COMP251: DATA STRUCTURES & ALGORITHMS

Instructor: Maryam Siahbani

Computer Information System
University of Fraser Valley

* Some slides from "Algorithms and Data Structures" by Douglas Wilhelm Harder

Shortest Path

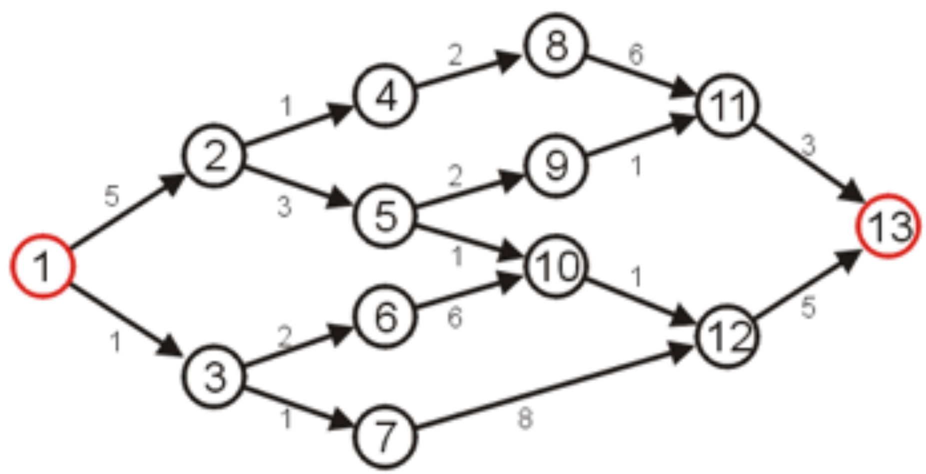
Shortest Path

Given a weighted directed graph, one common problem is finding the shortest path between two given vertices

- Recall that in a weighted graph, the ***length*** of a path is the sum of the weights of each of the edges in that path

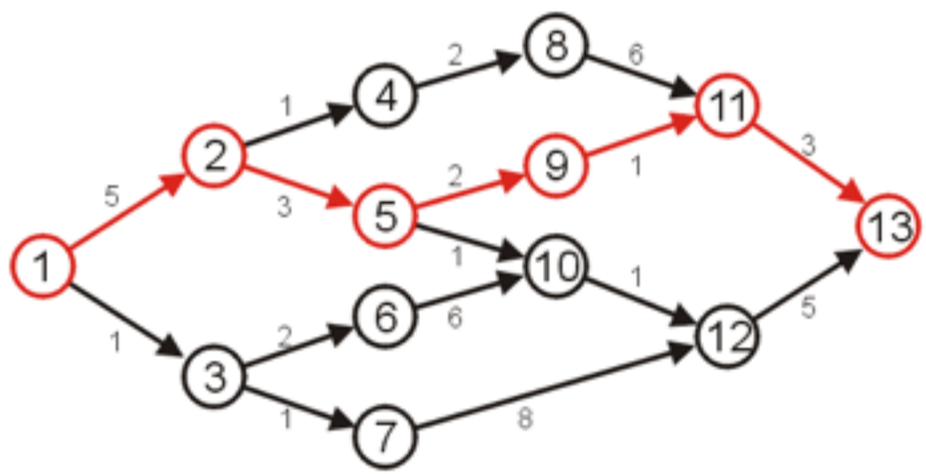
Shortest Path

Given this graph, suppose we wish to find the shortest path from vertex 1 to vertex 13



Shortest Path

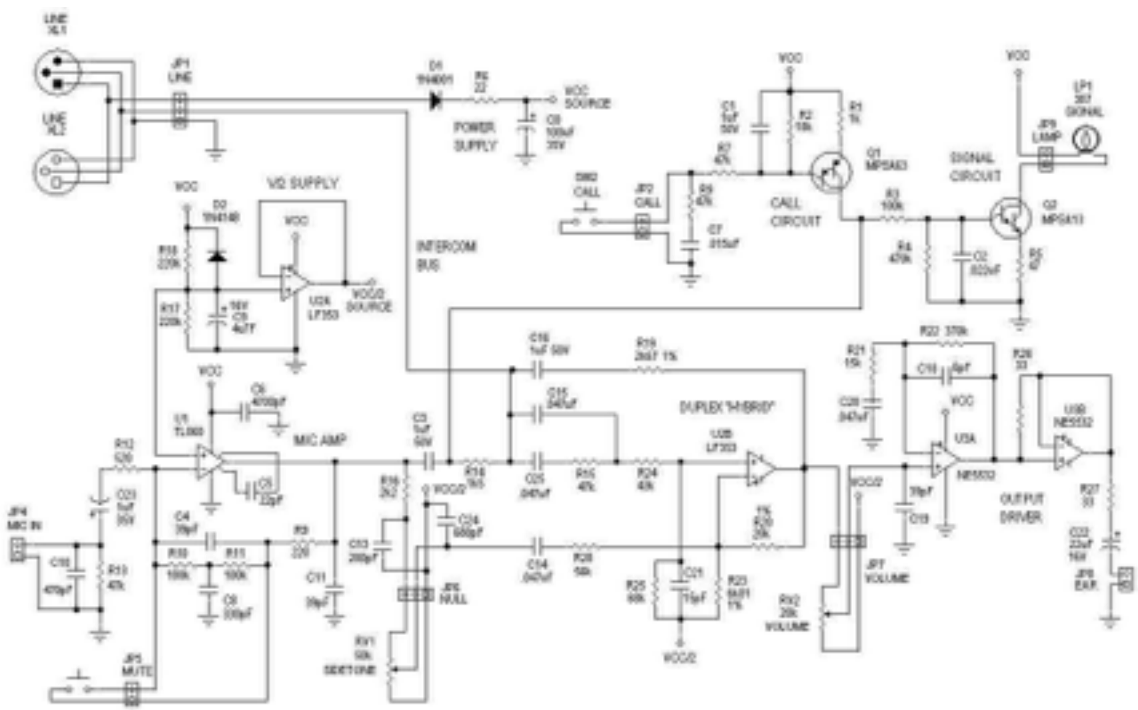
After some consideration, we may determine that the shortest path is as follows, with length 14



Other paths exists, but they are longer

Applications

One application is circuit design: the time it takes for a change in input to affect an output depends on the shortest path



http://www.hp.com/

Applications

The Internet is a collection of interconnected computer networks

- Information is passed through *packets*

Packets are passed from the source, through routers, to their destination

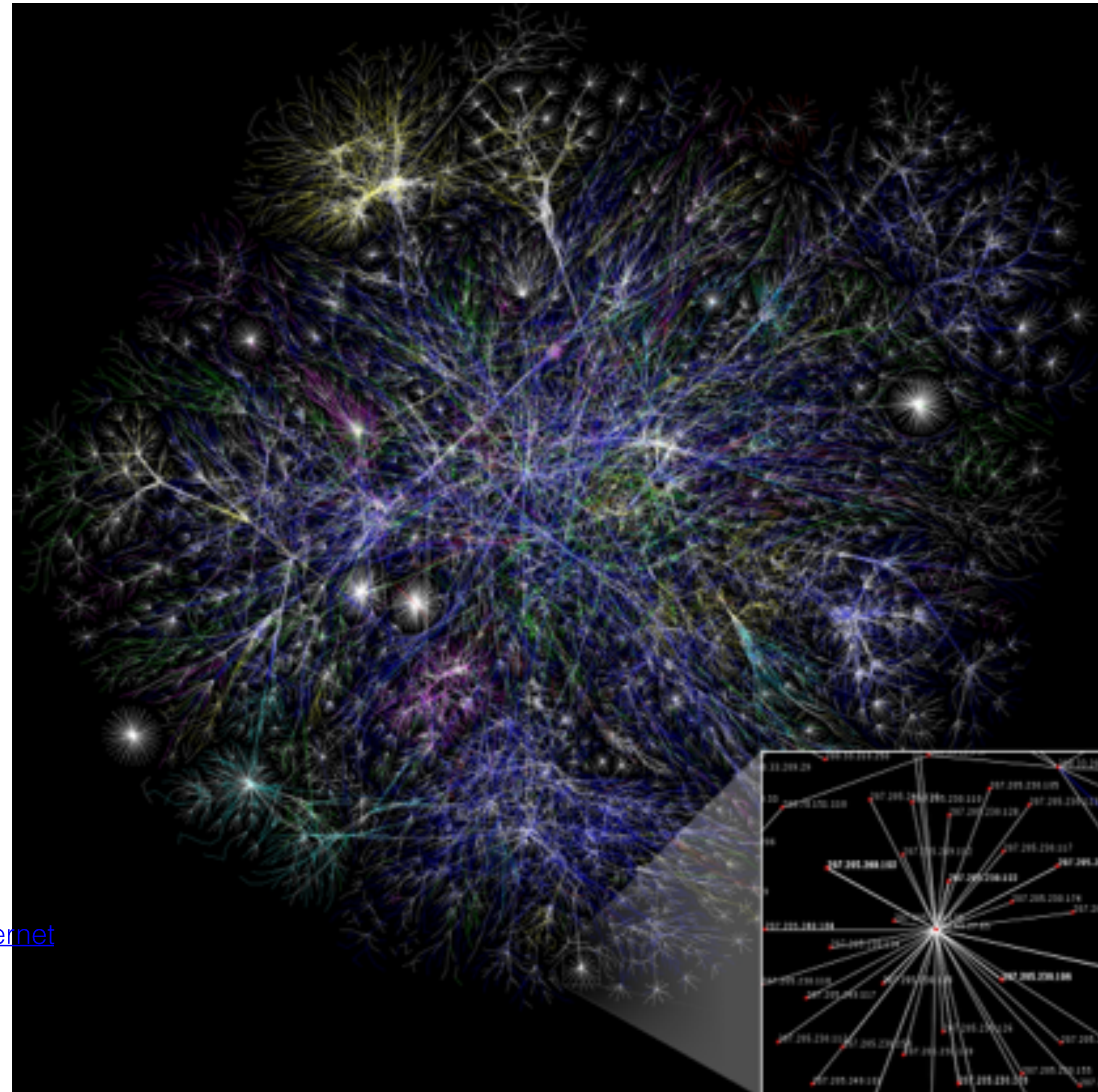
Routers are connected to either:

- individual computers, or
- other routers

These may be represented as graphs

Applications

A visualization of the graph of the routers and their various connections through a portion of the Internet (based on the January 15, 2005 data found on opte.org).



http://en.wikipedia.org/wiki/Internet

Applications

The path a packet takes depends on the IP address

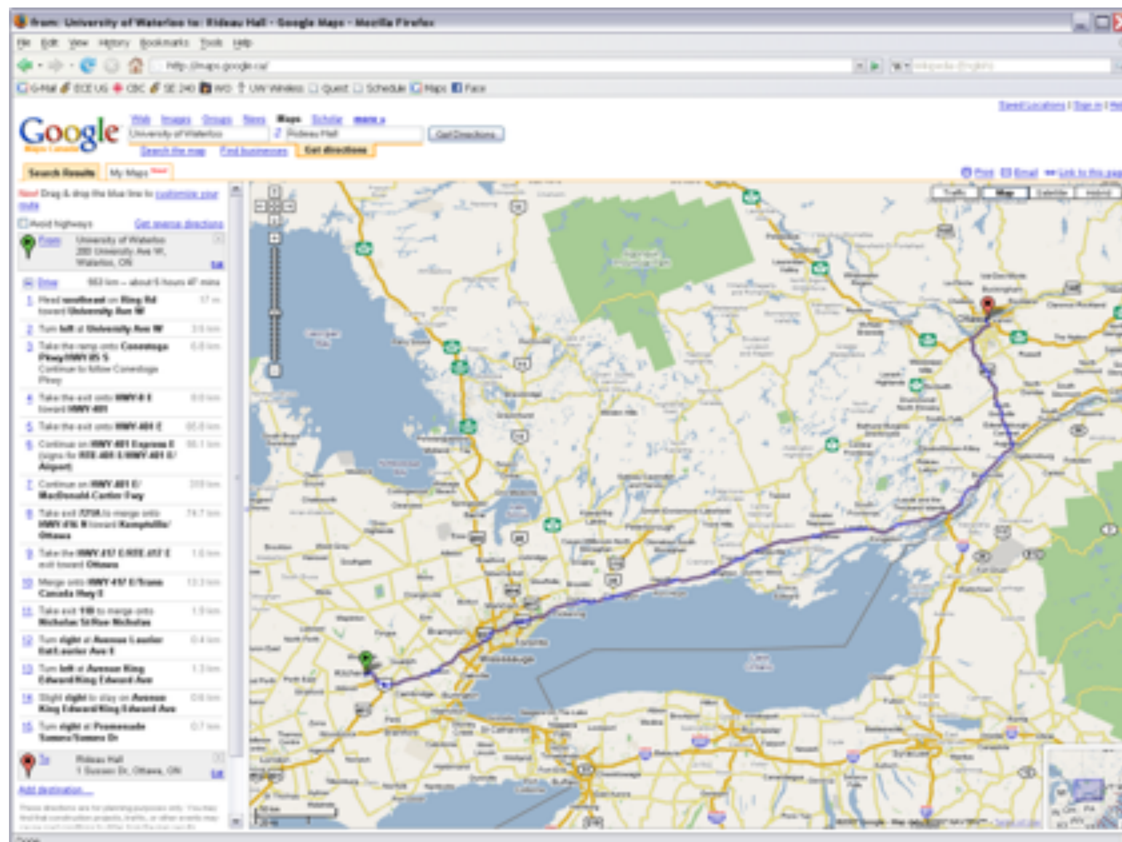
Metrics for measuring the shortest path may include:

- low latency (minimize time), or
- minimum hop count (all edges have weight 1)

Applications

In software engineering, one obvious problem is finding the shortest route between to points on a map

- Shortest path, however, need not refer to distance...



http://maps.google.ca/

Applications

A company will be interested in minimizing the cost which includes the following factors:

- distance
- time
- cost of fuel

Shortest Path

The goal of this algorithm will be to find the shortest path and its length

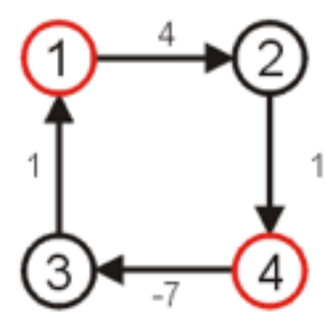
We will make the assumption that the weights on all edges is a positive number

Shortest Path

The goal of this algorithm will be to find the shortest path and its length

We will make the assumption that the weights on all edges is a positive number

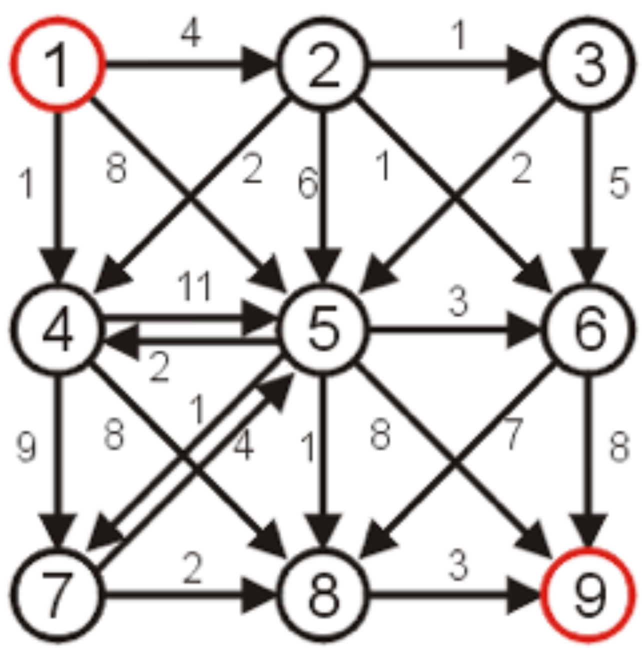
- Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total *length*
- Thus, a shortest length would be undefined for such a graph
- Consider the shortest path from vertex 1 to 4...



