CS3340 Asn 3

Daniel Rawana – 250911447

q1.

0	0	1	0	2	3	4	5	6	7	8	9	10	11	0	12	13	14

q2.

q3.

```
PRINT_LCS (c, A, p, q)
 if p == 0 || q == 0 then return
 if c[p,q] == c[p-1, q-1] then
        PRINT_LCS (c, A, p-1, q-1)
        Print the sequence
Else if c[p-1,q] >= c[p, q-1]
        PRINT_LCS (c, A, p-1, q)
Else
        PRINT_LCS (c, A, p, q-1)
```

q4.

```
Knapsack(n, W)
Initialize an n+1 by W + 1 table called K
For y = 1 to W
        K[0,y] = 0
For x = 1 to n
        K[x, 0] = 0
for i = 1 to n
        for j = 1 to W
                Check if j < i's weight, if it is
                        K[i,j] = K[i-1, j]
        K[i,j] = max (K[i-1,j], K[i-1, j-i.weight] + i.value)
```

Pick the lightest as well as the most valuable item. This solution is optimal because if there was an item that we included but a smaller and more valuable item i existed, we could swap item j in the knapsack with item i. It will increase the total value because i is more valuable and it will also fit because i is lighter than j.

5. If we use a 1D array called arr, arr[W+1] can be used such that arr[i] stores the maximum using all items and i capacity of the knapsack.

```
unboundedKnapsack(int W,  int n, int[] val, int[] weight)
    int arr[] = new int[W+1]
    from i = 0 to W
        for j = 0 to n
            if weight[j] <= i
                arr[i] = max (arr[i], arr[i-weight[j]] + val[j])


    return arr[W] after the loops
```

q6.
In order to find the maximum spanning tree, we must negate all edge weights and apply the MST algorithm rule. Multiply -1 to all edge weights. We must then apply Kruskal's or Prim's algorithm to find the minimum spanning tree.

```
Set of vertices = setOfVert
Set of edges = setOfEdges
Return maximumtree (setOfVert, setOfEdges)
For edge in setOfEdges: (set all edge weights to negative)
    edge.weight = -edge.weight
Mst = Kruskals(setOfVert, setOfEdges)
Maximum = setOfEdges
For edge in Mst.edges
    Maximum.remove(edge)
Return Maximum
```

q7.
A->C = 2

A->B = 5

B->C = -6

Dijkstra's algorithm assumes that any node originating from it will lead to greater distance as it marks the vertex as "closed". Therefore, the algorithm finds the shortest path to it and will never visit this node again. However, this doesn't hold true in the case of negative weights as shown above.

q8.
If we apply Floyd Warshall's algorithm for example:

1->2 = 4

2->3 = 3

3->1 = -5

This graph has a negative edge but doesn't contain a negative cycle.

Distances:

(1,1) = 0 (1,2) = 4   (1,3) = 7   (2,1) = -2   (2,2) = 0

(2,3) = 3 (3,1) = -5 (3,2) = -1 (3,3) = 0

Therefore, from the above example we can say the algorithm is still correct in a graph given a negative edge but no negative cycle.

q9.

We will first apply the Floyd Warshall algorithm on the graph.

```
Min = ∞

For each vertice (u,v)

    If (dist(u,v) + dist(v,u) < min)

        Min = dist(u,v) + dist(v,u)

        Pair = (u,v)

        Return path(u,v) + path (v, u)
```