Path & Reachability (Cont)

COMP9312_22T2

UNSW
SYDNEY

Outline

- Reachability
  - Transitive closure
  - Optimal Tree cover
  - Two-Hop labelling

- **Shortest Path**
  - **Dijkstra’s algorithm**
  - **A* algorithm**
  - **Floyd-Warshall algorithm**


Shortest
Distance/Path

Shortest path

1. **Single-source shortest path problem** (finding the shortest paths between a given vertex v and all other vertices in the graph)

- BFS
- Dijkstra's algorithm (very similar to Prim's algorithm; assumes all weights are positive)
- If we only want to find the shortest path to one given goal node, A* algorithm

2. **All pair shortest path** (find the pairwise shortest distances as well as the corresponding paths)

- Floyd-Warshall algorithm

BFS-based shortest path

Find the shortest path from vertex u to v.

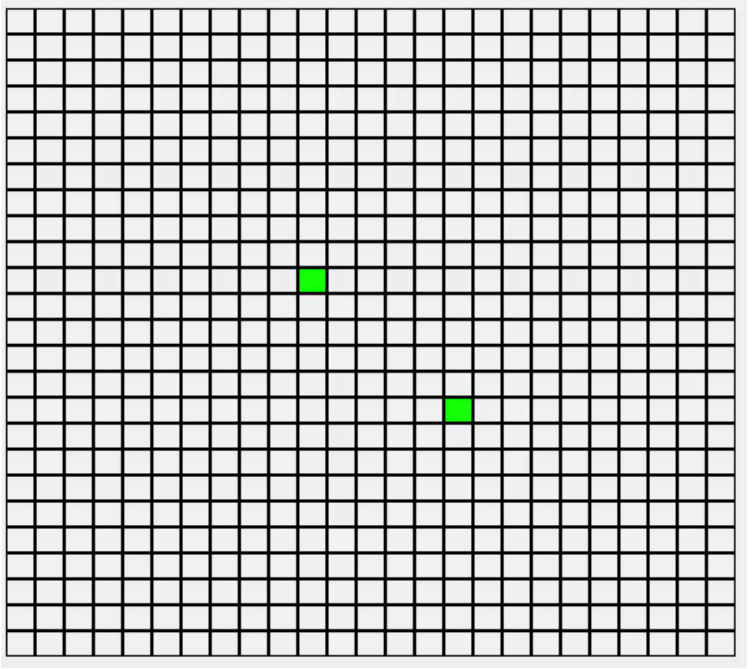
A naive BFS-based approach:

(1) conduct a BFS from the source vertex u

(2) for each visited vertex, record its parent in the searching tree

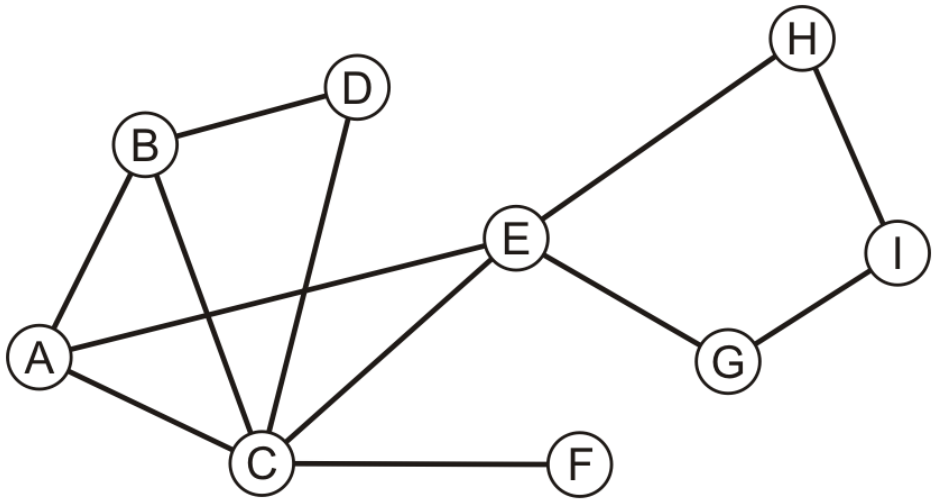
(3) when the target vertex v is visited, terminate BFS.

(4) recursively return the path from u to v.



Example

Find the shortest path from A to D by performing a breadth-first traversal



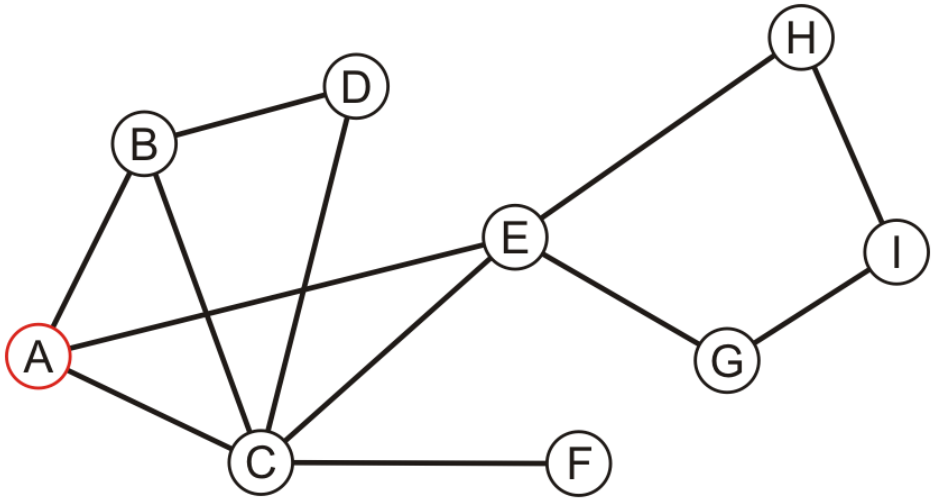
Push the first vertex A into the queue:

A					

Example

Pop A and push B, C, E.

Parents of visited vertices:
B: A
C: A
E: A



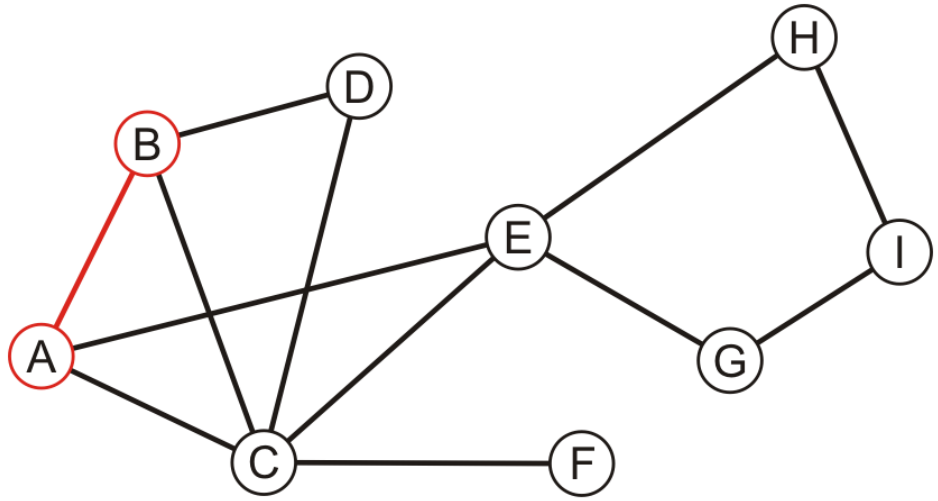
B	C	E			

Example

Pop B and push D into the queue.

Since D is found, terminate BFS! Based on the recorded parent vertices, one shortest path is A->B->D.

- Parents of visited vertices:
  B: A
  C: A
  D: B
  E: A



C	E	D			

BFS-based shortest path

Question:

1. Can we guarantee that this path is the shortest? If so, is it the only shortest path?
   Yes, we can. No, in the above example, A->C->D is another path with length 2.
2. Time complexity? Space complexity?
   The time complexity is O(m) which is the same with BFS. Slightly faster in practice because of the early termination. The space complexity is O(n) since we need to record the parent vertex of each visited vertex.
3. Can we use the above algorithm to solve the single-source shortest path problem and the all pairs shortest path problem?

   Yes.

Dijkstra's Algorithm

Motivation

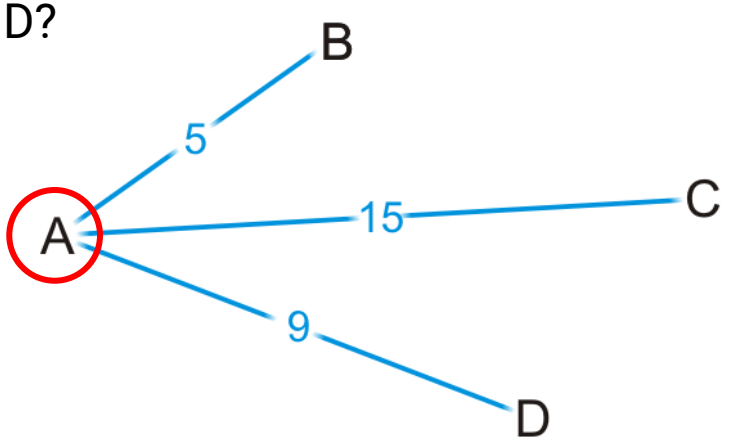
Single-source shortest path problem (finding the shortest paths between a given vertex v and all other vertices in the graph)

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

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

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

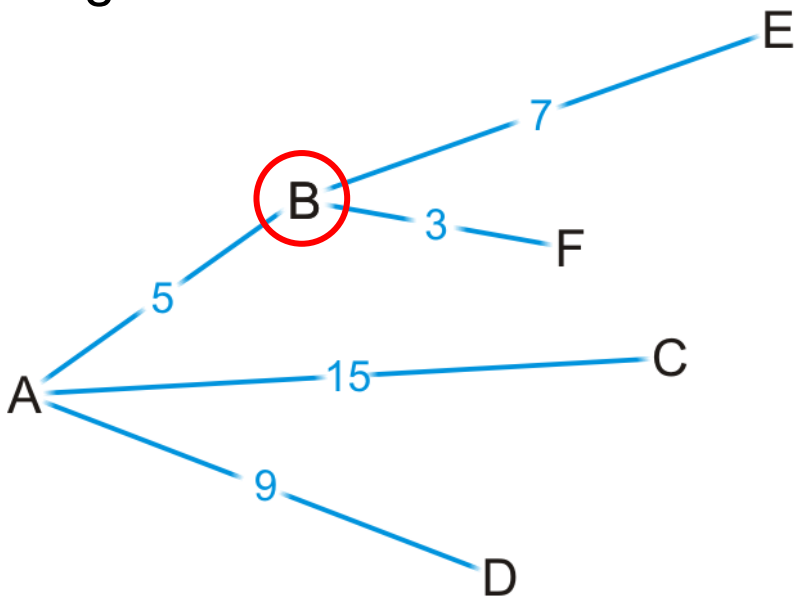


Motivation

Let’s see where we can go from B

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

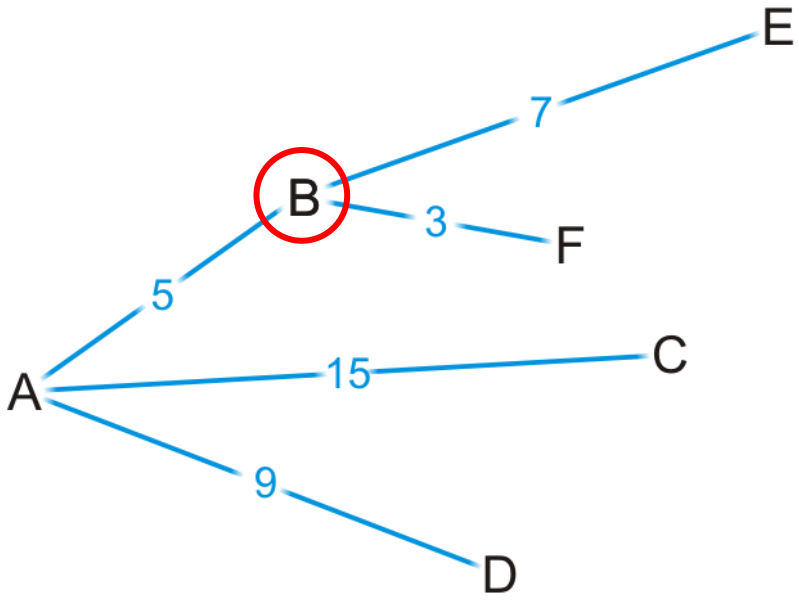


Motivation

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

- Why or why not?

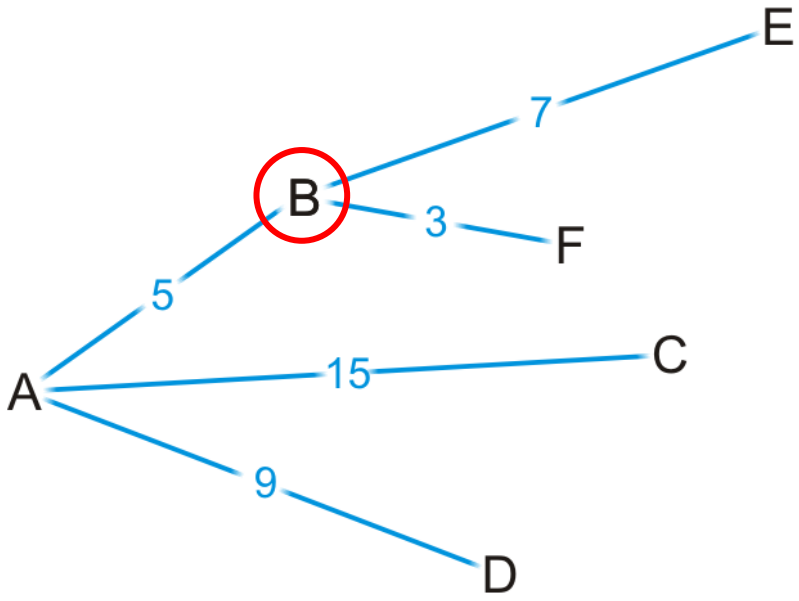
Yes, because 5+3<9, 5+3<15, and 3 is the smallest weight from B.



Motivation

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

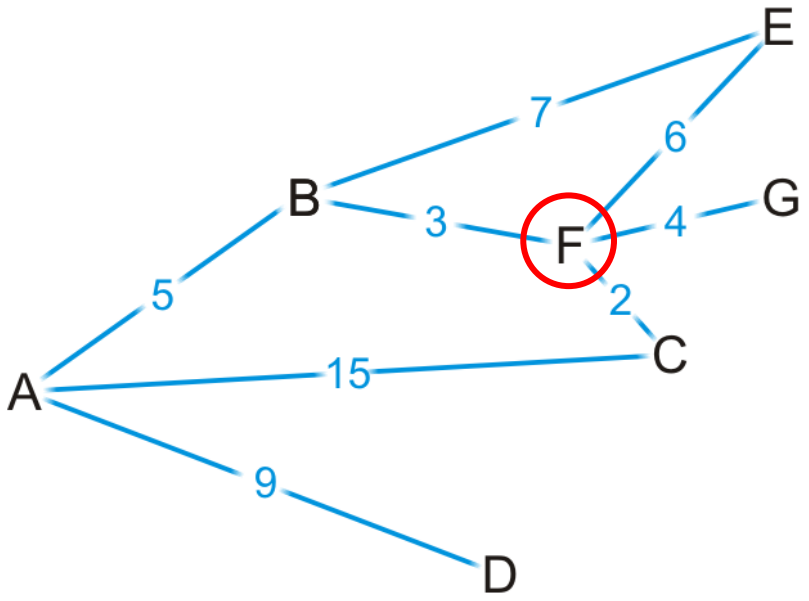
No.



Motivation

Okay, let's visit vertex F

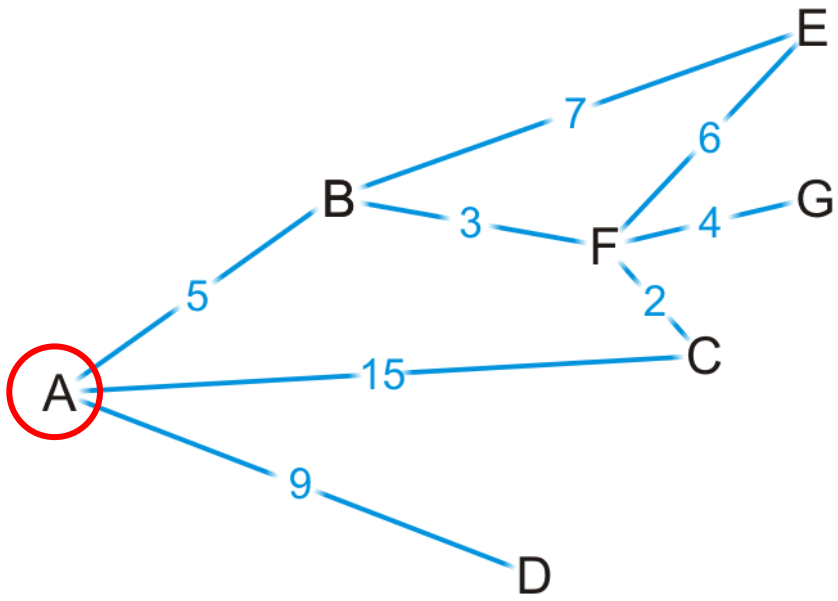
- We know the shortest path is (A, B, F) and it's of length 8



Motivation

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

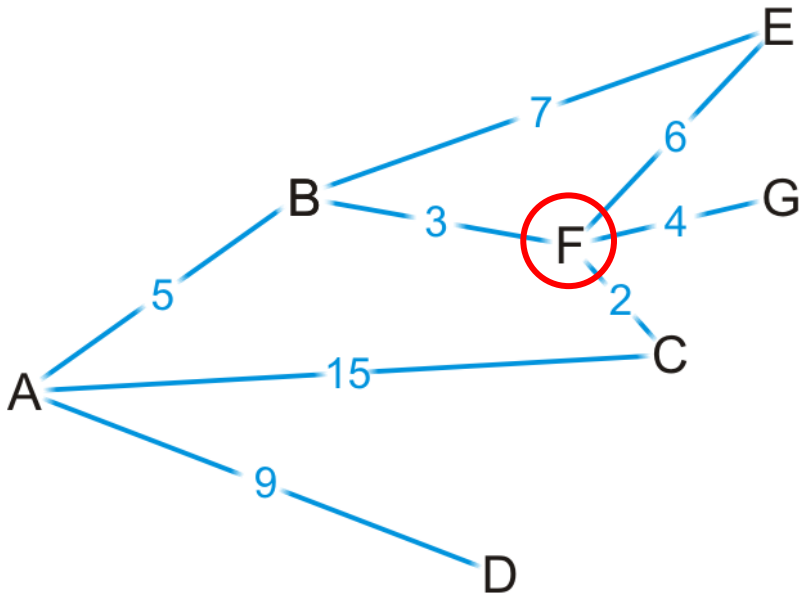
- (A, B, F, E) of length 8 + 6 = 14
- (A, B, F, G) of length 8 + 4 = 12
- (A, B, F, C) of length 8 + 2 = 10



Motivation

By observation:

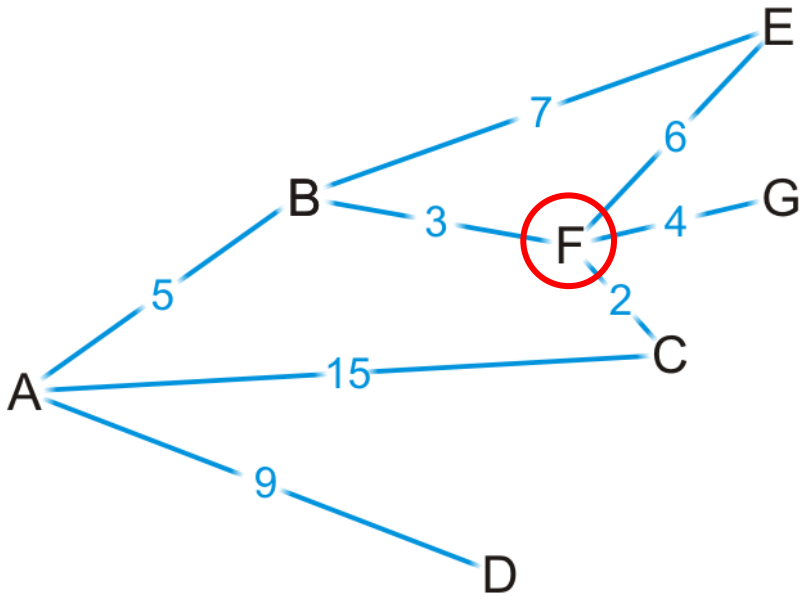
- The path (A, B, F, E) is longer than (A, B, E)
- The path (A, B, F, C) is shorter than the path (A, C)



Motivation

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

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

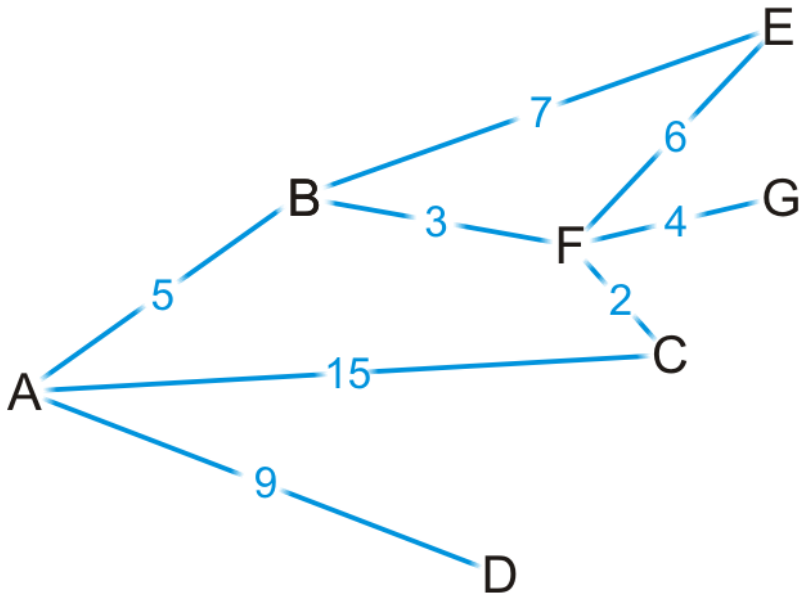


Motivation

At this point, we have the shortest distances to B and F

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

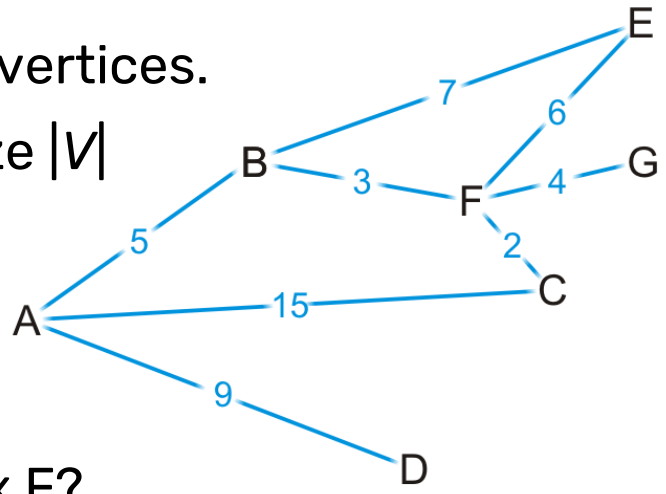
D. Because D is the unvisited vertex which has the shortest distance right now.



Dijkstra's algorithm

- Like Prim's algorithm, we initially don't know the distance to any vertex except the initial vertex. Thus, we require **an array of distances**, **all initialized to infinity except for the source vertex**, which is initialized to 0.
- Each time we visit a vertex, we will examine all adjacent vertices.

  We need to track visited vertices—a Boolean table of size $|V|$



How to track the shortest path to each vertex?

Do I have to store (A, B, F) as the shortest path to vertex F?

We really only have to record that the shortest path to vertex F came from vertex B

We would then determine that the shortest path to vertex B came from vertex A

Thus, **we need an array of previous vertices**, all initialized to null

Dijkstra's algorithm

We will iterate $|V|$ times:

- Find that unvisited vertex v that has a minimum distance to it. Mark it as having been visited.
- Consider every adjacent vertex w that is unvisited:
  - Is the distance to v plus the weight of the edge (v, w) less than our currently known shortest distance to w. If so, update the shortest distance to w and record v as the previous pointer.
- Continue iterating until all vertices are visited or all remaining vertices have a distance to them of infinity.

Example

Consider the game of *Risk* from Parker Brothers

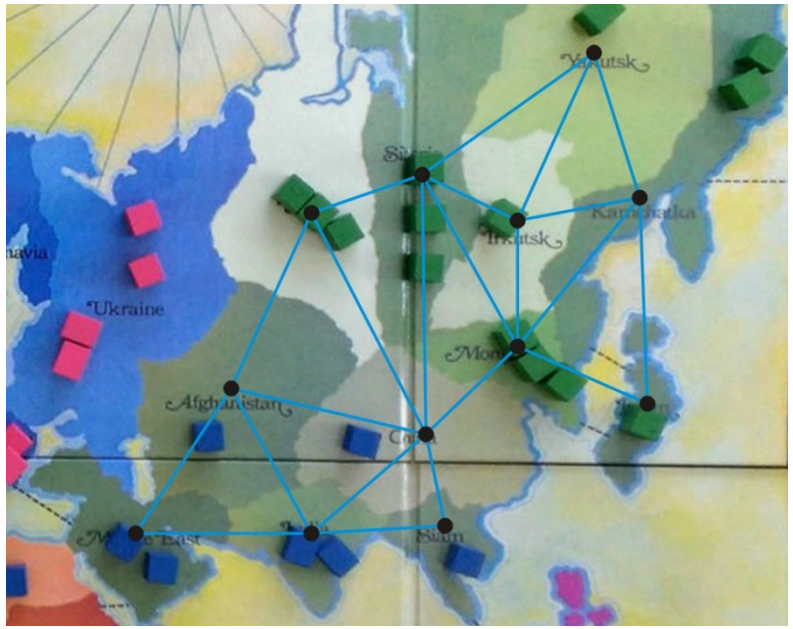
- A game of world domination
- The world is divided into 42 connected regions
- The regions are vertices and edges indicate adjacent regions

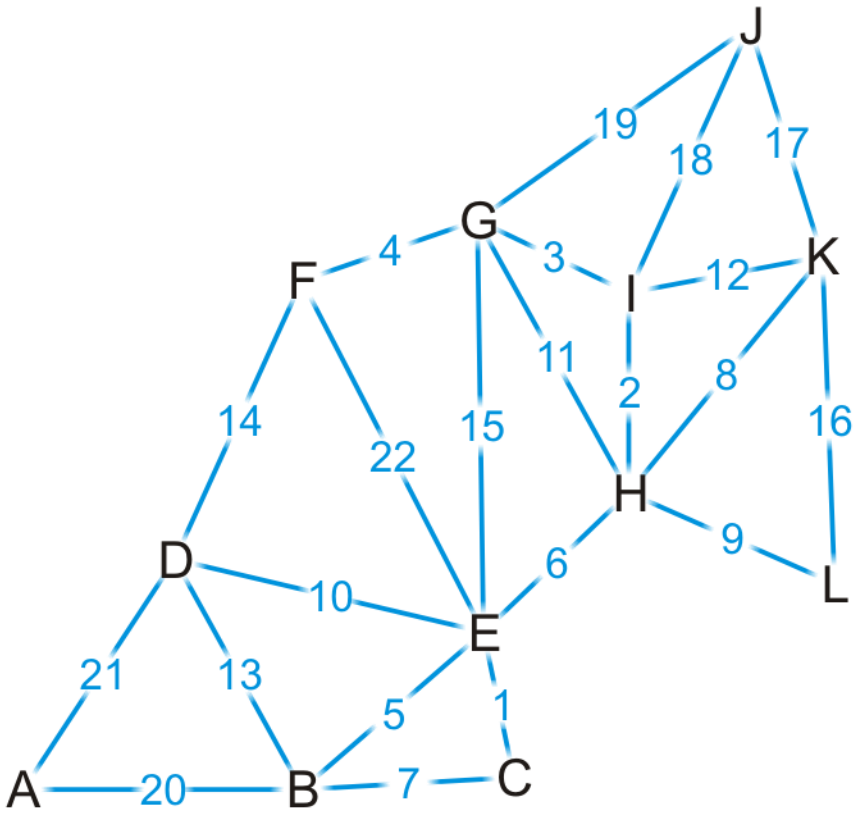
Example

We'll focus on Asia. Here is our abstract representation.

Let us give a weight to each of the edges.

Find the shortest distance from Kamchatka

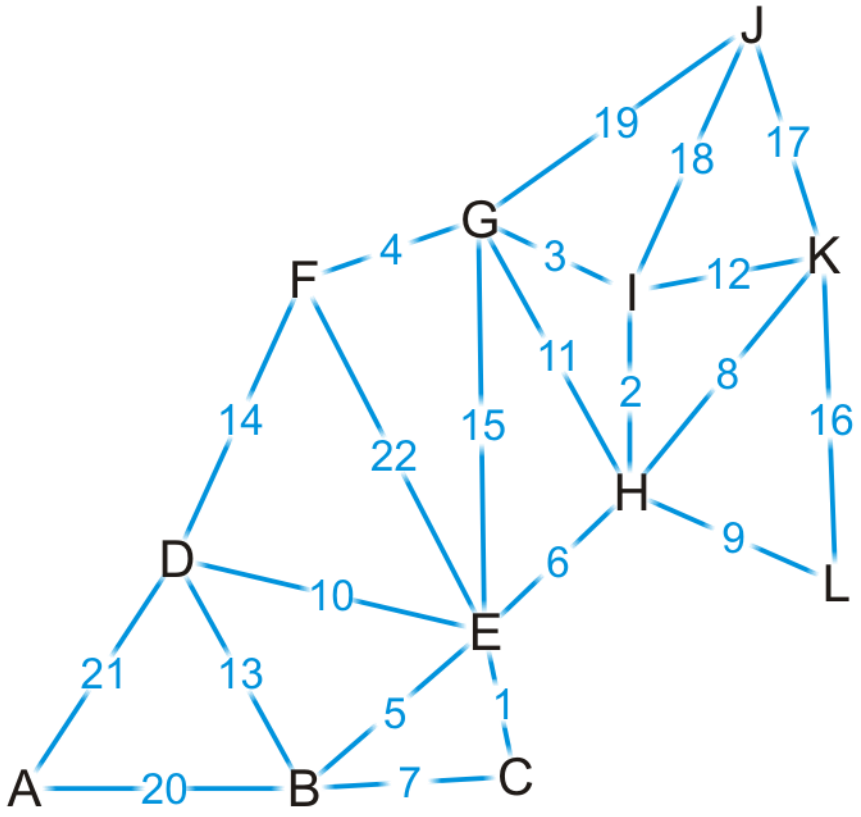




http://thunderbird37.com/tag/paker-brothers/

Example

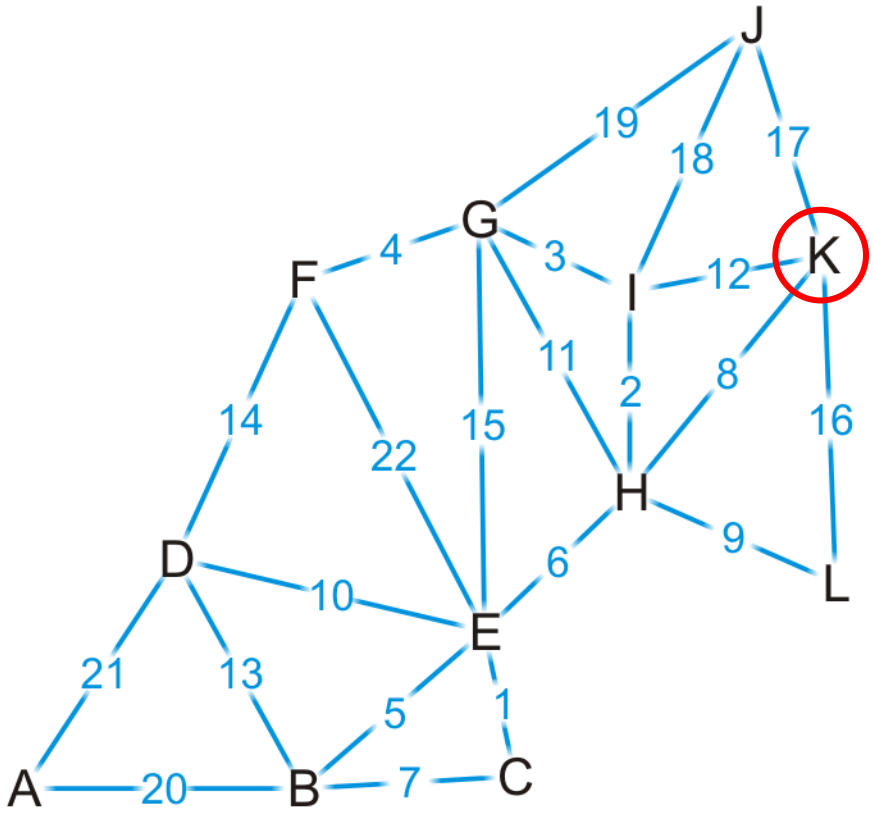
We set up our table as follows.



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

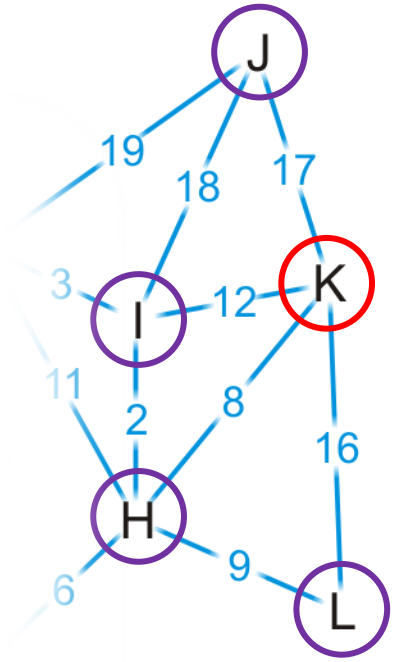
We visit vertex K



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

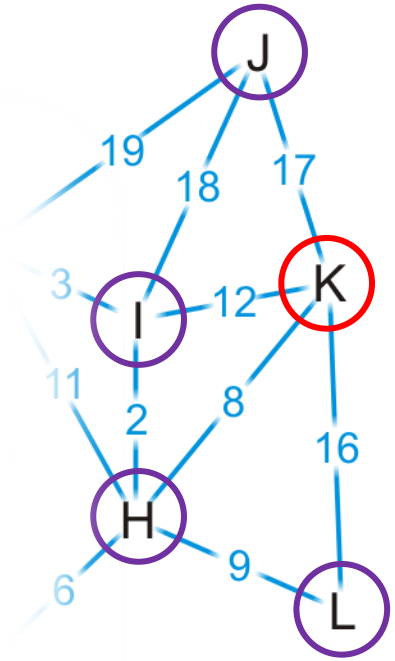
Vertex K has four neighbors: H, I, J and L



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

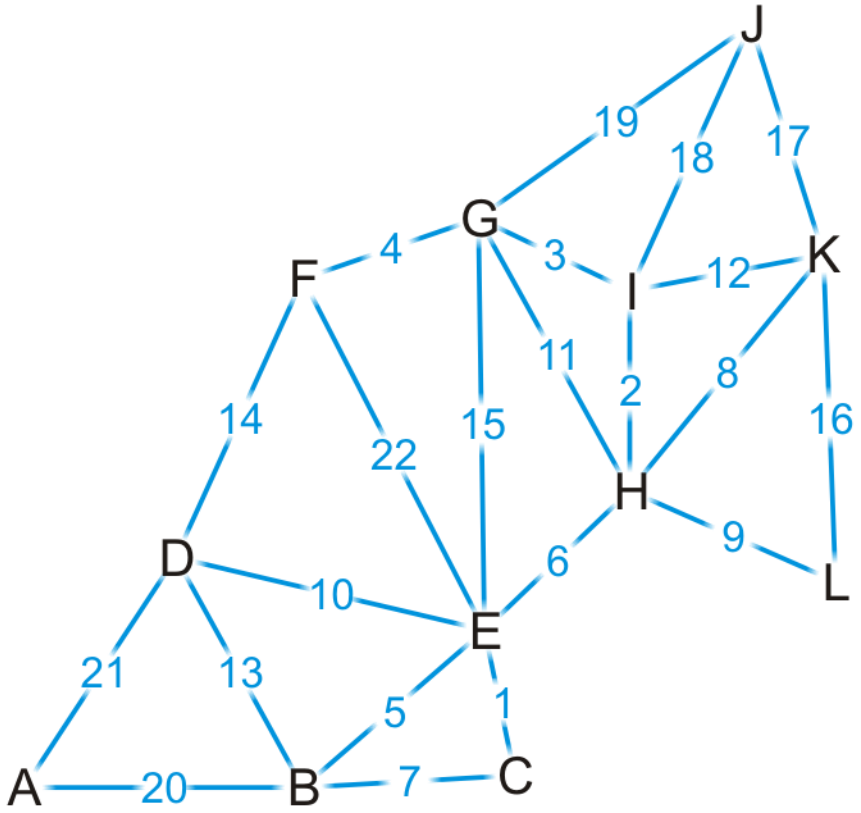


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	8	K
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We're finished with vertex K.

