CS 600.226: Data Structures

Michael Schatz

Nov 28, 2018

Lecture 35: Topological Sorting

Assignment 9: StringOmics

Out on: November 16, 2018

Due by: November 30, 2018 before 10:00 pm

Collaboration: None

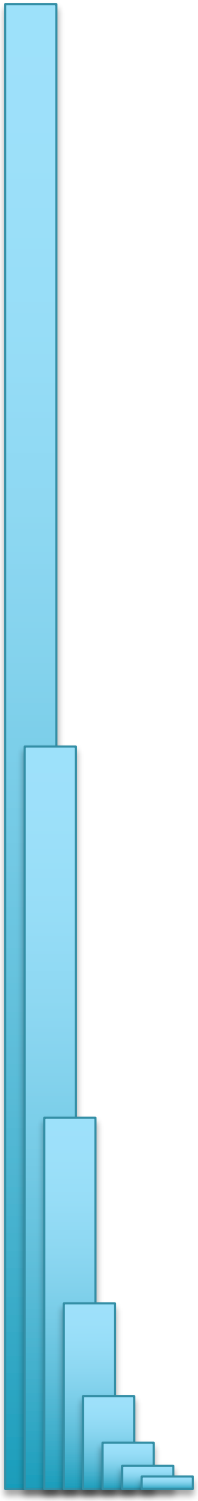
Grading:

Packaging 10%,
Style 10% (where applicable),
Testing 10% (where applicable),
Performance 10% (where applicable),
Functionality 60% (where applicable)

Overview

The ninth assignment focuses on data structures and operations on strings. In this assignment you will implement encoding and decoding using the Burrows Wheeler Transform as well as encoding and decoding in a simple form of run length encoding. In the final problem you will be asked to measure the space savings using run length encoding with and without applying the Burrows Wheeler Transform first.

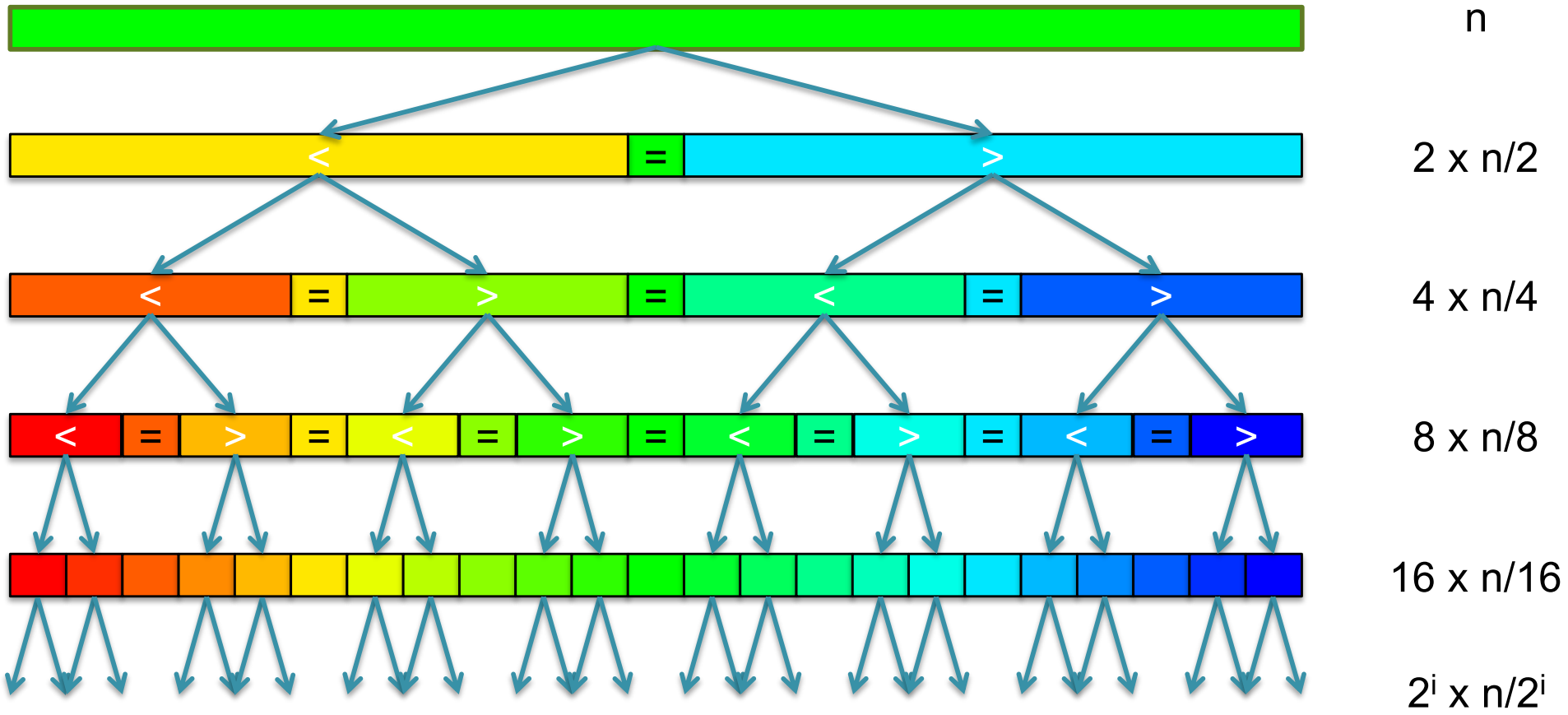
Remember: javac –Xlint:all & checkstyle *.java & Junit (No JayBee)



Part 1: Really Advanced Sorting

Quicksort

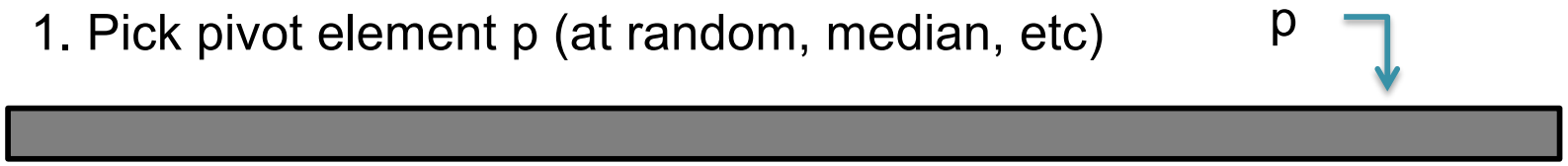
- Selection sort is slow because it rescans the entire list for each element
  - How can we split up the unsorted list into independent ranges?
  - Hint 1: Binary search splits up the problem into 2 independent ranges (hi/lo)
  - Hint 2: Assume we know the median value of a list



[How many times can we split a list in half?]

In-place Partitioning

1. Pick pivot element p (at random, median, etc)



2. Move p to front. Invariant: Elements 1..i-1 are <p; j+1..n-1 are >p

3. while (i < j), compare a[i] with p

x<p

x>p

4. Repeat …

In-place Partitioning

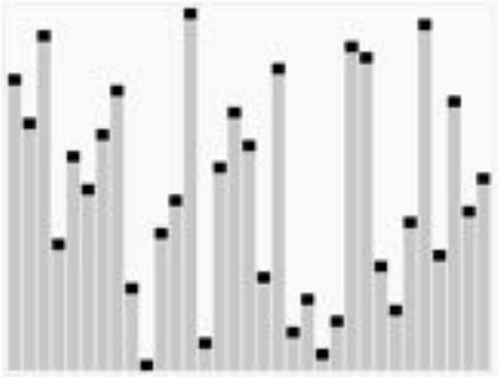
4. Repeat …

5. Finish

6. Swap

7. Recurse

Advanced Sorting Review

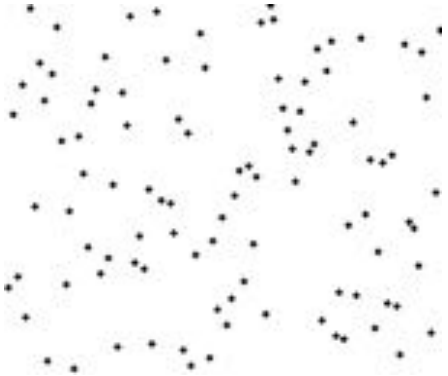


Heap Sort

Add everything to a heap, remove from biggest to smallest

O(n lg n) worst case

Big constants

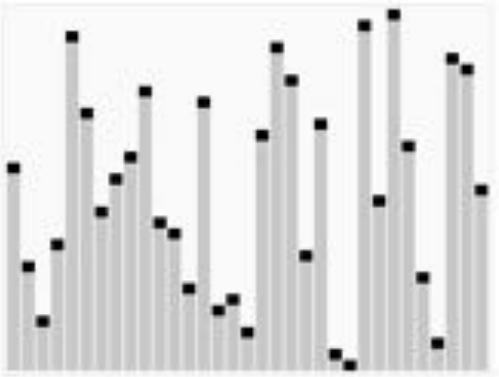


Merge Sort

Divide input into n lists, merge pairs of sorted lists as a tree

O(n lg n) worst case

O(n) space overhead



QuickSort

Recursively partition input into low/high based on a pivot

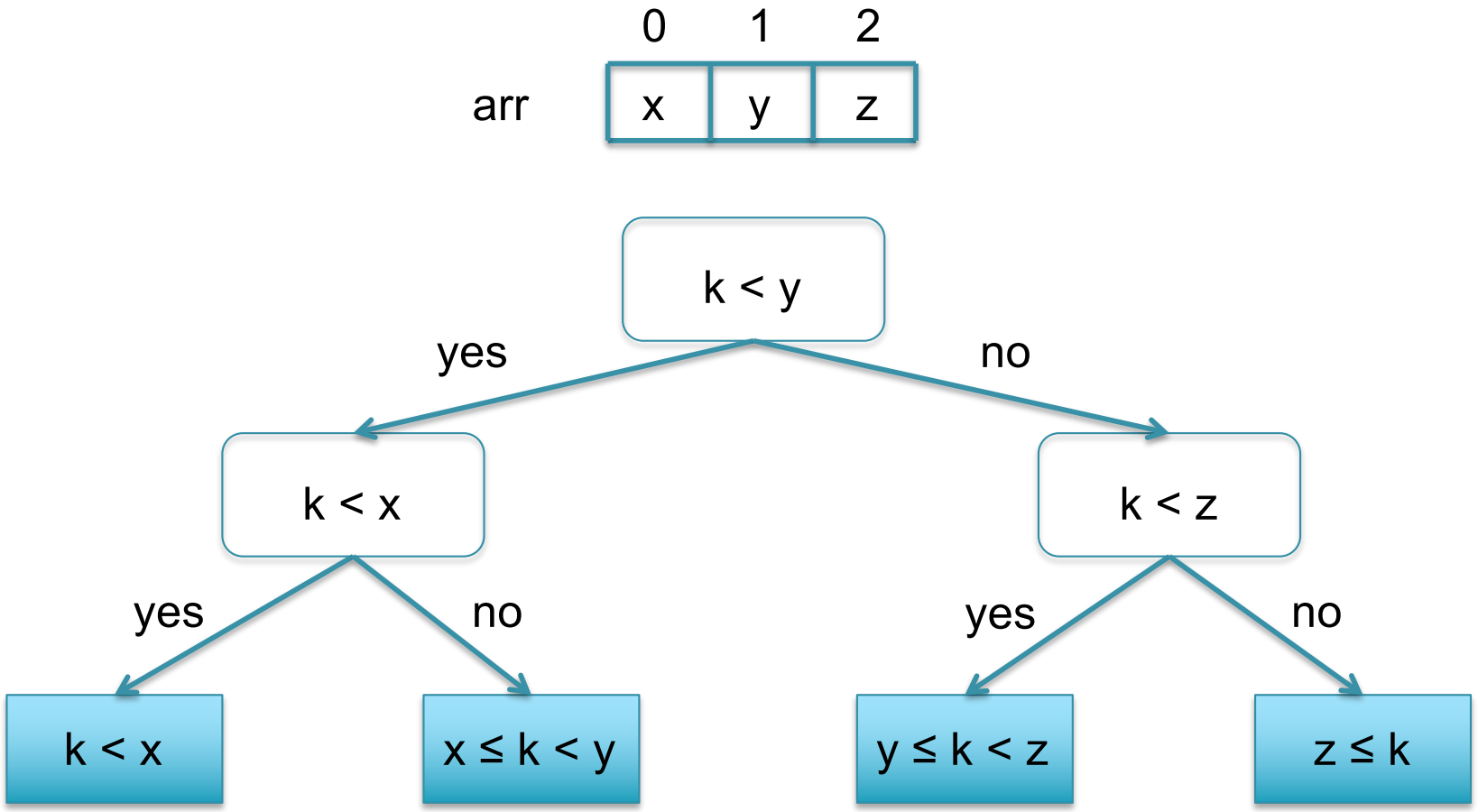
$O(n^2)$ worse case,

O(n lg n) typical

Very fast in practice

Decision Tree of Searching

How many comparisons are made to binary search in an array with 3 items?



The decision tree encodes the execution over all possible input values
The decision tree is a binary tree, each node encodes exactly 1 comparison
The decision tree for searching has n leaf nodes (since there are n "slots" for k)

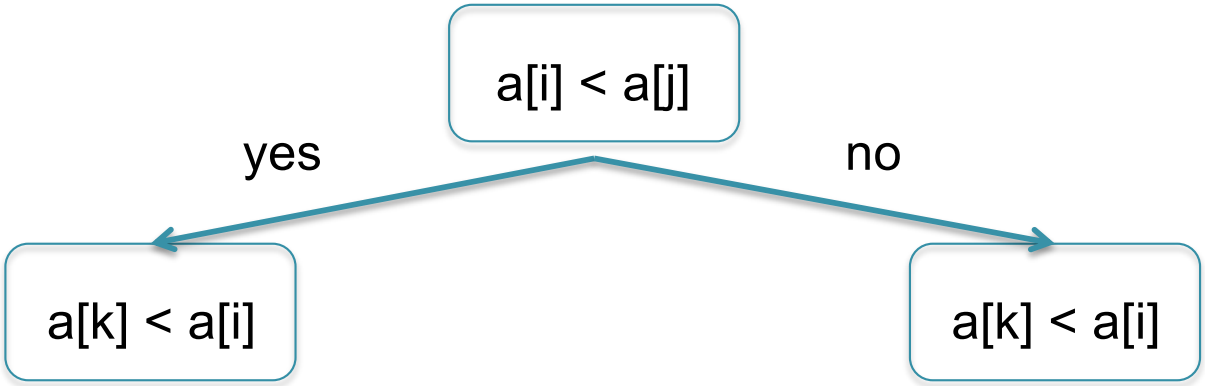
Searching by comparisons requires at least O(lg n) comparisons (lower bound)

Decision Tree of Sorting

Notice that sorting 3 items (a,b,c) may have 3 != 6 possible permutations

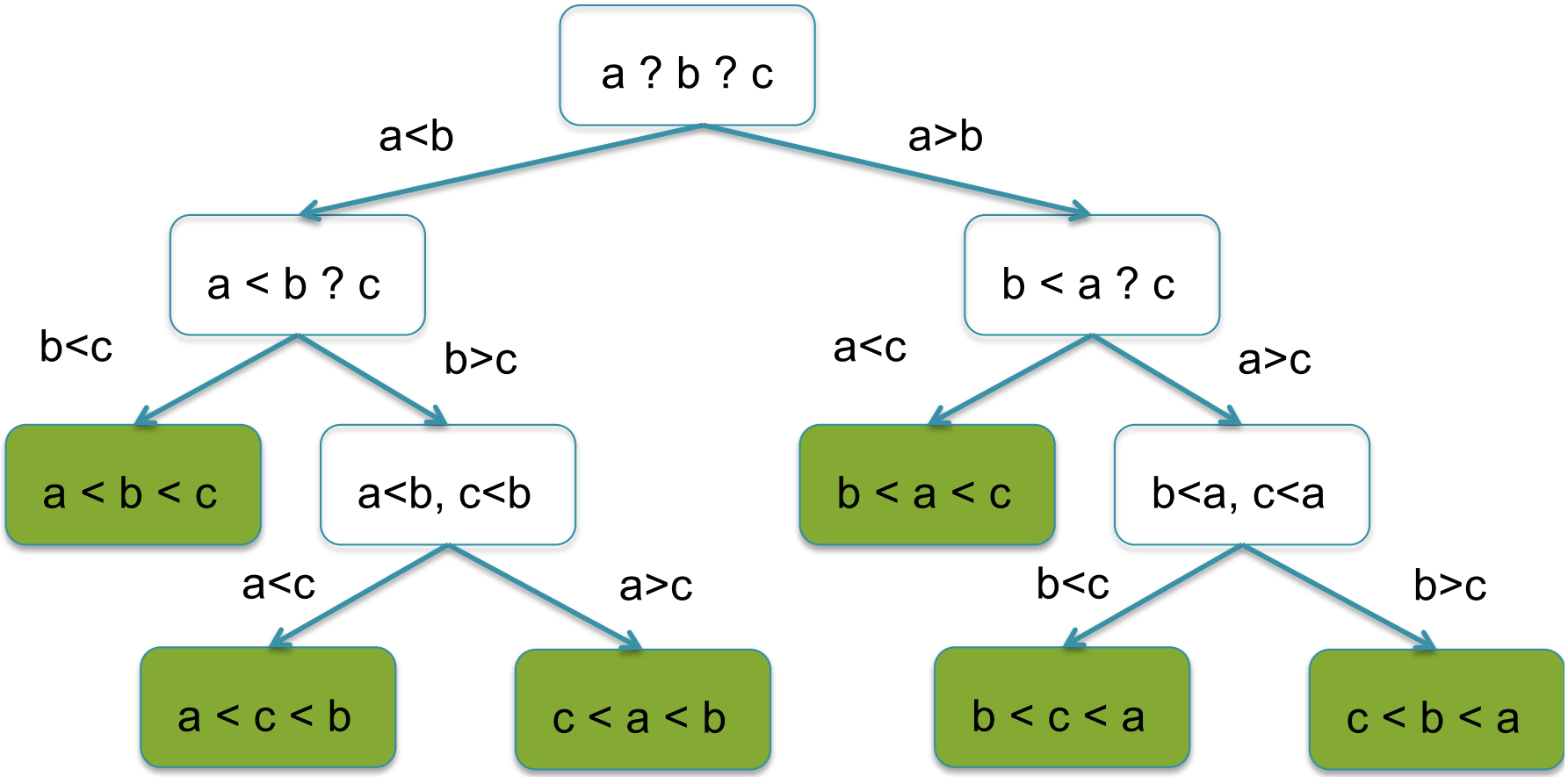
a < b < c
a < c < b
b < a < c
b < c < a
c < a < b
c < b < a

Initially we don't know which one of these permutations is the correctly sorted version, but we can make pairwise comparisons to figure it out



In the worst case, how many comparisons are needed?

