CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 32 Apr 8, 2013

DAGs

Lecture Goals

1. HW, last week and next
2. DAG overview
3. Dijkstra's Algorithm
4. Critical Path

Upcoming Homework Assignment

HW #9 **Due: Friday, Apr 12**

DAGs

DAGs are Directed Acyclic Graphs. They are graphs that have directed edges, and from any given node it is not possible to traverse edges and get back to where you started. This HW is about writing algorithms to find the shortest and longest paths in a graph. We'll implement **Dijkstra's algorithm** (which doesn't *require* a DAG), and a **critical path algorithm** (which *does* require a DAG).

Questions on Graph HW?

Questions on the HW that is due later today?

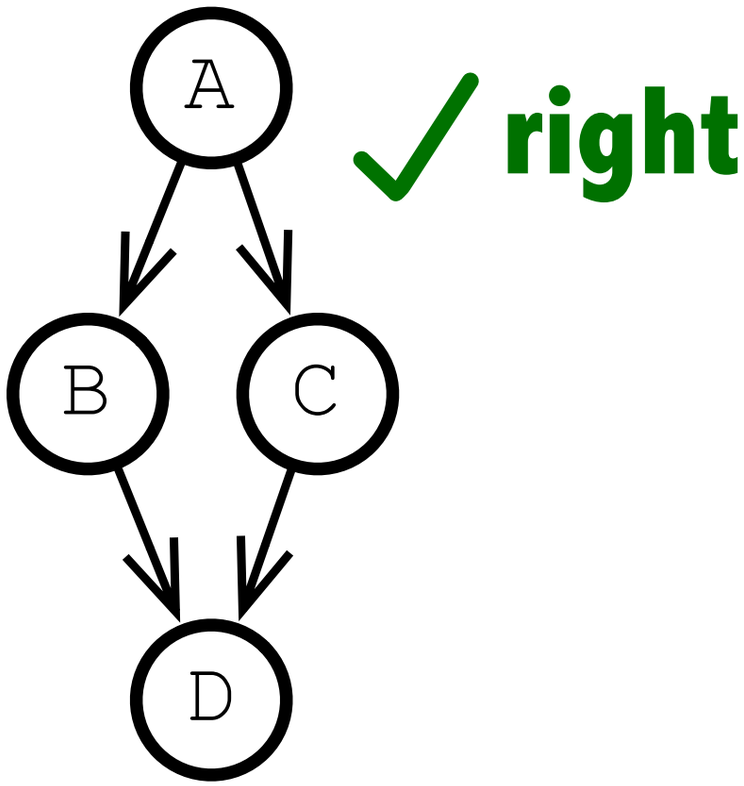
Next week's HW will use last week's code (I mean it this time, since there is no STL graph type).

DAGs

Directed Acyclic Graphs are *everywhere*.

As a reminder: they are directed graphs that don't allow *cycles* (indicated with *back edges*). If you start from any given node, you will not ever be able to follow edges and get back to where you started.

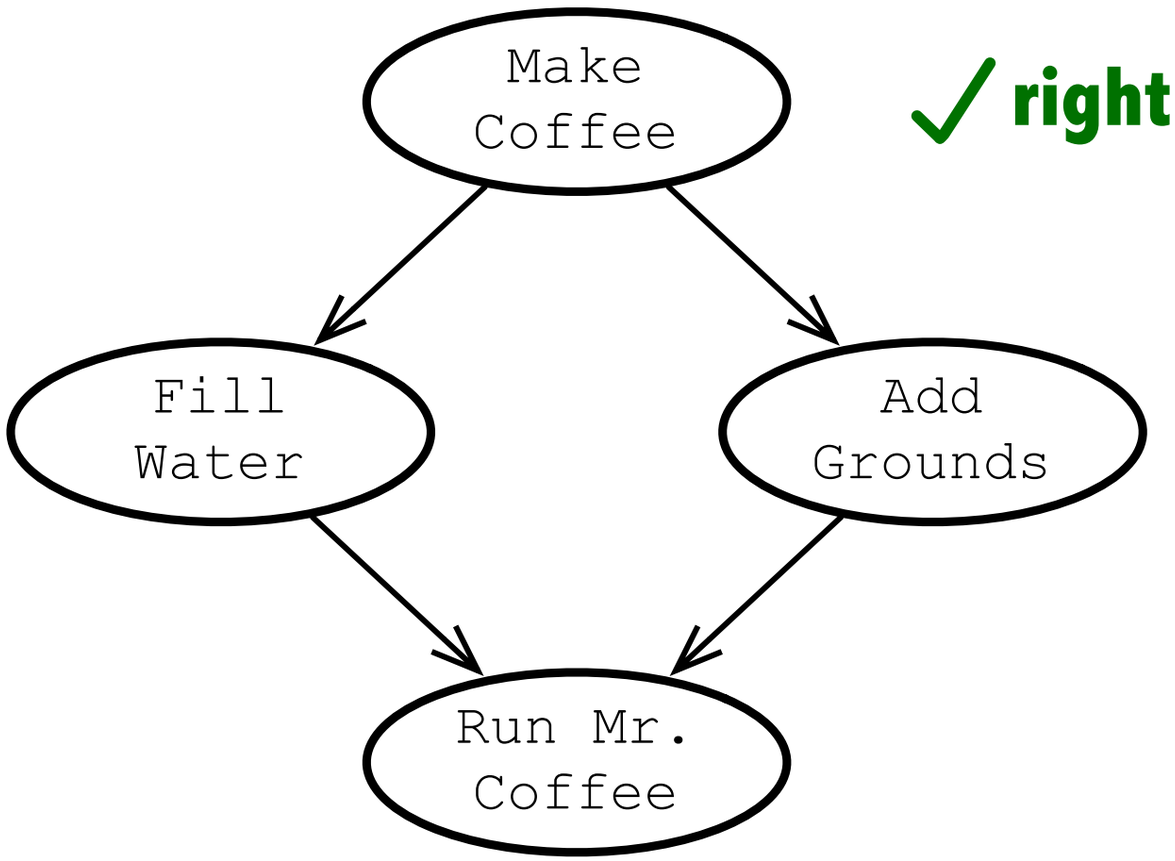
Simple DAG example



✓ **right**

No matter where we start (node A, B, whatever), we can never get back to where we started if we obey the arrow directions.

Coffee

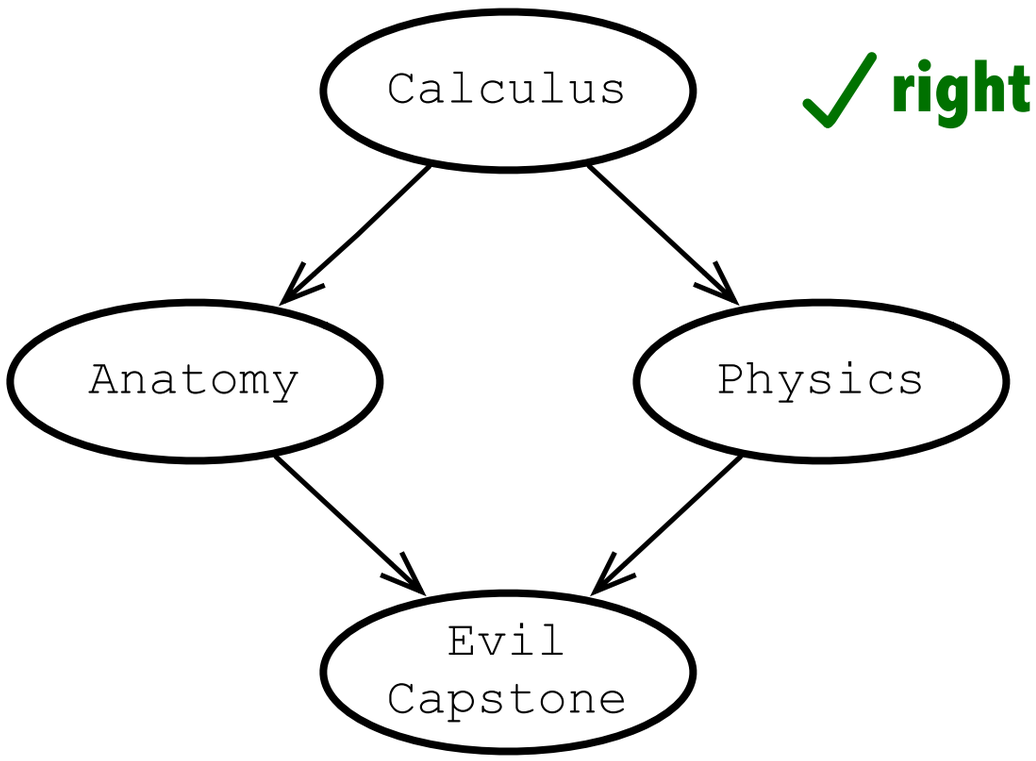


✓ right

The nodes could indicate processes that you have to do; the edges represent the order you have to do them.

We will have to do *all* the tasks, but it doesn't matter if we fill the water or add the grounds first. It only matters that we do it *before* we activate Mr. Coffee.

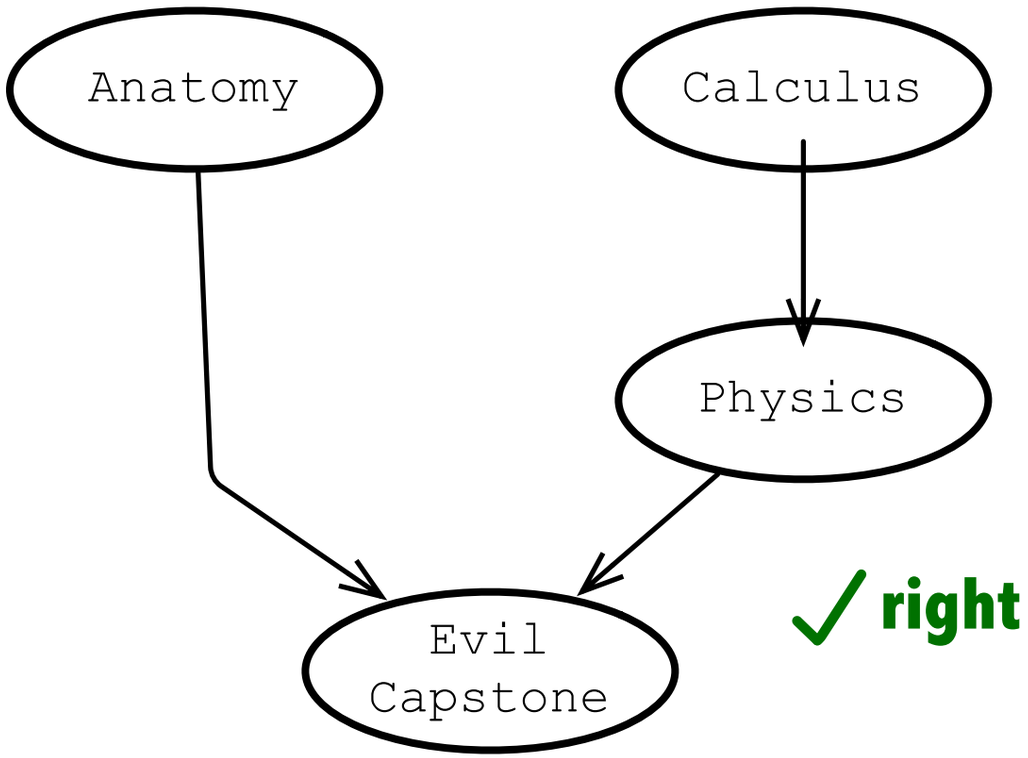
Taking Classes



✓ right

An Evil Scientist major at the University of Doom has to take classes in a certain order. Calculus is a pre-requisite for both Anatomy and Physics.

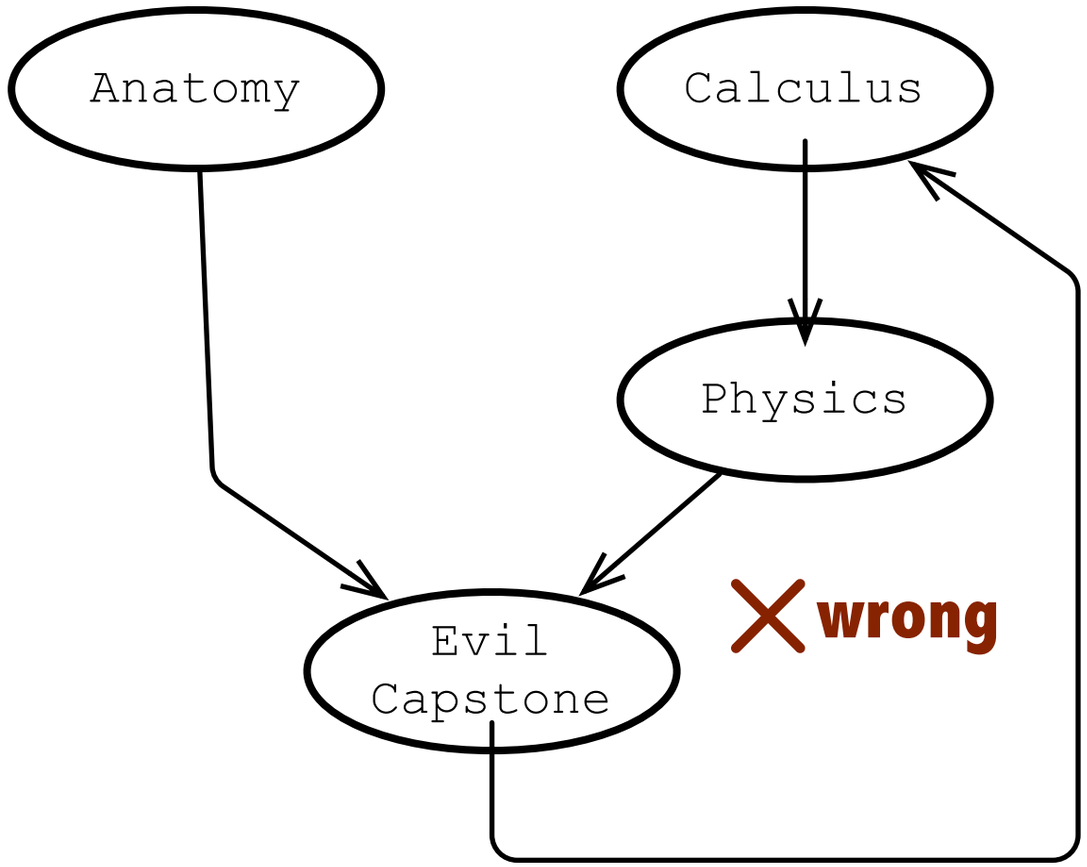
Taking Classes



This would also be correct. We've removed the calculus pre-req for Anatomy. So now we can take anatomy and calculus concurrently, whereas earlier we had to do calc first and *then* anatomy.

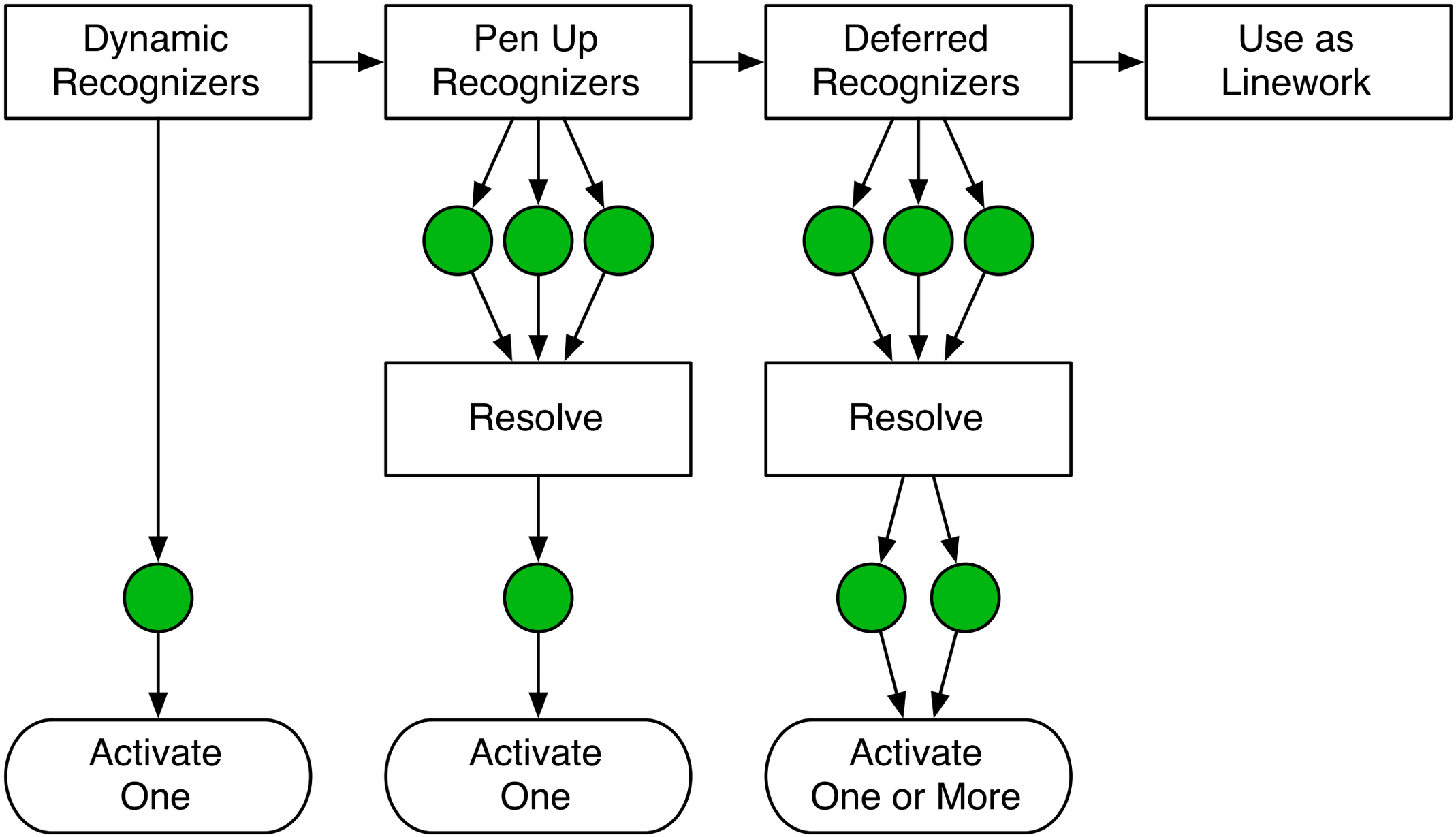
Impossible Pre-Reqs

If we changed the pre-req rules to be like this, we would be at Doom University learning to be evil *forever* (or, we wouldn't get started at all, I'm not sure).

