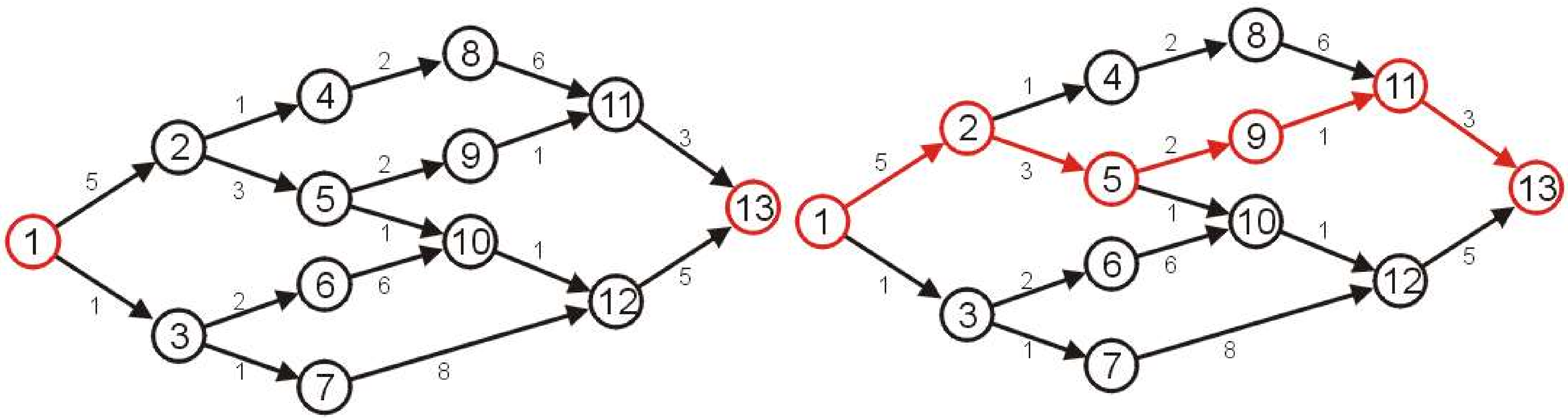
Data Structures

22. Shortest Path Trees

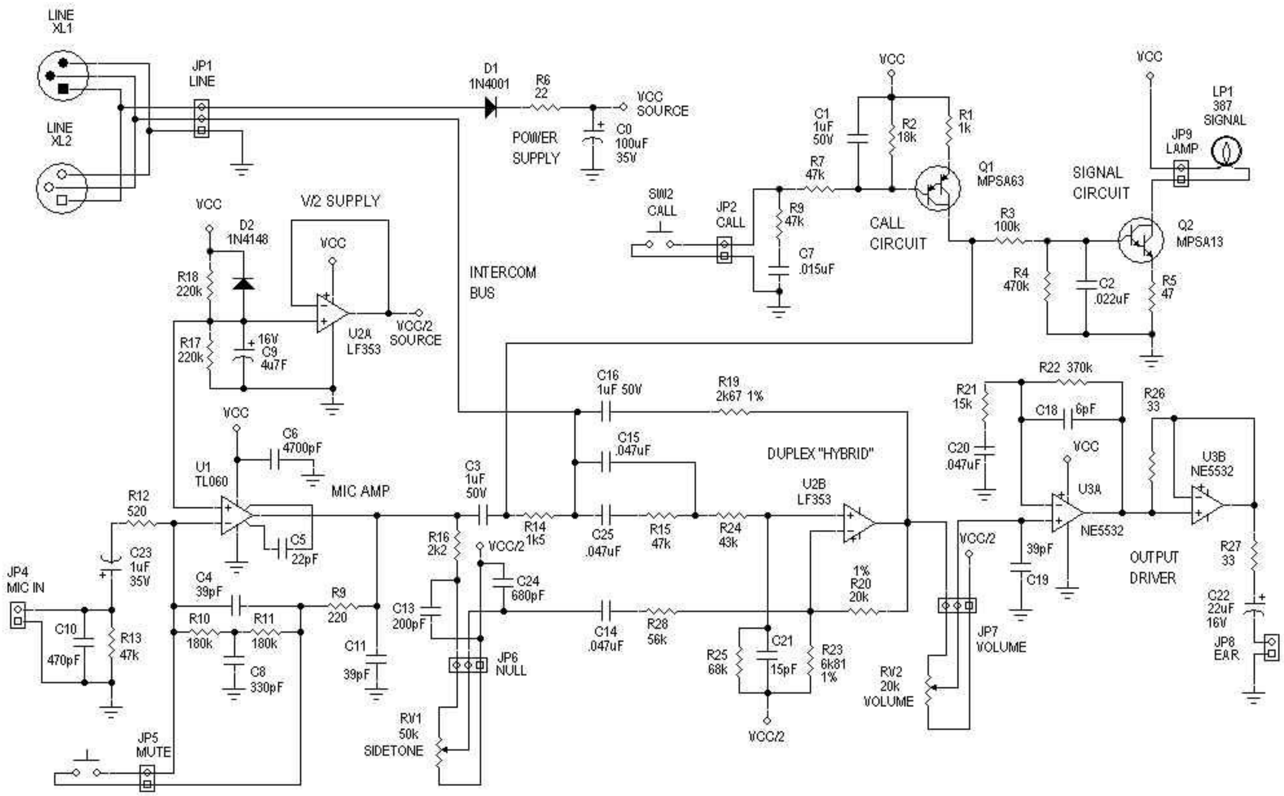
Shortest Path

- Given a weighted graph
  - Problem is to find the shortest path between two given vertices
- Length of a path in a weighted graph
  - Sum of the weights of each of the edges in that path
- Example: Shortest path from vertex 1 to vertex 13
  - Other paths exists but they are longer



Application – Circuit Design

- The time it takes for a change in input to affect an output depends on the shortest path

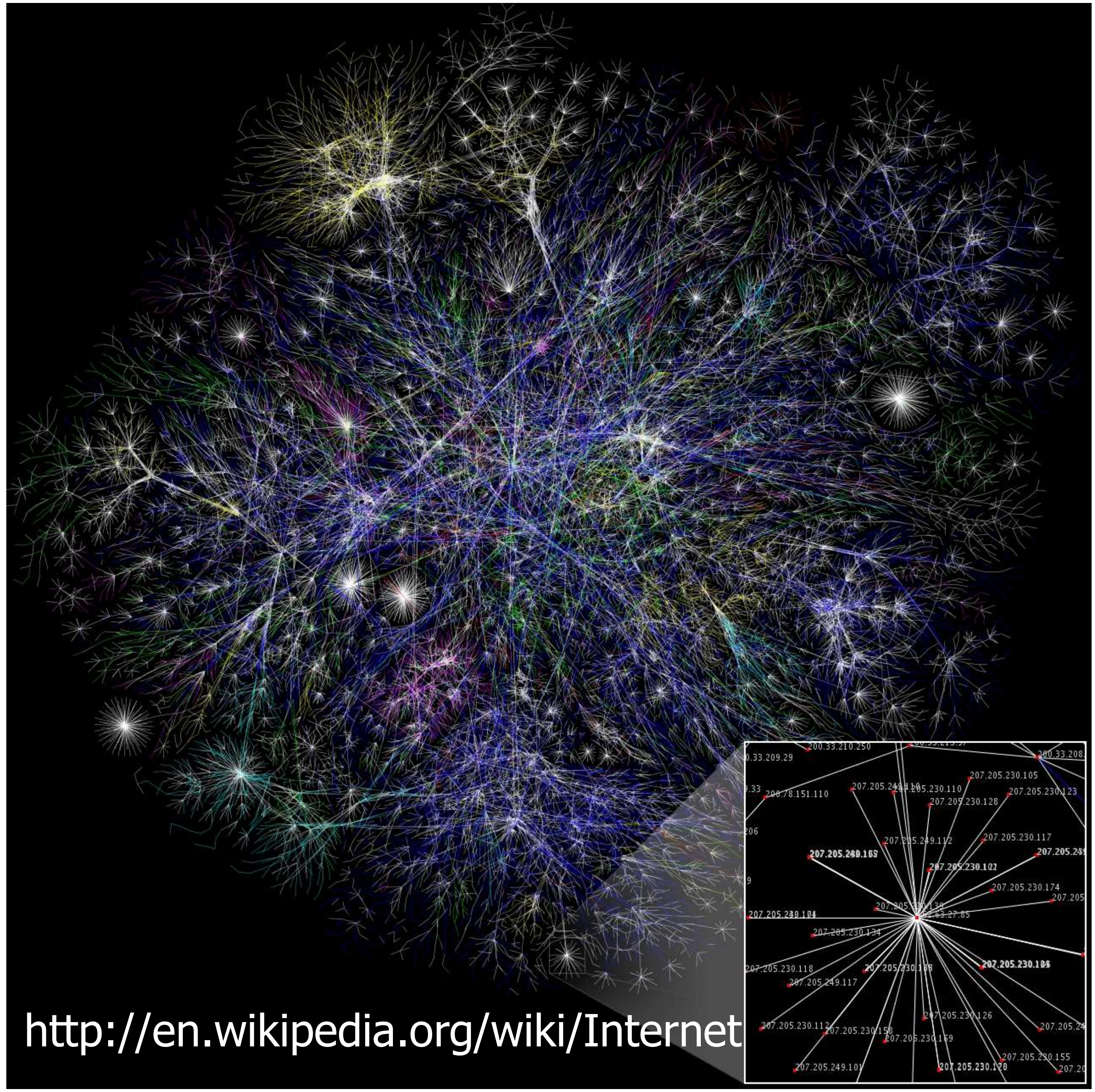


Application – Computer Networks

- 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

Application – Computer Networks

- A visualization of the graph of the routers and their various connections through a portion of the Internet



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

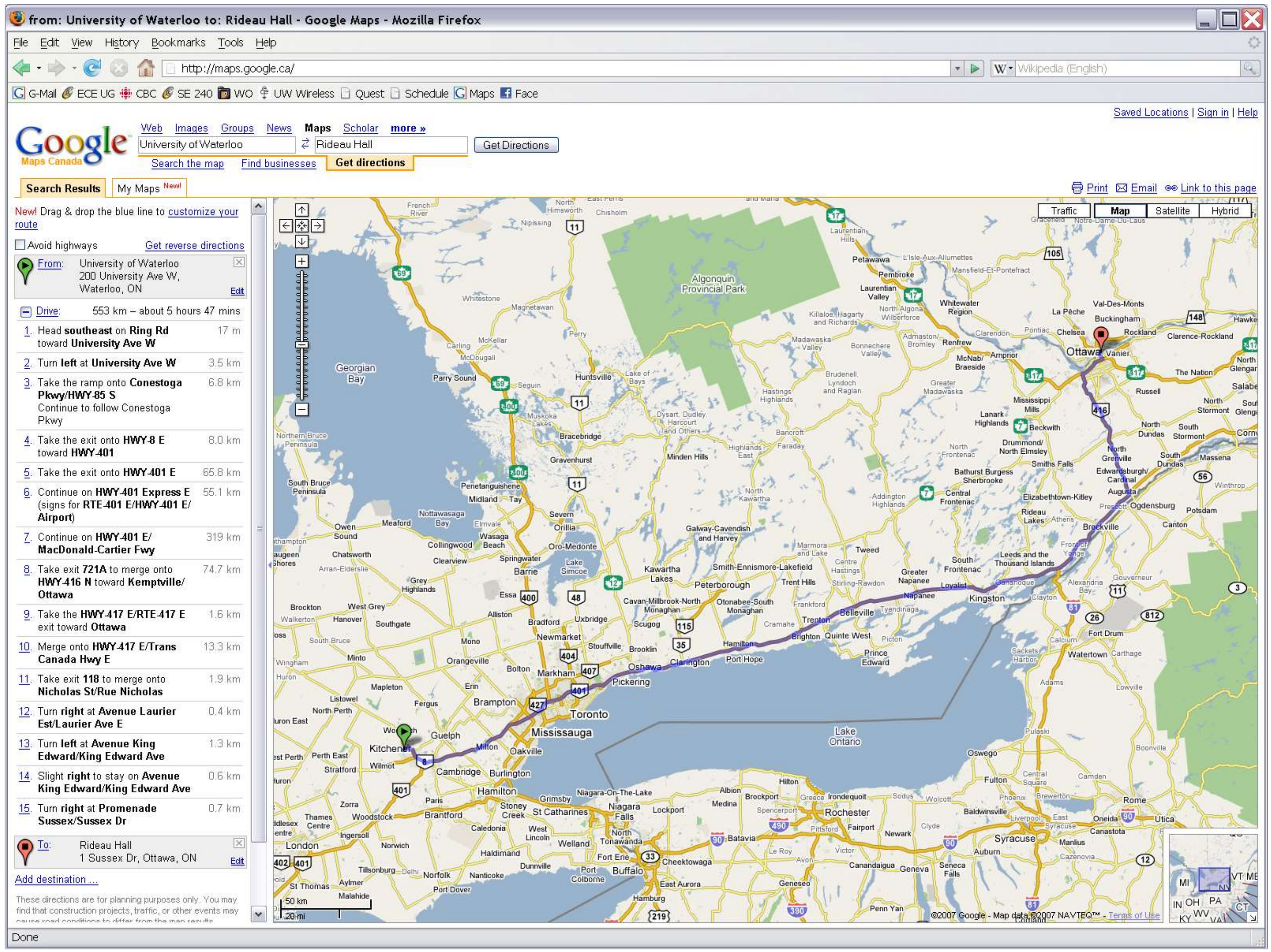
Application – Computer Networks

- The path a packet takes depends on the IP address

- Metrics for measuring the shortest path may include
  - Low latency (minimize time)
  - Minimum hop count (all edges have weight 1)

Application – Traffic

- Find the shortest route between to points on a map
  - Shortest path, however, need not refer to distance…



Variants of Shortest Path

Given a graph `G = (V, E)`

- Single-source shortest paths
  - Find shortest path from a given source vertex s to each vertex $v \in V$
- Single-destination shortest paths
  - Find shortest path to a given destination vertex t from each vertex v
- Single-pair shortest path
  - Find shortest path from u to v for given vertices u and v
- All-pairs shortest-paths
  - Find shortest path from u to v for every pair of vertices u and v

Single Source Shortest Path

Dijkstra's Algorithm

- Problem: From a given source vertex $s \in V$, find the shortest-paths and their weights $w(s,v)$ for all $v \in V$

- Idea of the Algorithm
  - Maintain a set S of vertices whose shortest-path distances from s are known
  - At each step add to S the vertex $v \in V-S$ whose distance estimate from s is minimal
  - Update the distance estimates of vertices adjacent to v

Dijkstra's Algorithm - Pseudocode

```
dist[s] = 0                          // distance to source vertex is zero
p[s] = NULL
for all v ∈ V–{s}
   dist[v] = ∞                       // set all other distances to infinity
S = Ø                                //S, the set of visited vertices is initially empty
Q = V                                //Q, the queue initially contains all vertices
while Q is not emppty                //while the queue is not empty
   u =  mindistance(Q)               //select the element of Q with the min distance
   S = S∪{u}                         // add u to list of visited vertices
   for all v ∈ neighbors[u]
      if dist[v] > dist[u] + w(u,v)     //if new shortest path found
         dist[v] = dist[u] + w(u,v)     //set new value of shortest path
         p[v] = u
```

