Hw02: Graph Searching Algorithms

Total: 20pts. Due: 3:29PM Feb. 5th on turnin system

You can work on this assignment in a team of two members.

Include your answers to Q1 and Q2 in a file named hw02-part1.pdf and submit it through D2L.

1. [3pts] Show the $d$ and $\pi$ values that result from running breadth-first search on the following undirected graph, using vertex U as the source.

   R - S
   V - W
   T - U
   X - Y

2. [3pts] Show how the depth-first search works on the following graph. Assume that the for loop of lines 6-10 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.
3. [14pts] Implement the graph depth-first search algorithm. I will provide a driver program `dfs.java` and some helping classes for you to start with and test your implementation. The driver program is based on listing 13.1 from textbook of data structure I. This code includes a StackX class, a Vertex Class, a Graph class, and a driver application. You will make the following changes to `dfs.java`.

1. [1pt] Add instance variables to class `Vertex` to represent: a vertex’s color, a vertex’s discovery time, a vertex’s finishing time, and a vertex’s parent.
2. [4pts] Add a public method `dfsRec()` into class `Graph` to implement the method in Q2. This method performs DFS recursively without explicitly using stack. Note that you may need a private recursive helping method in addition to the specified public method.
3. [1pts] Add a public method `displayVisitTime()` into class `Graph`. This method prints out each vertex’s label, discovery time, finishing time, and parent in DFS.
4. [1pts] Add an instance variable `adjList` into class `Graph` so that the graph can be represented by adjacency list. Note `adjList` is an array of `LinkedList`. You can use `java.util.LinkedList`, instead of implementing the `LinkedList` yourself. Change the `Graph` constructor accordingly.
5. [1pts] Add statements to `Graph` method `addEdge` to update the adjacency list when a new edge is created.
6. [4pts] Add a public method `dfsRecList()` in class `Graph` to implement the `dfs` search when the graph is represented by an adjacency list. Basically, you implement the same algorithm as in step 2. The only difference is that the graph is represented by adjacency list.

```
DFS(G)
1   for each vertex u ∈ V[G]
2       color[u] ← WHITE
3       π [u] ← NIL
4   end-for
5   time ← 0
6   for each vertex u∈V[G]
7       if color[u] = WHITE
8          DFS-VISIT(u)
9      end-if
10  end-for

DFS-VISIT(U)
1   color[u] ← GRAY       //White vertex u has just been discovered
2   time ← time + 1
3   d[u] ← time
4   for each v ∈ adj[u]   //Explore edge (u, v).
5       if color[v] = WHITE
6          π[v] ← u
7          DFS-VISIT(v)
8     end-if
9   end-for
10  color[u] ← BLACK     //Blacken u; it is finished.
11  f[u] ← time ← time +1
```
7. [1pt] The driver program *dfs.java* and sample inputs/outputs will be posted on the course website. Your algorithms are supposed to work with *dfs.java* without making any additional changes, especially the package name, method invocation, etc.

8. [1pt] Submit your *dfs.java* only through turnin before due time; No class files or directories should be submitted.