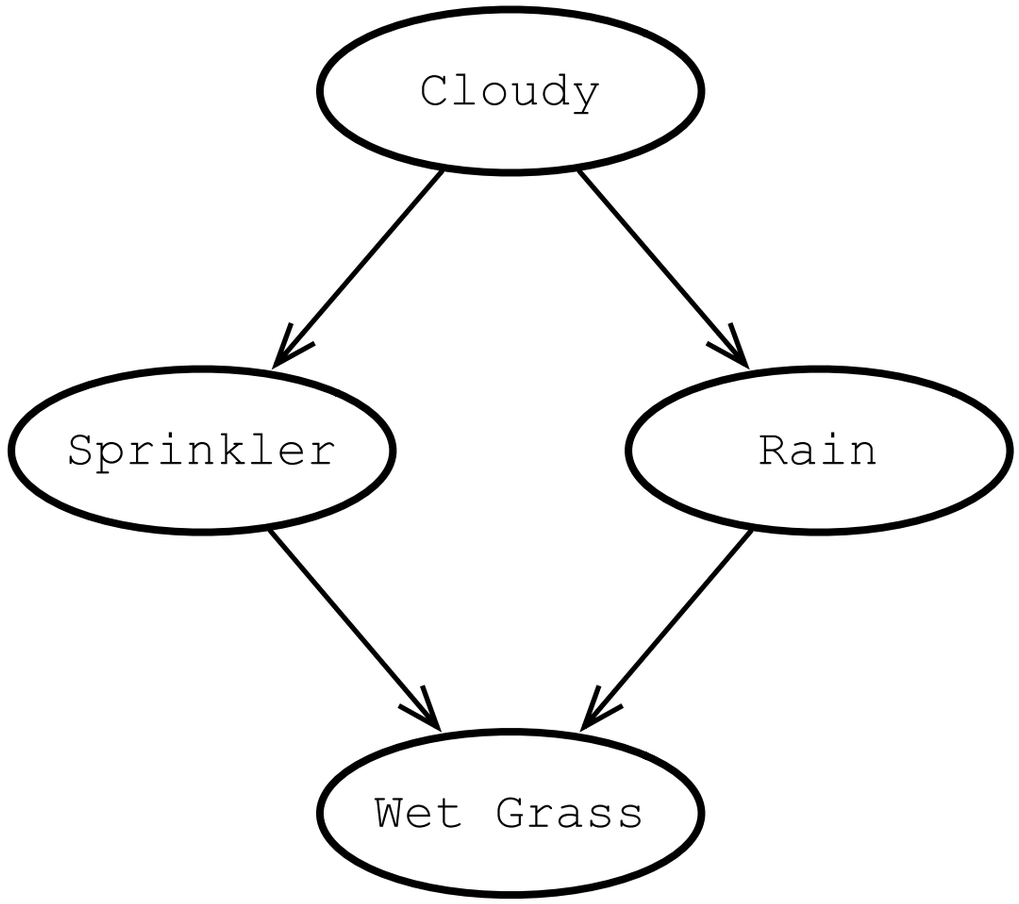


Here there is an endless loop that cycles between three classes, involving Calculus. (This is actually not unlike what it feels like to be an engineering major.)

From my thesis



In Bayesian Modeling

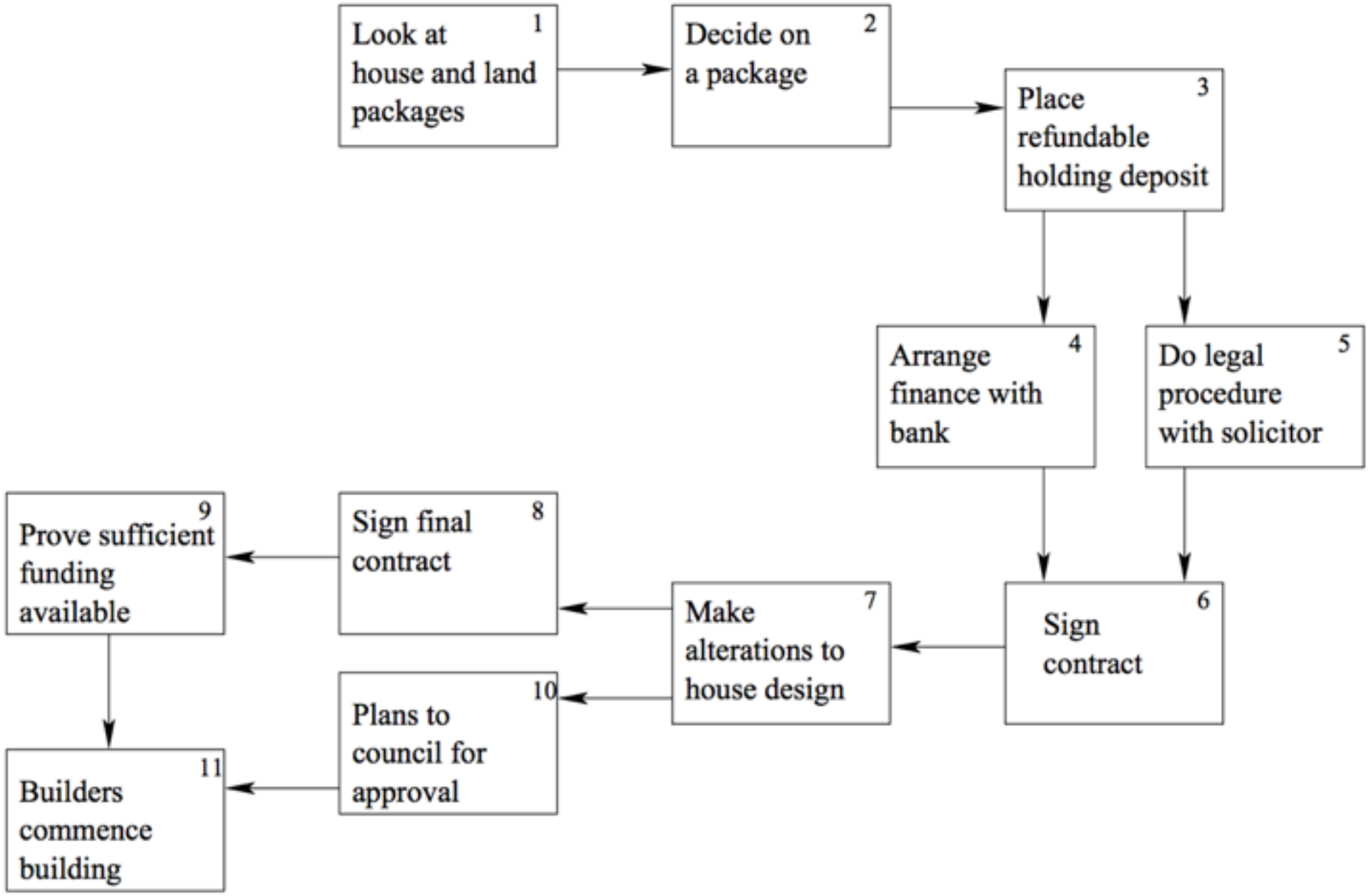


What's the probability that the grass is wet, given that we know it is raining?

What's the probability that it is raining, given that we know that the grass is wet?

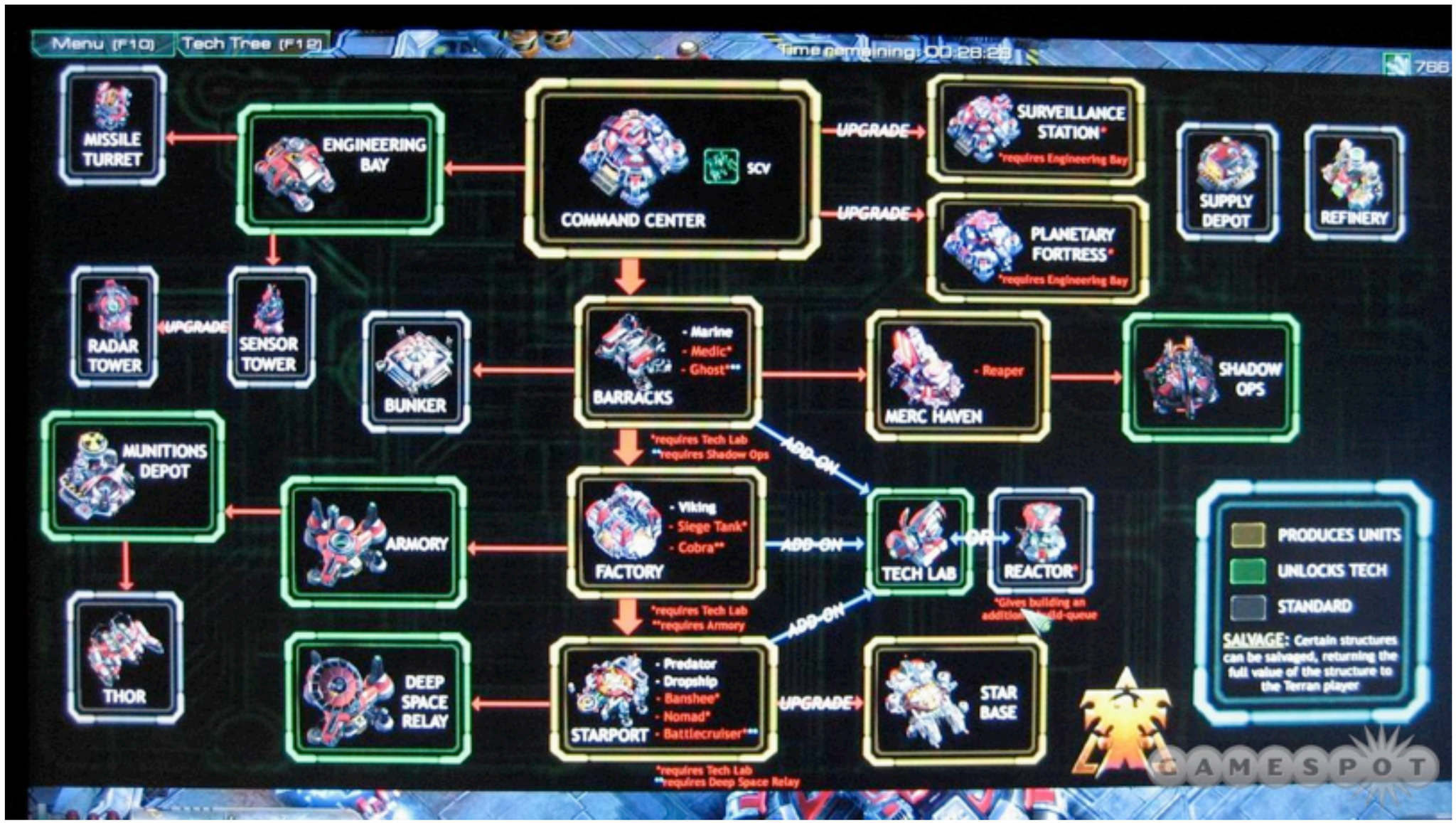
Bayesian Networks are often modeled as directed graphs. *The math gets easier when we ensure there are no cycles*. But a BN might not be a DAG.

In Project Management



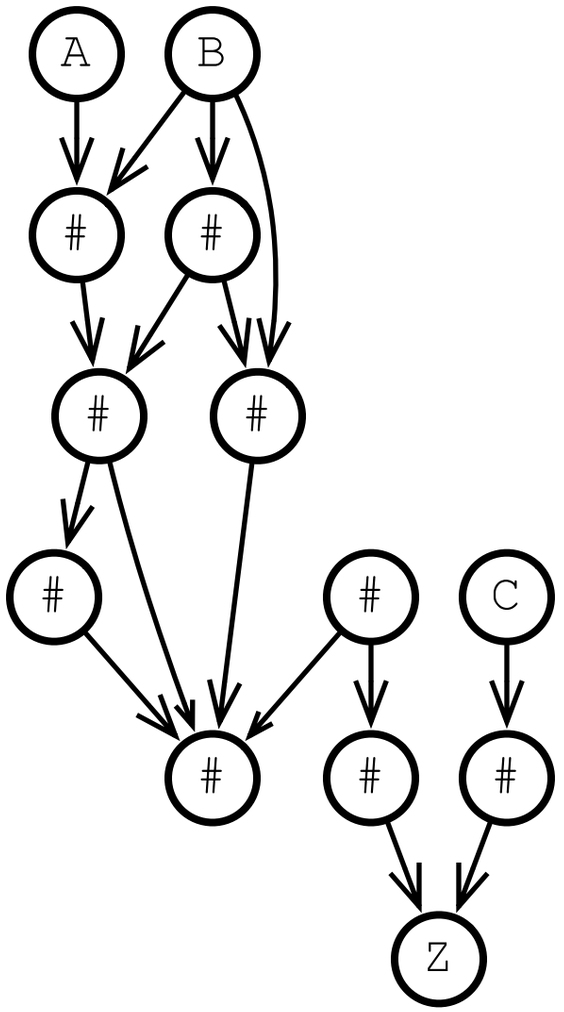
Mitchell, M. (2001) *Use of Directed Acyclic Graph Analysis in Generating Instructions for Multiple Users*. In Proc. Australian Symp. on Information Visualization.

In Video Games



A “tech tree” is usually a DAG. Not always, but usually.

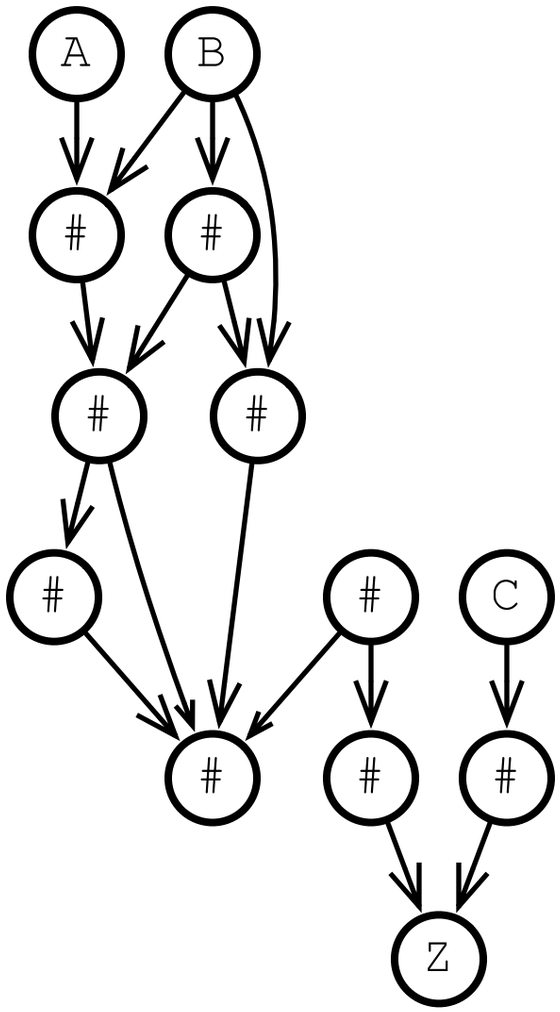
Problem: Path Length



One very common problem is determining the path length from one node to another. Since there might be several paths, we have a choice when computing this length.

Are we interested in the shortest path? The longest path? Are we required to fully explore each node or do we simply need to discover them?

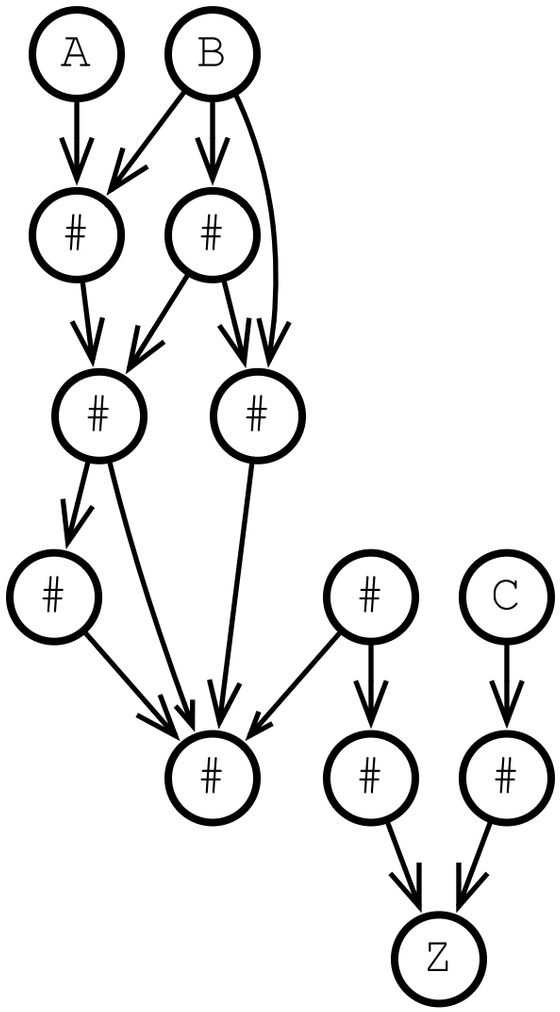
Problem: Path Length



Question:

What's the shortest to Z starting from A or B?

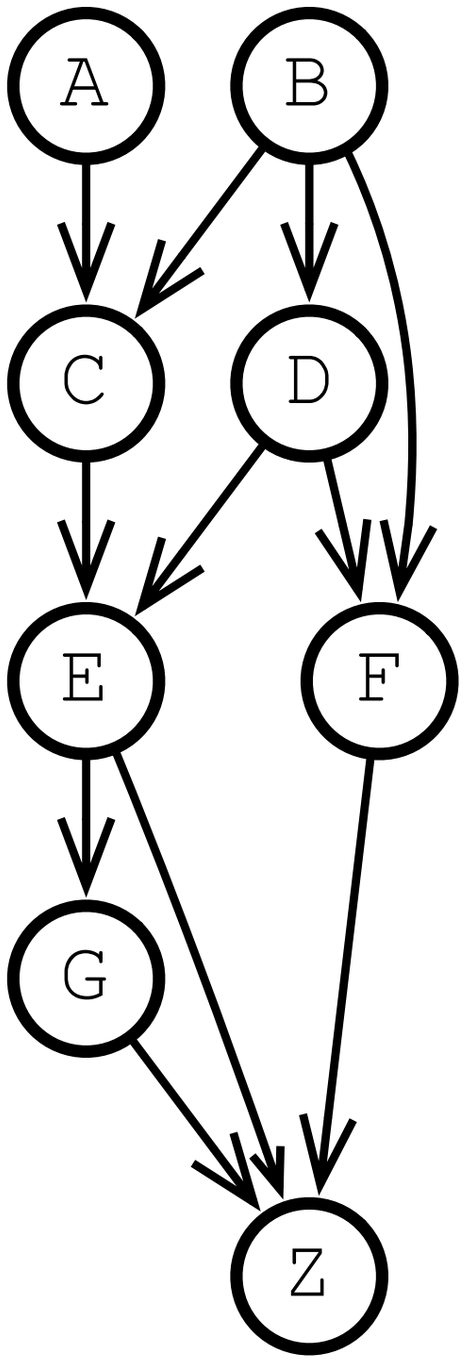
Problem: Path Length



Answer:

There's no path from either A or B to Z.
So, I dunno... infinite?

