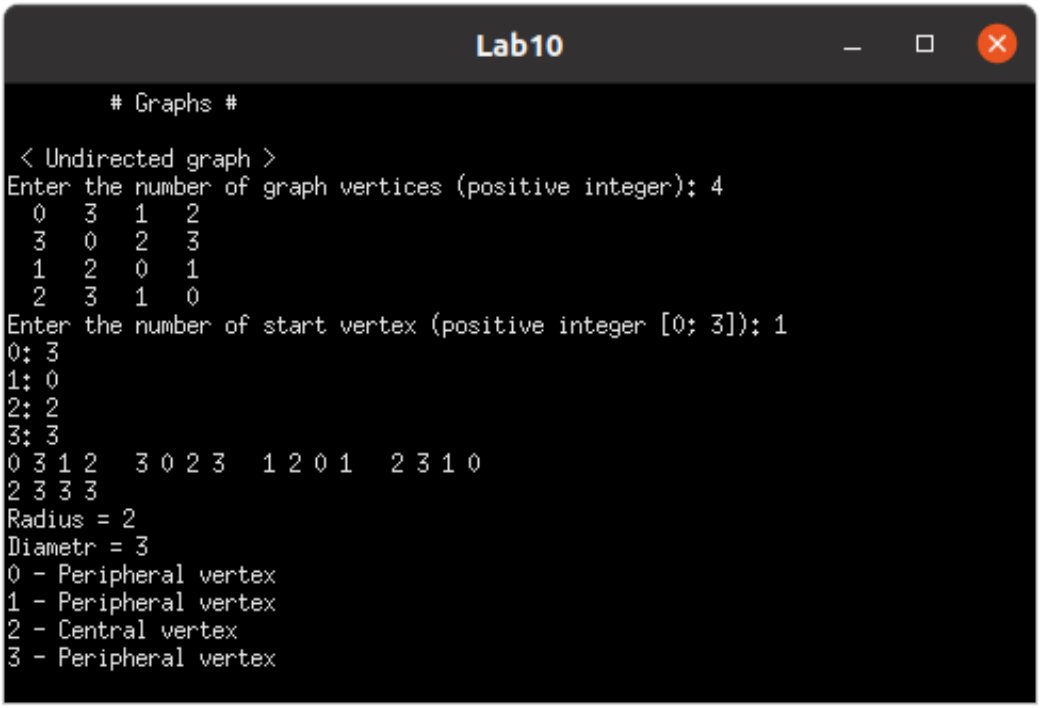
Логика работы программ аналогична лабораторной работе №9. Однако, касаемо задания №3, стоит отметить, что привычной Microsoft Visual Studio сборке проектов в исполняемый файл в операционных системах на базе GNU/Linux нет,

поэтому для сборки использовалась следующая команда, определяющая компилятор, ключ, выходной и входной файлы: `g++ -o prog main.cpp`.

