Shortest path

Textbook Ch 24, 25

Outline

- Definition and applications
- Dijkstra’s algorithm
- Floyd-Warshall algorithm

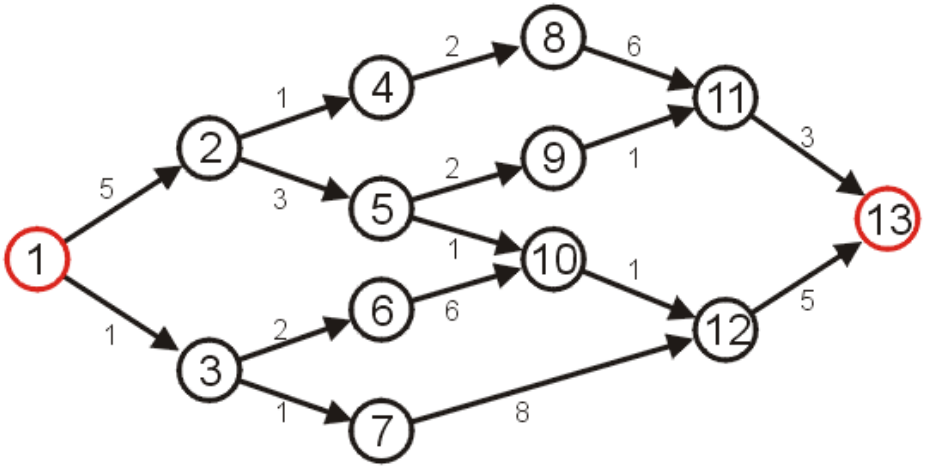
Shortest Path

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

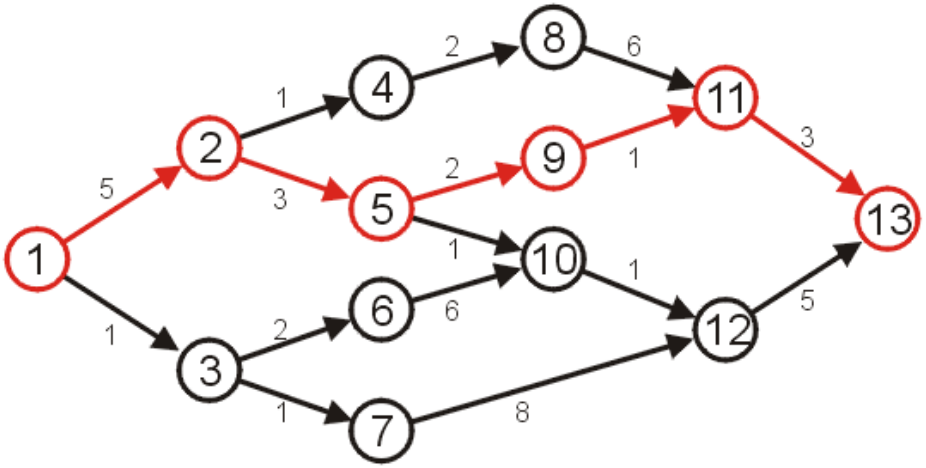
Shortest Path

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



Shortest Path

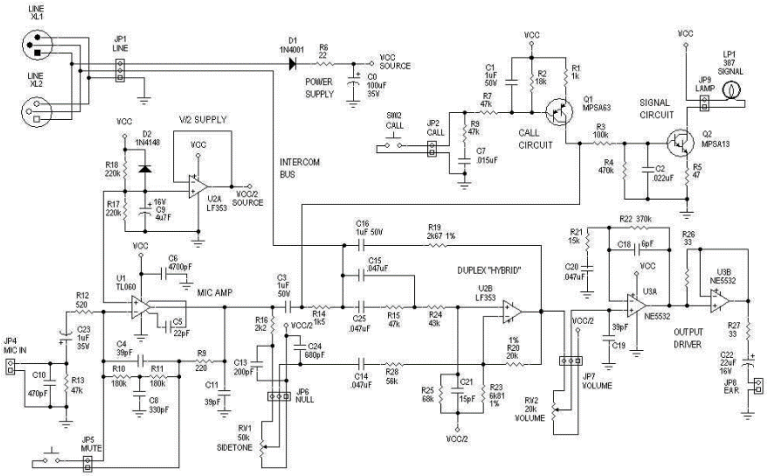
After some consideration, we may determine that the shortest path is as follows, with length 14



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



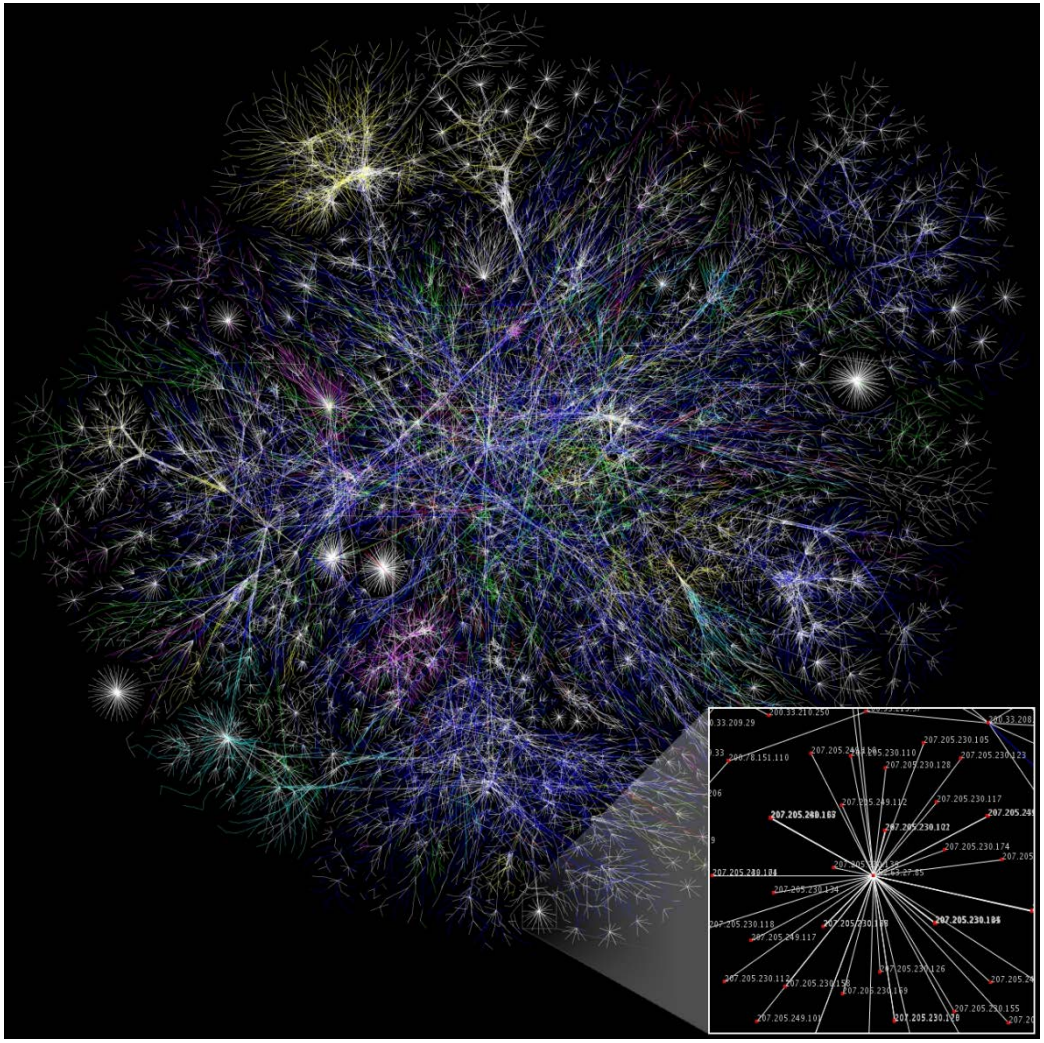
http://www.hp.com/

Applications

The Internet is a collection of interconnected devices

- Routers, individual computers

These may be represented as graphs



Applications

Information is passed through *packets.*

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

We would like to pass packets through the shortest path.

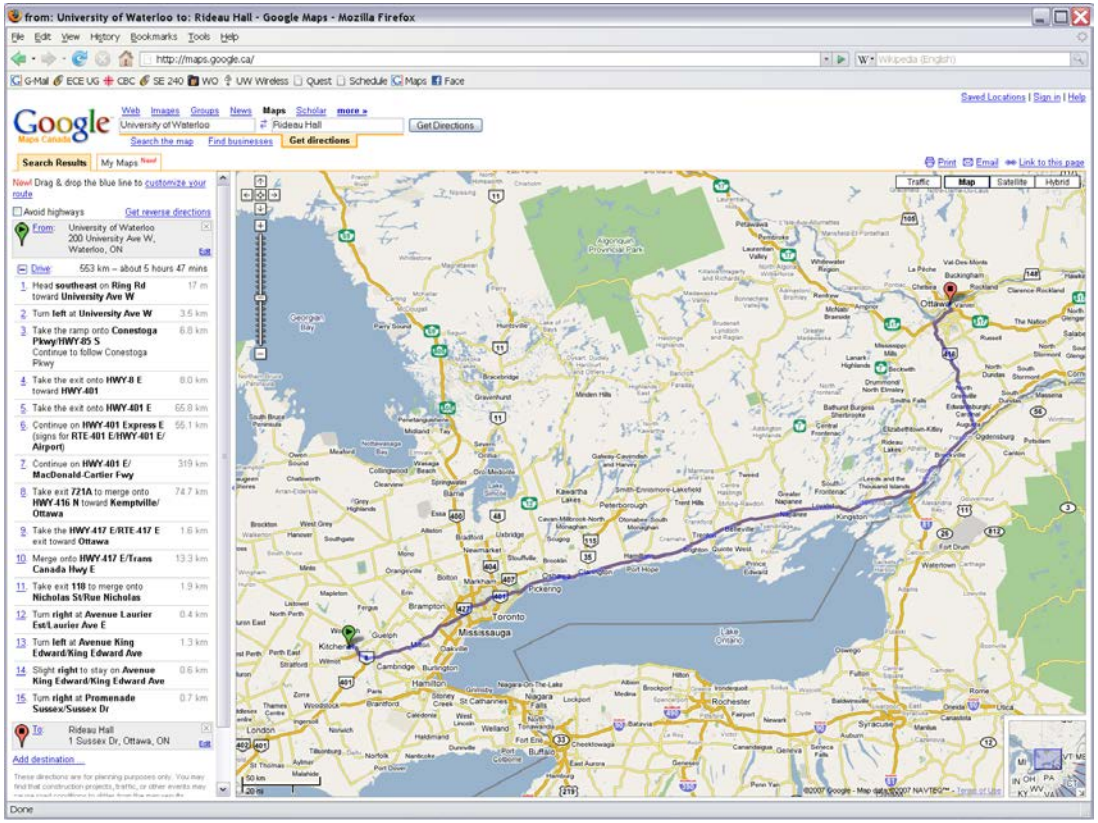
Metrics for measuring the shortest path may include:

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

Applications

Another obvious application is finding the shortest route between two points on a map

The shortest path using distance as a metric is obvious, however, a driver may be more interested in minimizing time

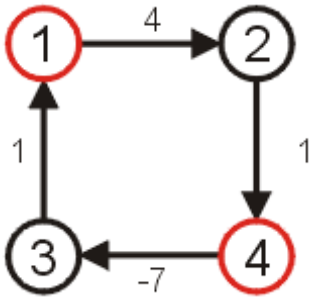


Shortest Path

The goal is to find the shortest path and its length

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

- Why this assumption?
- If we have negative weights, 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...



Algorithms

Algorithms for finding the shortest path include:

- Dijkstra's algorithm
- A* search algorithm
- Bellman-Ford algorithm
- Floyd-Warshall algorithm

Outline

- Definition and applications
- Dijkstra’s algorithm
- Floyd-Warshall algorithm

Dijkstra’s algorithm

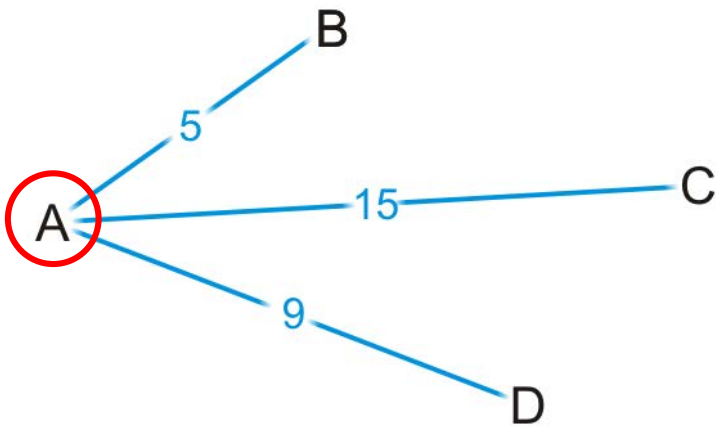
Dijkstra’s algorithm solves the single-source shortest path problem

- It is very similar to Prim’s algorithm
- Assumption: all the weights are positive

Strategy

Suppose you are at vertex A

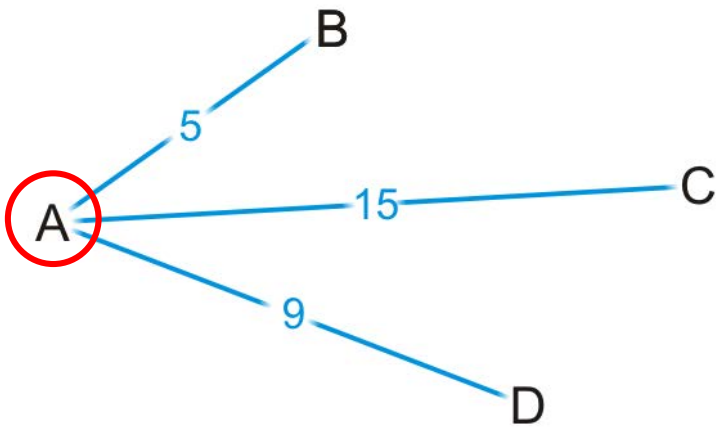
- You are aware of all vertices adjacent to it
- This information is either in an adjacency list or adjacency matrix



Strategy

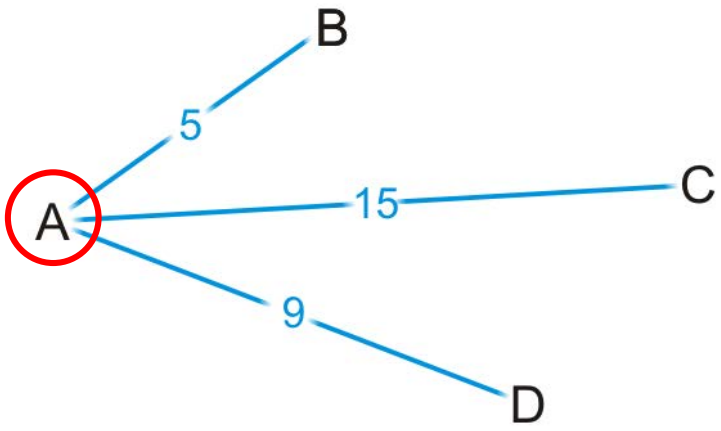
Is 5 the shortest distance to B via the edge (A, B)?

- Why or why not?



Strategy

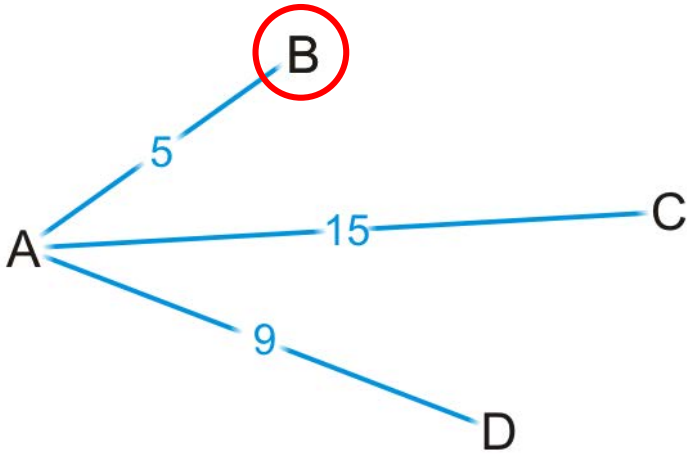
Are you guaranteed that the shortest path to C is (A, C), or that (A, D) is the shortest path to vertex D?



Strategy

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

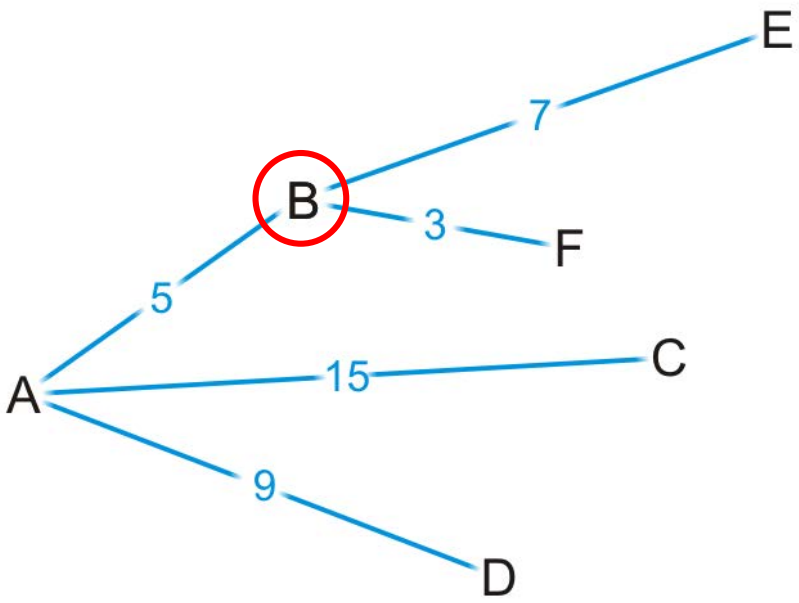
- Let's see where we can go from B



Strategy

By some simple arithmetic, we can determine that

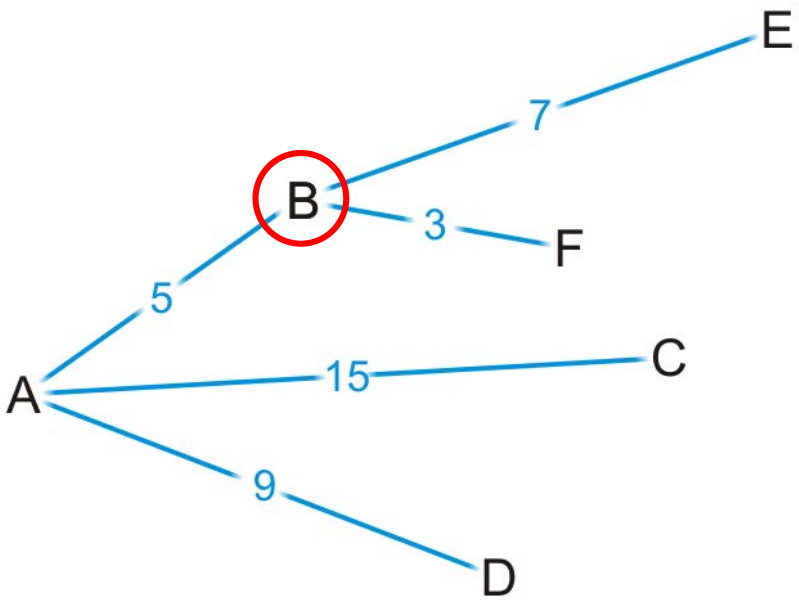
- There is a path (A, B, E) of length 5 + 7 = 12
- There is a path (A, B, F) of length 5 + 3 = 8



Strategy

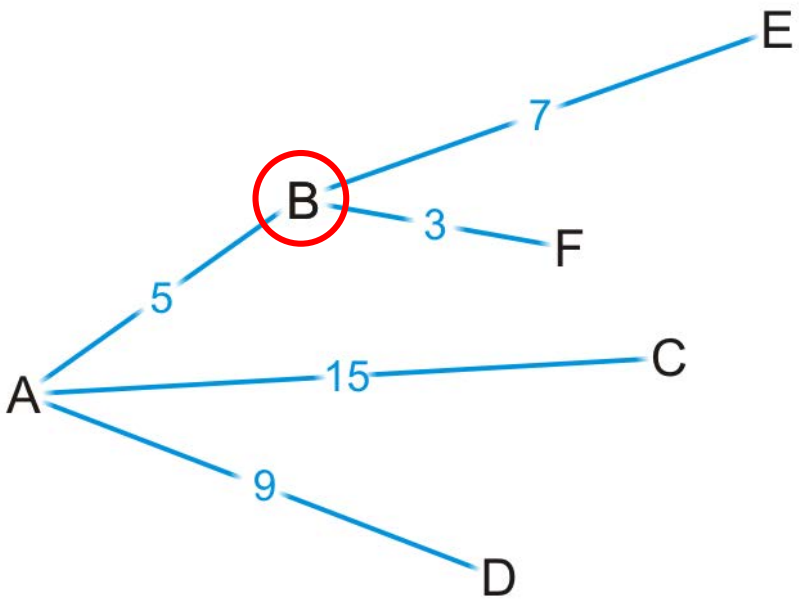
Is (A, B, F) is the shortest path from vertex A to F?

- Why or why not?



Strategy

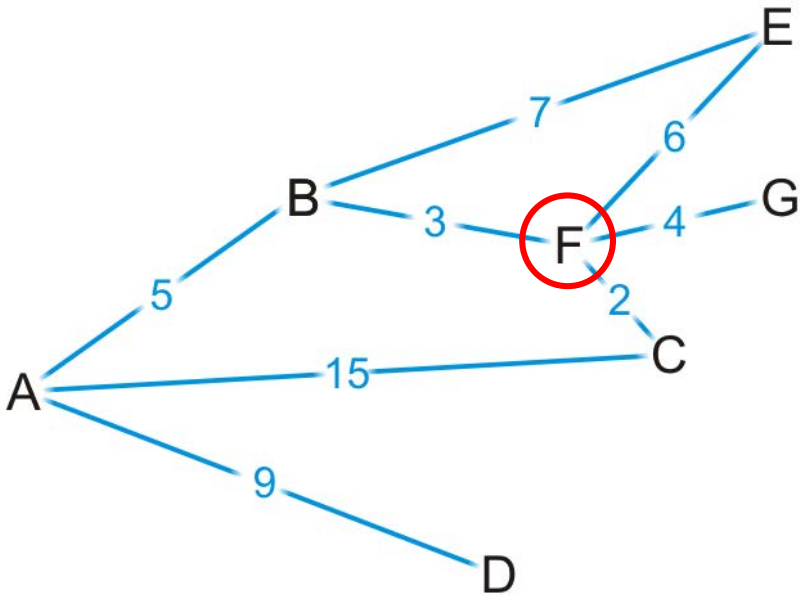
Are we guaranteed that any other path we are currently aware of is also going to be the shortest path?



Strategy

Okay, let's visit vertex F

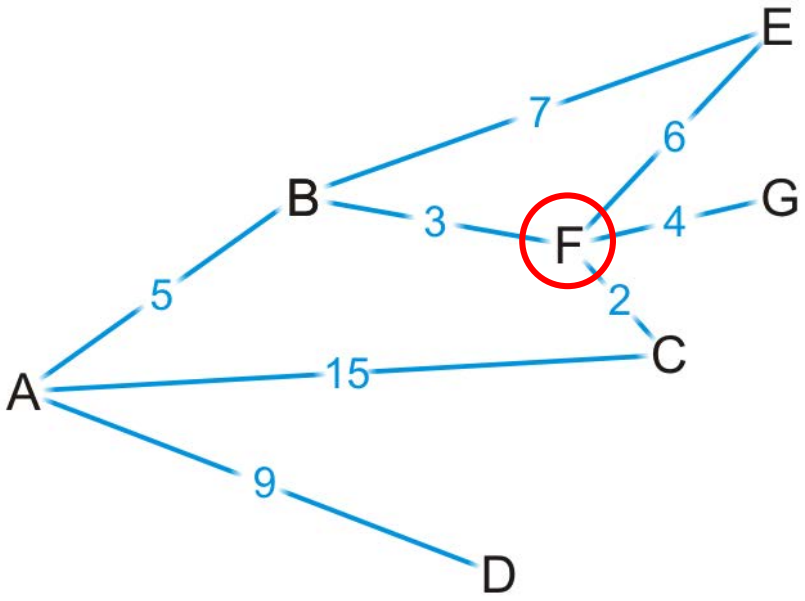
- We know the shortest path is (A, B, F) and it's of length 8



Strategy

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

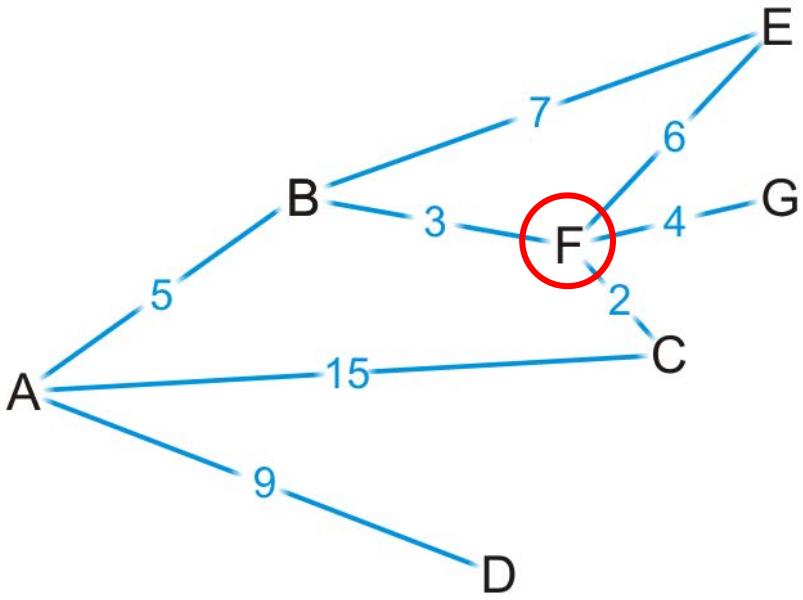
- (A, B, F, E) of length 8 + 6 = 14
- (A, B, F, G) of length 8 + 4 = 12
- (A, B, F, C) of length 8 + 2 = 10



Strategy

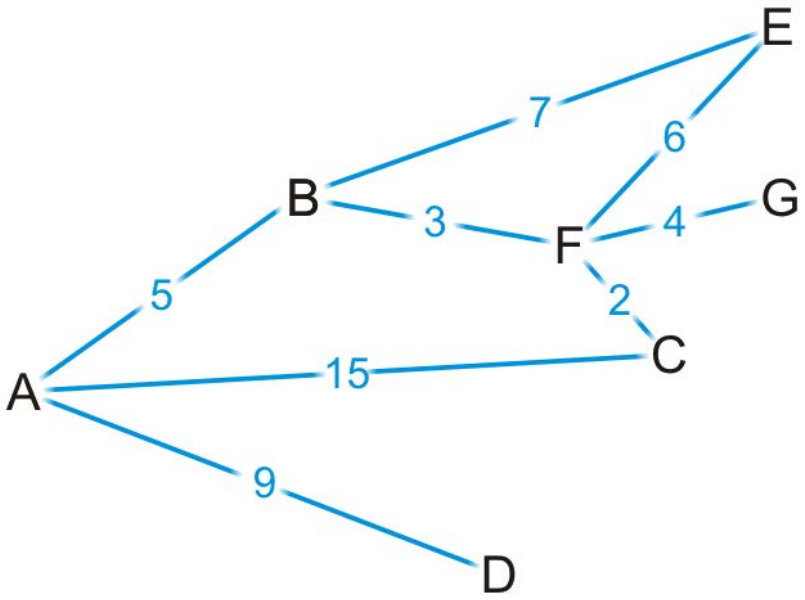
By observation:

- 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

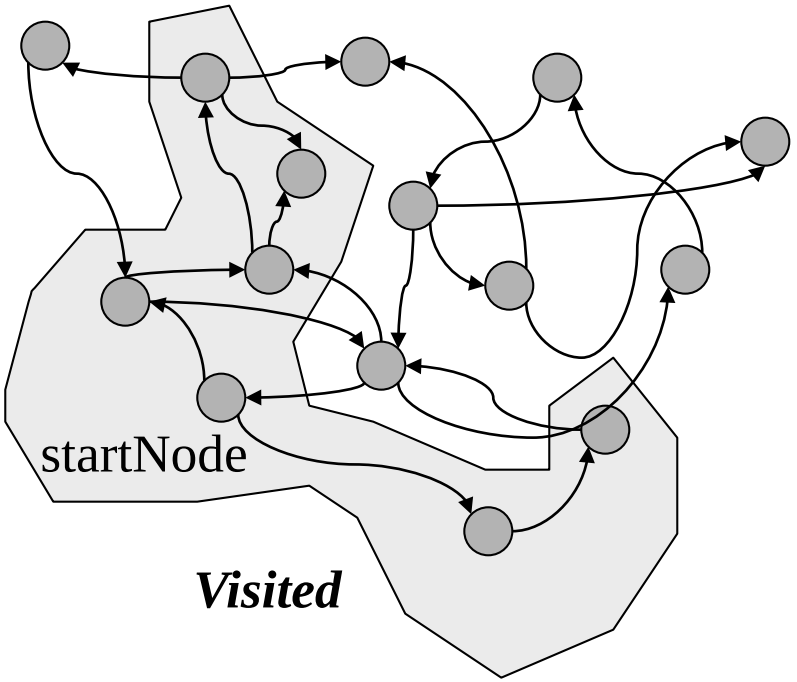
Which remaining vertex are we currently guaranteed to have the shortest distance to?



Strategy

In general

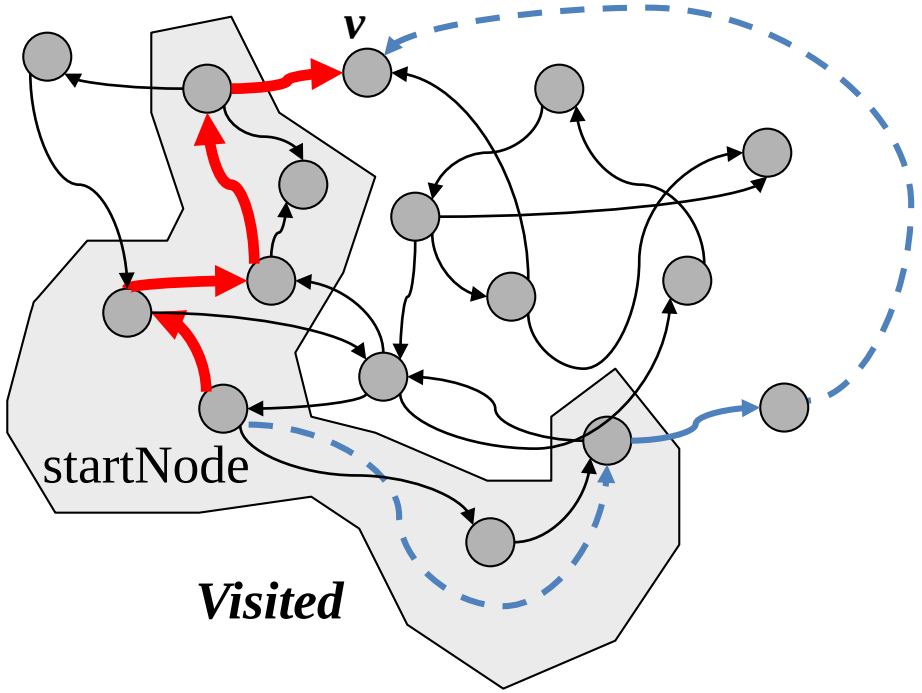
- We know the shortest distance to some of the vertices (marked as visited)
- We also know the shortest distance to each unvisited vertex **through visited vertices** (call this the “known distance”)



Strategy

Consider the unvisited vertex ***v*** that has the shortest known distance

- We are guaranteed that the known distance to ***v*** is the shortest distance from the start node to it
- Proof by contradiction



Dijkstra's algorithm

We need to track the known shortest distance to each vertex

- We require an array of distances, all initialized to infinity except for the source vertex, which is initialized to 0

Do we need to track the shortest path to each vertex?