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

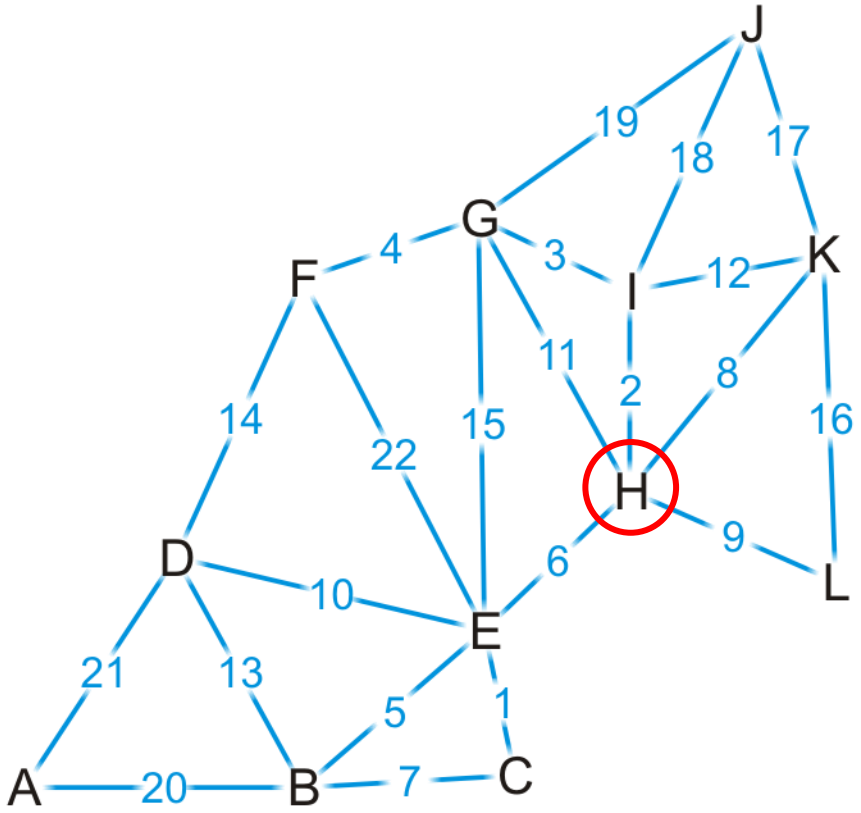


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	8	K
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

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



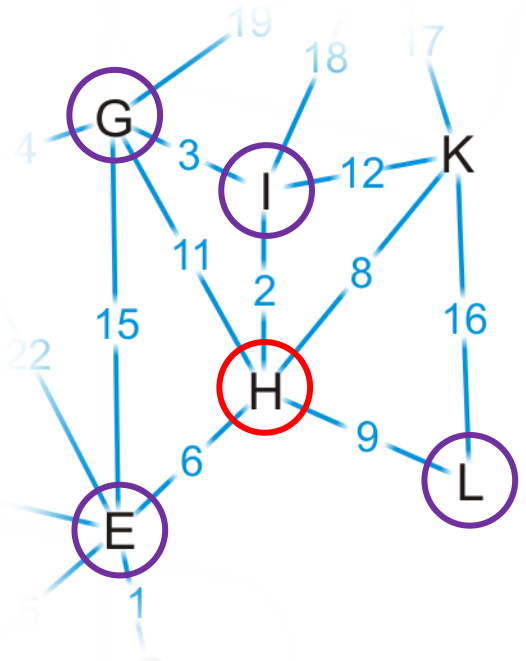
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	**T**	**8**	**K**
I	F	12	K
J	F	17	K
K	T	0	Ø
L	F	16	K

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

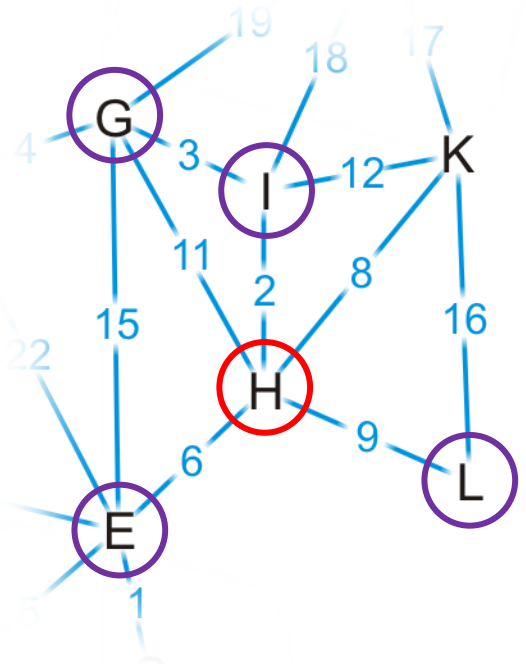
Which of these are shorter than any known path?



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

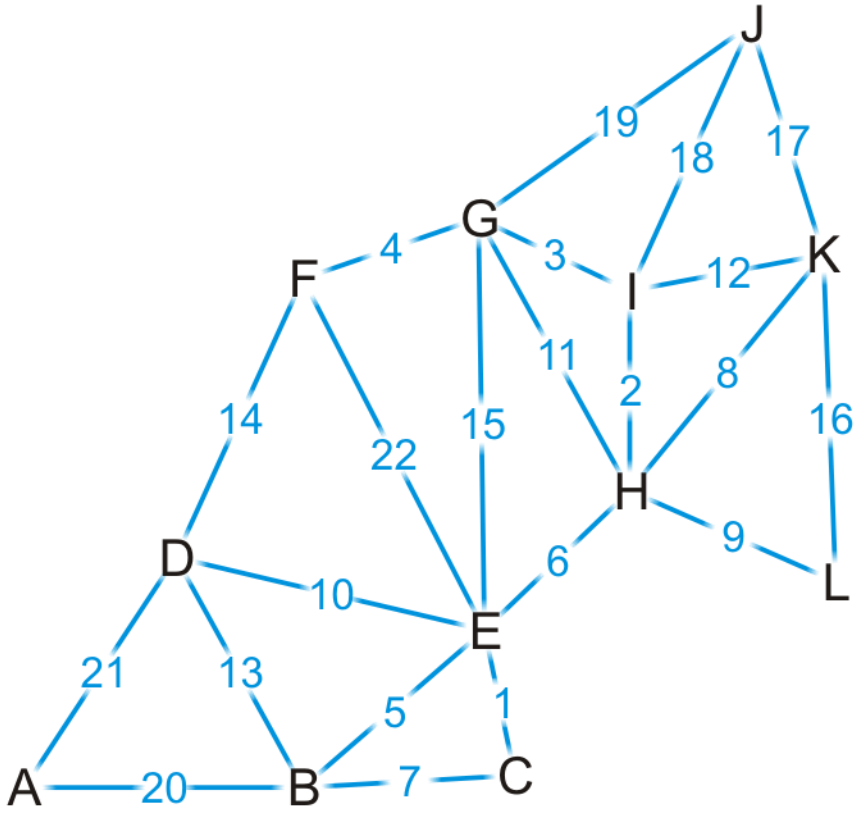


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	**14**	**H**
F	F	∞	Ø
G	F	**19**	**H**
H	**T**	**8**	**K**
I	F	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

We are finished with vertex H.

Which vertex do we visit next?

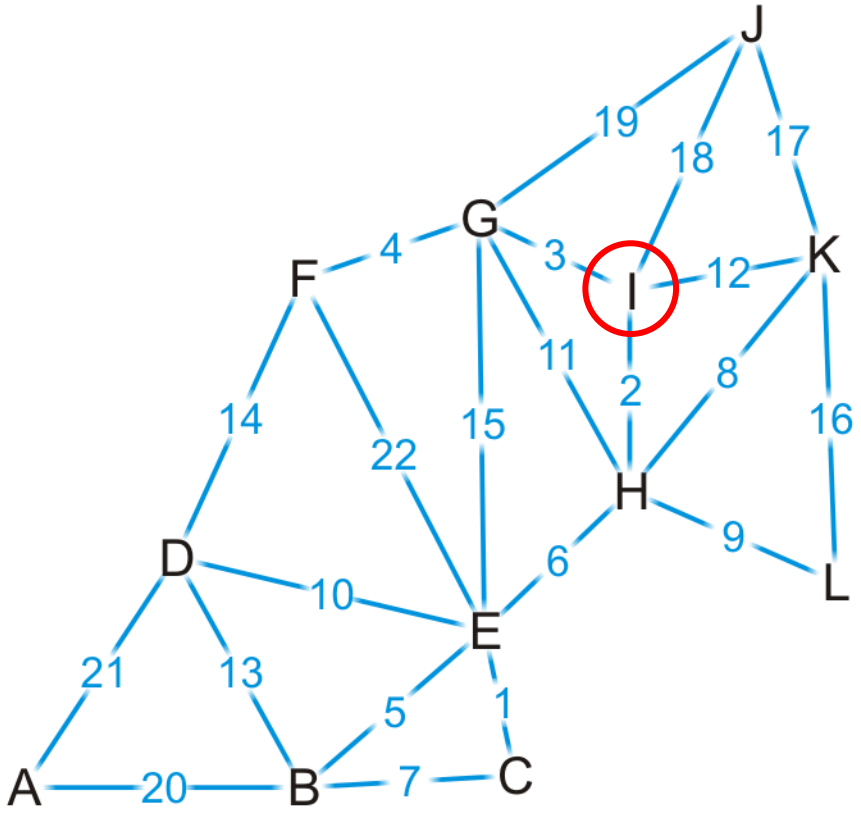


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	F	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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

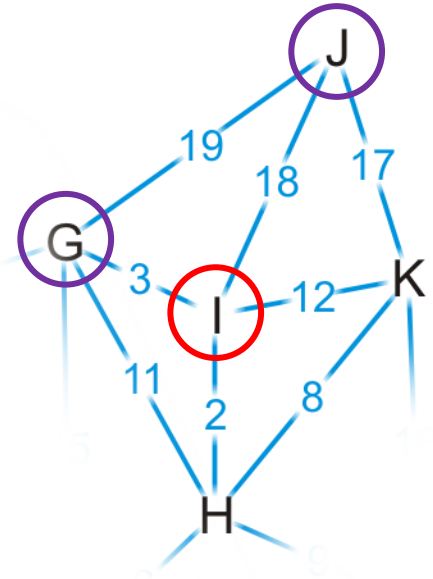


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	19	H
H	T	8	K
I	**T**	**10**	**H**
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

Consider these paths:

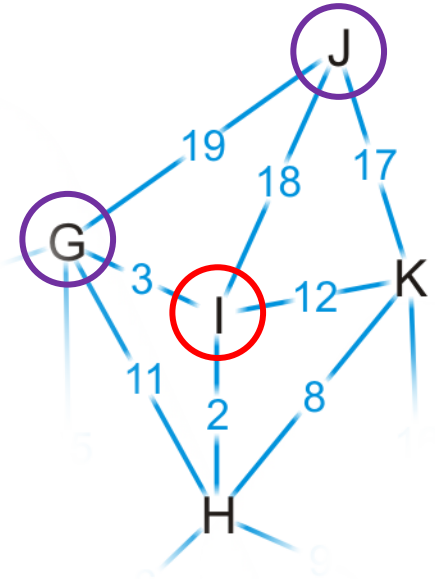
(K, H, I, G) of length 10 + 3 = 13 (K, H, I, J) of length 10 + 18 = 28



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

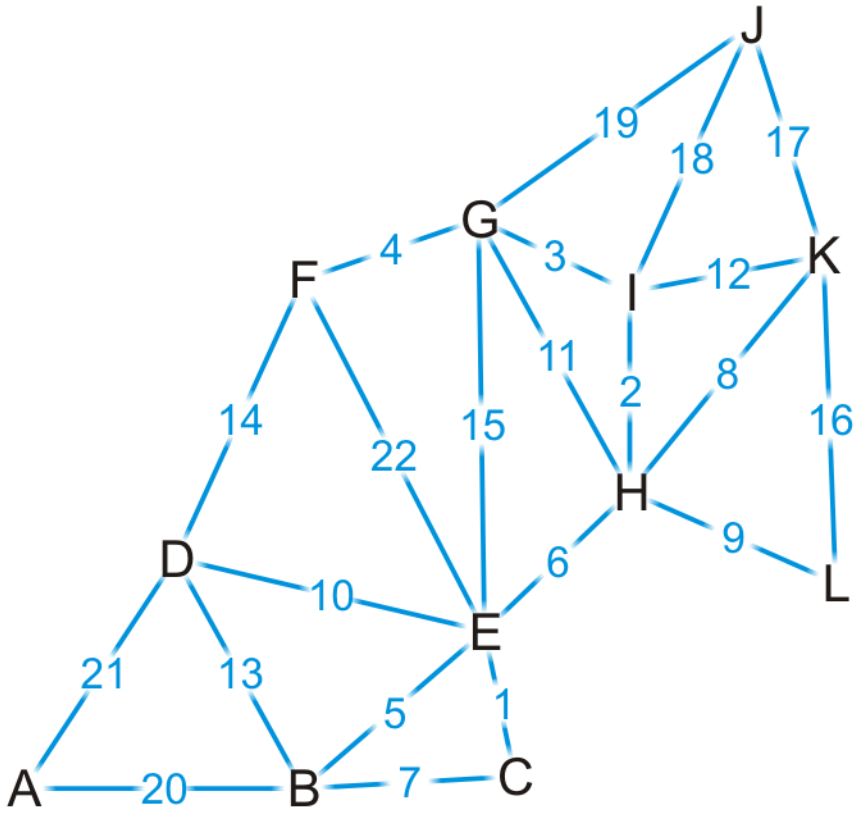
We have discovered a shorter path to vertex G, but (K, J) is still the shortest known path to vertex J



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?

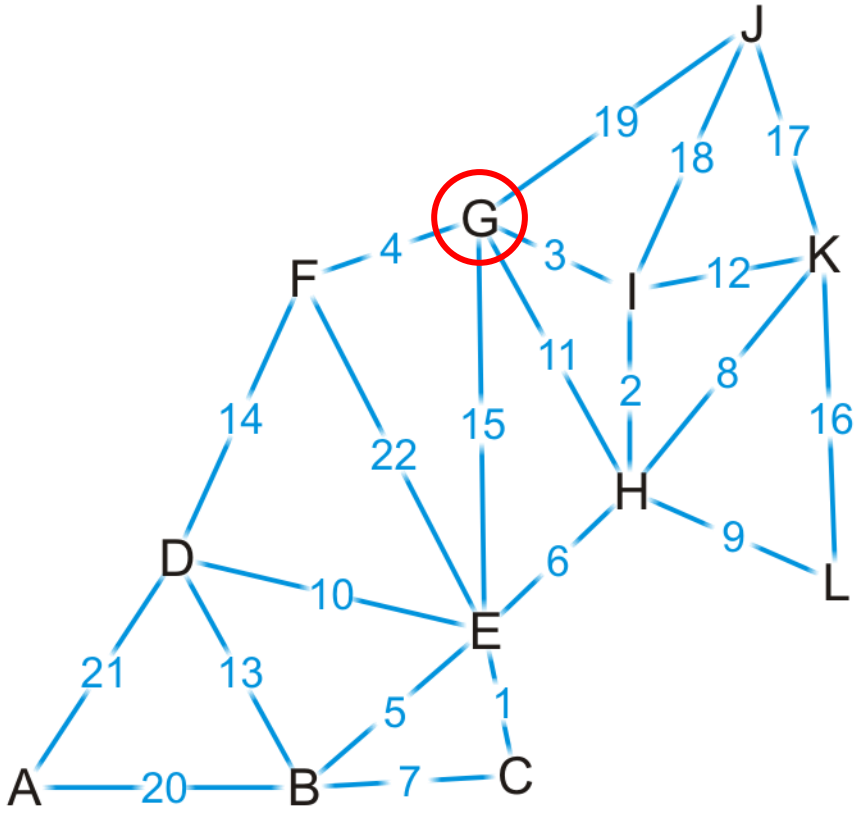


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	F	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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.



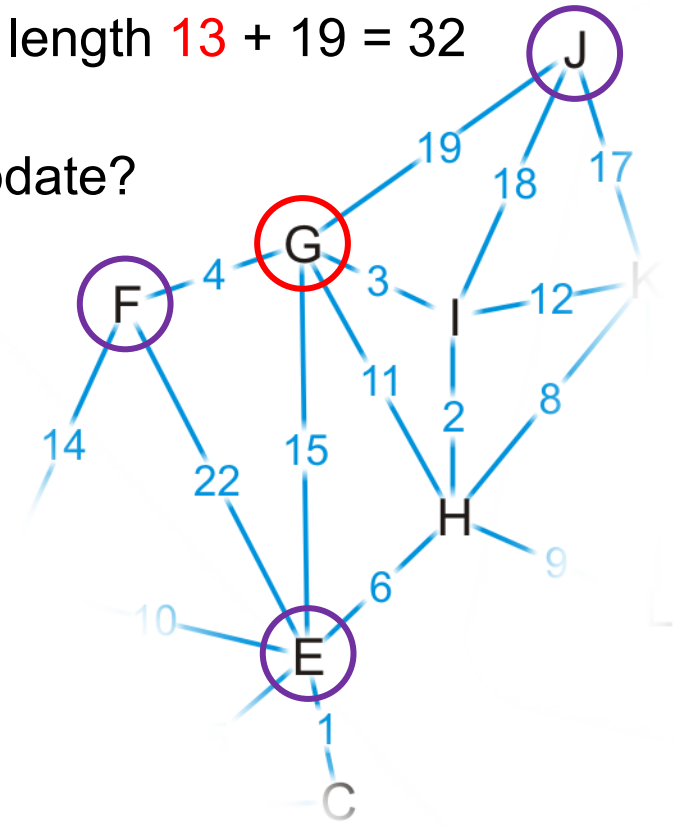
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	∞	Ø
G	**T**	**13**	**I**
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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

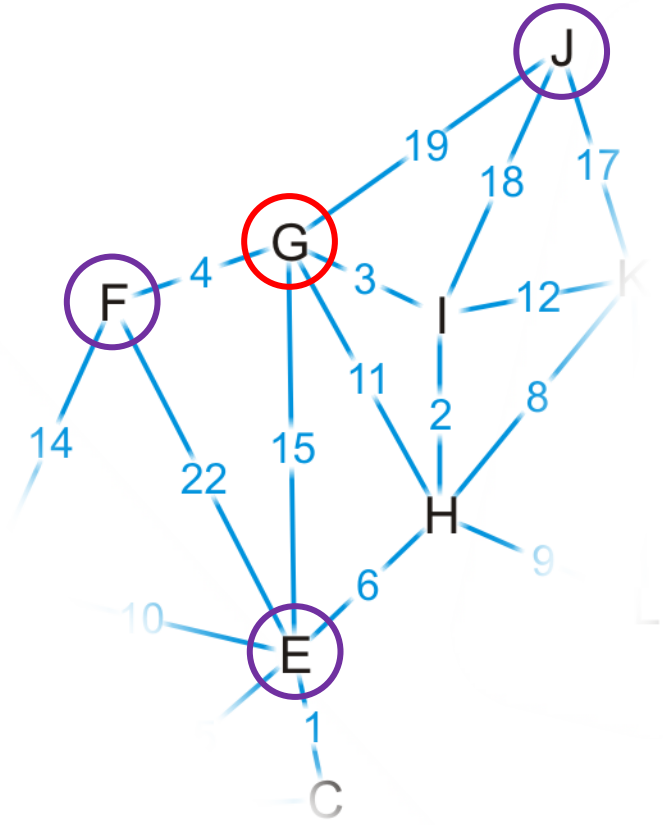
- Which do we update?



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

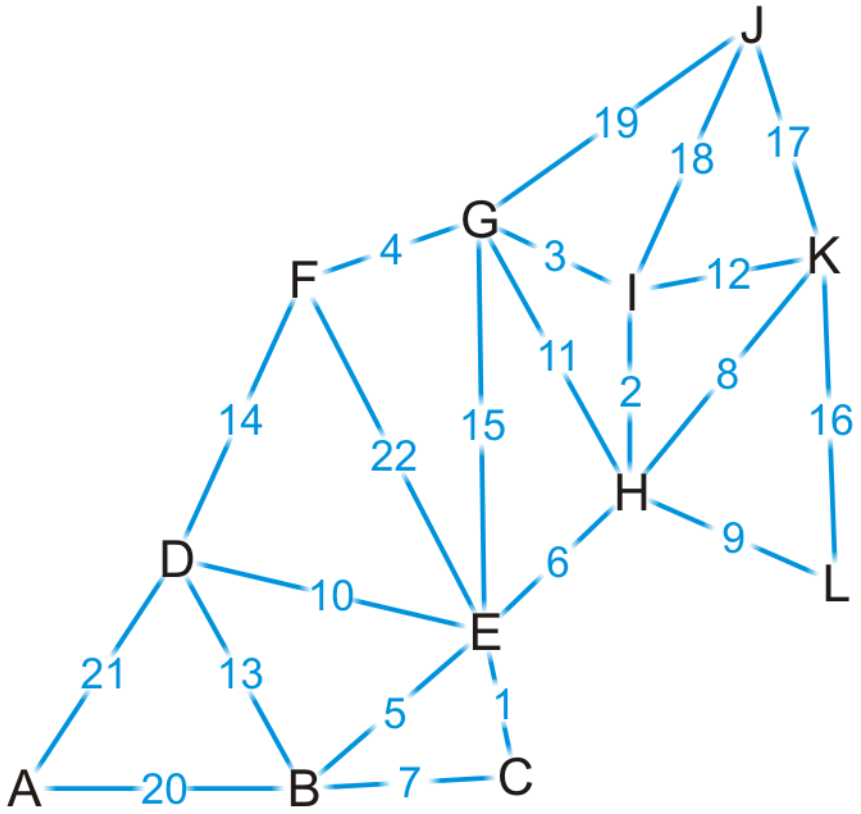
We have now found a path to vertex F



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?

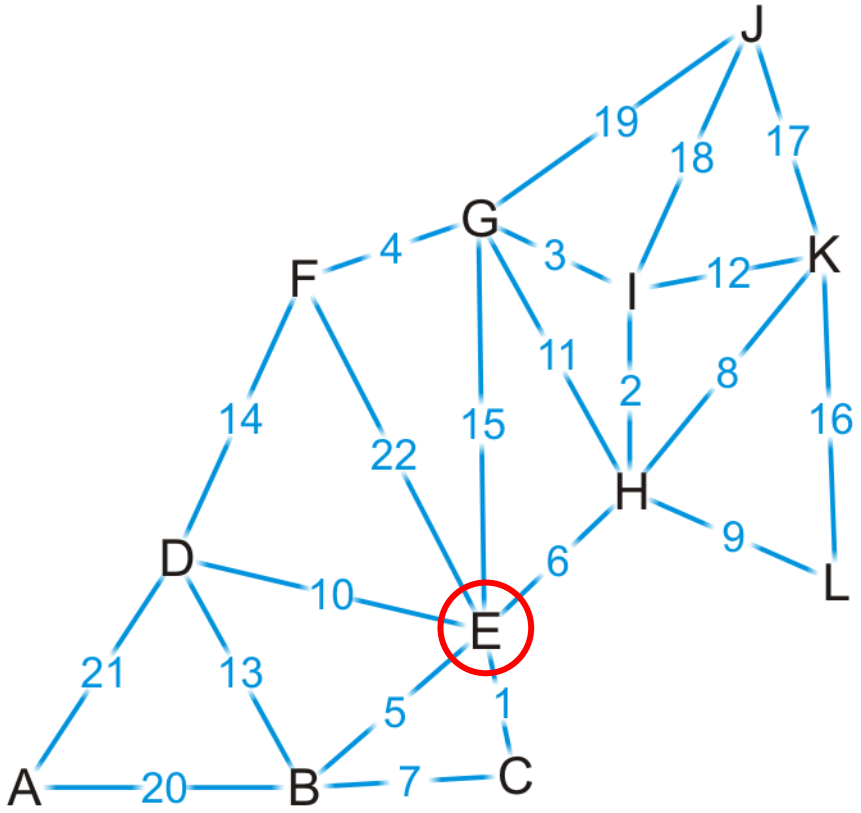


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

Vertex E has four unvisited neighbors: B, C, D and F.

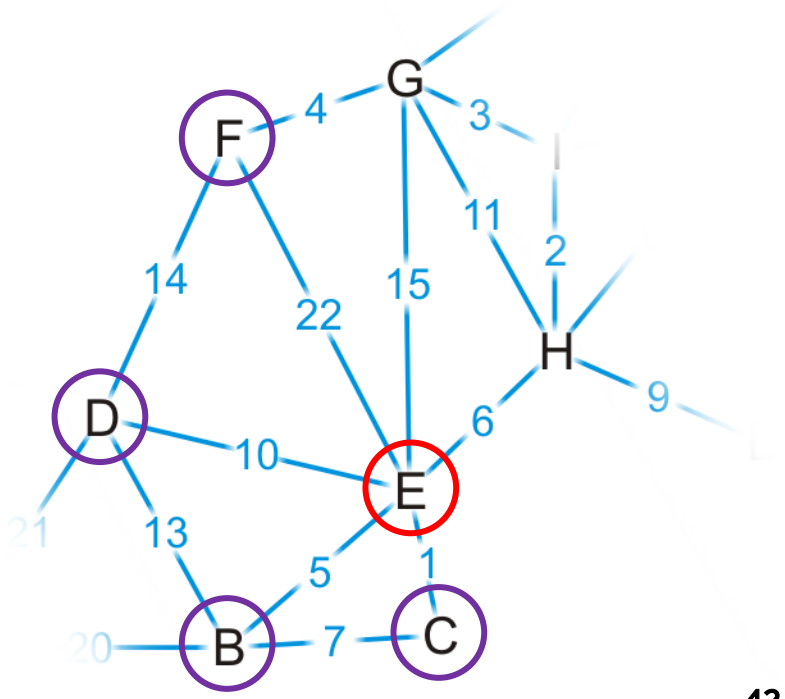


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	**T**	**14**	**H**
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

Vertex E 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

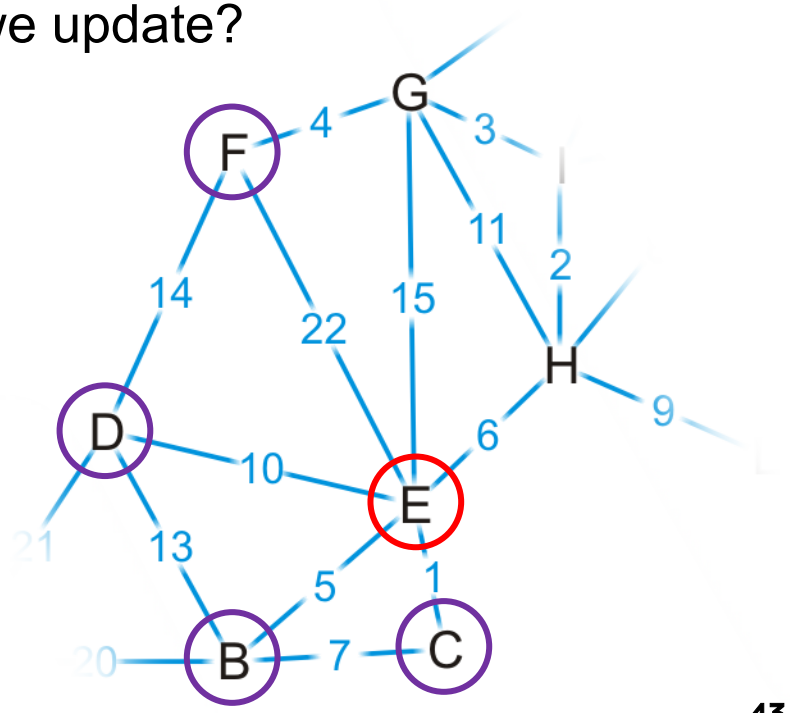
Consider these paths:

- (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

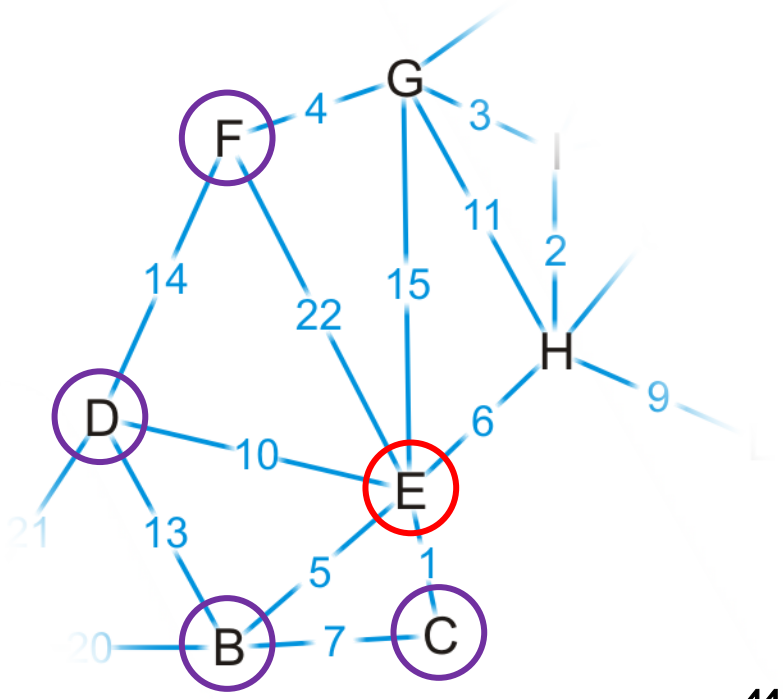
- Which do we update?



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

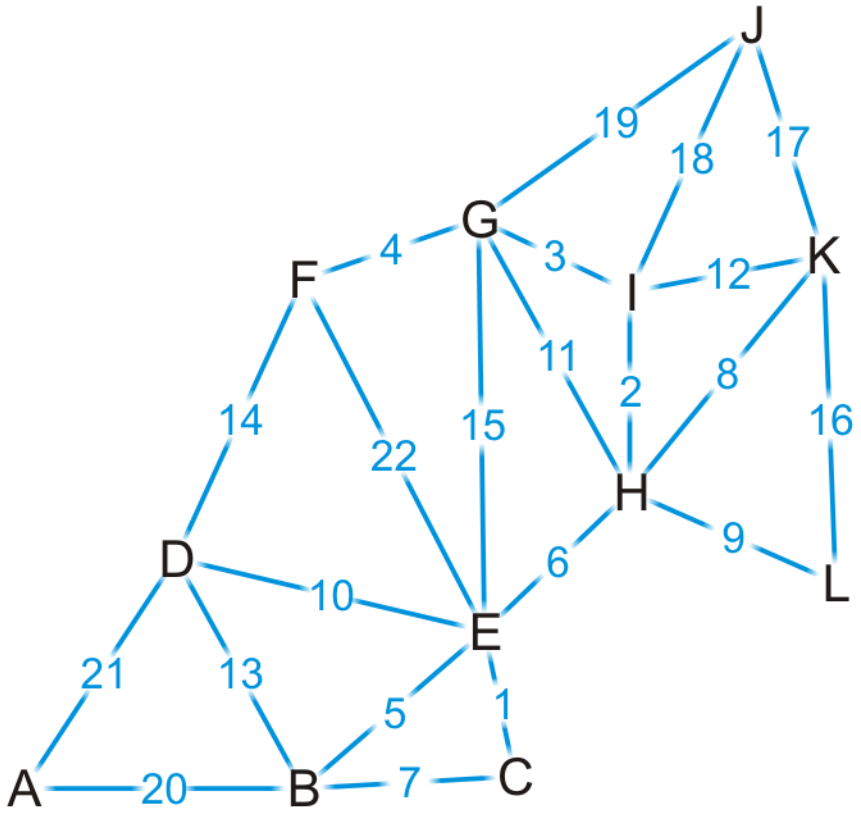
We've discovered paths to vertices B, C, D



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?

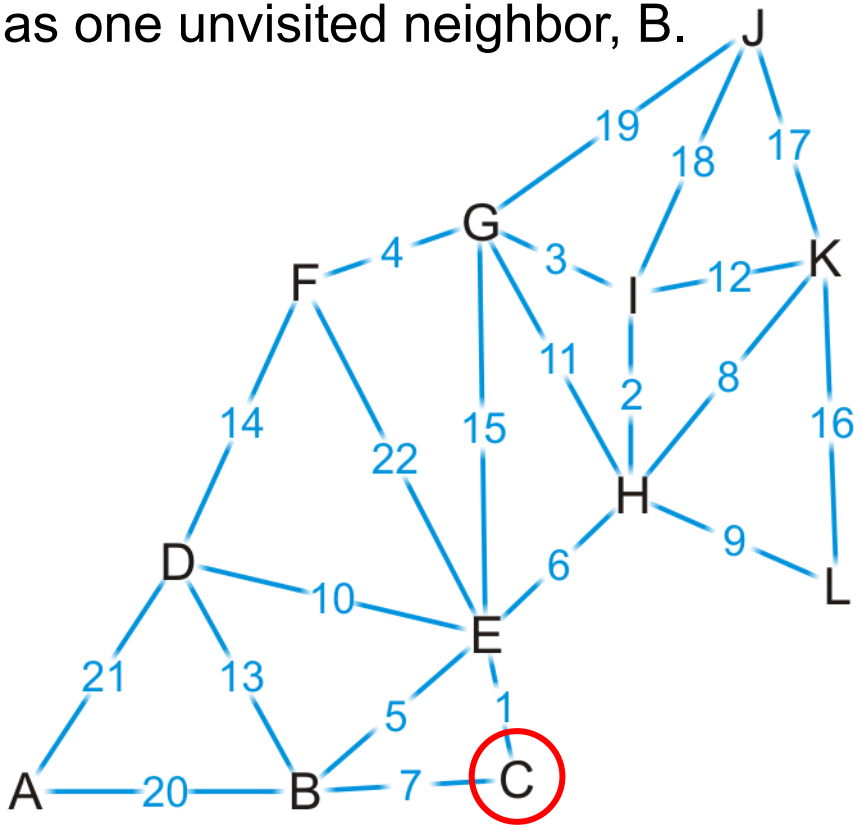


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	F	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

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.

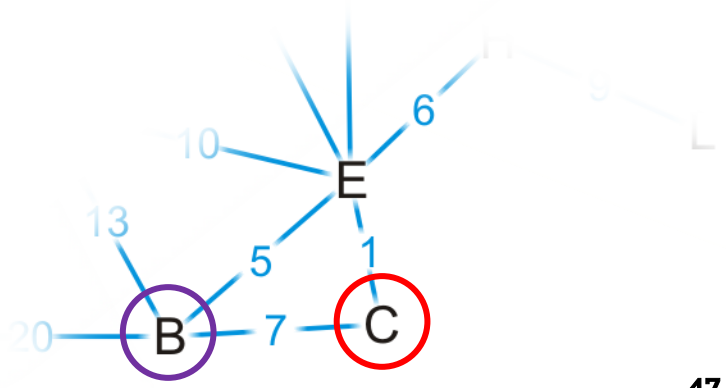


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	**T**	**15**	**E**
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

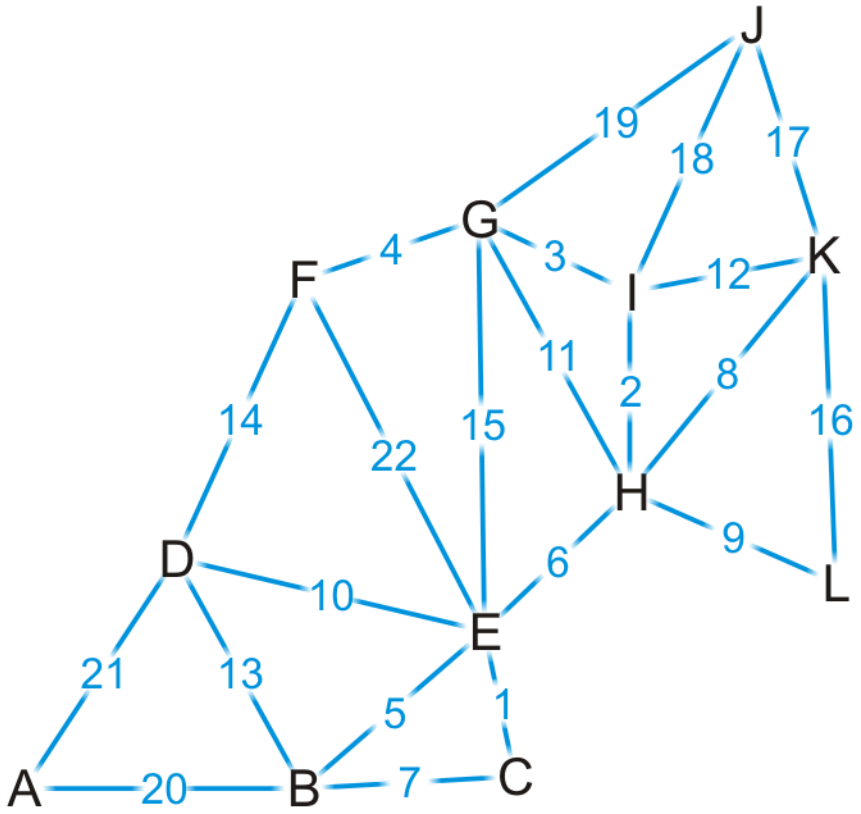
We have already discovered a shorter path through vertex E.