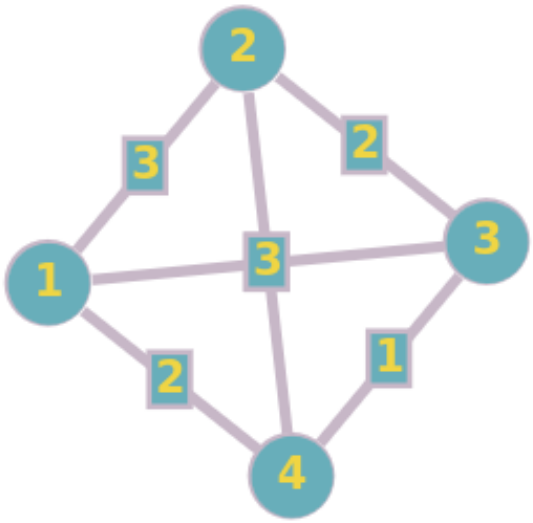
Результаты работы программы

Задания 1 и 2

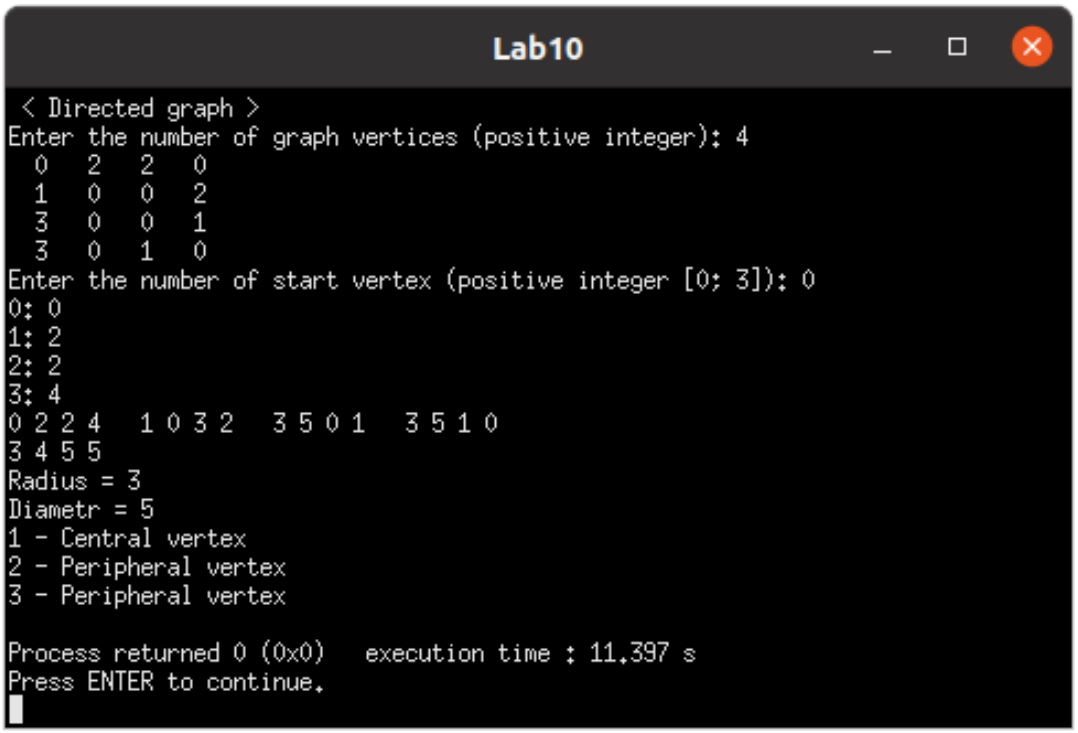
Построение матрицы смежности взвешенного неориентированного графа. Нахождение расстояний от заданной вершины, радиуса, диаметра, периферийных и центральных вершин.



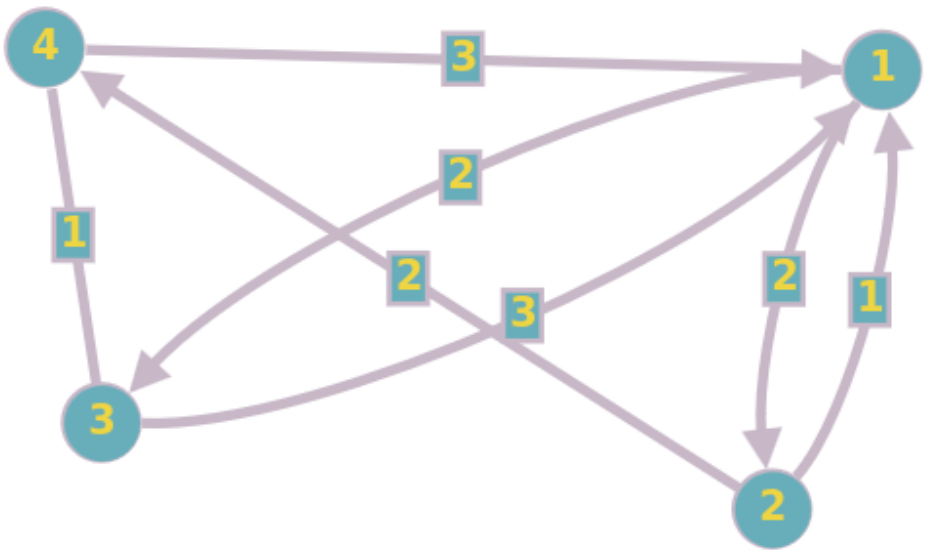
Граф, сгенерированный по матрице смежности.