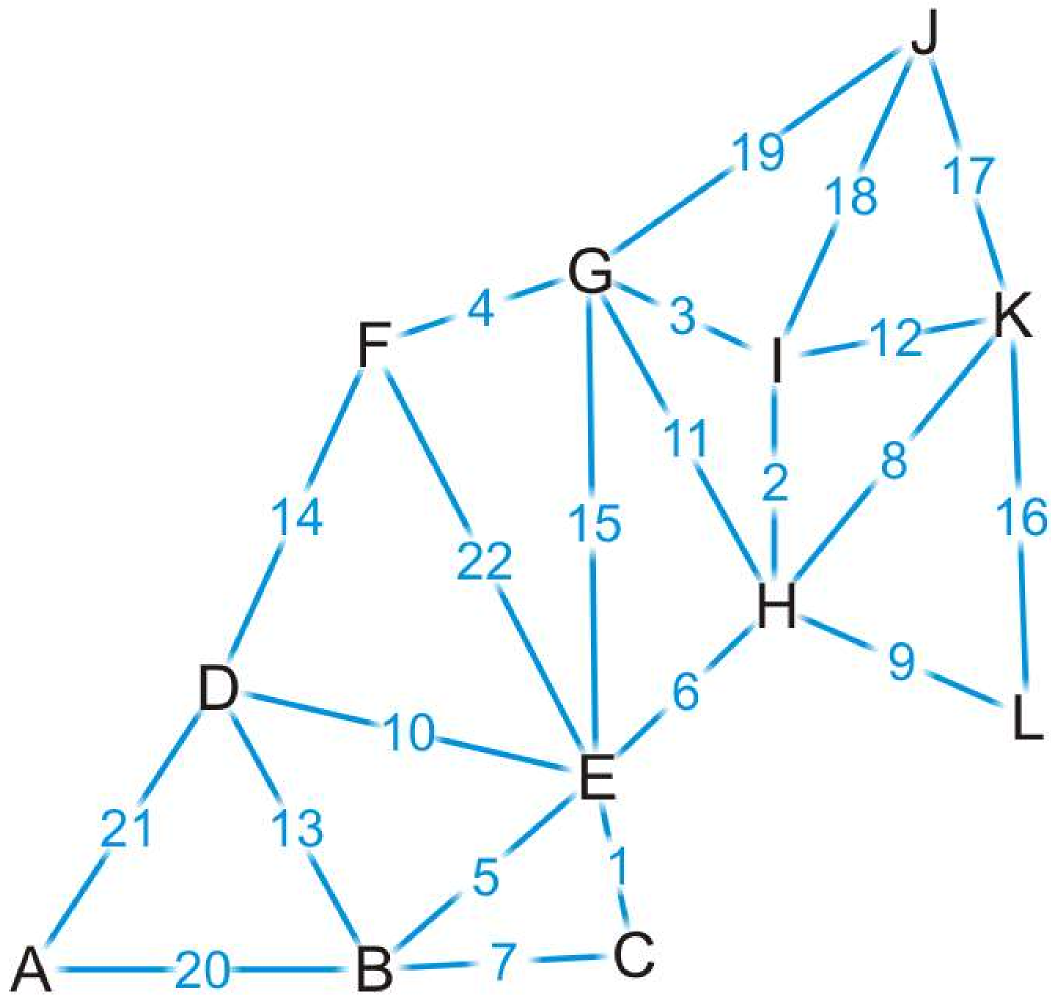
Dijkstra's Vs. Prim's Algorithm

```
Dijkstra's Algorithm
dist[s] = 0
p[s] = NULL
for all v ∈ V–{s}
   dist[v] = ∞
S = ∅
Q = V
while (Q not empty)
   u =  mindistance(Q)
   S = S∪{u}
   for all v ∈ neighbors[u] && v∈V-S
      if dist[u] + w(u, v) < dist[v]
         dist[v] = dist[u] + w(u, v)
         p[v] = u
```

```
Prim's Algorithm
Q = V[G];
for each u ∈ Q
   key[u] = ∞;
key[r] = 0;
p[r] = NULL;
while (Q not empty)
   u = ExtractMin(Q);
   for each v ∈ Adj[u]
      if (v ∈ Q and w(u,v) < key[v])
         p[v] = u;
         key[v] = w(u,v);
```

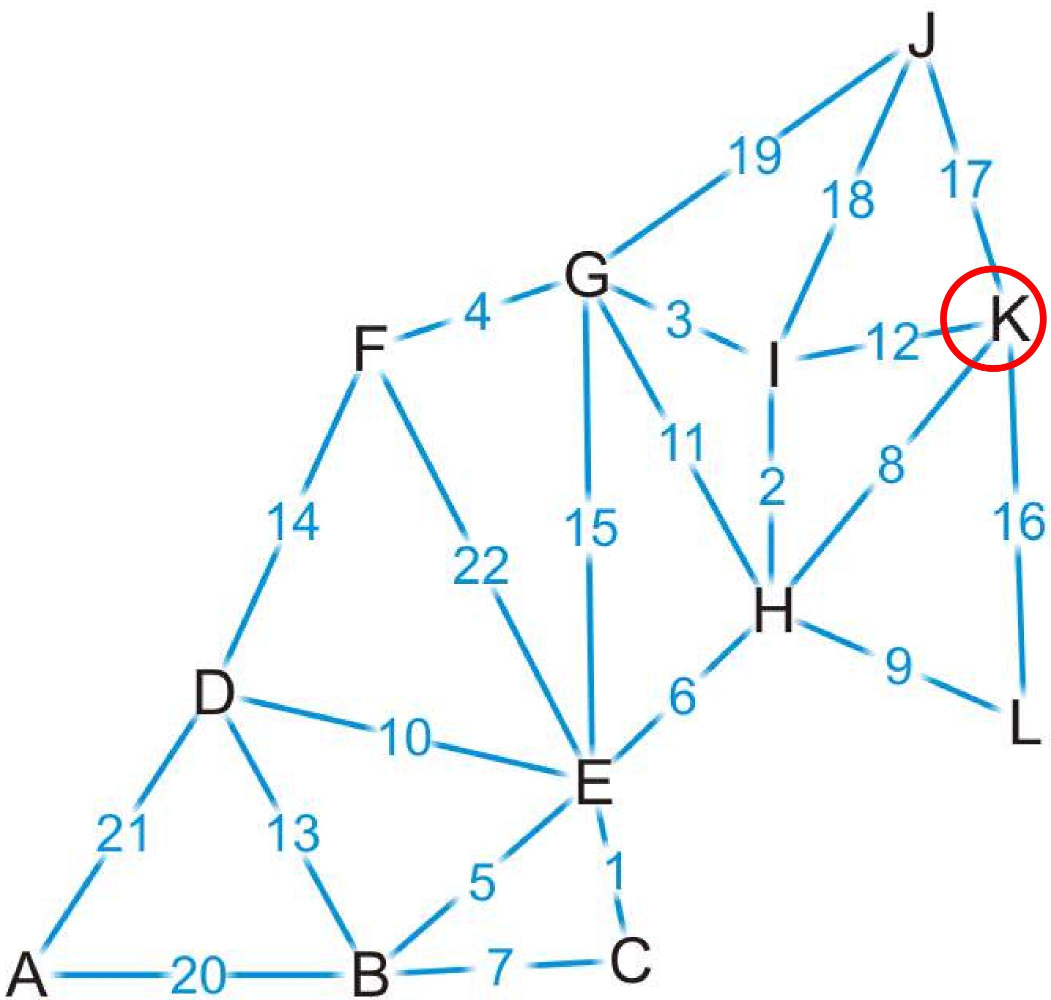
Dijkstra's Algorithm – Example

- Find the shortest path from K to every other vertex



Dijkstra's Algorithm – Example

- We visit vertex K

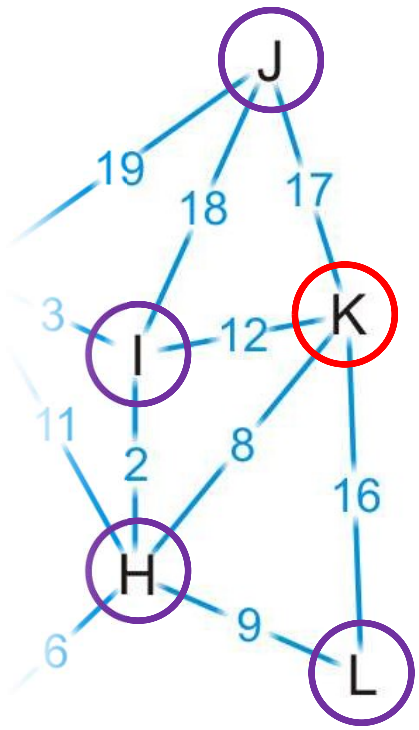


SPT

Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

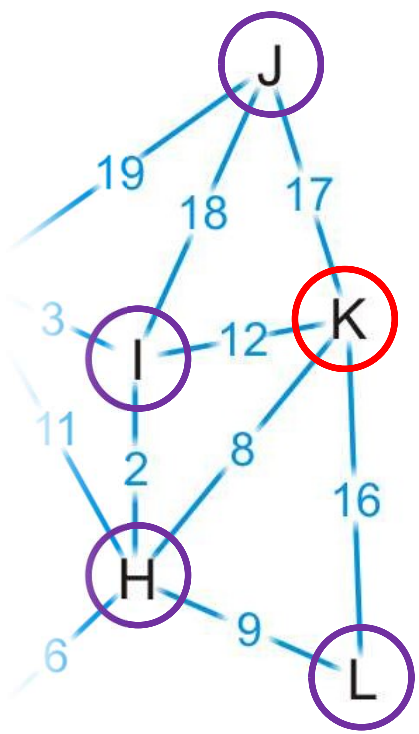
- Vertex K has four neighbors: H, I, J and L



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

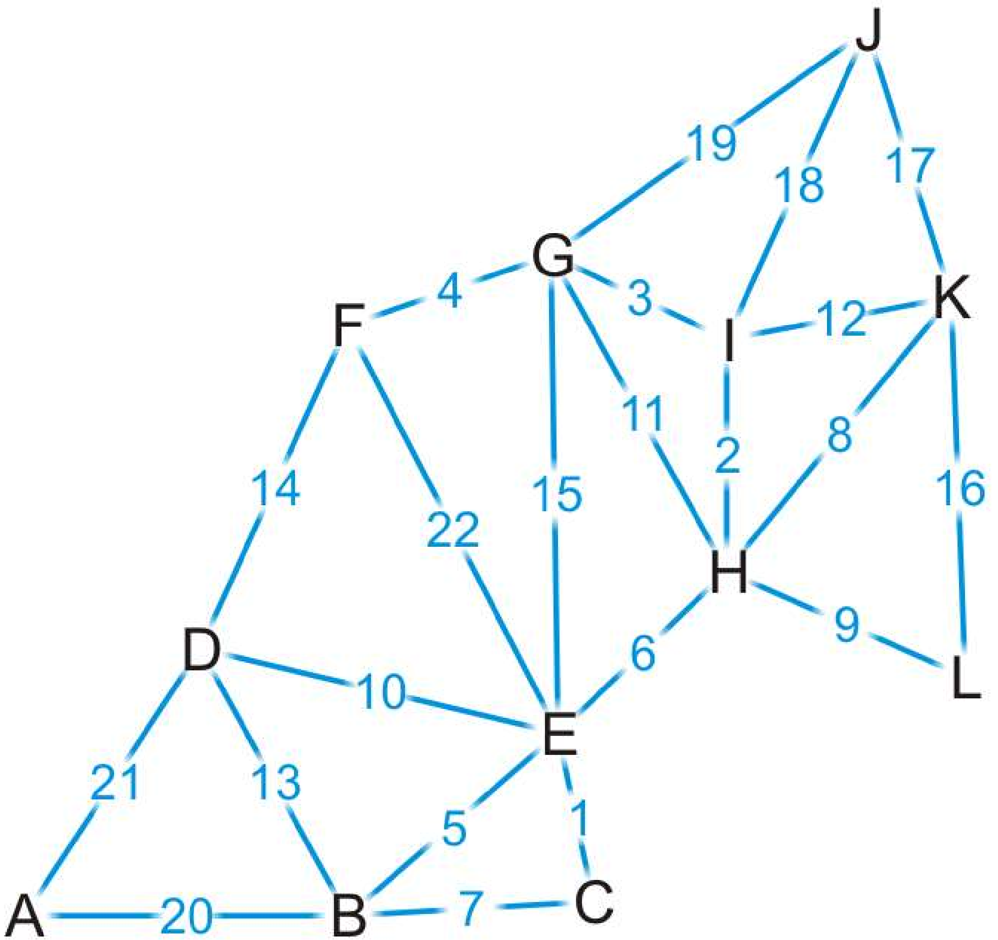
- We have now found at least one path to each of these vertices



Vertex	S	Distance	Parent
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**

Dijkstra's Algorithm – Example

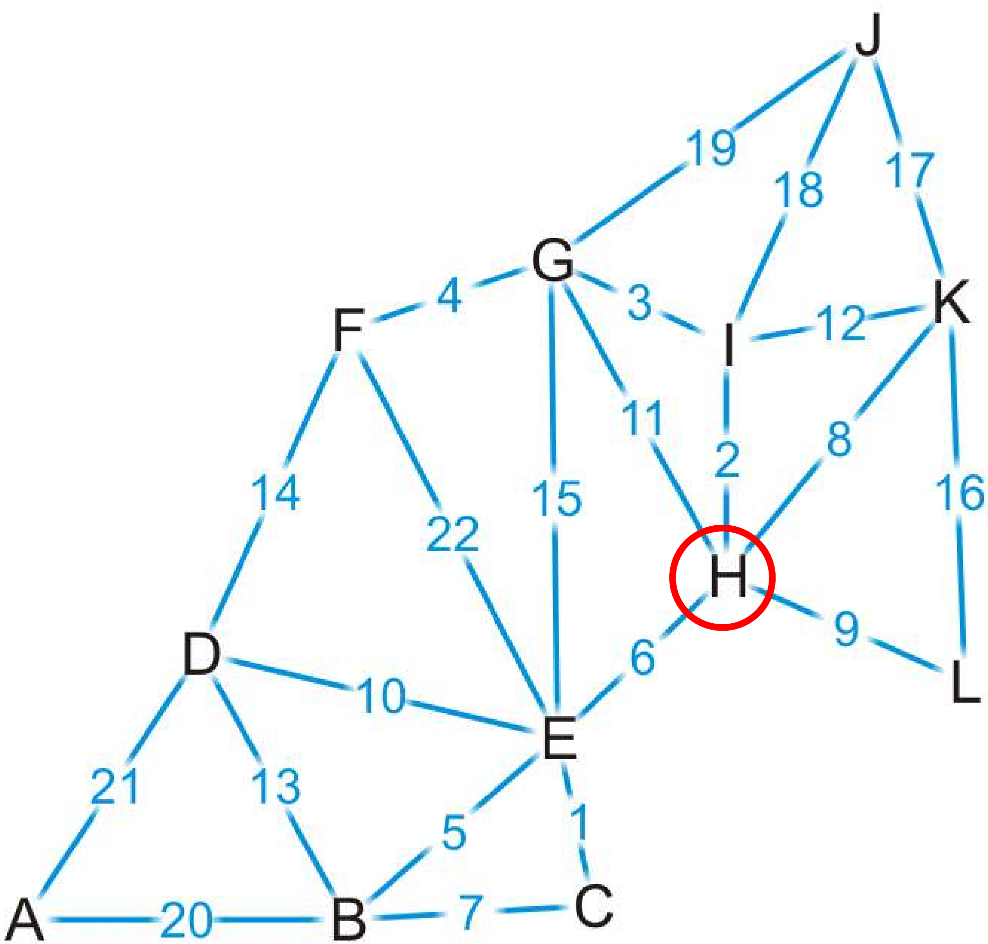
- We're finished with vertex K
  - To which vertex are we now guaranteed we have the shortest path?
    - ➢ `mindistance(Q)`



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

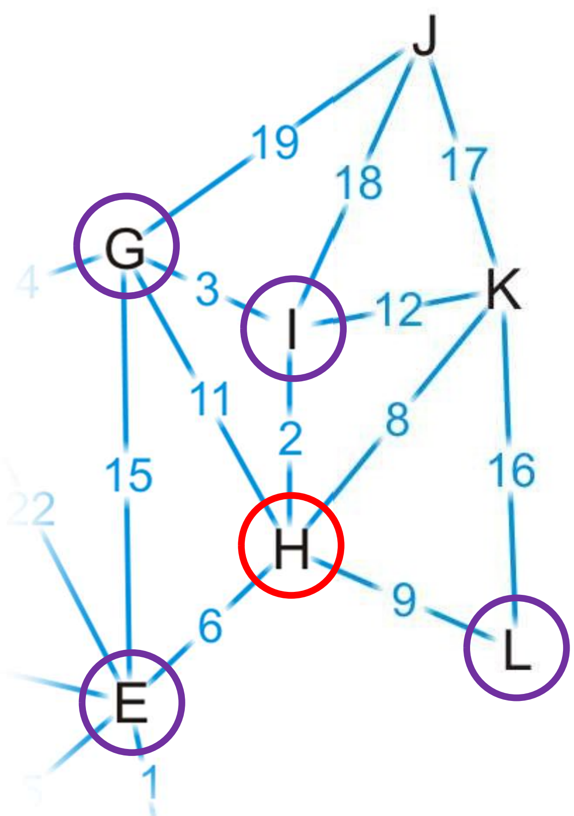
- We visit vertex H: the shortest path is (K, H) of length 8
  - Vertex H has four unvisited neighbors: E, G, I, L



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

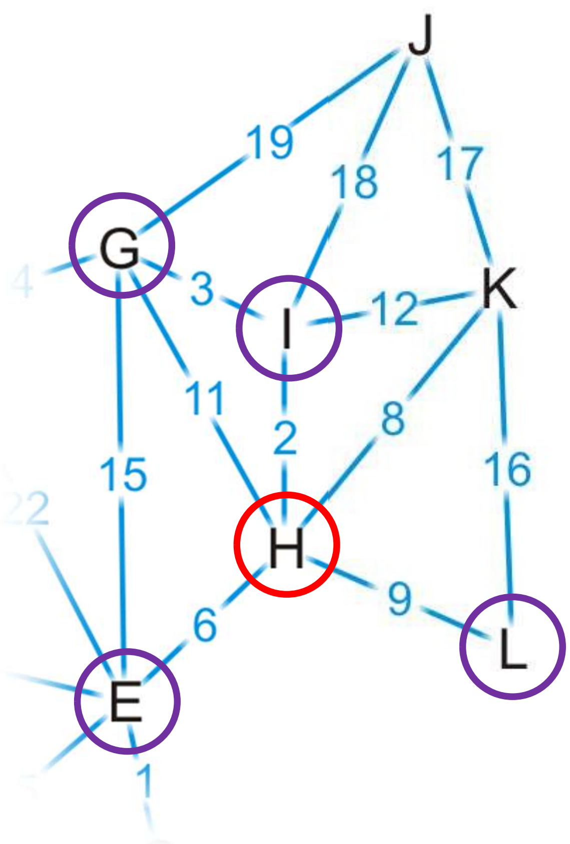
- Consider these paths:
  - (K, H, E) of length 8 + 6 = 14
  - (K, H, I) of length 8 + 2 = 10
  - (K, H, G) of length 8 + 11 = 19
  - (K, H, L) of length 8 + 9 = 17
- Which of these are shorter than any known path?



