Итоговая работа

по дисциплине «Дискретная математика»

Выполнил

студент гр.3530904/80004 < > Мащенко Б.Б.

Преподаватель <> Тышкевич А.И.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

**Оглавление**

[Задача 3](#_Toc39062984)

[Код исходных данных 4](#_Toc39062985)

[Решение задачи №1 и №2 6](#_Toc39062986)

[Код программы для задачи №1 и №2 7](#_Toc39062987)

[Решение задачи №3 9](#_Toc39062988)

[Код программы для задачи №3 9](#_Toc39062989)

[Решение задачи №4 12](#_Toc39062990)

[Код программы для задачи №4 12](#_Toc39062991)

# Задача

(1) По алгоритму Флойда-Уоршалла найти матрицу весов маршрутов И матрицу номеров первых вершин этих маршрутов. Обратите внимание, граф простой, поэтому каждое ребро «проходимо» в любом направлении. То есть, если указано, например, w({v2, v5}) = 12, то возможен как маршрут через вершины 2 и 5, так и в обратном направлении через 5 и 2. В обоих случаях вес ребра равен 12. Поэтому матрица весов симметричная.

(2) Для указанной пары вершин vb ---> vt извлечь из матрицы маршрутов маршрут от вершины vb к вершине vt в виде последовательности номеров вершин.

(3) По алгоритму Дейкстры для этой же пары вершин найти вес и маршрут минимального веса.

(4) По алгоритму Прима для данного графа найти минимальный остов в виде списка ребер остова.

Исходные данные:

Var 51 : v3 ---> v5 w({v1,v2})=33, w({v1,v3})=22, w({v1,v8})=19, w({v2,v4})=13, w({v2,v7})=32, w({v3,v7})=21, w({v3,v8})=11, w({v4,v6})=4, w({v4,v8})=12, w({v5,v6})=31, w({v5,v7})=27, w({v6,v7})=28, w({v6,v8})=6, w({v7,v8})=7.

# Код исходных данных

public class SourceData *{* public static final Edge*[] graph* = new Edge*[] {* new Edge*(*1, 2, 33*)*,  
 new Edge*(*1, 3, 22*)*,  
 new Edge*(*1, 8, 19*)*,  
 new Edge*(*2, 4, 13*)*,  
 new Edge*(*2, 7, 32*)*,  
 new Edge*(*3, 7, 21*)*,  
 new Edge*(*3, 8, 11*)*,  
 new Edge*(*4, 6, 4*)*,  
 new Edge*(*4, 8, 12*)*,  
 new Edge*(*5, 6, 31*)*,  
 new Edge*(*5, 7, 27*)*,  
 new Edge*(*6, 7, 28*)*,  
 new Edge*(*6, 8, 6*)*,  
 new Edge*(*7, 8, 7*)*,  
 *}*;  
  
 public static final int *INF* = 1000;  
 public static final int *NODE\_COUNT* = *getNodeCount()*;

public static final int *from* = 3;  
 public static final int *to* = 5;

private SourceData*() {}* private static int getNodeCount*() {* int *count* = -1;  
 for *(*Edge *edge*: *graph) {* if *(edge*.first > *count) {  
 count* = *edge*.first;  
 *}* if *(edge*.second > *count) {  
 count* = *edge*.second;  
 *}  
 }* return *count*;  
 *}  
}*class Edge *{* final int first;  
 final int second;  
 final int weight;  
  
 Edge*(*int *first*, int *second*, int *weight) {* this.first = *first*;  
 this.second = *second*;  
 this.weight = *weight*;  
 *}*

@Override  
public boolean equals*(*Object *o) {* if *(*this == *o)* return true;  
 if *(o* == null || getClass*()* != *o*.getClass*())* return false;  
 Edge *edge* = *(*Edge*) o*;  
 return *(*first == *edge*.first && second == *edge*.second  
 || first == *edge*.second && second == *edge*.first*)* &&  
 weight == *edge*.weight;  
*}*@Override  
public int hashCode*() {* return Objects.*hash(*first, second, weight*)*;  
*}*