Shortest Path

Consider the following graph

- All edges have positive weight
- There exists cycles—it is not a DAG



Algorithms

Algorithms for finding the shortest path include:

- Dijkstra’s algorithm
- A* search algorithm
- Bellman-Ford algorithm

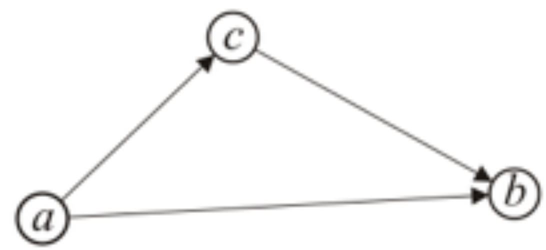
Dijkstra’s algorithm

Dijkstra’s algorithm works on graphs where the weights on all edges is positive

Triangle Inequality

If the distances satisfy the triangle inequality,

- That is, the distance between a and b is less than the distance from a to c plus the distance from c to b,

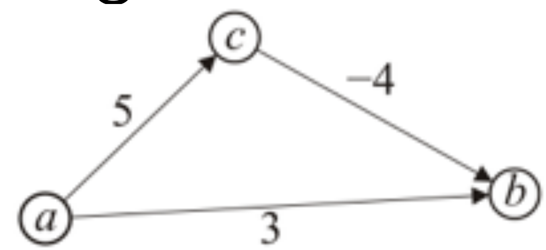


we can use the A* search which is faster than Dijkstra's algorithm

- All Euclidean distances satisfy the triangle inequality

Negative Weights

If some of the edges have negative weight, so long as there are no cycles with negative weight, the Bellman-Ford algorithm will find the minimum distance



–It is slower than Dijkstra's algorithm

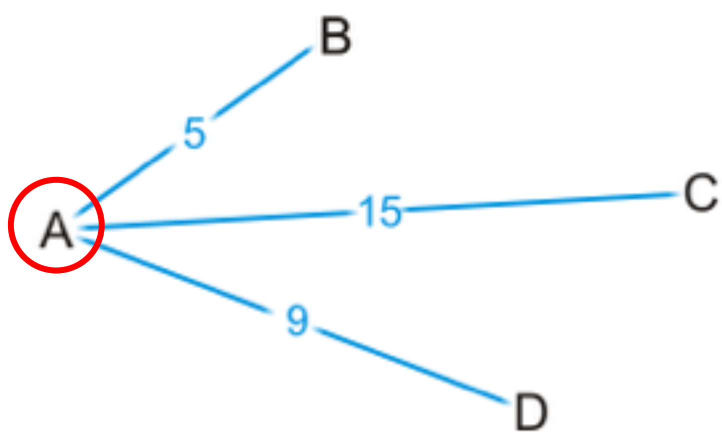
Dijkstra’s algorithm

Dijkstra’s algorithm solves the single-source shortest path problem

Strategy

Suppose you are at vertex A

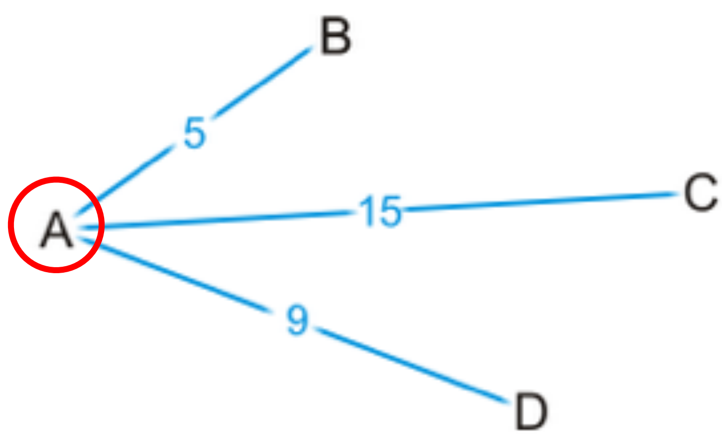
- You are aware of all vertices adjacent to it
- This information is either in an adjacency list or adjacency matrix



Strategy

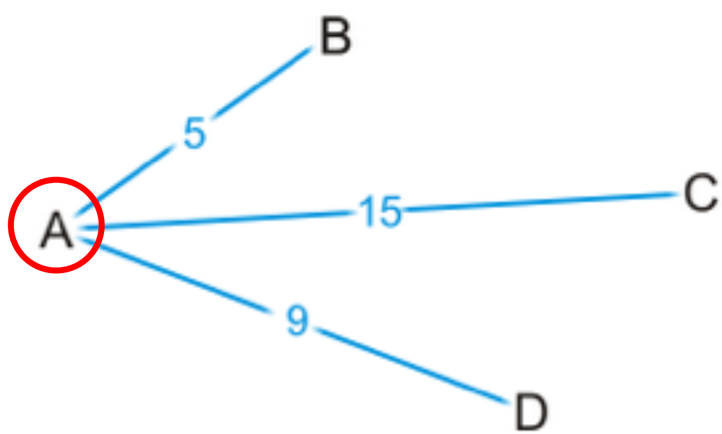
Is 5 the shortest distance to B via the edge (A, B)?

– Why or why not?



Strategy

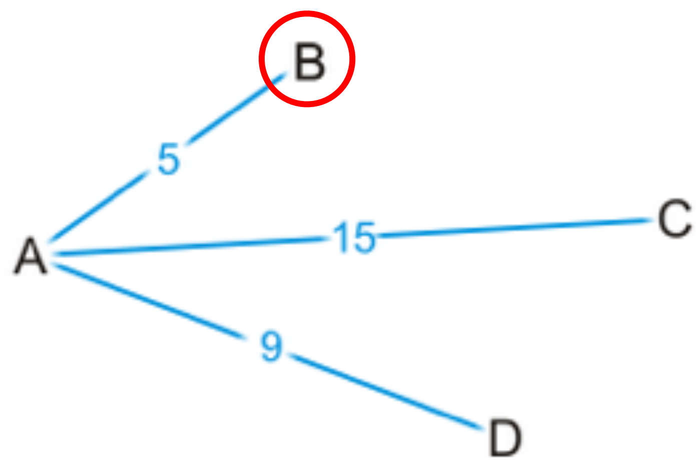
Are you guaranteed that the shortest path to C is (A, C), or
that (A, D) is the shortest path to vertex D?



Strategy

We accept that (A, B) is the shortest path to vertex B from A

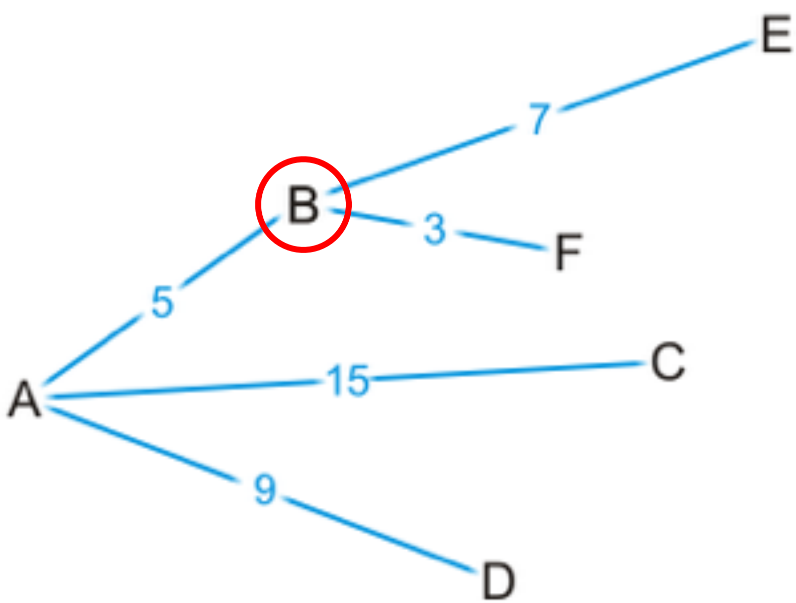
– Let's see where we can go from B



Strategy

By some simple arithmetic, we can determine that

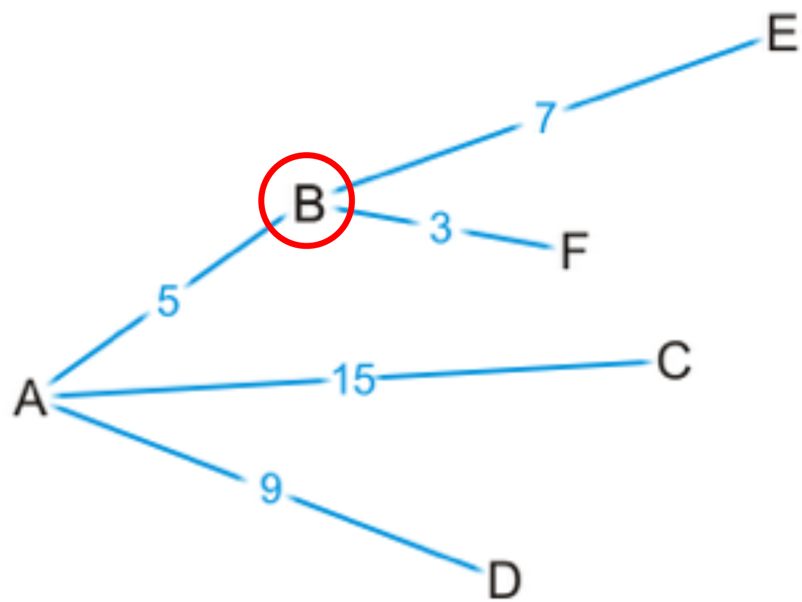
- There is a path (A, B, E) of length 5 + 7 = 12
- There is a path (A, B, F) of length 5 + 3 = 8



Strategy

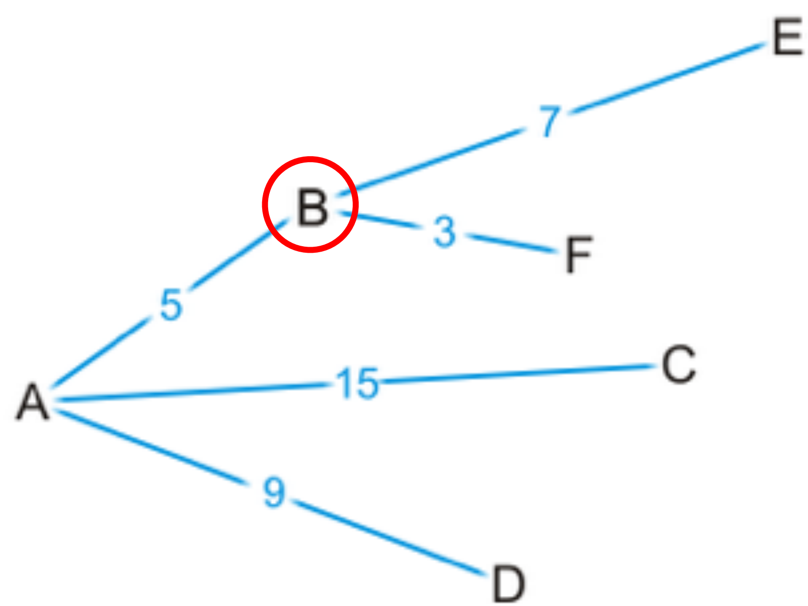
Is (A, B, F) is the shortest path from vertex A to F?

– Why or why not?



Strategy

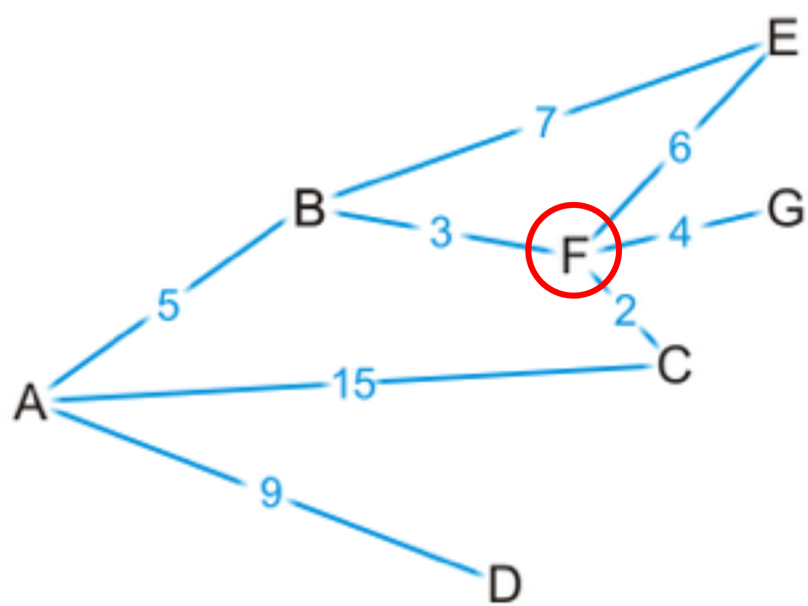
Are we guaranteed that any other path we are currently aware of is also going to be the shortest path?



Strategy

Okay, let's visit vertex F

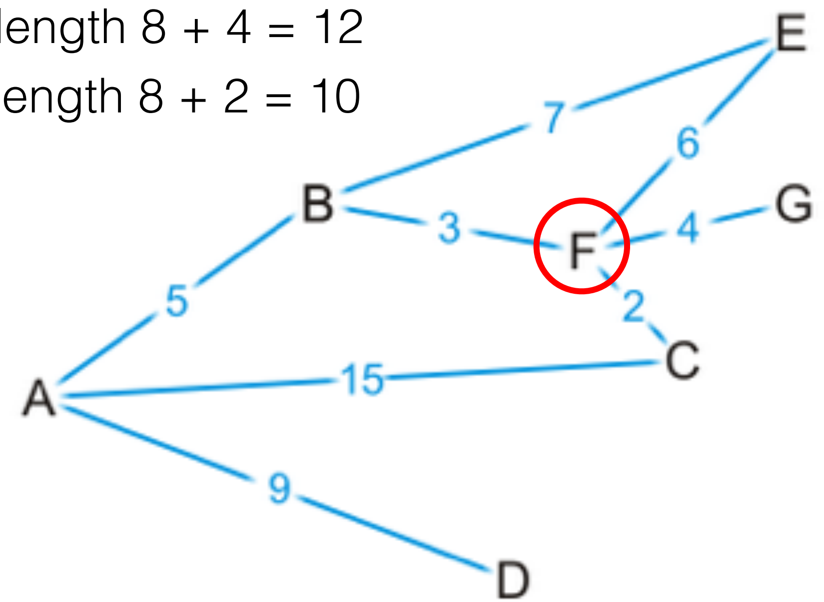
- We know the shortest path is (A, B, F) and it's of length 8



Strategy

There are three edges exiting vertex F, so we have paths:

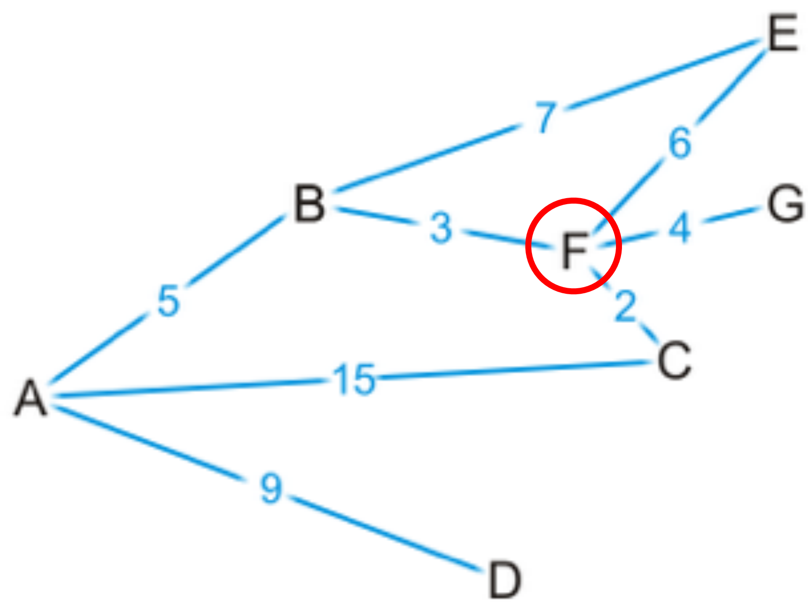
- (A, B, F, E) of length 8 + 6 = 14
- (A, B, F, G) of length 8 + 4 = 12
- (A, B, F, C) of length 8 + 2 = 10



Strategy

By observation:

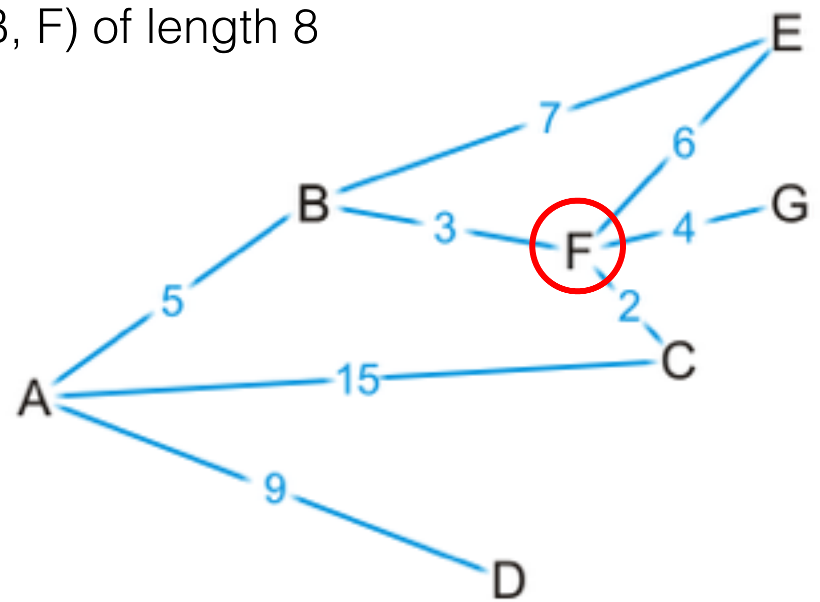
- The path (A, B, F, E) is longer than (A, B, E)
- The path (A, B, F, C) is shorter than the path (A, C)



Strategy

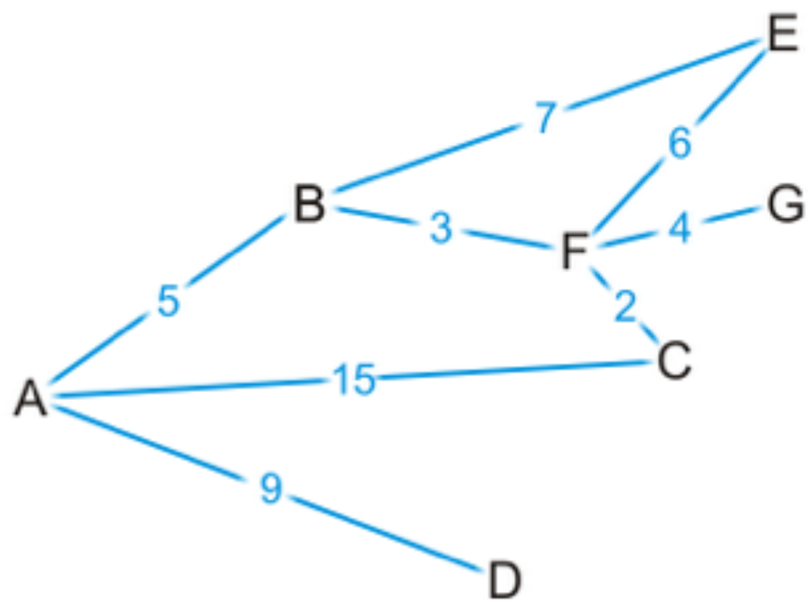
At this point, we've discovered the shortest paths to:

- Vertex B: (A, B) of length 5
- Vertex F: (A, B, F) of length 8



Strategy

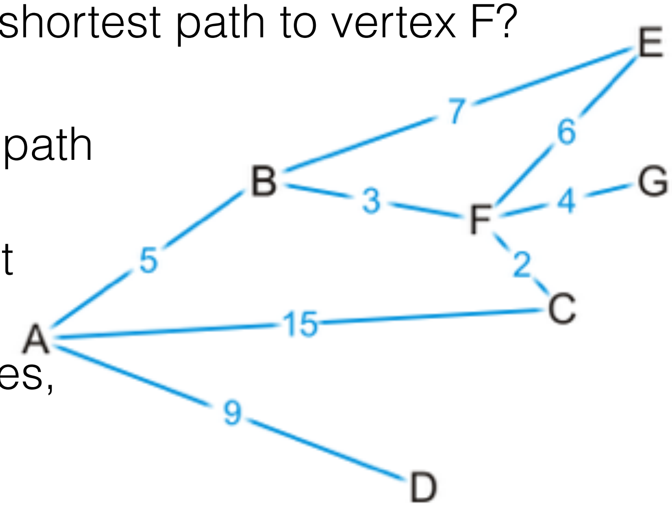
- At this point, we have the shortest distances to B and F
  - Which remaining vertex are we currently guaranteed to have the shortest distance to?



Dijkstra’s algorithm

We initially don’t know the distance to any vertex except the initial vertex

- We require an array of distances, all initialized to infinity except for the source vertex, which is initialized to 0
- Each time we visit a vertex, we will examine all adjacent vertices
  - We need to track visited vertices—a Boolean table of size $|V|$
- Do we need to track the shortest path to each vertex?
  - That is, do I have to store (A, B, F) as the shortest path to vertex F?



- We really only have to record that the shortest path to vertex F came from vertex B
  - We would then determine that the shortest path to vertex B came from vertex A
  - Thus, we need an array of previous vertices, all initialized to null

Dijkstra's algorithm

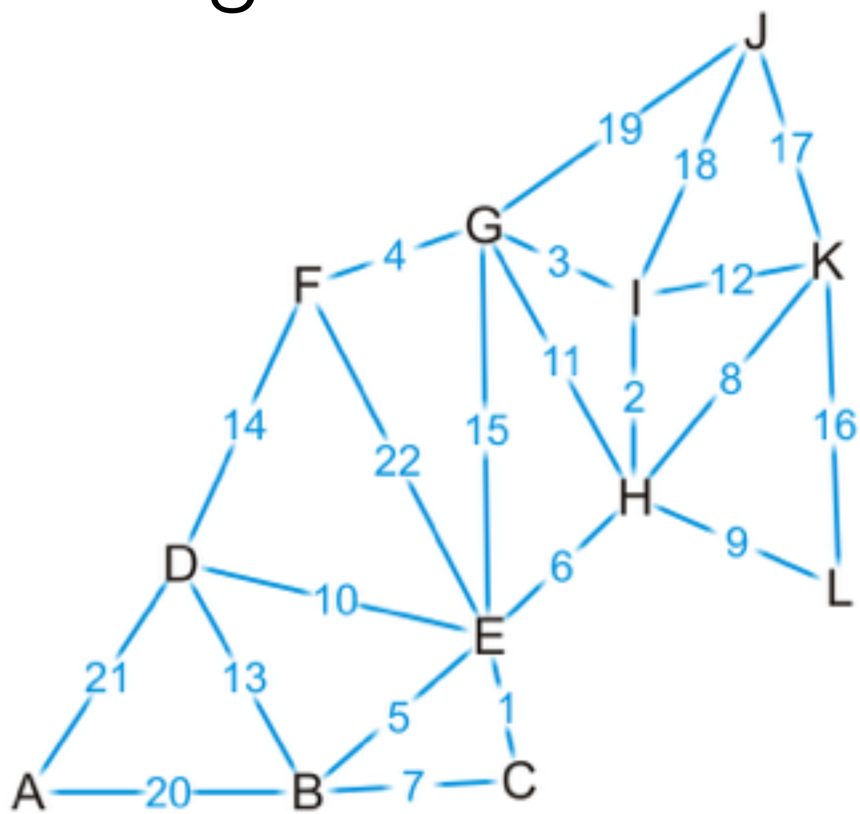
Thus, we will iterate $|V|$ times:

- Find that unvisited vertex v that has a minimum distance to it
- Mark it as having been visited
- Consider every adjacent vertex w that is unvisited:
  - Is the distance to v plus the weight of the edge (v, w) less than our currently known shortest distance to w
  - If so, update the shortest distance to w and record v as the previous pointer

- Continue iterating until all vertices are visited or all remaining vertices have a distance to them of infinity

Example

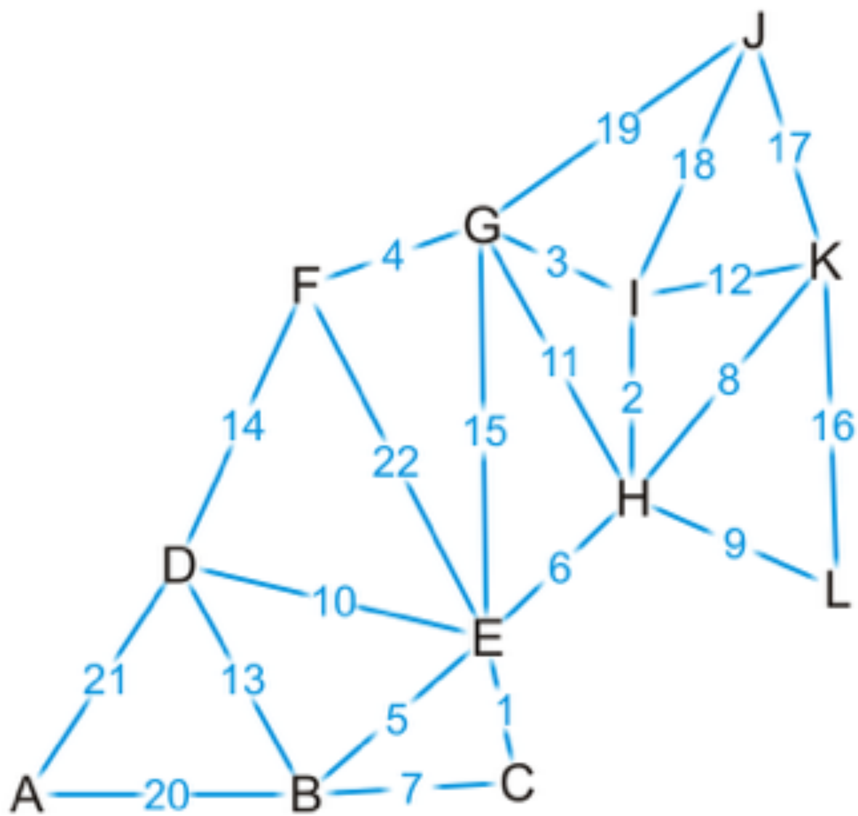
Find the shortest distance from K to every other region



Example

We set up our table

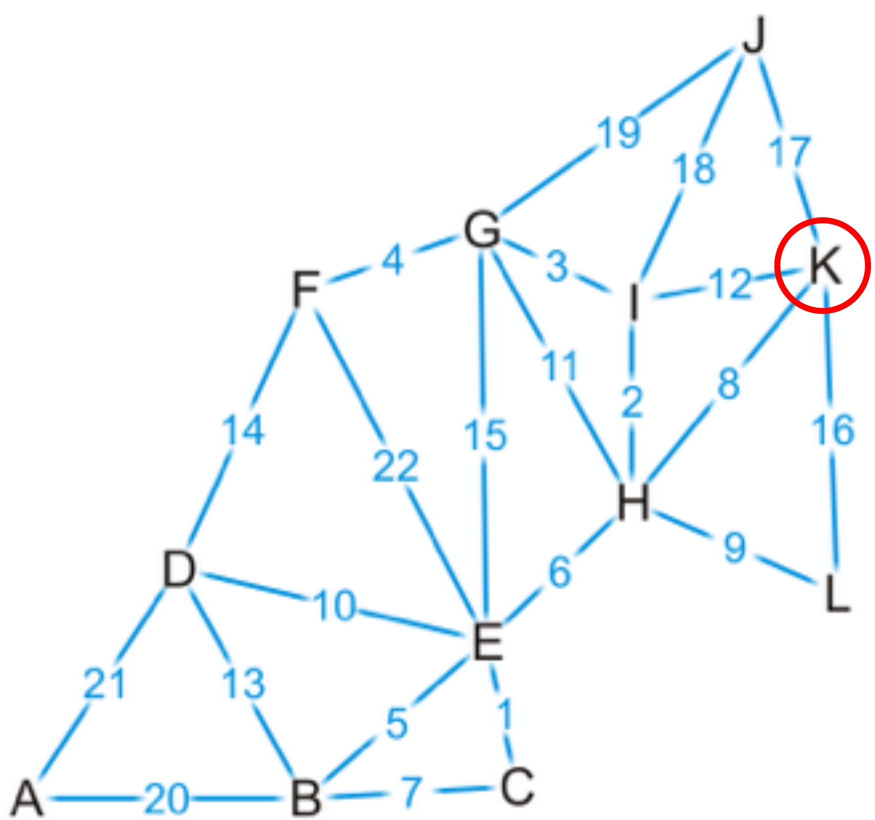
- Which unvisited vertex has the minimum distance to it?



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	∞	Ø
I	F	∞	Ø
J	F	∞	Ø
K	F	0	Ø
L	F	∞	Ø

Example

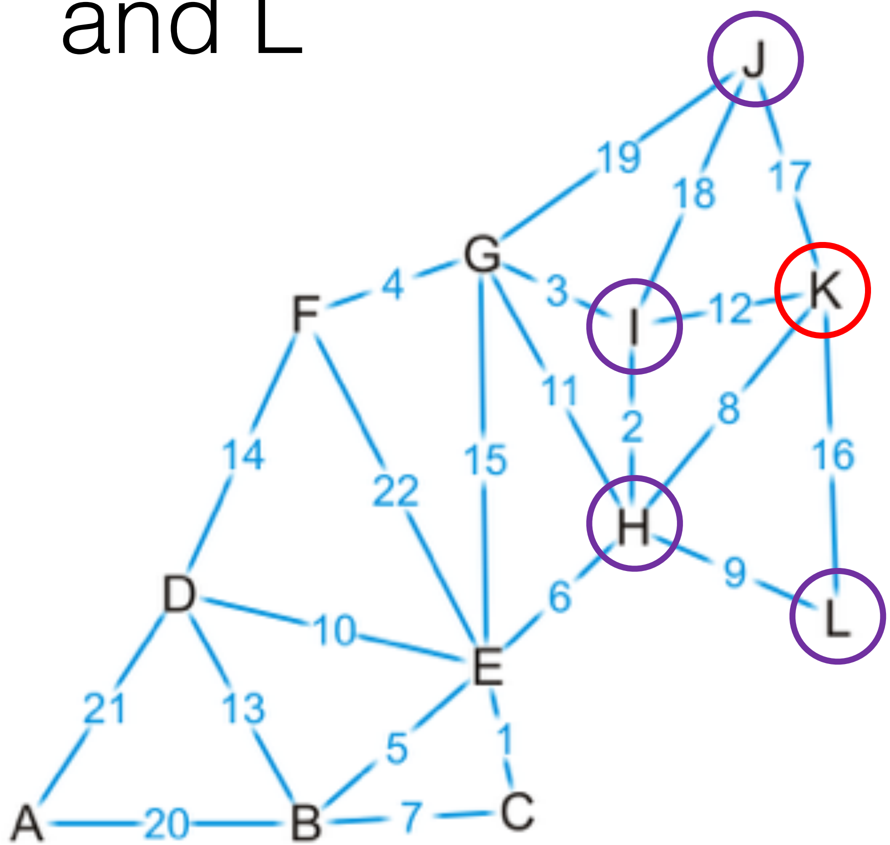
We visit vertex K



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	∞	Ø
I	F	∞	Ø
J	F	∞	Ø
K	T	0	Ø
L	F	∞	Ø

