|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 6** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «**Основные алгоритмы работы с графами.**»** | |
|  | |
| Выполнил студент группы ИКБО-03-21 | Хречко С.В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

# **Цель работы**

Получение практических навыков по выполнению операций над структурой данных граф.

# **Постановка задачи**

1. Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального задания. Реализовать метод ввода графа с клавиатуры, наполнение графа осуществлять с помощью метода добавления одного ребра. Реализовать метод вывода графа и методы, выполняющие задачи, определенные вариантом индивидуального задания.

2. Разработать программу, демонстрирующую работу всех методов класса.

3. Произвести тестирование программы на графе, предложенном в таблице 2.

4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №9. Условие задания:

|  |  |
| --- | --- |
| Матрица  смежности | Определить медиану неориентированного графа.  Составить программу нахождения кратчайших путей методом «Йена».  Используя результат алгоритма вывести путь между вводимыми парами вершин. |

# **Решение**

Граф — математическая абстракция реальной системы любой природы, объекты которой обладают парными связями. Граф как математический объект есть совокупность двух множеств — множества самих объектов, называемого множеством вершин, и множества их парных связей, называемого множеством рёбер. Элемент множества рёбер есть пара элементов множества вершин.

Графы находят широкое применение в современной науке и технике. Они используются и в естественных науках (физике и химии) и в социальных науках (например, социологии), но наибольших масштабов применение графов получило в информатике и сетевых технологиях.

По заданию требовалось реализовать метод добавления одного ребра. Метод принимает индексы вершин, а также вес добавляемого ребра и добавляет ребро в матрицу смежности.

|  |
| --- |
| void addEdge(int from, int to, int weight) {          adjMatrix[from][to] = weight;          adjMatrix[to][from] = weight;      } |

По заданию требовалось реализовать метод чтения графа с клавиатуры. Так как по заданию нужно заполнять граф, используя метод добавления одного ребра, было решено вводить граф как список ребер, после чего добавлять по одному ребру в матрицу смежности. Метод считывает данные с клавиатуры, после чего для каждого введенного ребра вызывает метод добавления ребра.

|  |
| --- |
| void readGraph() {//the task wants method of reading graph from keyboard          cout << "Enter number of vertices and number of edges" << endl;          int n;          cin >> n;          adjMatrix.resize(n, vector<int>(n, 0));          int m;          cin >> m;          cout << "Enter edges (<from number> <to number> <weight>) (numbers are from 1 to n)" << endl;          for (int i = 0; i < m; i++) {              //even though in memory we store graph as matrix, we will read it as list of edges              //because the task wanted us to use addEdge method              int from, to, weight;              cin >> from >> to >> weight;              addEdge(from-1, to-1, weight);          }      } |

По заданию требовалось реализовать метод вывода графа. Метод выводит матрицу смежности графа в отформатированном виде, используя библиотеку iomanip.

|  |
| --- |
| void printGraph() {          cout << setw(3) << "#";          for (int j = 0; j < adjMatrix[0].size(); j++){              cout << setw(3) << j+1;          }          cout << endl;          for (int i = 0; i < adjMatrix.size(); i++) {              cout << setw(3) << i+1;              for (int j = 0; j < adjMatrix[i].size(); j++) {                  cout << setw(3) << adjMatrix[i][j];              }              cout << endl;          }      } |

По заданию требовалось реализовать метод нахождения медианы графа. Медиана – это вершина графа, сумма путей до которой от остальных вершин минимальна. Для нахождения медианы метод для каждой вершины вызывает метод поиска суммы путей до других вершин, после чего ищет минимальное из полученных значений.

|  |
| --- |
| int findMedian() {          int min = INT\_MAX;          int minIndex = -1;          for (int i = 0; i < adjMatrix.size(); i++) {              int sum = findSumToAllVertexes(i);              if (sum < min) {                  min = sum;                  minIndex = i;              }          }          return minIndex;      } |

Для нахождения медианы графа используется вспомогательный метод поиска суммы путей от одной вершины до всех других. Метод находит кратчайшие пути от переданной вершины до других используя алгоритм Дейкстры. После чего суммирует полученные значения и возвращает сумму.

|  |
| --- |
| int findSumToAllVertexes(int from) {          vector<int> dist(adjMatrix.size(), INT\_MAX);          vector<bool> visited(adjMatrix.size(), false);          dist[from] = 0;          for (int i = 0; i < adjMatrix.size(); i++) {              int min = INT\_MAX;              int minIndex = -1; //closest vertex              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && dist[j] < min) {                      min = dist[j];                      minIndex = j;                  }              }              visited[minIndex] = true;              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && adjMatrix[minIndex][j] != 0 && dist[minIndex] != INT\_MAX && dist[minIndex] + adjMatrix[minIndex][j] < dist[j]) {                      dist[j] = dist[minIndex] + adjMatrix[minIndex][j];                  }              }          }          int sum = 0;          for (int i = 0; i < dist.size(); i++) {              sum += dist[i];          }          return sum;      } |

