Graphs for Beginners 1

Jul 1.2016

Please pair in groups of 2, 3, or 4 and collaborate. Each section lists points earned(100 means superb).

A. Create a Graph Node – 30 points

Background: Look up what a graph is in Wikipedia. We'll be implementing a graph. Very important: Each node will only contain a single letter, which we'll store as a string for maximum flexibility.

1. In YouTube search for: "graph introduction arnaldo" and click on the edX course link:
https://www.youtube.com/watch?v=22YQcWrnbN8&list=PLO9y7hOkmmSFuBbTCGUMEeaP9pvoQZDcY

Vocabulary: Node: An entity in a graph storing info (aka Vertex).
Edge: A line from one vertex to another.
Directed Graph(Digraph): Graph where edges has arrows.
Weight: Piece of info associated w/ an edge, i.e. cost.

2. Create vertex and edge classes in your language and IDE. Then create a Graph class to hold them. Note: If it's hard to represent strings in your language store an int for data instead. (15 points)

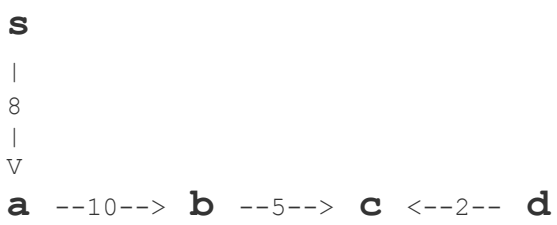
```
enum Status { Unvisited, InProgress, Visited }

class Vertex {
        String data;
        List<Edge> edges;
        Status visited;
}

class Edge {
        Vertex to; int weight;
}

class Graph { Set<Vertex> vertices; }
```

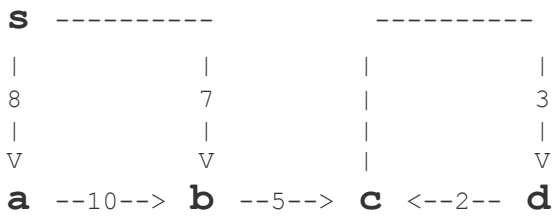
3. Manually create the following graph according to the diagram(15 points):



Ex:

```
Graph g = new Graph();
g.vertices = new HashSet<Vertex>();
Vertex s = new Vertex("s");
Vertex a = new Vertex("a");
Vertex b = new Vertex("b");
Vertex c = new Vertex("c");
Vertex d = new Vertex("d");
g.vertices.add(s);
g.vertices.add(a);
g.vertices.add(b);
g.vertices.add(c);
g.vertices.add(d);
s.edges.add(new Edge(a,8));
...
```

Add some more edges so that the graph looks like this:



B. Depth-first search – 30 points

1. Write your own recursive DFS routine based on this algorithm:

```
DepthFirstSearch(Vertex c)
      If c is the goal
          Exit
      Else
          Mark c "Visit In Progress"
```

```
Foreach neighbor n of c
    If n "Unvisited"
        Depth-First-Search( n )
Mark c "Visited"
```

2. Modify the algorithm so that it prints out the path it travelled to get to the goal.

C. Breadth-First Traversal – 30 points

1. Search for the following video on YouTube: "Graphs – Breadth First Search zooce" (5 points)
https://www.youtube.com/watch?v=EuwG9nk0VxQ

2. Implement your own breadth-first traversal here's the algorithm (10 points):

```
breadthFirst(start)
  q ← empty queue
  q.enqueue(start)
  while (q.hasElements())
     node ← q.dequeue()
     if node.visited != Visited
          visit(node)
          node.visited = Visited
          for each adjacentNode in node.edges
               q.enqueue(adjacentNode)
```

3. Modify the algorithm/data-structure so that it searches for a target, and prints out the path from start to goal when it finds it. (15 points)

D. All Paths from A to B in DAG – 30 points

1. A DAG is a directed, acyclic graph, which means there are no cycles. Assume an input graph is acyclic. Using depth-first search, figure out a way to print out all the paths from some node A to another node B.

2. Implement an algorithm to do this.

E. Implement Depth-first traversal – 10 points

1. Depth-first traversal is similar to the DFS in problem B.
2. Here's an algorithm:

```
DepthFirstSearch(Vertex c)
      Mark c "Visit In Progress"
      Foreach neighbor n of c
            If n "Unvisited"
                Depth-First-Search( n )
            Mark c "Visited"
```