Vertex	S	Distance	Parent
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**

Dijkstra's Algorithm – Example

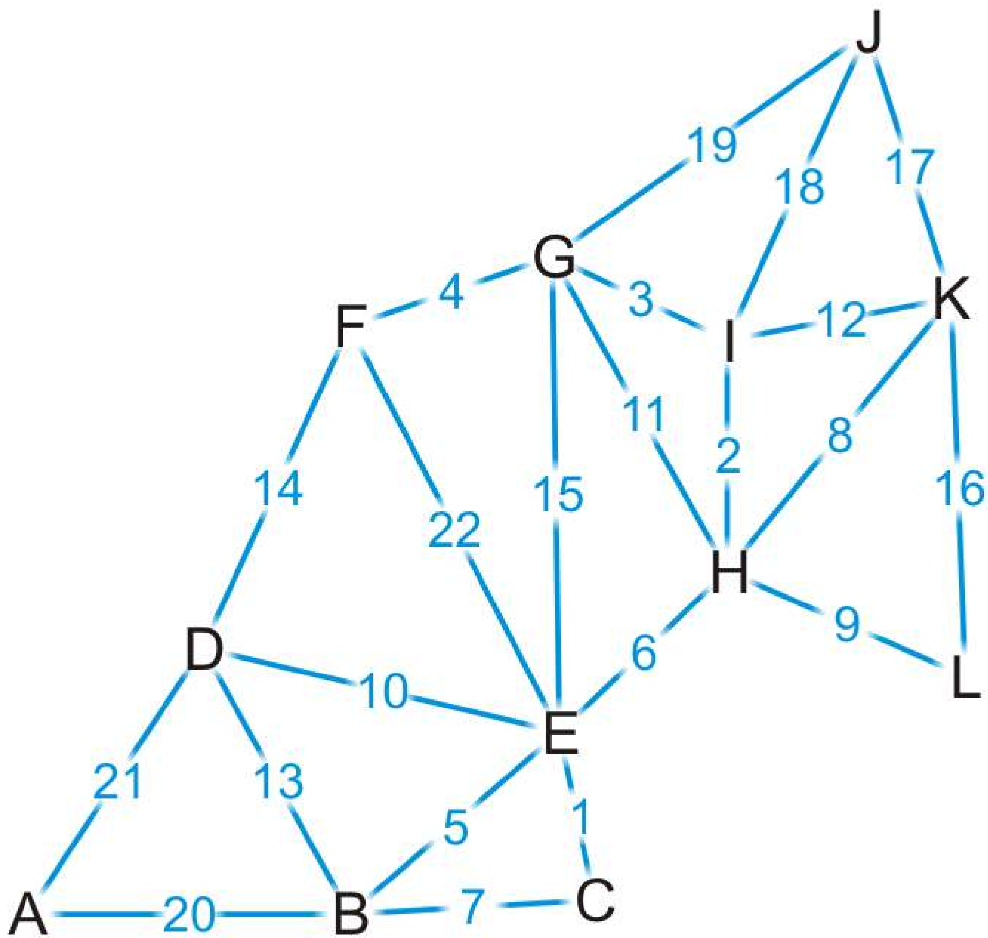
- We already have a shorter path (K, L), but we update the others



Vertex	S	Distance	Parent
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**

Dijkstra's Algorithm – Example

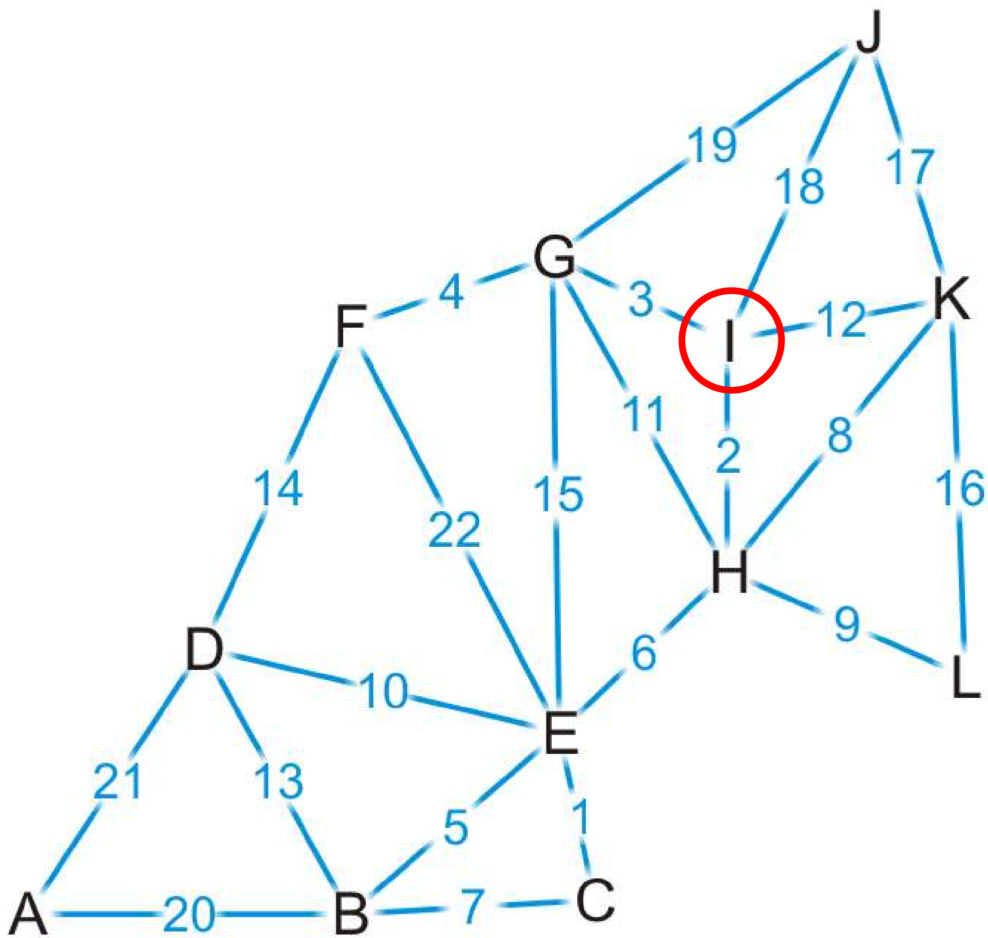
- We are finished with vertex H
  - Which vertex do we visit next?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

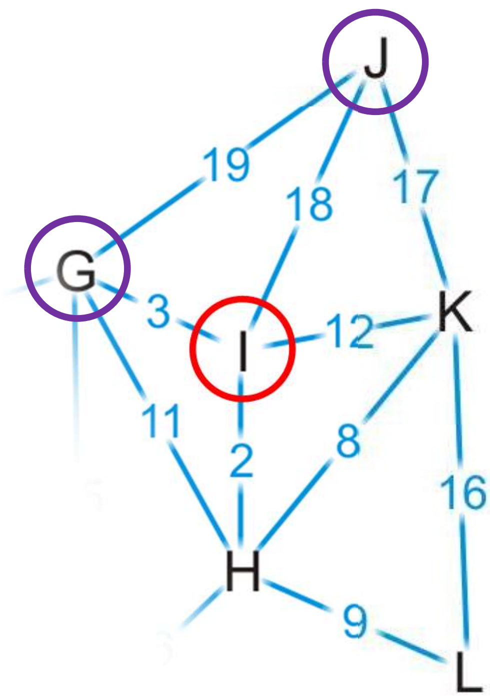
- The path (K, H, I) is the shortest path from K to I of length 10
  - Vertex I has two unvisited neighbors: G and J



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

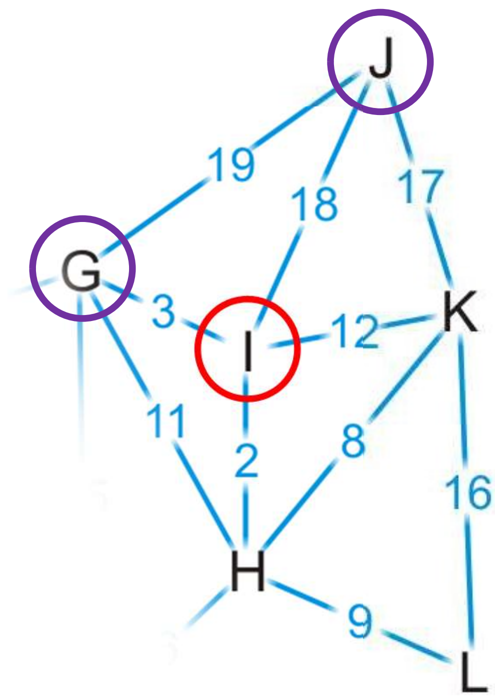
- Consider these paths:
  - (K, H, I, G) of length 10 + 3 = 13
  - (K, H, I, J) of length 10 + 18 = 28



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

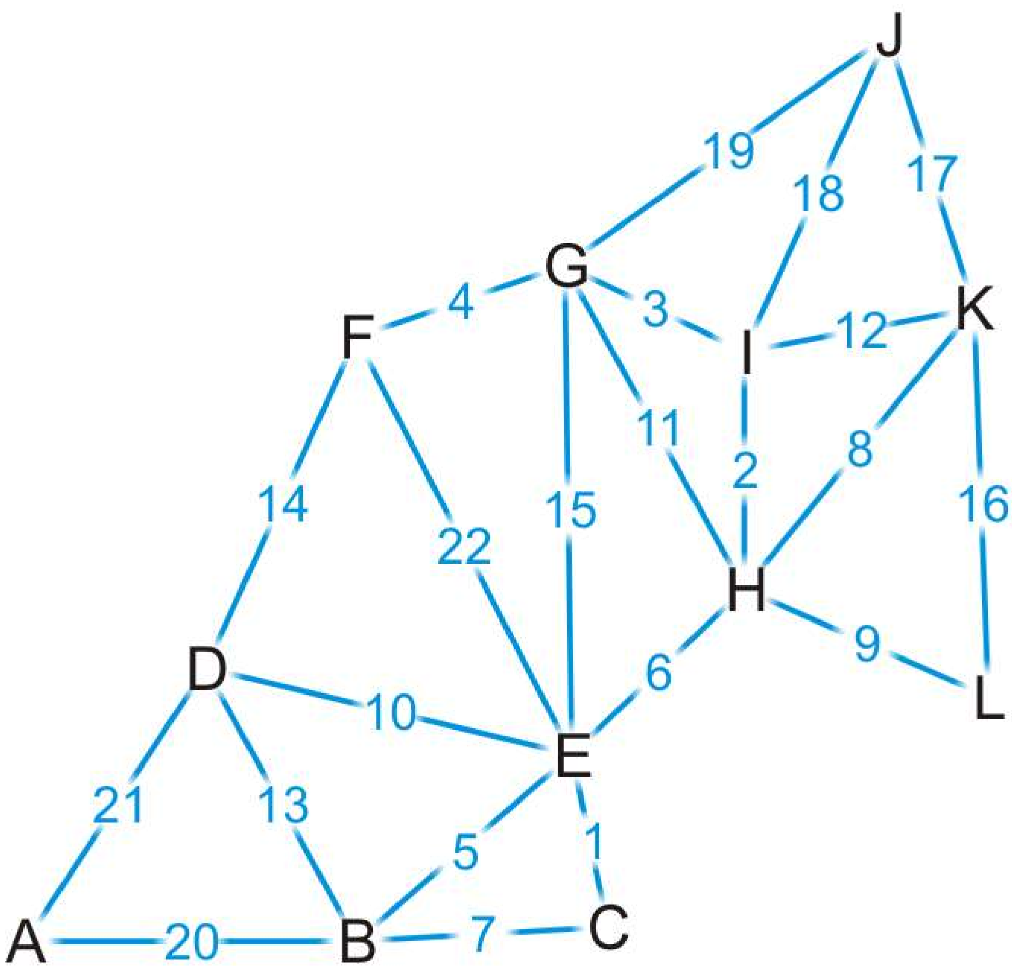
- We have discovered a shorter path to vertex G, but (K, J) is still the shortest known path to vertex J



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

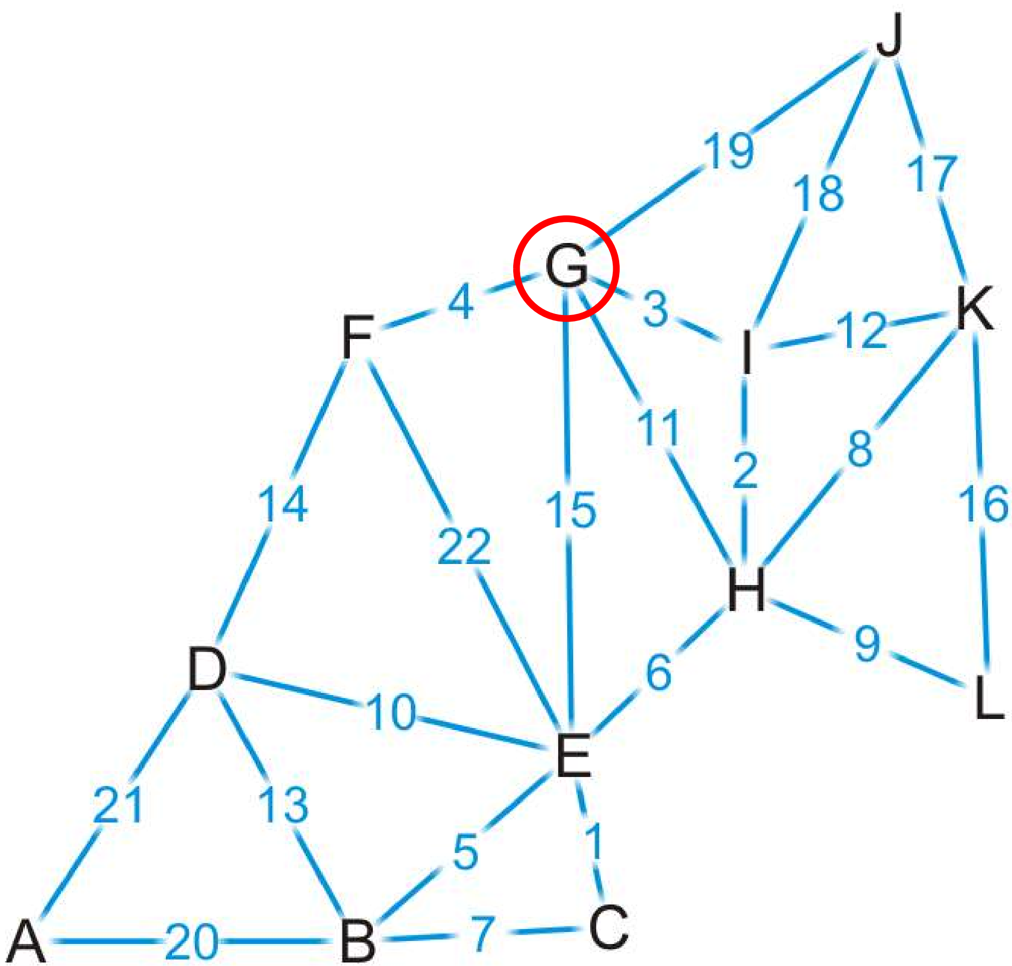
- Which vertex can we visit next?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

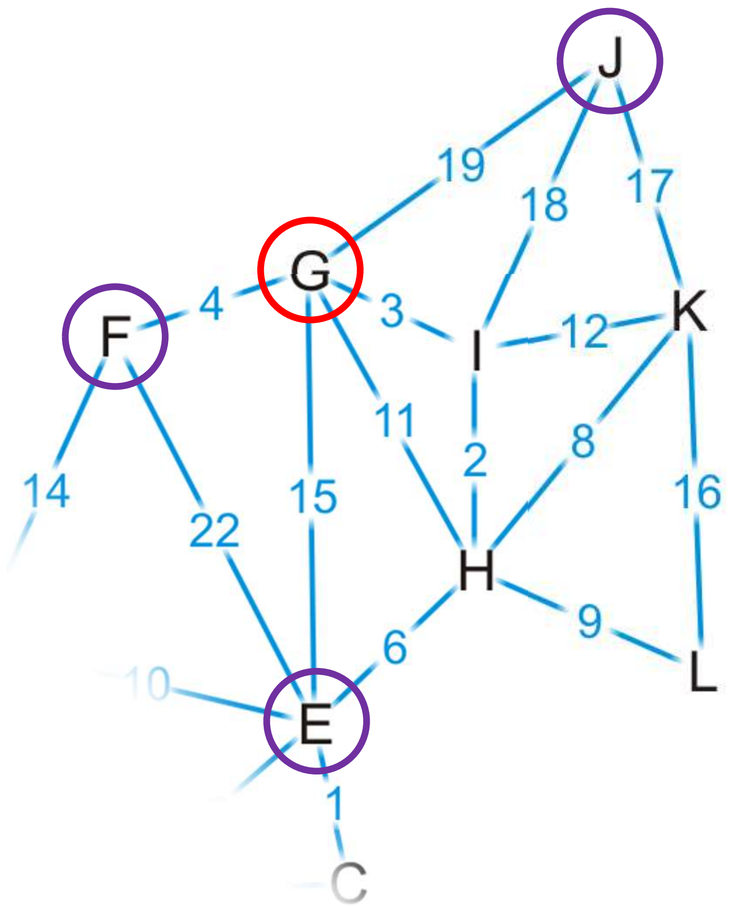
- The path (K, H, I, G) is the shortest path from K to G of length 13
  - Vertex G has three unvisited neighbors: E, F and J



