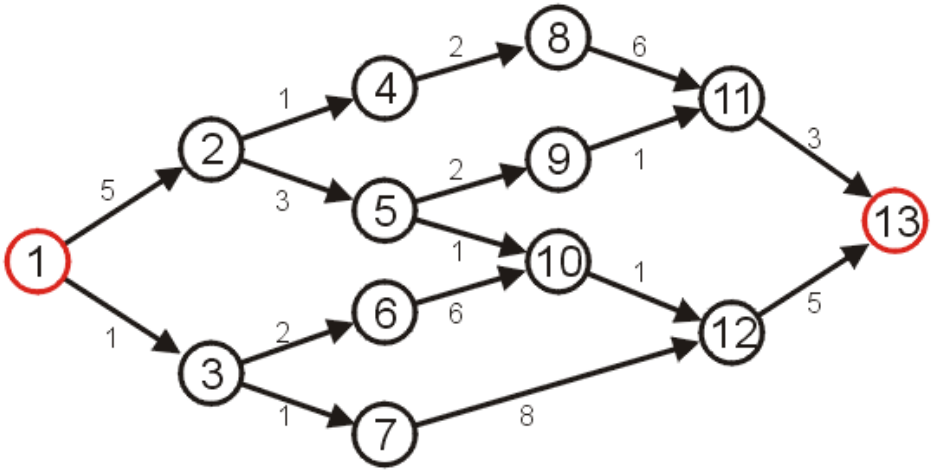
Data Structures

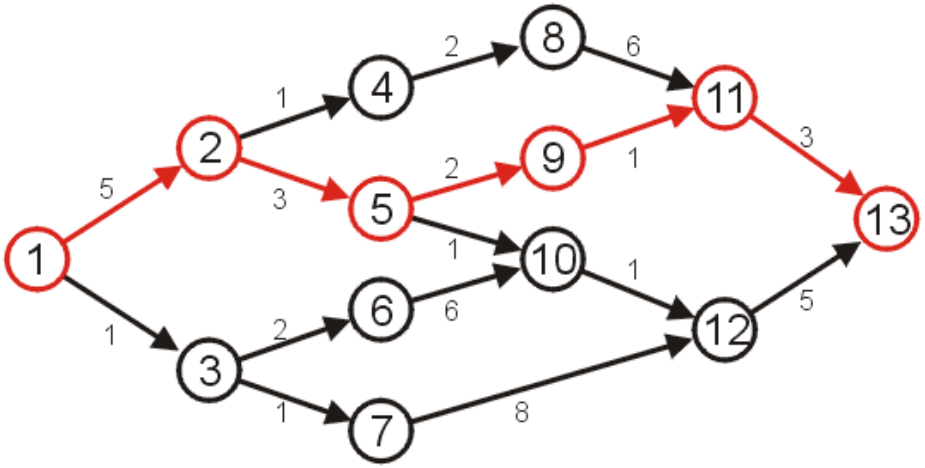
---

26. Shortest Path Trees

Shortest Path

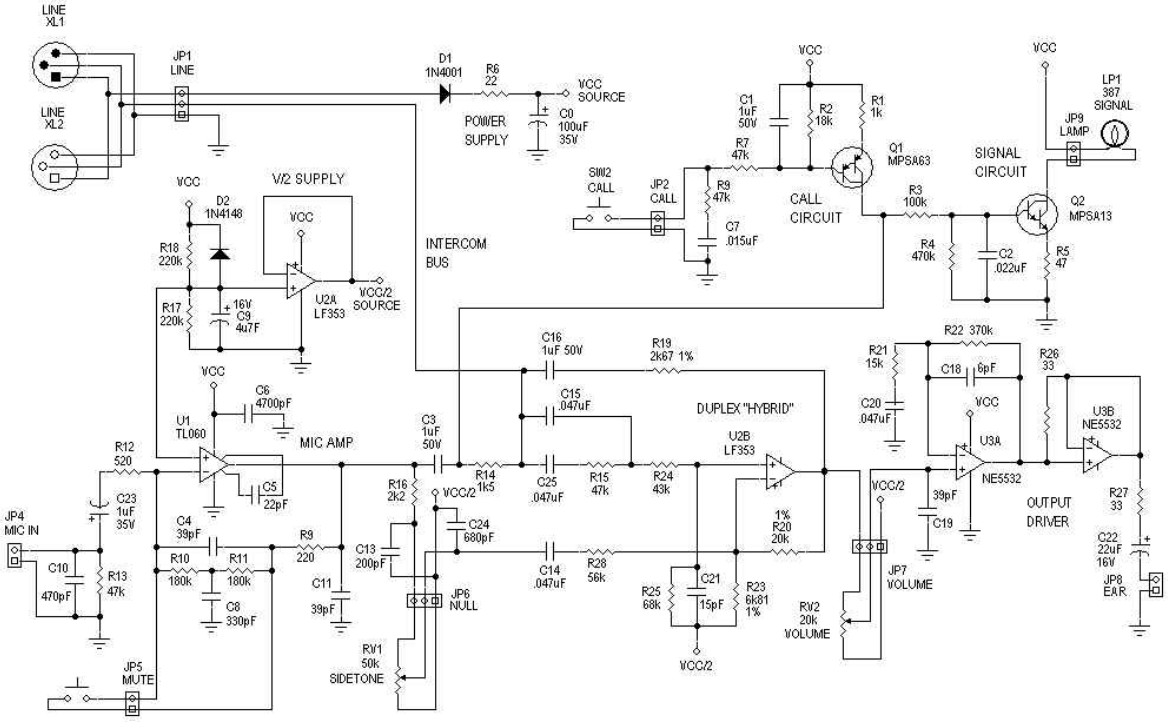
- Given a weighted graph
  - Problem is to find the shortest path between two given vertices
- Length of a path in a weighted graph
  - Sum of the weights of each of the edges in that path
- Example: Shortest path from vertex 1 to vertex 13
  - Other paths exists but they are longer





Application – Circuit Design

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

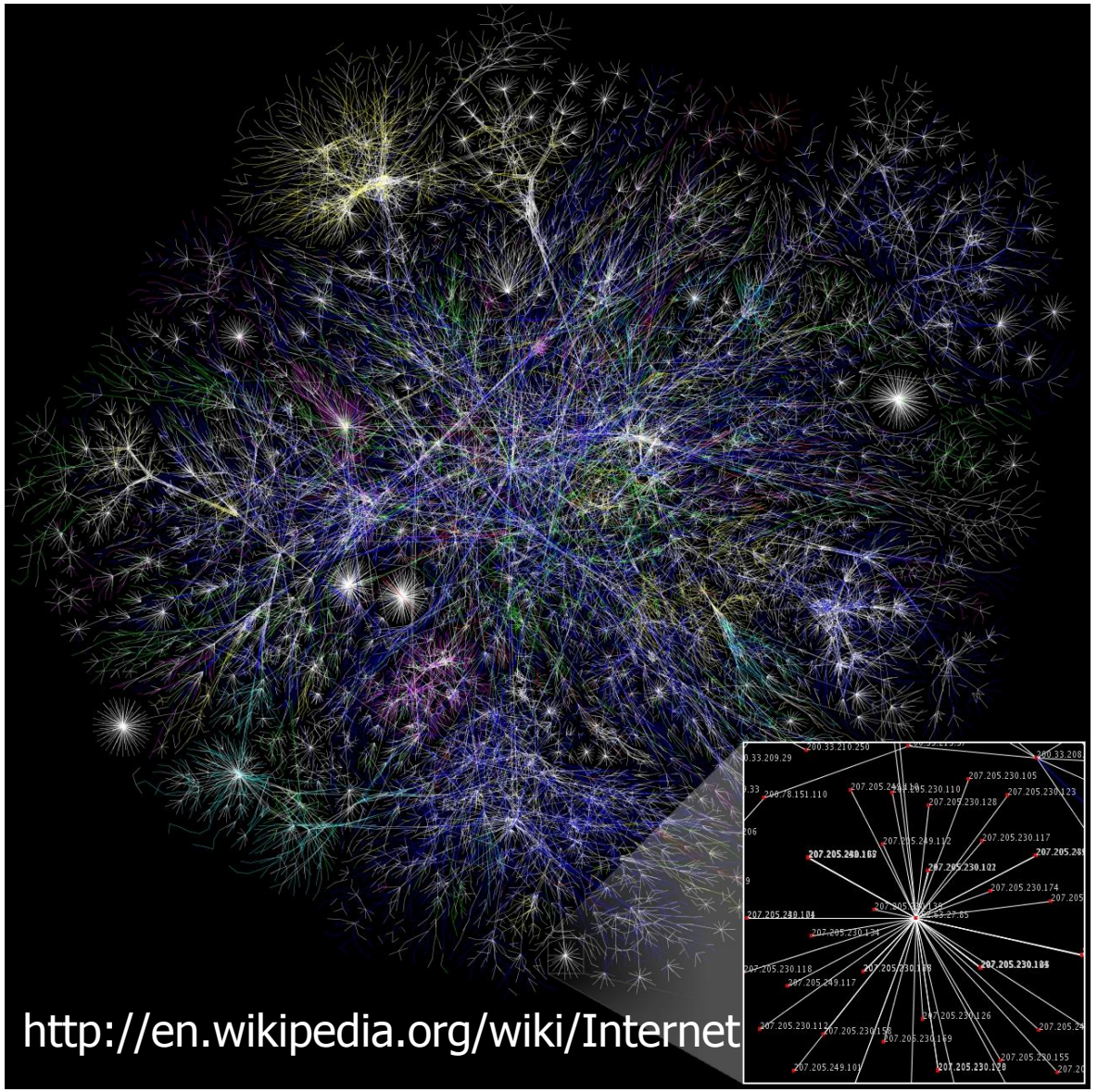


Application – Computer Networks

- 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



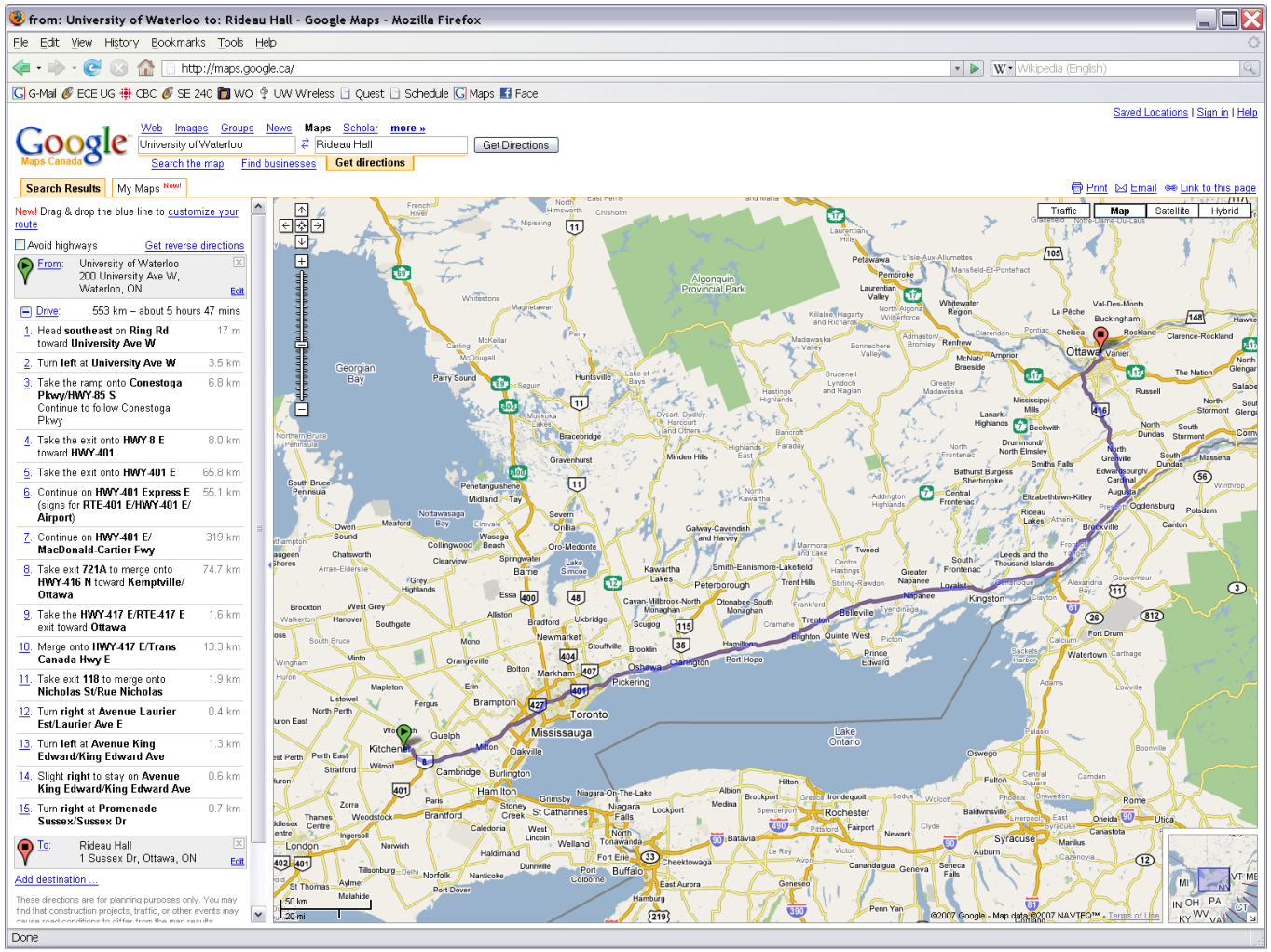
Application – Computer Networks

- 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 = ∅                                //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

