|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт кибербезопасности и цифровых технологий

Кафедра КБ-2 «Институт кибербезопасности и цифровых технологий»

**ОТЧЕТ   
о выполнении домашней работе**

**«Алгоритмы на графах»**

**по дисциплине   
«Технологии и методы программирования»**

**Вариант № 46**

Выполнил: студент 2 курса

группы БББО-01-21

Мысливец Л.В.

шифр 21Б0746

Москва 2022 г.

**Задание на № 1.**

В рамках домашней работы №1 требуется программно реализовать в виде программы абстрактный тип данных «Граф».

Операторы (операции) АТД «Граф» *функционально* должны выполнять следующие операции (названия операций – *примерные*):

* FIRST(*v*) - возвращает *индекс* первой вершины, смежной с вершиной *v.* Если вершина *v* не имеет смежных вершин, то возвращается "нулевая" вершина L.
* NEXT(*v*, *i*)- возвращает *индекс* вершины, смежной с вершиной *v*, следующий за индексом *i*. Если *i —* это индекс последней вершины, смежной с вершиной *v*, то возвращается L.
* VERTEX(*v*, *i*) - возвращает *вершину* с индексом *i* из множества вершин, смежных с *v.*
* ADD\_V(<имя>, <метка, mark>) - добавить УЗЕЛ
* ADD\_Е(v, w, c) - добавить ДУГУ (здесь c — вес, цена дуги (v,w))
* DEL\_V(<имя>) - удалить УЗЕЛ
* DEL\_Е(v, w) – удалить ДУГУ
* EDIT\_V(<имя>, <новое значение метки или маркировки>) - изменить метку (маркировку) УЗЛА
* EDIT\_Е(v, w, <новый вес дуги>) - изменить вес ДУГИ

Используя разработанный АТД и указанный набор операций, необходимо реализовать задание, согласно варианту 46 с использованием методов АТД «Граф».

