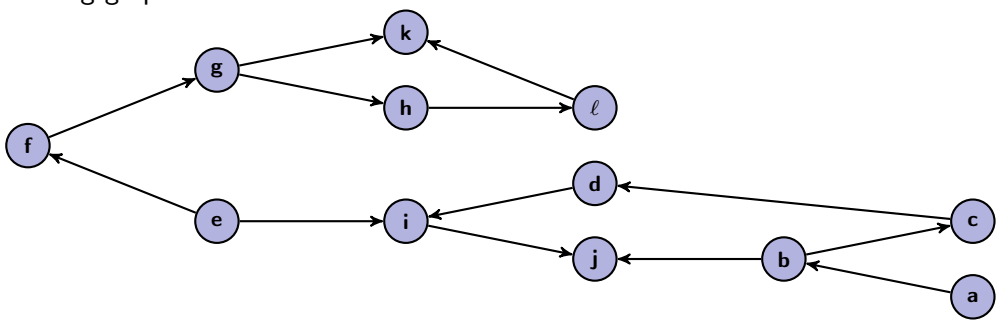
CSE 332: Data Structures and Parallelism

Section 9: Graphs Solutions

0. It Rhymes with Flopological Sort

Consider the following graph:



(a) Does this graph have a topological sort? Explain why or why not. If you answered that it does not, remove the **MINIMUM** number of edges from the graph necessary for there to be a topological sort and carefully mark the edge(s) you are removing. Otherwise, just move on to the next part.

Solution:

Yes, it does. This is a DAG (i.e., it has no cycles).

For the remaining parts, work with this (potentially) new version of the graph.

(b) Find a topological sort of the graph. Do not bother showing intermediary work.

Solution:

There are many. One example is e, f, g, h, l, k, a, b, c, j, d, i.

(c) If this graph represented various tasks in a `ForkJoin` algorithm, what would the work of the algorithm be assuming each individual task is $\Theta(1)$.

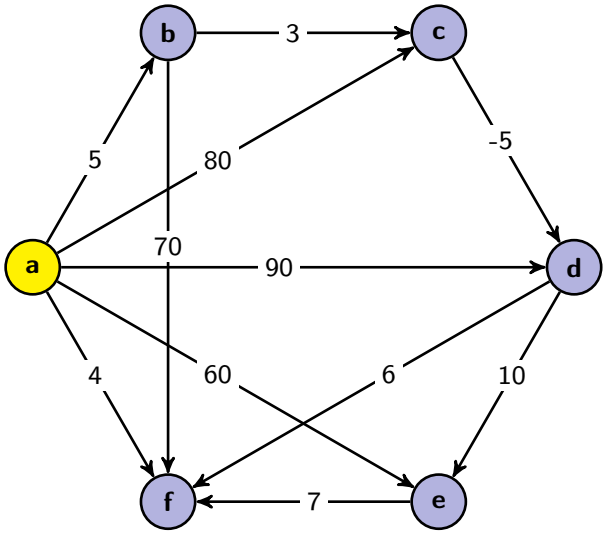
Hint: There are 12 tasks...

Solution:

There are a constant number of tasks; so, the work is $\Theta(1)$.

1. Better Find the Shortest Path Before It Catches You!

Consider the following graph:



(a) Use Dijkstra's Algorithm to find the **lengths** of the shortest paths from **a** to each of the other vertices. For full credit, you must show the worklist at every step, but how you show it is up to you.

Solution:

Vertex	Cost	Done?
a	0	✓
b	~~∞~~ 05	✓
c	~~∞~~ ~~80~~ 08	✓
d	~~∞~~ ~~90~~ 03	✓
e	~~∞~~ ~~60~~ 13	✓
f	~~∞~~ 04	✓

(b) Are any of the lengths you computed using Dijkstra's Algorithm in part (a) incorrect? Why or why not?

Solution:

In this case, no. In general, Dijkstra's Algorithm does not necessarily work correctly with negative edge weights, but here, it actually returns the right result.

