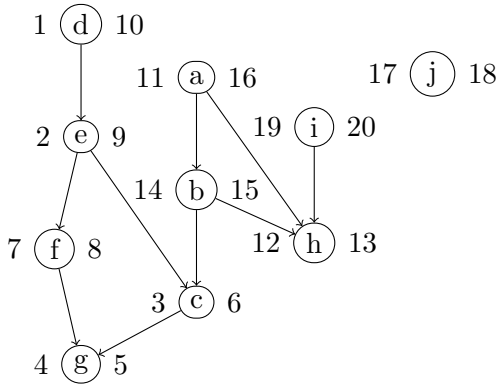
Problem 1.



The drawing above shows a directed acyclic graph (DAG) $G = (V, E)$, on which the depth-first-search-based topological-sort algorithm has been run. Each node is labeled with its discovery time and its finish time during the DFS. Note that the drawing above is *not* the layout from the DFS forest, and in this run ties were *not* always broken in alphabetical order.

(a) List the nodes in the topological order output by the algorithm (given this DFS outcome).

(b) Reconstruct the DFS forest. State the parent of each node in the forest. Write "none" for each root.

(c) Which edges are forward edges? Back edges? Cross edges?

(d) Optionally, draw the DFS forest.

Remember to draw the nodes so that the roots are ordered left-to-right by discovery time, and likewise for the children of each node.

(continued)

Problem 1 answer

(a) Here's a comma-separated list of the nodes in the topological order as computed by the algorithm:

$$i, j, a, b, h, d, e, f, c, g$$

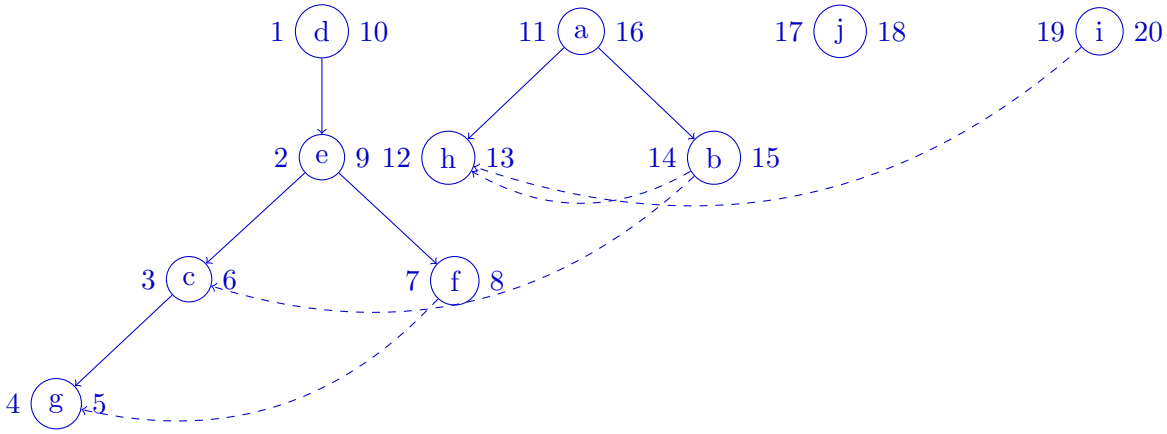
(b)

node:	parent
a:	*none*
b:	a
c:	e
d:	*none*
e:	d
f:	e
g:	c
h:	a
i:	*none*
j:	*none*

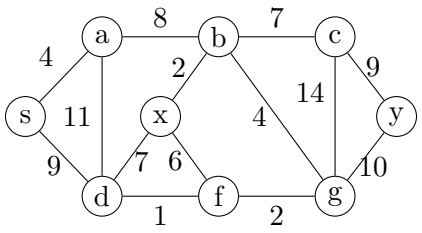
(c) Here are comma-separated lists of the forward, back, and cross edges.

Forward edges:	*none*
Back edges:	*none*
Cross edges:	$(f, g), (b, c), (b, h), (i, h)$

Problem 1(d) answer (OPTIONAL!)



Problem 2. Manually simulate Prim's algorithm on the graph G below, starting from source s. For each of the eight iterations of the algorithm, list the edge that the algorithm chooses in that iteration.