**Вариант № 46.**

**Способ представления графа: Матрица инцидентности**

**Алгоритм: Вывести на экран все существующие пути в ациклическом орграфе**

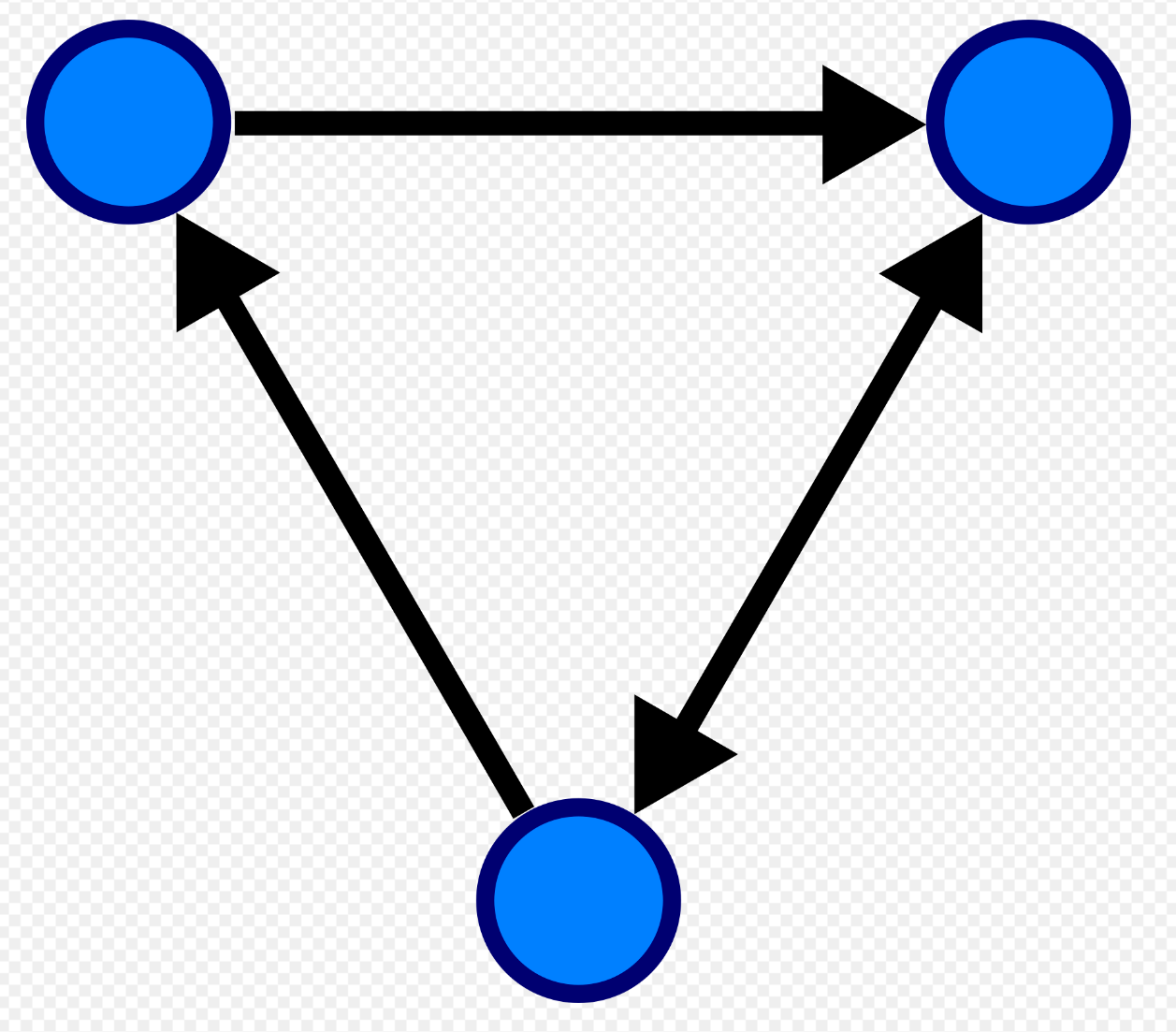
**Теория о графах.**

**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.

**Ориентированный граф.**

**Ориентированный граф** (кратко **орграф**) — (мульти) граф, ребрам которого присвоено направление. Направленные рёбра именуются также *дугами*, а в некоторых источниках и просто рёбрами. Граф, ни одному ребру которого не присвоено направление, называется неориентированным графом или неорграфом.

Ниже приведен пример орграфа:

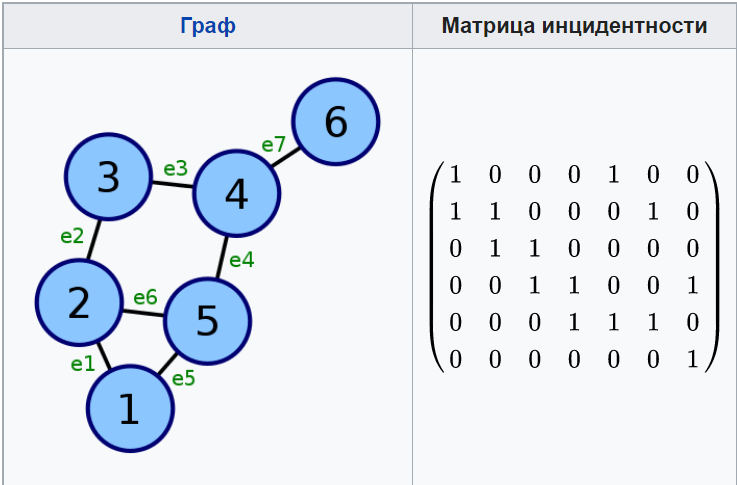


**Матрица инцидентности.**

**Матрица инцидентности** — одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро(дуга) и вершина). Столбцы матрицы соответствуют ребрам, строки — вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

В случае ориентированного графа каждой дуге <x,y> ставится в соответствующем столбце: «1» в строке вершины x и «-1» в строке вершины y; если связи между вершиной и ребром нет, то в соответствующую ячейку ставится «0».

Ниже приведен пример матрицы инцидентности:



Строки соответствуют вершинам от 1 до 6, а столбцы — ребрам e1–e7. Например, единицы во втором столбце во 2-й и 3-й строчках означают, что ребро e2 соединяет вершины 2 и 3.

**Обходы графов в глубину.**

**Поиск в глубину или обход в глубину** — это рекурсивный алгоритм по поиску всех вершин графа или дерева. Обход подразумевает под собой посещение всех вершин графа.