- Ex: do I have to store (A, B, F) as the shortest path to vertex F?
- No. We only have to record that the shortest path to vertex F came from vertex B
  - The shortest path to F is the shortest path to B followed by the edge (B, F)
- Thus, we need an array of previous vertices, all initialized to null

We need to track visited vertices whose shortest paths have been found

- a Boolean table of size $|V|$

Dijkstra's algorithm

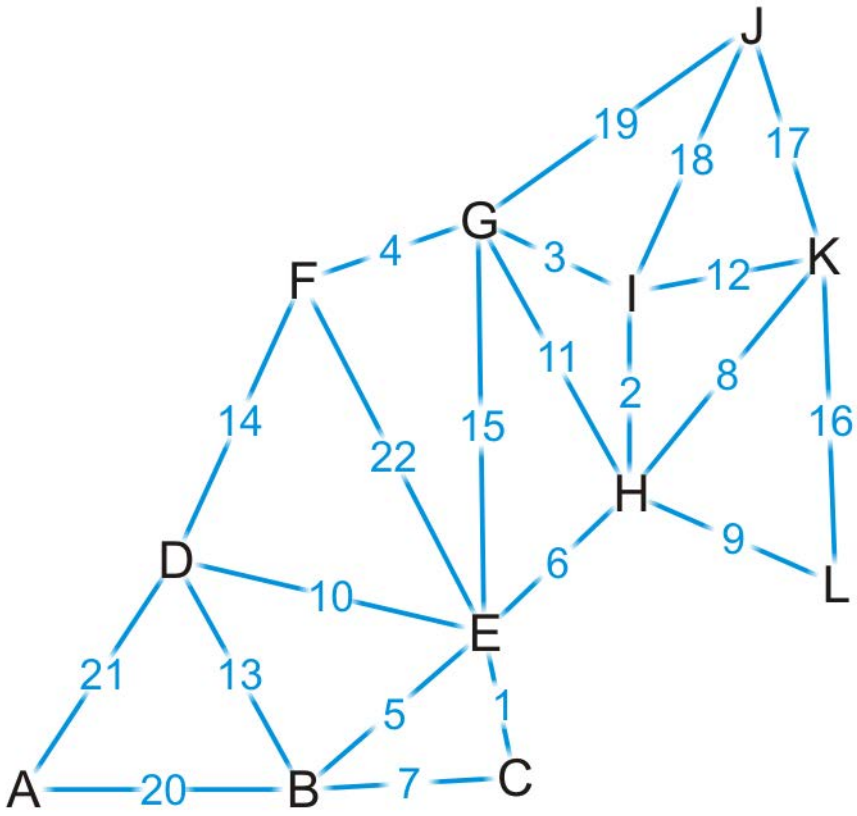
We will iterate $|V|$ times:

- Find the unvisited vertex v that has a minimum distance to it
- Mark it as visited
- Consider its 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 of infinity

Example

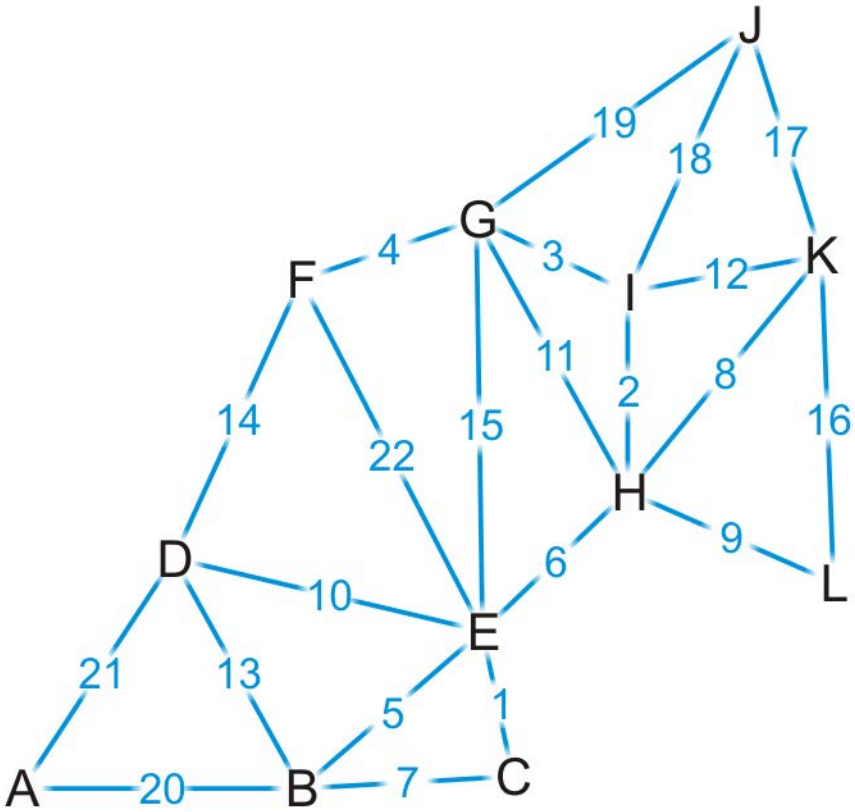
Find the shortest distance from K to every other vertex



Example

We set up our table

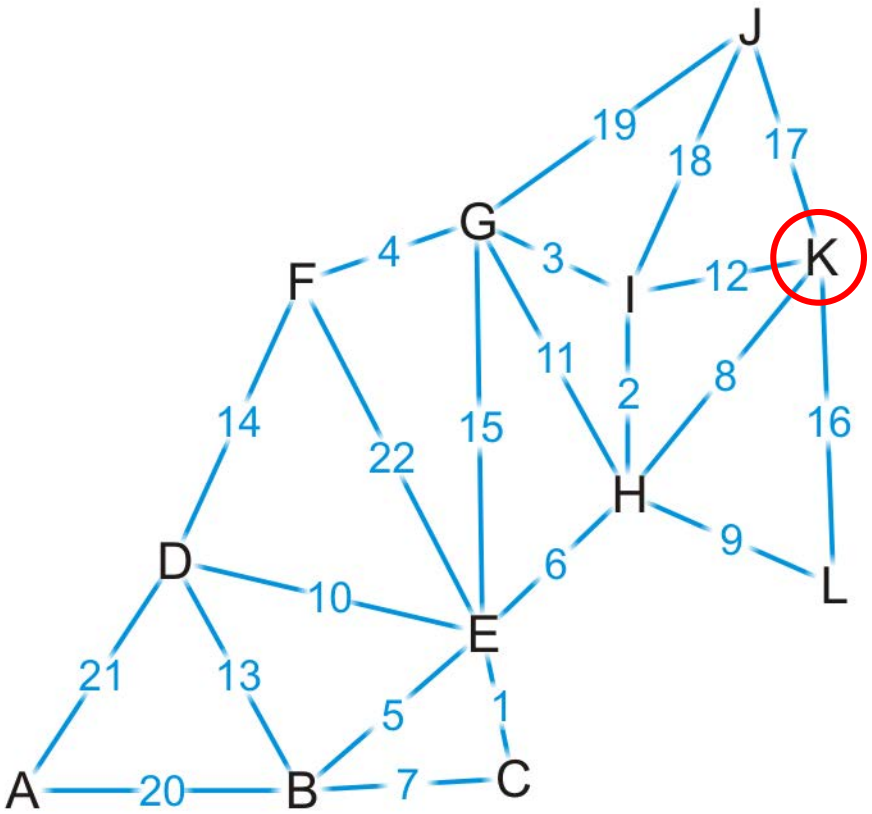
- Which unvisited vertex has the minimum distance to it?



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

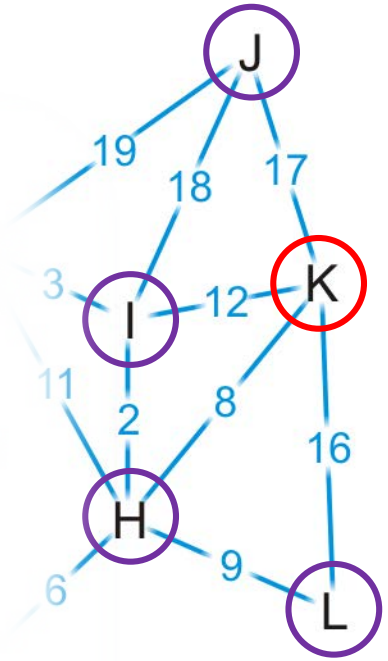
We visit vertex K



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

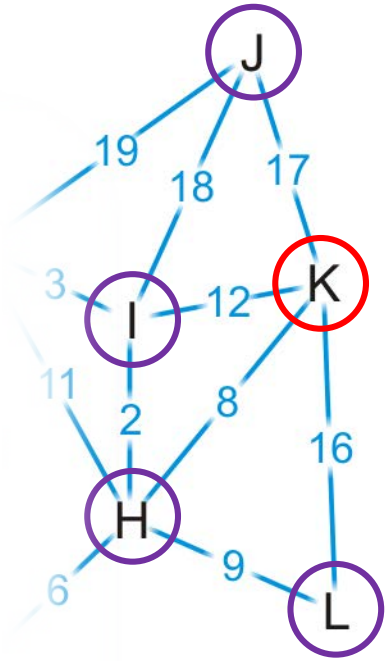
Vertex K has four neighbors: H, I, J and L



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

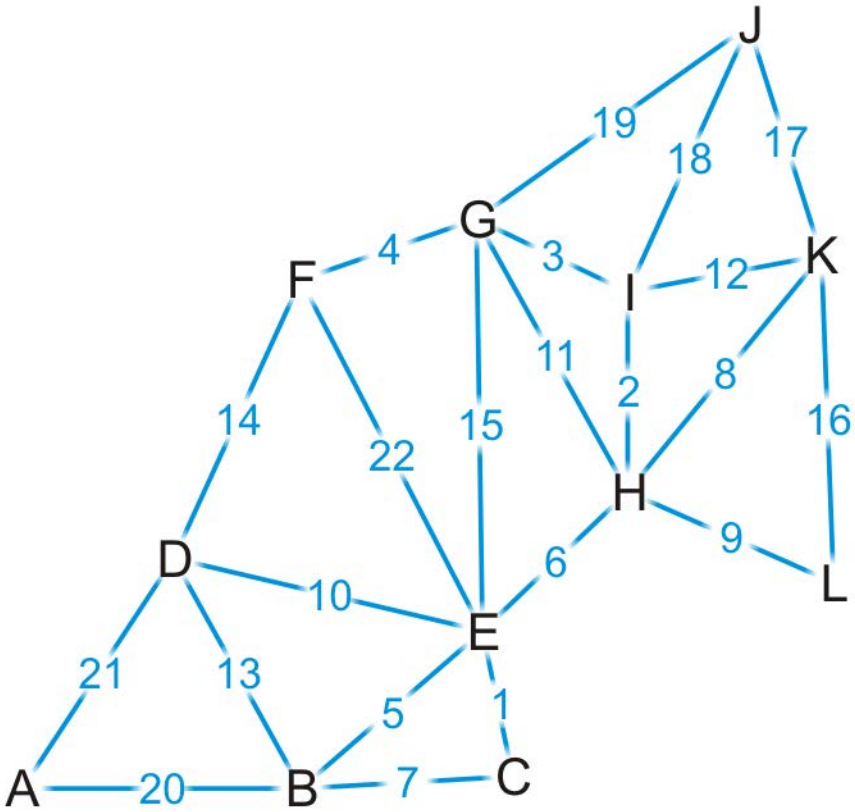


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	**F**	**8**	**K**
I	**F**	**12**	**K**
J	**F**	**17**	**K**
K	**T**	**0**	**Ø**
L	**F**	**16**	**K**

Example

We're finished with vertex K

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

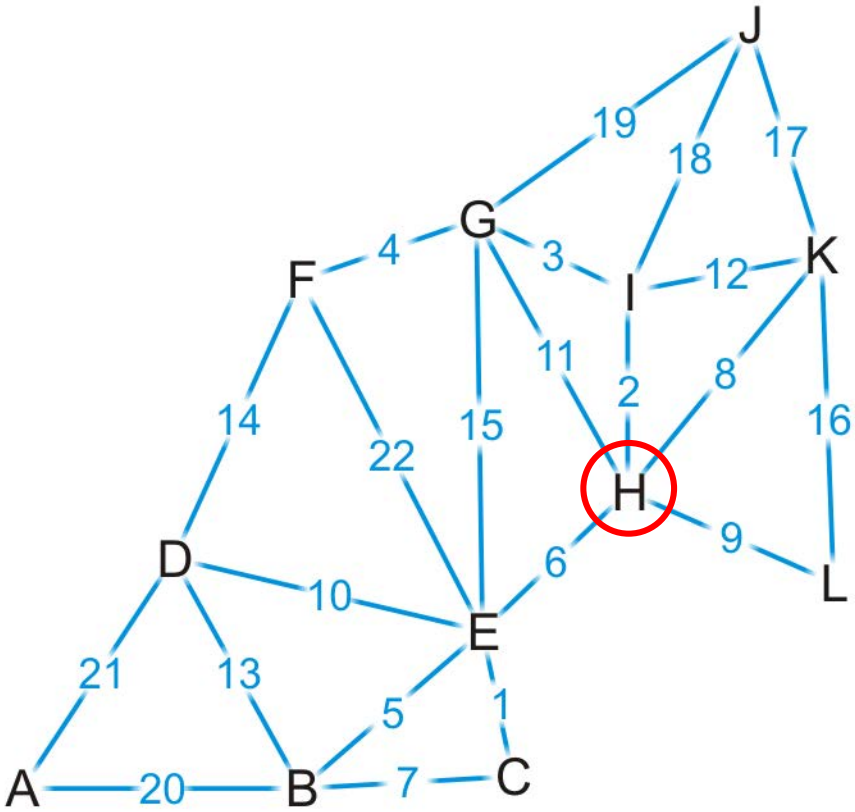


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	8	K
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

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:

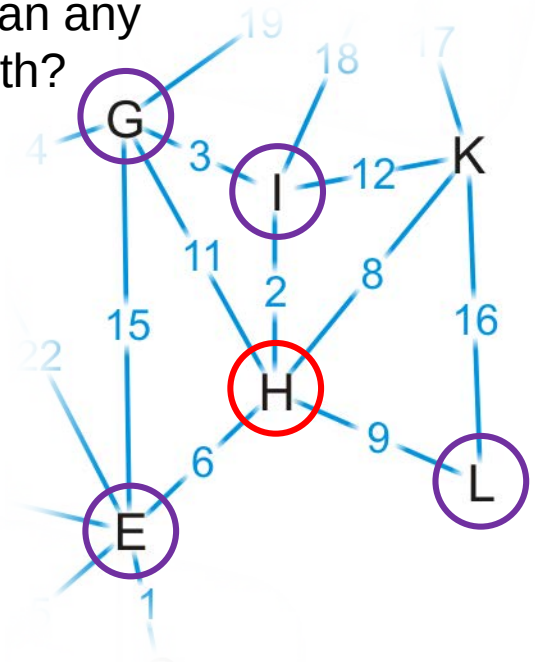
(K, H, E) of length 8 + 6 = 14

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

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

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

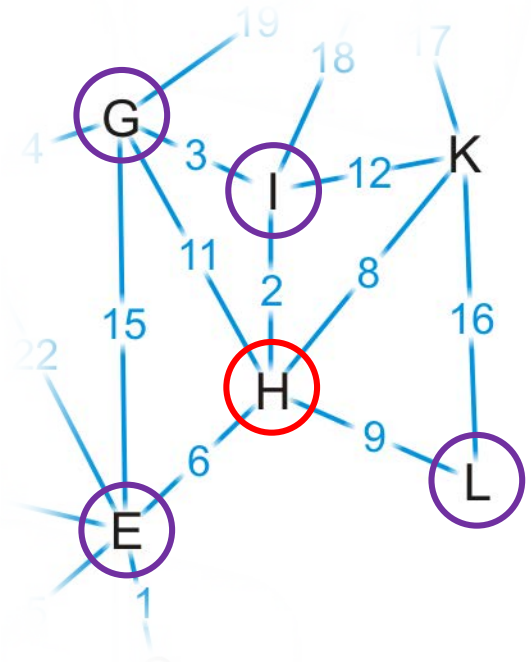
– Which of these are shorter than any known path?



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

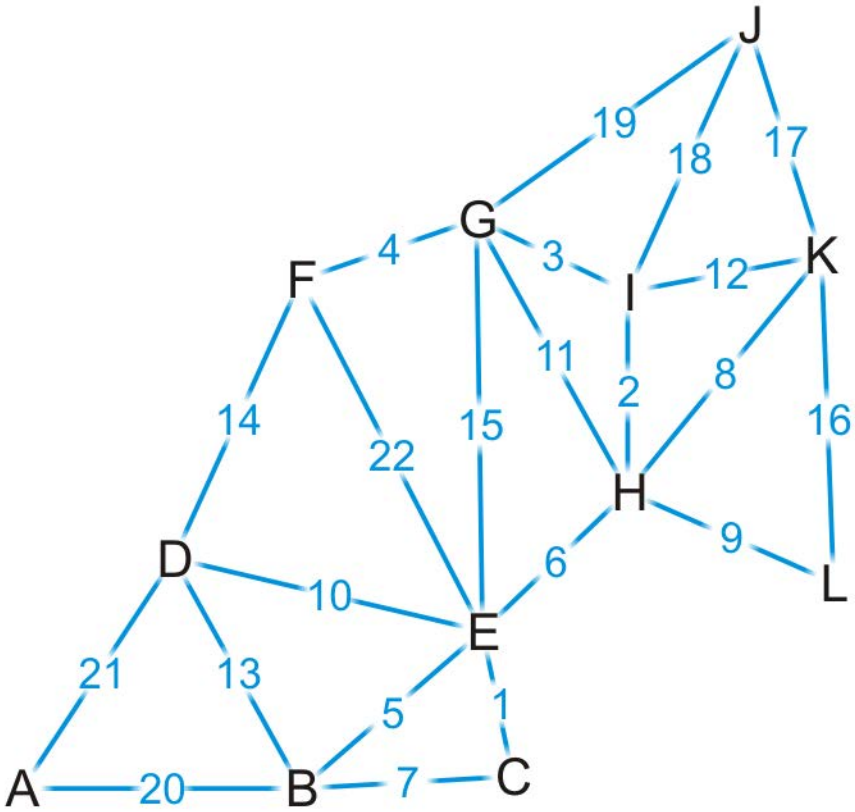


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	**14**	**H**
F	F	∞	Ø
G	F	**19**	**H**
H	**T**	**8**	**K**
I	F	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We are finished with vertex H

– Which vertex do we visit next?

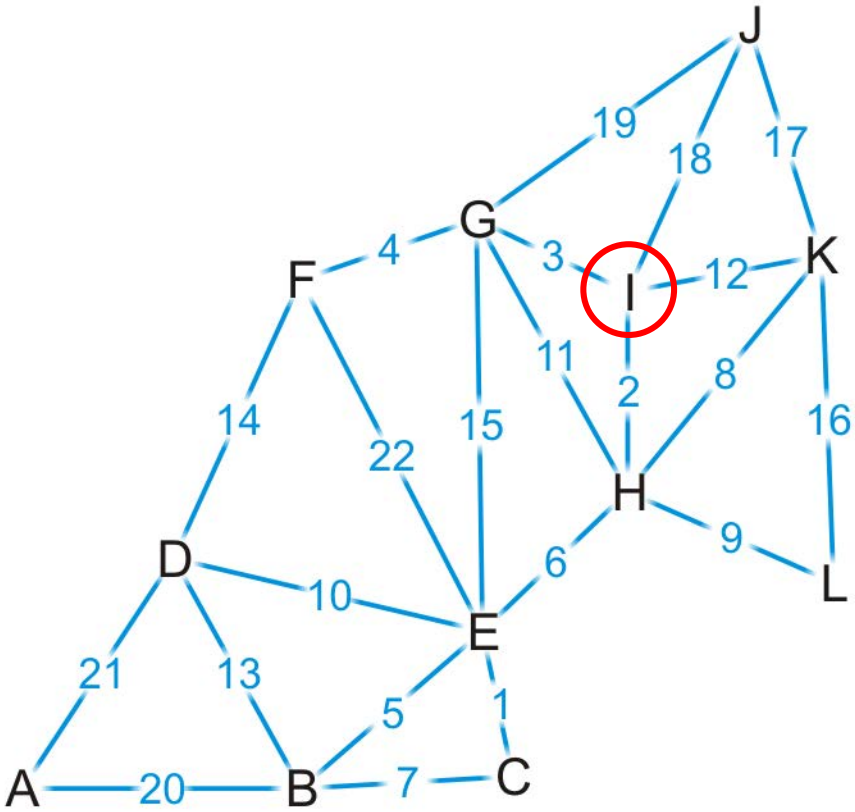


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	F	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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



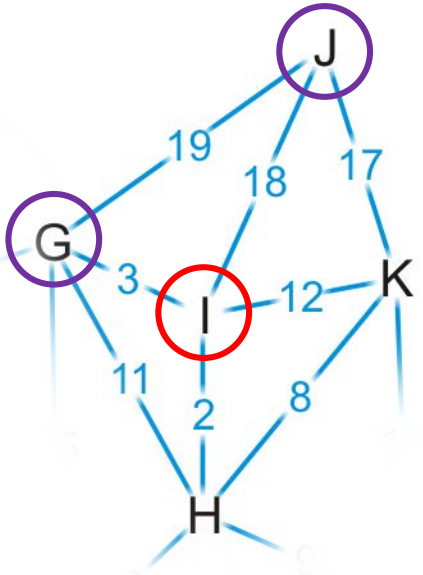
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	**T**	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Consider these paths:

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

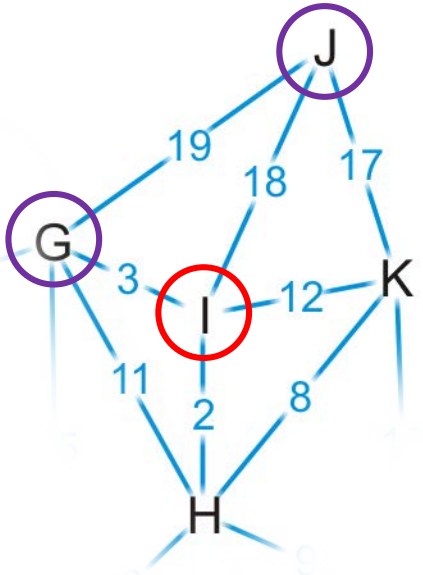
(K, H, I, J) of length 10 + 18 = 28



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

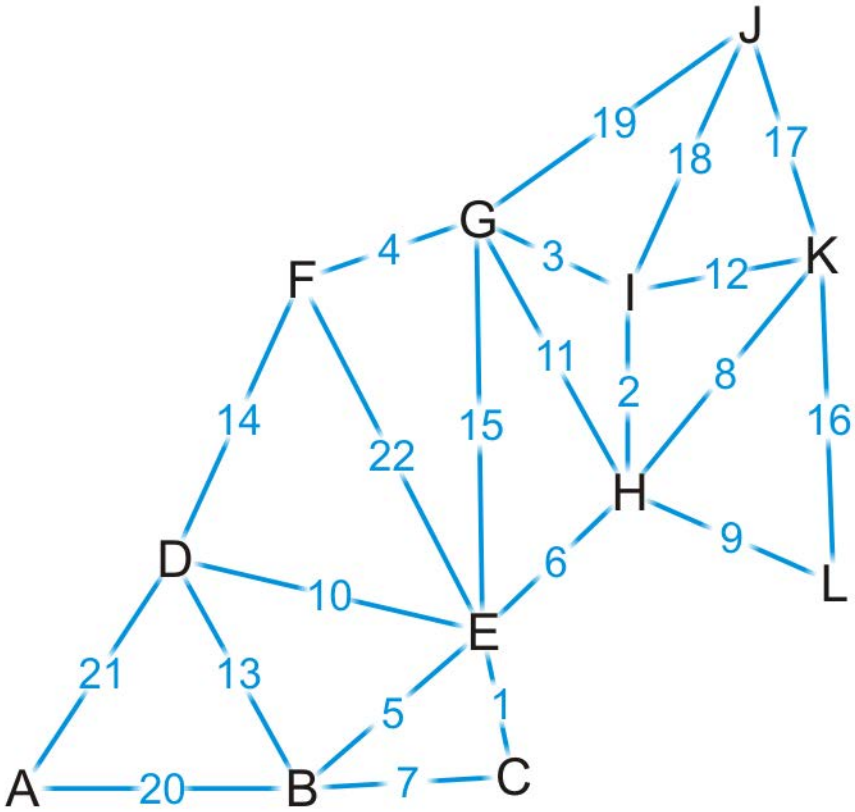
We have discovered a shorter path to vertex G, but (K, J) is still the shortest known path to vertex J



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?

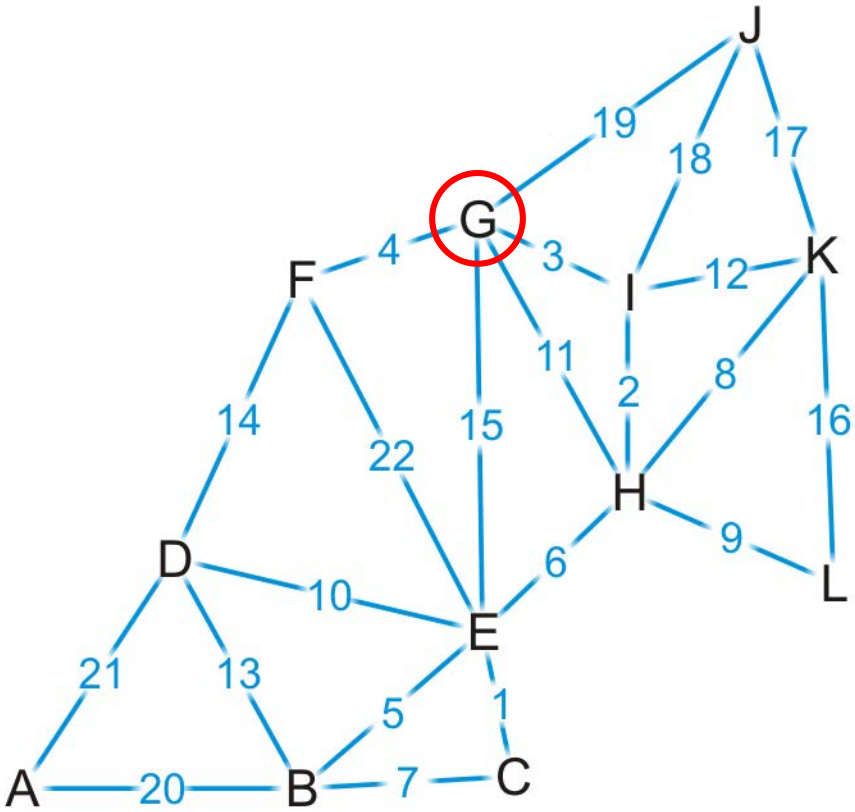


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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

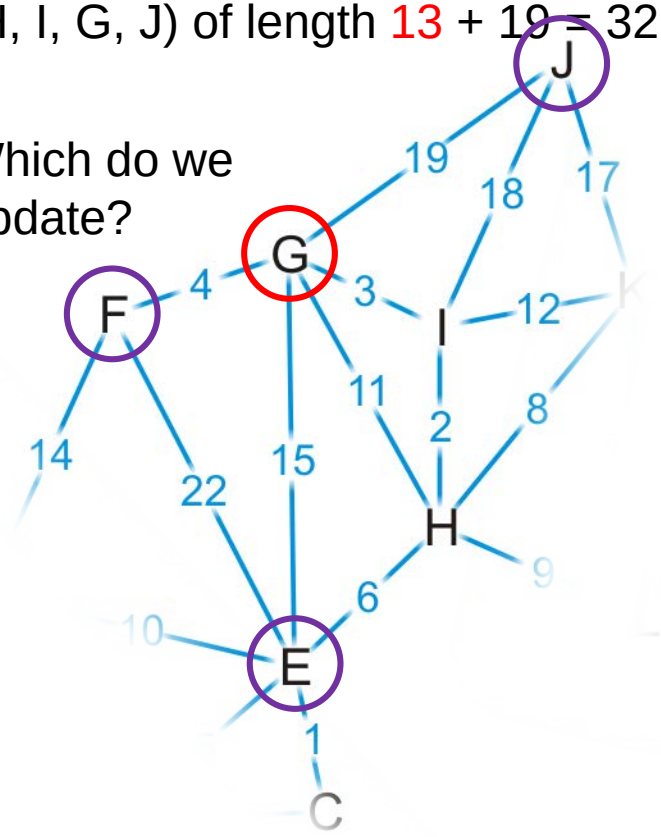
Consider these paths:

(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

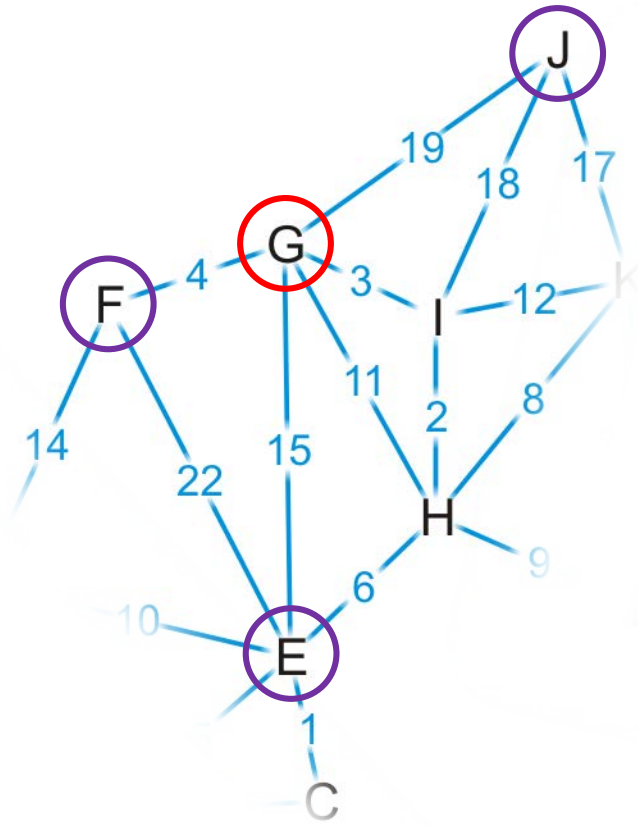
– Which do we update?



