

**Содержание**

[Реферат 3](#_Toc122947676)

[Введение 4](#_Toc122947677)

[1.Постановка задачи 5](#_Toc122947678)

[2.Теоретическая часть задания 6](#_Toc122947679)

[3.Описание алгоритма программы 7](#_Toc122947680)

[4.Описание программы 9](#_Toc122947681)

[5.Тестирование 13](#_Toc122947682)

[6.Ручной расчет задачи 16](#_Toc122947683)

[Заключение 17](#_Toc122947684)

[Список литературы 18](#_Toc122947685)

[Приложение А. 19](#_Toc122947686)

[Листинг программы. 19](#_Toc122947687)

**Реферат**

Отчет 24 стр, 8 рисунков.

Реализация алгоритма Дейкстры

Цель исследования – разработка программы, способной производить поиск кратчайших путей из вершины графа, по алгоритму Дейкстры.

В работе рассмотрен алгоритм Дейкстры. Установлено, что с помощью данного алгоритма можно произвести поиск кратчайших путей из одной вершины графа ко всем.

Введение

Алгоритм Дейкстры – алгоритм на графах, изобретенный нидерландским ученым Эдсгером Дейкстрой в 1959 году. Алгоритм находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании, например, его используют протоколы маршрутизации OSPF и IS-IS.

В качестве среды разработки мною была выбрана среда IntelliJ IDEA CE, язык программирования – Java.

Целью данной курсовой работы является разработка программы на языке Java, который является современным и широко используемым. Именно с его помощью в данной курсовом проекте реализуется алгоритм Дейкстры, осуществляющий поиск кратчайшего пути от одной вершины графа до всех остальных.

1.Постановка задачи

Требуется разработать программу, которая выявит кратчайшие пути от одной вершины графа до других, используя алгоритм Дейкстры.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки данных на экран должна выводиться матрица смежности графа. Необходимо предусмотреть различные исходы работы алгоритма, чтобы программа не выдавала ошибок.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №13.

2.Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается линией, которая указывает достижимость данной вершины, граф с такими ребрами называется неориентированным.

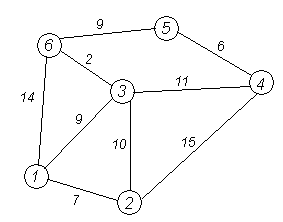


Рисунок 1 - Пример графа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем. После этого по данной матрице строится еще одна матрица смежности, которая показывает вес дуги.

Каждой вершине из *V* сопоставим метку — минимальное известное расстояние от этой вершины до *a*. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

3.Описание алгоритма программы

Для программной реализации алгоритма понадобится два массива: dist – массив, в который записываются минимальные пути, marked, куда записываются посещенные вершины. Имеется граф graph=(i, j). Каждая из вершин графа пока не занесена в словарь с посещенными вершинами.

В качестве исходного пункта выбирается вершина 0. В массиве dist задается нулевое значение для начальной вершины. Пока не просмотрены все вершины выполняется цикл: мы получаем все связанные вершины с текущей, и если вершина не записана в массив marked, то есть не посещена, то в переменную w записывается ее вес(сумма путей), если текущий вес меньше чем тот что уже записан, то в этом случае сохраняем новые значения связи. Далее выбирается вершина с минимальным весом с помощью функции findMinVert, если данная вершина больше нуля, то мы ее добавляем в массив marked. После завершения цикла в массиве dist будут записаны кратчайшие пути от начальной вершины ко всем остальным.