**Алгоритм поиска в глубину:**

Стандартная реализация поиска в глубину помещает каждую вершину (узел, node) графа в одну из двух категорий:

1. Пройденные (Visited).
2. Не пройденные (Not Visited).

Цель алгоритма состоит в том, чтобы пометить каждую вершину как “Пройденная”, избегая при этом циклов.

Алгоритм поиска в глубину работает следующим образом:

1. Начните с того, что поместите любую вершину графа на вершину стека.
2. Возьмите верхний элемент стека и добавьте его в список “Пройденных”.
3. Создайте список смежных вершин для этой вершины. Добавьте те вершины, которых нет в списке “Пройденных”, в верх стека.
4. Необходимо повторять шаги 2 и 3, пока стек не станет пустым.

**Листинг программы с реализацией АТД «Граф».**

class Graph<T> {

// Мы используем Hashmap для хранения ребер в графе

private Map<T, List<T>> map = new HashMap<>();

private Map<T, String> markslist = new HashMap<>();

private Map<String, Integer> weightlist = new HashMap<>();

//добавить узел

public void addVertex(T s) {

map.put(s, new LinkedList<T>());

}

//добавить узел и (маркер)

public void addVertex(T s, String mark) {

markslist.put(s, mark);

map.put(s, new LinkedList<T>());

}

// Эта функция создает дугу

// между source и destination

public void addEdge(

T source, T destination, int weight, boolean bidirectional) {

String indexEdge = (source + "" + destination);

if (!map.containsKey(source))

addVertex(source);

if (!map.containsKey(destination))

addVertex(destination);

map.get(source).add(destination);

weightlist.put(indexEdge, weight);

if (bidirectional == true) {

String indexEdgeRev = (destination + "" + source);

map.get(destination).add(source);

weightlist.put(indexEdgeRev, weight);

}

}

public void addEdge(T source, T destination, String mark1, String mark2,

int weight, boolean bidirectional) {

String indexEdge = (source + "" + destination);

if (!map.containsKey(source))

addVertex(source, mark1);

if (!map.containsKey(destination))

addVertex(destination, mark2);

map.get(source).add(destination);

weightlist.put(indexEdge, weight);

if (bidirectional == true) {

String indexEdgeRev = (destination + "" + source);

map.get(destination).add(source);

weightlist.put(indexEdgeRev, weight);

}

}

//удалить узел

public void deleteVertex(T s) {

map.remove(s);

}

//удалить дугу

public void deleteEdge(T source, T destination) {

map.get(source).remove(destination);

}

public void nextValue(T sourse, int i) {

if (map.containsKey(i + 1)) {

if (map.get(sourse).isEmpty()) { //если в заданном значении ниче нет

for (T kek : map.keySet()) { //проверяем для всех значений графа

if (map.get(kek).contains(

sourse)) { //если кто-то из значений имеет наше значение

if ((int) kek > i) { //и если оно больше i

System.out.println(kek); //выводим его

}

}

}

}

for (T kek : map.get(sourse)) {

System.out.print(kek + ">" + i + ":");

if ((int) kek > i) {

System.out.println("индекс вершины, смежной с вершиной " + sourse

+ ", следующий за индексом " + i + ": ");

}

}

} else {

System.out.println("такой вешины нет \uF04C");

}

}

public void first(T sourse) {

System.out.println(

"первая смежная вершина с " + sourse + ":" + map.get(sourse).get(0));

}

public void editVertex(int index, String mark) {

for (T keys : map.keySet()) {

if (keys.equals(index)) {

markslist.replace(keys, mark);

}

}

}

public void editEdge(T source, T destination, int newweight) {

String indexEdge = (source + "" + destination);

for (T v : map.keySet()) {

if (v.equals(source)) {

for (T w : map.get(v)) {

if (w.equals(source)) {

weightlist.replace(indexEdge, newweight);