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

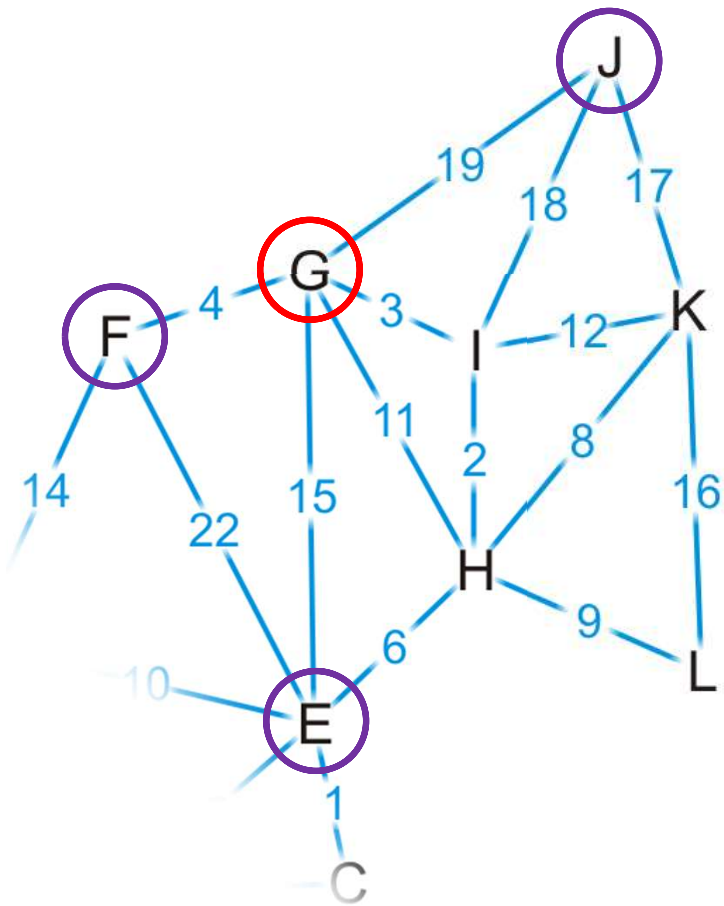
- Consider these paths:
  - (K, H, I, G, E) of length 13 + 15 = 28
  - (K, H, I, G, F) of length 13 + 4 = 17
  - (K, H, I, G, J) of length 13 + 19 = 32



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

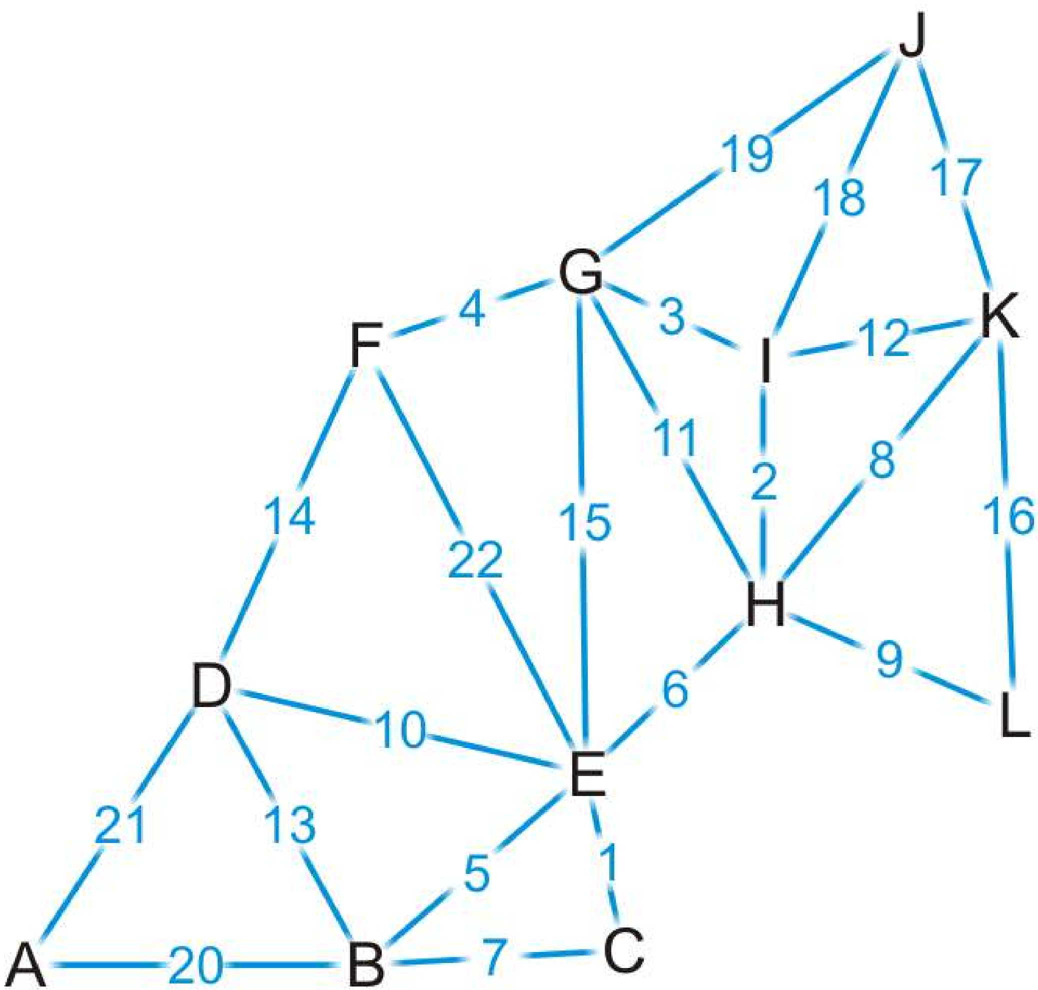
- We have now found a path to vertex F



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

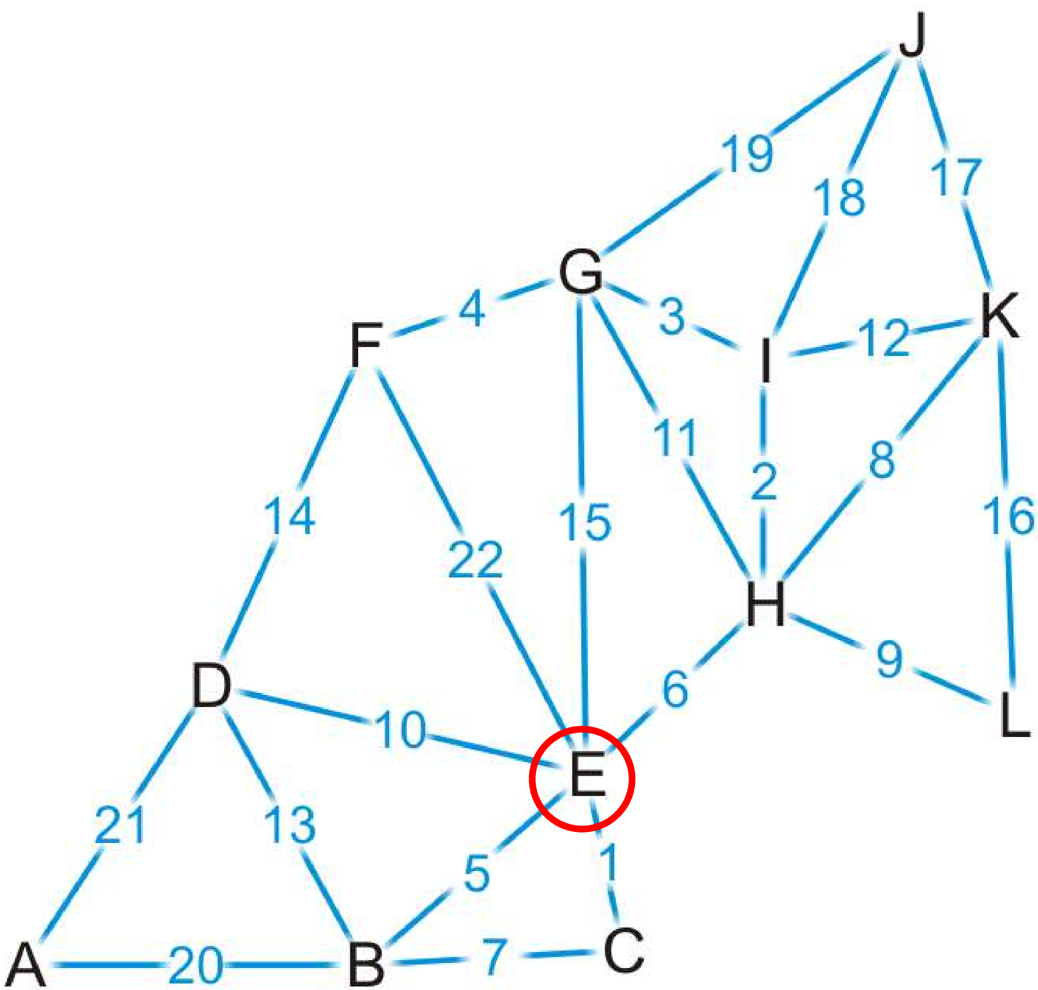
- Where do we visit next?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

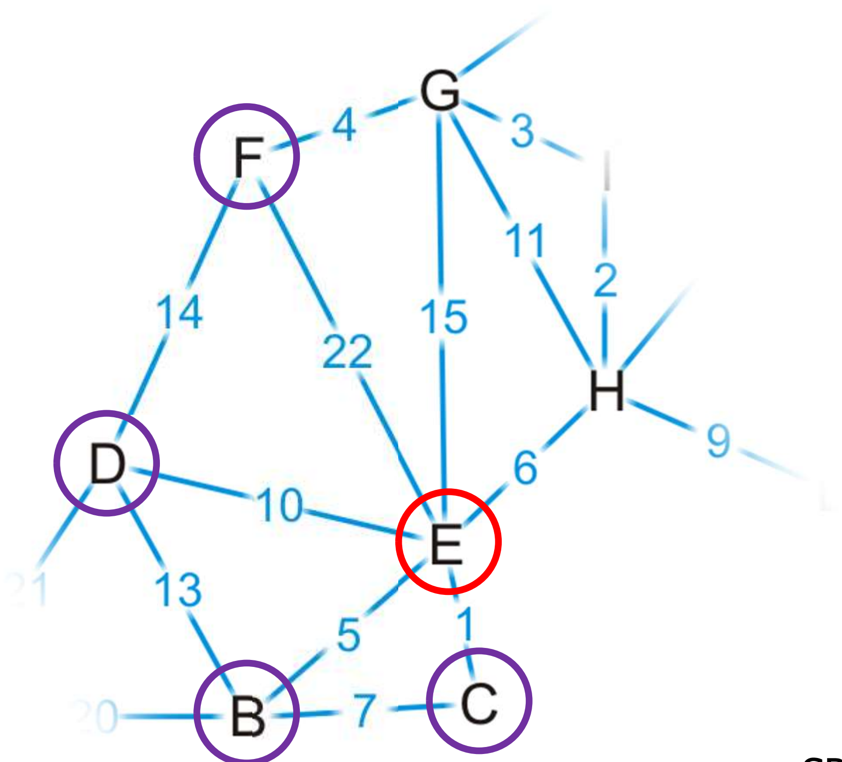
- The path (K, H, E) is the shortest path from K to E of length 14
  - Vertex G has four unvisited neighbors: B, C, D and F



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

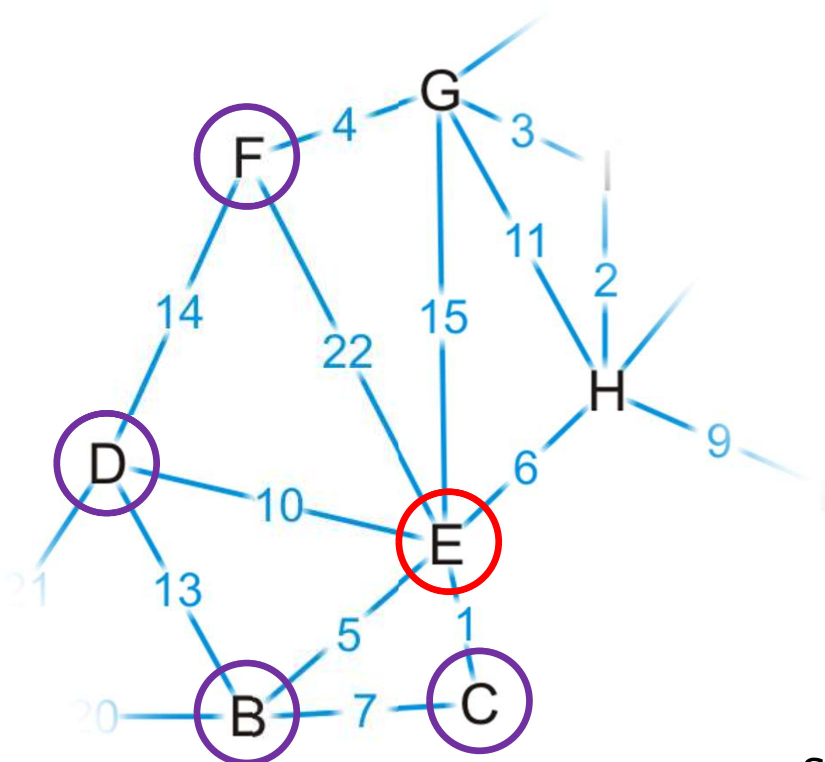
- The path (K, H, E) is the shortest path from K to E of length 14
  - Vertex G has four unvisited neighbors: B, C, D and F



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

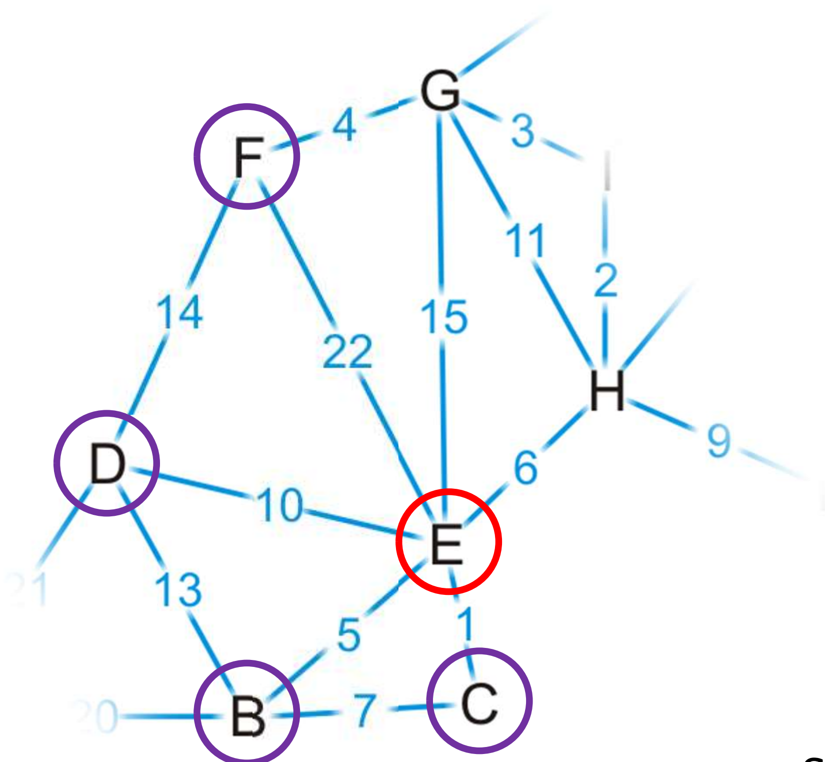
- Consider these paths:
  - (K, H, E, B) of length 14 + 5 = 19
  - (K, H, E, C) of length 14 + 1 = 15
  - (K, H, E, D) of length 14 + 10 = 24
  - (K, H, E, F) of length 14 + 22 = 36