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

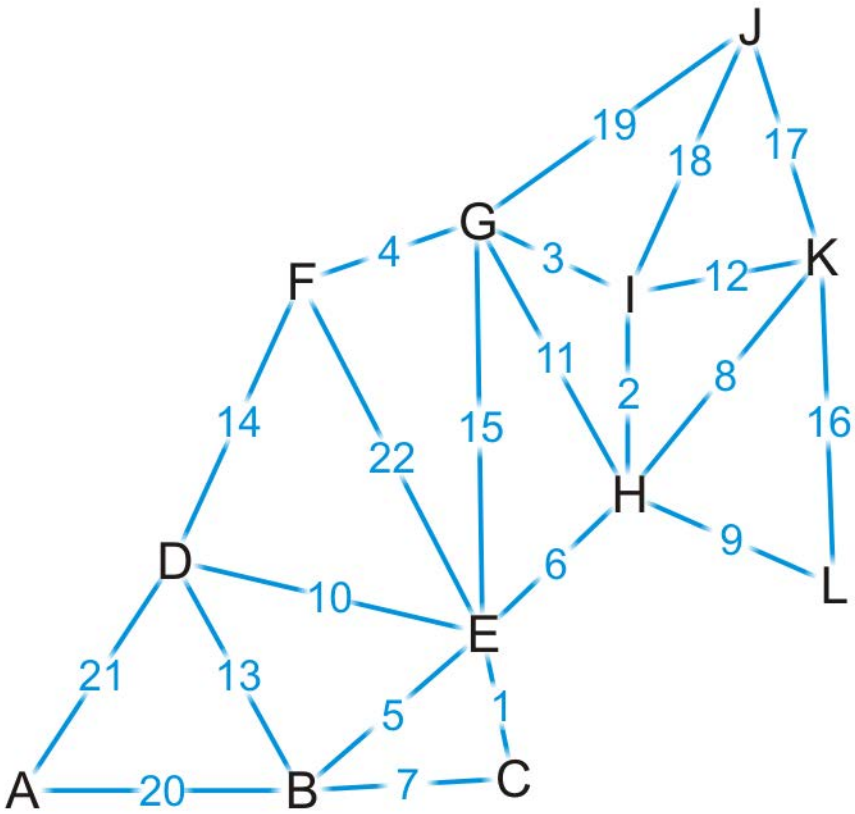
We have now found a path to vertex F



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?

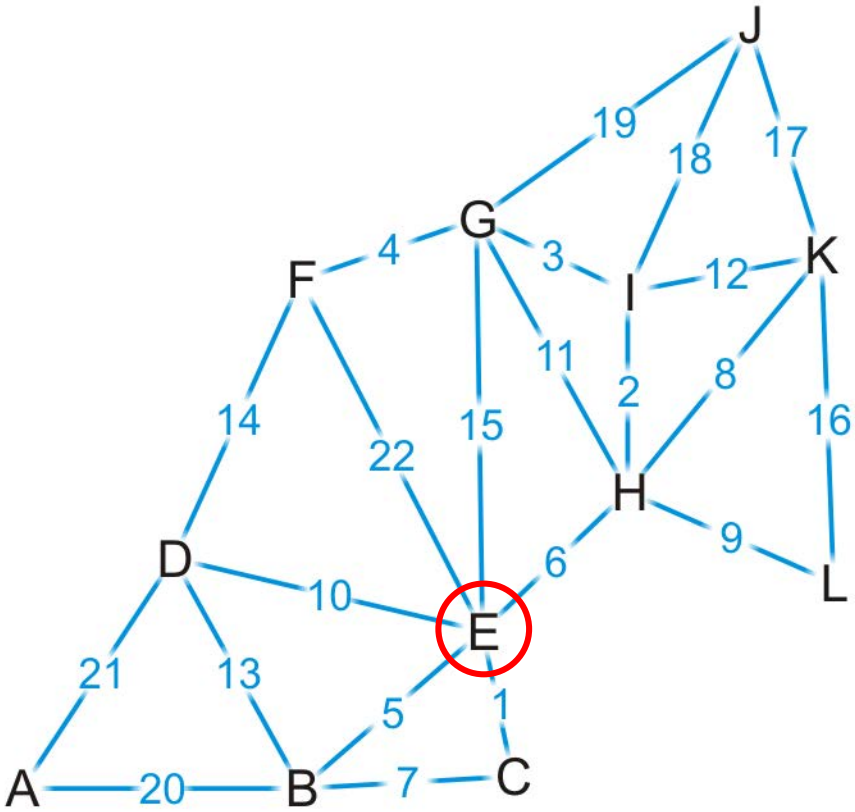


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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

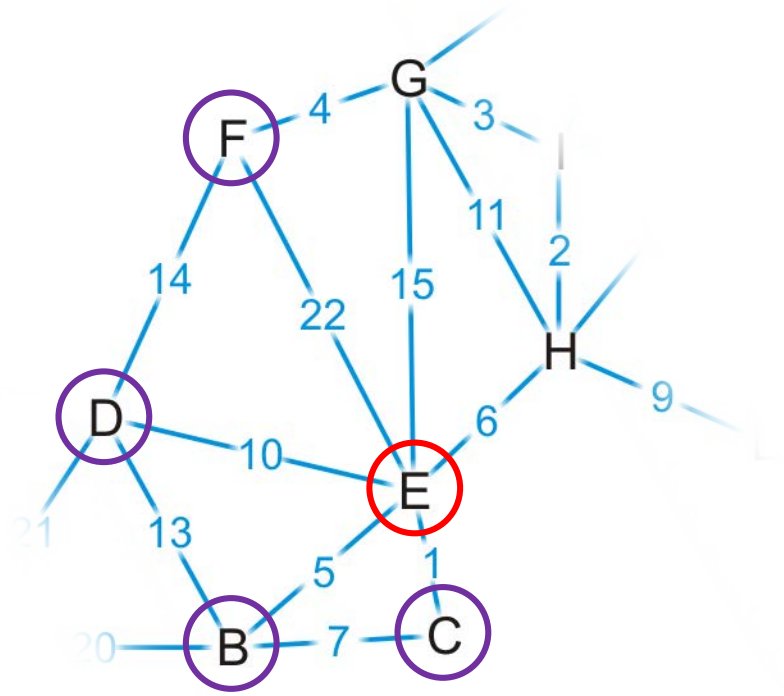


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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:

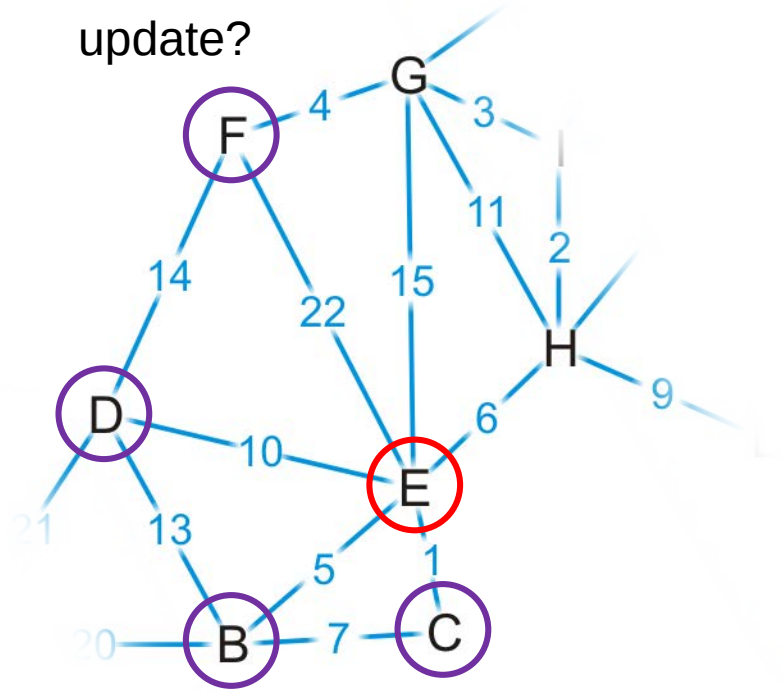
(K, H, E, B) of length 14 + 5 = 19

(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

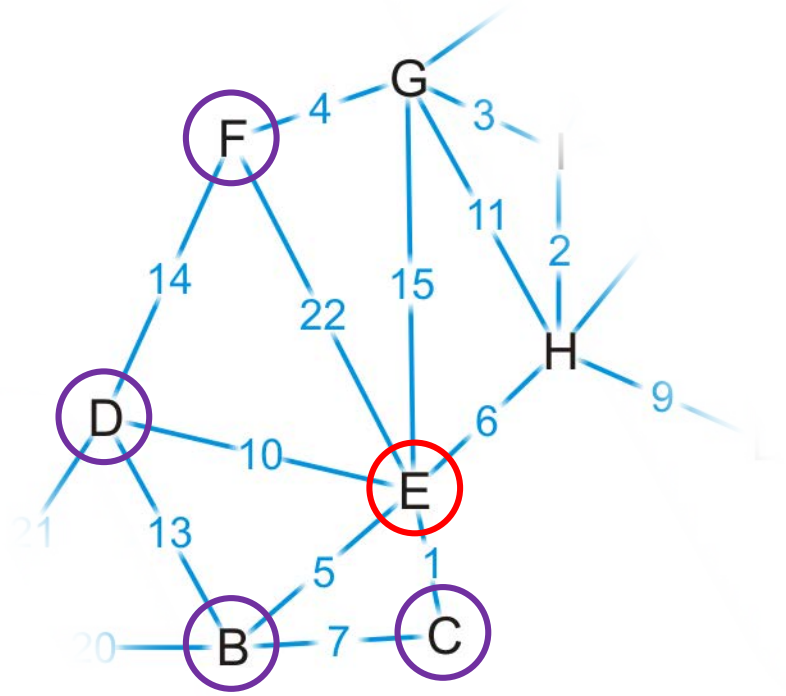
– Which do we update?



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

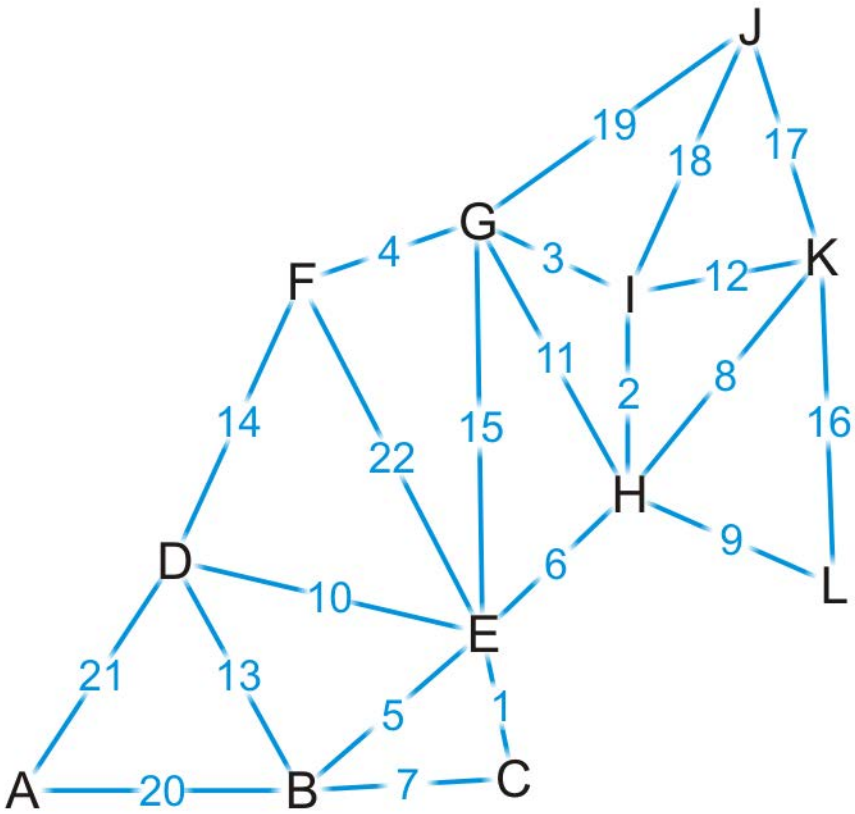
We've discovered paths to vertices B, C, D



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?

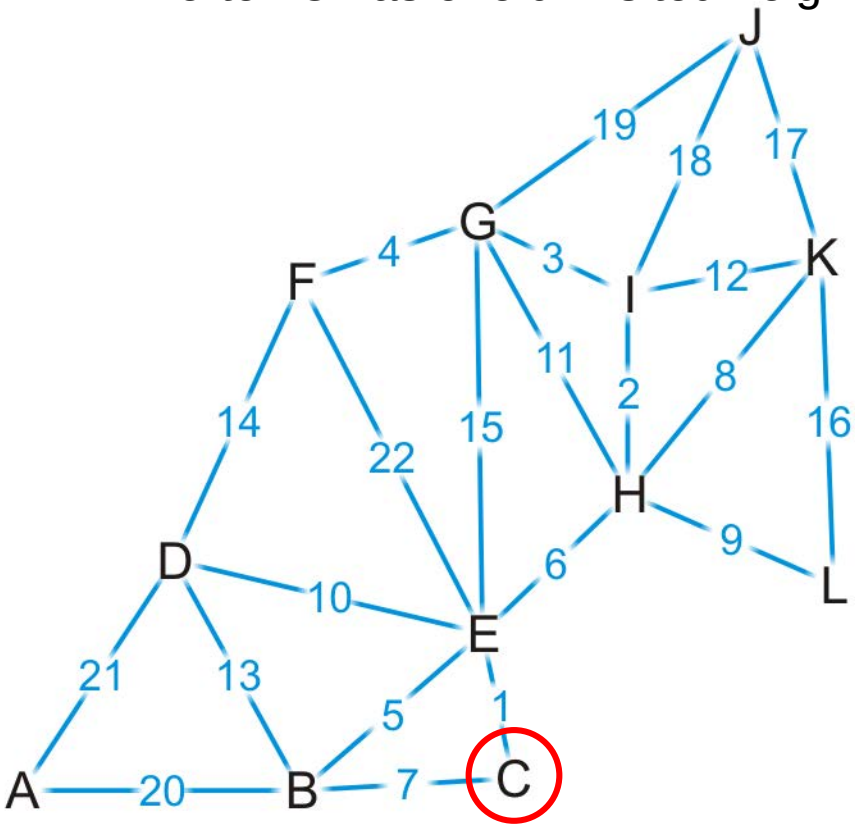


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	F	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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

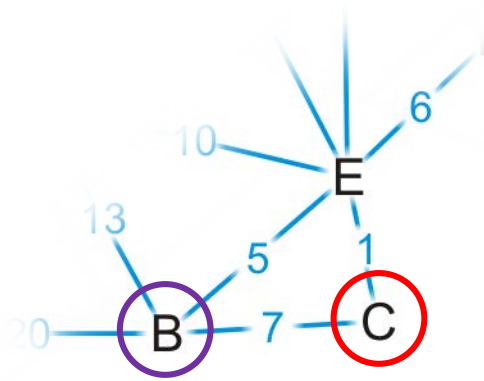


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	**T**	**15**	**E**
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

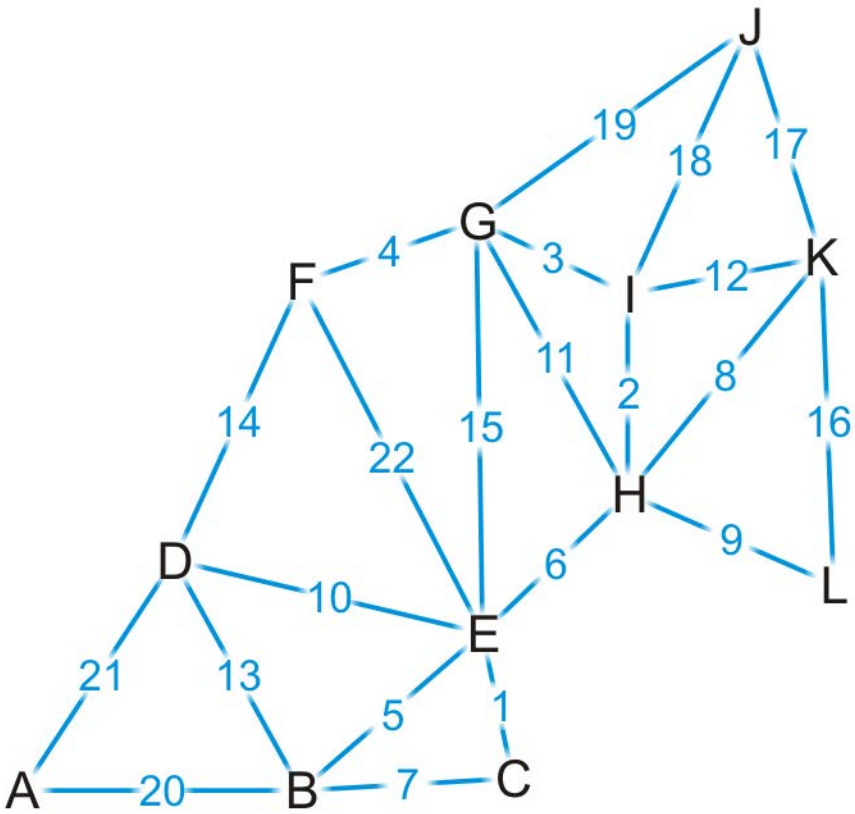
- We have already discovered a shorter path through vertex E



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?

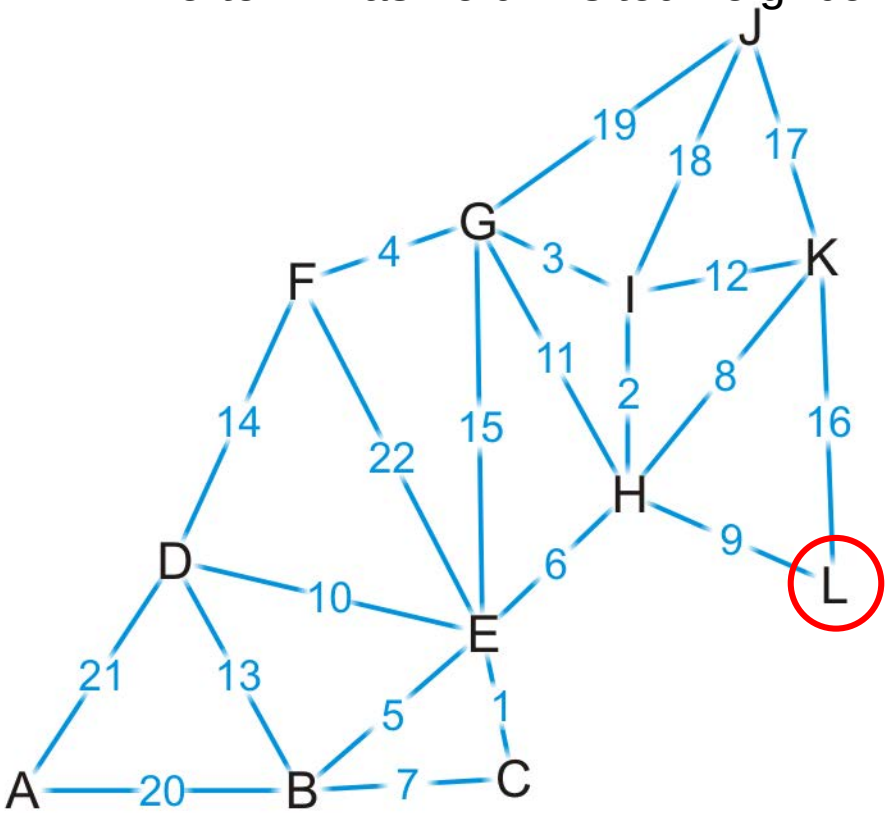


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

- Vertex L has no unvisited neighbors

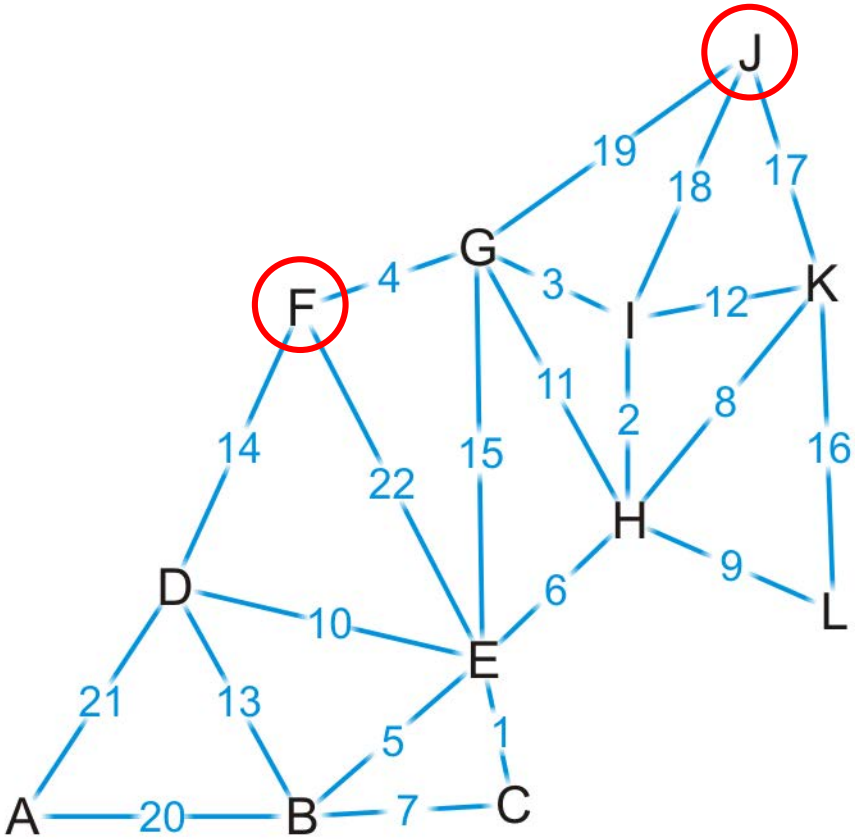


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	**T**	**16**	**K**

Example

Where to next?

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

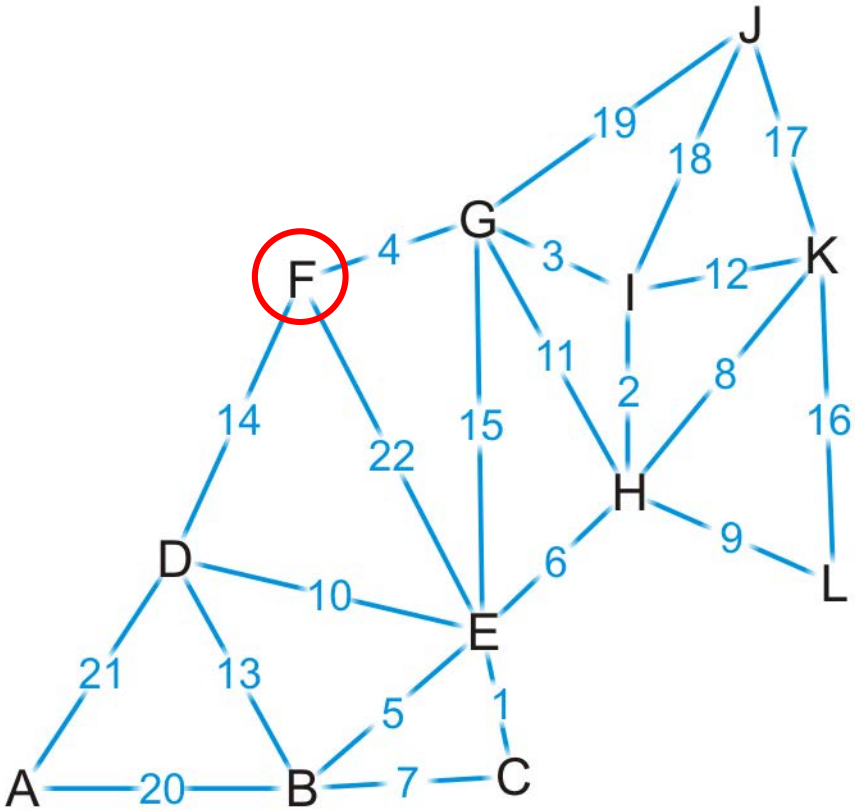


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Let's visit vertex F first

- It has one unvisited neighbor, vertex D

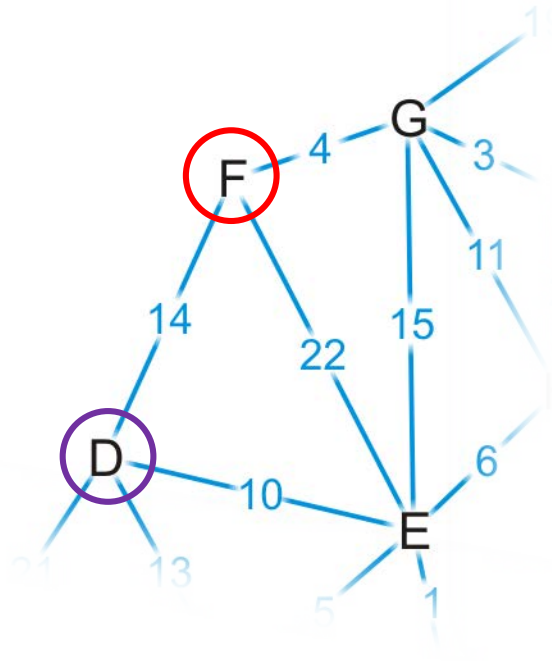


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

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

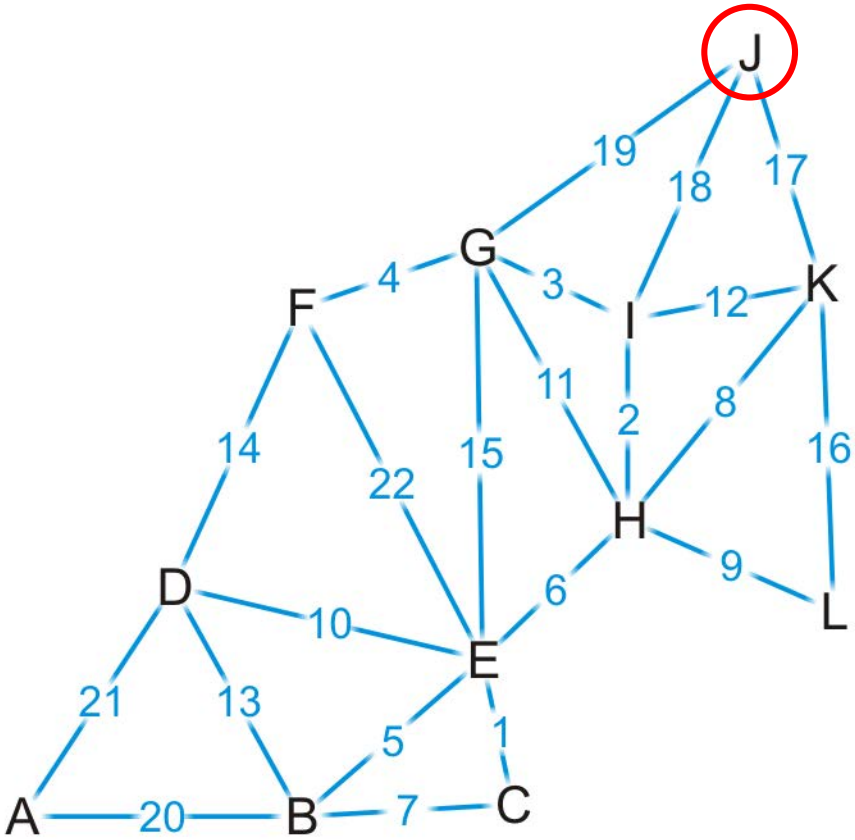


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Now we visit vertex J

– It has no unvisited neighbors



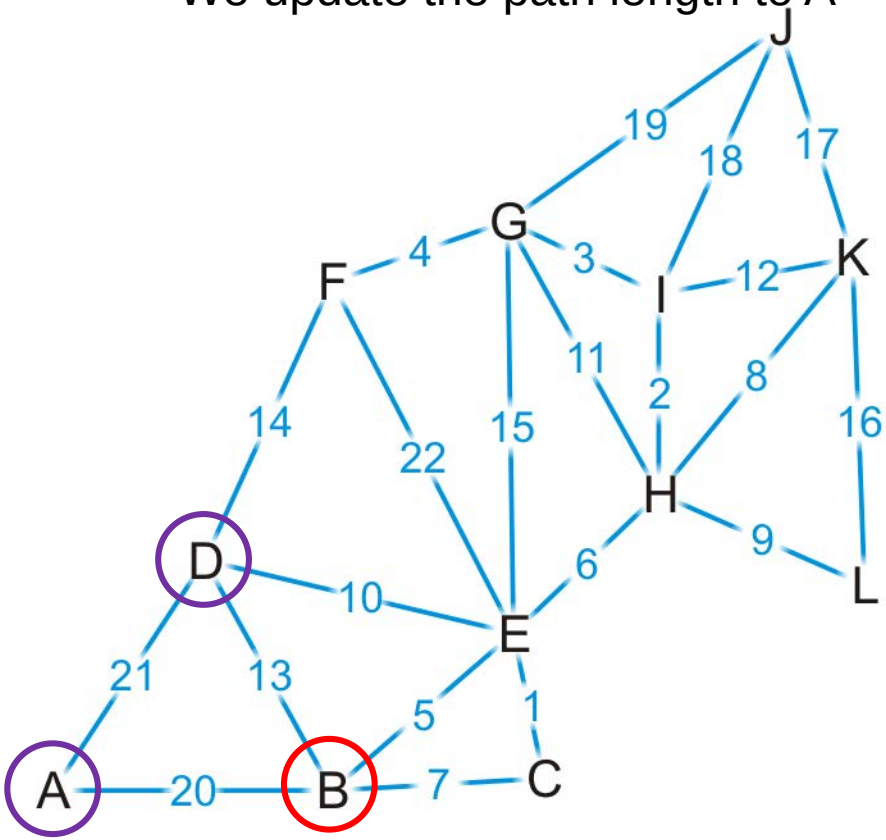
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	**T**	**17**	**K**
K	T	0	Ø
L	T	16	K