По заданию требовалось реализовать вывод пути между двумя заданными вершинами, используя алгоритм Йена. Метод представляет собой алгоритм Йена для нахождения одного кратчайшего пути, аналогично алгоритму Дейкстры. Метод во многом схож с методом поиска сумм от одной вершины до других, но в отличие от него, данный метод также запоминает путь, чтобы затем его можно было вывести.

|  |
| --- |
| int findShortestPath(int from, int to, vector<int>& path) { //returns weight of path          vector<int> dist(adjMatrix.size(), INT\_MAX);          vector<int> prev(adjMatrix.size(), -1);          vector<bool> visited(adjMatrix.size(), false);          dist[from] = 0;          for (int i = 0; i < adjMatrix.size(); i++) {              int min = INT\_MAX;              int minIndex = -1; //closest vertex              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && dist[j] < min) {                      min = dist[j];                      minIndex = j;                  }              }              visited[minIndex] = true;              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && adjMatrix[minIndex][j] != 0 && dist[minIndex] != INT\_MAX && dist[minIndex] + adjMatrix[minIndex][j] < dist[j]) {                      dist[j] = dist[minIndex] + adjMatrix[minIndex][j];                      prev[j] = minIndex;                  }              }          }          path.clear();          int current = to;          while (current != -1) {              path.push\_back(current);              current = prev[current];          }          return dist[to];      } |

Пользовательский интерфейс программы предлагает пользователю ввести количество вершин и ребер графа, а затем и сами ребра с их весами. После чего предлагает пользователю ввести две вершины , для того чтобы вывести путь между ними.

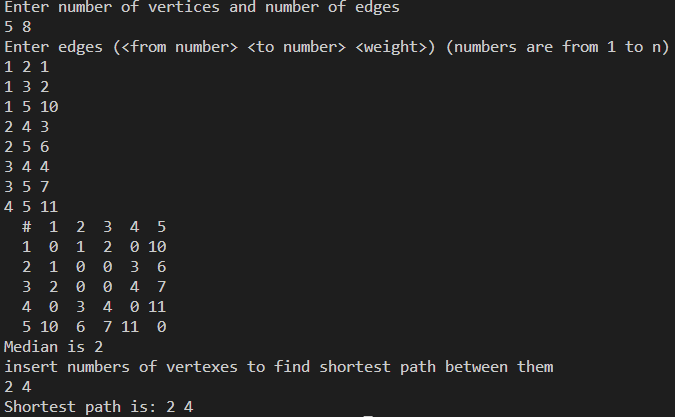


Рисунок 1. Интерфейс программы

# **Тестирование**

В соответствии с задание, тестирование будет проводиться на одном из графов данных в условии. Данный граф представлен на рисунке 2.

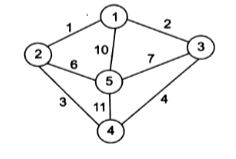


Рисунок 2. Граф для тестирования

Тестирование построения ввода и вывода графа. Граф имеет 5 вершин и 8 ребер. Результаты представлены на рисунке 3. Сравнив выведенные данные с изображением графа на рисунке 2, можно убедиться в работоспособности методов.

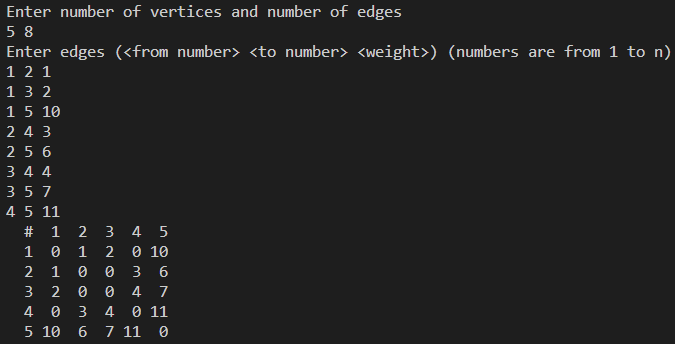


Рисунок 3. Тестирование ввода и вывода графа

Тестирование метода нахождения медианы графа представлено на рисунке 4. Ожидаемый результат, который можно получить из изображения графа на рисунке 2, это, что медианой является вершина 2, так как к вершине 5 ведут крайне тяжеловесные ребра.

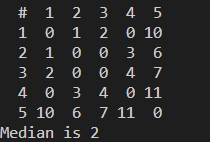


Рисунок 4. Тестирование метода нахождения медианы

Тестирование метода нахождения пути между двумя вершинами представлено на рисунках 5 и 6.

