Jade Yu Cheng
ICS 311
Homework 10
Oct 7, 2008

Question for lecture 13

Problem 23-3 on p. 577

Bottleneck spanning tree

A bottleneck spanning tree T of an undirected graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. We say that the value of the bottleneck spanning tree is the weight of the maximum0weight edge in T.

a. Argue that a minimum spanning tree is a bottleneck spanning tree.

Answer: Assume we have the MST for graph G. The largest weight edge of the MST is $w(u, v) = a$. The argument is converted to arguing that this edge $w(u, v)$ is not the part of the bottleneck spanning tree of the same graph G.

Case 1. The largest edge $w'(u', v') = b$ in the bottleneck spanning tree is larger than edge $w(u, v), b > a$. In this case, it's clear that the edge $w'(u', v')$ can't be the return value of the bottleneck spanning tree of graph G. Otherwise there exists another tree structure that provides a smaller largest edge. It's a conflict of the definition of bottleneck spanning tree.

Case 2. The largest edge $w'(u', v') = b$ in the bottleneck spanning tree is smaller than edge $w(u, v)$, $b < a$. In this case, we consider that the edge $w(u, v)$ in MST separates all vertices into two parts, left sub-tree T_l and right sub-tree T_r. This assumption is based on the definition of the tree structure. No cycles exist in tree structures, and all edges connect two sub-parts of the tree graph. Image is shown as FIGURE 1.

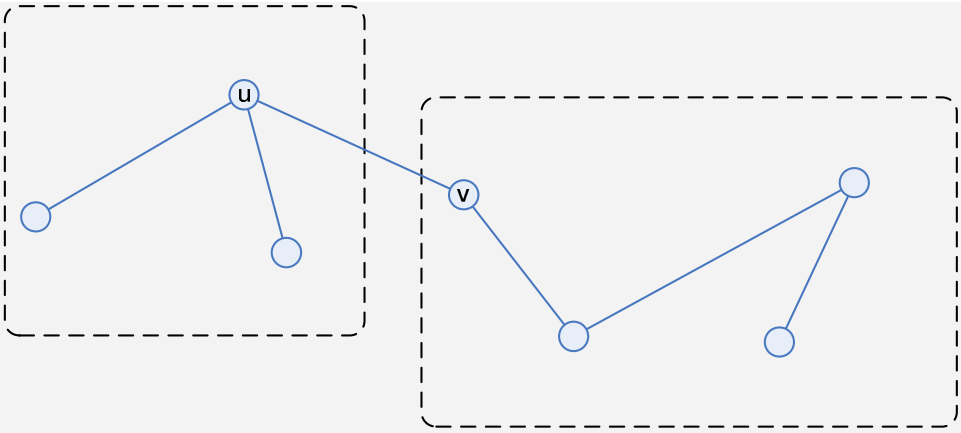

FIGURE 1. MST with its largest edge $w(u, v)$.

Suppose we have a bottleneck spanning tree for graph G, which doesn't take edge $w(u, v)$. Instead, based on our assumption, we believe in this bottleneck spanning tree, the largest weight edge $w'(u', v')$ occurs somewhere else and it's smaller than $w(u, v)$, $b < a$.

If we delete $w(u, v)$ in the picture above, based on the definition of tree graph, there should be another edge added to connect T_l and T_r. We assume $e(x, y) = c$ was included in the bottleneck spanning tree in order to stitch T_l and T_r together. Image is shown as FIGURE 2.

