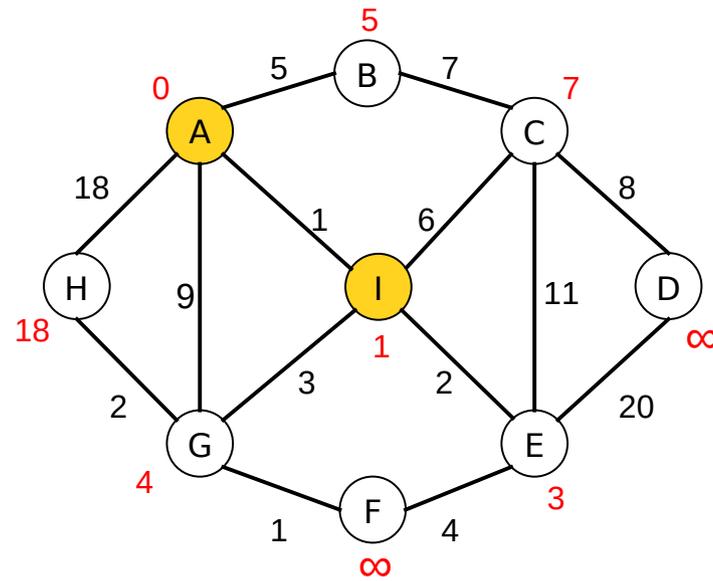


# Dijkstra's Algorithm

An Illustration



Slides by **Sean Szumlanski** for **CS106B**, Programming Abstractions

*Summer 2025*

# Dijkstra's Algorithm

**Goal:** Find the lowest-cost path from some start vertex (source) to every other vertex in the graph.

**Assumptions:** Edges all have non-negative weights.

**Motivation:** Sending a message to every other node in a network as fast as possible.

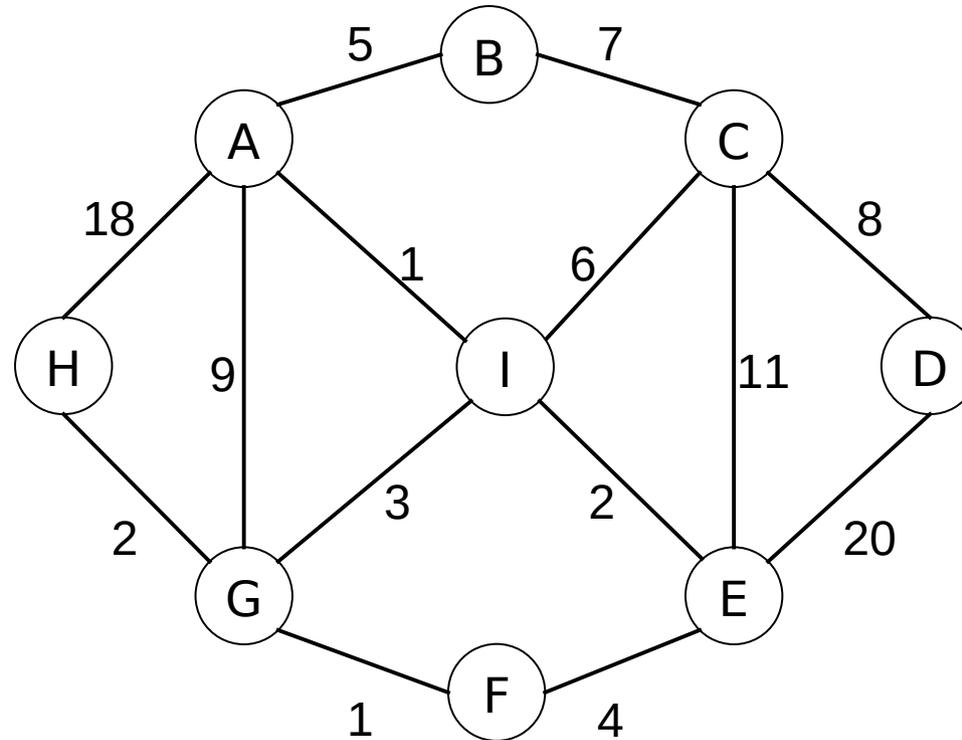
Shipping from a central distribution center, taking the shortest path to all destinations.

Modeling the spread of infectious diseases through social networks.

Evaluating degrees of separation between humans based on social network activity.

# Dijkstra's Algorithm

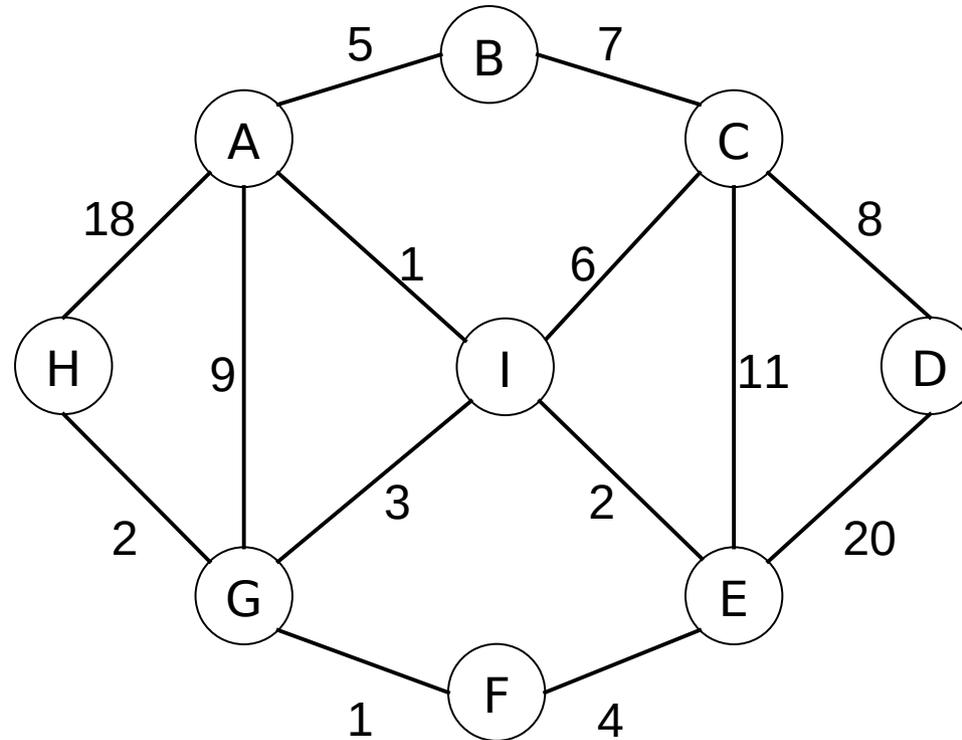
(calculating the cheapest path from a source vertex to all other vertices)



Let's trace through the algorithm to see how it works.

# Dijkstra's Algorithm

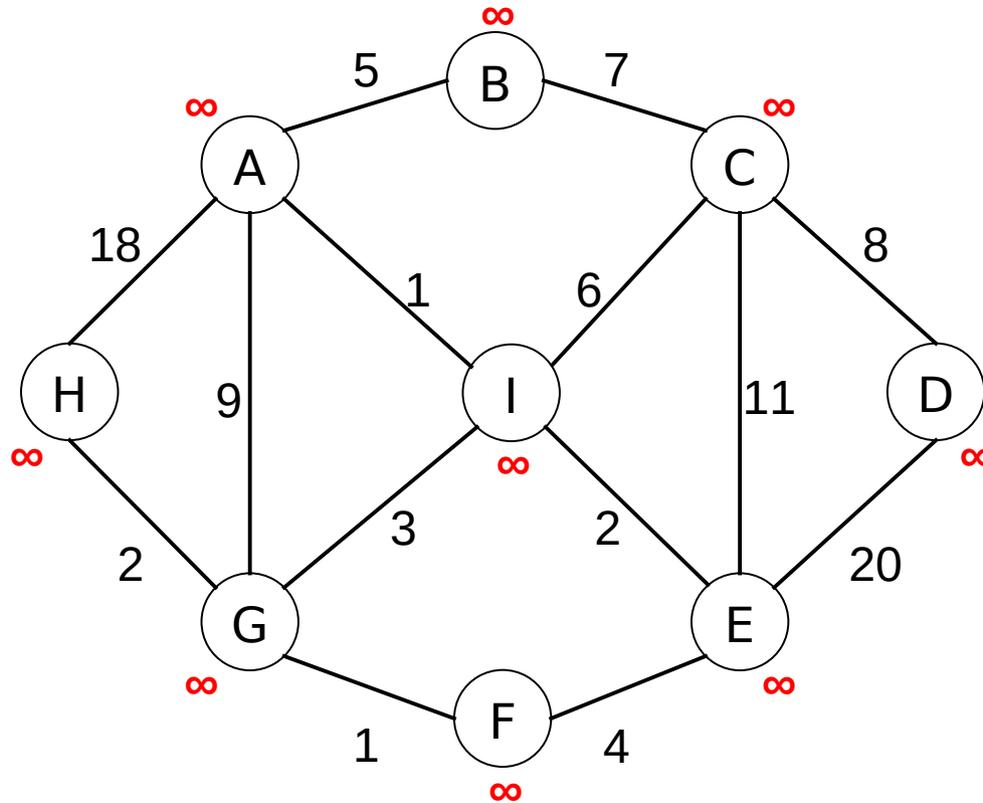
(calculating the cheapest path from a source vertex to all other vertices)



1: Initialize a value at each vertex to infinity ( $\infty$ ). Call these values  $\text{dist}[i]$ .

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

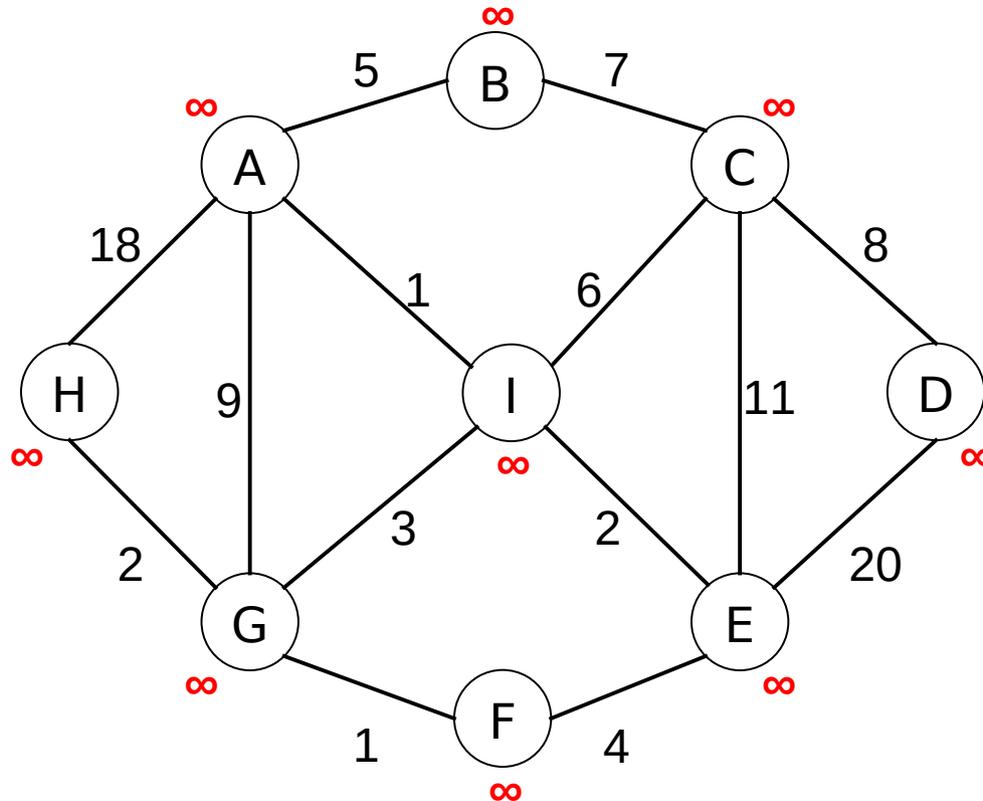


1: Initialize a value at each vertex to infinity ( $\infty$ ). Call these values  $\text{dist}[i]$ .

**Note:** These  $\infty$  values represent the cost of reaching each vertex from our source, using only intermediary vertices whose shortest paths we have already found.