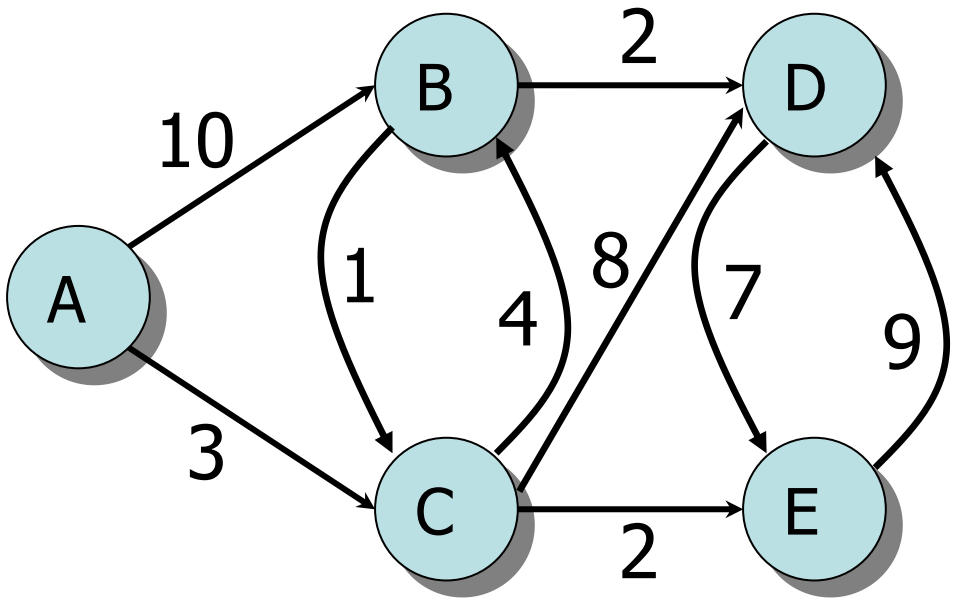
Dijkstra's Algorithm

```
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



Q: A B C D E

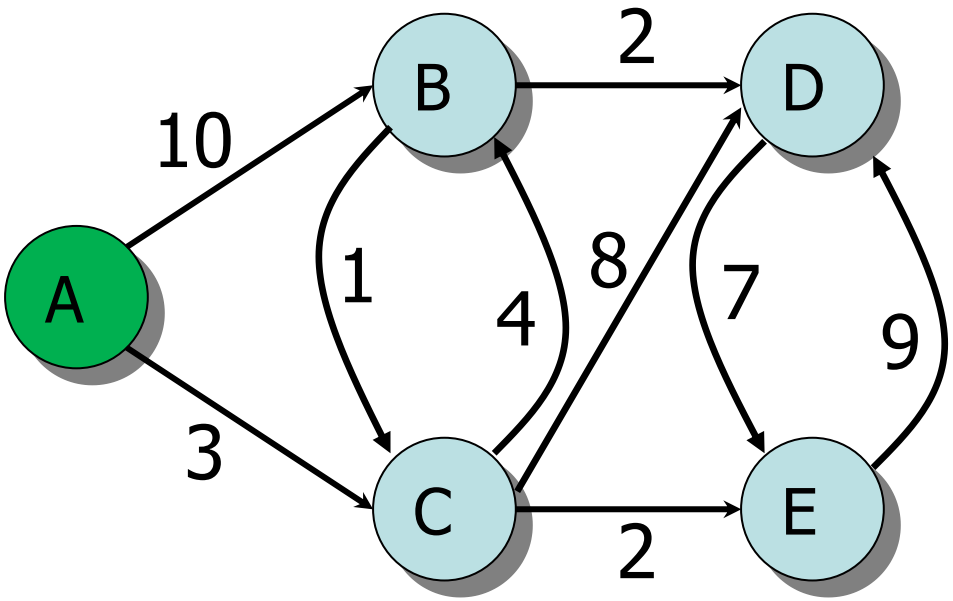
0 ∞ ∞ ∞ ∞

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

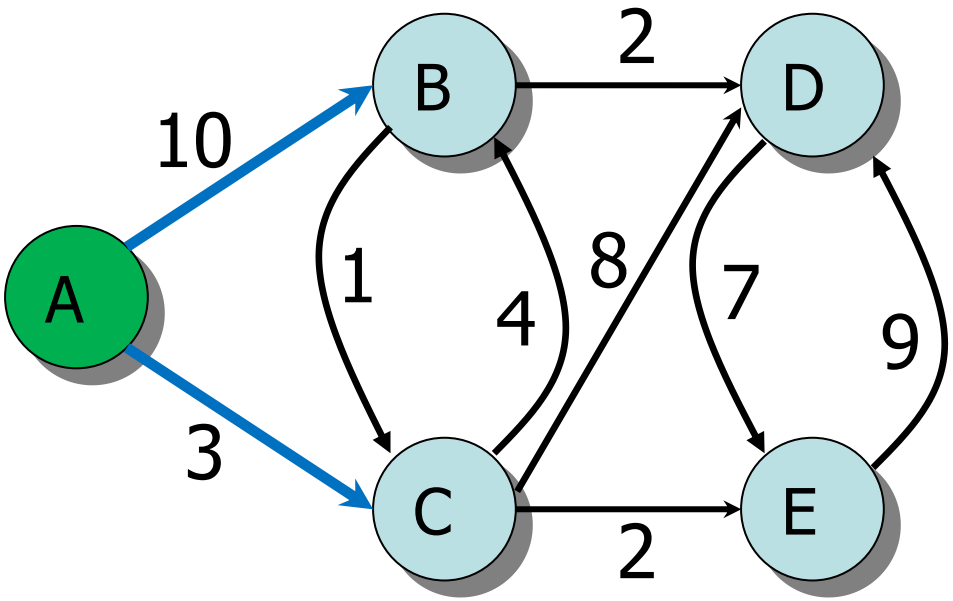
0	∞	∞	∞	∞

A ← EXTRACT-MIN(Q)

S: {A}

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

Dijkstra's Algorithm – Example



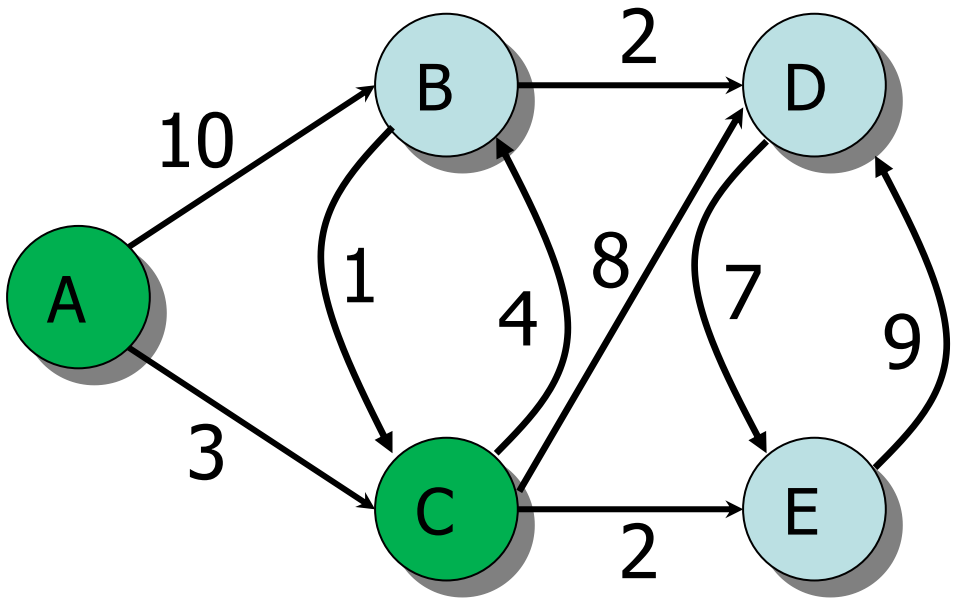
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

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



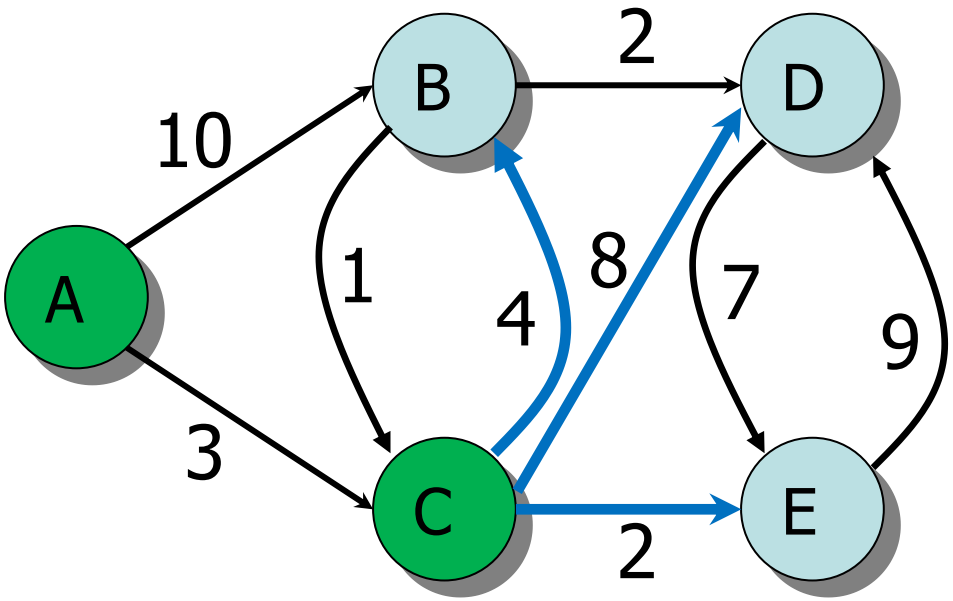
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

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



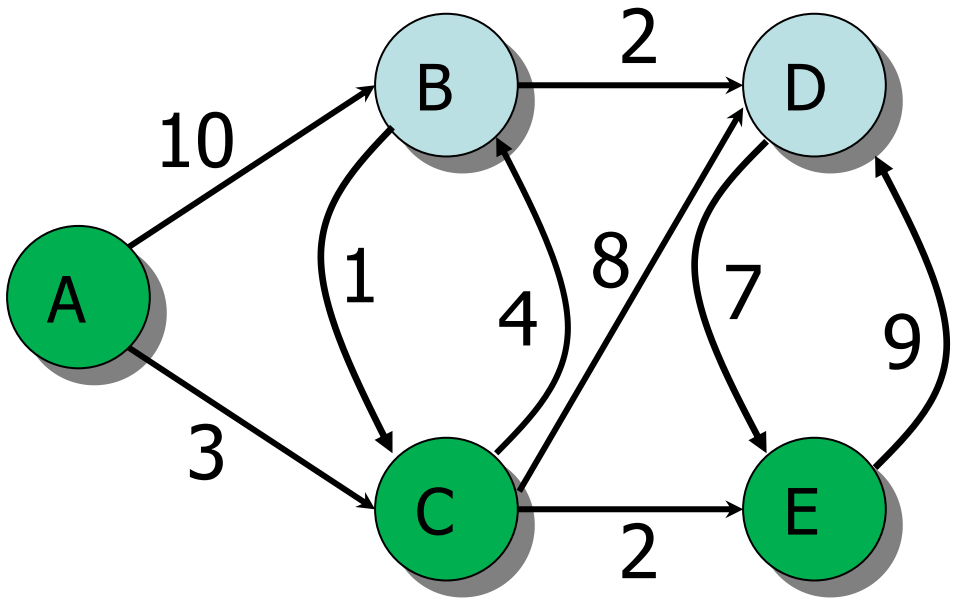
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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



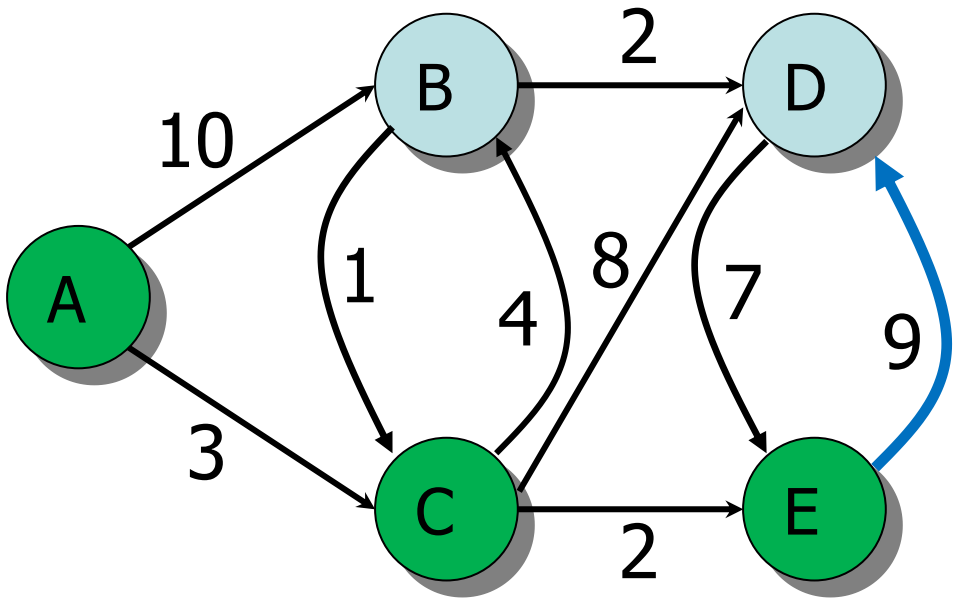
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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



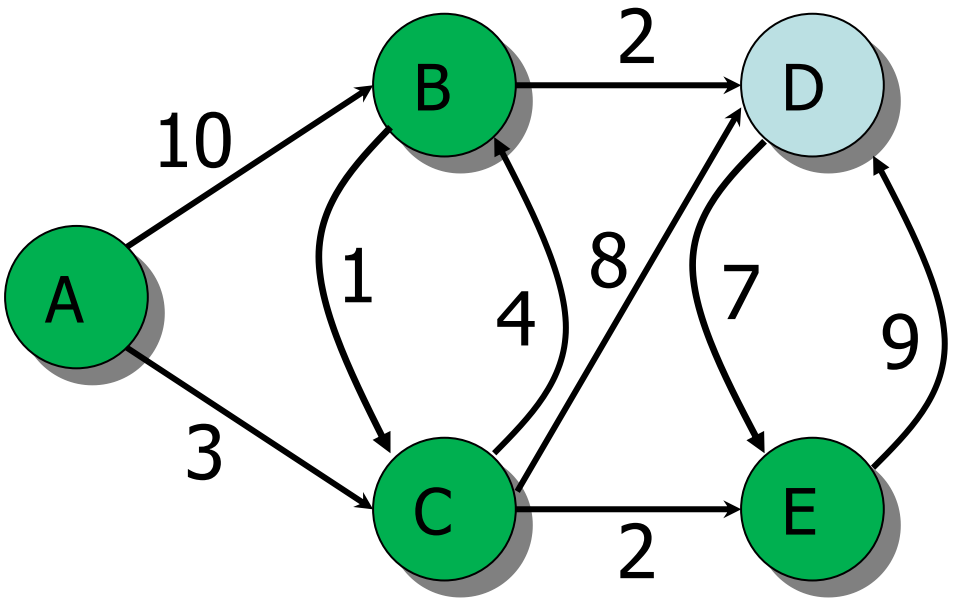
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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



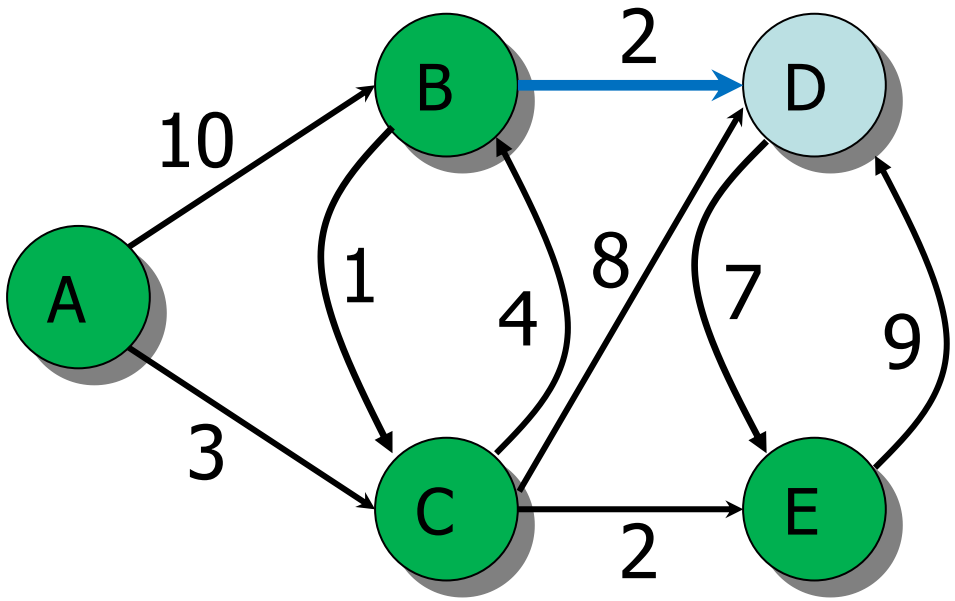
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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



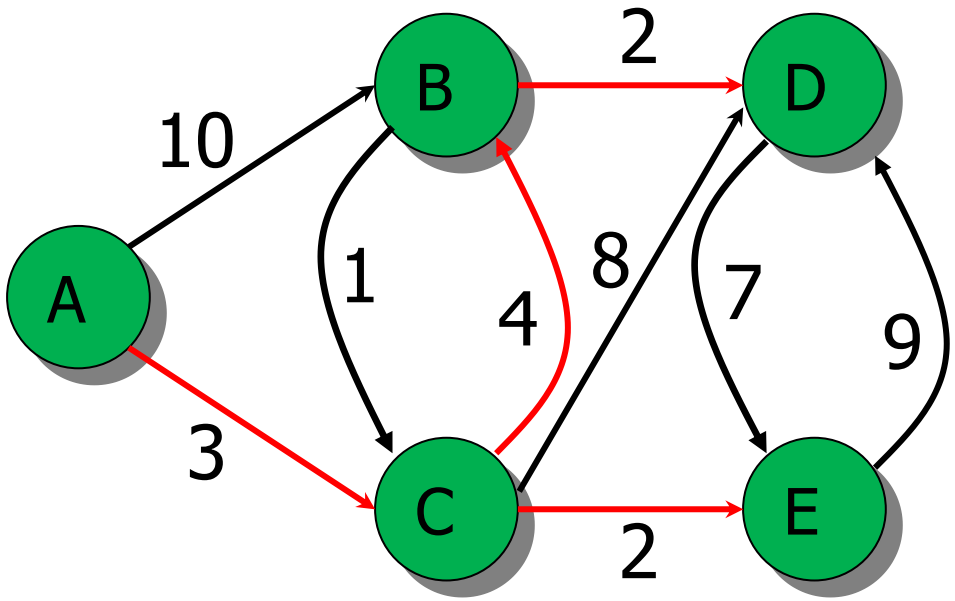
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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	

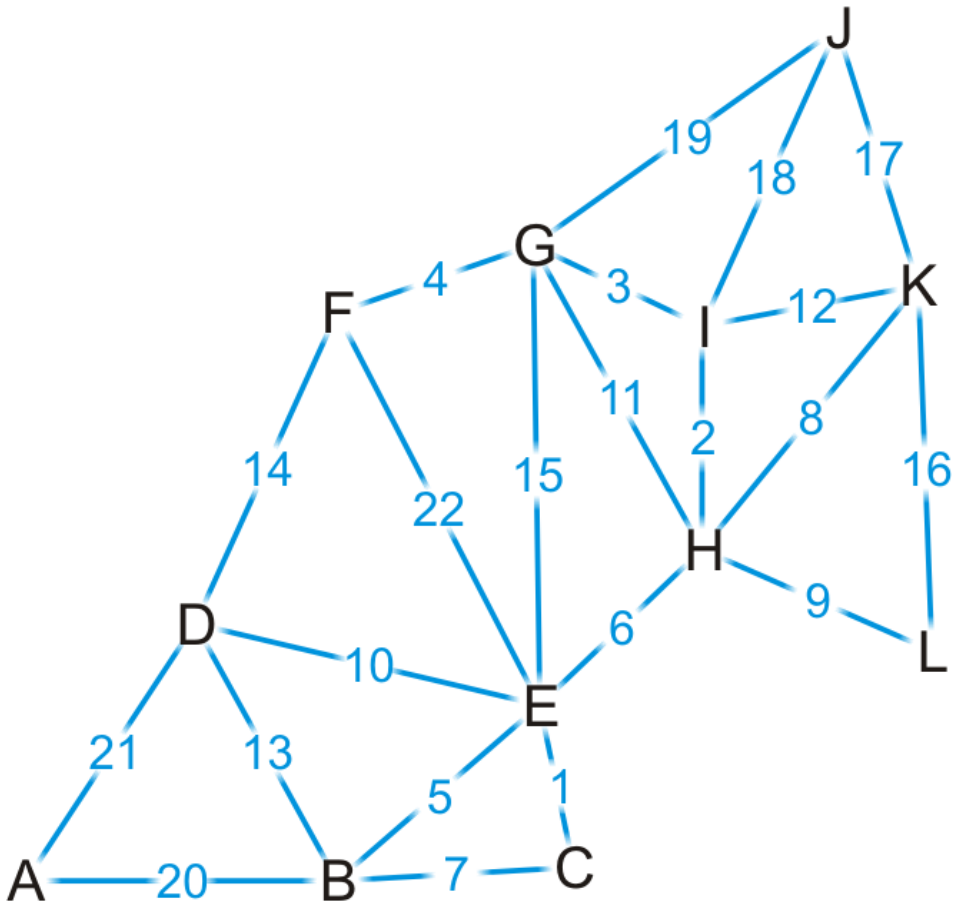
D ← EXTRACT-MIN(Q)

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

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

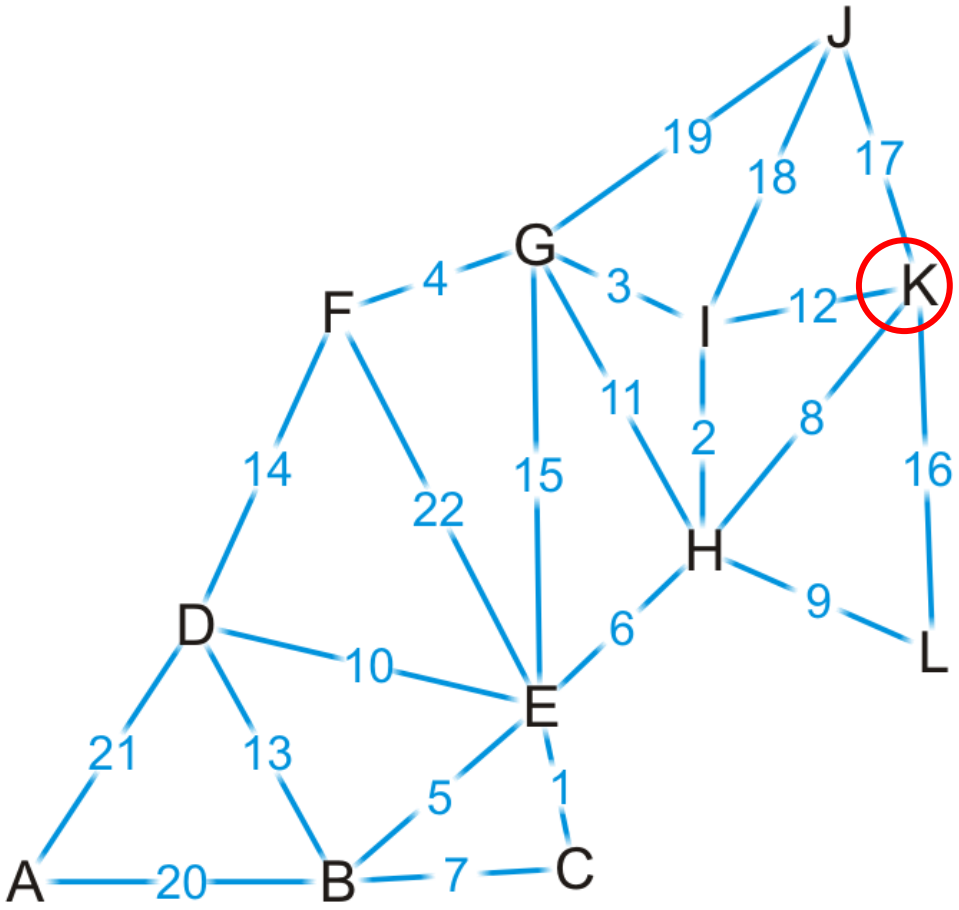
Dijkstra's Algorithm – Example

- Find the shortest path from K to every other vertex



Dijkstra's Algorithm – Example

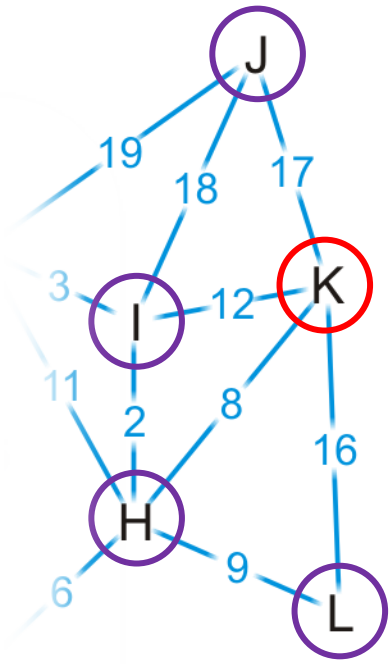
- We visit vertex K



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

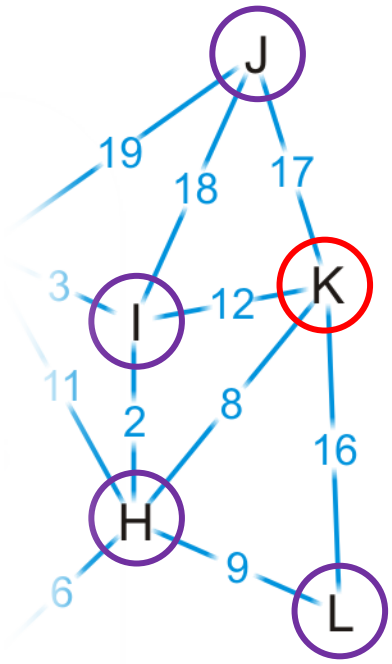
- Vertex K has four neighbors: H, I, J and L



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

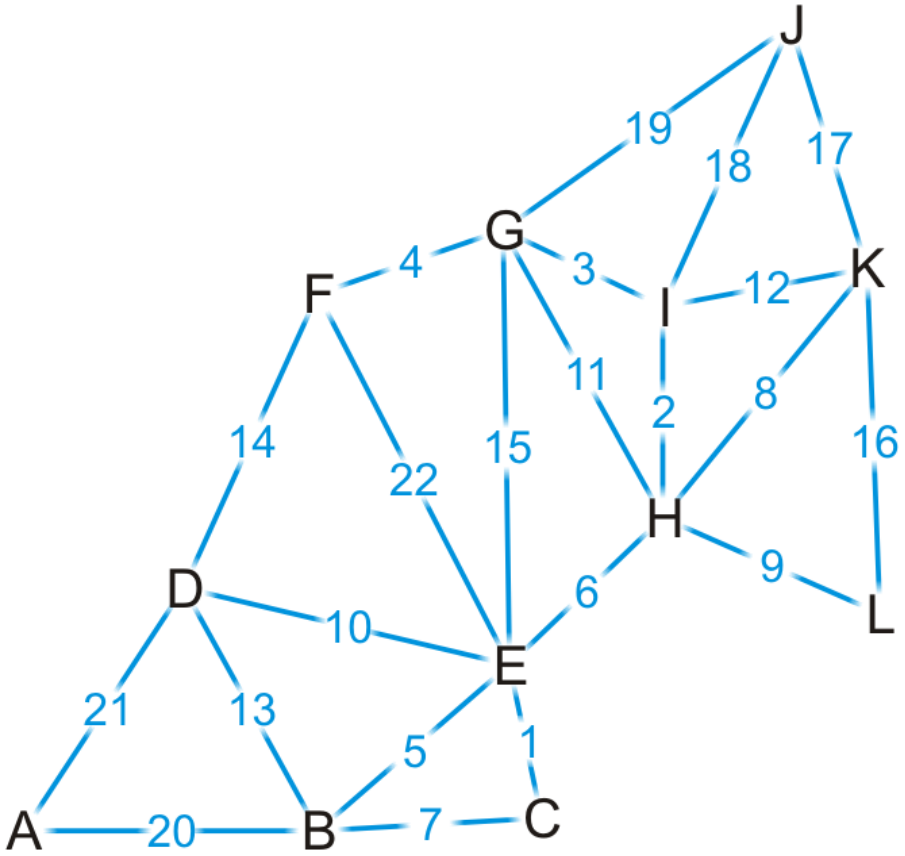
- We have now found at least one path to each of these vertices



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

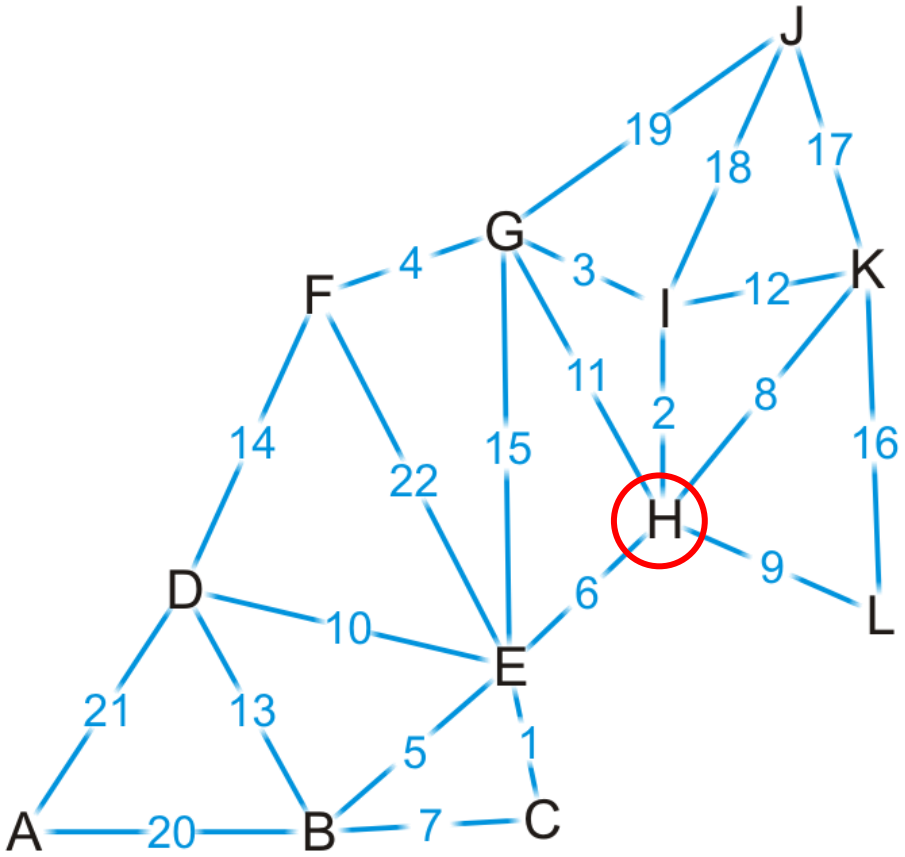
- We're finished with vertex K
  - To which vertex are we now guaranteed we have the shortest path?
    - `mindistance(Q)`



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



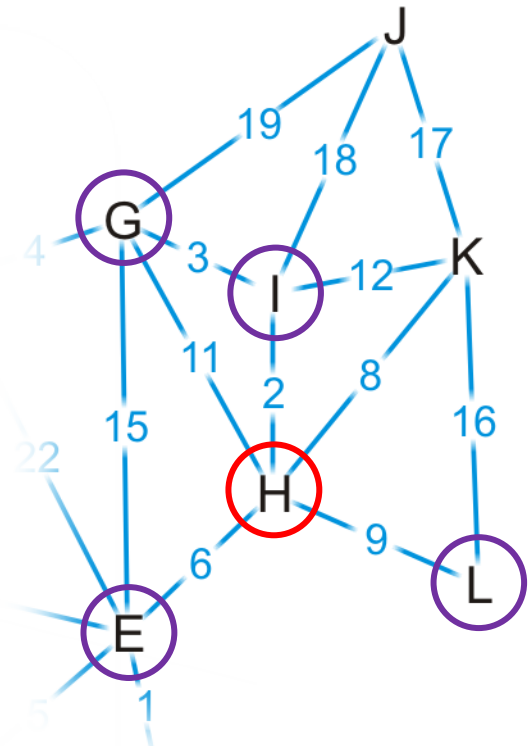
Vertex	S	Distance	Parent
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	**T**	**8**	**K**
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Dijkstra's Algorithm – Example

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

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

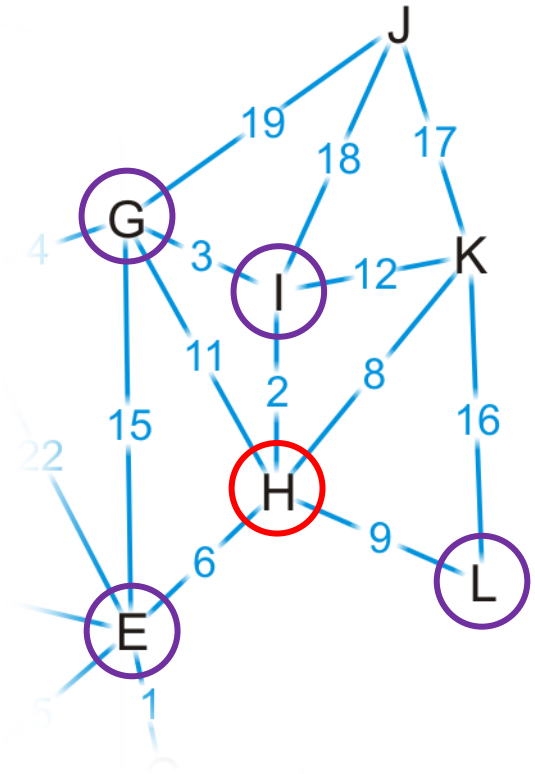
(K, H, L) of length 8 + 9 = 17



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

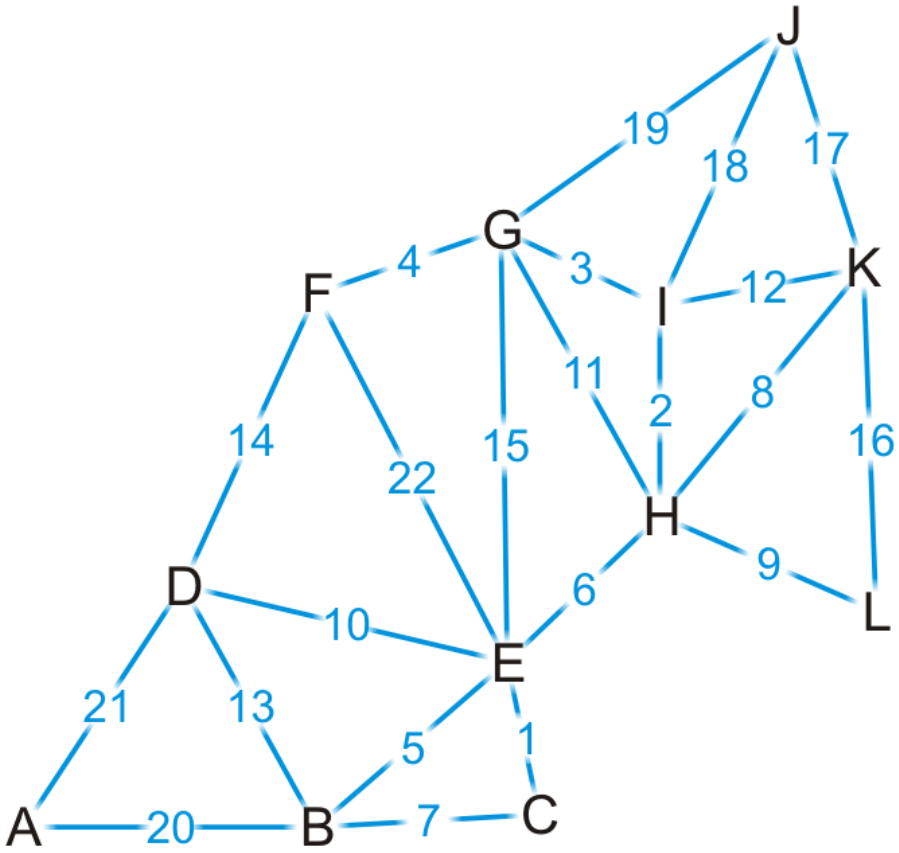
- We already have a shorter path (K, L), but we update the others



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

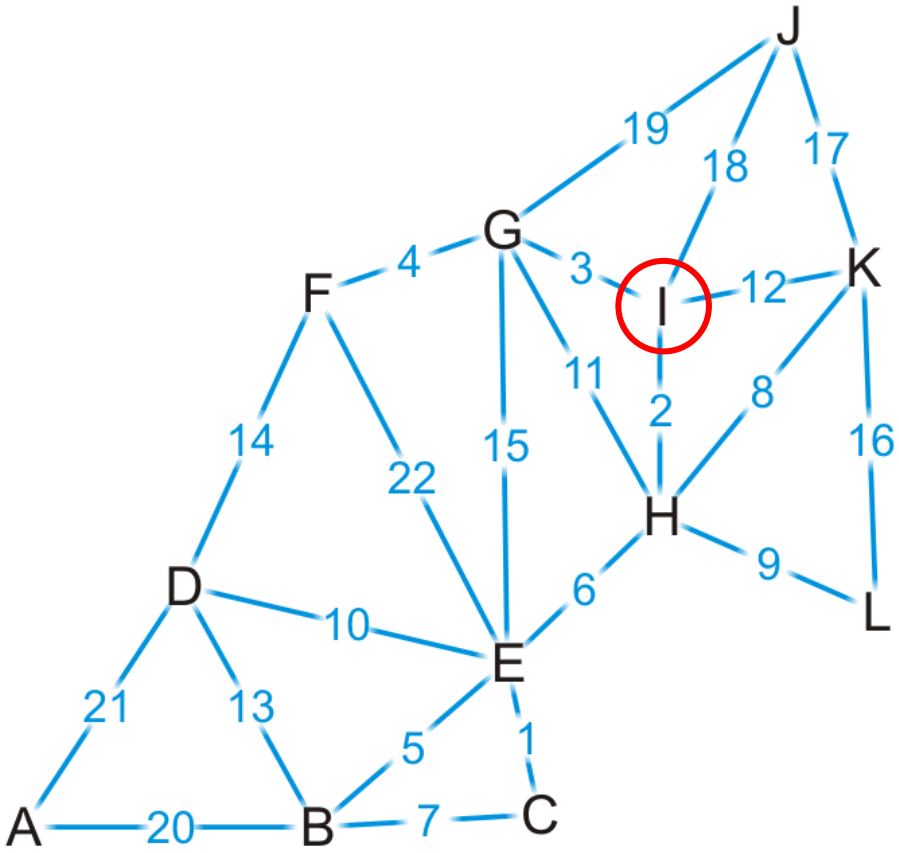
- We are finished with vertex H
  - Which vertex do we visit next?



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

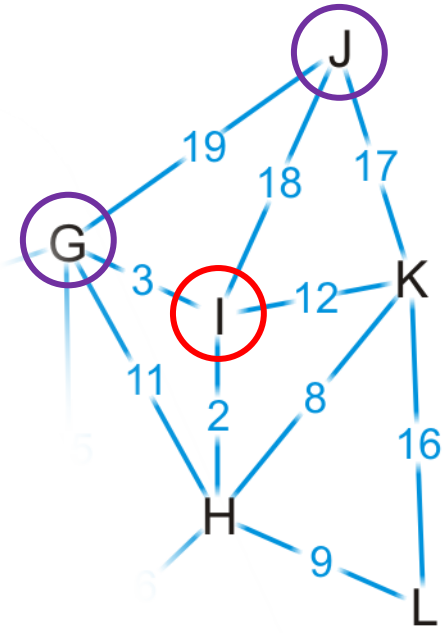
- The path (K, H, I) is the shortest path from K to I of length 10
  - Vertex I has two unvisited neighbors: G and J



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