Example

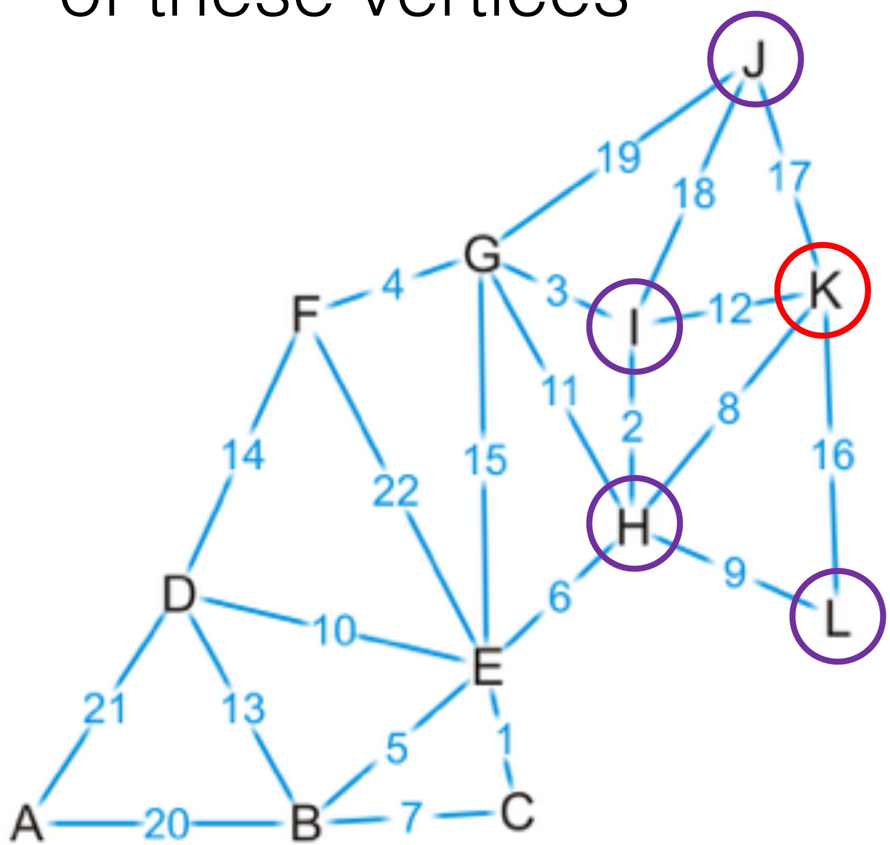
Vertex K has four neighbors: H, I, J and L



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	∞	Ø
I	F	∞	Ø
J	F	∞	Ø
K	**T**	**0**	**Ø**
L	F	∞	Ø

Example

We have now found at least one path to each of these vertices

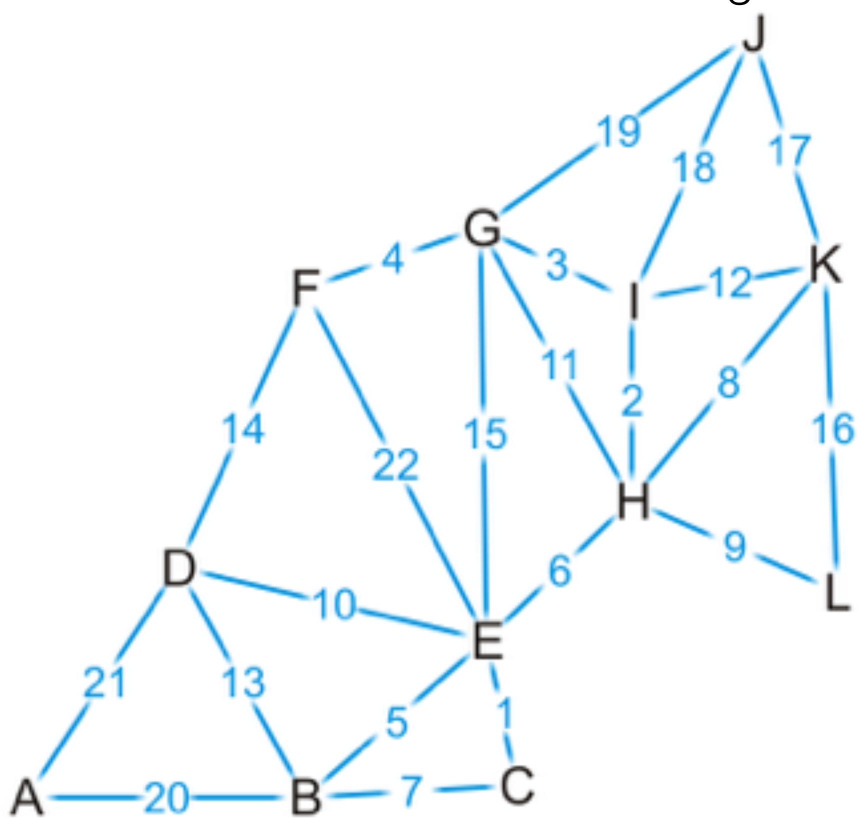


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	8	K
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We're finished with vertex K

– To which vertex are we now guaranteed we have the shortest path?

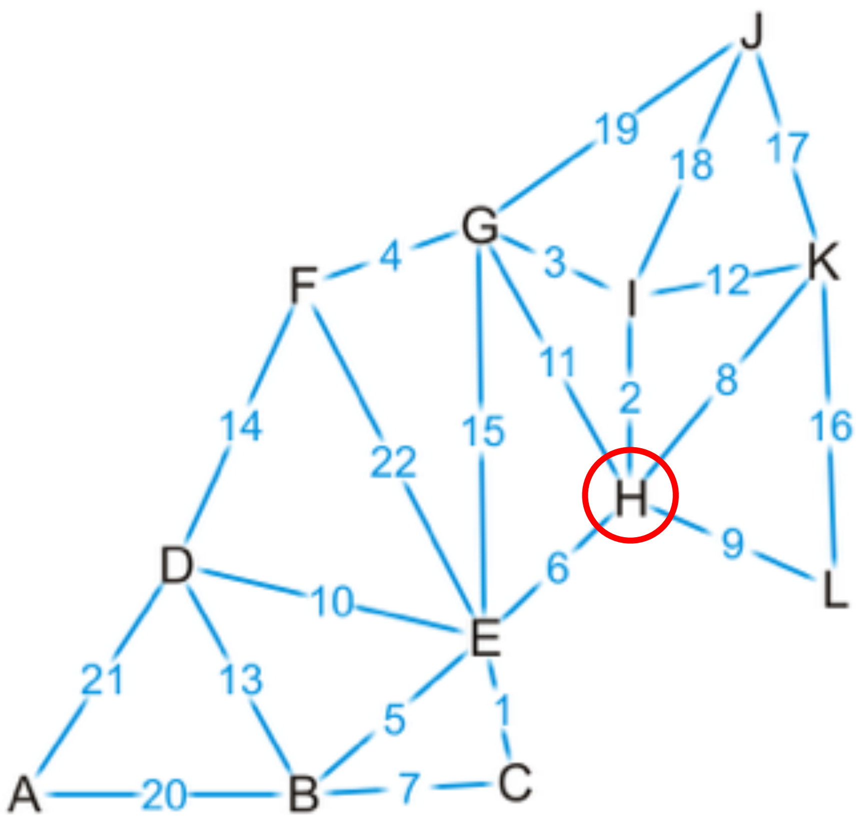


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	8	K
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We visit vertex H: the shortest path is (K, H) of length 8

- Vertex H has four unvisited neighbors: E, G, I, L



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	**T**	**8**	**K**
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Consider these paths:

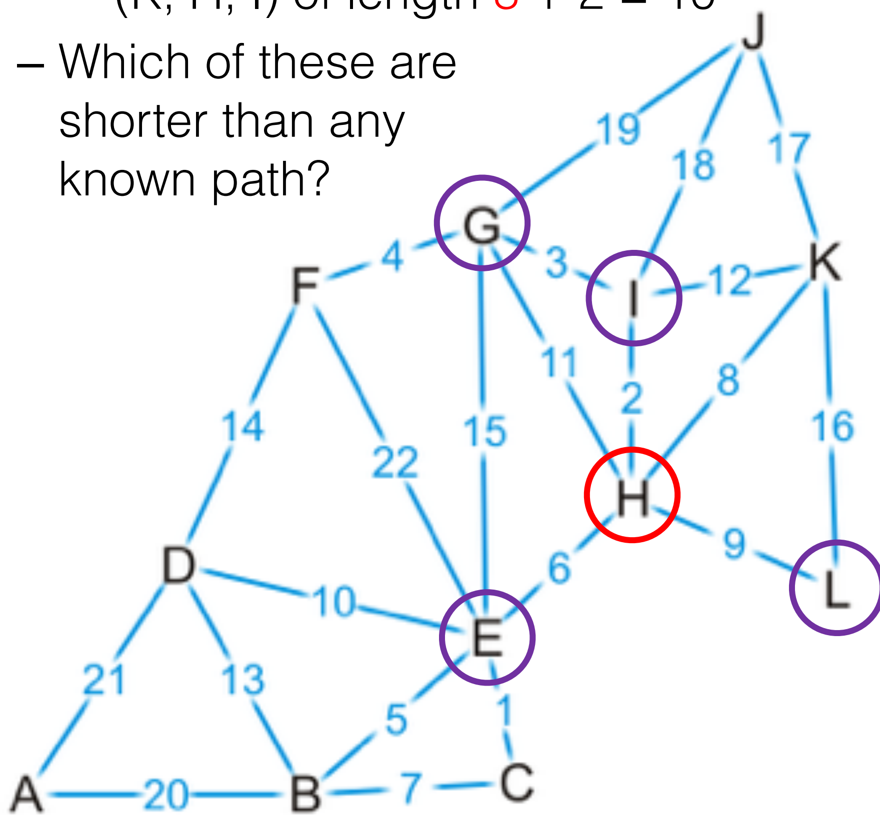
(K, H, E) of length 8 + 6 = 14

(K, H, I) of length 8 + 2 = 10

(K, H, G) of length 8 + 11 = 19

(K, H, L) of length 8 + 9 = 17

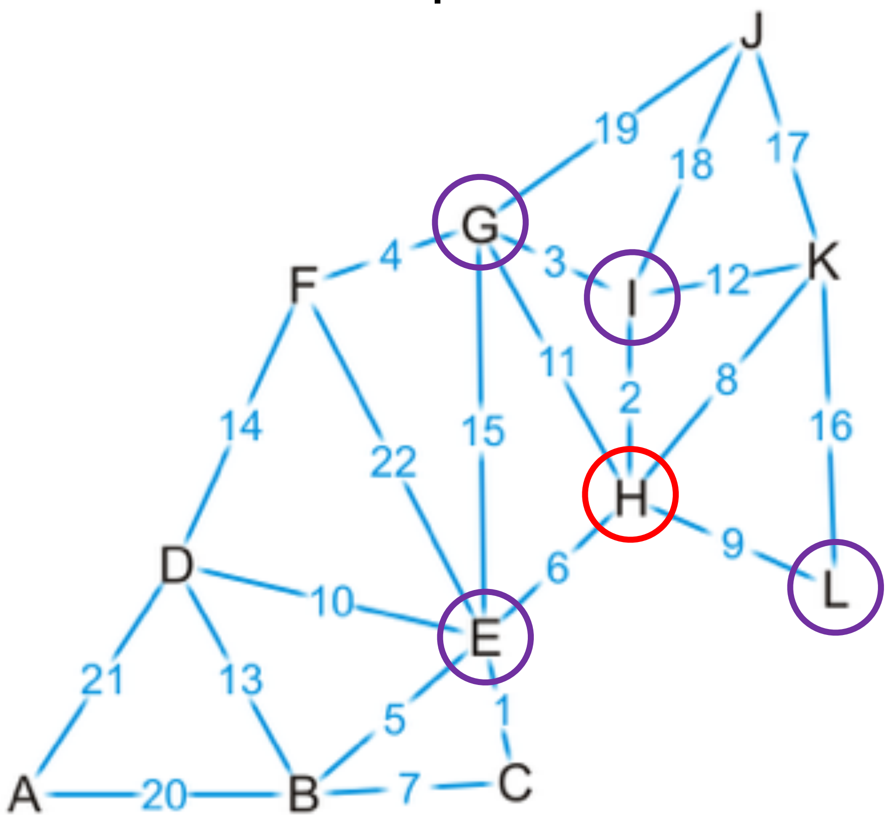
– Which of these are shorter than any known path?



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	**T**	**8**	**K**
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We already have a shorter path (K, L), but we update the other three

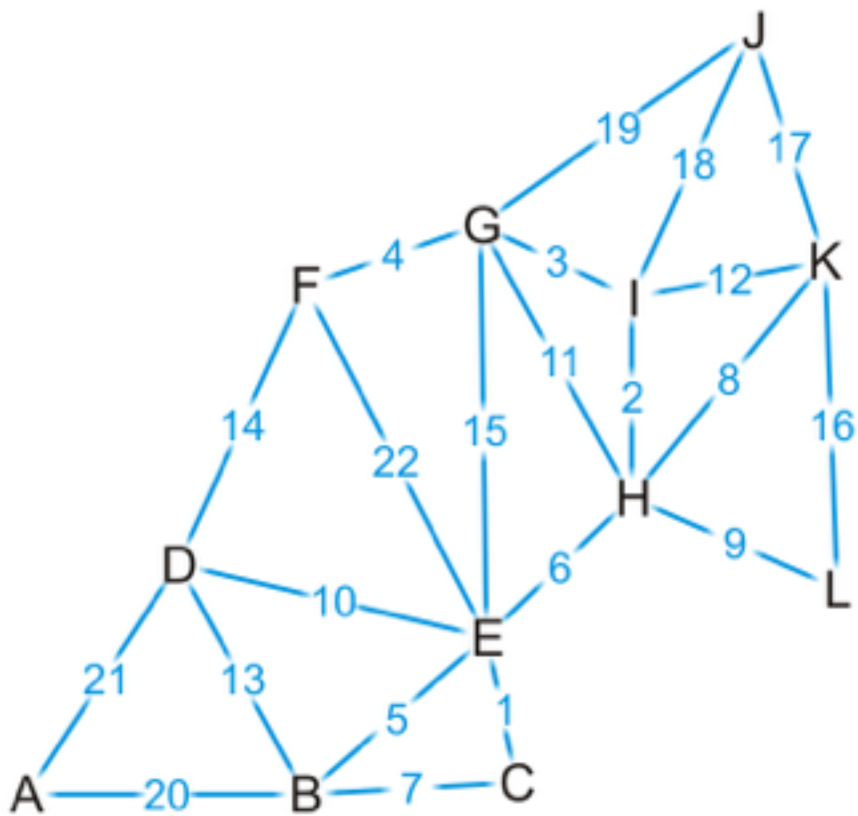


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	**14**	**H**
F	F	∞	Ø
G	F	**19**	**H**
H	**T**	**8**	**K**
I	F	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We are finished with vertex H

– Which vertex do we visit next?