# Dijkstra's Algorithm

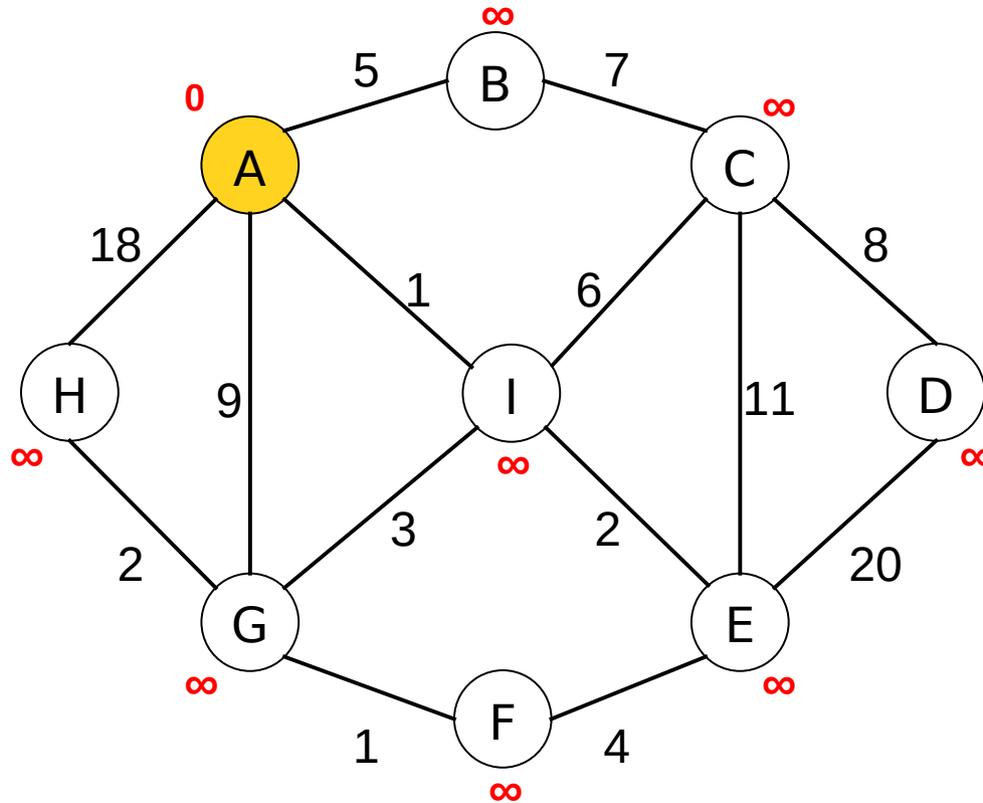
(calculating the cheapest path from a source vertex to all other vertices)



- 2: Initialize the value at our source vertex to zero and mark the source vertex as visited.

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

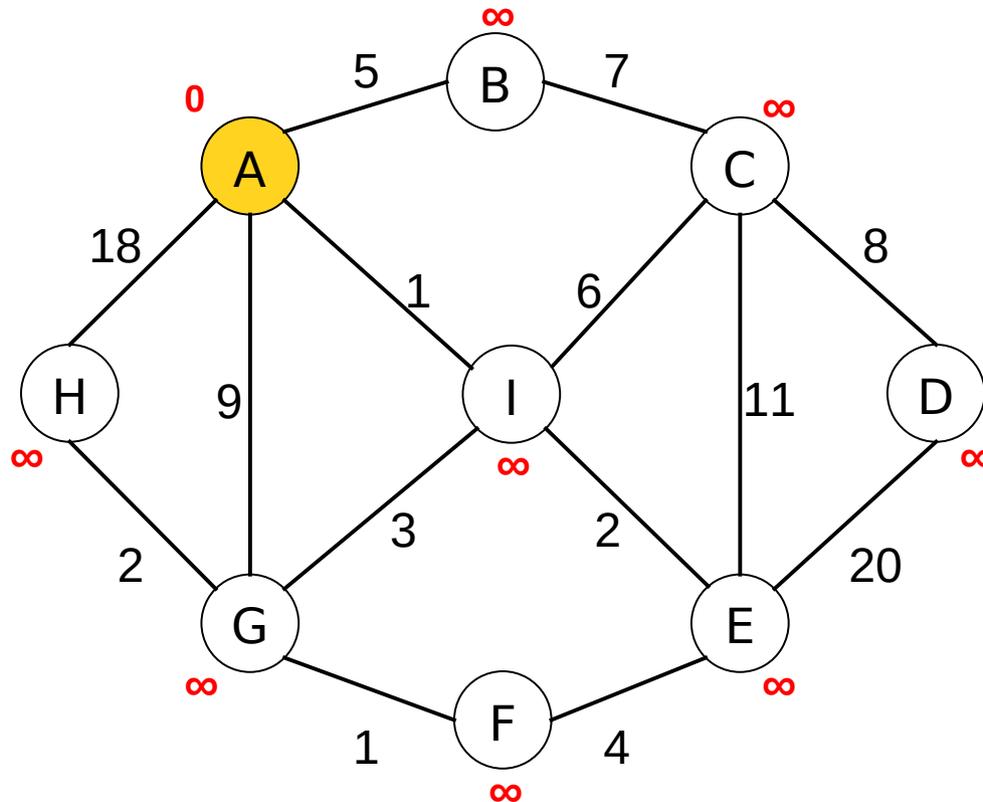


- 2: Initialize the value at our source vertex to zero and mark the source vertex as visited.

**Note:** Clearly we can get from vertex A to vertex A at a cost of zero...

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

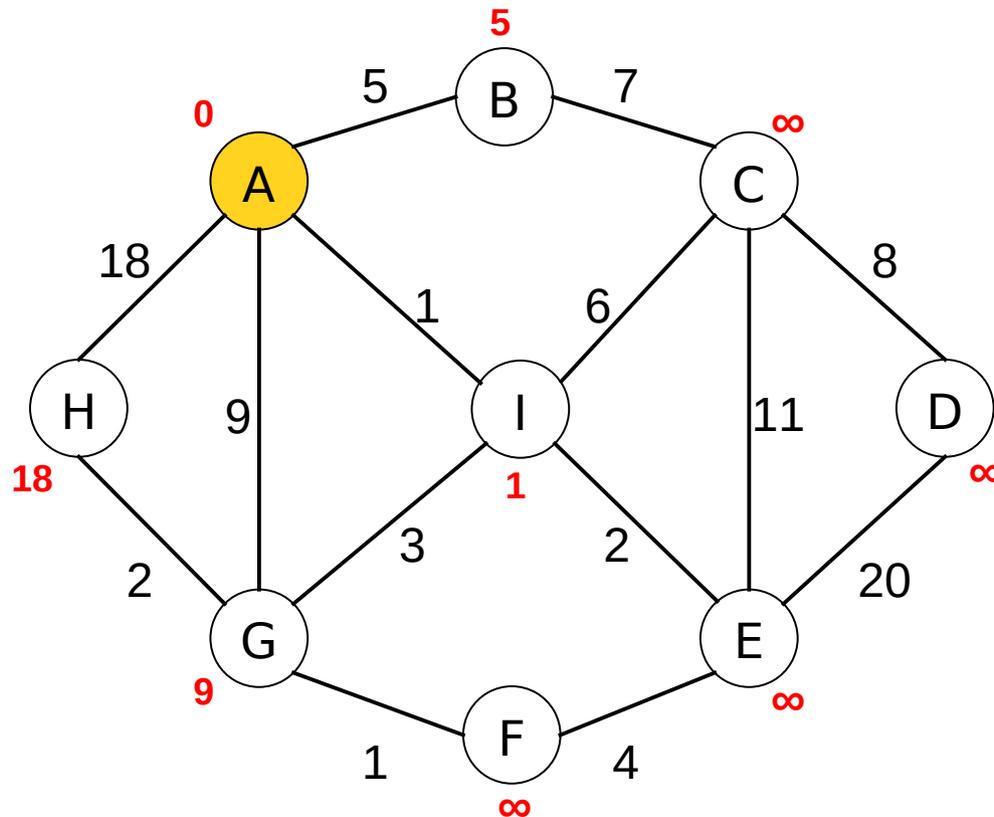


3: Update the cost of getting from the source vertex to every other vertex:

$$\text{MIN}\{\text{weight}[\text{source}, i], \text{dist}[i]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



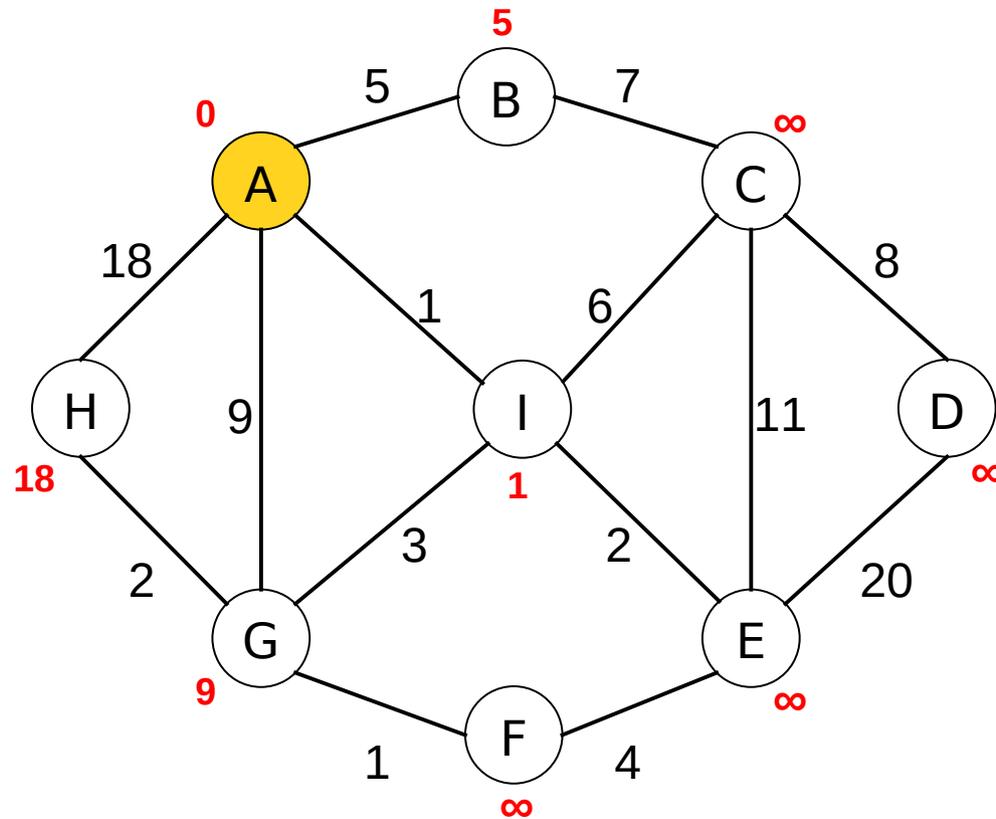
3: Update the cost of getting from the source vertex to every other vertex:

$$\text{MIN}\{\text{weight}[\text{source}, i], \text{dist}[i]\}$$

**Note:** These  $\text{dist}[i]$  values now represent the lowest-cost path from A using no intermediate vertices. (Unless we had negative edge weights...)

# Dijkstra's Algorithm

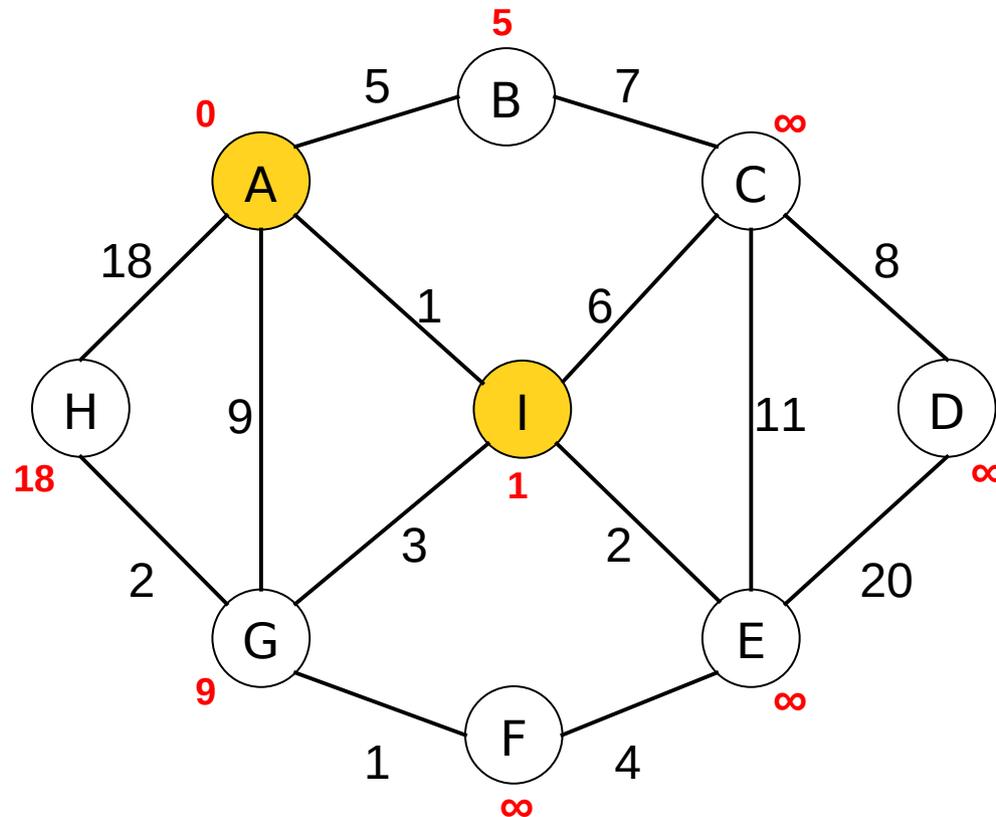
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