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?

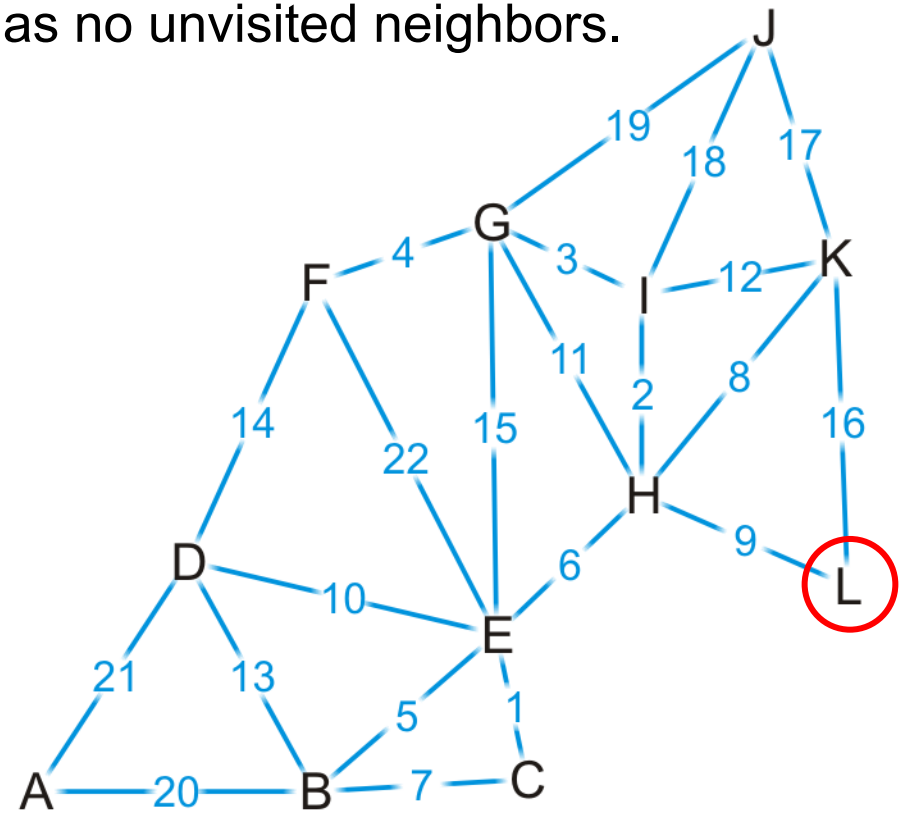


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	F	16	K

Example

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

Vertex L has no unvisited neighbors.

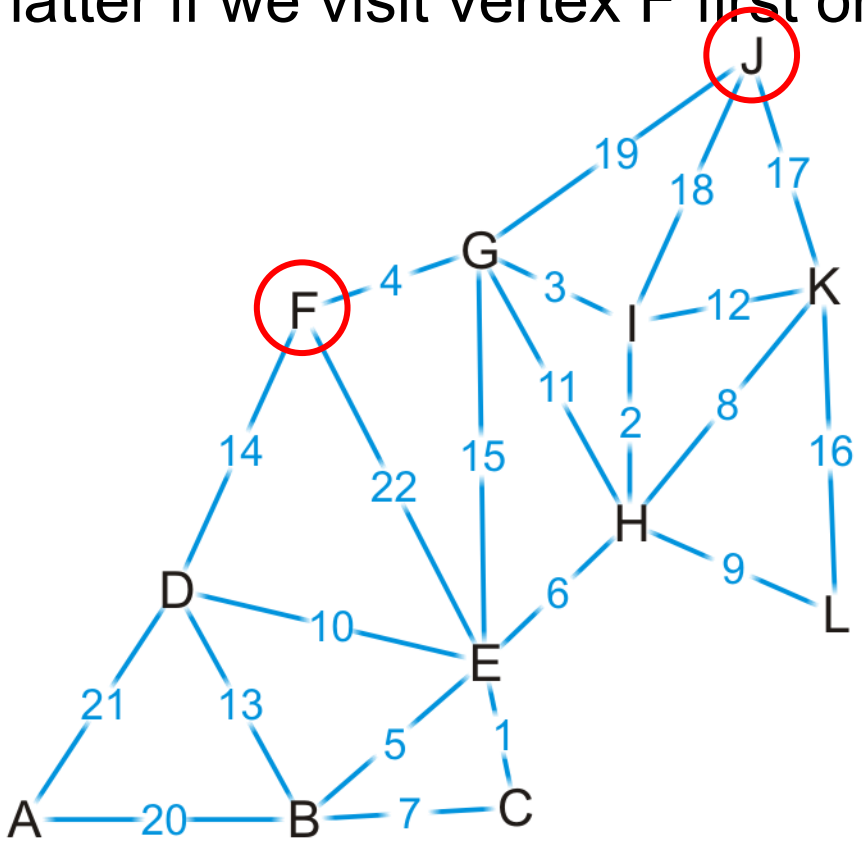


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	**T**	**16**	**K**

Example

Where to next?

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

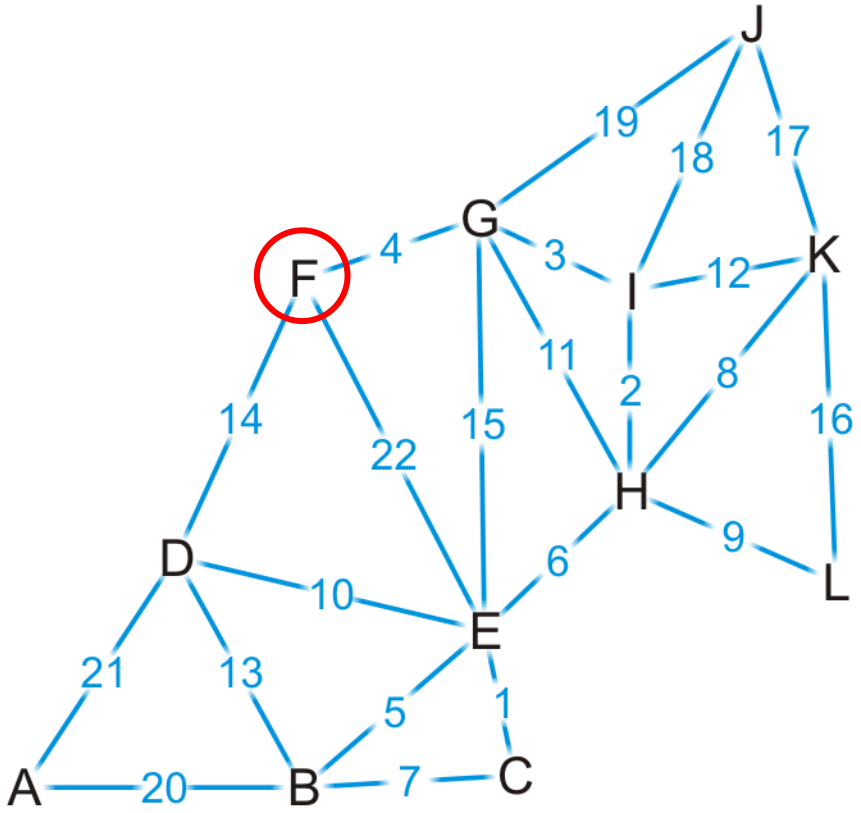


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	F	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Let's visit vertex F first.

It has one unvisited neighbor, vertex D

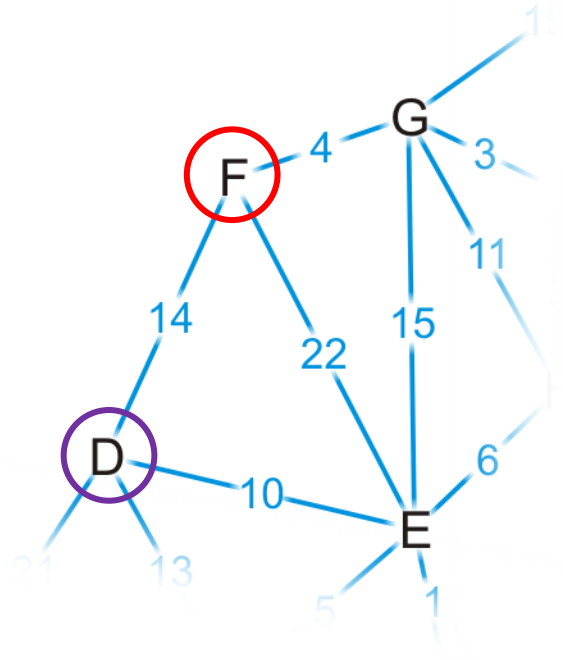


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

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.

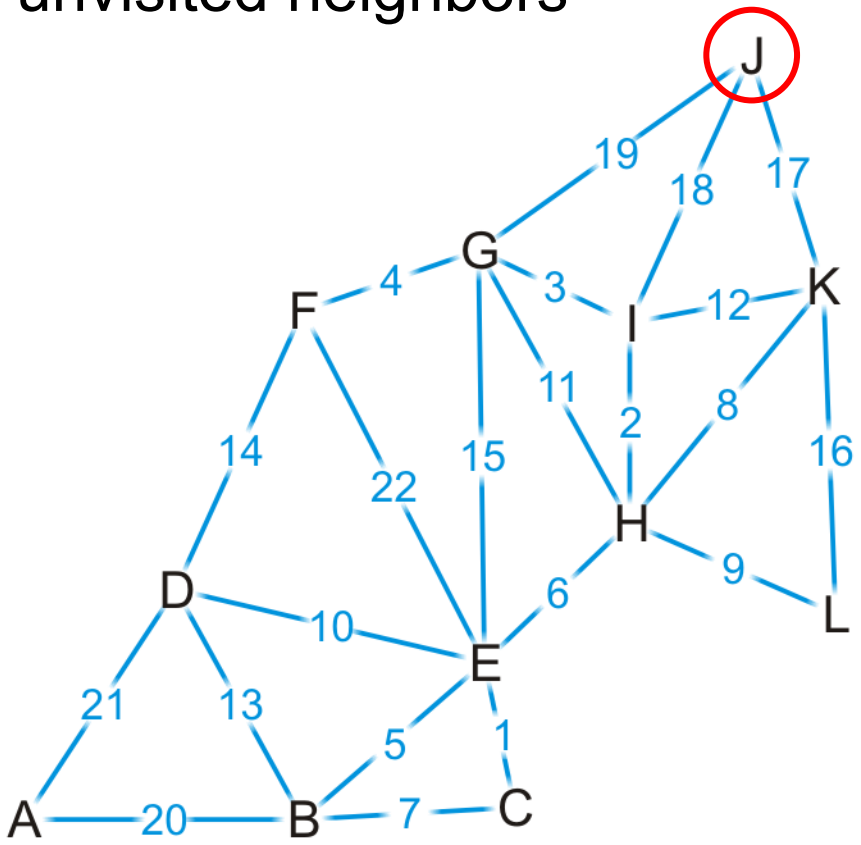


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	**T**	**17**	**G**
G	T	13	I
H	T	8	K
I	T	10	H
J	F	17	K
K	T	0	Ø
L	T	16	K

Example

Now we visit vertex J.

It has no unvisited neighbors

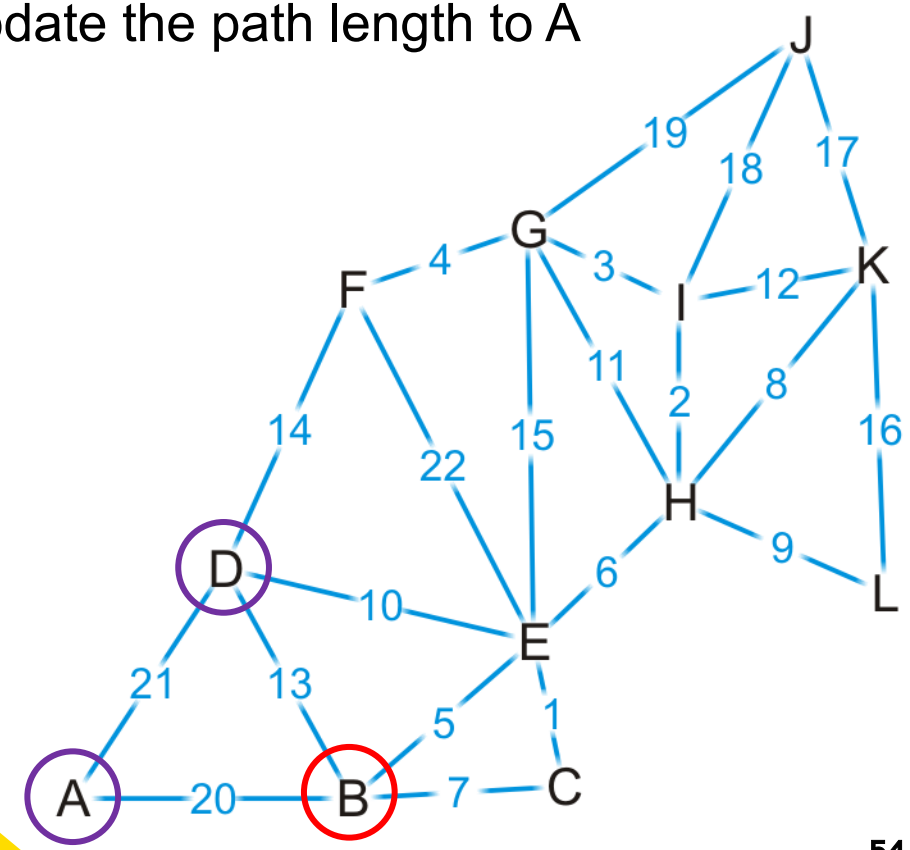


Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	19	E
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	**T**	**17**	**K**
K	T	0	Ø
L	T	16	K

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



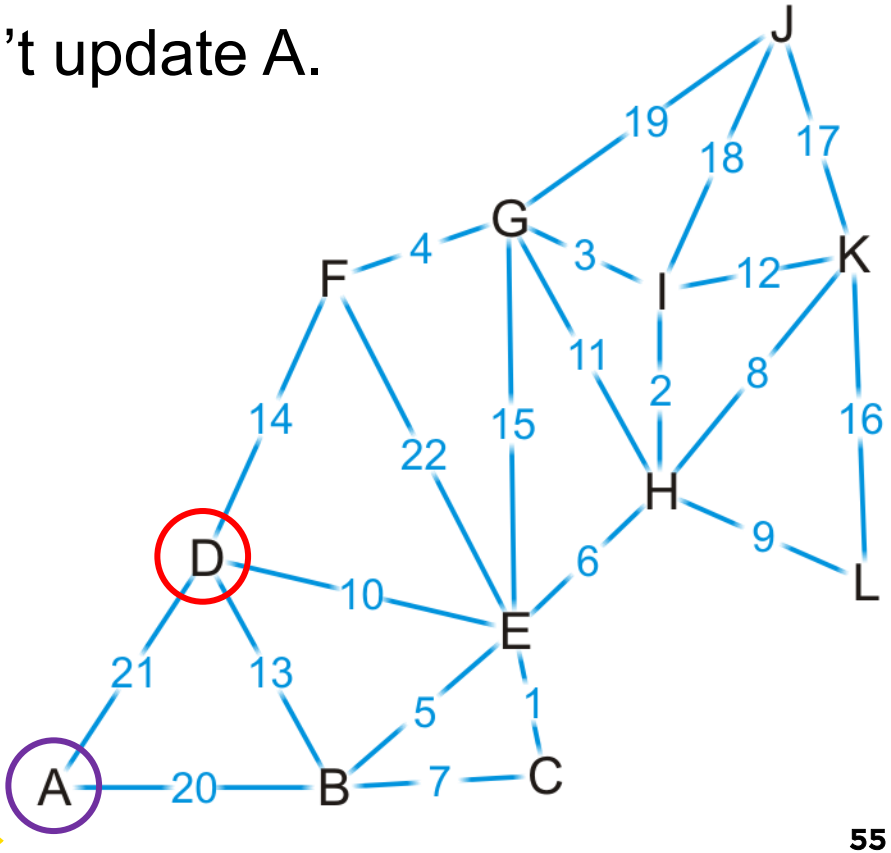
Vertex	Visited	Distance	Previous
A	F	**39**	**B**
B	**T**	**19**	**E**
C	T	15	E
D	F	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Next we visit vertex D

The path (K, H, E, D, A) is of length 24 + 21 = 45

We don't update A.



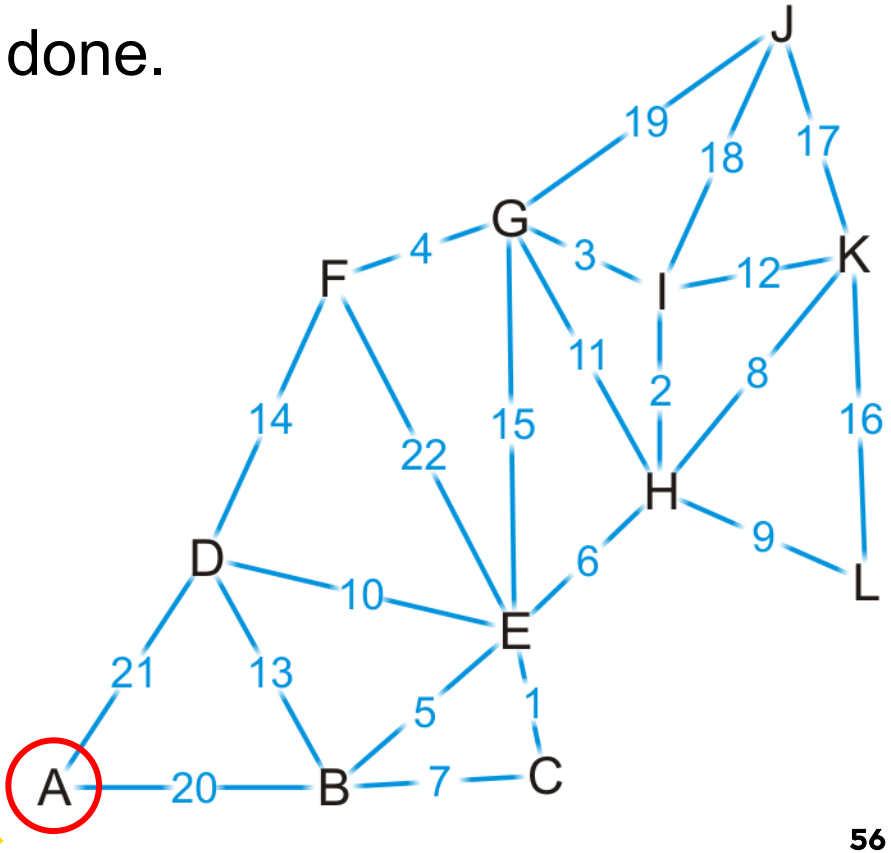
Vertex	Visited	Distance	Previous
A	F	39	B
B	T	19	E
C	T	15	E
D	**T**	**24**	**E**
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Example

Finally, we visit vertex A.

It has no unvisited neighbors and there are no unvisited vertices left.

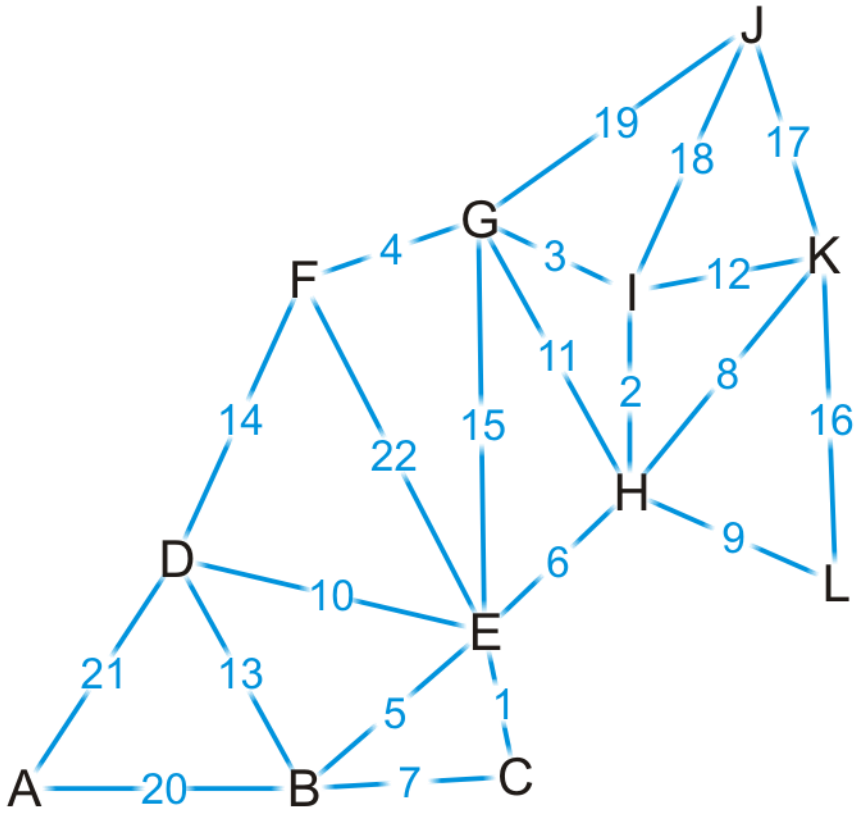
We are done.



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

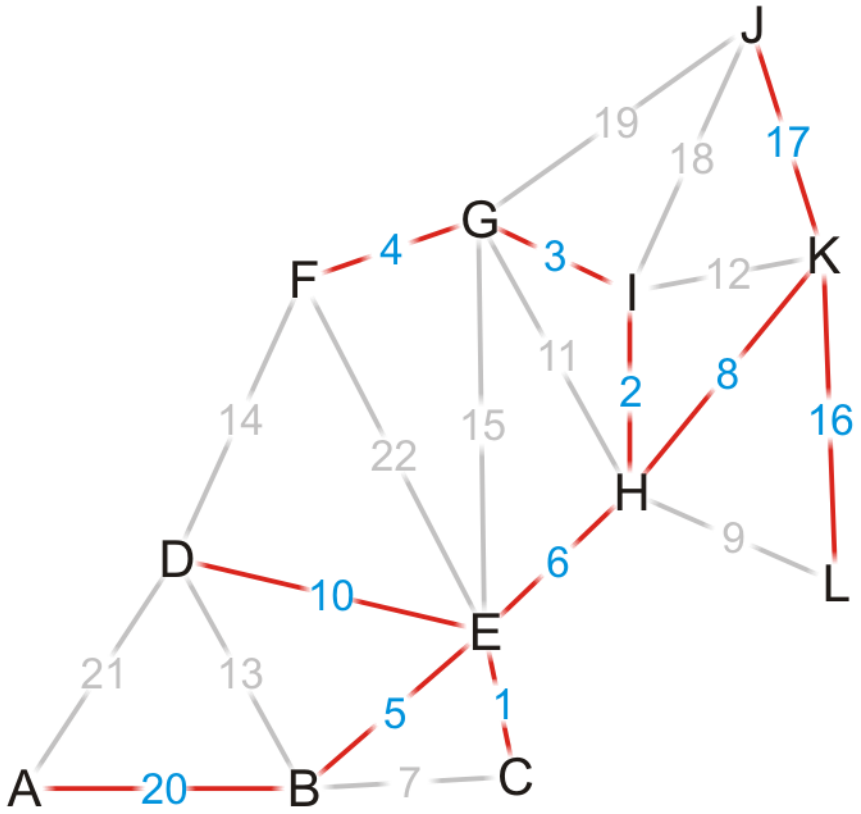
Thus, we have found the shortest path from vertex K to each of the other vertices



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



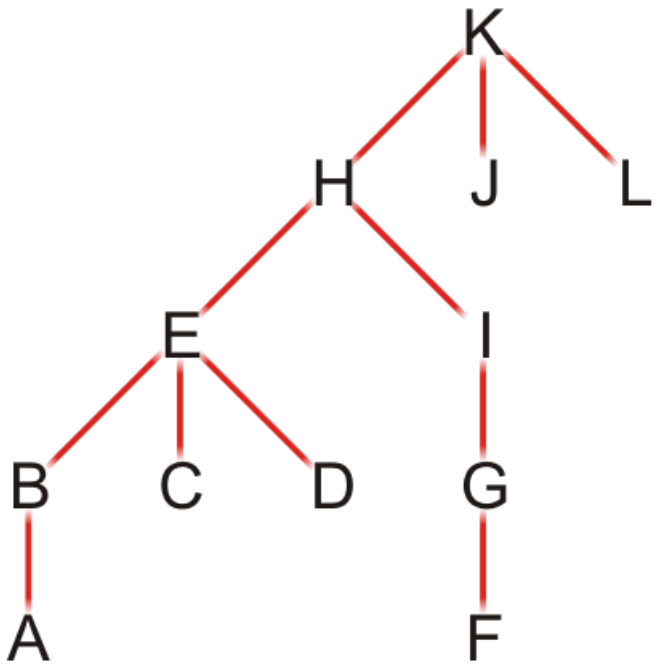
Vertex	Visited	Distance	Previous
A	T	39	B
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

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 $O(|V|)$ memory and run time

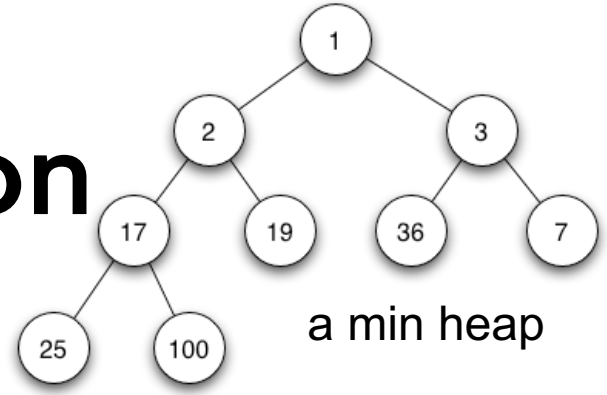
We iterate $|V| - 1$ times, each time finding the unvisited *closest* vertex

- Iterating through the table requires is $O(|V|)$ time
- Each time we find a vertex, we must check all of its neighbors
- With an adjacency matrix, the run time is $O(|V|(|V| + |V|)) = O(|V|^2)$
- With an adjacency list, the run time is $O(|V|^2 + |E|) = O(|V|^2)$ as $|E| = O(|V|^2)$

Can we do better?

Recall, we only need the vertex with the shortest distance next. We can use min-heap similar as the Prim's algorithm.

Min-heap-based optimization



a min heap

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

- The min heap requires $O(|V|)$ memory which **contains the shortest distance of all the vertices from the source vertex**

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

- Obtain the shortest distance from the min heap O(1), maintain the heap $O(\log(|V|))$
- The work required for this is $O(|V| \log(|V|))$

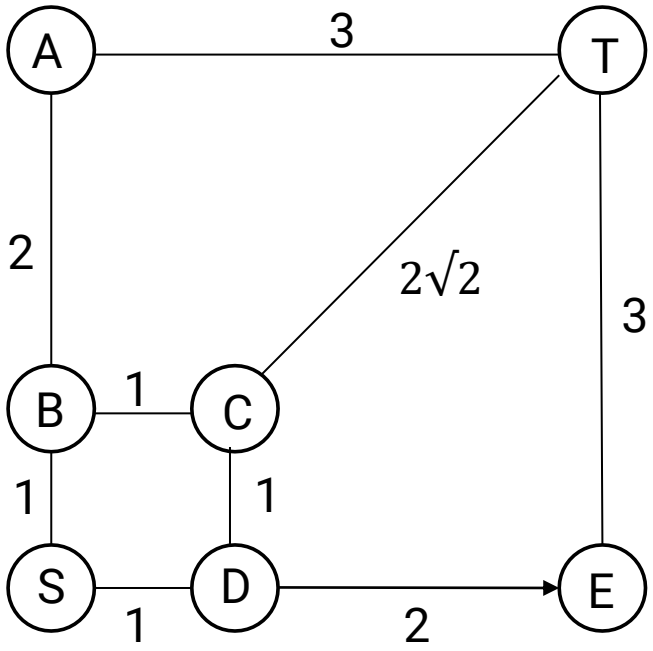
Is this all the work that is necessary?

- Recall that for the *closest* vertex in an iteration, **we try to update the distance of all its neighbors**, thus there are O(|E|) updates in total and each update in the heap requires $O(\log(|V|))$.
- Thus, the work required for this is $O(|E| \log(|V|))$

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

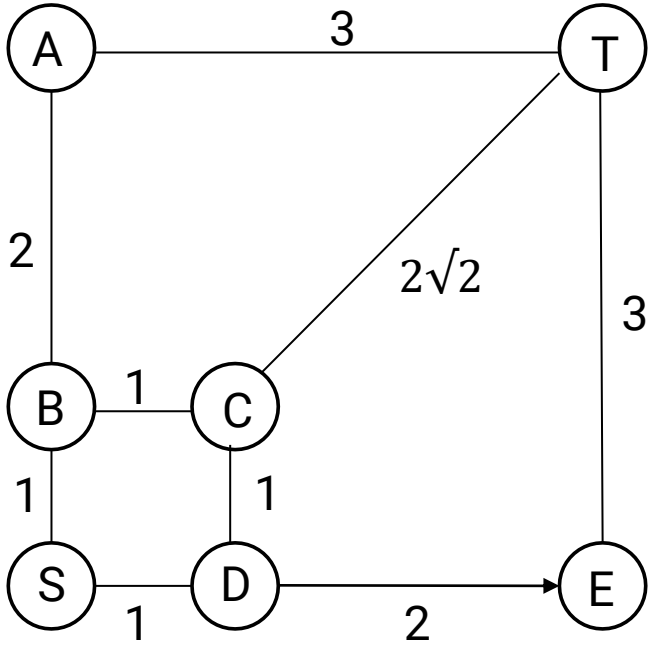
Quick Exercise

Given the graph below, apply Dijkstra's Algorithm to find the shortest paths from S.



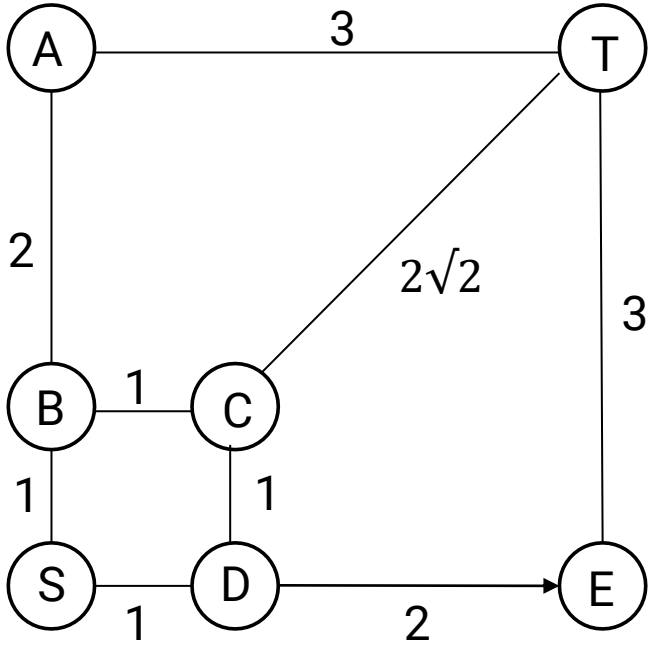
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



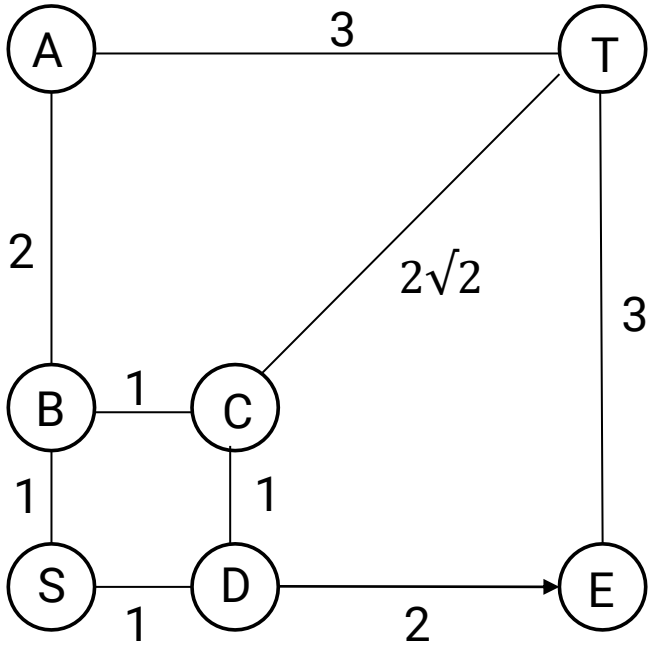
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



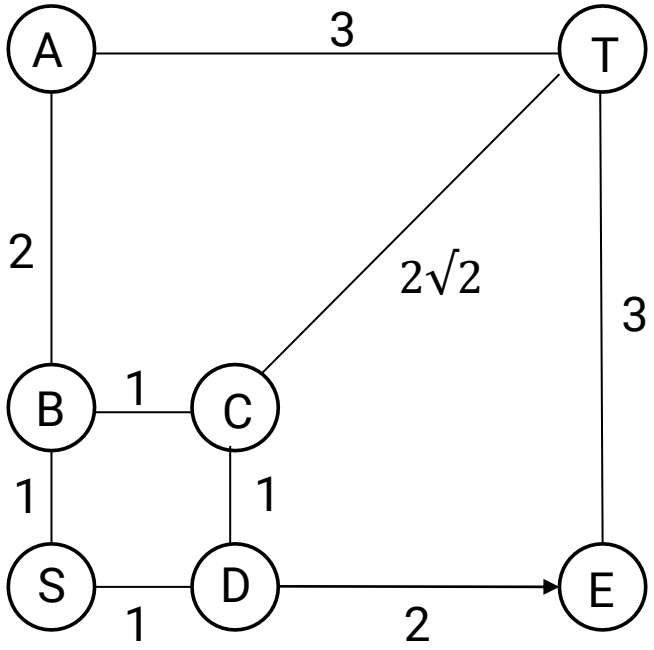
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	1	S
C	F	∞	Ø
D	F	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



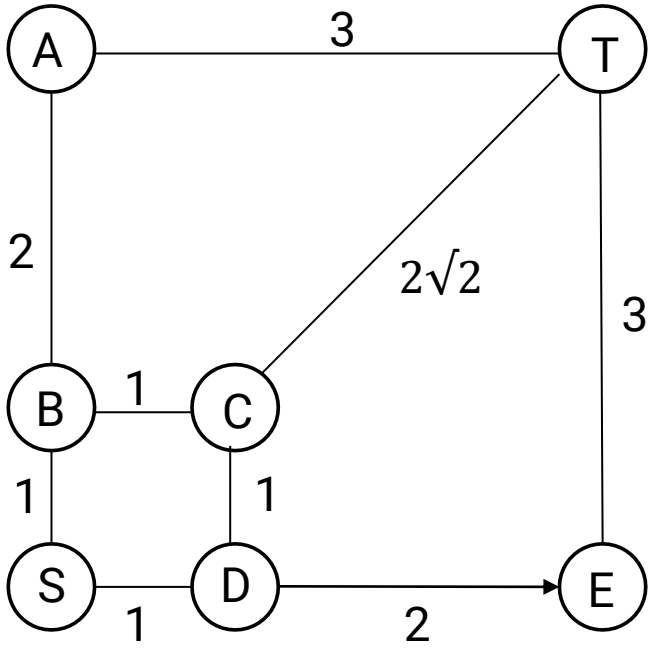
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	T	1	S
C	F	∞	Ø
D	F	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



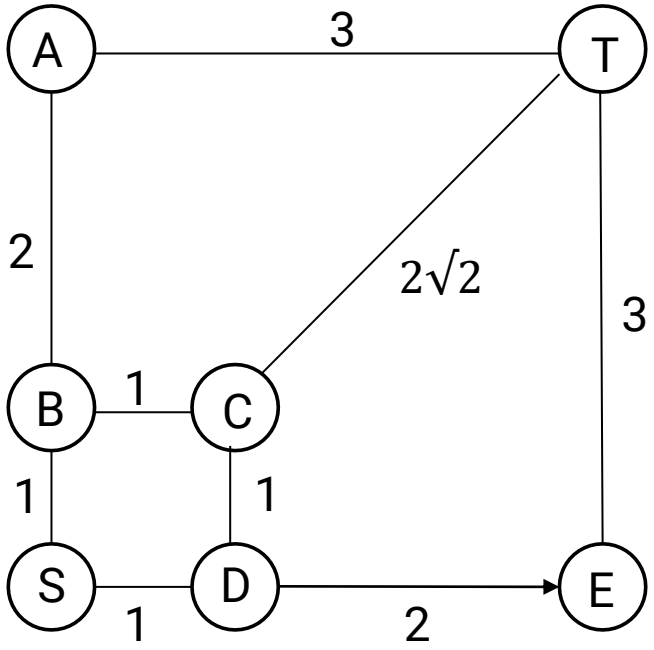
Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	T	1	S
C	F	∞	Ø
D	F	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