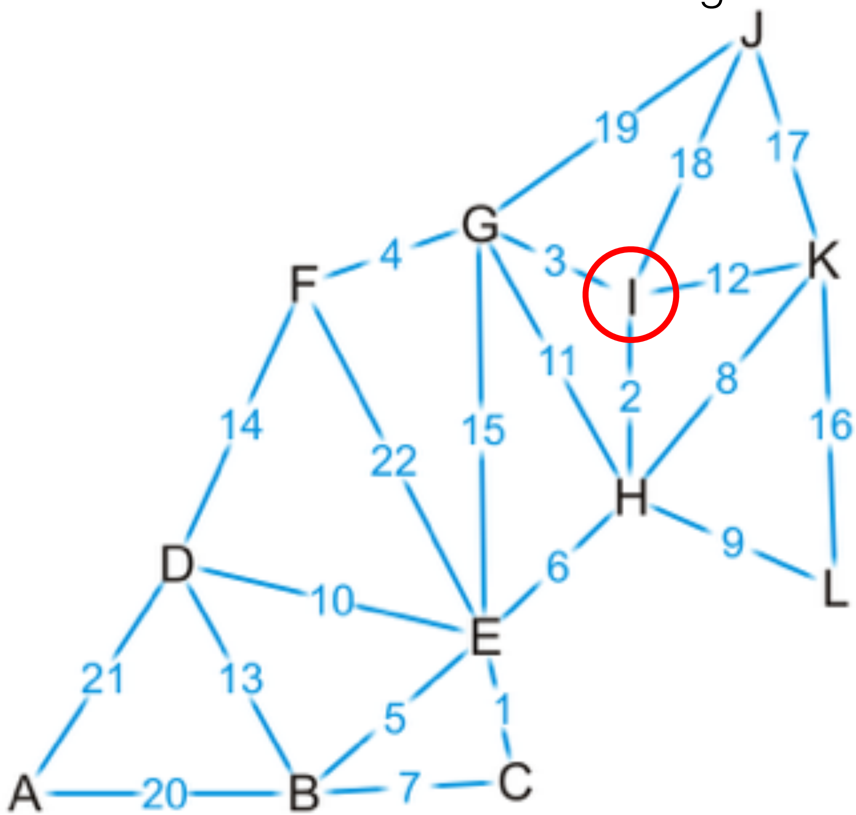


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	F	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

The path (K, H, I) is the shortest path from K to I of length 10

- Vertex I has two unvisited neighbors: G and J

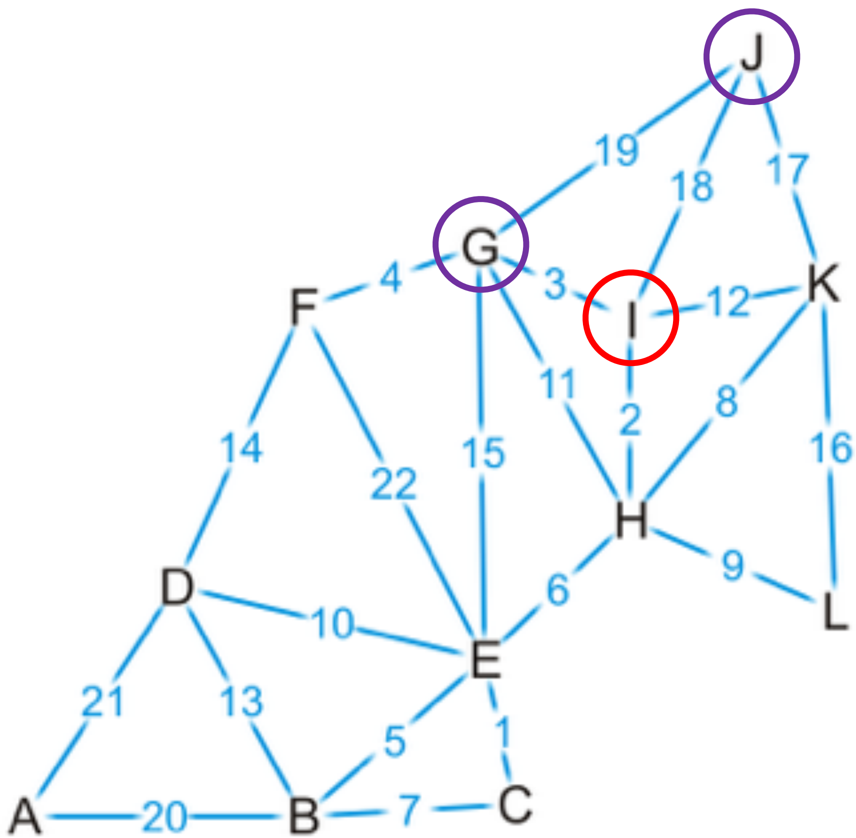


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	**T**	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Consider these paths:

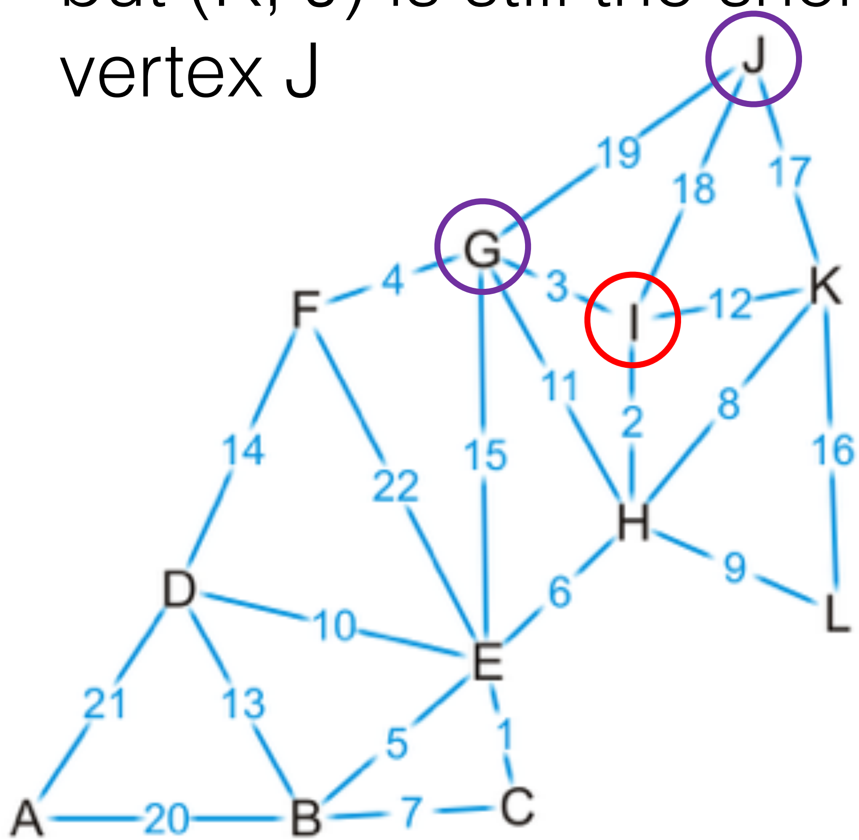
(K, H, I, G) of length 10 + 3 = 13 (K, H, I, J) of length 10 + 18 = 28



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	**T**	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

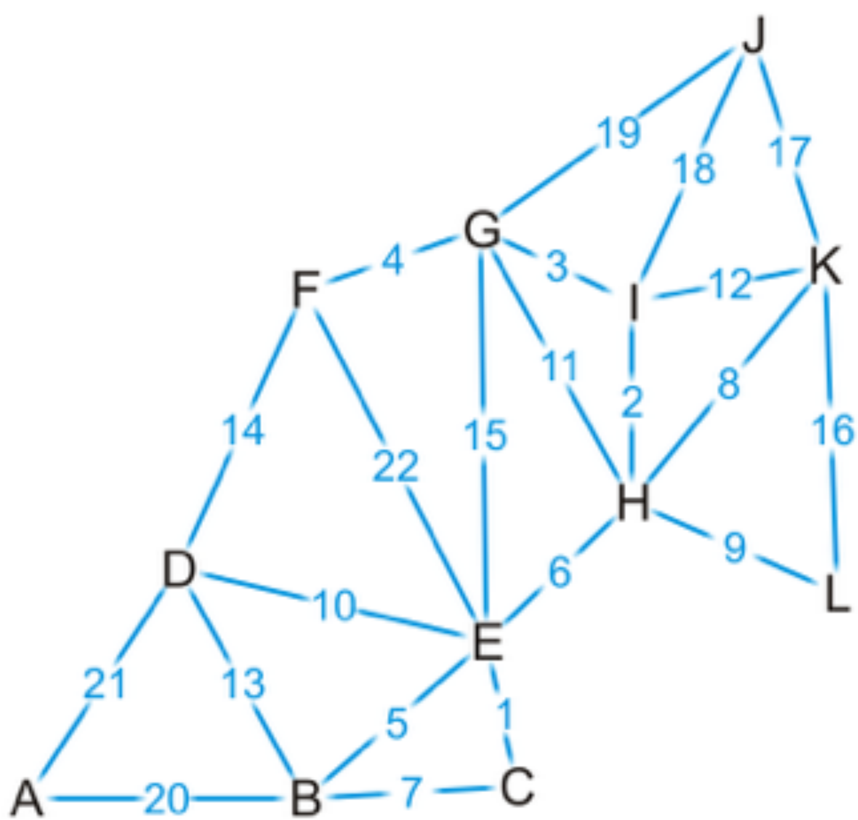
We have discovered a shorter path to vertex G, but (K, J) is still the shortest known path to vertex J



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	**13**	**I**
H	T	8	K
I	**T**	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Which vertex can we visit next?

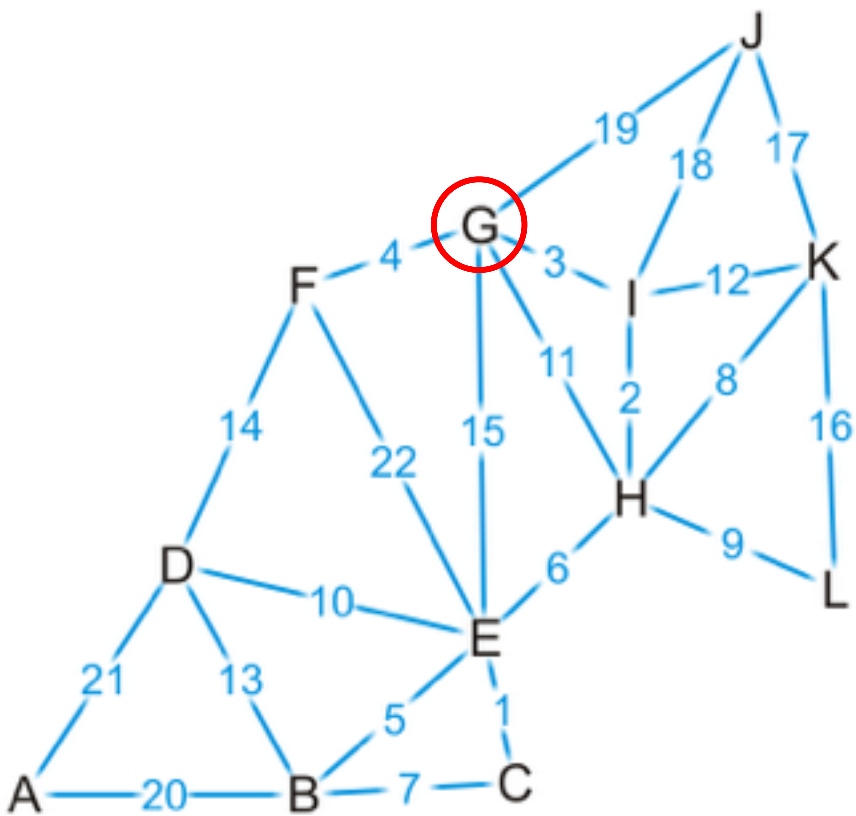


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

The path (K, H, I, G) is the shortest path from K to G of length 13

– Vertex G has three unvisited neighbors: E, F and J



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

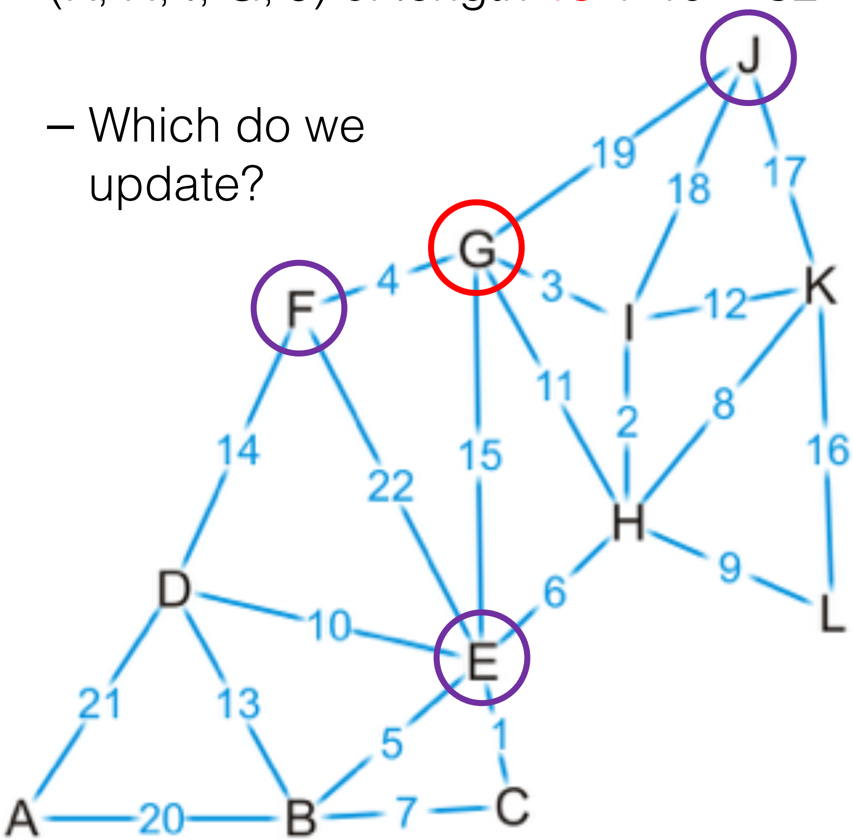
Consider these paths:

(K, H, I, G, E) of length 13 + 15 = 28

(K, H, I, G, J) of length 13 + 19 = 32

(K, H, I, G, F) of length 13 + 4 = 17

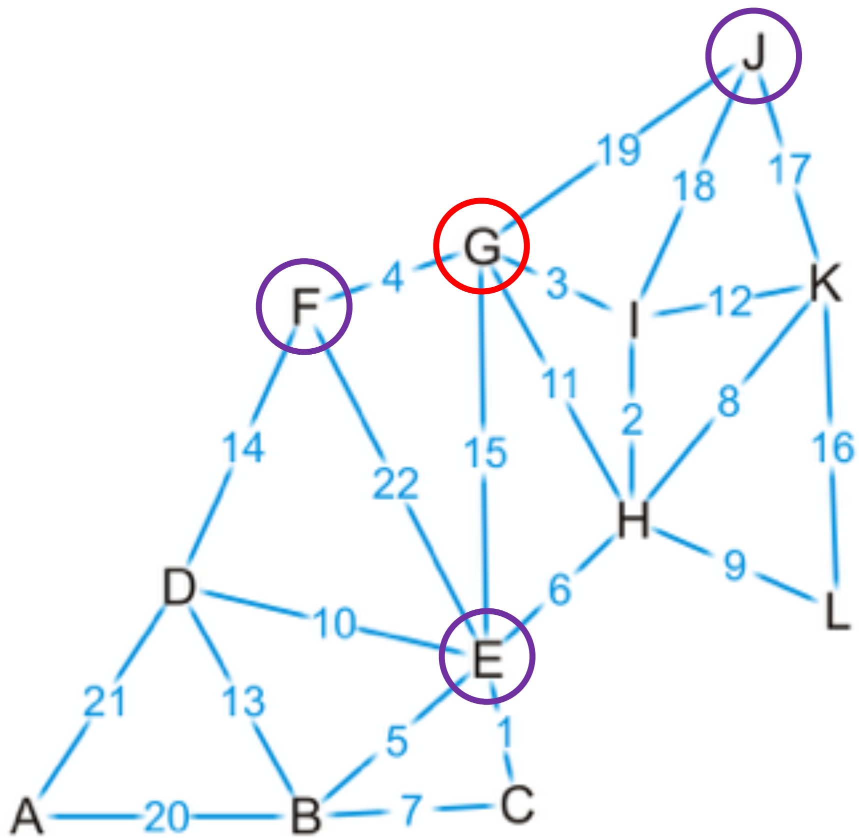
– Which do we update?



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	**T**	**13**	**I**
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

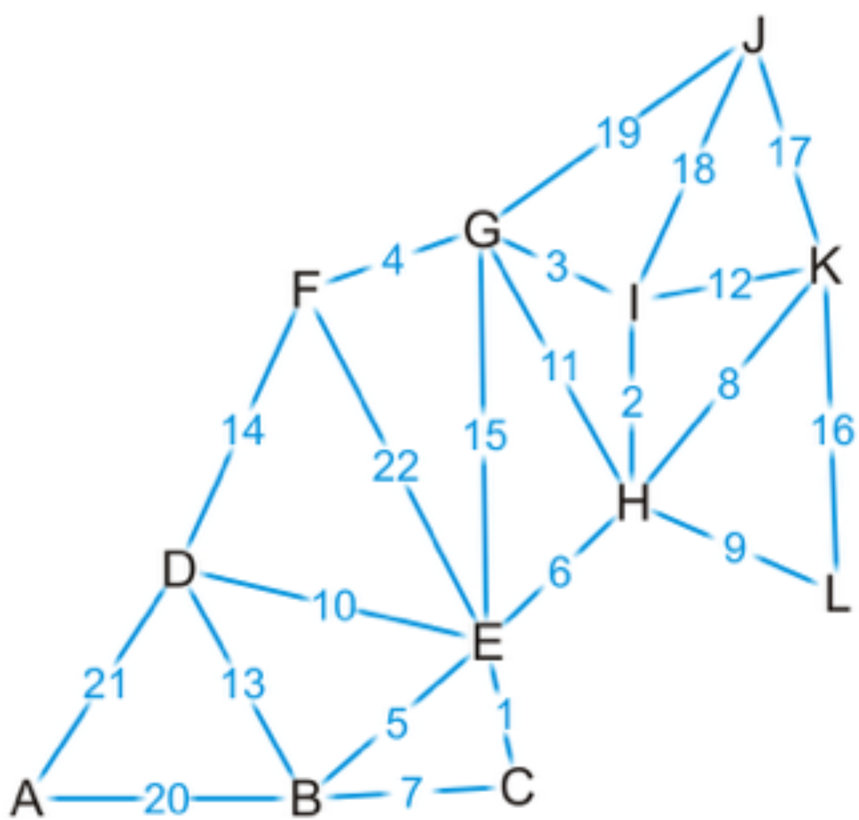
We have now found a path to vertex F



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	**17**	**G**
G	**T**	**13**	**I**
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Where do we visit next?