Ниже представлен псевдокод алгоритма и метода findMinVert()

findMinVert():

1. minVertex = -1;
2. цикл: i = 0; i < dist.length; i++;
   1. если не marked[i] и (minVertex == -1 или dist[i] < dist[minVertex];
      1. minVertex = i;
3. вернуть minVertex

Сам алгоритм:

1. dist[] = [graph.length];  
   marked[] = [graph.length];
2. цикл: i = 0; i < graph.length; i++
   1. dist[i] = Integer.MAX\_VALUE;  
      marked[i] = false;
3. dist[src\_node] = 0;
4. цикл: i = 0; i < graph.length; i++
   1. count = 0; count < graph.length - 1; count++
      1. u = *findMinVert*(dist, marked);  
         marked[u] = true;
         1. v = 0; v < graph.length; v++
         2. если не marked[v] и graph[u][v] не равно 0 и dist[u] не равно Integer.*MAX\_VALUE* и dist[u] + graph[u][v] < dist[v]
            1. dist[v] = dist[u] + graph[u][v];
5. *вывод минимального пути*  
   *и запись в файл*

Полный код можно увидеть в Приложении А.

4.Описание программы

Для написания данной программы использовался язык программирования Java. Это универсальный и современный язык программирования, который завоевал свою популярность у программистов, благодаря богатой стандартной библиотеке и комфортным порогом входа.

Проект был создан в виде консольного приложения.

Данная программа является многомодульной, поскольку состоит из нескольких методов: createMatrix, printArray, goToFile, main, infoToFile, dijkstra, findMinVert, printMinPath, readFromFile.

Работа начинается с выбора генерации графа. На выбор пользователю предлагается 2 варианта: Случайная генерация с вводом только количества вершин или прочитать матрицу смежности из файла, в который пользователь ввел ее заранее. Если пользователь выбрал случайную генерацию, то программа запросит ввести количество вершин, после чего программа сгенерирует граф заданного размера.

Далее выводятся элементы массива, а именно матрица смежности сгенерированного графа и предложит выбрать вершину, для которой будет выполняться алгоритм. (рис.2).

Если же пользователь выбрал ввод матрицы смежности из файла, то программа сразу выведет данную матрицу и предложит выбрать вершину, для которой будет выполняться алгоритм. (рис.3).

Scanner choose = new Scanner(System.*in*);  
int chs;  
  
while (true) {  
 System.*out*.println("choose output's method: " + "1. Read from file " + "2. Random graph adjacency table");  
 if (choose.hasNextInt()) {  
 chs = choose.nextInt();  
 if (chs == 1) {  
 int[][] arr = *readFromFile*();  
 *printArray*(arr);  
 System.*out*.println(" Enter the number of node for Dijkstra algoritm: ");  
 while (true) {  
 if (choose.hasNextInt()) {  
 int getNum = choose.nextInt();  
 if (getNum < arr.length) {  
 *dijkstra*(arr, getNum);  
 System.*exit*(0);  
  
 } else {  
 System.*out*.println("Error, input correct value!");  
 }  
 }  
 }  
 } else if (chs == 2) {  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("Enter the value of nodes: ");  
 int valueOfNodes = sc.nextInt();  
 int[][] graph = *createMatrix*(valueOfNodes);  
 System.*out*.println("Graph adjacency table:");  
 *printArray*(graph);  
 *goToFile*(graph);  
 System.*out*.println(" Enter the number of node for Dijkstra algoritm: ");  
 while (true) {  
 if (choose.hasNextInt()) {  
 int getNum = choose.nextInt();  
 if (getNum < graph.length) {  
 *dijkstra*(graph, getNum);  
 System.*exit*(0);  
 } else {  
 System.*out*.println("Error, input correct value!");  
 }  
 }  
 }  
 }  
 System.*out*.println("Error, enter the 1 or 2!");  
 }  
}

Ниже можно увидеть оформление начального запроса и дальнейшее действия с ним.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 - Случайная генерация матрицы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 - Ввод матрицы смежности из файла

После ввода вершины, программа выводит ответ и записывает его в файл (рис.4), (рис.5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 - вывод ответа

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 - вывод ответа и матрицы в файлы

5.Тестирование

Среда разработки IntelliJ IDEA CE предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем количества вершин графа и вводе матрицы смежности из файла.