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

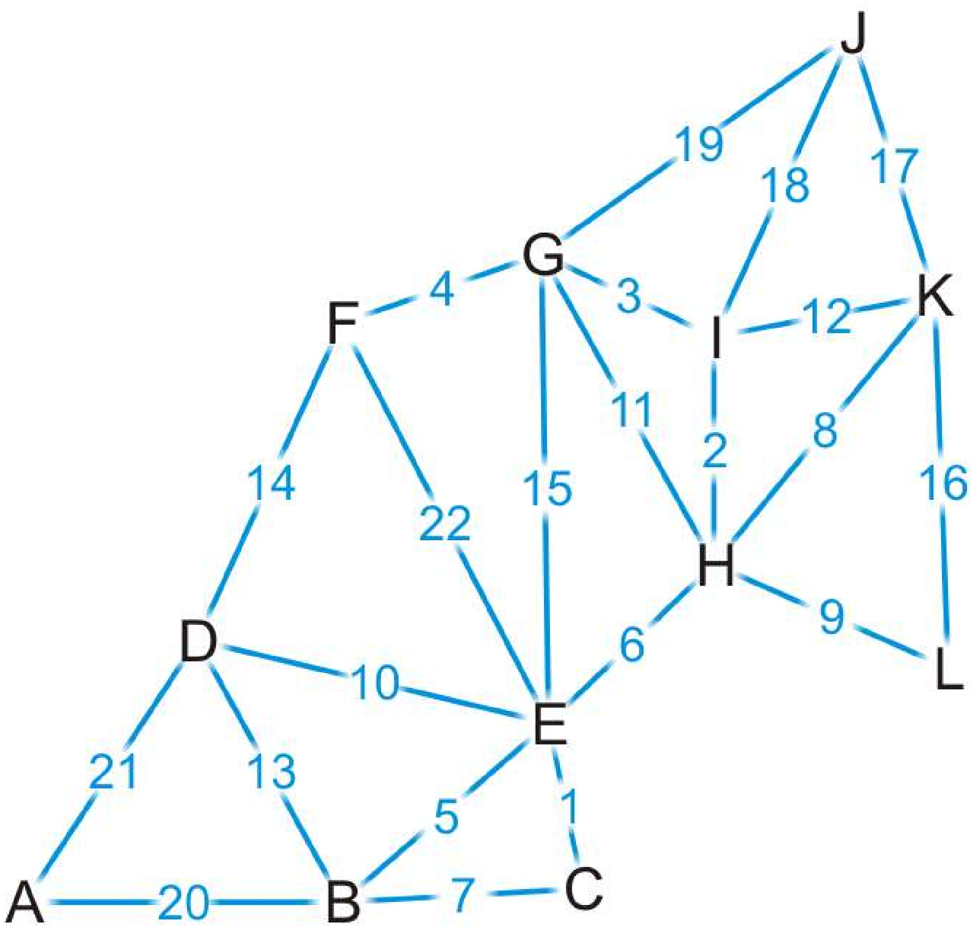
- We've discovered paths to vertices B, C, D



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

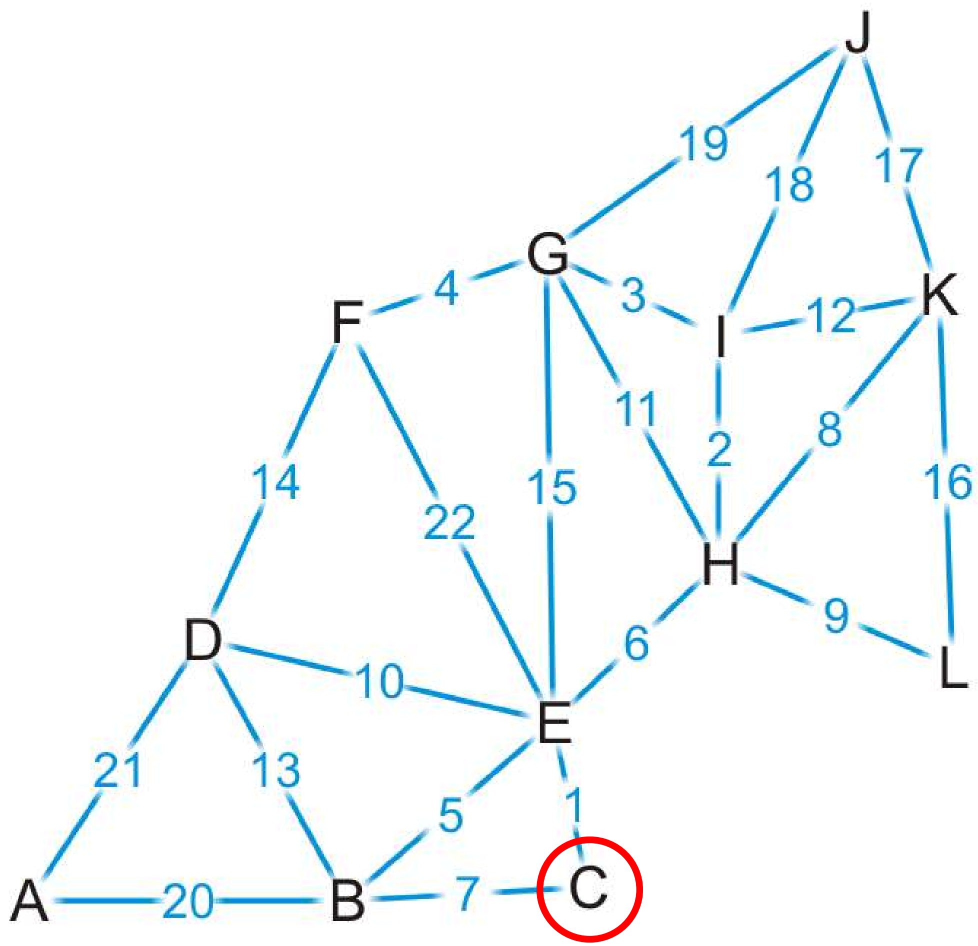
- Which vertex is next?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

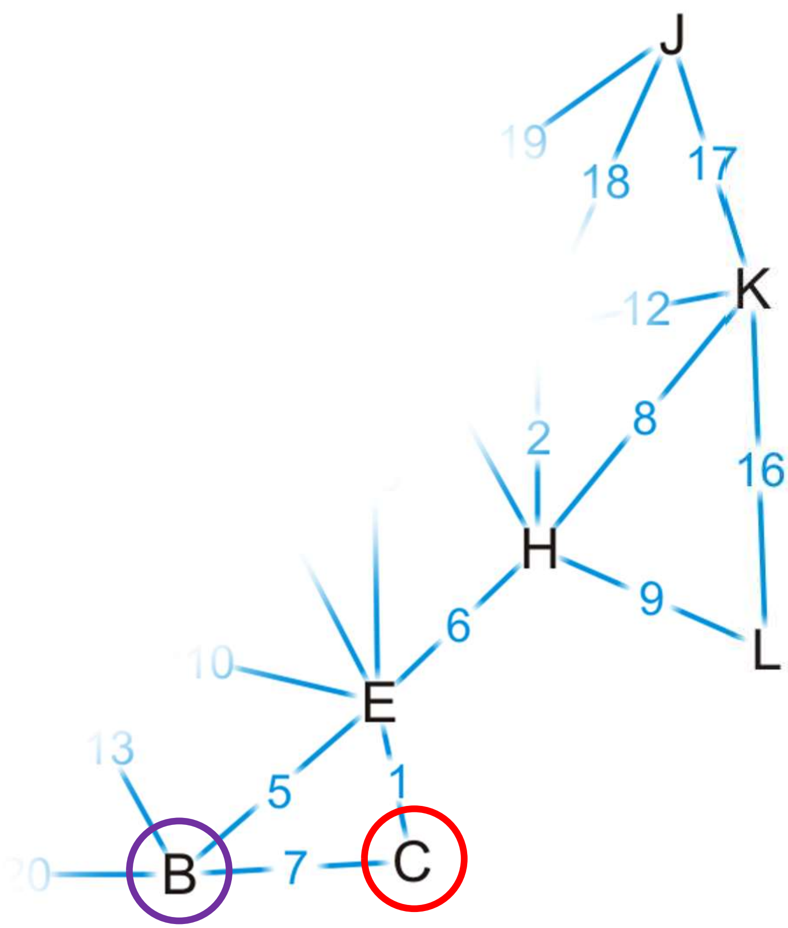
- We’ve found that the path (K, H, E, C) of length 15 is the shortest path from K to C
  - Vertex C has one unvisited neighbor, B



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

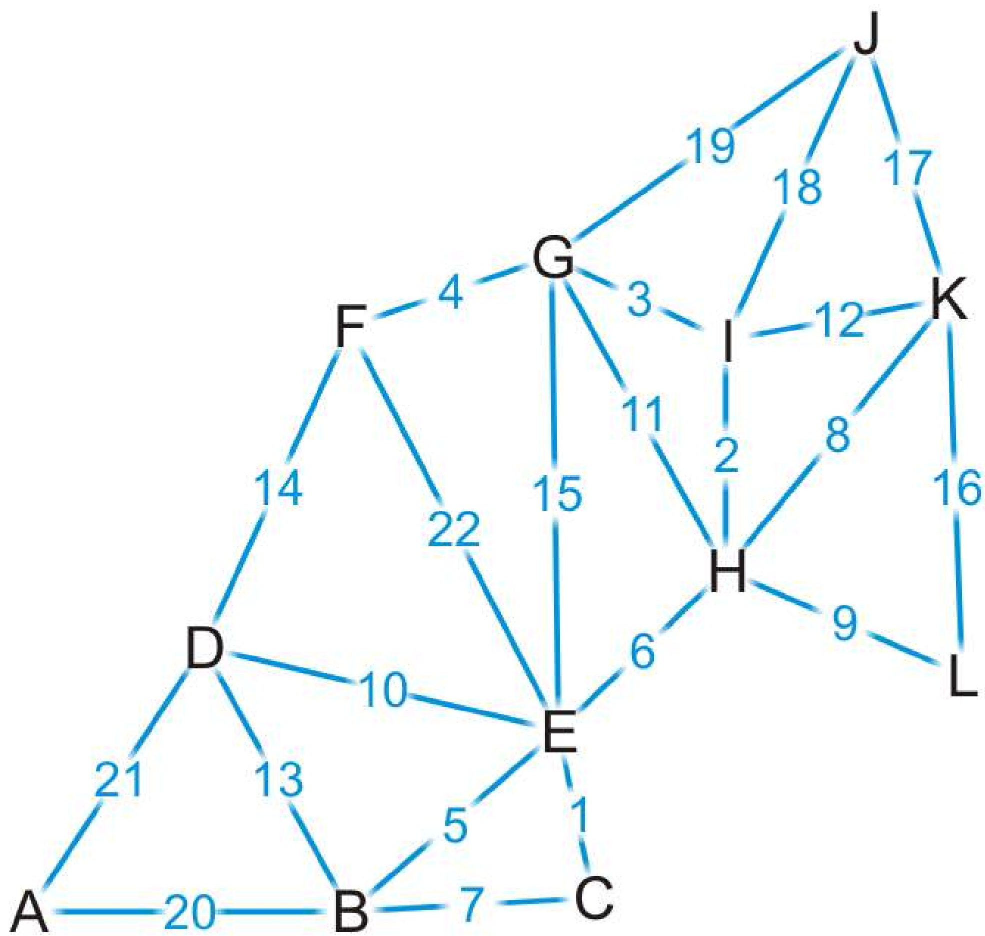
- The path (K, H, E, C, B) is of length 15 + 7 = 22
  - We have already discovered a shorter path through vertex E



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

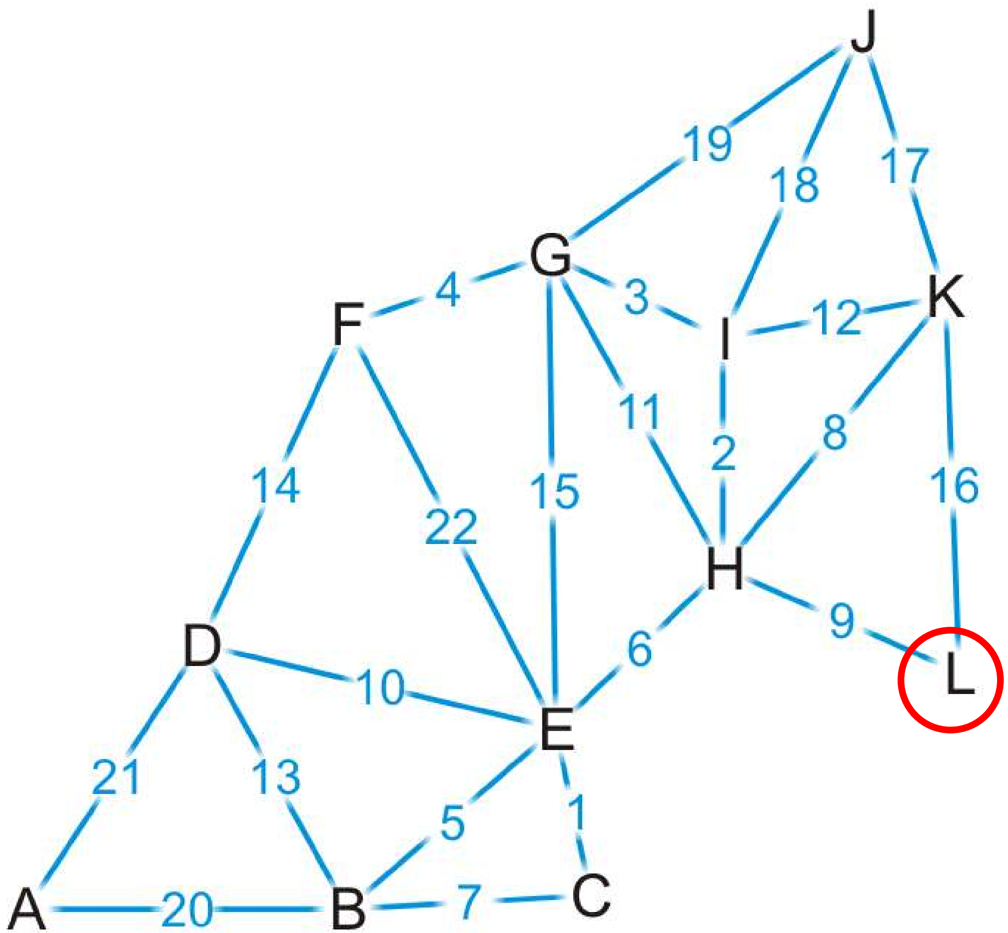
- Where to next?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

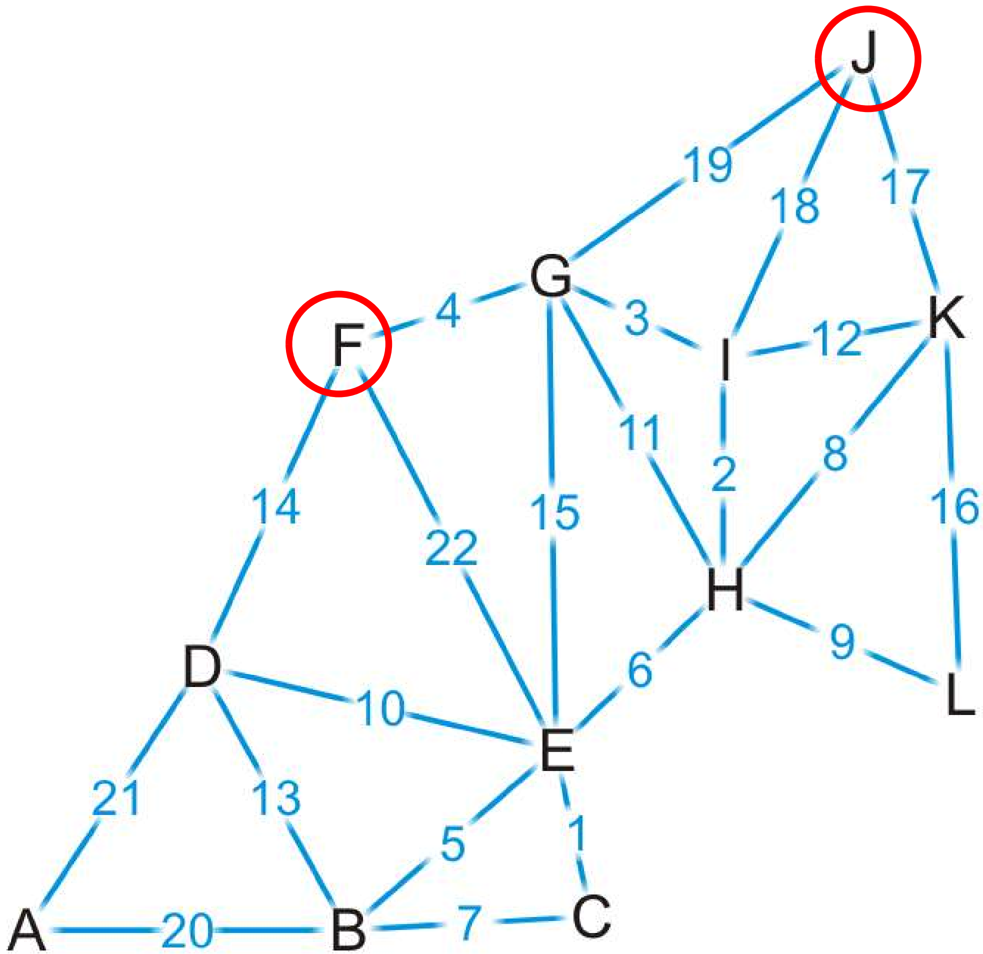
- We now know that (K, L) is the shortest path between these two points
  - Vertex L has no unvisited neighbors