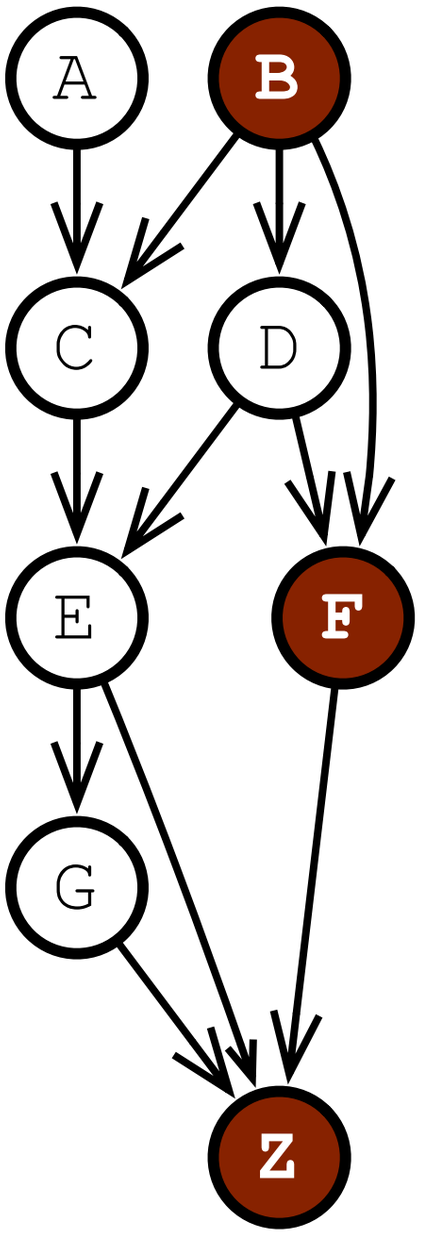
Problem: Path Length



In this week's HW we'll only need to discover a node in order move on to others. There are applications where you'd need to finish a node first.

Starting from B, what's the shortest path to Z? The longest?

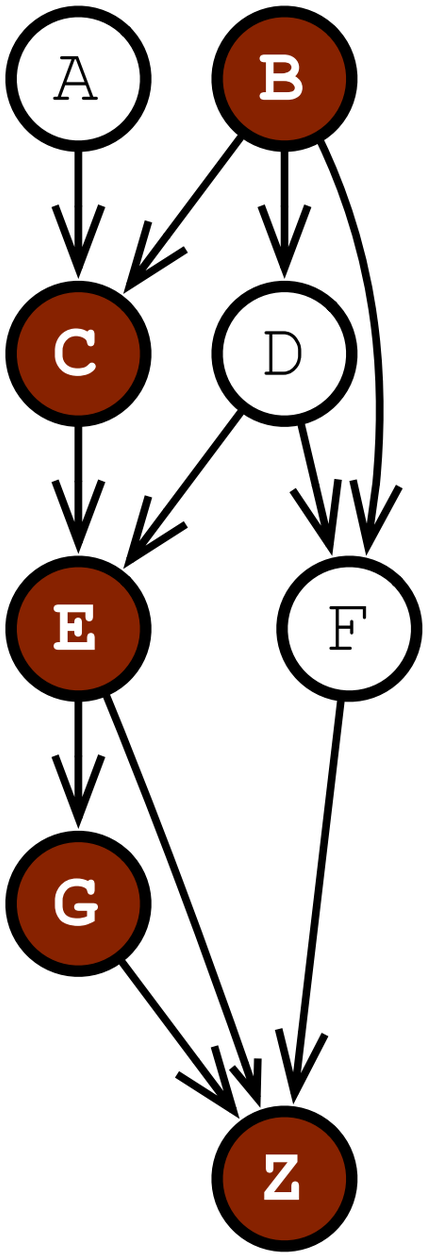
Problem: Path Length



Shortest Path

Since we're only interested in hop count (number of edges traversed) we can just do a BFS to find the shortest path. Easy.

Problem: Path Length



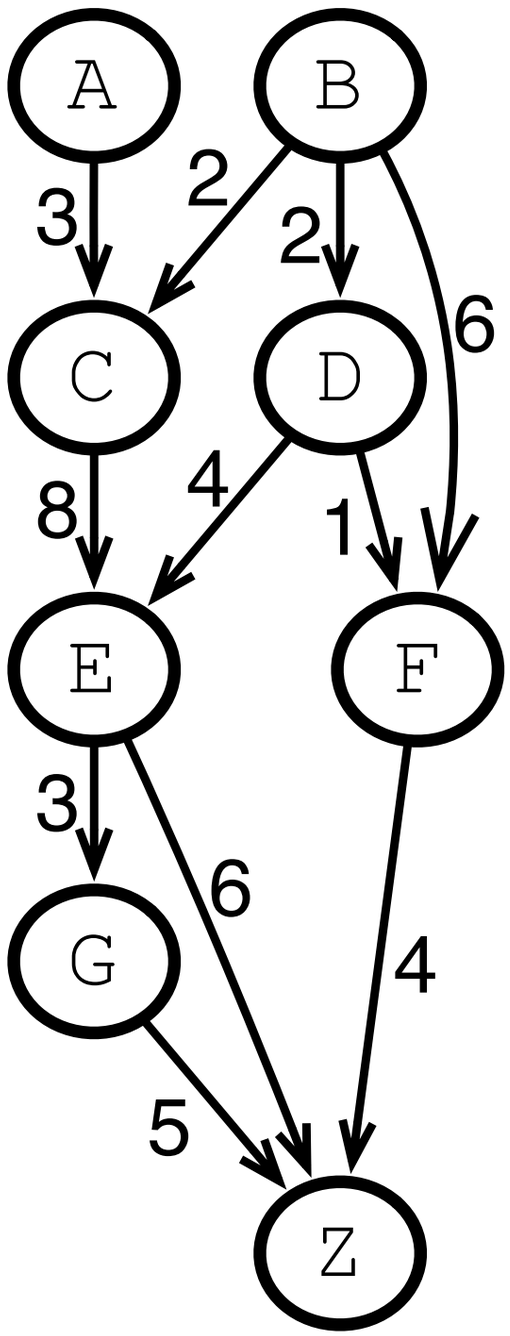
Longest Path

To find the longest path, we can modify our BFS to frame the bfs rank as the longest path between nodes.

This tells us the worst-case scenario of transitions. If we're modeling what could go wrong, this is definitely something we're interested in knowing.

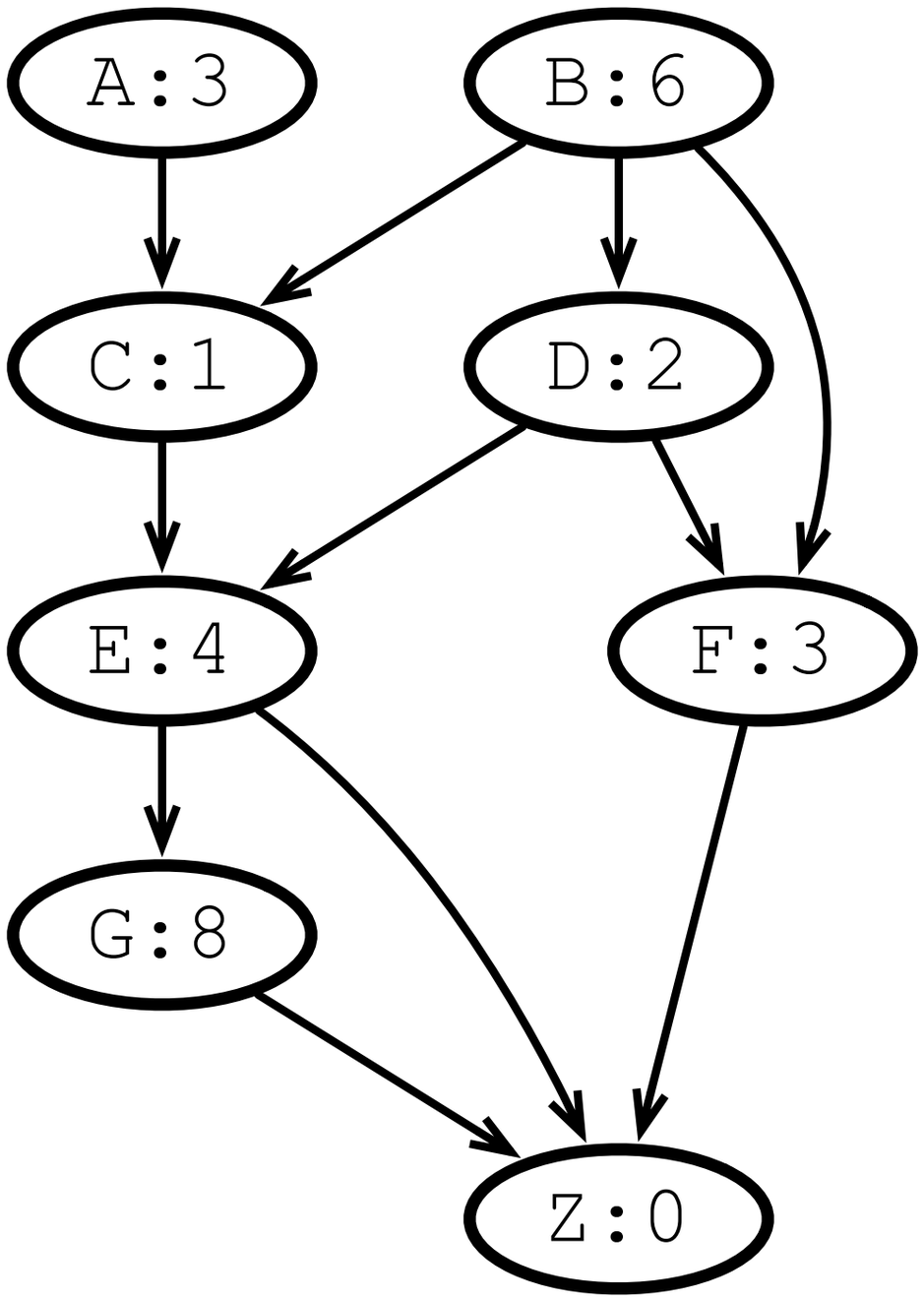
Note there may be more than one: B, **D**, E, G, Z is just as long.

Edges With Weights



We can give the edges *weights*, which is the generic term for the cost associated with traversing that edge. It might represent money, time, or use of some other resource.

Nodes With Weights



Nodes can also have weights associated with them. We aren't going to use this for the homework, but be aware that this can be a useful thing.

The final homework will probably be based on nodes with weights.

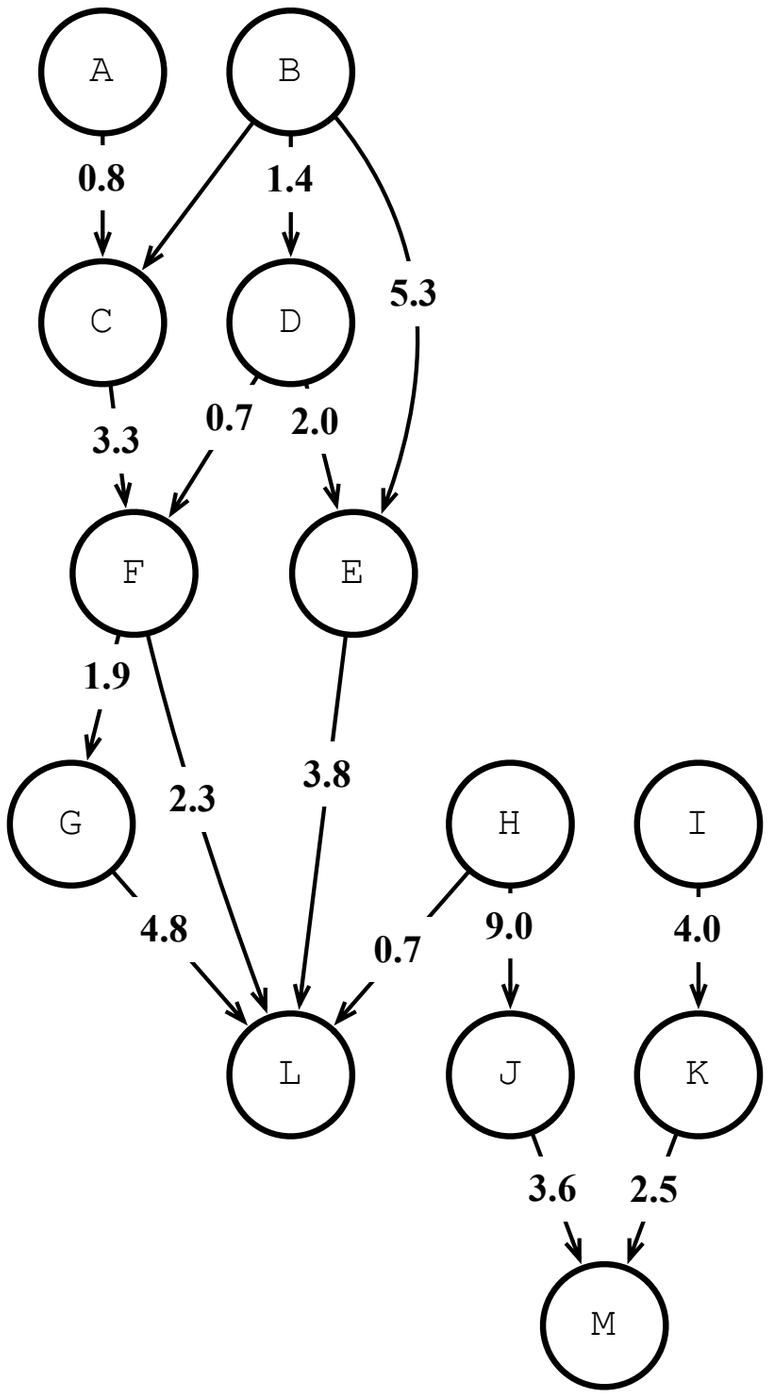
Algos for Shortest Path

Lots of algorithms for finding shortest paths:

- Dijkstra's
- Bellman—Ford
- A* (uses heuristics; very important)
- Floyd—Warshall

We're going to cover Dijkstra's algorithm.

Dijkstra's Algorithm

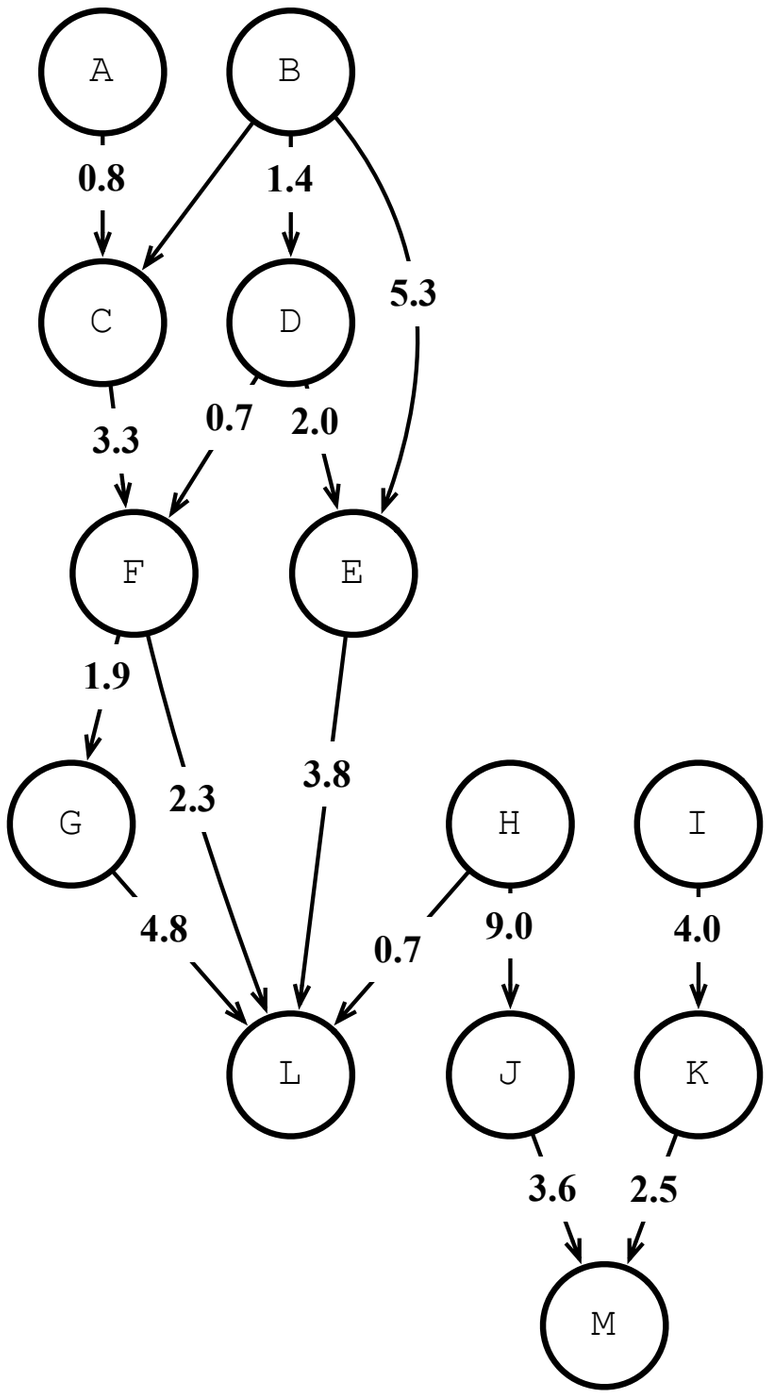


This works on an arbitrary Graph (does not need to be a DAG).

Edges have *non-negative* weight. There are other algos that work with negative weight, but not this one.