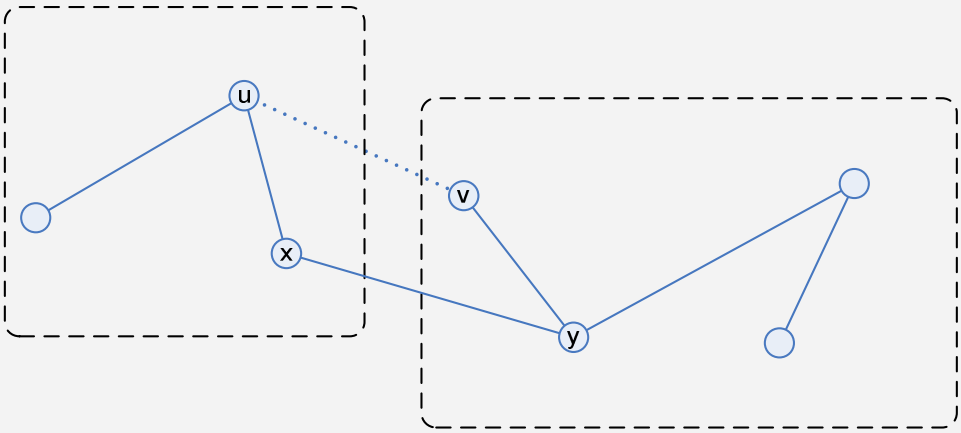

FIGURE 2. Bottleneck spanning tree.
The largest edge $w'(u', v')$ is not shown on this image.
Sub-tree T_l and T_r are not necessarily the same as the corresponding MST.

According to the bottleneck spanning tree, the relationship of $e(x, y) = c$ and $w'(u', v') = b$ should be $c \leq b$. At the same time, according to the MST definition, the relationship of $e(x, y) = c$ and $w(u, v) = a$ should be $c > a$. Therefore $a < b$ because $c \leq b$ and $c > a$. It indicates the length of $w(u, v)$ is smaller than the length of $w'(u', v')$. This is a conflict with the assumption of this case: The

largest edge $w'(u', v') = b$ in the bottleneck spanning tree is smaller than edge $w(u, v)$, $b < a$.

Therefore we proved that $w(u, v)$ is also in the bottleneck spanning tree and is the largest edge of it. Therefore MST is also a bottleneck spanning tree.

b. Part (a) shows that finding a bottleneck spanning tree is no harder than finding a minimum spanning tree. In the remaining parts, we will show that one can be found in linear time.

   Give a linear-time algorithm that given a graph G and an integer b, determines whether the value of the bottleneck spanning tree is at most b.

Answer: Consider we have a connect graph G shown as FIGURE 3. In order to decide if the return value of a bottleneck spanning tree is no larger than a given value b, we simply need to decide whether the edges that are larger than this given value b are all necessary for the graph to maintain a tree structure.

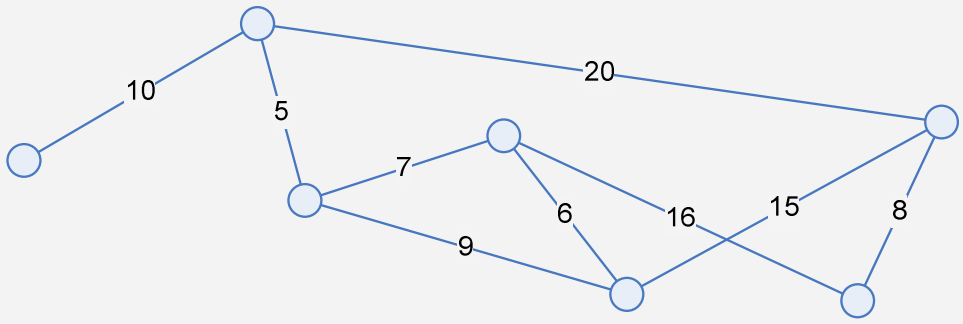

FIGURE 3. Connect graph G.

There are two cases based on the input value of b. If the left over graph is a connected graph, the deleted edges are not necessary for maintaining the tree structure of this graph G. If the left over graph is no longer a connect graph, at lest one of the deleted edges is essential for maintaining the tree structure.

Here is an example of case 1. The connected graph is shown as FIGURE 3, and the given value is $b = 18$. By deleting the edges that are larger than b, we would get a graph shown as FIGURE 4.