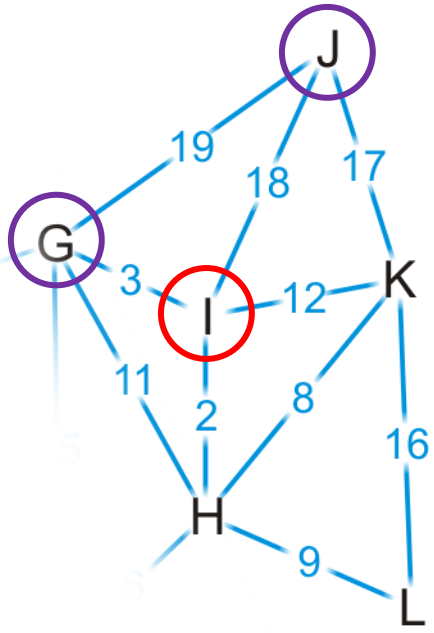
- Consider these paths:
  - (K, H, I, G) of length 10 + 3 = 13
  - (K, H, I, J) of length 10 + 18 = 28



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

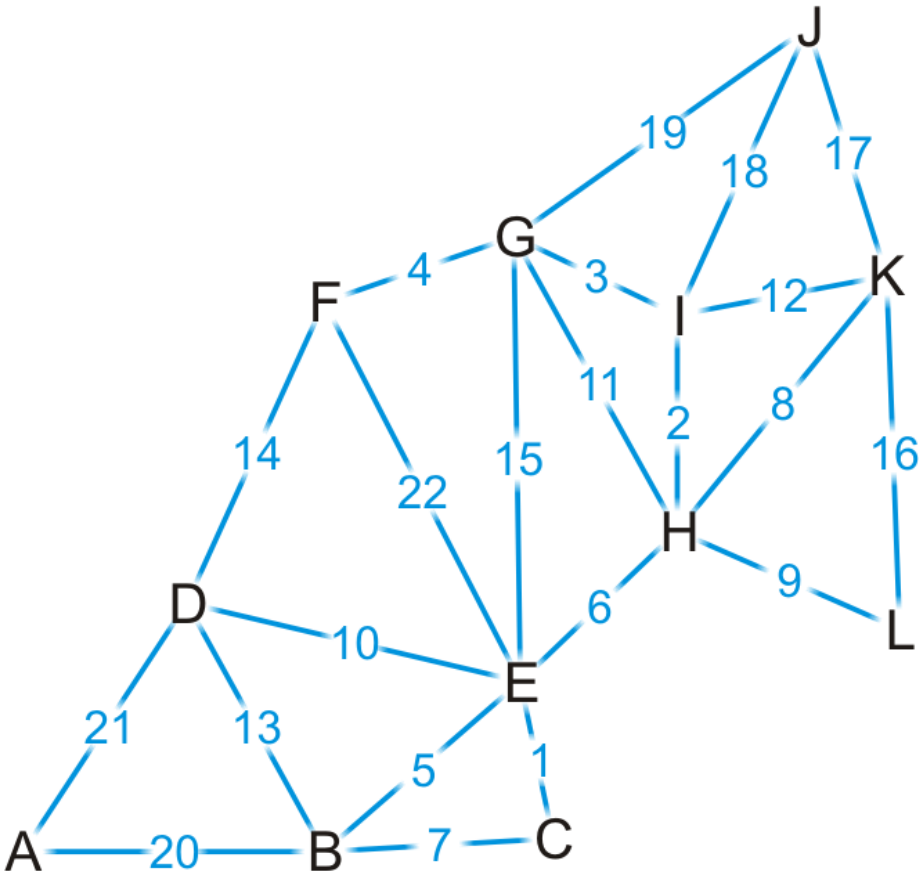
- We have discovered a shorter path to vertex G, but (K, J) is still the shortest known path to vertex J



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

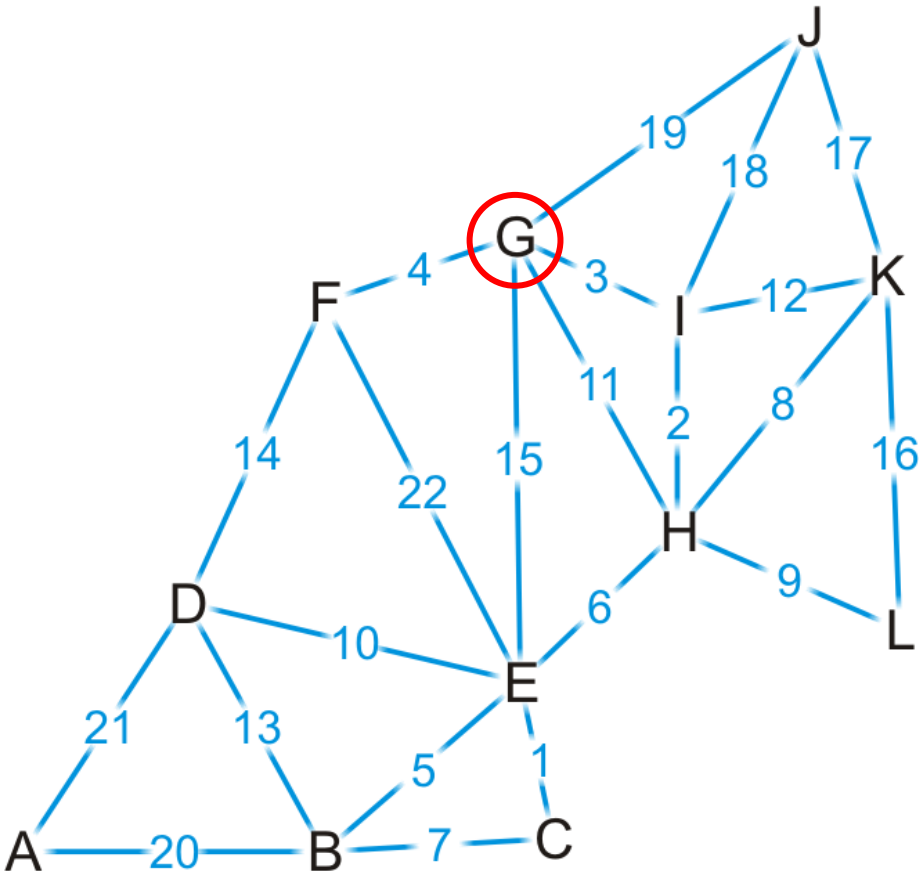
- Which vertex can we visit next?



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

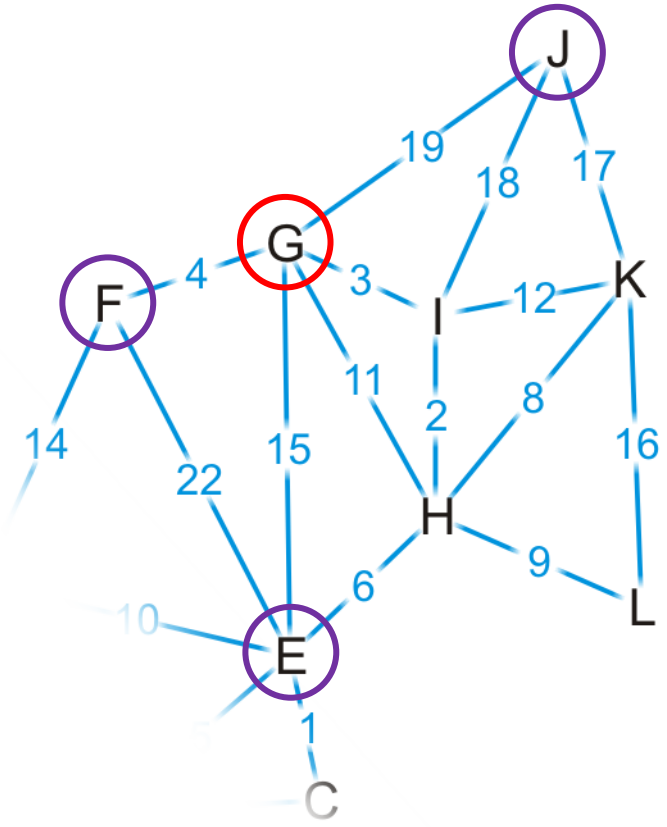
- The path (K, H, I, G) is the shortest path from K to G of length 13
  - Vertex G has three unvisited neighbors: E, F and J



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

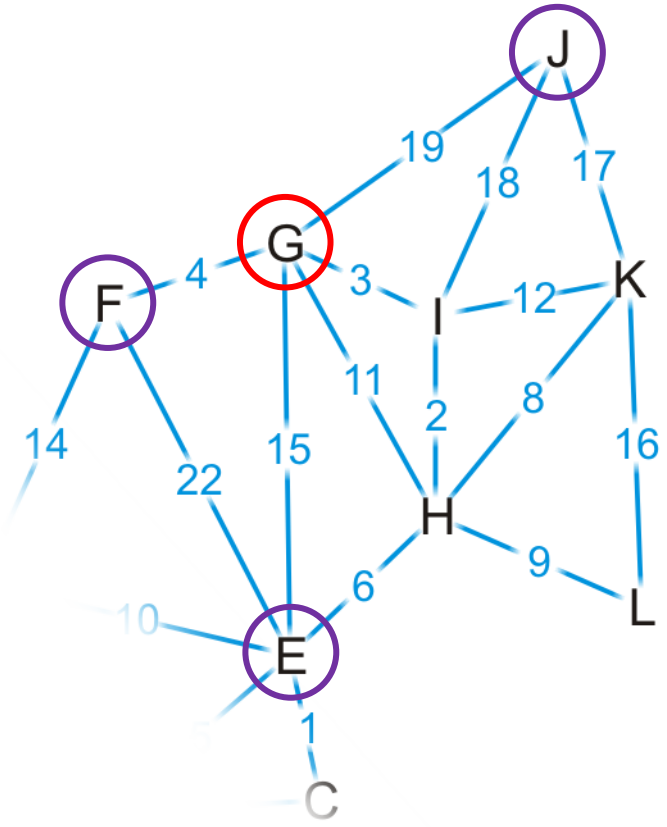
- Consider these paths:
  - (K, H, I, G, E) of length 13 + 15 = 28
  - (K, H, I, G, F) of length 13 + 4 = 17
  - (K, H, I, G, J) of length 13 + 19 = 32



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

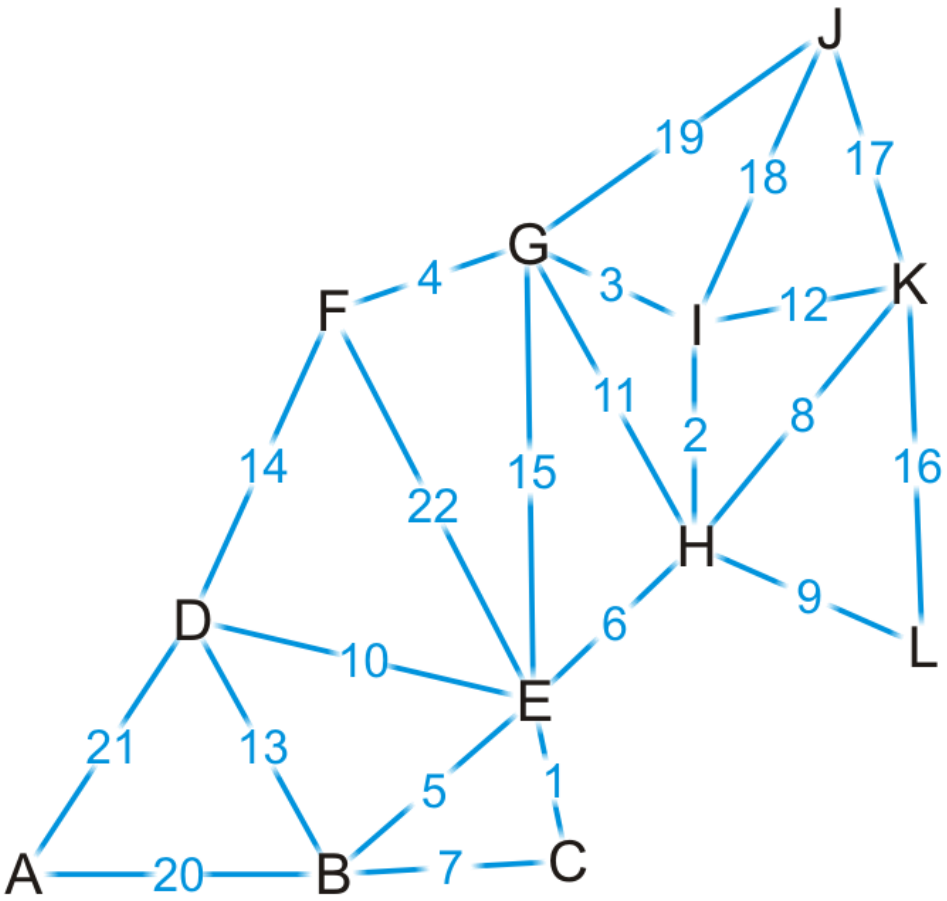
- We have now found a path to vertex F



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

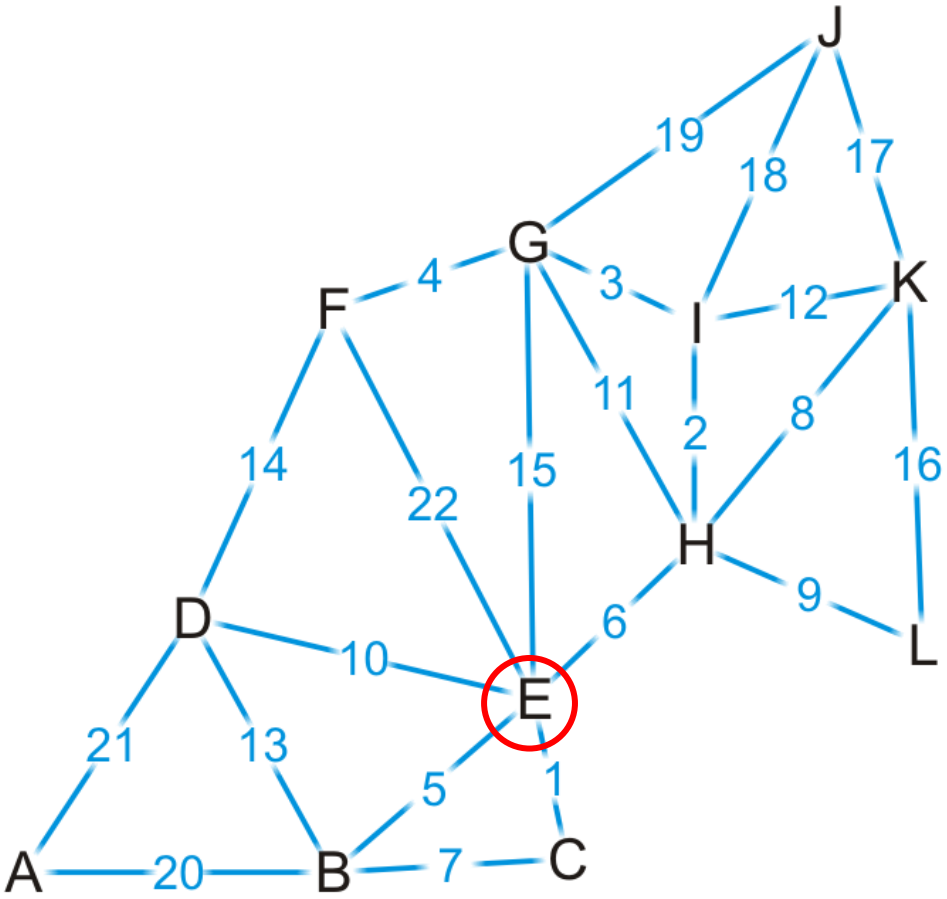
- Where do we visit next?



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

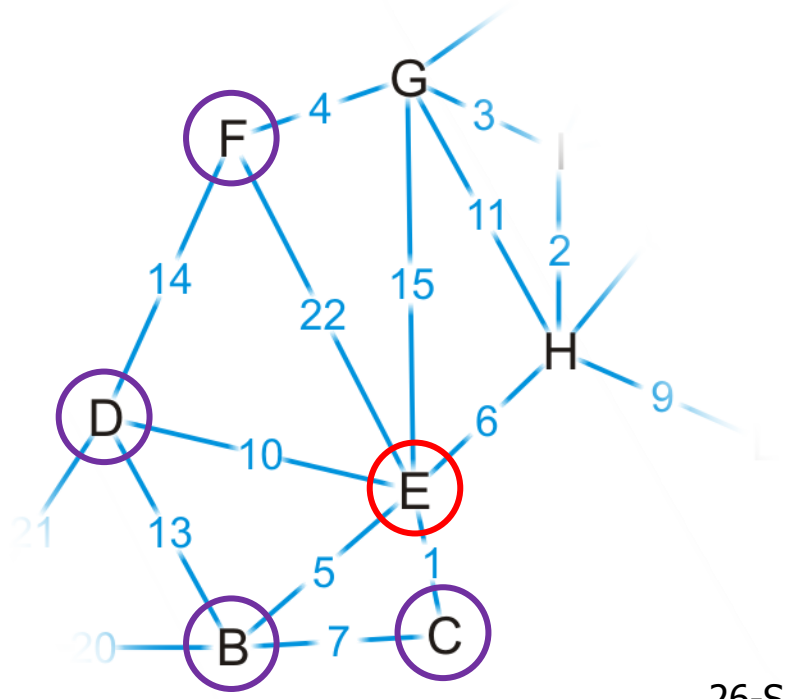
- The path (K, H, E) is the shortest path from K to E of length 14
  - Vertex G has four unvisited neighbors: B, C, D and F



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

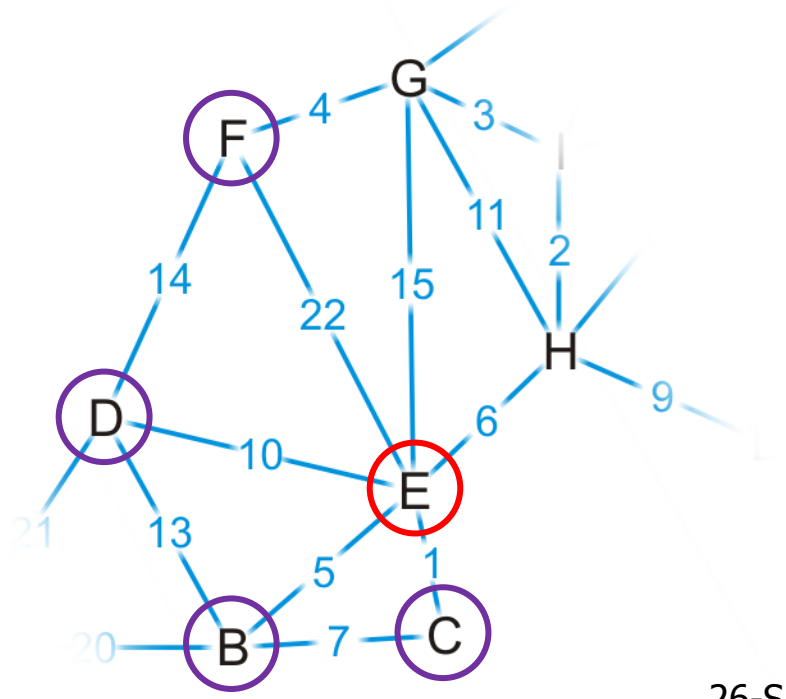
- The path (K, H, E) is the shortest path from K to E of length 14
  - Vertex G has four unvisited neighbors: B, C, D and F



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

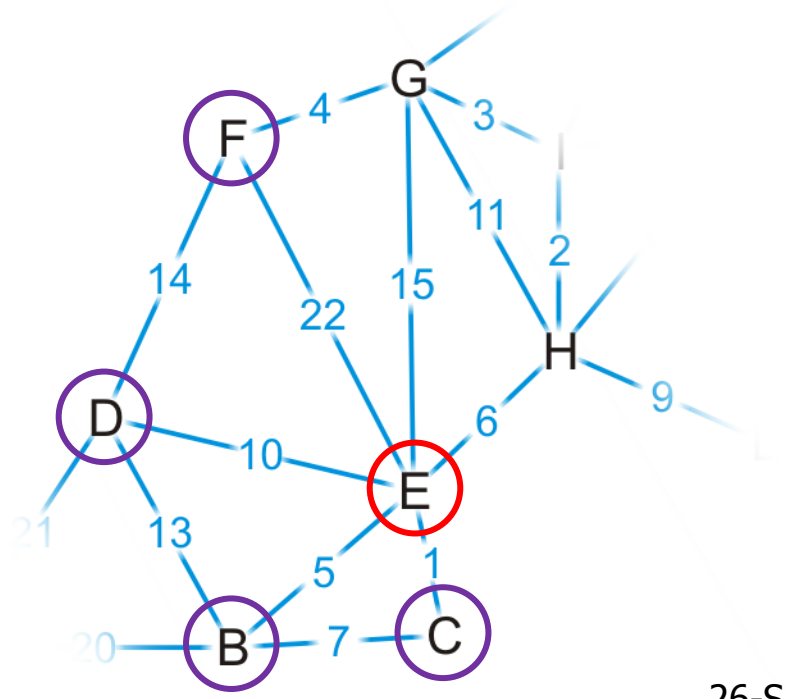
- Consider these paths:
  - (K, H, E, B) of length 14 + 5 = 19
  - (K, H, E, C) of length 14 + 1 = 15
  - (K, H, E, D) of length 14 + 10 = 24
  - (K, H, E, F) of length 14 + 22 = 36



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

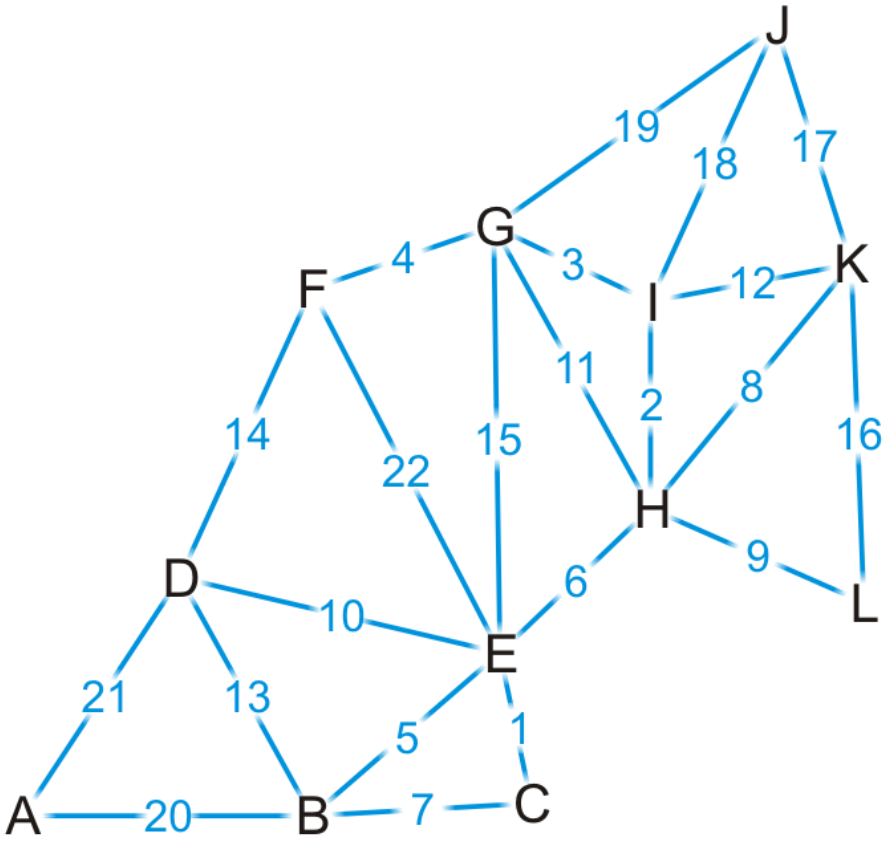
- We've discovered paths to vertices B, C, D



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

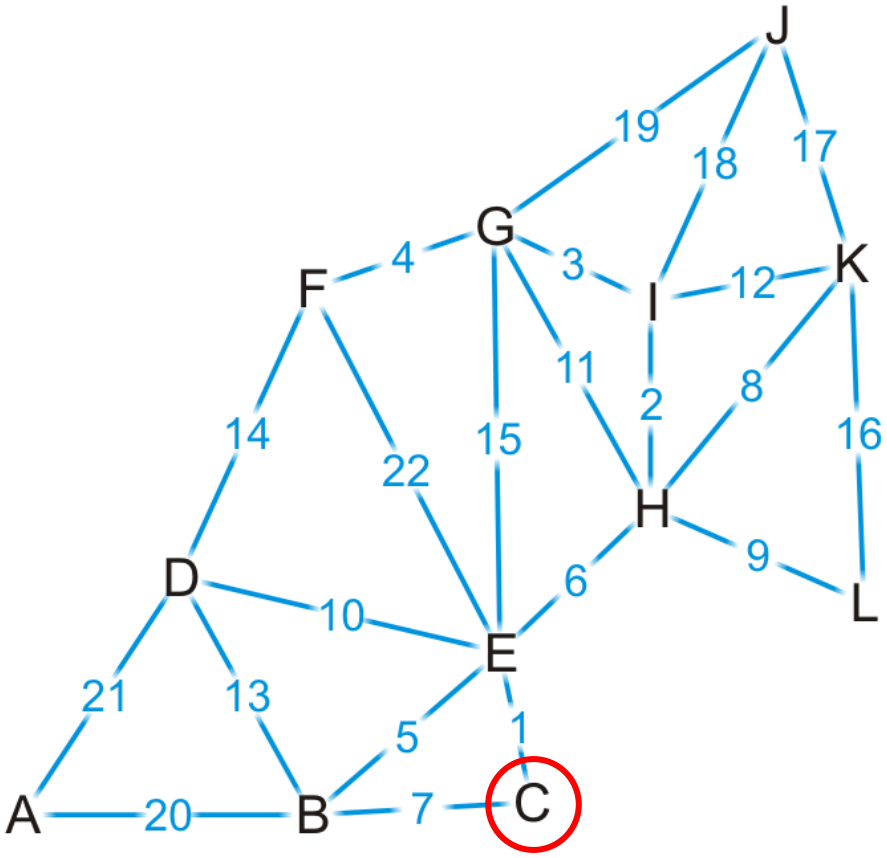
- Which vertex is next?



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

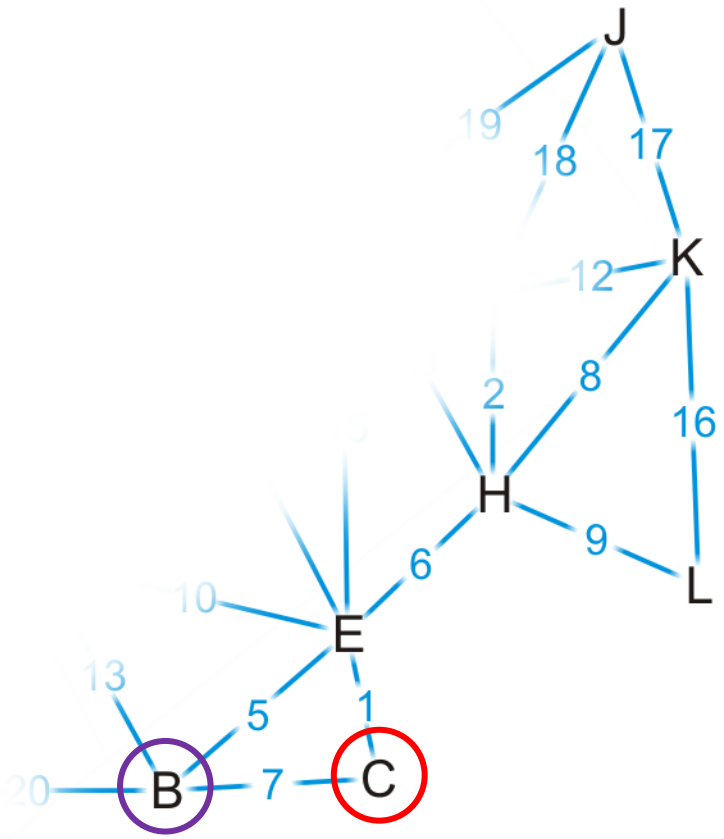
- We've found that the path (K, H, E, C) of length 15 is the shortest path from K to C
  - Vertex C has one unvisited neighbor, B



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

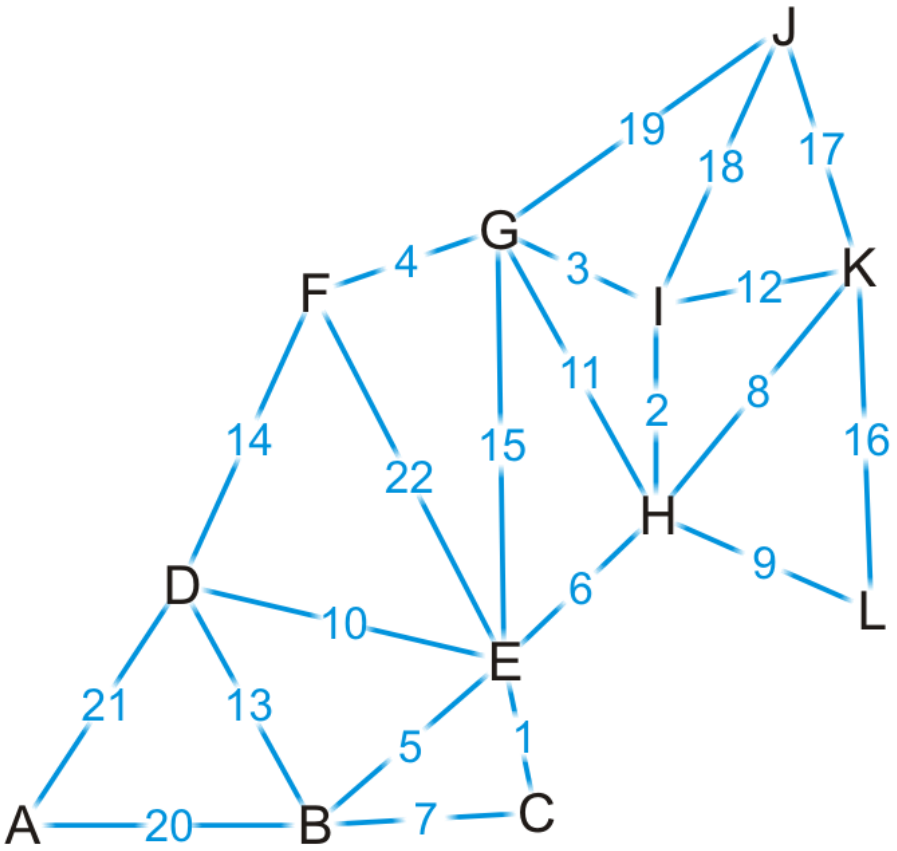
- The path (K, H, E, C, B) is of length 15 + 7 = 22
  - We have already discovered a shorter path through vertex E



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

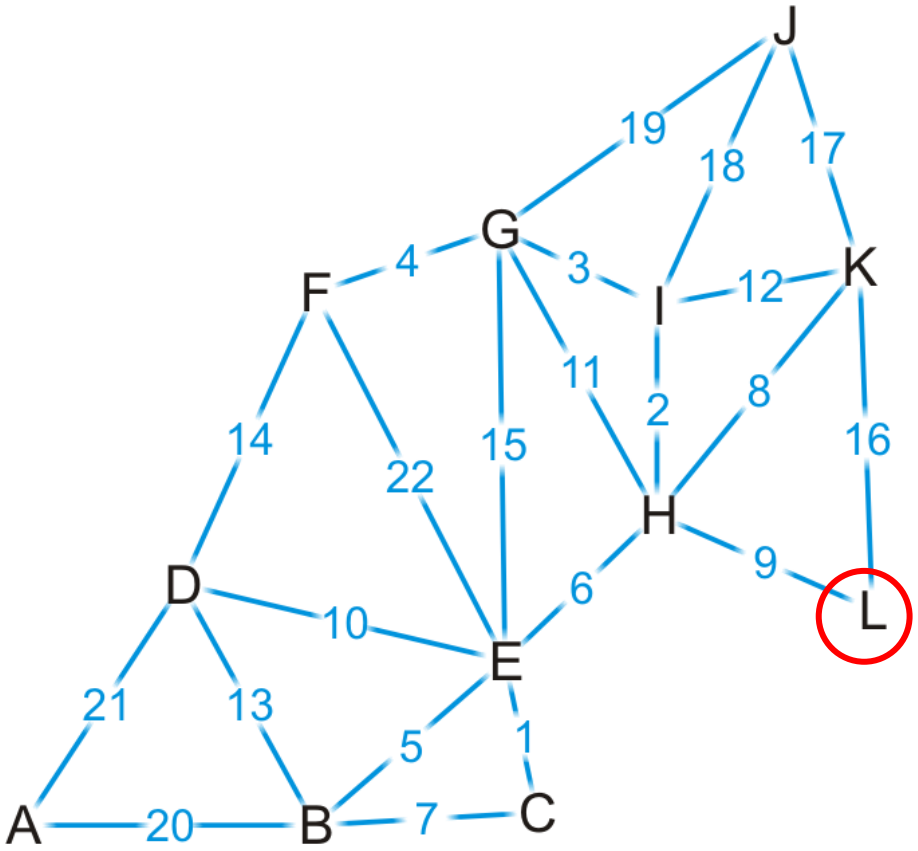
- Where to next?



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

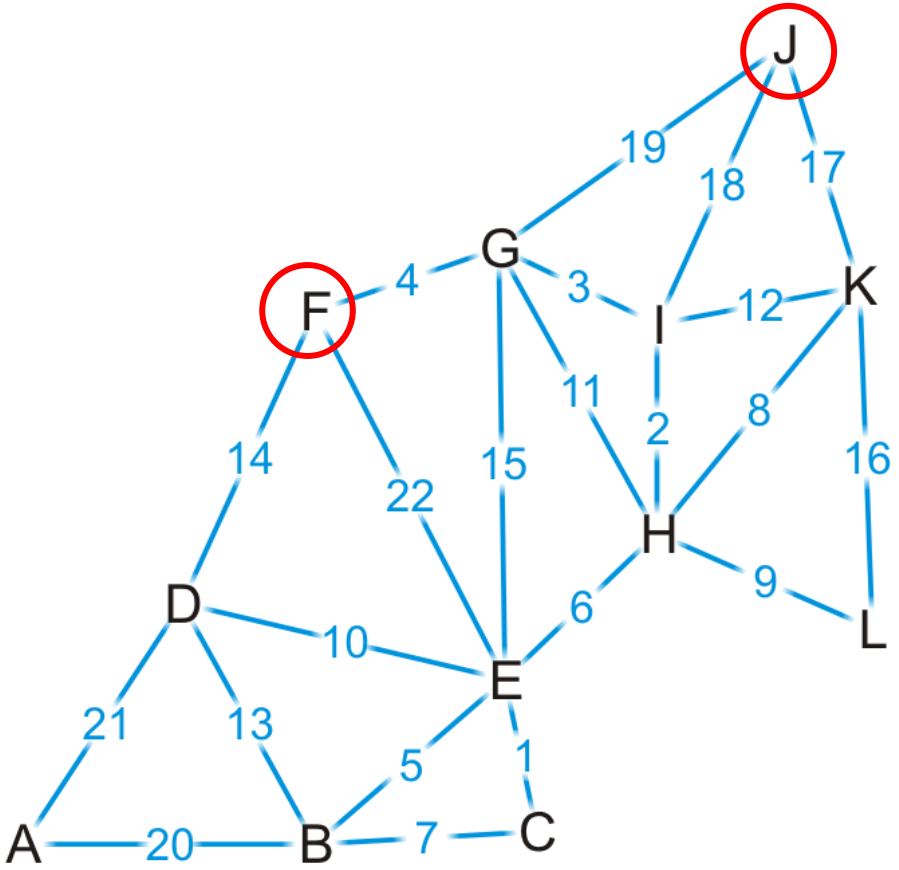
- We now know that (K, L) is the shortest path between these two points
  - Vertex L has no unvisited neighbors



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

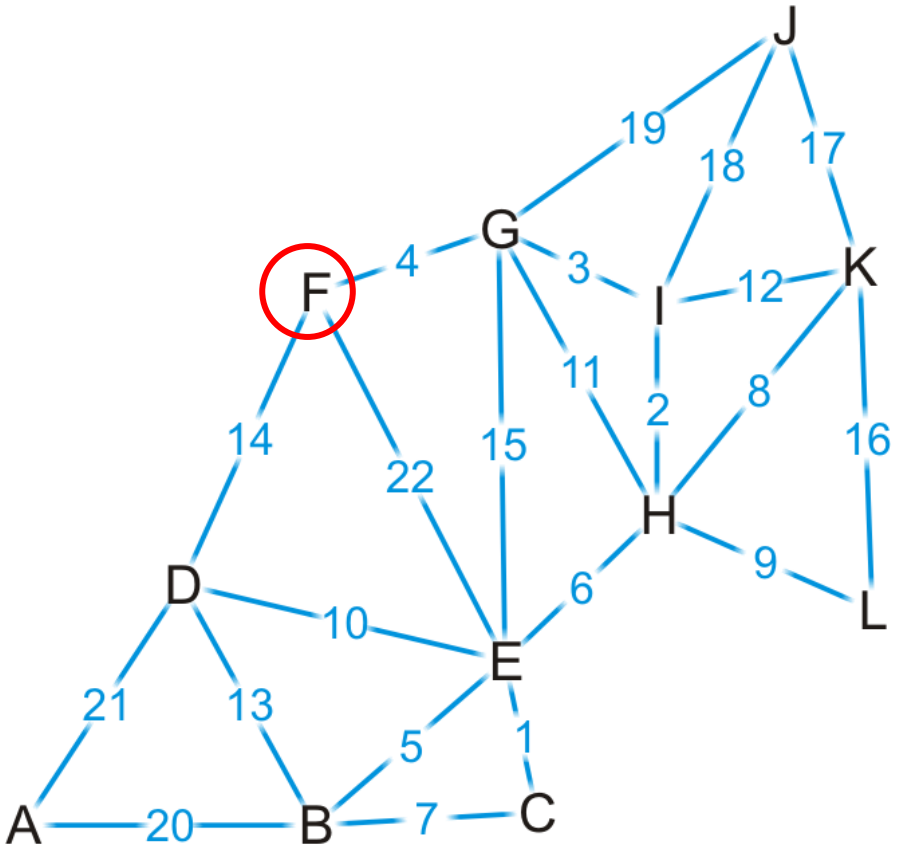
- Where to next?
  - Does it matter if we visit vertex F first or vertex J first?



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

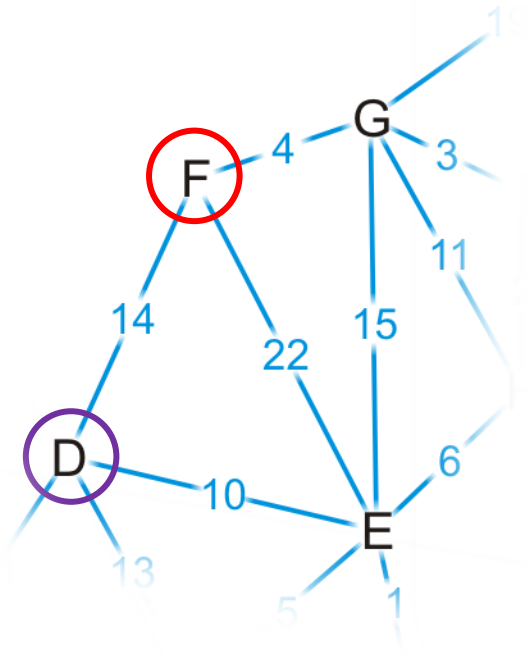
- Let's visit vertex F first
  - It has one unvisited neighbor, vertex D



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

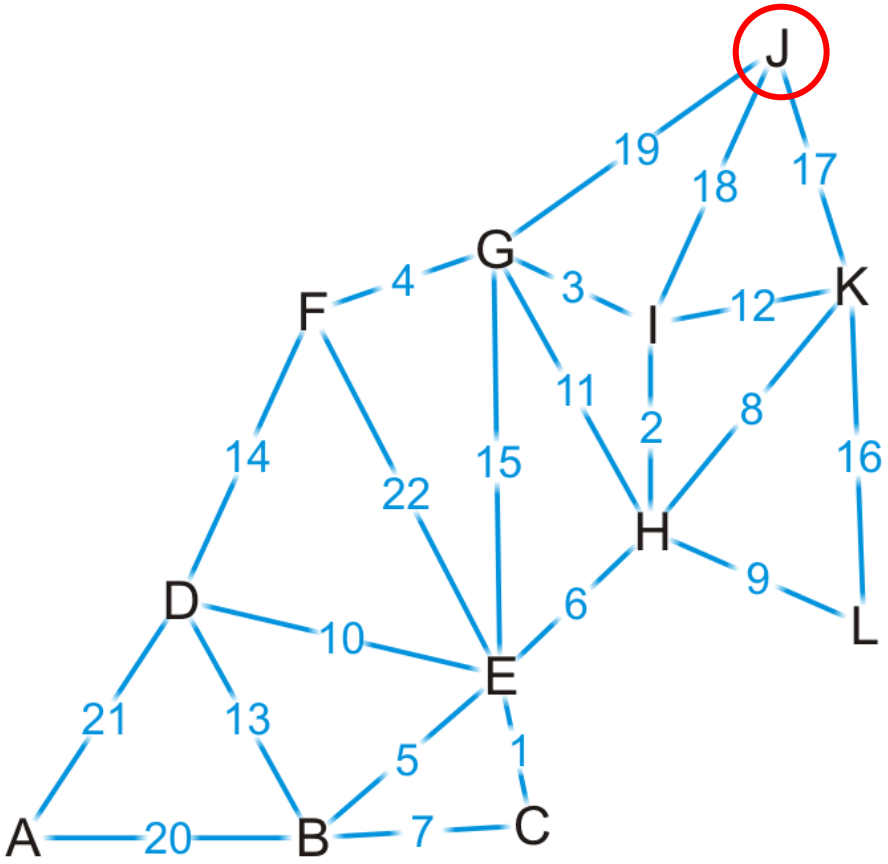