Example

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

(K, H, E, B, A) of length 19 + 20 = 39 (K, H, E, B, D) of length 19 + 13 = 32

– We update the path length to A

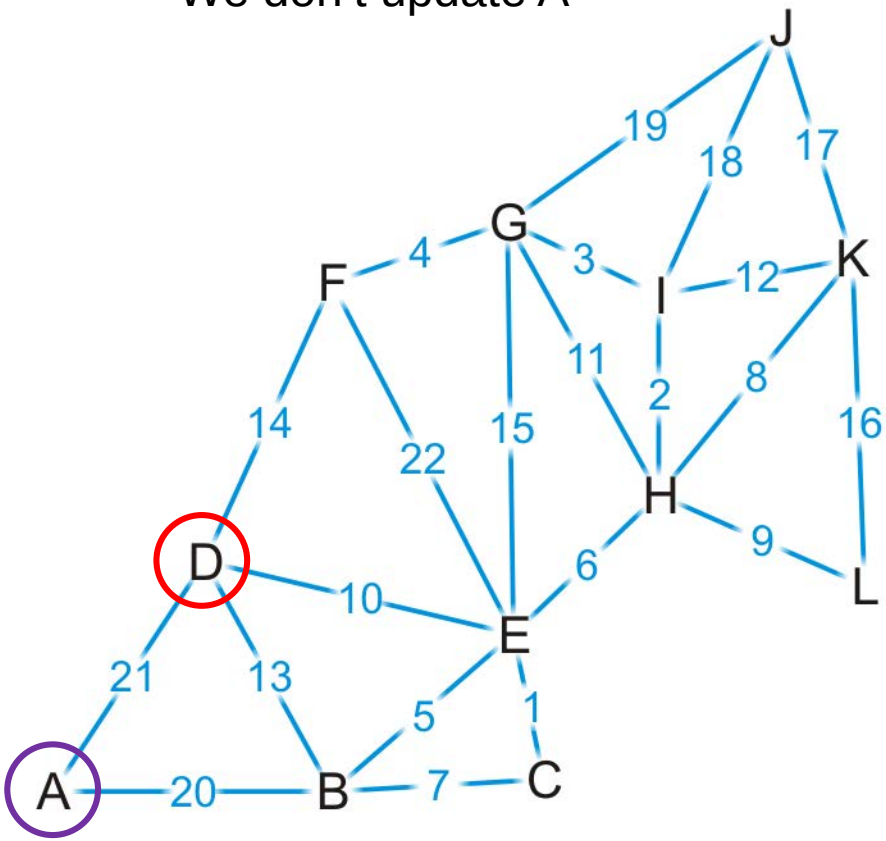


Vertex	Visited	Distance	Previous
A	F	**39**	**B**
B	**T**	**19**	**E**
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Next we visit vertex D

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

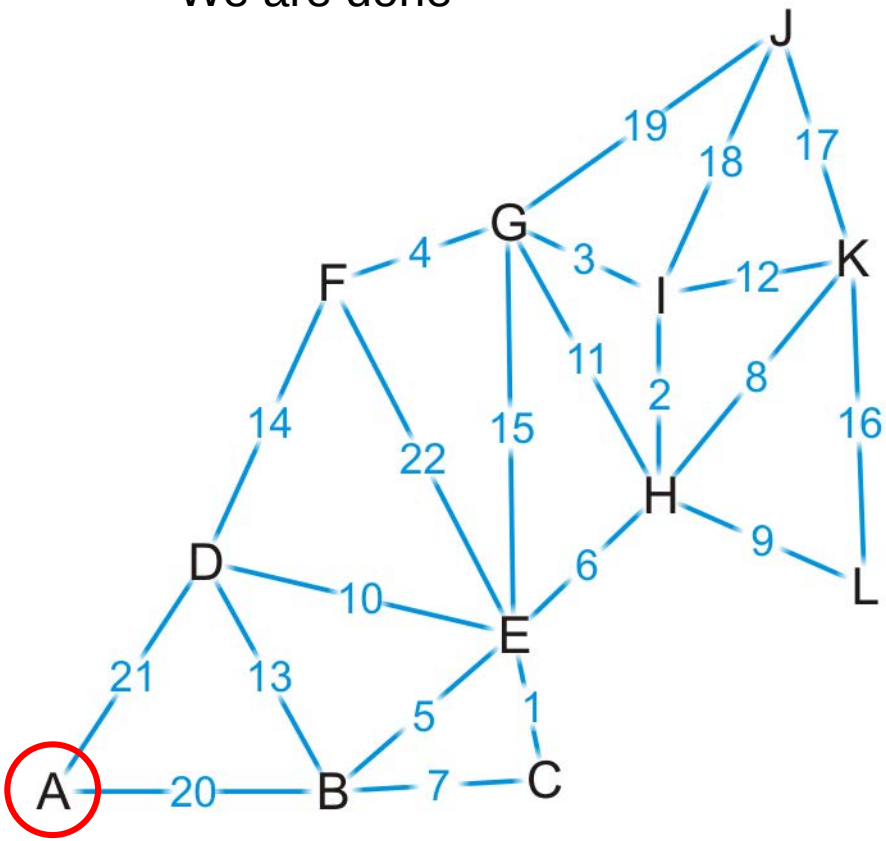


Vertex	Visited	Distance	Previous
A	F	39	B
B	T	19	E
C	T	15	E
D	**T**	**24**	**E**
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Finally, we visit vertex A

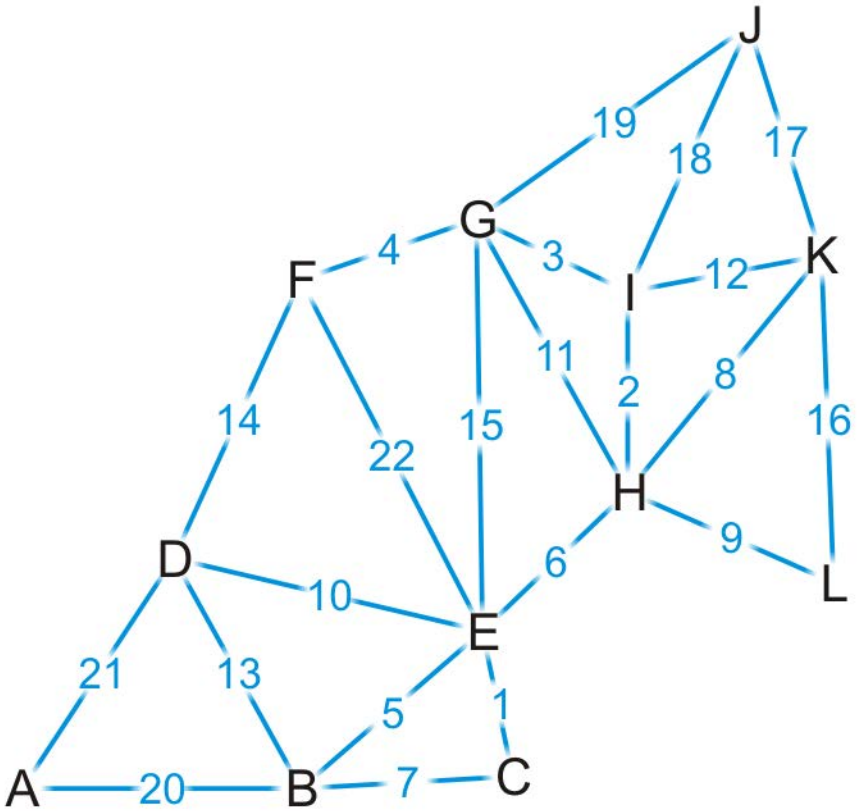
- It has no unvisited neighbors and there are no unvisited vertices left
- We are done



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

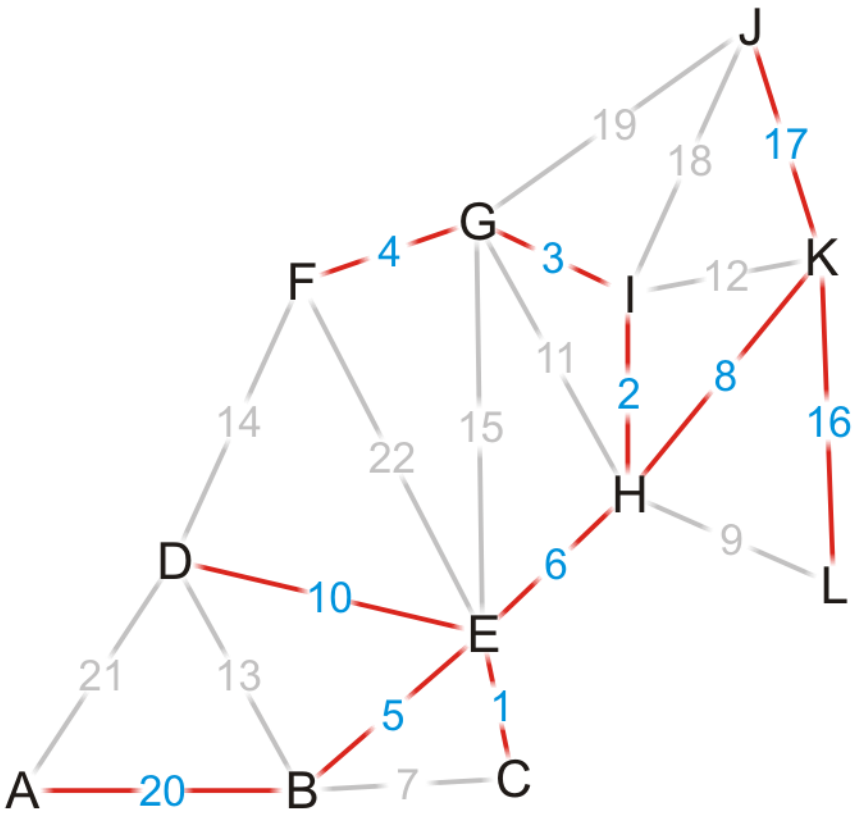
Thus, we have found the shortest path from vertex K to each of the other vertices



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

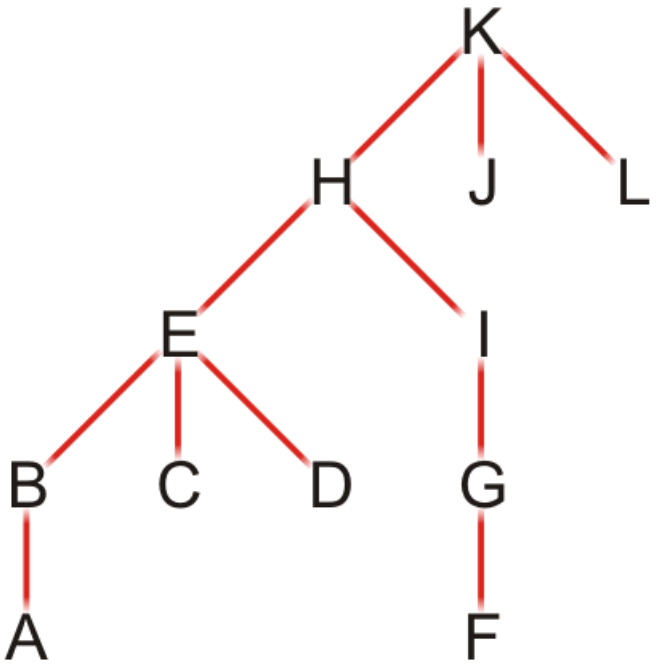


Vertex	Visited	Distance	Previous
A	T	39	B
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Note that this 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

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 visiting 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
- With an adjacency matrix, the run time is $\Theta(|V|(|V| + |V|)) = \Theta(|V|^2)$
- With an adjacency list, the run time is $\Theta(|V|^2 + |E|) = \Theta(|V|^2)$ as $|E| = O(|V|^2)$

Can we do better?

- How about using a priority queue to find the closest vertex?
  - Assume we are using a binary heap

Implementation and analysis

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

- The priority queue will also requires $O(|V|)$ memory
- We must use an adjacency list, not an adjacency matrix

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 distance being updated
- 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

We could use a different heap structure:

- A Fibonacci heap is a node-based heap
- Pop is still $O(\ln(|V|))$, but inserting and moving a key is $\Theta(1)$
- Thus, because we are only calling pop $|V| - 1$ times, the overall run-time reduces to $O(|E| + |V| \ln(|V|))$

Implementation and analysis

Thus, we have two run times when using

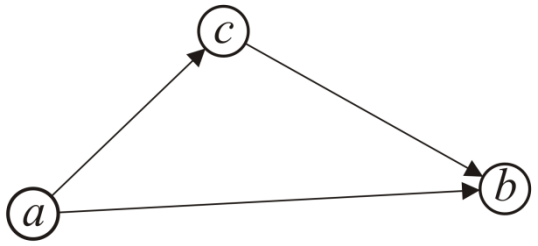
- A binary heap: $O(|E| \ln(|V|))$
- A Fibonacci heap: $O(|E| + |V| \ln(|V|))$

Questions: Which is faster if $|E| = \Theta(|V|)$? How about if $|E| = \Theta(|V|^2)$?

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,

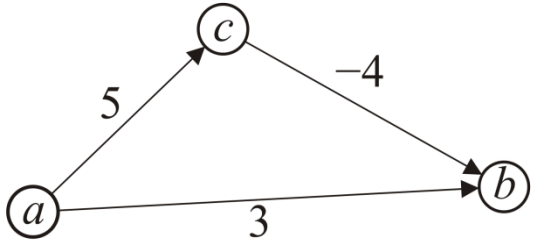


we can use the A* search which is faster than Dijkstra's algorithm

- 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

Outline

- Definition and applications
- Dijkstra’s algorithm
- Floyd-Warshall algorithm

Background

Dijkstra's algorithm finds the shortest path between two nodes

- Run time: $O(|E| \ln(|V|))$

If we wanted to find the shortest path between all pairs of nodes, we could apply Dijkstra's algorithm to each vertex:

- Run time: $O(|V| |E| \ln(|V|))$

Background

Now, Dijkstra's algorithm has the following run times:

- Best case:

  If $|E| = \Theta(|V|)$, running Dijkstra for each vertex is $O(|V|^2 \ln(|V|))$

- Worst case:

  If $|E| = \Theta(|V|^2)$, running Dijkstra for each vertex is $O(|V|^3 \ln(|V|))$

Problem

Question: for the worst case, can we find a $o\left(|V|^3 \ln |V|\right)$ algorithm?

We will look at the Floyd-Warshall algorithm

- It works with positive or negative weights with no negative cycle

Strategy

First, let's consider only edges that connect vertices directly:

$$d_{i,j}^{(0)} = \begin{cases} 0 & \text{If } i = j \\ w_{i,j} & \text{If there is an edge from } i \text{ to } j \\ \infty & \text{Otherwise} \end{cases}$$

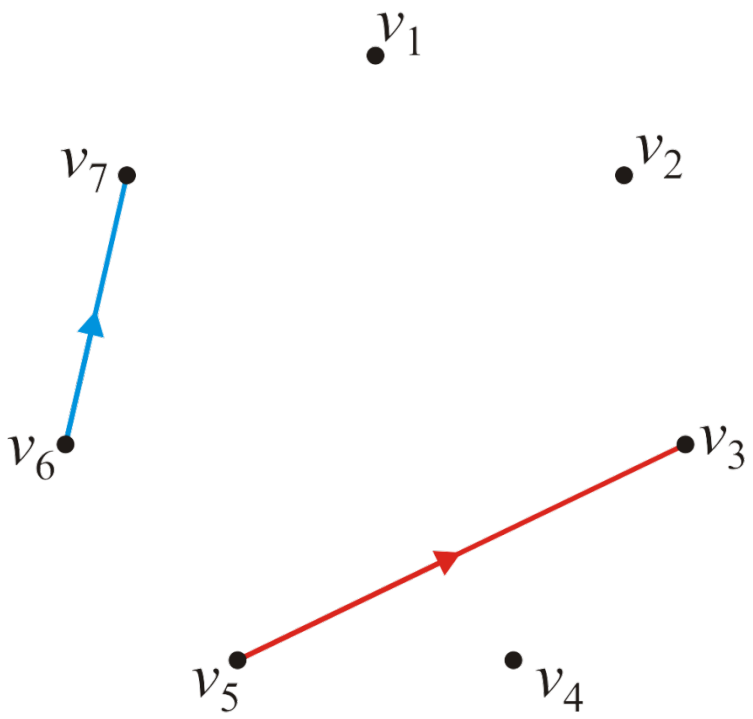
Here, $w_{i,j}$ is the weight of the edge connecting vertices i and j

- Note, this can be a directed graph; *i.e.*, it may be that $d_{i,j}^{(0)} \neq d_{j,i}^{(0)}$

Strategy

Consider this graph of seven vertices

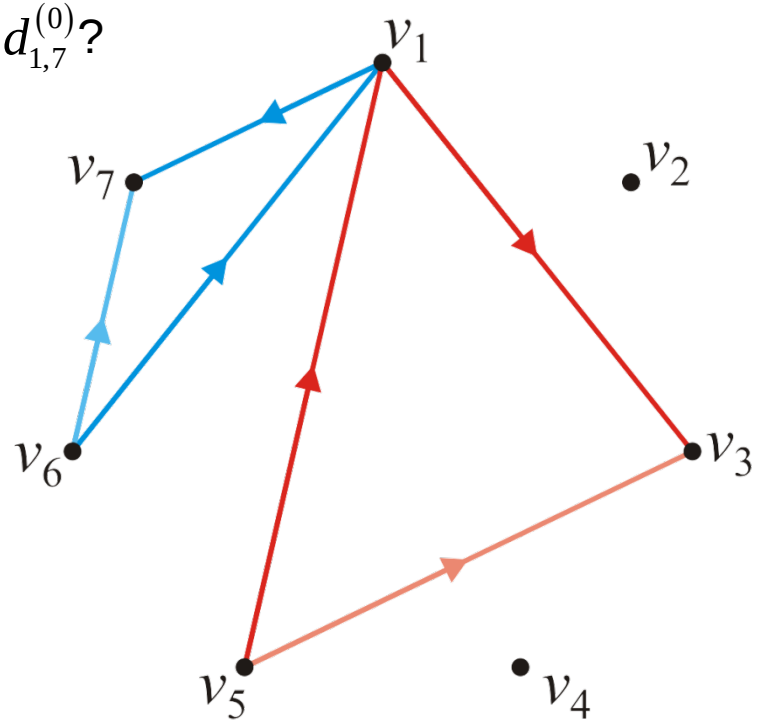
– The edges defining the values $d_{5,3}^{(0)}$ and $d_{6,7}^{(0)}$ are highlighted



Strategy

Suppose now, we want to see whether or not the path going through vertex v_1 is shorter than a direct edge?

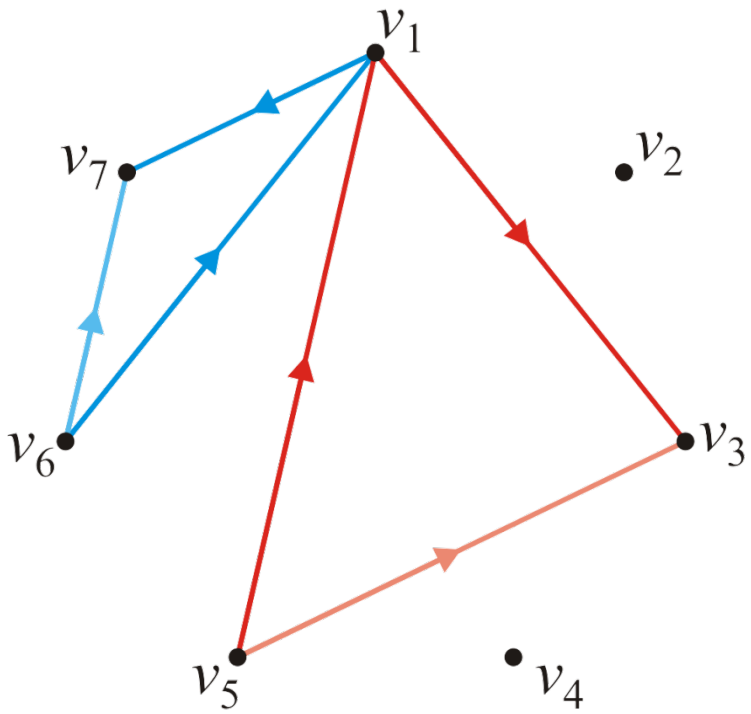
- Is $d_{5,3}^{(0)} > d_{5,1}^{(0)} + d_{1,3}^{(0)}$?
- Is $d_{6,7}^{(0)} > d_{6,1}^{(0)} + d_{1,7}^{(0)}$?



Strategy

Thus, for each pair of edges, we will define $d_{i,j}^{(1)}$ by calculating:

$$d_{i,j}^{(1)} = \min\left\{ d_{i,j}^{(0)}, d_{i,1}^{(0)} + d_{1,j}^{(0)} \right\}$$



Strategy

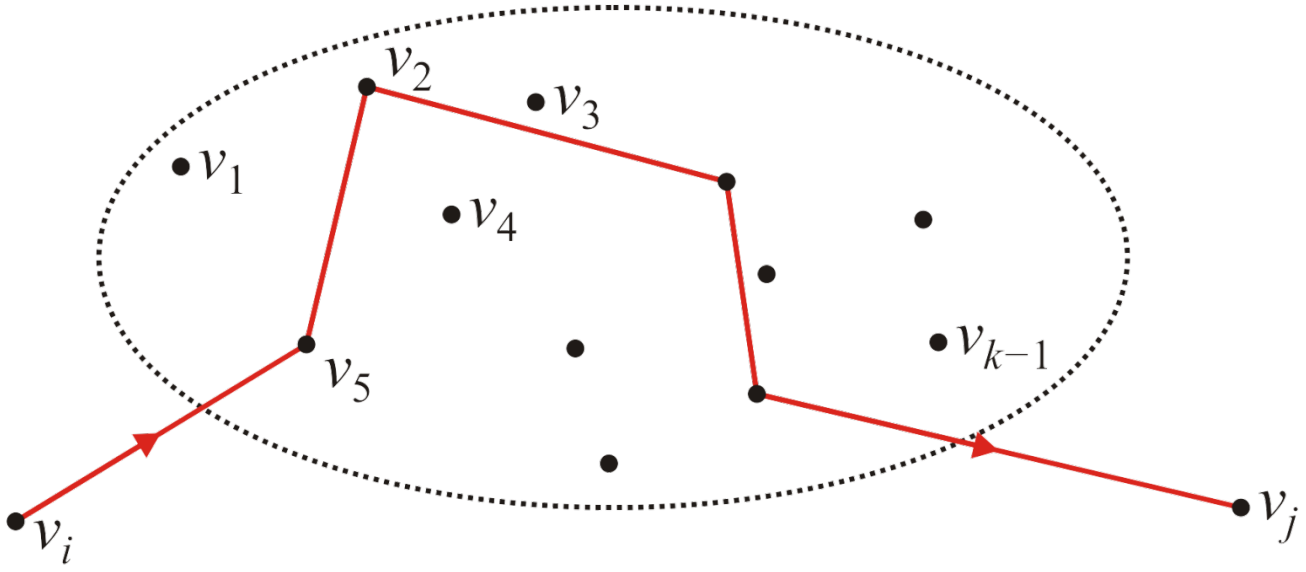
We need just run the algorithm for each pair of vertices:

```
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] );
    }
}
```

The General Step

Define $d_{i,j}^{(k-1)}$ as the shortest distance, but only allowing intermediate visits to vertices v_1, v_2, …, v_{k-1}

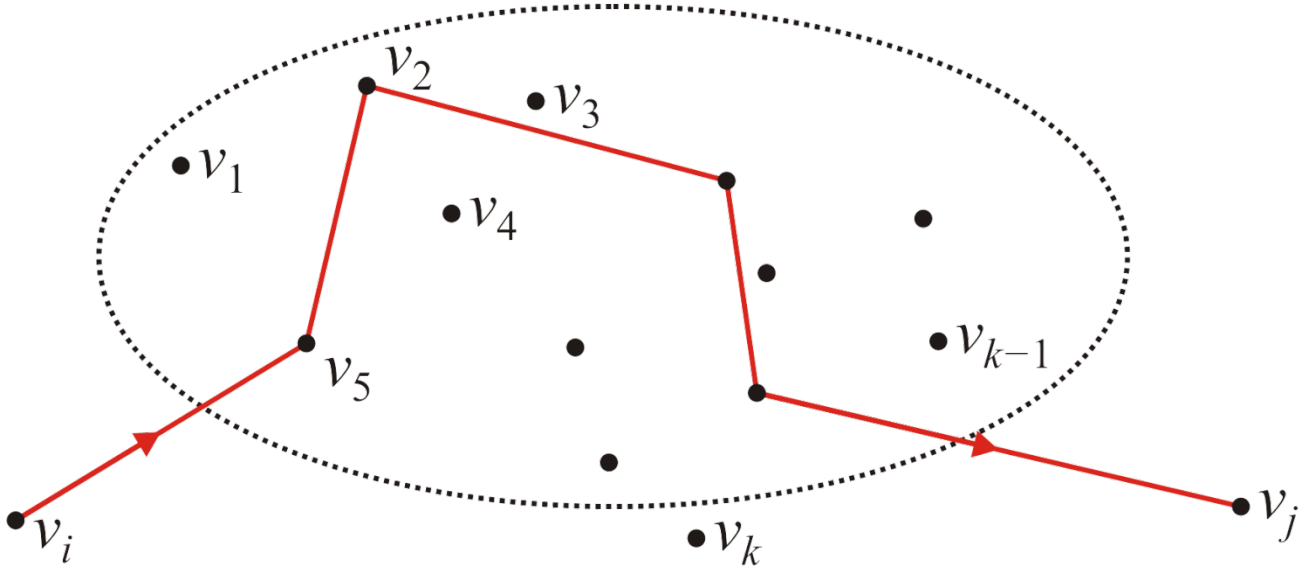
- Suppose we have an algorithm that has found these values for all pairs



The General Step

How could we find $d_{i,j}^{(k)}$; that is, the shortest path allowing intermediate visits to vertices v_1, v_2, …, v_{k-1} , v_k?

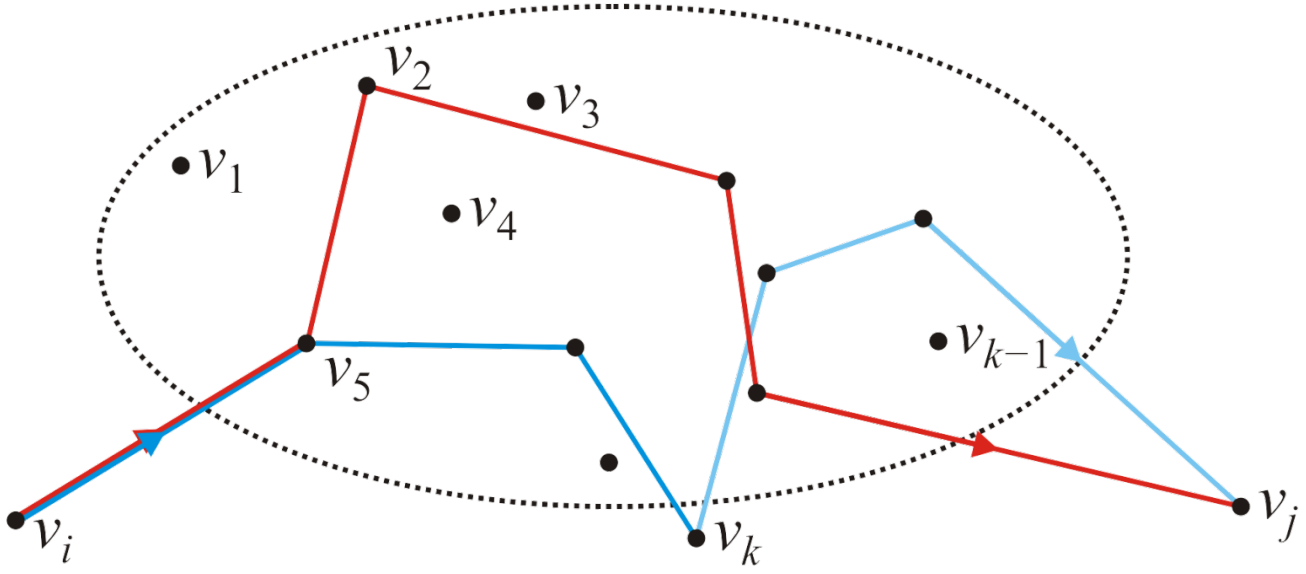
– Two possibilities: the shortest path includes or does not include v_k



The General Step

If the shortest path includes v_k, then it must consist of:

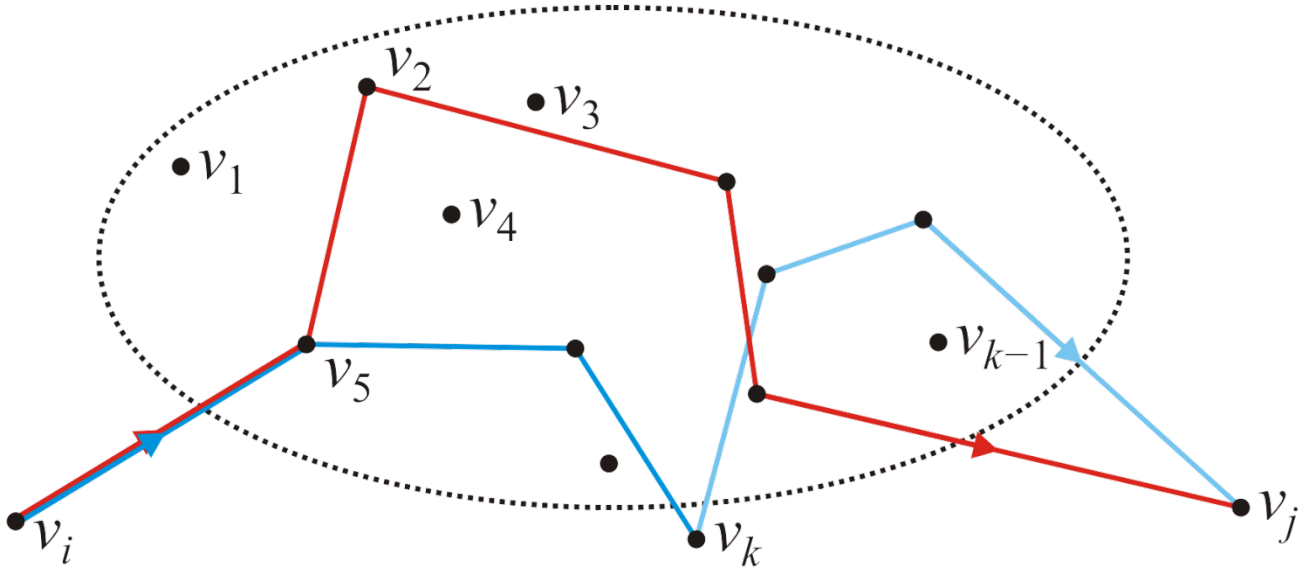
- the shortest path from v_i to v_k
- and then the shortest path from v_k to v_j
- both only allowing intermediate visits to vertices $v_1, v_2, \ldots, v_{k-1}$