Построение матрицы смежности взвешенного ориентированного графа. Нахождение расстояний от заданной вершины, радиуса, диаметра, периферийных и центральных вершин.

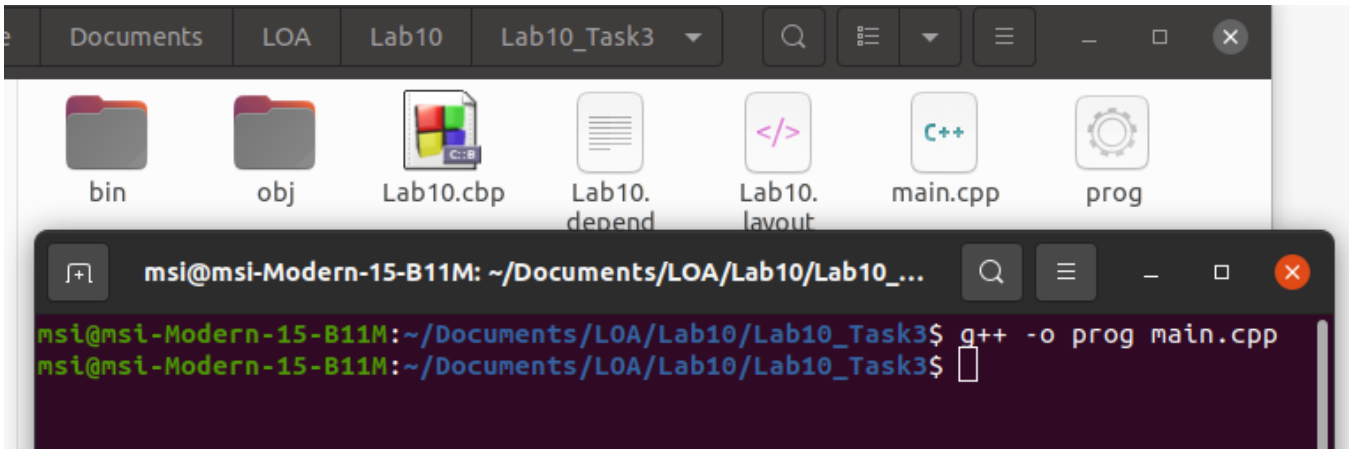


Граф, сгенерированный по матрице смежности.