@Override  
public String toString*() {* return "Edge{" +  
 "first=" + first +  
 ", second=" + second +  
 ", weight=" + weight +  
 '}';  
*}  
}*

Исходные данные:

INF = 1000

Matrix

0 33 22 1000 1000 1000 1000 19

33 0 1000 13 1000 1000 32 1000

22 1000 0 1000 1000 1000 21 11

1000 13 1000 0 1000 4 1000 12

1000 1000 1000 1000 0 31 27 1000

1000 1000 1000 4 31 0 28 6

1000 32 21 1000 27 28 0 7

19 1000 11 12 1000 6 7 0

# Решение задачи №1 и №2

*Результаты работы программы:*

Matrix

0 33 22 29 53 25 26 19

33 0 34 13 48 17 30 23

22 34 0 21 45 17 18 11

29 13 21 0 35 4 17 10

53 48 45 35 0 31 27 34

25 17 17 4 31 0 13 6

26 30 18 17 27 13 0 7

19 23 11 10 34 6 7 0

Paths

1 1 1 6 7 8 8 1

2 2 8 2 6 4 8 6

3 4 3 6 7 8 8 3

8 4 8 4 6 4 8 6

8 4 8 6 5 5 5 7

8 4 8 6 6 6 8 6

8 4 8 6 7 8 7 7

8 4 8 6 7 8 8 8

Result

3 -> 8 -> 7 -> 5

Наименьшая стоимость пути из 3 в 5 = 45

Кратчайший путь из вершины 3 в вершину 5 виден из матрицы вершин маршрутов: paths[3][5] = 7, paths[3][7] = 8, paths[3][8] = 3. Тогда путь будет   
3 -> 8 -> 7 -> 5

# Код программы для задачи №1 и №2

import java.util.ArrayDeque;  
import java.util.Arrays;  
import java.util.Deque;  
  
public class Task1 *{* private static final int*[][] matrix* = new int*[*SourceData.*NODE\_COUNT][*SourceData.*NODE\_COUNT]*;  
 private static final int*[][] paths* = new int*[*SourceData.*NODE\_COUNT][*SourceData.*NODE\_COUNT]*;  
 private static final Deque*<*Integer*> result* = new ArrayDeque*<>()*;  
  
 public static void main*(*String*[] args) {  
 setupMatrix()*;  
 *printMatrix()*;  
 *doAlgorithm()*;  
 *printMatrix()*;  
 *printPaths()*;  
 *printResult(*SourceData.*from*, SourceData.*to)*;