- The path (K, H, I, G, F, D) is of length 17 + 14 = 31
  - This is longer than the path we've already discovered



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

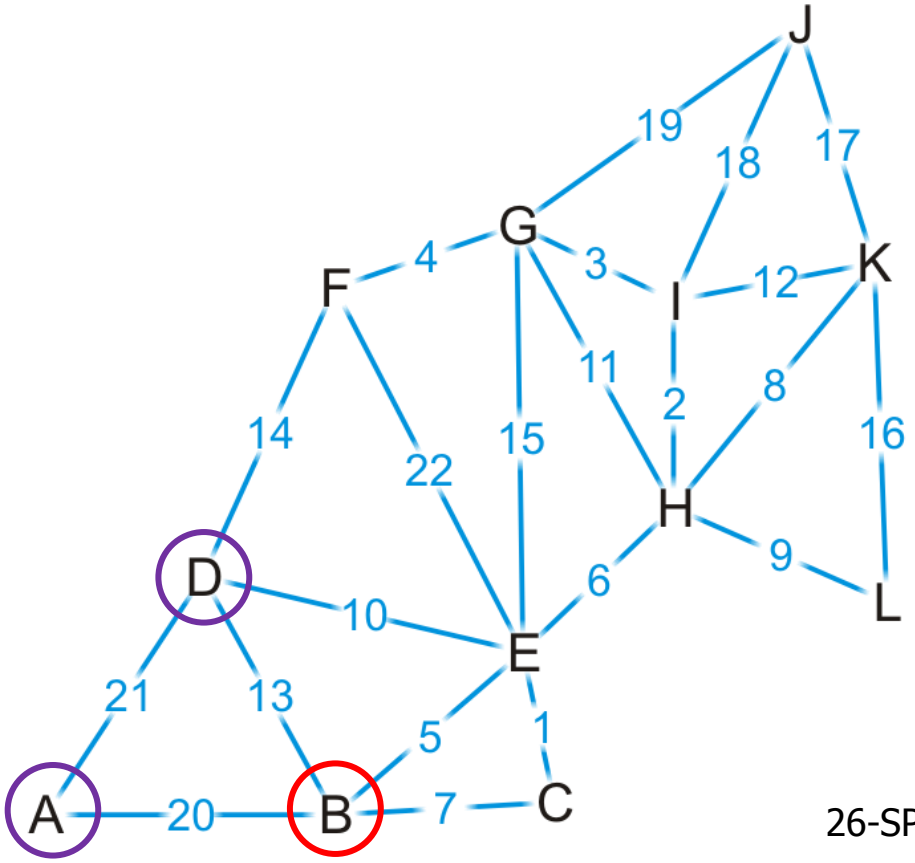
- Now we visit vertex J
  - It has no unvisited neighbors



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

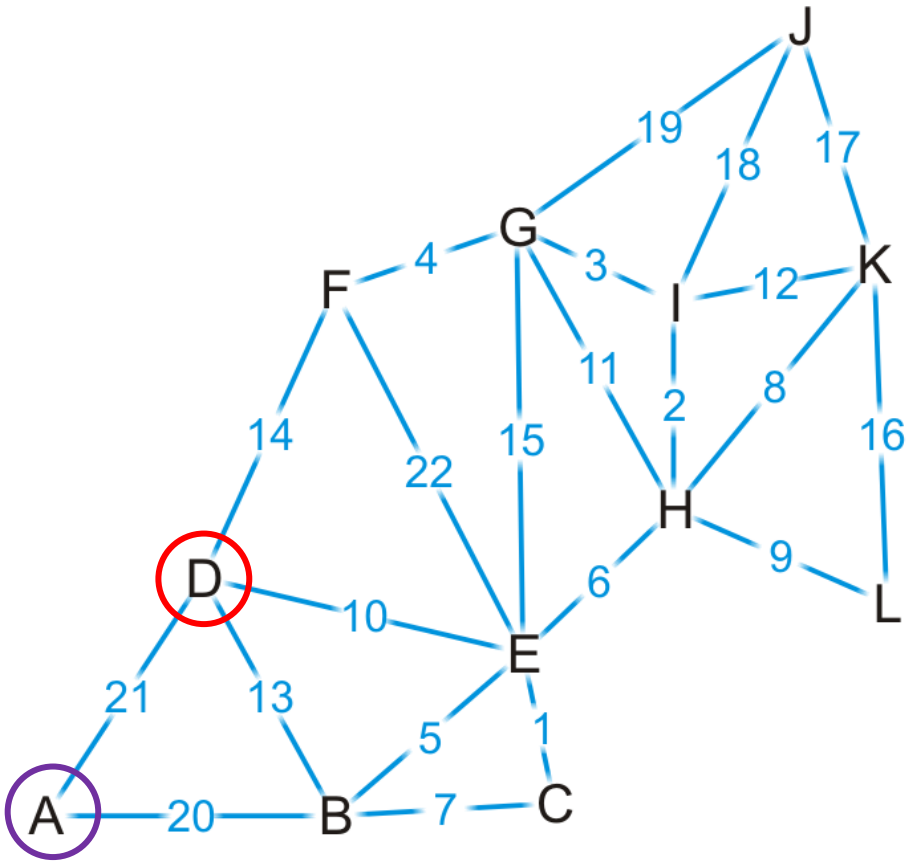
- Next we visit vertex B, which has two unvisited neighbors:
  - (K, H, E, B, A) of length 19 + 20 = 39
  - (K, H, E, B, D) of length 19 + 13 = 32
- We update the path length to A



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

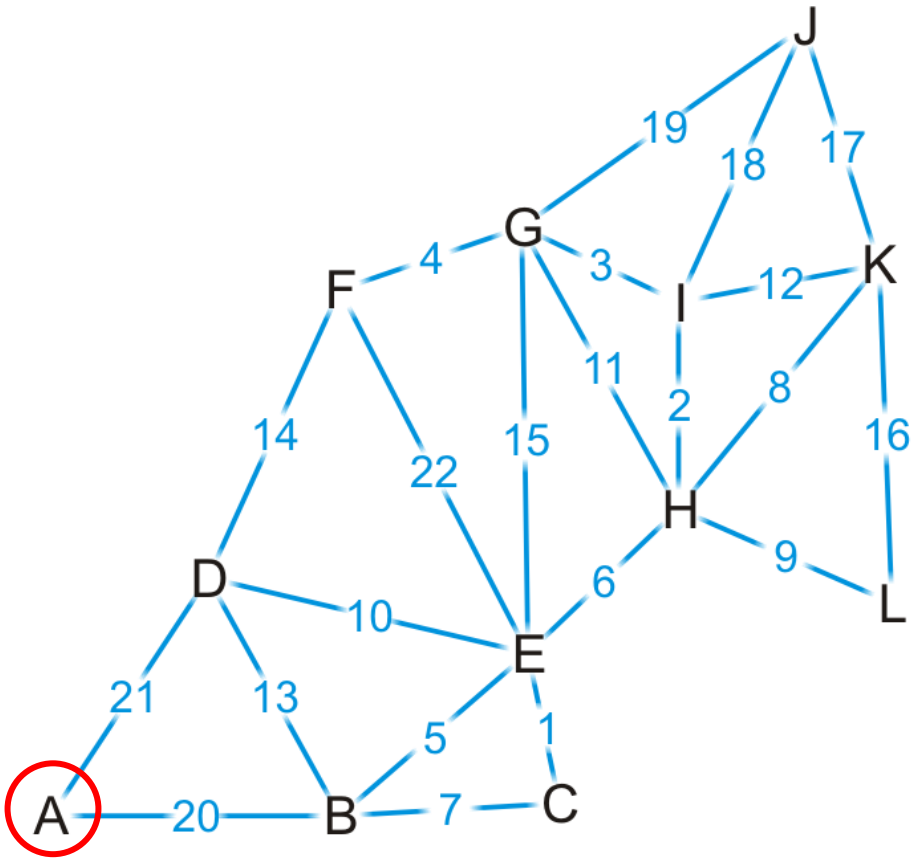
- Next we visit vertex D
  - The path (K, H, E, D, A) is of length 24 + 21 = 45
  - We don't update A



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

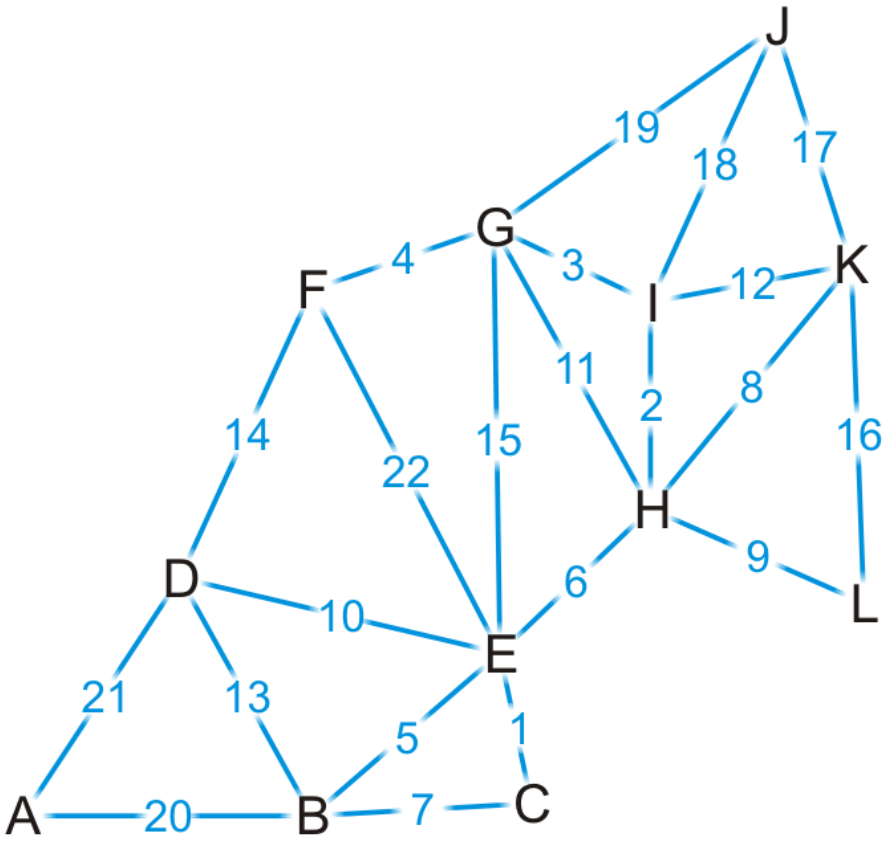
- Finally, we visit vertex A
  - It has no unvisited neighbors and there are no unvisited vertices left
  - We are done



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

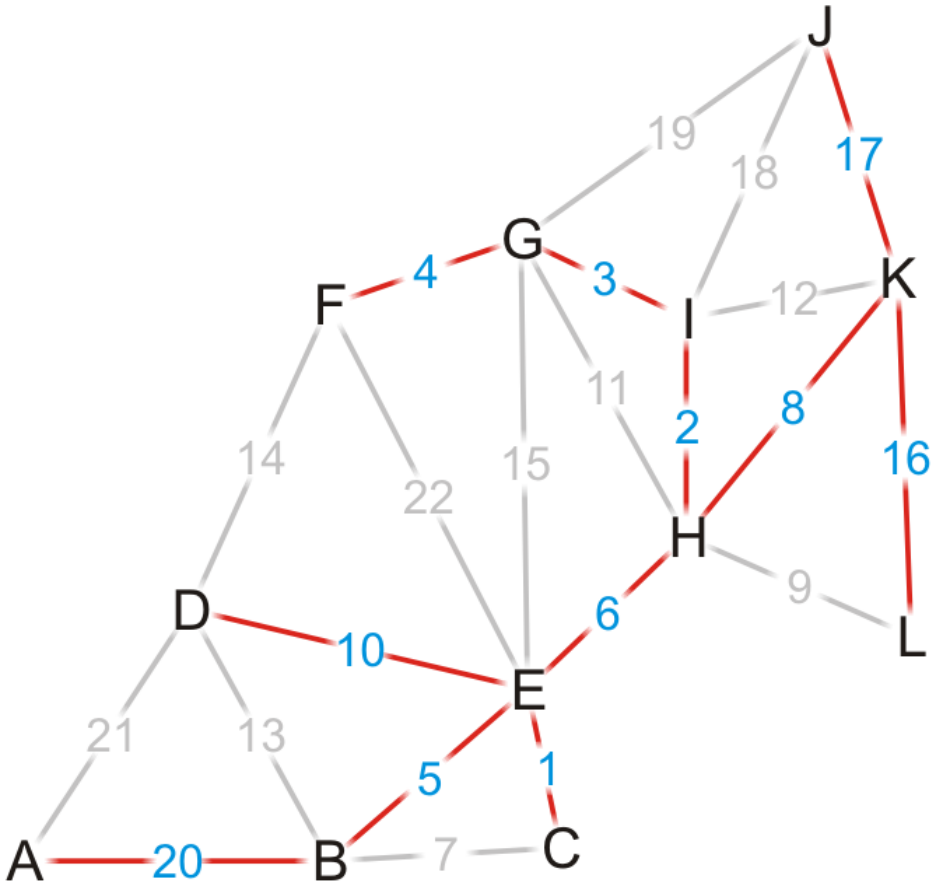
- Thus, we have found the shortest path from vertex K to each of the other vertices



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

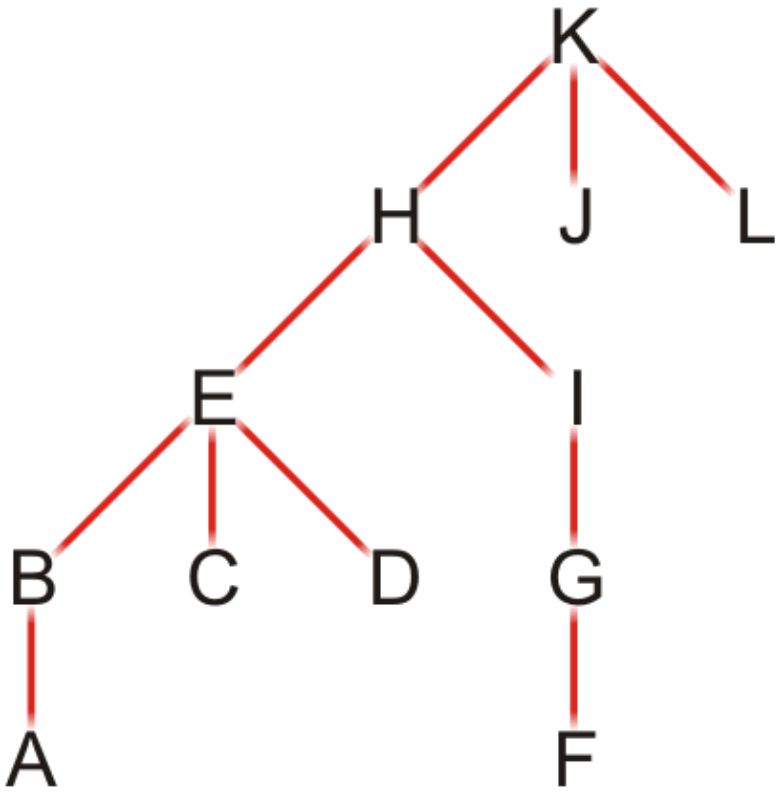
- Using the previous pointers, we can reconstruct the paths



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

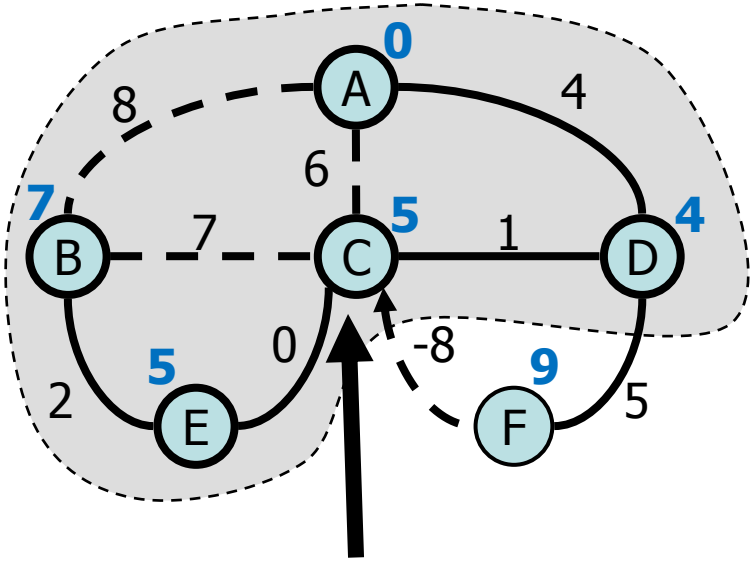
- Does the algorithm change if graph is directed?
  - No

Negative Edges

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

- If a node with a negative incident edge is visited late
  - It could mess up distances for vertices already in the set S

S: {A, C, E, D}



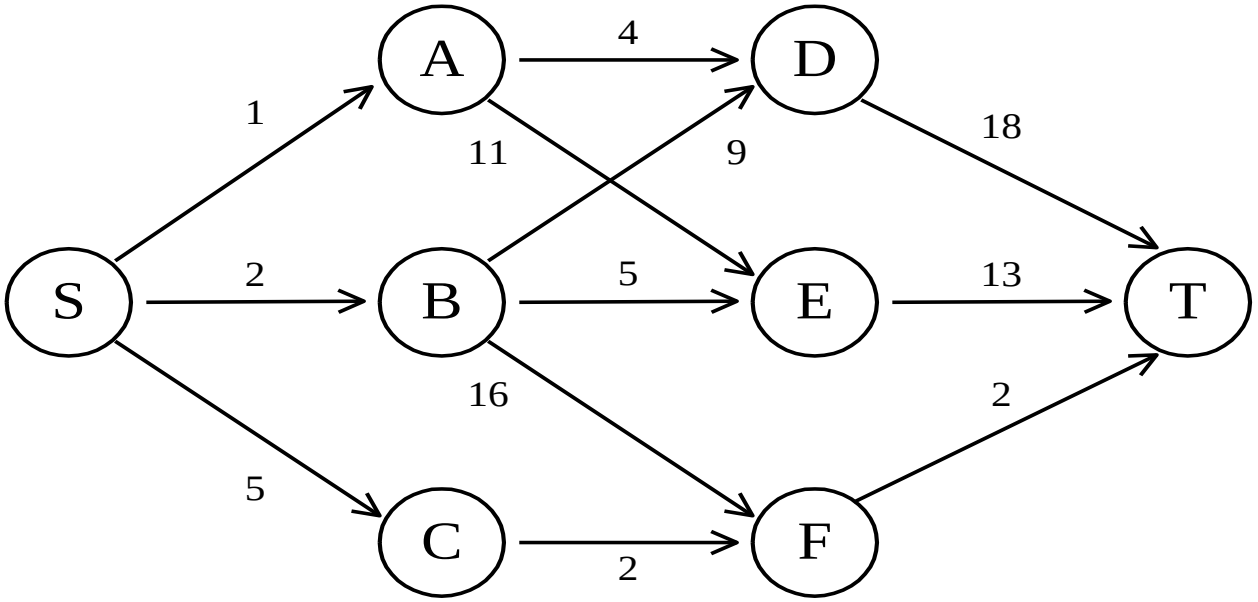
C's true distance is 1, but it is already in the set S with `dist(C)=5!`

All Pairs Shortest Path

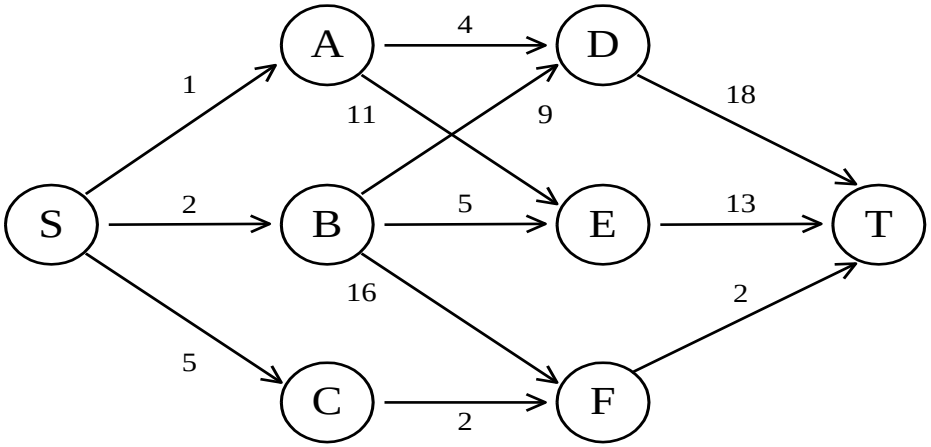
Dynamic Programming

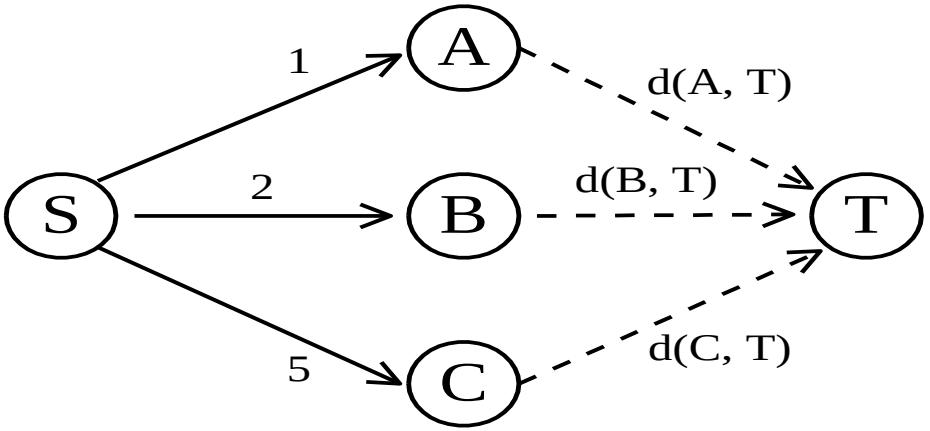
- Algorithm design method that can be used when the solution to a problem may be viewed as the result of a sequence of decisions

- Example: How to find shortest path from S to T?

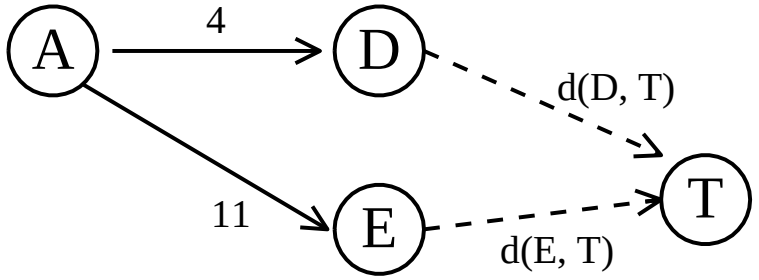


Dynamic Programming – Backward Reasoning



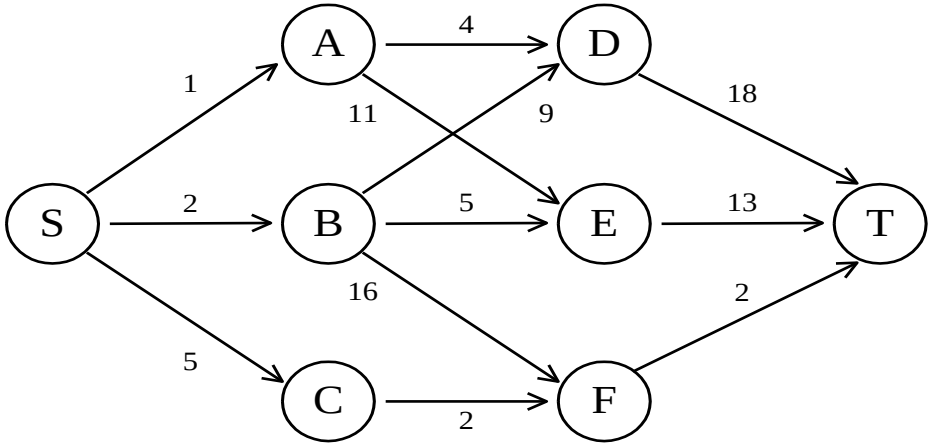


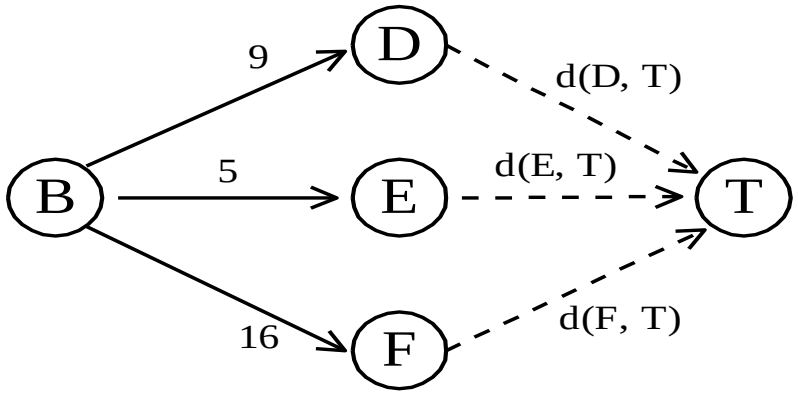
- d(S, T) = min{1+d(A, T), 2+d(B, T), 5+d(C, T)}
- d(A,T) = min{4+d(D,T), 11+d(E,T)} = min{4+18, 11+13} = 22



Dynamic Programming – Backward Reasoning

- d(B, T) = min{9+d(D, T), 5+d(E, T), 16+d(F, T)}
  = min{9+18, 5+13, 16+2} = 18

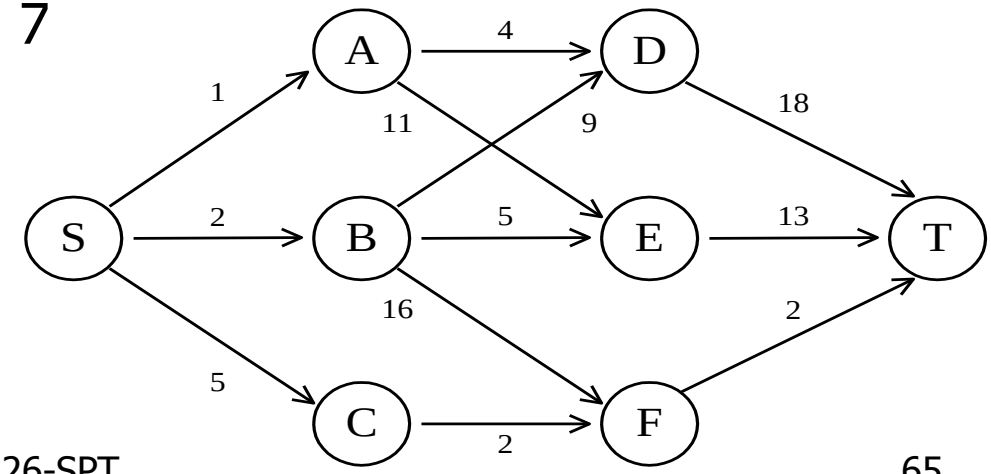




- d(C, T) = min{ 2+d(F, T) } = 2+2 = 4
- d(S, T) = min{1+d(A, T), 2+d(B, T), 5+d(C, T)}
  = min{1+22, 2+18, 5+4} = 9

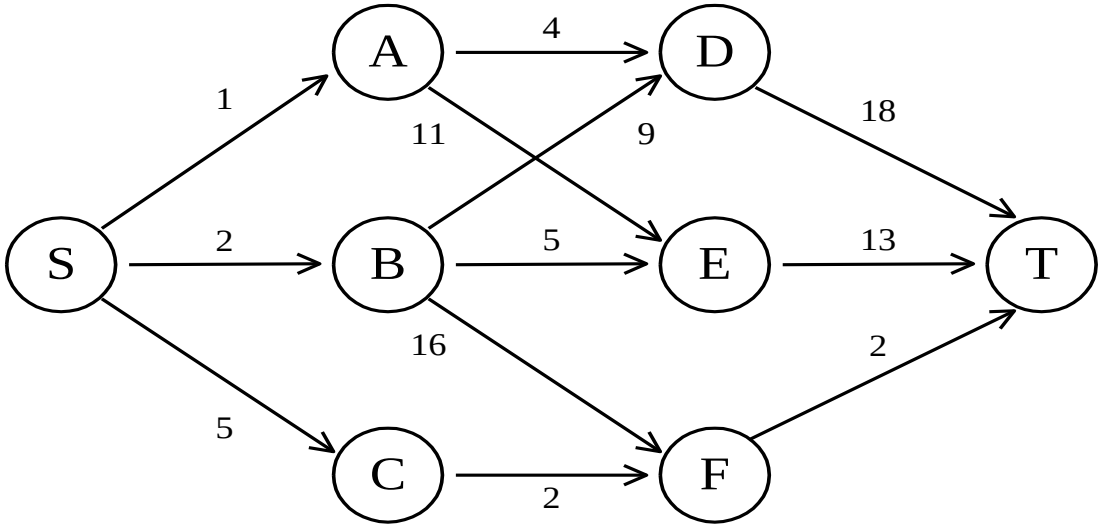
Dynamic Programming – Forward Reasoning

- d(S, A) = 1
- d(S, B) = 2
- d(S, C) = 5
- d(S,D) = min{d(S,A)+d(A,D), d(S,B)+d(B,D)}
  = min{ 1+4, 2+9 } = 5
- d(S,E) = min{d(S,A)+d(A,E), d(S,B)+d(B,E)}
  = min{ 1+11, 2+5 } = 7
- d(S,F) = min{d(S,B)+d(B,F), d(S,C)+d(C,F)}
  = min{ 2+16, 5+2 } = 7



Dynamic Programming – Forward Reasoning

- d(S,T) = min{d(S, D)+d(D, T), d(S,E) + d(E,T), d(S, F)+d(F, T)}
  = min{ 5+18, 7+13, 7+2 }
  = 9



All Pairs Shortest Path

- Find the shortest path between every pair of vertices of a graph
- Graph may contain negative edges but no negative cycles
- Can we use Dijkstra's algorithm for this purpose? How?
  - Set each vertex as source and call Dijkstra's algorithm
- We will discuss an algorithm By Floyd

