Nova School of Science & Technology

Universidade Nova de Lisboa

Lost

Relatório de Análise e Desenho de Algoritmos 2020/2021(MIEI)

3º Trabalho

Alunos : José Murta (55226) e Diogo Rodrigues (56153)

Curso: Mestrado Integrado em Engenharia Informática

Turno: Po5

Docente prático: Margarida Mamede

Complexidade Temporal

Método calculate(isJohn):

1. isJohn == true
   - Criação do vetor length $\Theta(1)$
   - Inicialização do length $\Theta(|V|)$
   - $\leq |V|$ execuções de updateLengths $O(|V| \times |X|)$
     - Cada execução: $\Theta(|X|)$
2. isJohn == false
   - Criação do vetor length $\Theta(1)$
   - Inicialização do length $\Theta(|V|)$
   - $\leq |V|$ execuções de updateLengths $O(|V| \times |Y|)$
     - Cada execução: $\Theta(|Y|)$

Total do método: $O(|V| \times |Y|)$

Método createEdgesJohn:

- Percorrer todos os nós * $\Theta(|V|)$
  - Percorrer as 4 adjacências
    - Verificações e adições à lista de arcos $\Theta(1)$

*(exceto os nós da primeira e ultima linhas que são sempre do tipo "W", que de qualquer forma iriam ser ignorados pela nossa implementação)

Total do método = $\Theta(4 \times |V|) = \Theta(|V|)$

Método createEdgesKate:

- Percorrer todos os nós $\Theta(|V|)$
  - Percorrer as 4 adjacências
    - Verificações e adições à lista de arcos $\Theta(1)$

Total do método = $\Theta(4 \times |V|) = \Theta(|V|)$

Total: $O(|V| \times |Y|)$

Legenda:

- $|V| = |R \times C|$ = número de vértices do grafo
- $|X|$ = número de arcos do grafo do John
- $|Y|$ = número de arcos do grafo da Kate

Complexidade Espacial

Matriz com o mapa da ilha (característica dos nós) $\Theta(|R \times C|)$

- Matriz com as características/tipo das células da ilha

Vetor de Magical Wheels $\Theta(|M|)$

- Vetor com 3 posições para cada magical wheel (Índice da linha, índice da coluna, e tempo)

Vetor length do algoritmo de Bellman Ford $\Theta(|R \times C|)$

- Vetor com os tempos mínimos da origem para todos os nós do grafo

Lista de arcos do grafo do John $\Theta(|X|)$

- Vetor com 3 posições (tail, head e tempo) para cada arco válido para o caminho do John, sendo X o número desses arcos.

Lista de arcos do grafo da Kate $\Theta(|Y|)$

- Vetor com 3 posições (tail, head e tempo) para cada arco válido para o caminho da Kate, sendo Y o número desses arcos.

Devido à forma como implementámos o problema enunciado, bem como as características do mesmo, o número de arcos do grafo da Kate é sempre superior ao número de arcos do John e do número de posições da matriz com o mapa da ilha (nós do grafo).

Total: $\boldsymbol{\Theta(|Y|)}$

Conclusões

Ao longo da implementação deste projeto, pensámos em diversas alternativas tendo chegado à conclusão que esta, baseada na sugestão de resolução indicada pelos docentes, seria a mais vantajosa, tanto em complexidade espacial como em temporal.

Inicialmente, explorámos uma outra alternativa mas mais tarde concluímos que não era vantajosa para a resolução adequada deste problema. Esta alternativa consistia em adicionar todos os arcos à lista (que neste caso seria única), independentemente das características da head e da tail de cada arco, e posteriormente, dentro do próprio algoritmo de Bellman Ford, filtrar então os arcos consoante as suas características. Tal seria um abordagem pior, pois existiriam arcos que iriam ser constantemente ignorados para o cálculo dos tempos mínimos, contrariamente à nossa solução atual que apenas contempla os arcos necessários para esse cálculo.

Outro aspeto que podíamos explorar para a nossa implementação, seria utilizar o algoritmo de Dijkstra para calcular o tempo mínimo para o caminho da Kate, visto que o seu grafo nunca terá arestas de peso negativo.

