Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рязанский государственный радиотехнический университет

имени В. Ф. Уткина»

Кафедра САПР ВС

Отчет о лабораторной работе №2

**«Поиск кратчайших путей»**

по дисциплине «Дискретная математика»

**Выполнил:**

ст. гр. 245

Бригада № 1

Сокол Илья

Лапин Кирилл

Москвитин Дмитрий

**Проверил:**

доц. каф. САПР ВС

Бакулева М.А.

доц. каф. САПР ВС

Орешков В.И.

Рязань 2024

**Цель работы:** изучение алгоритмов поиска кратчайших путей на графах на примере метода динамического программирования.

**Задание:** создать пометки для всех вершин L(xi) и найти кратчайший путь от x1 до xn.

**Блок-схема:**

Построим блок-схему алгоритма создания пометок для всех вершин (рисунок 1):

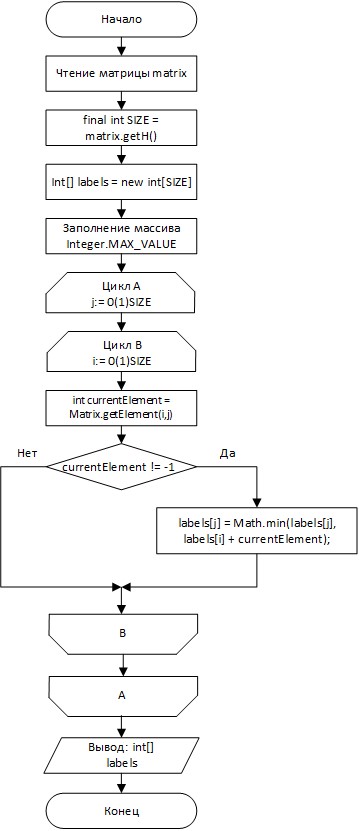
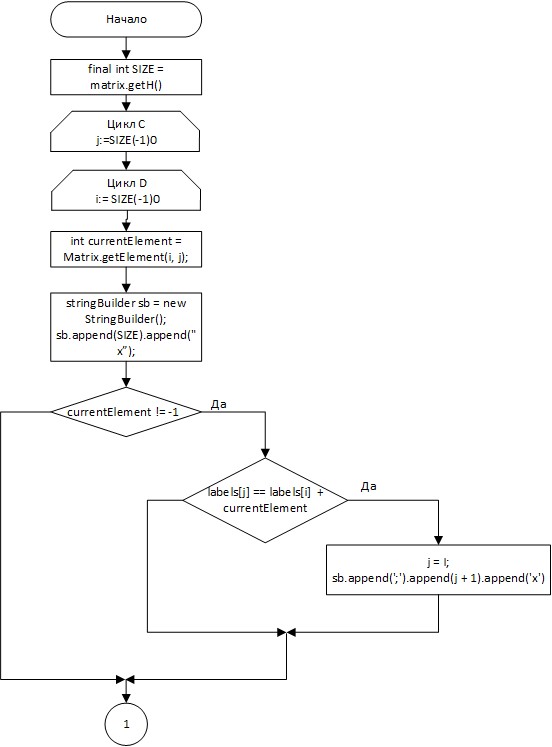


Рисунок 1 – Алгоритм создания пометок всех вершин

Построим блок-схему алгоритма нахождения кратчайшего пути от x1 до x2 (рисунок 2-3):

  
Рисунок 2 – Алгоритм нахождения кратчайшего пути

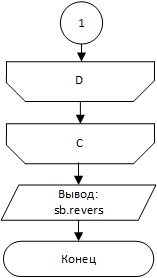


Рисунок 3 – Продолжение алгоритма нахождения кратчайшего пути

**Листинг программы:**

Разработанная программа на языке Java представлена в таблицах 1-5:

Таблица 1 – Главный класс программы

|  |
| --- |
| package application;  /\*  Сделать пометки всех вершин L(x\_i)  Найти кратчайший путь от x\_1 до x\_n  \*/  import exceptions.FileInputException;  import java.sql.SQLOutput;  public class TaskMain {  public static void main(String[] args) {  Matrix matrix = null;  try {  matrix = new Matrix(InOut.*readMatrix*("files/first.txt"));  } catch (FileInputException e) {  System.*out*.println(e.getMessage());  }  System.*out*.println("Исходная матрица смежности:");  InOut.*printMatrix*(matrix);   System.*out*.println(PathFinder.*findShortestPath*(matrix));  } } |

