FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

UNIVERSIDADE NOVA DE LISBOA
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT

Engenharia Informática

Análise e Desenho de Algoritmos

Trabalho Prático 3

A tale never loses in the telling

Trabalho realizado por:

Rafael Gameiro nº50677

Rui Santos nº50833

Índice

Complexidade Espacial

Vetor (*interfDegree*):	$\Theta(\|V\|)$
Vetor de Listas Ligadas (*Friendships*):	$\Theta(\|V\| + \|A\|)$
Vetor (*selected*):	$\Theta(\|V\|)$
Vetor (*noise*):	$\Theta(\|V\|)$
PriorityQueue (*connected*):	$O(\|V\|)$
Total:	$\Theta(\|V\| + \|A\|)$

Complexidade Temporal

Classe Main:

Preenchimento do *interfDegree*	$\Theta(\|V\|)$
Preenchimento do *friendships*	$\Theta(\|A\|)$

Classe BrokenPhone:

Criação do vetor *selected*	$\Theta(\|1\|)$
Criação do vetor *noise*	$\Theta(\|1\|)$
Criação da priorityQueue *connected*	$\Theta(\|1\|)$
Ciclo (executado |V| vezes)	
Inicialização do vetor *noise*	$\Theta(\|V\|)$
Ciclo (executado |f| vezes)	
Definição do noise dos sucessores do nó *source* (*s*)	$\Theta(\|f\|)$
Ciclo (executado no máximo |V| - 1 vezes)	
Remoção do mínimo da priorityQueue *connected*	$\Theta(\|1\|)$

<u>método exploreNodes():</u>

Ciclo (executado no máximo |A| - 1 vezes)	
Obtenção dos arcos incidentes do nó	$O(\|A\|)$
Inserção de um novo Pair na priorityQueue *connected*	$O(\|V\| * \log \|V\|)$

sendo f = $friendships[s].length$

Total: $O(|V| * \log |V|)$

Conclusão

O nosso algoritmo executa a expressão para o cálculo do valor mínimo do noise tal como apresentada no enunciado.
O ponto fraco do nosso algoritmo é este utilizar a classe *BigInteger* para o cálculo do noise. Esta classe substitui as operações normais entre números, permitindo obter valores bastante elevados que não caberiam num inteiro, ou até mesmo num long. Por esse motivo o custo das operações não é constante ($\Theta(1)$).

Anexos

```java
import java.io.BufferedReader;

/**
 * @author Rafael Gameiro (50677)
 * @author Rui Santos (50833)
 *
 */
public class Main {

    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws IOException {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String input = br.readLine();
        String[] inputArray = input.split(" ");
        int P = Integer.parseInt(inputArray[0]);
        int s = Integer.parseInt(inputArray[1]);
        int t = Integer.parseInt(inputArray[2]);
        int N = Integer.parseInt(inputArray[3]);

        int[] interfDegree = new int[P];
        List<Integer>[] friendships = new List[P];

        for(int i = 0 ; i < P ; i++) {
            int degree = Integer.parseInt(br.readLine());
            interfDegree[i] = degree;
            friendships[i] = new LinkedList<>();
        }

        int nFriendships = Integer.parseInt(br.readLine());
        for(int i = 0; i < nFriendships; i++) {
            input = br.readLine();
            inputArray = input.split(" ");
            int x = Integer.parseInt(inputArray[0]);
            int y = Integer.parseInt(inputArray[1]);
            friendships[x].add(y);
            friendships[y].add(x);
        }

        BrokenPhone bp = new BrokenPhone(s,t,N,interfDegree,friendships);
        BigInteger result = bp.dijkstra();
        System.out.println(result);

    }
}
```

```java
import java.math.BigInteger;

/**
 * @author Rafael Gameiro (50677)
 * @author Rui Santos (50833)
 *
 */
public class BrokenPhone {

    public static final BigInteger DEFAULT_VAL = new BigInteger("-1");
    public static final BigInteger MODULE_POW = new BigInteger("" + (long) Math.pow(2, 45));

    int s, t;
    BigInteger N;
    int[] interfDegree;
    List<Integer>[] friendships;

    public BrokenPhone(int s, int t, int N, int[] interfDegree, List<Integer>[] friendships) {
        this.s = s;
        this.t = t;
        this.N = new BigInteger("" + N);
        this.interfDegree = interfDegree;
        this.friendships = friendships;
    }
```

```java
    /**
     * Computes the Dijkstra algorithm
     * Firstly, initialises the variables (selected, noise and connected)
     * and after that, it will set all the position of the
     * noise array with the constant DEFAULT_VAL, except for the source's successors
     * The algorithm starts by taking the Pair(node, noise) object with the less noise
     * and checks if the object is the target person
     * If it is not, the it will explore it's successors
     * After that returns the value of the noise
     *
     * @return the noise of the person target after applying the module of 2^45
     */
    public BigInteger dijkstra() {

        boolean[] selected = new boolean[friendships.length];
        BigInteger[] noise = new BigInteger[friendships.length];

        Queue<Pair> connected = new PriorityQueue<>();

        for (int i = 0; i < friendships.length; i++)
            noise[i] = DEFAULT_VAL;

        for (int node : friendships[s]) {
            BigInteger value = new BigInteger("0");
            noise[node] = value;
            connected.add(new Pair(node, value));
        }

        do {
            Pair pair = connected.remove();
            selected[pair.getKey()] = true;
            if (pair.getKey() == t)
                break;
            exploreNodes(pair, selected, noise, connected);

        } while (!connected.isEmpty());

        return noise[t].mod(MODULE_POW);
    }
```

```java
    /**
     * The method will run through all the node's successors and for each successor,
     * it will check if that successor was already explored
     * If a successor is the target, it will set the newNoise value,
     * as the value in the noise array, node position
     * Checks if it's the first time computing the successor's noise or the
     * newNoise value is less than the previous computed noise of the successor and
     * if it is, replaces the value in the noise array and inserts the newNoise
     * value in the priority queue
     *
     * @param pair      current Pair object
     * @param selected  array with the info of the explored nodes
     * @param noise     array with the info of noise for each node
     * @param connected priority queue with the nodes already explored
     */
    private void exploreNodes(Pair pair, boolean[] selected, BigInteger[] noise, Queue<Pair> connected) {
        BigInteger newNoise = null;
        for (int node : friendships[pair.getKey()]) {
            if (!selected[node]) {
                newNoise = new BigInteger("" + interfDegree[pair.getKey()]);
                newNoise = newNoise.add(noise[pair.getKey()].multiply(N));

                if (noise[node].compareTo(DEFAULT_VAL) == 0 || newNoise.compareTo(noise[node]) == -1) {
                    noise[node] = newNoise;
                    connected.add(new Pair(node, newNoise));
                }
            }
        }
    }

}
```

```java
import java.math.BigInteger;

/**
 * @author Rafael Gameiro (50677)
 * @author Rui Santos (50833)
 *
 */
public class Pair implements Comparable<Pair>{

    private int key;
    private BigInteger value;

    public Pair(int key, BigInteger value) {
        this.key = key;
        this.value = value;
    }

    /**
     *
     * @return the number that corresponds to a person
     */
    public int getKey() {
        return key;
    }

    /**
     *
     * @return the current value of noise reaching this person
     */
    public BigInteger getValue() {
        return value;
    }

    @Override
    public int compareTo(Pair pair) {
        return this.getValue().compareTo(pair.getValue());
    }

}
```