Finalmente, os nossos métodos de preenchimento da lista de arcos do grafo do John e da Kate, têm algum código com o mesmo comportamento que resulta em pequenas porções de código repetido. Isto poderia ser evitado, utilizando só um método, mas acabaria por tornar esse método menos direto e claro.

```java
/**
 * @author Jose Murta 55226 && Diogo Rodrigues 56153
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {

    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader input = new BufferedReader ( new InputStreamReader(System.in));
        int nTestCases = Integer.parseInt(input.readLine());
        for(int i = 0; i < nTestCases; i++) {
            String[] firstLine = input.readLine().split(" ");
            int nRows = Integer.parseInt(firstLine[0]);
            int nCols = Integer.parseInt(firstLine[1]);
            int nWheels = Integer.parseInt(firstLine[2]);
            int finalNode = -1;

            char [][] map = new char [nRows][nCols];
            for(int r = 0; r < nRows; r++) {
                String[] lines = input.readLine().split("");
                for(int c = 0; c< nCols; c++) {
                    map[r][c] = lines[c].charAt(0);
                    if(lines[c].equals("X")) {
                        finalNode = r*nCols+c;
                    }
                }
            }

            int[] wheels = new int[3*nWheels];
            for(int j= 0; j <nWheels*3; j+=3) {
                String[] aux = input.readLine().split(" ");
                wheels[j] = Integer.parseInt(aux[0]);
                wheels[j+1] = Integer.parseInt(aux[1]);
                wheels[j+2] = Integer.parseInt(aux[2]);
            }

            String[] lastLine =  input.readLine().split(" ");
            int[] pos= new int[4];
            for(int p = 0; p<4; p++) {
                pos[p] = Integer.parseInt(lastLine[p]);
            }

            Lost l = new Lost(nRows, nCols, map, wheels, pos, finalNode);
            System.out.println("Case #"+ (i+1));
            int answerJohn = l.calculate(true);
            int answerKate = l.calculate(false);

            if(answerJohn == Integer.MIN_VALUE) {
                System.out.println("John Lost in Time");
            }
            else if(answerJohn == Integer.MAX_VALUE) {
                System.out.println("John Unreachable");
            }
            else {
                System.out.println("John "+answerJohn);
            }

            if(answerKate == Integer.MAX_VALUE) {
                System.out.println("Kate Unreachable");
```

```
63              }
64              else {
65                  System.out.println("Kate "+answerKate);
66              }
67
68          }
69
70      }
71
72  }
```

```java
/**
 * @author Jose Murta 55226 && Diogo Rodrigues 56153
 */

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class Lost {

    private int nCols, numNodes, finalNode;
    private char[][] map;
    private int[] wheels, pos;
    private List<Integer> listJohn, listKate;

    /**
     *
     * @param nRows - number of rows of the island
     * @param nCols - number of columns of the island
     * @param map - matrix with the island's map
     * @param wheels - position and time traveled of the wheels
     * @param pos - John and Kate's initial position
     * @param finalNode - the island's exit
     */
    public Lost(int nRows, int nCols, char [][] map, int[] wheels, int[] pos, int finalNode) {
        this.nCols = nCols;
        this.map = map;
        this.wheels = wheels;
        this.pos = pos;
        this.numNodes = nRows * nCols;
        this.listJohn = createEdgesJohn();
        this.listKate = createEdgesKate();
        this.finalNode = finalNode;
    }

    /**
     * Calculates the time traveled from the start cell to the exit cell
     *
     * @param isJohn - true if the player is John, false if the player is Kate
     * @return the time traveled from the start to the exit cell
     */
    public int calculate(boolean isJohn) {
        int[] le = new int[numNodes];
        List<Integer> list;

        for (int i = 0; i < numNodes; i++) {
            le[i] = Integer.MAX_VALUE;
        }

        int initial = -1;
        if(isJohn) {
            initial = pos[0] * nCols + pos[1];
            list = listJohn;
        }
        else {
            initial = pos[2] * nCols + pos[3];
            list = listKate;
        }

        le[initial] = 0;