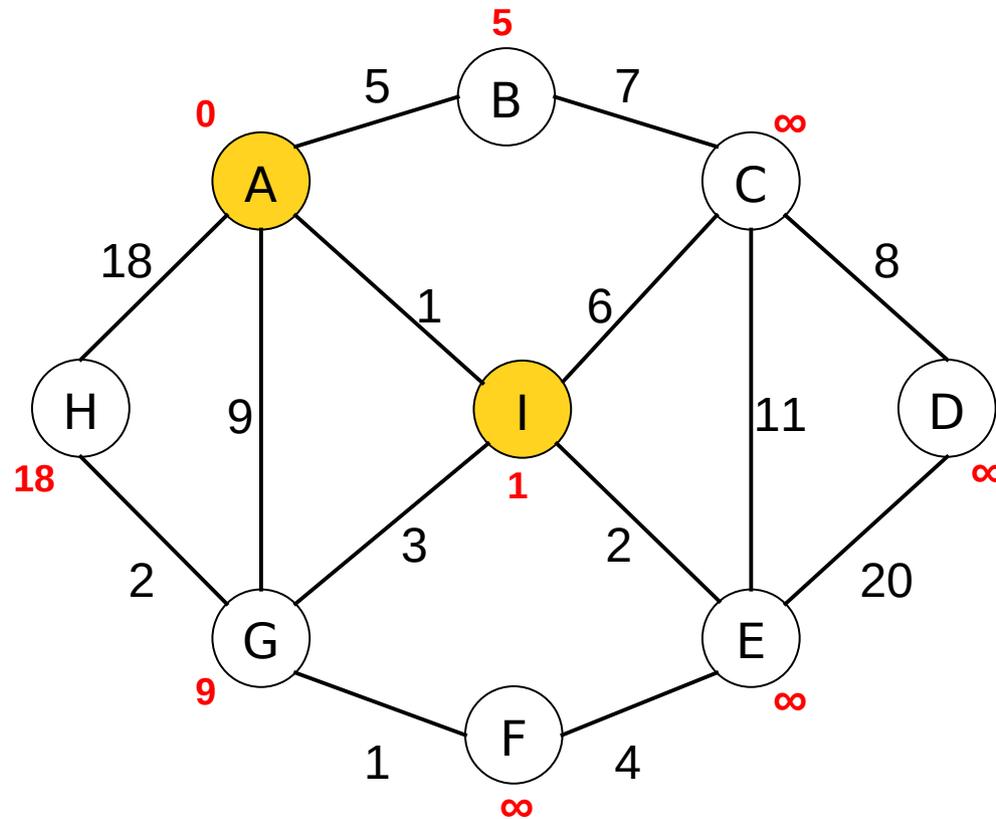


4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.

**Note:** Clearly we have found a shortest path from vertex A to vertex I, since any other path must go through edges of greater (or equal) weight.

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

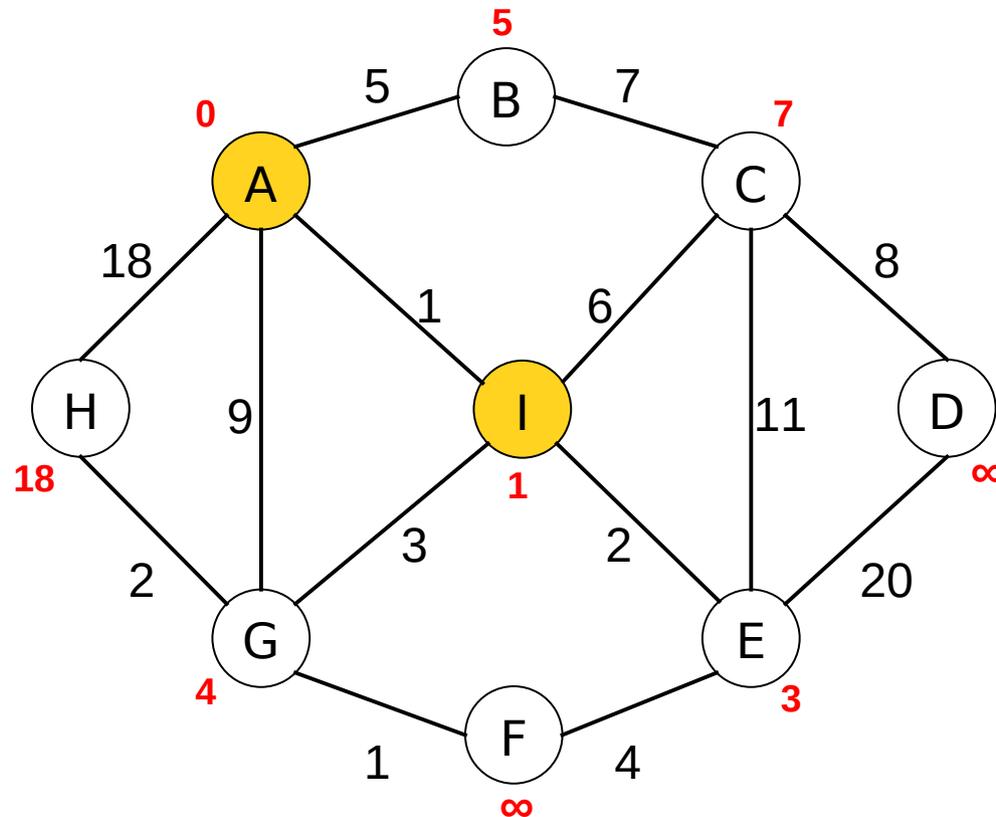


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



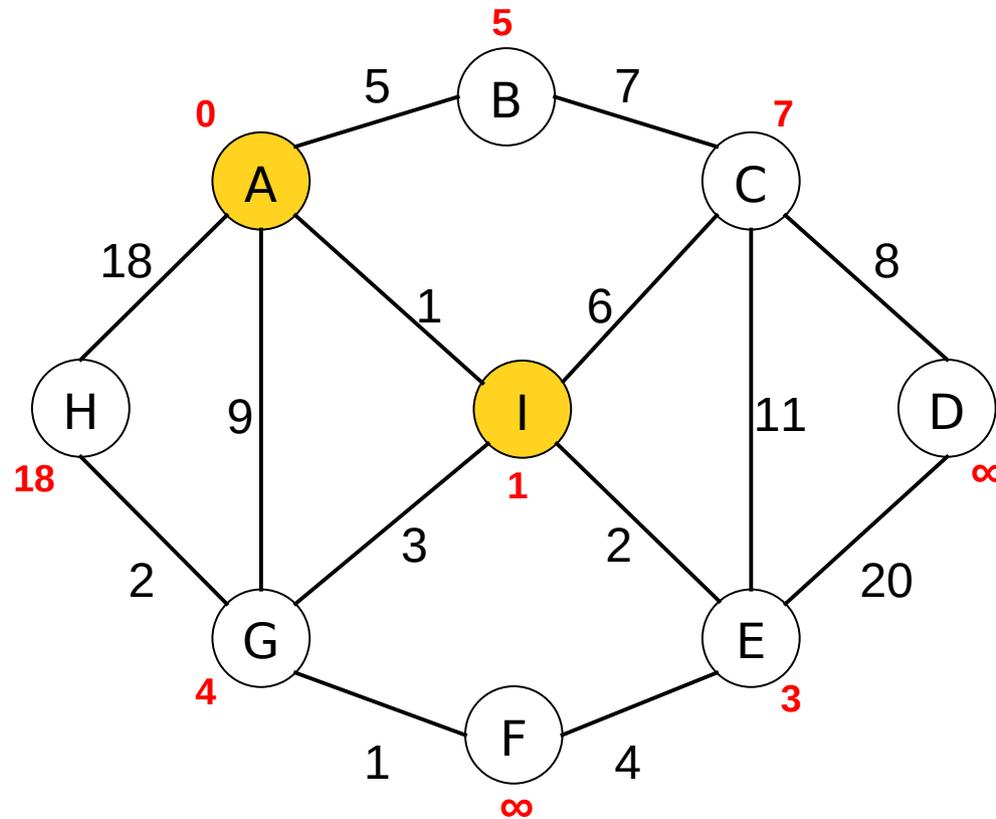
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertex I to be used as an intermediary vertex along the path.

# Dijkstra's Algorithm

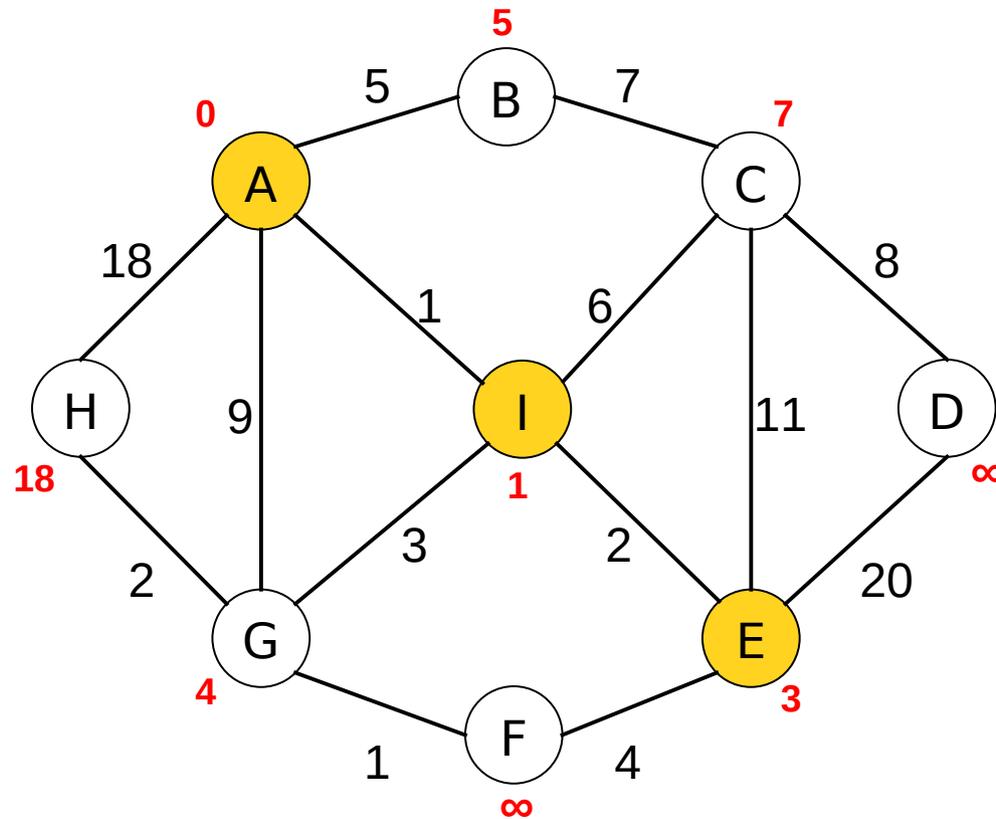
(calculating the cheapest path from a source vertex to all other vertices)



- 4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

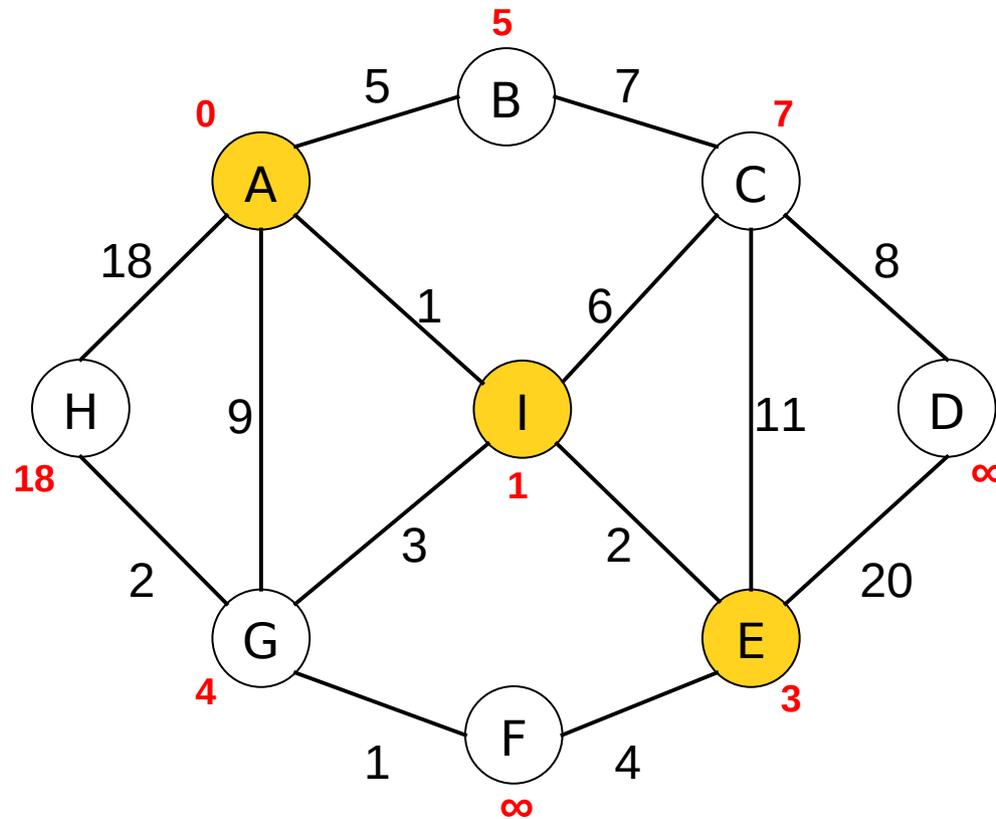
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