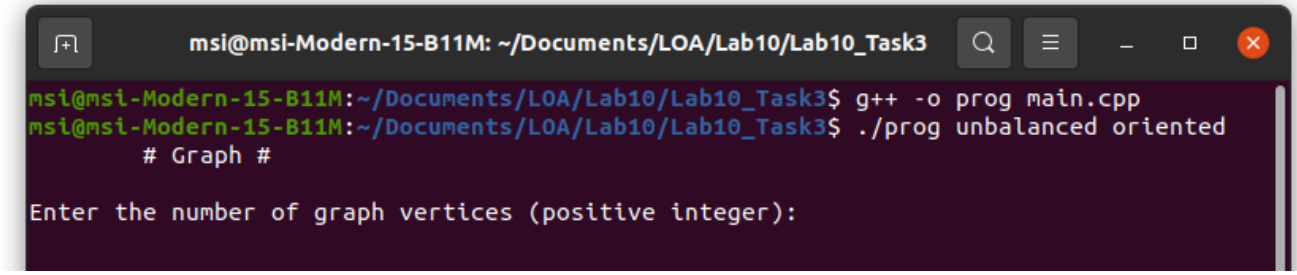


Задание 3

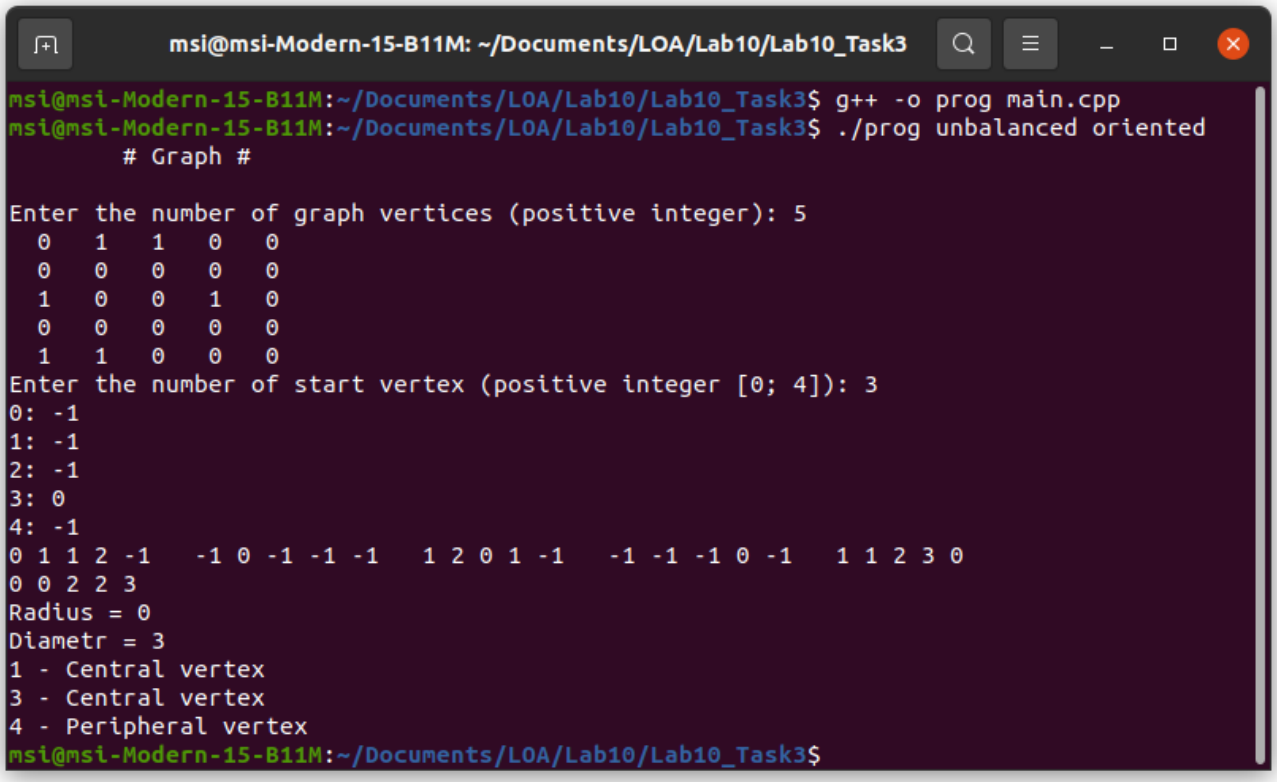
Сборка исполняемых файлов программы в терминале.



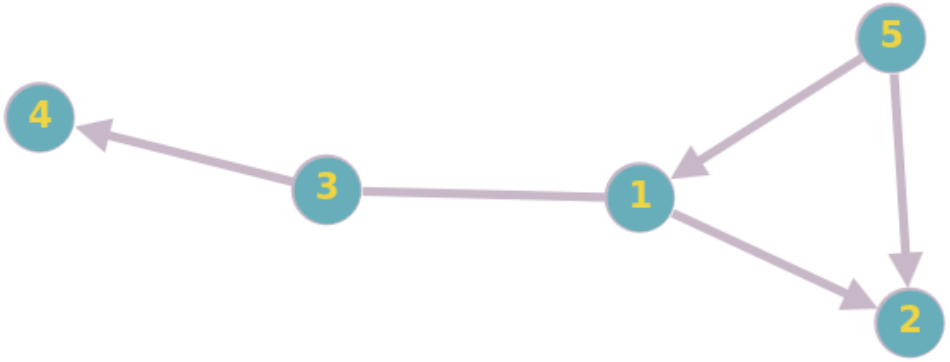
Запускаем программу, передав ей произвольные параметры из списка (Комментарии к коду см. Приложение Б)



Теперь вводим количество вершин графа и начальную вершину, относительно которой происходит поиск расстояний.