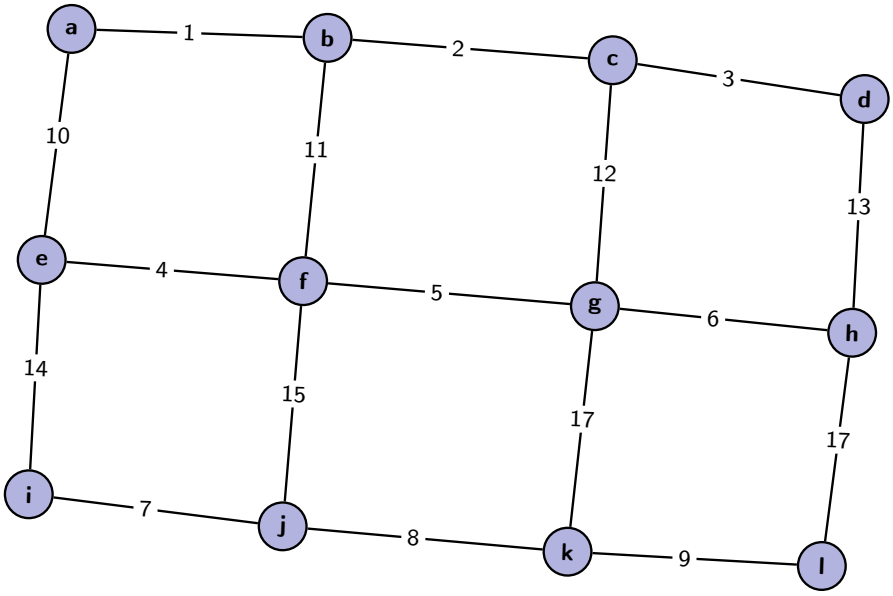
(c) Explain how you would use Dijkstra's Algorithm to recover the actual paths (rather than just the lengths).

Solution:

Keep an extra dictionary which maps nodes to their predecessors. (A predecessor is the node we took an edge from to get to the node.) Then, walk from the target vertex back toward the source vertex using the predecessor map.

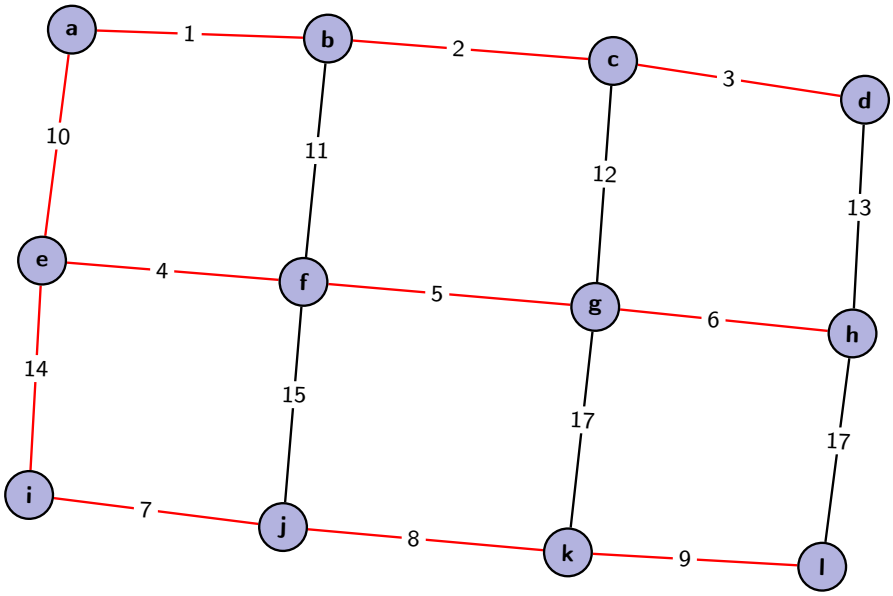
2. LMNST!

Consider the following graph:



(a) Find an MST of this graph using both of the two algorithms we've discussed in lecture. Make sure you say which algorithm you're using and show all iterations of the worklist.

Solution:



- Using Prim's algorithm:

Vertex	Cost	Done?
a	0	✓
b	~~∞~~ 01	✓
c	~~∞~~ 02	✓
d	~~∞~~ 03	✓
e	~~∞~~ 10	✓
f	~~∞~~ ~~11~~ 04	✓
g	~~∞~~ ~~12~~ 05	✓
h	~~∞~~ ~~13~~ 06	✓
i	~~∞~~ 14	✓
j	~~∞~~ ~~15~~ 07	✓
k	~~∞~~ ~~17~~ 08	✓
l	~~∞~~ ~~17~~ 09	✓

- Using Kruskal's algorithm:
  - Sorted Edges:
    (a, b), (b, c), (c, d), (e, f), (f, g), (g, h), (i, j), (j, k), (k, l), (a, e), (b, f), (c, g), (d, h), (e, i), (f, j), (g, k), (h, l)
  - UF Forests:
    - {a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}, {j}, {k}, {l}
    - {a, b, c, d}, {e, f, g, h}, {i, j, k, l}
    - {a, b, c, d, e, f, g, h}, {i, j, k, l}
    - {a, b, c, d, e, f, g, h, i, j, k, l}

(b) Using just the graph, how can you determine if *it's possible* that there are multiple MSTs of the graph? Does this graph have multiple MSTs?

Solution:

A graph can only have multiple MSTs if it has multiple edges of the same weight. This graph has two 17's, but neither of them are used in the MST. So, there's only one MST here.

(c) What is the asymptotic runtime of the algorithms that you used to compute the MSTs?

Solution:

Prim's Algorithm takes $\mathcal{O}(|V|\lg(|V|) + |E|\lg(|V|)$, and Kruskal's Algorithm takes $\mathcal{O}(|E|\lg(|E|))$.