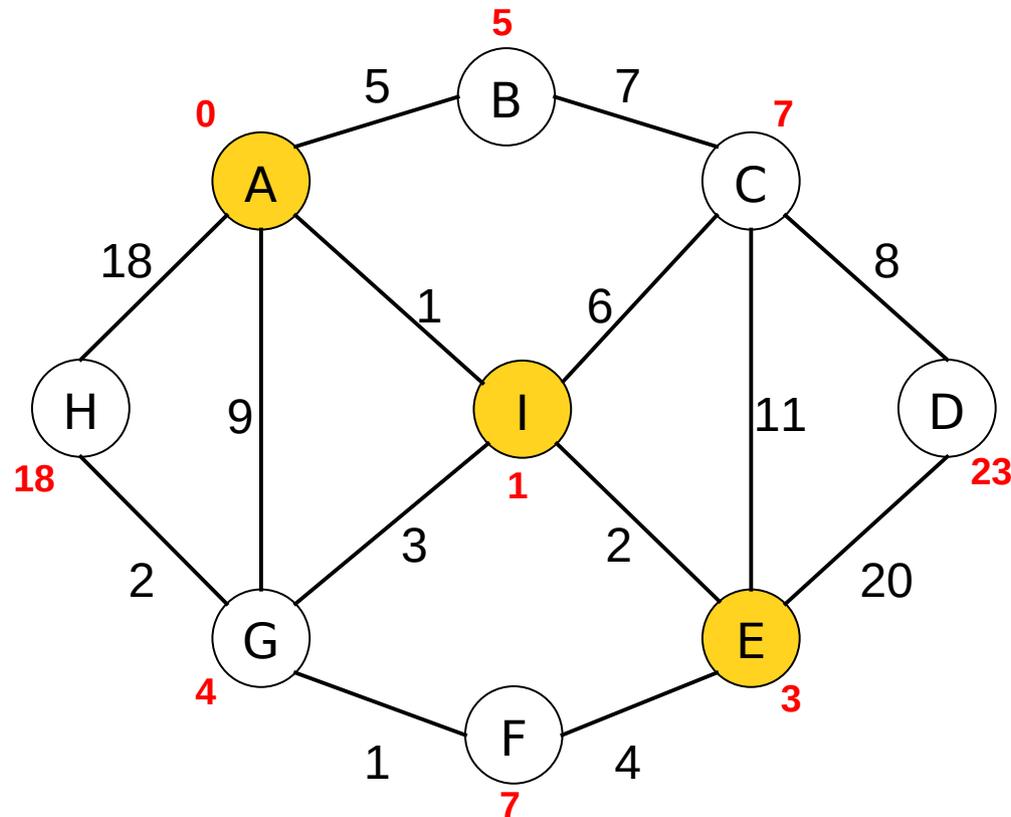


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



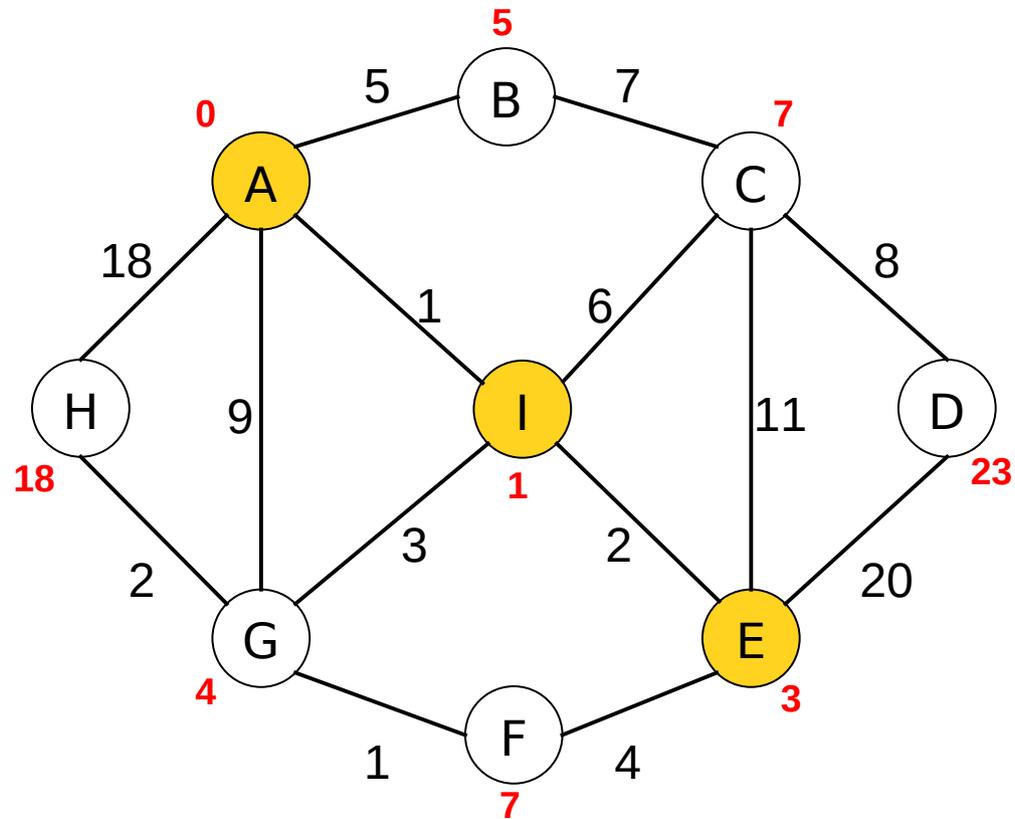
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I and E to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

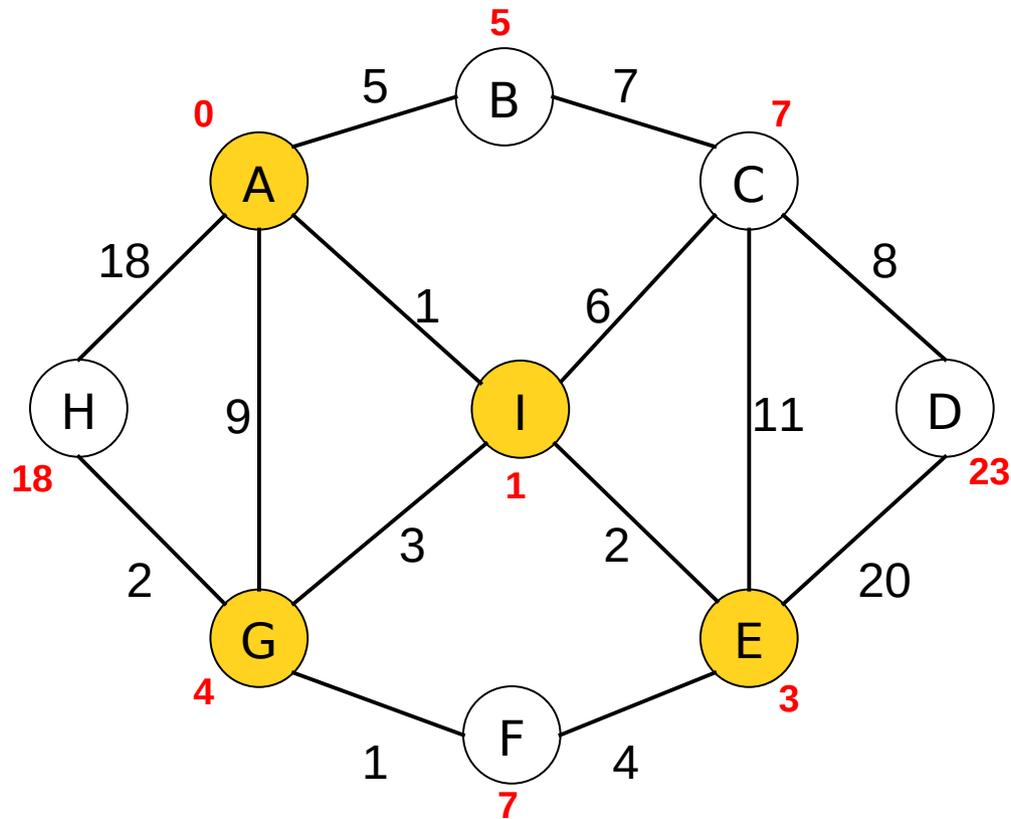
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

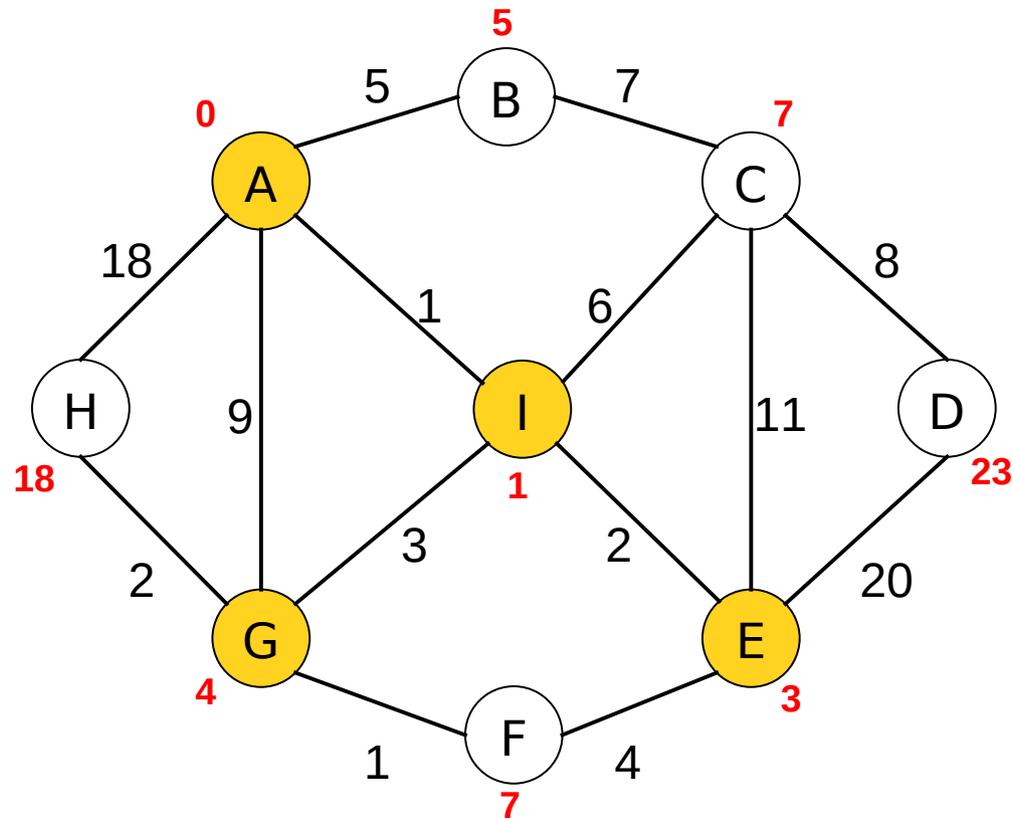
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