Decision Tree of Sorting

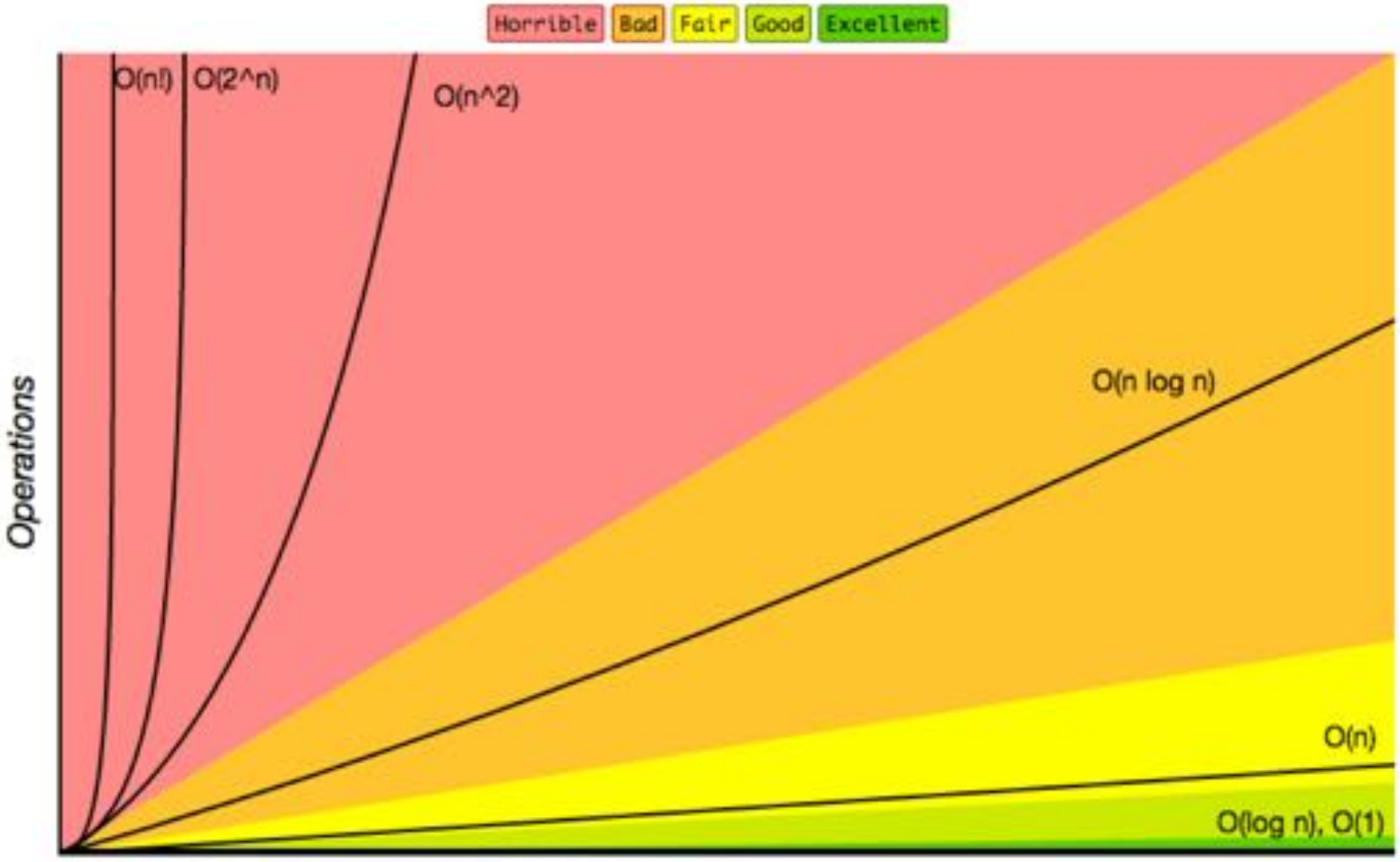


Notice that we have all 3 != 6 possibilities as leaf nodes

How tall is a tree with n! leaf nodes?

O(lg n!) whattt???

Growth of functions



Saying the minimum height is O(1) is true, but not interesting ☺
Want to say “minimum height is >= something” as tightly as possible

Decision Tree of Sorting

What is O(lg n!)

$$\lg(n!) = \lg(n * (n-1) * (n-2) * \ldots * 2 * 1)$$

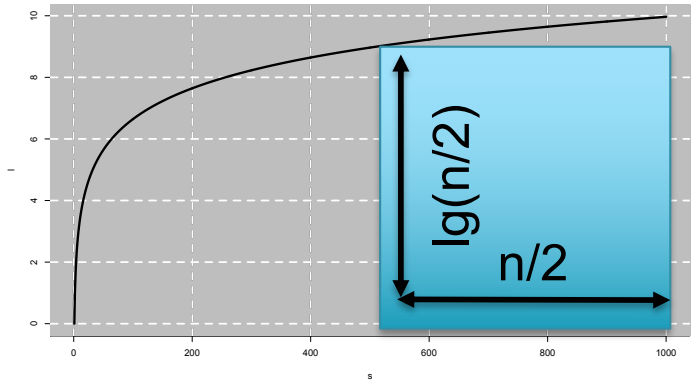
$$\lg(n!) = \lg(n) + \lg(n-1) + \lg(n-2) + \ldots + \lg(2) + \lg(1)$$

$$\lg(n!) \leq \lg(n) + \lg(n) + \lg(n) + \ldots + \lg(n) + \lg(n)$$

$$\lg(n!) \leq n \lg n$$

This is true, but the wrong direction to prove a lower bound.
Need to show lg(n!) ≥ something instead

$$\begin{aligned}
&= \log n + \log(n-1) + \log(n-2) + \cdots + \log 2 \\
&= \sum_{i=2}^{n} \log i \\
&= \sum_{i=2}^{n/2-1} \log i + \sum_{i=n/2}^{n} \log i \\
&\geq 0 + \sum_{i=n/2}^{n} \log \frac{n}{2} \\
&= \frac{n}{2} \cdot \log \frac{n}{2} \\
&= \Omega(n \log n)
\end{aligned}$$



Because lg grows so slowly,
Just sum from n/2 to n

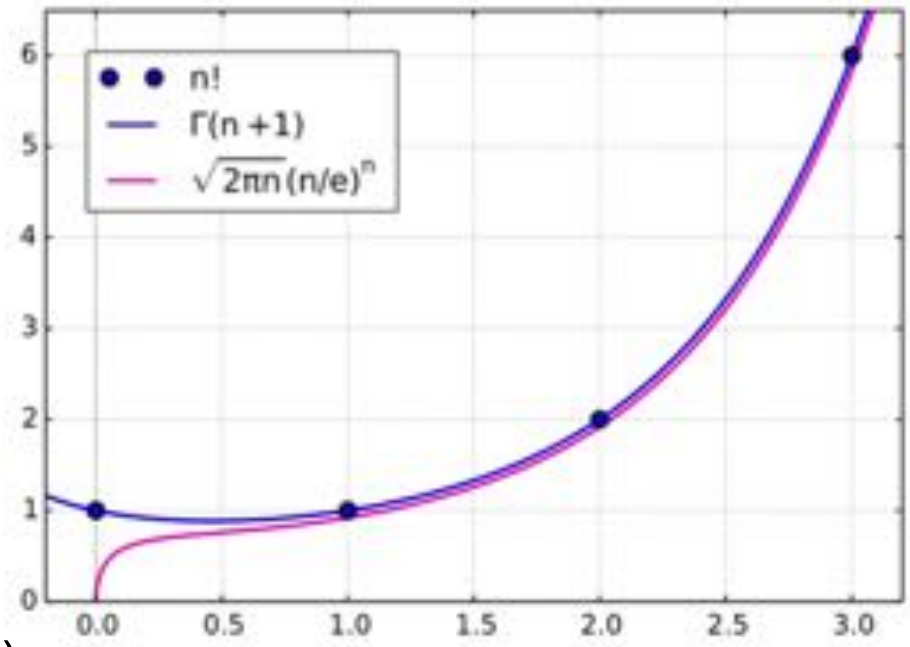
***Any* comparison based sorting algorithm requires at least O(n lg n)**

Alternate Analysis

What is O(lg n!)

Stirling's aproximation

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$



lg(n!) ~ lg((√2πn) (n/e)^n)

lg(n!) ~ lg(√2πn) + lg((n/e)^n)

lg(n!) ~ lg((2πn)^{½}) + lg((n/e)^n)

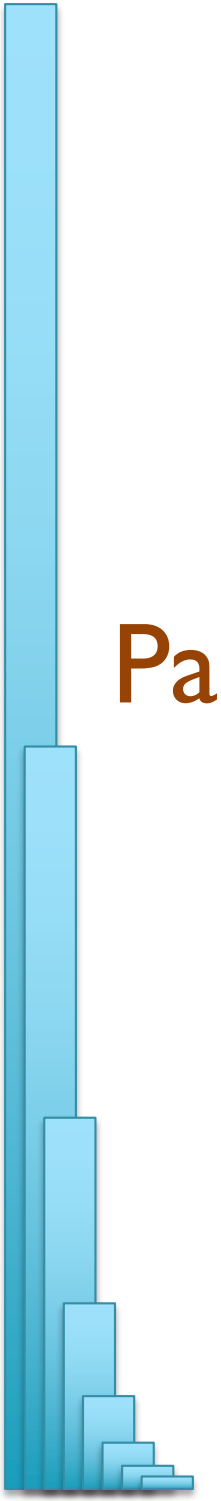
lg(n!) ~ ½ lg(2πn) + n lg(n/e)

lg(n!) ~ ½ lg(2πn) + n lg(n) - n lg(e)

lg(n!) ~ O(lg n + n lg n - n)

lg(n!) ~ O(n lg n)

Same result, if you have Stirling's approximation memorized ☺



Part 2: Really Really Advanced Sorting

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data ☺

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

6,13,14,19,29,31,39,50,61,63,64,78

```
for(i = 1 to 100) { range[i] = 0; }
for(i = 1 to n) { range[list[i]] = 1; }
for(i = 1 to 100) { if (range[i] == 1){print i}}
```

<- Mark it was present

What if 13 occurred twice?

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

6,13,14,19,29,31,39,50,61,63,64,78

```
for(i = 1 to 100) { range[i] = 0; }
for(i = 1 to n) { range[list[i]]++; }
for(i = 1 to 100) { for (j = 0; j < range[i]; j++){print i}}
```

<- Counting sort

Sorting in Linear Time

Counting Sort

Input:
0,4,2,2,0,0,1,1,0,1,0,2,4,2

Count Array:

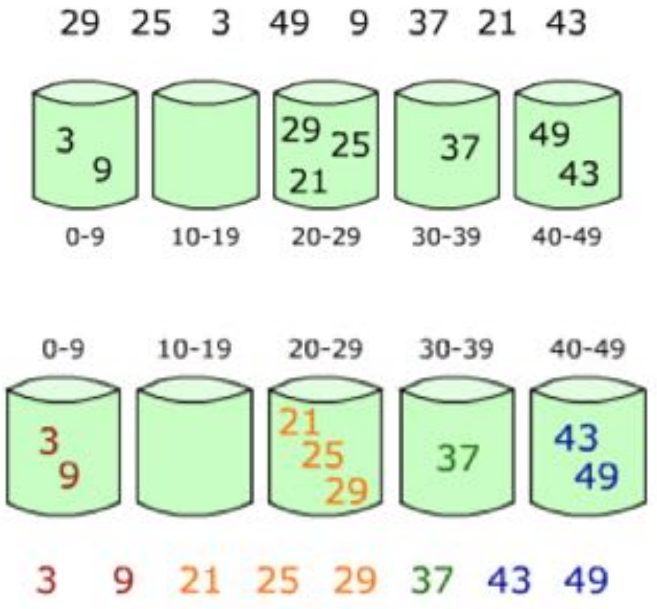
```
Idx: 0 1 2 3 4
Cnt: 5 3 4 0 2
```

Sorted:
0,0,0,0,0,1,1,1,2,2,2,2,4,4

Uses the input values as indices into array of counts

Very fast, but requires a (potentially very) large array to store the counts

Bucket Sort



Divide data into k buckets, sort data within each bucket, and output in sorted order

Sort in O(n) if k is O(n) and O(1) items per bucket

Sorting in Linear Time

What does bucket sort remind you of?

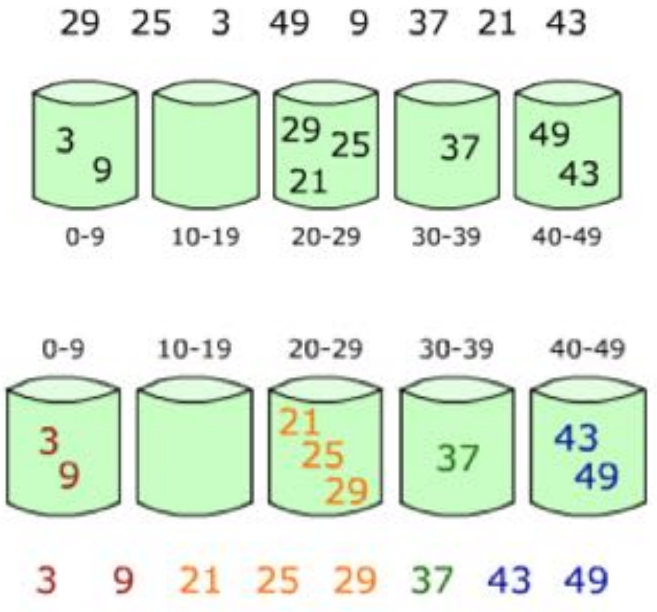
Just like hashing, want a small (~1) number of items per bucket, meaning number of buckets must be at least n (but not too much more than n)

Just like hashing, gets bad performance if the values to sort are clustered together

More advanced radix sort uses bucket sort to achieve linear time integer sorting over essentially unlimited range of integer values

