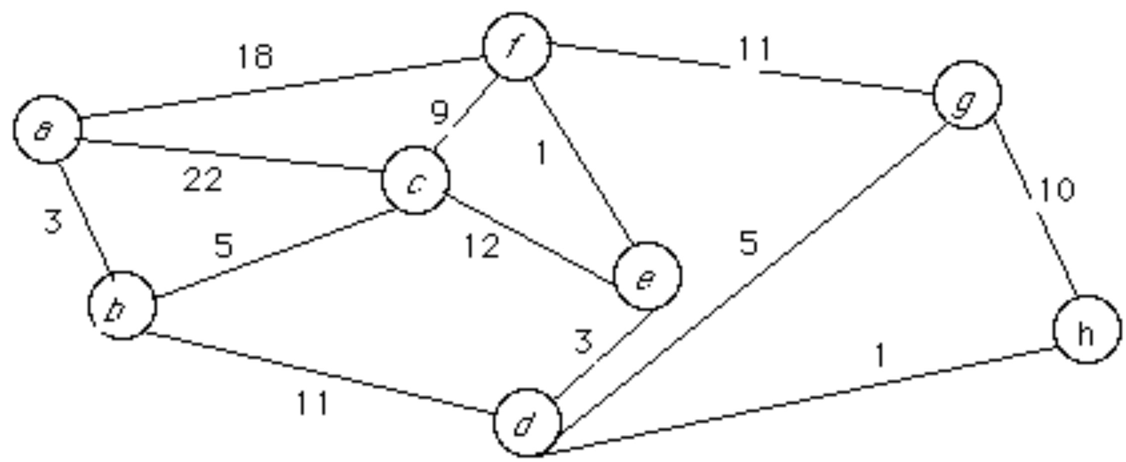
# CS2223
# HW#5

**DUE**: Tuesday , December 2

1. (5 points) Consider the algorithm *Dijkstra** which is dervived from our text's version of *Dijkstra* (on pg. 199) by changing *minimizing* to *maximizing* and by changing *min* to *max*. Prove or give a counterexample to the following
**CONJECTURE**: For any connected graph with positive edge lengths, *Dijkstra** will return the lengths of longest paths from vertex 1 to all other vertices.

2. (10 points) For a graph $G$ and for any pair of vertices $v$ and $x$ in $G$, we define the *bottleneck length* of a path from $v$ to $x$ as the length of the longest edge on the path, and a *bottleneck shortest path* from $v$ to $x$ as a path from $v$ to $x$ of minimum bottleneck weight. For example, for the graph




the path $(a, b, c, f, e, d, g)$ from $a$ to $g$ has bottleneck weight 9, the path $(a, f, e, d, g)$ from $a$ to $g$ has bottleneck weight 18, and the path $(a, b, d, h)$ from $a$ to $h$ has bottleneck weight 11. The *bottleneck shortest path* from $a$ to $h$ is $(a, b, c, f, e, d, h)$.
**THEOREM**: For any weighted connected graph $G$, there is a subgraph $T$ of $G$ which is a tree such that for every pair of vertices $v$, $x$ in $G$, $T$ contains a bottleneck shortest path from $v$ to $x$.

(*a*) Describe an algorithm (precise pseudo-code will suffice) which will accept as input a weighted connected graph $G$, and will construct a subgraph of $G$ which is a tree and which contains a *bottleneck shortest path* from any vertex of $G$ to any other vertex of $G$. Your algorithm's execution time should be in $O(a^2)$, where $a$ is the number of edges of $G$.

(*b*) Prove that your algorithm works correctly.

3. (12 points) Write, test and time the following two programs.
(*a*) a straightforward implementation *quicksort*, either from the text or derived from the version presented in class or elsewhere.
(*b*) a modification of the above program where "sufficiently small" lists are sorted by a sorting routine which is faster for smaller lists.Determine the optimal size of "sufficiently small", that is, determine the optimal value of $s$ such that all lists of size greater than $s$ should be Quicksorted and all lists of size less than or equal to $s$ should be sorted using the other routine. How much faster is your program after this modification?