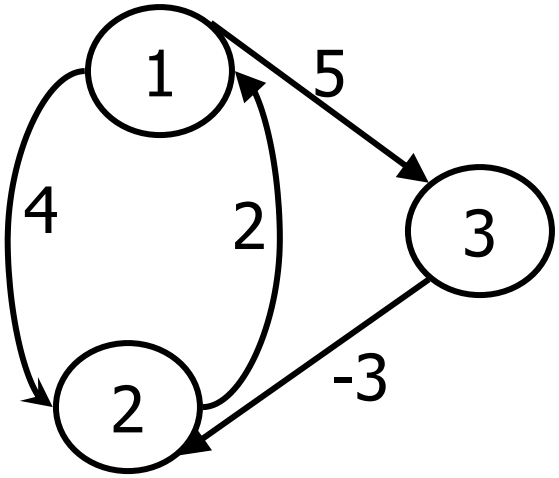
Floyd's Algorithm – Basics and Notation

- Assume vertices are numbered as `1,2,3,…,n=|V|` for simplicity
- Uses `n×n` matrix `D` (`n` is the number of vertices in the graph `G`)
- Shortest paths are computed in matrix `D`
- We make n=|V| iterations over matrix D
  - D^k denotes the value of the matrix after k^{th} iterations
  - D^k`[i,j]` indicates minimum weight path between vertices `i` and `j`
    - That does not pass through a vertex numbered higher than `k`

Initialization

- D^0 considers only edges that connect vertices directly
  - `D⁰[i,j] = 0` `if i==j`
  - `D⁰[i,j] = w(i,j)` `if there is an edge from i to j`
  - `D⁰[i,j] = ∞` `Otherwise`

- In C++, we would define a two-dimensional array
  - `double D[num_vertices][num_vertices];`



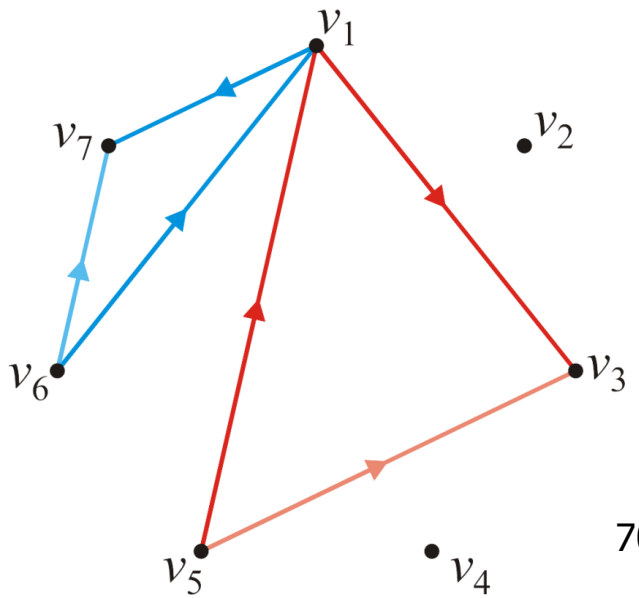
$W = D^0 =$

	1	2	3
1	0	4	5
2	2	0	∝
3	∝	-3	0

First Iteration

- For each pair of vertices, we will define $D^1[i,j]$ by calculating
  - $D^1[i,j] = \min(D^0[i,j], D^0[i,1] + D^0[1,j])$

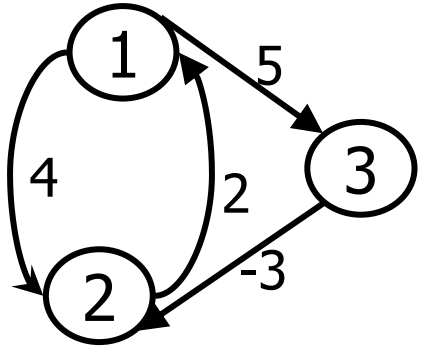
```
for ( int i = 0; i < num_vertices; ++i ) {
    for ( int j = 0; j < num_vertices; ++j ) {
        D[i][j] = std::min( D[i][j], D[i][0] + D[0][j] );
    }
}
```



First Iteration – Example

$D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0



Vertex 1 can be intermediate node

$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$D^1[2,3] = \min(D^0[2,3], D^0[2,1]+D^0[1,3])$
$= \min (\infty, 7)$
$= 7$

$D^1[3,2] = \min(D^0[3,2], D^0[3,1]+D^0[1,2])$
$= \min (-3,\infty)$
$= -3$

Second Iteration

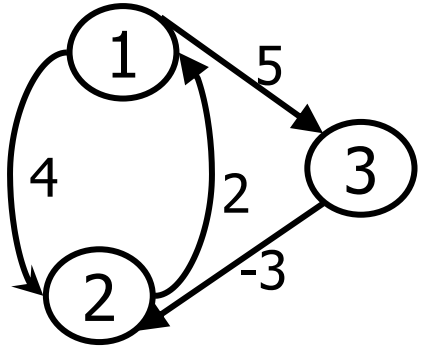
- With vertex `1` as intermediate we have found the shortest paths from `i` to `j`, `i` to `2` and `2` to `j`
  - The only possible shorter path including vertex `2` would be the path from `i` to `2` continuing from there to `j`

- Thus, calculate
  - $D^2[i,j] = \min (D^1[i,j], D^1[i,2] + D^1[2,j])$

Second Iteration – Example

$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0



Vertex 1, 2 can be intermediate nodes

$D^2 =$

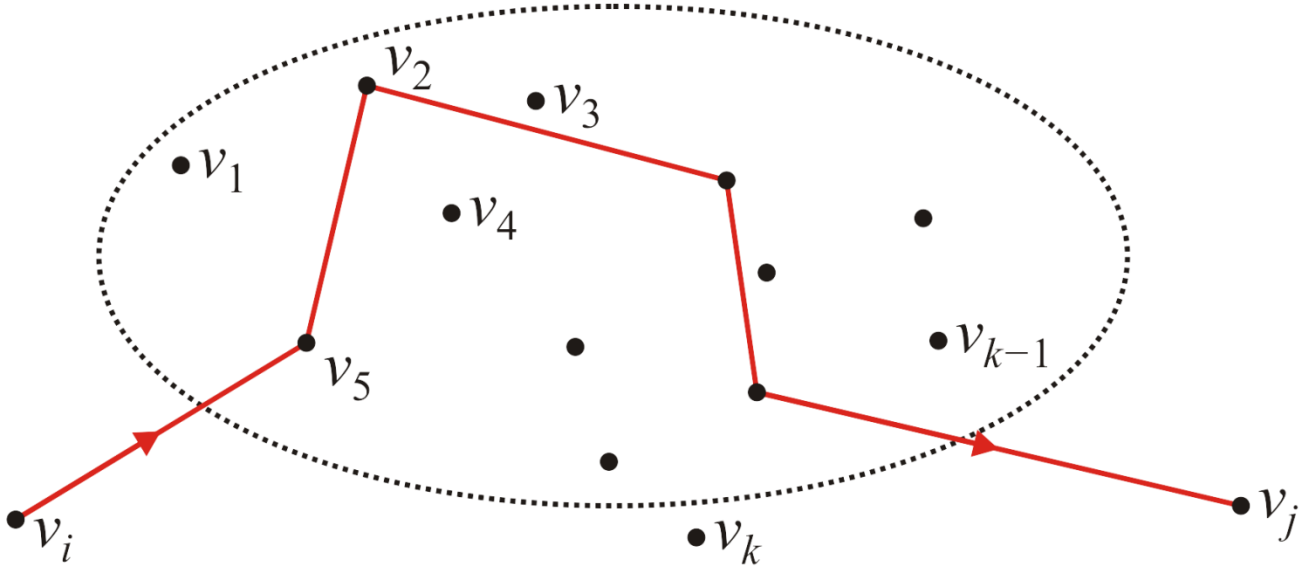
	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$D^2[1,3] = \min(D^1[1,3], D^1[1,2]+D^1[2,3])$
$= \min (5, 4+7)$
$= 5$

$D^2[3,1] = \min(D^1[3,1], D^1[3,2]+D^1[2,1])$
$= \min (\infty, -3+2)$
$= -1$

Generalization to Kth Iteration

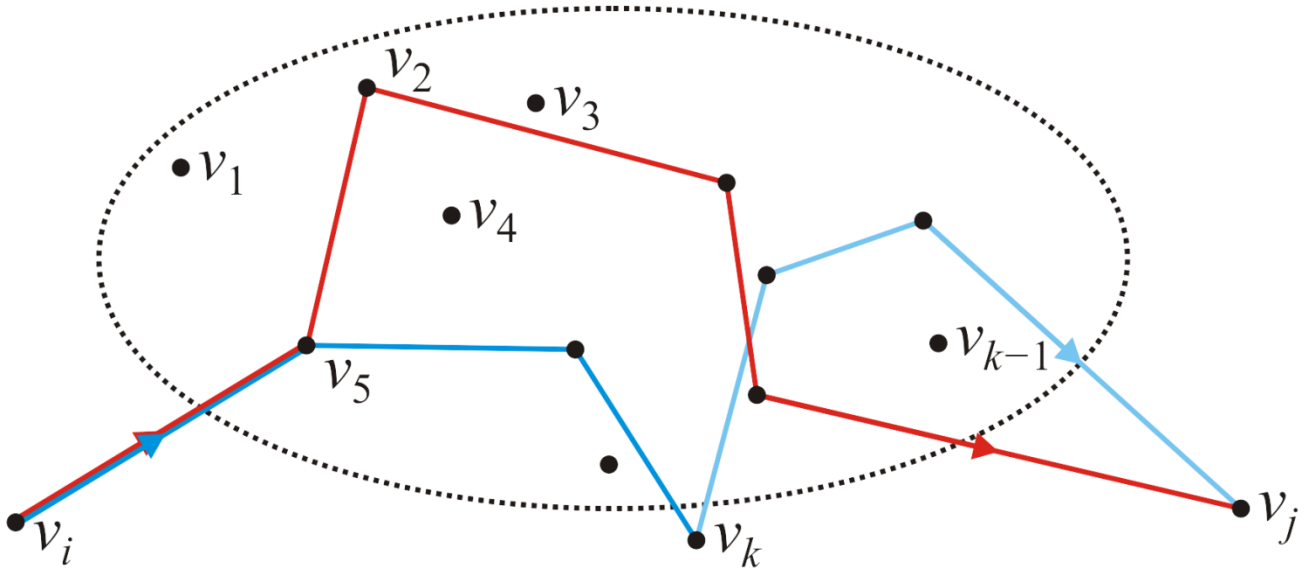
- With vertices `1, 2, … k-1` as intermediate we have found the shortest paths from `i` to `j`, `i` to `k` and `k` to `j`
  - The only possible shorter path including vertex `k` would be the path from `i` to `k` continuing from there to `j`



Generalization to Kth Iteration

- With vertices `1, 2, … k-1` as intermediate we have found the shortest paths from `i` to `j`, `i` to `k` and `k` to `j`
  - The only possible shorter path including vertex `k` would be the path from `i` to `k` continuing from there to `j`

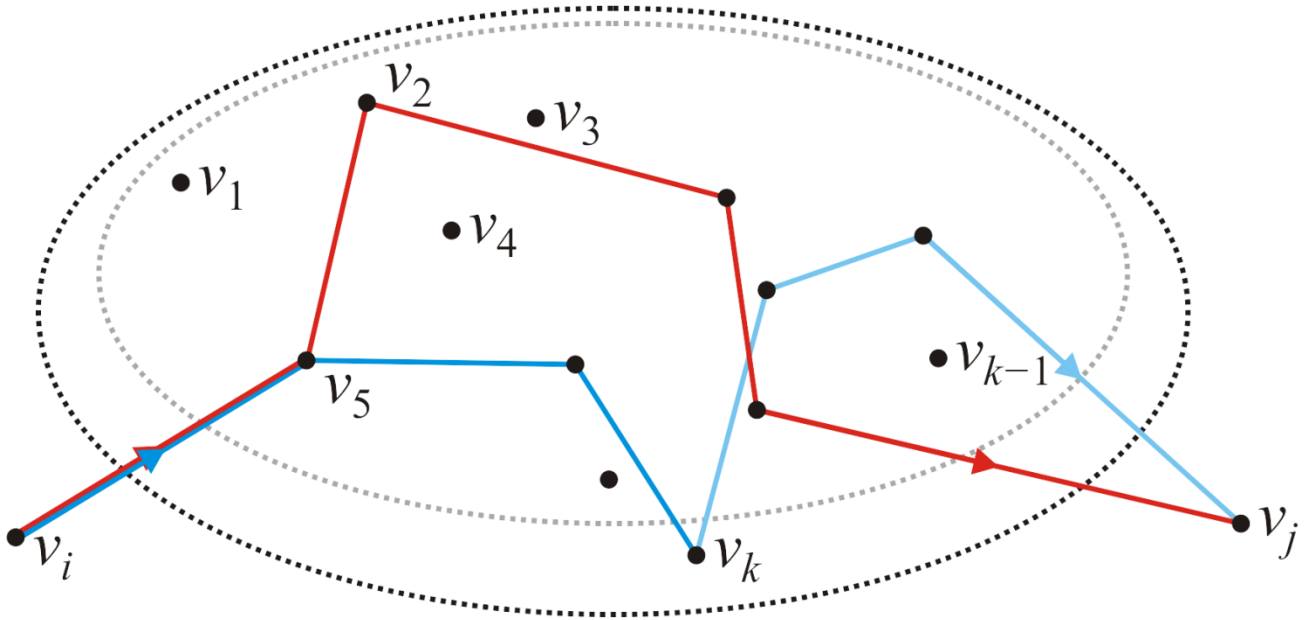
- Calculate: $D^k[i,j] = \min(D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j])$



Generalization to Kth Iteration

- With vertices `1, 2, … k-1` as intermediate we have found the shortest paths from `i` to `j`, `i` to `k` and `k` to `j`
  - The only possible shorter path including vertex `k` would be the path from `i` to `k` continuing from there to `j`

- Calculate: $D^k[i,j] = \min (D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j])$



Floyd's Algorithm – Pseudocode