It lets you compute a single-source minimum distance to any other node. This means you *need to tell it where to start.*

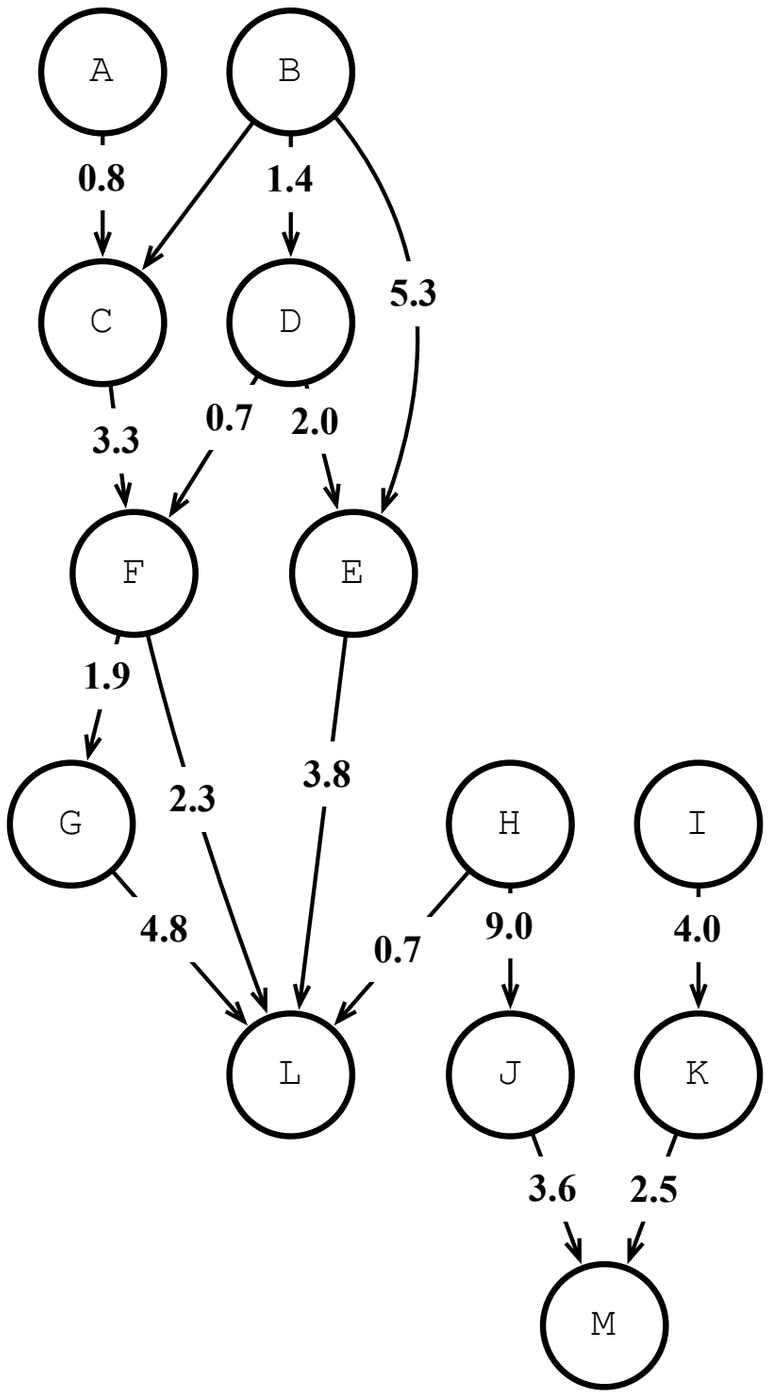
Dijkstra's Algorithm



The general idea is to do a breadth-first search beginning at a *source* node, and compute path distances, where edge weights are considered distances.

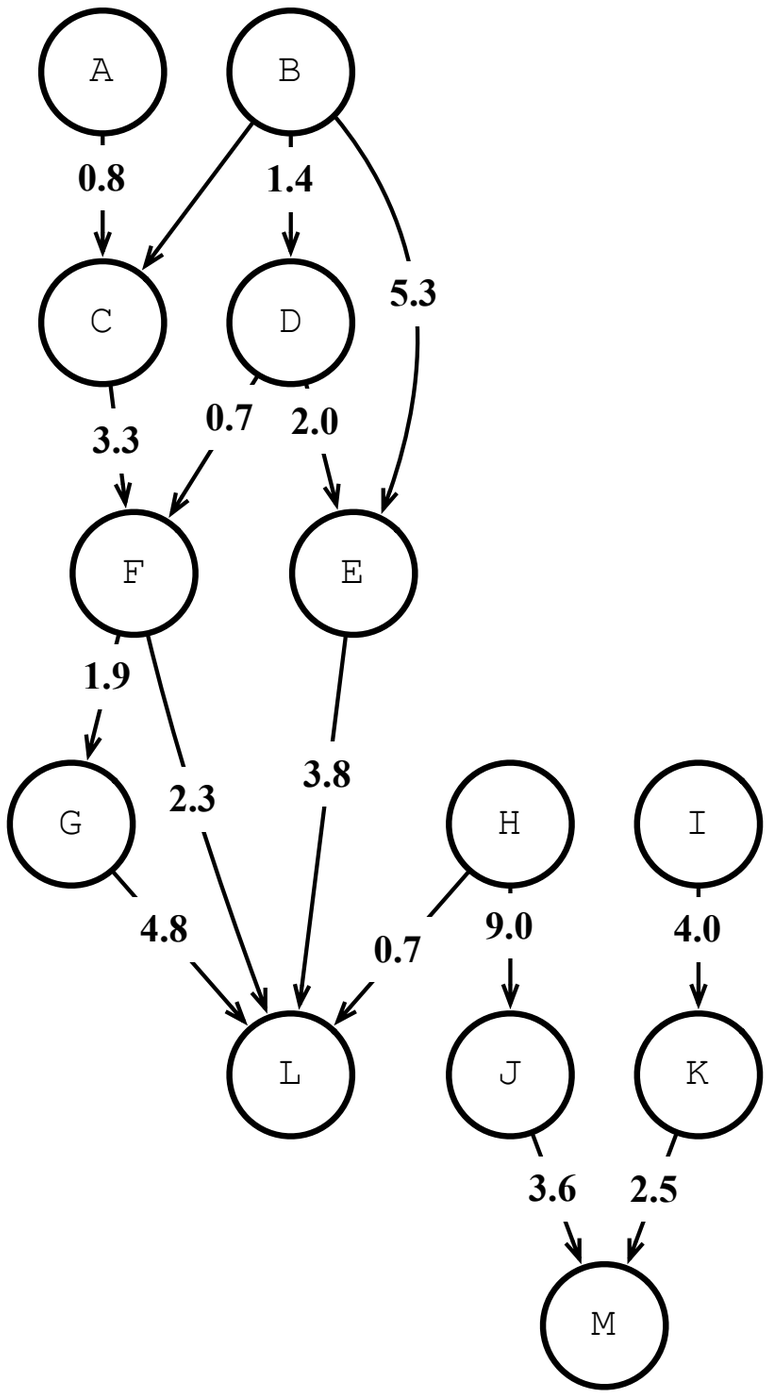
The algorithm records the minimum distance to a node by adding an explored node's distance to the next one. It also maintains precedence in a tree called the *shortest path tree* (a.k.a. the *minimum spanning tree*).

Dijkstra's Algorithm



I'm going to show the algorithm graphically first because the pseudocode is actually sort of confusing, but the visual is pretty clear.

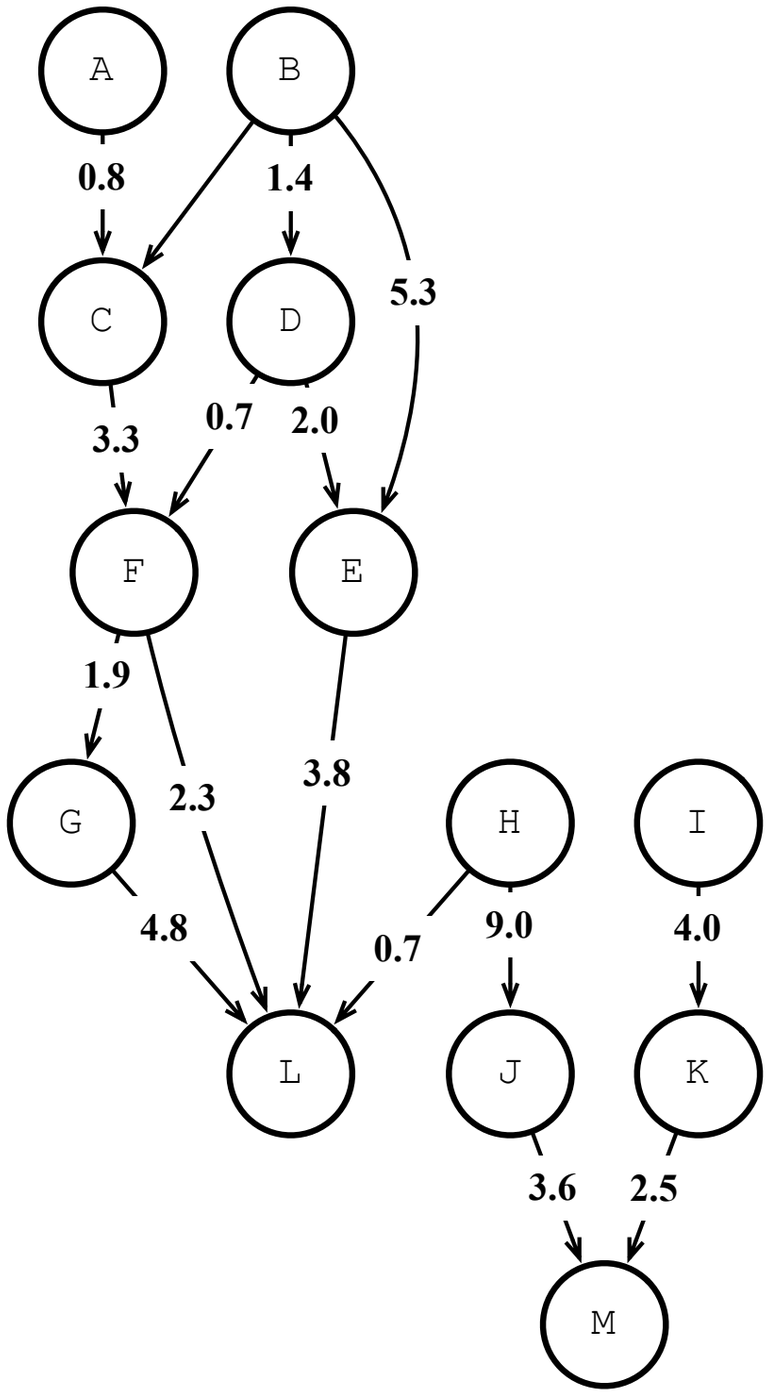
Dijkstra's Algorithm



First thing to do is set the path distance to +infinity (or a suitably large number), and to clear whatever precedence information might be stored from a previous run.

Also create a priority queue **Q** to hold nodes. They are prioritized in ascending order of path distance.

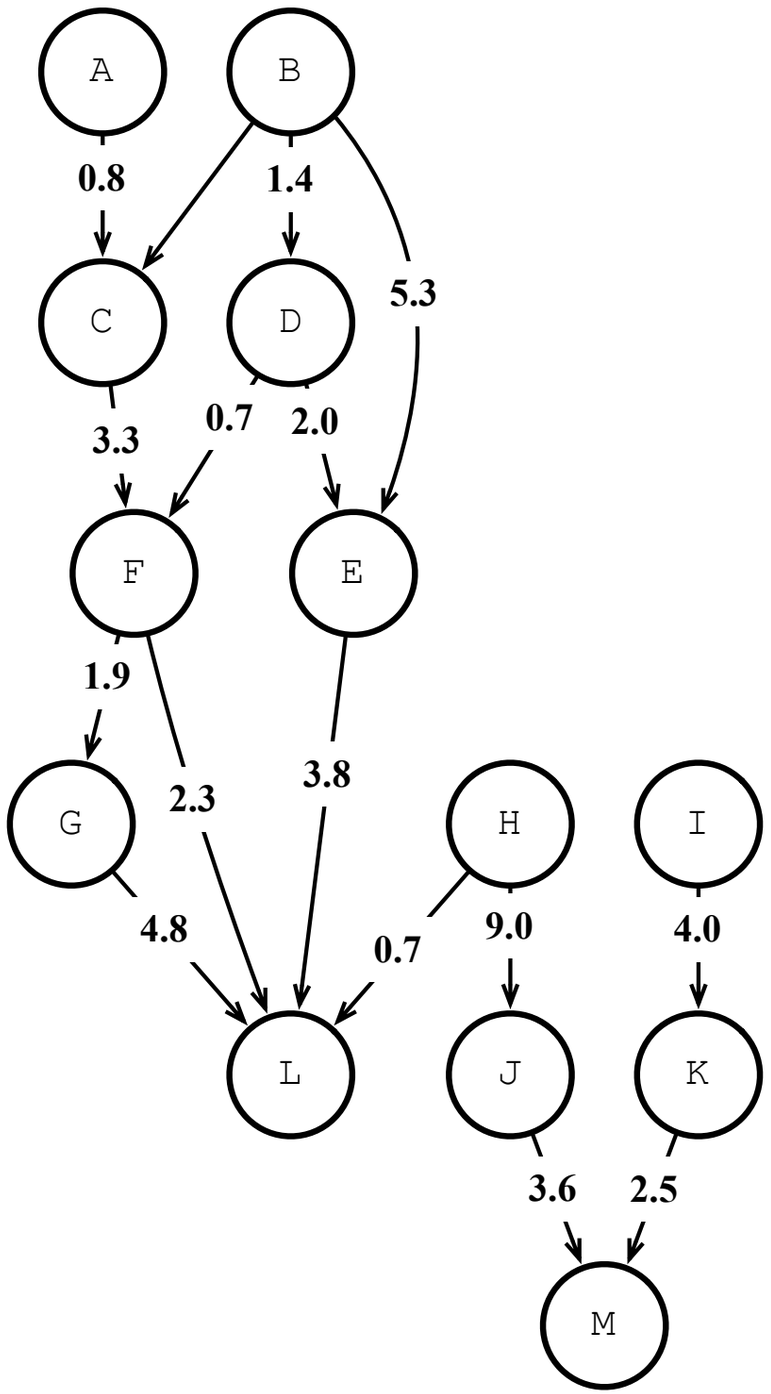
Dijkstra's Algorithm



Say we are going to start the process at node **B**, and we are searching for node **L**. This means we will probably stop the algorithm before exploring everything, because we stop when we have an answer.

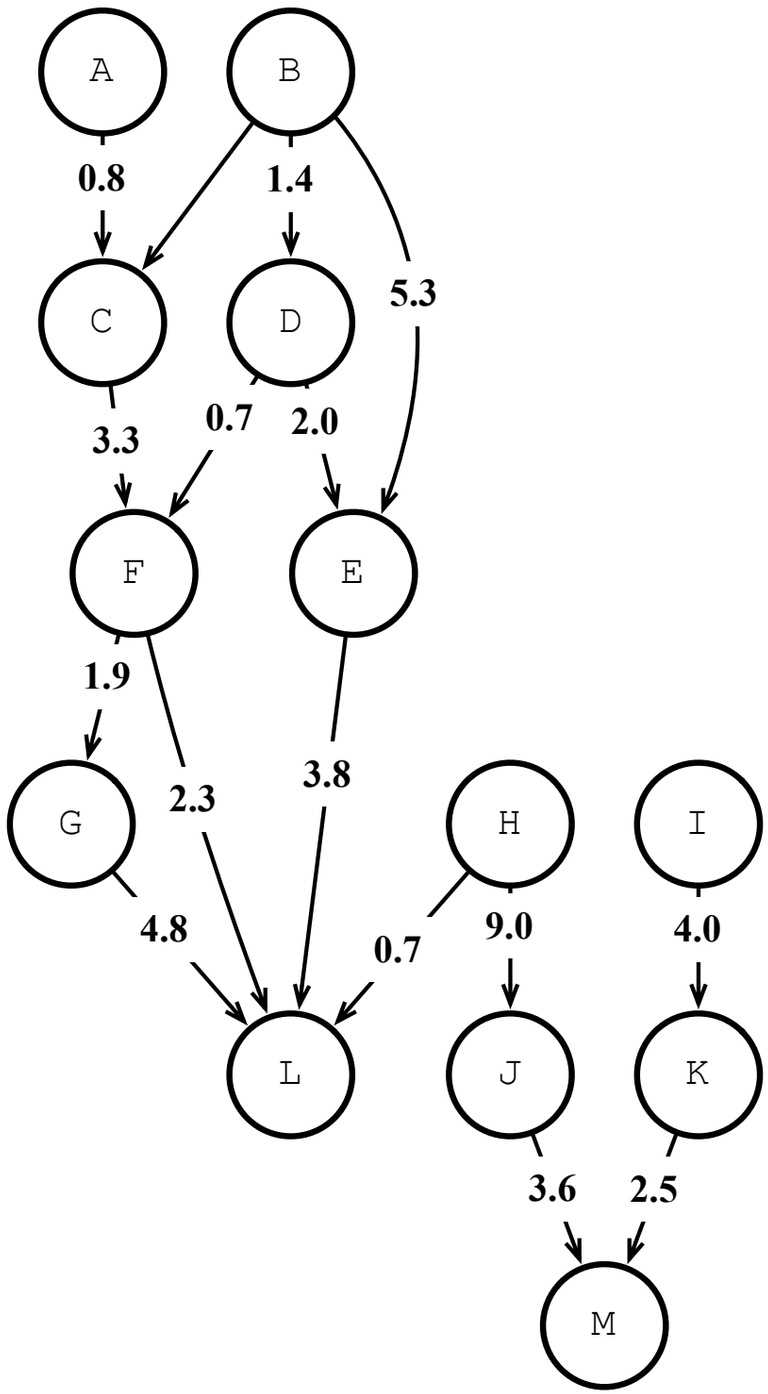
So: set node A's distance to 0. Then, add all the nodes in the graph.

