Chuangxin Xia
301373629
CMPT 706 HW4

1.

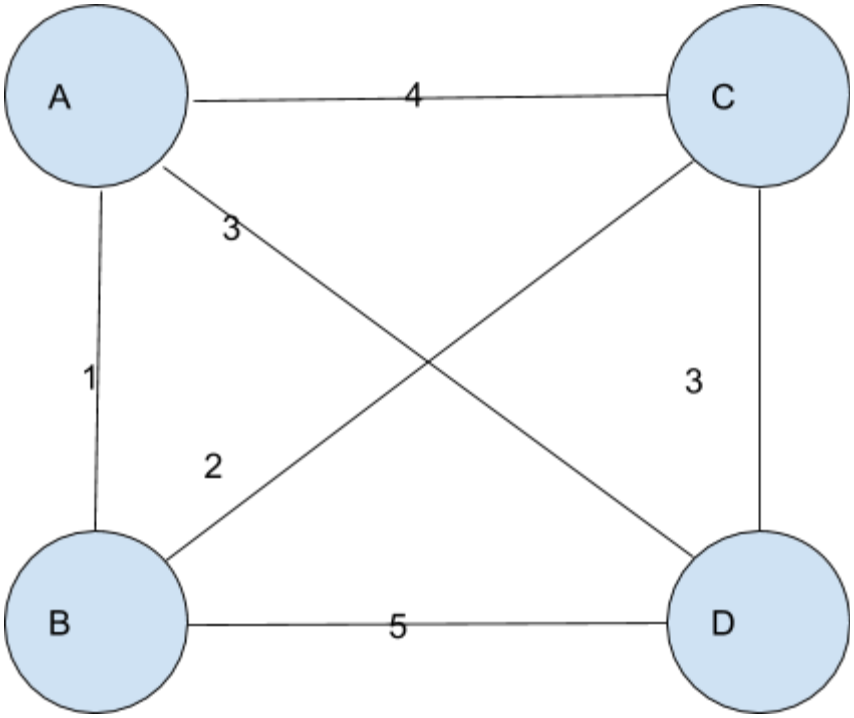
```
def prob1(s, s_sub)
    j=0
    for(int i = 0; i < s.length, i++) {
          if (j == s_sub.length) break;
          if (s[i] == s_sub[j]) j++;
    }
    if (j == s_sub.length) return true;
    else return false;
```

2.

a) A minimum cost spanning tree must have edges with lowest weights that connect all the vertices and no cycle. We can have the algorithm run all the edges and compare with new edge. If the new edge has lower weight and replace another edge with no cycle. So it runs in O(|E|)

b) Since T is no longer a minimum spanning tree, it means when the edge is added, there's a cycle because of minimum spanning tree's cycle property. All we need to do is find the cycle in T, remove the edge with higher weight that's causing the cycle and replace it with the new edge.

3.



a) FALSE. Let G be {(A,B), (A,D), (C,D)}, it is a minimum bottleneck tree with bottleneck being 3. However, it is not a minimal spanning tree.
b) TRUE. Proof by contradiction. Assume that there is a MST T that is not minimum bottleneck tree. Let the bottleneck tree be T'. There's a bottleneck edge in T that is greater than all the edge in T'. If I could remove the largest edge from T and replace it with a smaller with and still complete a MST. Then the assumption that T is MST in the first place is not true.

4.

Since the input is a tree, so we are aware of the leaf node, those with degree 1. Our final set of edge will include those leaf node for sure since there's only one edge going into those vertices. So we add all the leaf vertices to our set, and remove then along with the connecting edge. We repeat until all the vertices has been visit and have our set. If there are unvisited vertices, then there's no perfect match for such tree.

5.

It's going to look exactly like the original Huffman algorithm except each time we choose 3 smallest frequencies and sum up to make a new node instead of 2. The input needs to be odd set of nodes, otherwise, add offset node with complete a odd set.

The proof of correctness should be the same as the original Huffman. The lower frequency ones are always lower, which means more frequent ones will be higher up in the tree with shorter codewords.