Hemuppgift 8- Introduktion till Datalogi

Peter Boström

2009-03-20

Innehåll

1 Grattis på födelsedagen <3

A Källkod

Gammal källkod från förra veckan används även.
Path.java

A.1 Path.java

```
1  package kth.csc.inda;
2
3  import java.io.BufferedReader;
4  import java.io.FileNotFoundException;
5  import java.io.FileReader;
6  import java.io.IOException;
7
8  import java.util.ArrayDeque;
9  import java.util.Deque;
10 import java.util.Stack;
11 import java.util.StringTokenizer;
12
13 /**
14  * @author lemming
15  * Ännu ett fall öfr Karl Martin Frost!
16  */
17 public class Path {
18
19     /**
20      * @param args ignored
21      */
22     public static void main(String[] args) {
23         if(args.length != 2){
24             System.out.println("Usage: java kth.csc.inda.Path FROM TO");
25             return;
26         }
27         int from, to;
28         try{
29             from = Integer.parseInt(args[0]);
30             to = Integer.parseInt(args[1]);
31         }
32         catch(Exception e){
33             System.out.println("Usage: java kth.csc.inda.Path FROM TO");
34             return;
35         }
36         BufferedReader in;
37         int nodes = 0;
38         int numbers[] = new int[3]; // max 3 ävrden öfr en nod, åfrn, till,
               vikt
39         UndirectedGraph graph = new ListGraph(1); // will be replaced
               before usage
40         try{
41             in = new BufferedReader(new FileReader("Distances.txt"));
42             String line;
43             while((line = in.readLine())!= null){
44                 StringTokenizer st = new StringTokenizer(line);
```

```
45                  int size = 0;
46                  while(st.hasMoreTokens()){
47                      String token = st.nextToken();
48                      if(token.contains("//"))
49                          break;
50                      if(size == 3){
51                          System.err.println("Broken file");
52                          return;
53                      }
54
55                      numbers[size++] = Integer.parseInt(token);
56                  }
57                  if(size == 0)
58                      continue;
59                  if(nodes == 0){
60                      if(size != 1){
61                          System.err.println("Missing graph size");
62                          return;
63                      }
64                      nodes = numbers[0];
65
66                      if(Math.min(from, to) < 0 || Math.max(from, to) >=
                            nodes){
67                          System.err.println("Start or end out of bounds");
68                          return;
69                      }
70                      graph = new ListGraph(nodes);
71                      continue;
72                  }
73                  if(size == 1){
74                      System.err.println("Multiple graph sizes or incomplete
                            nodes");
75                      return;
76                  }
77                  if(size == 2) // Unweighed edge
78                      graph.addEdge(numbers[0], numbers[1]);
79
80                  graph.addEdge(numbers[0], numbers[1], numbers[2]);
81              }
82          }
83          catch(FileNotFoundException e){
84              System.err.println("File not found: \"Distances.txt\"");
85              return;
86          }
87          catch(IOException e){
88              System.err.println("IO error: " + e.getMessage());
89              return;
90          }
91          catch(Exception e){
92              System.err.println("Parse error: " + e.getMessage());
93              return;
94          }
95
96          int visited[] = new int[nodes]; // which place the visitor came
                from, used for backtracing.
97
```

```
98          for(int i = 0; i < nodes; ++i)  // init array with -1 as default,
                as 0 can a node which it origined from
99              visited[i] = -1;
100
101         // deque used for bfs
102         Deque<Integer> deque = new ArrayDeque<Integer>(nodes);
103         deque.add(from);
104
105         // bfs
106         Integer node;
107         while((node = deque.poll()) != null){
108             VertexIterator it = graph.adjacentVertices(node);
109             while(it.hasNext()){
110                 int next = it.next();
111                 if(visited[next] != -1)
112                     continue;
113                 visited[next] = node;
114                 if(next == to) // framme! :D
115                     break;
116                 deque.add(next);
117             }
118             if(visited[to] != -1)
119                 break;
120         }
121
122         // if node wasn't reached
123         if(visited[to] == -1){
124             System.out.println(); //empty line
125             return;
126         }
127
128         // search backwards to find way to get here
129         Stack<Integer> stack = new Stack<Integer>();
130         int pathNode = to;
131
132         stack.push(to);
133         do{
134             pathNode = visited[pathNode];
135             stack.push(pathNode);
136         }while(pathNode != from);
137
138         // print out path with fewest jumps
139
140         while((pathNode = stack.pop()) != to)
141             System.out.print("" + pathNode + " ");
142         System.out.println(to);
143     }
144 }
```