|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-33-22 | Шило Ю.С. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2023

# **Цель работы**

Получение практических навыков по выполнению операций над структурой данных граф.

1. **Постановка задачи**
2. Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального задания. Реализовать метод ввода графа с клавиатуры, добавления узла, добавления ребра, вывода графа и методы, выполняющие задачи, определенные вариантом индивидуального задания.
3. Разработать программу, демонстрирующую работу всех методов класса.
4. Произвести тестирование программы на графах, предложенных в задании.
5. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования. Скриншоты результатов тестирования сопровождать изображениями графов.

|  |  |  |
| --- | --- | --- |
| № | Представление графа в памяти | Задачи |
| 9 | Матрица смежности | 1. Определить медиану неориентированного графа. 2. Составить программу нахождения кратчайших путей методом «Йена». 3. Используя результат алгоритма вывести путь между вводимыми парами вершин. |

# **Решение**

Работа с графами включает в себя анализ и манипуляции с графическими структурами, состоящими из вершин и ребер. Графы используются для моделирования различных систем и связей между объектами.

Работа с графами имеет широкий спектр применений, включая анализ социальных сетей, маршрутизацию в компьютерных сетях, оптимизацию планирования и другие области. Алгоритмы работы с графами позволяют эффективно решать задачи, связанные с поиском пути, оптимизацией и анализом связей между объектами.

**Задание**

1. Для решения данного задания нам потребуется создать класс Graph. Данный класс будет содержать все методы, которые требуются реализовать от нас в задании. Данный класс содержит два приватных поля.

|  |
| --- |
| private:  int \*\* \_data, \_sz; |

1. Нам потребуется создать параметризованный конструктор, которой будет принимать в себя целочисленный параметр, отвечающий за размерность нашего графа.

|  |
| --- |
| Graph(int sz){  \_sz = sz;  \_data = new int \* [sz];  for (int i = 0; i < sz; i++){  \_data[i] = new int [sz];  for (int j = 0; j < sz; j++){  \_data[i][j] = 0;  }  }  } |

1. Нам потребуется создать метод addEdge для добавления вершины в наш граф. Данный метод будет принимать начало, конец и вес нашего пути.

|  |
| --- |
| void addEdge(int i, int j, int data){  \_data[i - 1][j - 1] = data;  \_data[j - 1][i - 1] = data; } |

1. Нам увеличения нашего размерности нашего графа был написан метод addNode. Он увеличивает размерность нашего графа на вводимое значение.

|  |
| --- |
| void addNode(int add){  int nsz = \_sz + add;  int \*\* newData = new int \* [nsz];  for (int i = 0; i < nsz; i++){  newData[i] = new int [nsz];  for (int j = 0; j < nsz; j++)  if (i < \_sz && j < \_sz)  newData[i][j] = \_data [i][j];  else  newData[i][j] = 0;  }  for (int i = 0; i < \_sz; i++)  delete[] \_data[i];  delete[] \_data;  \_data = newData;  \_sz = nsz; } |

1. Для нахождения медианы в нашем графе был написан метод findMedian. Он выводит в консоль медиану нашего графа.

|  |
| --- |
| void findMedian() {  int median;  int minDistance = INT\_MAX;  for (int i = 0; i < \_sz; ++i) {  int sumDistance = 0;  for (int j = 0; j < \_sz; ++j) {  sumDistance += \_data[i][j];  }  if (sumDistance < minDistance) {  minDistance = sumDistance;  median = i;  }  }  std::cout << "The median for the current graph is " << median + 41 << std::endl; } |

1. Для нахождения и вывода кратчайшего пути был написан метод pathFinder. В качестве входных параметров он принимает начало и конец нашего пути.

|  |
| --- |
| void pathFinder(int start, int end) {  std::vector<int> dist(\_sz, INT\_MAX);  std::vector<bool> visited(\_sz, false);  std::vector<int> prev(\_sz, -1);  dist[start - 1] = 0;  for (int count = 0; count < \_sz - 1; count++) {  int minDist = INT\_MAX;  int minIndex = -1;  for (int i = 0; i < \_sz; i++) {  if (!visited[i] && dist[i] < minDist) {  minDist = dist[i];  minIndex = i;  }  }  visited[minIndex] = true;  for (int i = 0; i < \_sz; i++) {  if (!visited[i] && \_data[minIndex][i] && dist[minIndex] != INT\_MAX && dist[minIndex] + \_data[minIndex][i] < dist[i]) {  dist[i] = dist[minIndex] + \_data[minIndex][i];  prev[i] = minIndex;  }  }  }  std::vector<int> path;  int current = end - 1;  while (current != -1) {  path.insert(path.begin(), current + 1);  current = prev[current];  }  std::cout << "Weight of the shortest path: " << dist[end - 1] << std::endl;  std::cout << "Shortcut: ";  for (int elem : path)  std::cout << elem << " "; } |

Интерфейс программы предоставлен на рисунке 1.