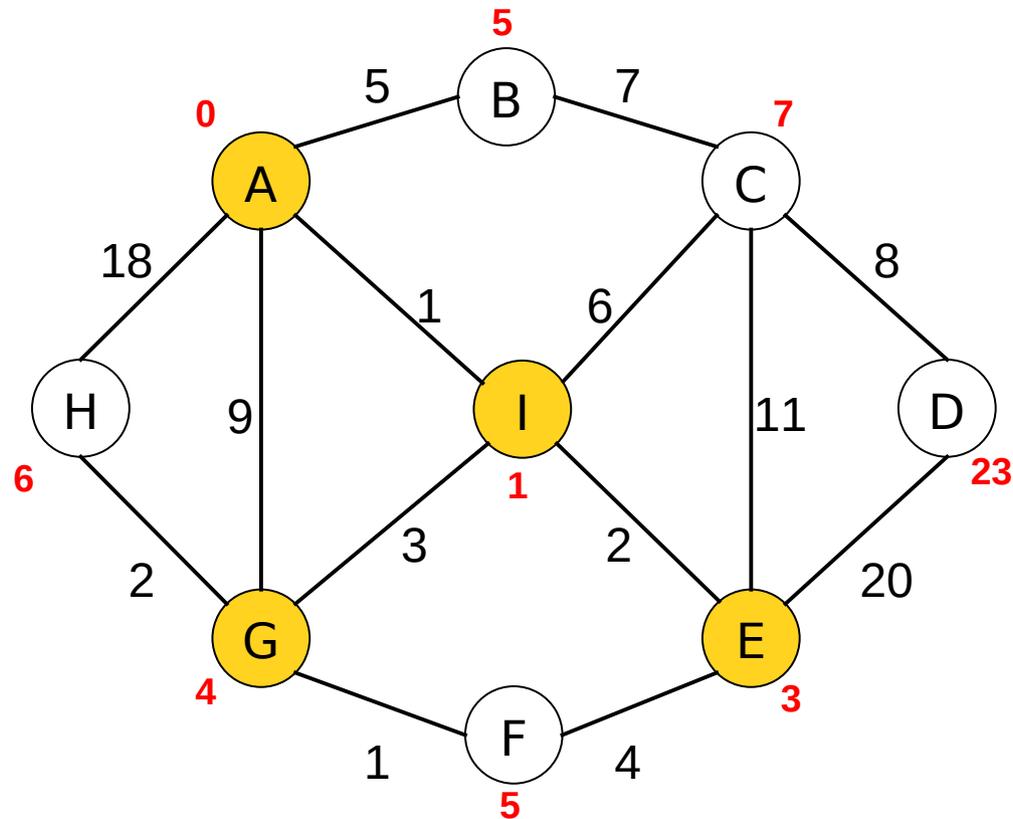


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



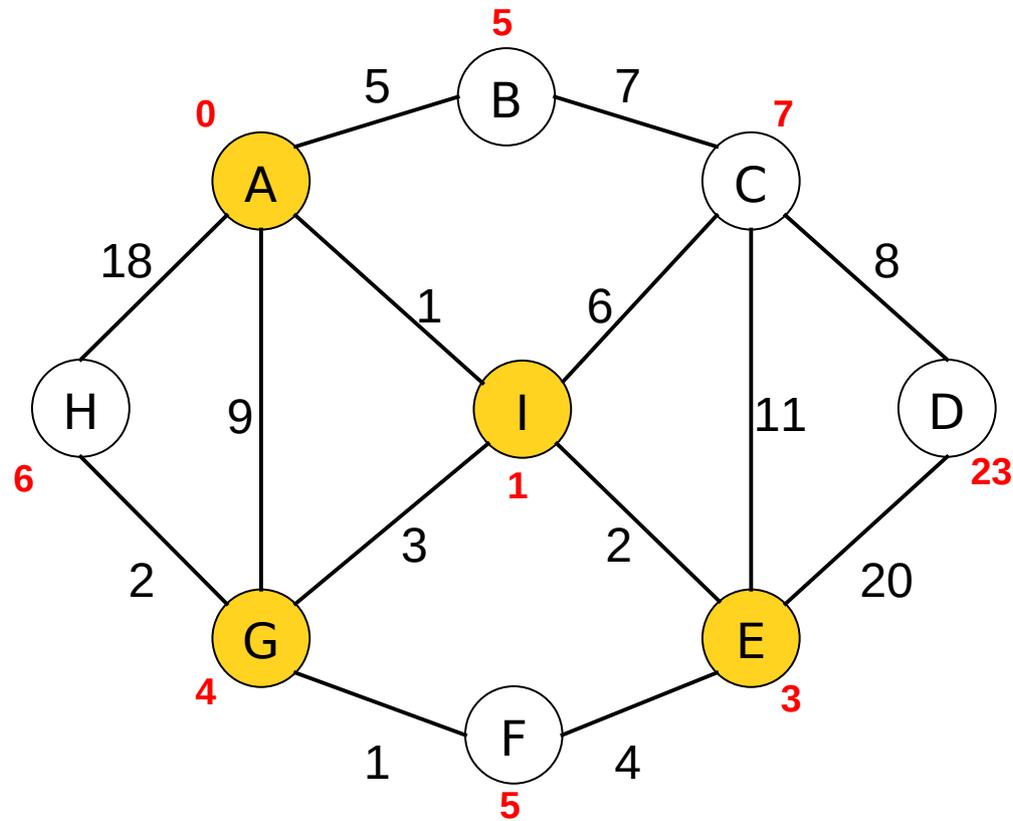
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I, E, and G to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

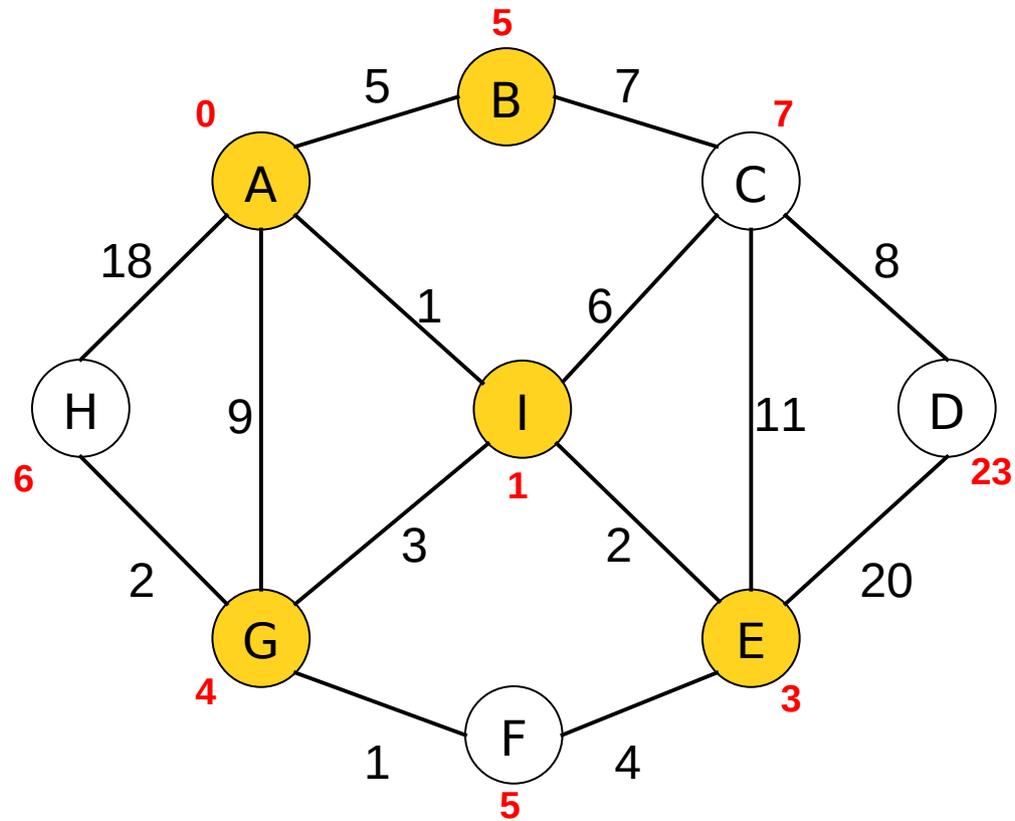
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

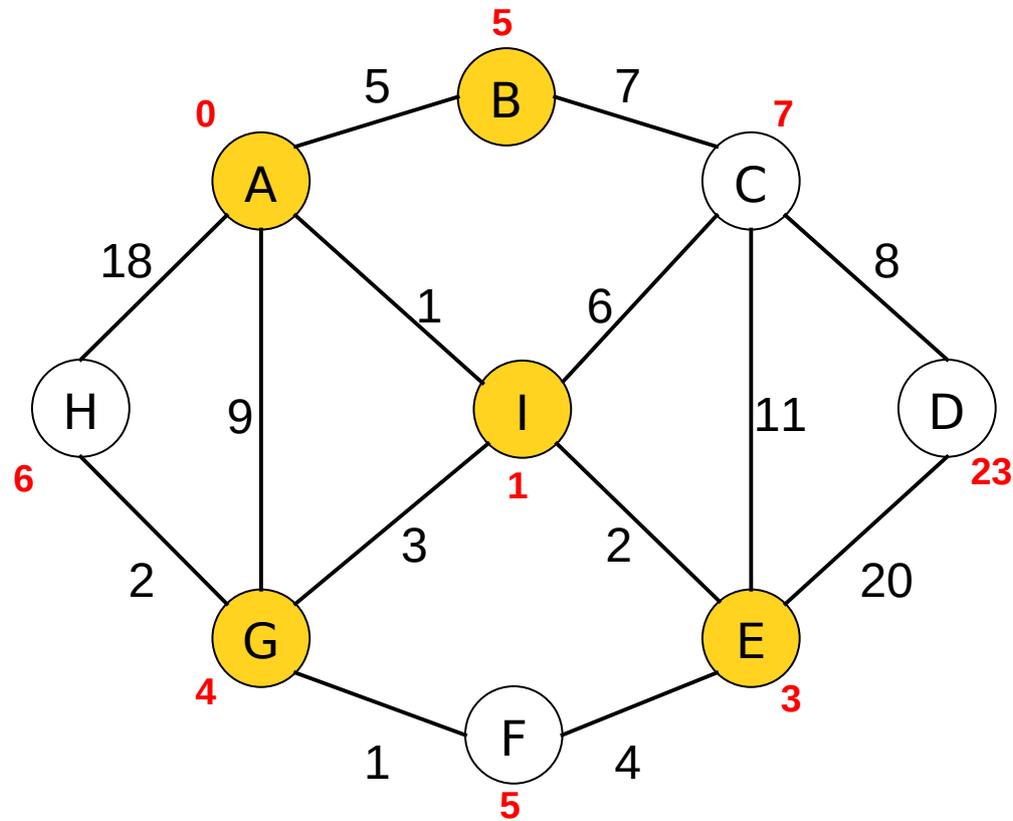
(calculating the cheapest path from a source vertex to all other vertices)