Dijkstra's Algorithm



Node	Dist	Pred.
B	0	?
A	?	?
C	?	?
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm

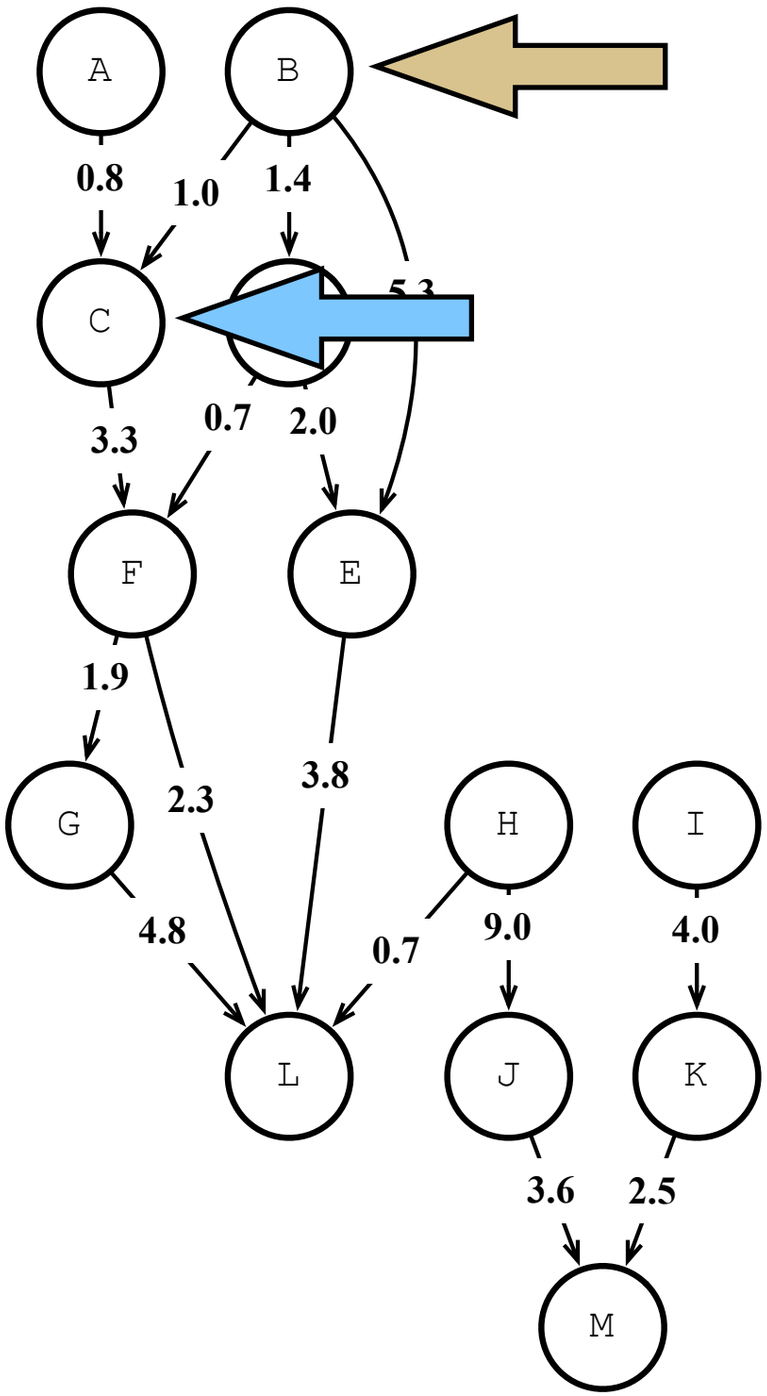


The main loop of the algorithm is:

Remove **u** from Q.
If u's distance is infinity, stop.
For all neighbors **v** of u:

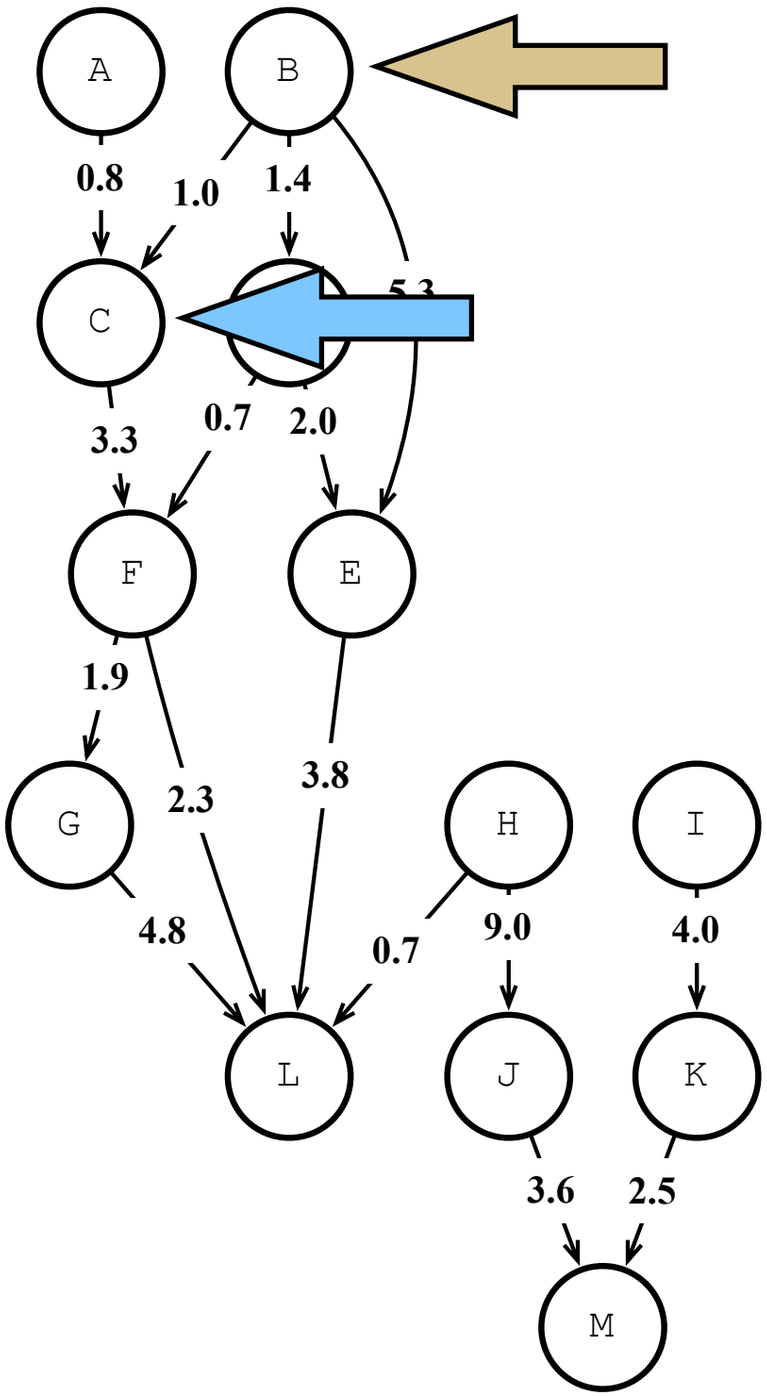
```
    dist = u.dist + dist_between(v, u)
    if dist < v.dist:
        v.dist = dist
        v.predecessor = u
        remove/insert v in Q
```

Dijkstra's Algorithm



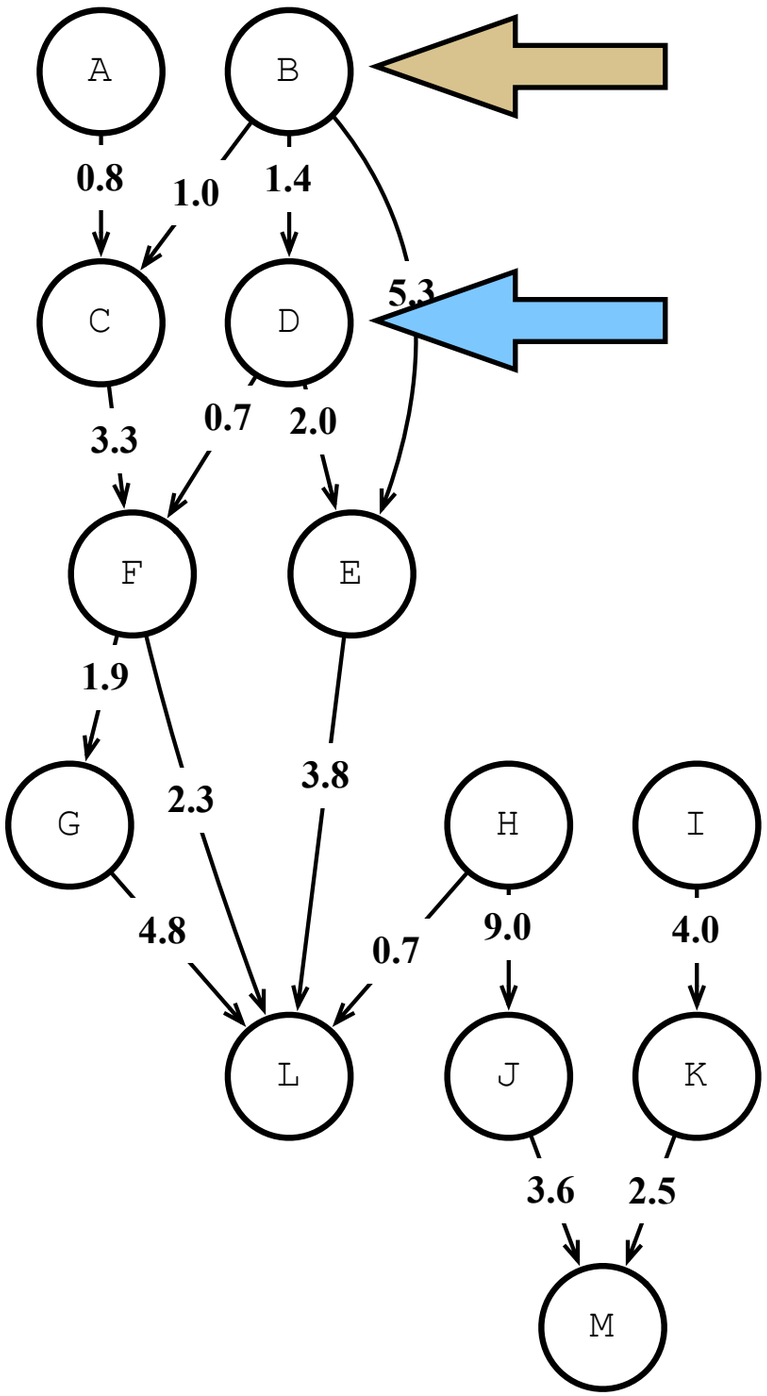
Node	Dist	Pred.
B	0	?
A	?	?
C	1.0	B
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



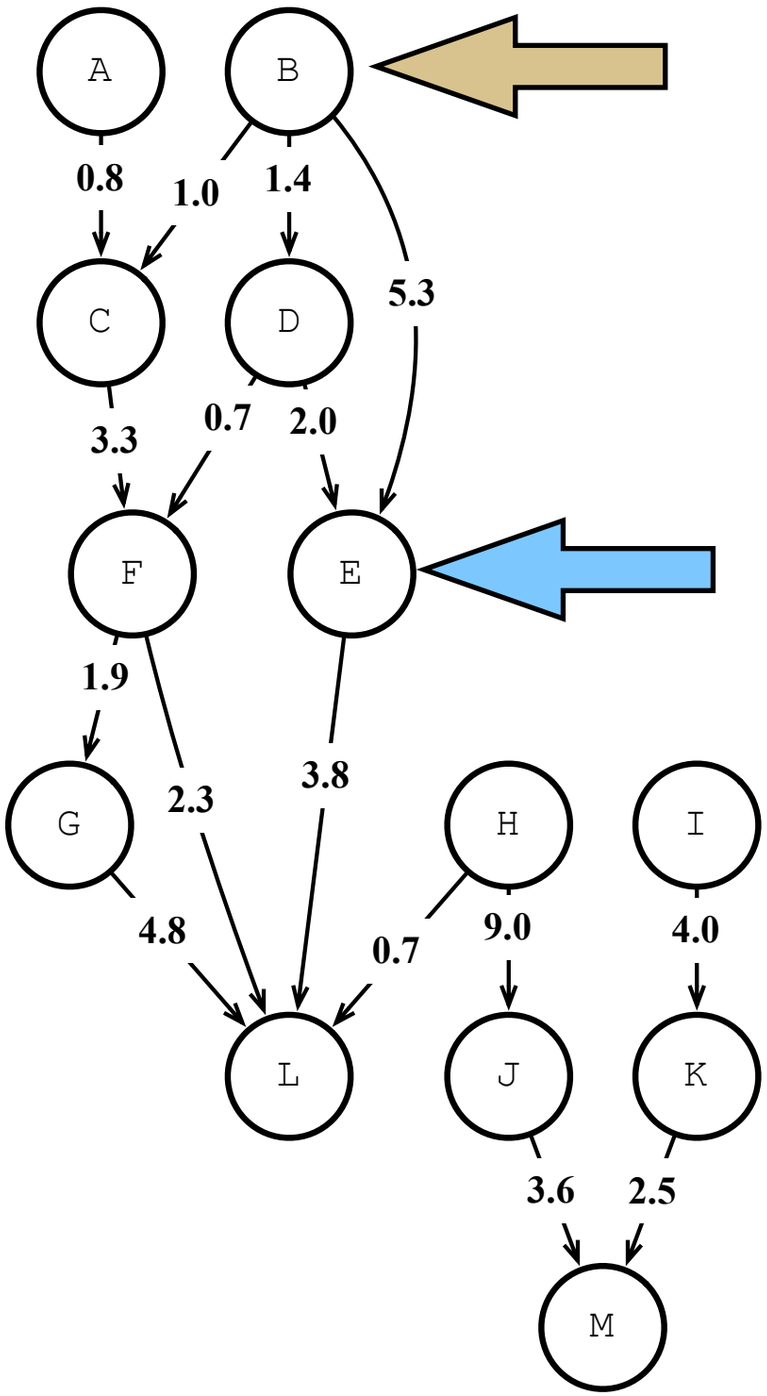
Node	Dist	Pred.
B	0	?
C	1.0	B
A	?	?
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



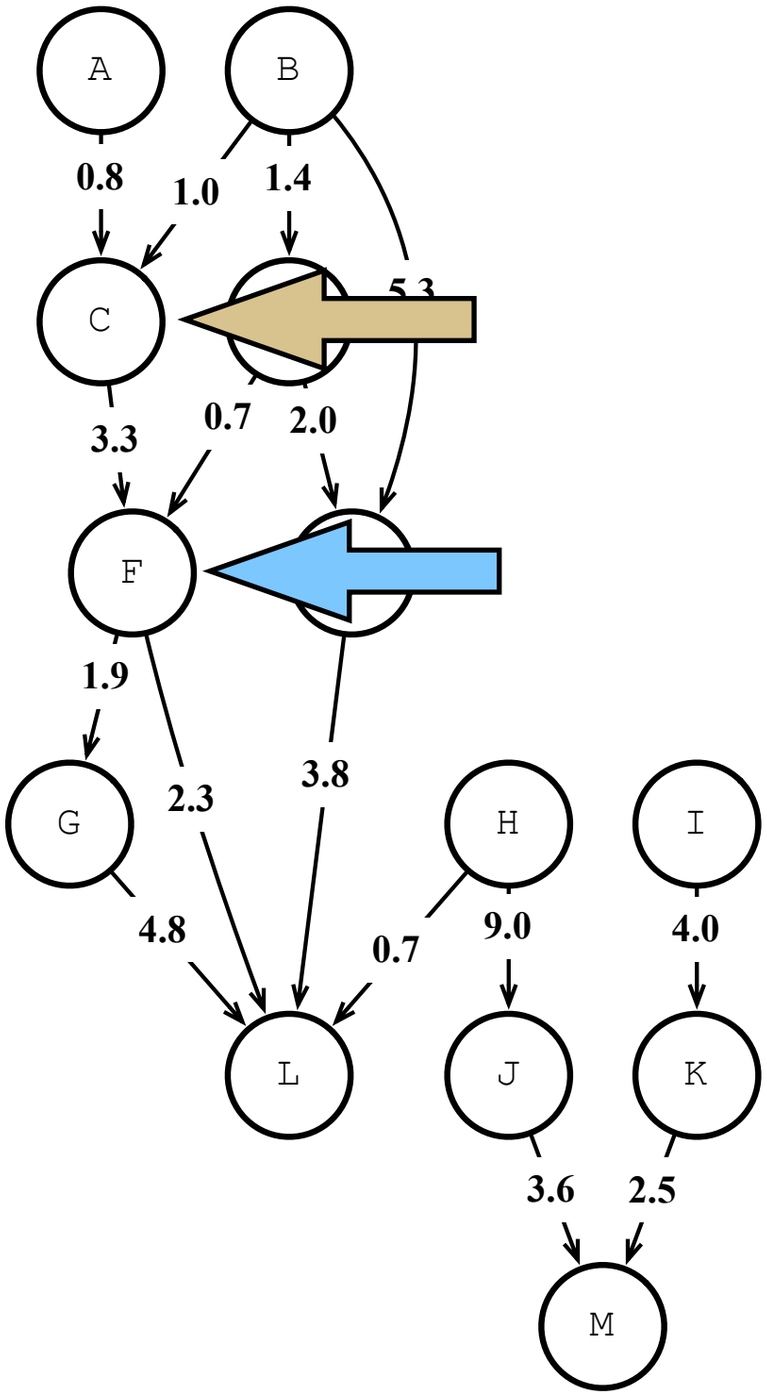
Node	Dist	Pred.
B	0	?
C	1.0	B
A	?	?
D	1.4	B
E	?	?
F	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



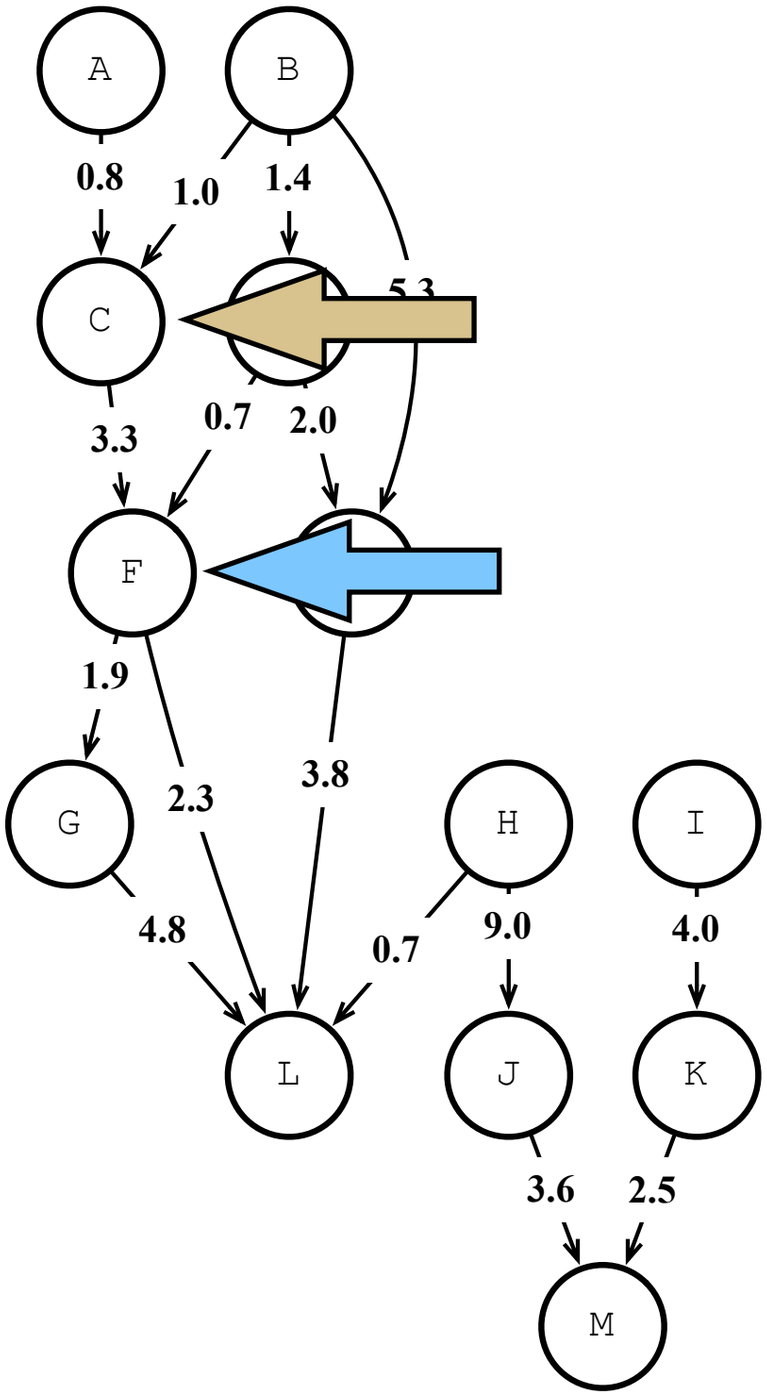
Node	Dist	Pred.
B	0	?
C	1.0	B
D	1.4	B
A	?	?
E	5.3	B
F	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