}

}

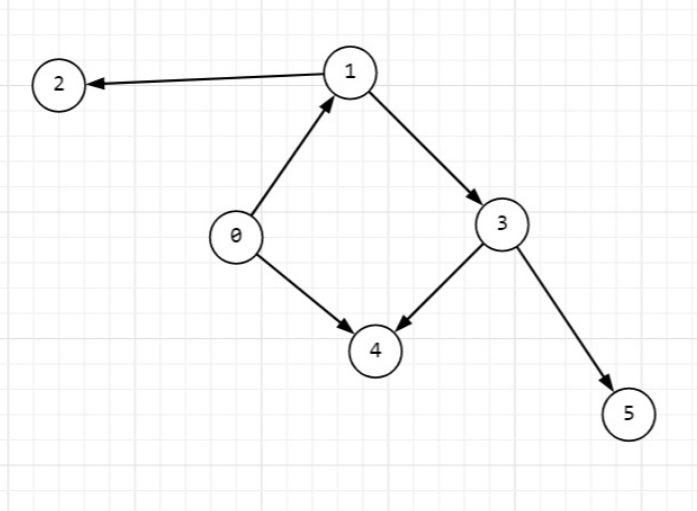
}

}

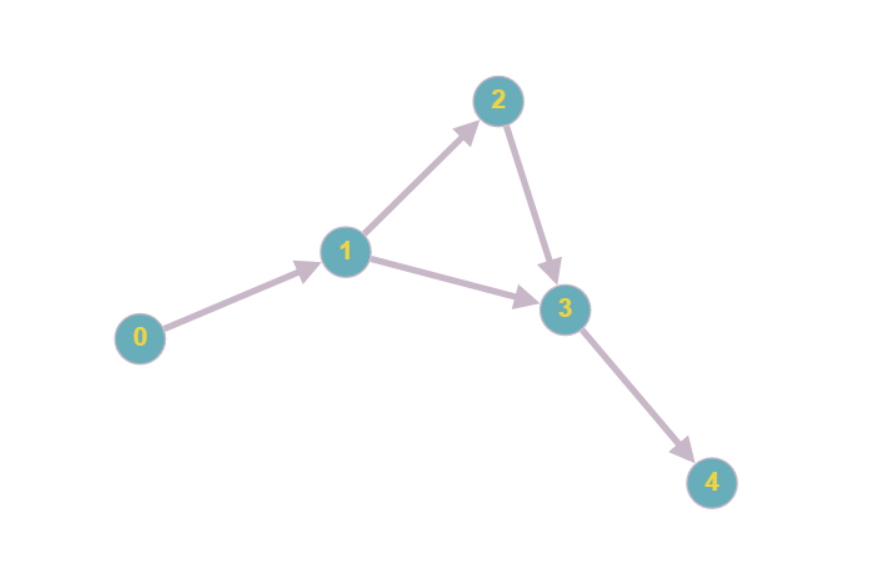
}

**Контрольный пример**

пример 1(рисунок):