*}* private static void printResult*(*int *from*, int *to) {* int *currentNode* = *to* - 1;  
 while *(currentNode* != *from* - 1*) {  
 result*.push*(currentNode* + 1*)*;  
 *currentNode* = *paths[from* - 1*][currentNode]*;  
 *}  
 result*.push*(from)*;  
 System.*out*.println*(*"\tResult"*)*;  
 while *(*!*result*.isEmpty*()) {* if *(result*.size*()* == 1*) {* System.*out*.println*(result*.pop*())*;  
 *}* else *{* System.*out*.print*(result*.pop*()* + " -> "*)*;  
 *}  
 }  
 }* private static void printMatrix*() {* System.*out*.println*(*"\tMatrix"*)*;  
 for *(*int*[] array* : *matrix) {* for *(*int *path* : *array) {* System.*out*.print*(path* + " "*)*;  
 *}* System.*out*.println*()*;  
 *}* System.*out*.println*()*;  
 *}* private static void printPaths*() {* System.*out*.println*(*"\tPaths"*)*;  
 for *(*int*[] array* : *paths) {* for *(*int *path* : *array) {* System.*out*.print*(path* + 1 + " "*)*;  
 *}* System.*out*.println*()*;  
 *}* System.*out*.println*()*;  
 *}* private static void doAlgorithm*() {* for *(*int *k* = 0; *k* < *matrix*.length; *k*++*) {* for *(*int *i* = 0; *i* < *matrix*.length; *i*++*) {* for *(*int *j* = 0; *j* < *matrix*.length; *j*++*) {* final int *newPath* = *matrix[i][k]* + *matrix[k][j]*;  
 if *(matrix[i][k]* < SourceData.*INF* && *matrix[k][j]* < SourceData.*INF) {* if *(newPath* < *matrix[i][j]) {  
 matrix[i][j]* = *newPath*;  
 *paths[i][j]* = *paths[k][j]*;  
 *}  
 }  
 }  
 }  
 }  
 }* private static void setupMatrix*() {* for *(*int *i* = 0; *i* < *matrix*.length; *i*++*) {* Arrays.*fill(matrix[i]*, SourceData.*INF)*;  
 Arrays.*fill(paths[i]*, -1*)*;  
 *matrix[i][i]* = 0;  
 *paths[i][i]* = *i*;  
 *}* for *(*Edge *edge* : SourceData.*graph) {  
 matrix[edge*.first - 1*][edge*.second - 1*]* = *matrix[edge*.second - 1*][edge*.first - 1*]* = *edge*.weight;  
 *paths[edge*.second - 1*][edge*.first - 1*]* = *edge*.second - 1;  
 *paths[edge*.first - 1*][edge*.second - 1*]* = *edge*.first - 1;  
 *}  
 }  
}*

# Решение задачи №3

*Результат работу программы*

Массив весов кратчайших расстояний от вершины 3 до других вершин:

Distance

22 50 0 21 45 17 18 11

Массив вершин кратчайших расстояний от вершины 3 до других вершин:

Paths

3 7 0 6 7 8 8 3

Кратчайший путь из вершины 3 в 5

Result

3 -> 8 -> 7 -> 5

Стоимость = dist[5] = 45

Кратчайший путь из вершины 3 в вершину 5 виден из массива вершин маршрутов: paths[5] = 7, paths[7] = 8, paths[8] = 3. Тогда путь будет   
3 -> 8 -> 7 -> 5

# Код программы для задачи №3

import java.util.ArrayDeque;  
import java.util.Arrays;  
import java.util.Deque;  
  
class Task2 *{* private static final int*[] dist* = new int*[*SourceData.*NODE\_COUNT]*;  
 private static final int*[] paths* = new int*[*SourceData.*NODE\_COUNT]*;  
 private static final boolean*[] used* = new boolean*[*SourceData.*NODE\_COUNT]*;  
 private static final Deque*<*Integer*> result* = new ArrayDeque*<>()*;  
  