```
double D[num_vertices][num_vertices];

// Initialize the matrix d
//   ...

// Run Floyd-Algorithm
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            D[i][j] = std::min( D[i][j], D[i][k] + D[k][j] );
        }
    }
}
```

Floyd's Algorithm – Pseudocode

```
double D[num_vertices][num_vertices];

// Initialize the matrix d
//   ...

// Run Floyd-Algorithm
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            D[i][j] = std::min( D[i][j], D[i][k] + D[k][j] );
        }
    }
}
```

```
if( D[i][j] > D[i][k] + D[k][j] ){
    D[i][j] = D[i][k] + D[k][j];
}
```

- Runtime $O(|V|^3)$
- This algorithm finds the shortest distances, but what are the paths corresponding to those shortest distances?

Tracking Shortest Paths

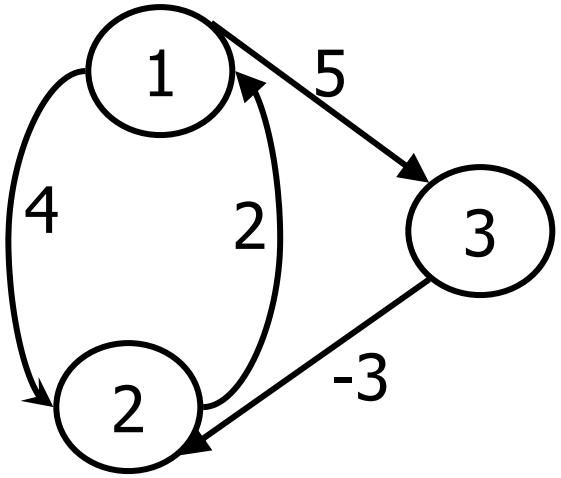
- Use another matrix `P`
- `P[i,j]` holds the vertex `k` that led to the smallest value of `D[i,j]`
- If `P[i,j]=0` there is no intermediate edge involved
  - Shortest path is direct edge between `i` and `j`

```
// Run Floyd-Algorithm
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            if( D[i][j] > D[i][k] + D[k][j] ){
                D[i][j] = D[i][k] + D[k][j];
                P[i][j] = k;
            }
        }
    }
}
```

Floyd's Algorithm – Example Revisited

$W = D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0



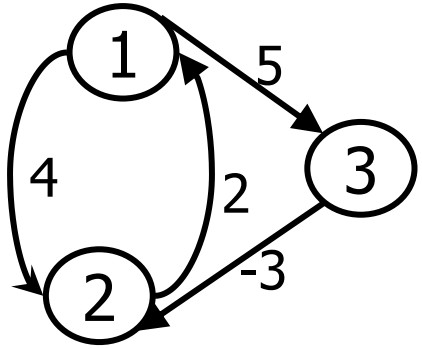
P =

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

Floyd's Algorithm – Example Revisited

$D^0 =$

	1	2	3
1	0	4	5
2	2	0	∝
3	∝	-3	0



$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∝	-3	0

Vertex 1 can be intermediate node

$D^1[2,3] = \min(D^0[2,3], D^0[2,1]+D^0[1,3])$
$= \min (\propto, 7)$
$= 7$

$D^1[3,2] = \min(D^0[3,2], D^0[3,1]+D^0[1,2])$
$= \min (-3,\propto)$
$= -3$

$P =$

	1	2	3
1	0	0	0
2	0	0	1
3	0	0	0

Floyd's Algorithm – Example Revisited

$D^1 =$

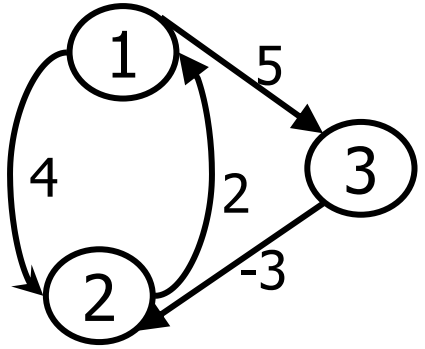
	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$P =$

	1	2	3
1	0	0	0
2	0	0	1
3	2	0	0



Vertex 1, 2 can be intermediate nodes

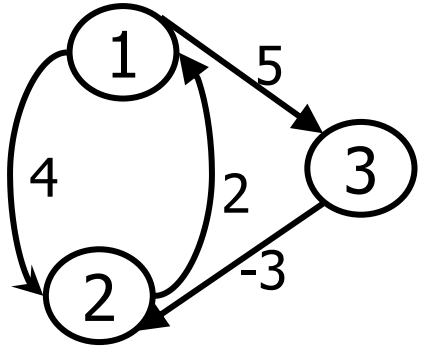
$D^2[1,3] = \min(D^1[1,3], D^1[1,2]+D^1[2,3])$
$= \min (5, 4+7)$
$= 5$

$D^2[3,1] = \min(D^1[3,1], D^1[3,2]+D^1[2,1])$
$= \min (\infty, -3+2)$
$= -1$

Floyd's Algorithm – Example Revisited

$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0



$D^3 =$

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

Vertex 1, 2, 3 can be intermediate nodes

$D^3[1,2] = \min(D^2[1,2], D^2[1,3]+D^2[3,2])$
$= \min(4, 5+(-3))$
$= 2$

$D^3[2,1] = \min(D^2[2,1], D^2[2,3]+D^2[3,1])$
$= \min(2, 7+(-1))$
$= 2$

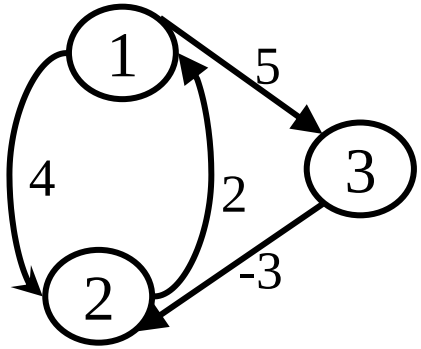
$P =$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

Printing Intermediate Vertices

- To print the intermediate nodes on shortest path from vertex i to j