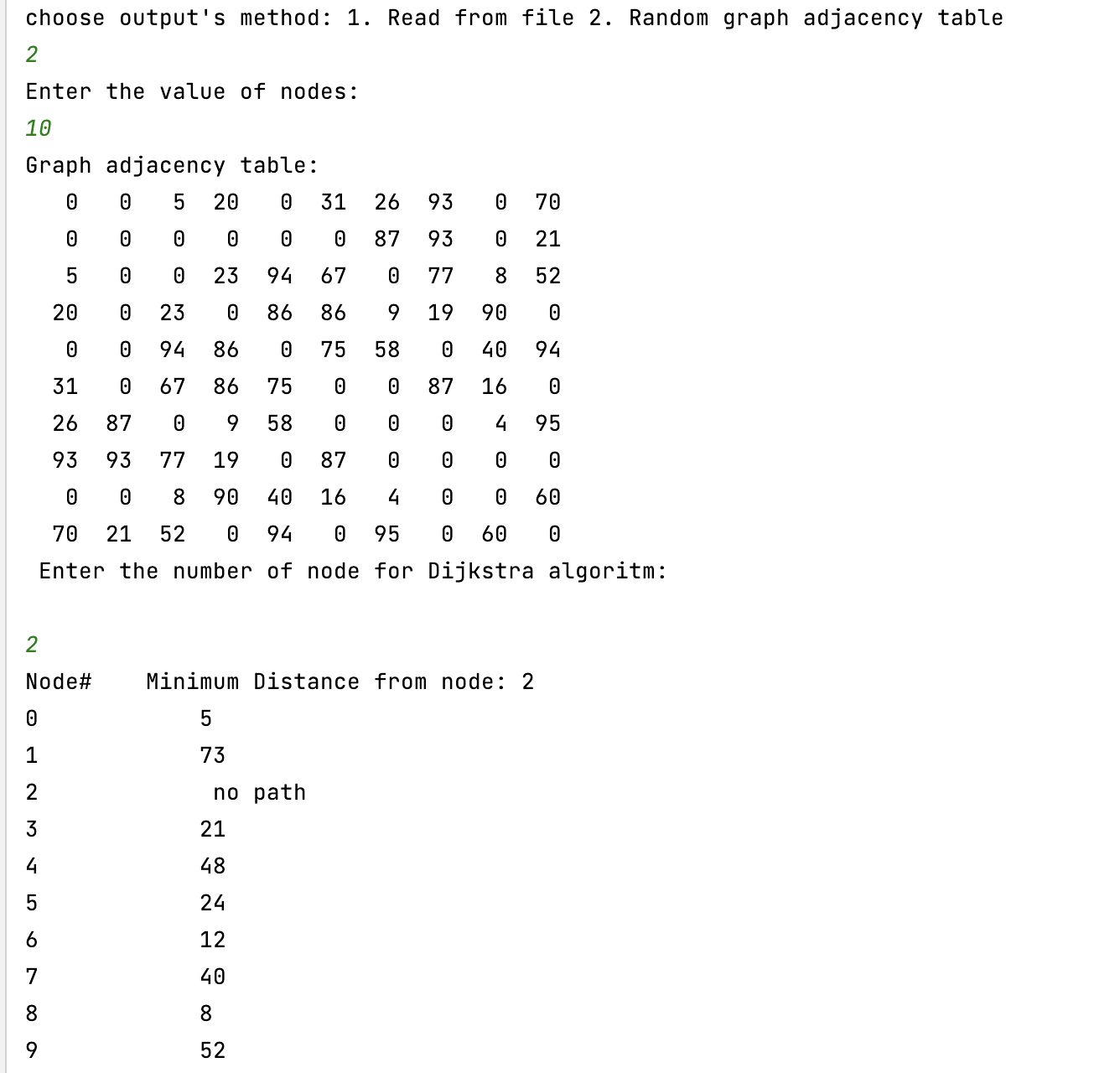


Рисунок 6 - Тестирование при вводе количества вершин = 10

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 - Тестирование при вводе матрицы смежности из файла

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе: сгенерировать матрицу или вывести матрицу из файла | Верно |
| Выбор генерации матрицы | Вывод сообщения о количестве вершин в орграфе | Верно |
| Выбор вывода матрицы из  файла | Вывод матрицы | Верно |
| Выбор стартовой вершины | Вывод сообщения о выборе стартовой вершины | Верно |
| Вывод ответа | Вывод вершин графа и под ними кратчайшие пути из выбранной вершины | Верно |

В результате тестирования было выявлено, что программа успешно находит кратчайшие пути, использую алгоритм Дейкстры.