Problem 2 answer For each of the eight iterations of the algorithm, here is the edge that the algorithm chooses:

Iteration 1: (s, a)
Iteration 2: (a, b)
Iteration 3: (b, x)
Iteration 4: (b, g)
Iteration 5: (g, f)
Iteration 6: (f, d)
Iteration 7: (b, c)
Iteration 8: (c, y)

Problem 3. Consider the following conjecture and attempted proof.

Conjecture 1. *Let G be any connected edge-weighted graph G with at least two vertices. Let v be any vertex, and, among edges leaving v, let (v, w^*) be one of minimum weight. Suppose every other edge incident to v has strictly larger weight. Then (v, w^*) is in* every *minimum spanning tree of G.*

Proof (long form).

1. Consider any such graph G, vertex v, and edge (v, w^*),
2.1. Consider any spanning tree T of G that does not contain (v, w^*).
2.2. Since T is connected and contains all vertices, T contains some edge incident to v.
2.3. Let (v, w') be such an edge. Note that $w' \neq w^*$ as (v, w^*) is not in T.
2.4. Let $T' = \{(v, w^*)\} \cup T \setminus \{(v, w')\}$ be obtained from T by removing (v, w') and adding (v, w^*).
2.5. Then $\mathsf{wt}(v, w^*)$ is strictly less than $\mathsf{wt}(v, w')$. That is, $\mathsf{wt}(v, w^*) < \mathsf{wt}(v, w')$.
2.6. So $\mathsf{wt}(T') = \mathsf{wt}(T) + \mathsf{wt}(v, w^*) - \mathsf{wt}(v, w') < \mathsf{wt}(T)$.
2.7. And by Observation 5 of Lecture Note 5 (page 26), T' is a spanning tree.
2.8. By the previous two steps, the weight of spanning tree T' is strictly less than the weight of T.
2.9. So T is not a minimum spanning tree.
3. By Block 2, every spanning tree of G that doesn't contain (v, w^*) isn't a minimum spanning tree.
4. So every minimum spanning tree of G contains (v, w^*). □

(a) Is the reasoning in the proof correct?

(b) If not, what is the first step that makes an assertion that is not necessarily true, given the assumptions?

(c) What exactly is the mistake in the justification given in that step?

(continued)

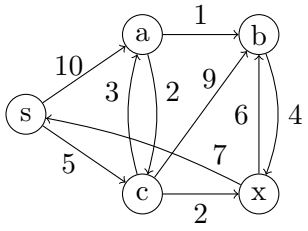
Problem 3 answer

(a) No, the reasoning in the proof is not correct.

(b) Step 2.7

(c) You have to carefully choose the edge to remove. In this proof, one must remove an edge that is in the path from v to w* in the original spanning tree. If one does not carefully choose this edge, then the resulting graph could potentially not be a spanning tree (an example of choosing such an edge and the result not being a spanning tree can be seen in LN 5 8.1 Figure Under Conjecture 3).

Problem 4.



(a) Simulate Dijkstra's algorithm on the directed graph G above with source s. In each iteration of the algorithm, for what vertex v does the algorithm set $d[v]$ and what does it set $d[v]$ to?

(b) Let G' be the graph obtained from G (shown above) by reducing the weight of edge (a, c) to -10. What is the array d of distances returned by Dijkstra's algorithm given input G' with source s? Show $d[s]$, $d[a]$, $d[b]$, $d[c]$, and $d[x]$.

This graph has a negative edge weight, so Dijkstra's algorithm might not output correct distances!