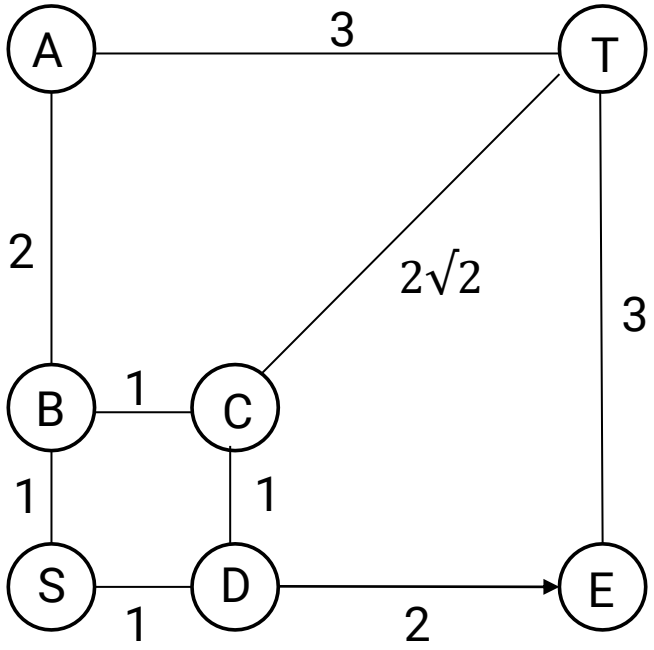
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	F	2	B
D	F	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



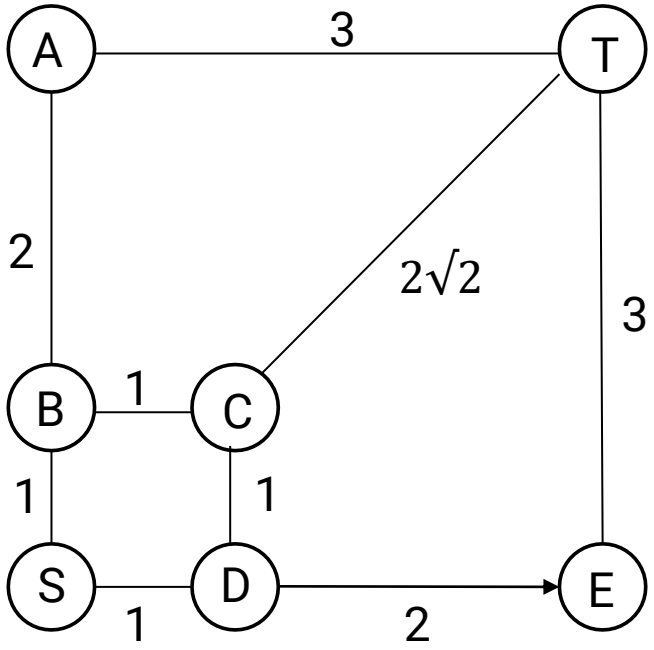
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	F	2	B
D	T	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



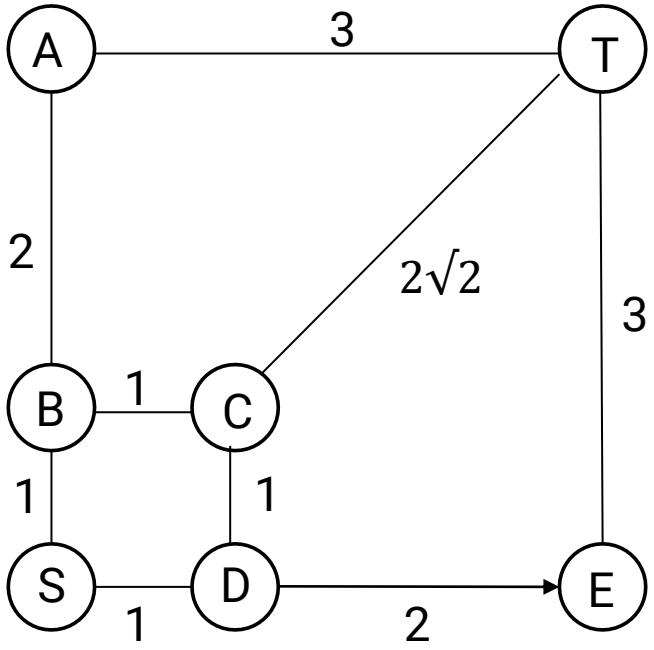
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	F	2	B
D	T	1	S
E	F	∞	Ø
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



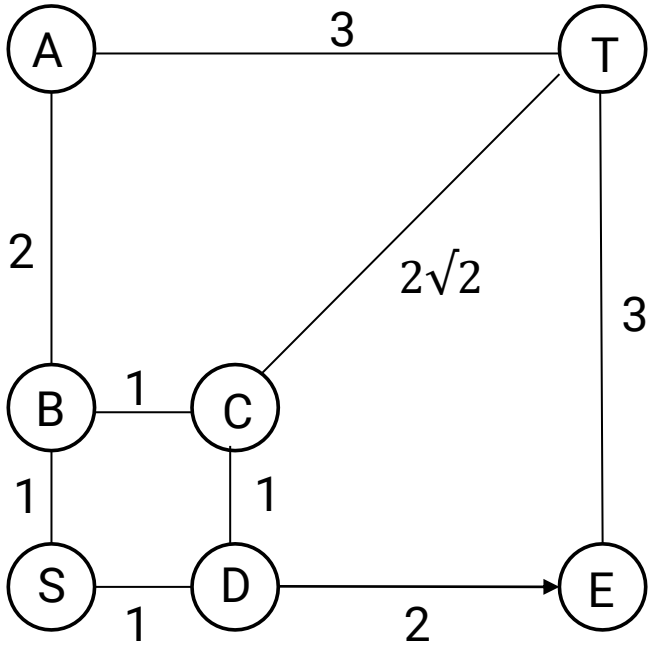
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	F	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



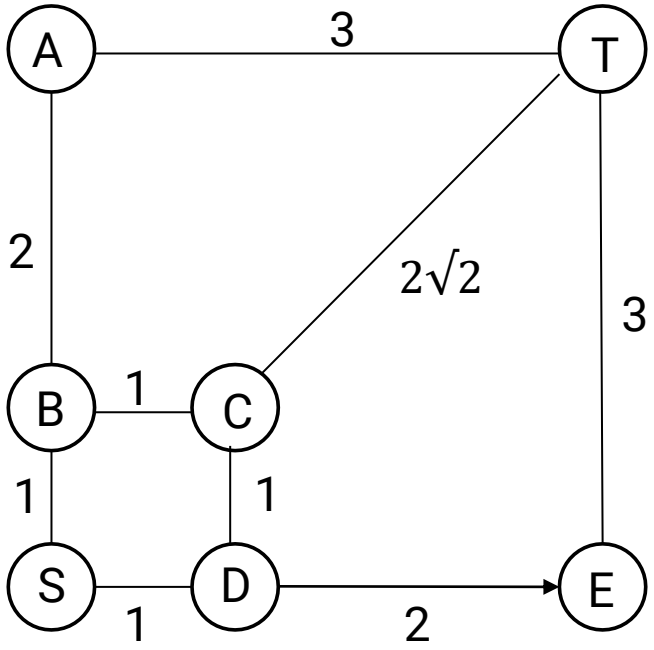
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



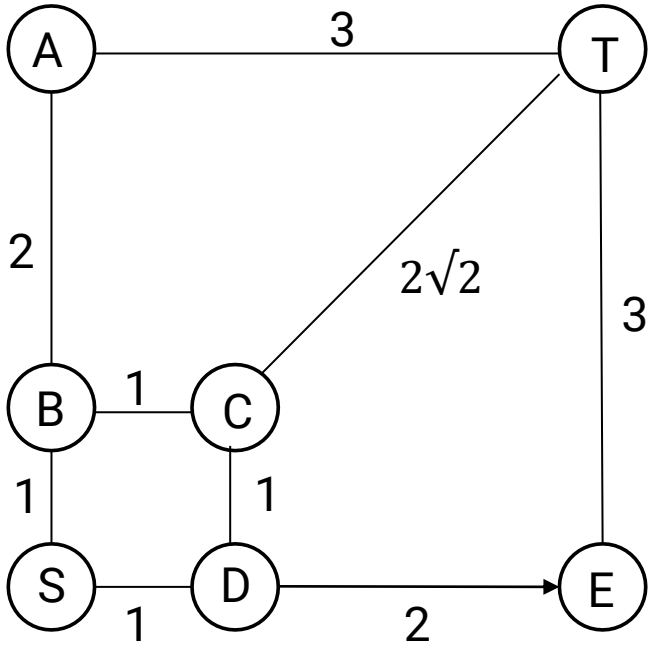
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	∞	Ø

Quick Exercise



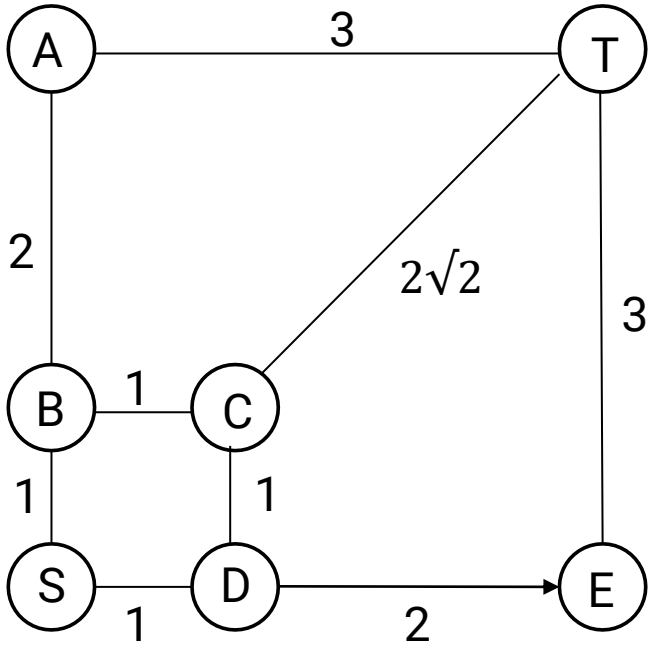
Vertex	Visited	Distance	Previous
A	F	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	$2+2\sqrt{2}$	C

Quick Exercise



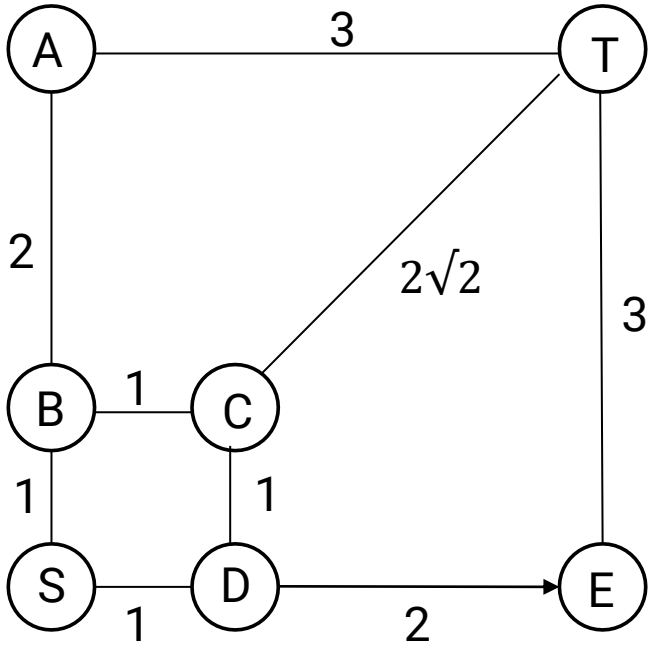
Vertex	Visited	Distance	Previous
A	T	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	$2+2\sqrt{2}$	C

Quick Exercise



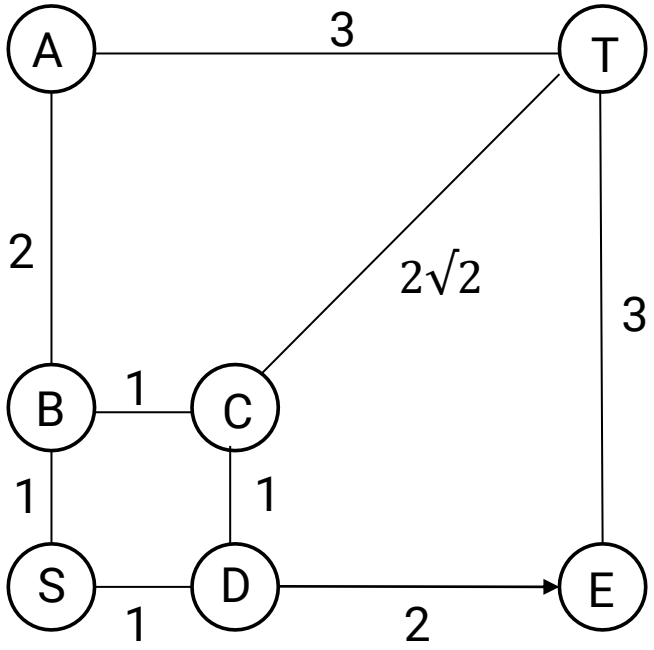
Vertex	Visited	Distance	Previous
A	T	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	F	3	D
S	T	0	Ø
T	F	$2+2\sqrt{2}$	C

Quick Exercise



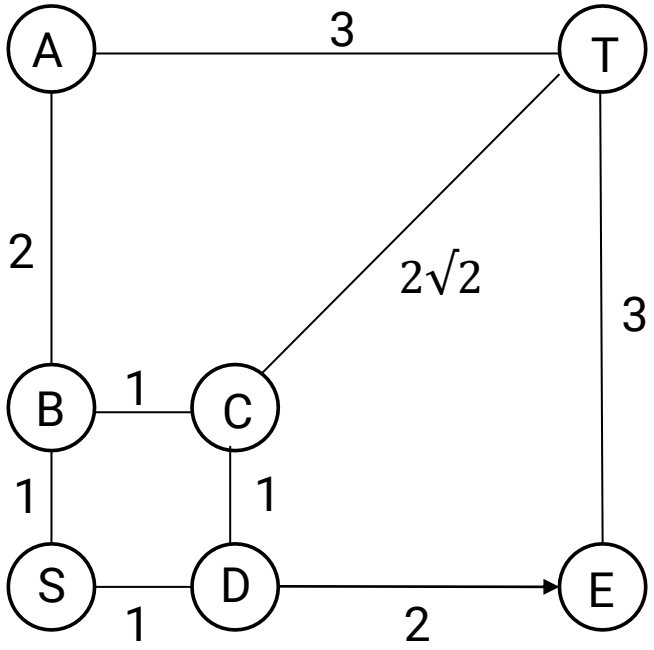
Vertex	Visited	Distance	Previous
A	T	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	T	3	D
S	T	0	Ø
T	F	$2+2\sqrt{2}$	C

Quick Exercise



Vertex	Visited	Distance	Previous
A	T	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	T	3	D
S	T	0	Ø
T	F	$2+2\sqrt{2}$	C

Quick Exercise

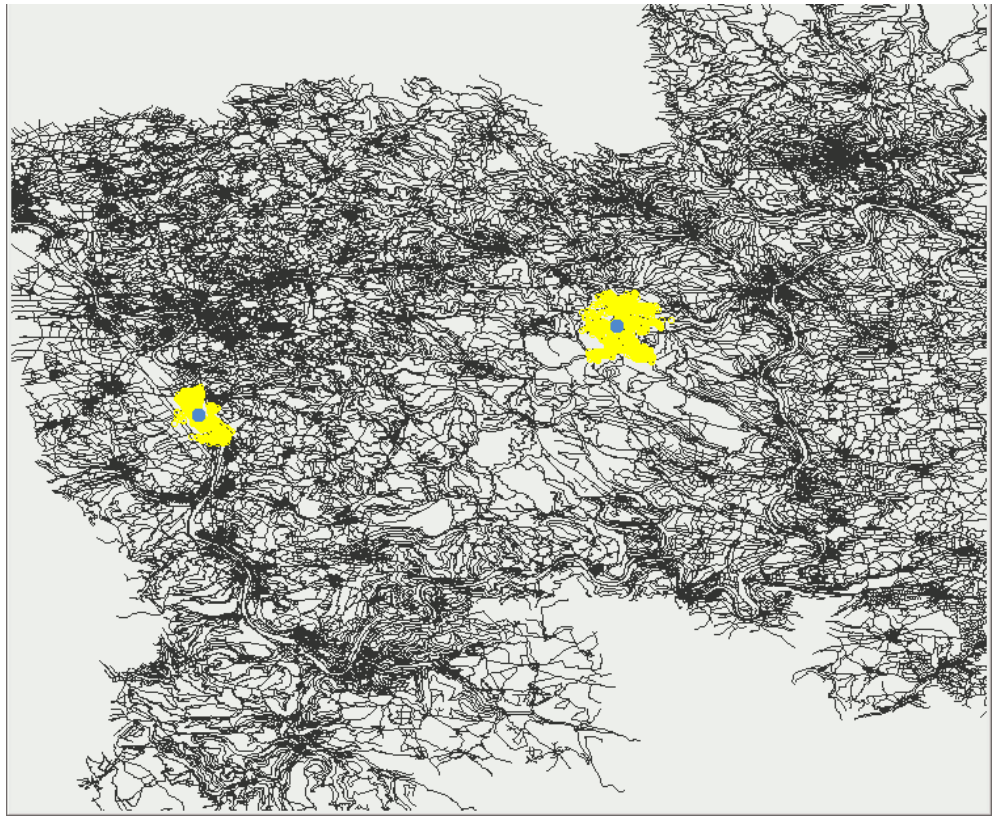


Vertex	Visited	Distance	Previous
A	T	3	B
B	T	1	S
C	T	2	B
D	T	1	S
E	T	3	D
S	T	0	Ø
T	T	$2+2\sqrt{2}$	C

Summary for Dijkstra's algorithm

Single source shortest distance/path for weighted graphs

Compute shortest distance/path for a pair of vertices in practice:



Bidirectional search

A* Search

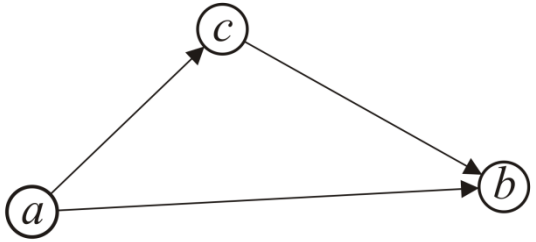
Motivation: Triangle Inequality

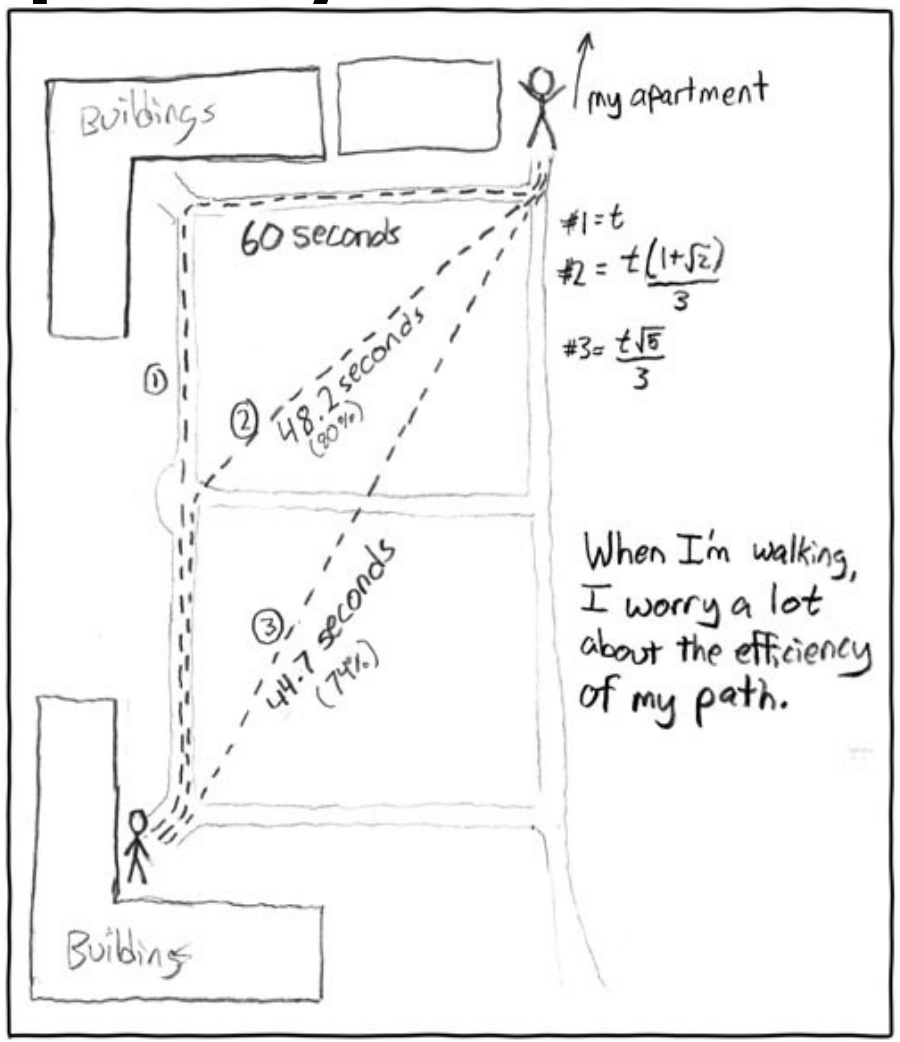
Starting from a specific starting node of a graph, if we aim to find the shortest path to a given goal node.

We can use the **A* search** which is faster than Dijkstra's algorithm, which uses a hypothetical shortest distance to weight the paths.

The requirement is that 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.

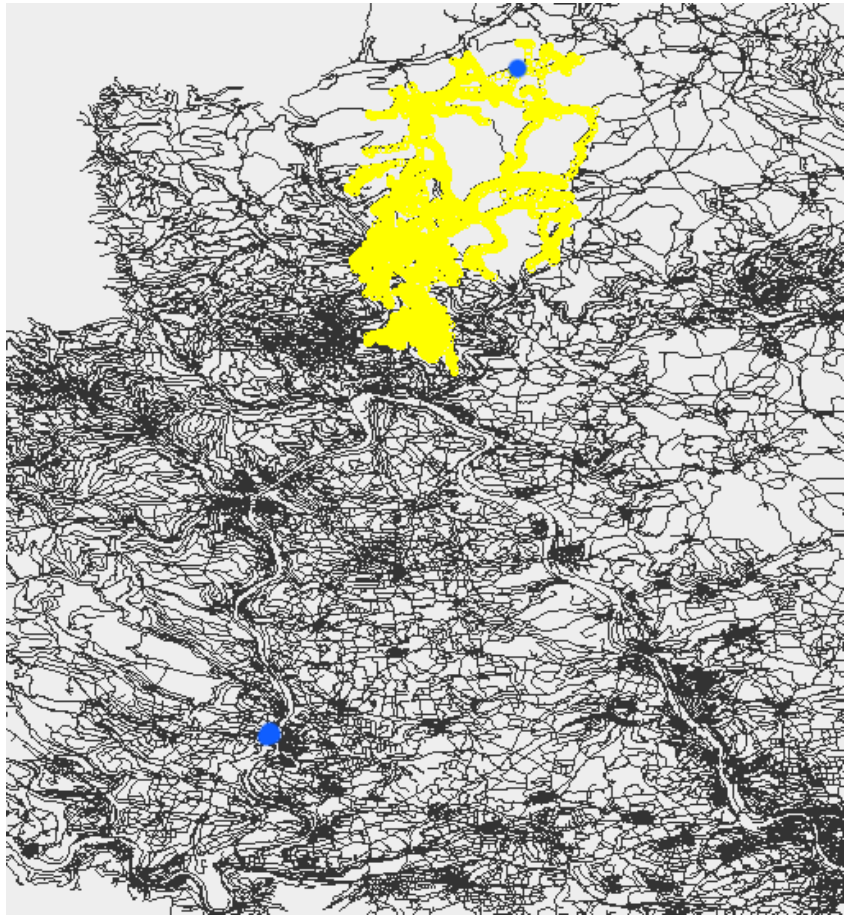
- All Euclidean distances satisfy the triangle inequality





http://xkcd.com/85/

A quick view of A* search



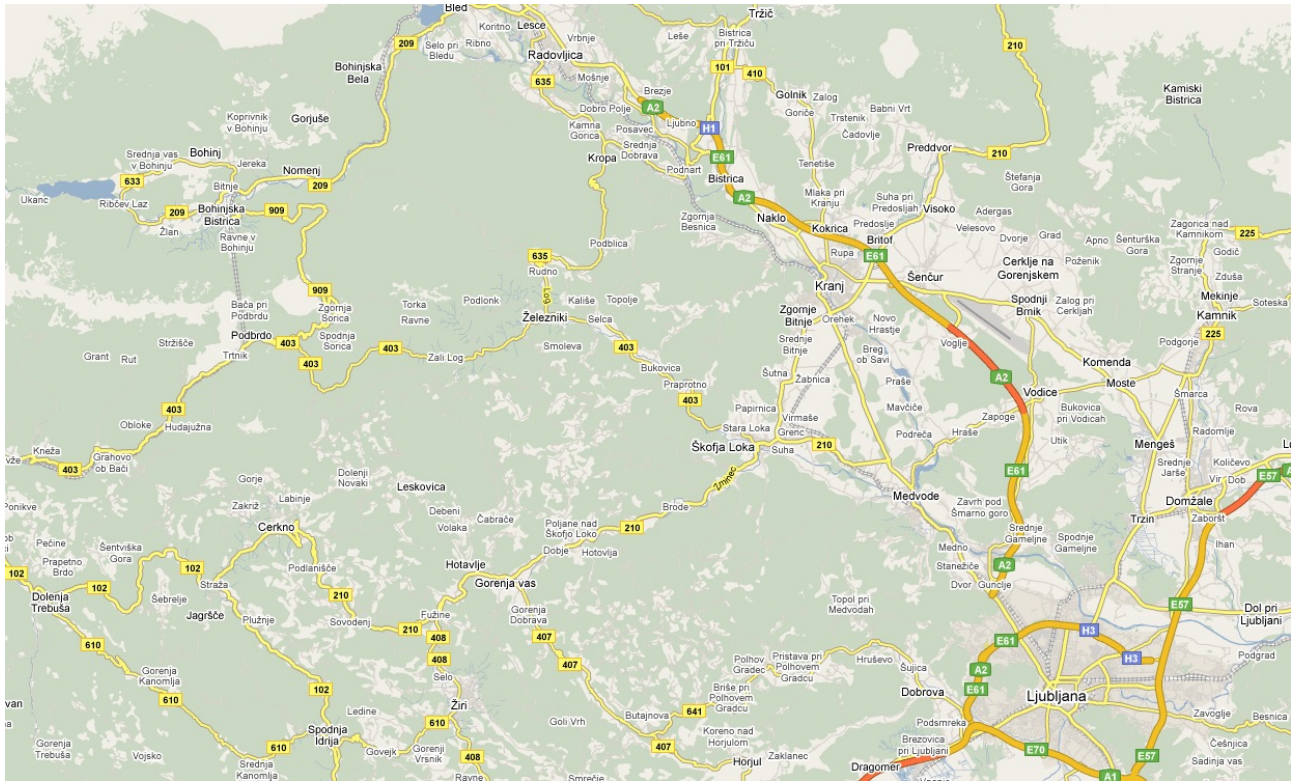
Background

Assume we have a heuristic lower bound for the length of a path between any two vertices.

For instance, for a graph embedded in a plane, the shortest distance is the Euclidean distance. We can use this to guide our search for a path.

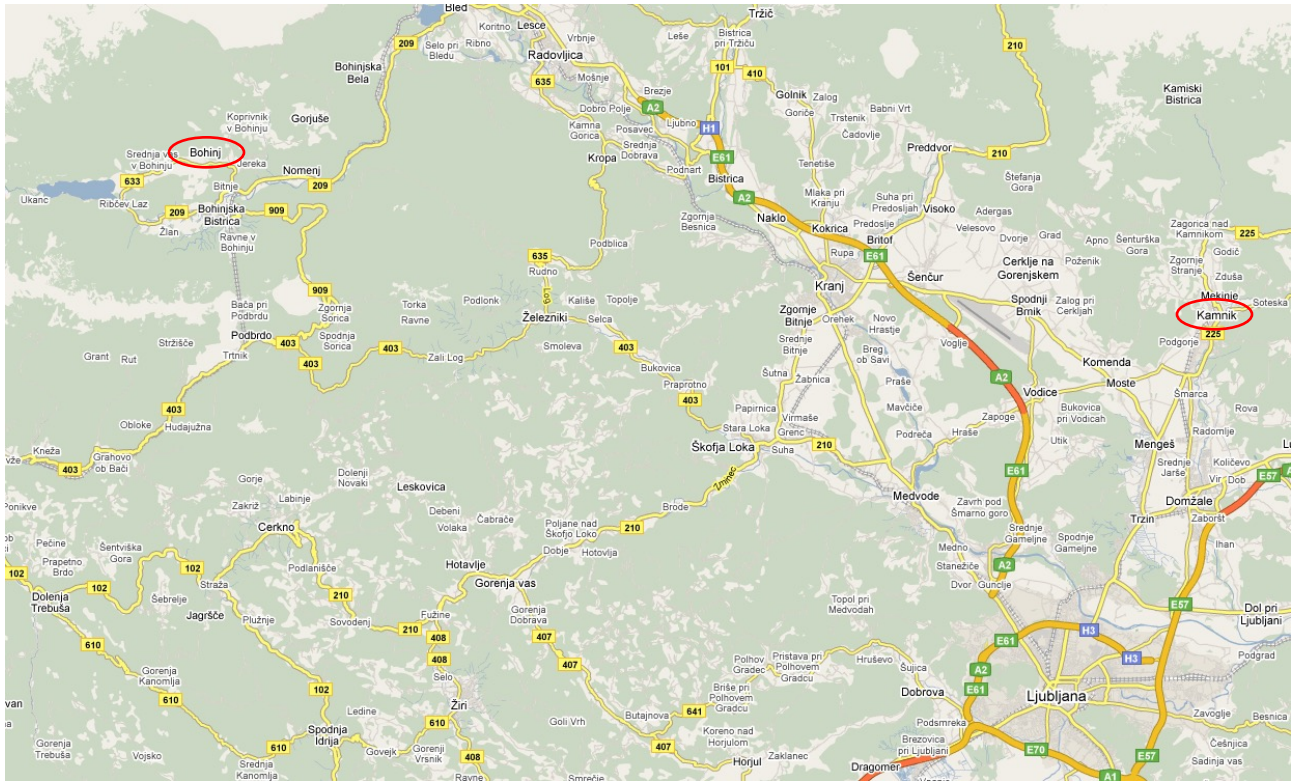
Idea

Consider this map of Slovenia



Idea

Suppose we want to go from Kamnik to Bohinj



Idea

A lower bound for the length of the shortest path to Bohinj is h(Kamnik, Bohinj) = 53 km

Idea

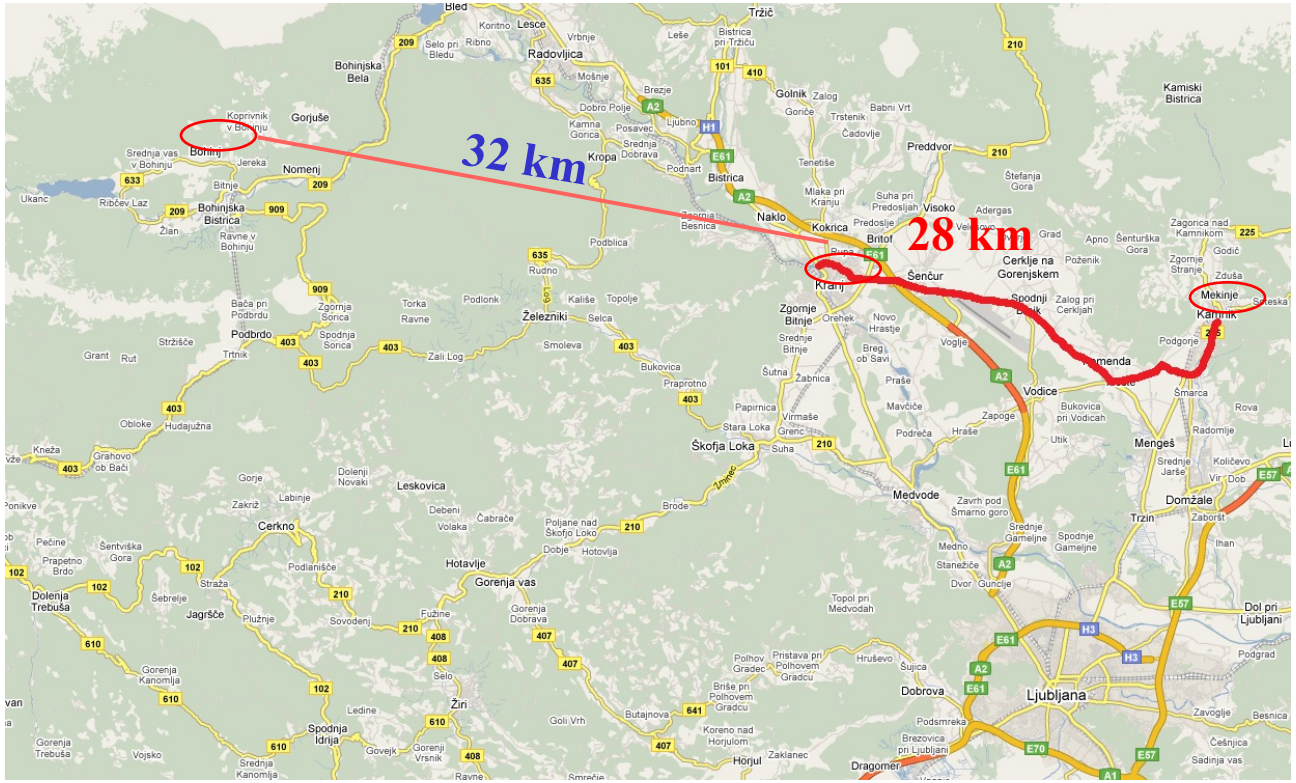
Any actual path must be **at least as long as 53 km**

Idea

Suppose we have a 28 km shortest path from Kamnik to Kranj: d(Kamnik, Kranj) = 28 km

Idea

A lower bound on the shortest distance from Kranj to the destination is now
h(Kranj, Bohinj) = 32 km



Idea

Thus, we weight of the path up to Kranj is

w(Kranj) = d(Kamnik, Kranj) + h(Kranj, Bohinj) = 60 km

Idea

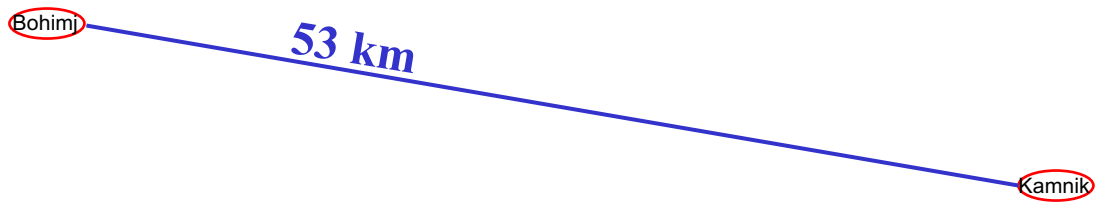
Any path extending this given path to Bohimj must be at least 60 km

Idea

The value $w(\text{Kranj})$ represents the shortest possible distance from Kamnik to Bohinj given that we follow the path to Kranj

As with Dijkstra's algorithm, we must start with the null path starting at Kamnik:

w(Kamnik) = d(Kamnik, Kamnik) + h(Kamnik, Bohimj)
= 0 km + 53 km



Algorithm Description

Suppose we are finding the shortest path from vertex a to a vertex z

The A* search algorithm initially:

- Marks each vertex as unvisited
- Starts with an array containing only the initial vertex
  - The value of any vertex v in the array is the weight w(v) which assumes we have found the a shortest path to v

Algorithm Description

The algorithm then iterates:

- Visit the vertex *u* with the least weight
  - Mark the vertex *u* of the path as visited
- Ignore all visited adjacent vertices *v*
- For each remaining unvisited adjacent vertex *v*:
  - Determine if w(*v*) = d(*a*, *u*) + d(*u*, *v*) + h(*v*, *z*) is less than the current weight of *v*, and if so, update the path leading to *v* and its weight

Continue iterating until the destination vertex *z* is visited

Comparison with Dijkstra's Algorithm

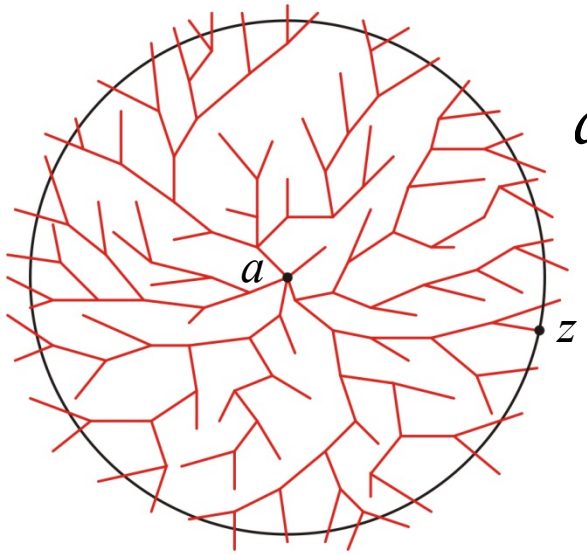
This differs from Dijkstra's algorithm which gives weight only to the known path

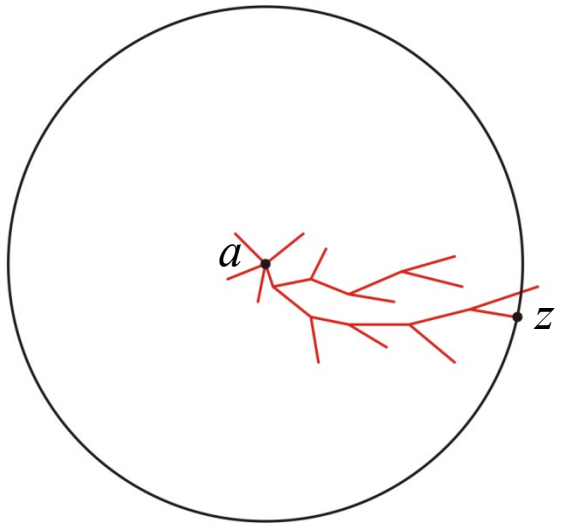
Difference:

- Dijkstra's algorithm radiates out from the initial vertex
- The A* search algorithm directs its search towards the destination

Comparison with Dijkstra's Algorithm

Graphically, we can suggest the behaviour of the two algorithms as follows:

Suppose we 

a to z:


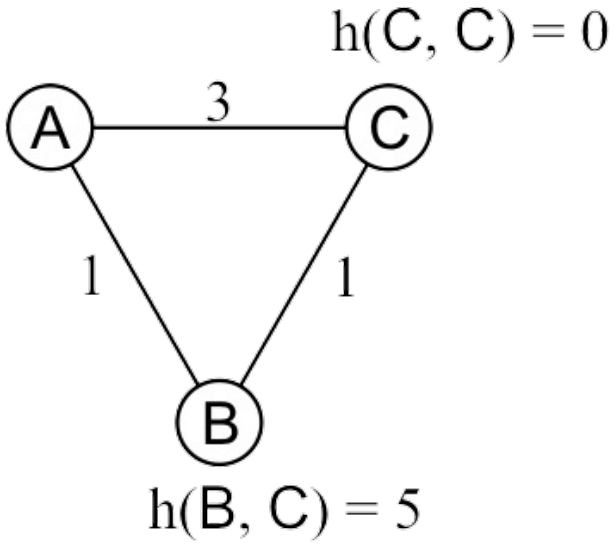
Representative search patterns for Dijkstra's and the A* search algorithms

Optimally Guarantees?