Vertex	S	Distance	Parent
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**

Dijkstra's Algorithm – Example

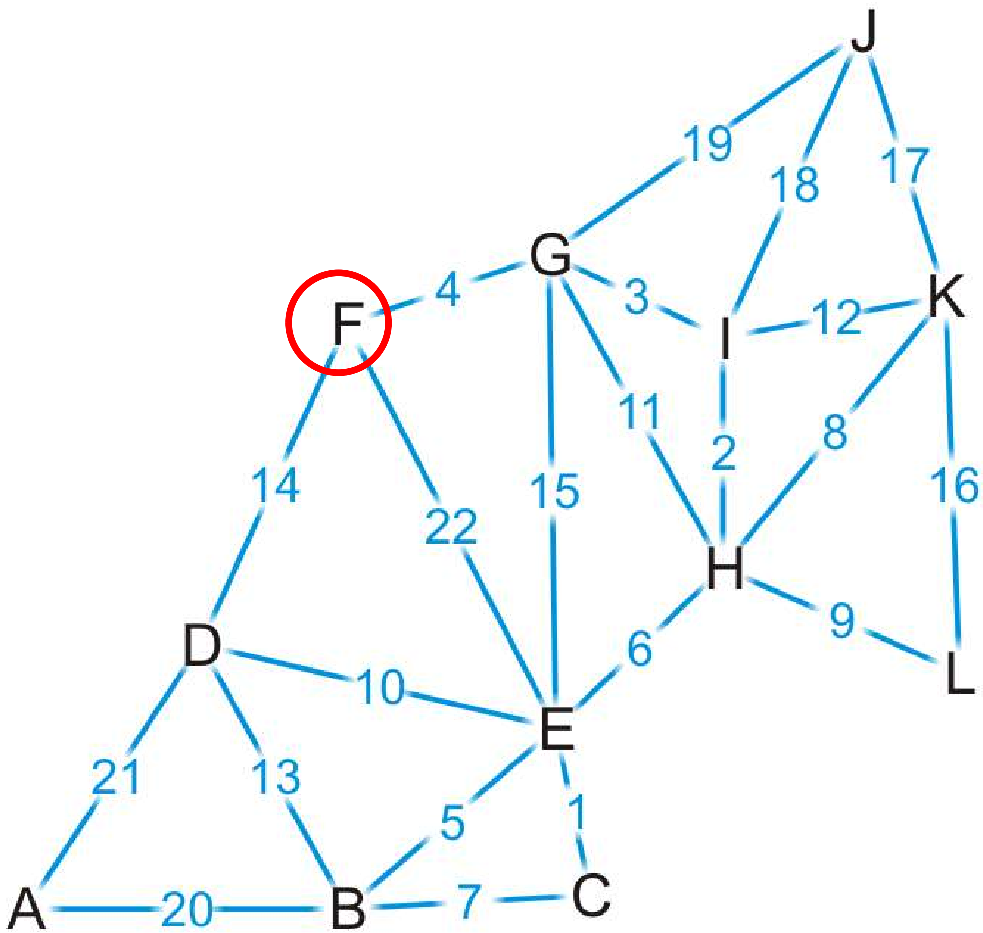
- Where to next?
  - Does it matter if we visit vertex F first or vertex J first?



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

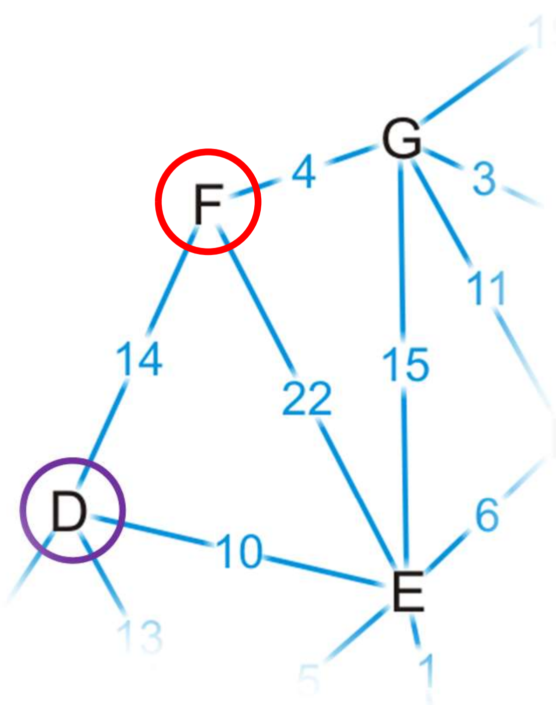
- Let's visit vertex F first
  - It has one unvisited neighbor, vertex D



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

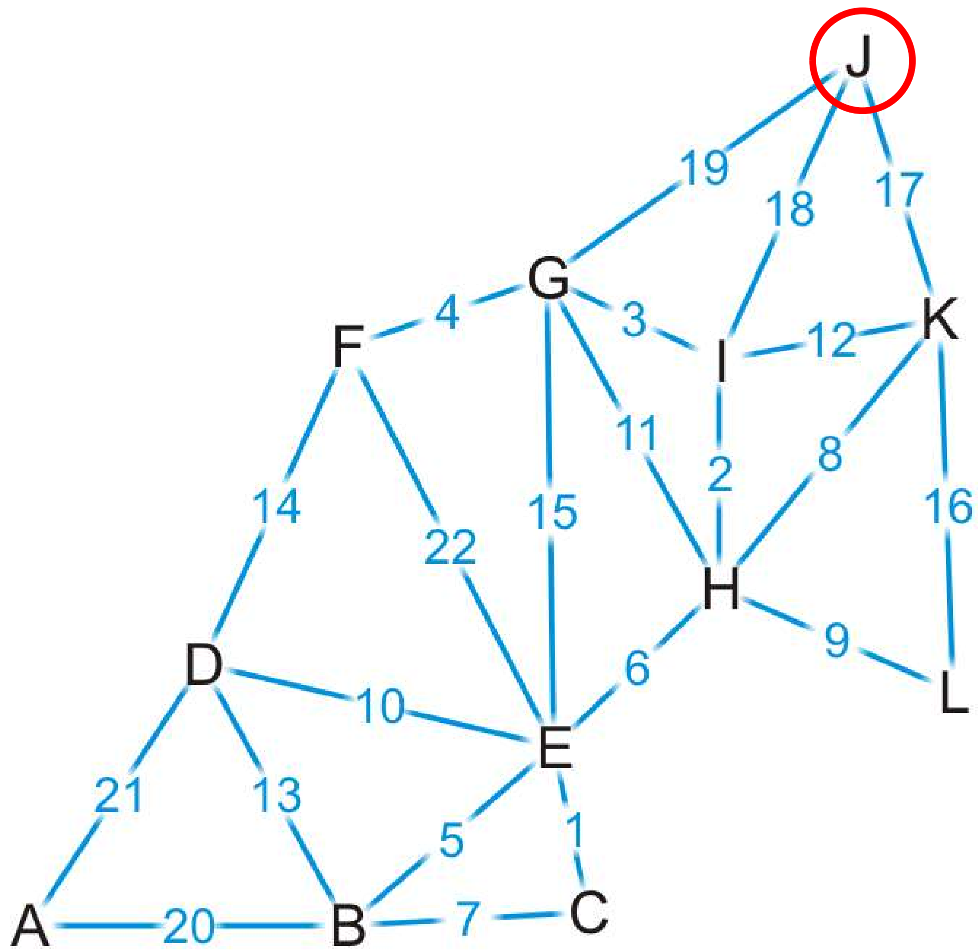
- The path (K, H, I, G, F, D) is of length 17 + 14 = 31
  - This is longer than the path we've already discovered



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

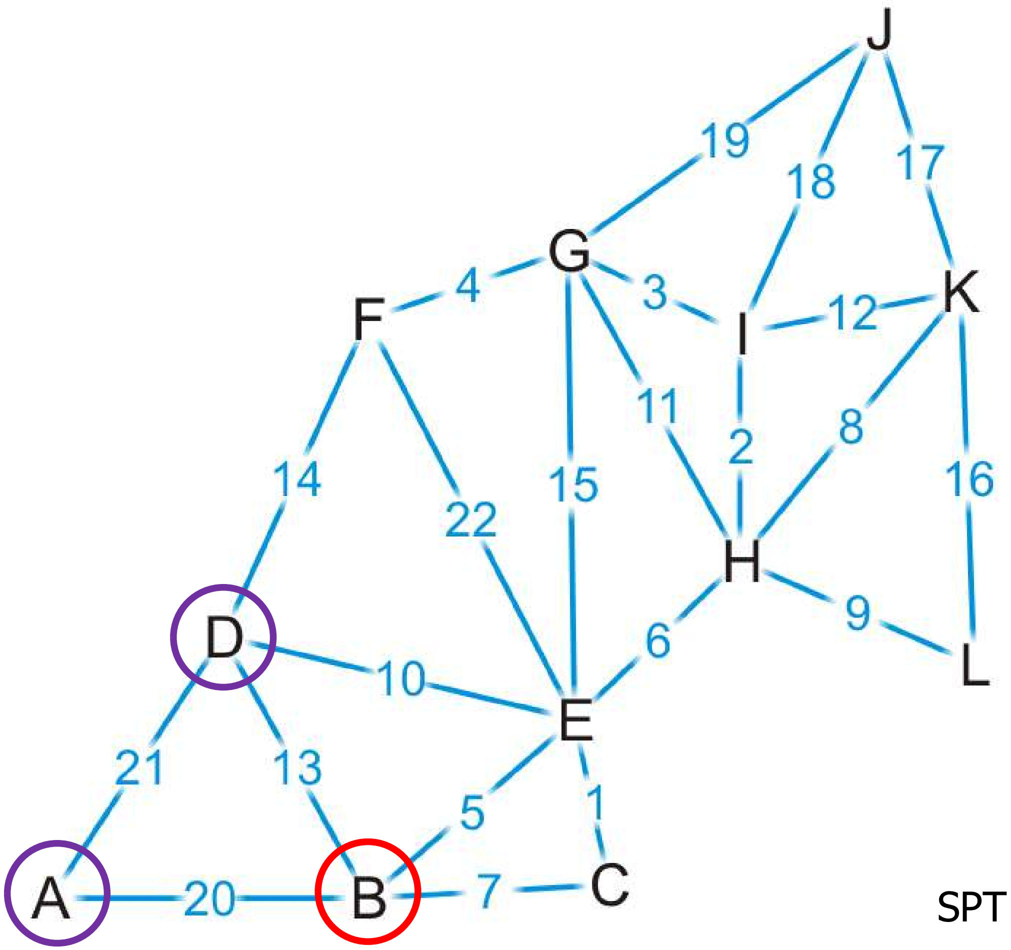
- Now we visit vertex J
  - It has no unvisited neighbors



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

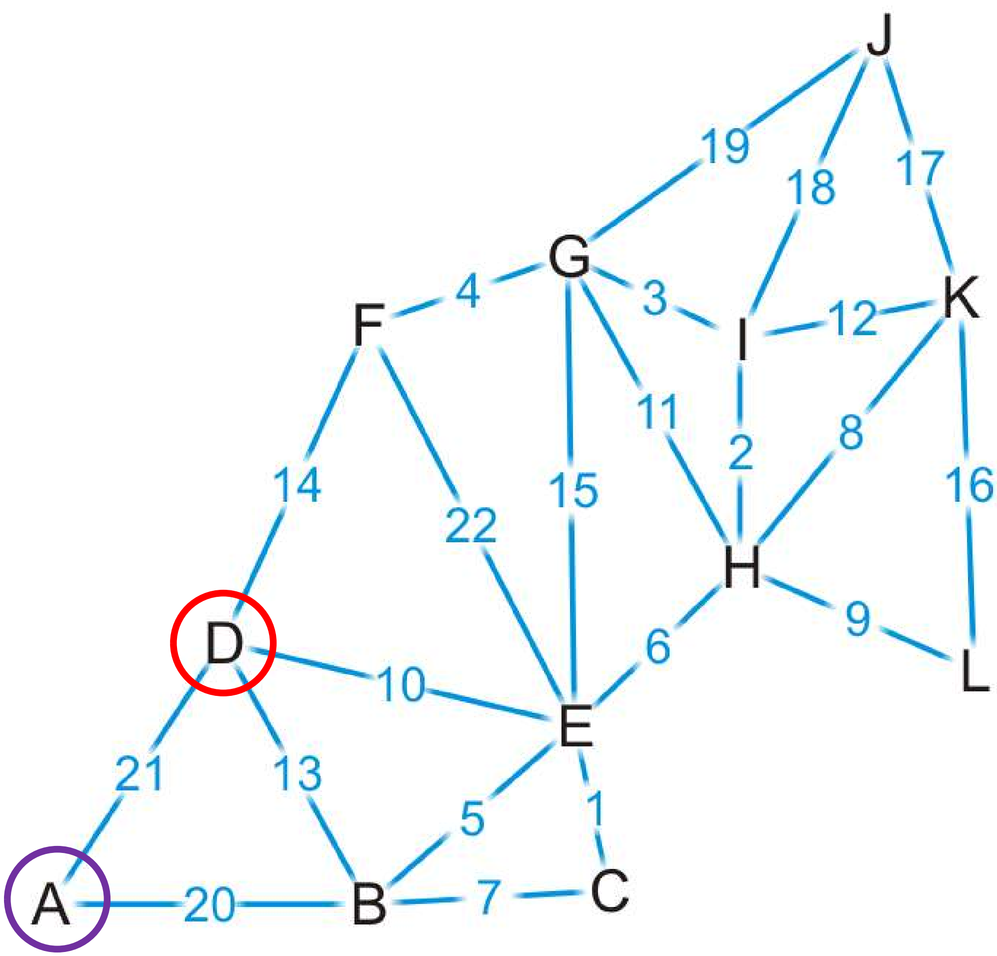
- Next we visit vertex B, which has two unvisited neighbors:
  - (K, H, E, B, A) of length 19 + 20 = 39
  - (K, H, E, B, D) of length 19 + 13 = 32
- We update the path length to A



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

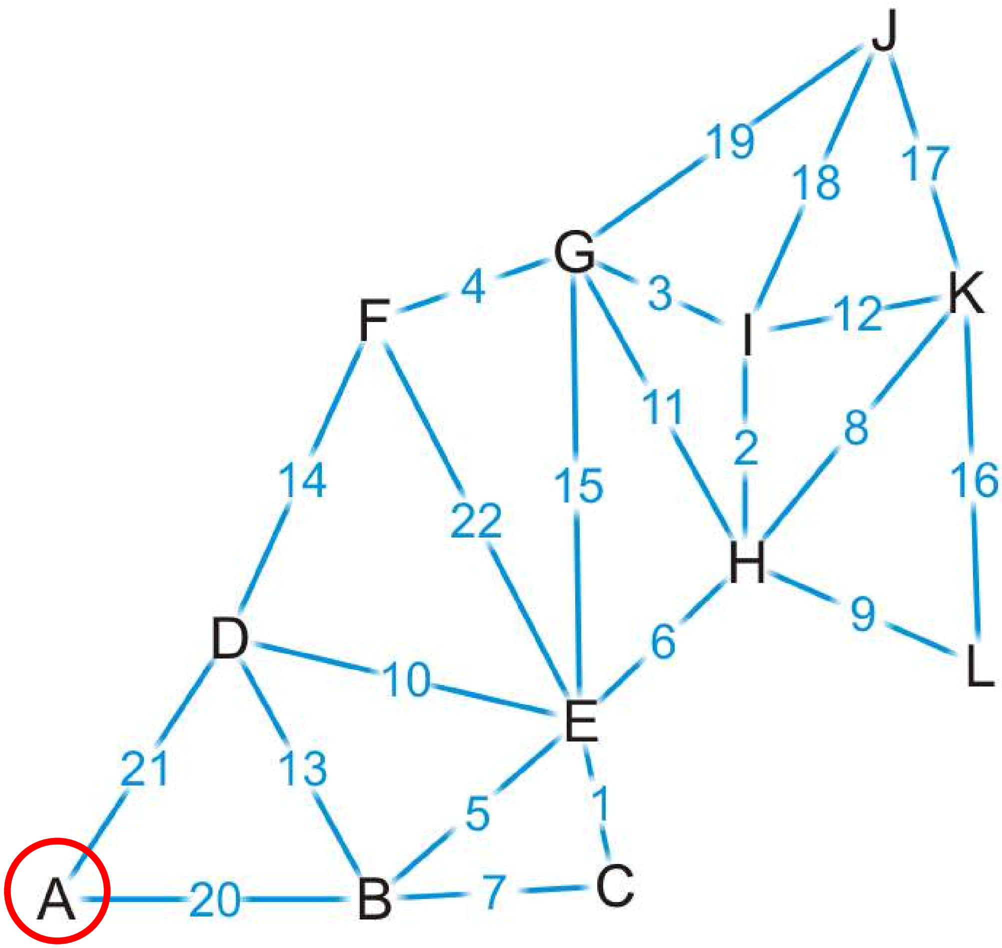
- Next we visit vertex D
  - The path (K, H, E, D, A) is of length 24 + 21 = 45
  - We don't update A



