AOA Final Exam 1

1. (20 points) The following statements may or may not be correct. In each case, either prove it or give a counter example.

   (a) If the lightest edge in a graph is unique, then it must be part of every MST.
   Yes. Lets assume we have a MST which does not contain the min weight edge. Now the inclusion of this edge to the MST will result in a cycle. Now there will always be another edge in the cycle which can be removed to remove the cycle and still maintain the graph(MST) connected. Removing this edge, and leaving (u, v), we now have a tree with a lower total weight than the original T. Because T was assumed to be an MST, this is a contradiction. Therefore, any unique light edge of a cut must be part of every MST.

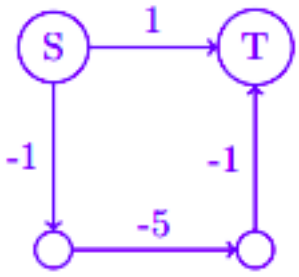
   (b) Djikstra's algorithm works correctly even when there are negative edges in graph.

   (c) If there is a path from u to v in a directed graph G and if d[u] $<$ d[v] in a dfs search of G, then v is a descendant of u in the depth first forest produced.

2. (15 points) Give an efficient algorithm which takes as input a directed graph G(V,E) and determines whether of not there is a vertex s $\in$ V from which all other vertices are reachable.

3. (15 points) Professor Lucent suggests the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges.

   Add a large constant to each edge weight so that all the weights become positive. Then run Djikstra's algorithm starting at node s and return the shortest path found to node t.
   Is this a valid method? Either prove that it works correctly or give a counter example.
   The above algorithm doesn't work when the actual shortest path has more edges than other potential shortest paths. In this case, the paths with more edges have their weights increased more than the path with fewer edges. We can see this in the following counterexample:



   The shortest path is "down-right-up" (weight -7). After adding M = 5 to each edge, we increase the actual shortest path by fifteen 15. The path "right" only increases by 5 and so the algorithm returns this path as the shortest path.

   OR
   No, consider the graph on 3 vertices:
   s → a = 1,
   a → t = 1,
   s → t = 10.
   Clear the shortest path is s → a → t with total length 2.

But if we add a constant c to all edge lengths, then the length becomes 2 + 2c while the length of s → t is 10 + c. Once c > 9, we will incorrectly return s → t instead.

4. (30 points) You are given a set of cities, along with the pattern of highways between them, in the form of undirected graph G(V,E). Each segment of highway e ∈ E connects two of the cities and you know its length in miles, le. You want to get from city s to city t. There is one problem : your car can hold only enough gas to cover L miles. There are gas stations in each city but not between cities. Therefore, you can only take a route if every one of its edges has length le ≤ L.

   (a) Given the limitation on your car's fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t.
   Since we can only take a route if every one of its edges has a length less than or equal to L, we can simply remove edges of a greater value and see if the city t is still reachable from s. A simple Breadth First Search should be able to verify this. BFS step is O(|V | + |E|).

   (b) You are now planning to buy a new car and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give $O((V + E)\ logV)$ algorithm to determine this.
   Define an edge u, v to be tense if max(dist(u), w(u, v)) < dist(v) and relax a tense edge (u, v) by setting dist(v) to be max(dist(u), w(u, v)). Then run Dijkstra's with these new changes.

```
procedure dijkstra(G, l, s)
Input: Graph G = (V; E), directed or undirected;
positive edge lengths l for each edge e in E
vertex s in V
Output: For all vertices u reachable from s, dist(u) is set
to the distance from s to u.
for all u in V:
dist(u) = infinity
prev(u) = nil
dist(s) = 0
H = makequeue(V) // using dist-values as keys
while H is not empty:
    u = deletemin(H)
    for all edges (u, v) in E:
        if dist(v) > max(dist(u), l(u, v)) //change here
        dist(v) = max(dist(u), l(u, v))
        prev(v) = u
        decreasekey(H, v)
```

5. (20 points) Milk packing is such a low margin business It is Important to keep the price of of your product as low as possible help Mary milk makers get the milk they have to sell cheapest possible manner. Merry milk makers company has several farmers from which they may buy milk and each one has potentially different price at which they sell to milk producing plant. Moreover, as cow can only produce so much milk in a day, the farmers only have so much milk to sell per day. Each day Merry milk makers can buy an integral amount of milk from each farmer less than or equal to farmer's limit.

   Given the Merry milk makers daily requirement of milk along with the cost per gallon and amount of available milk for each farmer, calculate the minimum amount of money that it takes to fulfil Merry milk makers requirement.

   Note: The total milk produced by farmers will be enough to meet demands of Merry Milk Makers

6. (20 points) A subsequence is a palindrome if it is the same whether read left to right of right to left - for instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences including A, C, G, C, A and A, A, A, A (on the other hand the subsequence A, C, T) is not a palindrome. Devise an algorithm that takes a string n [1...n] and returns the length of the longest palindromic subsequence. Its running time should be $O(n^2)$

```
if (s[i - 1] == rev[j - 1]) lps[i][j] = lps[i - 1][j - 1] + 1
else lps[i][j] = max(lps[i - 1][j], lps[i][j - 1])

LPS(string s)

    int n = s.length();
    string rev = s;
    reverse(begin(rev), end(rev));
    vector<vector<int>> lps(n + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j)
            if (s[i - 1] == rev[j - 1])
                lps[i][j] = lps[i - 1][j - 1] + 1;
            else
                lps[i][j] = max(lps[i - 1][j], lps[i][j - 1]);


    return lps[n][n];
```