Граф, сгенерированный по матрице смежности.



Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации на языке C++ алгоритма поиска расстояний от указанной вершины в ориентированном взвешенном графе, представленном матрицей смежности.

Ссылка на *GitHub* репозиторий с лабораторной работой

https://github.com/KulakhmetovS/Lab10

Приложение А

Листинг программы

Файл Lab10_Task2/main.cpp

```
#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
#include <stdio.h>
#include <stdlib.h>


using namespace std;

int** Creategraph(int **, int); //Создание графа
int** CreategraphO(int **, int); //Создание ориентированного графа
void BFSD(int **, int *, int, int);  //Обход в ширину
void RadiusDiameter(int **, int *, int, int);
void Vertices(int **, int, int, int);

int size;   //Размер графа


int main()
{
   int i, j, v;
   int **graph = NULL, *DIST = NULL, **graphO = NULL, *DISTO = NULL;


   printf("\t# Graphs #\n\n < Undirected graph >\n");
   printf("Enter the number of graph vertices (positive integer): ");
   scanf("%d", &size);

   // Creating the graph
   graph = Creategraph(graph, size);
   DIST = (int *)(malloc(sizeof(int *) * size));  // Array for visited vertices
   DISTO = (int *)(malloc(sizeof(int *) * size));

   // Printing the matrix
   for(i = 0; i < size; i++)
   {
      DIST[i] = -1;
      DISTO[i] = -1;
     // visited[i] = 0;
      for(j = 0; j < size; j++)
          printf("%3d ", graph[i][j]);
      printf("\n");
   }

   int num = size - 1;

   printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
   scanf("%d", &v);

   // <---------- ! ---------->
   BFSD(graph, DIST, size, v);
   // <---------- ! ---------->
   //printf("\n");
```

```
    for(i = 0; i < size; i++)
        printf("%d: %d\n", i, DIST[i]);

    RadiusDiameter(graph, DIST, size, 0);

    printf("\n\n < Directed graph >\n");

    printf("Enter the number of graph vertices (positive integer): ");
    scanf("%d", &size);
    graphO = CreategraphO(graphO, size);

    for(i = 0; i < size; i++)
    {
        for(j = 0; j < size; j++)
            printf("%3d ", graphO[i][j]);
        printf("\n");
    }

    //for(i = 0; i < size; i++)
        //visited[i] = 0;

    printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
    scanf("%d", &v);

    // <---------- ! ---------->
    BFSD(graphO, DISTO, size, v);
    // <---------- ! ---------->
    //printf("\n");
    for(i = 0; i < size; i++)
        printf("%d: %d\n", i, DISTO[i]);

    RadiusDiameter(graphO, DISTO, size, 0);


    delete[] DISTO;
    delete[] graphO;
    //delete[] visited;
    delete[] graph;
    delete[] DIST;

    return 0;
}


int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 4;
```

```
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}

void BFSD(int **graph, int *DIST, int size, int v)
{
    queue<int> q;
    q.push(v);
    DIST[v] = 0;

    while(!q.empty())
    {
        v = q.front();
        //printf("%d ", v);
        q.pop();

        for(int i = 0; i < size; i++)
        {
            if((graph[v][i] > 0) && (DIST[i] == -1))
            {
                q.push(i);
                DIST[i] = DIST[v] + graph[v][i];
            }
        }
    }

}

int** CreategraphO(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = 0; j < size; j++)
        {
            graph[i][j] = rand() % 4;
            //graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}

void RadiusDiameter(int **graph, int *DIST, int size, int v)
{
    int i, j, temp;
    int *arr;
```

```
    arr = (int *)(malloc(sizeof(int *) * size));

    for(i = 0; i < size; i++)
        DIST[i] = -1;

    for(i = 0; i < size; i++)
    {
        BFSD(graph, DIST, size, i);

        for(j = 0; j < size; j++)
            printf("%d ", DIST[j]);
        printf("  ");

        // __________ Сортировка пузырьком __________
        for (int k = 0; k < size - 1; k++)
        {
            for (int n = 0; n < size - k - 1; n++) {
                if (DIST[n] > DIST[n + 1]) {
                    // меняем элементы местами
                    temp = DIST[n];
                    DIST[n] = DIST[n + 1];
                    DIST[n + 1] = temp;
                }
            }
        }
        // _____________________________________________

        arr[i] = DIST[size - 1];

        for(j = 0; j < size; j++)
            DIST[j] = -1;
        //printf("  ");

    }

      // __________ Сортировка пузырьком __________
        for (int k = 0; k < size - 1; k++)
        {
            for (int n = 0; n < size - k - 1; n++) {
                if (arr[n] > arr[n + 1]) {
                    // меняем элементы местами
                    temp = arr[n];
                    arr[n] = arr[n + 1];
                    arr[n + 1] = temp;
                }
            }
        }
        // _____________________________________________

    printf("\n");
    for(i = 0; i < size; i++)
        printf("%d ", arr[i]);

    printf("\nRadius = %d\nDiametr = %d\n", arr[0], arr[size - 1]);

    Vertices(graph, size, arr[0], arr[size - 1]);

    delete[] arr;
}
```

```
void Vertices(int **graph, int size, int radius, int diameter)
{
   int i, j, temp;
   int *DIST;
   DIST = (int *)(malloc(sizeof(int *) * size));

   for(i = 0; i < size; i++)
      DIST[i] = -1;

   for(i = 0; i < size; i++)
   {
      BFSD(graph, DIST, size, i);

      // __________ Сортировка пузырьком _________
      for (int k = 0; k < size - 1; k++)
      {
         for (int n = 0; n < size - k - 1; n++) {
            if (DIST[n] > DIST[n + 1]) {
               // меняем элементы местам
               temp = DIST[n];
               DIST[n] = DIST[n + 1];
               DIST[n + 1] = temp;
            }
         }
      }
      // ____________________________________________


      if(DIST[size - 1] == radius) printf("%d - Central vertex\n", i);
      else if(DIST[size - 1] == diameter) printf("%d - Peripheral vertex\n", i);

      for(j = 0; j < size; j++)
         DIST[j] = -1;
   }

   delete[] DIST;
}
```