The General Step

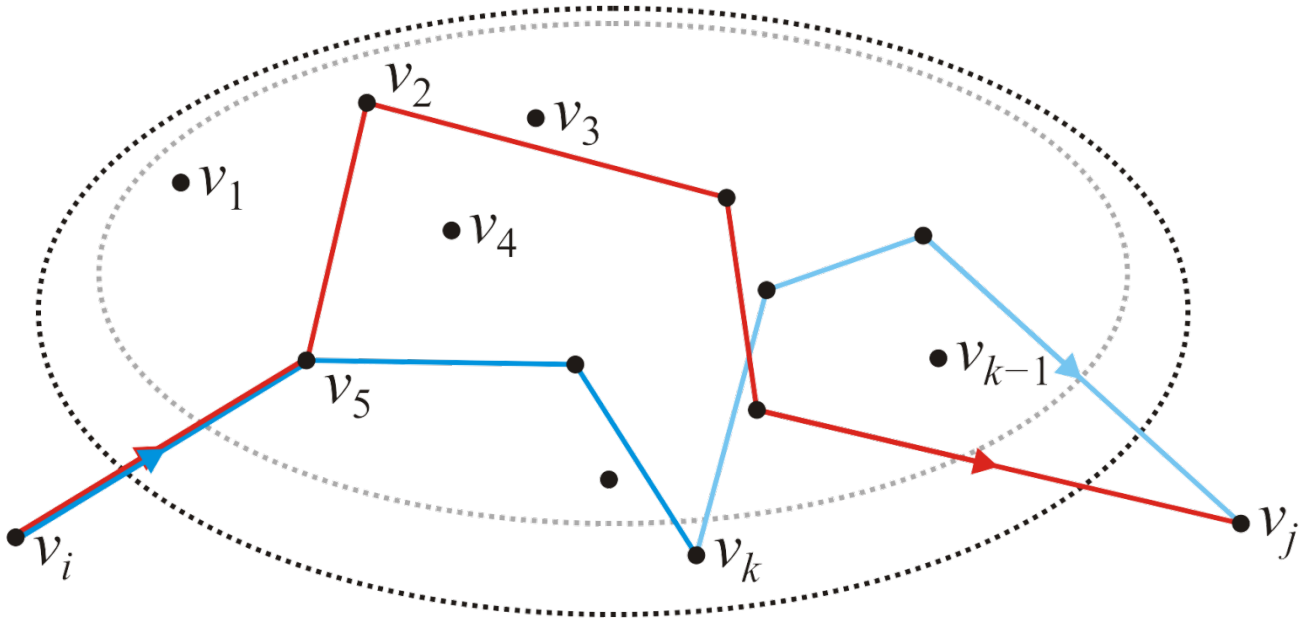
With $v_1, v_2, \ldots, v_{k-1}$ as intermediates, we already know the shortest paths from v_i to v_j, v_i to v_k and v_k to v_j

Thus, we calculate $d_{i,j}^{(k)} = \min\left\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right\}$



The General Step

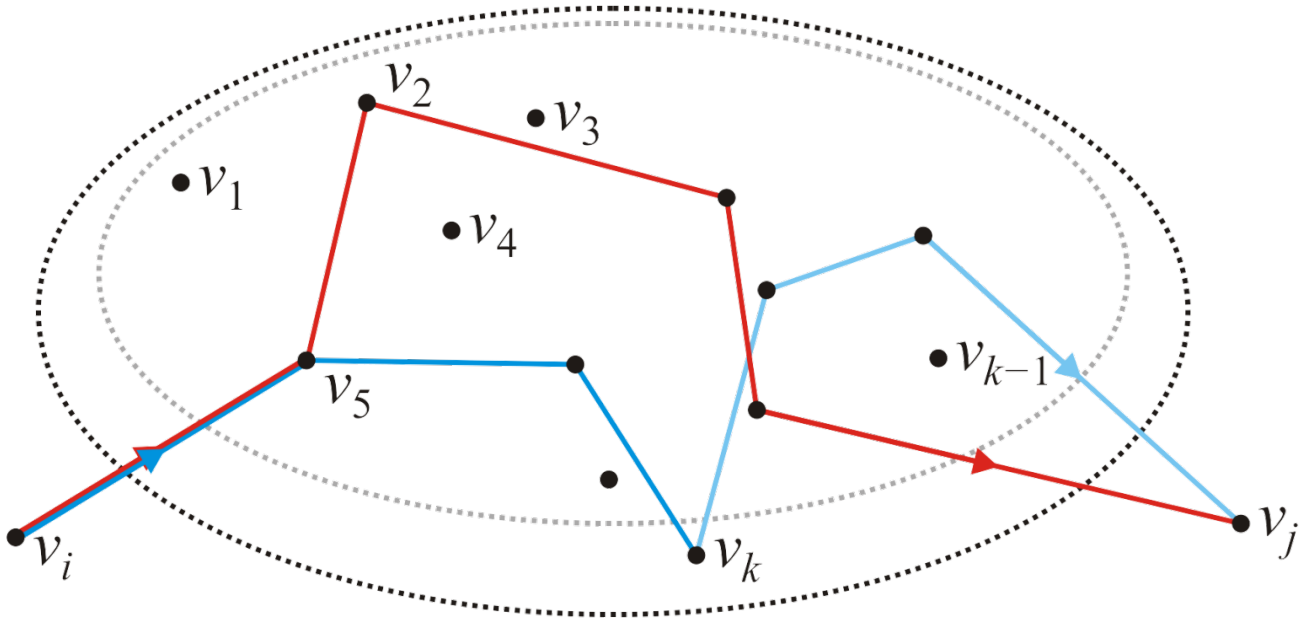
Finding $d_{i,j}^{(k)}$ for all pairs of vertices gives us all shortest paths from v_i to v_j possibly going through vertices v_1, v_2, …, v_k



The General Step

The calculation is straight forward:

```
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] );
    }
}
```



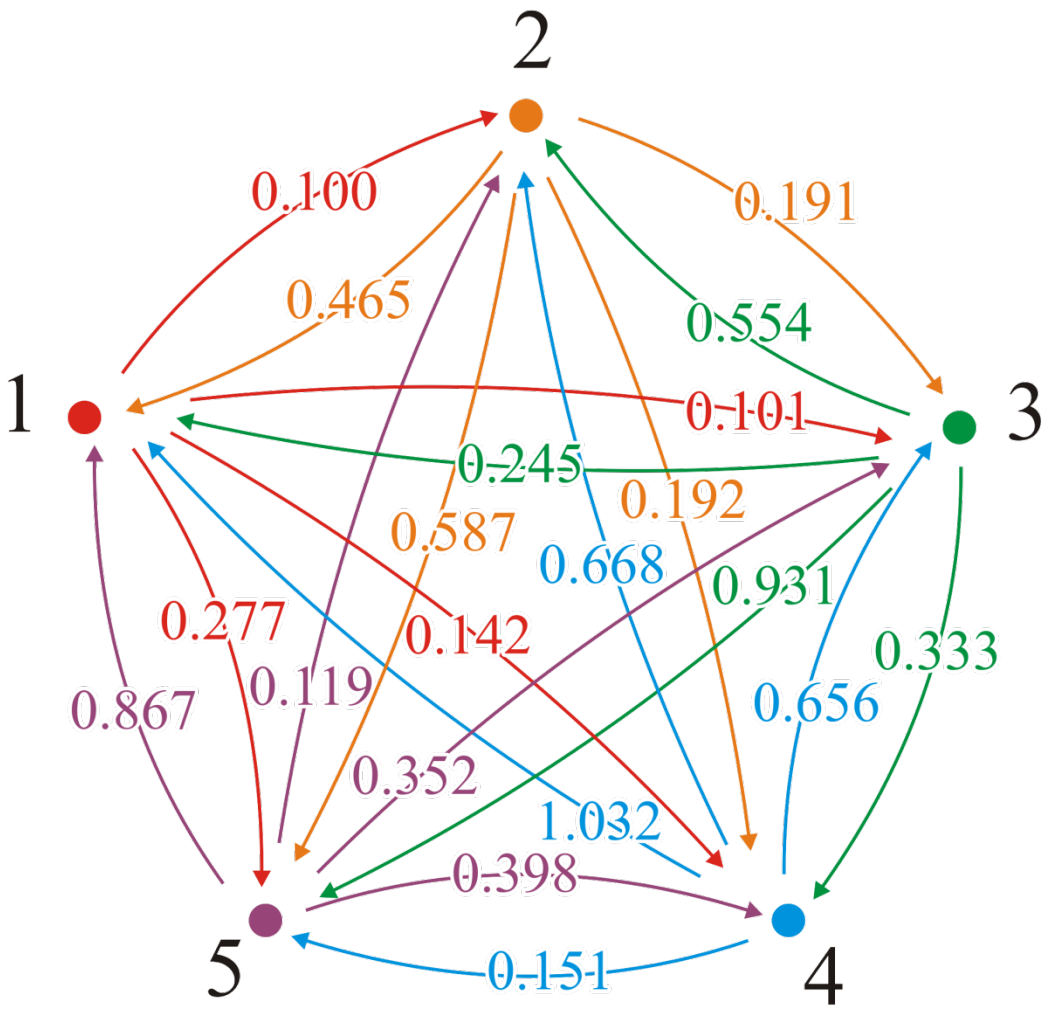
The Floyd-Warshall Algorithm

```
// Initialize the matrix d
//   ...

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] );
        }
    }
}
```

Run time? $\Theta\left(|V|^3\right)$

Example

Consider this graph

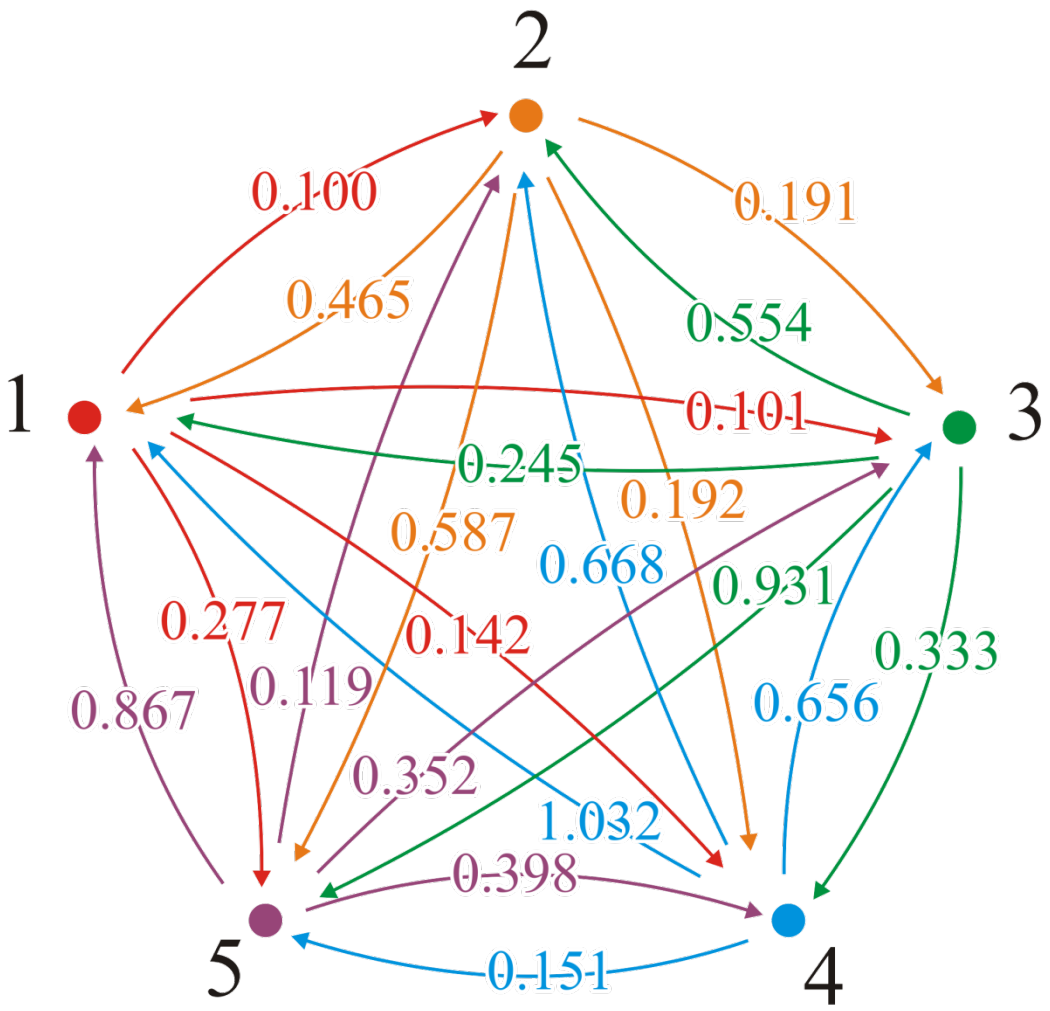


Example

The adjacency matrix is

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

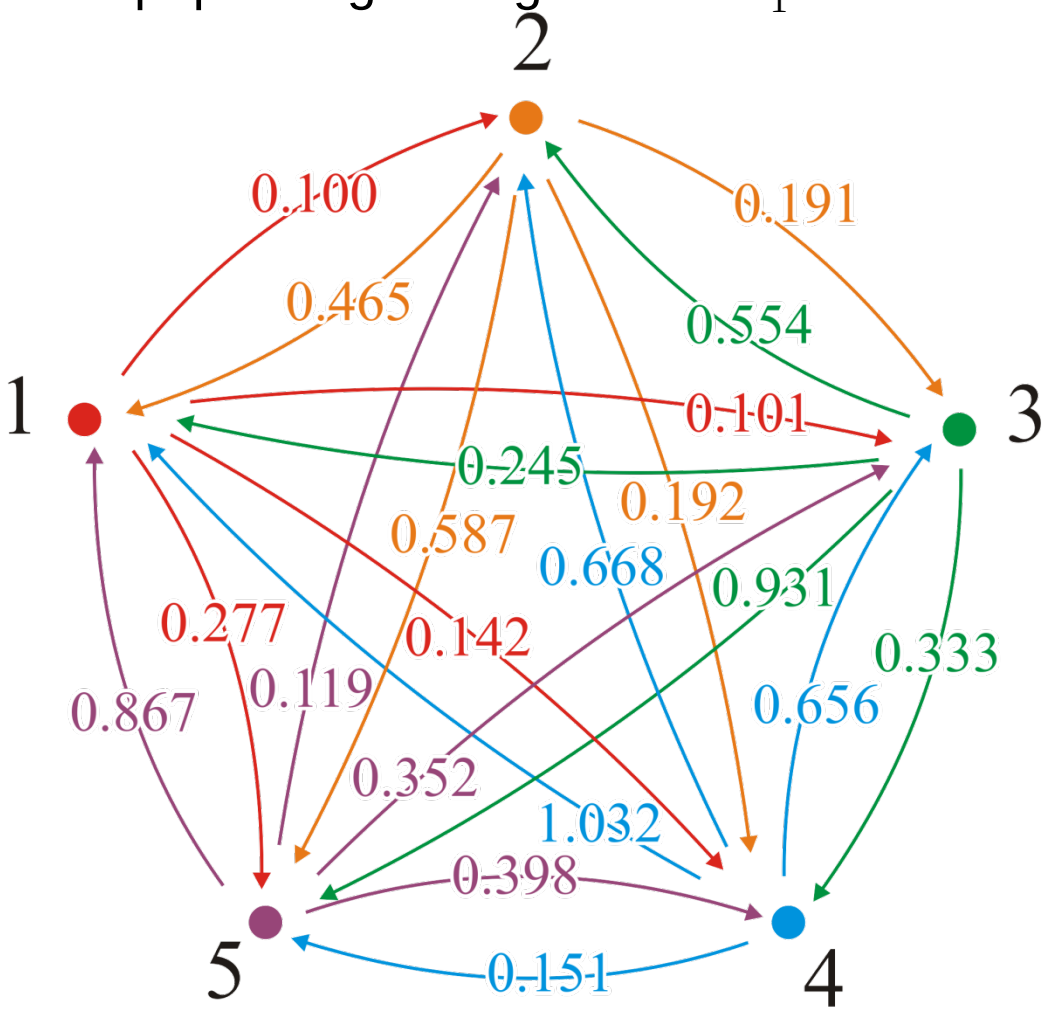
This would define our matrix $\mathbf{D} = (d_{ij})$



Example

With the first pass, $k = 1$, we attempt passing through vertex v_1

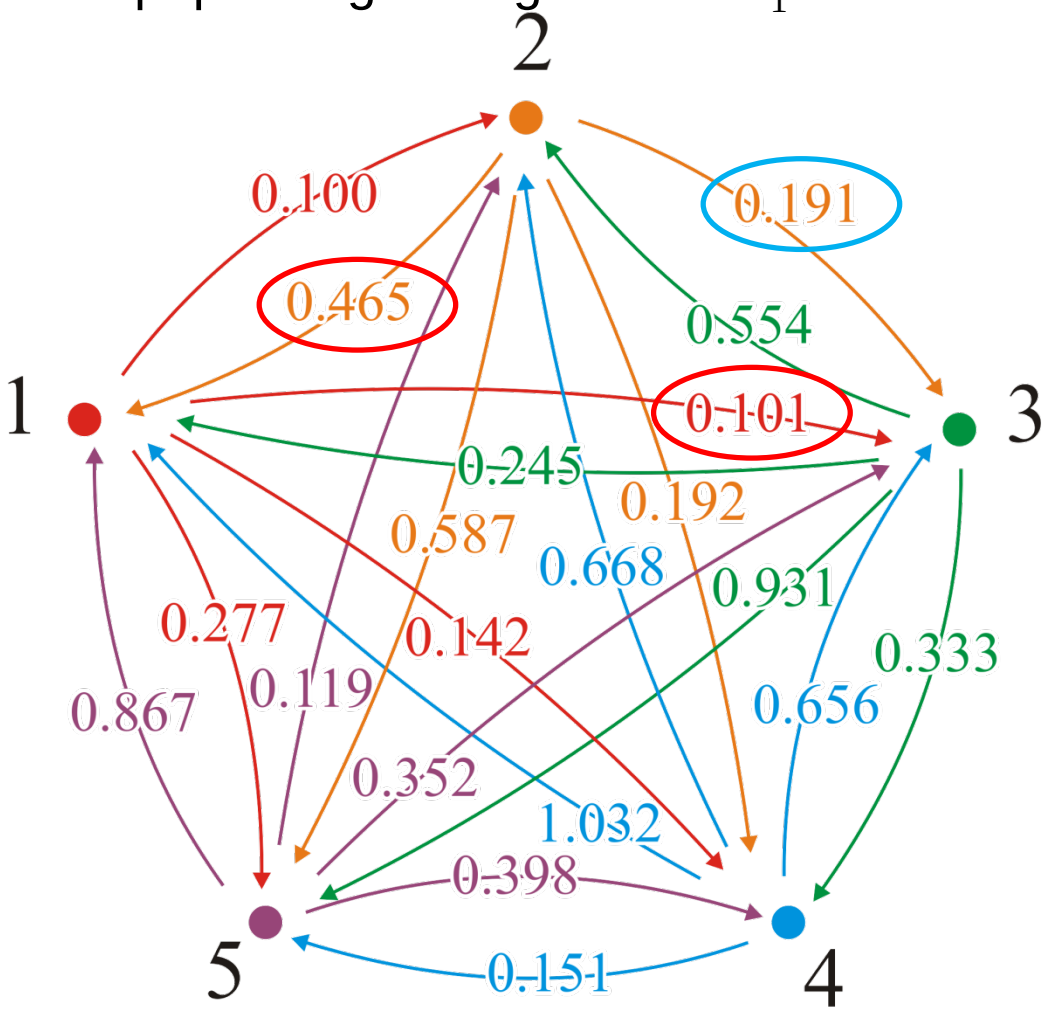
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$



Example

With the first pass, $k = 1$, we attempt passing through vertex v_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 would start:

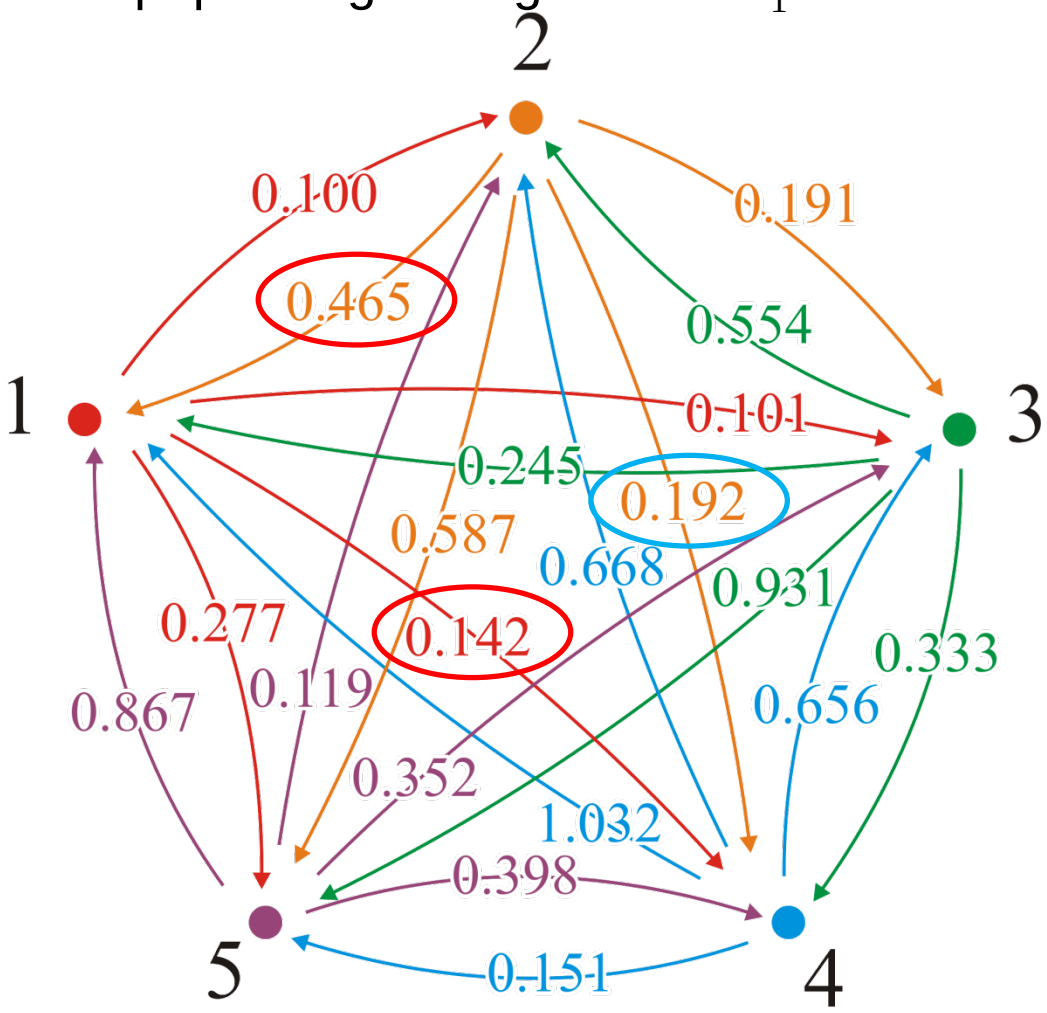
$(2, 3) \rightarrow (2, 1, 3)$

$0.191 \not> 0.465 + 0.101$

Example

With the first pass, $k = 1$, we attempt passing through vertex v_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 would start:

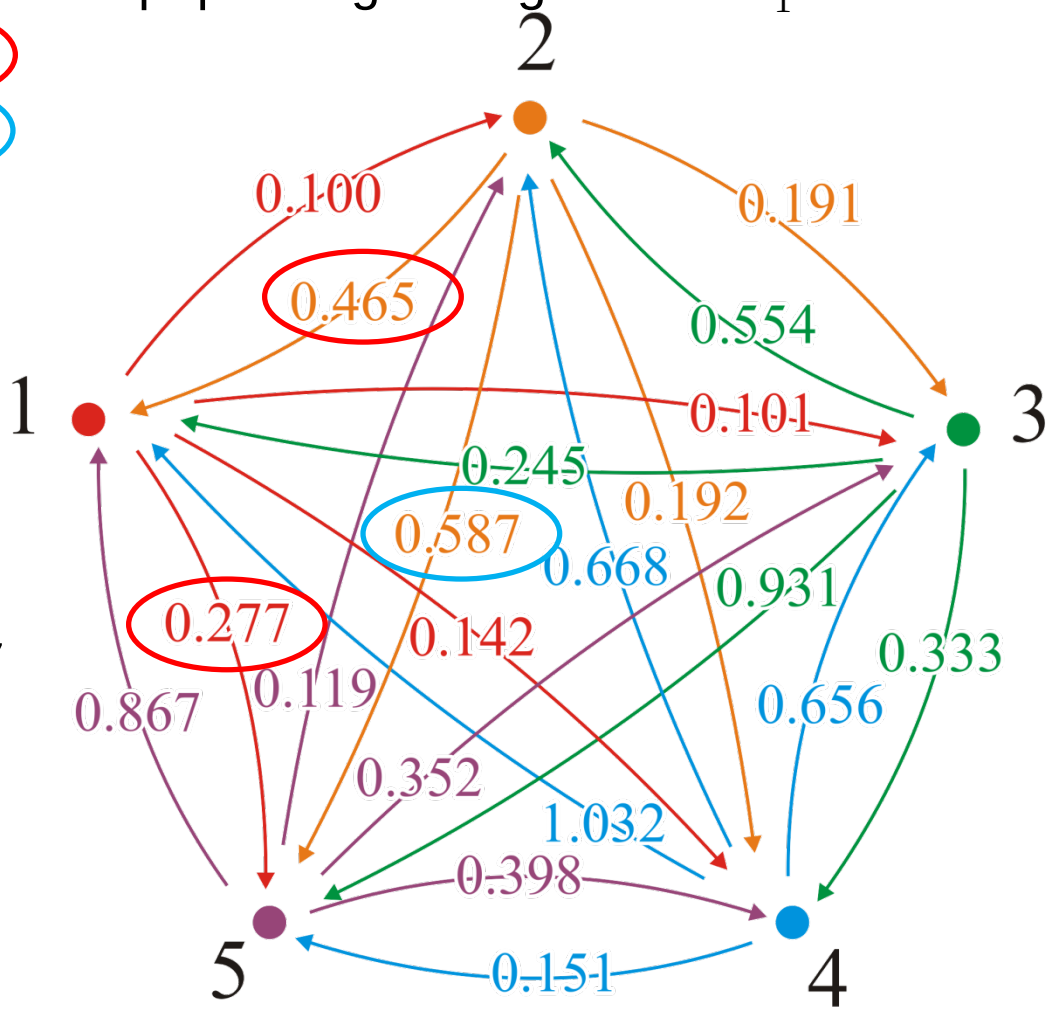
$(2, 4) \rightarrow (2, 1, 4)$

$0.192 \not> 0.465 + 0.142$

Example

With the first pass, $k = 1$, we attempt passing through vertex v_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 would start:

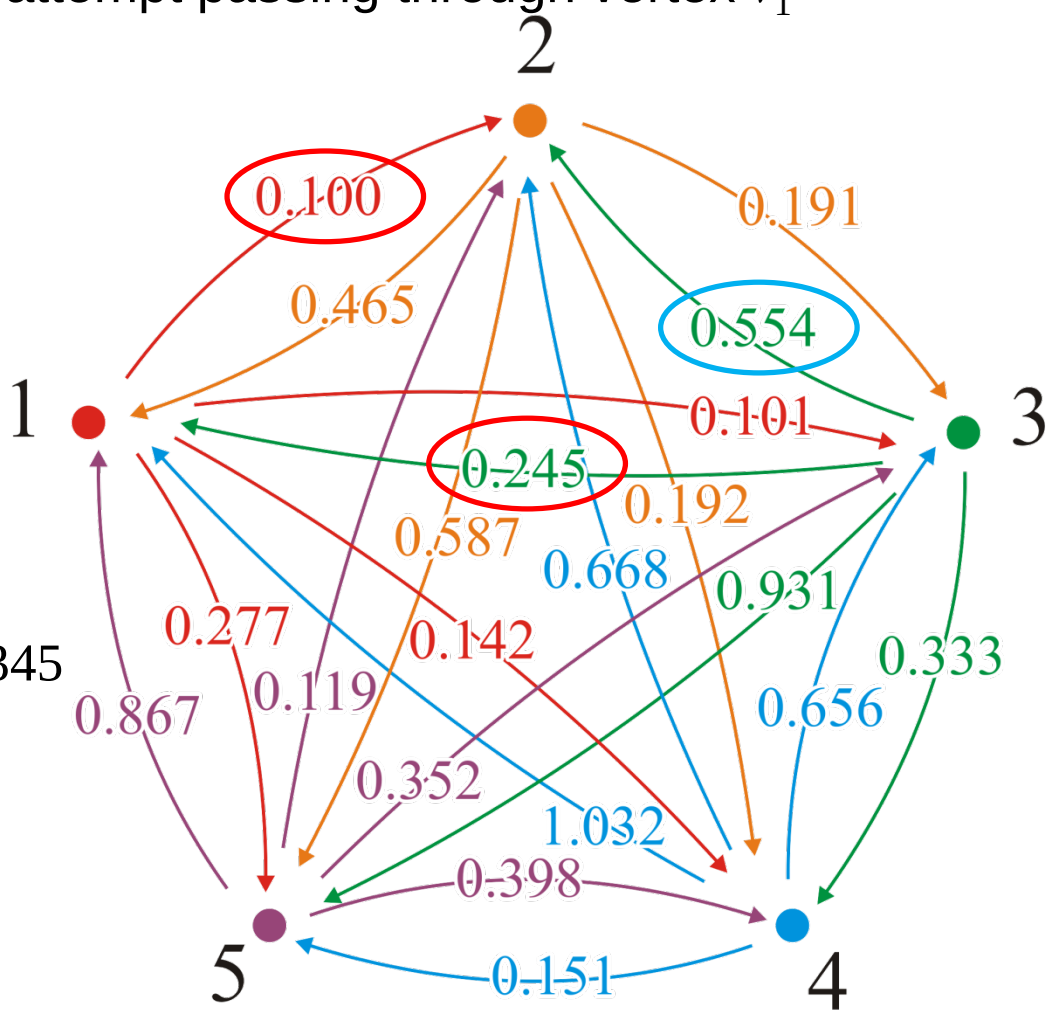
$(2, 5) \rightarrow (2, 1, 5)$

$0.587 \not> 0.465 + 0.277$

Example

With the first pass, $k = 1$, we attempt passing through vertex v_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:

$(3, 2) \rightarrow (3, 1, 2)$

$0.554 > 0.245 + 0.100 = 0.345$

Example

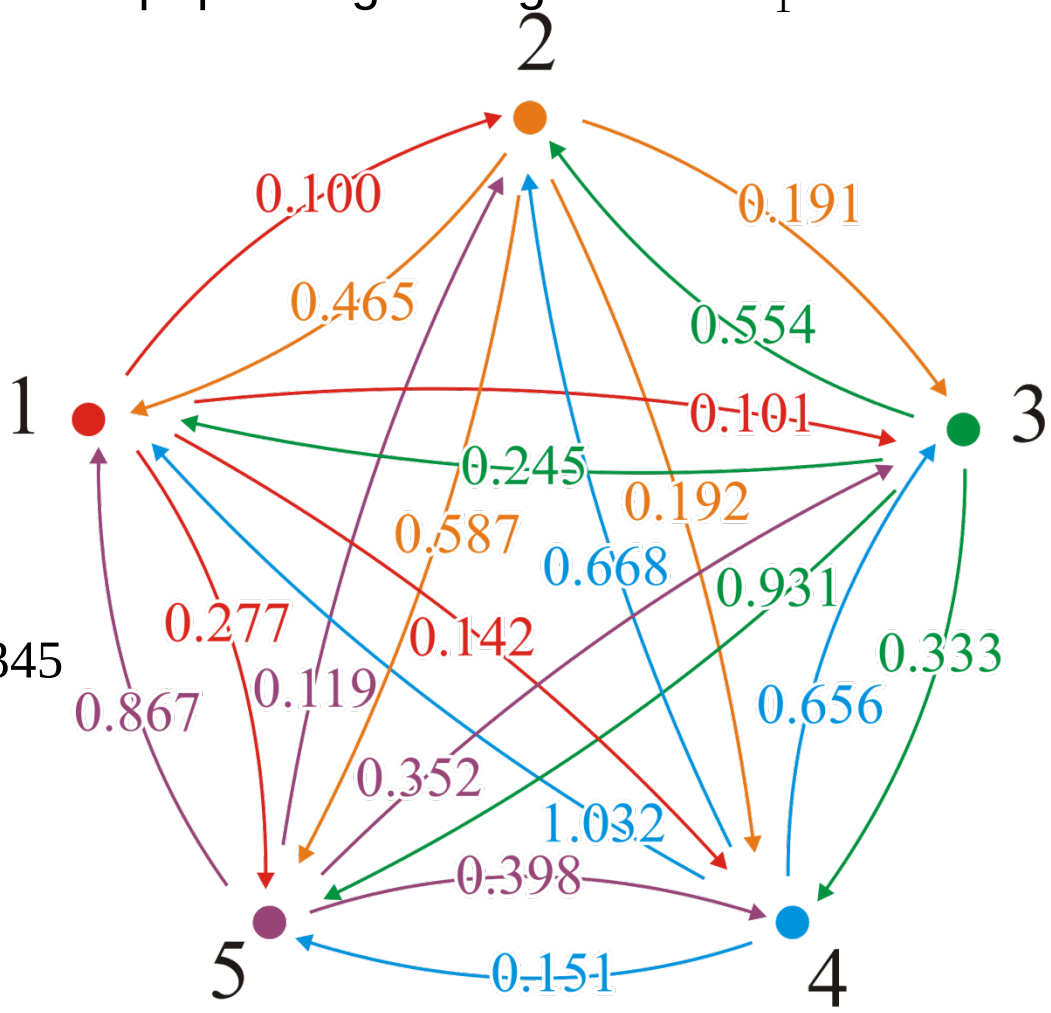
With the first pass, $k = 1$, we attempt passing through vertex v_1

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

We update the table

$(3, 2) \rightarrow (3, 1, 2)$

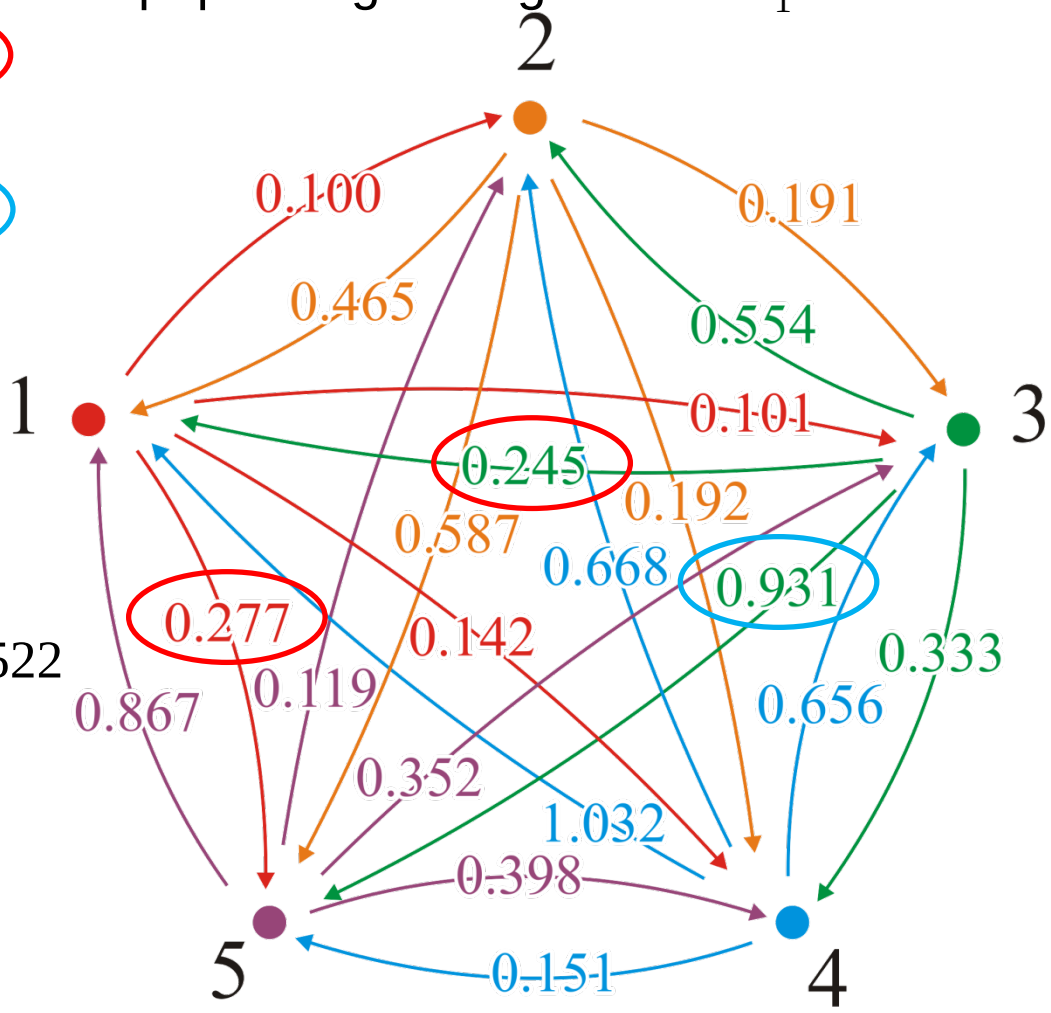
$0.554 > 0.245 + 0.100 = 0.345$



Example

With the first pass, $k = 1$, we attempt passing through vertex v_1

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



And a second shorter path:

$(3, 5) \rightarrow (3, 1, 5)$

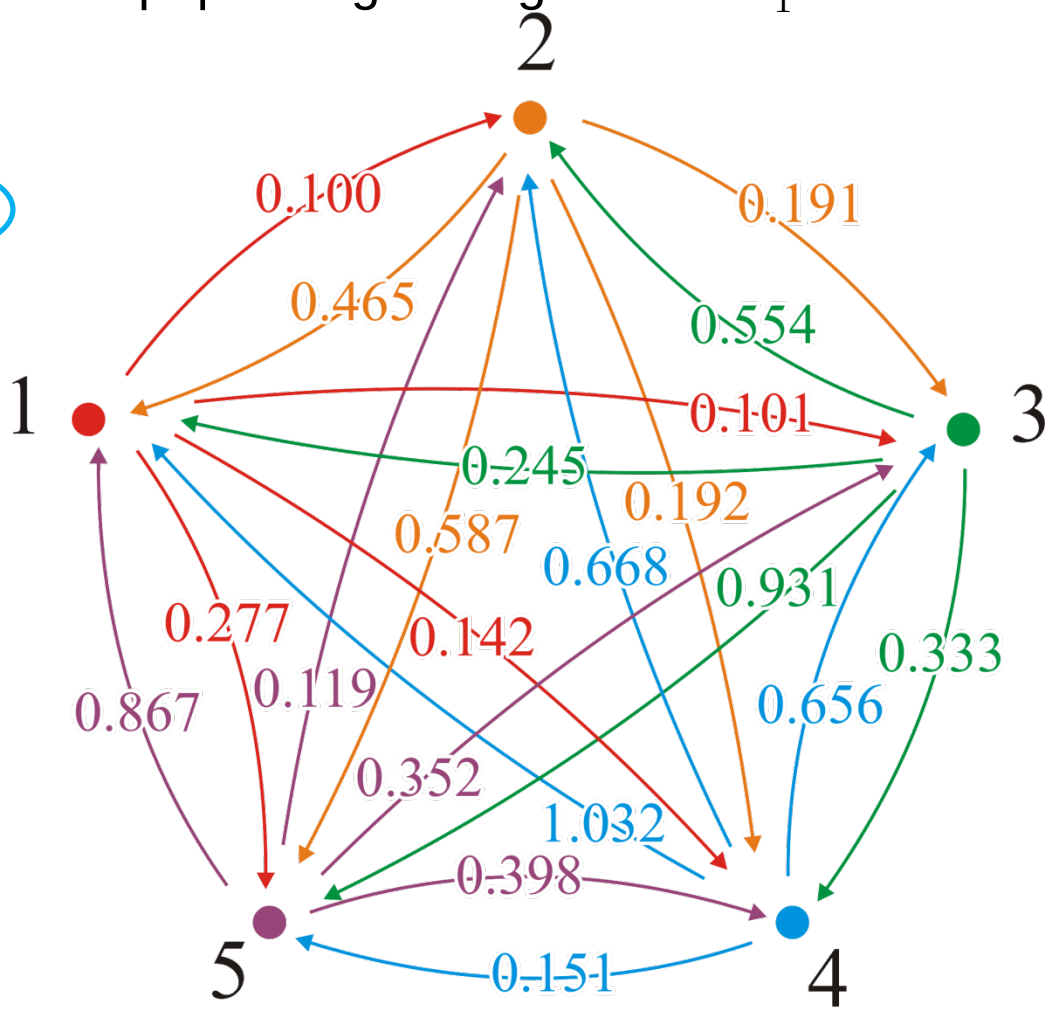
$0.931 > 0.245 + 0.277 = 0.522$

Example

With the first pass, $k = 1$, we attempt passing through vertex v_1

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

We update the table

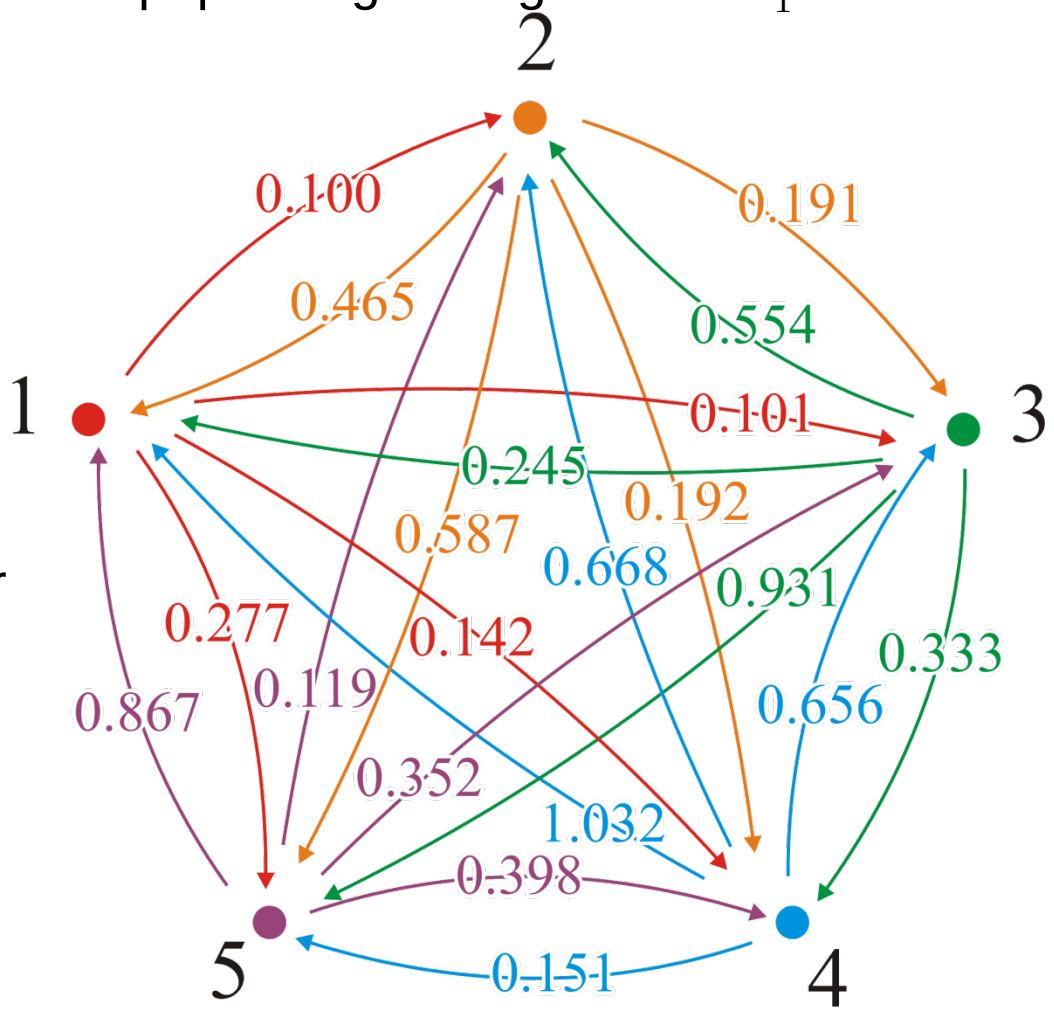


Example

With the first pass, $k = 1$, we attempt passing through vertex v_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…
We find that no other shorter paths through vertex v_1 exist



Example

With the next pass, $k = 2$, we attempt passing through vertex v_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:

$(5, 1) \rightarrow (5, 2, 1)$

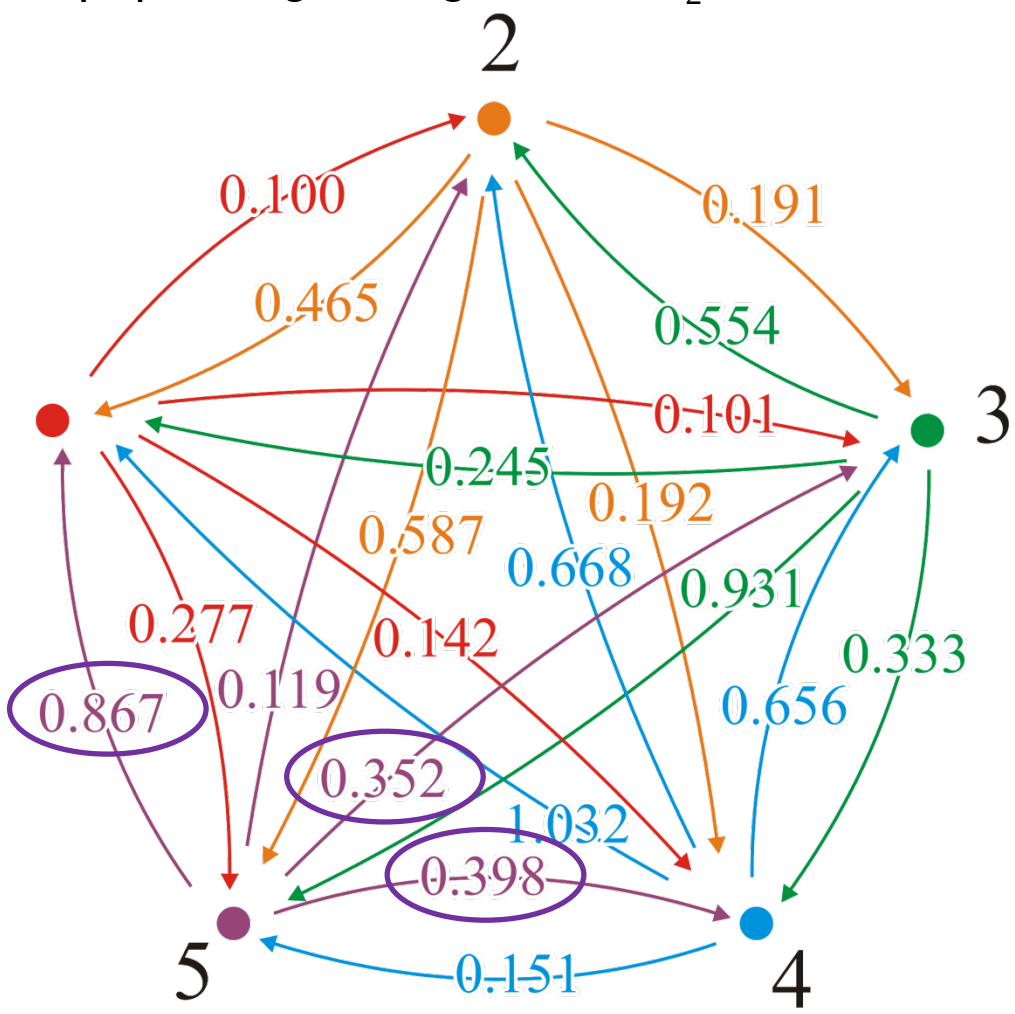
$0.867 > 0.119 + 0.465 = 0.584$

$(5, 3) \rightarrow (5, 2, 3)$

$0.352 > 0.119 + 0.191 = 0.310$

$(5, 4) \rightarrow (5, 2, 4)$

$0.398 > 0.119 + 0.192 = 0.311$

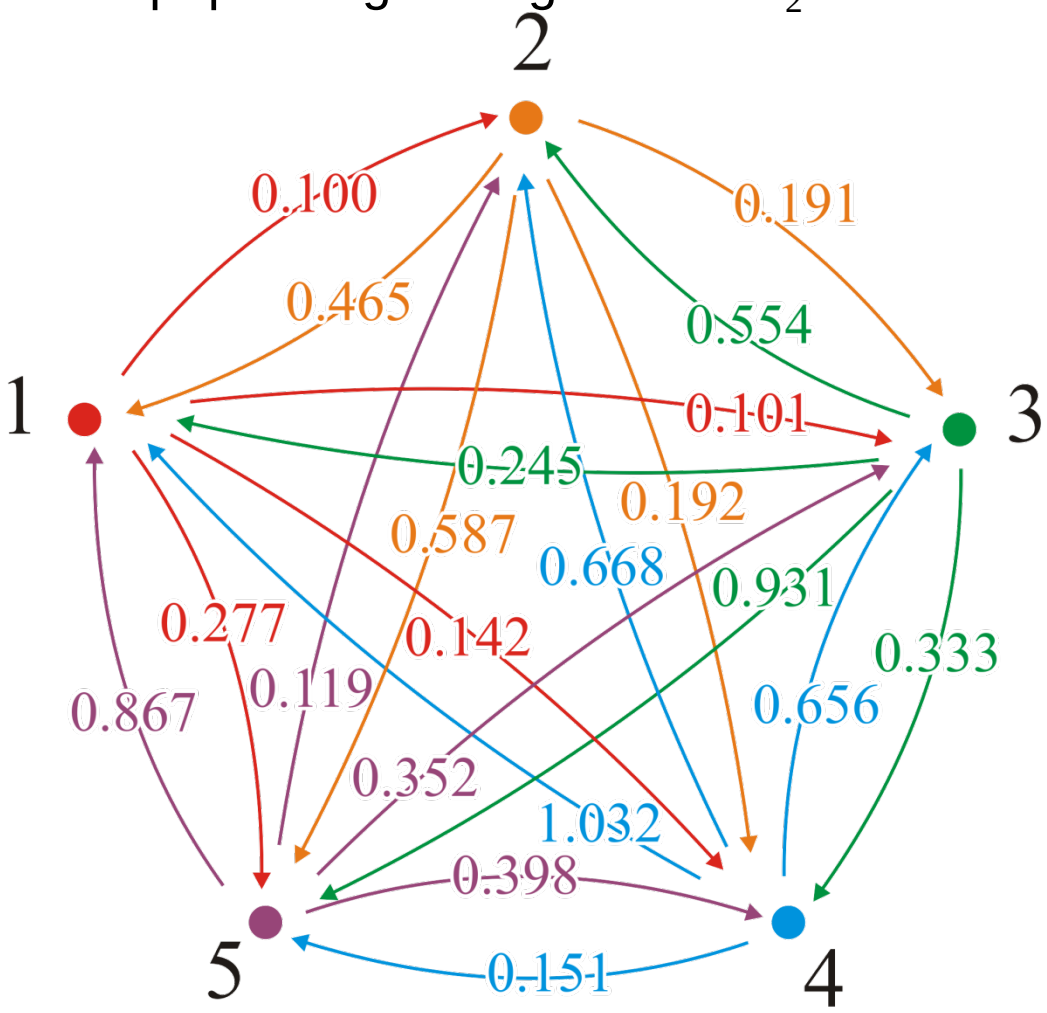


Example

With the next pass, $k = 2$, we attempt passing through vertex v_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 update the table



Example

With the next pass, $k = 3$, we attempt passing through vertex v_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:

$(2, 1) \rightarrow (2, 3, 1)$

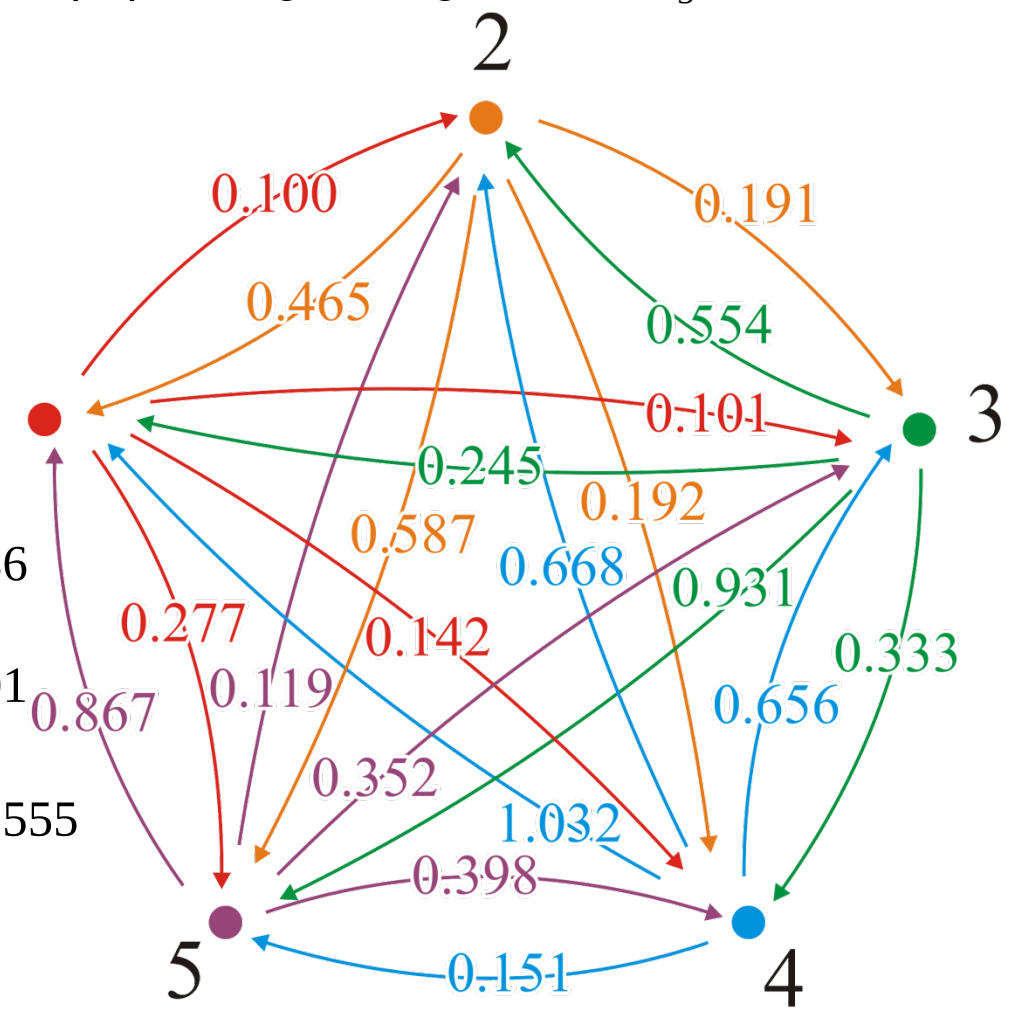
$0.465 > 0.191 + 0.245 = 0.436$

$(4, 1) \rightarrow (4, 3, 1)$

$1.032 > 0.656 + 0.245 = 0.901$

$(5, 1) \rightarrow (5, 3, 1)$

$0.584 > 0.310 + 0.245 = 0.555$

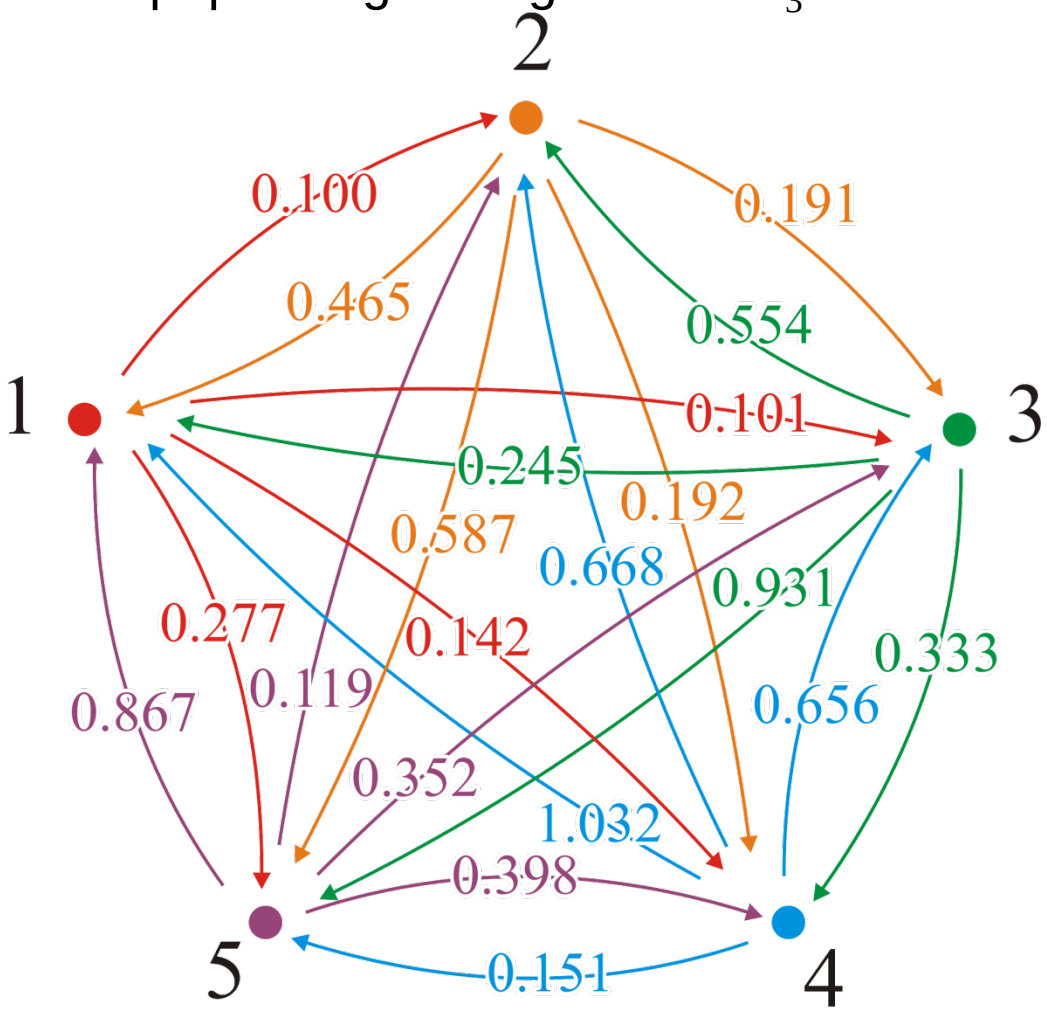


Example

With the next pass, $k = 3$, we attempt passing through vertex v_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 update the table



Example

With the next pass, $k = 4$, we attempt passing through vertex v_4

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

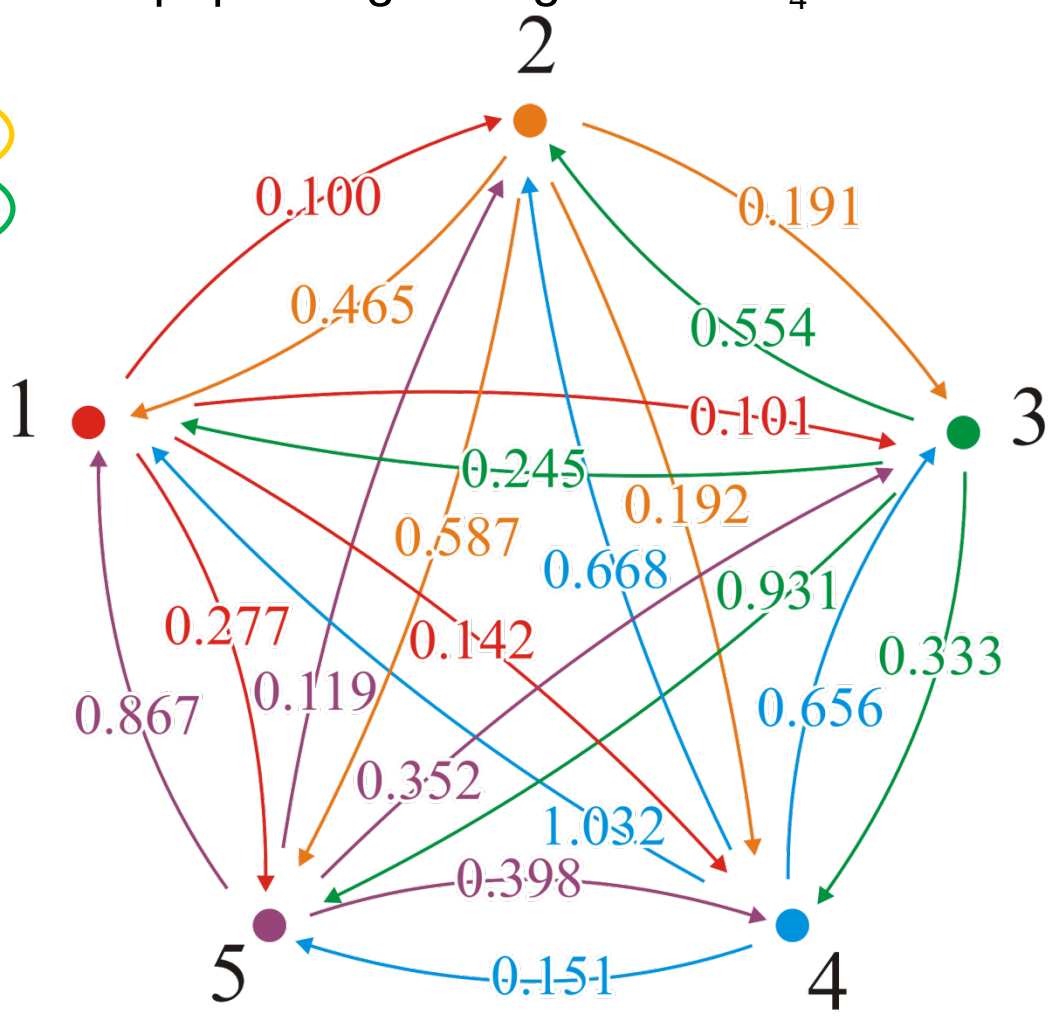
There are two shorter paths:

$(2, 5) \rightarrow (2, 4, 5)$

$0.587 > 0.192 + 0.151$

$(3, 5) \rightarrow (3, 4, 5)$

$0.522 > 0.333 + 0.151$

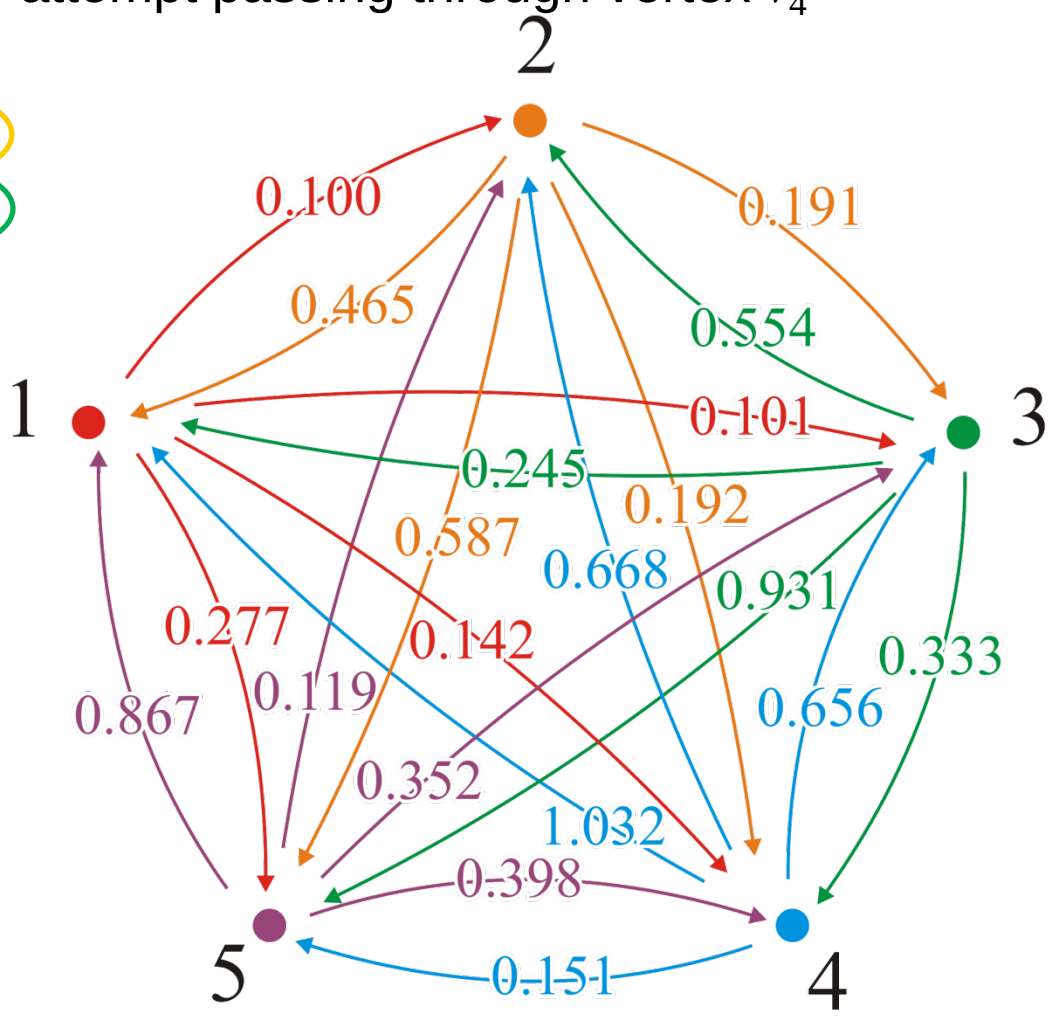


Example

With the next pass, $k = 4$, we attempt passing through vertex v_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 update the table



Example

With the last pass, $k = 5$, we attempt passing through vertex v_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 three shorter paths:

$(4, 1) \rightarrow (4, 5, 1)$

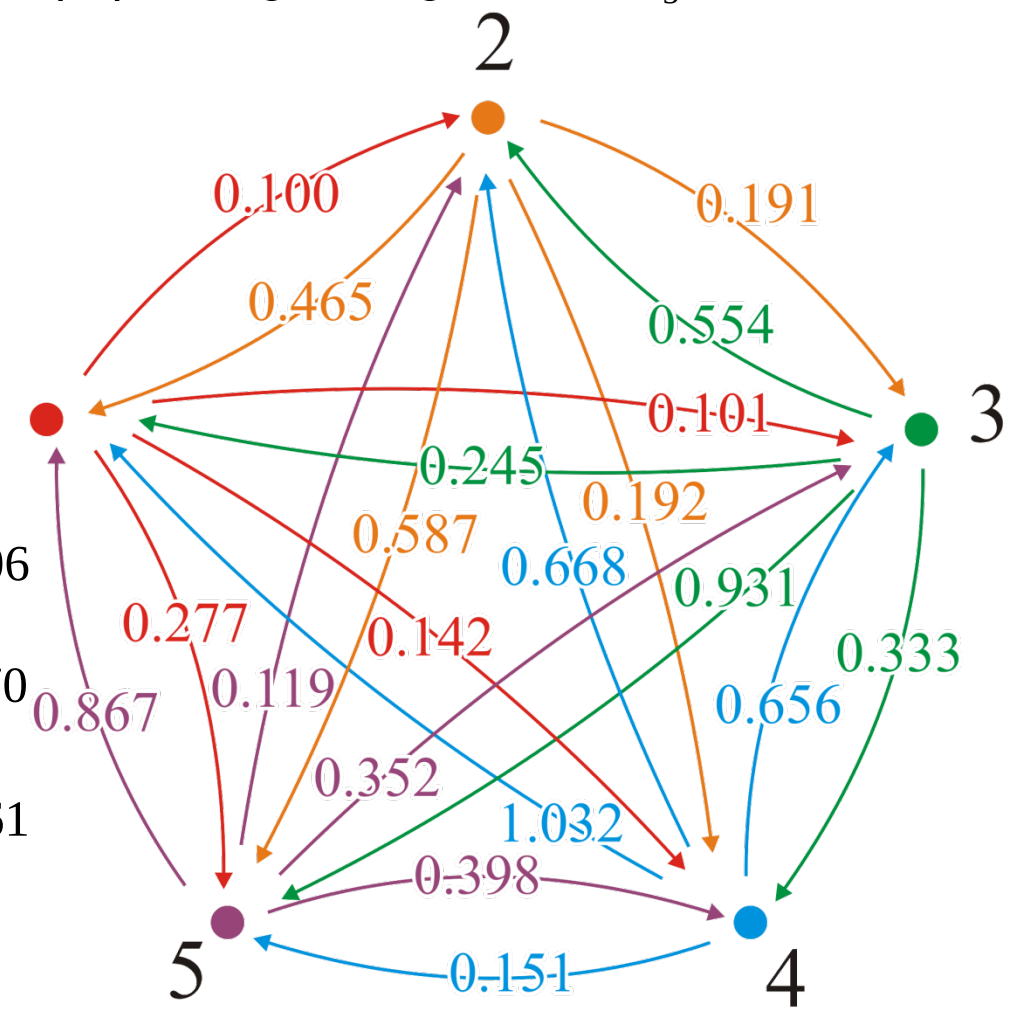
$0.901 > 0.151 + 0.555 = 0.706$

$(4, 2) \rightarrow (4, 5, 2)$

$0.668 > 0.151 + 0.119 = 0.270$

$(4, 3) \rightarrow (4, 5, 3)$

$0.656 > 0.151 + 0.310 = 0.461$

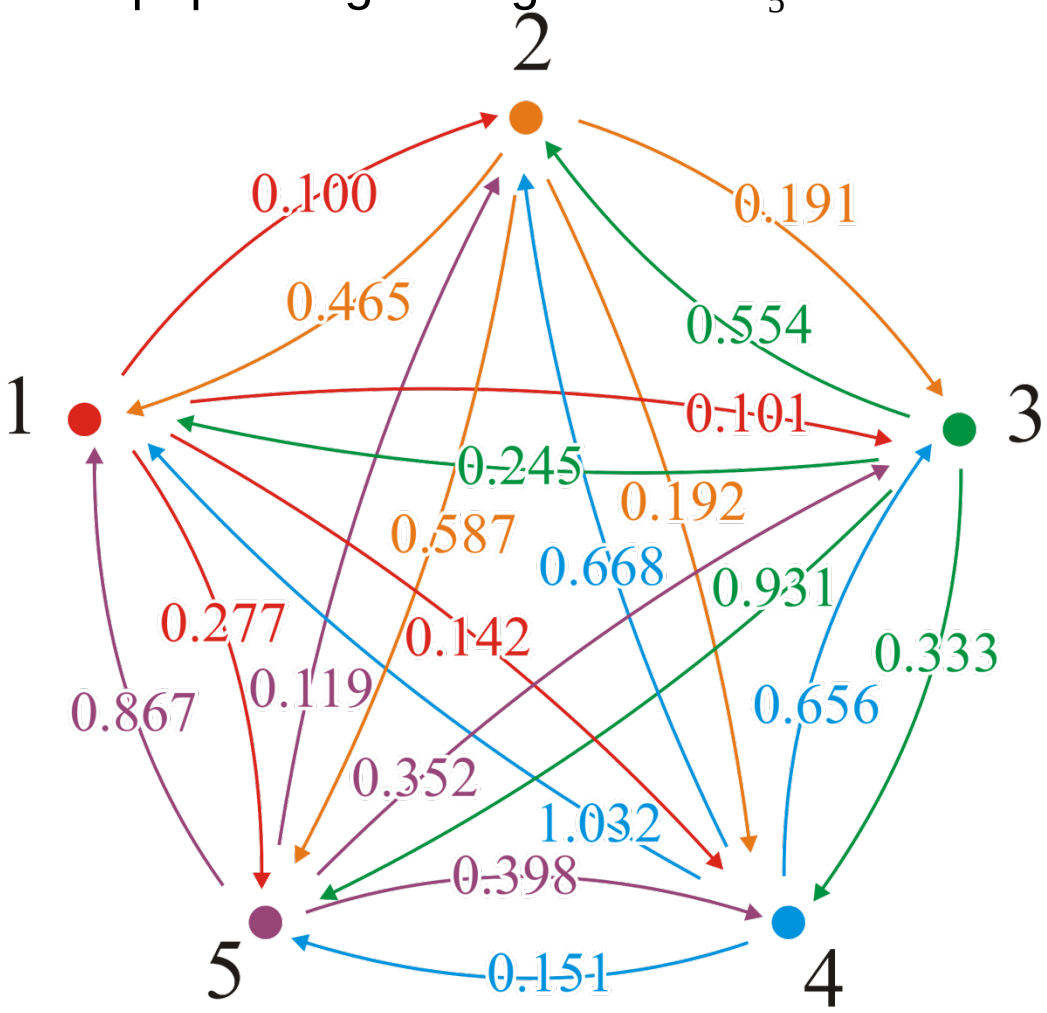


Example

With the last pass, $k = 5$, we attempt passing through vertex v_5

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

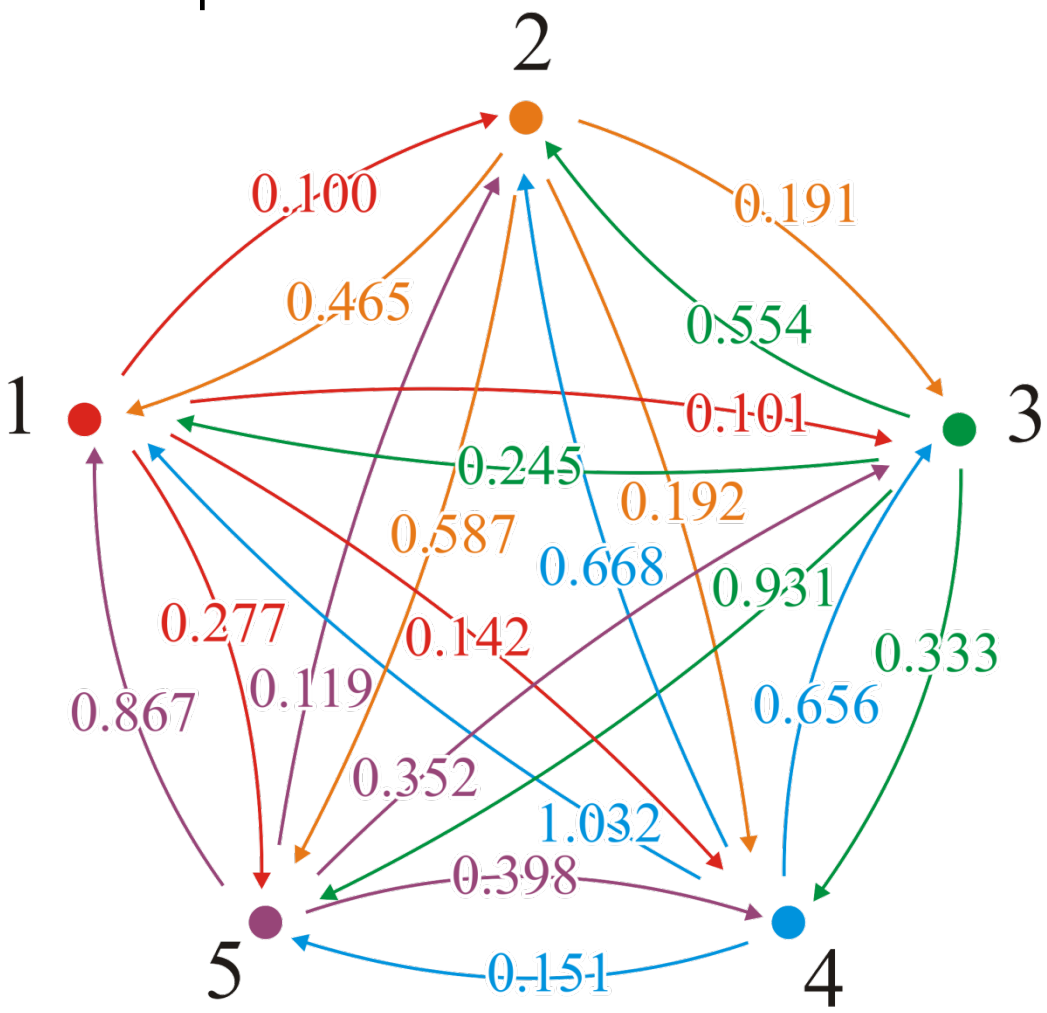
We update the table



Example

Thus, we have a table of all shortest paths:

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



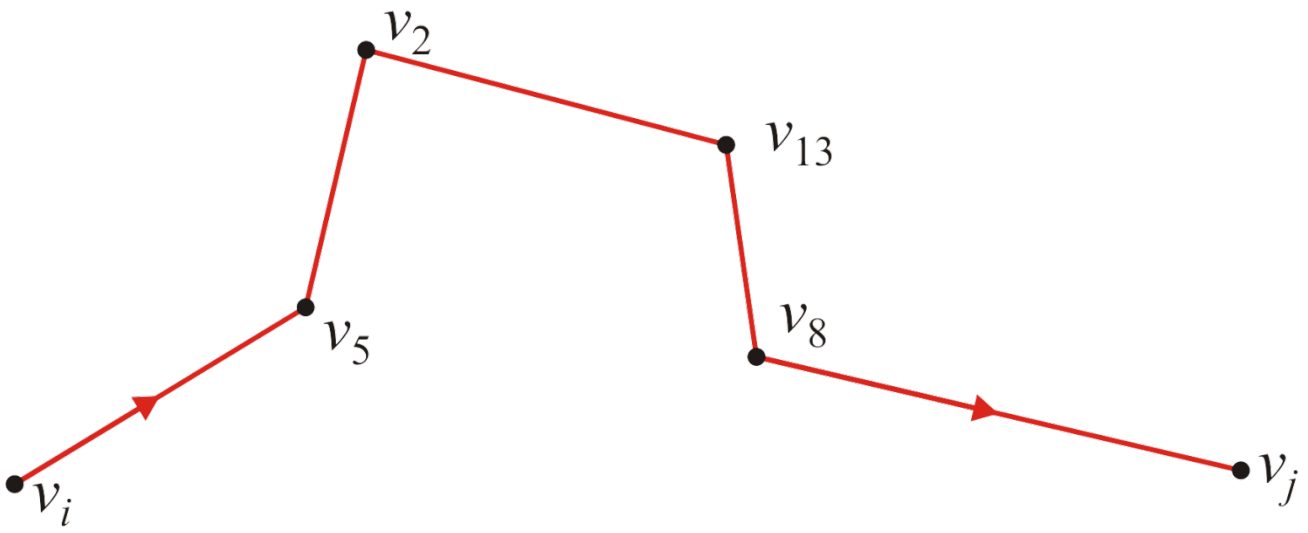
What Is the Shortest Path?

This algorithm finds the shortest distances, but what are the paths corresponding to those shortest distances?

- Recall that with Dijkstra's algorithm, we could find the shortest paths by recording the previous node
- Here we use a similar approach, but we choose to store the next node instead of the previous node

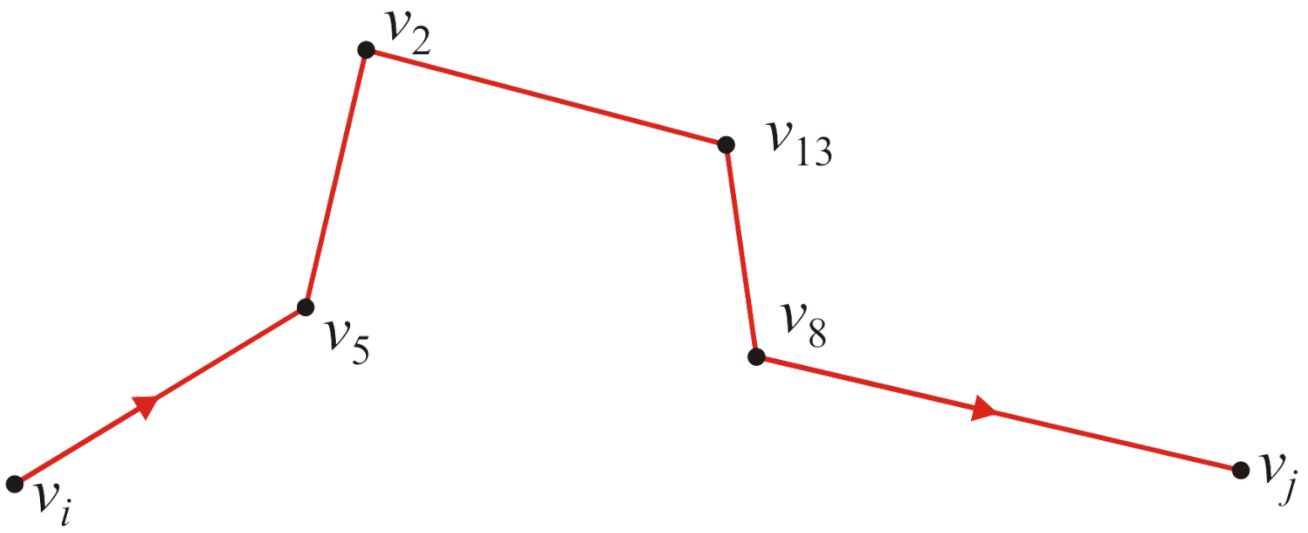
What Is the Shortest Path?

Suppose the shortest path from v_i to v_j is as follows:



What Is the Shortest Path?

Does this path consist of (v_i, v_5) and the shortest path from v_5 to v_j?

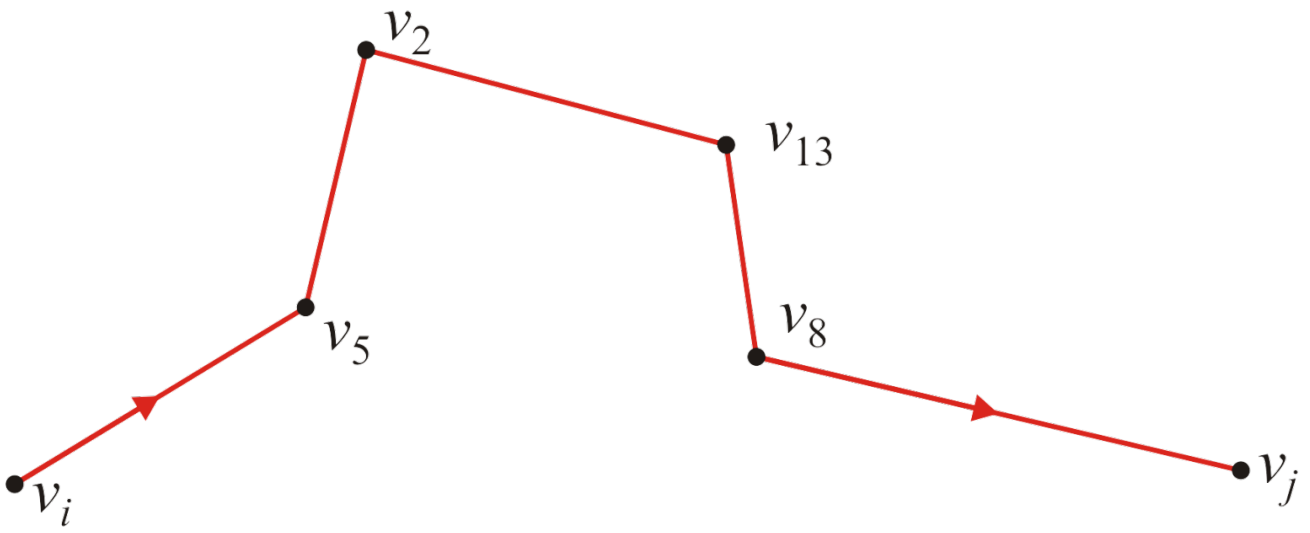


Yes

- If there was a shorter path from v_5 to v_j, then we would also find a shorter path from v_i to v_j

What Is the Shortest Path?

Does this path consist of (v_i, v_5) and the shortest path from v_5 to v_j?

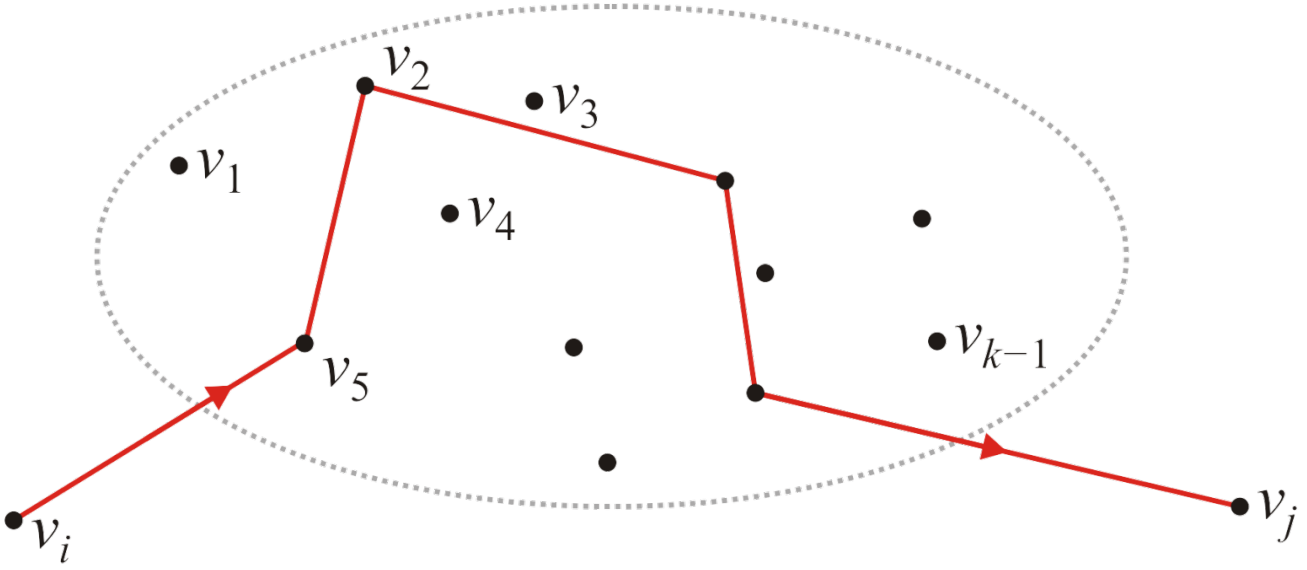


To find the shortest path from v_i to v_j, we only need to know that v_5 is the next vertex in the path — the rest of the path would be recursively recovered as the shortest path from v_5 to v_j

What Is the Shortest Path?

Now, suppose we have the shortest path from v_i to v_j which passes through the vertices $v_1, v_2, \ldots, v_{k-1}$

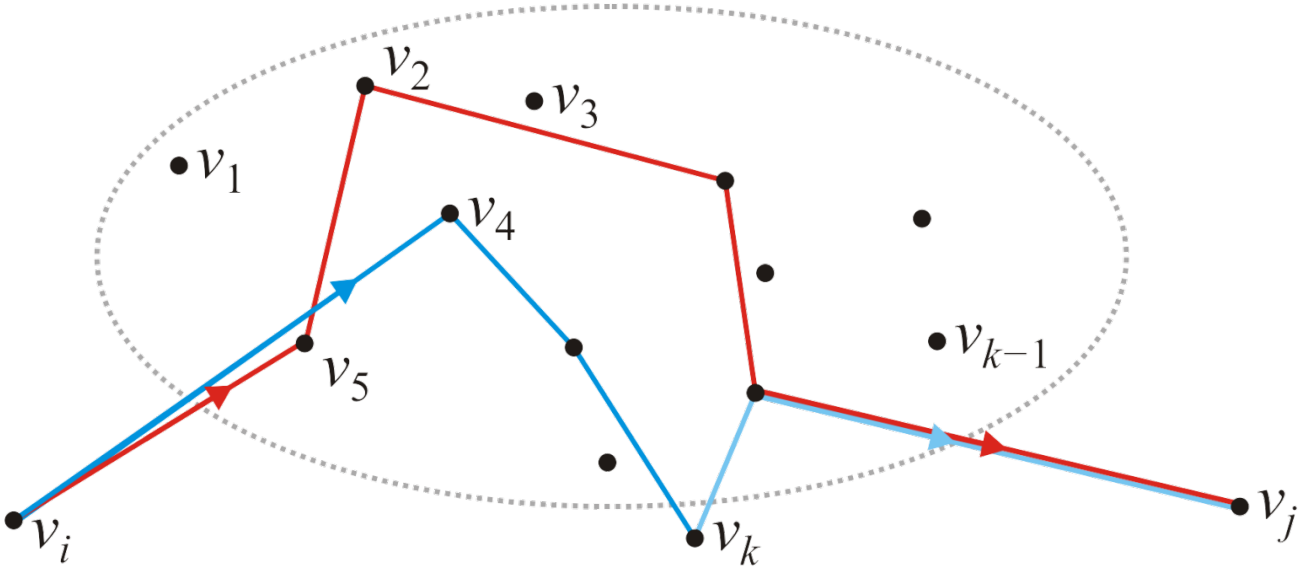
- In this example, the next vertex in the path is v_5



What Is the Shortest Path?

What if we find a shorter path passing through v_k?

– Now the next vertex in the new path should be the next vertex in the shortest path from v_i to v_k , which is v_4 in this example



What Is the Shortest Path?

Let us store the next vertex in the shortest path. Initially:

$$p_{i,j} = \begin{cases} \varnothing & \text{If } i = j \\ j & \text{If there is an edge from } i \text{ to } j \\ \varnothing & \text{Otherwise} \end{cases}$$

What Is the Shortest Path?