Таблица 2 – Класс объекта Матрица

|  |
| --- |
| package application;  import java.util.Arrays; import java.util.Objects;  public class Matrix implements Cloneable {  // Fields  private int[][] matrix;  private int v; // vertical size  private int h; // horizontal size   // Constructors  public Matrix() {};   public Matrix(int v, int h) {  this.v = v;  this.h = h;  this.matrix = new int[v][h];  }   public Matrix(int[][] matrix) {  this.matrix = matrix;  this.v = matrix.length;  this.h = matrix[0].length;  }   public Matrix(int size) {  this.matrix = new int[size][size];  this.v = size;  this.h = size;  }   // Getters and setters  public void setV(int v) {  this.v = v;  }   public void setH(int h) {  this.h = h;  }   public void setMatrix(int[][] matrix) {  this.matrix = matrix;  }   public void setElement(int i, int j, int value) {  this.matrix[i][j] = value;  }   public int getV() {  return v;  }   public int getH() {  return h;  }   public int[][] getMatrix() {  return matrix;  }   public int getElement(int i, int j) {  return matrix[i][j];  }   public int[] getRow(int i) {  return matrix[i];  }   // Methods  @Override  protected Object clone() throws CloneNotSupportedException {  return super.clone();  }   @Override  public boolean equals(Object object) {  if (this == object) return true;  if (object == null || getClass() != object.getClass()) return false;  Matrix matrix1 = (Matrix) object;  return getV() == matrix1.getV() && getH() == matrix1.getH() &&  Arrays.deepEquals(getMatrix(), matrix1.getMatrix());  }   @Override  public int hashCode() {  int result = Objects.hash(getV(), getH());  result = 31 \* result + Arrays.deepHashCode(getMatrix());  return result;  }   @Override  public String toString() {  return "Matrix{" +  "matrix=" + Arrays.deepToString(matrix) +  ", v=" + v +  ", h=" + h +  '}';  } } |

Таблица 3 – Класс считывания и вывода объекта Матрица

|  |
| --- |
| package application;  import exceptions.FileInputException; import java.io.File; import java.io.FileNotFoundException; import java.util.Scanner;  public class InOut {  public static int[][] readMatrix(String filename) throws FileInputException {  int[][] inMatrix;  try {  File file = new File(filename);  Scanner scanner = new Scanner(file);   int rows = scanner.nextInt();  int cols = scanner.nextInt();  inMatrix = new int[rows][cols];   for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  if (scanner.hasNextInt()) {  inMatrix[i][j] = scanner.nextInt();  } else {  scanner.close();  throw new FileInputException("File doesn't contain enough" +  "elements for matrix of size" + rows + "x" + cols);  }  }  }  scanner.close();  } catch (FileNotFoundException e) {  throw new FileInputException("File not found: " + e.getMessage());  }  return inMatrix;  }   public static void printMatrix(Matrix matrixObject) {  if (matrixObject == null) {  System.*out*.println("Matrix is empty or invalid");  return;  }   StringBuilder sb = new StringBuilder();  int[][] matrix = matrixObject.getMatrix();   for (int[] row : matrix) {  for (int element : row) {  String elString = String.*format*("%3d", element);  sb.append(elString).append("\t");  }  sb.append("\n");  }  System.*out*.println(sb.toString());  }   public static void printLabels(int[] array) {  if (array == null) {  System.*out*.println("Array is empty or invalid");  return;  }   StringBuilder sb = new StringBuilder();  for (int i = 0; i < array.length; i++) {  String elString = "L(x" + (i + 1) + ") = " + array[i];  sb.append(elString).append('\n');  }  System.*out*.println(sb.toString());  } } |