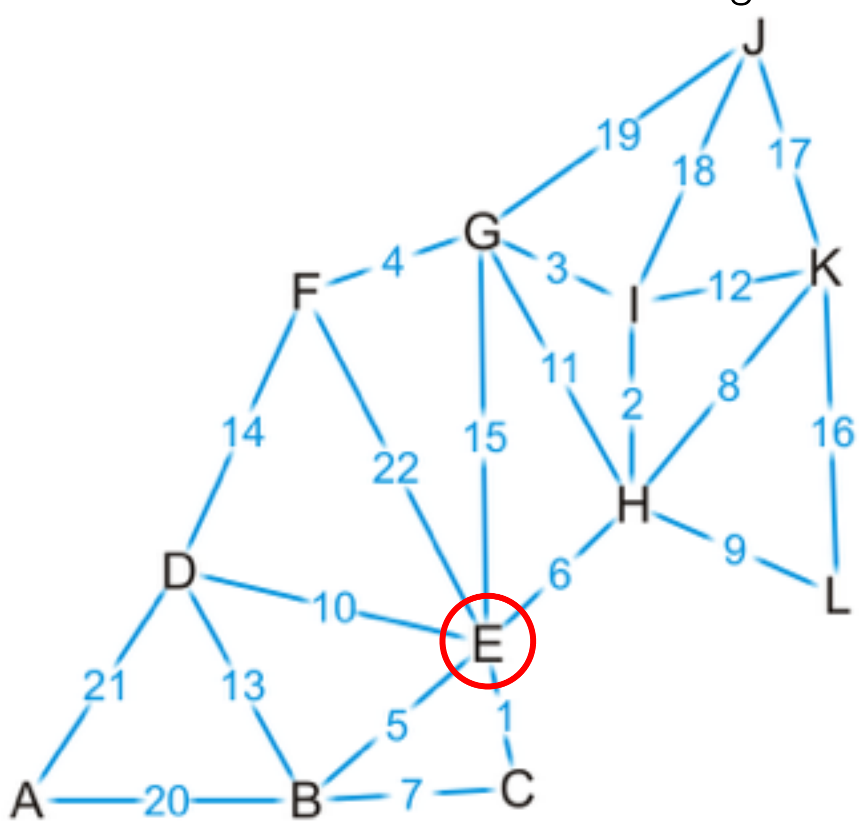


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

The path (K, H, E) is the shortest path from K to E of length 14

– Vertex G has four unvisited neighbors: B, C, D and F

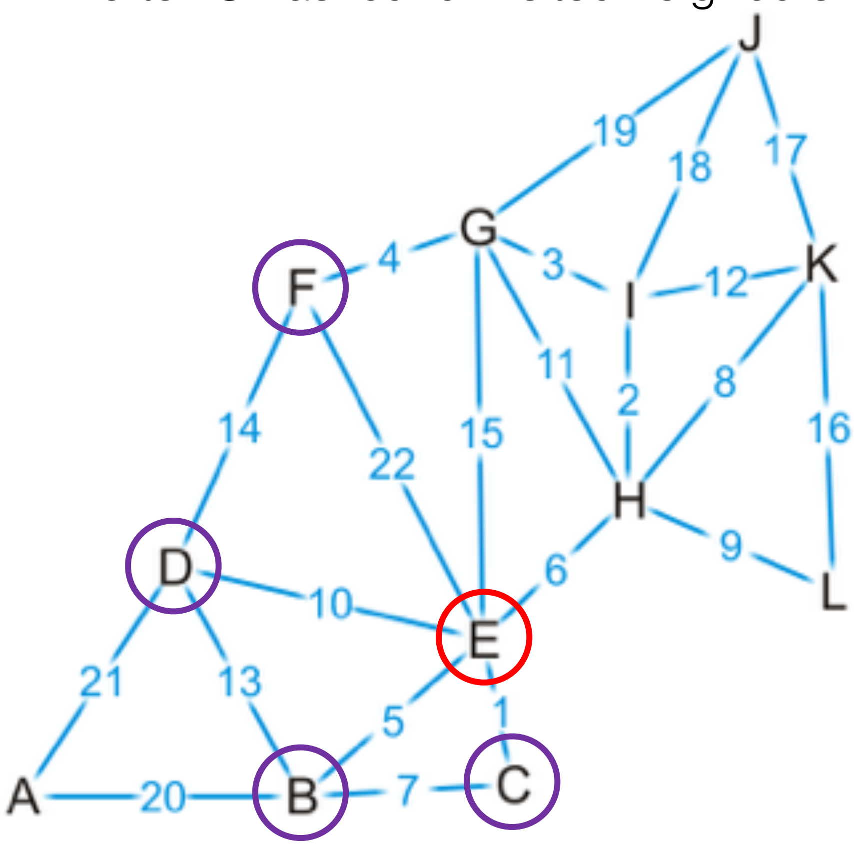


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

The path (K, H, E) is the shortest path from K to E of length 14

– Vertex G has four unvisited neighbors: B, C, D and F



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Consider these paths:

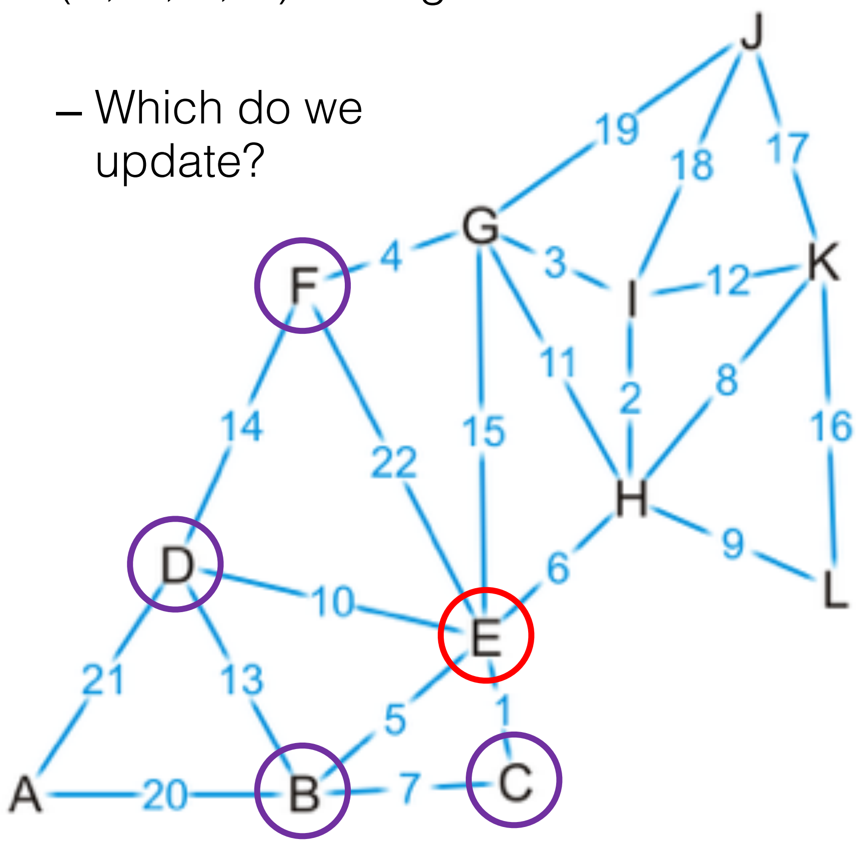
(K, H, E, B) of length 14 + 5 = 19

(K, H, E, D) of length 14 + 10 = 24

(K, H, E, C) of length 14 + 1 = 15

(K, H, E, F) of length 14 + 22 = 36

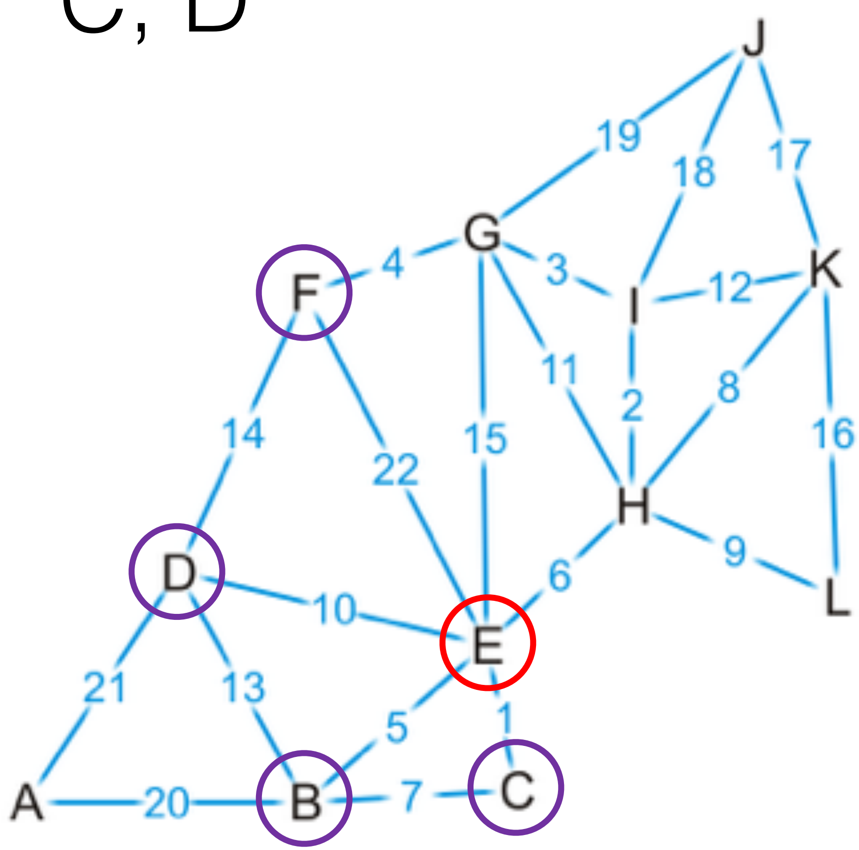
– Which do we update?



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

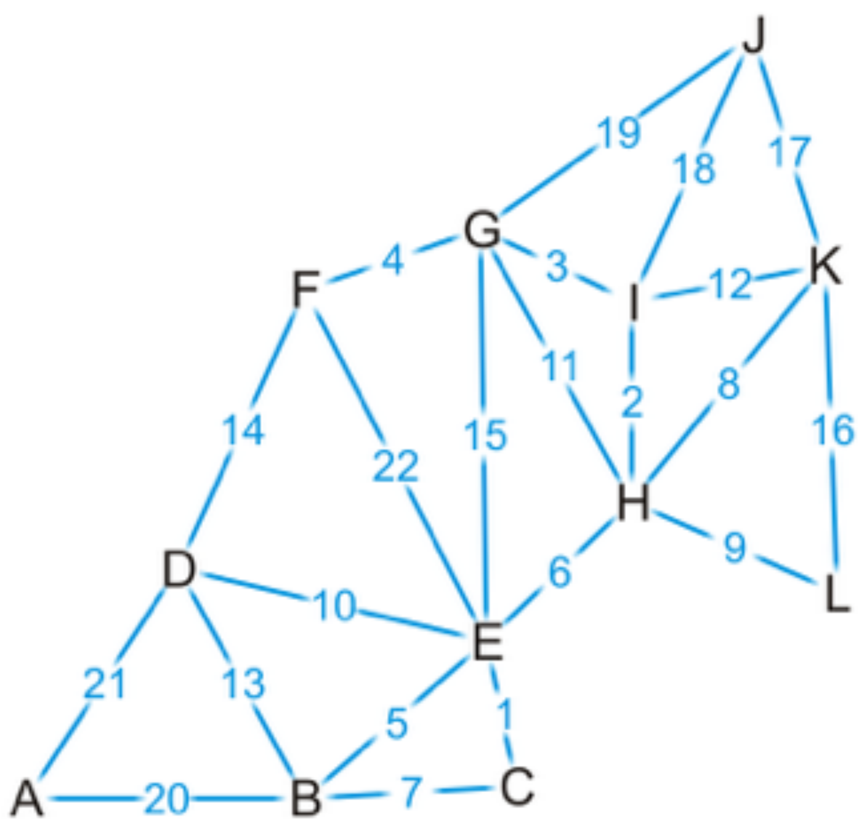
We've discovered paths to vertices B, C, D



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	**19**	**E**
C	F	**15**	**E**
D	F	**24**	**E**
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Which vertex is next?

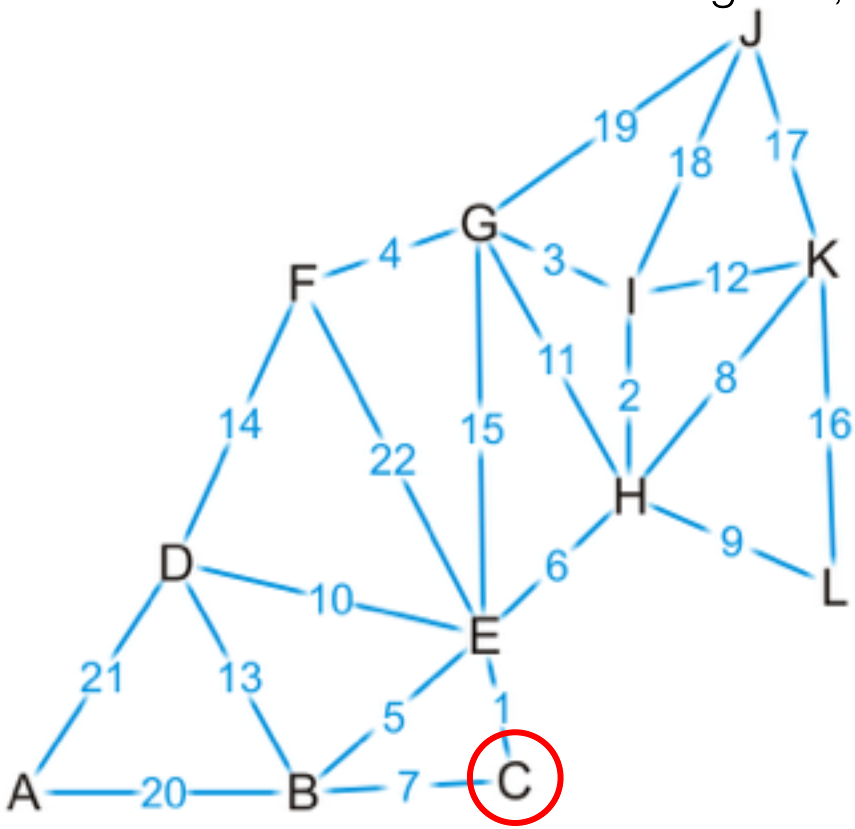


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	F	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We've found that the path (K, H, E, C) of length 15 is the shortest path from K to C

