ICS 202 – Data Structures
Fall Semester 2018/2019 (181)
Assignment # 4–Hints

Question 1: This problem is about reachability tests from a start vertex (vertex 0). Run dfs(0) on the input graph to record vertices that are reachable from vertex 0. Then to check which vertices are dominated by vertex X, we (temporarily) turn off all the outgoing edges of vertex X and re-run dfs(0). Now, a vertex Y is not dominated by vertex X if dfs(0) initially cannot reach vertex Y or dfs(0) can reach vertex Y even after all outgoing edges of vertex X are (temporarily) turned off. Vertex Y is dominated by vertex X otherwise.

Tips: We do not have to physically delete vertex X from the input graph. We can simply add a statement inside our DFS routine to stop the traversal if it hits vertex X.

Question 2: Represent the directed Graph as adjacency list. If task 'y' depends on 'x' then there is an edge from 'x' to 'y'. Implement a topological sort!

Question 3: The task is to find that the provided digraph is strongly connected or not. Represent the input as a digraph (with one-way streets as one directed edge and two-ways streets as 2 directed edges.) and perform the test to see if the resultant digraph is strongly connected or not.

Question 4: Straightforward implementation of Dijkstra's Shortest Path Algorithm.

Question 5: This problem seems a bit tricky! After a careful look, you can discover that what you need is a maximum (not a minimum) spanning tree. The edges which are left out will be the least cost edges which if added to the tree will create loops. So you need to place camera on these edges.

Algorithm:

- Sort the edges of the graph in decreasing weight (descending order) to perform Kruskal's 'Maximum' Spanning tree.
- Sum the cost of the left over edges which were not included in the Spanning tree, this is the total cost.

Question 6: This is about Flood Fill - Labeling/Coloring the Connected Components.

After representing a graph as an adjacency matrix, call a recursive traversal, floodfill (int r, int c), if the location (r,c) is not 'W' return 0 and if (r,c) is 'W', count +1 and call (floodfill()) 8 times for all its neighbour and replace 'W' with, say, 'V' so that you don't count it again:

Algorithm:

- if(graph[r][c]) is not water because it is land or the coordinates are out of the map boundaries.
- return 0;
- graph[r][c] = 'V'
- return 1 + floodfill(r+1, c) + floodfill(r, c+1) + floodfill(r+1, c+1) + floodfill(r-1, c-1) +... all the eight neighbours.

- reset the 'V's to 'W's before the next query.