Файл Lab10_Task3/main.cpp

```
#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


using namespace std;

int** Creategraph(int **, int); //Создание графа (взвешенный)
int** CreategraphO(int **, int); //Создание ориентированного графа (взвешенный)
int** creategraph(int **, int); //Создание графа
int** creategrapho(int**, int); //Создание ориентированного графа
void BFSD(int **, int *, int, int);  //Поиск расстояний путём обхода в глубину
void RadiusDiameter(int **, int *, int, int);
void Vertices(int **, int, int, int);
```

```
/*Параметры
balanced - взвешенный
unbalanced - невзвешенный
oriented - оринтированный
unoriented - неориентированный
*/

int size;   //Размер графа


int main(int argc, char *argv[])
{
   int i, j, v;
   int **graph = NULL, *DIST = NULL;


   printf("\t# Graph #\n\n");
   printf("Enter the number of graph vertices (positive integer): ");
   scanf("%d", &size);

   // Creating the graph

   if(strcmp(argv[1], "balanced") == 0)
   {
      if(strcmp(argv[2], "unoriented") == 0)
      {
         graph = Creategraph(graph, size);
      }
      else if(strcmp(argv[2], "oriented") == 0)
      {
         graph = CreategraphO(graph, size);
      }
   }
   else if(strcmp(argv[1], "unbalanced") == 0)
   {
      if(strcmp(argv[2], "unoriented") == 0)
      {
         graph = creategraph(graph, size);
      }
      else if(strcmp(argv[2], "oriented") == 0)
      {
         graph = creategrapho(graph, size);
      }
   }

   DIST = (int *)(malloc(sizeof(int *) * size));  // Array for visited vertices

   // Printing the matrix
   for(i = 0; i < size; i++)
   {
      DIST[i] = -1;
      for(j = 0; j < size; j++)
         printf("%3d ", graph[i][j]);
      printf("\n");
   }

   int num = size - 1;

   printf("Enter the number of start vertex (positive integer [0; %d]): ", num);
   scanf("%d", &v);
```

```
    // <---------- ! ---------->
    BFSD(graph, DIST, size, v);
    // <---------- ! ---------->
    //printf("\n");
    for(i = 0; i < size; i++)
        printf("%d: %d\n", i, DIST[i]);

    RadiusDiameter(graph, DIST, size, 0);

    delete[] graph;
    delete[] DIST;

    return 0;
}


int** Creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 4;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}

void BFSD(int **graph, int *DIST, int size, int v)
{
    queue<int> q;
    q.push(v);
    DIST[v] = 0;

    while(!q.empty())
    {
        v = q.front();
        //printf("%d ", v);
        q.pop();

        for(int i = 0; i < size; i++)
        {
            if((graph[v][i] > 0) && (DIST[i] == -1))
            {
                q.push(i);
                DIST[i] = DIST[v] + graph[v][i];
            }
```

```
        }
    }

}

int** CreategraphO(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = 0; j < size; j++)
        {
            graph[i][j] = rand() % 4;
            //graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}

void RadiusDiameter(int **graph, int *DIST, int size, int v)
{
    int i, j, temp;
    int *arr;
    arr = (int *)(malloc(sizeof(int *) * size));

    for(i = 0; i < size; i++)
        DIST[i] = -1;

    for(i = 0; i < size; i++)