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

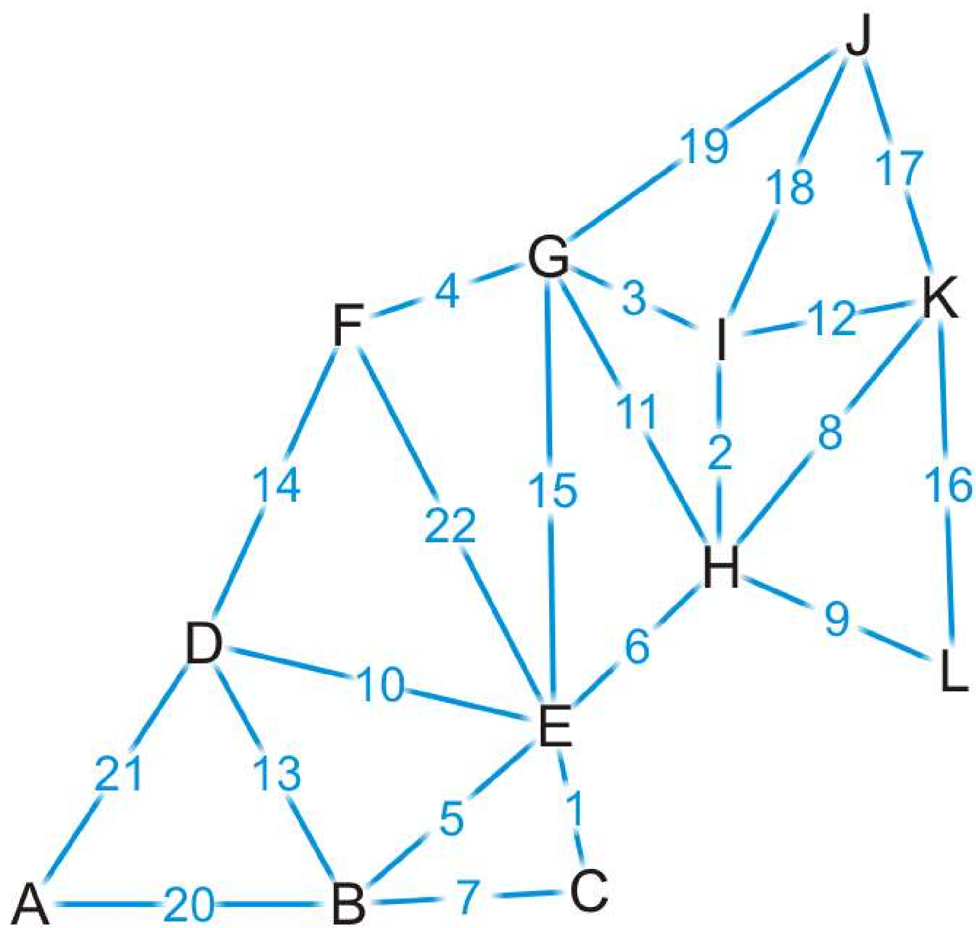
- Finally, we visit vertex A
  - It has no unvisited neighbors and there are no unvisited vertices left
  - We are done



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

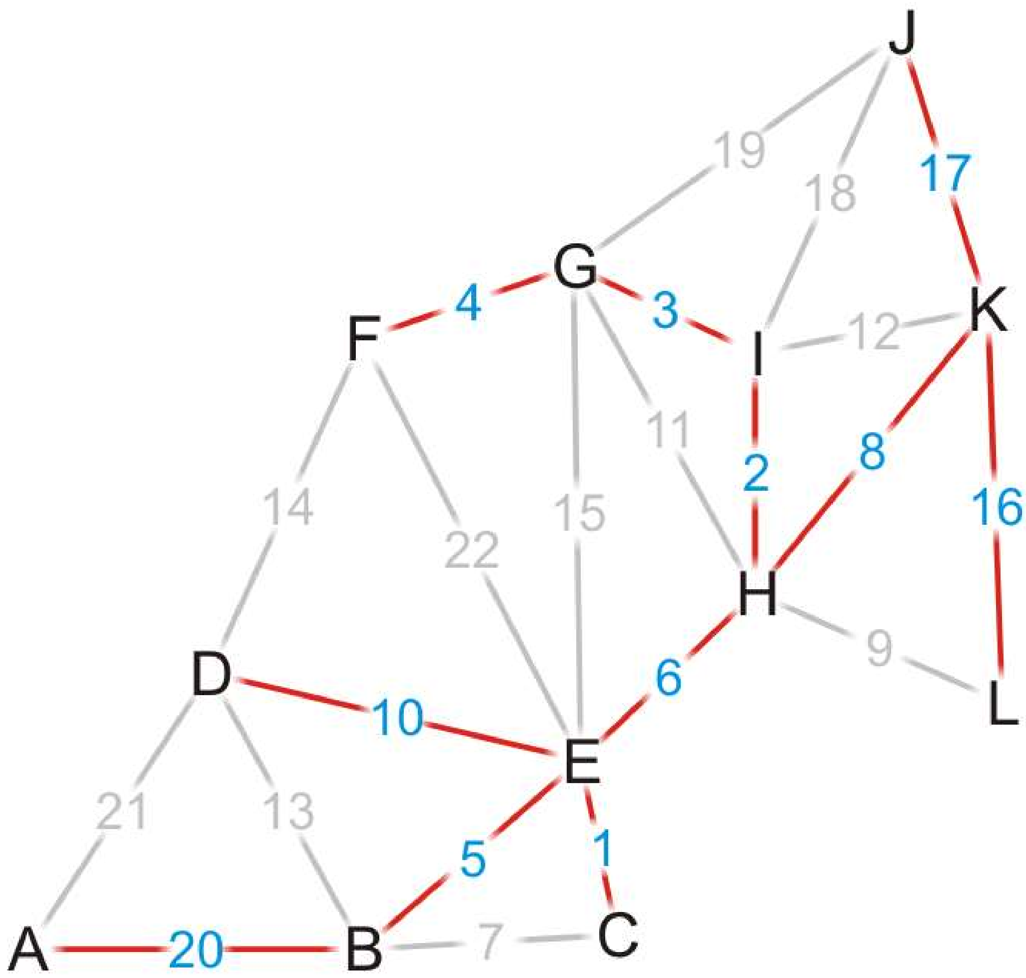
- Thus, we have found the shortest path from vertex K to each of the other vertices



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

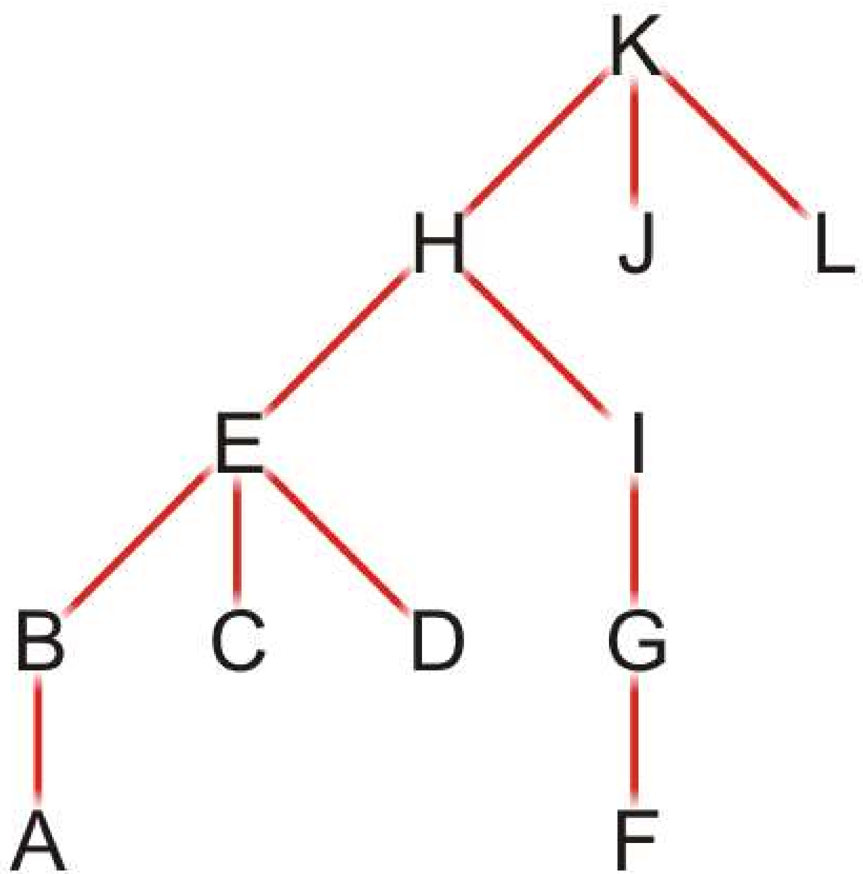
- Using the previous pointers, we can reconstruct the paths



Vertex	S	Distance	Parent
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

Dijkstra's Algorithm – Example

- The table defines a rooted parental tree
  - The source vertex K is at the root
  - The previous pointer is the parent of the vertex in the tree

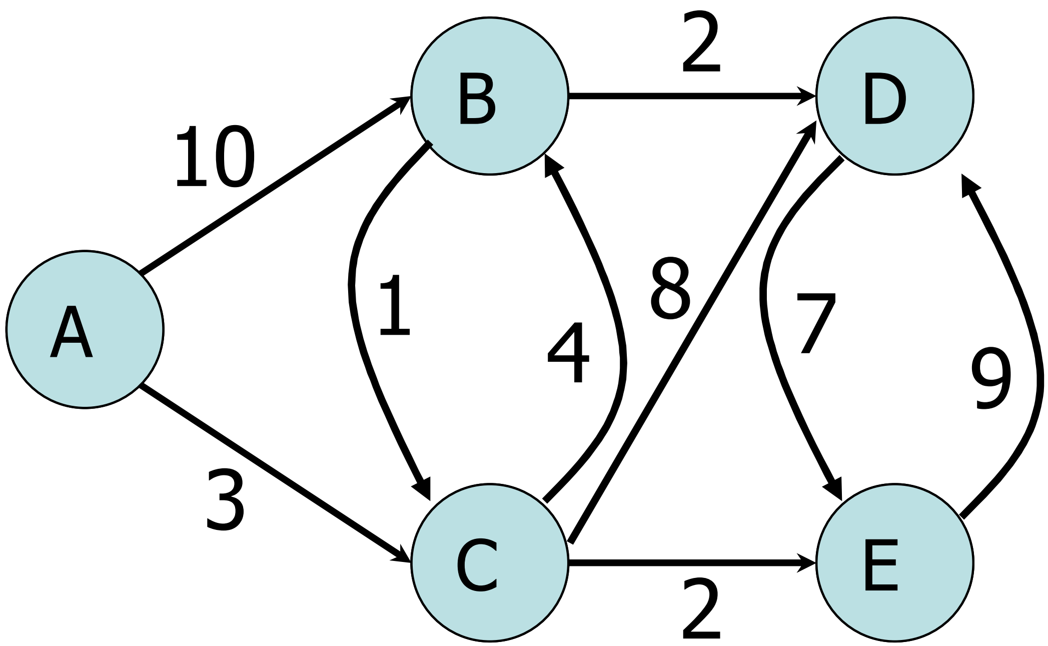


Vertex	Previous
A	B
B	E
C	E
D	E
E	H
F	G
G	I
H	K
I	H
J	K
K	Ø
L	K

Comments on Dijkstra's Algorithm

- 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
- To find the shortest path between vertices v_j and v_k?
  - Apply the same algorithm, but stop when visiting vertex v_k
- Does the algorithm change if graph is directed?
  - No

Dijkstra's Algorithm – Example



Q: A B C D E

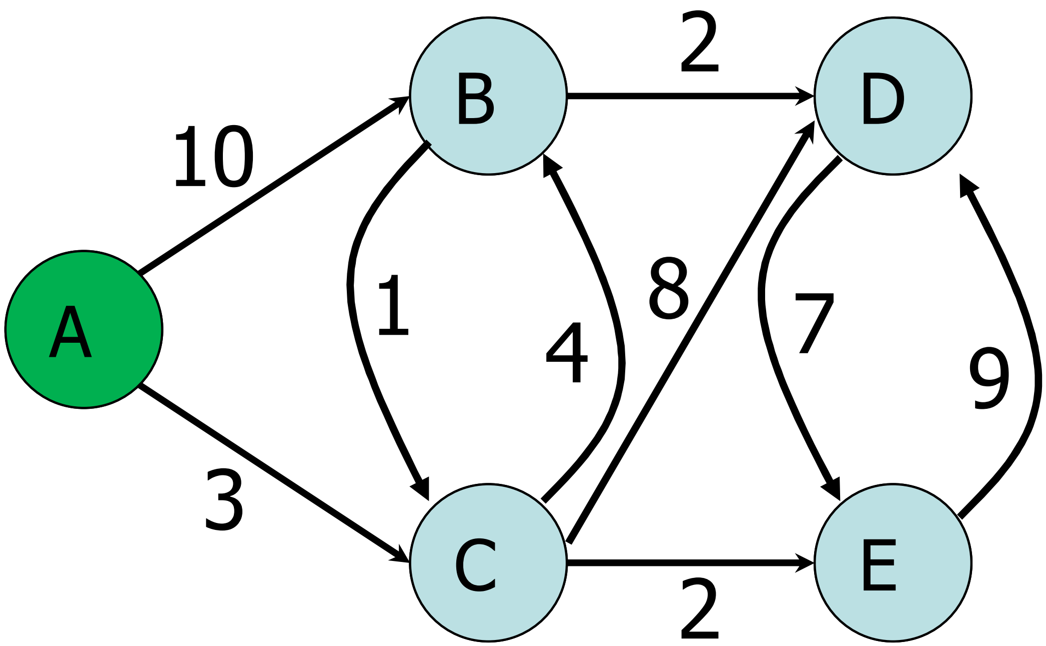
A	B	C	D	E
0	∞	∞	∞	∞

Initialization

S: {}

Vertex	Distance	Parent
A	0	Ø
B	∞	Ø
C	∞	Ø
D	∞	Ø
E	∞	Ø

Dijkstra's Algorithm – Example



Q: A B C D E

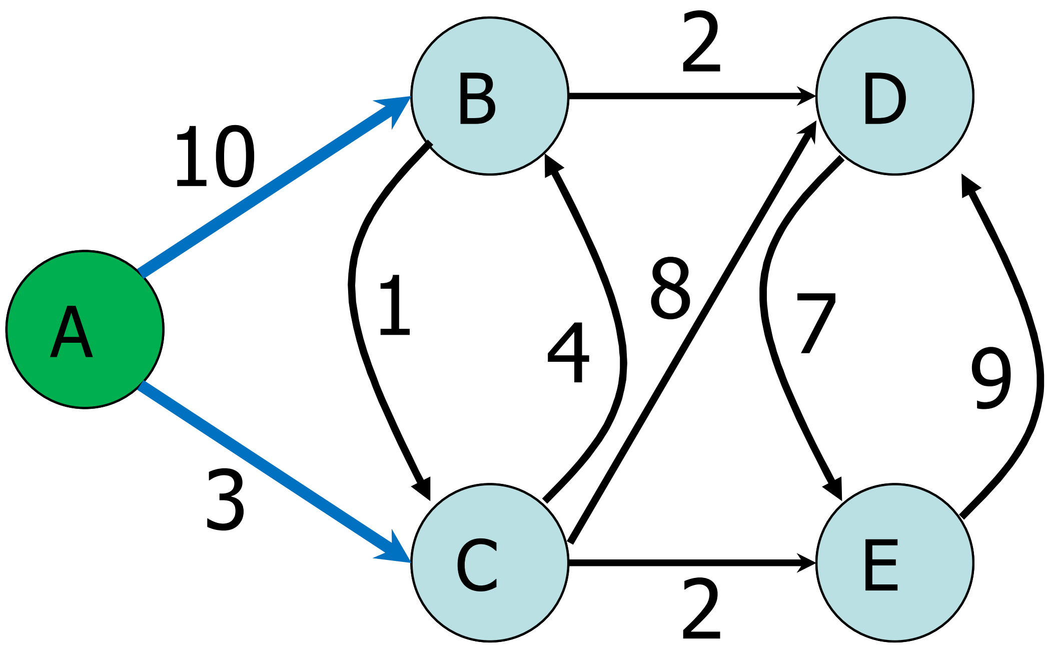
A	B	C	D	E
0	∞	∞	∞	∞

A ← EXTRACT-MIN(Q)