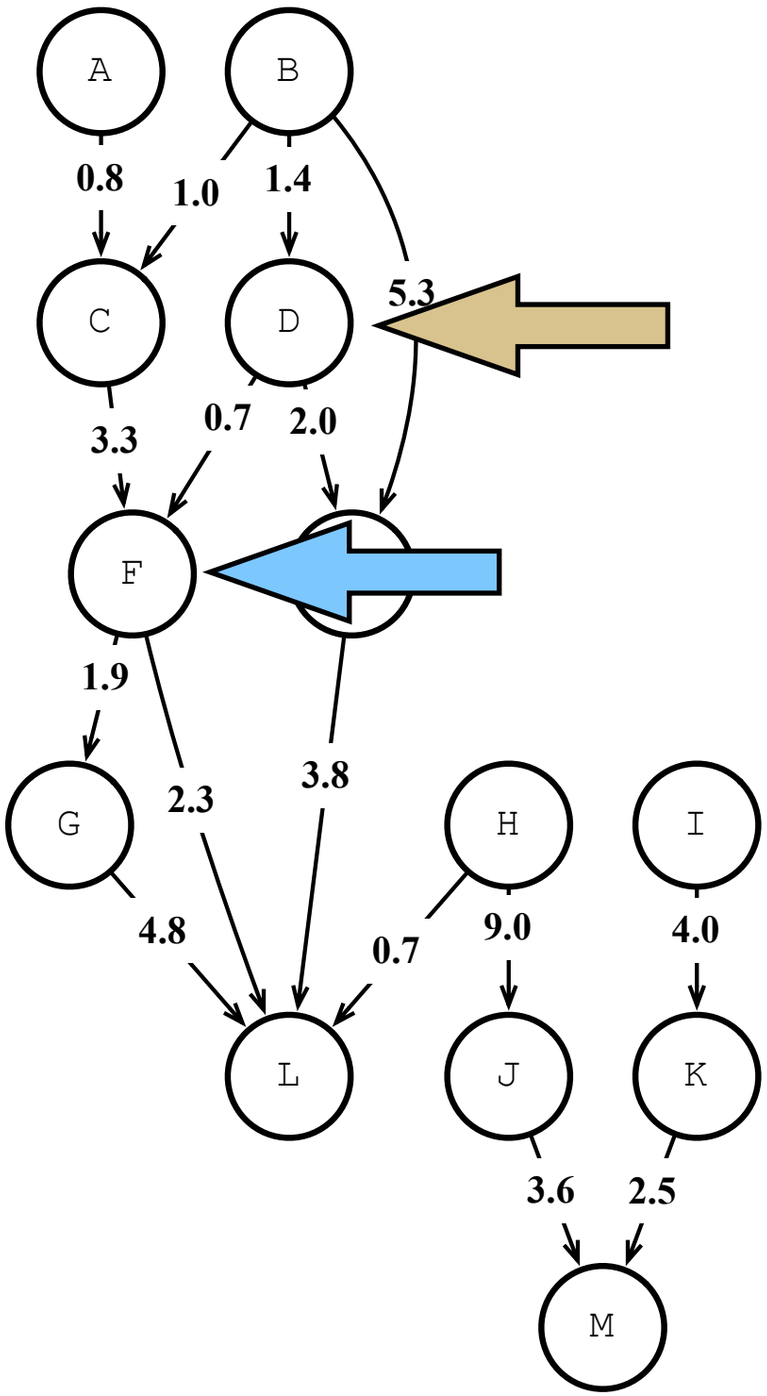
Node	Dist	Pred.
C	1.0	B
D	1.4	B
E	5.3	B
A	?	?
F	1.0 + 3.3 = 4.3	C
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



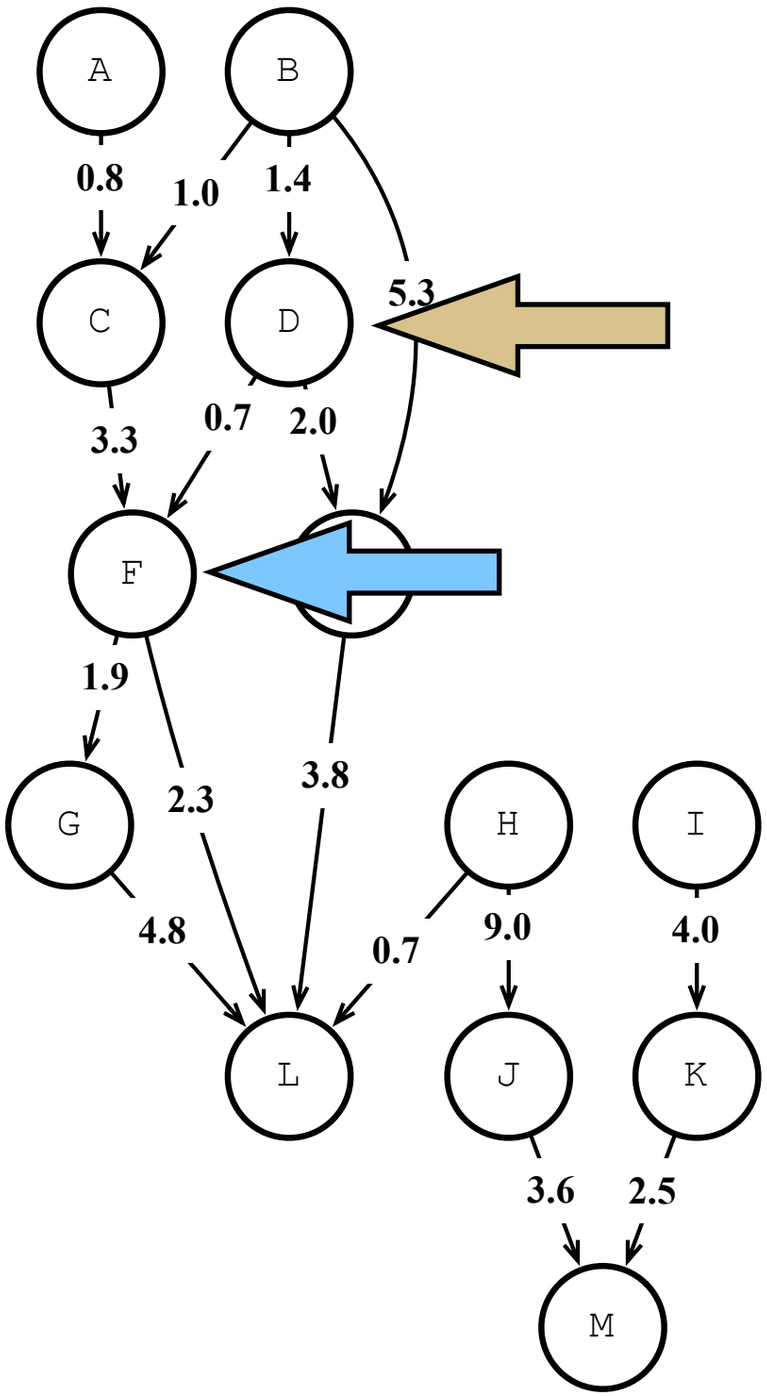
Node	Dist	Pred.
C	1.0	B
D	1.4	B
F	4.3	C
E	5.3	B
A	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



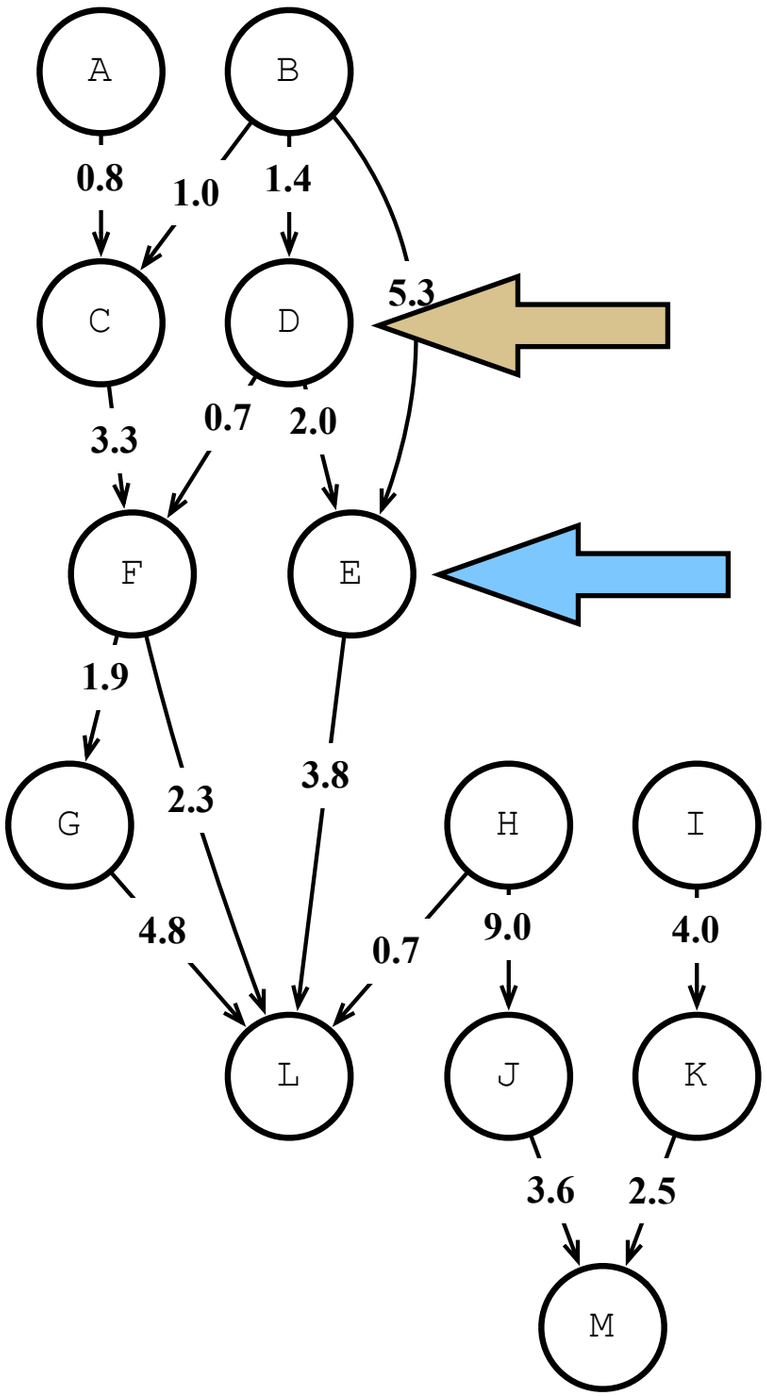
Node	Dist	Pred.
D	1.4	B
F	4.3 > 1.4 + 0.7	D
E	5.3	B
A	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



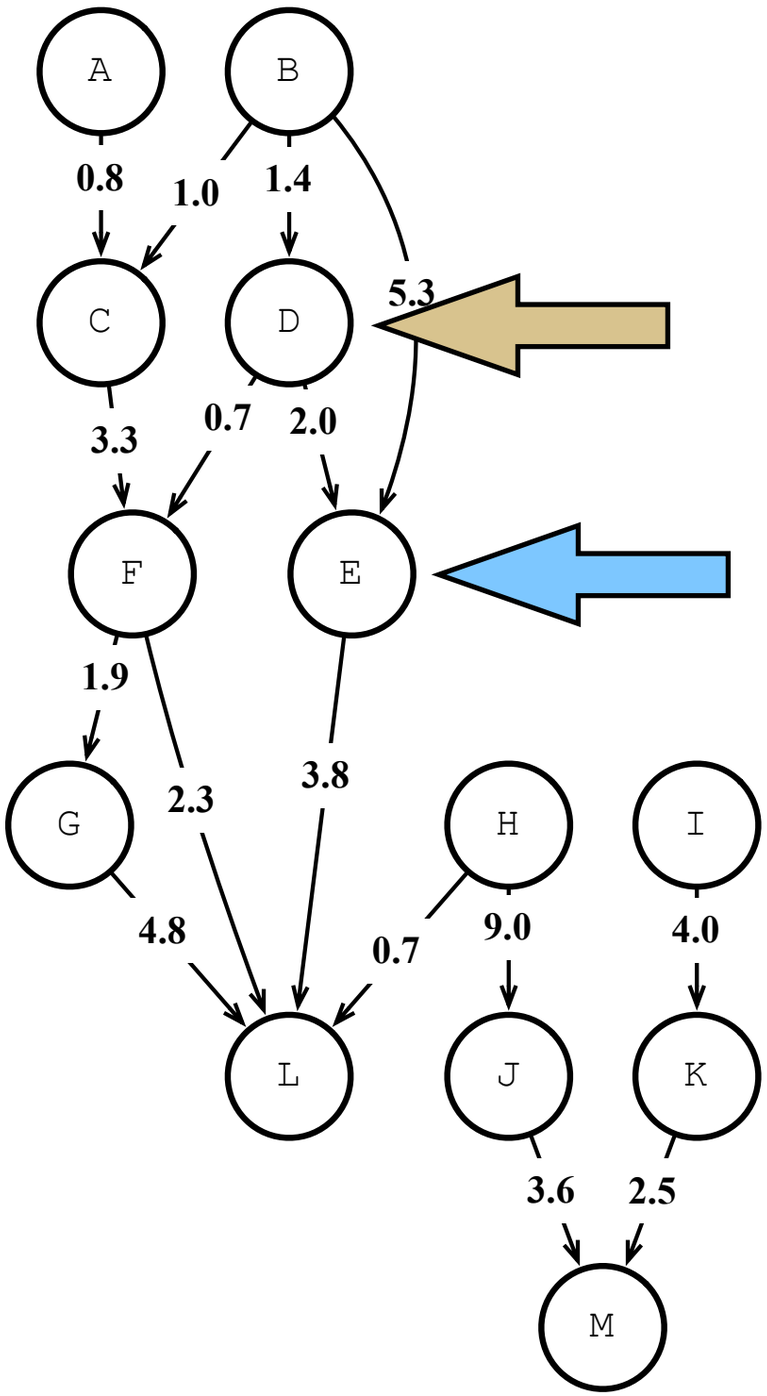
Node	Dist	Pred.
D	1.4	B
F	2.1	D
E	5.3	B
A	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



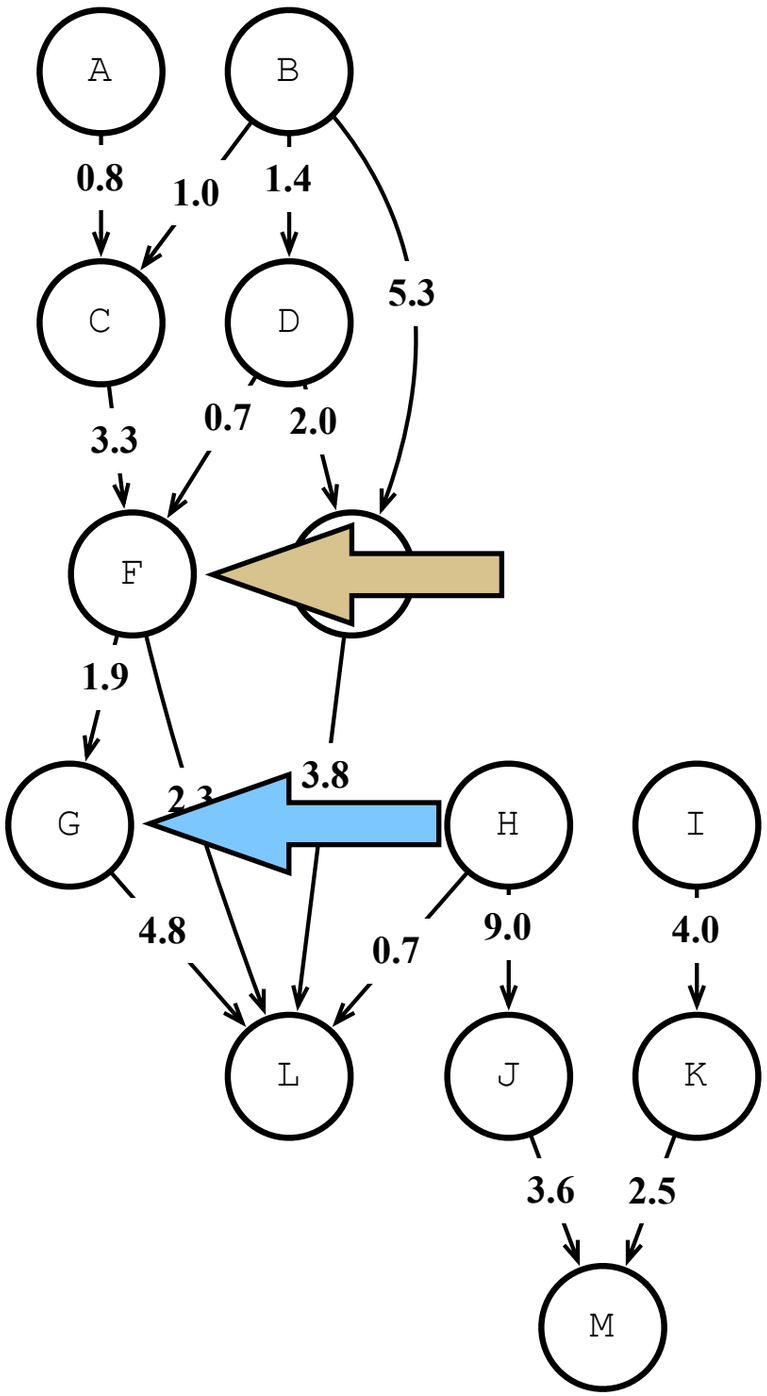
Node	Dist	Pred.
D	1.4	B
F	2.1	D
E	5.3 > 1.4 + 2.0	D
A	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



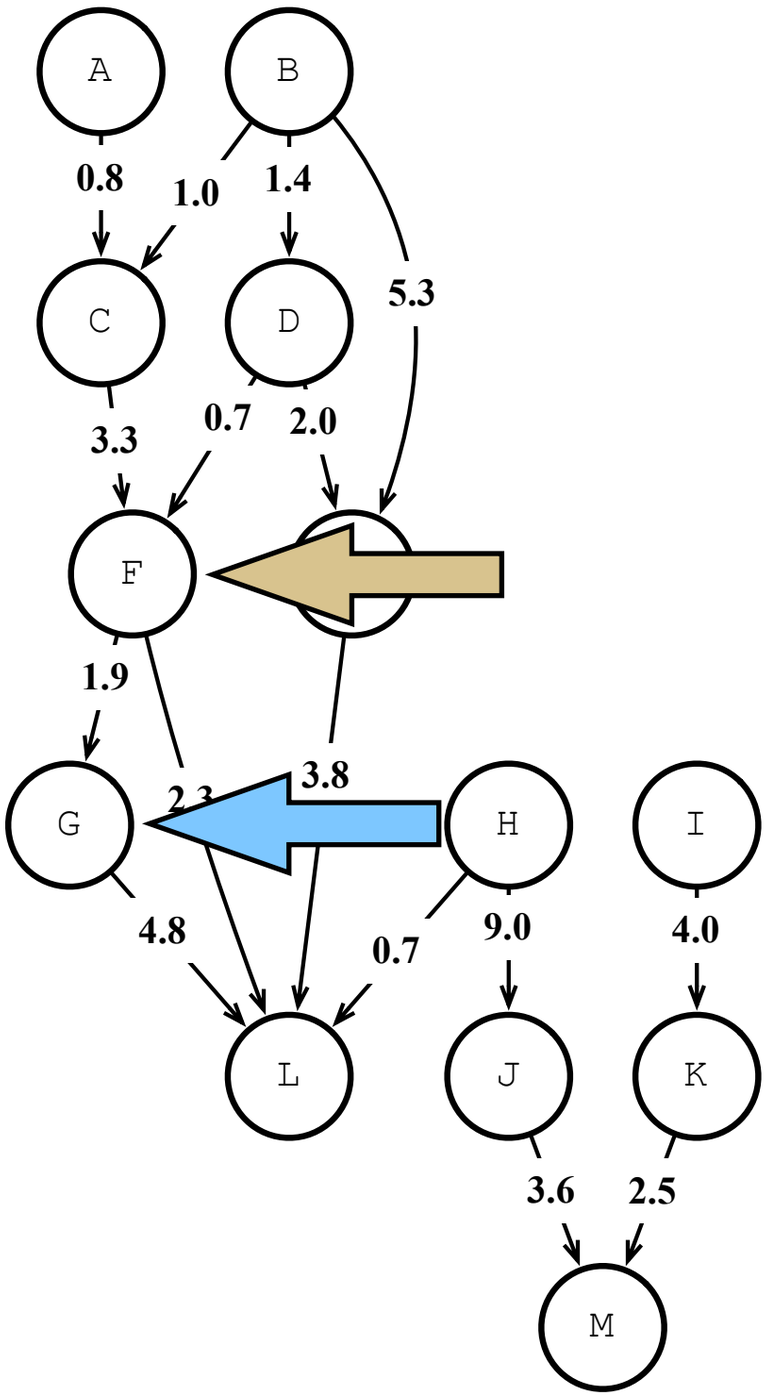
Node	Dist	Pred.
D	1.4	B
F	2.1	D
E	3.4	D
A	?	?
G	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



