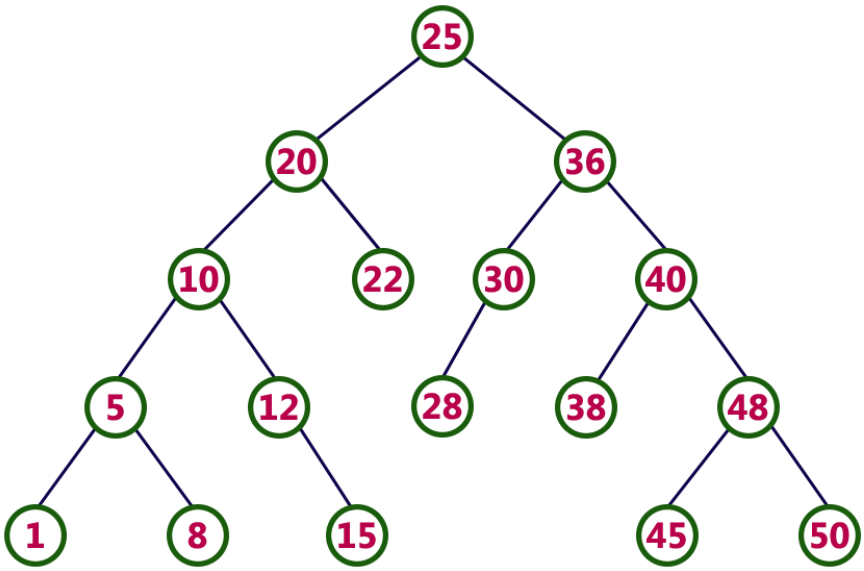
1. Given arbitrarily big string with alphabet [Implement using singly or doubly linked list]
    a. Store the each alphabet into linked list return the first and last node. (Store)
    b. Print each alphabet with number of duplicates (traversal through linked list) (Print)
    c. Sort the alphabet based on number of duplicates (Sort)
    d. Remove extra alphabet if it occurs more than T times consecutively and print the final linked list (Remove)

    Example
    Input:
    SSSKKBBBBHHHJJJJKKKK 3
    Store
    Print
    Ascend/Sort
    Remove 3
    Output:
    S K
    S 3 K 6 B 4 H 3 J 4
    KBJSH
    SSSKKBBBHHHJJJKKK

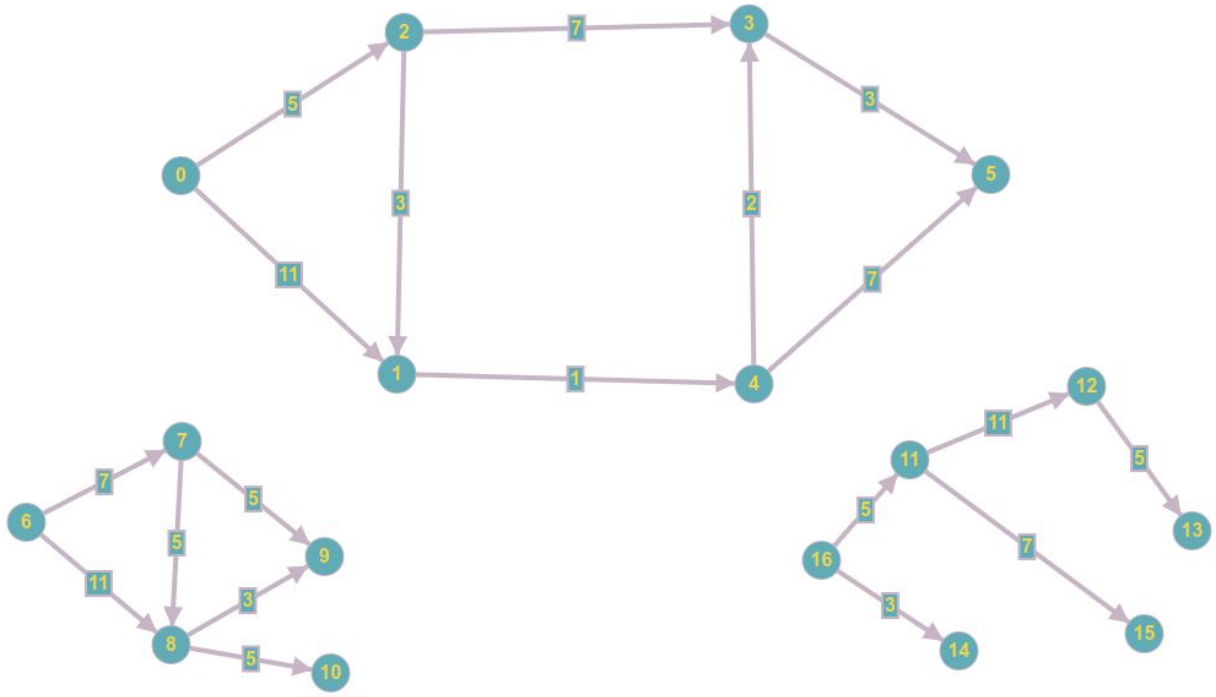
2. Construct a Binary Search tree with the given set of keys (with no duplication of keys), Implement the following functionalities:
        A. Find predecessor of a given node (P node)
        B. Find successor of a given node (S node)
        C. Find the minimum and maximum values in the subtree rooted at the given node. (M node)
        D. Given two nodes find the lowest common ancestor (C node1 node2)