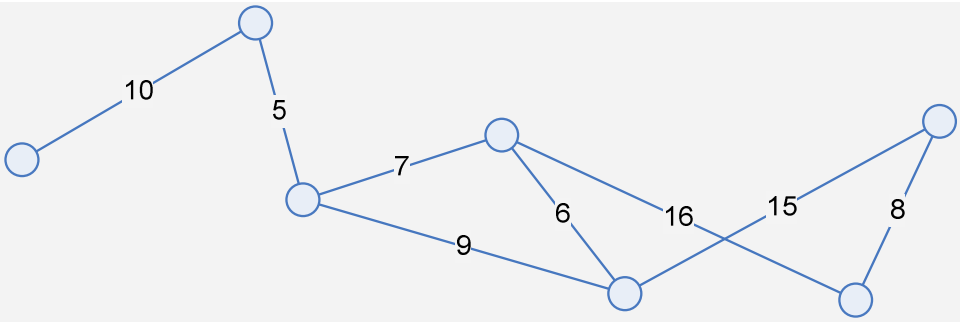

FIGURE 4. Derived graph of G after deleting all edges larger than $b = 18$.

Here is an example of case 2. The connected graph is shown as FIGURE 3, and the given value is $b = 14$. By deleting the edges that are larger than b, we would get a graph shown as FIGURE 5.

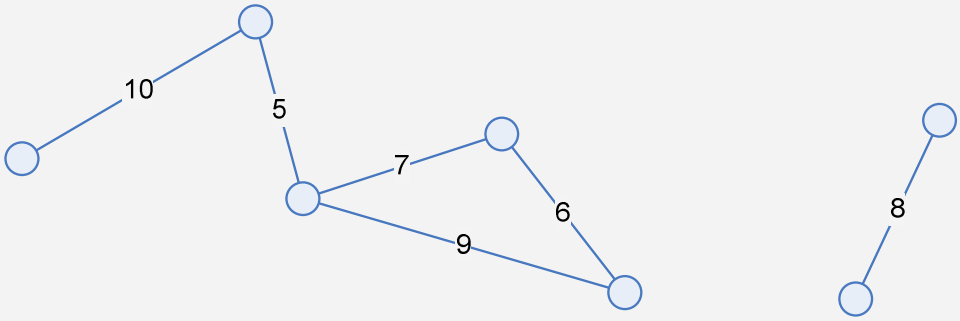

FIGURE 5. Derived graph of G after deleting all edges larger than $b = 14$.

From the images, we can easily decide that the return value of a bottleneck spanning tree of this graph G has to be larger than 14 and smaller than 18, since we observed the fact that deleting all edges larger than 18 didn't change the feature of a connected graph but deleting all edges larger than 14 failed to maintain this feature.

To decide which edges to delete takes at most $O(E)$ time because we go through all edges once. To decide if the left over graph is still a connect graph takes $O(V + E)$ time because we can use either DFS or BFS. These algorithms provide the tree traversal in a linear time. Therefore the combined run-time of this method would be $O(2E + V)$, linear time.

c. Use your algorithm for Part (b) as subroutine in a linear-time algorithm of the bottleneck spanning tree problem.

Answer: To solve the bottleneck spanning tree problem is the same problem as finding the largest edge needed for the graph to maintain as a connect graph. We can repeatedly check for different values of b as shown in the two examples above.

The problem will come down to something like: the solution is larger than a and smaller than b, while $a + 1 = b$. Then the solution becomes obvious.

The fastest way to get to the right value is using a binary search. If we have the largest edge in the graph as n, we repeatedly look of $n/2, n/4, n/8$.. until we find the solution.

If the graph is more close to a complete graph, it has lots of edges, then the run-time of the method in Part (b) would be determined by V. Its run-time would be $O(v)$. If this is the case, repeatedly looking for about half of the edges make the problem as a whole holds a linear run-time:

$$O(v) + O(v/2) + \cdots + O(1) = O(2v) = O(v)$$

If the graph is more close to a collection of vertices, then the run-time of the method in Part (b) would be determined by E. Its run-time would be $O(E)$. But this is clearly not the case for this problem.

Therefore the run-time of solving the bottleneck spanning tree problem would be linear.