6.Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 6 вершинами (рисунок 7).

Начинаем алгоритм из вершины 0. Проверяем, если есть путь из вершины 0 в другие вершины, то идем дальше. Далее вычисления будут представлены в виде таблицы (таблица 2).

Таблица 2 - Ручной расчет программы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 |  | ∞ | ∞ | 84 | ∞ | ∞ |
| 2 |  | 123 | ∞ |  | 107 | ∞ |
| 3 |  |  | 128 |  |  | ∞ |
| 4 |  |  |  |  |  | 138 |

Таким образом, можно сделать вывод, что программа работает верно.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 8 - Тестирование работы программы

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры по поиску кратчайших путей из одной вершины ко всем остальным в IntelliJ IDEA CE.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Дейкстры. Углублены знания языка программирования Java.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Богомолов Э.В. Алгоритм Дейкстры. Разбор задач // URL: <https://habr.com/ru/company/otus/blog/599621/> (дата обращение: 20.10.2022)
2. Ананий В. Левитин Глава 9. Жадные методы: Алгоритм Дейкстры 189-195 стр.
3. Шилдт Герберт Java. Полное руководство, 10-е изд. ­1488 стр.

Приложение А.

Листинг программы.

import java.io.\*;  
import java.util.\*;  
  
public class Graph {  
  
 public static void main(String[] args) throws IOException {  
 Scanner choose = new Scanner(System.*in*);  
 int chs;  
  
 while (true) {  
 System.*out*.println("choose output's method: " + "1. Read from file " + "2. Random graph adjacency table");  
 if (choose.hasNextInt()) {  
 chs = choose.nextInt();  
 if (chs == 1) {  
 int[][] arr = *readFromFile*();  
 *printArray*(arr);  
 System.*out*.println(" Enter the number of node for Dijkstra algoritm: ");  
 while (true) {  
 if (choose.hasNextInt()) {  
 int getNum = choose.nextInt();  
 if (getNum < arr.length) {  
 *dijkstra*(arr, getNum);  
 System.*exit*(0);  
  
 } else {  
 System.*out*.println("Error, input correct value!");  
 }  
 }  
 }  
 } else if (chs == 2) {  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("Enter the value of nodes: ");  
 int valueOfNodes = sc.nextInt();  
 int[][] graph = *createMatrix*(valueOfNodes);  
 System.*out*.println("Graph adjacency table:");  
 *printArray*(graph);  
 *goToFile*(graph);  
 System.*out*.println(" Enter the number of node for Dijkstra algoritm: ");  
 while (true) {  
 if (choose.hasNextInt()) {  
 int getNum = choose.nextInt();  
 if (getNum < graph.length) {  
 *dijkstra*(graph, getNum);  
 System.*exit*(0);  
 } else {  
 System.*out*.println("Error, input correct value!");  
 }  
 }  
 }  
 }  
 System.*out*.println("Error, enter the 1 or 2!");  
 }  
 }  
 }  
  
 public static int[][] createMatrix(int value) {  
 int[][] gr = new int[value][value];  
 for (int i = 0; i < gr.length; i++) {  
 for (int j = i; j < gr[i].length; j++) {  
 if (i == j) {  
 gr[i][j] = 0;  
 } else {  
 gr[i][j] = (int) (Math.*random*() \* 2);  
 if (gr[i][j] == 1) {  
 gr[i][j] = (int) (Math.*random*() \* 100);  
 }  
 gr[j][i] = gr[i][j];  
 }  
 }  
 }  
 return gr;  
 }  
  
 public static void printArray(int[][] arr) {  
 for (int i = 0; i < arr.length; i++) {  
 for (int j = 0; j < arr[i].length; j++) {  
  