- Range may be a polynomial in n, even up to n^{100}

Bucket Sort

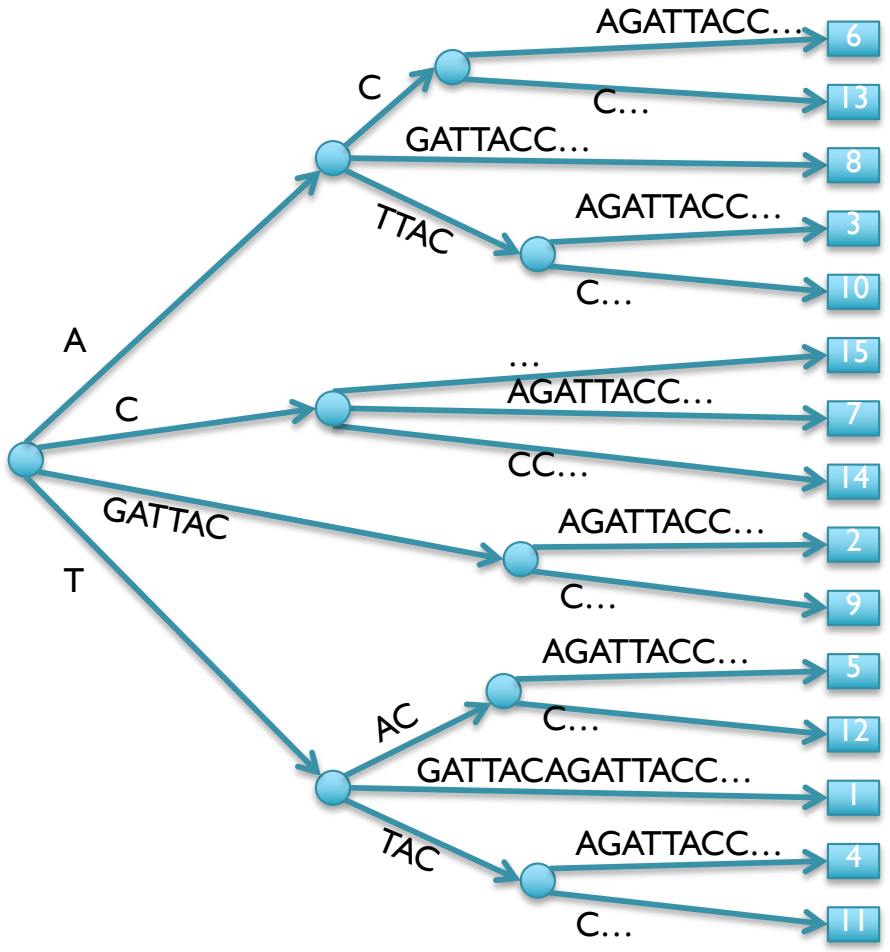


Divide data into k buckets, sort data within each bucket, and output in sorted order

Sort in O(n) if k is O(n) and O(1) items per bucket

Sorting Suffixes in linear Time

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

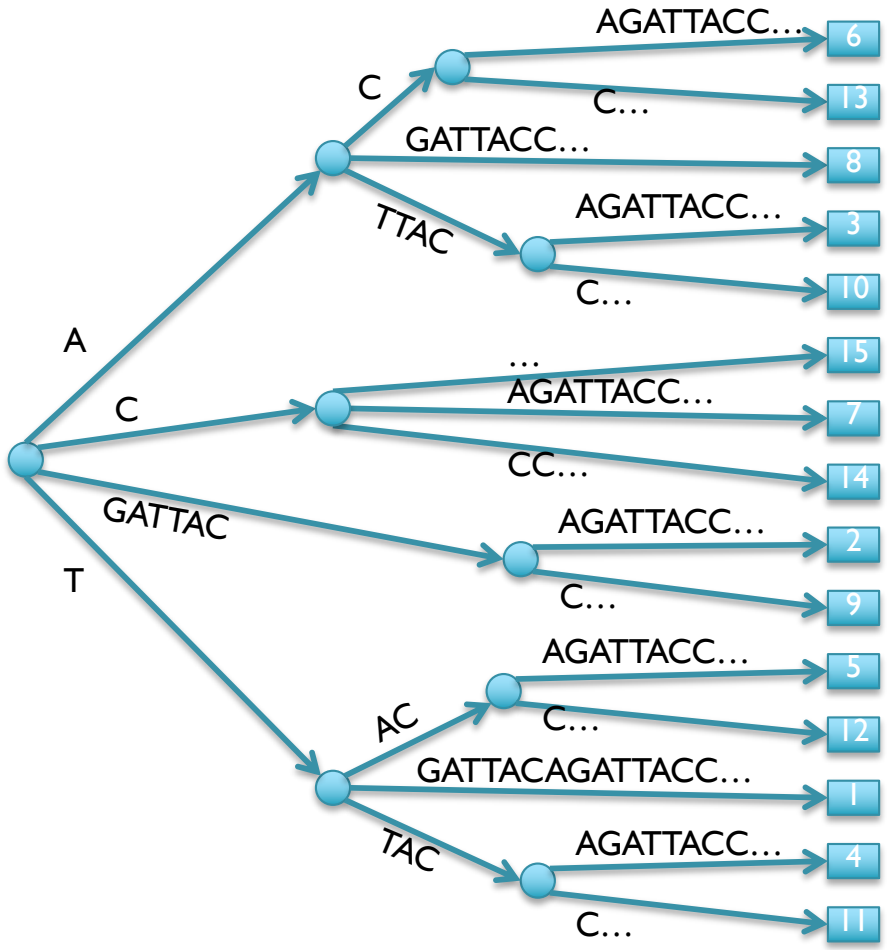


Suffix Tree = Tree of suffixes (indexes **all** substrings of a sequence)

- 1 Leaf ($) for each suffix, path-label to leaf spells the suffix
- Nodes have at least 2 and at most 5 children (A,C,G,T,$)

Sorting Suffixes in linear Time

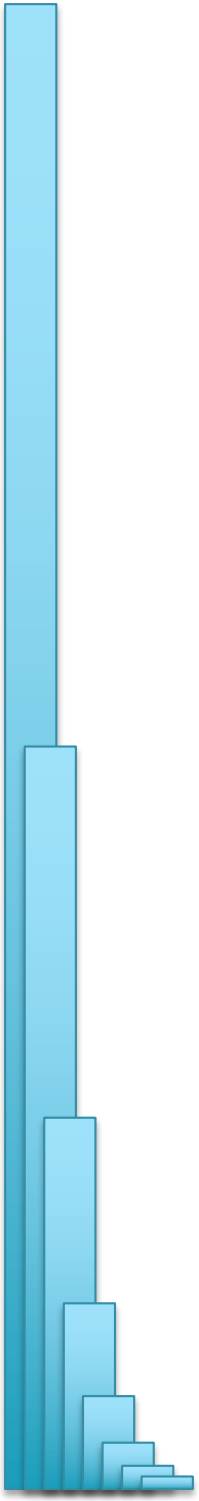
#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11



Linear time construction takes about a month of grad school to understand ☺

Can search for any query is linear time (independent of the size of the text)

Can compute the suffix array or BWT in linear time

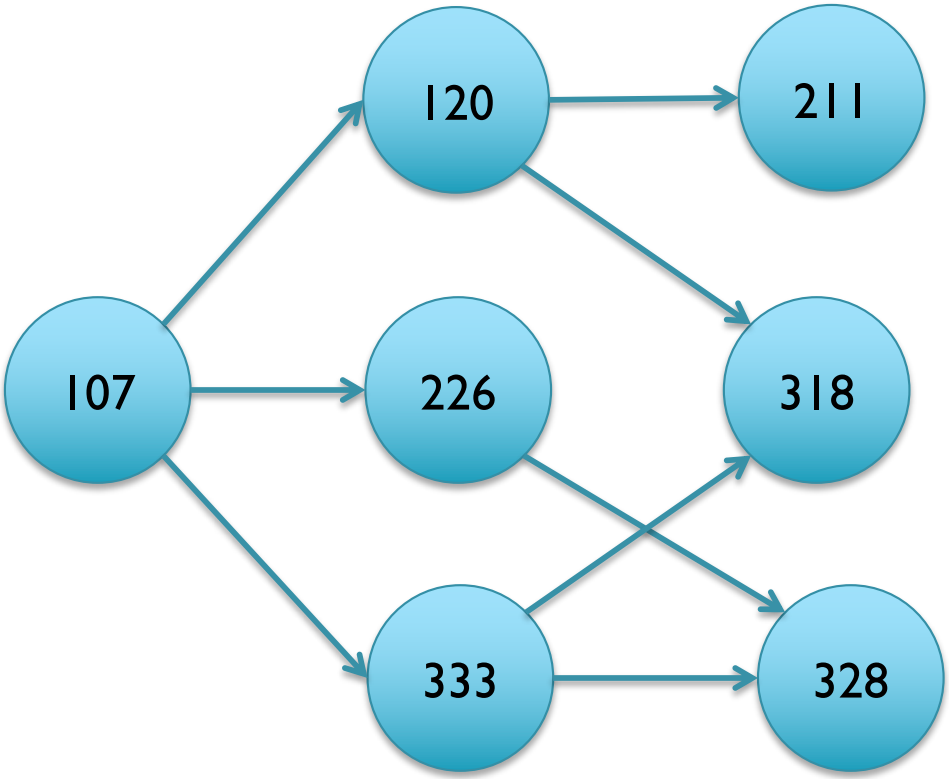


Part 3: Topological Sort aka Sorting graphs

Scheduling Challenges

- Consider the following class prerequisites:
  - 107 is required before taking 120, 226, or 333
  - 120 is required before taking 211 or 318
  - 226 is required before taking 328
  - 333 is required before taking 318 or 328

What courses should I take now so I can take all of these by senior year? ☺

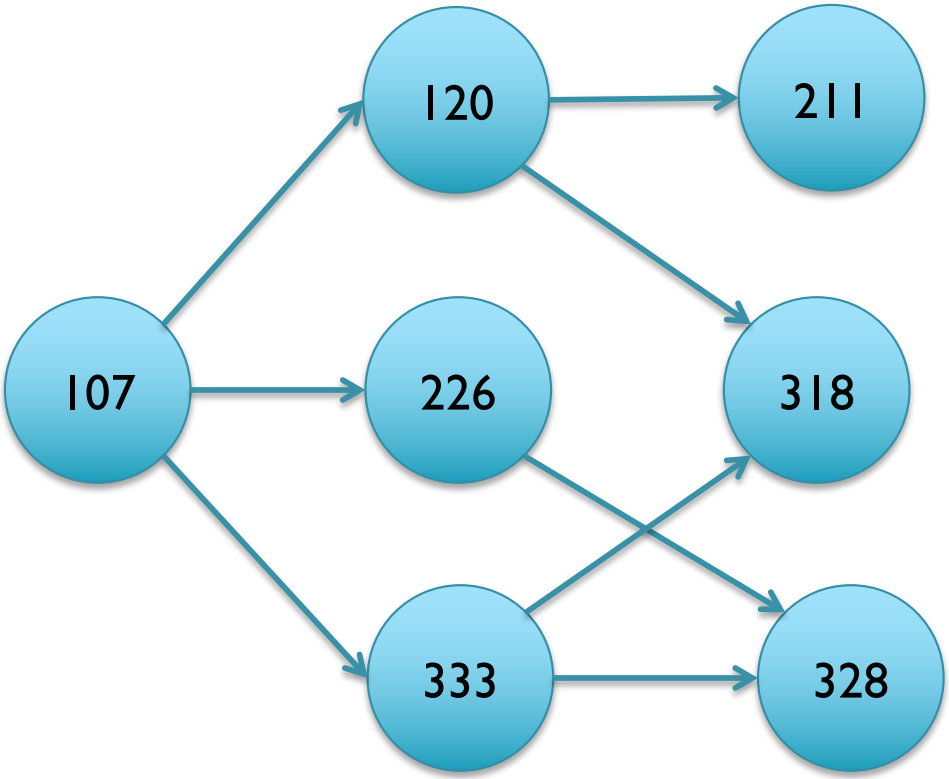


Any ideas?

Scheduling Challenges

- Consider the following class prerequisites:
  - 107 is required before taking 120, 226, or 333
  - 120 is required before taking 211 or 318
  - 226 is required before taking 328
  - 333 is required before taking 318 or 328

What courses should I take now so I can take all of these by senior year? ☺



Topological sorting

Analysis of directed, acyclic graphs (DAGs) with no weights on the edges

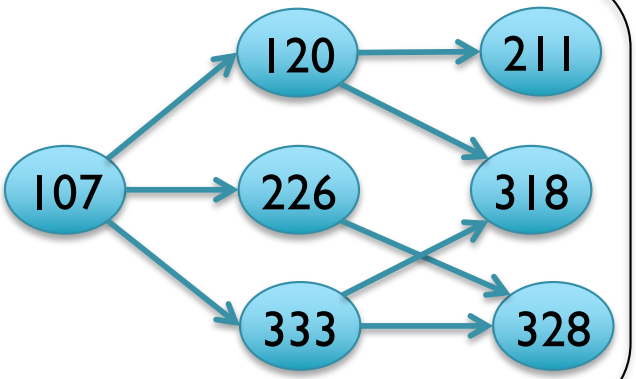
Vertices represent "***tasks***", edges represent some "***before***" relationship

Goal: Find a valid sequence of tasks that fulfill the ordering constraints

Topological sort

1. ***Pick a node that has no prior constraints***

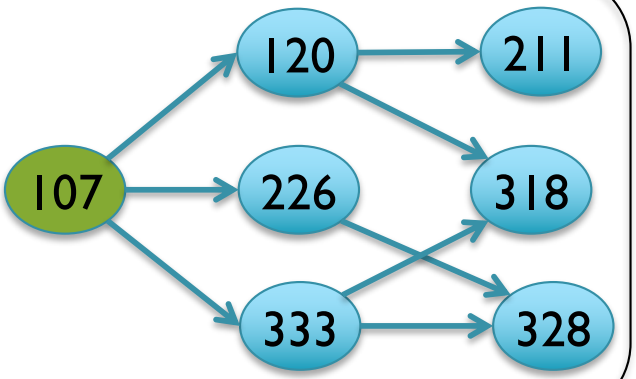
Since we are working with DAGs there must be at least 1 such node with indegree 0



Topological sort

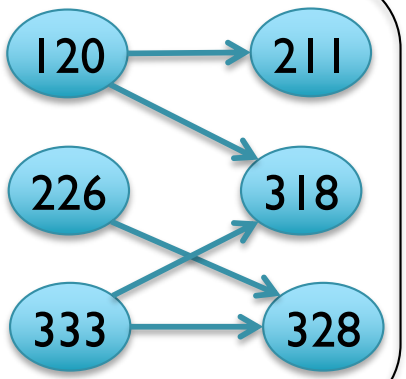
1. ***Pick a node that has no prior constraints***

Since we are working with DAGs there must be at least 1 such node with indegree 0



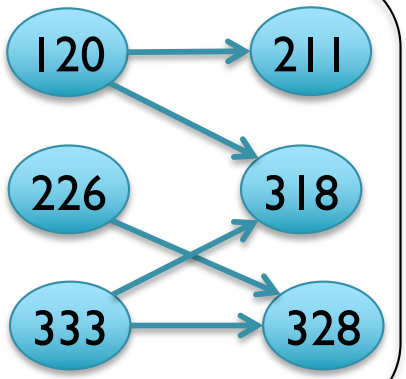
2. Remove that node from the DAG and all incident edges

The node you just removed starts a valid sequence of classes: 107



3. Arbitrarily pick another "free" node

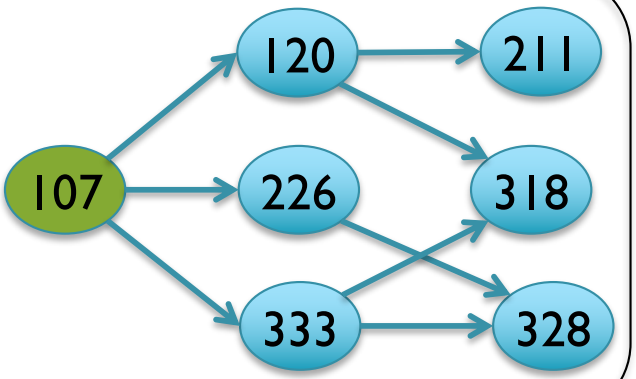
Removing one node will generally "open up" one or more additional nodes



Topological sort

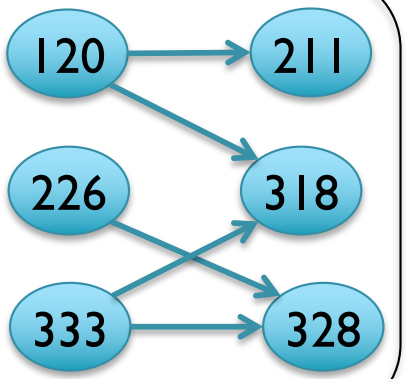
1. ***Pick a node that has no prior constraints***

Since we are working with DAGs there must be at least 1 such node with indegree 0



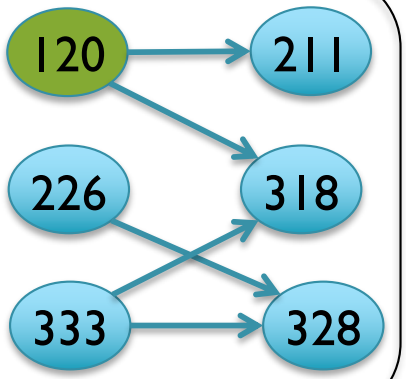
2. Remove that node from the DAG and all incident edges

