Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**Отчет**

По лабораторной работе №5

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Обход графа в ширину»

Выполнил студенты гр.20ВВ3

Мирясов Н.А.

Проверили:

Митрохин М.А.

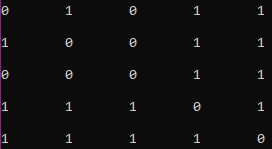
Юрова О.В.

Пенза, 2021

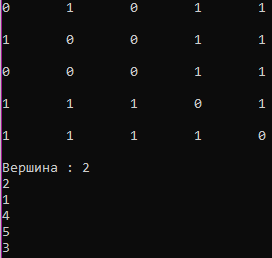
**Цель:** изучить обход графа в ширину.

**Задание 1:**

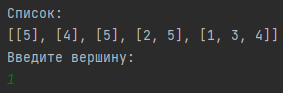
1. Сгенерировали (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Вывели матрицу на экран:



1. Для сгенерированного графа осуществили процедуру обхода в ширину:

****

**3.**\* Реализовали процедуру обхода в ширину для графа, представленного списками смежности:





**Задание 2**

1. Для матричной формы представления графов реализовали алгоритм

обхода в ширину с использованием очереди, построенной на основе

структуры данных «список», самостоятельно созданной в лабораторной

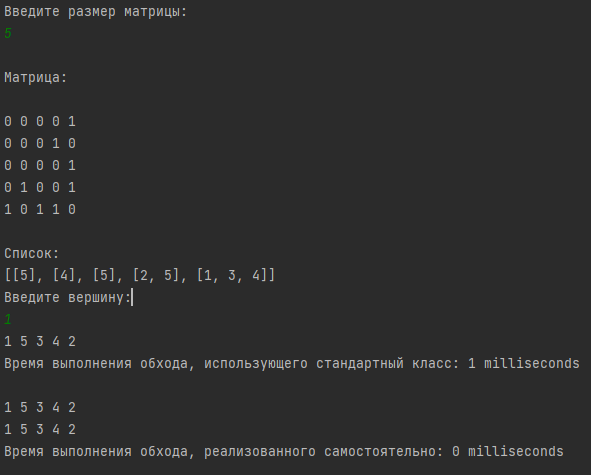
работе № 3.

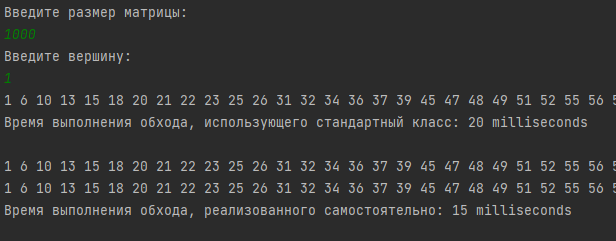


2. Оценили время работы двух реализаций алгоритмов обхода в ширину

(использующего стандартный класс queue и использующего очередь,

реализованную самостоятельно) для графов разных порядков.





**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<>();  
 boolean[] vis = new boolean[n];  
 boolean[] vis1 = new boolean[n];  
 boolean[] vis2 = new boolean[n];  
 System.*out*.println("Введите вершину:");  
 Scanner in1 = new Scanner(System.*in*);  
 int v = in1.nextInt() - 1;  
  
 *BFS*(n, v, vis, queue, matrix);  
 System.*out*.println();  
 *BFS1*(n, v, vis1, queue1, adjLists);  
  
 Queue1 myQueue = new Queue1(n);  
 System.*out*.println();  
 *BFS2*(n, v, vis2, matrix, myQueue);  
 }  
  
 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 BFS(int n, int v, boolean[] vis, Queue<Integer> queue, int[][] matrix){  
 long time1;  
 long time2;  
 time1 = System.*currentTimeMillis*();  
 vis[v] = true;  
 queue.offer(v);  
 while (queue.peek() != null){  
 int s = queue.poll();  
 System.*out*.print(s + 1 + " ");  
 for (int i = 0; i < n; i++){  
 if (matrix[s][i] == 1 & !vis[i]){  
 vis[i] = true;  
 queue.offer(i);  
 }  
 }  
 }  
 time2 = System.*currentTimeMillis*();  
 System.*out*.println();  
 System.*out*.println("Время выполнения обхода, использующего стандартный класс: " + (time2 - time1) + " milliseconds");  
 }  
  
 public static void BFS1(int n, int v, boolean[] vis1, Queue<Integer> queue1, ArrayList<LinkedList<Integer>> adjLists){  
 vis1[v] = true;  
 queue1.offer(v);  
 while (queue1.peek() != null){  
 int s = queue1.poll();  
 System.*out*.print(s + 1 + " ");  
 for (int i = 0; i < n; i++){  
 if (!vis1[i] & adjLists.get(s).contains(i + 1)){  
 vis1[i] = true;  
 queue1.offer(i);  
 }  
 }  
 }  
 }  
  
 public static void BFS2(int n, int v, boolean[] vis2, int[][] matrix, Queue1 myQueue){  
 long time1;  
 long time2;  
 time1 = System.*currentTimeMillis*();  
 vis2[v] = true;  
 myQueue.insert(v);  
 while (!myQueue.isEmpty()){  
 int s = myQueue.remove();  
 System.*out*.print(s + 1 + " ");  
 for (int i = 0; i < n; i++){  
 if (matrix[s][i] == 1 & !vis2[i]){  
 vis2[i] = true;  
 myQueue.insert(i);  
 }  
 }  
 }  
 time2 = System.*currentTimeMillis*();  
 System.*out*.println();  
 System.*out*.println("Время выполнения обхода, реализованного самостоятельно: " + (time2 - time1) + " milliseconds");  
 }  
}  
  
class Queue1 {  
 private int[] queue;  
 private int maxSize;  
 private int nElem;  
 private int front;  
 private int rear;  
  
 public Queue1(int maxSize) {  
 this.maxSize = maxSize;  
 queue = new int[maxSize];  
 rear = -1;  
 front = 0;  
 nElem = 0;  
 }  
  
 public void insert(int elem) {  
 if (rear == maxSize - 1) {  
 rear = -1;  
 }  
  
 queue[++rear] = elem;  
 nElem++;  
 }  
  
 public int remove() {  
 int temp = queue[front++];  
  
 if (front == maxSize) {  
 front = 0;  
 }  
 nElem--;  
 return temp;  
 }  
  
 public boolean isEmpty() {  
 return (nElem == 0);  
 }  
}

**Вывод:** изучили процедуру обхода графа в ширину, с помощью встроенных методов, и написанных самостоятельно и также применили его на списки смежности. Выяснили, что стандартный класс показывает меньшую производительность, чем написанный вручную.