CS2020: Data Structures and Algorithms (Accelerated)

Problems 1–3

For Week09

1 Lecture 13/15 Material - Graph Data Structures

1.1 Explain Several Small Random Graphs

At the beginning of DG6, your DG leader will make sure you and everyone else in his group know these graph terminologies by drawing several **small random graphs** and then ask whether those graph are: **connected/disconnected**, **weighted/unweighted**, **directed/undirected**, **cyclic/acyclic**, **clearly sparse/borderline/clearly dense/complete graph**; **a DAG/a tree/a bipartite graph**; ask how many **vertices/edges** in the graphs; ask how many **in-degrees/out-degrees** of certain vertices in the graphs; ask how many **vertices/edges** in the graphs; ask to identify **components** in the graphs; ask to identify any **path** between two connected vertices and count its **path length** (for unweighted graph, assume that edge weight = 1), etc.

1.2 Verifying Your Understanding of Various Graph Data Structures

Next, using the **small random graphs** drawn earlier, your DG leader will ask some of you to store those graphs as either **Adjacency Matrix**, **Adjacency List**, or **Edge List**.

1.3 Another Conversion Problem :)

In PS6, you were asked to convert a graph stored in an Adjacency List into an Adjacency Matrix. Now, with the presence of Edge List (Lecture15), we can have more combinations. Discuss how to convert a graph stored in Edge List into Adjacency List; Edge List into Adjacency Matrix; etc.

2 Lecture 14 Material - Graph Traversal

2.1 Verifying Your Understanding of BFS and DFS Graph Traversal

Again, still using the same set of **small random graphs** drawn earlier, your DG leader will ask some other students to perform **Breadth-First Search (BFS)** or **Depth-First Search (DFS)** from **various sources** in the graph and perhaps with **various neighborhood ordering** (in the lecture, we adopt a convention that neighbors are listed in ascending order by vertex ID, but this is not always the case). Everyone should verify that the BFS/DFS traversals performed by the selected students are correct (the BFS/DFS traversal should be unique if everyone agree on the same graph, the same source, and the same neighborhood ordering scheme).

2.2 Quick Introduction to Strongly Connected Component - Not Examinable

Observe the following small **Directed Graph**:

```
A -> B -> C
     ^    |
     |    v
     E <- D -> F <-> G
```

In a directed graph, we have a concept of **strongly connected component** (SCC). A directed graph is called **strongly connected** if there is a path from each vertex in the graph to every other vertex. In particular, this means the graph has paths in each direction; a path from any vertex **U** to any vertex **V** and vice versa: a path from **V** to **U**.

The directed graph shown above is **NOT** strongly connected, because (for example) vertex **A** can reach every other vertices, but the other vertices cannot reach **A**. However, the directed graph above have **three SCCs**: {A} (obvious), {B, C, D, E} (every vertex in this SCC can visit each other, please verify this!), and {F, G} (same reasoning as the previous SCC, please verify this too!).

Now assume that you have an algorithm to identify these SCCs[1]. Notice that when you group vertices in these SCCs together, you will obtain a DAG. For example, this is the resulting DAG if the three SCCs are grouped into three larger vertices:

```
A -> BCDE -> FG
```

Now, your DG leader will spend some time to draw some **small random directed graphs**. He will ask some students to 'visually inspect' these directed graphs, identify the SCCs, and redraw these directed graphs into DAGs of SCCs.

3 Lecture 15 Material - Minimum Spanning Tree

3.1 Verifying Your Understanding of MST Problem & Kruskal's Algorithm

In the last part of DG6, your DG leader will draw yet another random graph. This time, he will draw some **small random connected undirected weighted graphs** where the weight of all edges are distinct. He will ask some students to perform the **Kruskal's** algorithm on these graphs.

Recall that Kruskal's algorithm start from an empty tree. It repeatedly chooses unprocessed edge in the original graph based on increasing edge weight. Every time the next unprocessed edge does not cause a cycle, Kruskal's algorithm will always greedily take it.

3.2 Multiple MSTs

Next, your DG leader will draw some more small random connected undirected weighted graphs, but this time **several edges have the same weight**. Try to find all possible MSTs and verify that the MST solution (the actual MS tree) may not be unique although the MST cost is always unique. Verify that Kruskal's algorithm will definitely find one of those MST :).

[1]We learn this in CS3233: Tarjan's and/or Kosaraju's SCC algorithm; These two algorithms are not part of the current CS2020 syllabus. Consider taking CS3233 in S2 AY2011/2012 if you want to know more :)

3.3 Forest of Trees?

Yet another twist, your DG leader will now draw some more small random undirected weighted graphs, but this time the graph is **disconnected**. Run Kruskal's algorithm again on them. What will you get? How to modify Kruskal's algorithm to detect that the given input is disconnected?

3.4 Verifying Your Understanding of Prim's Algorithm - Not Examinable

If you still have time, you can discuss with your DG about Prim's algorithm (i.e. you can scrutinize PrimDemo.java together or run Prim's algorithm on random graphs drawn earlier).

Note that this Prim's algorithm is only briefly discussed in Lecture15. Steven guarantees that for every MST problem that *probably* appear in Quiz 2/Final Exam, you can always choose to run Kruskal's algorithm instead.

4 Discussion on Current Problem Set (PS7)

If you still have time, you can spend some time during DG6 to discuss the solutions for problems in PS7. Note that you are allowed to discuss! You are only prohibited from coding the solution together/copying the solution. Write down the list of collaborators in your solution.