The A* search algorithm will **NOT** always find the optimal path with a poor heuristic distance

- Find the shortest path from A to C:
  - **w(B) = 1 + 5 = 6**
  - **w(C) = 3 + 0 = 3**

- Therefore, C is visited next and as it is the destination, we are finished

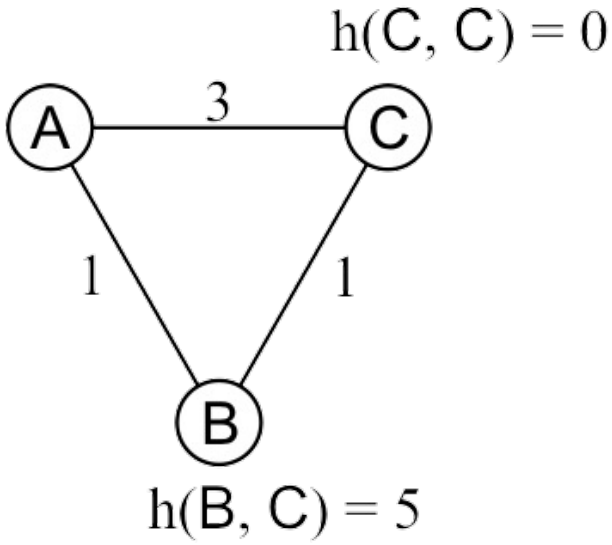


Admissible Heuristics

This heuristic overestimates the actual distance from B to C.

The Euclidean distance doesn't suffer this problem:

The path the crow flies is always shorter than the road the wolf runs.



Admissible Heuristics

Admissible heuristics h:

- Let $\mathrm{d}(u, v)$ represent the actual shortest distance from u to v
- A heuristic $\mathrm{h}(u, v)$ is *admissible* if $\mathrm{h}(u, v) \leq \mathrm{d}(u, v)$
- The heuristic is *optimistic* or a *lower bound* on the distance

When the heuristic is admissible, then it is guaranteed to return the shortest path.

Using the Euclidean distance between two points on a map is clearly an admissible heuristic

- The flight of the crow is shorter than the run of the wolf

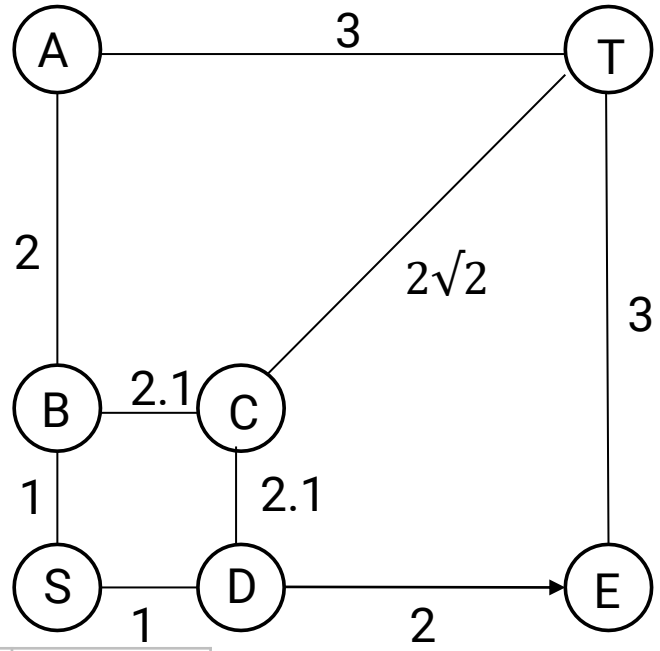
Quick Exercise

Given the graph, apply A* Algorithm to find the shortest path from S to T. Use Euclidean distance as the heuristic.

$$d(u, v) = \sqrt{(u_x - v_x)^2 + (u_y - u_y)^2}$$

The coordinates of the vertices:

A = (0, 3)
B = (0, 1)
C = (1, 1)
D = (1, 0)
E = (3, 0)
S = (0, 0)
T = (3, 3)

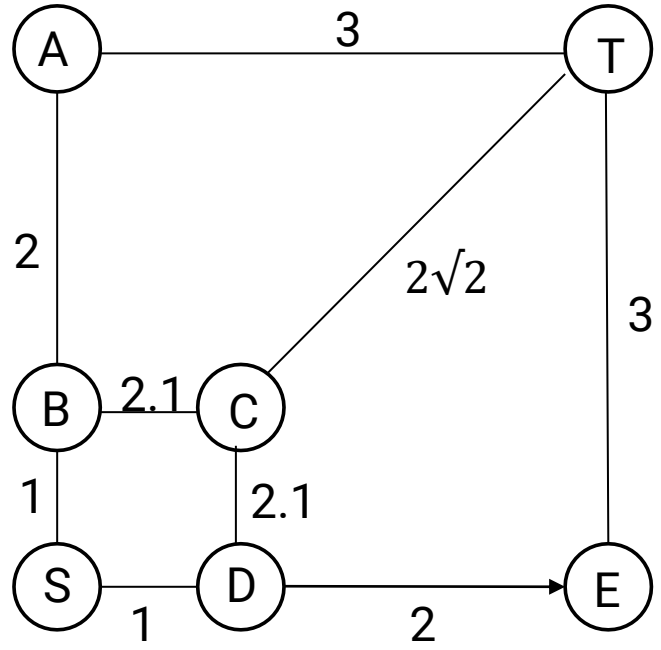


Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	∞	3	∞	Ø
B	F	∞	$\sqrt{13}$	∞	Ø
C	F	∞	$2\sqrt{2}$	∞	Ø
D	F	∞	$\sqrt{13}$	∞	Ø
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø

Quick Exercise

Since S has the minimum weight, we update the distances of its neighbors B and D.

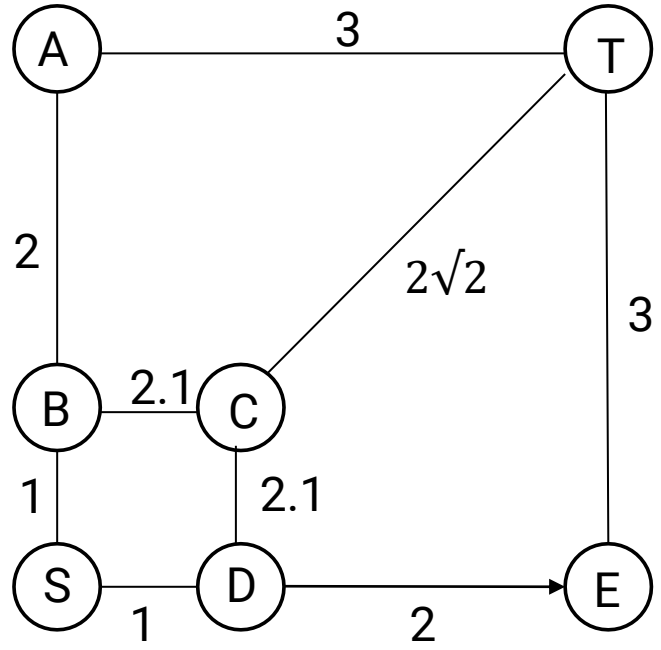
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	∞	3	∞	Ø
B	F	∞	$\sqrt{13}$	∞	Ø
C	F	∞	$2\sqrt{2}$	∞	Ø
D	F	∞	$\sqrt{13}$	∞	Ø
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

The distances of B and D are updated. They now have the smallest weights (total).
Next, we visit B.

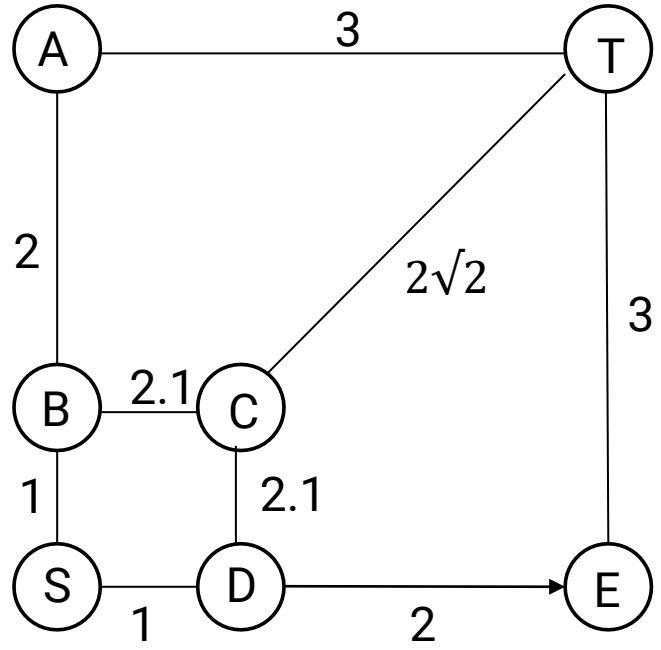
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	∞	3	∞	Ø
B	F	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	F	∞	$2\sqrt{2}$	∞	Ø
D	F	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

Mark B as visited.
Next, we update the distances of A and C.

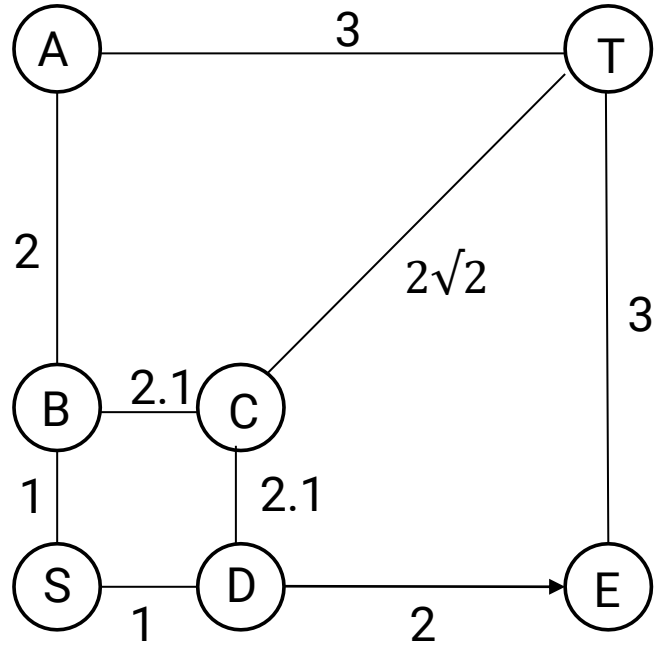
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	∞	3	∞	Ø
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	F	∞	$2\sqrt{2}$	∞	Ø
D	F	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

After updating the distances and the weights, D has the minimum weight and will be visited next.

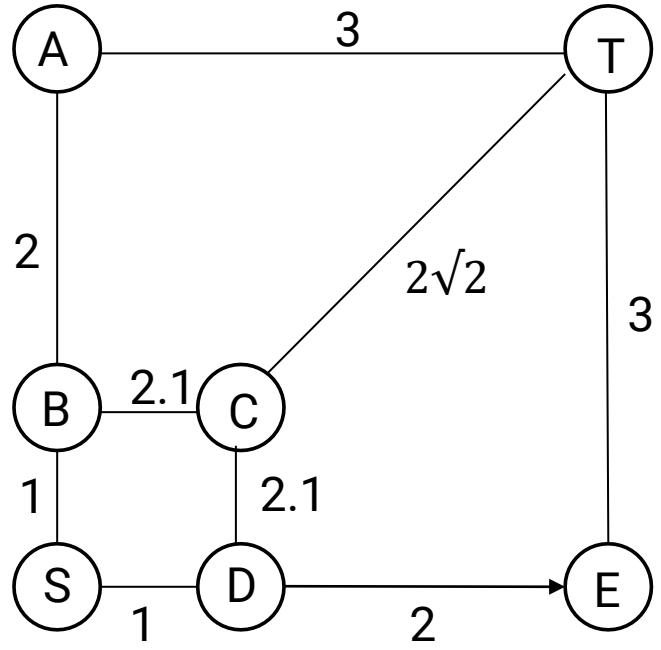
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	F	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	F	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

Mark D as visited.

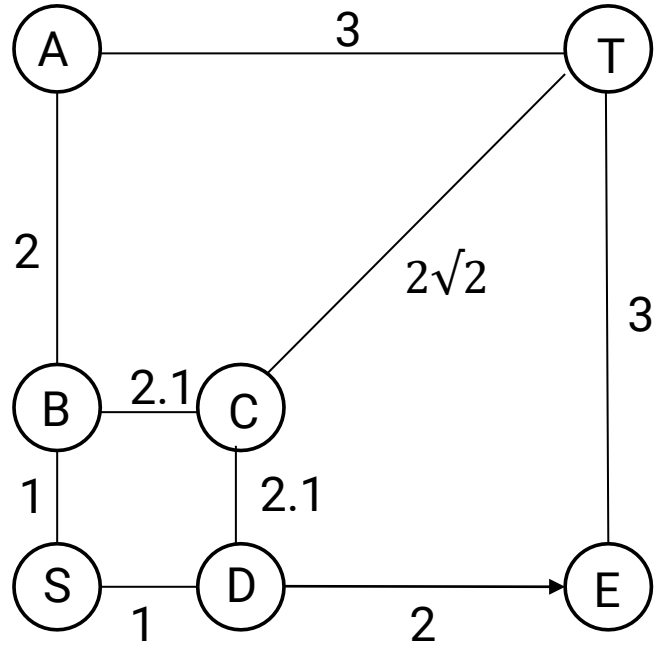
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	F	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	∞	3	∞	Ø
S	T	0	$2\sqrt{3}$	$2\sqrt{3}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

Update the distances of E.

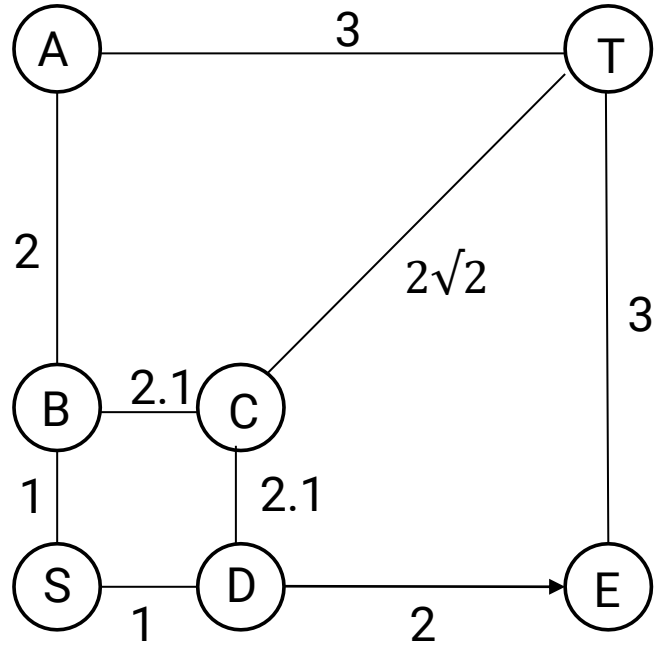
Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	F	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	3	3	6	D
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø



Quick Exercise

Mark C as visited.
Update the distances of C's neighbor: T.

Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	T	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	3	3	6	D
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø

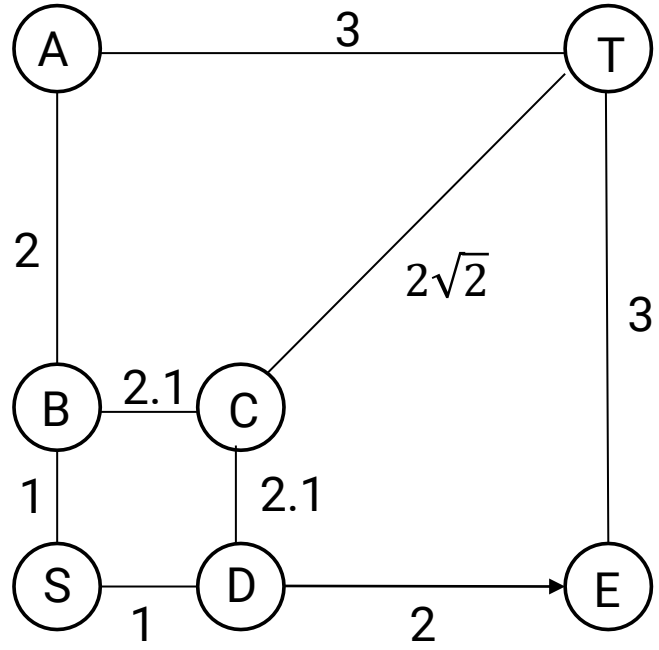


Quick Exercise

Note that in this exercise, the vertices A and E remain unvisited.
This is because they are pruned by the heuristic function.
The weight suggests that vertex C is guaranteed to be a better option to explore.

Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	T	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	3	3	6	D
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	T	$3.1+2\sqrt{2}$	0	$3.1+2\sqrt{2}$	C

The target vertex is now found.



All-pairs Shortest Path

All-pairs Shortest Path

This topic:

- We will look at the Floyd-Warshall algorithm for:
  - Finding all shortest distances (the shortest distance between all pairs of nodes)
  - Finding the paths corresponding to these distances
- We conclude by finding the transitive closure

Problem

Observation: Any algorithm that finds the shortest path between all pairs must consider, each pair of vertices; therefore, a lower bound on the execution would be $O(|V|^2)$.

We will look at the Floyd-Warshall algorithm.

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

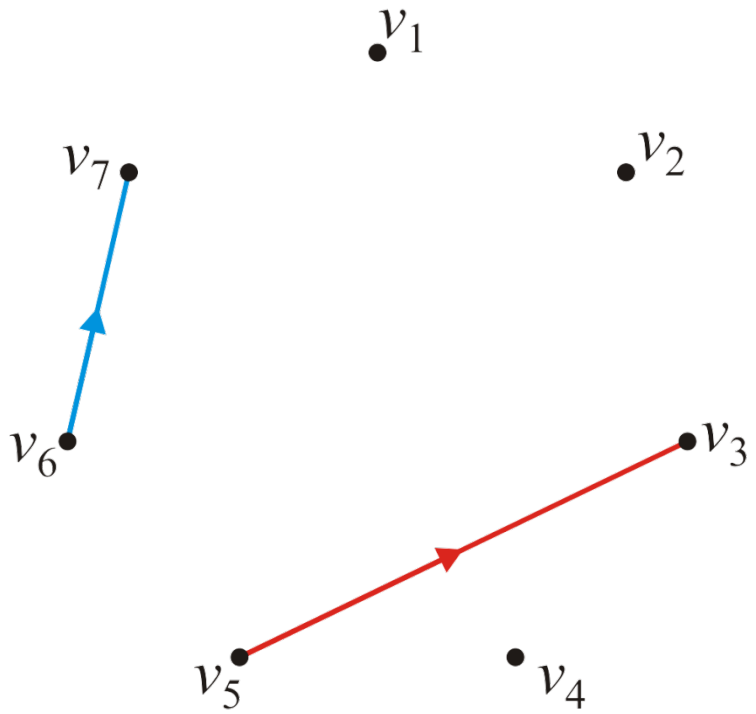
In C++, we would define a two-dimensional array

```
double d[num_vertices][num_vertices];
```

Strategy

Consider this graph of seven vertices

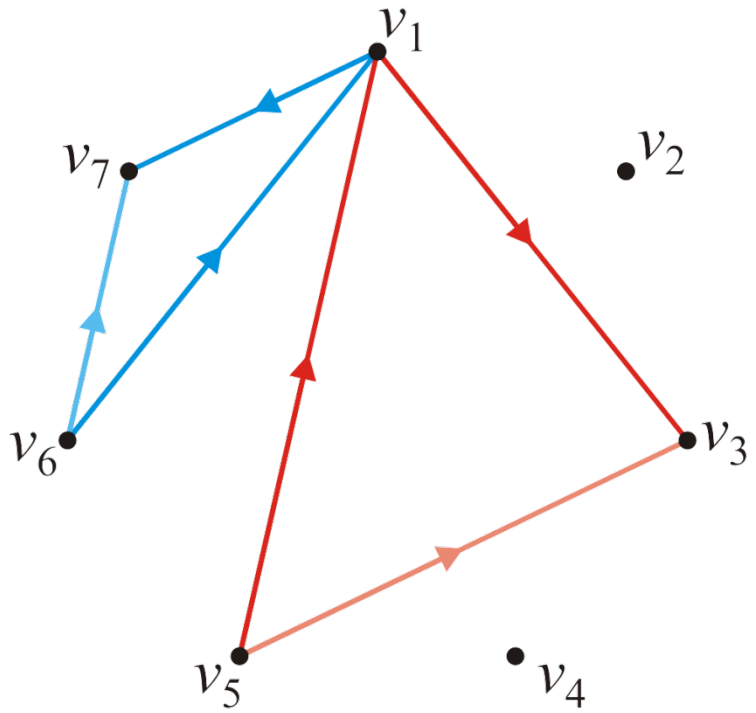
The edges defining the values $d_{6,7}^{(0)}$ and $d_{5,3}^{(0)}$ are highlighted



Strategy

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

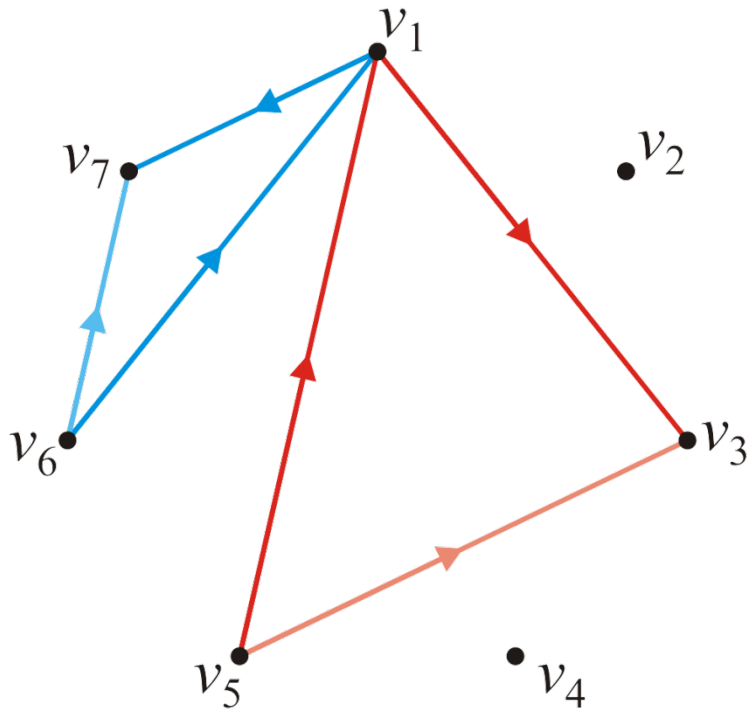
- Is $d^{(0)}_{5,3} > d^{(0)}_{5,1} + d^{(0)}_{1,3}$?
- Is $d^{(0)}_{6,7} > d^{(0)}_{6,1} + d^{(0)}_{1,7}$?



Strategy

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

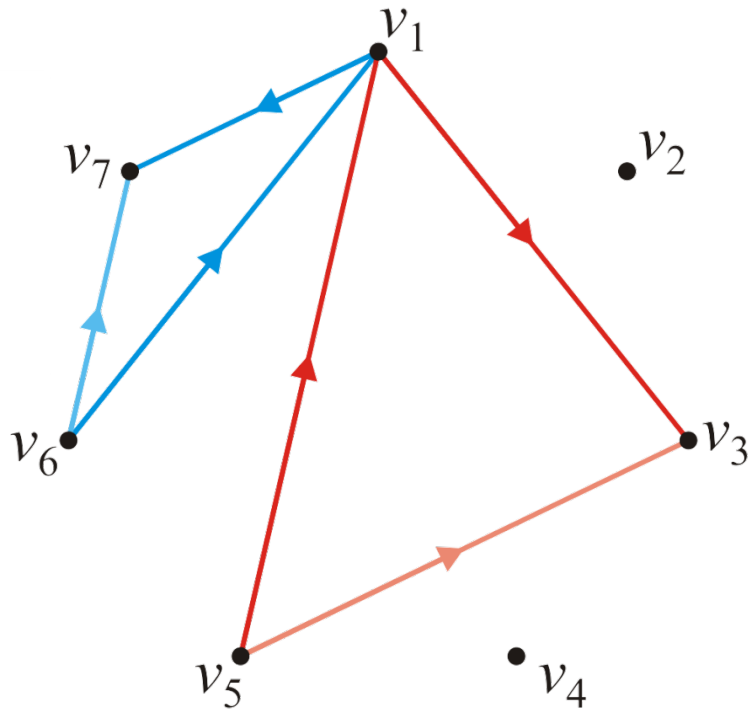
$$d_{i,j}^{(1)} = \min\left\{d_{i,j}^{(0)}, d_{i,1}^{(0)} + d_{1,j}^{(0)}\right\}$$



Strategy

Note that $d^{(1)}_{1,j} = d^{(0)}_{1,j}$ and $d^{(1)}_{i,1} = d^{(0)}_{i,1}$; thus, we need just run the algorithm for each pair of vertices:

```
for ( int i = 1; i <= n; ++i ) {
    for ( int j = 1; j <= num_n; ++j ) {
        d[i][j] = min( d[i][j], d[i][1] + d[1][j] );
    }
}
```

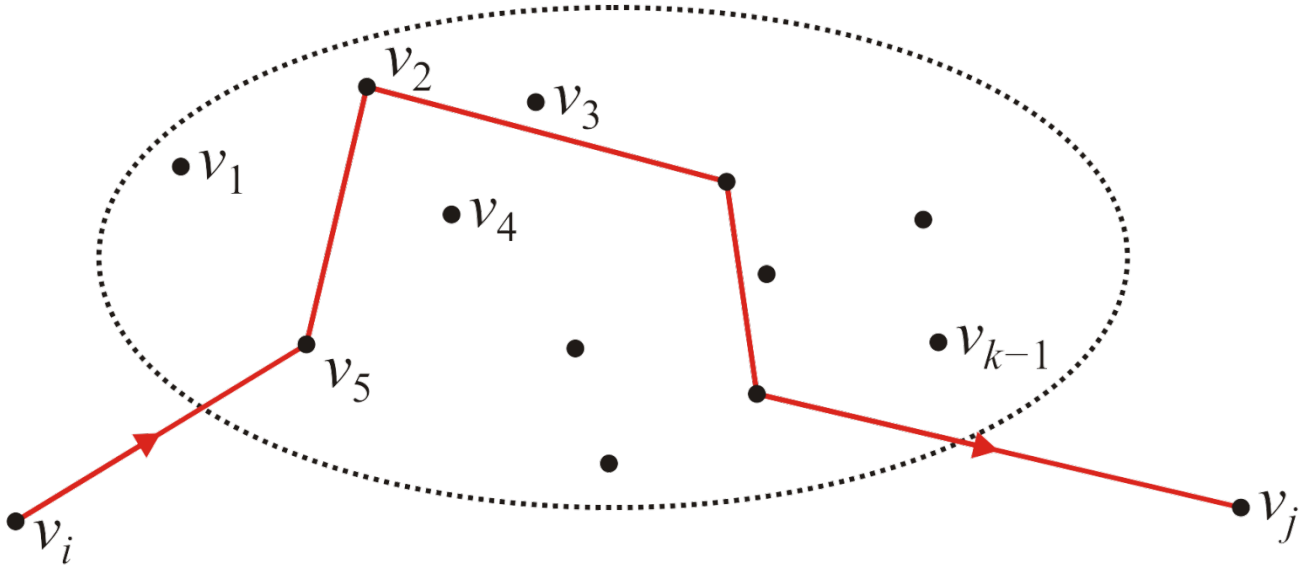


The General Step

Optional

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

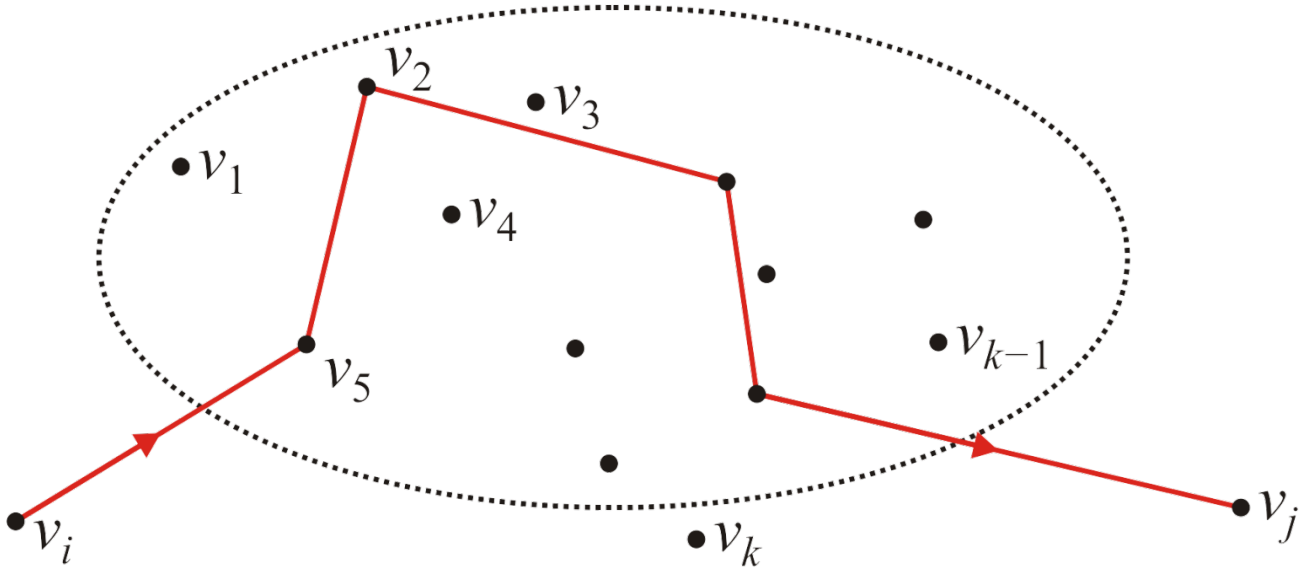
- Suppose we have an algorithm that has found these values for all pairs



The General Step

Optional

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 ?