- 4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

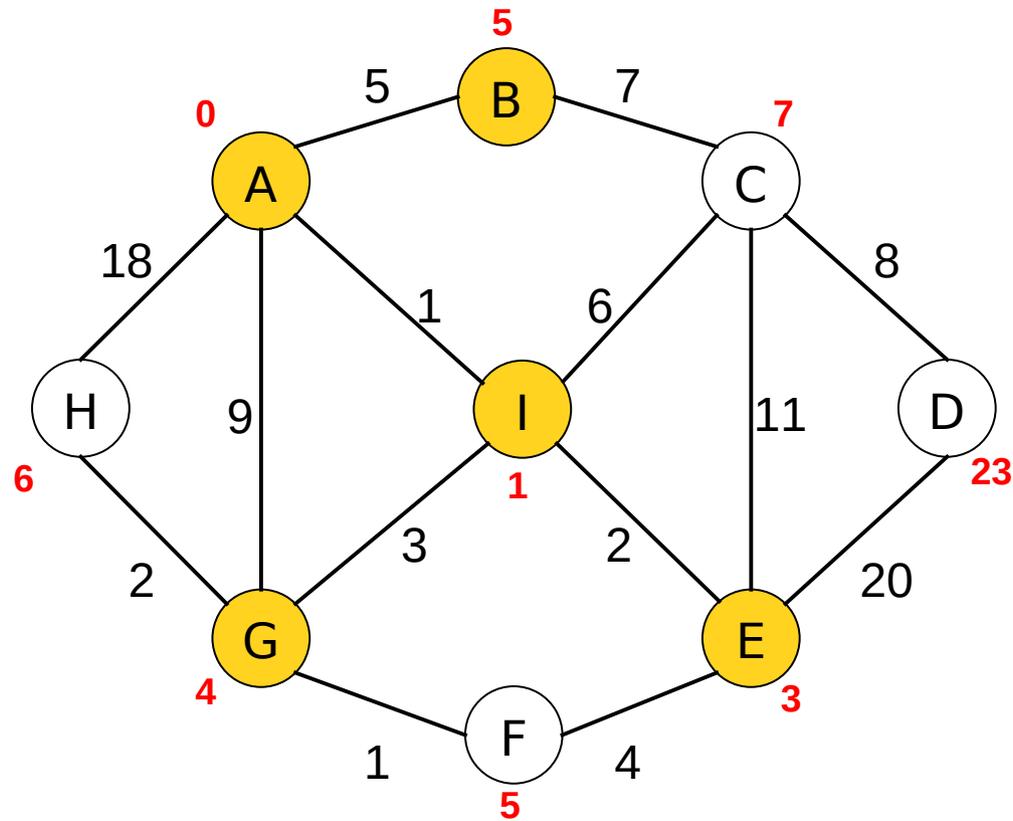


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



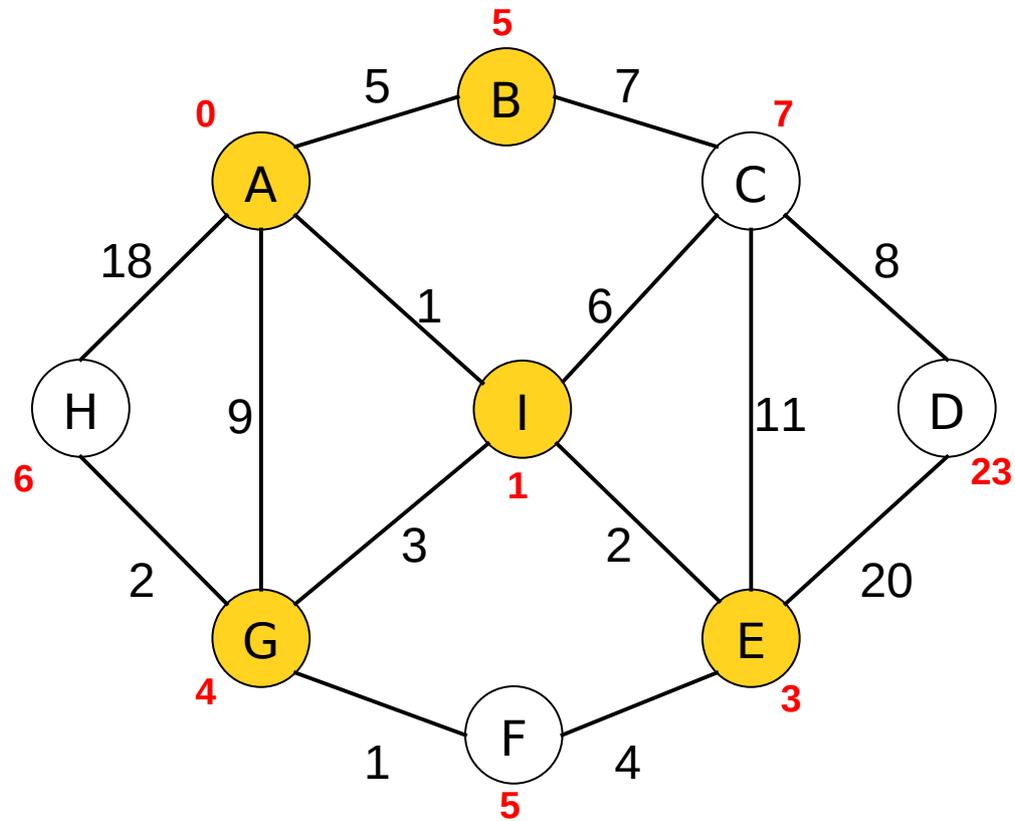
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I, E, G, and B to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

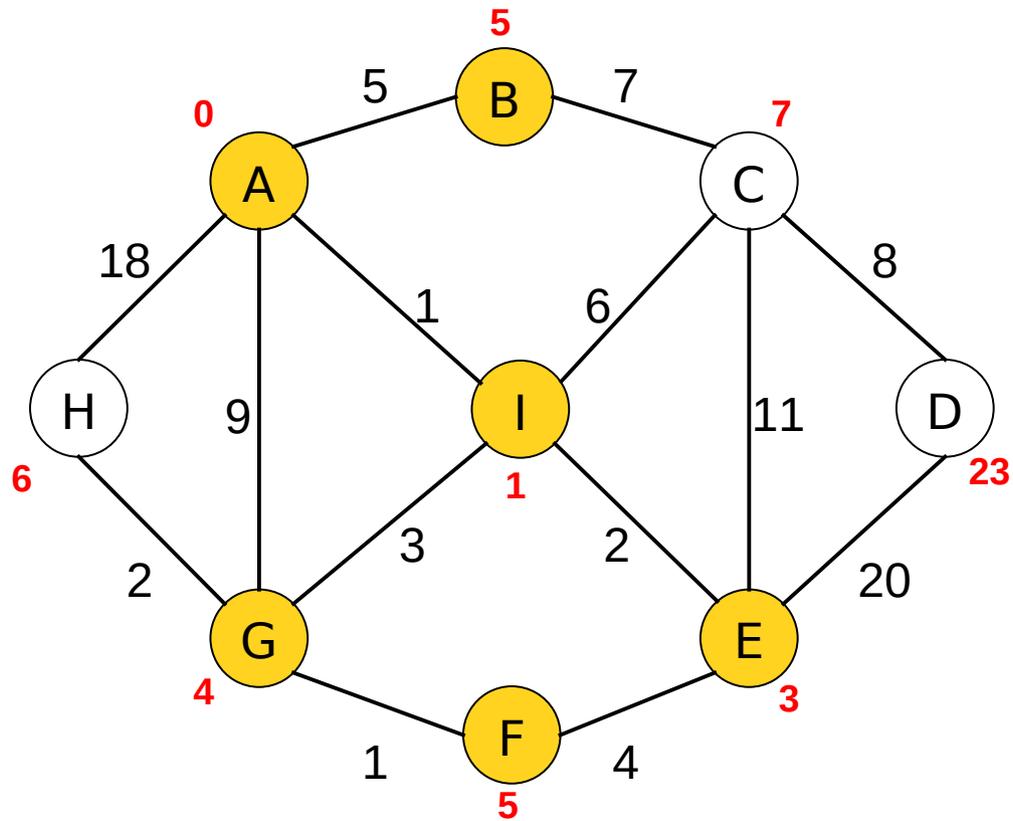
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

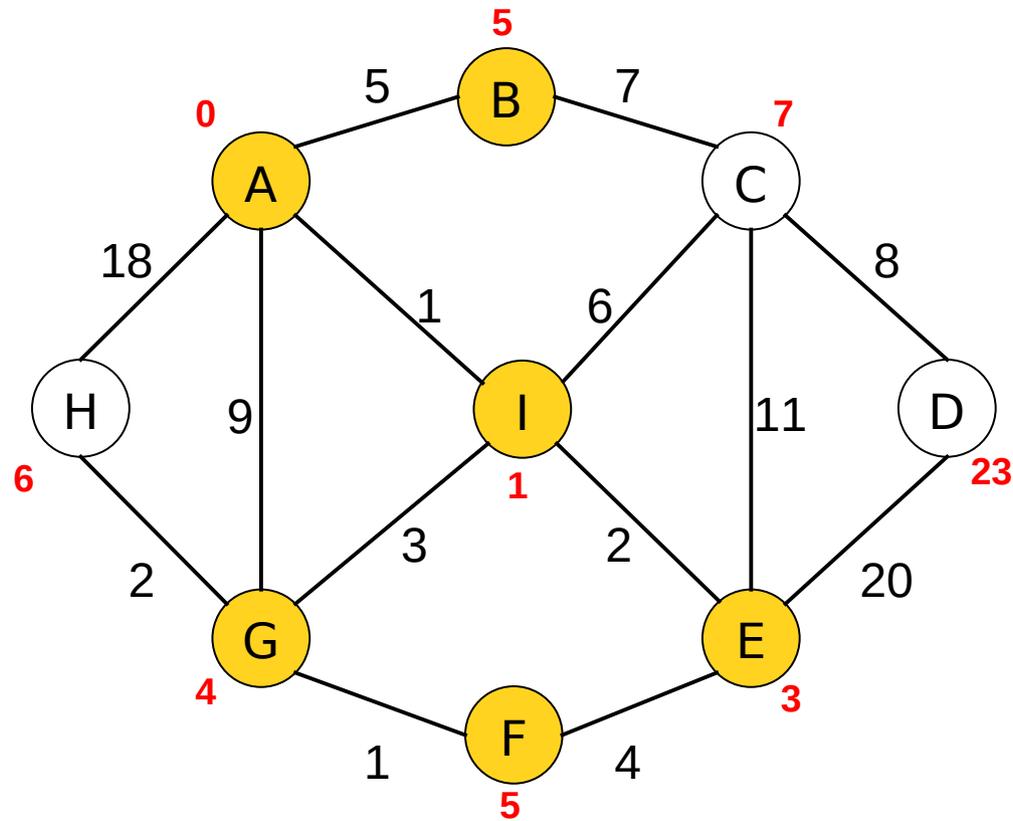
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

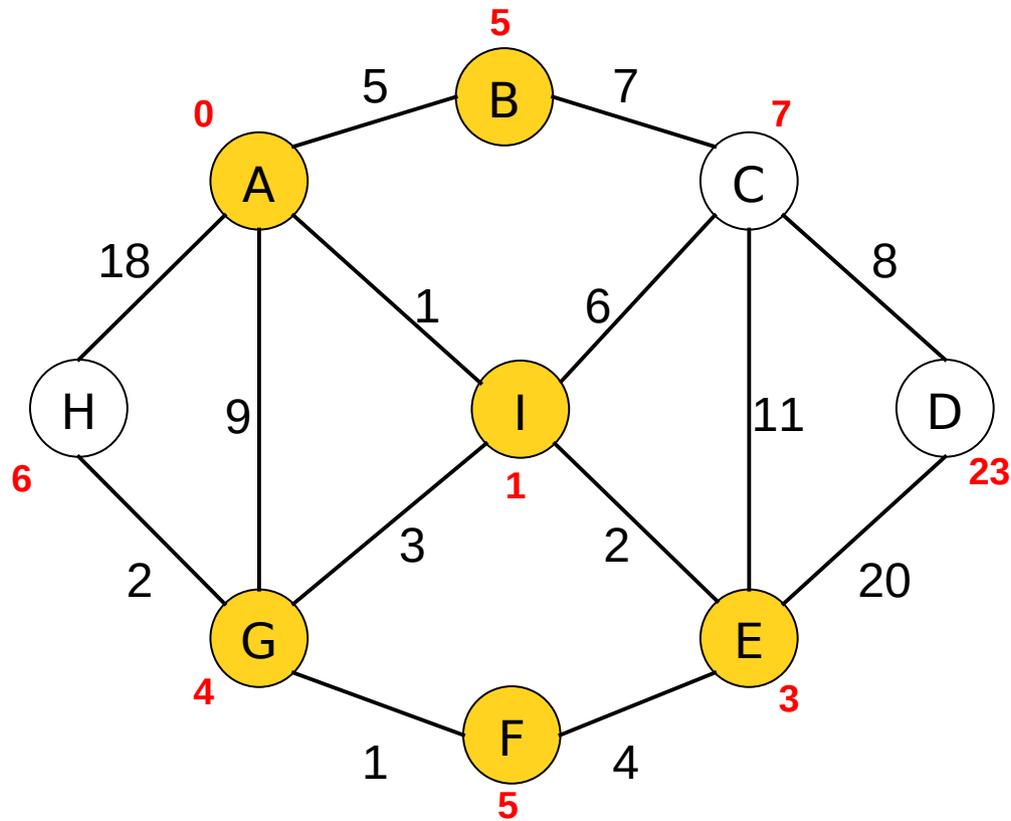


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



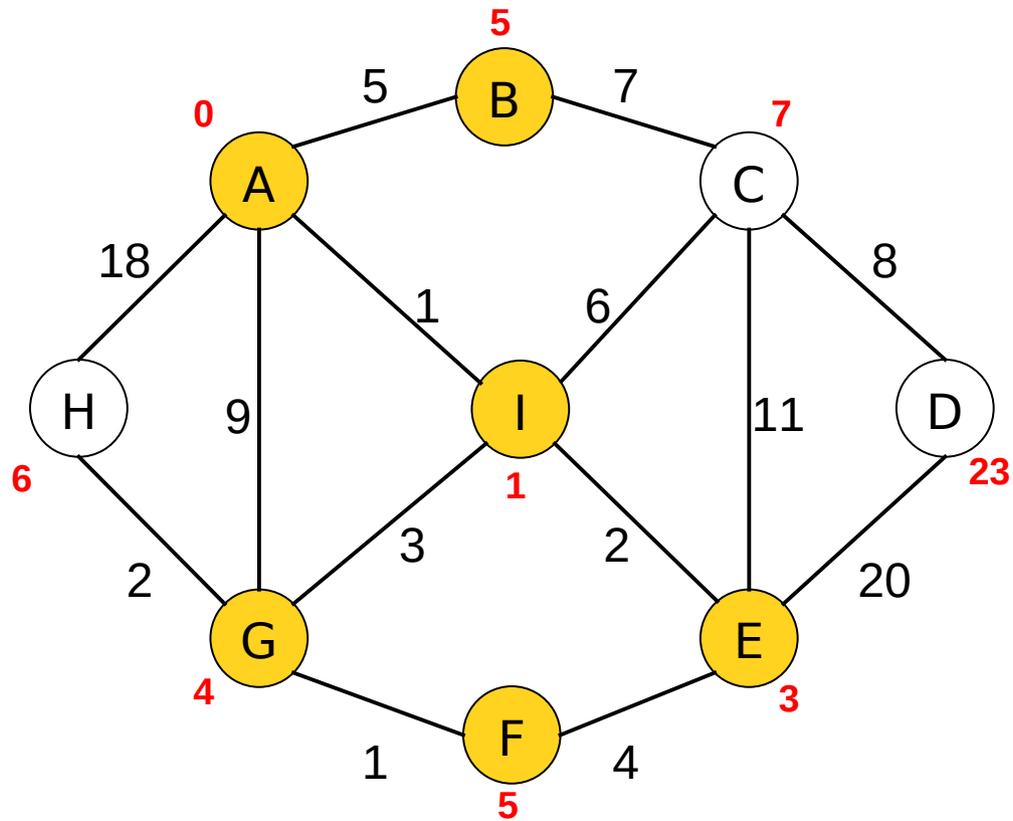
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I, E, G, B, and F to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