The node you just removed starts a valid sequence of classes: 107



3. Arbitrarily pick another "free" node

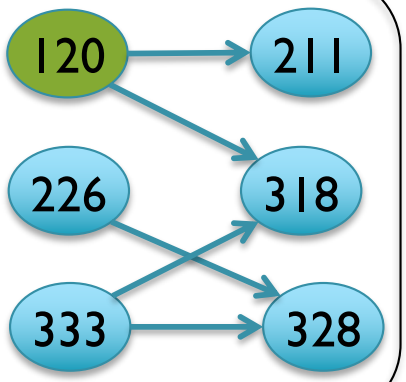
We could pick 120, 226, or 333. Picking 120 immediately opens up 211, but no new courses open up by picking 226 or 333



Topological sort

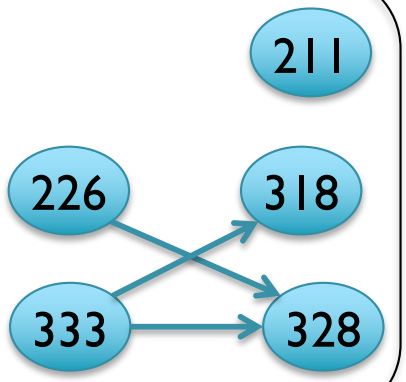
3. Arbitrarily pick another "free" node

We could pick 120, 226, or 333. Picking 120 immediately opens up 211, but no new courses open up by picking 226 or 333



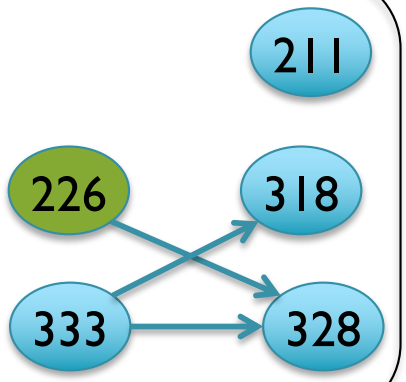
4. Remove that node

The valid sequence grows: 107, 120



5. Repeat!

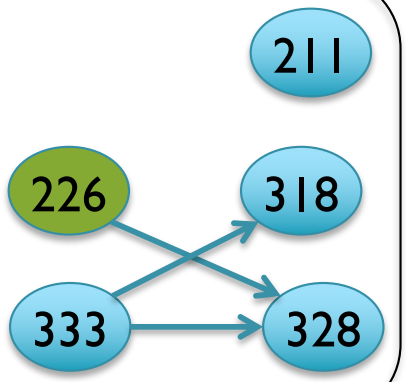
The valid sequence grows: 107, 120



Topological sort

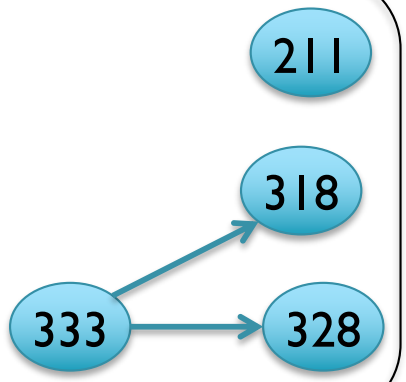
5. Repeat!

The valid sequence grows: 107, 120



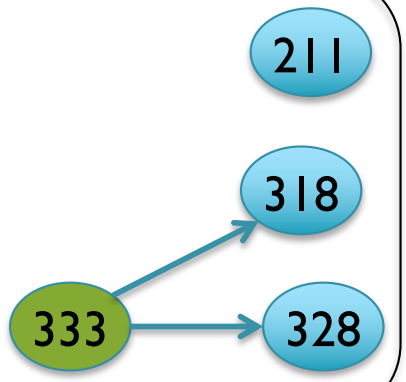
6. Repeat!

The valid sequence grows: 107, 120, 226



7. Repeat!

The valid sequence grows: 107, 120, 226



Topological sort

8. Repeat!

The valid sequence grows: 107, 120, 226, 333

211

318

328

9. Repeat!

The valid sequence grows: 107, 120, 226,333, 328

211

318

10. Repeat!

The valid sequence grows: 107, 120, 226, 333, 328, 211

318

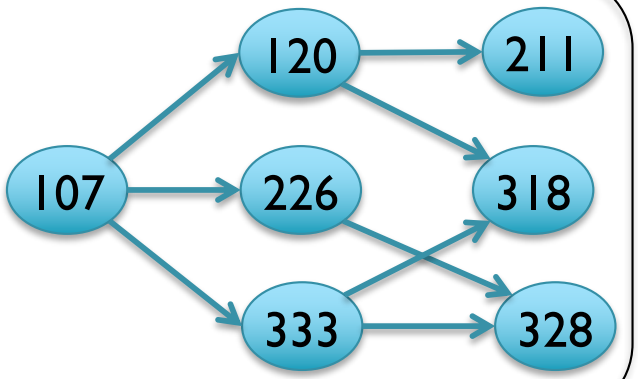
Topological sort

10. Repeat!

The valid sequence grows: 107, 120, 226, 333, 328, 211, 318

We just found one of potentially many valid sequences of courses

Alt1: 107, 226, 120, 211, 333, 328, 318
Alt2: 107, 333, 226, 328, 120, 318, 211



What is the running time to find a valid sort?

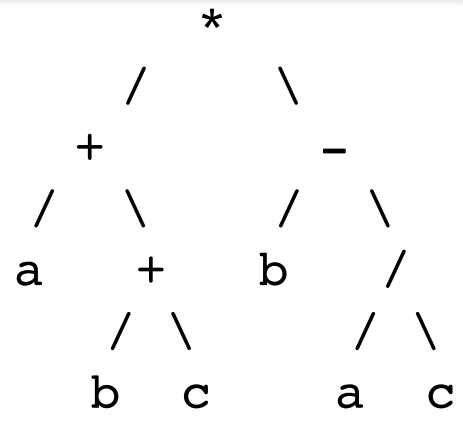
Linear Time: O(|V|) + O(|E|)

Data Dependency Graphs

Please evaluate this expression: (a + b + c) * (b – (a / c))

Figure out the prerequisites for each pairwise step: load the variables first, multiple and divide before addition and subtraction

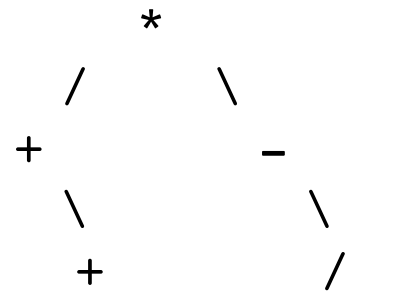
Hooray: the expression tree will do this for us, working from the bottom up.



We clearly need to load the values a,b,c first, but should we add next or divide next?

Under an abstract compute model, it doesn't matter which comes first.

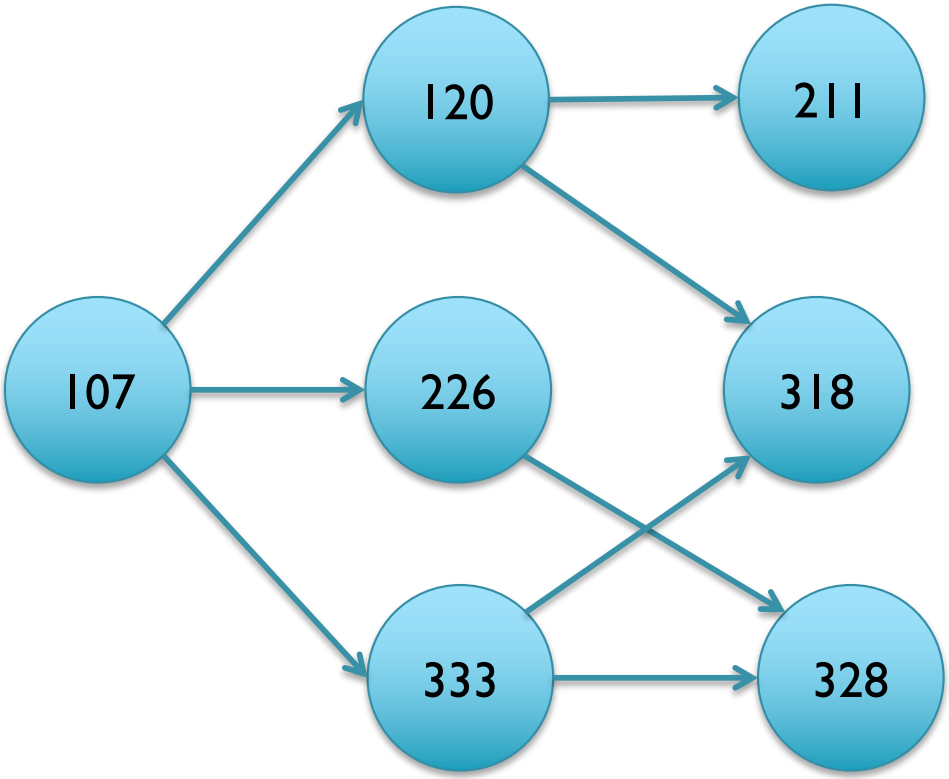
In practice division (8 units) takes much longer than addition/subtraction (1 unit) or multiplication (2 units) and may be offloaded to a separate functional unit that can work in parallel



Left first: 1+1+8+1+2=13

Right first: 8+0+0+1+2=11

Topological Sorting

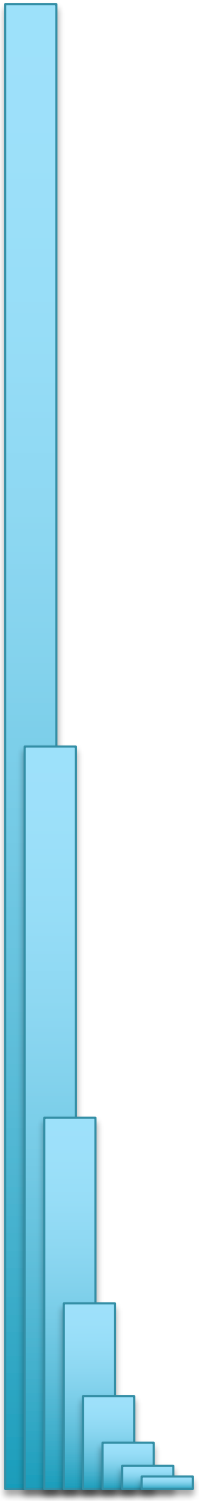


Topological sorting

Analysis of directed, acyclic graphs (DAGs) with no weights on the edges

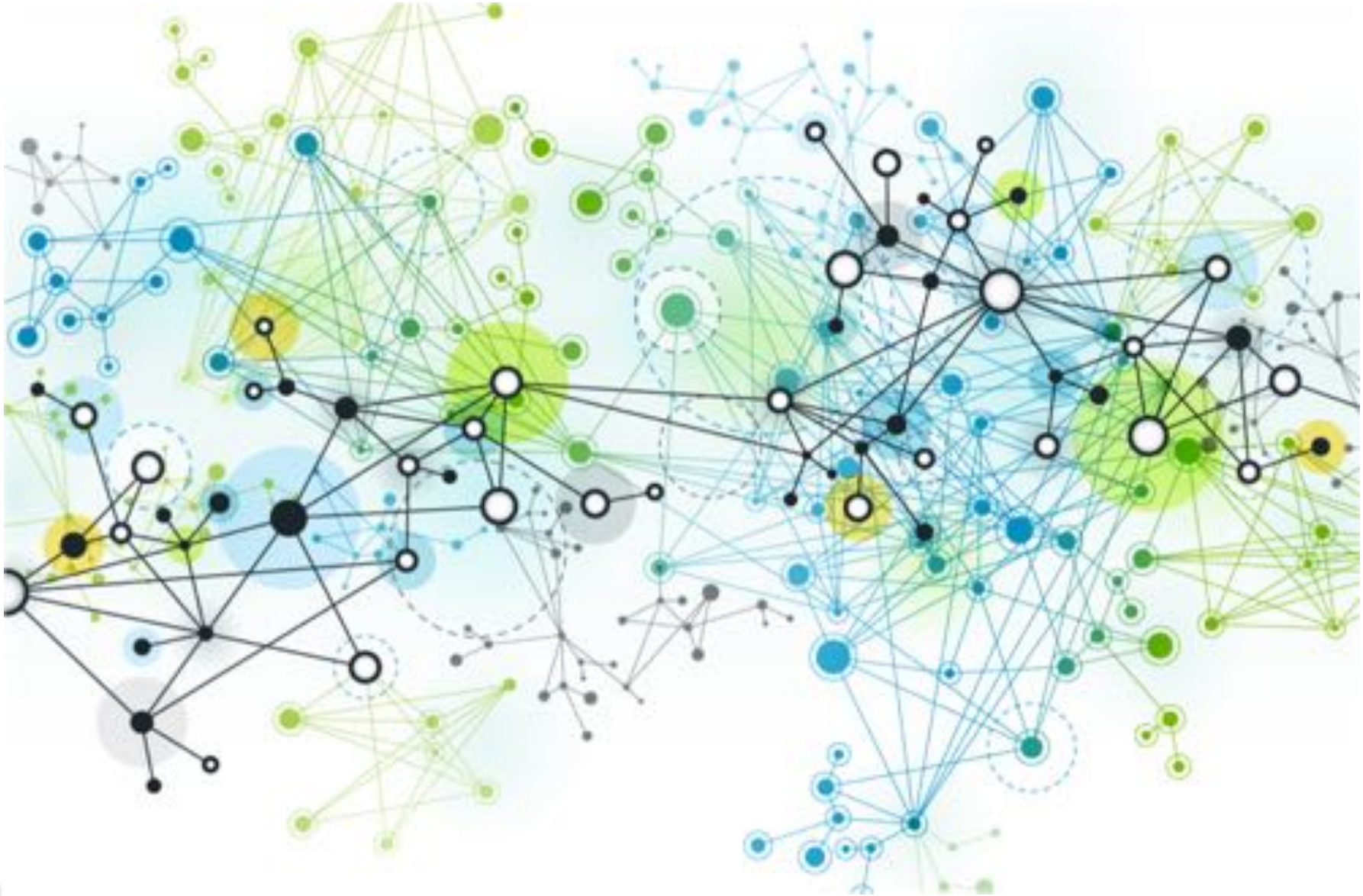
Vertices represent "***tasks***", edges represent some "***before***" relationship

Goal: Find a valid sequence of tasks that fulfill the ordering constraints



Part 4: Dijkstra's Algorithm aka Shortest Path Revisited

Graphs are Everywhere!



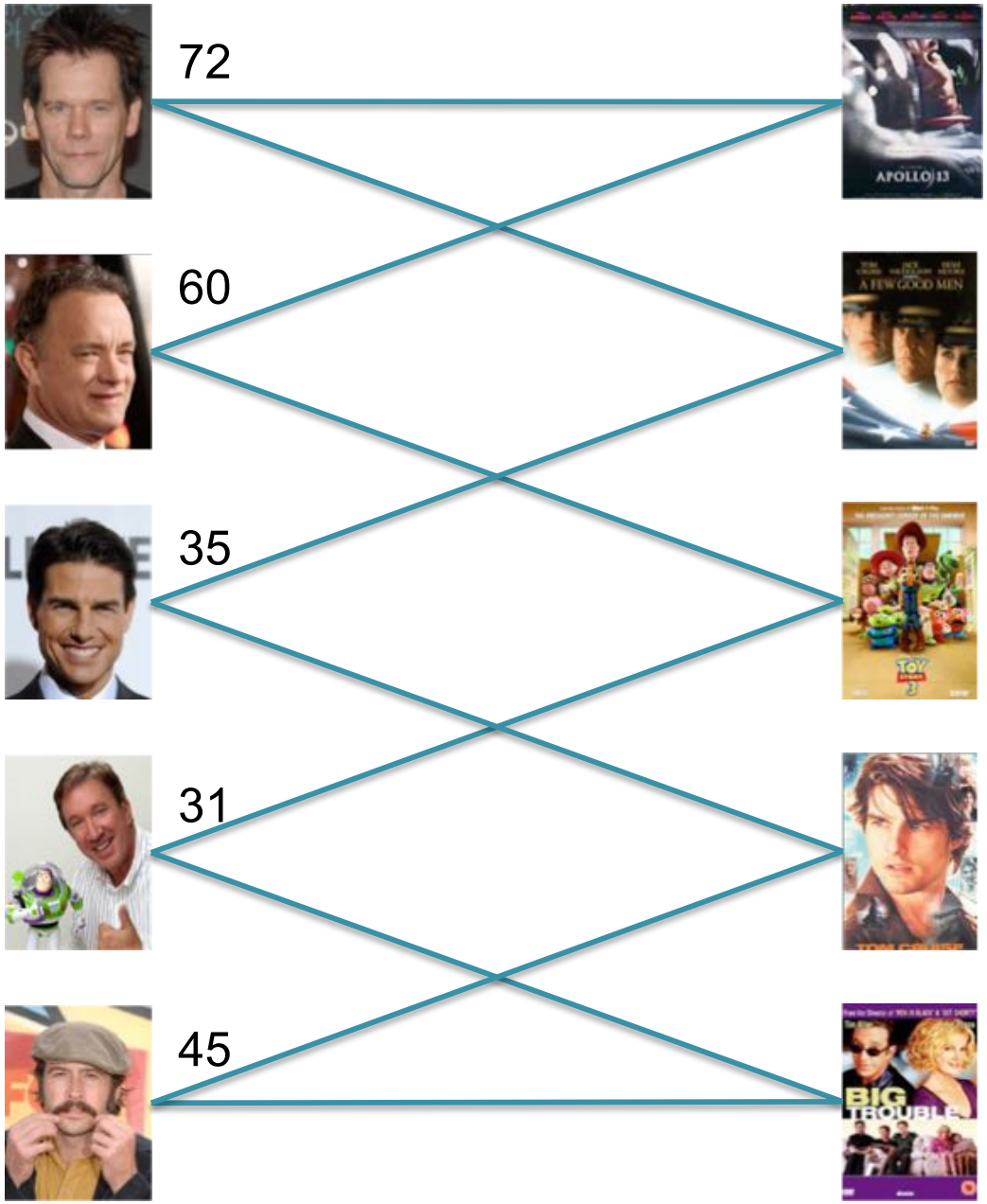
One of the most important queries
is finding the (shortest) path from A to B!