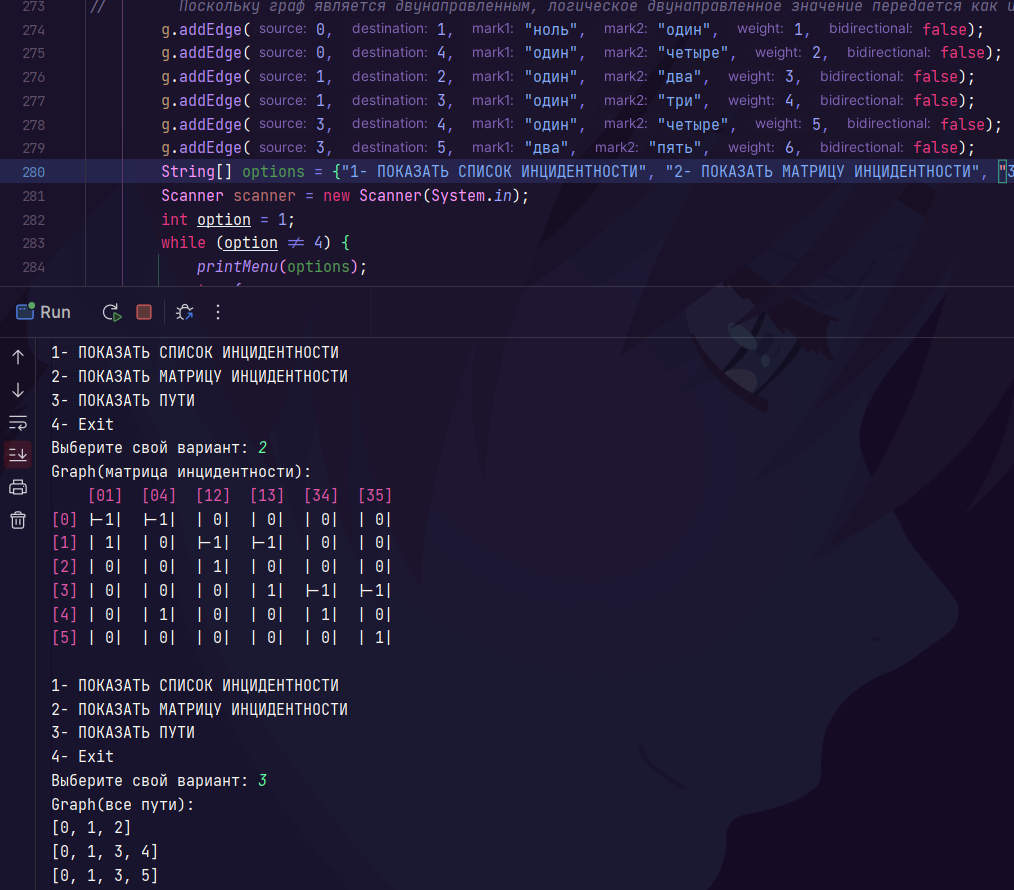


пример 2(рисунок):

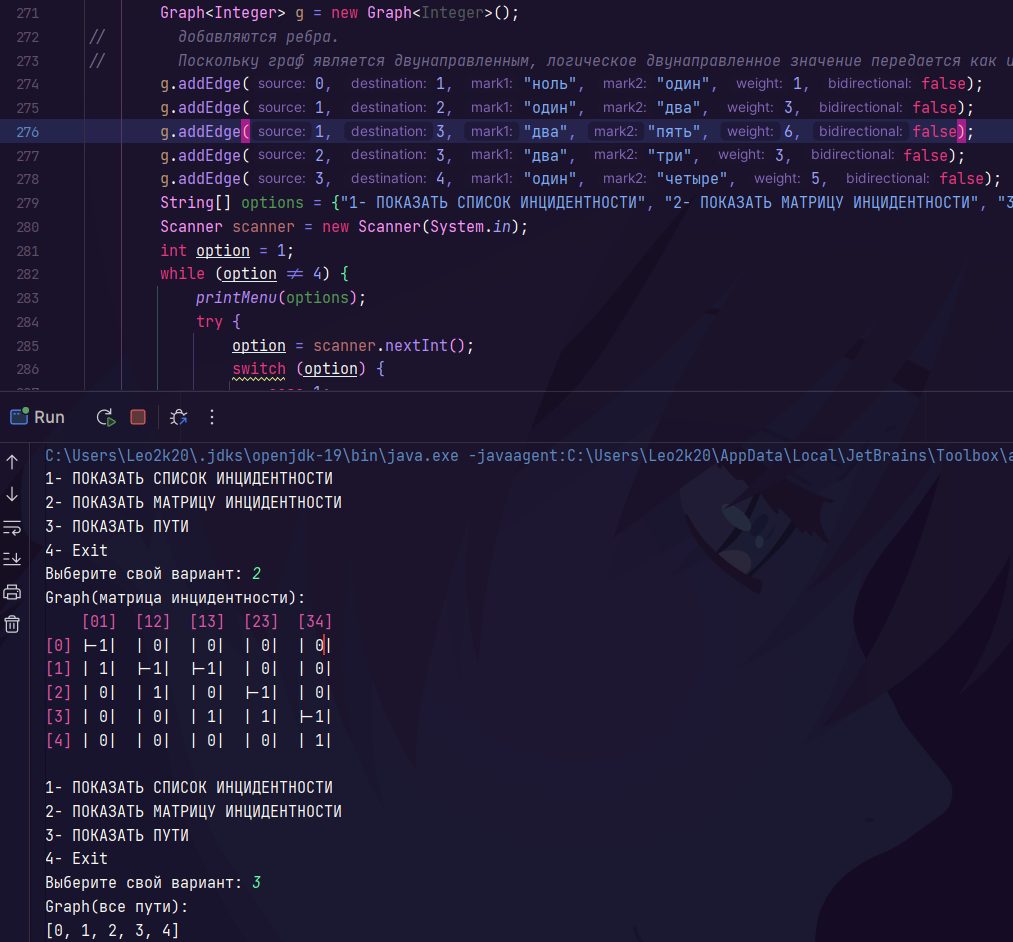


**Скриншот работы программы:**

скриншот для примера 1:



скриншот для примера 2:



**Выводы.**

По результатам экспериментов было установлено, что метод обхода графов в глубину, при невысокой сложности самого алгоритма, очень эффективен при работе с графами.

