Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**Отчет**

По лабораторной работе №6

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Поиск расстояний в графе»

Выполнил студент гр.20ВВ3

Мирясов Н.А.

Проверили:

Митрохин М.А.

Юрова О.В.

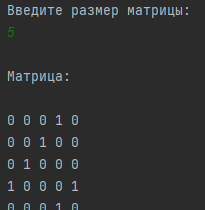
Пенза, 2021

**Цель:** изучить алгоритм поиска расстояний в графе.

**Задание 1:**

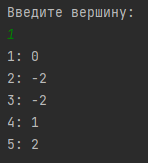
1. Сгенерировали (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Вывели матрицу на экран.



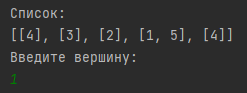
2. Для сгенерированного графа осуществили процедуру поиска

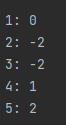
расстояний, реализованную в соответствии с приведенным выше описанием.



3.\* Реализовали процедуру поиска расстояний для графа, представленного

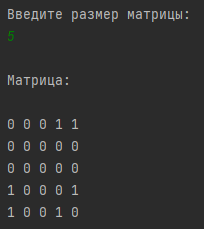
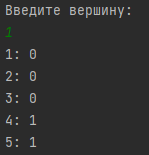
списками смежности.





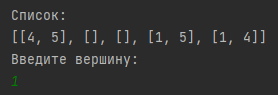
**Задание 2:**

1. Реализовали процедуру поиска расстояний на основе обхода в глубину.

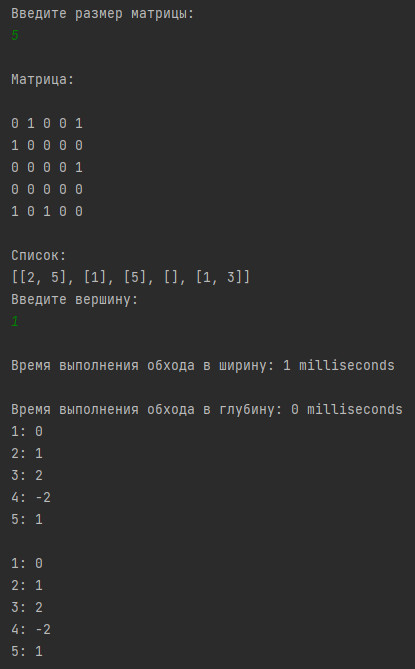
2. Реализовали процедуру поиска расстояний на основе обхода в глубину

для графа, представленного списками смежности.

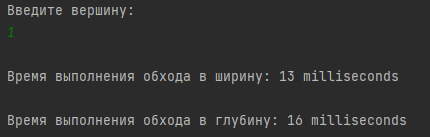
 

3. Оценили время работы реализаций алгоритмов поиска расстояний на

основе обхода в глубину и обхода в ширину для графов разных порядков.



Размер графа: 1000



**Листинг задания 1 - 2:**

import java.util.\*;  
  
public class mission1 {  
 public static void main(String[] args) {  
 System.*out*.println("Введите размер матрицы:");  
 Scanner in = new Scanner(System.*in*);  
 int n = in.nextInt();  
 int[][] matrix = new int[n][n];  
 ArrayList<LinkedList<Integer>> adjLists = new ArrayList<>();  
  
 *rand*(n, matrix);  
 *addEdge*(n, matrix, adjLists);  
 *output*(n, matrix, adjLists);  
   
 Queue<Integer> queue = new LinkedList<>();  
 Queue<Integer> queue1 = new LinkedList<>();  
 int[] vis = new int[n];  
 int[] vis1 = new int[n];  
 System.*out*.println("Введите вершину:");  
 Scanner in1 = new Scanner(System.*in*);  
 int v = in1.nextInt() - 1;  
  
 for (int i = 0; i < n; i ++){  
 vis[i] = -1;  
 vis1[i] = -1;  
 }  
  
  
 *BFSD*(n, v, vis, queue, matrix);  
 *BFSD1*(n, v, vis1, queue1, adjLists);  
  
 int count = 0;  
 int[] vis2 = new int[n];  
 int[] res = new int[n];  
 long time1;  
 long time2;  
 time1 = System.*currentTimeMillis*();  
 *DFS*(n, count, v, vis2, res, matrix);  
 time2 = System.*currentTimeMillis*();  
 System.*out*.println();  
 System.*out*.println("Время выполнения обхода в глубину: " + (time2 - time1) + " milliseconds");  
  
 for (int i = 0; i < n; i++) {  
 System.*out*.println(i + 1 + ": " + (vis[i] - 1));  
 }  
 System.*out*.println();  
 for (int i = 0; i < n; i++) {  
 System.*out*.println(i + 1 + ": " + (vis1[i] - 1));  
 }  
 }  
  
 public static void rand(int n, int[][] matrix) {  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 matrix[i][j] = (int) ((Math.*random*() \* 2) + 0);  
 matrix[j][i] = matrix[i][j];  
  
 if (i == j) {  
 matrix[i][j] = 0;  
 }  
 }  
 }  
 }  
  
 public static void addEdge(int n, int[][] matrix, ArrayList<LinkedList<Integer>> adjLists) {  
 for (int i = 0; i < n; i++)  
 adjLists.add(new LinkedList<>());  
  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if (matrix[i][j] == 1)  
 adjLists.get(i).add(j + 1);  
 }  
 }  
  
 }  
  
 public static void output(int n, int[][] matrix, ArrayList<LinkedList<Integer>> adjLists) {  
 System.*out*.println("\nМатрица:\n");  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
  
 System.*out*.println("\nСписок:");  
 System.*out*.println(adjLists);  
 }  
  
 public static void BFSD(int n, int v, int[] vis, Queue<Integer> queue, int[][] matrix){  
 long time1;  
 long time2;  
 time1 = System.*currentTimeMillis*();  
 vis[v] = 1;  
 queue.offer(v);  
 while (queue.peek() != null){  
 int s = queue.poll();  
 for (int i = 0; i < n; i++){  
 if (matrix[s][i] != 0 & vis[i] == -1){  
 queue.offer(i);  
 vis[i] = 1 + vis[s];  
 }  
 }  
 }  
 time2 = System.*currentTimeMillis*();  
 System.*out*.println();  
 System.*out*.println("Время выполнения обхода в ширину: " + (time2 - time1) + " milliseconds");  
 }  
  
 public static void BFSD1(int n, int v, int[] vis1, Queue<Integer> queue1, ArrayList<LinkedList<Integer>> adjLists){  
 vis1[v] = 1;  
 queue1.offer(v);  
 while (queue1.peek() != null){  
 int s = queue1.poll();  
 for (int i = 0; i < n; i++){  
 if (vis1[i] == -1 & adjLists.get(s).contains(i + 1)){  
 vis1[i] = 1 + vis1[s];  
 queue1.offer(i);  
 }  
 }  
 }  
 }  
  
 public static void DFS(int n, int count, int v, int[] vis, int[] res, int[][] matrix){  
  
 count += 1;  
 vis[v] = 1;  
 for (int i = 0; i < n; i++){  
 if (matrix[v][i] == 1 & vis[i] == 0)  
 *DFS*(n, count, i, vis, res, matrix);  
 if (matrix[v][i] == 1)  
 res[i] = count;  
 }  
 count -= 1;  
 if (count > res[n - 1])  
 return;  
 res[v] = count;  
  
 }  
}

**Вывод:** изучили алгоритм поиска расстояний в графе с помощью обхода в глубину и в ширину, сравнили эффективность обоих методов.