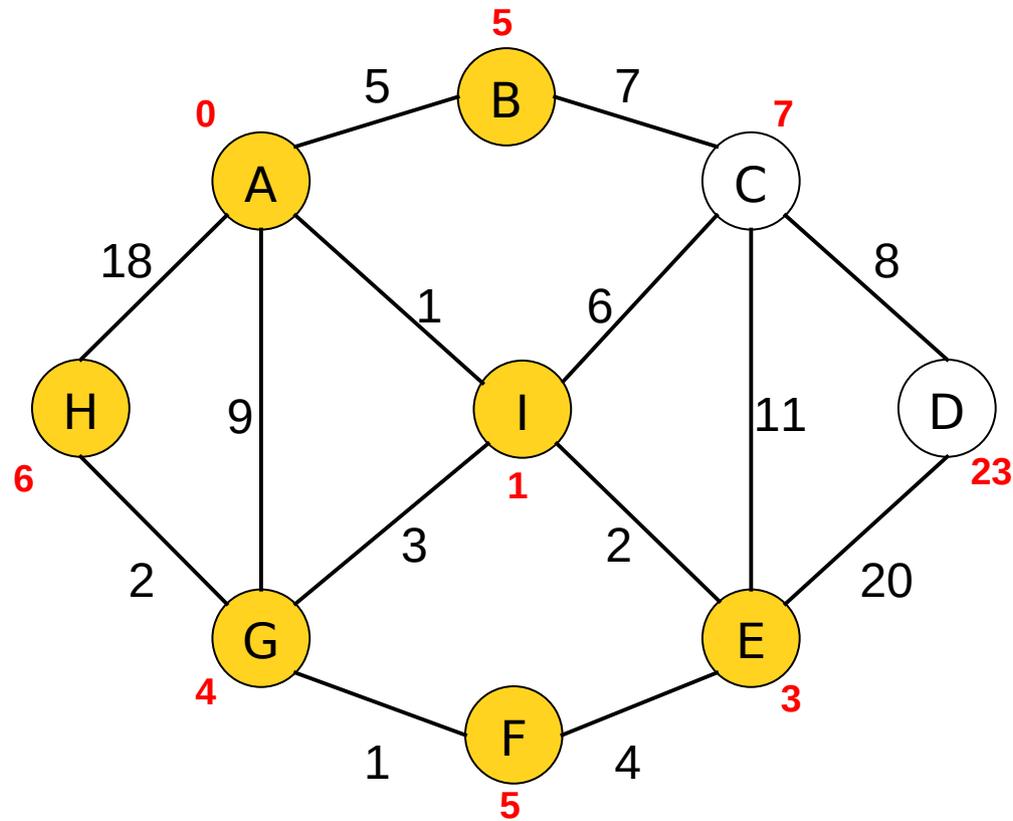
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

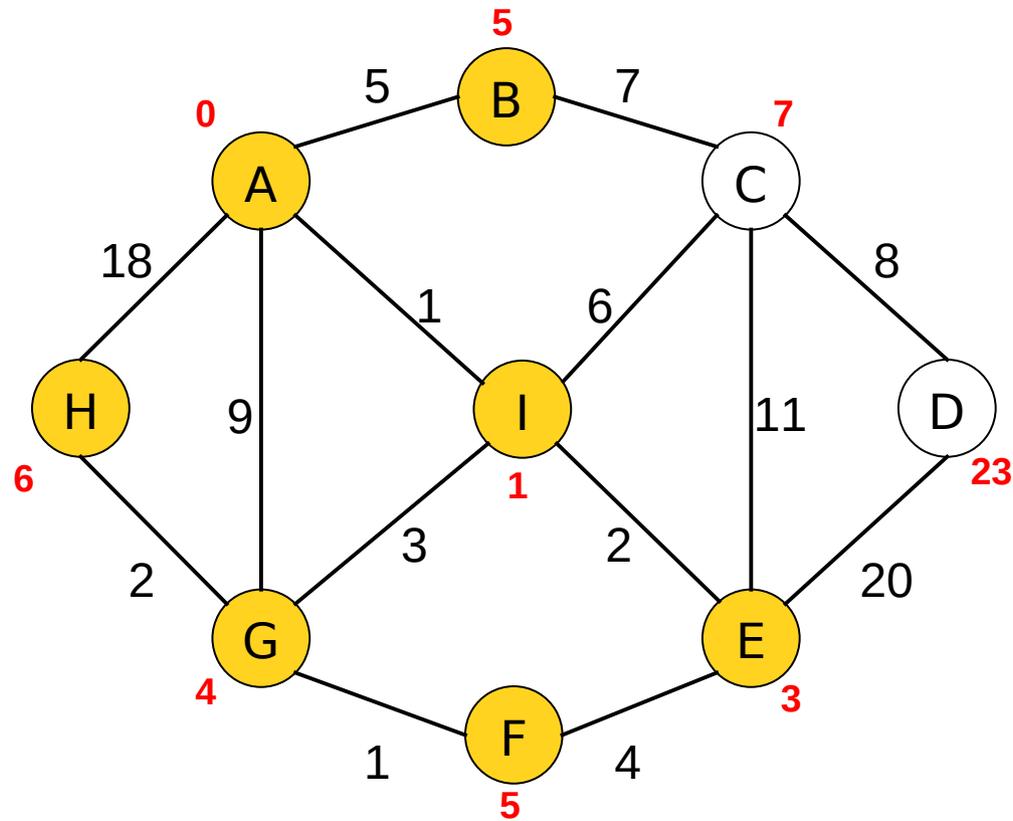
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



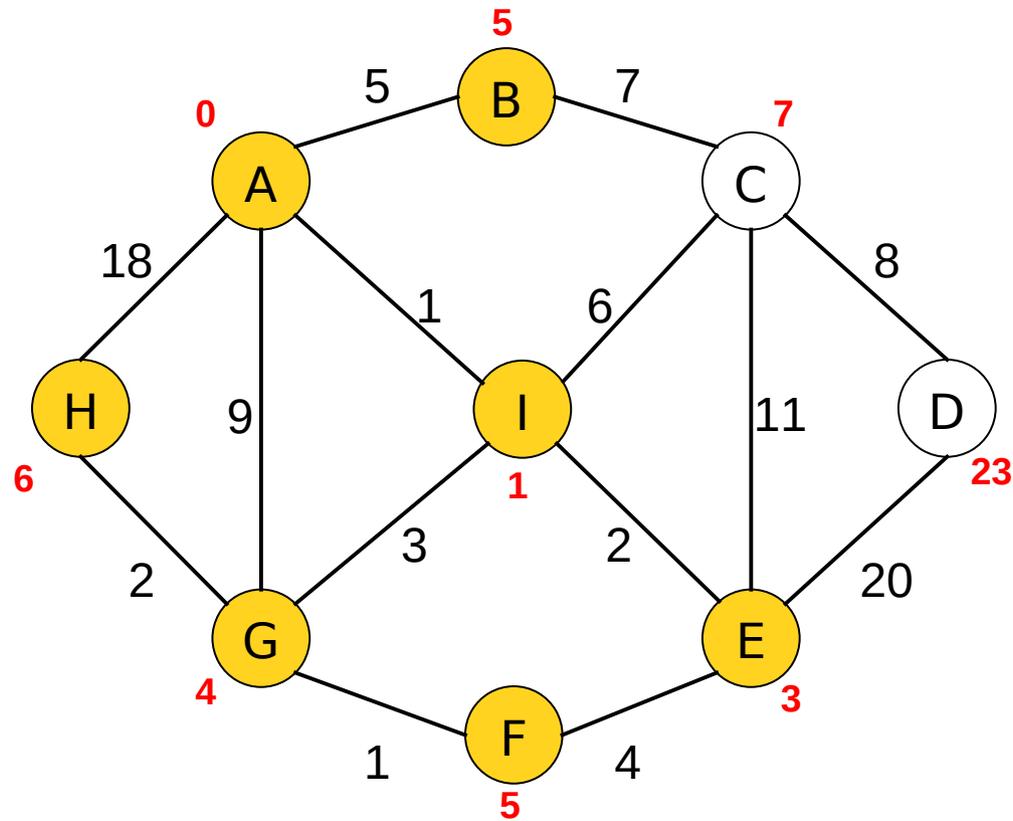
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I, E, G, B, F, and H to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

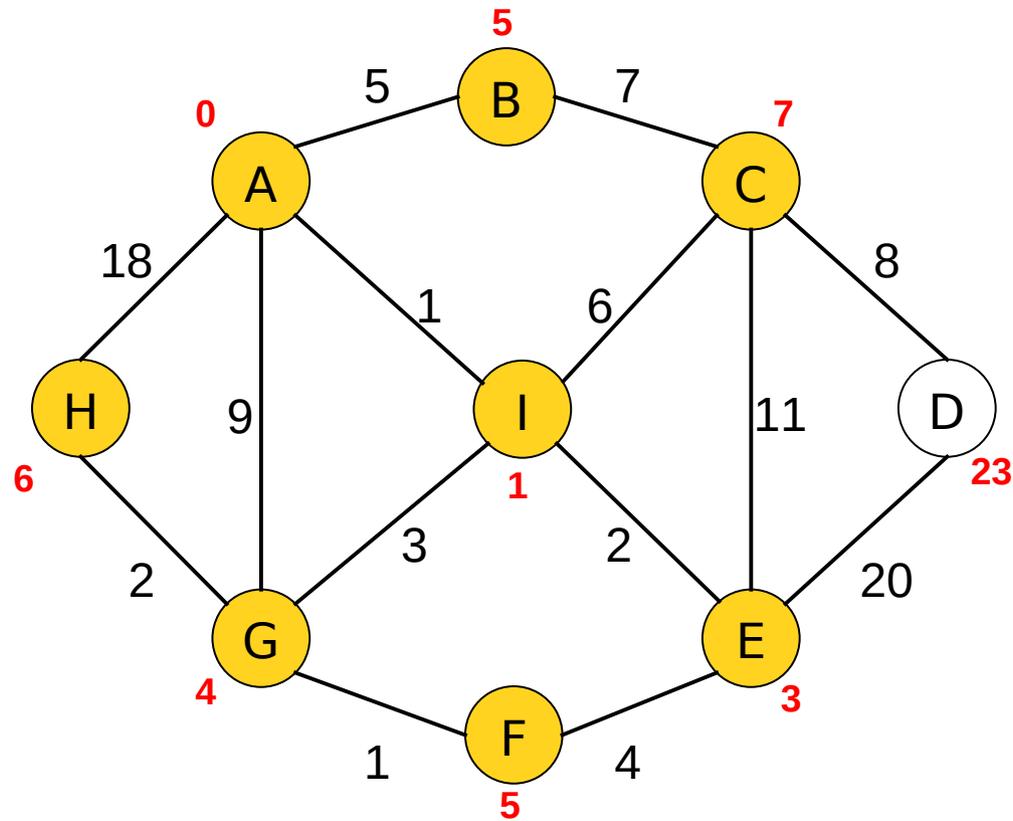
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

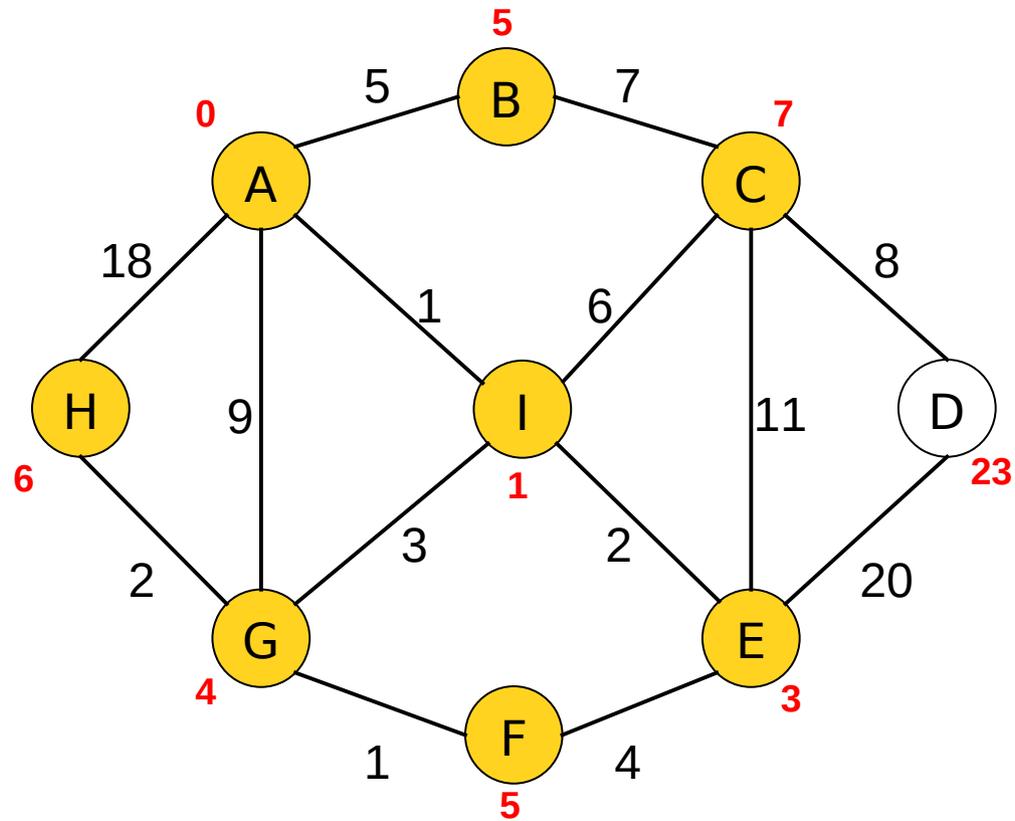
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

