Aditya Shirode, Rhythm Shah, Shrenuj Gandhi

3. [9 points] The *bottleneck weight* of a spanning tree T of a weighted, undirected graph G is the weight of the largest weight edge in T. A *bottleneck spanning tree* is a spanning tree with minimum bottleneck weight.

   (a) Prove that a minimum spanning tree is also a bottleneck spanning tree.

   (b) Design a linear time algorithm that, given a graph G and an integer b determines whether or not there exists a spanning tree T of G with bottleneck weight $\leq b$.

   (c) Use your algorithm in part (b) to design a linear time algorithm for finding a bottleneck spanning tree. Hint: Use contraction of sets of vertices into a single vertex as in my descriptions of various MST algorithms.

Answer:

A *bottleneck spanning tree* T of an undirected, weighted graph G is - a spanning tree of G whose largest edge weight is the minimum over all spanning trees of G.

We say that, the value of bottleneck spanning tree T is the weight of the maximum weighted (bottleneck) edge in the tree.

(a) Let us assume that a minimum spanning tree is not a bottleneck spanning tree.

For an undirected, weighted graph G -
Let MST be the minimum spanning tree with e as the largest weight edge.
Let BST be the bottleneck spanning tree with e' as the largest weight edge.

Then based on our assumption, we can say that $e' \leq e$.
But if this is true, then MST cannot be a minimum spanning tree, as it contains an edge is not minimal.

Hence, out assumption is wrong and a minimum spanning tree is always a bottleneck spanning tree.

(b)
```
isBST(V,E):
    cc ← DFS(V,E)
    return cc
end

Algorithm(G,b):
    E' ← Remove all edges from E which are greater than b
                                                        Θ(E)
    cc ← isBST(V,E')                                    Θ(|E| + |V|)

    if |cc| = 1, then:
        "There exists a spanning tree with bottleneck weight ≤ b"
    else:
        "No spaning tree with bottleneck weight ≤ b"
end
```

As seen from above algorithm, this takes *linear* time.

$isBST()$ function is a modified DFS search, which returns the number of connected components. (As mentioned in class notes)
We remove all the edges which have weights below the given bottleneck weight b, and see if the remaining tree edges form a *Spanning Tree*. (i.e. if there are more than one connected components, it means that the tree is not a ST).
If there is only one connected component, then it means there exists a spanning tree with bottleneck weight $\leq b$.

(c)
```
BST(V, E):
  q ← MEDIAN(E)                                              Θ(E)
  ▷ E_A ← edges ≥ E[q]
  ▷ E_B ← edges ≤ E[q]
  cc ← isBST(V,E)                                            Θ(|E| + |V|)

  if |cc| = 1, then:
    return BST(V, E_B)
  else:
    V_C ← construct a set of connected components into single vertex ⋃
           vertices missing from CC                          Θ(V)
    E_C ← Edges from E_A that connects the components in CC ⋃
           edges that connect to vertices not in CC
                                                             Θ(E)
    return CC ⋃ BST(V_C, E_C)
```

Overall: O($|E| + |V|$)