CS211 ALGORITHMS & DATA STRUCTURES II

LAB 10

Dr. Phil Maguire

STRING SEARCHING ALGORITHMS

Pen and Paper Exercise

The goal is to find the pattern ABRACADABRA in the string ABRACABABRABRACABRACADABRA.

Show step-by-step how the Knuth-Morris-Pratt algorithm would find the pattern. Find how many comparisons are carried out.

You should first write out the Partial Match Table and then show clearly the progress of the algorithm through the string.

Programming Exercise

Write a program that reads in a file representing a connected undirected weighted graph in the form of an adjacency matrix and outputs the longest path between any two vertices in the graph.

For example, if you load in the file `graph.txt` the output should look something like this:

```
20
EBCJ
```

The idea is to find the two vertices for which the shortest distance between them is as long as possible. For example, in the graph the shortest distance between E and J is 20. There are no other vertices whose shortest path is as long as this. For any other two vertices, there will be a path between them which is shorter than 20.

In effect, all that needs to be done is to run Dijkstra's algorithm for every possible starting position and find the longest Dijkstra path between any two vertices.

This code might be useful as a guide, but it also might be best to code the algorithm from scratch. Just code up what you do on paper, loop for all starting positions, and output the longest Dijkstra path in the graph.

```
public class Graph{

    public static void main(String[] args){

        FileIO io = new FileIO();
        String[] original = io.load("C:\\graph.txt");
        int size=original.length-1;
        int[][] array = new int[size][size];

        for(int i=1;i<=size;i++){   //load up the data into array
            for(int j=1;j<=size;j++){
                array[i-1][j-
1]=Integer.parseInt(original[i].split("\t")[j]);
            }
        }

        int furthestdistance=0; //keep track of the longest path found
so far
        String furthestroute="";
        for(int  i=0;i<size;i++){        //loop  through  all  possible
starting positions

            int visited=0;   //how many vertices have been visited so
far?
            Vertex[]  vertices  =  new  Vertex[size];     //initiate  the
vertex objects
            for(int j=0;j<size;j++){
                vertices[j]=new Vertex(original[0].split("\t")[j+1]);
            }

            vertices[i].visited=true;          //we  visit  the  starting
position
            visited++;  //we've visited one vertex so far
            vertices[i].distance=0;  //set  this  vertex  as  starting
point
            for(int  j=0;j<size;j++){         //update  distances  from
starting point
                if(array[j][i]>0){   //only  update  those  that  can  be
reached from here
                    vertices[j].distance=array[j][i];
                }
            }

            while(visited<size){    //while not all vertices visited
```

```
                …fill this in



            }

            for(int  x=0;x<size;x++){       //check  for  a  new  record
distance
                if(vertices[x].distance>furthestdistance){         //go
through all distances in the graph
                    furthestdistance=vertices[x].distance;
                    int visiting=x;
                    String solution=""; //build up the path taken to
get this particular record beating distance
                    while(visiting>=0){  //the  starting  position  has
route of -1
                        solution=vertices[visiting].label+""+solution;
                        visiting=vertices[visiting].route;       //step
back along the route taken to generate this distance
                    }
                    solution=vertices[i].label+""+solution;  //this  is
the starting position
                    furthestroute=solution;   //keep   track   of   this
record path
                }
            }
        }
        System.out.println(furthestdistance);        //print   out   the
results
        System.out.println(furthestroute);
    }
}


class Vertex{   //everything you need for a vertex

    public boolean visited=false;   //has it been visited?
    int  distance=Integer.MAX_VALUE;       //the  shortest  route  from
starting position to this vertex
    int route=-1;   //keep track of the last step taken to reach this
vertex for the shortest path - what vertex did it come from?
    String label;   //name of the vertex

    public Vertex (String labelin){      //give the vertex a name when
instantiated
        label=labelin;
    }
}
```