    {
        BFSD(graph, DIST, size, i);

        for(j = 0; j < size; j++)
            printf("%d ", DIST[j]);
        printf("  ");

        // _________ Сортировка пузырьком _________
        for (int k = 0; k < size - 1; k++)
        {
            for (int n = 0; n < size - k - 1; n++) {
                if (DIST[n] > DIST[n + 1]) {
                    // меняем элементы местами
                    temp = DIST[n];
                    DIST[n] = DIST[n + 1];
                    DIST[n + 1] = temp;
                }
            }
        }
        // ___________________________________________
```

```
        arr[i] = DIST[size - 1];

        for(j = 0; j < size; j++)
            DIST[j] = -1;
        //printf("  ");

    }

       // _________ Сортировка пузырьком _________
        for (int k = 0; k < size - 1; k++)
        {
            for (int n = 0; n < size - k - 1; n++) {
                if (arr[n] > arr[n + 1]) {
                    // меняем элементы местами
                    temp = arr[n];
                    arr[n] = arr[n + 1];
                    arr[n + 1] = temp;
                }
            }
        }
        // ___________________________________________

    printf("\n");
    for(i = 0; i < size; i++)
        printf("%d ", arr[i]);

    printf("\nRadius = %d\nDiametr = %d\n", arr[0], arr[size - 1]);

    Vertices(graph, size, arr[0], arr[size - 1]);

    delete[] arr;
}

void Vertices(int **graph, int size, int radius, int diameter)
{
    int i, j, temp;
    int *DIST;
    DIST = (int *)(malloc(sizeof(int *) * size));

    for(i = 0; i < size; i++)
        DIST[i] = -1;

    for(i = 0; i < size; i++)
    {
        BFSD(graph, DIST, size, i);

        // _________ Сортировка пузырьком _________
        for (int k = 0; k < size - 1; k++)
        {
            for (int n = 0; n < size - k - 1; n++) {
                if (DIST[n] > DIST[n + 1]) {
                    // меняем элементы местам
                    temp = DIST[n];
                    DIST[n] = DIST[n + 1];
                    DIST[n + 1] = temp;
                }
            }
        }
        // ___________________________________________
```

```
        if(DIST[size - 1] == radius) printf("%d - Central vertex\n", i);
        else if(DIST[size - 1] == diameter) printf("%d - Peripheral vertex\n", i);

        for(j = 0; j < size; j++)
            DIST[j] = -1;
    }

    delete[] DIST;
}

int** creategraph(int **graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}

int** creategrapho(int** graph, int size)
{
    srand(time(NULL));

    int i = 0, j = 0;

    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * size));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = 0; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            //graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            //if(graph[i][j] == 1);
        }

    return graph;
}
```

Приложение Б
Параметры, вводимые в терминале для программы в задании №3

```
20 /*Параметры
21 balanced - взвешенный
22 unbalanced - невзвешенный
23 oriented - оринтированный
24 unoriented - неориентированный
25 */
```