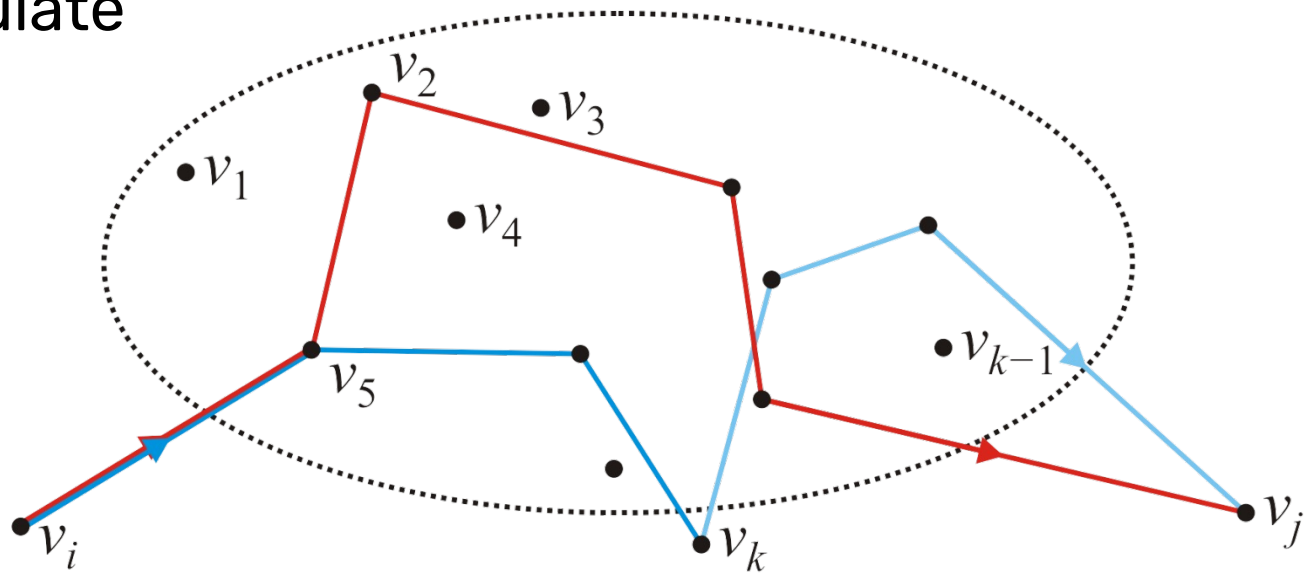
The General Step

Optional

With v_1, v_2, …, v_{k-1} as intermediates, have assumed we have found the shortest paths from v_i to v_j, v_i to v_k and v_k to v_j. The only possible shorter path including v_k would be the path from v_i to v_k continuing from there to v_j

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

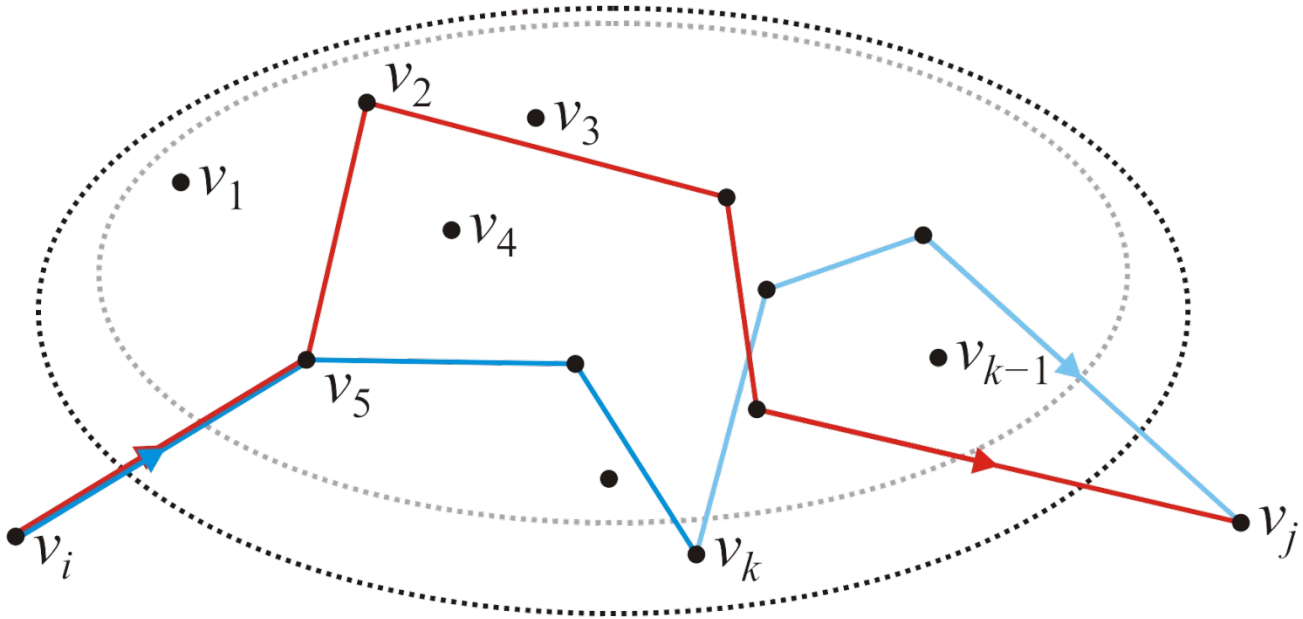
Thus, we calculate



The General Step

Optional

Finding this 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 Floyd-Warshall Algorithm

Thus, we have found the Floyd-Warshall algorithm:

```
double d[num_vertices][num_vertices];

// Initialize the matrix d:  Theta(|V|^2)
//   ...

// Run Floyd-Warshall
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] = min( d[i][j], d[i][k] + d[k][j] );
        }
    }
}
```

Run time? $O(|V|^3)$

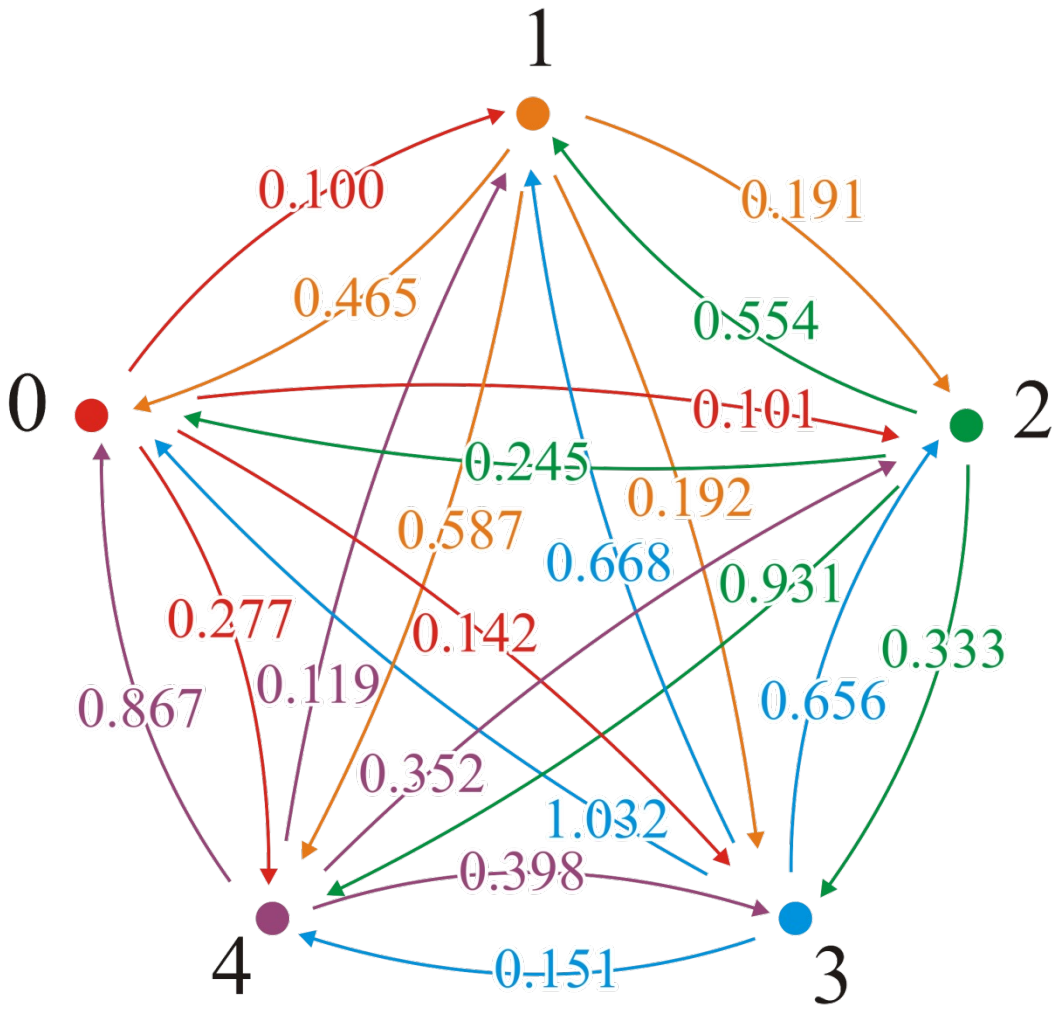
Example

Consider this graph

The adjacency matrix is

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

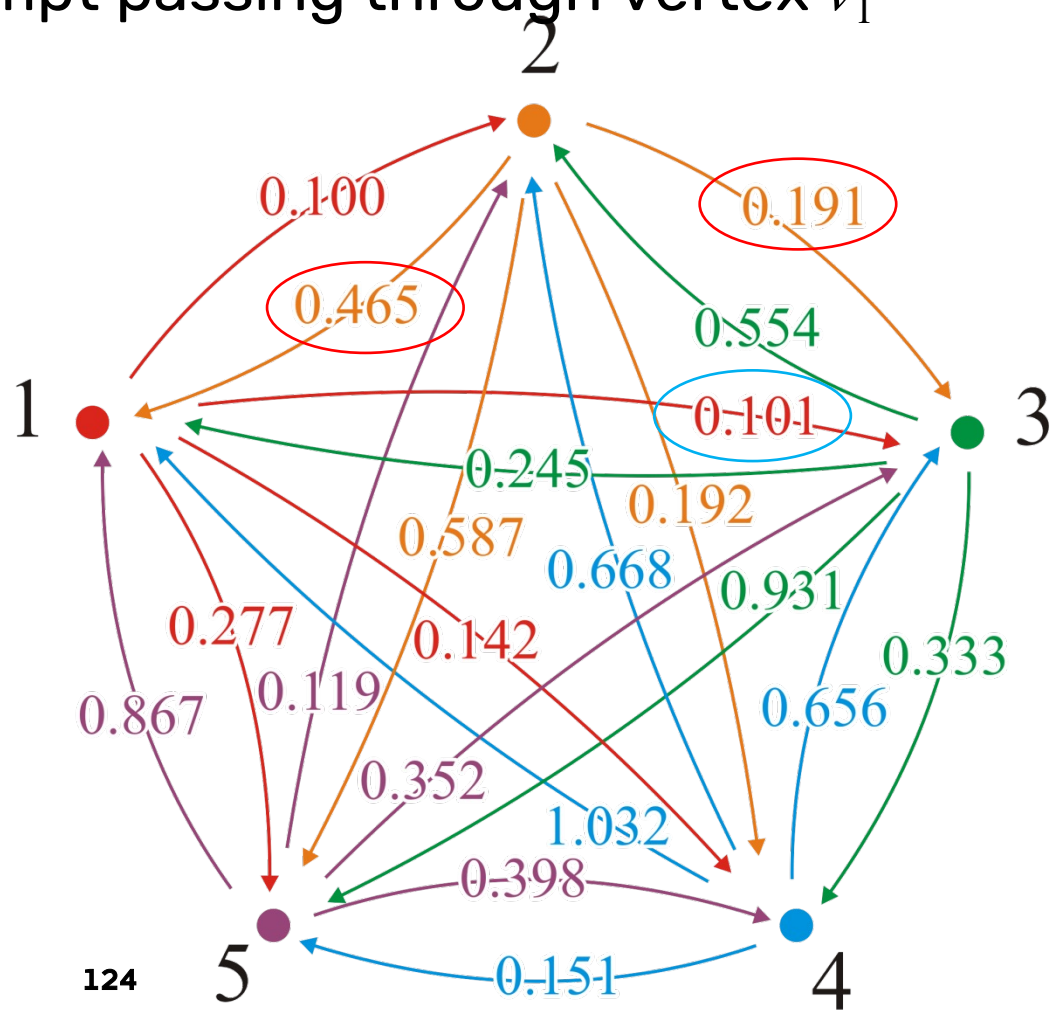
This would define our matrix $\mathbf{D} = (d_{ij})$



Example

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

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



We would start:

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

$$0.191 \not> 0.465 + 0.101$$

Example

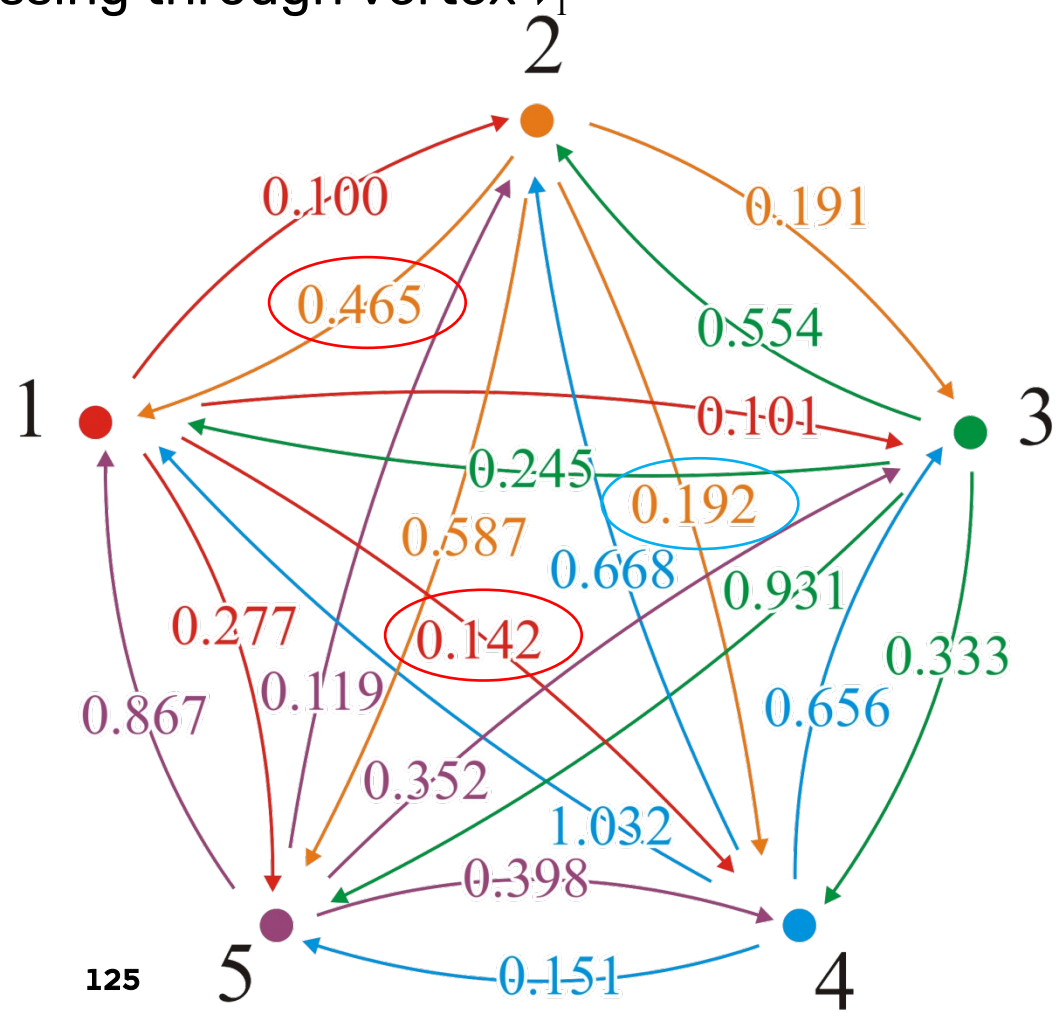
With the first pass, $k = 1$, we attempt passing through vertex v_1

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

We would start:

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

$$0.192 \not> 0.465 + 0.142$$



Example

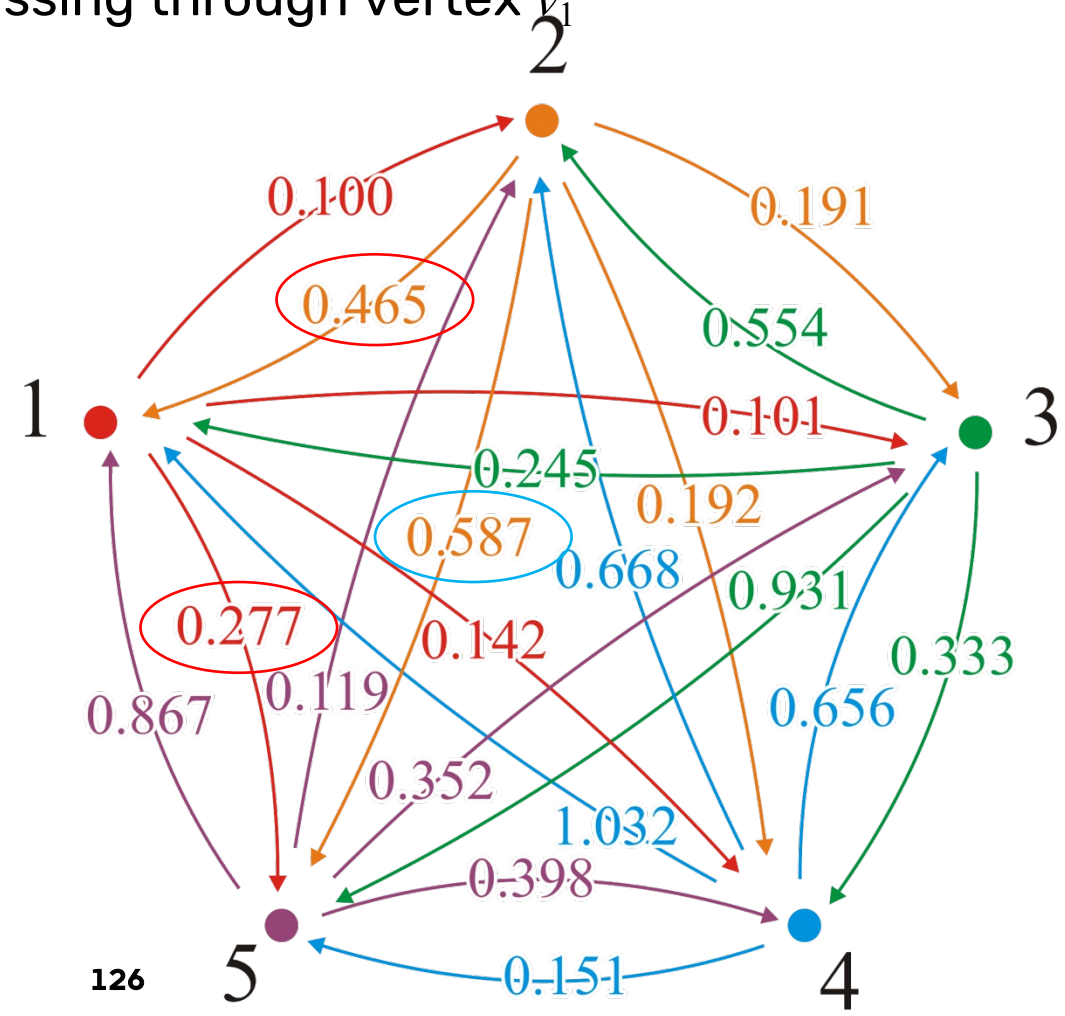
With the first pass, $k = 1$, we attempt passing through vertex v_1

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

We would start:

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

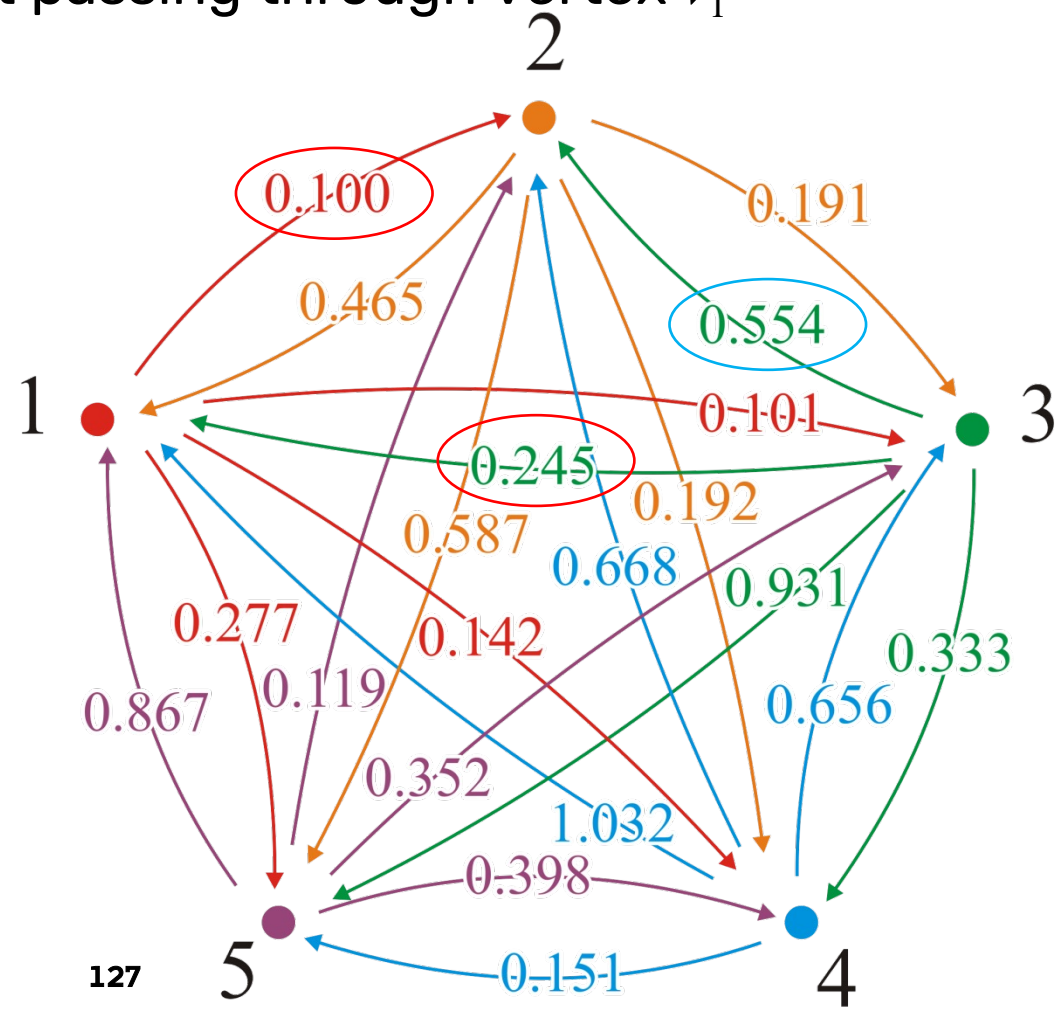
$0.587 \not> 0.465 + 0.277$



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:

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

$0.554 > 0.245 + 0.100 = 0.345$

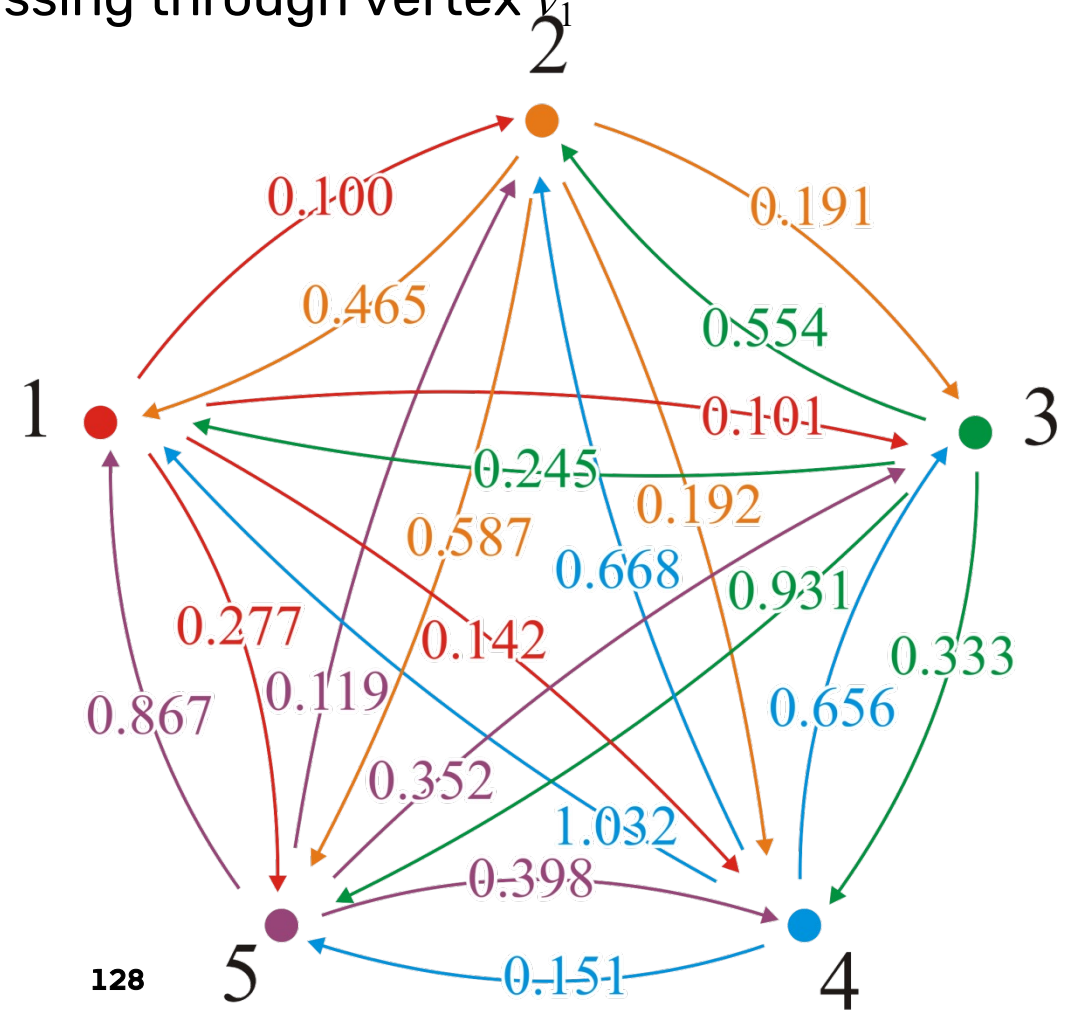
Example

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

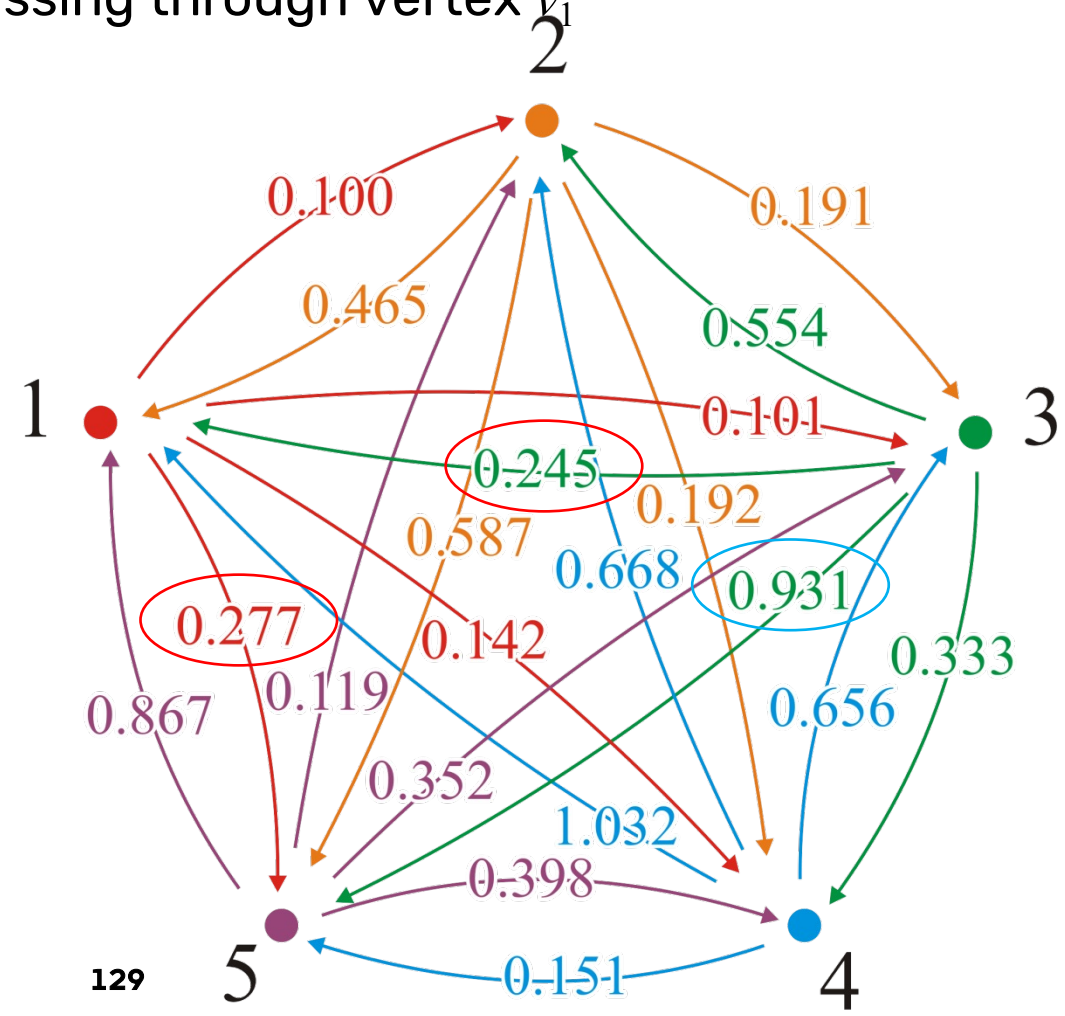
$0.554 > 0.245 + 0.100 = 0.345$



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:

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

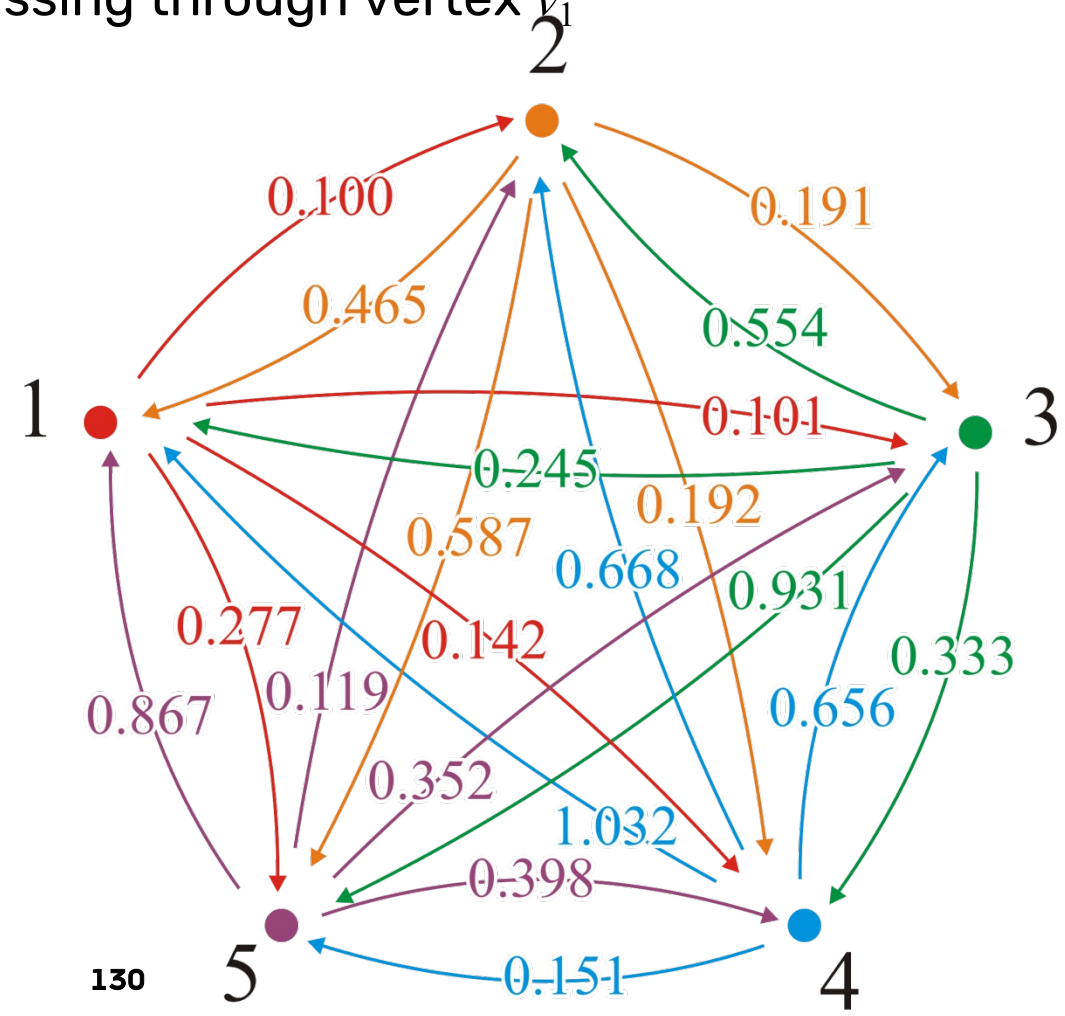
$0.931 > 0.245 + 0.277 = 0.522$

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

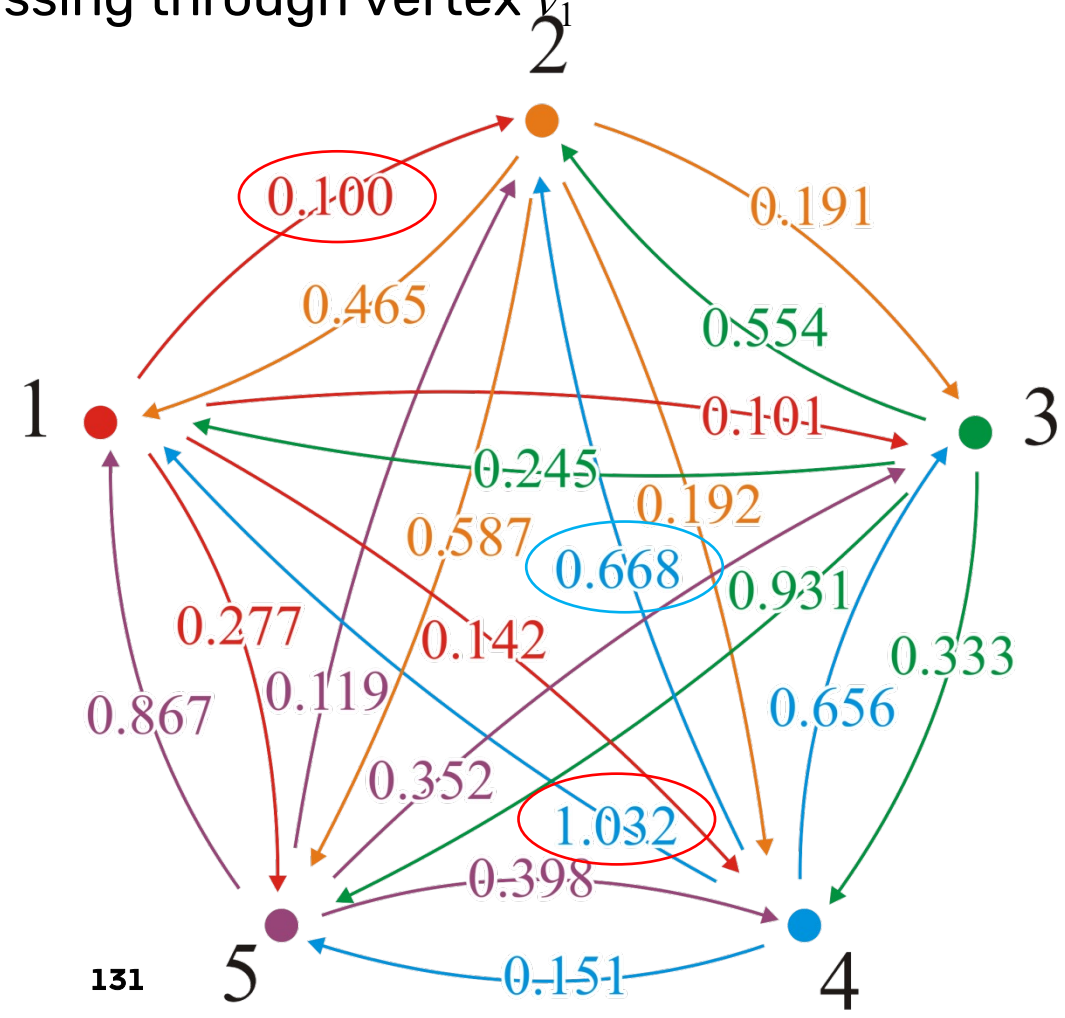
$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Continuing:

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

$0.668 \not> 1.032 + 0.100$

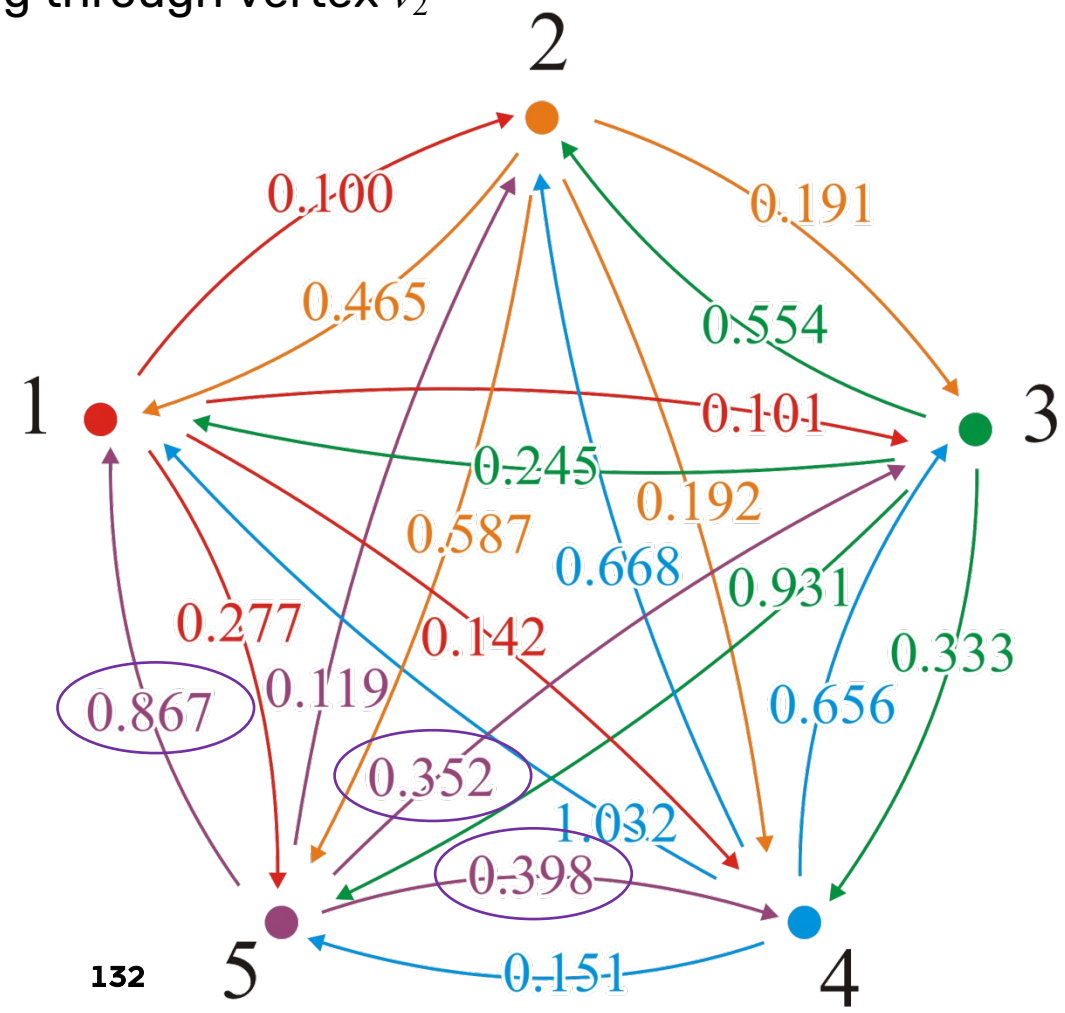
In fact, no other shorter paths through vertex v_1 exist