**Литература:**

1. Ахо А. В. Структуры данных и алгоритмы. – Издательский дом Вильямс, 2000.
2. Белов В. В., Воробьев Е. М., Шаталов В. Е. - Теория графов. — М.: Высш. школа, 1976. — С. 392.
3. Берж К. - Теория графов и ее приложения. М.: ИЛ, 1962. 320c.

**Приложение 1. Листинг программы**

import static java.lang.System.exit;

import java.util.\*;

class Graph<T> {

private Map<T, List<T>> map = new HashMap<>();

private Map<T, String> markslist = new HashMap<>();

private Map<String, Integer> weightlist = new HashMap<>();

public void addVertex(T s) {

map.put(s, new LinkedList<T>());

}

public void addVertex(T s, String mark) {

markslist.put(s, mark);

map.put(s, new LinkedList<T>());

}

public void addEdge(

T source, T destination, int weight, boolean bidirectional) {

String indexEdge = (source + "" + destination);

if (!map.containsKey(source))

addVertex(source);

if (!map.containsKey(destination))

addVertex(destination);

map.get(source).add(destination);

weightlist.put(indexEdge, weight);

if (bidirectional == true) {

String indexEdgeRev = (destination + "" + source);

map.get(destination).add(source);

weightlist.put(indexEdgeRev, weight);

}

}

public void addEdge(T source, T destination, String mark1, String mark2,

int weight, boolean bidirectional) {

String indexEdge = (source + "" + destination);

if (!map.containsKey(source))

addVertex(source, mark1);

if (!map.containsKey(destination))

addVertex(destination, mark2);

map.get(source).add(destination);

weightlist.put(indexEdge, weight);

if (bidirectional == true) {

String indexEdgeRev = (destination + "" + source);

map.get(destination).add(source);

weightlist.put(indexEdgeRev, weight);

}

}

//удалить узел

public void deleteVertex(T s) {

map.remove(s);

}

//удалить дугу

public void deleteEdge(T source, T destination) {

map.get(source).remove(destination);

}

public void getVertexCount() {

System.out.println("У графа " + map.keySet().size() + " вершин(ы)");

}

public void getEdgesCount(boolean bidirection) {

int count = 0;

for (T v : map.keySet()) {

count += map.get(v).size();

}

if (bidirection == true) {

count = count / 2;

}

System.out.println("у графа " + count + " ребер.");

}

public void nextValue(T sourse, int i) {

if (map.containsKey(i + 1)) {

if (map.get(sourse).isEmpty()) { //если в заданном значении ниче нет

for (T kek : map.keySet()) { //проверяем для всех значений графа

if (map.get(kek).contains(

sourse)) { //если кто-то из значений имеет наше значение

if ((int) kek > i) { //и если оно больше i

System.out.println(kek); //выводим его

}

}

}

}

for (T kek : map.get(sourse)) {

System.out.print(kek + ">" + i + ":");

if ((int) kek > i) {

System.out.println("индекс вершины, смежной с вершиной " + sourse

+ ", следующий за индексом " + i + ": ");

}

}

} else {

System.out.println("такой вешины нет \uF04C");

}

}

public void first(T sourse) {

System.out.println(

"первая смежная вершина с " + sourse + ":" + map.get(sourse).get(0));

}

public void editVertex(int index, String mark) {

for (T keys : map.keySet()) {

if (keys.equals(index)) {

markslist.replace(keys, mark);

}

}

}

public void editEdge(T source, T destination, int newweight) {

String indexEdge = (source + "" + destination);

for (T v : map.keySet()) {

if (v.equals(source)) {

for (T w : map.get(v)) {

if (w.equals(source)) {

weightlist.replace(indexEdge, newweight);

}

}

}

}

}

Map<T, Boolean> visited = new HashMap<>();

ArrayList<T> path = new ArrayList<>();

public void dfs(T vertex) {

visited.put(vertex, true);

