Assignment 3

Dinushan Dayarathna

250725428

Question 1

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P[i]	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
Next()	0	0	1	1	2	3	4	5	6	7	8	9	2	3	4	5	6	7

Question 2

The KMP string matching algorithm is already scanning the text, so the state of the KMP shows the length of the longest substring of string T. Finding the longest substring of T will require us to update a maximum length counter and recording the location in the string.

Question 3

```
Print_LCS (c, X, Y, i, j)
        Return if i = 0 or j = 0
        If X[ i ] = Y [ j ]
                Print_LCS (c, X, Y, i – 1, j – 1)
                Print X[ i ]
        Else If c[ i – 1, j ] >= c[ i, j – 1]
                Print_LCS(c, X, Y, i – 1, j)
        Else
                Print_LCS(c, X, Y, i, j – 1)
```

This algorithm will run in O(m + n) time since it will run for i + j times, visiting each location only once.

Question 4

At each step, simply pick the lightest (most valuable) item that can be picked. This is an optimal solution: for example, if there were some item j in the knapsack and an item i that is smaller and more valuable outside, then we can replace j with i. We know for sure that it will fit since i < j and the value of the knapsack will increase too.

Question 5

Given a knapsack of weight W and n items and having the possibility that each item appears more than once, we can use a simple 1D array of size W + 1. The array at position i will store the maximum value which can be achieved using all items and i capacity of knapsack. The difference from the classical format is that we are using a 1D array here instead of a 2D array in the classic solution, since the number of items never changes, all the items will always be available. Can calculate the content of the 1D array using the recursive formula here:

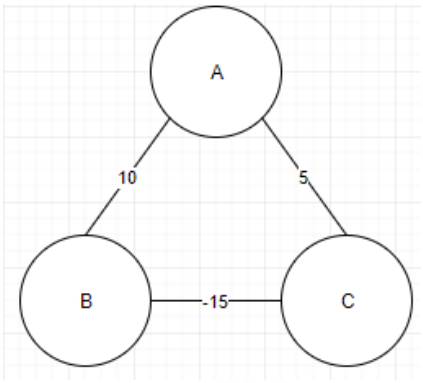
Array [i] = 0
Array [i] = max (Array [i], Array [i – wt [j] + val [j] } // Where j varies from 0 to n – 1 such that wt [j] <= i
Result = Array [W]

Question 6

Modifying the minimum spanning tree algorithm to find the maximum spanning tree can be done as follows for a network G:

1. First, decreasingly sort the edges of G by weight. We will represent the set of edges comprising the maximum weight spanning tree as T, currently T = Ø.
2. Add the first edge in the order to T.
3. Add the next edge to T if and only if it does not form a cycle in T. Exit if there are no remaining edges, G will be disconnected
4. If T has n – 1 edges (n = number of vertices in G) the algorithm will stop and present T, else jump to step 3.

Question 7



Following Dijkstra's algorithm, it will mark a vertex as closed once it thinks that it has found the shortest path to that node. The algorithm will never change the value at that node again, it will assume that the path developed will be the shortest. However, looking at the three nodes, ABC, and having A as the source node, C will be relaxed to have value 5 and then be marked as closed. The algorithm will never find the shorter path to C which is A → B → C which should have value -5.

Question 8

Given a graph G (V,E), with negative weights but no negative cycle in G, the all-pair-shortest-path algorithm or the Floyd-Warshall algorithm is still correct. As described in the textbook, the algorithm works by analyzing the intermediate vertices of a shortest path. Intermediate vertices of a simple path $p = \{v_1, v_2 \ldots v_l\}$ is any vertex of p other than v_1 or v_2. Since the algorithm takes into account all of the intermediate vertices and so will take into account any negative weights associated with any edges that are connected to nodes. Since the algorithm will consider negative weights it will be able to find the shortest path to a particular node since all intermediate edges will be considered, not just immediate edges.

Question 9

We can use the Floyd-Warshall algorithm to find a cycle in graph G with minimum weight. The algorithm allows negative weights to be present but we assume that there are no negative-weight cycles, which is valid with the question criteria. The algorithm will run in $O(V^3)$ time. The algorithm runs as follows:

```
Min = infinity
For each pair of vertices 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)
```

The path(u,v) + path(v,u) is the cycle that is discovered of minimum weight in G. If there are self loops in G, then those will need to be checked separately as this algorithm does not cover them.