– Vertex C has one unvisited neighbor, B

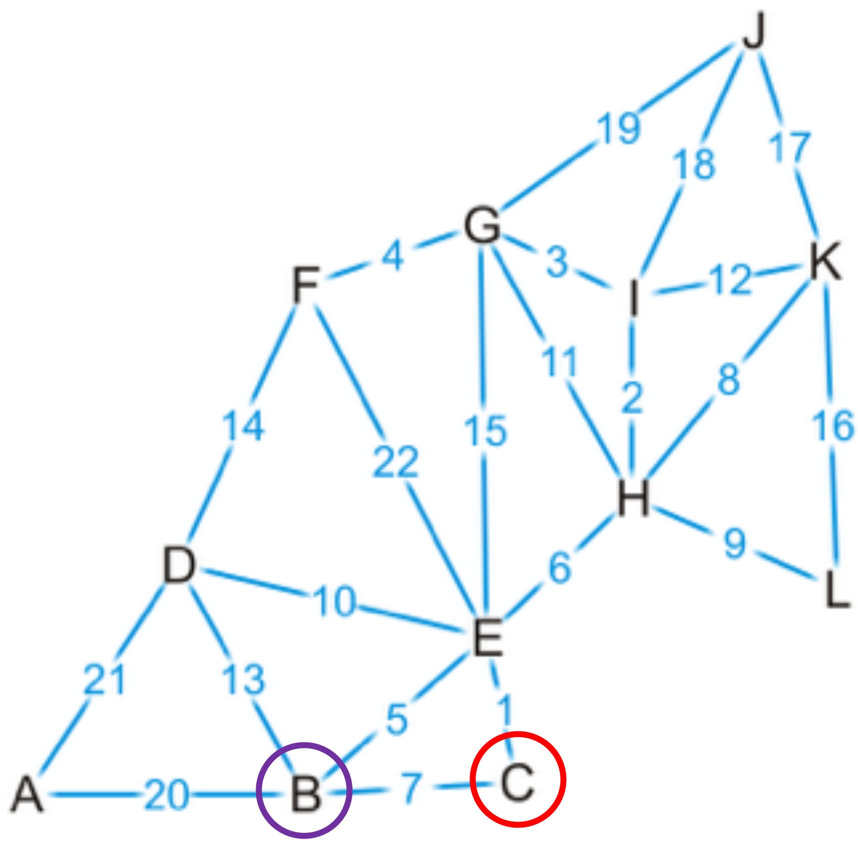


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

The path (K, H, E, C, B) is of length 15 + 7 = 22

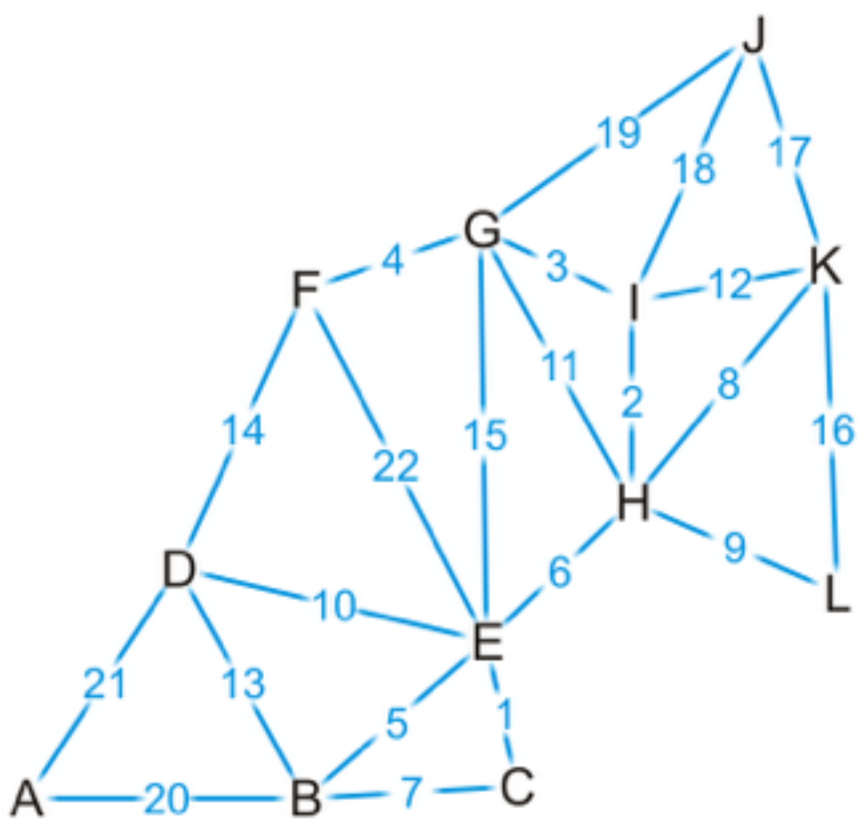
– We have already discovered a shorter path through vertex E



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	**T**	**15**	**E**
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Where to next?

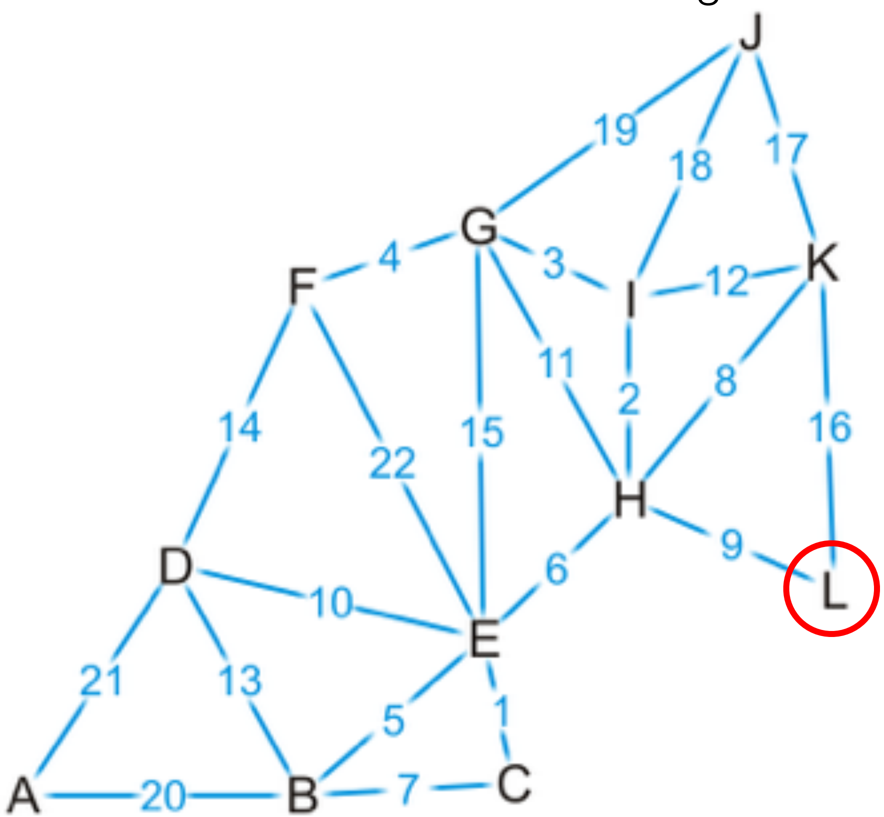


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We now know that (K, L) is the shortest path between these two points

- Vertex L has no unvisited neighbors

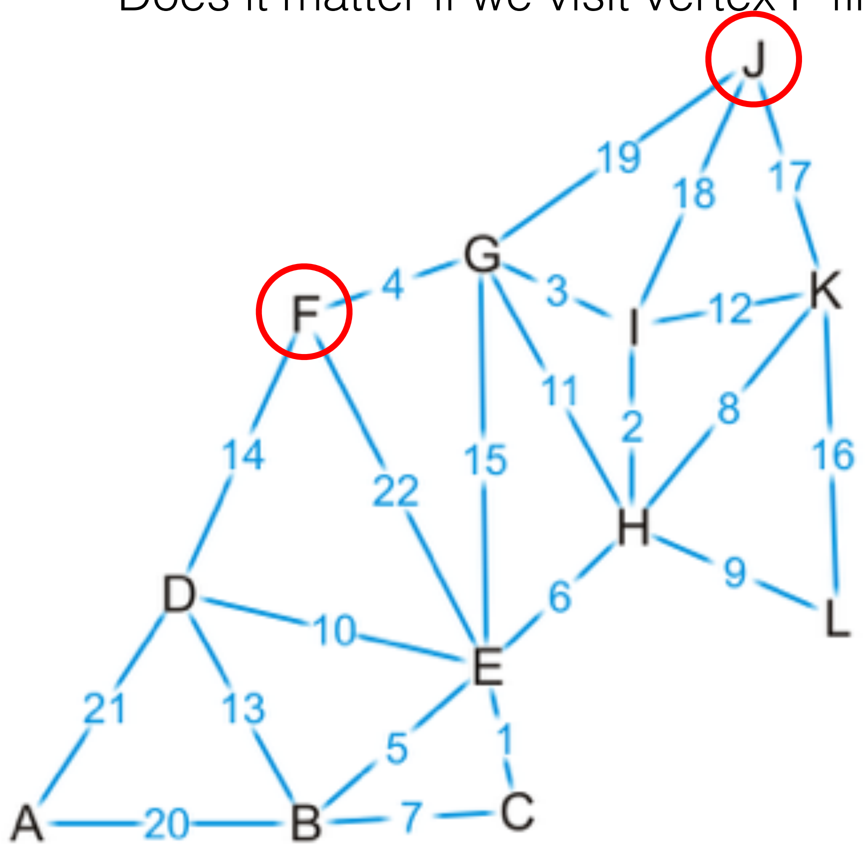


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	**T**	**16**	**K**

Example

Where to next?

– Does it matter if we visit vertex F first or vertex J first?

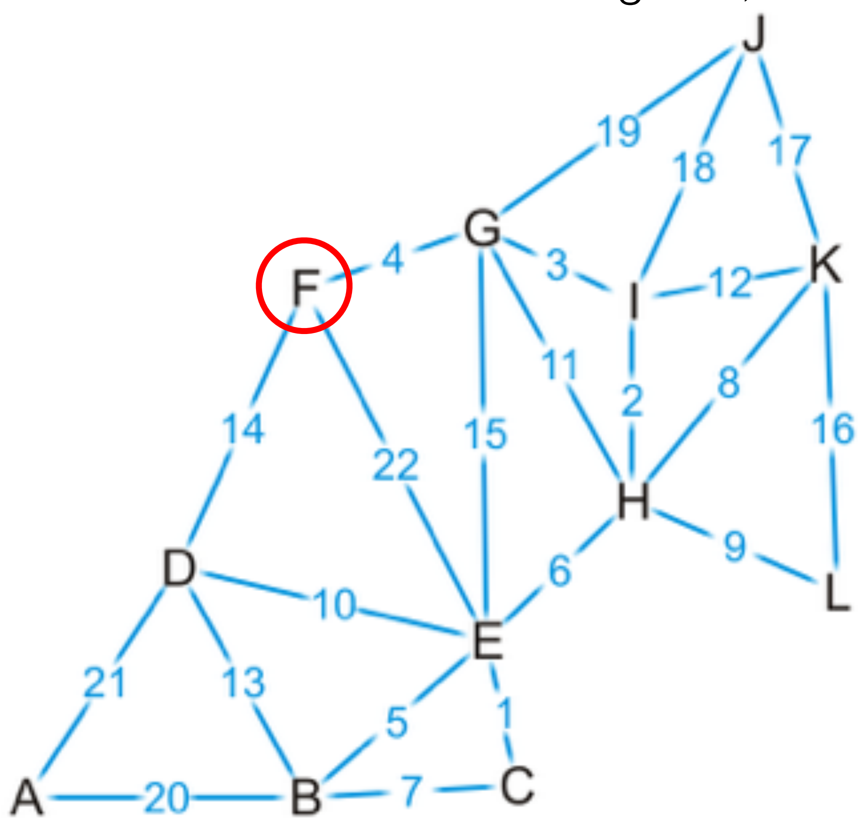


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Let's visit vertex F first

– It has one unvisited neighbor, vertex D

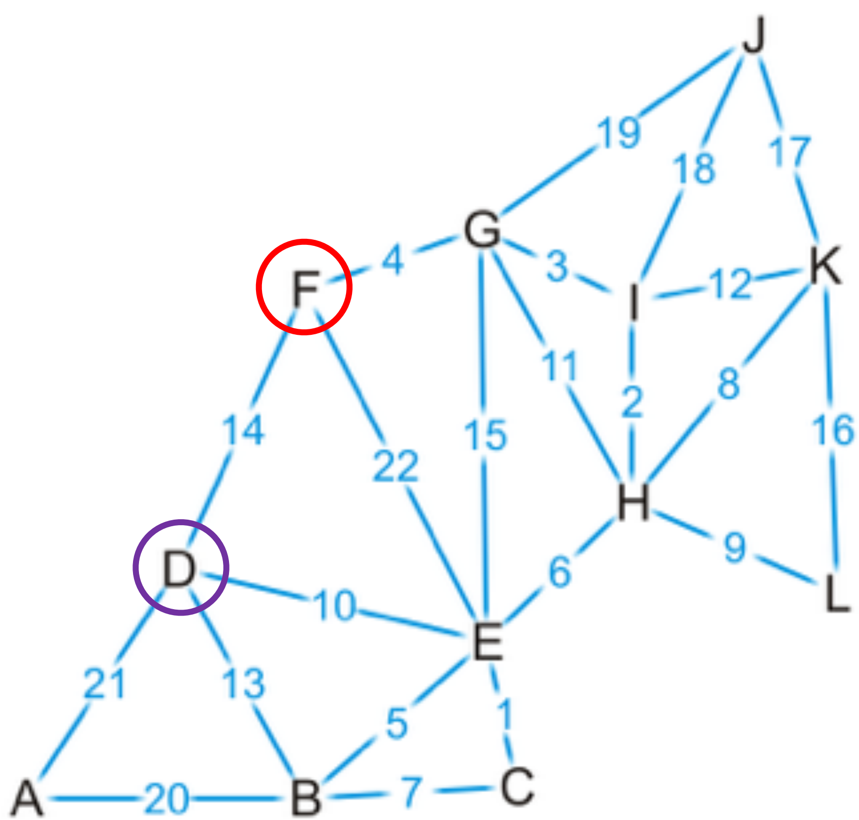


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

The path (K, H, I, G, F, D) is of length 17 + 14 = 31

– This is longer than the path we've already discovered

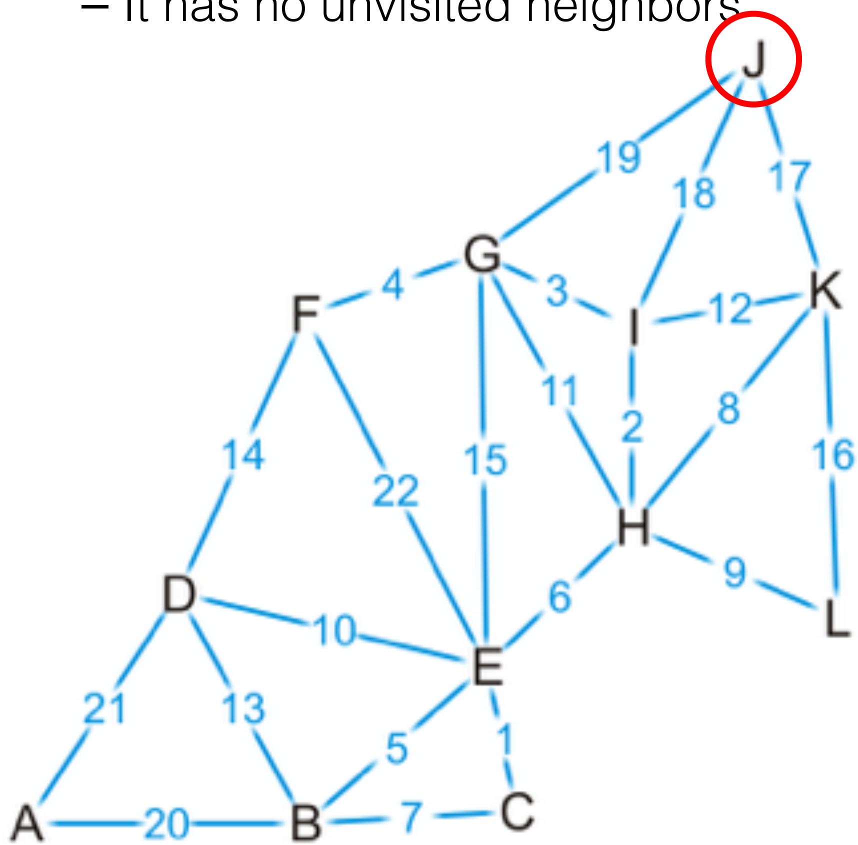


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Now we visit vertex J

– It has no unvisited neighbors



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	**T**	**17**	**K**
K	T	0	Ø
L	T	16	K