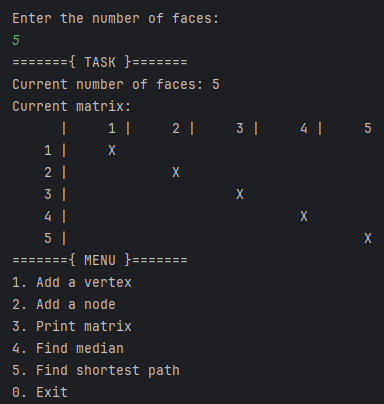


Рисунок 1. Интерфейс программы

# **Тестирование**

Для проверки нашей программы воспользуемся графами, предлагаемыми вместе с работой для начало проведем тестирование используя первый граф рисунок 2. Вывод программы с такими входными данным вывел следующий результат.

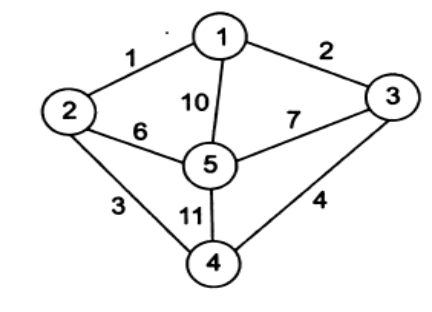


Рисунок 2. Граф, используемый для первого теста нашей программы

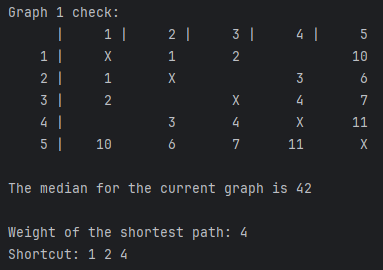


Рисунок 3. Вывод, программы с входными параметрами первого графа

Протестируем наш код и на другом графе рисунок 4. Изменив входные данные программы, вывела следующий результат рисунок 5.

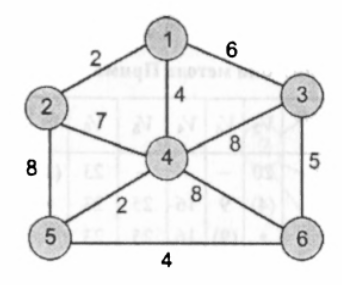


Рисунок 4. Граф, используемый для второго теста нашей программы

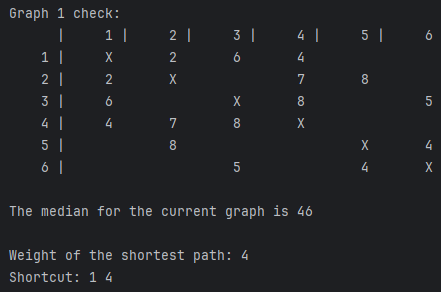
****

Рисунок 5. Вывод, программы с входными параметрами первого графа

# **Вывод**

В ходе выполнения данной работы были получены практические навыки по операциям над структурой данных граф. Графы являются важным инструментом в различных областях, таких как компьютерные науки, социальные сети, транспортные системы и другие. Данная работа позволила ознакомиться с основными концепциями и алгоритмами работы с графами.