```
Path (i, j)
{
     k= P[ i, j ];
     if (k == 0) {
          return;
     }
     Path (i , k);
     print( k);
     Path (k, j);
}
```



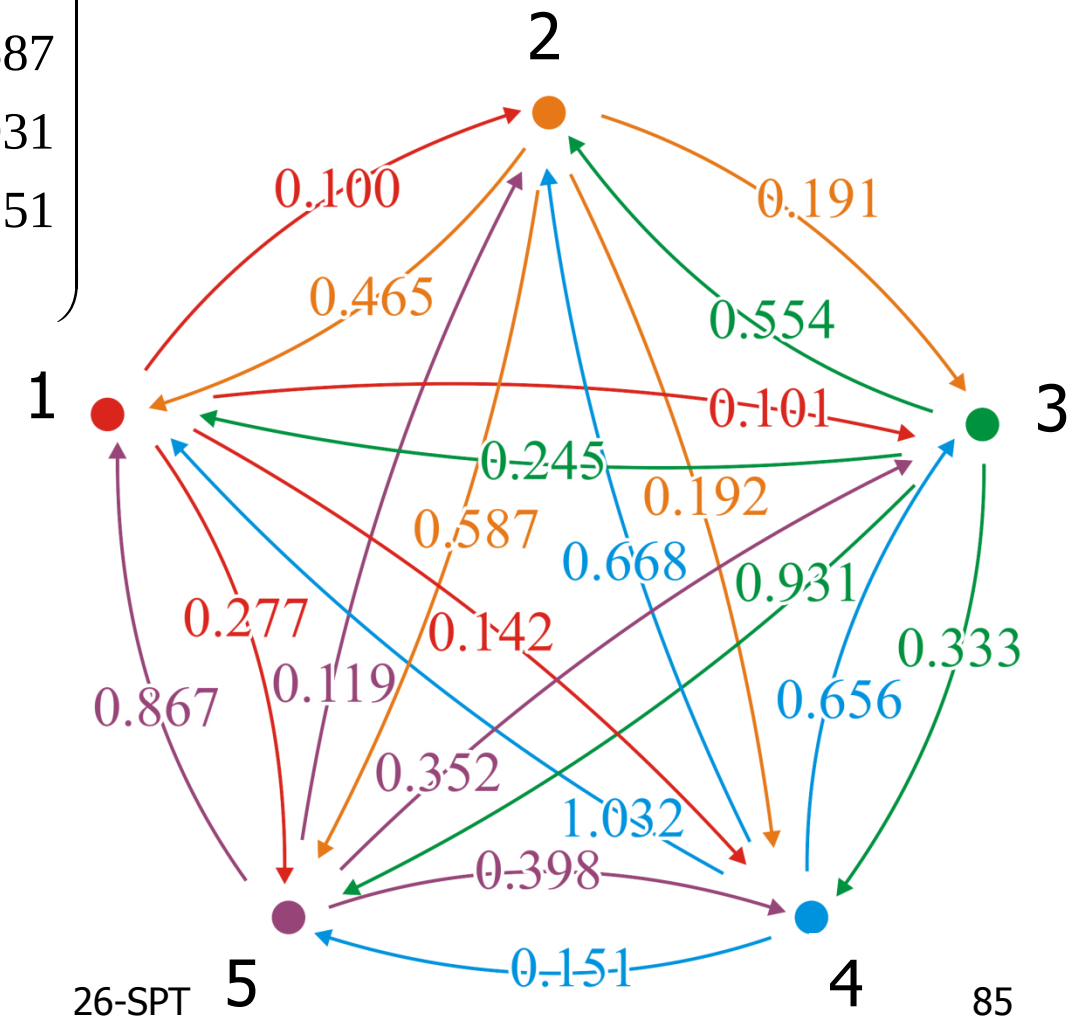
P =

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

Floyd's Algorithm – Another Example

The D^0 matrix

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

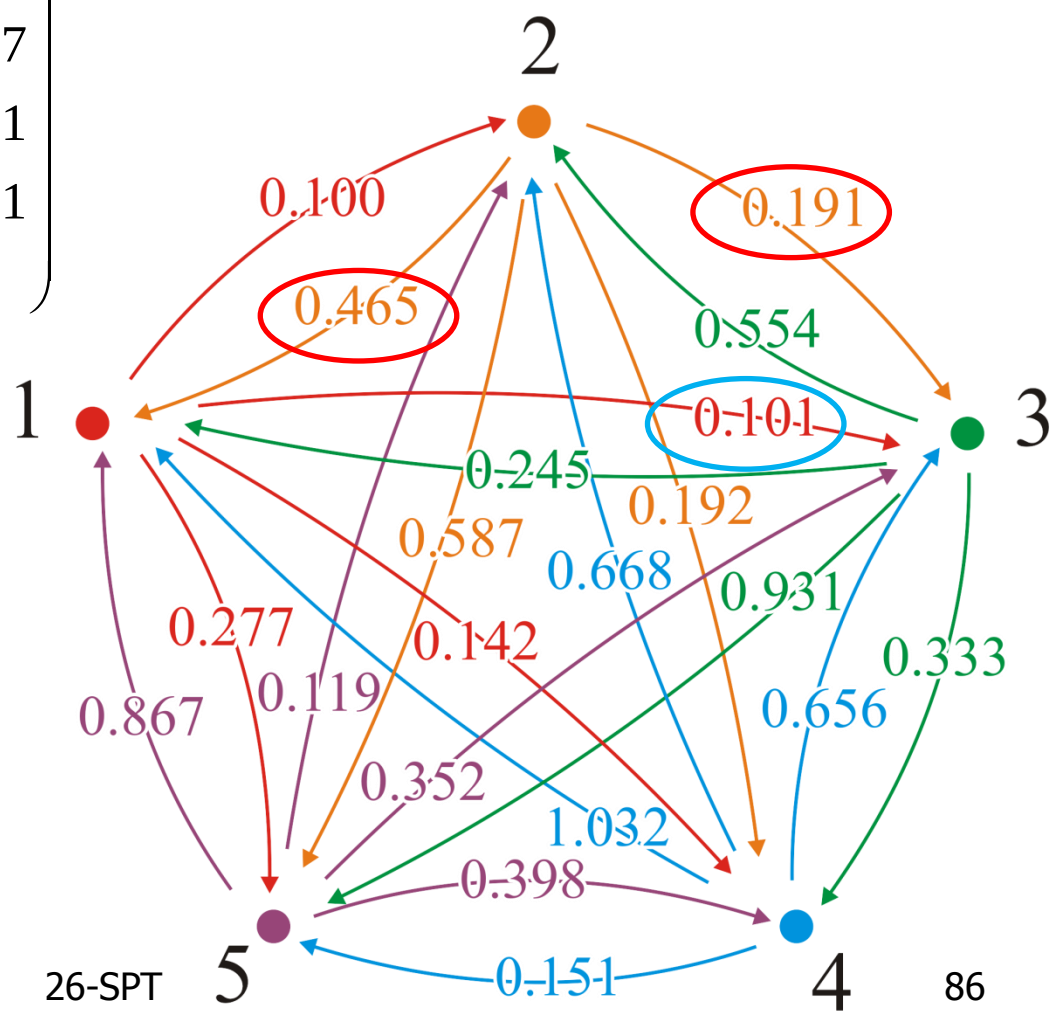


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

We can check:
(2, 3) → (2, 1, 3)
0.191 ≯ 0.465 + 0.101

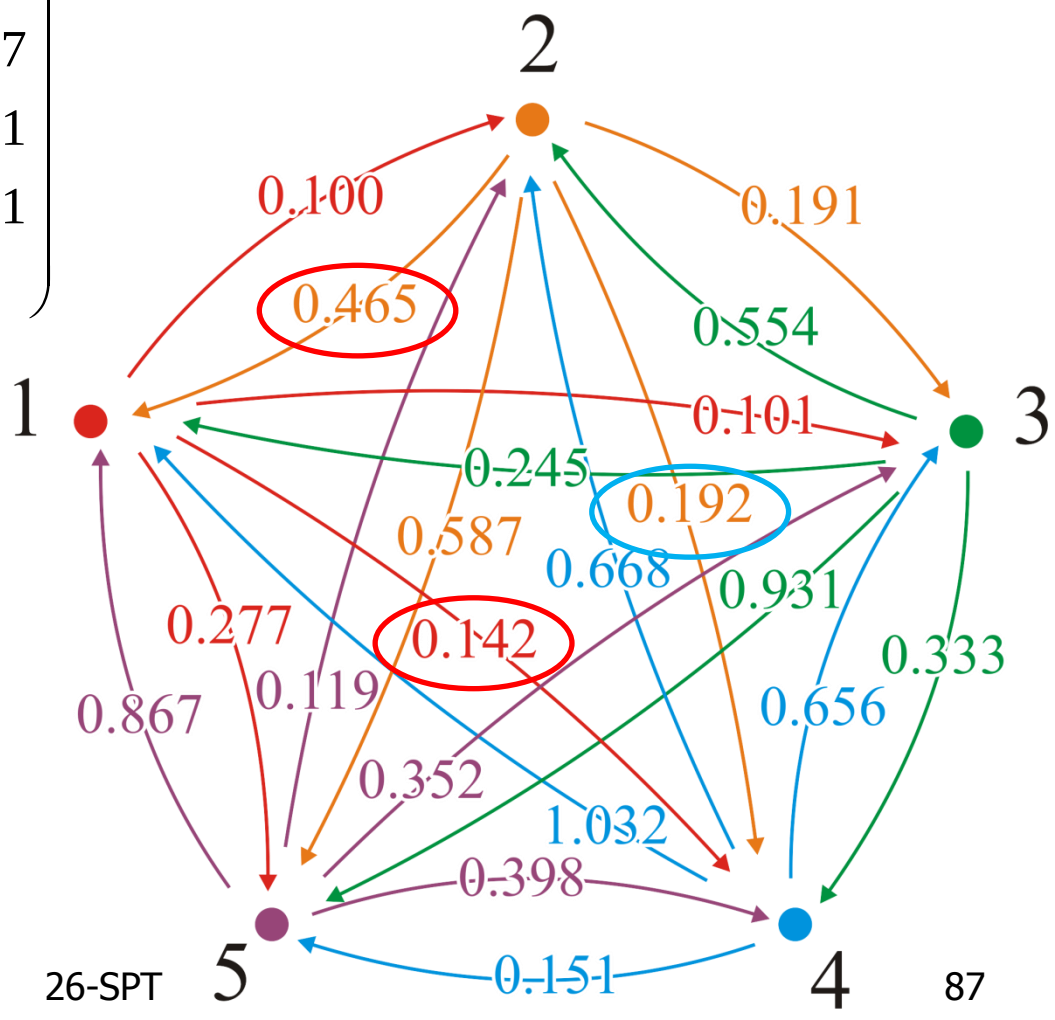


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

We can check:
(2, 4) → (2, 1, 4)
0.192 ≯ 0.465 + 0.142

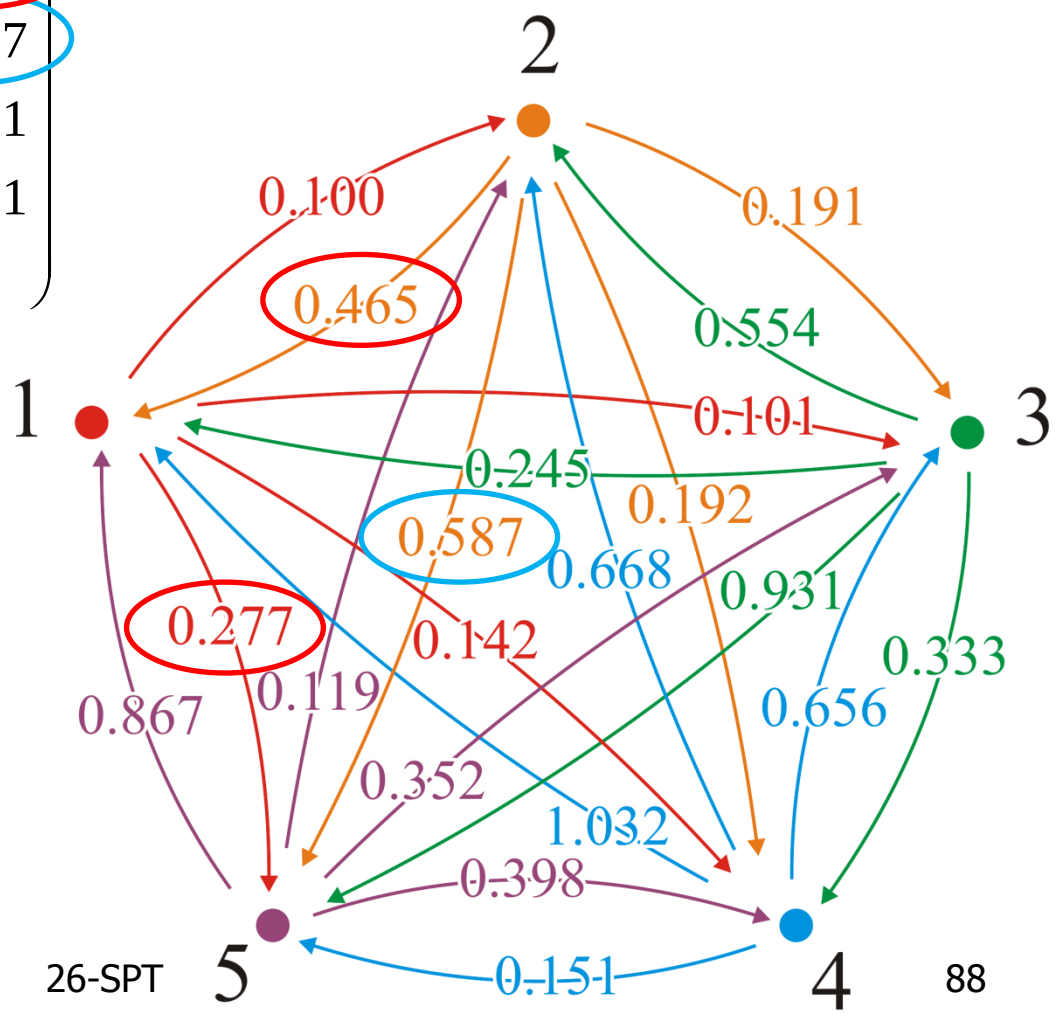


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

We can check:
(2, 5) → (2, 1, 5)
0.587 ≯ 0.465 + 0.277

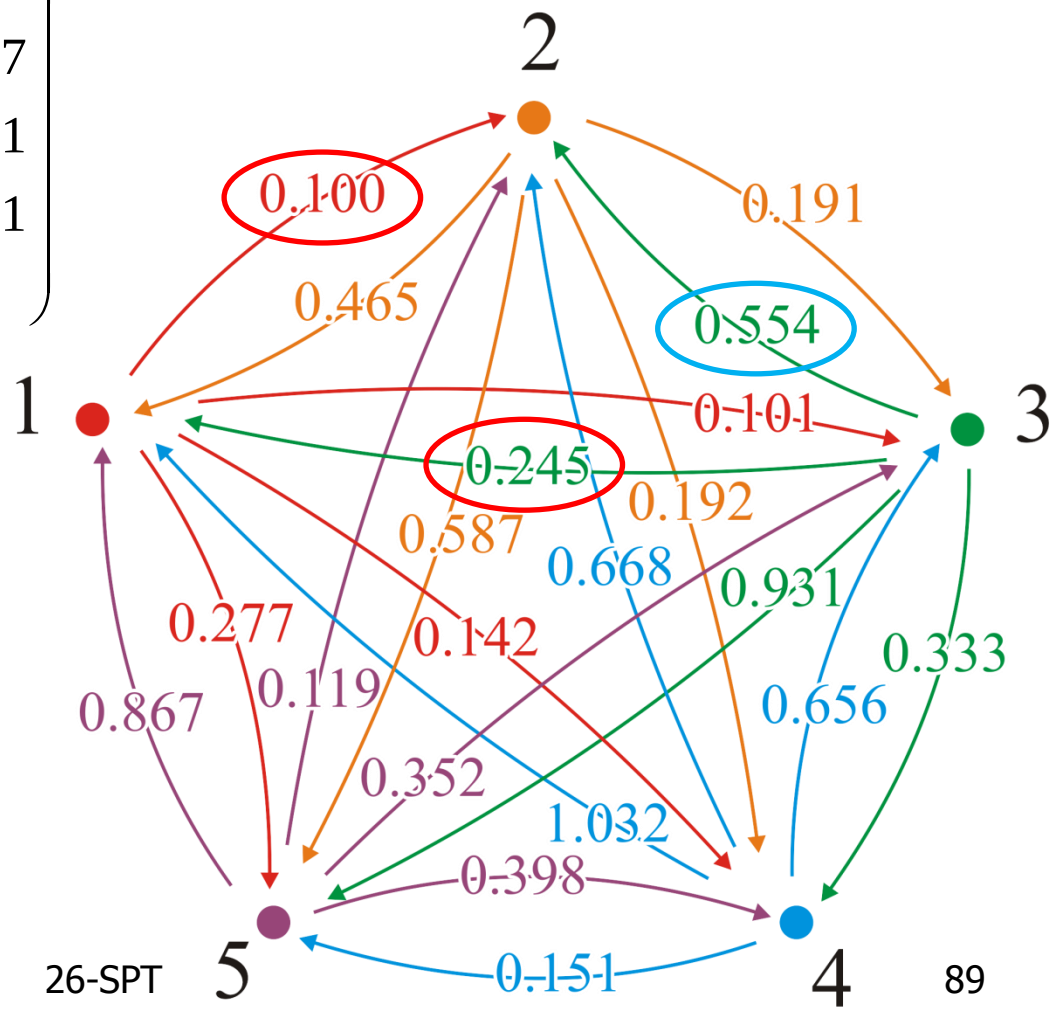


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Here is a shorter path:
(3, 2) → (3, 1, 2)
0.554 > 0.245 + 0.100 = 0.345



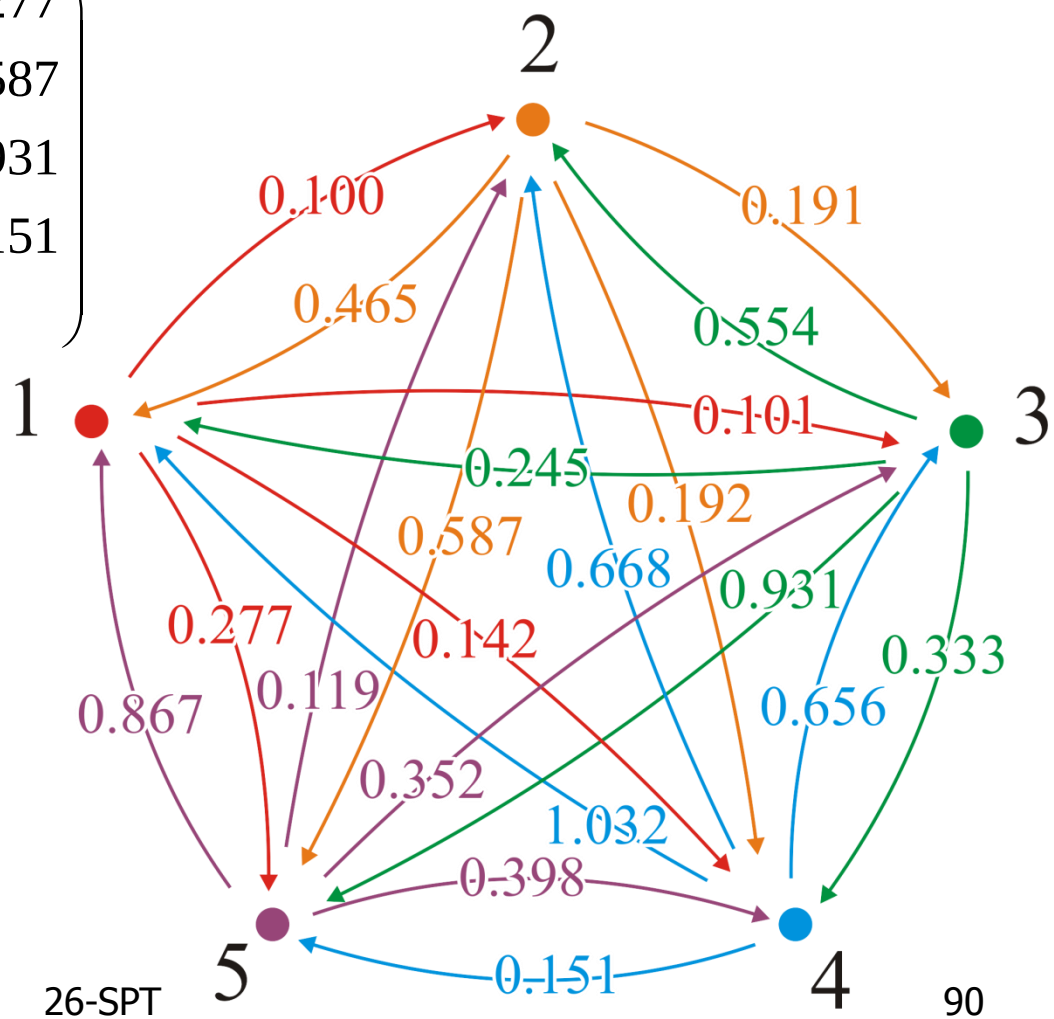
Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Here is a shorter path:
(3, 2) → (3, 1, 2)
0.554 > 0.245 + 0.100 = 0.345

We can update matrix!

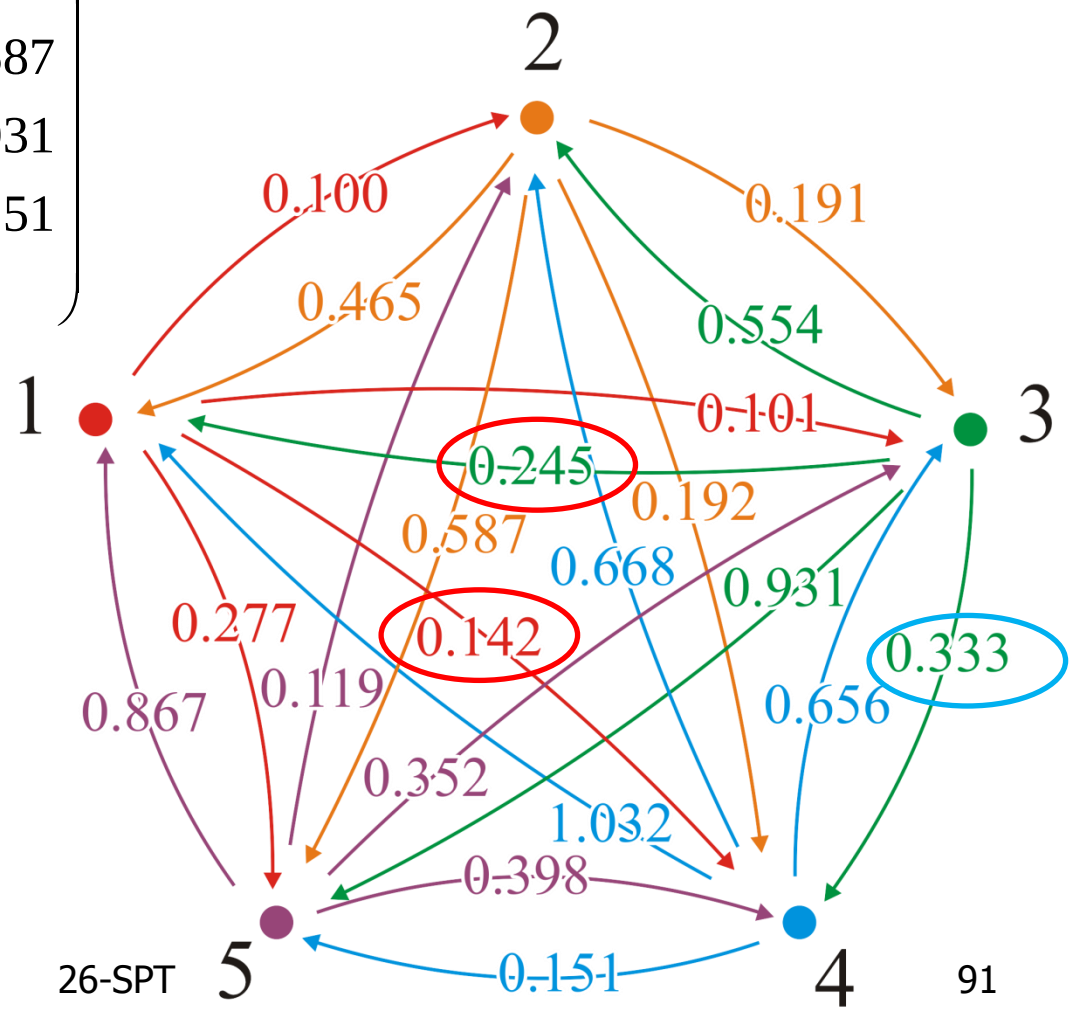


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

We can check:
(3, 4) → (3, 1, 4)
0.333 ≯ 0.245 + 0.142

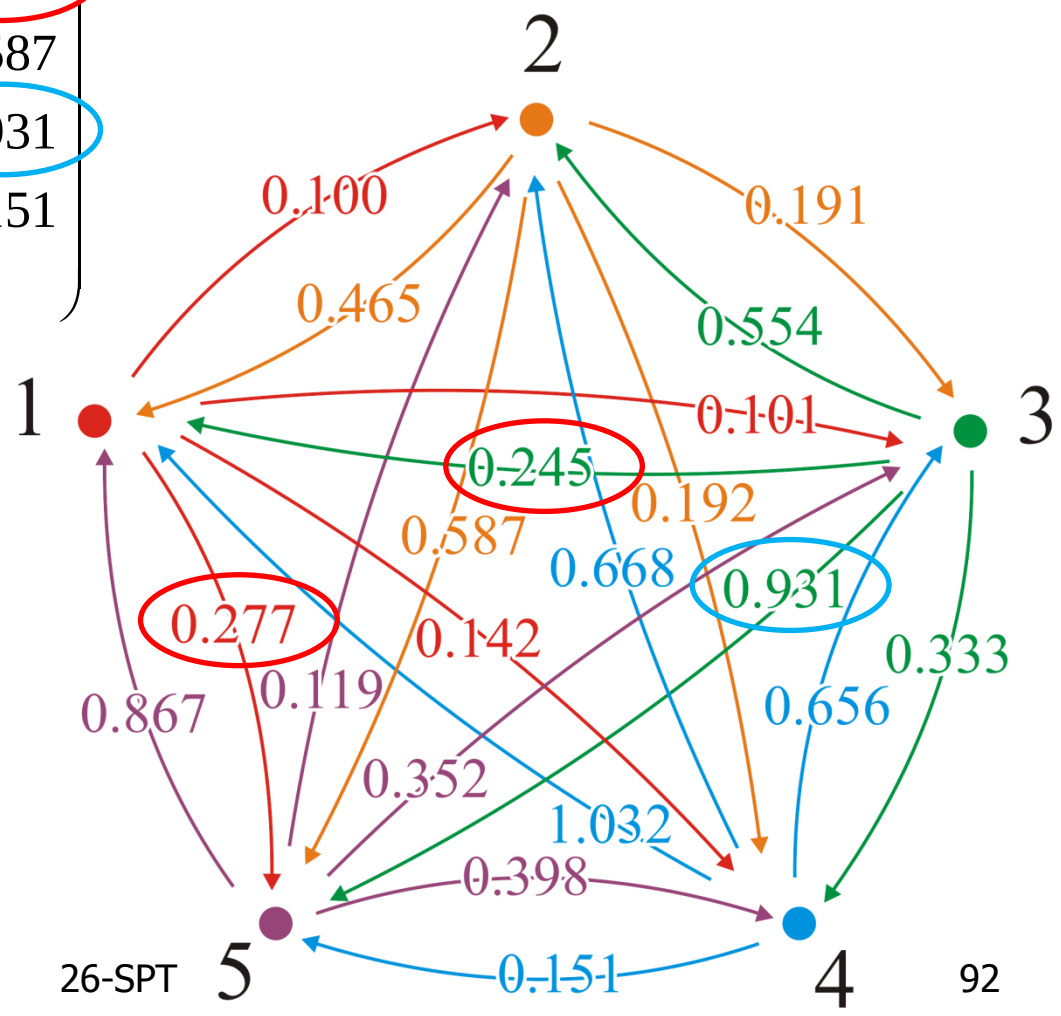


Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Another shorter path:
(3, 5) → (3, 1, 5)
0.931 > 0.245 + 0.277 = 0.522



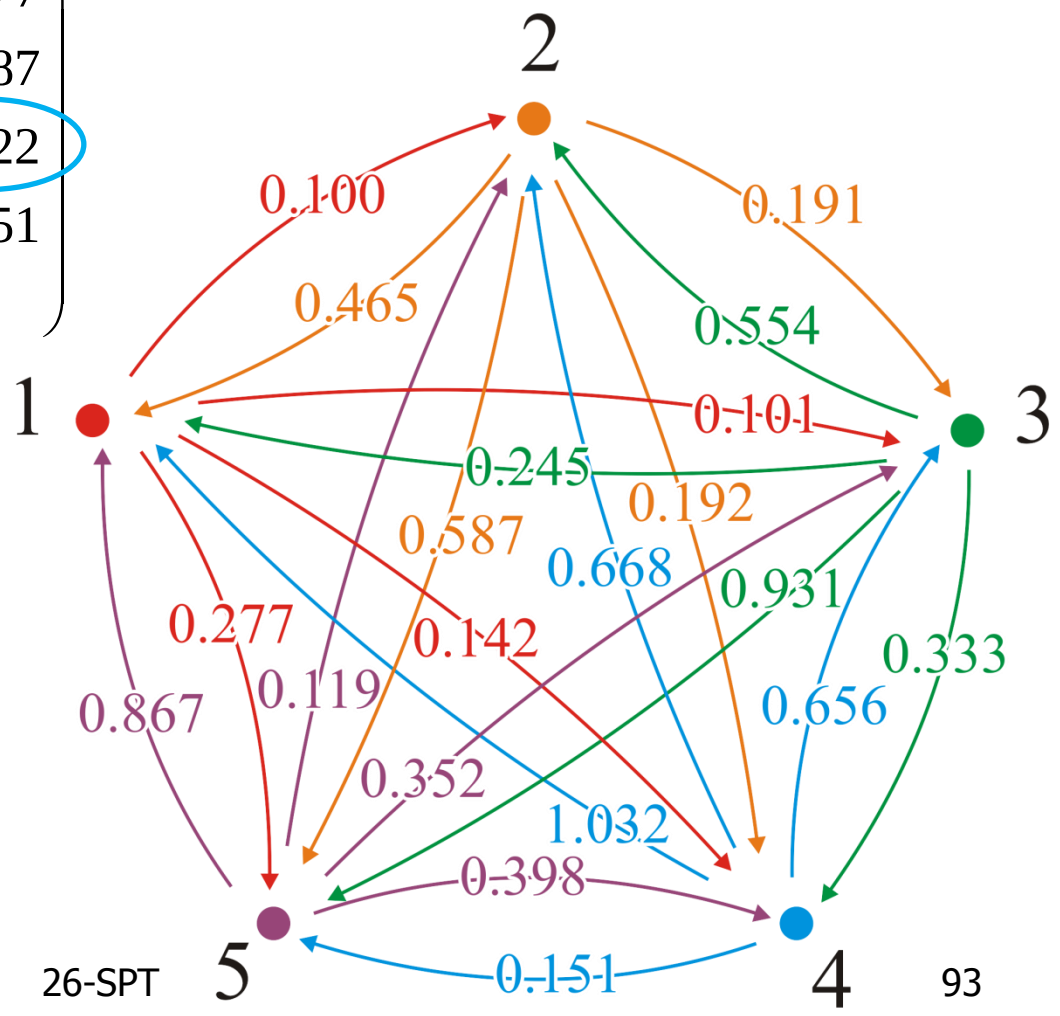
Floyd's Algorithm – Another Example

First attempt passing through vertex 1

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Another shorter path:
(3, 5) → (3, 1, 5)
0.931 > 0.245 + 0.277 = 0.522

We can update matrix!



Floyd's Algorithm – Another Example

First attempt passing through vertex 1

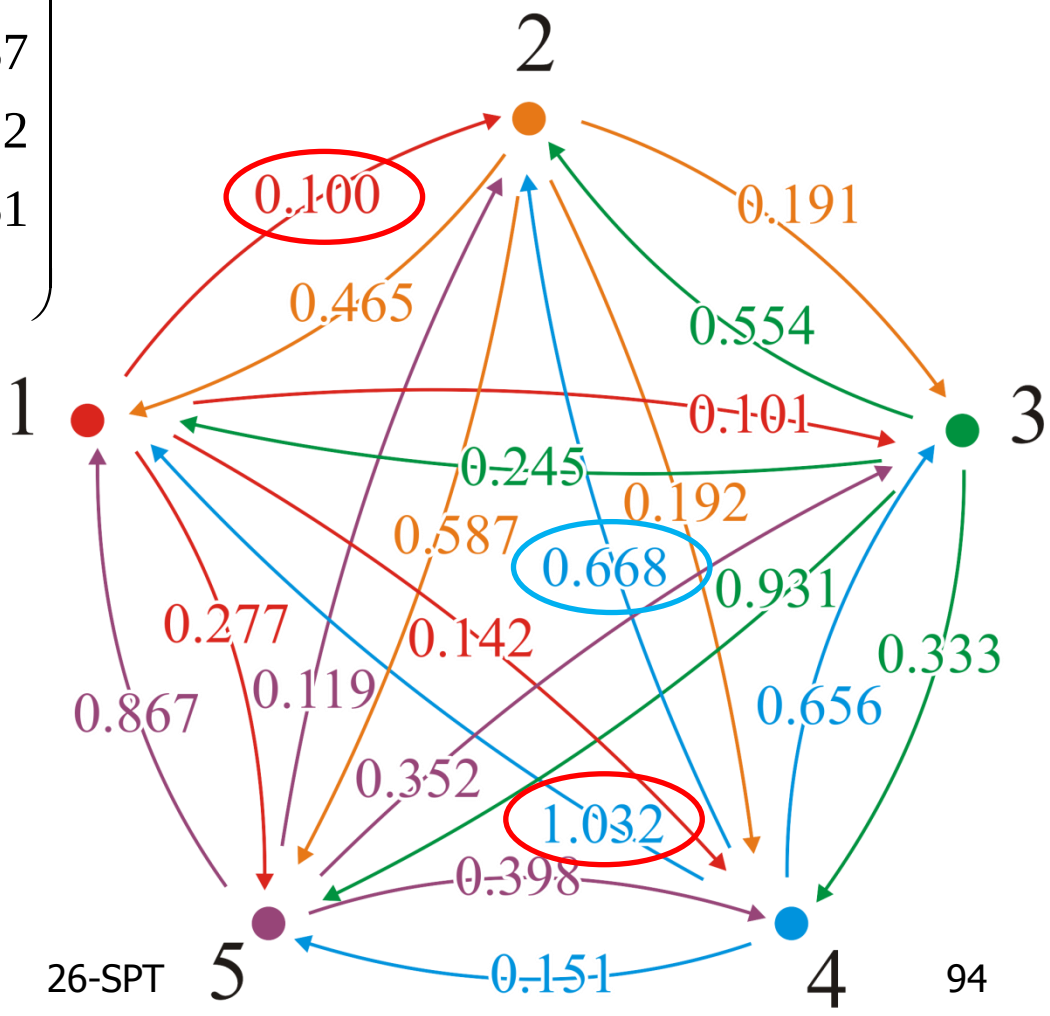
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Continuing:

(4, 2) → (4, 1, 2)

0.668 ≯ 1.032 + 0.100

In fact, no other shorter paths through vertex 1 exist!



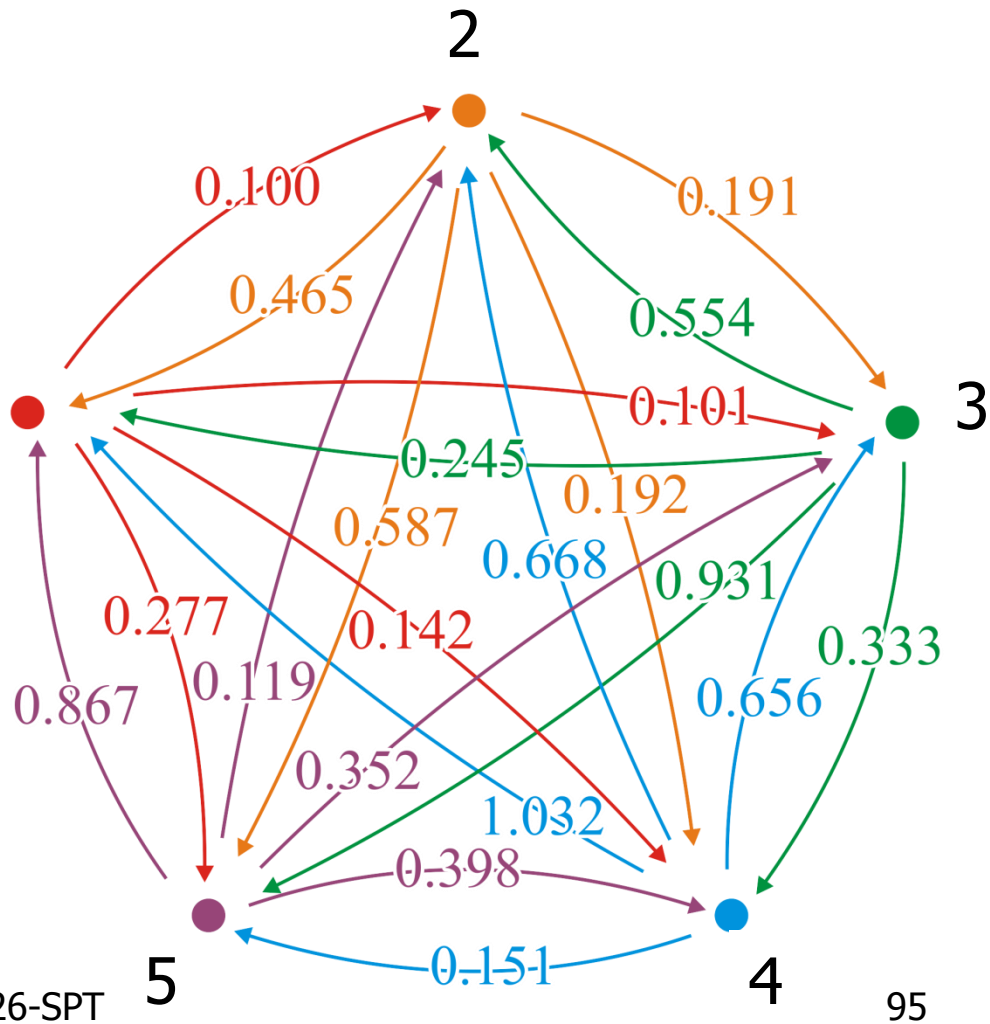
Floyd's Algorithm – Another Example

The D^0 matrix

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

The D^1 matrix

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$



Floyd's Algorithm – Another Example

Now attempt passing through vertex 2

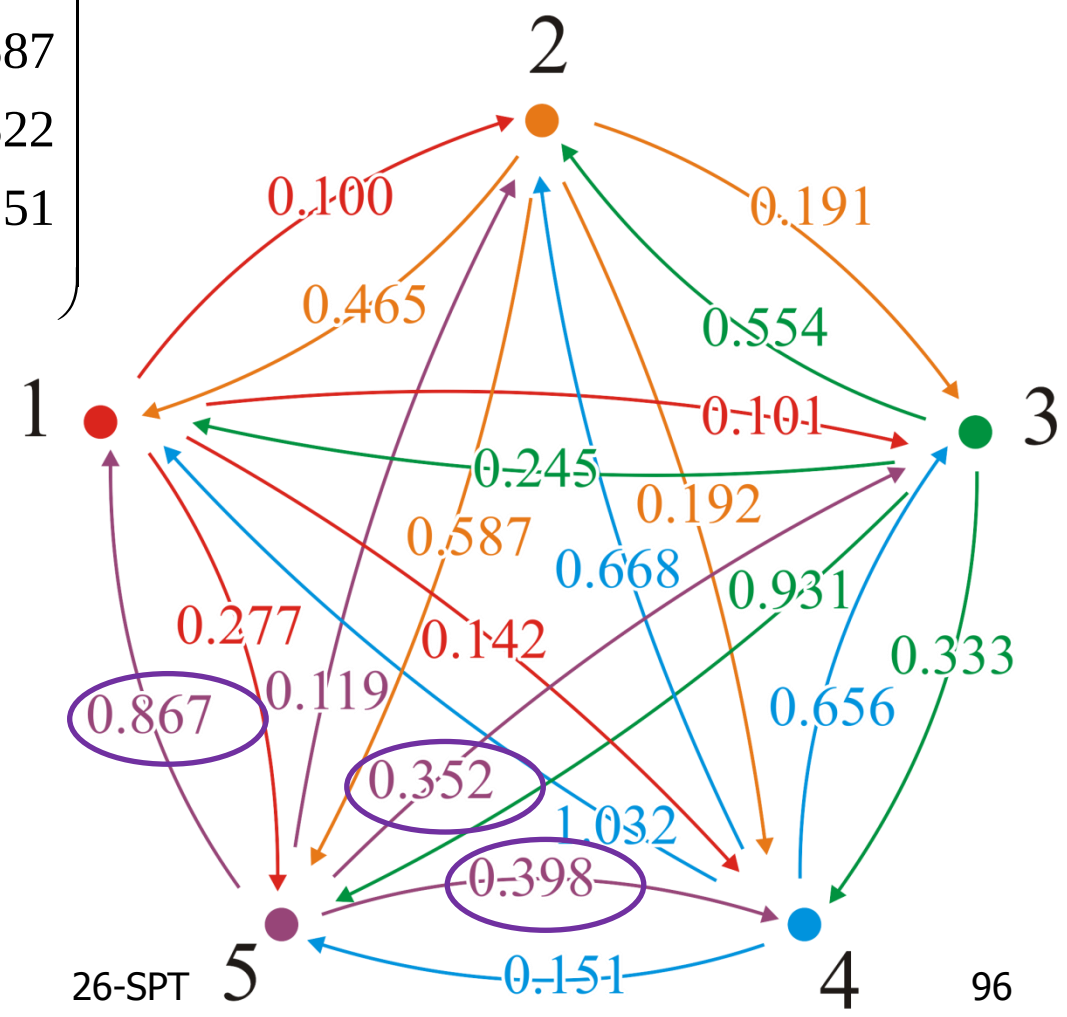
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

There are three shorter paths:

(5, 1) → (5, 2, 1)
0.867 > 0.119 + 0.465 = 0.584

(5, 3) → (5, 2, 3)
0.352 > 0.119 + 0.191 = 0.310

(5, 4) → (5, 2, 4)
0.398 > 0.119 + 0.192 = 0.311

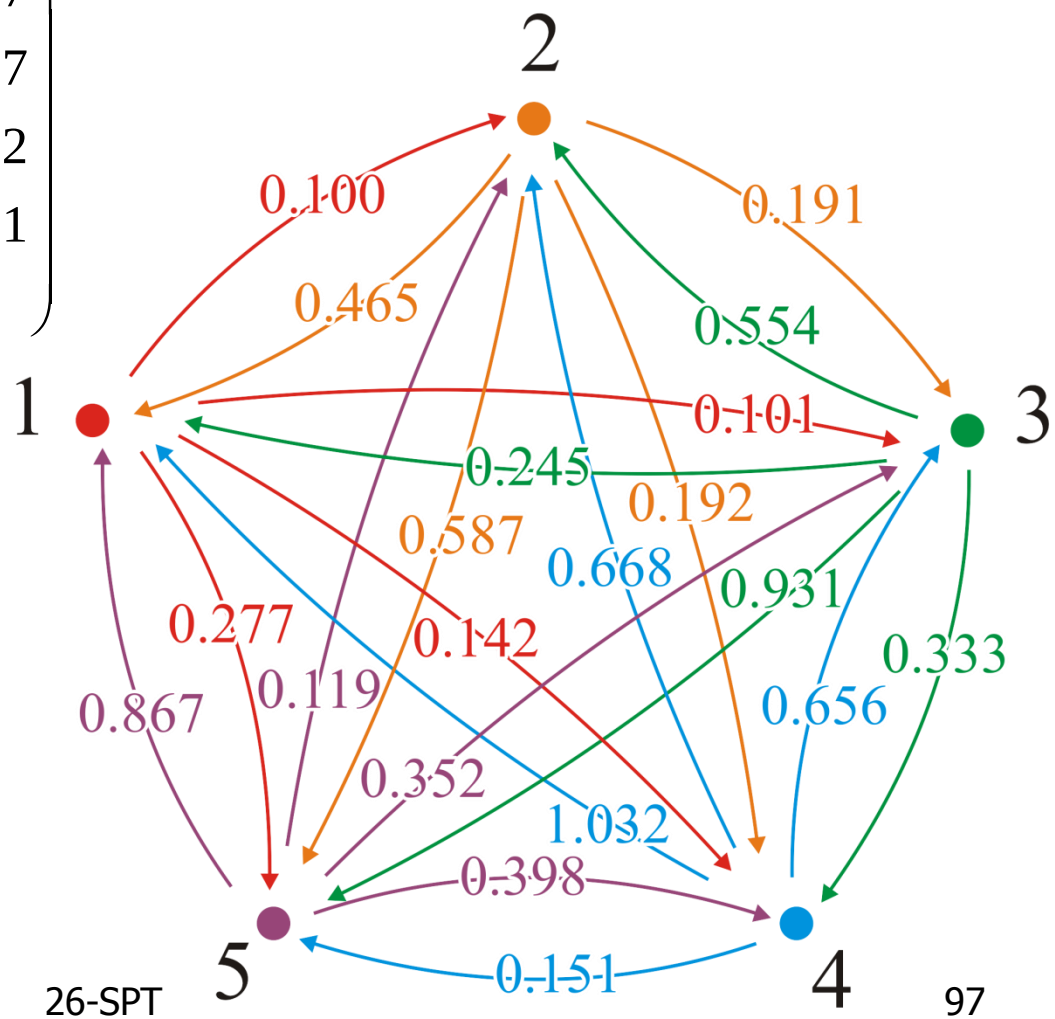


Floyd's Algorithm – Another Example

Now attempt passing through vertex 2

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.584 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

We can update matrix to get D^2