Example :
Inputs is a list of keys, to be inserted in the BST. The keys will be separated by spaces, and the input will end with a new line character.
Input:
25 20 36 10 22 30 40 5 12 28 38 48 1 8 15 45 50

Functionality Input:
M 10
M 40
M 12
P 10
P 38
S 10
S 36
C 38 50
C 25 38
Output:
1 15
38 50
12 15
8
36
12
38
40
25

3. Basic functionalities of graph
   In a graph G,
   a. – **neighbors(v)** : returns all neighbors of v
   b. – **vertices()** : returns all vertices in G
   c. – **edgeWeight(v1, v2)** : returns the weight of the edge between v1 and v2.
   d. – **containsVertex(v)** : returns whether G contains a vertex v.
   e. – **containsEdge(v1, v2)** : returns whether G contains an edge between v1 and v2.
4. Construct a directed graph G with the input and implement the following:



   a. Display all the connected components and print its vertices (Find)
      UPDATE: For part (a), treat the graph as undirected.
   b. Find the shortest path from a single vertex to all other vertices in the same connected component. If G is connected, the program should print shortest distance to all the other vertices. (SP sourceVertex) print its distance

Input is of the form (E v1 v2 wt) where 'wt' represents the weight of the edge from v1 to v2.

Example

Input:

E 0 2 5
E 0 1 11
E 2 1 3

E 2 3 7
E 1 4 1
E 4 3 2
E 4 5 7
E 3 5 3
E 6 7 7
E 6 8 11
E 7 8 5
E 7 9 5
E 8 9 3
E 8 10 5
E 16 11 5
E 16 14 3
E 11 12 11
E 11 15 7
E12 13 5

Functionality input:
Find
SP 0

Output:
3 components
0 1 2 3 4 5
6 7 8 9 10
11 12 13 14 15 16
0 0 0
0 1 8
0 2 5
0 3 11
0 4 9
0 5 14

5. Given an undirected graph with edge weights, implement Kruskal's algorithm with the help of Disjoint Set (Union Find) Data Structure. The Disjoint Set Data Structure should support the function calls Makeset(x), Union(x, y), and Findset(x). The input will be given the same form as in the previous problem, but here we have to consider them as undirected graphs.

Output Format:

If spanning tree exists, output the minimum spanning tree as a list of edges like below:

(1, 2), (3, 4), (1, 4), (2, 5), (3, 6)
If spanning tree does not exist, print a message stating the same.