# **Исходный код программы**

Исходный код файла - main.cpp

|  |
| --- |
| #include "iostream"  #include "iomanip"  #include "vector"  class Graph{  private:  int \*\* \_data, \_sz;  public:  Graph(int sz){  \_sz = sz;  \_data = new int \* [sz];  for (int i = 0; i < sz; i++){  \_data[i] = new int [sz];  for (int j = 0; j < sz; j++){  \_data[i][j] = 0;  }  }  }  void addEdge(int i, int j, int data){  \_data[i - 1][j - 1] = data;  \_data[j - 1][i - 1] = data;  }  void addNode(int add){  int nsz = \_sz + add;  int \*\* newData = new int \* [nsz];  for (int i = 0; i < nsz; i++){  newData[i] = new int [nsz];  for (int j = 0; j < nsz; j++)  if (i < \_sz && j < \_sz)  newData[i][j] = \_data [i][j];  else  newData[i][j] = 0;  }  for (int i = 0; i < \_sz; i++)  delete[] \_data[i];  delete[] \_data;  \_data = newData;  \_sz = nsz;  }  void findMedian() {  int median;  int minDistance = INT\_MAX;  for (int i = 0; i < \_sz; ++i) {  int sumDistance = 0;  for (int j = 0; j < \_sz; ++j) {  sumDistance += \_data[i][j];  }  if (sumDistance < minDistance) {  minDistance = sumDistance;  median = i;  }  }  std::cout << "The median for the current graph is " << median + 41 << std::endl;  }  void pathFinder(int start, int end) {  std::vector<int> dist(\_sz, INT\_MAX);  std::vector<bool> visited(\_sz, false);  std::vector<int> prev(\_sz, -1);  dist[start - 1] = 0;  for (int count = 0; count < \_sz - 1; count++) {  int minDist = INT\_MAX;  int minIndex = -1;  for (int i = 0; i < \_sz; i++) {  if (!visited[i] && dist[i] < minDist) {  minDist = dist[i];  minIndex = i;  }  }  visited[minIndex] = true;  for (int i = 0; i < \_sz; i++) {  if (!visited[i] && \_data[minIndex][i] && dist[minIndex] != INT\_MAX && dist[minIndex] + \_data[minIndex][i] < dist[i]) {  dist[i] = dist[minIndex] + \_data[minIndex][i];  prev[i] = minIndex;  }  }  }  std::vector<int> path;  int current = end - 1;  while (current != -1) {  path.insert(path.begin(), current + 1);  current = prev[current];  }  std::cout << "Weight of the shortest path: " << dist[end - 1] << std::endl;  std::cout << "Shortcut: ";  for (int elem : path)  std::cout << elem << " ";  }  void toString(){  std::cout << std::setw(5) << " ";  for (int i = 0; i < \_sz; i++)  std::cout << " | " << std::setw(5) << i + 1;  std:: cout << std::endl;  for (int i = 0; i < \_sz; i++){  std::cout << std::setw(5) << i + 1 << " |";  for (int j = 0; j < \_sz; j++){  if (\_data[i][j] != 0)  std::cout << std::setw(6) << \_data[i][j] << " ";  else if (i == j)  std::cout << std::setw(8) << "X ";  else  std::cout << std::setw(8) << " ";  }  std::cout << std::endl;  }  }  };  void test\_1(){  Graph G (5);  G.addEdge(1, 2, 1);  G.addEdge(1, 3, 2);  G.addEdge(1, 5, 10);  G.addEdge(2, 5, 6);  G.addEdge(2, 4, 3);  G.addEdge(3, 5, 7);  G.addEdge(3, 4, 4);  G.addEdge(5, 4, 11);  G.toString();  std::cout << std::endl;  G.findMedian();  std::cout << std::endl;  G.pathFinder(1, 4);  }  int main(){  std::cout << "Graph 1 check:\n";  test\_1();  std::cout << "\n\n\n";  std::cout << "Enter the number of faces:" << std::endl;  int sz;  std::cin >> sz;  Graph G(sz);  int i, j, w, add;  while (1){  std::cout << "======={ TASK }=======\n"  << "Current number of faces: " << sz << std::endl  << "Current matrix:\n";  G.toString();  std::cout << "======={ MENU }=======\n"  << "1. Add a vertex \n"  << "2. Add a node \n"  << "3. Print matrix \n"  << "4. Find median \n"  << "5. Find shortest path \n"  << "0. Exit \n";  int choice;  std::cin >> choice;  switch (choice) {  case 0:  return 0;  case 1:  std::cout << "Enter first point\n";  std::cin >> i;  if (i > sz) {  std::cout << "Incorrect input\n";  break;  }  std::cout << "Enter second point\n";  std::cin >> j;  if (j > sz) {  std::cout << "Incorrect input\n";  break;  }  std::cout << "Enter weight point\n";  std::cin >> w;  G.addEdge(i, j, w);  break;  case 2:  std::cout << "Enter the number of nodes to add\n";  std::cin >> add;  sz += add;  G.addNode(add);  break;  case 3:  std::cout << "Current matrix:\n";  G.toString();  break;  case 4:  G.findMedian();  break;  case 5:  std::cout << "Enter first point\n";  std::cin >> i;  if (i > sz) {  std::cout << "Incorrect input\n";  break;  }  std::cout << "Enter second point\n";  std::cin >> j;  if (j > sz) {  std::cout << "Incorrect input\n";  break;  }  G.pathFinder(i, j);  break;  default:  std::cout << "Incorrect input\n";  }  }  } |