S: {A}

Vertex	Distance	Parent
A	0	Ø
B	∞	Ø
C	∞	Ø
D	∞	Ø
E	∞	Ø

Dijkstra's Algorithm – Example



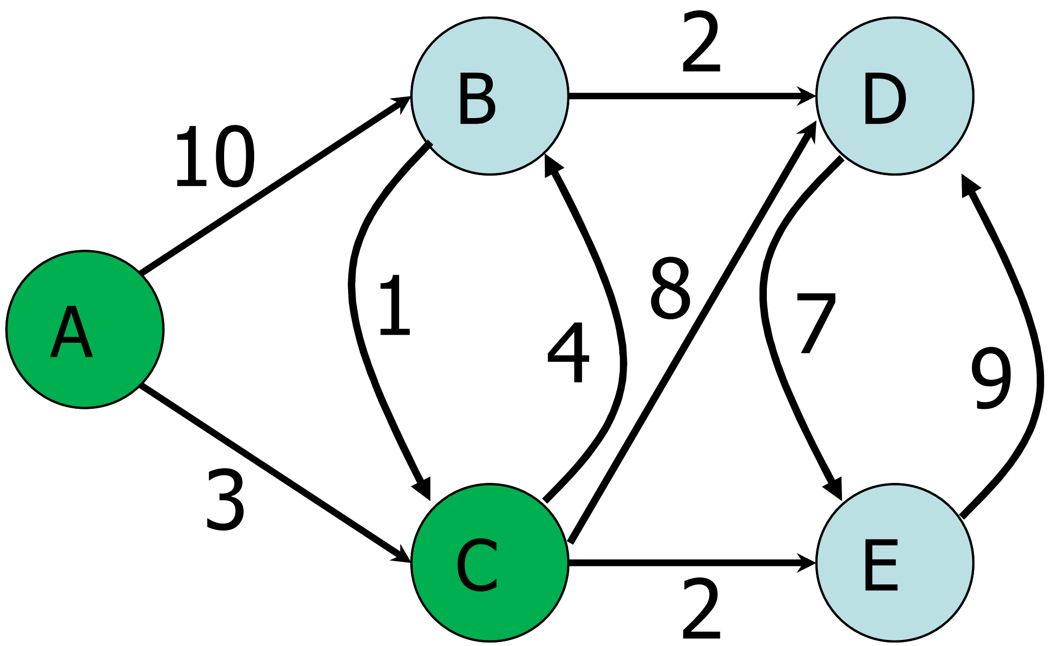
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

Update all neighbors of A

S: {A}

Vertex	Distance	Parent
A	0	Ø
B	10	A
C	3	A
D	∞	Ø
E	∞	Ø

Dijkstra's Algorithm – Example



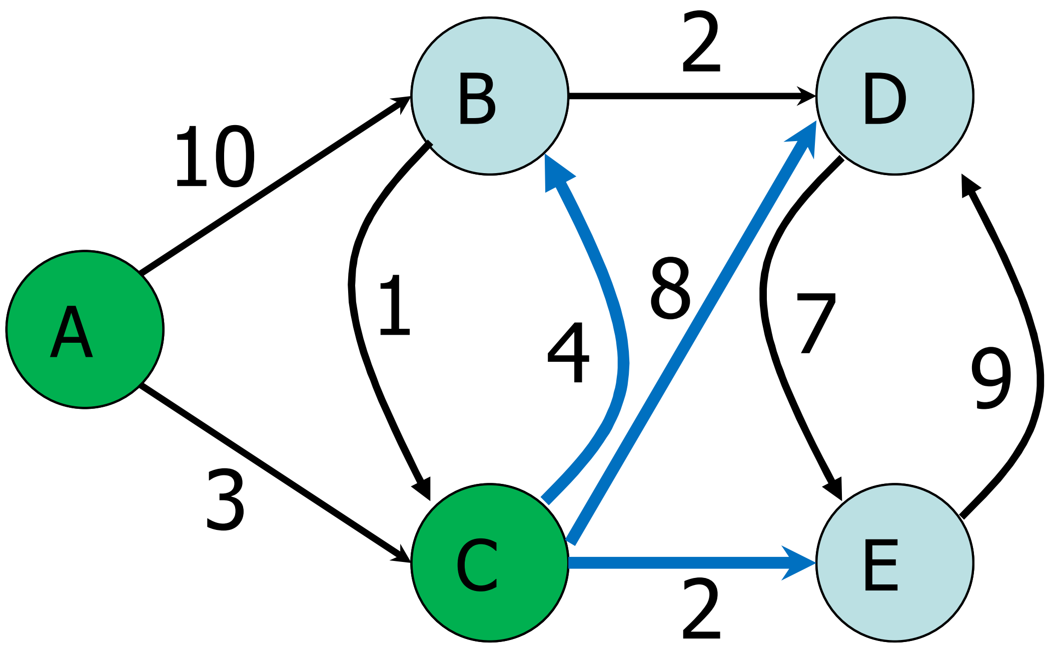
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

C ← EXTRACT-MIN(Q)

S: {A, C}

Vertex	Distance	Parent
A	0	Ø
B	10	A
C	3	A
D	∞	Ø
E	∞	Ø

Dijkstra's Algorithm – Example



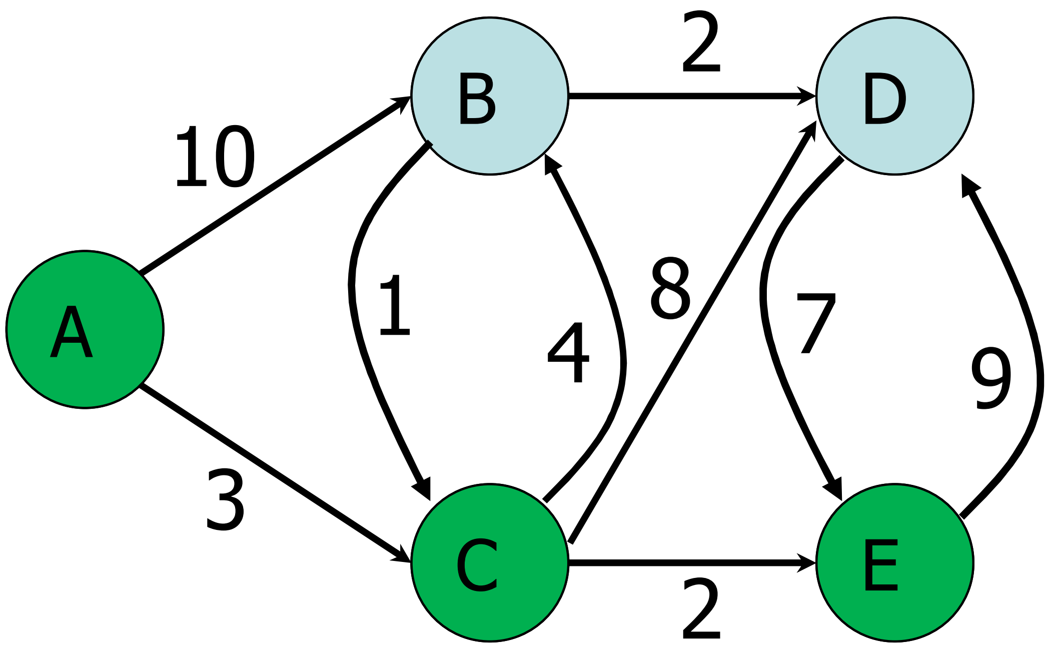
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

Update all neighbors of C

S: {A, C}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	11	C
E	5	C

Dijkstra's Algorithm – Example



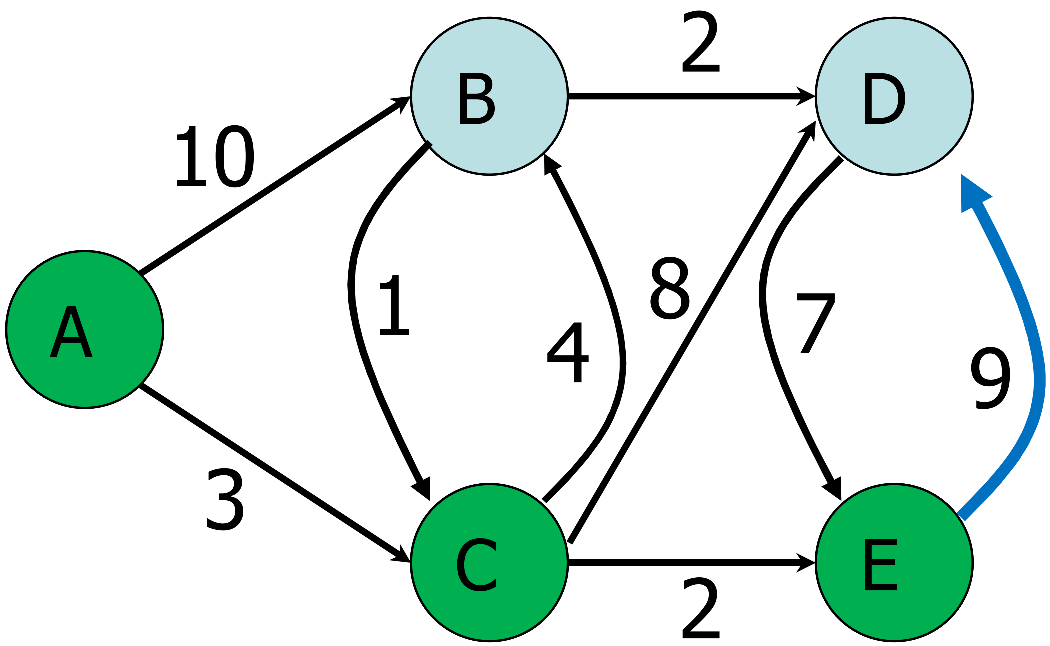
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

E ← EXTRACT-MIN(Q)

S: {A, C, E}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	11	C
E	5	C

Dijkstra's Algorithm – Example



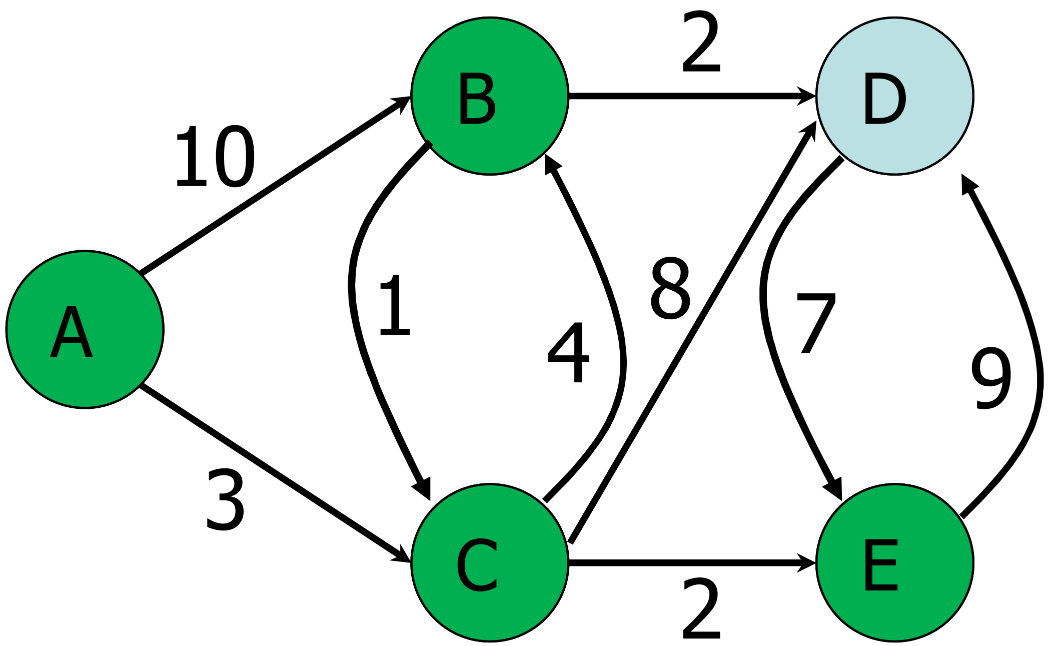
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

Update all neighbors of E

S: {A, C, E}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	11	C
E	5	C

Dijkstra's Algorithm – Example



Q:

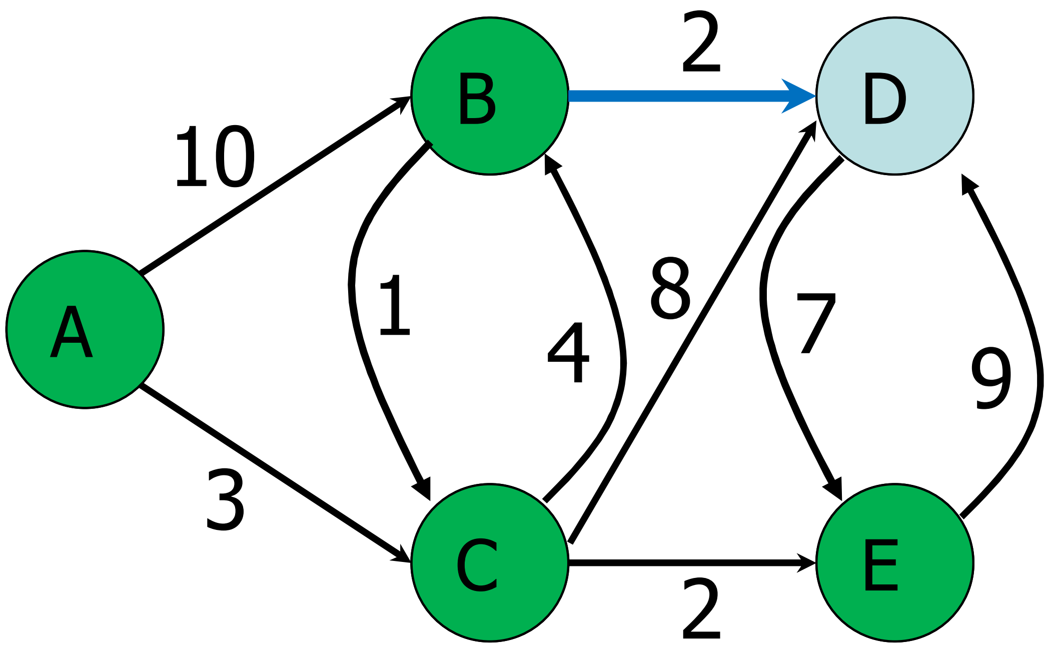
A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	

B ← EXTRACT-MIN(Q)

S: {A, C, E, B}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	11	C
E	5	C

Dijkstra's Algorithm – Example



Q: A B C D E

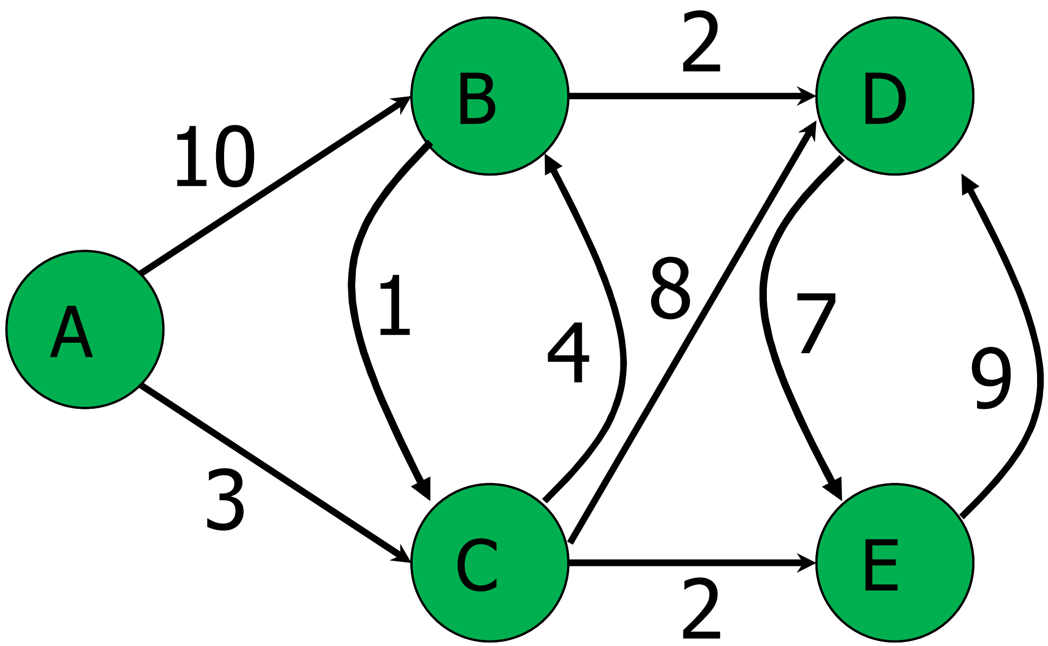
A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

Update all neighbors of B

S: {A, C, E, B}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	9	B
E	5	C

Dijkstra's Algorithm – Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

D ← EXTRACT-MIN(Q)

S: {A, C, E, B, D}

Vertex	Distance	Parent
A	0	Ø
B	7	C
C	3	A
D	9	B
E	5	C

Negative Edges

- Dijkstra’s algorithm is based on the greedy method
  - It adds vertices by increasing distance

Any Question So Far?