Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ»

Институт системной и программной инженерии

и информационных технологий (СПИНТех)

**Отчёт**

по дисциплине «Алгоритмы и структуры данных»

**Лабораторная работа №3**

**Вариант-10**

Руководитель

\_\_\_\_\_\_\_\_\_\_\_ Волков A. C.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Студент группы ПИН-23

\_\_\_\_\_\_\_\_\_\_\_ Исламов Р. Р.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

*Москва*

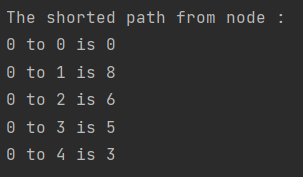
*2023*

# 1 Листинг

package Lab3;  
  
import java.util.\*;  
public class Dijkstra {  
 private int dist[];  
 private Set<Integer> settled;  
 private PriorityQueue<Node> pq;  
 private int V;   
 List<List<Node> > adj;  
  
 public Dijkstra(int V)  
 {  
 this.V = V;  
 dist = new int[V];  
 settled = new HashSet<Integer>();  
 pq = new PriorityQueue<Node>(V, new Node());  
 }  
  
 public void dijkstra(List<List<Node> > adj, int src)  
 {  
 this.adj = adj;  
  
 for (int i = 0; i < V; i++)  
 dist[i] = Integer.*MAX\_VALUE*;  
  
 pq.add(new Node(src, 0));  
  
 dist[src] = 0;  
 while (settled.size() != V) {  
 if(pq.isEmpty())  
 return ;  
 int u = pq.remove().node;  
  
 settled.add(u);  
  
 e\_Neighbours(u);  
 }  
 }  
  
 private void e\_Neighbours(int u)  
 {  
 int edgeDistance = -1;  
 int newDistance = -1;  
  
 for (int i = 0; i < adj.get(u).size(); i++) {  
 Node v = adj.get(u).get(i);  
  
 if (!settled.contains(v.node)) {  
 edgeDistance = v.cost;  
 newDistance = dist[u] + edgeDistance;  
  
 if (newDistance < dist[v.node])  
 dist[v.node] = newDistance;  
  
 pq.add(new Node(v.node, dist[v.node]));  
 }  
 }  
 }  
  
 public static void main(String arg[])  
 {  
 int V = 5;  
 int source = 0;  
  
 List<List<Node> > adj = new ArrayList<List<Node> >();  
  
 for (int i = 0; i < V; i++) {  
 List<Node> item = new ArrayList<Node>();  
 adj.add(item);  
 }  
  
 adj.get(0).add(new Node(1, 9));  
 adj.get(0).add(new Node(2, 6));  
 adj.get(0).add(new Node(3, 5));  
 adj.get(0).add(new Node(4, 3));  
  
 adj.get(2).add(new Node(1, 2));  
 adj.get(2).add(new Node(3, 4));  
  
 Dijkstra dijkstra = new Dijkstra(V);  
 dijkstra.dijkstra(adj, source);  
  
 System.*out*.println("The shorted path from node :");  
 for (int i = 0; i < dijkstra.dist.length; i++)  
 System.*out*.println(source + " to " + i + " is "  
 + dijkstra.dist[i]);  
 }  
}  
  
// Class to represent a node in the graph  
class Node implements Comparator<Node> {  
 public int node;  
 public int cost;  
  
 public Node()  
 {  
 }  
  
 public Node(int node, int cost)  
 {  
 this.node = node;  
 this.cost = cost;  
 }  
  
 @Override  
 public int compare(Node node1, Node node2)  
 {  
 if (node1.cost < node2.cost)  
 return -1;  
 if (node1.cost > node2.cost)  
 return 1;  
 return 0;  
 }  
}

}

# 2 Результат работы



# 3 Контрольные вопросы

1. Какова теоретическая сложность алгоритмов, рассмотренных в настоящей лабораторной работе? 2. В решении каких прикладных задач используются алгоритмы определения в графе кратчайших расстояний между заданными вершинами? 3. Может ли быть применен рассмотренный алгоритм Дейкстры при определении кратчайшего расстояния в ориентированном графе? 4. Как работает алгоритм Дейкстры? 5. Как работает алгоритм динамического программирования в задачах определения в графе кратчайших расстояний между вершинами?

1. Теоретическая сложность алгоритмов, рассмотренных в данной лабораторной работе, зависит от конкретного алгоритма, используемого в каждом случае. Например, алгоритм Крускала и алгоритм Прима для построения остовного дерева имеют сложность O(E log V), где E - количество ребер в графе, V - количество вершин. Алгоритм Дейкстры для нахождения кратчайшего пути имеет сложность O((V + E) log V), где V - количество вершин, E - количество ребер.
2. Алгоритмы определения кратчайших расстояний между заданными вершинами в графе применяются в различных прикладных задачах, таких как:

* Маршрутизация в компьютерных сетях: определение оптимального маршрута для передачи данных между узлами сети.
* Навигация: нахождение кратчайшего пути между местоположениями на карте или в GPS-навигации.
* Транспортное планирование: оптимизация маршрутов доставки для минимизации времени и затрат.
* Анализ социальных сетей: определение степени связности между людьми или сообществами в графе социальных связей.

1. Алгоритм Дейкстры может быть применен при определении кратчайшего расстояния в ориентированном графе, если граф не содержит отрицательных ребер. Однако в случае наличия отрицательных ребер, алгоритм Дейкстры может дать неправильный результат. В таких случаях обычно используются алгоритмы, такие как алгоритм Беллмана-Форда или алгоритм Флойда-Уоршелла.
2. Алгоритм Дейкстры работает следующим образом:

* Создается массив расстояний, инициализирующийся бесконечностями, за исключением стартовой вершины, для которой расстояние равно 0.
* Выбирается вершина с наименьшим расстоянием из массива расстояний (начиная с стартовой вершины).
* Обновляются расстояния до соседних вершин, если новое расстояние меньше текущего.
* Повторяются предыдущие два шага, пока все вершины не будут посещены.
* В результате получается массив кратчайших расстояний от стартовой вершины до всех остальных вершин.

1. Алгоритм динамического программирования в задачах определения кратчайших расстояний между вершинами графа используется для нахождения оптимальных путей с минимальными затратами. Он работает путем разбиения задачи на подзадачи и сохранения результатов этих подзадач для последующего использования. В контексте определения кратчайших путей, алгоритм динамического программирования может быть реализован, например, с использованием матрицы расстояний, где каждый элемент матрицы содержит кратчайшее расстояние между соответствующими вершинами графа.