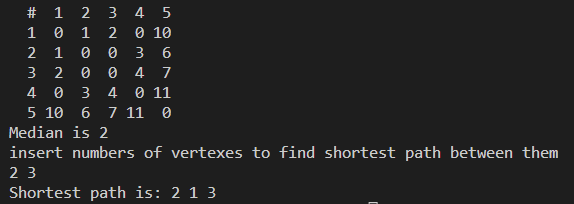


Рисунок 5. Тестирование нахождения пути

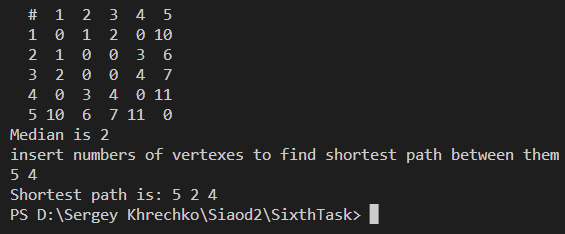


Рисунок 6. Тестирование нахождения пути

Из результатов выполнения программы видно:

1. Программа работает корректно, все функции работоспособны.

# **Вывод**

В результате выполнения работы я:

1. Освоил базовую работу с графами.
2. Научился создавать методы поиска кратчайшего пути в графе по алгоритму Дейкстры.

# **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <vector>  #include <iomanip>  using namespace std;  class Graph {  private:      vector<vector<int>> adjMatrix;  public:      Graph(int n) {          adjMatrix.resize(n, vector<int>(n, 0));      }      void addEdge(int from, int to, int weight) {          adjMatrix[from][to] = weight;          adjMatrix[to][from] = weight;      }      void readGraph() {//the task wants method of reading graph from keyboard          cout << "Enter number of vertices and number of edges" << endl;          int n;          cin >> n;          adjMatrix.resize(n, vector<int>(n, 0));          int m;          cin >> m;          cout << "Enter edges (<from number> <to number> <weight>) (numbers are from 1 to n)" << endl;          for (int i = 0; i < m; i++) {              //even though in memory we store graph as matrix, we will read it as list of edges              //because the task wanted us to use addEdge method              int from, to, weight;              cin >> from >> to >> weight;              addEdge(from-1, to-1, weight);          }      }      void removeEdge(int from, int to) {          adjMatrix[from][to] = 0;          adjMatrix[to][from] = 0;      }      void printGraph() {          cout << setw(3) << "#";          for (int j = 0; j < adjMatrix[0].size(); j++){              cout << setw(3) << j+1;          }          cout << endl;          for (int i = 0; i < adjMatrix.size(); i++) {              cout << setw(3) << i+1;              for (int j = 0; j < adjMatrix[i].size(); j++) {                  cout << setw(3) << adjMatrix[i][j];              }              cout << endl;          }      }      int findSumToAllVertexes(int from) {          vector<int> dist(adjMatrix.size(), INT\_MAX);          vector<bool> visited(adjMatrix.size(), false);          dist[from] = 0;          for (int i = 0; i < adjMatrix.size(); i++) {              int min = INT\_MAX;              int minIndex = -1; //closest vertex              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && dist[j] < min) {                      min = dist[j];                      minIndex = j;                  }              }              visited[minIndex] = true;              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && adjMatrix[minIndex][j] != 0 && dist[minIndex] != INT\_MAX && dist[minIndex] + adjMatrix[minIndex][j] < dist[j]) {                      dist[j] = dist[minIndex] + adjMatrix[minIndex][j];                  }              }          }          int sum = 0;          for (int i = 0; i < dist.size(); i++) {              sum += dist[i];          }          return sum;      }      int findMedian() {          int min = INT\_MAX;          int minIndex = -1;          for (int i = 0; i < adjMatrix.size(); i++) {              int sum = findSumToAllVertexes(i);              if (sum < min) {                  min = sum;                  minIndex = i;              }          }          return minIndex;      }      int findShortestPath(int from, int to, vector<int>& path) { //returns weight of path          vector<int> dist(adjMatrix.size(), INT\_MAX);          vector<int> prev(adjMatrix.size(), -1);          vector<bool> visited(adjMatrix.size(), false);          dist[from] = 0;          for (int i = 0; i < adjMatrix.size(); i++) {              int min = INT\_MAX;              int minIndex = -1; //closest vertex              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && dist[j] < min) {                      min = dist[j];                      minIndex = j;                  }              }              visited[minIndex] = true;              for (int j = 0; j < adjMatrix.size(); j++) {                  if (!visited[j] && adjMatrix[minIndex][j] != 0 && dist[minIndex] != INT\_MAX && dist[minIndex] + adjMatrix[minIndex][j] < dist[j]) {                      dist[j] = dist[minIndex] + adjMatrix[minIndex][j];                      prev[j] = minIndex;                  }              }          }          path.clear();          int current = to;          while (current != -1) {              path.push\_back(current);              current = prev[current];          }          return dist[to];      }  };  int main(){      Graph g(0);      g.readGraph();      g.printGraph();      cout << "Median is " << g.findMedian() + 1 << endl;      cout << "insert numbers of vertexes to find shortest path between them" << endl;      int from, to;      cin >> from >> to;      vector<int> path;      g.findShortestPath(from - 1, to - 1, path);      cout << "Shortest path is: ";      for (int i = path.size() - 1; i >= 0; i--) {          cout << path[i] + 1 << " ";      }      cout << endl;      return 0;  } |