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:

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

$$0.867 > 0.119 + 0.465 = 0.584$$

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

$$0.352 > 0.119 + 0.191 = 0.310$$

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

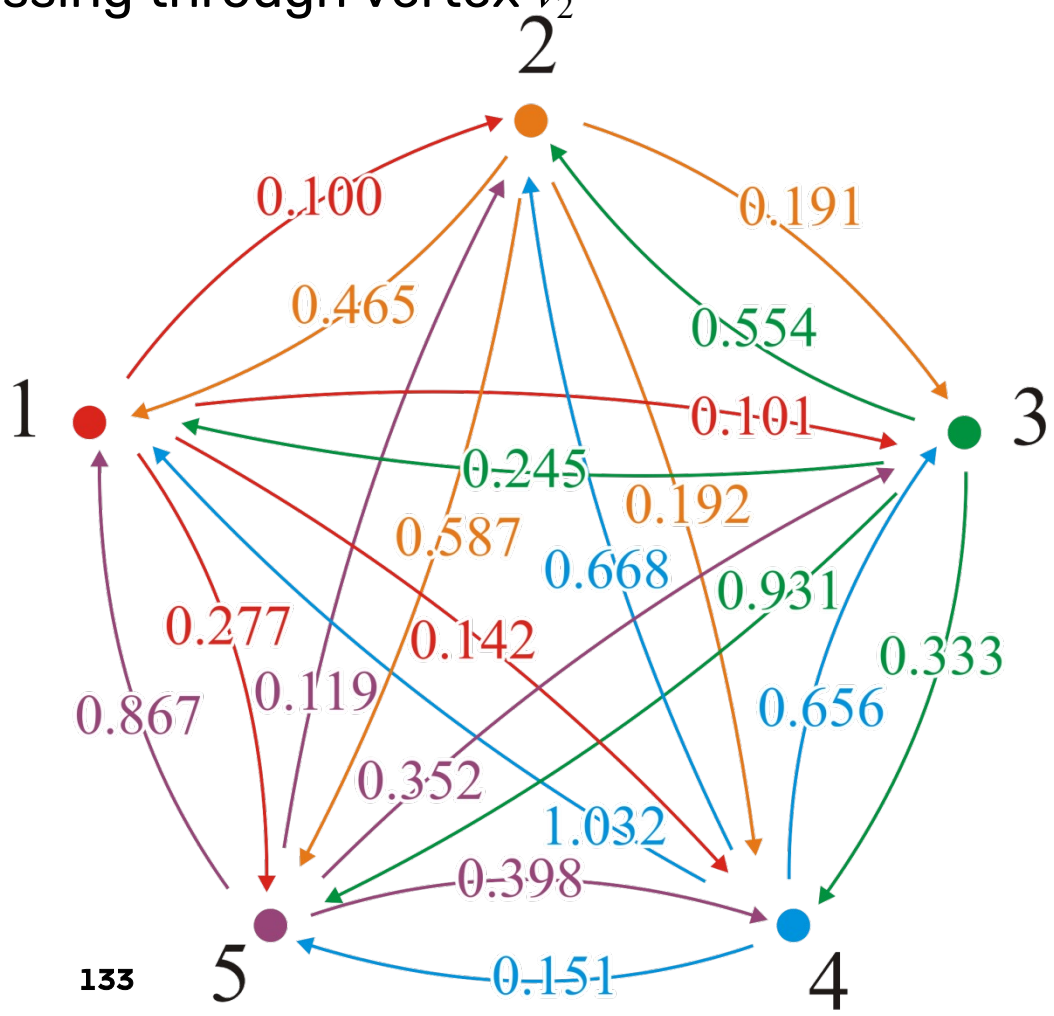
$$0.398 > 0.119 + 0.192 = 0.311$$

Example

With the next pass, $k = 2$, we attempt passing through vertex v_2

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

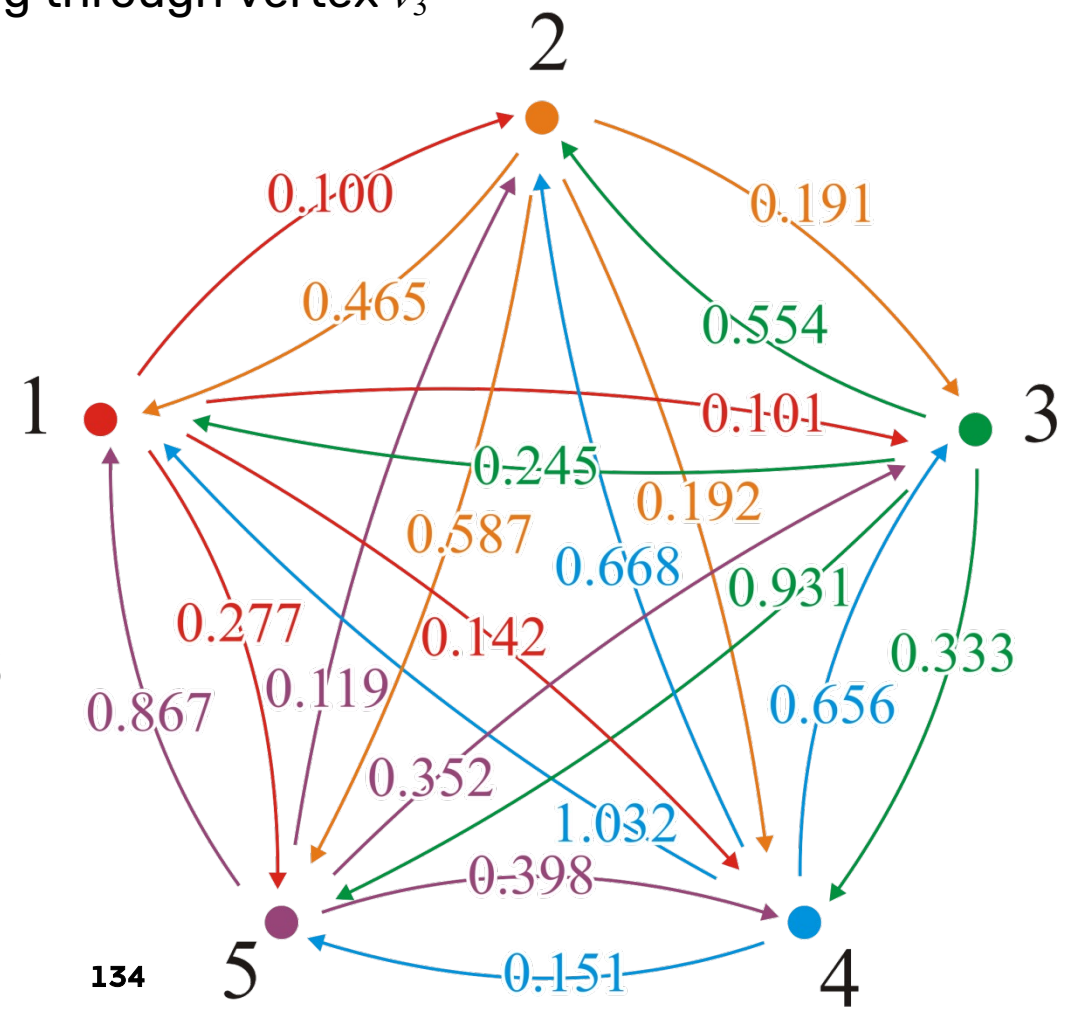
We update the table



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:

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

$$0.465 > 0.191 + 0.245 = 0.436$$

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

$$1.032 > 0.656 + 0.245 = 0.901$$

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

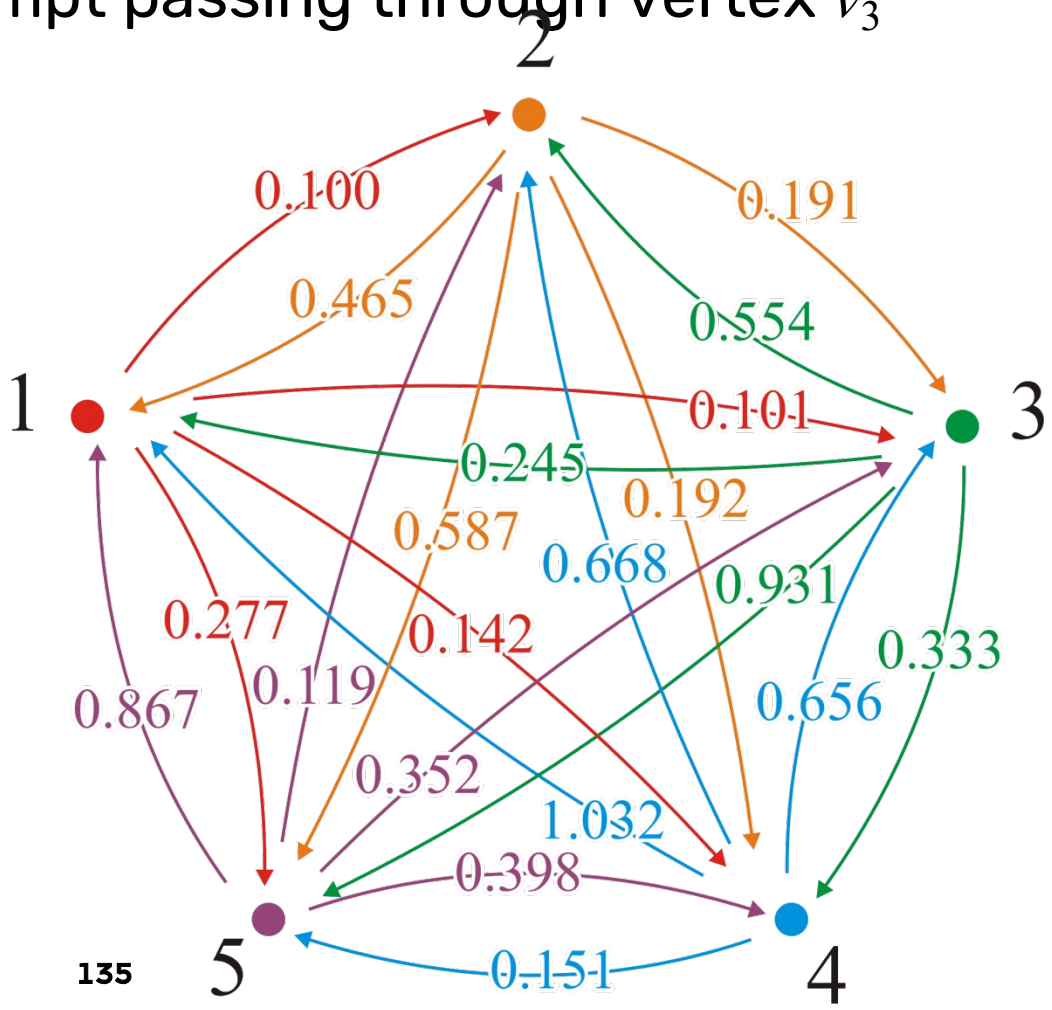
$$0.584 > 0.310 + 0.245 = 0.555$$

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

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

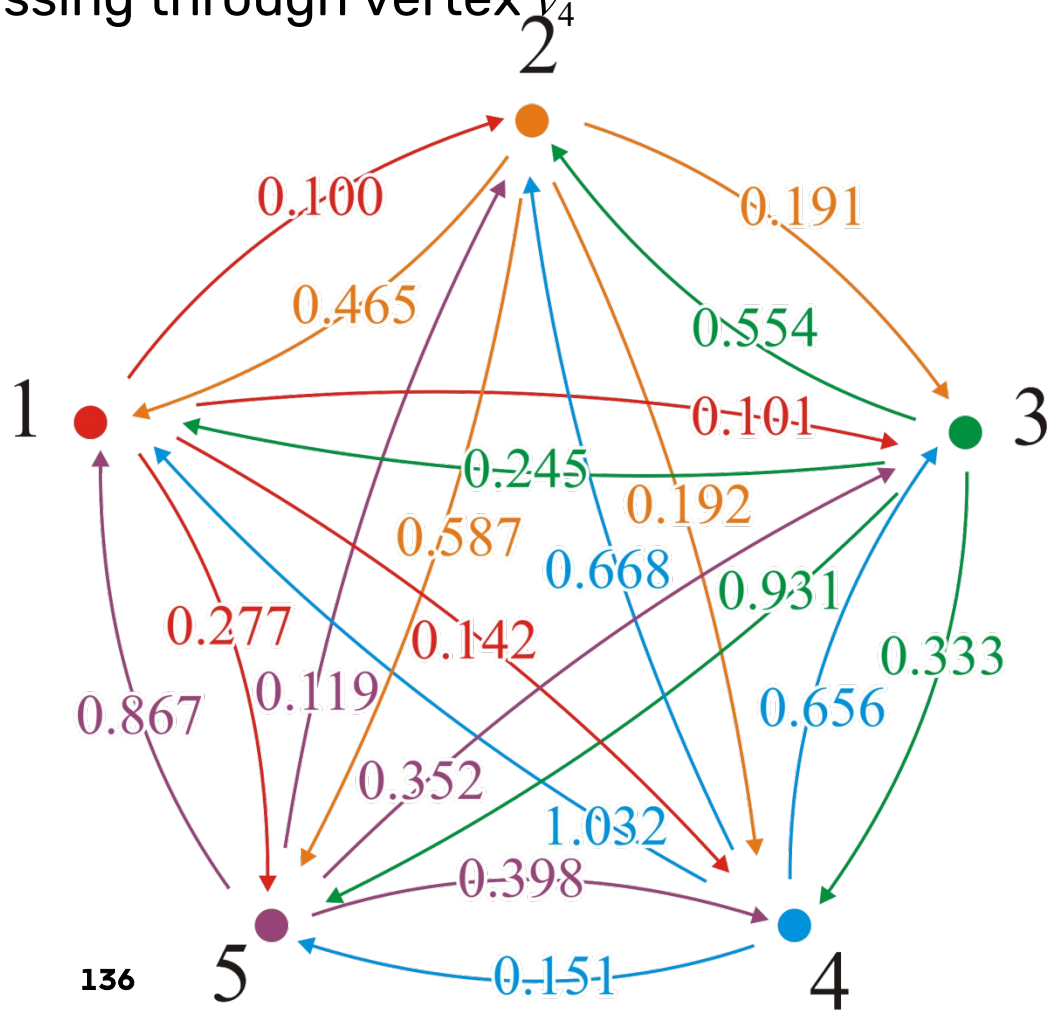
There are two shorter paths:

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

$$0.587 > 0.192 + 0.151$$

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

$$0.522 > 0.333 + 0.151$$

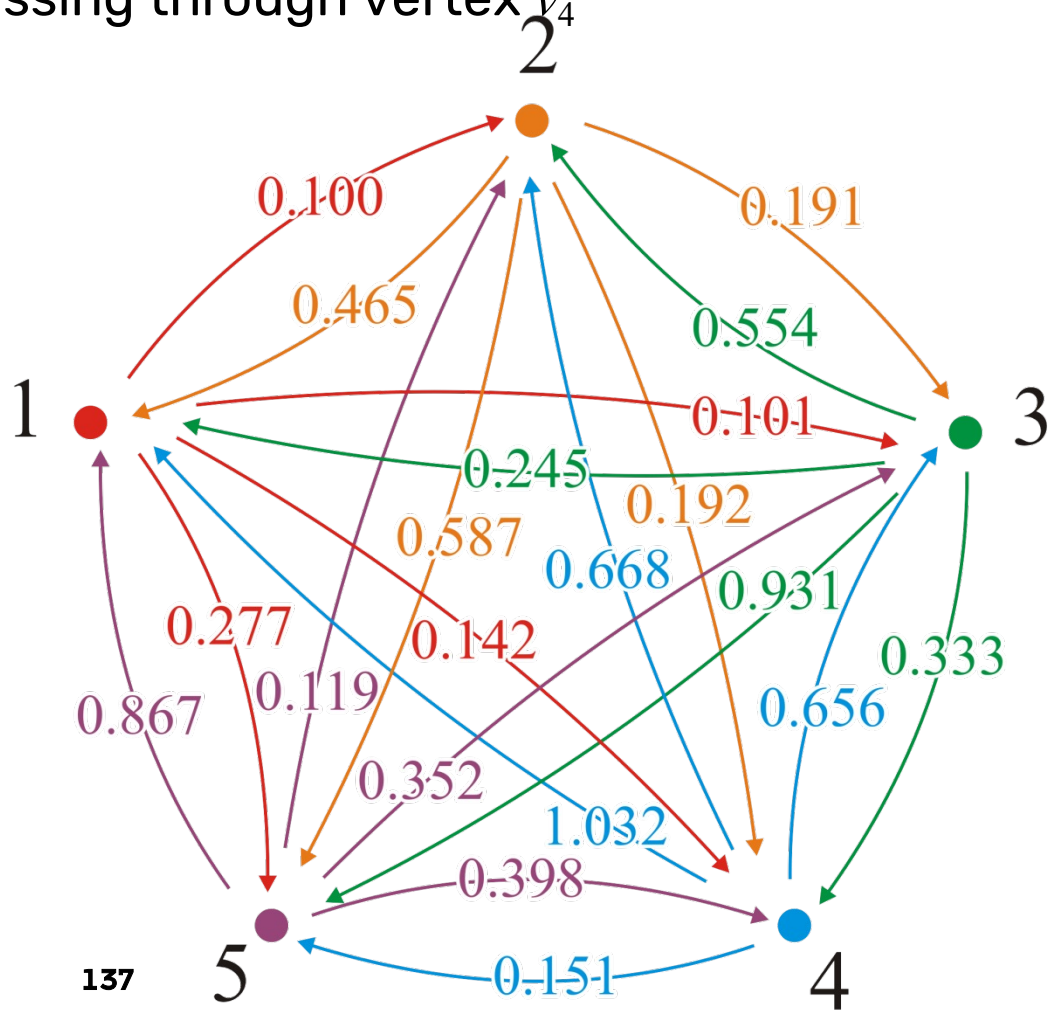


Example

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:

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

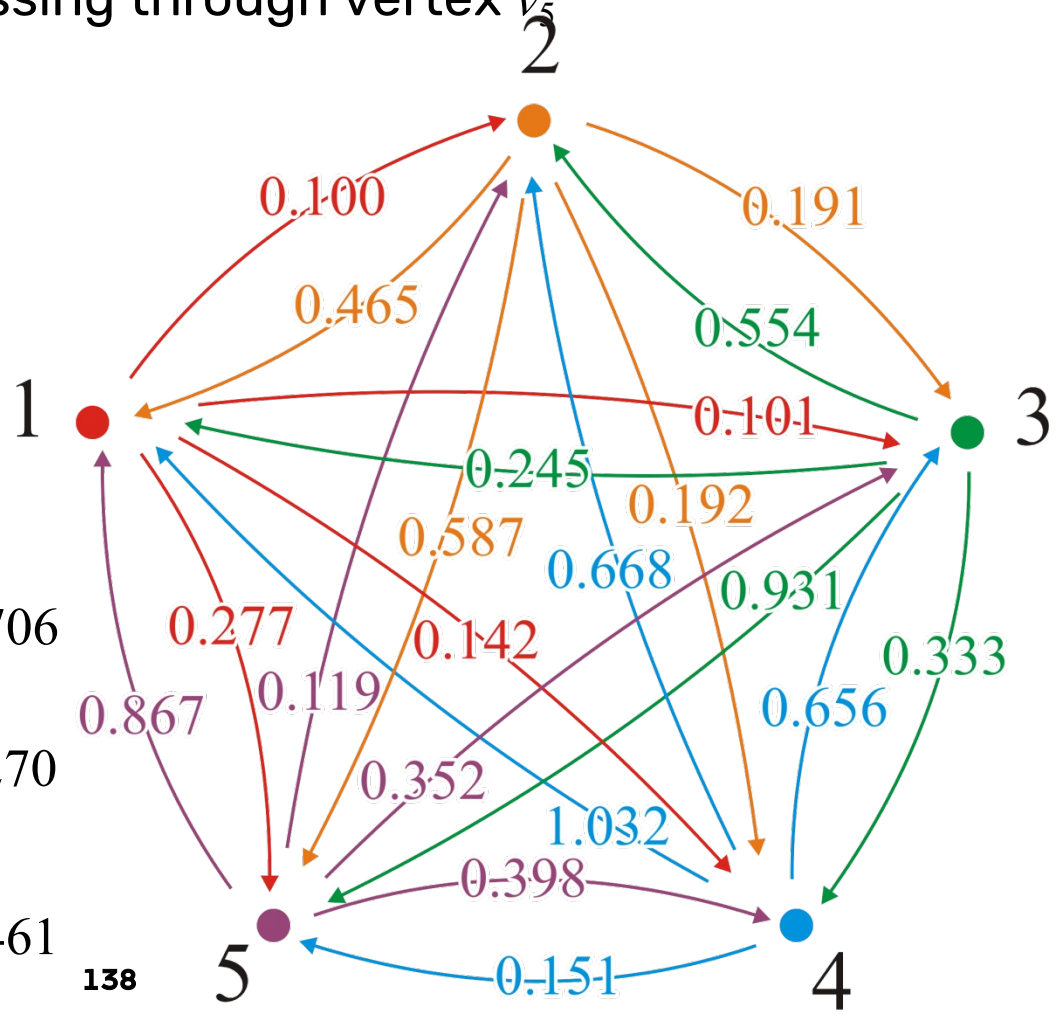
$$0.901 > 0.151 + 0.555 = 0.706$$

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

$$0.668 > 0.151 + 0.119 = 0.270$$

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

$$0.656 > 0.151 + 0.310 = 0.461$$

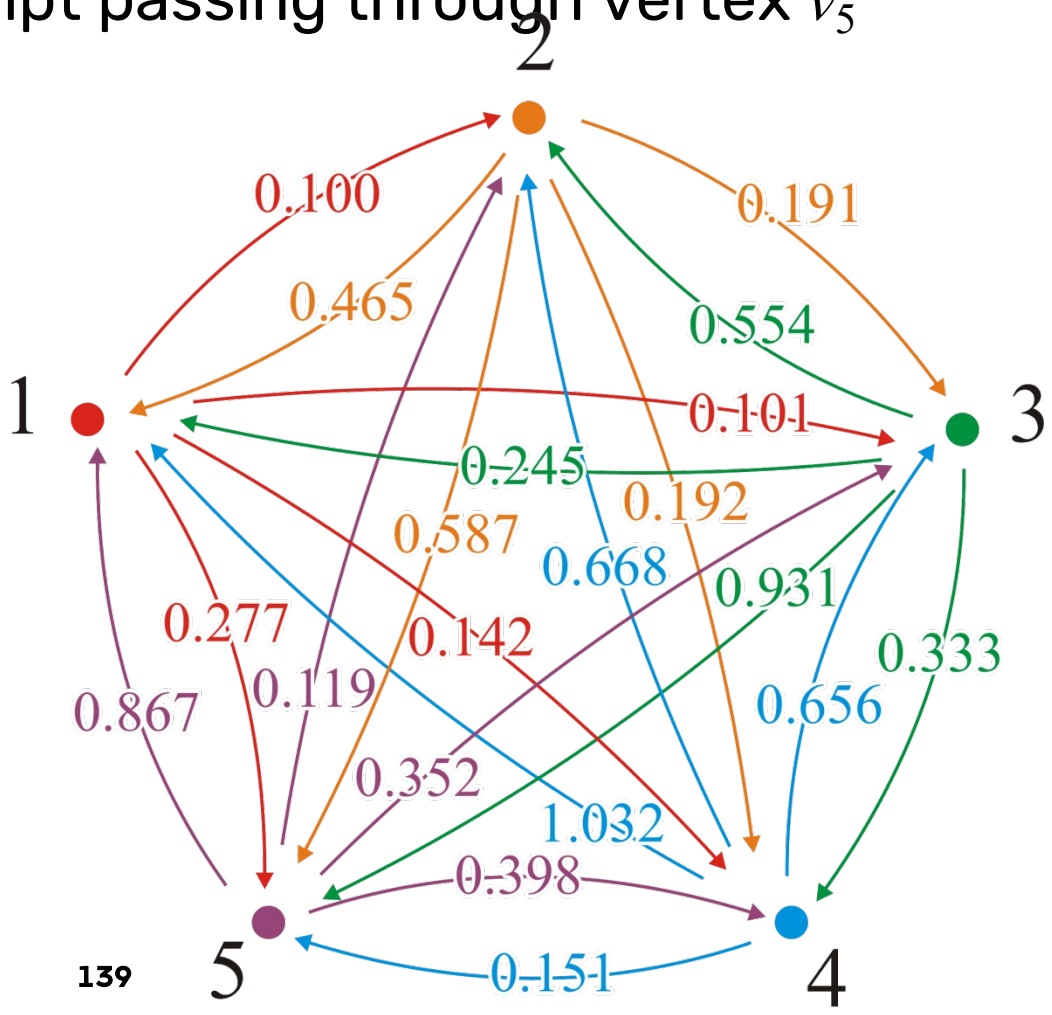


Example

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

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

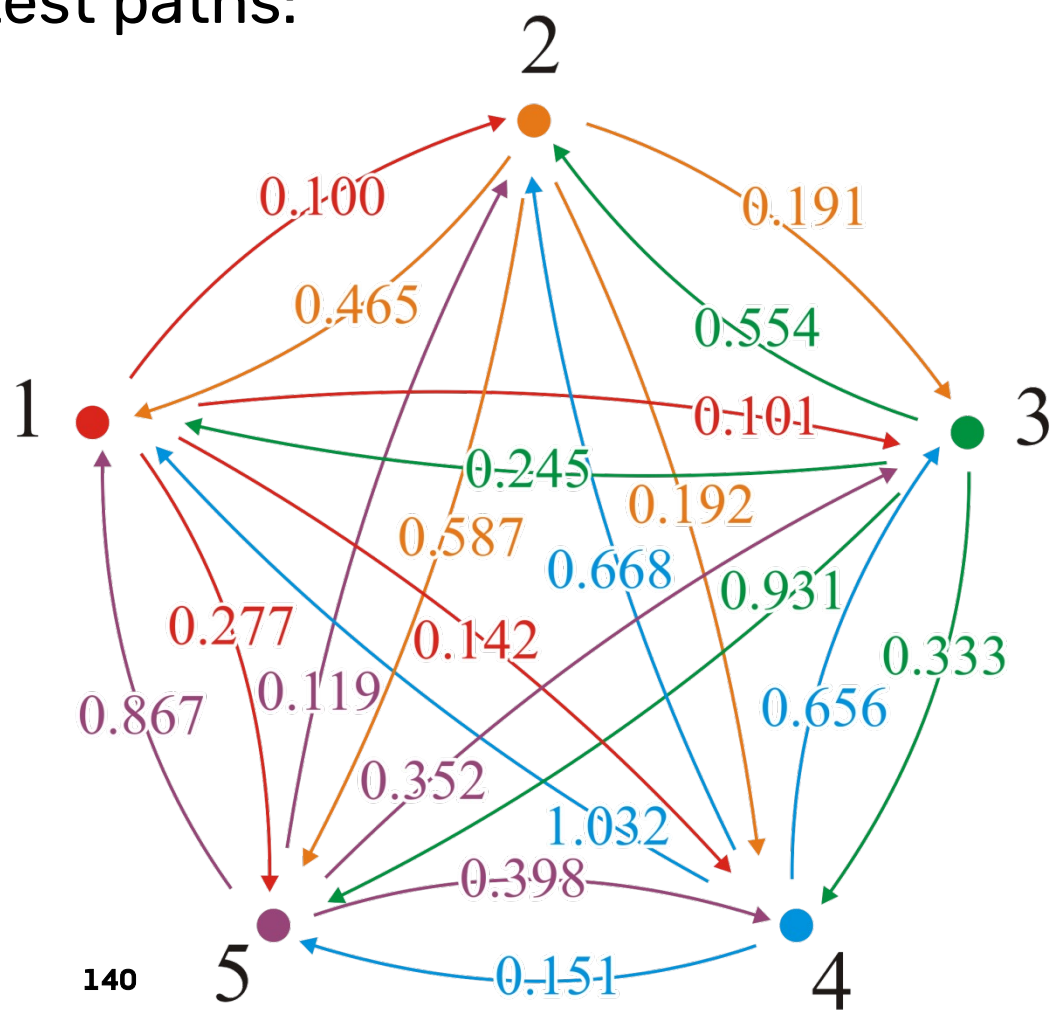
We update the table



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 about the Shortest Path?

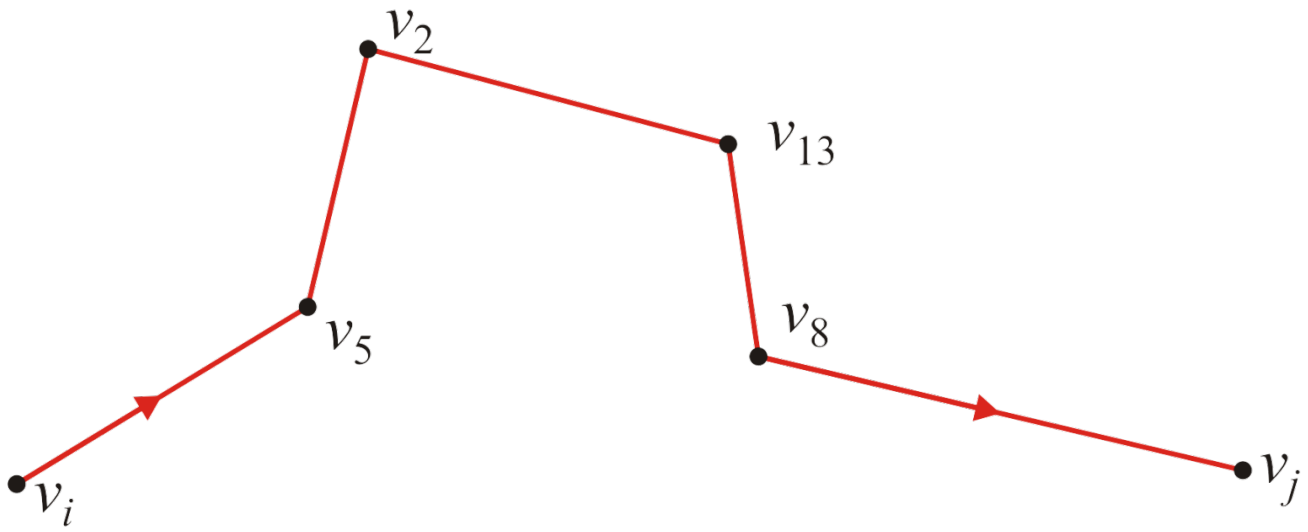
This algorithm finds the shortest distances, but what are the paths corresponding to those shortest distances?

Recall that with Dijkstra's algorithm, we could find the shortest paths by recording the previous node.

You would start at the end and work your way back... (AKA. Back-tracking)

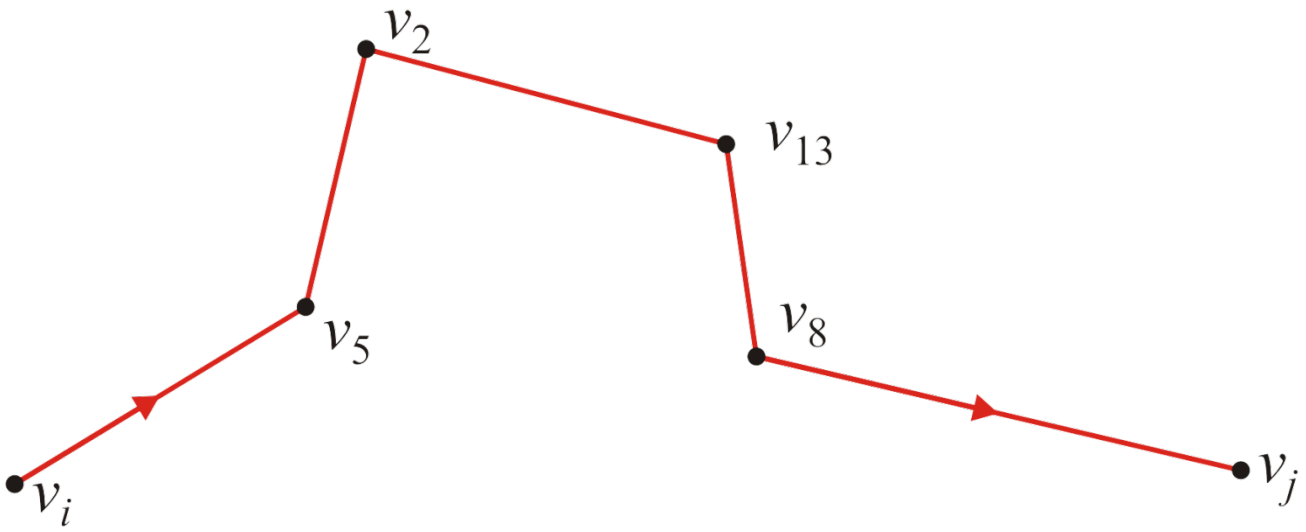
What about the Shortest Path?

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



What about the Shortest Path?

Is this path consists of (v_i, v_5) and the shortest path from v_5 to v_j?

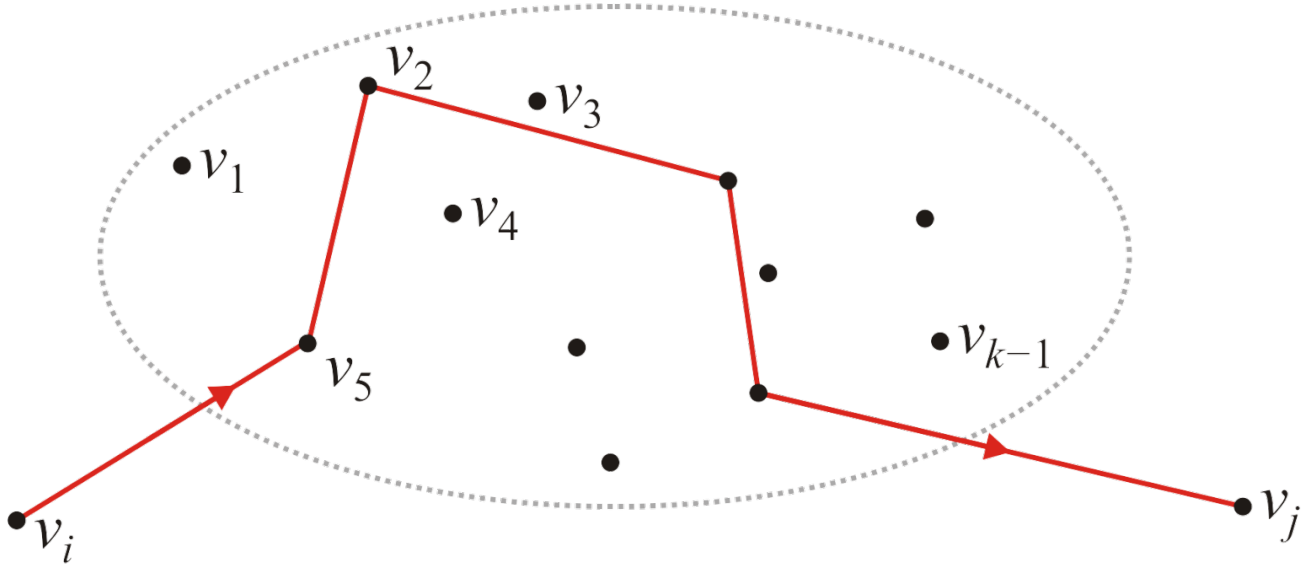


If there was a shorter path from v_i to v_j through v_5 that didn't follow v_2, v_{13}, *etc.*, then we would also find a shorter path from v_5 to v_j

What about the Shortest Path?

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

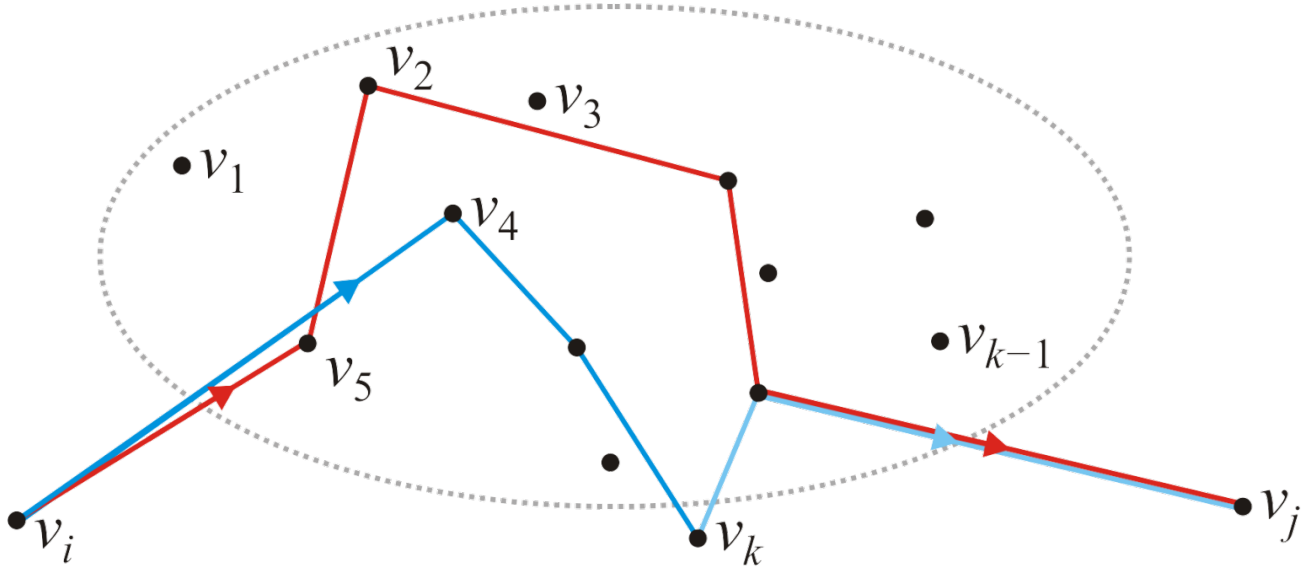
- In this example, the next vertex in the path is v_5



What about the Shortest Path?