Kevin Bacon and Bipartite Graphs

Find the ***shortest*** path from Kevin Bacon to Jason Lee

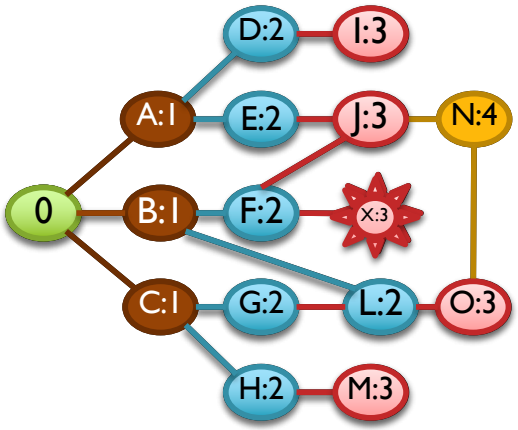
Breadth First Search: 4 hops

Bacon Distance: 2



BFS

```
BFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
   cur = list.getBegin()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```



0

A,B,C
B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

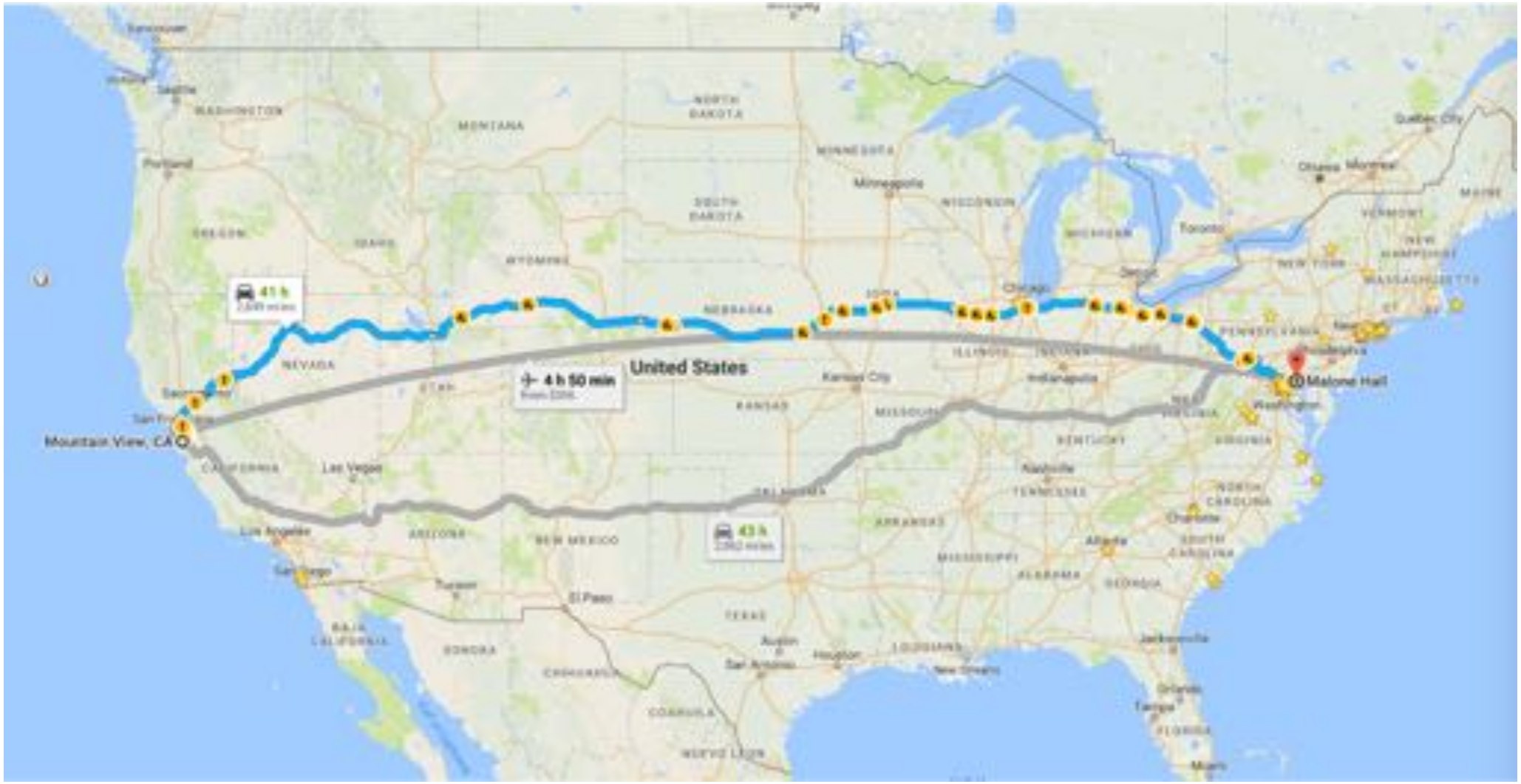
N

[How many nodes will it visit?]

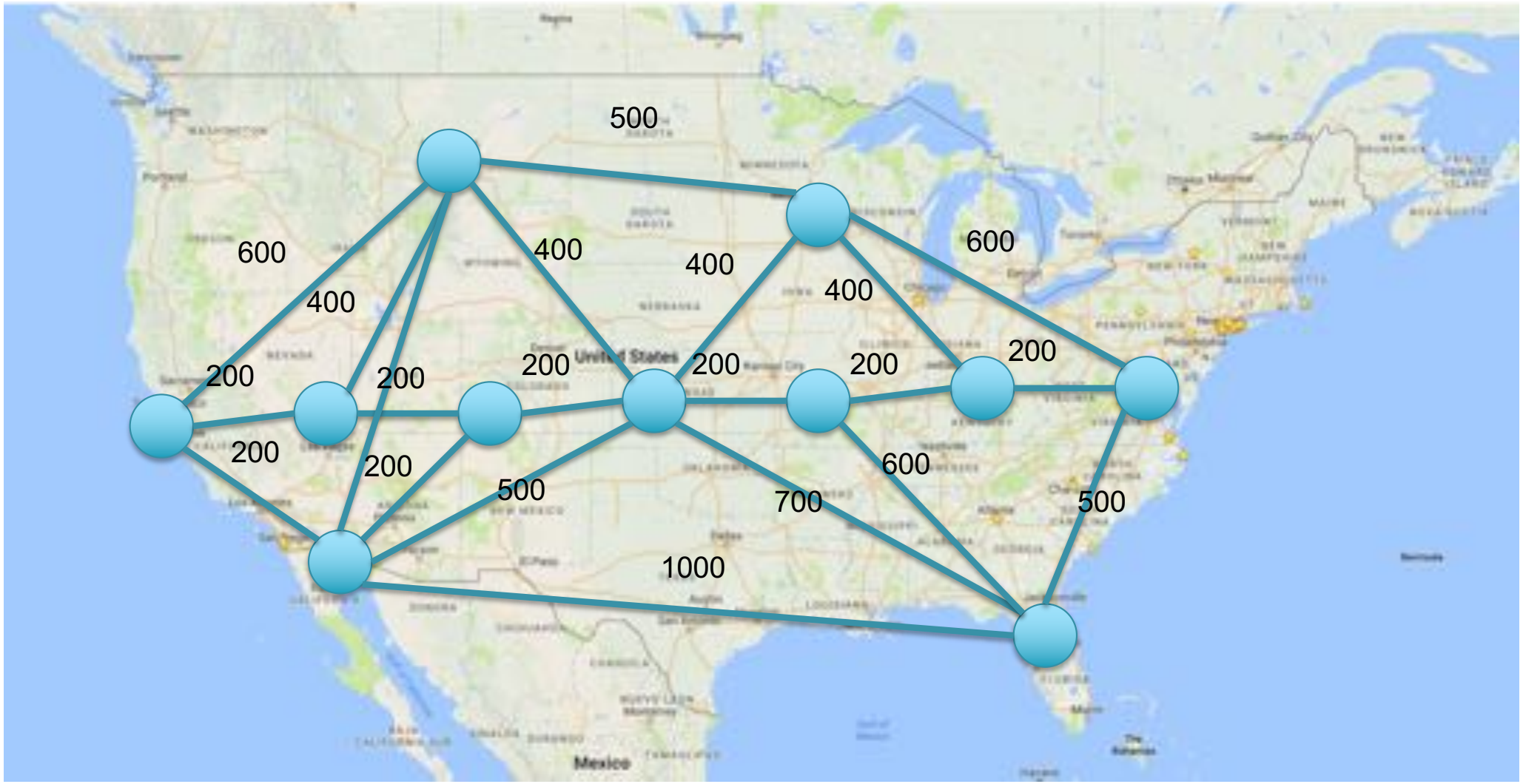
[What's the running time?]

[What happens for disconnected components?]

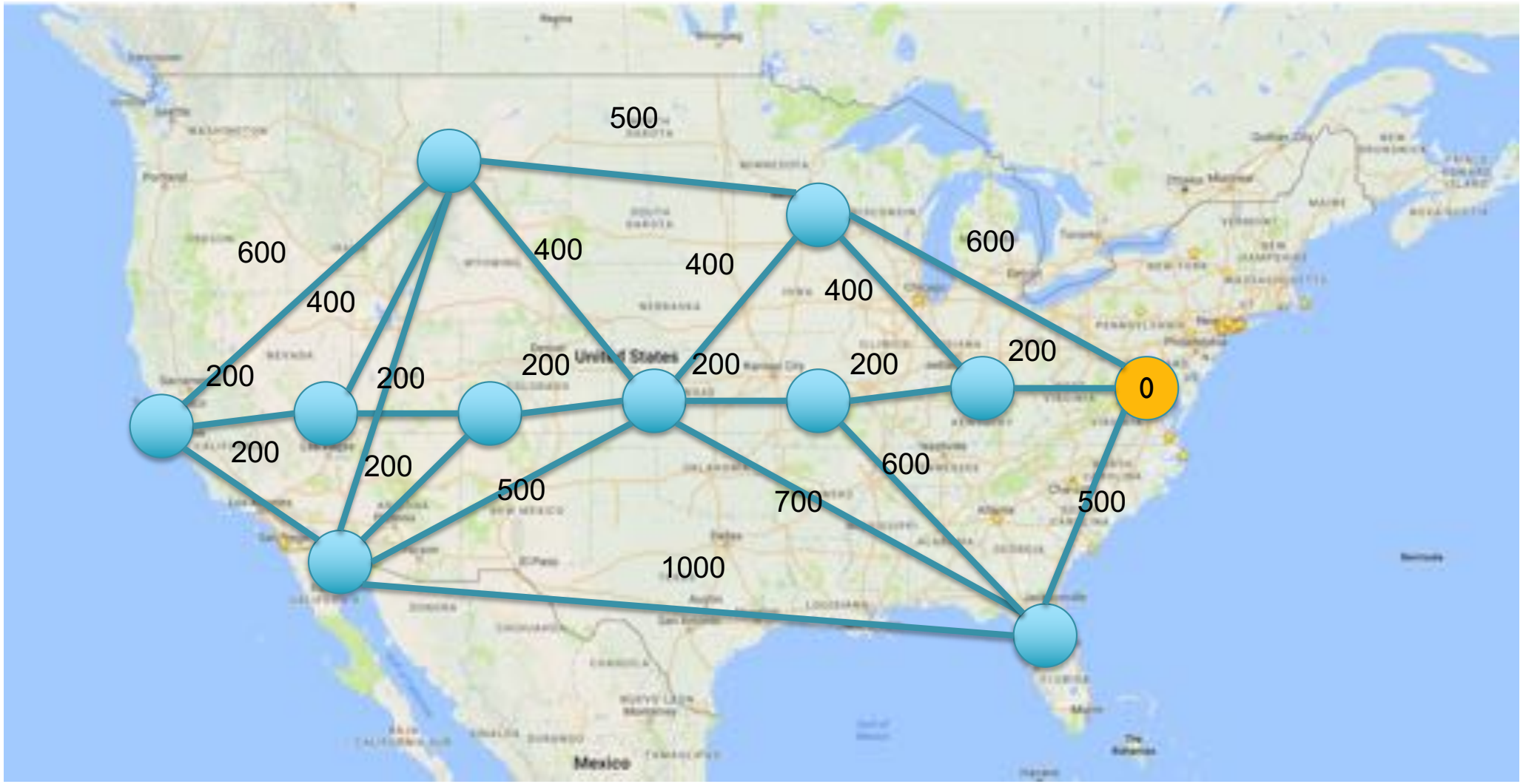
Breath First Searching



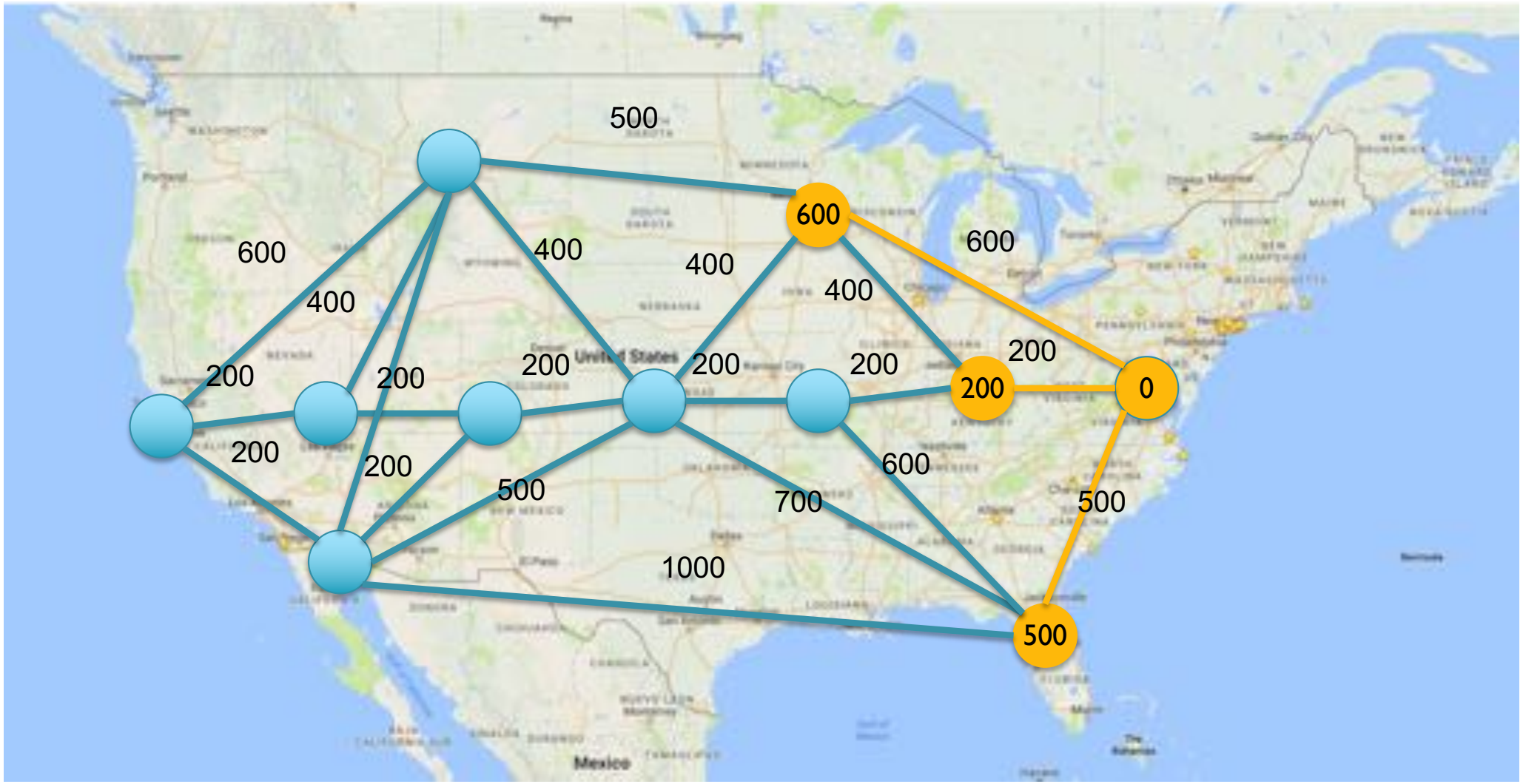
Why doesn't BFS work for maps?



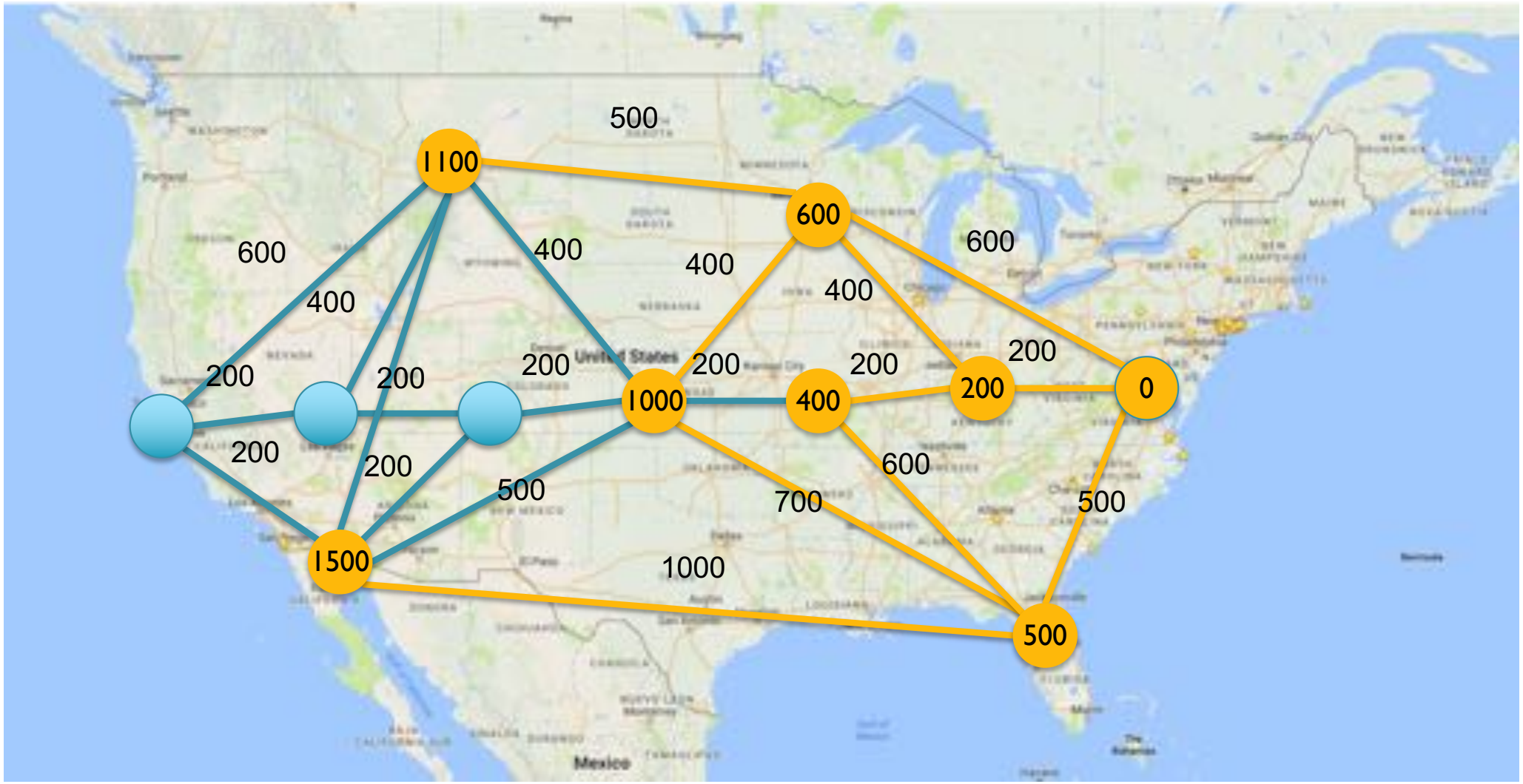
Why doesn't BFS work for maps?



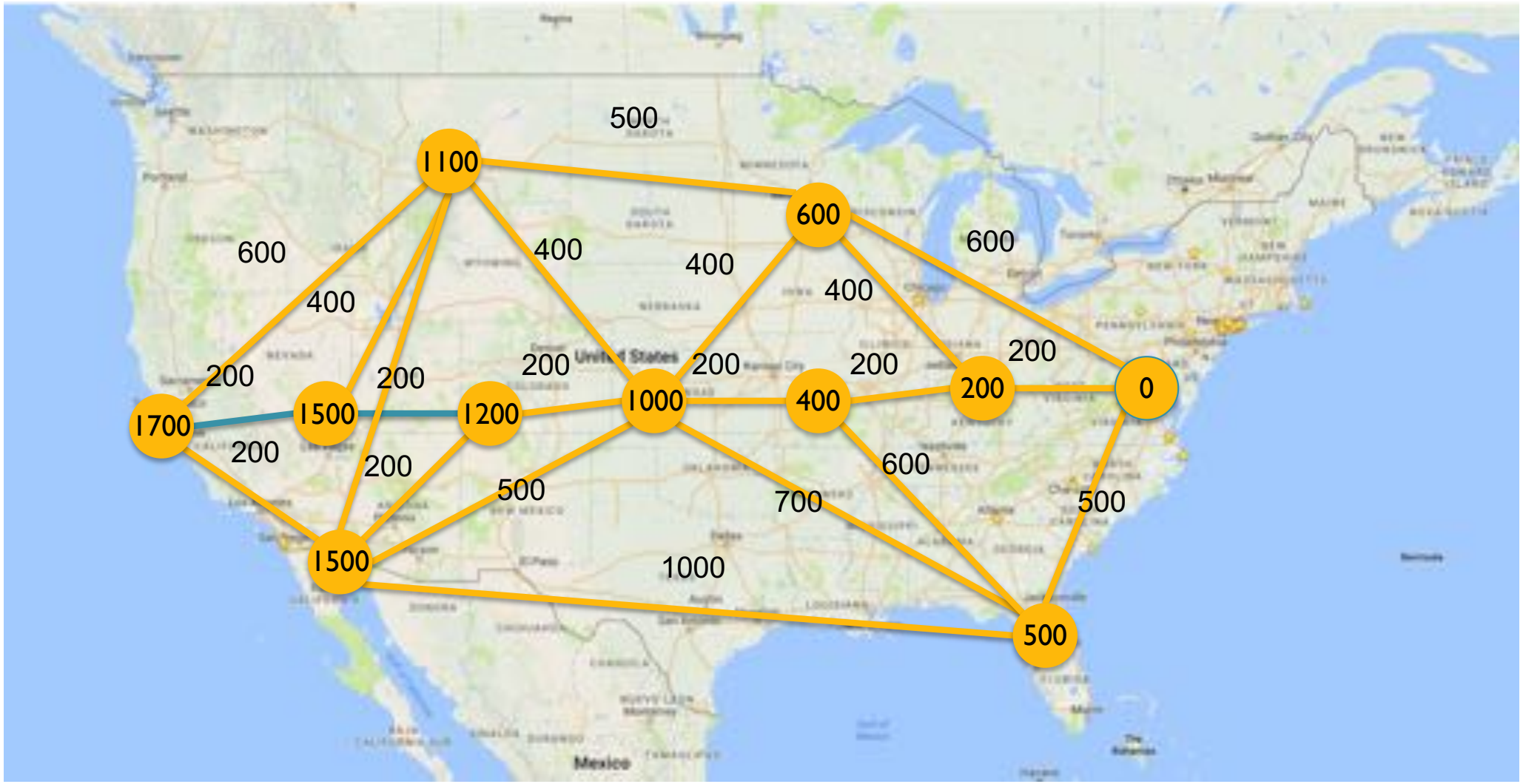
Why doesn't BFS work for maps?



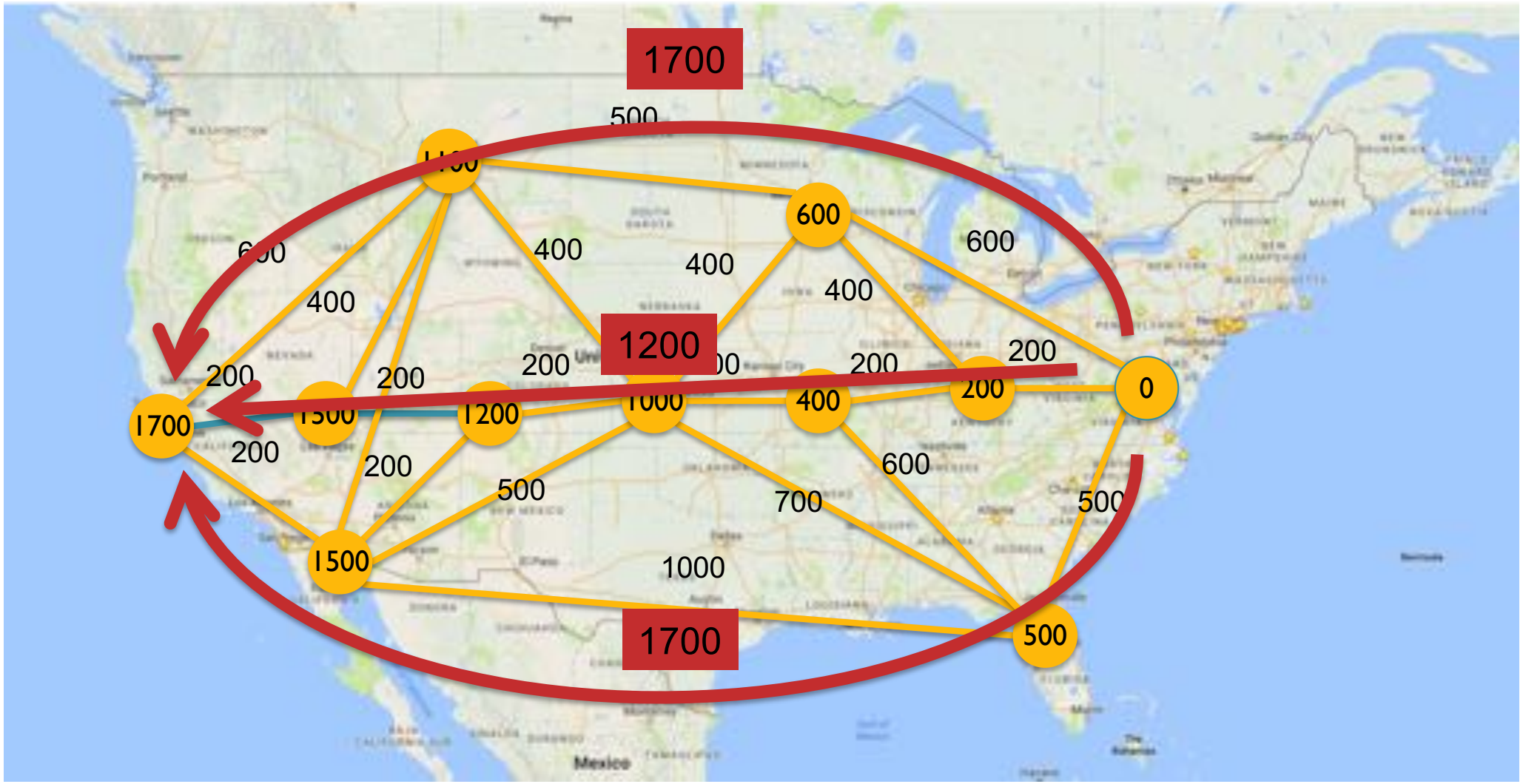
Why doesn't BFS work for maps?



Why doesn't BFS work for maps?

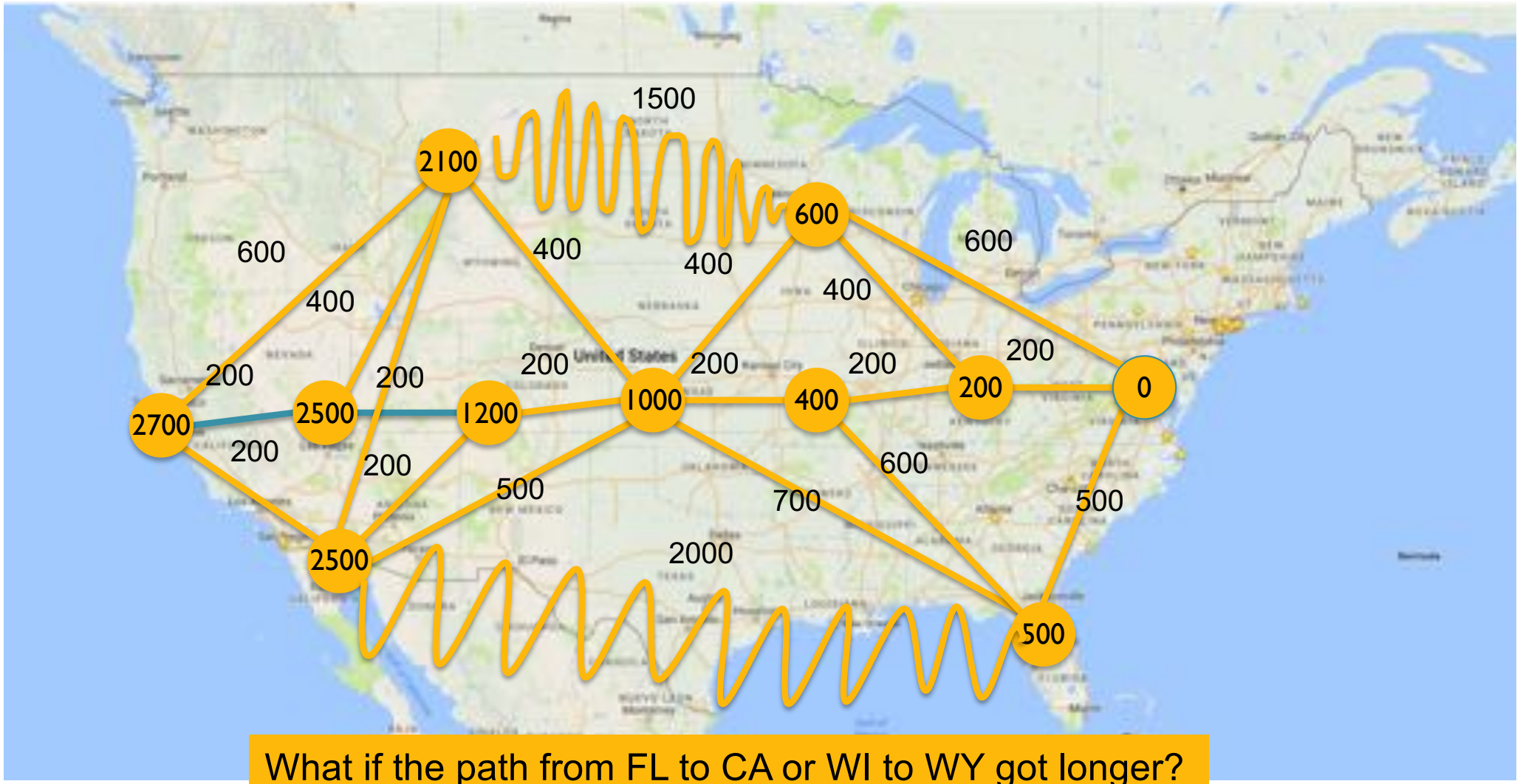


Why doesn't BFS work for maps?



Fewest hops != Shortest distance

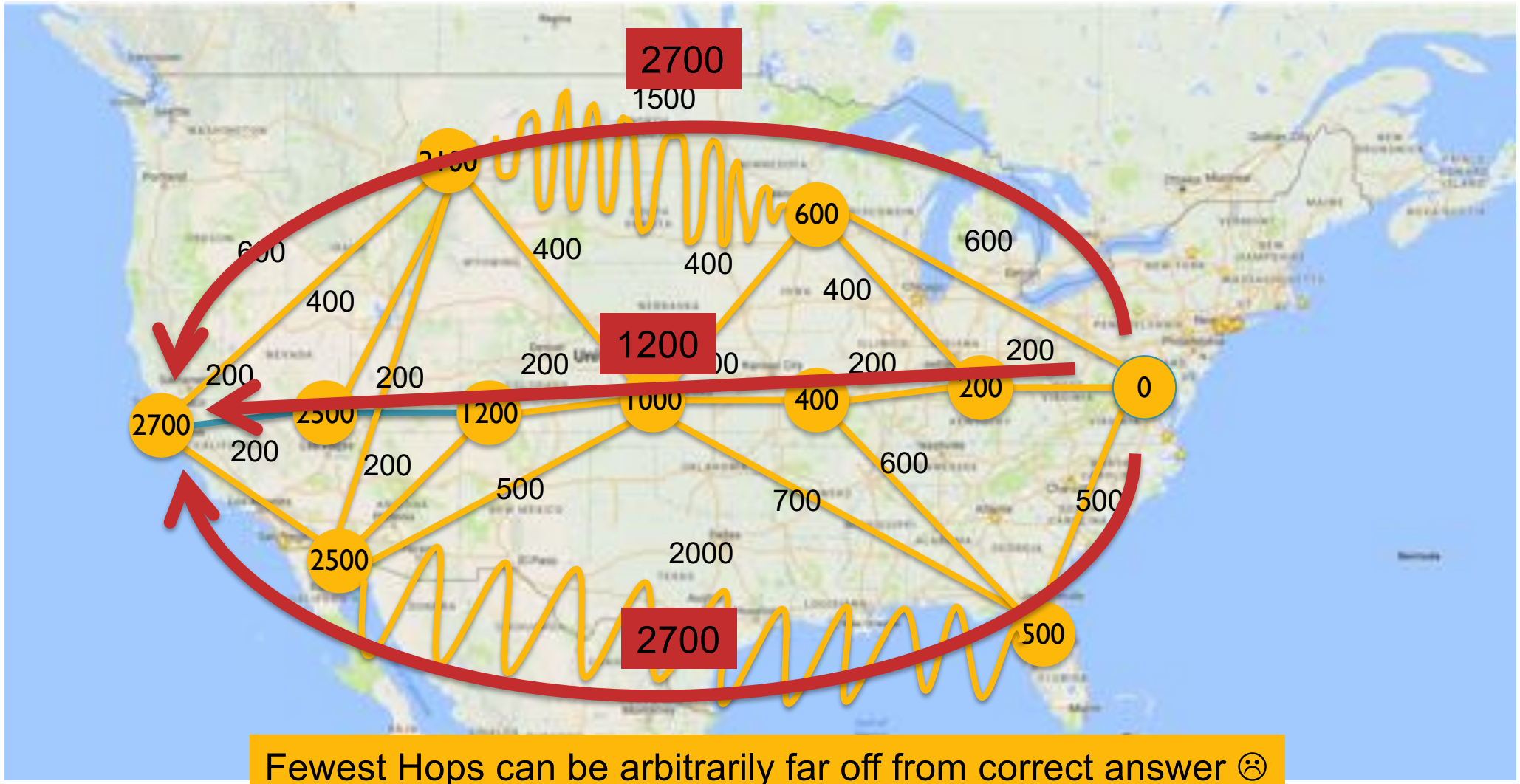
Why doesn't BFS work for maps?



What if the path from FL to CA or WI to WY got longer?

Fewest hops != Shortest distance

Why doesn't BFS work for maps?

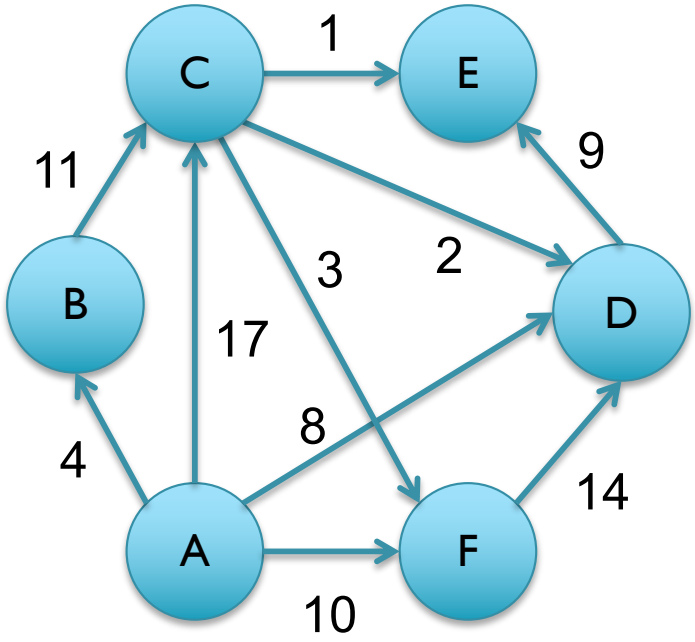


Fewest Hops can be arbitrarily far off from correct answer ☹

Fewest hops != Shortest distance

Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



Note, there may not be a path between those nodes:

- No roads from JHU to U. Hawaii ☹
- No path between nodes that only have outgoing edges

=> Report an infinite distance

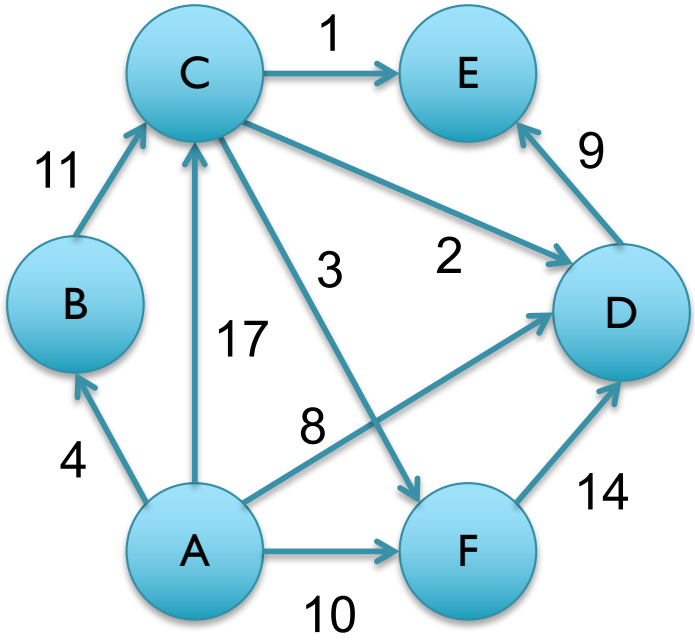
Most commonly applied to graphs with non-negative edge weights

- What might happen with negative weights?

Any ideas?

Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



Similar to BFS, maintain a search frontier of nodes that are further and further away