Таблица 4 – Класс Путь

|  |
| --- |
| package application;  import java.util.Arrays; import java.util.Objects;  public class Path {  final int[] labels;  final String pathString;   public Path(int[] labels, String pathString) {  this.labels = labels;  this.pathString = pathString;  }   public int[] getLabels() {  return labels;  }   public String getPathString() {  return pathString;  }   @Override  public boolean equals(Object object) {  if (this == object) return true;  if (object == null || getClass() != object.getClass()) return false;  Path path = (Path) object;  return Arrays.*equals*(labels, path.labels) && Objects.*equals*(pathString, path.pathString);  }   @Override  public int hashCode() {  int result = Objects.*hash*(pathString);  result = 31 \* result + Arrays.*hashCode*(labels);  return result;  }   @Override  public String toString() {  StringBuilder sb = new StringBuilder();   sb.append("Пометки вершин:\n");  for (int i = 0; i < labels.length; i++) {  String elString = "L(x" + (i + 1) + ") = " + labels[i];  sb.append(elString).append('\n');  }  int lastElement = labels.length;  String elString = "\nL(M) = " + labels[lastElement-1];  sb.append(elString).append('\n');   sb.append("Кратчайший путь: ").append("M(x1, x" + labels.length + ") = {").append(pathString).append("}");   return sb.toString();  } } |

Таблица 5 – Класс с методами обработки матрицы

|  |
| --- |
| package application;  import java.util.Arrays;  public class PathFinder {  public static Path findShortestPath(Matrix matrix) {  Matrix clonedMatrix = null;   try {  clonedMatrix = (Matrix) matrix.clone();  } catch (CloneNotSupportedException e) {  e.printStackTrace();  }   // Расстановка пометок вершин  int[] labels = *findLabels*(clonedMatrix);   // Поиск кратчайшего пути  String pathString = *findWay*(labels, clonedMatrix);   return new Path(labels, pathString);  }   public static int[] findLabels(Matrix matrix) {  Matrix clonedMatrix = null;   try {  clonedMatrix = (Matrix) matrix.clone();  } catch (CloneNotSupportedException e) {  e.printStackTrace();  }   // Размерность матрицы  final int SIZE = clonedMatrix.getH();   // Создание массива пометок  int[] labels = new int[SIZE];  Arrays.*fill*(labels,1, SIZE, Integer.*MAX\_VALUE*);   // Расстановка пометок  for (int j = 0; j < SIZE; j++) {  for (int i = 0; i < SIZE; i++) {  int currentElement = clonedMatrix.getElement(i, j);  if (currentElement != -1) {  labels[j] = Math.*min*(labels[j], labels[i] + currentElement);  }  }  }   return labels;  }   public static String findWay(int[] labels, Matrix matrix)  throws IllegalArgumentException {  Matrix clonedMatrix = null;   try {  clonedMatrix = (Matrix) matrix.clone();  } catch (CloneNotSupportedException e) {  e.printStackTrace();  }   // Размерность матрицы  final int SIZE = clonedMatrix.getH();   if (labels == null || labels.length != SIZE) {  throw new IllegalArgumentException("Label array is invalid");  }   // Поиск пути  StringBuilder sb = new StringBuilder();  sb.append(SIZE).append("x");  for (int j = SIZE - 1; j >= 0; j--) {  for (int i = SIZE - 1; i >= 0; i--) {  int currentElement = clonedMatrix.getElement(i, j);  if (currentElement != -1 && labels[j] == labels[i] + currentElement) {  j = i; // переход на столбец смежной вершины  sb.append(';').append(j + 1).append('x');  }  }  }  return sb.reverse().toString();  } } |

**Пример работы программы:**

Для проверки работы разработанной программы возьмем следующие исходные данные (рисунок 4):

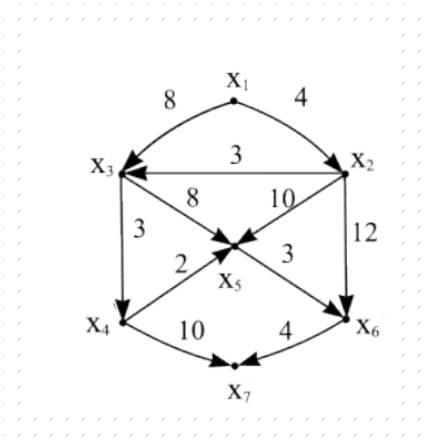


Рисунок 4 – Пример графа

Построим для данного графа соответствующую матрицу смежности (рисунок 5):