ListIterator<T> ite = map.get(vertex).listIterator();

if (map.get(vertex).isEmpty()) {

path.add(vertex);

System.out.println(path);

path.remove(vertex);

}

while (ite.hasNext()) {

T adj = ite.next();

path.remove(vertex);

if (visited.get(adj) == null) {

path.add(vertex);

dfs(adj);

}

}

}

@Override

public String toString() {

StringBuilder builder = new StringBuilder();

for (T v : map.keySet()) {

if ((map.get(v).isEmpty())) {

builder.append(v.toString() + "(" + markslist.get(v) + ")"

+ ":я пустой");

} else {

builder.append(v.toString() + "(" + markslist.get(v) + ")"

+ "=>");

for (T w : map.get(v)) {

builder.append(w.toString() + "{" + weightlist.get(v + "" + w) + "}"

+ "; ");

}

}

builder.append("\n");

}

return (builder.toString());

}

public String showGraph() {

StringBuilder builder1 = new StringBuilder();

ArrayList<String> rowlist = new ArrayList();

ArrayList<String> columnlist = new ArrayList();

final String RESET = "\u001B[0m";

final String RED = "\u001B[31m";

//получаем массивы с элементами row и column

for (T v : map.keySet()) {

rowlist.add(v + "");

for (T w : map.get(v)) {

columnlist.add(v + "" + w);

}

}

//заподняем билдер элементами(подписанная матрица инцидентности)

builder1.append("\t");

for (int i = 0; i < columnlist.size(); i++) {

builder1.append(RED + ""

+ "[" + columnlist.get(i) + "] " + RESET);

}

builder1.append("\n");

//заполняем двумерный массив

int[][] matrix = new int[columnlist.size()][rowlist.size()];

for (int i = 0; i < columnlist.size(); i++) {

for (int j = 0; j < rowlist.size(); j++) {

String tmp1 = columnlist.get(i);

String tmp2 = rowlist.get(j);

if (columnlist.get(i).contains(rowlist.get(j))) {

if (columnlist.get(i).indexOf(rowlist.get(j)) == 0) {

matrix[j][i] = -1;

} else {

matrix[j][i] = 1;

}

} else {

matrix[j][i] = 0;

}

}

}

//заподняем билдер элементами(подписанная матрица инцидентности)(column)

int i = 0;

for (T v : map.keySet()) {

builder1.append(RED + "[" + v + "]" + RESET + " ");

for (int j = 0; j < columnlist.size(); j++) {

if (matrix[i][j] == -1) {

builder1.append("|" + matrix[i][j] + "| ");

} else {

builder1.append("| " + matrix[i][j] + "| ");

}

}

i++;

builder1.append("\n");

}

return (builder1.toString());

}

}

public class Main {

public static void printMenu(String[] options) {

for (String option : options) {

System.out.println(option);

}

System.out.print("Выберите свой вариант: ");

}

public static void main(String args[]) {

// Объект графа создан.

Graph<Integer> g = new Graph<Integer>();

// добавляются ребра.

// Поскольку граф является двунаправленным, логическое

// двунаправленное значение передается как истинное.

g.addEdge(0, 1, "ноль", "один", 1, false);

g.addEdge(1, 2, "один", "два", 3, false);

g.addEdge(1, 3, "два", "пять", 6, false);

g.addEdge(2, 3, "два", "три", 3, false);

g.addEdge(3, 4, "один", "четыре", 5, false);

String[] options = {

"1- ПОКАЗАТЬ СПИСОК ИНЦИДЕНТНОСТИ",

"2- ПОКАЗАТЬ МАТРИЦУ ИНЦИДЕНТНОСТИ",

"3- ПОКАЗАТЬ ПУТИ",

"4- Exit",

};

Scanner scanner = new Scanner(System.in);

int option = 1;

while (option != 4) {

printMenu(options);

try {

option = scanner.nextInt();

switch (option) {

case 1:

System.out.println("Graph(список инцидентности):\n" + g.toString());

break;

case 2:

System.out.println(

"Graph(матрица инцидентности):\n" + g.showGraph());

break;

case 3:

System.out.println("Graph(все пути):");

g.dfs(0);

break;

case 4:

exit(0);

}

} catch (Exception ex) {

System.out.println("Введите целое число от 1 до " + options.length);

scanner.next();

}

}

}

}