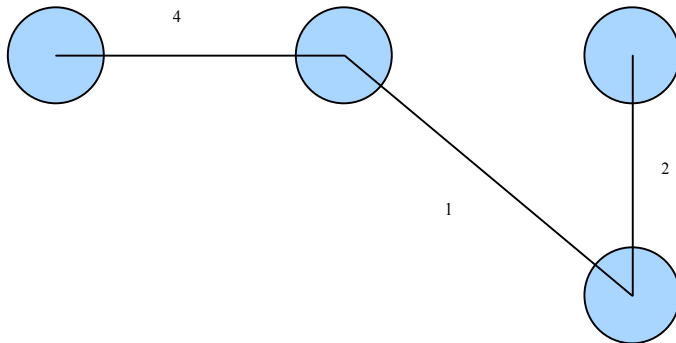
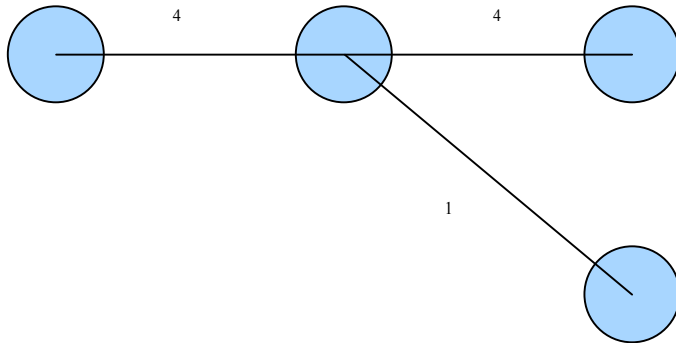
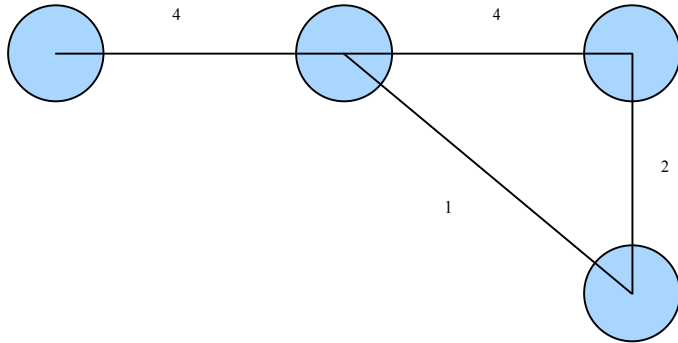


Solution for Problem 11 Bottleneck Spanning Trees

Solution adapted by ELL from an original by Frank Bellamy

a) The middle graph below is clearly a bottleneck spanning tree of the top graph, since its bottleneck edges have weight four and it is clearly impossible to have a spanning tree of this graph that does not have an edge of at least weight four. The middle graph is not, however, a minimum spanning tree, as the bottom graph is a minimum spanning tree and has a smaller weight than the middle graph.



b) Give an $O(n+e)$ algorithm for finding a bottleneck spanning tree in a weighted graph G .

We begin by defining the following procedure that we will use as a subroutine:

connected(G,b)

this procedure takes a graph and a designated edge b from G , and determines whether or not the subgraph G' is connected, where G' consists of all vertices of G but only those edges of G which have weight $\leq \text{weight}(b)$. This is done by modifying a depth-first search to only consider edges of small enough weight, and seeing if the search reaches all the nodes or not. The running time for this procedure is clearly the same as DFS, namely, $O(n+e)$.

BNST(G)

let b be an edge of median weight in G , let a be an edge of greatest weight less than the median weight in G , and let c be an edge of least weight greater than the median weight in G . The edge b is found in time $O(e)$ using the FastSelect algorithm, and then a simple linear time pass over the edges determines a and c . Thus, this step in total takes time $O(e)$.

let G' be the subgraph of G consisting of all vertices of G and those edges of weight $\leq \text{weight}(b)$ (which is half as many edges as G has). Generating G' can be done very straightforwardly in time $O(n+e)$.

if connected(G,b)

 then if connected(G,a)

 then Return BNST(G')

 else b is a bottleneck edge, so Return ($\text{weight}(b)$)

 else if connected(G,c)

 then c is a bottleneck edge, so Return ($\text{weight}(c)$)

 else begin

 let G'' be the graph formed by creating a single vertex for each connected component of G' . Since it is useful in the next step, during this process we also create a mapping in the form of an array from the vertices of G and G' to the vertices of G'' . This can be done straightforwardly in time $O(n+e)$.

 for each edge in G of weight $\geq \text{weight}(b)$ that connects two vertices in different connected components of G' , an edge is inserted into G'' connecting the two vertices corresponding to the corresponding two connected components. That edge has the same weight as the edge in G . If there is already an edge connecting those two vertices in G'' , then that edge's weight is changed to be the minimum of its current weight and the weight of the edge in G being considered. Since we have a mapping from vertices of G to vertices of G'' , this is straightforwardly takes time $O(e)$. Since G'' contains at most one edge for each edge of G of weight $> \text{weight}(b)$, and no edges for any edge of G of weight $\leq \text{weight}(b)$, G'' contains at most half as many edges as G .

 end

 Return BNST(G'')

Analysis of the running time of BNST

In analyzing the running time, the only tricky part is determining the cost of the recursion. Before addressing that issue we note that each constructed graph above is connected, so the number of vertices in each graph is no more than the number of graph edges plus 1. Accordingly we determine the running time strictly in terms of the number of edges.

Note also that the recursion takes as input either the graph G' or the graph G'' depending on which case occurs. But, either way, as discussed in the algorithm text, the number of edges in the recursion is half the number of the graph G .

Thus, let $T(e)$ be the running time of BNST on a graph with e edges. Thus $T(e) = T(e/2) + O(n+e)$, and since $n < e+1$, we rewrite this as $T(e) = T(e/2) + O(e)$. The solution to this familiar recurrence is $O(e)$.

FINISHING UP

BNST gives us the weight of the bottleneck edge(s). Once we have this value, it is straightforward to run a DFS of G to construct the actual bottleneck spanning tree, considering only edges of weight \leq the weight of the bottleneck edges. Like any standard graph searching algorithm, this takes time $O(n+e)$.