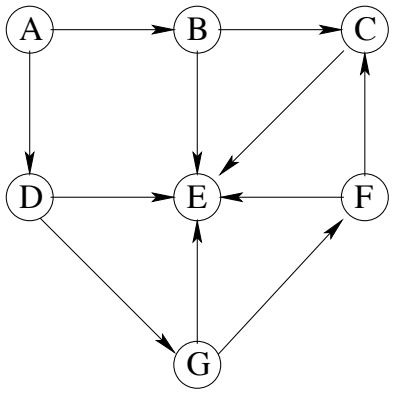
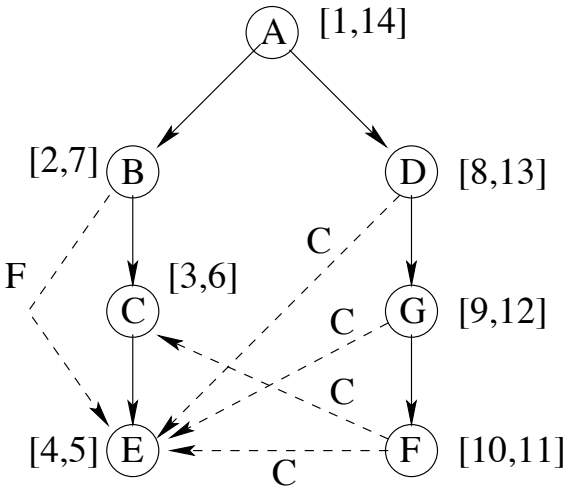


Figure 1: Graph for Problem 1. **Assume for all figures that the adjacency lists are ordered so that vertices are ordered in alphabetical order in an adjacency list.** For example, the adjacency list of A has the edge to B before the edge to D.

1. DFS Execution

Draw the DFS tree of the graph in Figure 1 that will result if the first call to DFS is on vertex A and label each vertex with its pre(in)- and post(out)-order numbers. As per our standing convention, draw tree edges as solid lines, and back, cross, and forward edges as dashed lines. Put a 'B', 'C', or 'F' on each dashed line according to whether it is a back, cross, or forward edge.



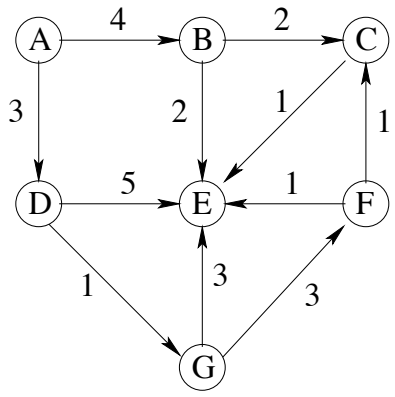


Figure 2: Graph for Problem 2.

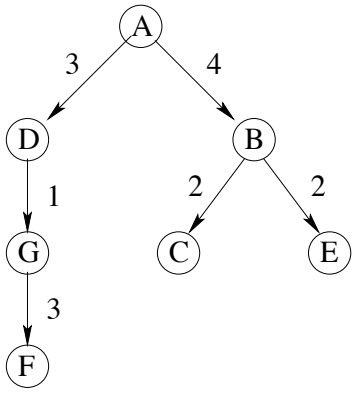
2. Dijkstra Execution

Execute Dijkstra's algorithm on the graph of Figure 2 starting at vertex A. If there are any ties, the vertex with the lower letter comes first.

(a) List the vertices in the order in which they are deleted from the priority queue and for each the shortest distance from A to the vertex.

A,0 D,3 B,4 G,4 C,6 E,6 F,7

(b) Draw the shortest paths tree that results



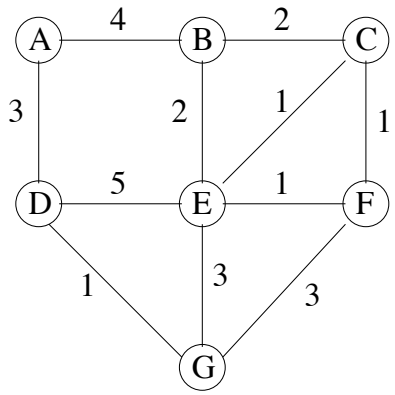


Figure 3: Graph for Problem 3.

3. Prim and Kruskal Executions

(a) Execute Prim's algorithm on the graph of Figure 3 starting at vertex A. If there are any ties, the vertex with the lower letter comes first. List the edges in the order in which they are added to the tree.

(A,D) (D,G) (E,G) (C,E) (C,F) (B,C)

(b) Execute Kruskal's algorithm on the graph of Figure 3 starting at vertex A. Assume that equal weight edges are ordered lexicographically by the labels of their vertices assuming that the lower labeled vertex always comes first when specifying an edge, e.g. (C, E) is before (C, F) which in turn is before (D, G). List the edges in the order in which they are added to the developing forest.

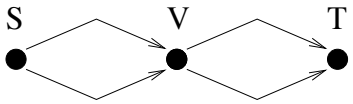
(C,E) (C,F) (D,G) (B,C) (A,D) (E,G)

4. True/False

For each statement below, say whether it is true or false, and a one sentence and/or one picture explanation.

(a) In an undirected graph, we write $s \sim_k t$ if and only if there are k vertex disjoint paths between vertices s and t. Is $\sim_k$ an equivalence relation for all choices of k?

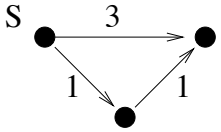
FALSE: $S \sim_k V$ and $V \sim_k T$ but $S \not\sim_k T$.



(b) $T(n) = 4T(n/2) + \frac{n}{\log n}$ is solvable with the Master Theorem.