Similar to BFS, initialize the nodes as unvisited, and record the distance from start in the node

Similar to BFS, leave breadcrumbs along the way so we can retrace the path

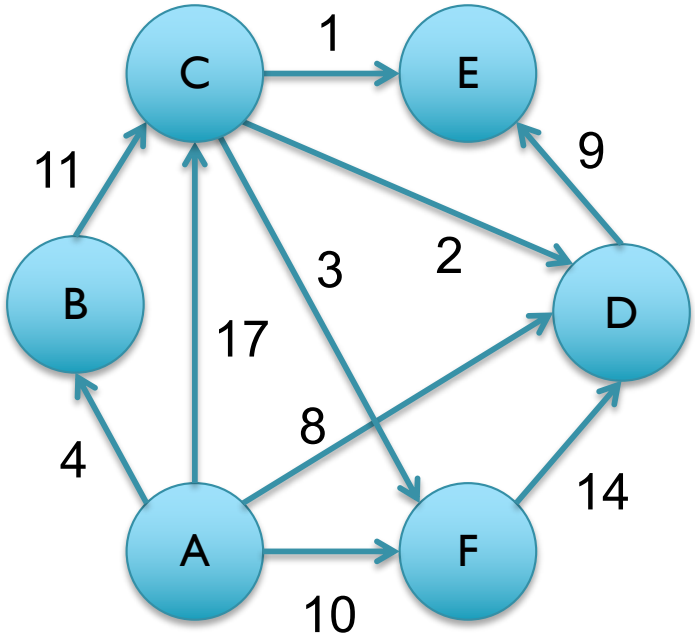
Unlike BFS, we may have to revise the distance if we later find a cheaper path (A->B->C vs A->C)

Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Lets start with node A

Initialize the distance to A as 0, and the distance everywhere else as infinity



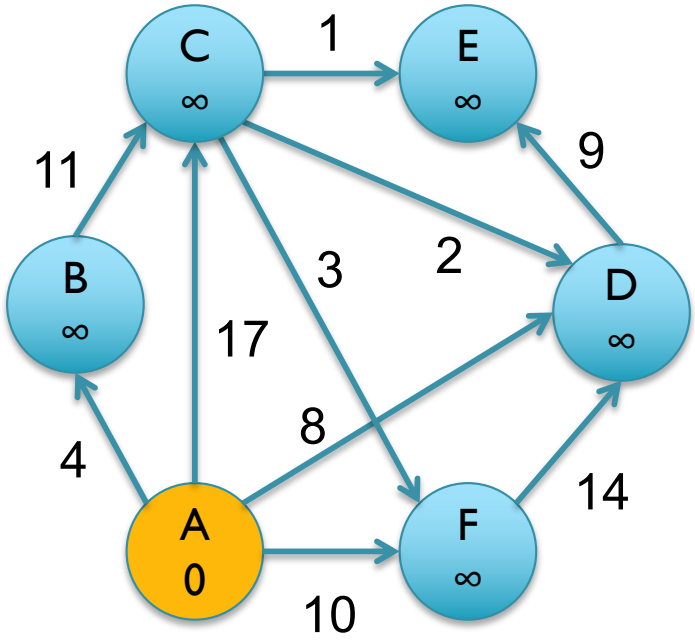
Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Lets start with node A

Initialize the distance to A as 0, and the distance everywhere else as infinity

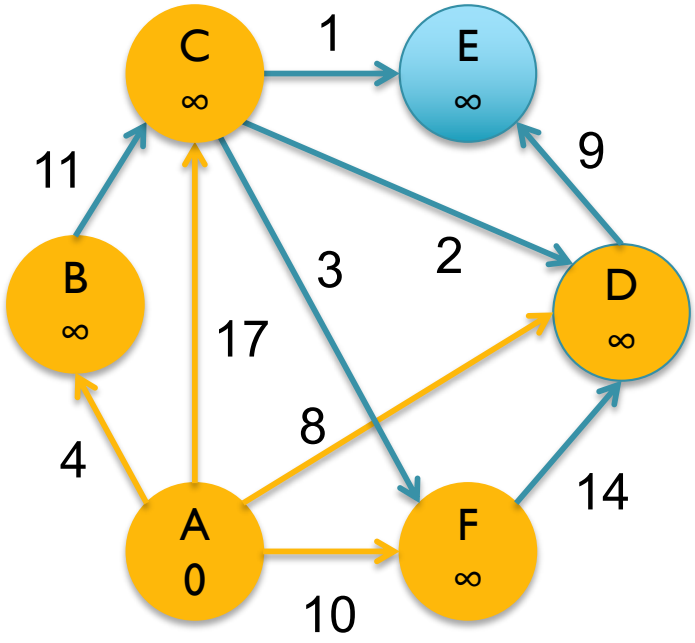
Lets explore all possible paths starting at A (all outgoing edges from A)



Simple Shortest Path Algorithm

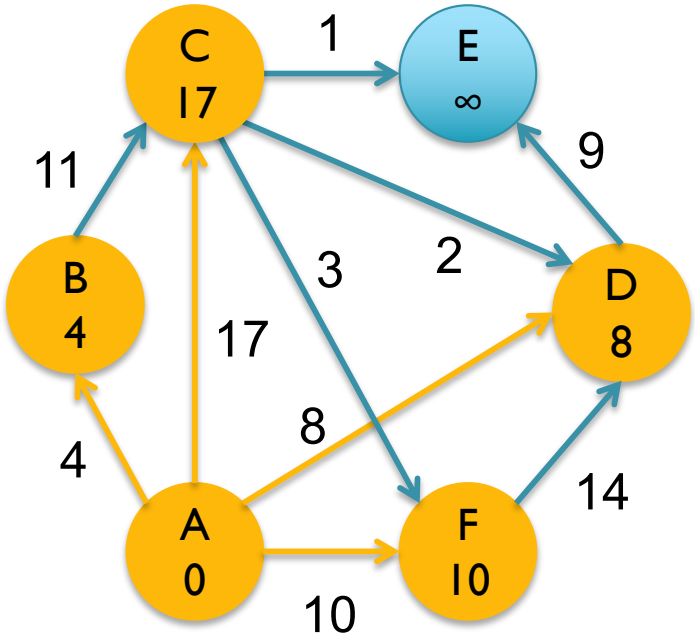
Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

- Lets start with node A
- Initialize the distance to A as 0, and the distance everywhere else as infinity
- Lets explore all possible paths starting at A (all outgoing edges from A)
- Revise if the total distance we just traveled is less than the previously recorded distance (boring this round)



Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



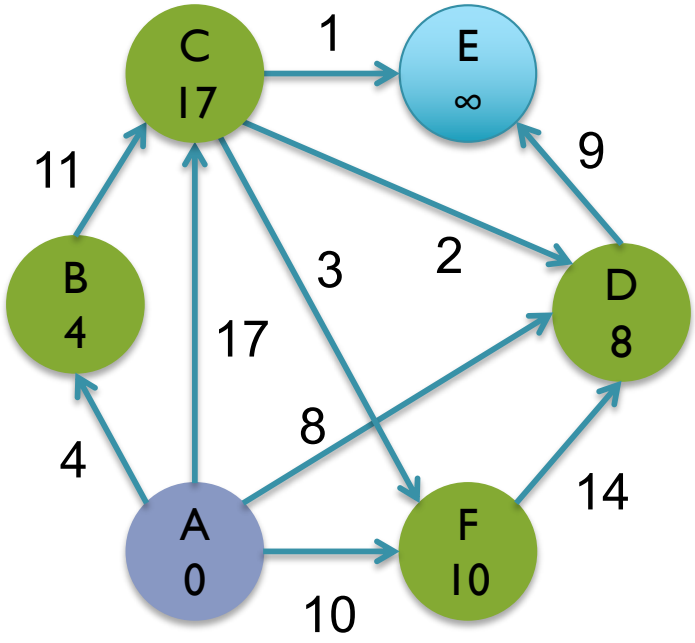
- Lets start with node A
- Initialize the distance to A as 0, and the distance everywhere else as infinity
- Lets explore all possible paths starting at A (all outgoing edges from A)
- Revise if the total distance we just traveled is less than the previously recorded distance (boring this round)
- Repeat! … repeat where?
- Repeat on all nodes that just had their distance updated!

Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Repeat on all nodes that just had their distance updated!

Notice that we only need to explore the edges out from the nodes that were just updated

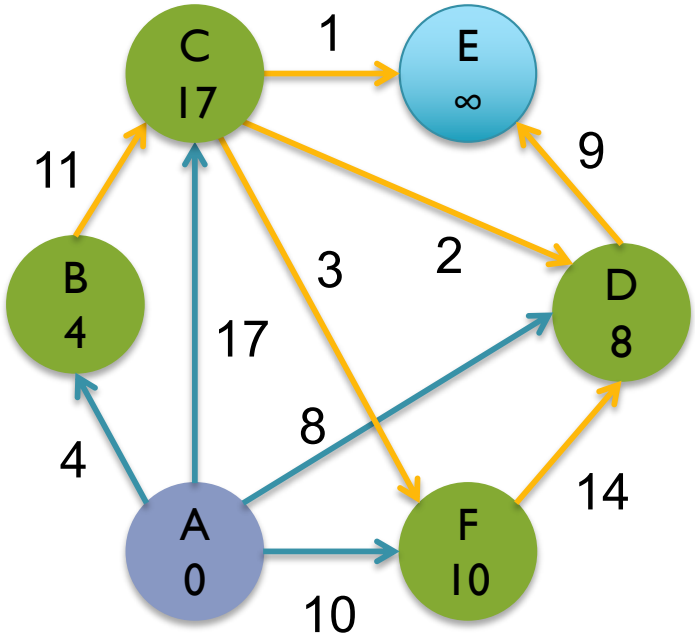


Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Repeat on all nodes that just had their distance updated!

Notice that we only need to explore the edges out from the nodes that were just updated



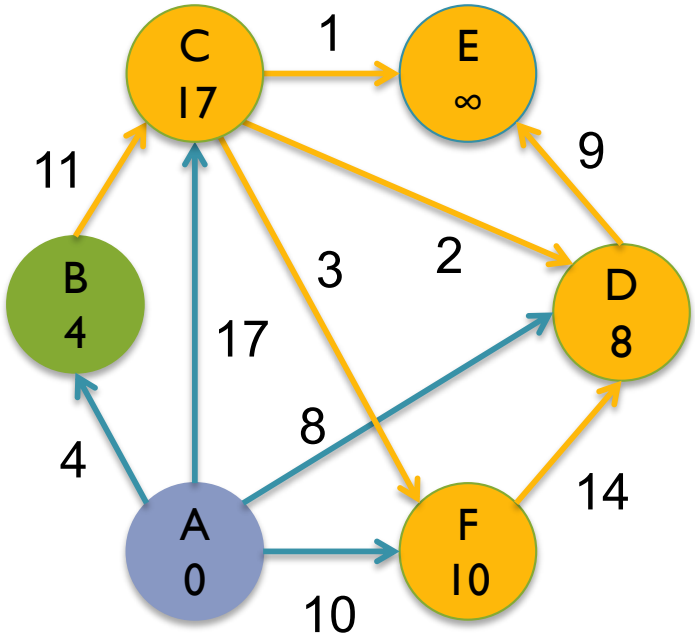
Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Repeat on all nodes that just had their distance updated!

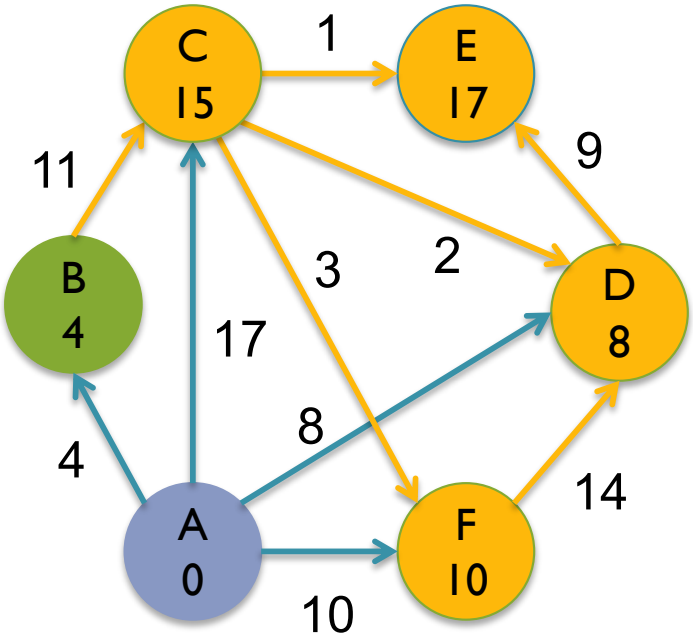
Notice that we only need to explore the edges out from the nodes that were just updated

Now set the distance of the active (yellow) nodes as the minimum of the incoming distances: distance to C is reset from 17 to 15 and E is set to 17 from D



Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



Repeat on all nodes that just had their distance updated!

Notice that we only need to explore the edges out from the nodes that were just updated

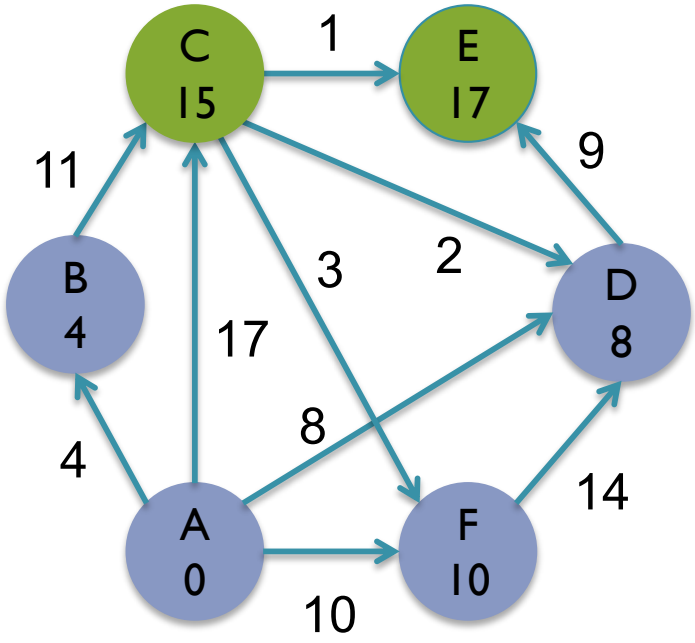
Now set the distance of the active (yellow) nodes as the minimum of the incoming distances: distance to C is reset from 17 to 15 and E is set to 17 from D

Even though the distance to C was previously set, mark that we need to repeat on its outgoing edges

Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Even though the distance to C was previously set, mark that we need to repeat on its outgoing edges

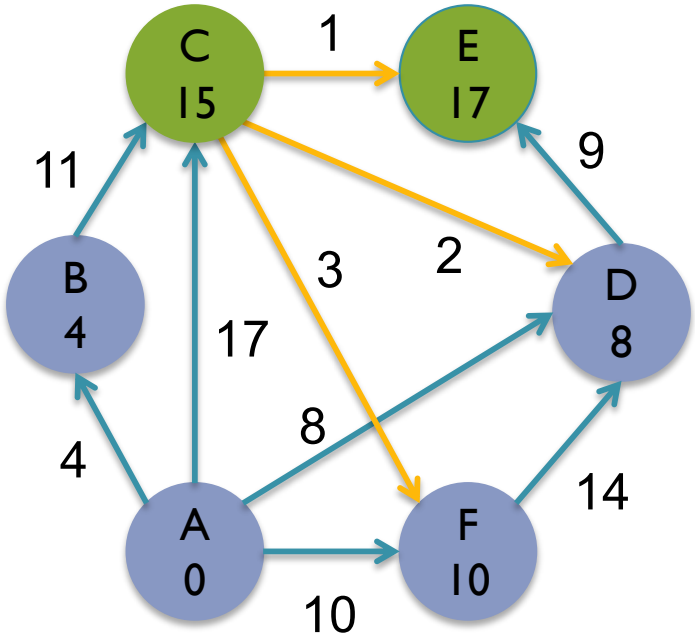


Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

Even though the distance to C was previously set, mark that we need to repeat on its outgoing edges

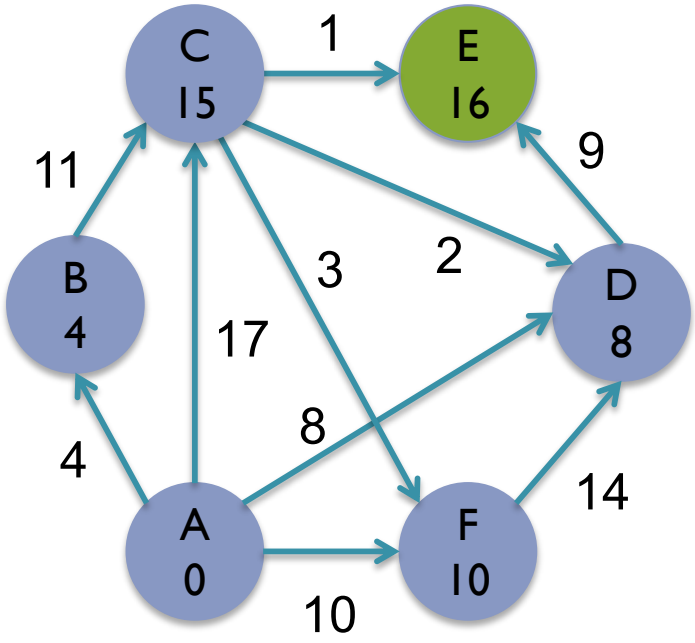
Notice now the path to E is about to get cheaper! (16 versus 17)



Simple Shortest Path Algorithm

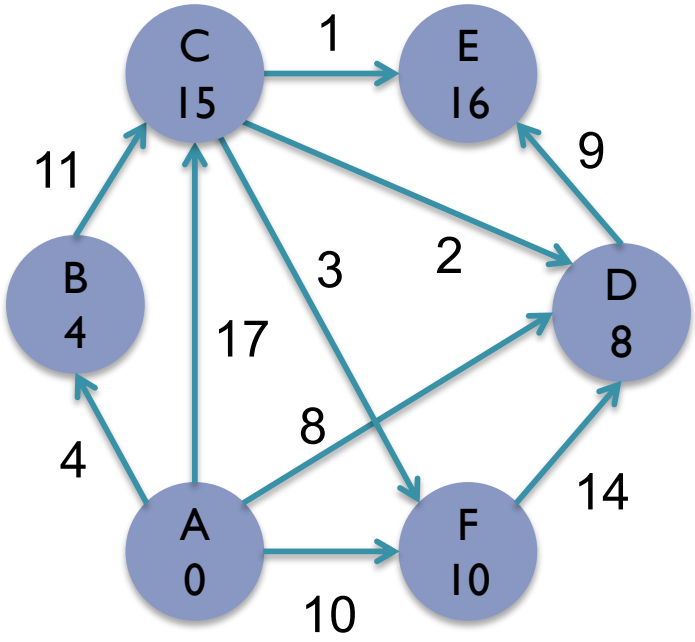
Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

E is the only node that was updated in the last round, but it doesn't have any outgoing edges



Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



E is the only node that was updated in the last round, but it doesn't have any outgoing edges