Floyd's Algorithm – Another Example

Now attempt passing through vertex 3

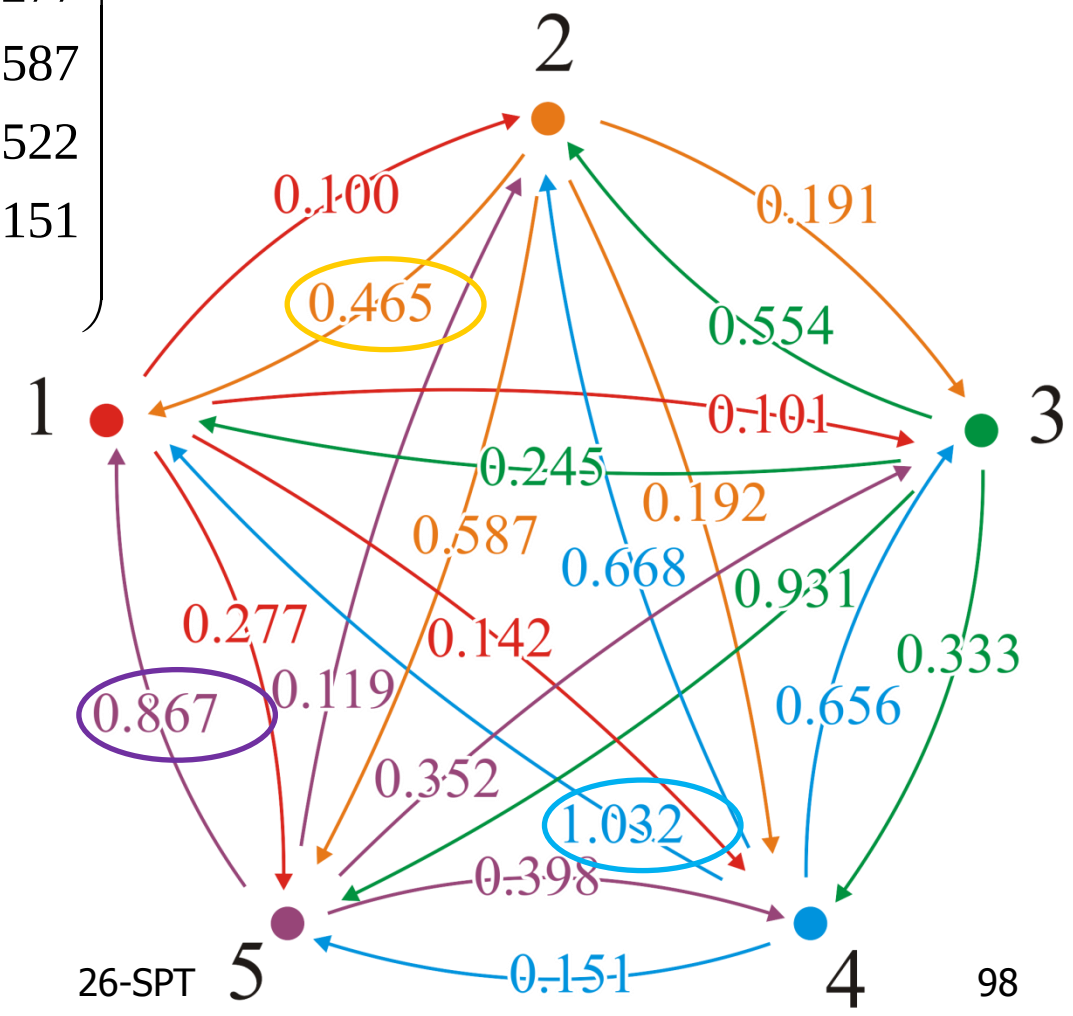
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.584 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

There are three shorter paths:

(2, 1) → (2, 3, 1)
0.465 > 0.191 + 0.245 = 0.436

(4, 1) → (4, 3, 1)
1.032 > 0.656 + 0.245 = 0.901

(5, 1) → (5, 3, 1)
0.584 > 0.310 + 0.245 = 0.555

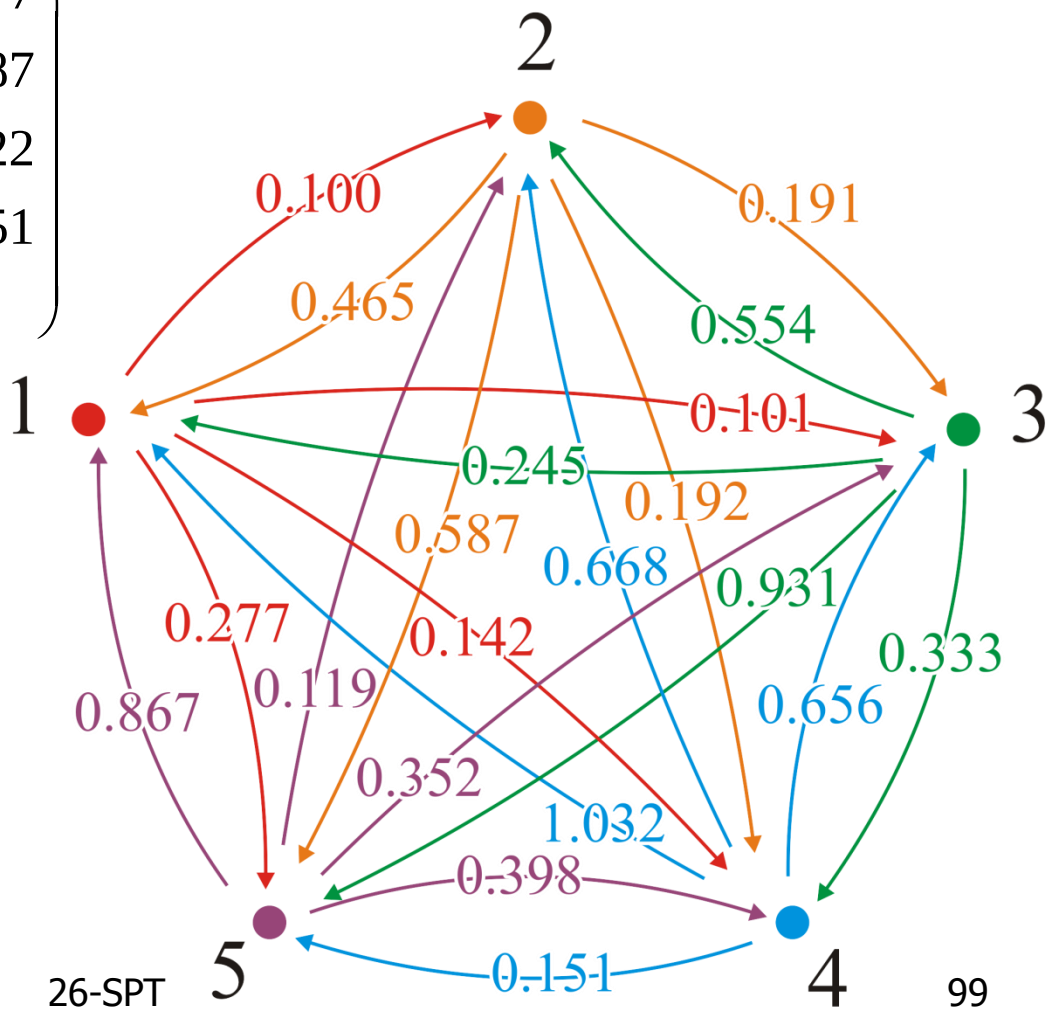


Floyd's Algorithm – Another Example

Now attempt passing through vertex 3

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.436 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 0.901 & 0.668 & 0.656 & 0 & 0.151 \\ 0.555 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

We can update matrix to get D^3



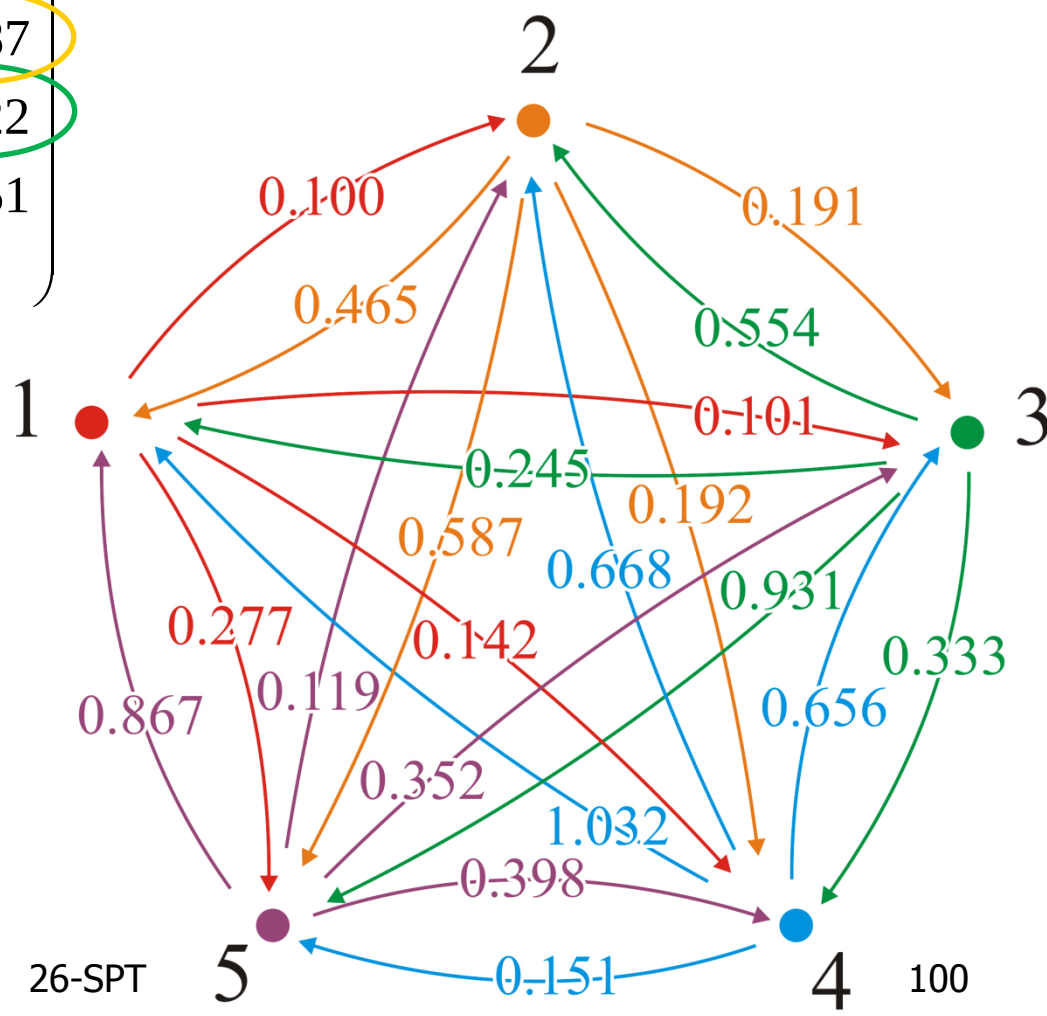
Floyd's Algorithm – Another Example

Now attempt passing through vertex 4

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.436 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.345 & 0 & 0.333 & 0.522 \\ 0.901 & 0.668 & 0.656 & 0 & 0.151 \\ 0.555 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

There are two shorter paths:
(2, 5) → (2, 4, 5)
0.587 > 0.192 + 0.151

(3, 5) → (3, 4, 5)
0.522 > 0.333 + 0.151

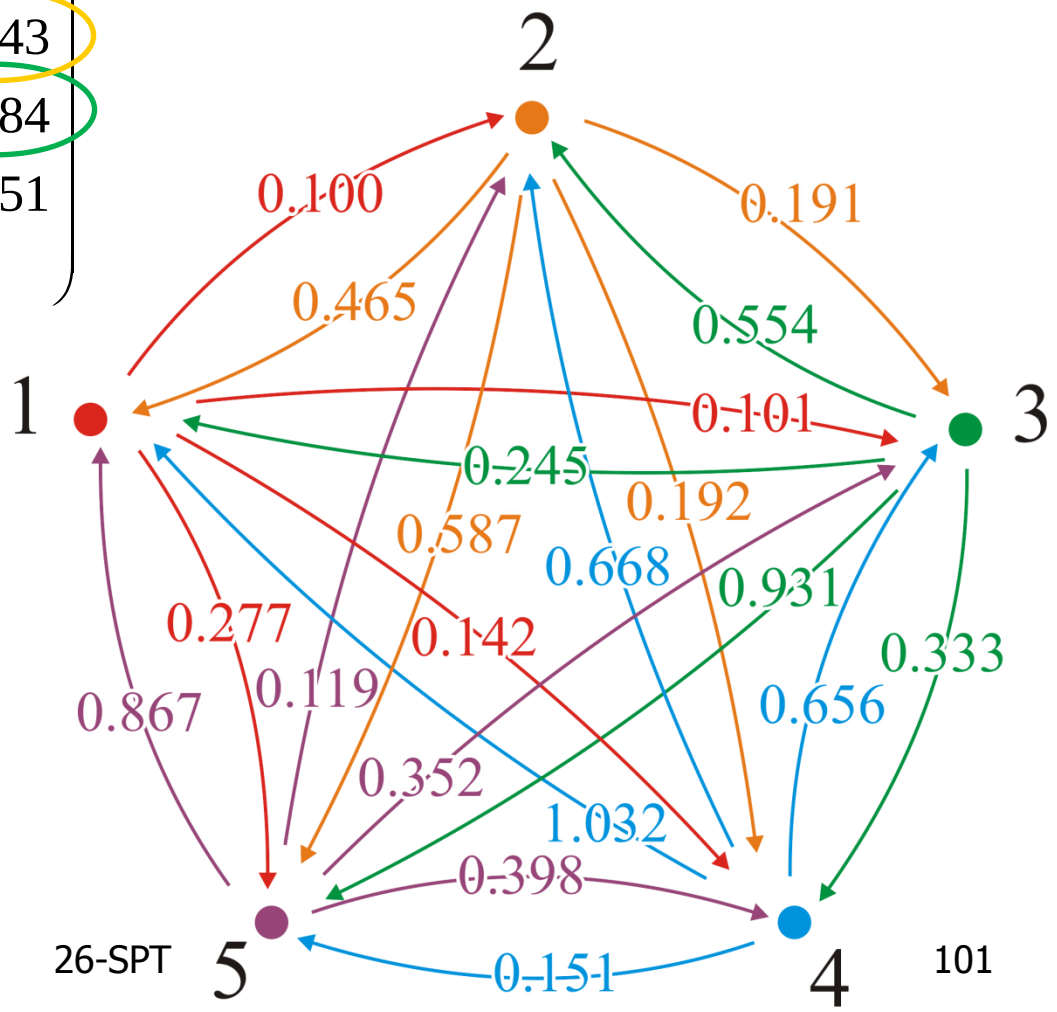


Floyd's Algorithm – Another Example

Now attempt passing through vertex 4

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.436 & 0 & 0.191 & 0.192 & 0.343 \\ 0.245 & 0.345 & 0 & 0.333 & 0.484 \\ 0.901 & 0.668 & 0.656 & 0 & 0.151 \\ 0.555 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

We can update matrix to get D^4



Floyd's Algorithm – Another Example

Now attempt passing through vertex 5

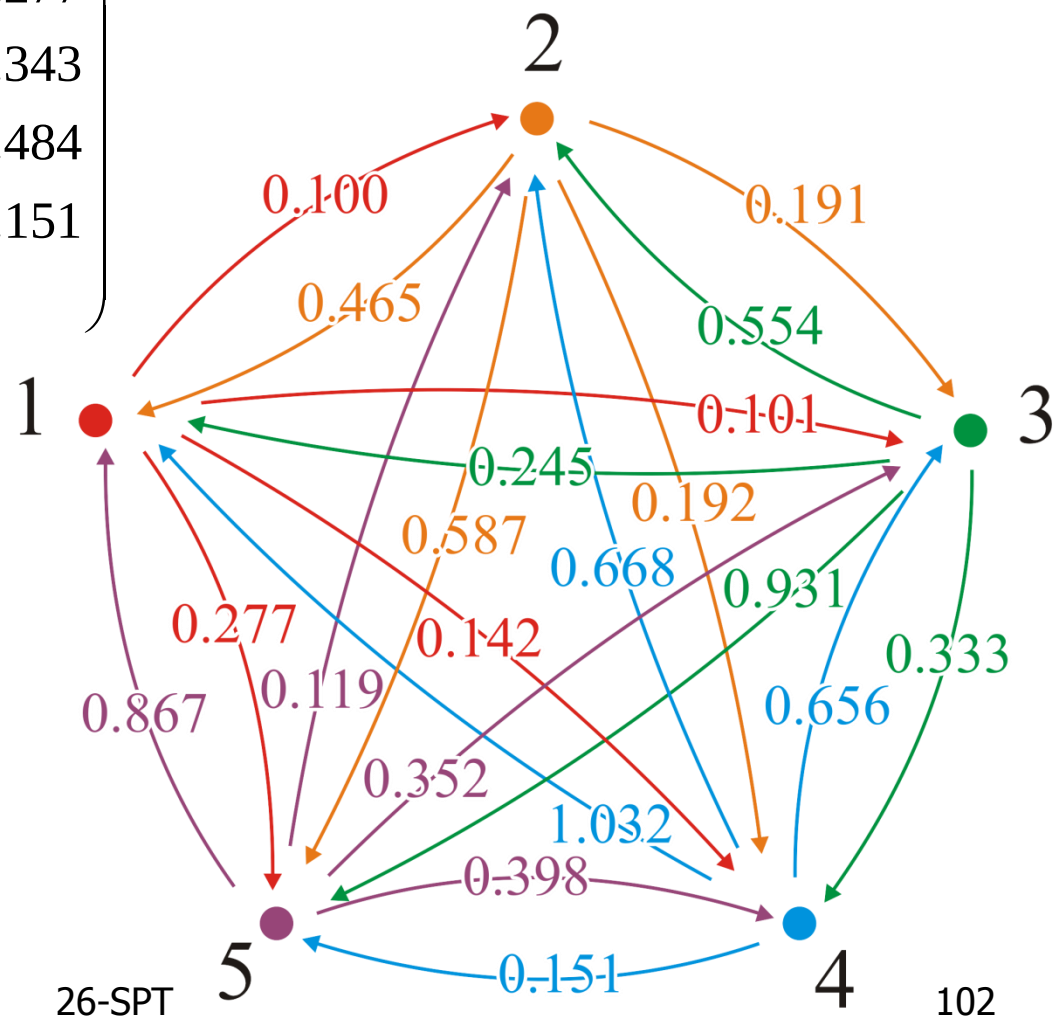
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.436 & 0 & 0.191 & 0.192 & 0.343 \\ 0.245 & 0.345 & 0 & 0.333 & 0.484 \\ 0.901 & 0.668 & 0.656 & 0 & 0.151 \\ 0.555 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

There are two shorter paths:

(4, 1) → (4, 5, 1)
0.901 > 0.151 + 0.555 = 0.706

(4, 2) → (4, 5, 2)
0.668 > 0.151 + 0.119 = 0.270

(4, 3) → (4, 5, 3)
0.656 > 0.151 + 0.310 = 0.461

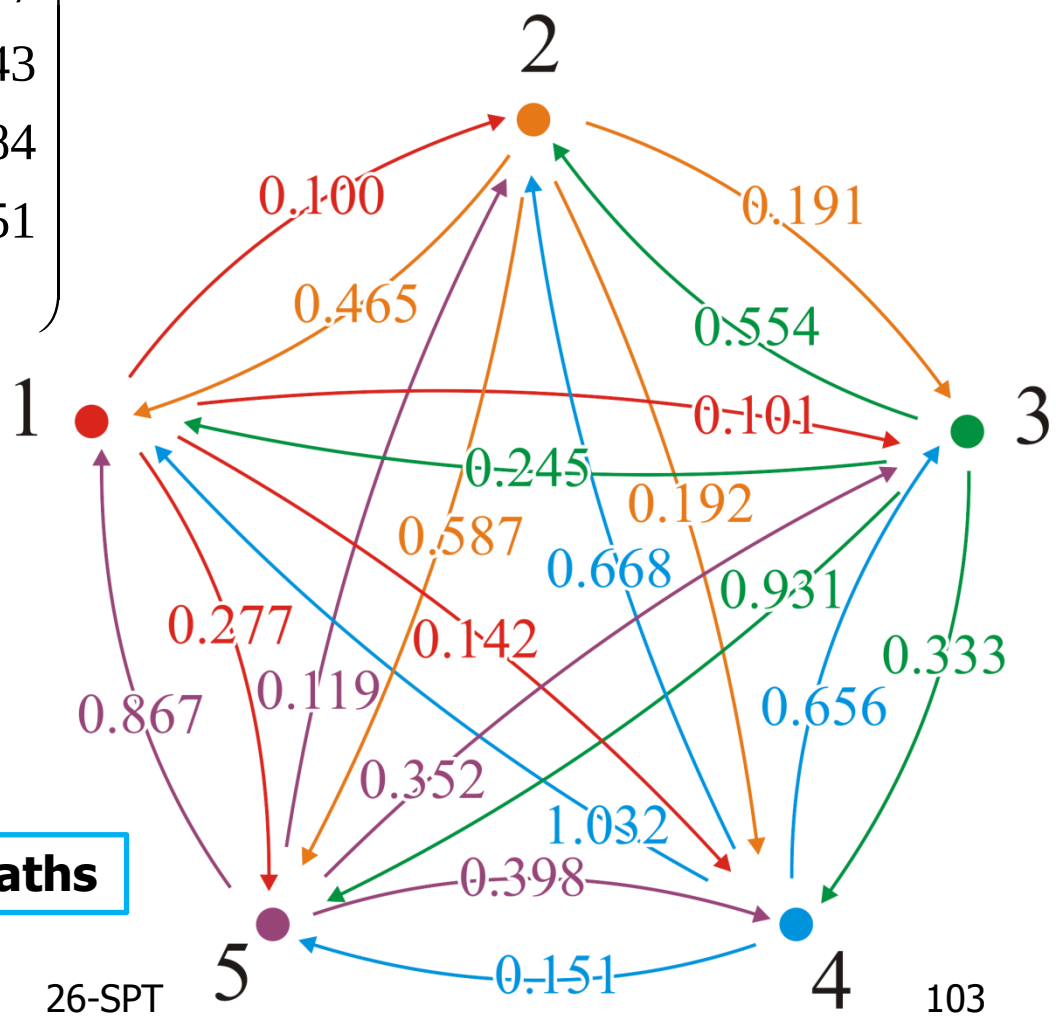


Floyd's Algorithm – Another Example

Now attempt passing through vertex 5

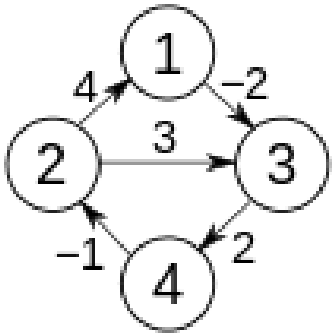
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.436 & 0 & 0.191 & 0.192 & 0.343 \\ 0.245 & 0.345 & 0 & 0.333 & 0.484 \\ 0.706 & 0.270 & 0.461 & 0 & 0.151 \\ 0.555 & 0.119 & 0.310 & 0.311 & 0 \end{pmatrix}$$

We can update matrix to get D^5

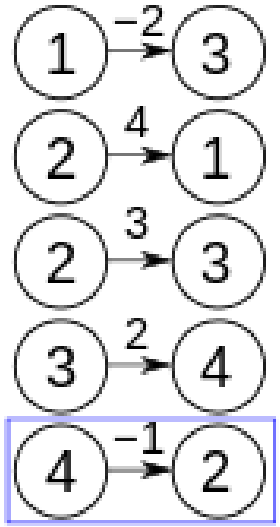


Thus, we have all pairs shortest paths

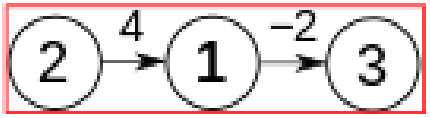
All Pairs Shortest Paths – Yet Another Example



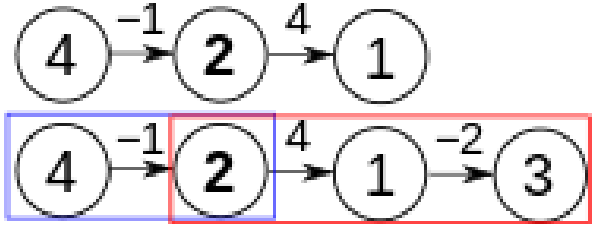
$k = 0$:



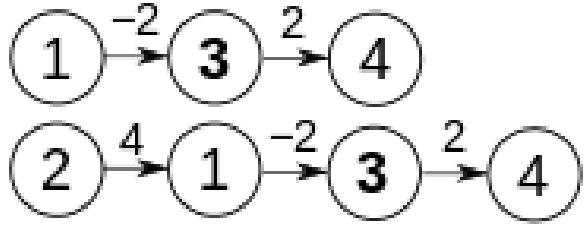
$k = 1$:



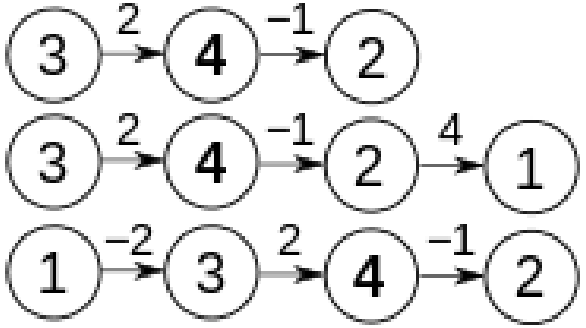
$k = 2$:



$k = 3$:



$k = 4$:



Any Question So Far?