```

```
62
63          boolean changes = false;
64          for (int j = 1; j < numNodes; j++) {
65              changes = updateLengths(le, list);
66              if (!changes) {
67                  break;
68              }
69          }
70
71          if (changes && updateLengths(le, listJohn) && isJohn) {
72              le[finalNode] = Integer.MIN_VALUE;
73          }
74
75          int a = le[finalNode];
76          return a;
77      }
78
79      /**
80       *
81       * @param le - array containing the time needed to travel to each node
82       * @param edges - edge's graph
83       * @return - true, if there were changes on the length array; false, otherwise
84       */
85      private boolean updateLengths(int[] le, List<Integer> edges) {
86          boolean changes = false;
87
88          Iterator<Integer> it = edges.iterator();
89          while(it.hasNext()) {
90              int tail = it.next();
91              int head = it.next();
92              int label = it.next();
93              if(le[tail] < Integer.MAX_VALUE) {
94                  int newLen = le[tail] + label;
95                  if( newLen < le[head]) {
96                      le[head] = newLen;
97                      changes = true;
98                  }
99              }
100         }
101         return changes;
102     }
103
104
105     /**
106      * Creates edge's graph from John
107      *
108      * @return the list representing John's graph
109      */
110     private List<Integer> createEdgesJohn() {
111         List<Integer> list = new LinkedList<Integer>();
112         int[] heads = new int[4];
113         for (int i = nCols; i < numNodes-nCols; i++) {
114             boolean found = false;
115
116             heads[0] = i + 1;
117             heads[1] = i - 1;
118             heads[2] = i + nCols;
119             heads[3] = i - nCols;
120
121
122             for (int j = 0; j <4 ; j++) {
123                 if (heads[j] >= 0 && heads[j] < numNodes) {
```

```
124                     if (getType(i) != 'W' && getType(i) != 'O' && getType(i) != 'X') {
125                         if (getType(heads[j]) !='W' && getType(heads[j]) != 'O') {
126                             list.add(i);
127                             list.add(heads[j]);
128                             list.add(1);
129                         }
130                         if (getType(i) != 'G' && !found) {
131                             found = true;
132                             int aux = Character.getNumericValue(getType(i)) - 1;
133                             list.add(i);
134                             int destination = wheels[aux * 3] * nCols + wheels[aux * 3 +
    1];
135                             list.add(destination);
136                             list.add(wheels[aux * 3 + 2]);
137                         }
138                     }
139                 }
140             }
141         }
142 
143         return list;
144     }
145 
146     /**
147      * Creates edge's graph from Kate
148      *
149      * @return the list representing Kate's graph
150      */
151     private List<Integer> createEdgesKate() {
152         List<Integer> list = new LinkedList<Integer>();
153         int[] heads = new int[4];
154         for (int i = 0; i < numNodes; i++) {
155             heads[0] = i + 1;
156             heads[1] = i - 1;
157             heads[2] = i + nCols;
158             heads[3] = i - nCols;
159 
160             if (i % nCols == 0) {
161                 heads[1] = -1;
162             }
163             if ((i+1) % nCols == 0) {
164                 heads[0] = -1;
165             }
166 
167             for (int j = 0; j < 4; j++) {
168                 if (heads[j] >= 0 && heads[j] < numNodes) {
169                     if (getType(i) != 'O'  && getType(i) != 'X') {
170                         if (getType(heads[j]) != 'O')  {
171                             list.add(i);
172                             list.add(heads[j]);
173                             if(getType(i) == 'W') {
174                                 list.add(2);
175                             }
176                             else {
177                                 list.add(1);
178                             }
179                         }
180                     }
181                 }
182             }
183         }
184 
```

```
185         return list;
186     }
187 
188     /**
189      * Gets the type of the cell/node
190      *
191      * @param node - the node's integer
192      * @return the type of the node
193      */
194     private char getType(int node) {
195         int line = node / nCols;
196         int col = node % nCols;
197         return map[line][col];
198     }
199 }
```