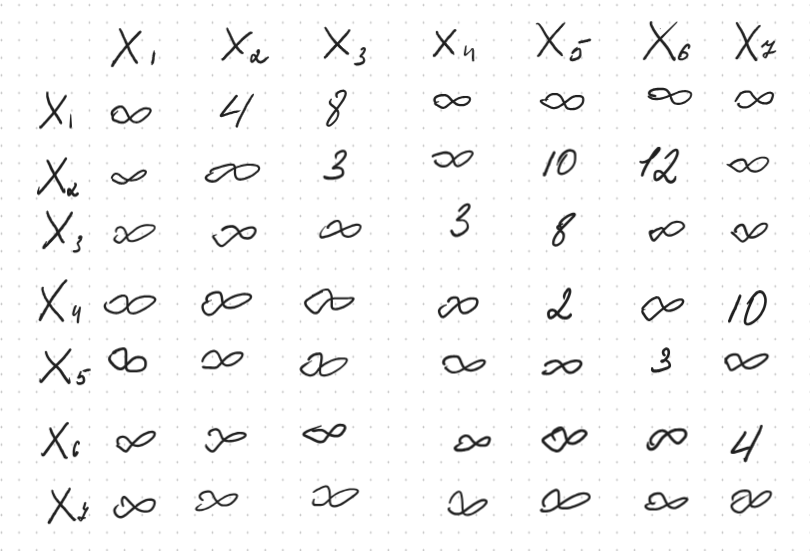


Рисунок 5 – Матрица смежности примера графа

Результат выполнения разработанной программы (рисунок 6):

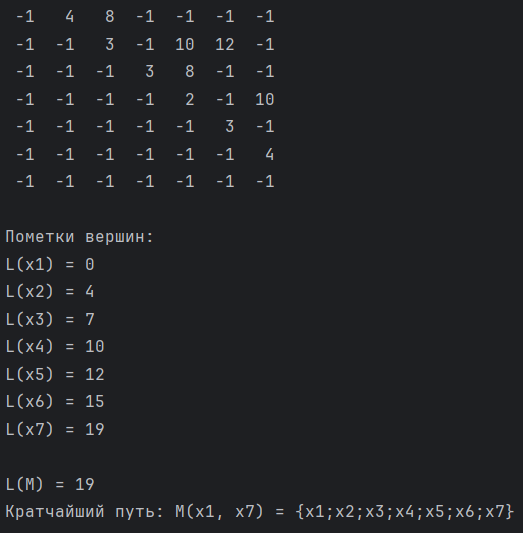


Рисунок 6 – Результат выполнения программы

Выполним проверку на следующем примере (рисунок 7-8):

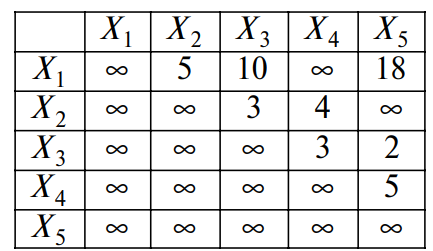


Рисунок 7 – Следующий пример

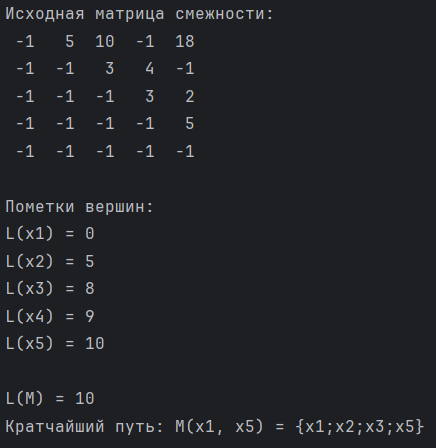


Рисунок 8 – Решение следующего примера

**Вывод**

Были изучены алгоритмы поиска кратчайших путей на графах на примере метода динамического программирования.