

Departamento de Informática

Análise e Desenho de Algoritmos

2015/2016

**Relatório**  
Trabalho Prático 2

Crimes in Whitechapel

Docente:

Margarida Mamede

Turno Pratico p.5

Rodolfo Ferreira nº 41654

João Elvas nº 41934

Apresentação do Problema

O problema consiste em saber dadas duas posições N, que correspondem a localizações e dada a estimativa da distancia percorrida dessas duas posições N quais as novos possíveis localizações do “Jack the Ripper”.

Sabe-se que N esta contido no intervalo [1:10 000]

E a distância percorrida está contida no intervalo [1;N]

Resolução do Problema

* Formalização do problema

Para resolver o problema apercebemo-nos que o mesmo poderia ser traduzido para linguagem informática, usando grafos e outras estruturas de dados para facilitar o cálculo da resposta.

* Conceção do algoritmo

Para a resolução deste problema usámos uma pesquisa de árvore em largura, usando o algoritmo Breadth-First Search Traversal, que foi adaptado dos slides da cadeira para otimizar o problema, pois este é bastante específico.

Em resumo, estamos a fazer uma pesquisa em largura de um determinado nó com n passos, sendo n o valor da distância recebido por input.

Implementação do Algoritmo

* Estrutura do programa

O programa esta dividido em duas classes, a class Main e a class Whitechapel .

A class Main, lê o input, converte-o e passam para a class WhiteChapel, onde o problema é resolvido.

A class WhiteChapel recebe o input, cria todas as estruturas de dados necessárias para a resolução do problema, aplica o algoritmo de Breadth-First Search Traversal e, no fim, retorna a resposta.

* Estruturas de dados

As estruturas de dados utilizadas foram:

Uma matriz, com a dimensão de todos os nós do problema por todos os seus sucessores;

Um vetor de booleanos, que guarda em cada índice os nós já explorados pelo algoritmo, este vetor tem o tamanho do número de localizações;

E, por fim, um vetor de inteiros, que tem a dimensão da primeira iteração do algoritmo. Isto corresponde a uma otimização feita por nós, pois em vez de estarmos a criar uma estrutura de dados para cada iteração do algoritmo e no fim compararmos todas, estamos a guardar apenas numa e fazemos a intersecção de respostas em todas as iterações, isto é possível pois ao intersectar duas respostas o resultado é sempre igual ou inferior ao primeiro vetor comparado.

Análise do Algoritmo

* Complexidade Espacial

|  |
| --- |
| N – numero de nós (localizações) |
| Array locations θ(N \* N) |
| Array found θ(N) |
| Array equals θ(N) |

* Complexidade Temporal

Main

|  |
| --- |
| N – número de nós (localizações)  NC – número e pistas |
| 1º ciclo θ (N) |
| 1º ciclo θ (NC) |
| = θ (N) + θ (NC) |

addSucessor(**int** node , String[] sucessores )

|  |
| --- |
| NS – números e sucessores |
| ciclo θ (NS) |

bfsExploreFirst(**int** node , **int** numberOfTimes)

ou

bfsExplore(**int** node , **int** numberOfTimes)

|  |
| --- |
| D – Distancia  Q – nós na Queue |
| 1º ciclo θ (D) |
| 2º ciclo θ (Q) |
| = θ (D) + θ (Q) |

calcNextQueue(Queue<Integer> waiting)

|  |
| --- |
| Q – nós na Queue  SX – sucessores do nó X |
| 1º ciclo θ (Q) |
| 2º ciclo θ (SX) |
| = θ (Q \* SX) |

comparation( **int**[] x , **int**[] y )

|  |
| --- |
| TX = Tamanho vetor X  TY = tamanho vetor Y |
| 1º ciclo θ (TX) |
| 2º ciclo θ (TY) |
| = θ (TX \* TY) |

String getAnswer()

|  |
| --- |
| NI = números no vetore de números iguais |
| 1º ciclo θ (NI) |
| Arrays.sort (NI\*log(NI)) |
| 3º ciclo θ (NI) |
| = θ (NI) + θ (NI\*log(NI))) + θ (NI) |

Conclusões

O nosso trabalho foi realizado com bastante deliberação, optámos por usar tipos de variáveis mais primitivos para facilitar consultas e comparações.

Após a sua conclusão achamos que poderiam haver otimizações, nomeadamente, achando uma maneira mais eficiente de fazer a comparação entre dois vetores.

Contudo, através da presença nas aulas teóricas, foi-nos possível concluir o nosso trabalho tendo alcançado os objetivos pedidos.