TRUE: $\frac{n}{\log n} = O(n^{(\log_2 4=2)-\epsilon})$ so the root dominated case applies.

(c) Suppose T is a shortest paths tree for Dijkstra's algorithm. After adding $c > 0$ to every edge in the graph, T is still a shortest paths tree for the modified graph.

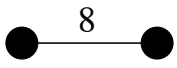


FALSE: Adding 2 changes the SP tree

(d) It is possible to find a heaviest weight path of exactly k edges from s to each vertex in a directed graph in $O(kE)$ time.

TRUE: k iterations of the Bellman-Ford inner loop with the negative of the original weights.

(e) The heaviest edge in a graph cannot belong to a minimum spanning tree.



FALSE: Clearly the one spanning tree has the largest edge

5. Recurrence Equation

Give a Θ characterization of the function $T(n)$ that satisfies the recurrence

$$T(n) = T(2n/3) + (\log_2 n)^2$$

Prove your answer correct.

Unwinding the recurrence yields by induction:

$$\begin{aligned} T(n) &= T((2/3)n) + (\log_2 n)^2 \\ &= T((2/3)^2 n) + (log_2(2/3)n)^2 + (\log_2 n)^2 \\ &= \dots \\ &= T((2/3)^k n) + \Sigma_{i=0}^{k-1}(log_2(2/3)^i n)^2 \end{aligned}$$

The recurrence bottoms out when $(2/3)^k n = O(1)$ which occurs when $k = \log_{3/2} n$. So we have:

$$\begin{aligned} T(n) &= \Sigma_{i=0}^{\log_{3/2} n - 1}(\log_2 n + i(\log_2 2/3))^2) \\ &\leq \Sigma_{i=0}^{\log_{3/2} n - 1} \log_2 n \\ &= O((\lg n)^3) \end{aligned}$$

and

$$\begin{aligned} T(n) &= \Sigma_{i=0}^{\log_{3/2} n - 1}(\log_2 n + i(\log_2 2/3))^2) \\ &\geq \Sigma_{i=0}^{(\log_{3/2} n)/2 - 1}(\log_2 n + i(\log_2 2/3))^2 \\ &\geq \Sigma_{i=0}^{(\log_{3/2} n)/2 - 1}(\log_2 n + (\log_{3/2} n)(\log_2 2/3)/2)^2 \\ &= \Sigma_{i=0}^{(\log_{3/2} n)/2 - 1}((\log_2 n)/2)^2 \\ &= \Omega((\lg n)^3) \end{aligned}$$

Therefore $T(n) = \Theta(\lg^3 n)$.

6. PERT Diagrams

A PERT diagram is a business standard for planning and charting a multi-task project. Each task to be performed is modeled as a vertex. The longest and shortest time, $short(v)$ and $long(v)$, are given for each task v. In addition, there are dependencies between the tasks modeled as directed edges, where $v \to w$ if task v must precede task w. For example, in a construction project the electrical and plumbing cannot be done until the framing has been completed.

Given a directed graph for a PERT chart, along with the *short* and *long* values for each vertex, give an algorithm that first checks that the diagram is acyclic, and then if so, gives the most optimistic (shortest) and most pessimistic (longest) completion times for the entire project.

(a) Run a DFS on the graph and make sure there are no backedges. Recall from lecture that a directed is cycle free if and only if a DFS of the graph has no backedges.

(b)
$optimistic \leftarrow 0$
$pessimistic \leftarrow 0$
For each vertex v in topological order **do**
{ **if** $indegree(v) = 0$ **then**
{ $longest(v) \leftarrow long(v)$
$shortest(v) \leftarrow short(v)$
}
else
{ $longest(v) \leftarrow \max_{w \to v}(longest(w)) + long(v)$
$shortest(v) \leftarrow \max_{w \to v}(shortest(w)) + short(v)$
}
if $outdegree(v) = 0$ **then**
{ $optimistic \leftarrow \max(shortest(v), optimistic)$
$pessimistic \leftarrow \max(longest(v), pessimistic)$
}
}

(c) The algorithm clearly takes $O(V + E)$ time for all aspects. An easy proof by induction shows that $shortest(v)$ is the shortest schedule for completing v and $longest(v)$ is the longest schedule. The topological sort guarantees that preceding values of these quantities are available when computing the values for v. *optimistic* and *pessimistic* are the best of these values over all sink nodes and so the times for the shortest and longest completion times.

7. Minimum Weight Cycle

Consider a postively weighted directed graph. Design the most efficient algorithm you can for finding a minimum weight simple cycle in the graph. Be sure to prove that your algorithm is correct and to argue carefully its running time.

Consider a minimum weight simple cycle and consider any edge $t \rightarrow s$ within it. The portion of the cycle from s to t is a simple path and it must be of minimum cost: if it were not then one could concatenate the better path from s to t with the edge $t \rightarrow s$ and obtain a cycle of lesser cost, a contradiction.

So we have shown that it certainly suffices to compute the shortest path from s to each vertex t and then check the cost of the cycle this path makes with the edge $t \rightarrow s$ (if it exists). By checking over all possible sources s and all edges to s we are guaranteed of seeing a minimum cycle at least once. In fact we see a cycle once for every edge in it. In summary we have the pseudo-code.

$best \leftarrow \infty$
For each vertex s **do**
{ Run Disjkstra's algorithm from s (distances from s in array $dist$)
For each edge $t \rightarrow s$ **do**
$best \leftarrow \min(best, weight(t \rightarrow s) + dist[t])$
}

The algorithm clearly takes $O(V(V+E)logV)$ time if one uses the heap implementation of Dijkstra, or $O(V^3)$ time if one uses the list implementation of the priority queue. Space is $O(V+E)$.