Problem 4 answer

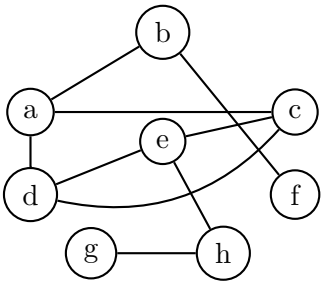
(a) For each of the four iterations of the algorithm, here is the vertex v for which $d[v]$ is set, and the value it is set to:

In iteration 1, the algorithm sets $d[c] = 5$
In iteration 2, the algorithm sets $d[x] = 7$
In iteration 3, the algorithm sets $d[a] = 8$
In iteration 4, the algorithm sets $d[b] = 9$

(b) Here is the array d that would be returned by the algorithm given the modified input:

$d[s] = 0$
$d[a] = 8$
$d[b] = 9$
$d[c] = 5$
$d[x] = 7$

Problem 5. You have been hired to clean up the streets of Boston using your street sweeper. In order to do the job for the lowest cost, you want to devise a way to sweep each street once in each direction. That is, for each street, you go up the street once and down the street once to sweep each side of the road. All roads are two way. To find a good route, you formally define the following Street Sweeper problem:



input: a connected, undirected graph $G = (V, E)$

output: a (not necessarily simple) cycle C that traverses each edge (u, w) once in each direction[1]

For this problem you are to define a correct algorithm for Street Sweeper by adding just a few lines to the code for depth-first search given in the template in the next page for Part (c) below.

(a) First, give a correct output for the graph above. Please start your cycle at node a. Give your cycle as a comma-separated list of directed edges as described in the footnote below. (It may help to first do a DFS, starting at a, and draw the DFS tree.)

Please validate your cycle using the checker at
https://repl.it/repls/WeeklyMiniatureNumericalanalysis.

(b) Explain clearly in just a few English sentences exactly how you modify DFS to obtain your algorithm. First state the high-level idea, then give more details. *Hint: in a DFS of an undirected graph, each edge occurs exactly once as a tree edge or once as a forward edge.*

(c) Define your algorithm in pseudocode, using the pseudo-code in the answer template as a starting point. Your algorithm can just *print* the edges in the cycle in the right order, it doesn't need to collect and return them.

(d) Give the best bound you can on the worst-case running time of your algorithm in terms of $n = |V|$ and $m = |E|$. Explain carefully why the bound you give holds for all inputs. You may use (without reproving them) any facts about DFS that we've observed in class or the lecture notes. Your algorithm should run in time linear in the graph size, $\Theta(|V| + |E|)$.

(e) Implement your algorithm and submit it to the Hacker-Rank Contest. Your algorithm should pass all the tests for that challenge. State the username that you used for your Hacker-Rank submission.

[1]Formally, the cycle C is specified as a sequence of *directed* edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{\ell-1}, v_\ell), (v_\ell, v_1)$. Each edge must start with the vertex that the edge before ends with, except the *first* edge must start with the vertex that the last edge ends with. Here, C should contain each edge $(u, w) \in E$ exactly twice: once as (u, w) and once as (w, u).

Problem 5 answer

(a)

$(a,b),(b,f),(f,b),(b,a),(a,c),(c,e),(e,d),(d,a),(a,d),(d,c),(c,d),(d,e),(e,h),(h,g),(g,h),(h,e),(e,c),(c,a)$

(b) You want to traverse each tree edge twice which happens with DFS, but you also need to traverse all non-tree edges (in an undirected graph, there are only forward edges) as well to sweep all graph edges. To do this, once you reach a vertex, you should sweep all edges out of that vertex to vertices that have already been visited and back before continuing the DFS.

(c) StreetSweeper($G = (V, E)$)

1. visited $= \emptyset$
2. ancestors $= \emptyset$
3. def DFS1(u):

4.1. add u to visited and to ancestors
4.2. for each neighbor w of u:
4.3. if w is not in visited:
4.4. print "tree edge:" (u, w)
4.5. DFS1(w)
4.6. print "tree edge:" (w, u)
4.7. else if w is not in ancestors:
4.8. print "forward edge:" (u, w)
4.9. print "forward edge:" (w, u)
4.10. remove u from ancestors

5. for each vertex v in V:
6. if v is not in visited: DFS1(v)

Please color the text you add blue.

(d) We iterate over the v vertices at most v times (outer loop), and through the inner and outer loops, we are visiting each edge twice in order to sweep it. Thus the complexity of the algorithm is $\Theta(2|E|+|V|)$ $= \Theta(|V| + |E|) = \Theta(n + m)$ which is linear in graph size. Using sets makes checking constant time.

(e) My username for my Hacker-Rank submission is sisolta75 .