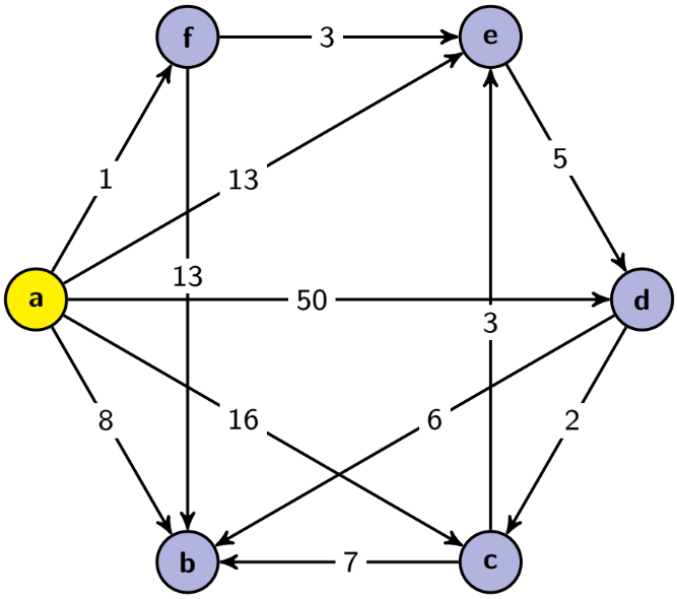
CSE 332: Data Structures and Parallelism

Section 10: Graphs Cont'd, P/NP Solutions

0. Velociraptors

Consider the following graph:



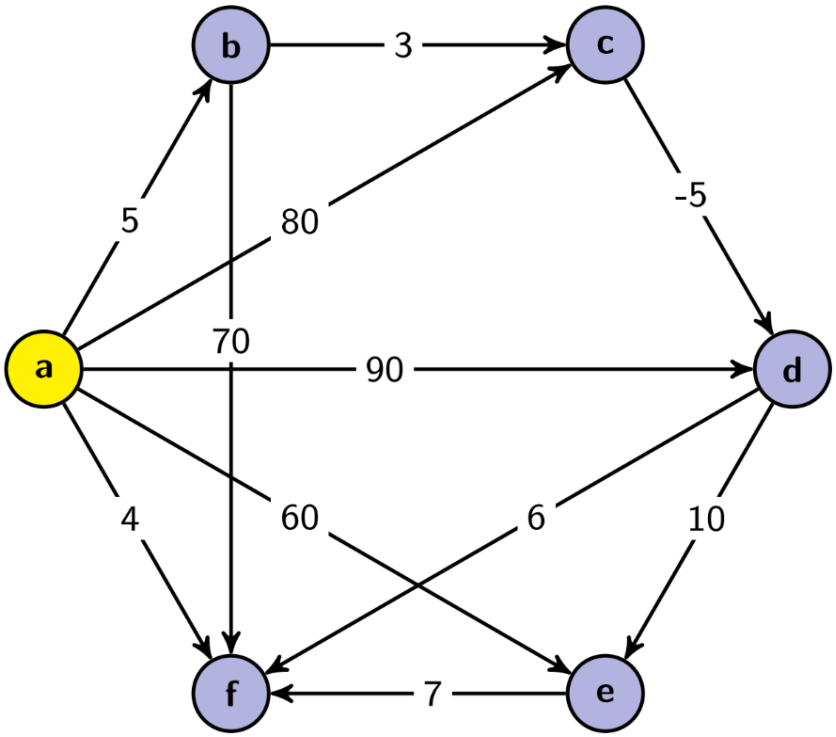
Suppose that you are at **a** and you are planning your escape from a bunch of hungry velociraptors (edge weights represent the expected number of velociraptors you will meet on this path). Run Dijkstra's Algorithm to **find the lengths of the shortest paths** (fewest number of velociraptors met) from **a** to each of the other vertices. Remember to store the path variable and **list the order vertices are added to the known set**.

Vertex	Known	Cost of Path	Path
a	True	0	
b	True	~~∞~~ 8	a
c	True	~~∞~~ ~~16~~ 11	~~a~~ d
d	True	~~∞~~ ~~50~~ 9	~~a~~ e
e	True	~~∞~~ ~~13~~ 4	~~a~~ f
f	True	~~∞~~ 1	a

Order added to known set: **a**, **f**, **e**, **b**, **d**, **c**

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. Show your work at every step.

Vertex	Known	Cost of Path	Path
a	True	0	
b	True	~~∞~~ 05	~~a~~
c	True	~~∞~~ ~~80~~ 08	~~a~~ b
d	True	~~∞~~ ~~90~~ 03	~~a~~ c
e	True	~~∞~~ ~~60~~ 13	~~a~~ d
f	True	~~∞~~ 04	a