 System.*out*.print(String.*format*("%4d", arr[i][j]));  
 }  
 System.*out*.println();  
 }  
 }  
  
 public static void goToFile(int[][] array) {  
 try (FileWriter writer = new FileWriter("Matrix.txt")) {  
 for (int i = 0; i < array.length; ++i) {  
 for (int j = 0; j < array[i].length; ++j) {  
 writer.write(String.*format*(array[i][j] + " " ));  
 }  
 writer.write("\r\n");  
 }  
 } catch (IOException ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
  
 public static void infoToFile(int[] arr, int node) {  
 try (FileWriter writer = new FileWriter("Info.txt")) {  
 writer.write("Node# \t Minimum Distance from node: " + node + "\n");  
  
 for (int i = 0; i < arr.length; i++) {  
 if (arr[i] == Integer.*MAX\_VALUE* || arr[i] == 0) {  
 writer.write(i + " " + " \t\t\t " + " no path " + "\n");  
 } else {  
 writer.write(i + " " + " \t\t\t " + arr[i] + "\n");  
 }  
 }  
 } catch (IOException ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
  
 static void dijkstra(int graph[][], int src\_node) {  
 int dist[] = new int[graph.length];  
 Boolean marked[] = new Boolean[graph.length];  
  
 for (int i = 0; i < graph.length; i++) {  
 dist[i] = Integer.*MAX\_VALUE*;  
 marked[i] = false;  
 }  
  
 dist[src\_node] = 0;  
  
 for (int count = 0; count < graph.length - 1; count++) {  
 int u = *findMinVert*(dist, marked);  
 marked[u] = true;  
 for (int v = 0; v < graph.length; v++)  
 if (!marked[v] && graph[u][v] != 0 && dist[u] != Integer.*MAX\_VALUE* && dist[u] + graph[u][v] < dist[v])  
 dist[v] = dist[u] + graph[u][v];  
 }  
 *printMinpath*(dist, src\_node);  
 *infoToFile*(dist, src\_node);  
 }  
  
 public static int findMinVert(int[] dist, Boolean[] marked) {  
 int minVertex = -1;  
 for (int i = 0; i < dist.length; i++) {  
 if (!marked[i] && (minVertex == -1 || dist[i] < dist[minVertex])) {  
 minVertex = i;  
 }  
 }  
 return minVertex;  
 }  
  
 static void printMinpath(int arr[], int node) {  
 System.*out*.println("Node# \t Minimum Distance from node: " + node);  
 for (int i = 0; i < arr.length; i++) {  
 if (arr[i] == Integer.*MAX\_VALUE* || arr[i] == 0) {  
 System.*out*.println(i + " \t\t\t " + " no path ");  
 } else {  
 System.*out*.println(i + " \t\t\t " + arr[i]);  
 }  
 }  
 }  
  
 public static int[][] readFromFile() throws FileNotFoundException {  
 Scanner scanner = new Scanner(new File("Matrix.txt"));  
 String[] firstLine;  
  
 if (scanner.hasNextLine()) {  
 firstLine = scanner.nextLine().split(" ");  
 } else {  
 throw new RuntimeException("File is empty");  
 }  
 int size = firstLine.length;  
 int[][] matrix = new int[size][size];  
 System.*out*.println("Size of matrix: " + size);  
  
 matrix[0] = Arrays.*stream*(firstLine).mapToInt(Integer::*valueOf*).toArray();  
 for (int i = 1; i < size; i++) {  
 if (scanner.hasNextLine()) {  
 matrix[i] = Arrays.*stream*(scanner.nextLine().split(" ")).mapToInt(Integer::*valueOf*).toArray();  
 } else {  
 throw new RuntimeException("Размер матрицы не соотвествует требованиям");  
 }  
 }  
 return matrix;  
 }  
}