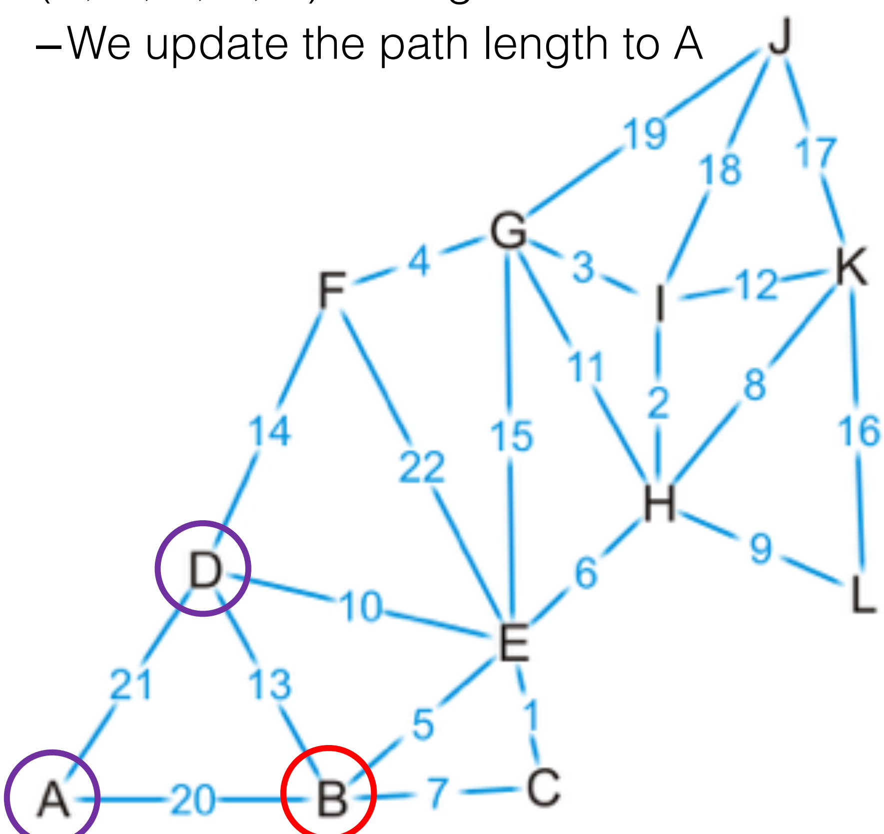
Example

Next we visit vertex B, which has two unvisited neighbors:

(K, H, E, B, A) of length 19 + 20 = 39

– We update the path length to A

(K, H, E, B, D) of length 19 + 13 = 32

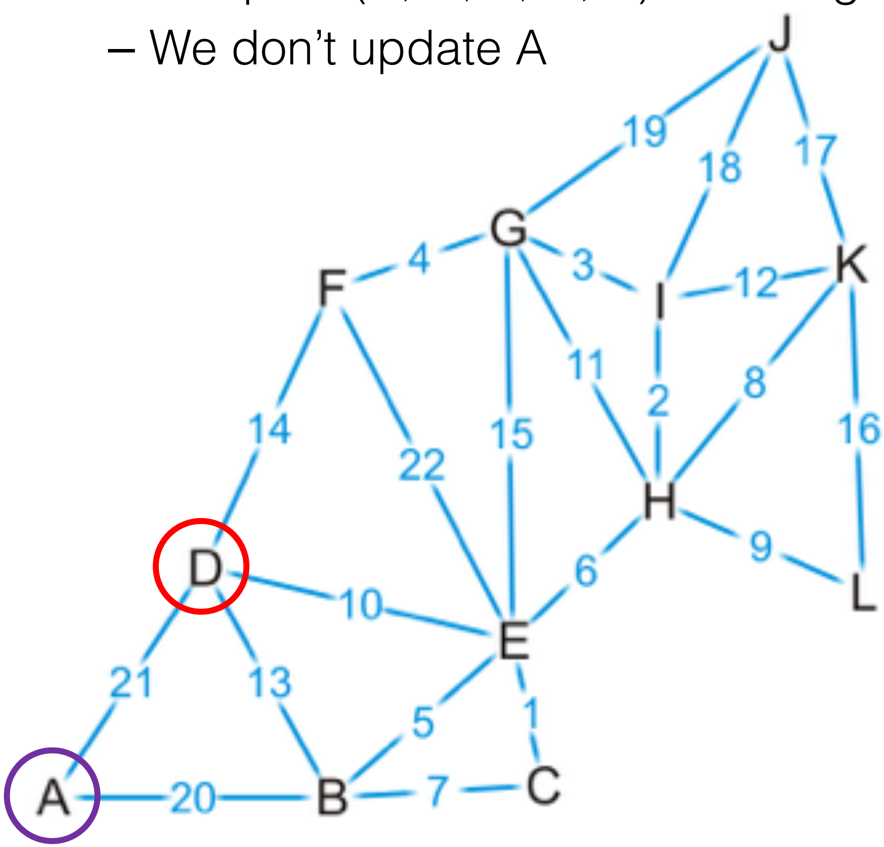


Vertex	Visited	Distance	Previous
A	F	39	B
B	T	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Next we visit vertex D

- The path (K, H, E, D, A) is of length 24 + 21 = 45
- We don't update A

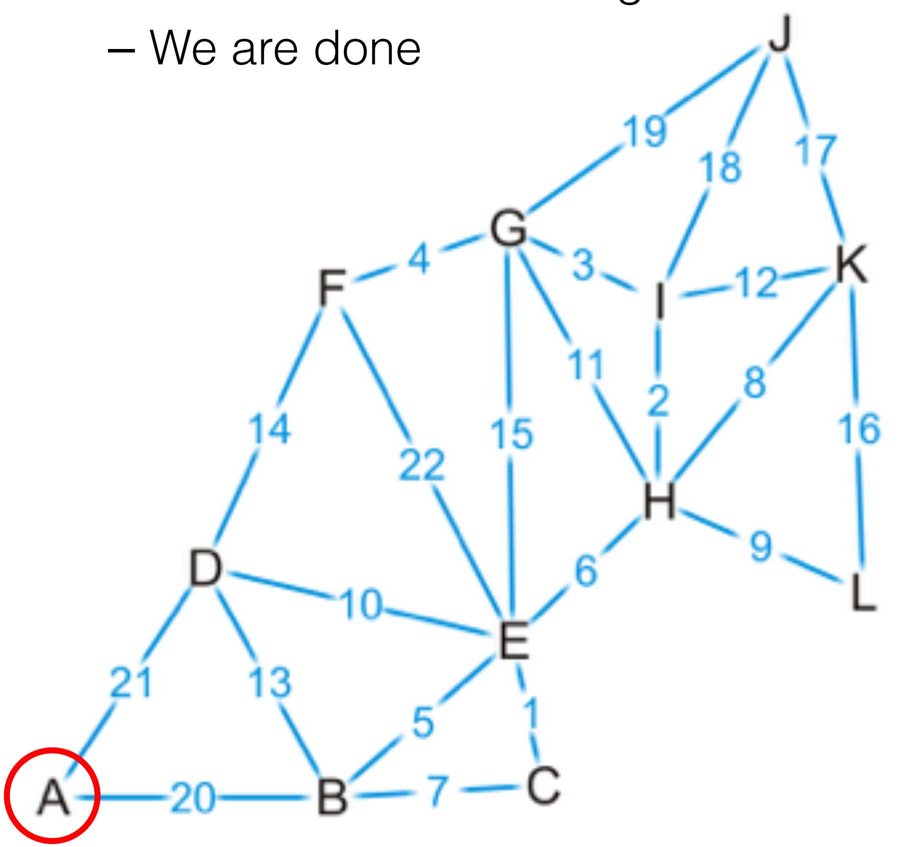


Vertex	Visited	Distance	Previous
A	F	39	B
B	T	19	E
C	T	15	E
D	**T**	**24**	**E**
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Finally, we visit vertex A

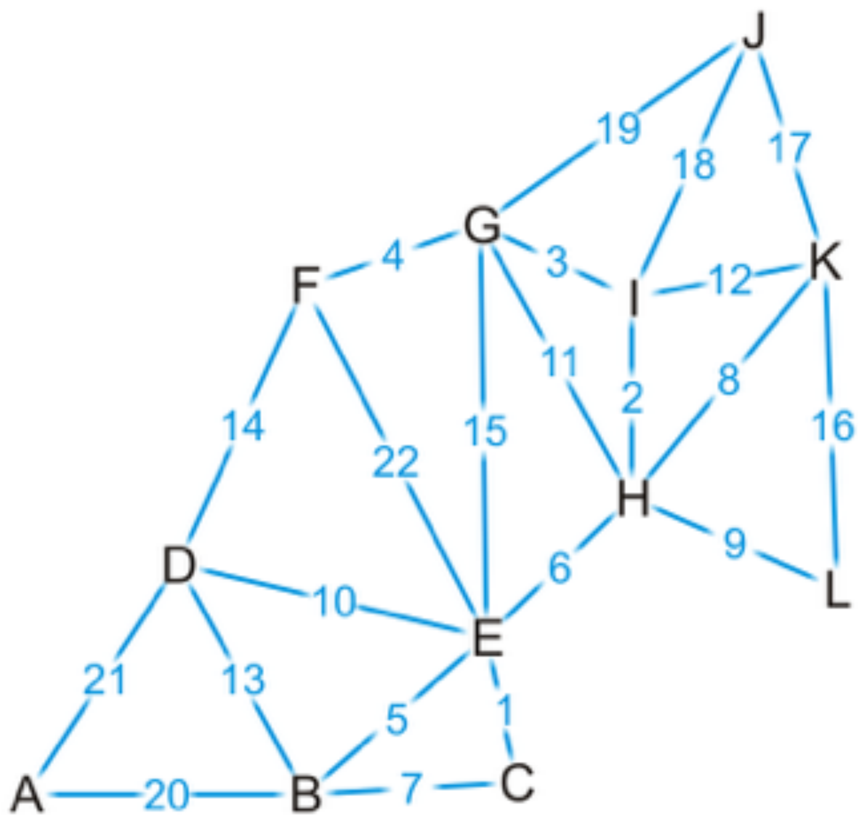
- It has no unvisited neighbors and there are no unvisited vertices left
- We are done



Vertex	Visited	Distance	Previous
A	**T**	**39**	**B**
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

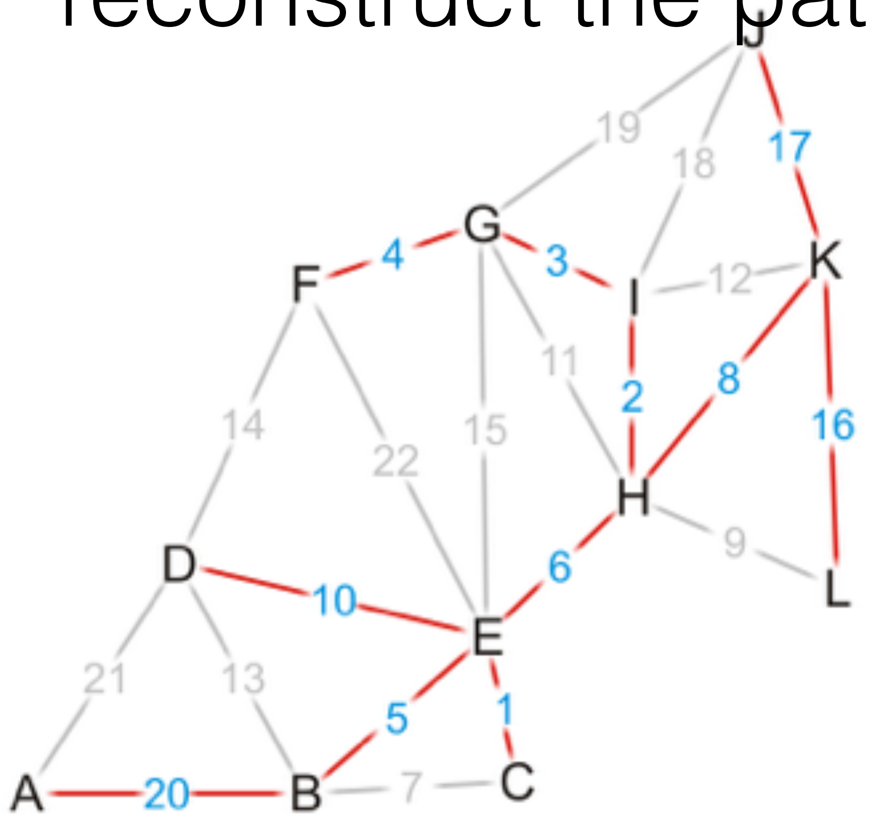
Thus, we have found the shortest path from vertex K to each of the other vertices



Vertex	Visited	Distance	Previous
A	T	39	B
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Using the *previous* pointers, we can reconstruct the paths



Vertex	Visited	Distance	Previous
A	T	39	B
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Comments on Dijkstra's algorithm

Questions:

- What if at some point, all unvisited vertices have a distance ∞?

- What if we just want to find the shortest path between vertices v_j and v_k?

- Does the algorithm change if we have a directed graph?

Comments on Dijkstra's algorithm

Questions:

- What if at some point, all unvisited vertices have a distance ∞?
  - This means that the graph is unconnected
  - We have found the shortest paths to all vertices in the connected subgraph containing the source vertex
- What if we just want to find the shortest path between vertices v_j and v_k?
- Does the algorithm change if we have a directed graph?

Comments on Dijkstra's algorithm

Questions:

- What if at some point, all unvisited vertices have a distance ∞?
  - This means that the graph is unconnected
  - We have found the shortest paths to all vertices in the connected subgraph containing the source vertex
- What if we just want to find the shortest path between vertices v_j and v_k?
  - Apply the same algorithm, but stop when we are <u>visiting</u> vertex v_k
- Does the algorithm change if we have a directed graph?

Comments on Dijkstra's algorithm

Questions:

- What if at some point, all unvisited vertices have a distance ∞?
  - This means that the graph is unconnected
  - We have found the shortest paths to all vertices in the connected subgraph containing the source vertex
- What if we just want to find the shortest path between vertices v_j and v_k?
  - Apply the same algorithm, but stop when we are <u>visiting</u> vertex v_k
- Does the algorithm change if we have a directed graph?
  - No

Implementation and analysis

The initialization requires $\Theta(|V|)$ memory and run time

We iterate $|V| - 1$ times, each time finding next closest vertex to the source

- Iterating through the table requires is $\Theta(|V|)$ time
- Each time we find a vertex, we must check all of its neighbors: $\Theta(|V|(|V| + |V|)) = \Theta(|V|^2)$

Can we do better?

- Recall, we only need the closest vertex
- How about a priority queue?
  - Assume we are using a binary heap
  - We will have to update the heap structure—this requires additional work

Implementation and analysis

The initialization still requires $\Theta(|V|)$ memory and run time

- The priority queue will also requires $O(|V|)$ memory

We iterate $|V|$ times, each time finding the ***closest*** vertex to the source

- Place the distances into a priority queue
- The size of the priority queue is $O(|V|)$
- Thus, the work required for this is $O(|V| \ln(|V|))$

Is this all the work that is necessary?

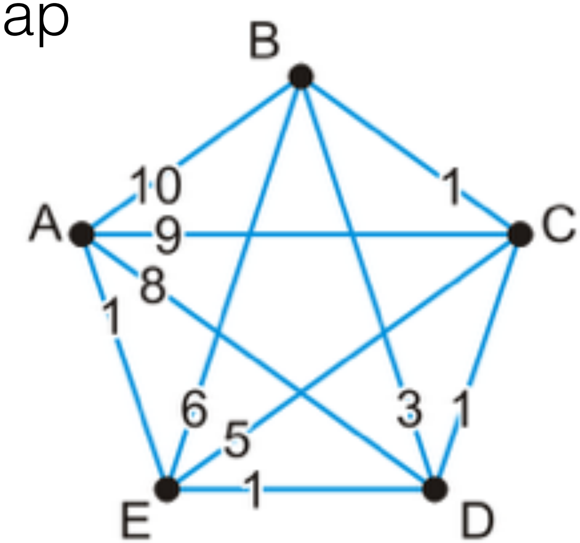
- Recall that each edge visited may result in a new edge being pushed to the very top of the heap
- Thus, the work required for this is $O(|E| \ln(|V|))$

Thus, the total run time is $O(|V| \ln(|V|) + |E| \ln(|V|)) = O(|E| \ln(|V|))$

Implementation and analysis

Here is an example of a worst-case scenario:

- Immediately, all of the vertices are placed into the queue
- Each time a vertex is visited, all the remaining vertices are checked, and in succession, each is pushed to the top of the binary heap



Summary

We have seen an algorithm for finding single-source shortest paths

- Start with the initial vertex
- Continue finding the next vertex that is closest

Dijkstra's algorithm always finds the next closest vertex

- It solves the problem in $O(|E| \ln(|V|))$ time