

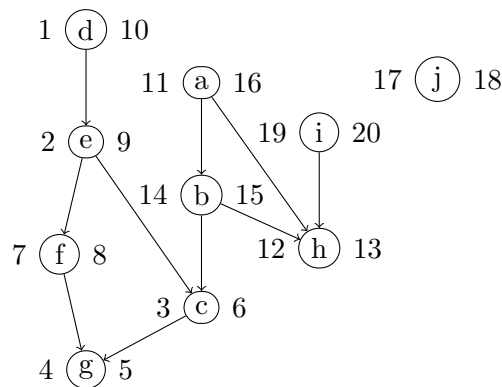
General instructions. Answer the problems given in the rest of this document. Edit this LaTeX template to add your answers, as described in the file named `00_README.txt`, then compile it to produce `homework_6.pdf`.

Don't know? Say so. If you don't know the answer to a problem or any part of the problem, we encourage you to simply write “*I don't know*” instead of giving an answer. (If you write “I don't know” you can also add any questions you might have.)

Gentle reminder: homework is for learning. As noted in [Lecture Note 1](#), engaging fully with the homeworks is the best way to learn the material. *The primary role of homeworks is to help you learn the material, and the primary role of feedback on homeworks is to help you improve your ideas.*

Please try to explain your ideas as clearly as you can. The clearer your explanations are, the more useful the feedback on them can be.

Collaboration and citation policy. If you are doing this homework as part of a course, follow the course's collaboration and citation policy. In any case, at the end of each answer, list the people you collaborated with, and cite every source that your answer uses ideas from (other than the lecture notes and the sources listed in the lecture notes). If there are none, write “none”.

Problem 1. (*Exercise 6.1*)

The drawing above shows a directed acyclic graph (DAG) $G = (V, E)$, on which the depth-first-search-based topological-sort algorithm from Section 6.3.5 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.

- List the nodes in the topological order output by the algorithm (given this DFS outcome).
- Reconstruct the DFS forest. State the parent of each node in the forest. Write “none” for each root.
- Which edges are forward edges? Back edges? Cross edges?
- Optionally, draw the DFS forest. If you do this, and your answers for Parts (b) and (c) are substantially incorrect, you may get partial credit here.

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.

In the Python code for the notes (<https://github.com/nealeyoung/algs101/tree/main/src/code>), the file `graph_traversal/depth_first_search.py` has Python code for DFS and the DFS-based topological-sort algorithm.

(continued)

Problem 1 answer

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

★...★

- (b) node: parent

a: ★...★

b: ★...★

c: ★...★

d: ★...★

e: ★...★

f: ★...★

g: ★...★

h: ★...★

i: ★...★

j: ★...★

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

Forward edges: ★...★

Back edges: ★...★

Cross edges: ★...★

Problem 1(d) answer (OPTIONAL!)

Problem 1 collaboration and citations

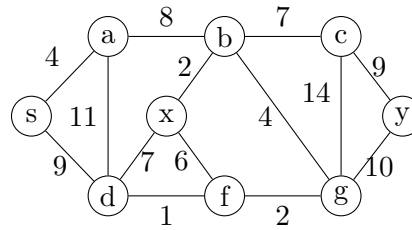
I collaborated on this problem with the following people:

• ★ ⋯ ★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

• ★ ⋯ ★

Problem 2. (*Exercise 5.11*) Manually simulate Prim's algorithm (from the lecture notes) 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.



In the Python code for the notes (<https://github.com/nealeyoung/algs101/tree/main/src/code>), the file `greedy_algorithms/min_spanning_tree.py` has Python code for Prim's algorithm.