Anexos

Class Main

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

/\*\*

\* @author Rodolfo Ferreira n 41654

\* @author Joao Elvas n 41934

\*/

public class Main {

public static void main(String[] args) throws IOException {

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

int numberOfLocations = Integer.parseInt(in.readLine());

Whitechapel whiteChapel = new Whitechapel(numberOfLocations);

for(int n = 1 ; n <= numberOfLocations ; n++) {

String[] sucessores = in.readLine().split(" ");

whiteChapel.addSucessor(n, sucessores);

}

int numberOfClues = Integer.parseInt(in.readLine());

int node;

int distance;

String[] tmp = in.readLine().split(" ");

node = Integer.parseInt(tmp[0]);

distance = Integer.parseInt(tmp[1]);

whiteChapel.bfsExploreFirst(node,distance);

if(numberOfClues == 1){

System.out.println(whiteChapel.getAnswer());

}

else{

for(int x = 1 ; x < numberOfClues ; x++){

tmp = in.readLine().split(" ");

node = Integer.parseInt(tmp[0]);

distance = Integer.parseInt(tmp[1]);

whiteChapel.bfsExplore(node, distance);

}

in.close();

System.out.println(whiteChapel.getAnswer());

}

}

}

Class Whitechapel

import java.util.Arrays;

import java.util.LinkedList;

import java.util.Queue;

/\*\*

\* Class que se destina a representar o Whitechapel

\*/

/\*\*

\* @author Rodolfo Ferreira n 41654

\* @author Joao Elvas n 41934

\*/

public class Whitechapel {

private static final String NO\_SOLUTION= "NO SOLUTION";

int locations [][]; // estrutura que guarda os sussesores no indice de cada no

int numNodes; // numero de nos

boolean []found ; // vector de boleanos que no indice do no diz se ja foi explorado ou nao

int [] equals; // vector que tem todos os numeros iguais depois das iteracoes

int equalNumbers ; // todos os numeros iguais

public Whitechapel(int numLocations ){

locations = new int [numLocations + 1][];

locations[0] = null;

found = new boolean [numLocations + 1];

equals = null;

numNodes = numLocations;

equalNumbers = 0;

}

/\*\*

\* coloca na estrutura de dados no indice = node todos os seus sucessores

\* @param node - indice do no

\* @param sucessores - todos os sucessores desse no

\*/

public void addSucessor(int node , String[] sucessores ) {

int [] tmp = new int [sucessores.length-1 ];

int i = 0;

while(!sucessores[i].equals("0")){

tmp[i]=Integer.parseInt(sucessores[i]);

i++;

}

locations[node]= tmp;

}

/\*\*

\* Preenche a estrutura equals na primeira iteracao

\* @param node indice do no a explorar

\* @param numberOfTimes

\*/

public void bfsExploreFirst(int node , int numberOfTimes){

Queue <Integer> waiting = new LinkedList<Integer>();

waiting.add(node);

found[node] = true;

for(int i = 0 ; i< numberOfTimes ; i++){

waiting = calcNextQueue(waiting);

}

int[] tmp = new int[waiting.size()];

int d = 0;

while(!waiting.isEmpty()){

tmp[d]=waiting.poll() ;

d++;

}

equals = tmp;

found = new boolean [numNodes + 1];

equalNumbers = equals.length;

}

/\*\*

\* Preenche a estrutura equals depois de numberOfTimes iteracoes

\* @param node indice do no a explorar

\* @param numberOfTimes - numero de vezes que se vai esplorar todos os nos da pilha

\*/

public void bfsExplore(int node , int numberOfTimes){

Queue <Integer> waiting = new LinkedList<Integer>();

waiting.add(node);

found[node] = true;

for(int i = 0 ; i< numberOfTimes ; i++){

waiting = calcNextQueue(waiting);

}

int[] tmp = new int[waiting.size()];

int d = 0;

while(!waiting.isEmpty()){

tmp[d]=waiting.poll();

d++;

}

comparation(equals,tmp);

found = new boolean [numNodes + 1];

}

/\*\*

\* calcula a prosima queue

\* @param waiting - queue da iteracao anterior

\* @return - prosima queue dessa iteracao

\*/

private Queue<Integer> calcNextQueue(Queue<Integer> waiting) {

Queue <Integer> processed = new LinkedList<Integer>();

do{

int nodeX = waiting.poll();

for(Integer i : locations[nodeX]) {

if(!found[i]) {

processed.add(i);

found[i] = true;

}

}

} while(!waiting.isEmpty() );

return processed;

}

/\*\*

\* Compara dois vectores de inteiros

\* @param x - vector a comparar

\* @param y - vector a comprar

\*/

private void comparation( int[] x , int[] y ){

int[] tmpVec = new int[x.length];

equalNumbers = 0;

for(int i = 0 ; i<x.length ; i++){

for(int z = 0 ; z < y.length ; z++){

if( x[i] != 0 && x[i]==y[z] ){

tmpVec[equalNumbers]=x[i];

equalNumbers++;

}

}

}

equals = tmpVec;

}

/\*\*

\* retorna uma String que responde ao problema

\* @return - String formatada que responde ao problema

\*/

public String getAnswer(){

String answer = new String() ;

if(equalNumbers == 0)

answer = NO\_SOLUTION;

else if(equalNumbers == 1 ){

answer = String.valueOf(equals[0]);}

else {

int[] integerArray = new int[equalNumbers];

for(int p=0 ; p< equalNumbers ; p++){

integerArray[p] = equals[p];

}

equals = integerArray;

Arrays.sort(equals);

answer = String.valueOf(equals[0]);

int i = 1;

while(i < equalNumbers ){

answer = answer.concat(" "+ equals[i]);

i++;

if(i== equals.length)

break;

}

}

return answer;

}

}