  
 public static void main*(*String*[] args) {  
 setupArray()*;  
 *doAlgorithm(*SourceData.*from)*;  
 *printArray()*;  
 *printPaths()*;  
 *printResult()*;  
 *}* private static void printResult*() {* int *currentNode* = SourceData.*to*;  
 while *(currentNode* != 3*) {  
 result*.push*(currentNode)*;  
 *currentNode* = *paths[currentNode* - 1*]*;  
 *}  
 result*.push*(currentNode)*;  
 System.*out*.println*(*"\tResult"*)*;  
 while *(*!*result*.isEmpty*()) {* if *(result*.size*()* == 1*) {* System.*out*.println*(result*.pop*())*;  
 *}* else *{* System.*out*.print*(result*.pop*()* + " -> "*)*;  
 *}  
 }  
 }* private static void doAlgorithm*(*int *currentNode) {* int *minNode* = -1;  
 int *minWeight* = Integer.*MAX\_VALUE*;  
 for *(*Edge *node* : SourceData.*graph) {* if *(*!*used[node*.second - 1*]* && *node*.first == *currentNode) {* if *(dist[node*.first - 1*]* + *node*.weight < *dist[node*.second - 1*]) {  
 dist[node*.second - 1*]* = *dist[node*.first - 1*]* + *node*.weight;  
 *paths[node*.second - 1*]* = *currentNode*;  
 *}* if *(dist[node*.second - 1*]* < *minWeight) {  
 minNode* = *node*.second;  
 *minWeight* = *dist[node*.second - 1*]*;  
 *}  
 }* else if *(*!*used[node*.first - 1*]* && *node*.second == *currentNode) {* if *(dist[node*.second - 1*]* + *node*.weight < *dist[node*.first - 1*]) {  
 dist[node*.first - 1*]* = *dist[node*.second - 1*]* + *node*.weight;  
 *paths[node*.first - 1*]* = *currentNode*;  
 *}* if *(dist[node*.first - 1*]* < *minWeight) {  
 minNode* = *node*.first;  
 *minWeight* = *dist[node*.first - 1*]*;  
 *}  
 }  
 }  
 used[currentNode* - 1*]* = true;  
 if *(minNode* != -1*) {  
 doAlgorithm(minNode)*;  
 *}  
 }* private static void printArray*() {* System.*out*.println*(*"\tDistance"*)*;  
 for *(*int *weight* : *dist) {* System.*out*.print*(weight* + " "*)*;  
 *}* System.*out*.println*()*;  
 System.*out*.println*()*;  
 *}* private static void printPaths*() {* System.*out*.println*(*"\tPaths"*)*;  
 for *(*int *weight* : *paths) {* System.*out*.print*(weight* + " "*)*;  
 *}* System.*out*.println*()*;  
 System.*out*.println*()*;  
 *}* private static void setupArray*() {* Arrays.*fill(dist*, SourceData.*INF)*;  
 *dist[*SourceData.*from* - 1*]* = 0;  
 Arrays.*fill(paths*, 0*)*;  
 Arrays.*fill(used*, false*)*;  
 *}  
}*

# Решение задачи №4

*Результаты работы программы:*

Минимальный остов:

Edge{first=3, second=8, weight=11}

Edge{first=6, second=8, weight=6}

Edge{first=4, second=6, weight=4}

Edge{first=7, second=8, weight=7}

Edge{first=2, second=4, weight=13}

Edge{first=1, second=8, weight=19}

Edge{first=5, second=7, weight=27}

# Код программы для задачи №4

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class Task3 *{* private static List*<*Edge*> nodes* = new ArrayList*<>(*SourceData.*graph*.length*)*;  
 private static List*<*Edge*> ostov* = new ArrayList*<>(*SourceData.*NODE\_COUNT* - 1*)*;  
 private static List*<*Integer*> nodesInOstov* = new ArrayList*<>(*SourceData.*NODE\_COUNT)*;  
  
 public static void main*(*String*[] args) {  
 setupData()*;  
 *doAlgorithm()*;  
 *printResult()*;  
 *}* private static void setupData*() {  
 nodes*.addAll*(*Arrays.*asList(*SourceData.*graph))*;  
 *nodesInOstov*.add*(*SourceData.*from)*;  
 *}* private static void printResult*() {  
 ostov*.forEach*(*System.*out*::println*)*;  
 *}* private static void doAlgorithm*() {* while *(nodesInOstov*.size*()* != SourceData.*NODE\_COUNT) {* Edge *minEdge* = null;  
 int *minNode* = -1;  
 for *(*Edge *node* : *nodes) {* if *(nodesInOstov*.contains*(node*.first*)* && !*nodesInOstov*.contains*(node*.second*)* && *(minEdge* == null || *node*.weight < *minEdge*.weight*)) {  
 minEdge* = *node*;  
 *minNode* = *node*.second;  
 *}* else if *(nodesInOstov*.contains*(node*.second*)* && !*nodesInOstov*.contains*(node*.first*)* && *(minEdge* == null || *node*.weight < *minEdge*.weight*)){  
 minEdge* = *node*;  
 *minNode* = *node*.first;  
 *}  
 }  
 nodesInOstov*.add*(minNode)*;  
 *ostov*.add*(minEdge)*;  
 *nodes*.remove*(minEdge)*;  
 *}  
 }  
}*