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

Iteration 1: ★★★★★
 Iteration 2: ★★★★★
 Iteration 3: ★★★★★
 Iteration 4: ★★★★★
 Iteration 5: ★★★★★
 Iteration 6: ★★★★★
 Iteration 7: ★★★★★
 Iteration 8: ★★★★★

Problem 2 collaboration and citations

I collaborated on this problem with the following people:

● ★★★★★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

● ★★★★★

Problem 3. (*Exercise 5.12*) 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 $\text{wt}(v, w^*)$ is strictly less than $\text{wt}(v, w')$. That is, $\text{wt}(v, w^*) < \text{wt}(v, w')$.
- 2.6. So $\text{wt}(T') = \text{wt}(T) + \text{wt}(v, w^*) - \text{wt}(v, w') < \text{wt}(T)$.
- 2.7. And by Observation 5.8 (Section 5.9.1), 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) ★ ⋯ ★

(b) ★ ⋯ ★

(c) ★ ⋯ ★

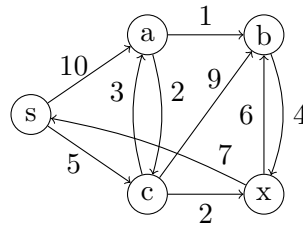
Problem 3 collaboration and citations

I collaborated on this problem with the following people:

- ★ ⋯ ★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

- ★ ⋯ ★

Problem 4. (Exercise 5.8)

- (a) Simulate Dijkstra's algorithm (Section 5.8) 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 some distances may be undefined.

In the Python code for the notes (<https://github.com/nealeyoung/algs101/tree/main/src/code>), the file `greedy_algorithms/Dijkstra.py` has Python code for Dijkstra's algorithm.

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 $d[\star\cdots\star] = \star\cdots\star$

In iteration 2, the algorithm sets $d[\star\cdots\star] = \star\cdots\star$

In iteration 3, the algorithm sets $d[\star\cdots\star] = \star\cdots\star$

In iteration 4, the algorithm sets $d[\star\cdots\star] = \star\cdots\star$

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

$d[s] = \star\cdots\star$

$d[a] = \star\cdots\star$

$d[b] = \star\cdots\star$

$d[c] = \star\cdots\star$

$d[x] = \star\cdots\star$

Problem 4 collaboration and citations

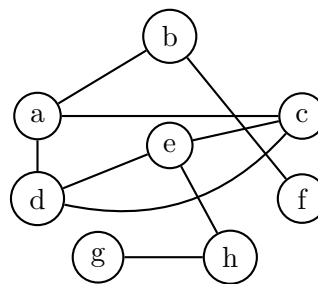
I collaborated on this problem with the following people:

- $\star\cdots\star$

My answers use ideas from the following sources (other than the lecture notes and textbooks):

- $\star\cdots\star$

Problem 5. (Exercise 6.5) 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 direction¹

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 (from Homework 5 Problem 2). 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 that comes with this template:

```
run python3 check_cycle.py "(a, b), (b, f), ...".
```

- (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 the lecture notes. For full credit, 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 for Homework 6 at <https://www.hackerrank.com/nealeyoung-algs101-street-sweeper>. If your algorithm passes all the tests for that challenge, you will get full credit for this part. State the username that you used for your Hacker-Rank submission.

Note that the academic-integrity policy applies to code you submit. You must write all of the code you submit on your own. Don't share code, or write code based on a document shared with any collaborator. You may adapt code provided in this homework or with the notes.

Note: use Hacker-Rank to validate your algorithm before submitting.

In the Python code for the notes (<https://github.com/nealeyoung/algs101/tree/main/src/code>), the file `graph_traversal/depth_first_search.py` has Python code for depth-first search.

¹Formally, the cycle C is specified as a sequence of *directed* edges $(v_1, v_2), (v_2, v_3), \dots, (v_{\ell-1}, v_\ell), (v_\ell, v_1)$. 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)

★...★

(b) ★...★

(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. else if w is not in ancestors:
 - 4.7. print “forward edge:” (u, w)
 - 4.8. 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) ★...★

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

Problem 5 collaboration and citations

I collaborated on this problem with the following people:

• ★ ⋯ ★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

• ★ ⋯ ★