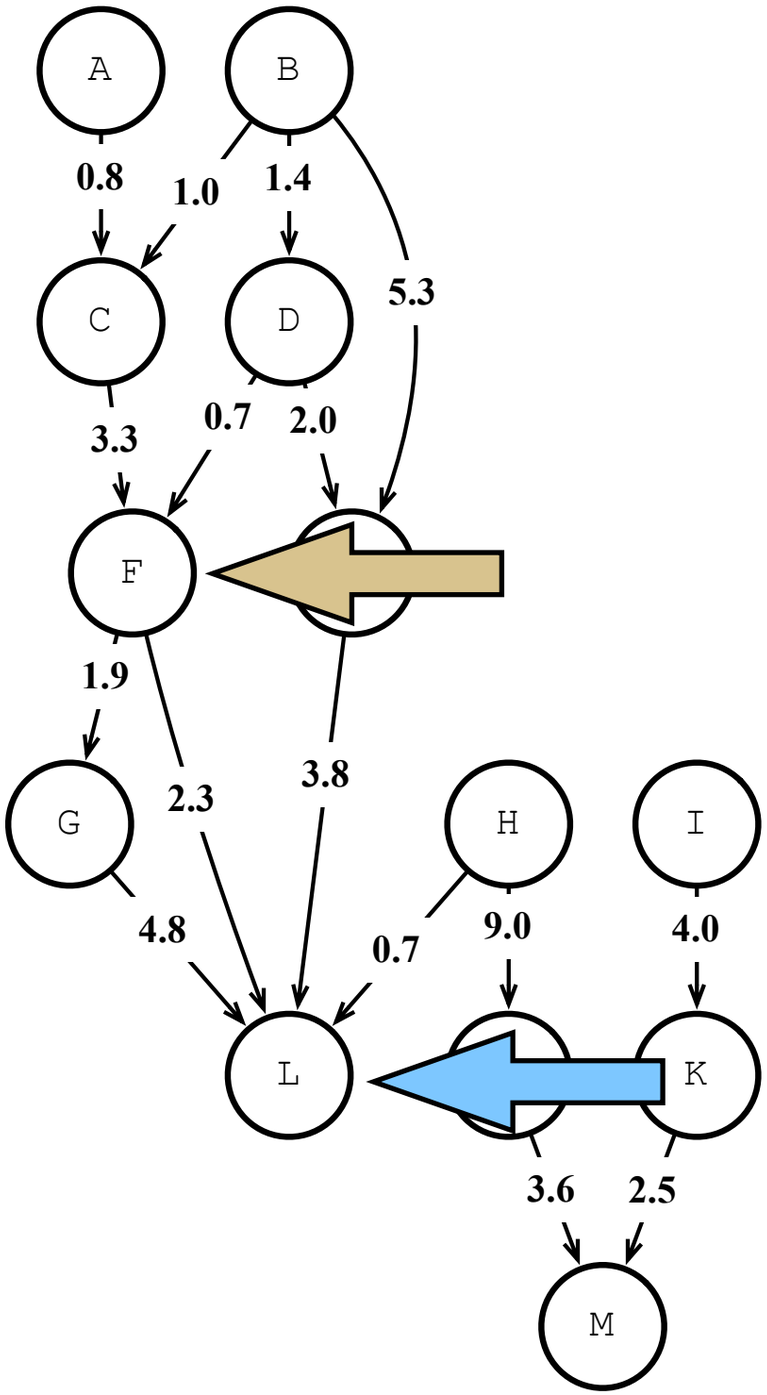
Node	Dist	Pred.
F	2.1	D
E	3.4	D
A	?	?
G	2.1+1.9	F
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



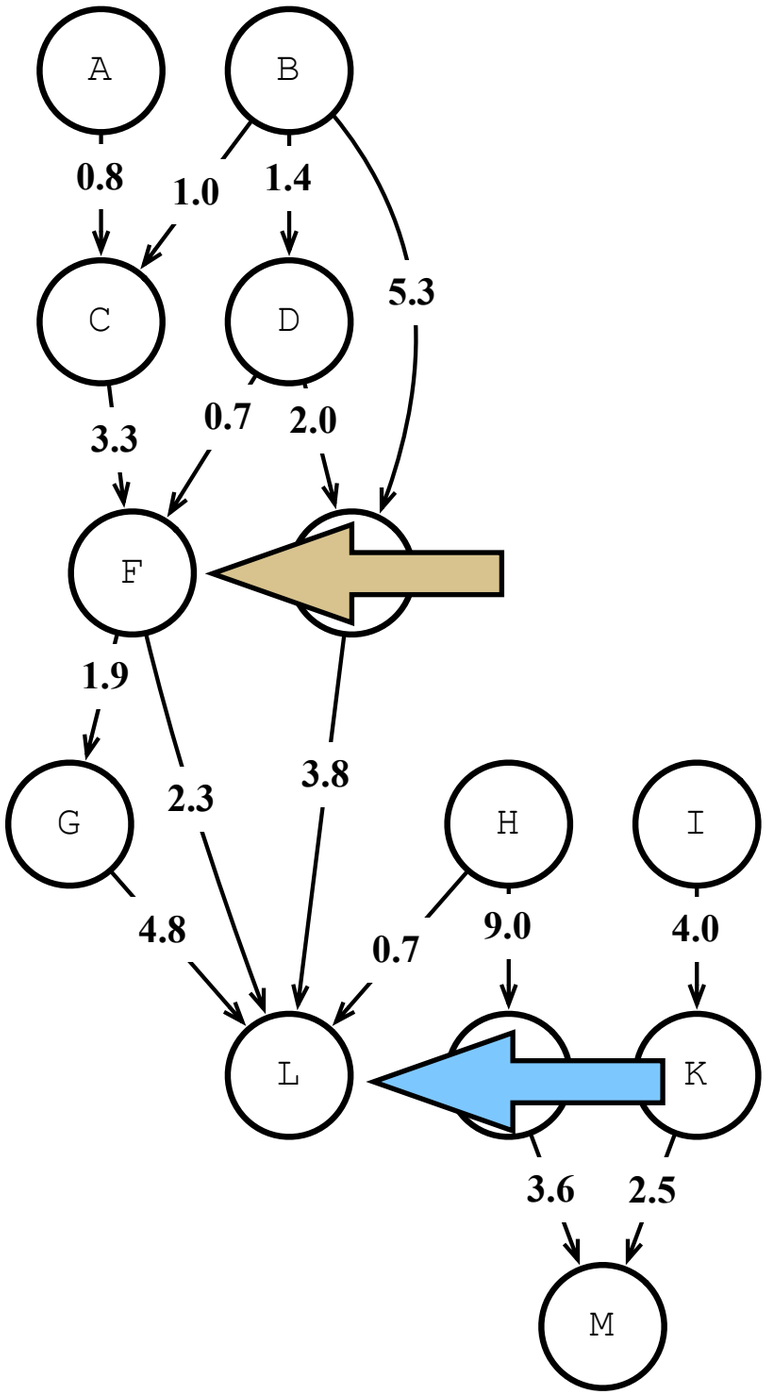
Node	Dist	Pred.
F	2.1	D
E	3.4	D
G	4.0	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	?	?
M	?	?

Dijkstra's Algorithm



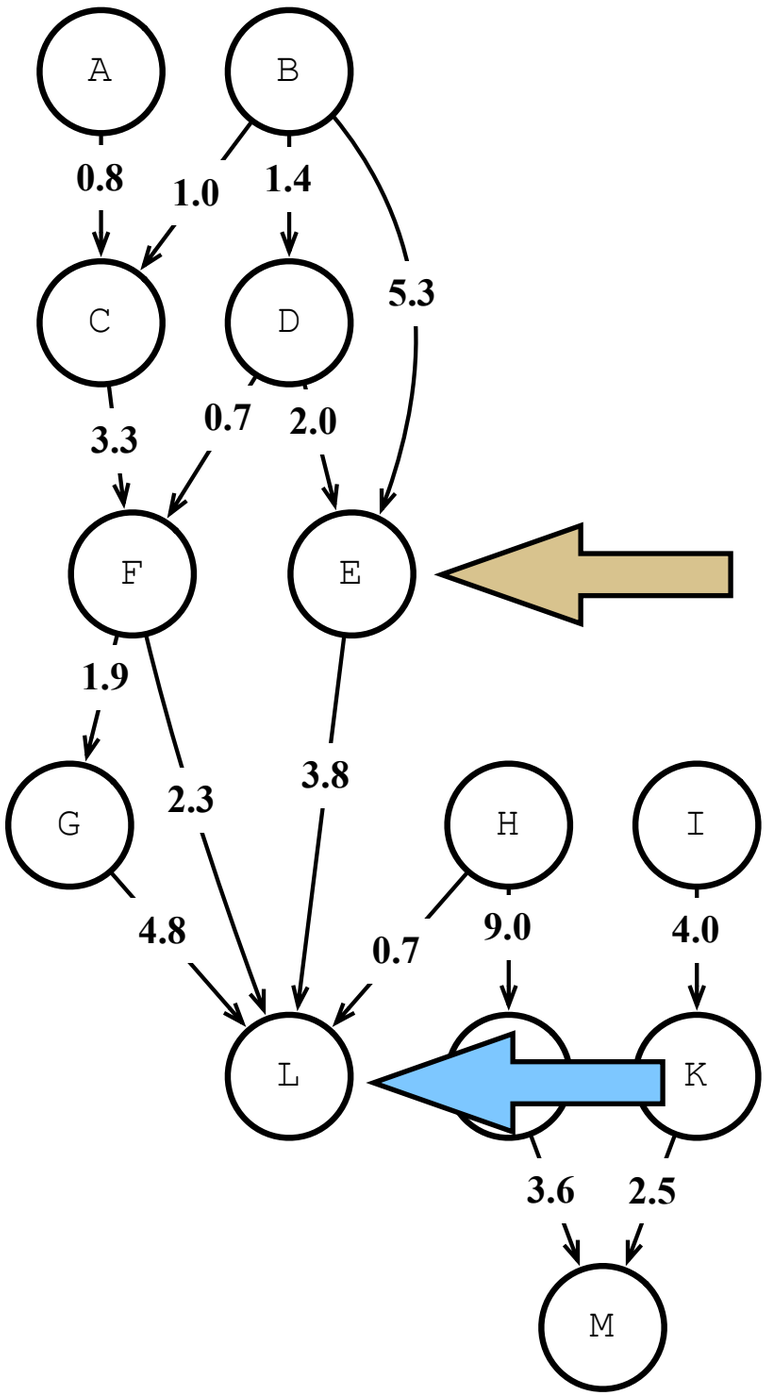
Node	Dist	Pred.
F	2.1	D
E	3.4	D
G	4.0	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
L	2.1+2.3	F
M	?	?

Dijkstra's Algorithm



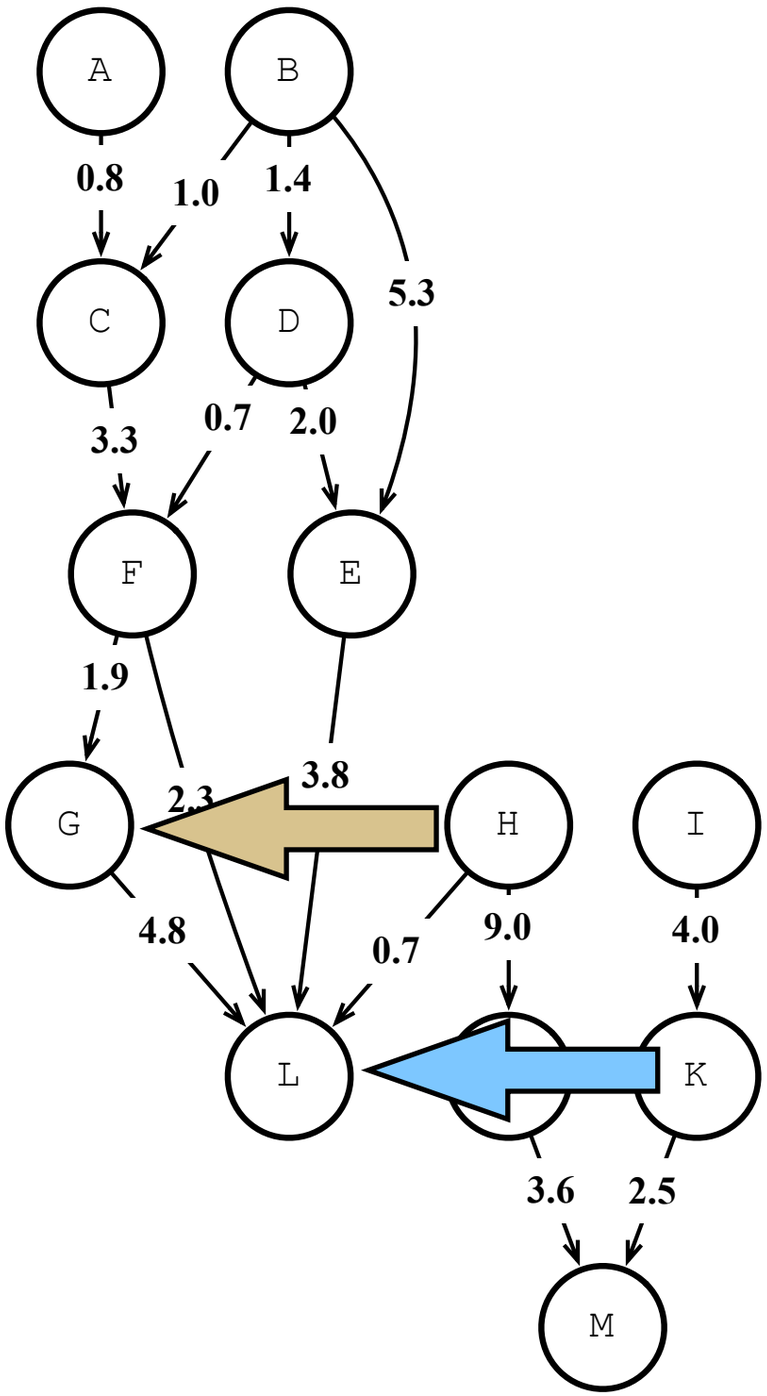
Node	Dist	Pred.
F	2.1	D
E	3.4	D
G	4.0	F
L	4.4	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
M	?	?

Dijkstra's Algorithm



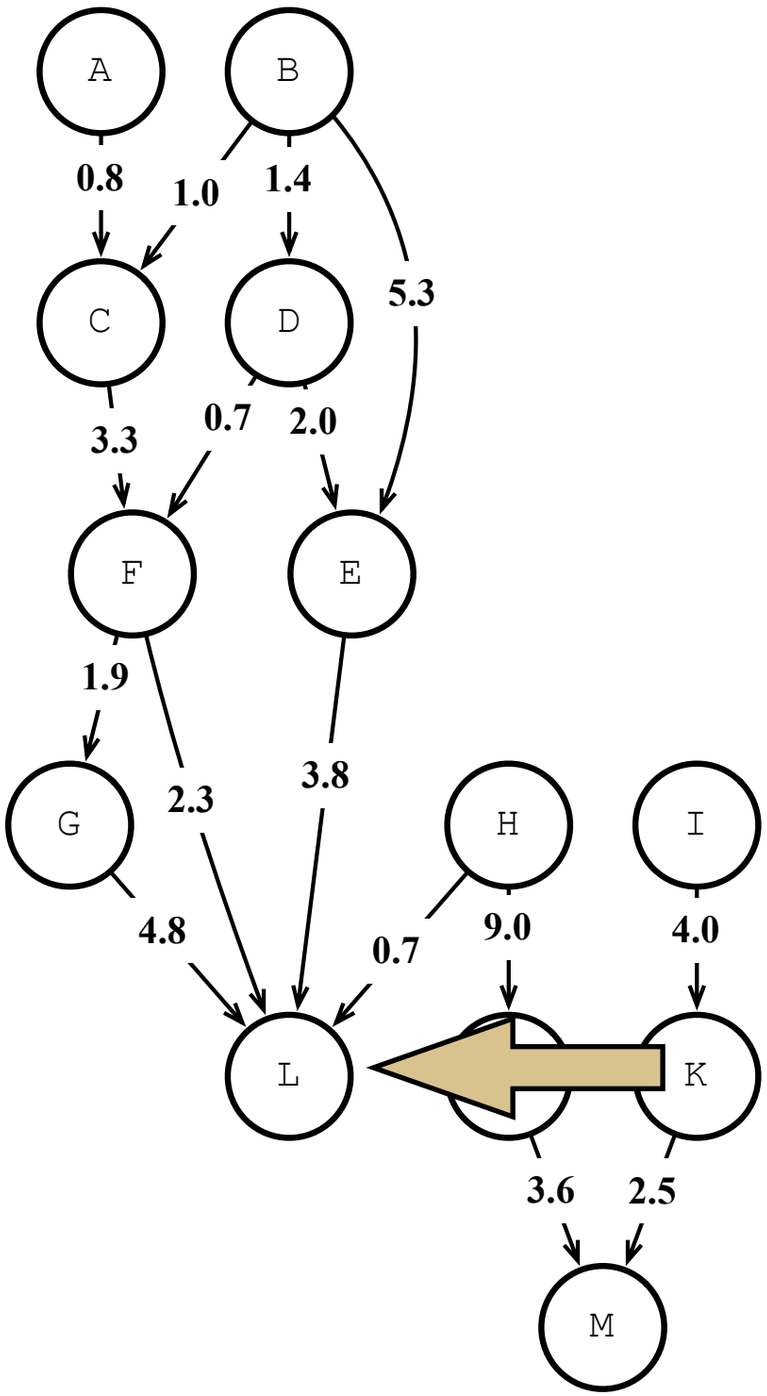
Node	Dist	Pred.
E	3.4	D
G	4.0	F
L	4.4 < 3.4+3.8	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
M	?	?