When we find a shorter path, update the next node:

$$p_{i,j} = p_{i,k}$$

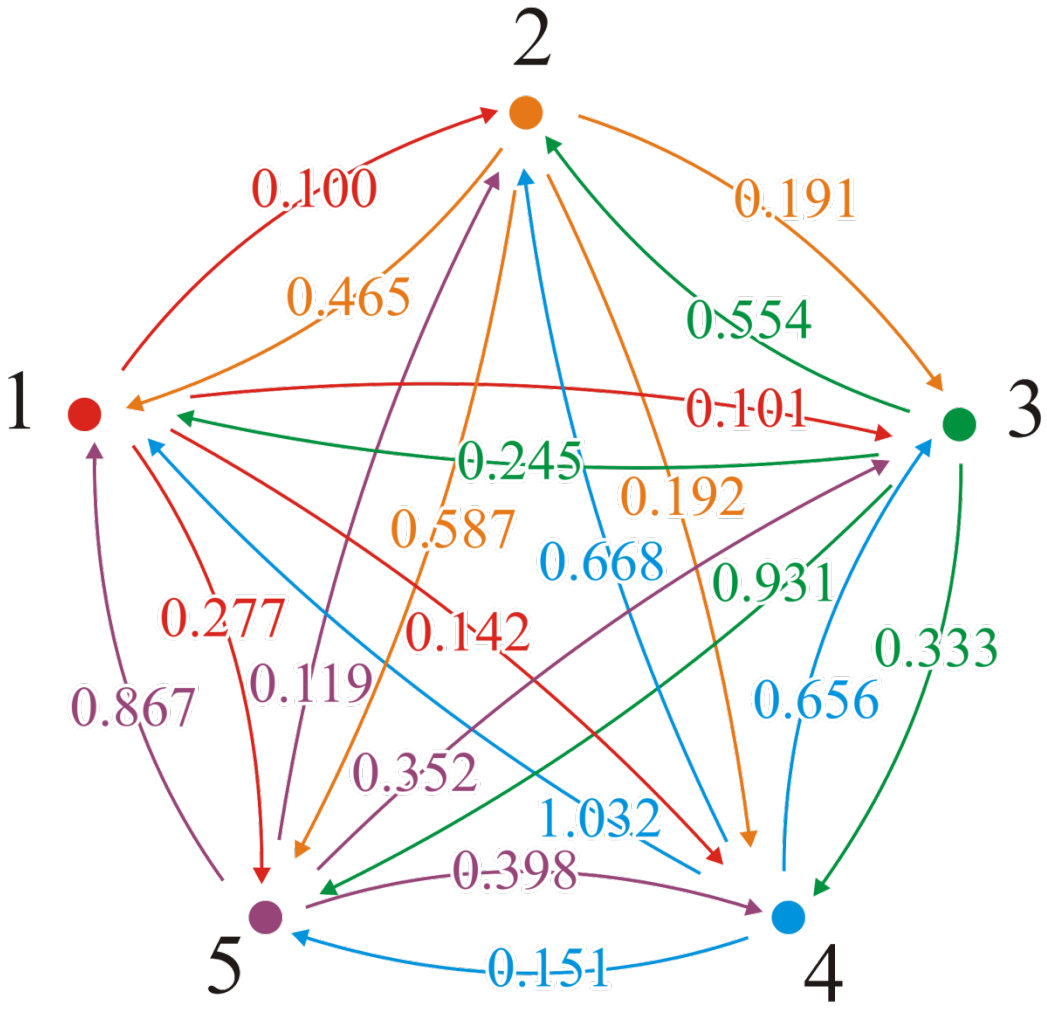
```
// Initialize the matrix p
//   ...

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] ) {
                p[i][j] = p[i][k];
                d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

Example

In our original example, initially, the next node is exactly that:

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 1 & - & 3 & 4 & 5 \\ 1 & 2 & - & 4 & 5 \\ 1 & 2 & 3 & - & 5 \\ 1 & 2 & 3 & 4 & - \end{pmatrix}$$

This would define our matrix $\mathbf{P} = (p_{ij})$



Example

With the first pass, $k = 1$, we attempt passing through vertex v_1

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 1 & - & 3 & 4 & 5 \\ 1 & 2 & - & 4 & 5 \\ 1 & 2 & 3 & - & 5 \\ 1 & 2 & 3 & 4 & - \end{pmatrix}$$

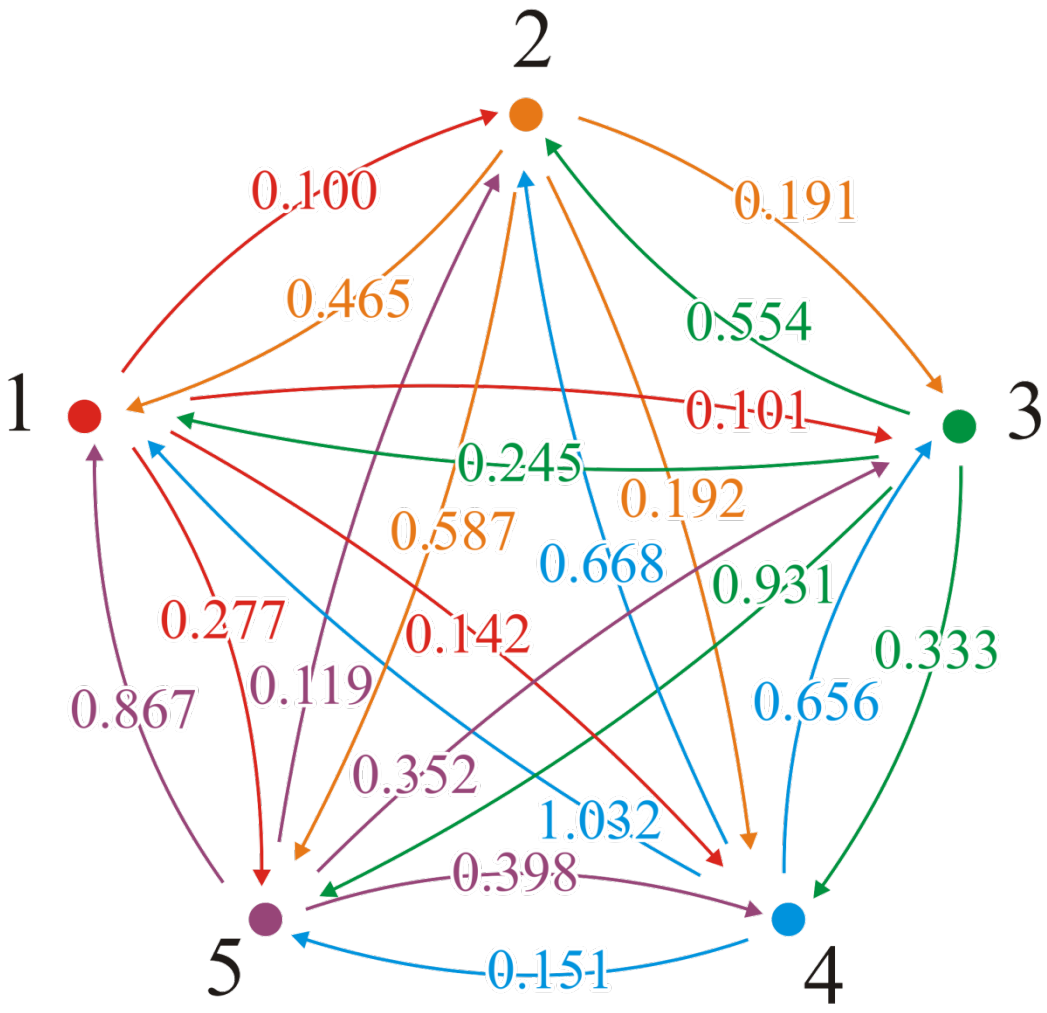
There are two shorter paths:

$(3, 2) \rightarrow (3, 1, 2)$

$0.554 > 0.245 + 0.100$

$(3, 5) \rightarrow (3, 1, 5)$

$0.931 > 0.245 + 0.277$

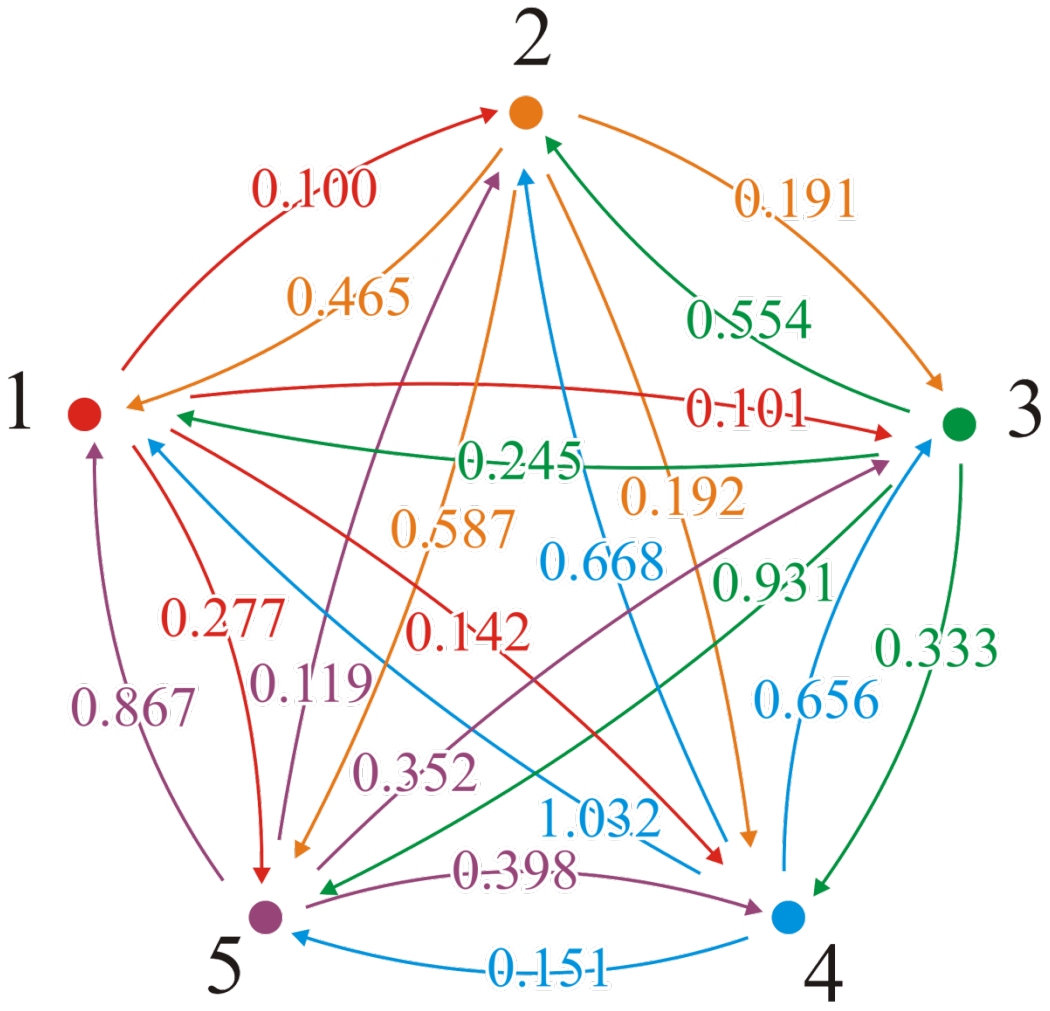


Example

With the first pass, $k = 1$, we attempt passing through vertex v_1

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 1 & - & 3 & 4 & 5 \\ 1 & 1 & - & 4 & 1 \\ 1 & 2 & 3 & - & 5 \\ 1 & 2 & 3 & 4 & - \end{pmatrix}$$

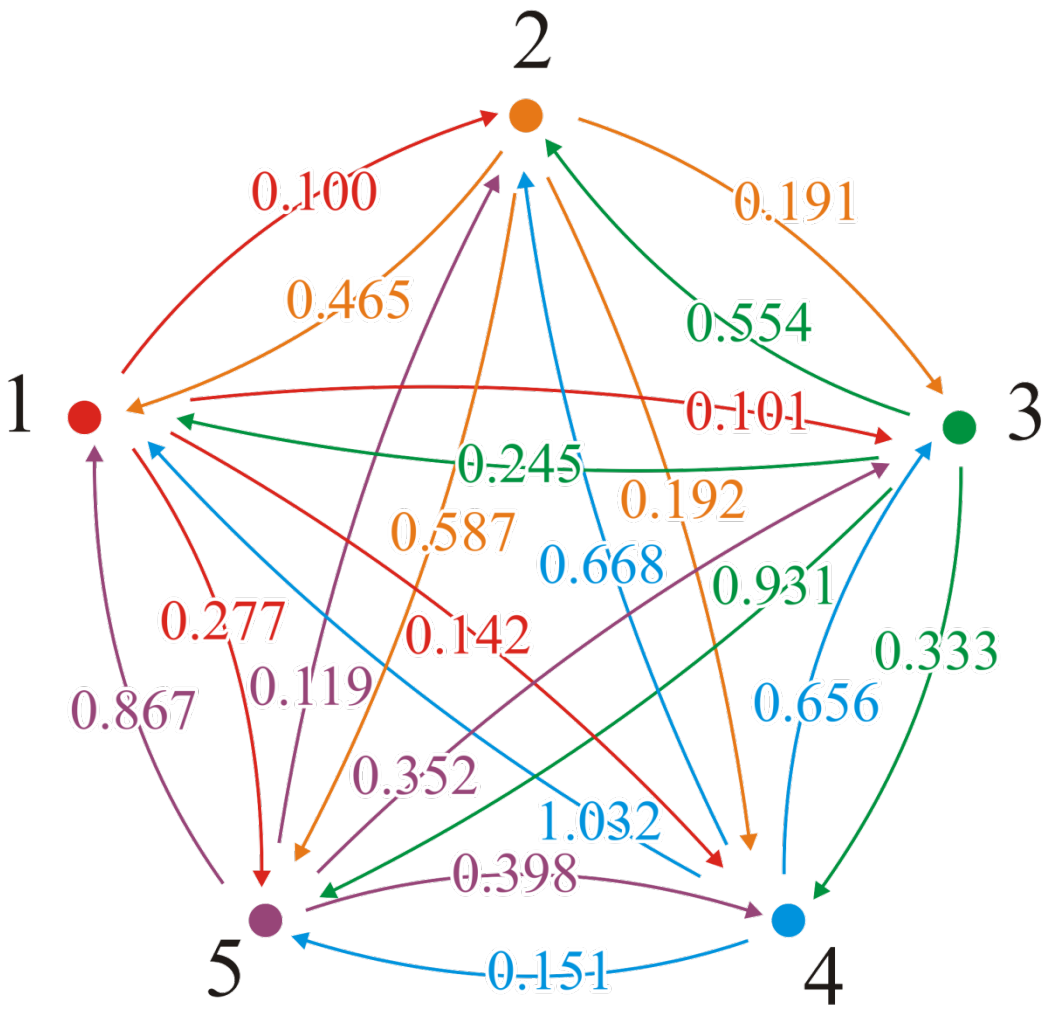
We update each of these



Example

After all the steps, we end up with the matrix $\boldsymbol{P} = (p_{i,j})$:

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$



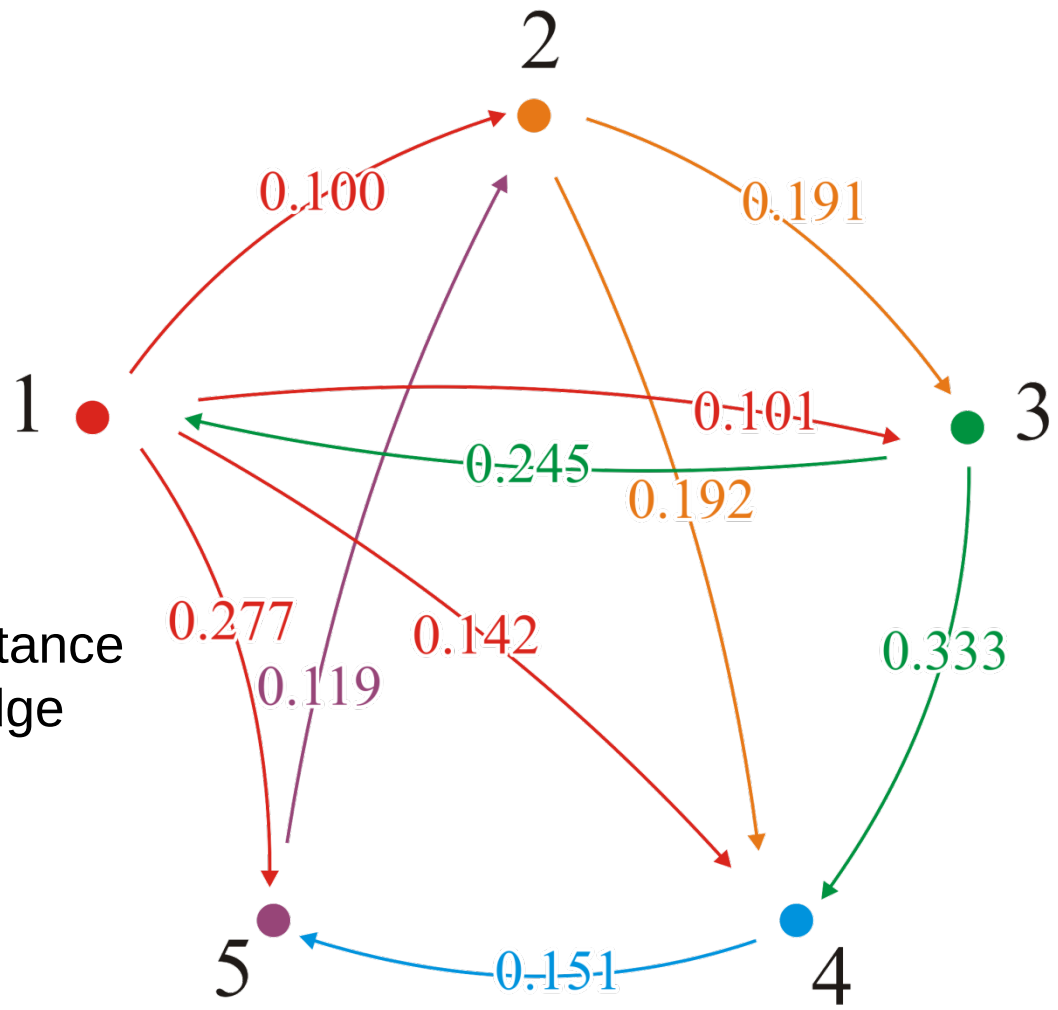
Example

These are all the adjacent edges that are still the shortest distance

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$

For each of these, $p_{i,j} = j$

In all cases, the shortest distance from vertex 1 is the direct edge



Example

From vertex v_2, $p_{2,3} = 3$ and $p_{2,4} = 4$; we go directly to vertices v_3 and v_4

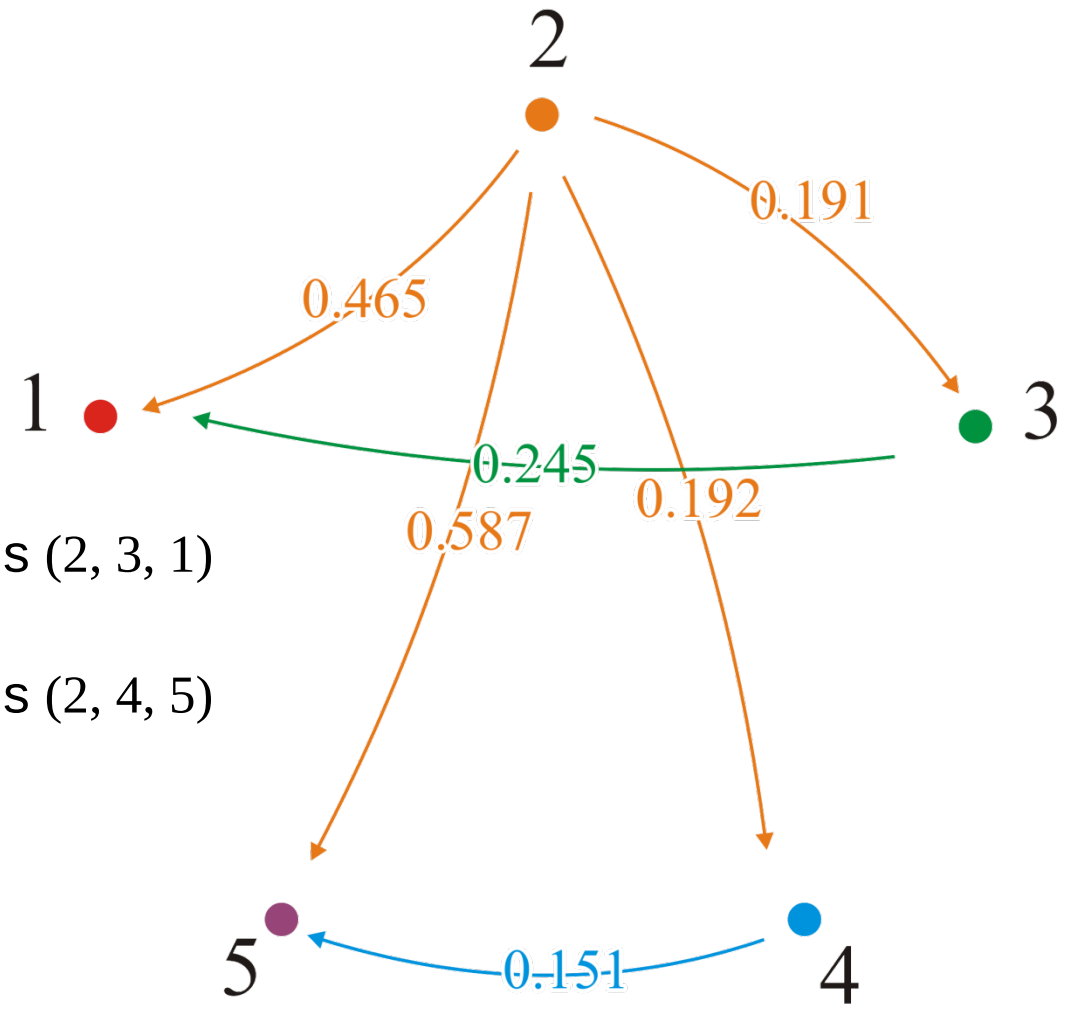
$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$

But $p_{2,1} = 3$ and $p_{3,1} = 1$;

the shortest path to v_1 is (2, 3, 1)

Also, $p_{2,5} = 4$ and $p_{4,5} = 5$;

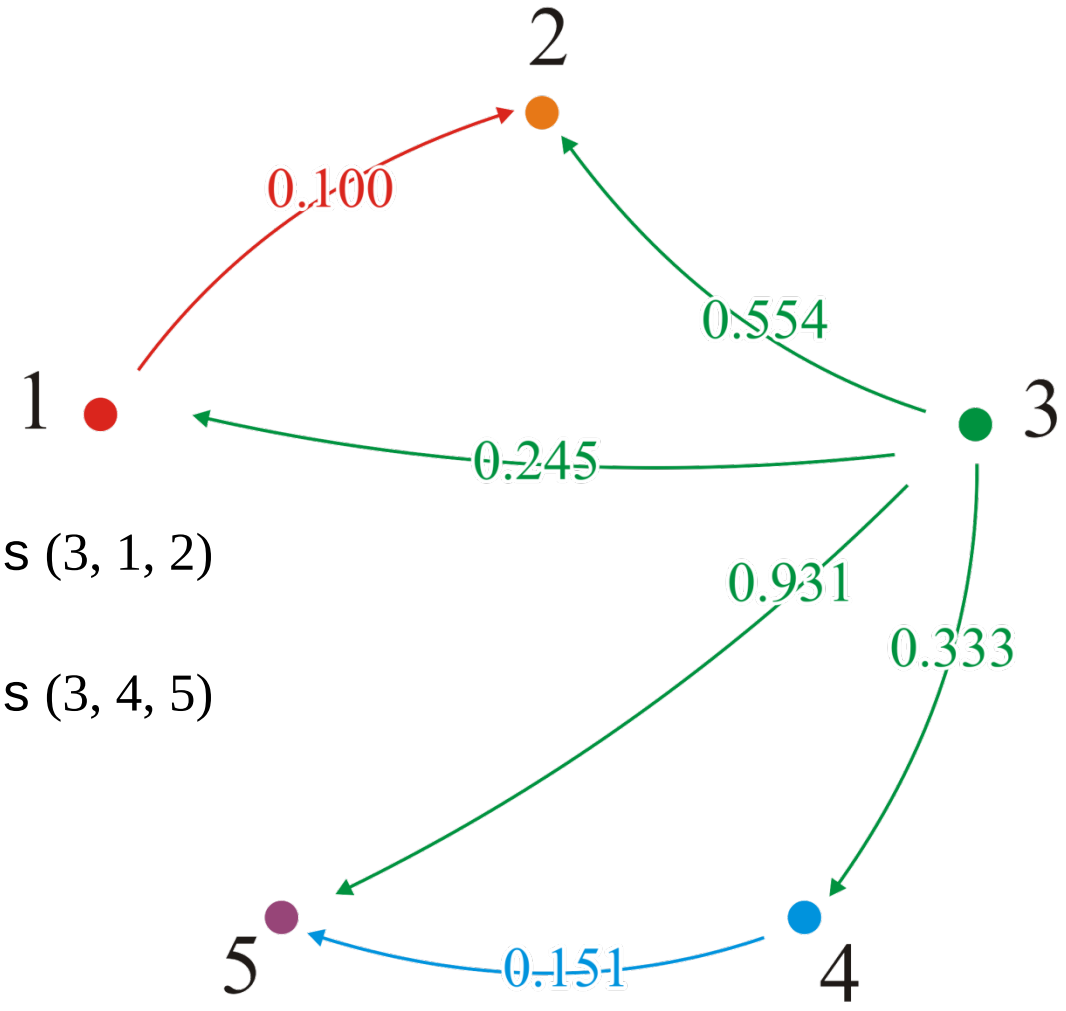
the shortest path to v_5 is (2, 4, 5)



Example

From vertex v_3, $p_{3,1} = 1$ and $p_{3,4} = 4$; we go directly to vertices v_1 and v_4

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$



But $p_{3,2} = 1$ and $p_{1,2} = 2$;

the shortest path to v_2 is (3, 1, 2)

Also, $p_{3,5} = 4$ and $p_{4,5} = 5$;

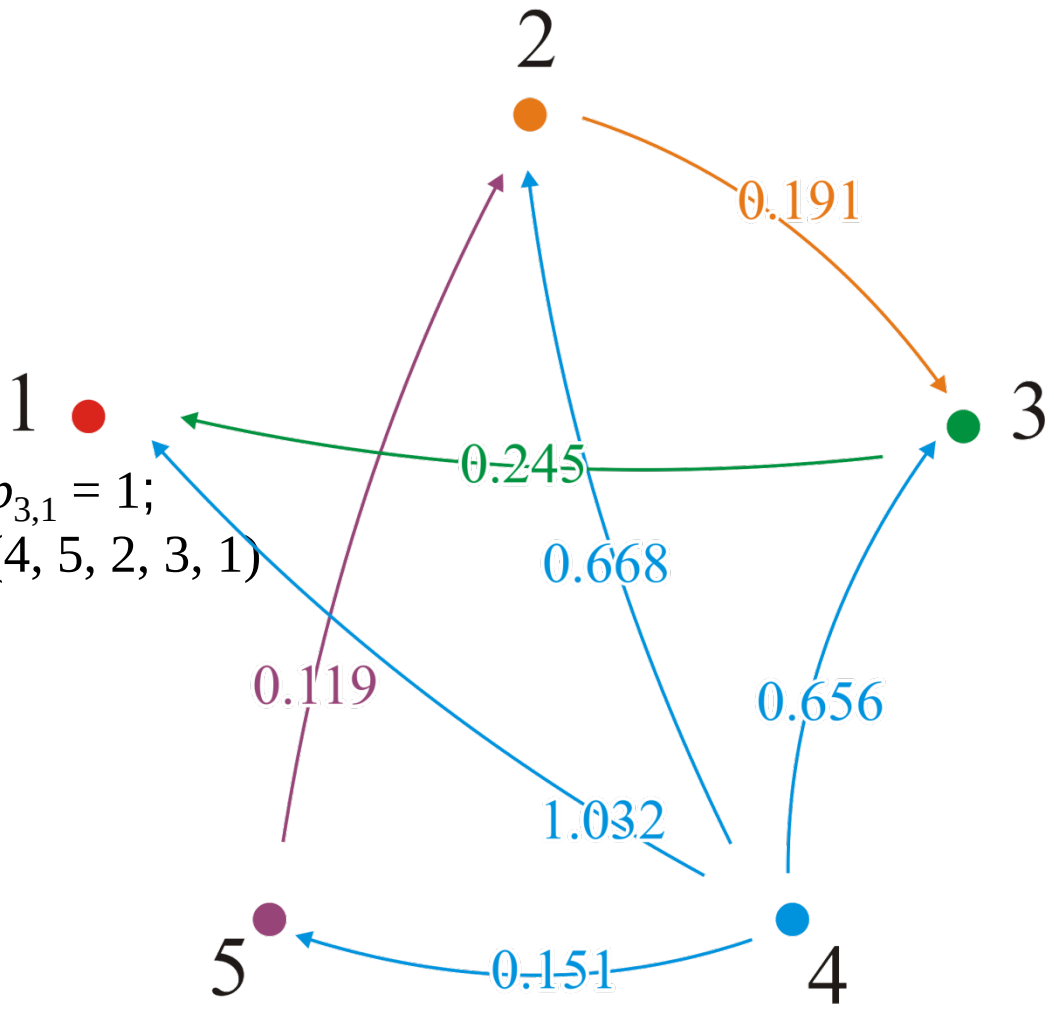
the shortest path to v_5 is (3, 4, 5)

Example

From vertex v_4, $p_{4,5} = 5$; we go directly to vertex v_5

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$

But $p_{4,1} = 5$, $p_{5,1} = 2$, $p_{2,1} = 3$, $p_{3,1} = 1$;
the shortest path to v_1 is (4, 5, 2, 3, 1)

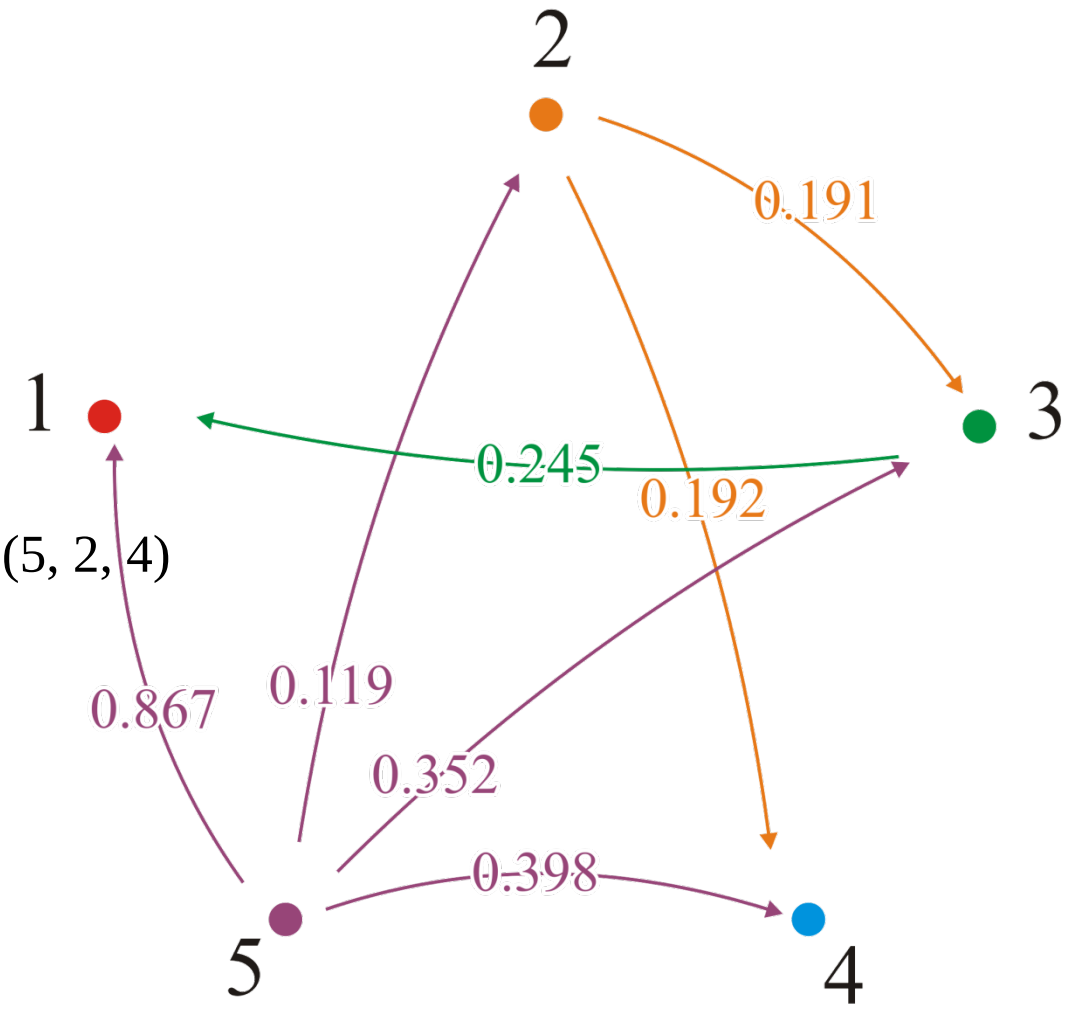


Example

From vertex v_5, $p_{5,2} = 2$; we go directly to vertex v_2

$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$

But $p_{5,4} = 2$ and $p_{2,4} = 4$;
the shortest path to v_4 is (5, 2, 4)



Summary

- Definition and applications
- Dijkstra’s algorithm
  - Single source shortest distance
- Floyd-Warshall algorithm
  - All-pairs shortest distance