Week 8 Graphs Review Session

CSE 373 19su

1. **Applying Algorithms**

   (a) Given a directed graph G and a node v in the graph, devise an algorithm that returns whether or not v is part of a cycle in G.

   (b) If **True**, give a (short) explanation why. If **False**, give a counterexample:

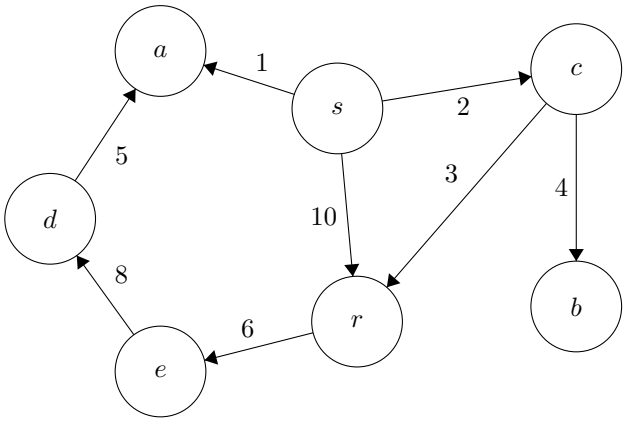
   Assume a connected graph G that we have run Dijkstra on, with start s and goal g. This means each node has its predecessor set correctly.

   After this, I increase the weight of some edge $u \rightarrow v$ on the shortest path between s and g (you don't know by how much). Although my shortest path may no longer be valid, I can look at all of v's incoming edges, select the cheapest one, and set v's new predecessor to be the other endpoint of that cheapest edge.

   This sets all of the predecessors correctly, and I have repaired my shortest path despite the edge change.

2. **Graph Traversal**

Consider the following weighted, direct graph:



(a) Starting from s, what is the path traversed when we add children alphabetically to the Queue using BFS? Ignore the edge weights.

(b) Starting from s, what is the path traversed when we add children alphabetically to the Stack using DFS? Ignore the edge weights.

3. **Disjoint Sets**

When answering the following problems, you should apply the optimizations mentioned in lecture when performing disjoint set operations, including **union-by-rank** and **path compression**. If you must break ties, use the smaller number.

(a) You are given a disjoint set containing numbers 1 to 8, each of the numbers initially in their own set of size 1. Perform the following operations and draw the final state of the disjoint (using an up-tree) set afterwards.

1 2 3 4 5 6 7 8

Union 1 2
Union 3 4
Union 5 6
Union 7 8
Union 1 4
Union 6 7
Union 4 5

(b) Represent your disjoint set using an array, with the index of the array representing an element of the disjoint set. Since our numbers start at 1, feel free to leave index 0 unused/blank. Store the rank of a set in its root node as described in lecture.

4. **Design Decisions**

For each situation, describe a graph that can be used to model it and a graph algorithm that can be used. What are the nodes? What are the edges? Is it weighted or directed?

(a) You are in Southcenter Mall looking at the map. Because you had a very large bubble tea earlier in the day, you need to find the closest restroom to your current location and get there as fast as possible.

(b) John Wick is at the Continental Hotel and he only has a pencil. He must take down all his enemies before they take him down. John doesn't care how far he has to walk or who he gets to first.

(c) Given a set of (x, y) points that visually form distinctly separated clusters, how do you cluster them into these groups using a modified graph algorithm? We might be able to see the groups by eye, but computers are unable to do so unless we tell them how to 'see' using the algorithm.

5. **Runtime Analysis**

Consider this recursive code:

```
// The Node class
public Node {
    public boolean visited; // True if this node has been visited
    public final Node[] children; // The children of this Node
    public final value; // The value stored in this node
}

// The recursive method
public static boolean mystery(Node node, int val) {
    node.visited = true;
    if (node.value == val) {
        return true;
    }
    Node[] children = node.children;
    boolean b;
    for (int i = 0; i < children.length; i++) {
        if (children[i].visited != true && mystery(children[i], val)) {
            return true;
        }
    }
    return false;
}
```

(a) How is this graph represented? What advantages/disadvantages does it have to using a Dictionary implementation?

(b) What is the recursive method doing?

(c) Which graph algorithm is this most similar to and why?

(d) What is the worst-case and its runtime?

(e) What is the best-case and its runtime?

(f) Is this version more or less efficient (time and/or memory) compared to a non-recursive version using the Dictionary implementation of graphs? Why? (Hint: how would you implement this method if this graph was a Dictionary?)