Dijkstra's Algorithm



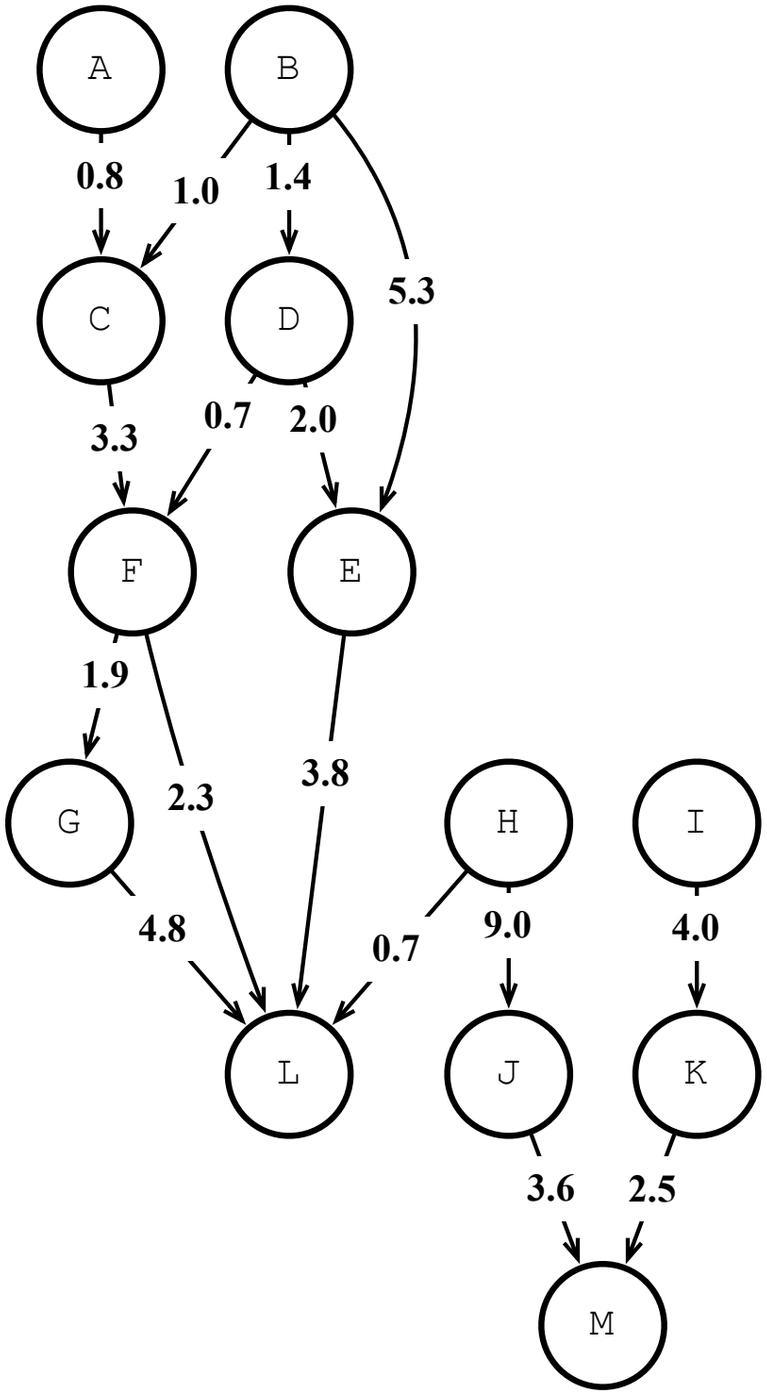
Node	Dist	Pred.
G	4.0	F
L	4.4 < 4.0+4.8	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
M	?	?

Dijkstra's Algorithm



Node	Dist	Pred.
L	4.4	F
A	?	?
H	?	?
I	?	?
J	?	?
K	?	?
M	?	?

Dijkstra's Algorithm



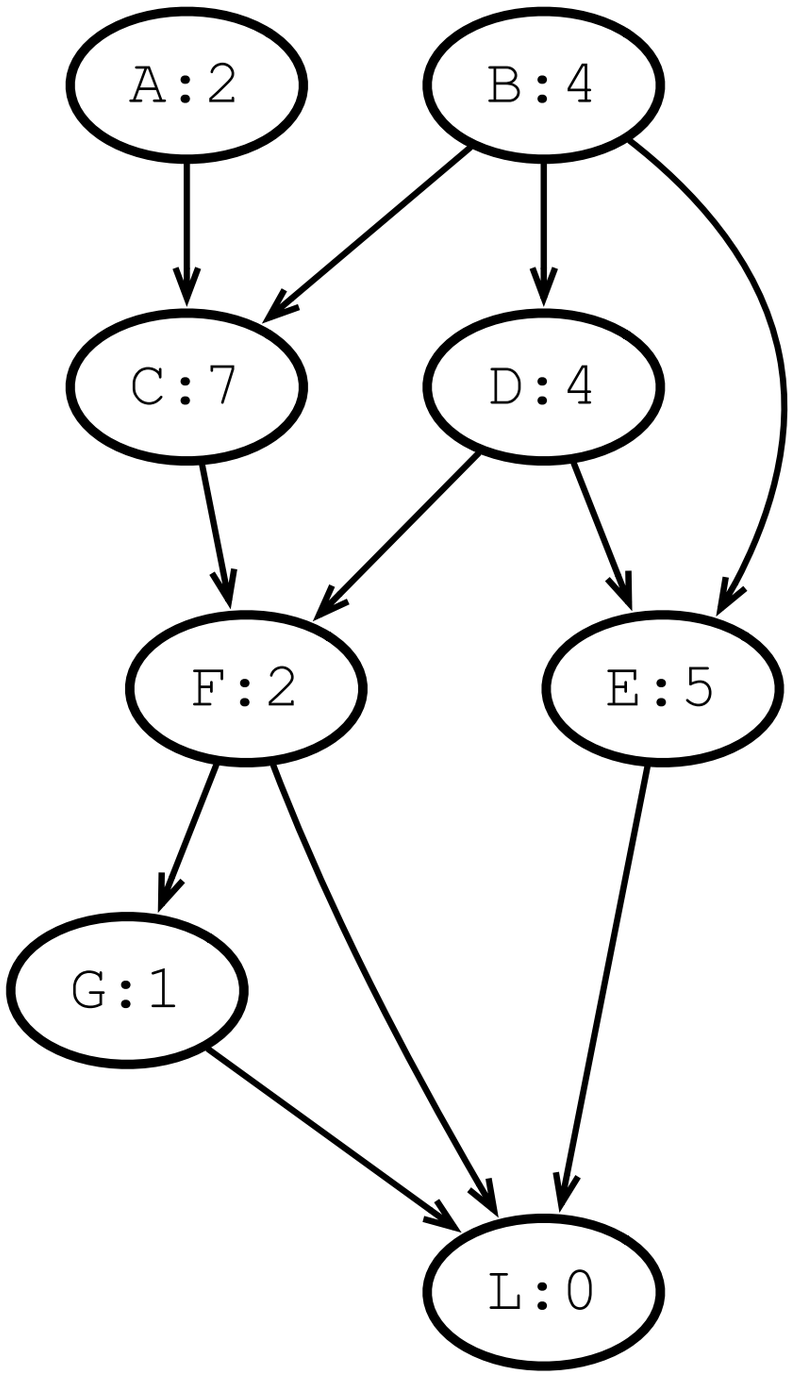
Found the destination node L.

Distance: 4.4

Predecessor Path:

L → F → D → B

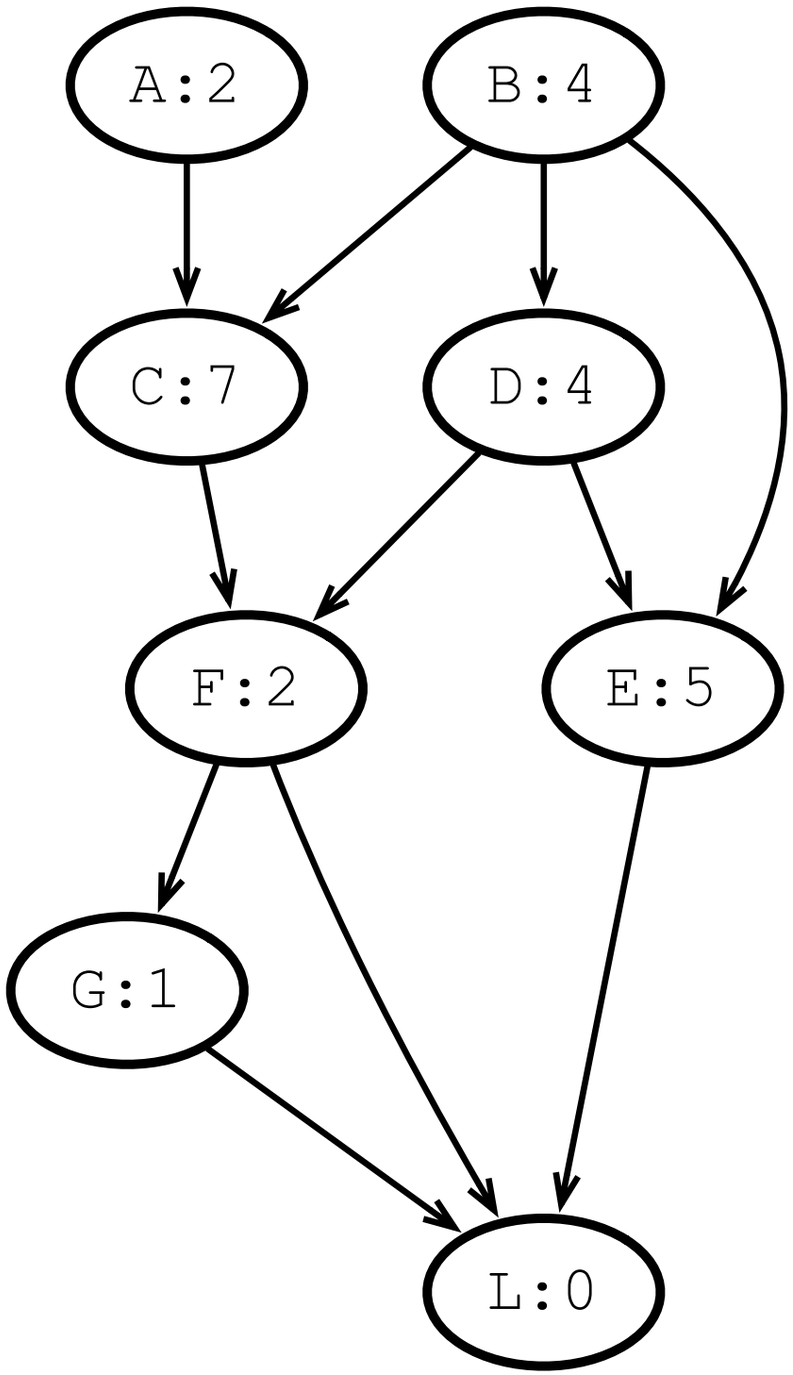
Critical Paths



Say you run a construction company, and you have a project with this DAG structure (these are always DAGs for the same reason course requirements are always DAGs).

Each node represents an activity (digging a hole, pouring concrete, etc.) Activities have durations in days.

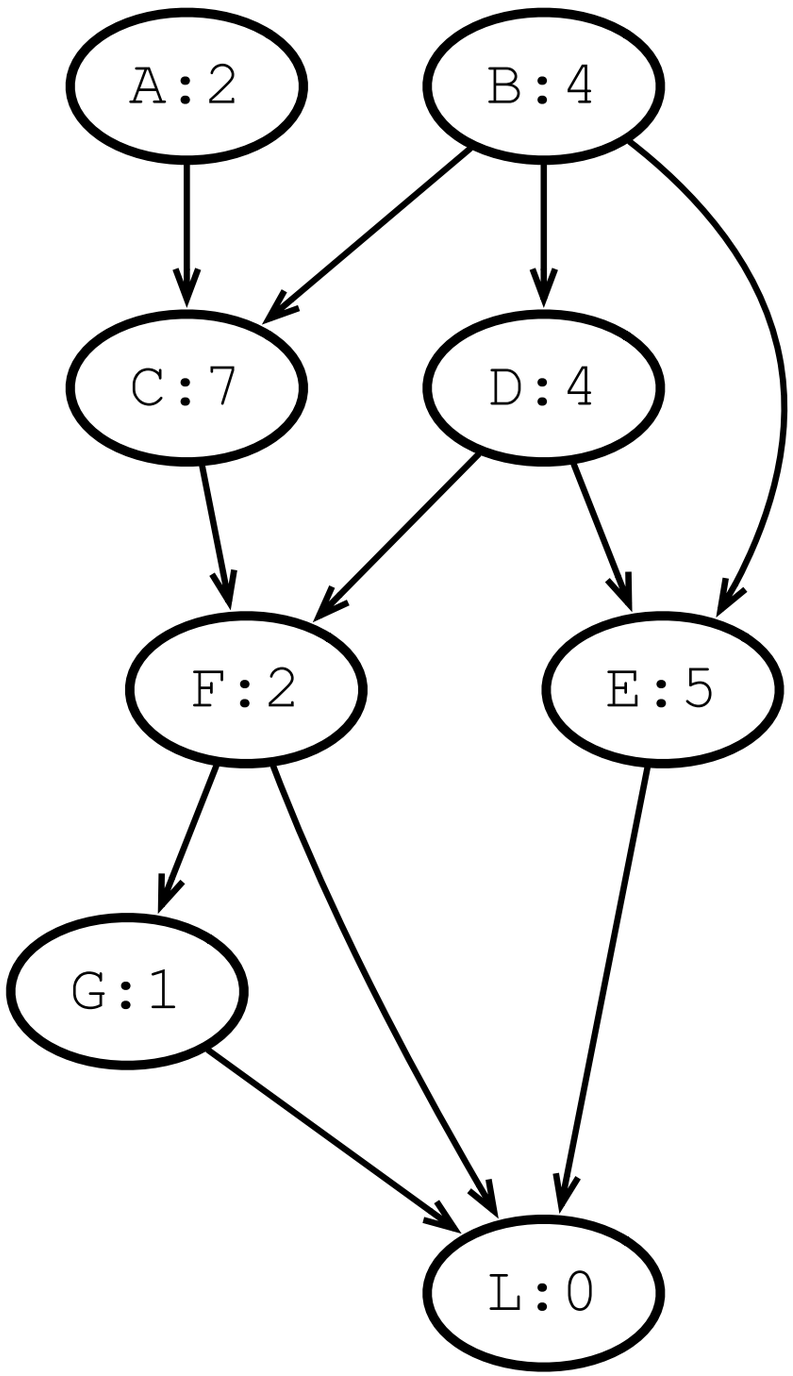
Critical Paths



You need to know how LONG the ENTIRE project is going to take.

Let's say that you can do the things that are on the same line at the same time. E.g. You can do A and B concurrently, and C and D concurrently. But you have to complete A *and* B before you can begin task C.

Critical Paths

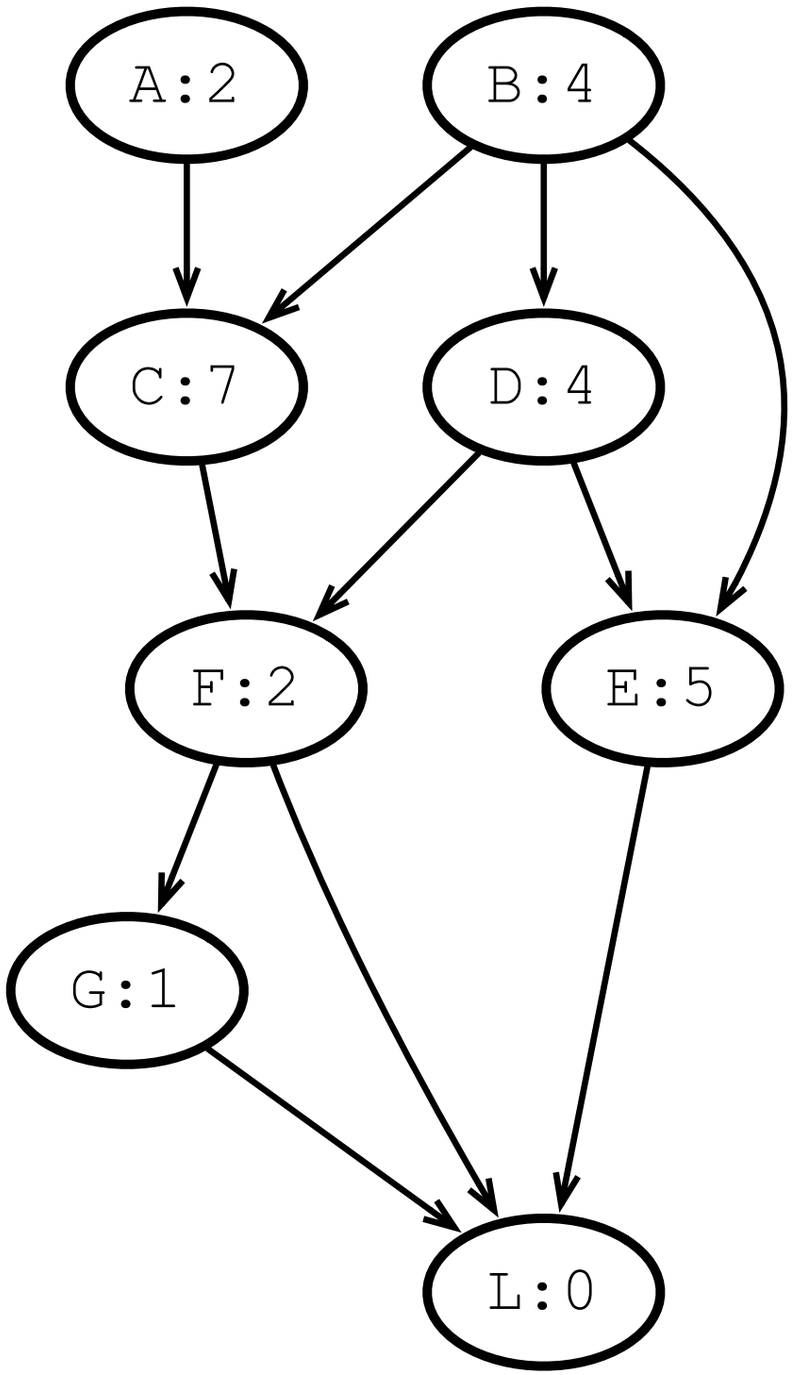


The overall project described by this DAG is called the *critical path*, and it is the sequence of tasks that requires the most time.

This is similar to what we just thought about with Dijsktra's algorithm.

Instead of edges, we look at nodes.
Instead of shortest path, we need the longest one.

Critical Paths



The last homework (after DAGs and FSMs) will involve this.

But to keep things interesting: the duration that each activity takes will actually be random, chosen from some distribution for each node.

E.g. Task D will take anywhere from 3 to 6 days, Task E will be from 4 to 6, and so on. Given uncertainty, can you compute the critical path?