Order added to known set: a, f, b, c, d, e

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

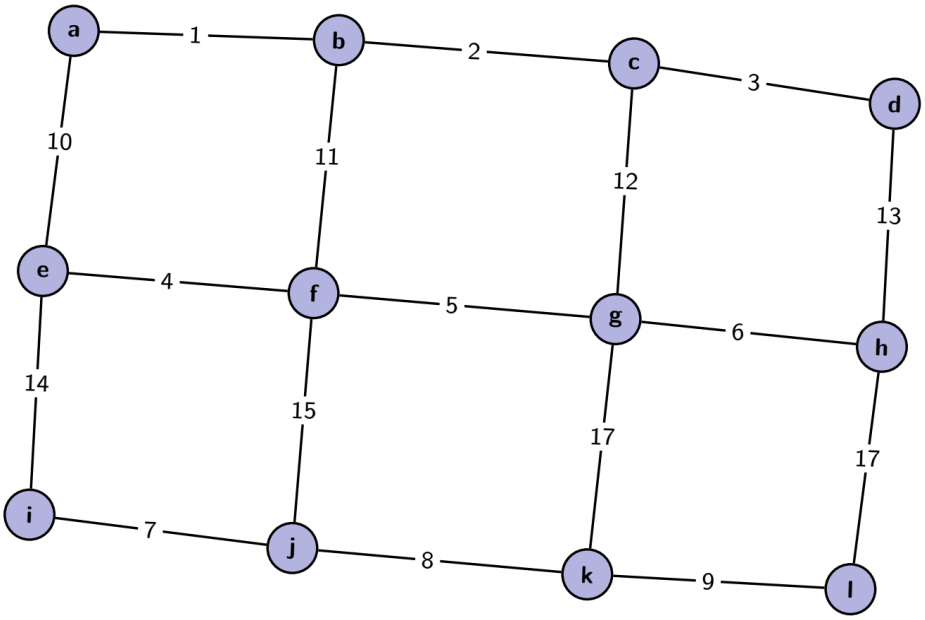
In this case, no. In general, Dijkstra's Algorithm does not necessarily work correctly with negative edge weights, but here, it actually returns the right result.

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

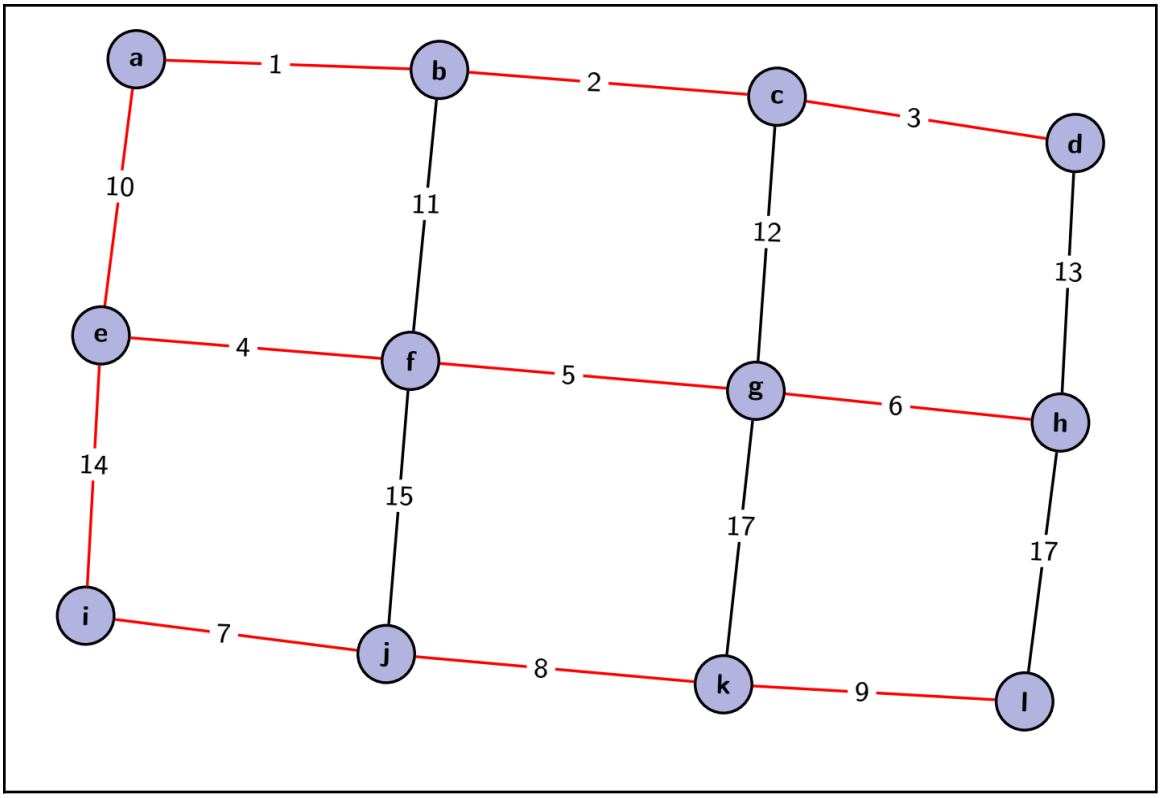
Keep an extra dictionary which maps nodes to their predecessors. (A predecessor is the node we took an edge from to get to the node.) Then, walk from the target vertex back toward the source vertex using the predecessor map.

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 your work.



Using Prim's algorithm:

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

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?

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

3. P, NP, NP-Complete

a) "NP" stands for:

Nondeterministic Polynomial

b) What does it mean for a problem to be in NP?

Given a candidate solution to a decision problem, we can verify whether the solution is correct in polynomial time.

c) For the following problems, circle ALL the sets they (most likely) belong to:

Is there a path of weight at most *k* from one vertex to another vertex in a weighted directed graph?

NP **P** NP-complete None of these

Is there a cycle that visits each edge in a graph exactly once?

NP **P** NP-complete None of these

Will this program run forever?

NP P NP-complete **None of these**

Can we find the prefix sum of an array in parallel using 10 processors?

NP **P** NP-complete None of these

Is there a path that starts and ends at the same vertex that visits every vertex exactly once?

NP P **NP-complete** None of these