All done!

Say we were only interested in the path between a specific pair of nodes, could we terminate sooner (A->F)?

Nope ☹

Simple Shortest Path Algorithm

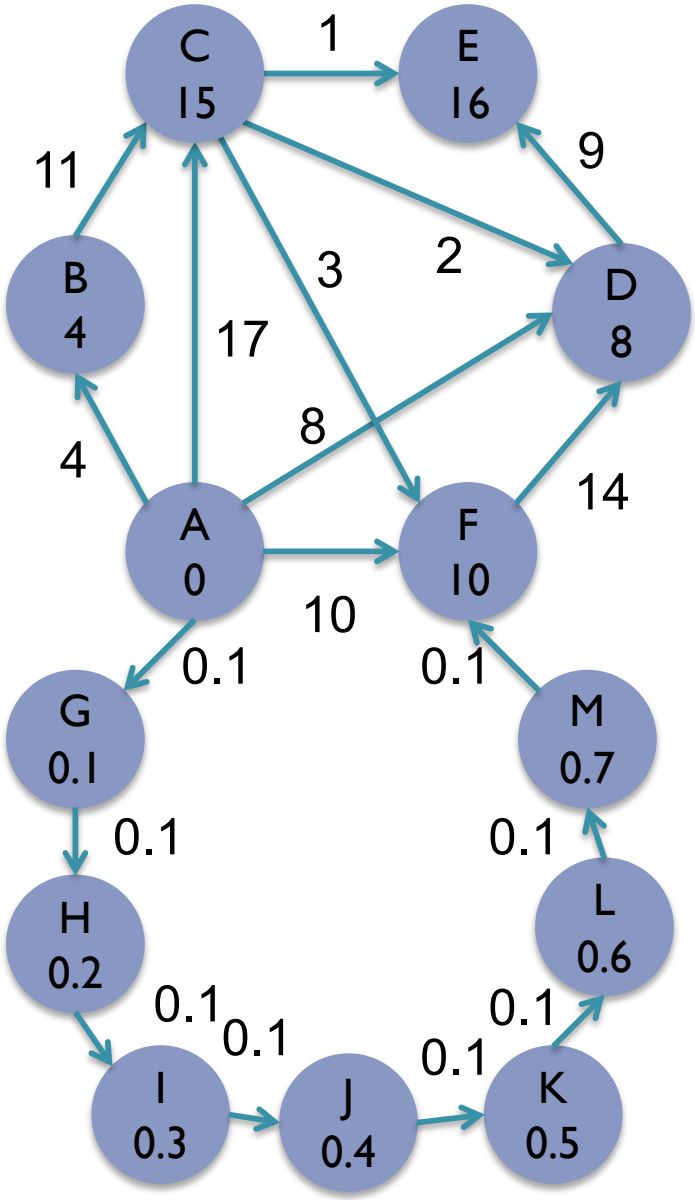
Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

E is the only node that was updated in the last round, but it doesn't have any outgoing edges

All done!

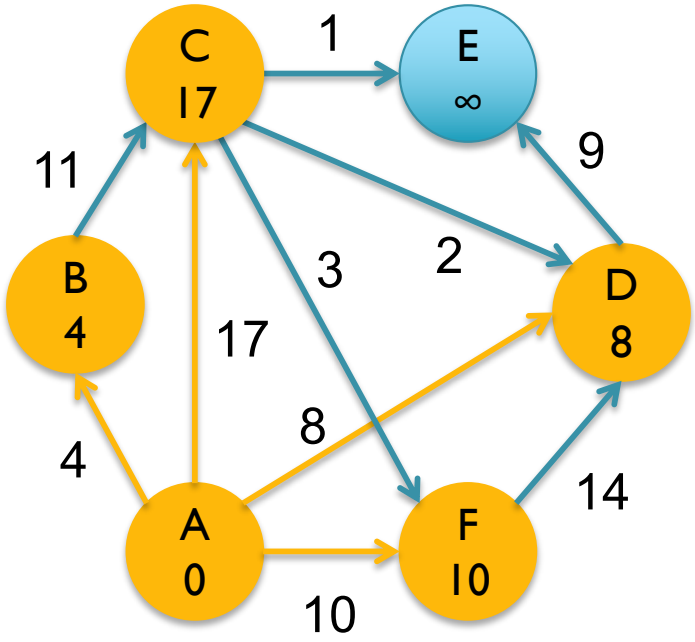
Say we were only interested in the path between a specific pair of nodes, could we terminate sooner (A->F)?

Nope ☹



Simple Shortest Path Algorithm

Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.



- Lets start with node A
- Initialize the distance to A as 0, and the distance everywhere else as infinity
- Lets explore all possible paths starting at A (all outgoing edges from A)
- Revise if the total distance we just traveled is less than the previously recorded distance (boring this round)
- Repeat! … repeat where?
- Repeat on all nodes that just had their distance updated!

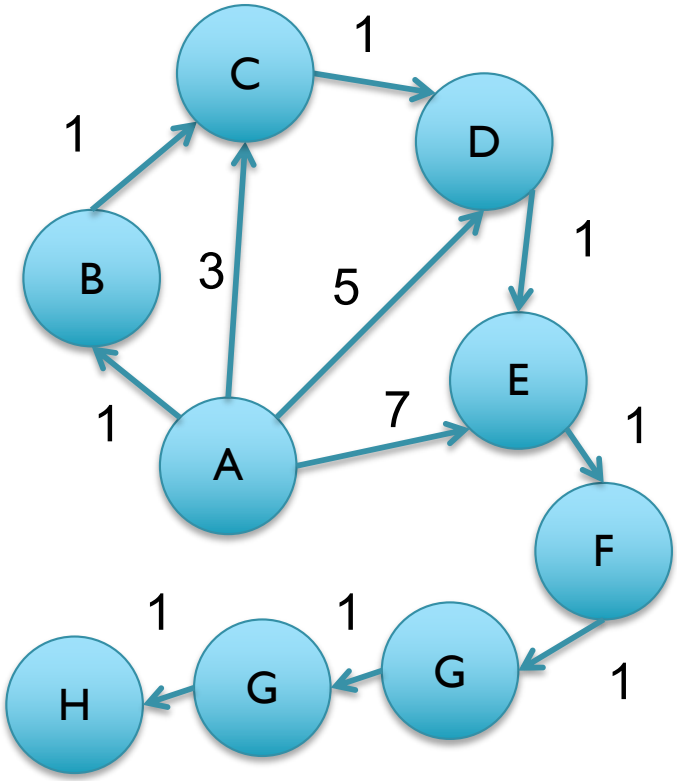
Whats wrong with this algorithm?

Correct but slow, same edges may be explored many times O(\|V\|*\|E\|)

Simple Shortest Path Algorithm

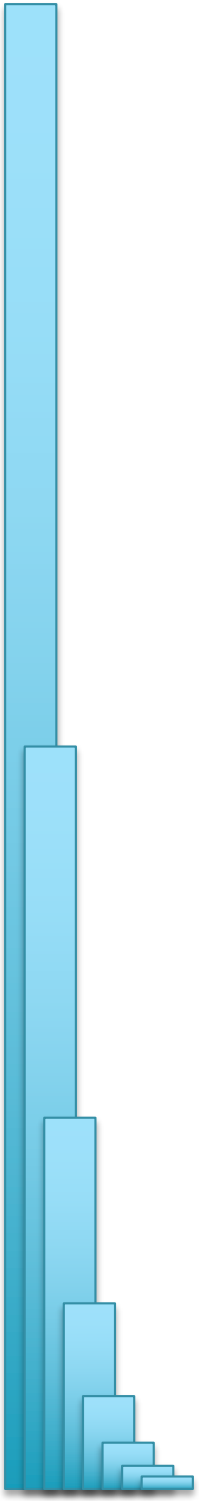
Given a weighted directed graph, find the shortest (minimum weight) path from one start node to one final node.

- Lets start with node A
- Initialize the distance to A as 0, and the distance everywhere else as infinity
- Lets explore all possible paths starting at A (all outgoing edges from A)
- Revise if the total distance we just traveled is less than the previously recorded distance (boring this round)
- Repeat! … repeat where?
- Repeat on all nodes that just had their distance updated!



Whats wrong with this algorithm?

Correct but slow, same edges may be explored many times O(\|V\| * \|E\|)



Next Steps

1. Reflect on the magic and power of Sorting!

1. Assignment 9 due on Friday November 30 @ 10pm