What if we find a shorter path passing through v_k?

- In this example, all we'd have to do is now remember that the new path has v_4 as the second node—the rest of the path would be recursively stored as the shortest path from v_4 to v_j



What about the Shortest Path?

In this case, let us store the shortest path moving forward.

P_{ij} stores the next vertex in the shortest path between i and j.

Initialize:

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

Now, if we find a shorter path, update the value

- This matrix will store the next vertex in the list in the shortest path starting at vertex v_i

What about the Shortest Path?

Thus, if we ever find a shorter path, update it the next node: $p_{i,j} := p_{i,k}$

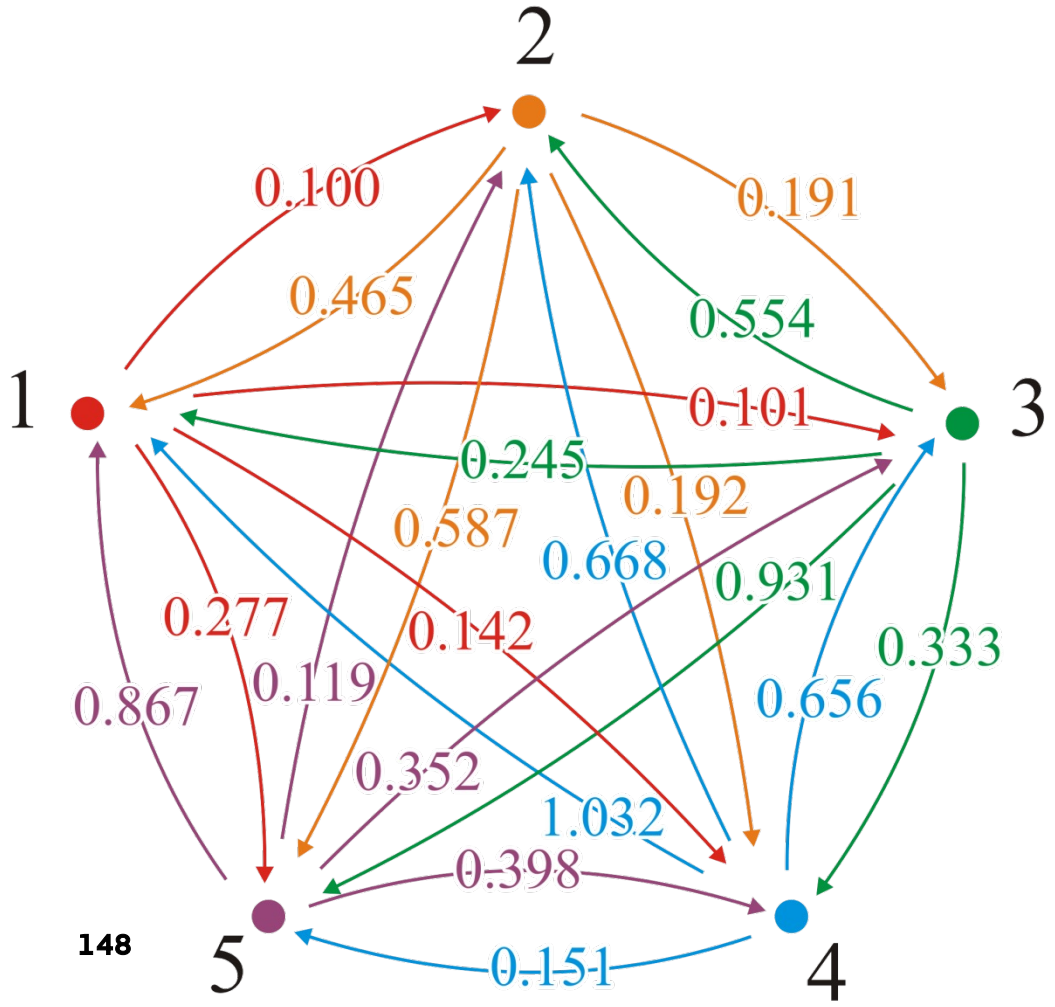
```
unsigned int p[num_vertices][num_vertices];

// Initialize the matrix p:  O(|V|^2)
//   ...

// Run Floyd-Warshall
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

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

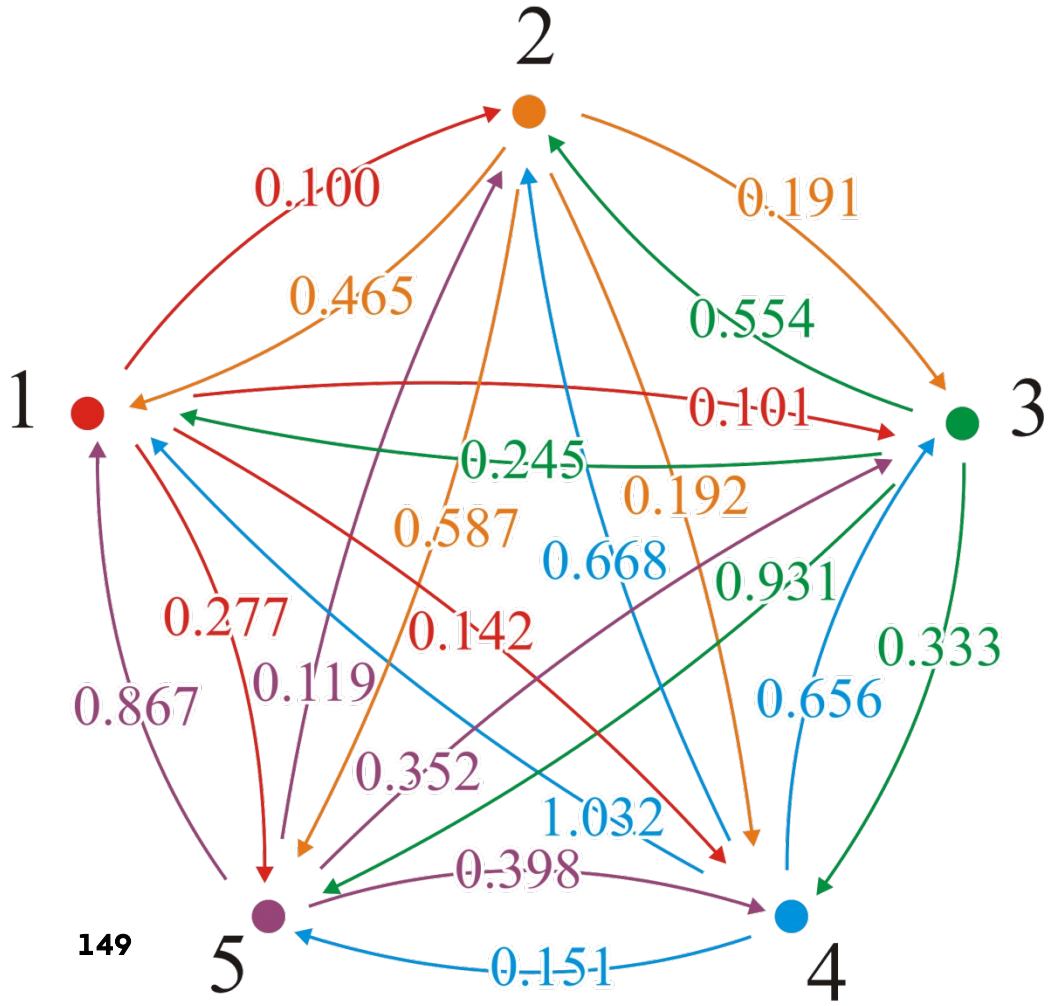
There are two shorter paths:

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

$$0.554 > 0.245 + 0.100$$

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

$$0.931 > 0.245 + 0.277$$

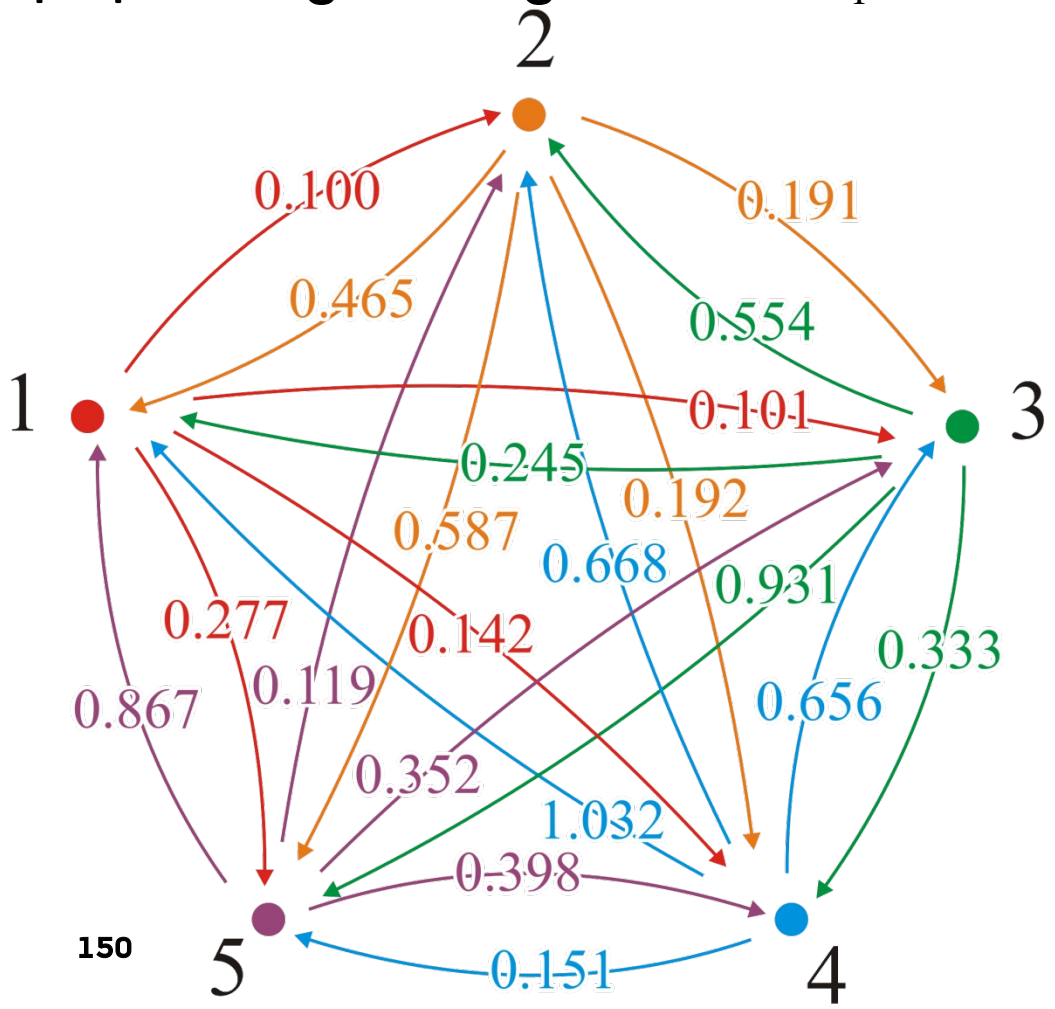


Example

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

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

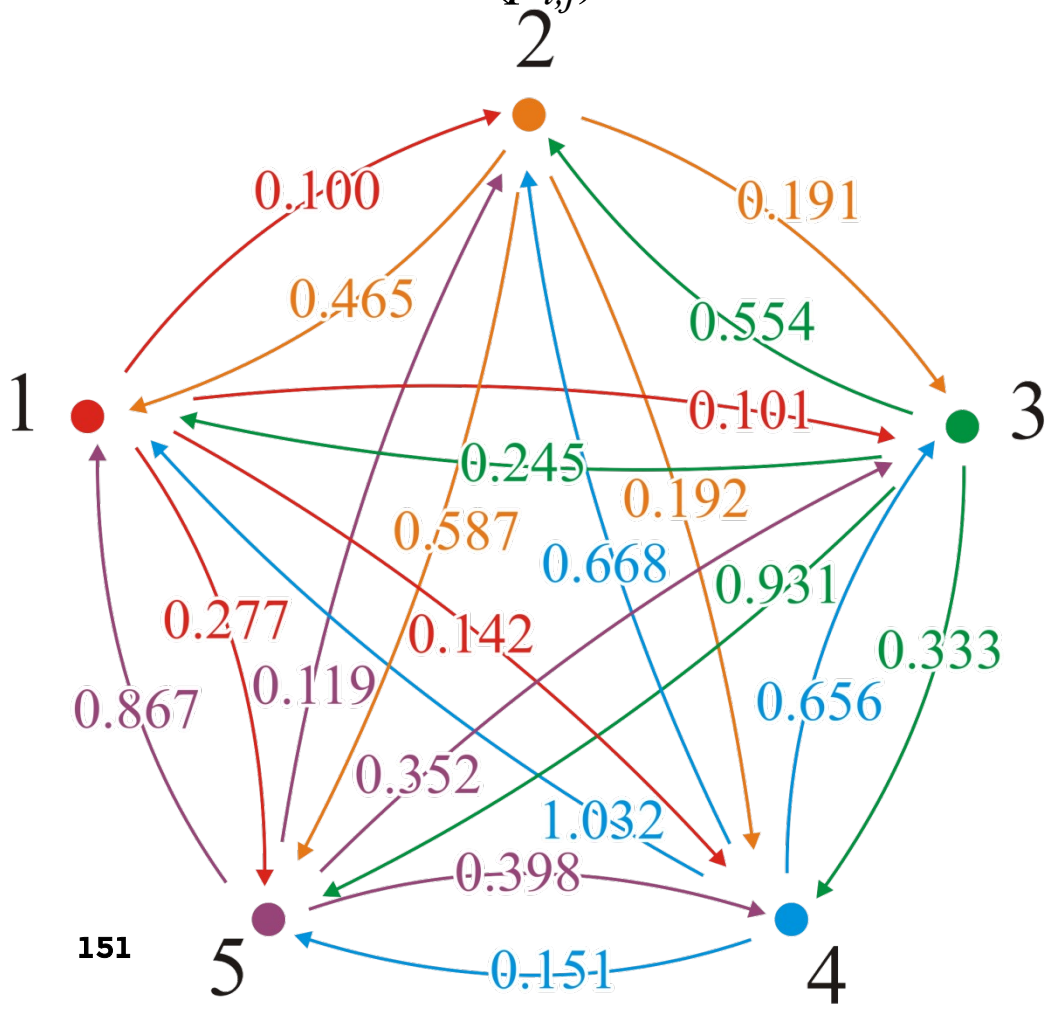
We update each of these



Example

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

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

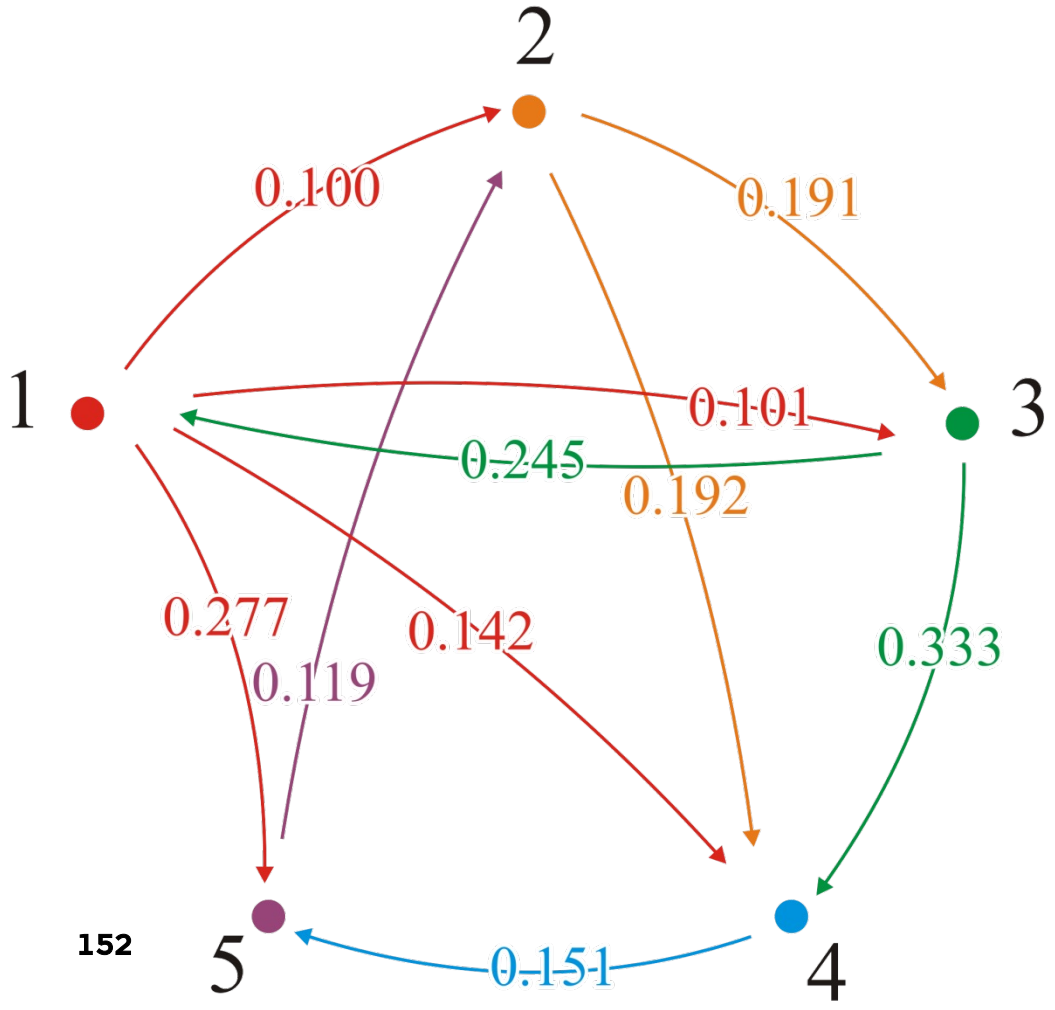


Example

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, $p_{i,j} = j$



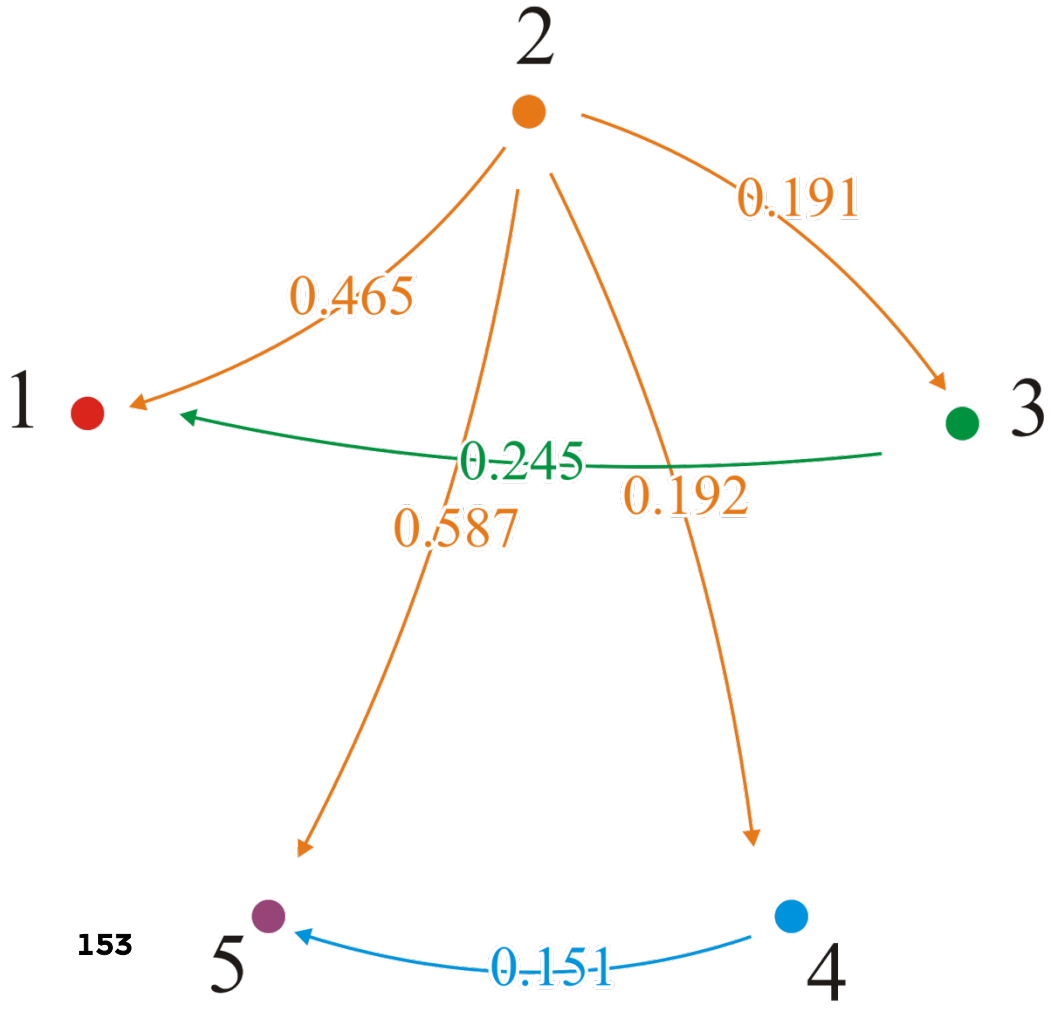
Example

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

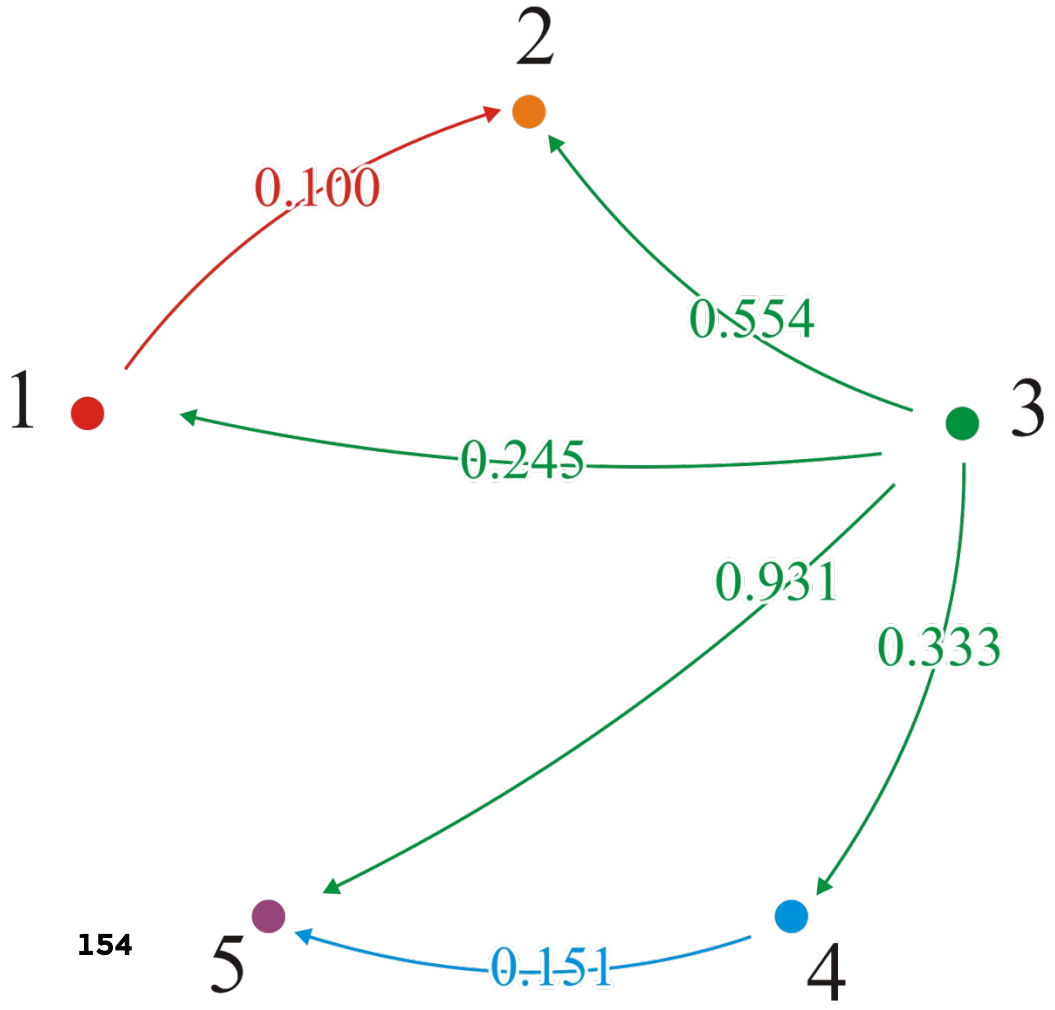
$$\begin{pmatrix} - & 2 & 3 & 4 & 5 \\ 3 & - & 3 & 4 & 4 \\ 1 & 1 & - & 4 & 4 \\ 5 & 5 & 5 & - & 5 \\ 2 & 2 & 2 & 2 & - \end{pmatrix}$$

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 $(3, 4, 5)$

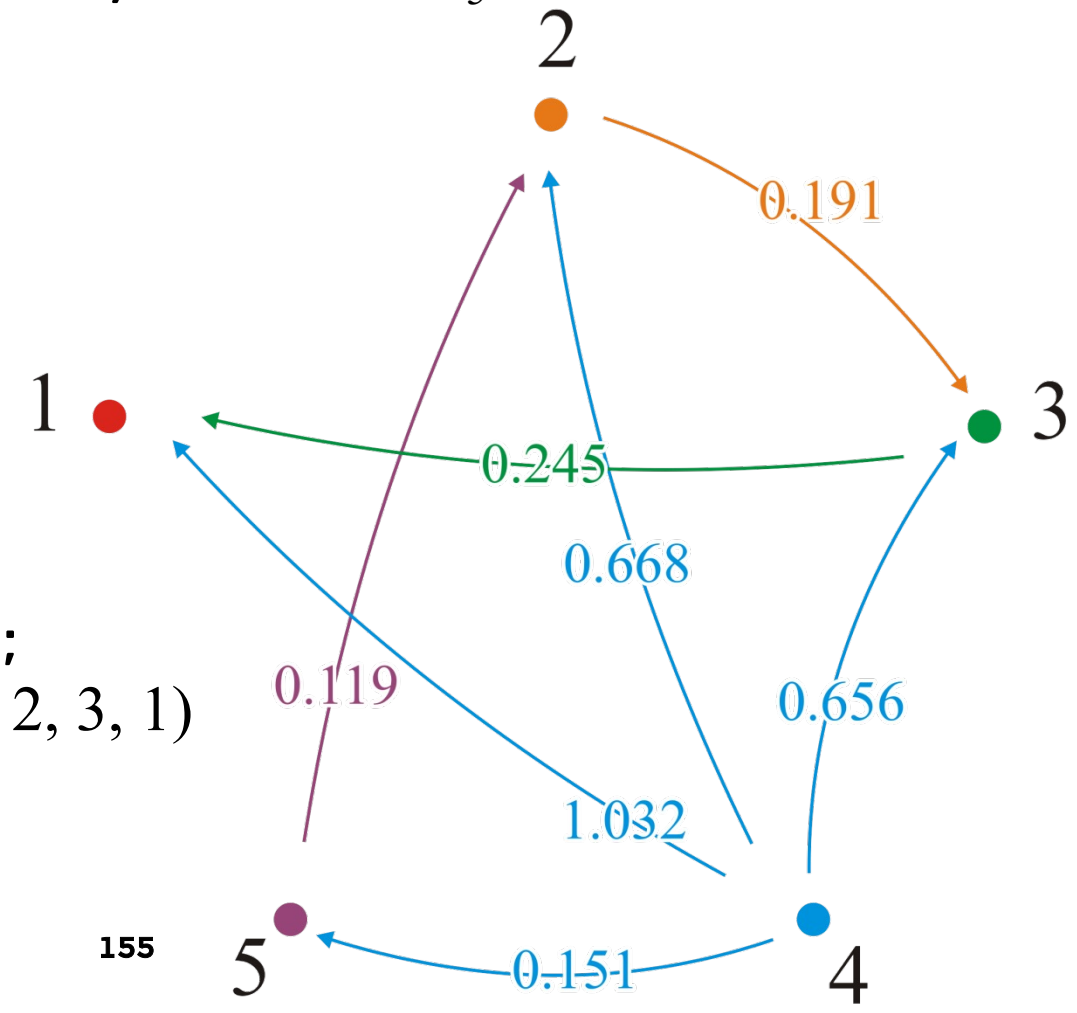


Example

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 $(4, 5, 2, 3, 1)$

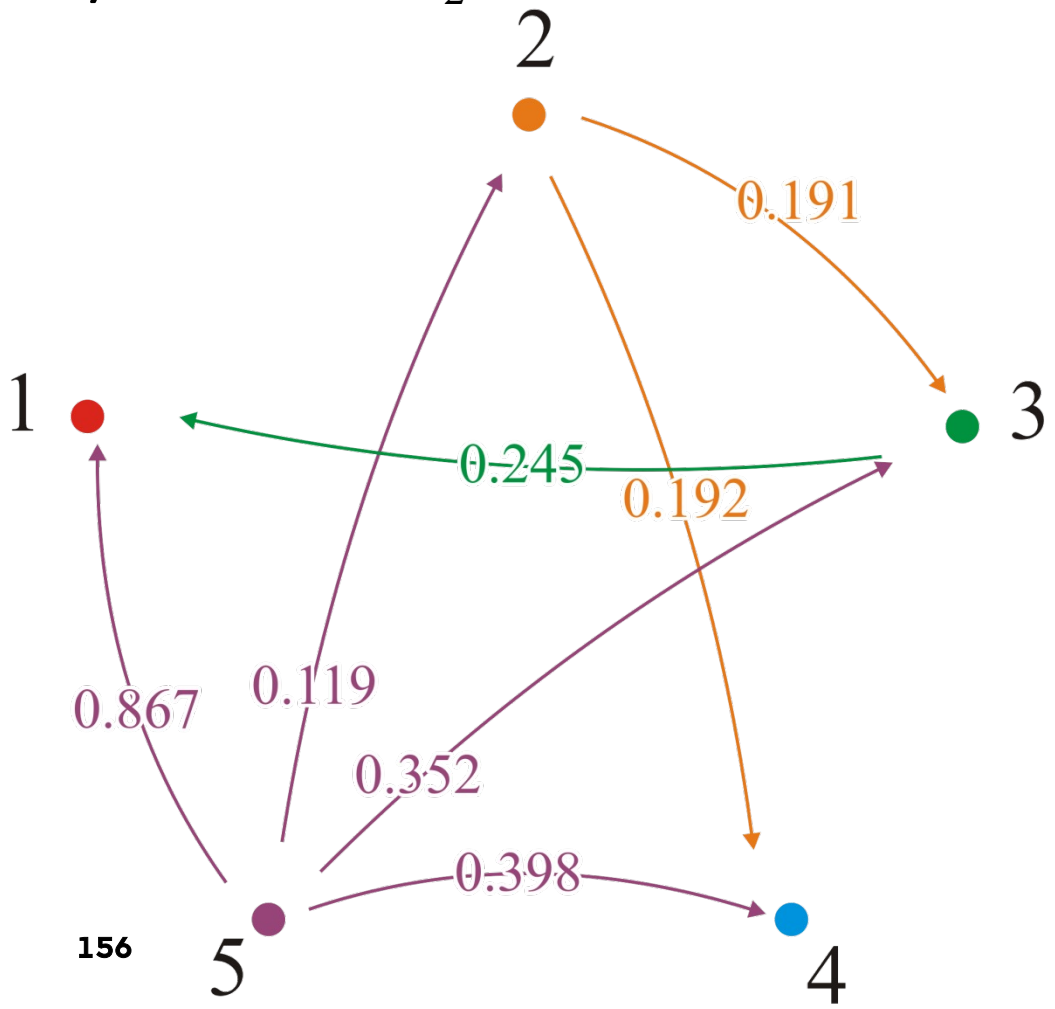


Example

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 $(5, 2, 4)$



Learning Outcome

- Understand the Dijkstra’s algorithm, A* algorithm, and Floyd-Warshall algorithm.

- Know how to use these algorithms to solve the shortest-path problems.

Index-based shortest distance

Drawbacks of online method

Optional

Drawbacks of online search methods (Dijkstra, A*…)

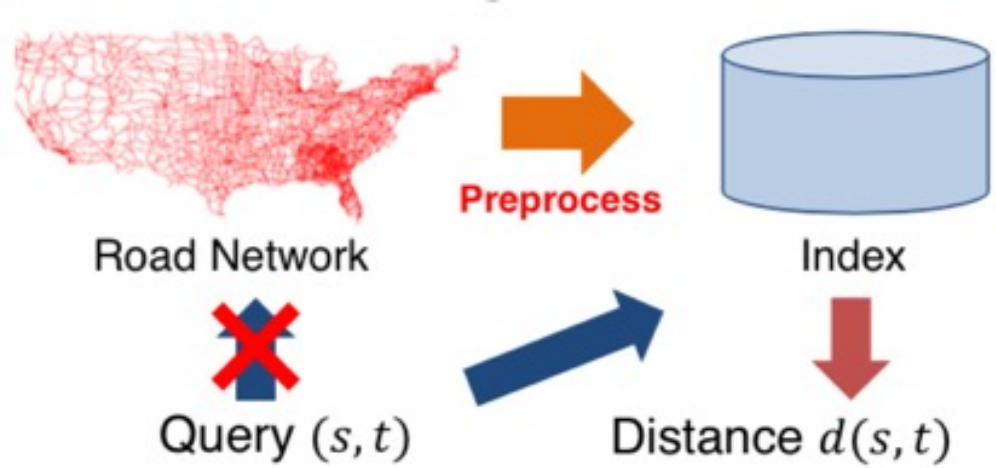
- Takes much time if distance is long.
- Explores too much unnecessary vertices.

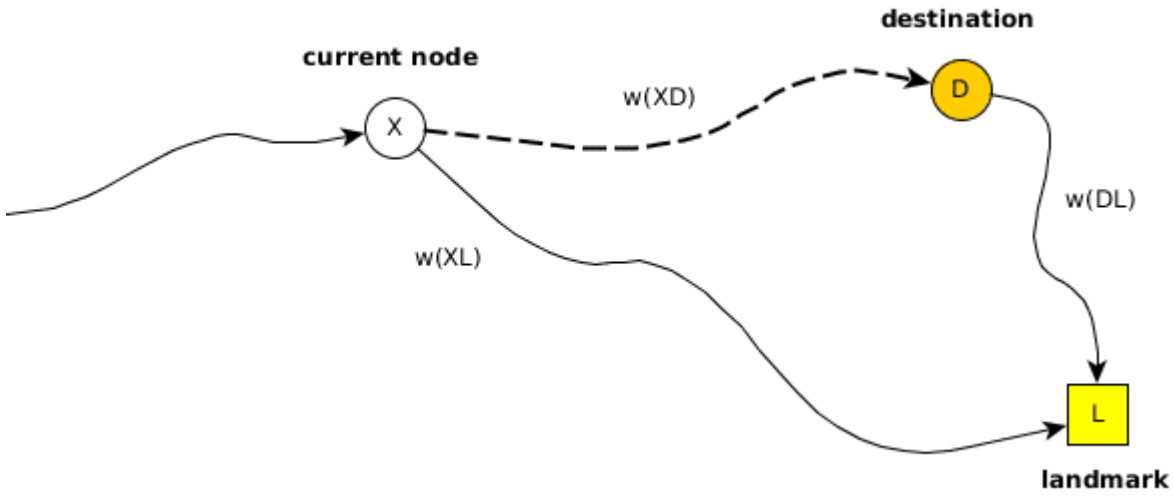
Index-based shortest distance

Optional

Further improve the query efficiency by precomputing small structure to maintain distances.

- Landmark-based global summary structure.
- Tree traversal.





https://dl.acm.org/doi/10.1145/2463676.2465315
https://dl.acm.org/doi/10.1145/3183713.3196913

What researchers are doing...

Optional

- Different Index Structure (Maintenance for dynamic updates)
- Shortest Distance vs Shortest Path
- Shortest Path Counting & Enumeration
- Various Settings (Distributed, Multi-Core . . .)
- Complex Graphs (Road Networks, Social Networks . . .)
- Variations (Label Constrained, Skyline . . .)
- . . .