Noah Buchanan
Problem Set 7
Algorithms

November 10, 2020

1. G^T:
   1 1 0 1
   0 1 0 0
   1 0 1 0
   1 0 0 1

2. 1]—>[1]—>[3]—>[4]—>/
   2]—>[1]—>[2]—>/
   3]—>[3]—>/
   4]—>[1]—>[4]—>/

3. (Made with the assumption that there are pointers to the parent of the child and the child to the parent)
   1]—>[2]—>/
   2]—>[1]—>[3]—>[4]—>/
   3]—>[2]—>/
   4]—>[2]—>[6]—>[7]—>/
   5]—>[6]—>/
   6]—>[5]—>[4]—>/
   7]—>[4]—>[9]—>/
   8]—>[9]—>/
   9]—>[8]—>[10]—>[7]—>/
   10]—>[9]—>/

4. The runtime of Depth-First Search is O(Edges + Vertices) or O(E+V). The total number of vertices visited is $\leq$ n since in the case that two vertices that are visited share the same edge and are isolated, besides that edge the other vertice is never visited. Notice however it is not simply ¡, it is possible that we visit all nodes individually. Since every node is visited

at most once as we just proved, we know that each edge is scanned only when u or v is visited these are arbitrary vertices with an edge between them for example, the edge $E(u,v)$(undirected) is counted only when either u or v is visted. Since this is the case we know that it is only possible at most for each individual edge to be counted twice assuming an undirected graph(counted only once for directed graphs). From this we get O(2E+V), now drop the constants. Thus the overall runtime is O(E+V).

The runtime of Breadth-First Search is O(E+V) as well. Each vertice is visited at most only once just as before in Depth-First Search. From the previous statement we can also assume this means each Vertice is only put into the Queue used within Breadth-First Search only once per vertice. Meaning that all vertices are visited which puts us at O(V) currently. For each vertice that we visit we count each edge at most once, meaning each edge can only be counted at most twice(or once for directed graphs just as before). From this we can assume O(2E+V) once again, constants are dropped and we get O(E+V).

UAFindMyClass:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;

/*******************************
 * Name: Noah Buchanan
 * Username: ua505
 * Problem Set: PS7
 * Due Date: November 9,2020
 *******************************/

public class UAFindMyClass {

  public ArrayList<Vertex> V = new ArrayList<>();
  public HashMap<Vertex, ArrayList<Vertex>> Adj = new HashMap<>();
  public final static Vertex nil = new Vertex();
  public char[][] input;
  public Vertex[][] vertices;
  public Vertex start = new Vertex();
  public Vertex end = new Vertex();
```

```
public int time;

public static class Vertex {

  boolean color;
  int distance;
  Vertex predecessor;
  int i = 0;
  int j = 0;
  int finish = 0;
}

public UAFindMyClass(String filename) throws IOException {

  BufferedReader br = new BufferedReader(new FileReader(filename));
  String line = "";
  int height = 1;
  int width = 0;

  line = br.readLine();
  width = line.length();

  while((line = br.readLine()) != null) {
    height++;
  }
  br.close();
  vertices = new Vertex[height][width];
  input = new char[height][width];
}

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

  UAFindMyClass G = new UAFindMyClass(args[1]);
  G.buildAdj(args[1]);

  if (args[0].equals("bfs")) {
    G.BFS(G, G.start);
    G.FillPath(G.start, G.end);
    G.writeOutput(args[2]);
  } else if (args[0].equals("dfs")) {
    G.DFS(G);
    G.FillPath(G.start, G.end);
    G.writeOutput(args[2]);
  } else {
    System.out.println("Wrong arguements given");
  }
```

```
}

public void writeOutput(String filename) throws IOException {
  BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
  for (int i = 0; i < input.length; i++) {
    for (int j = 0; j < input[0].length; j++) {
      bw.write(input[i][j]);
    }
    bw.newLine();
  }
  bw.close();
}

public void FillPath(Vertex start, Vertex s) {
  while (s != start && s != null) {
    if (input[s.i][s.j] != 'M') {
      input[s.i][s.j] = 'x';
    }
    s = s.predecessor;
  }
}

public void BFS(UAFindMyClass G, Vertex s) {
  for (Vertex u : G.V) {
    if (u != s) {
      u.color = false;
      u.distance = Integer.MAX_VALUE;
      u.predecessor = nil;
    }
  }
  s.color = true;
  s.distance = 0;
  s.predecessor = nil;
  Queue<Vertex> Q = new LinkedList<>();
  Q.add(s);
  while (!Q.isEmpty()) {
    Vertex u = Q.remove();

    for (Vertex v : G.Adj.get(u)) {
      if (v.color == false) {
        v.color = true;
        v.distance = u.distance + 1;
        v.predecessor = u;
        Q.add(v);
      }
```

```
    }
    u.color = true;
  }
}

public void DFS(UAFindMyClass G) {

  for (Vertex u : G.V) {
    u.color = false;
    u.predecessor = nil;
  }
  time = 0;
  DFSVISIT(G, start);
}

public void DFSVISIT(UAFindMyClass G, Vertex u) {
  time++;
  u.distance = time;
  u.color = true;
  for (Vertex v : G.Adj.get(u)) {
    if (v.color == false) {
      v.predecessor = u;
      DFSVISIT(G, v);
    }
  }
  u.color = true;
  time++;
  u.finish = time;
}

public void buildAdj(String filename) throws IOException {

  BufferedReader br = new BufferedReader(new FileReader(filename));
  String line = "";
  int count = 0;

  while ((line = br.readLine()) != null) {

    for (int i = 0; i < line.length(); i++) {
      input[count][i] = line.charAt(i);
      if (input[count][i] == ' ' || input[count][i] ==
      'S' || input[count][i] == 'M') {
        vertices[count][i] = new Vertex();
      }
    }
    count++;
```

```
}

for (int i = 0; i < input.length; i++) {
  for (int j = 0; j < input[0].length; j++) {

    if (input[i][j] == ' ' || input[i][j] == 'S' || input[i][j] == 'M') {

      vertices[i][j].i = i;
      vertices[i][j].j = j;
      V.add(vertices[i][j]);
      Adj.put(vertices[i][j], new ArrayList<Vertex>());

      if (input[i][j] == 'S') {
        start = vertices[i][j];
      }
      if (input[i][j] == 'M') {
        end = vertices[i][j];
      }

      if (j % (input[0].length-1) != 0) {

        if (input[i][j + 1] == ' ' || input[i][j + 1]
         == 'S' || input[i][j + 1] == 'M') {

          Adj.get(vertices[i][j]).add(vertices[i][j + 1]);
        }
        if (input[i][j - 1] == ' ' || input[i][j - 1]
         == 'S' || input[i][j - 1] == 'M') {

          Adj.get(vertices[i][j]).add(vertices[i][j - 1]);
        }
      }

      if (i % (input.length-1) != 0) {

        if (input[i + 1][j] == ' ' || input[i + 1][j]
        == 'S' || input[i + 1][j] == 'M') {

          Adj.get(vertices[i][j]).add(vertices[i + 1][j]);
        }
        if (input[i - 1][j] == ' ' || input[i - 1][j]
         == 'S' || input[i - 1][j] == 'M') {

          Adj.get(vertices[i][j]).add(vertices[i - 1][j]);
        }
      }
```

```
            }
          }
        }
        br.close();
    }

}
```