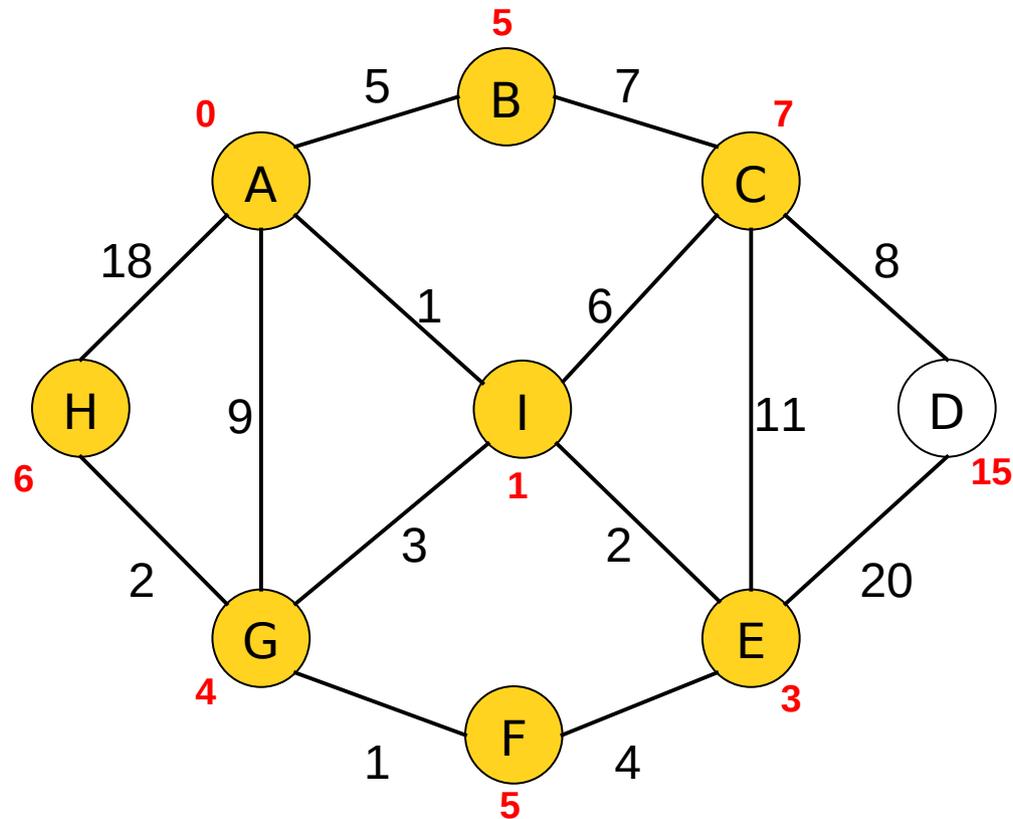


5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)



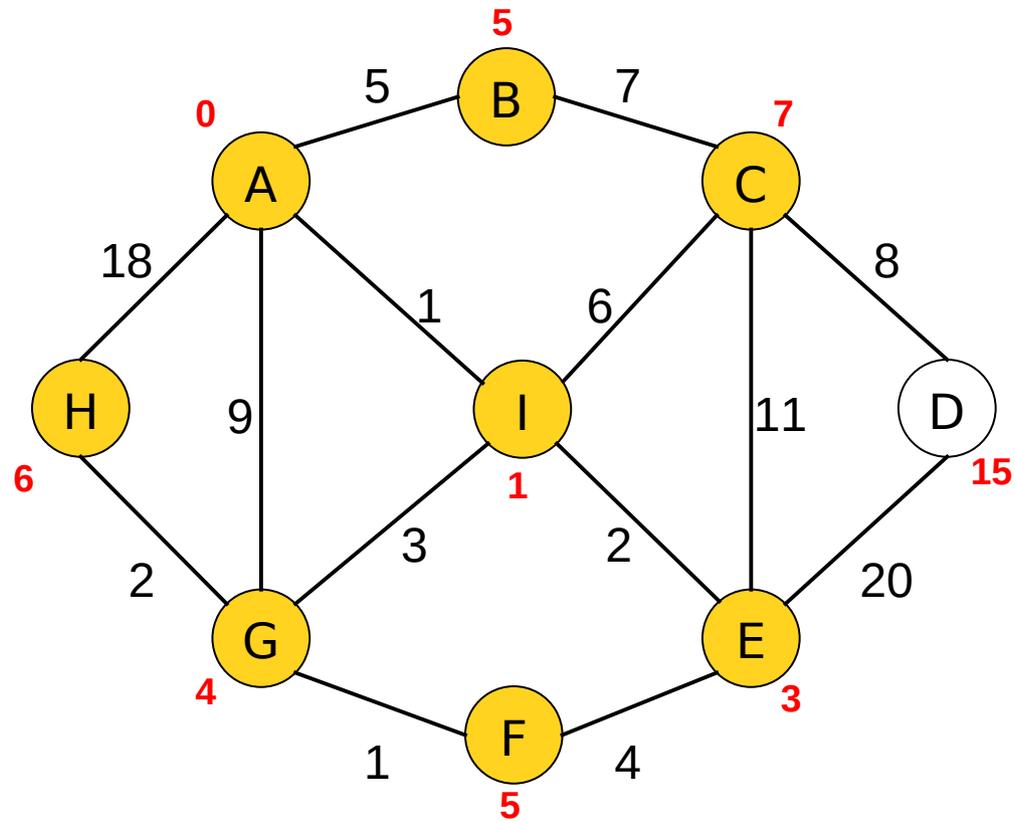
5: From that vertex,  $i$ , update the  $\text{dist}[j]$  values for all adjacent vertices,  $j$ :  
(again)

$$\text{MIN}\{\text{dist}[i] + \text{weight}[i, j], \text{dist}[j]\}$$

**Note:** The  $\text{dist}[i]$  values now indicate the lowest cost path from vertex A if we allow vertices I, E, G, B, F, H, and C to be used as intermediary vertices along the path.

# Dijkstra's Algorithm

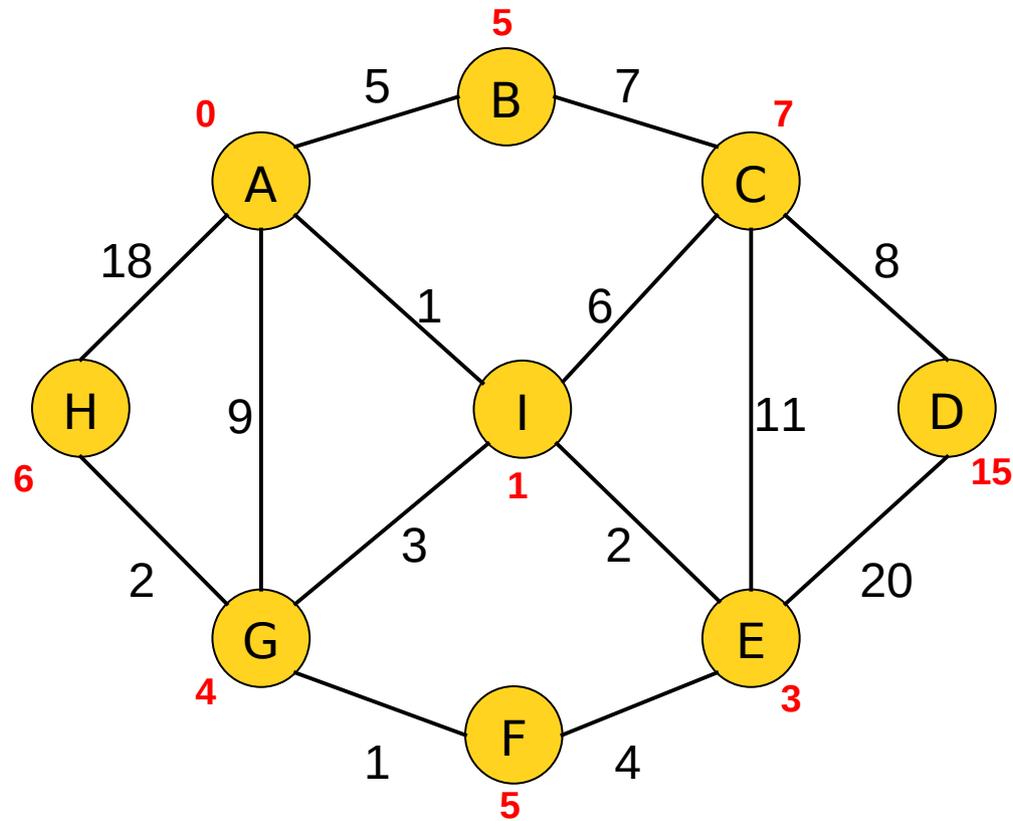
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

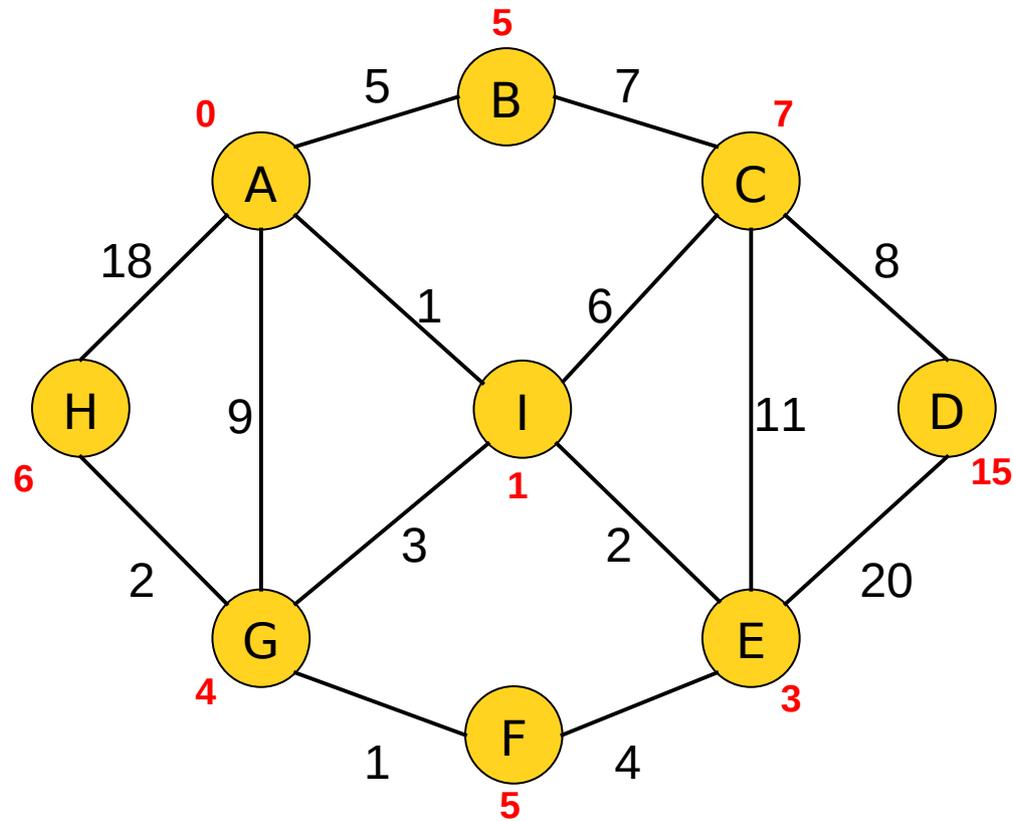
(calculating the cheapest path from a source vertex to all other vertices)



4: Choose the unvisited vertex with the smallest  $\text{dist}[i]$  value and visit it.  
(again)

# Dijkstra's Algorithm

(calculating the cheapest path from a source vertex to all other vertices)

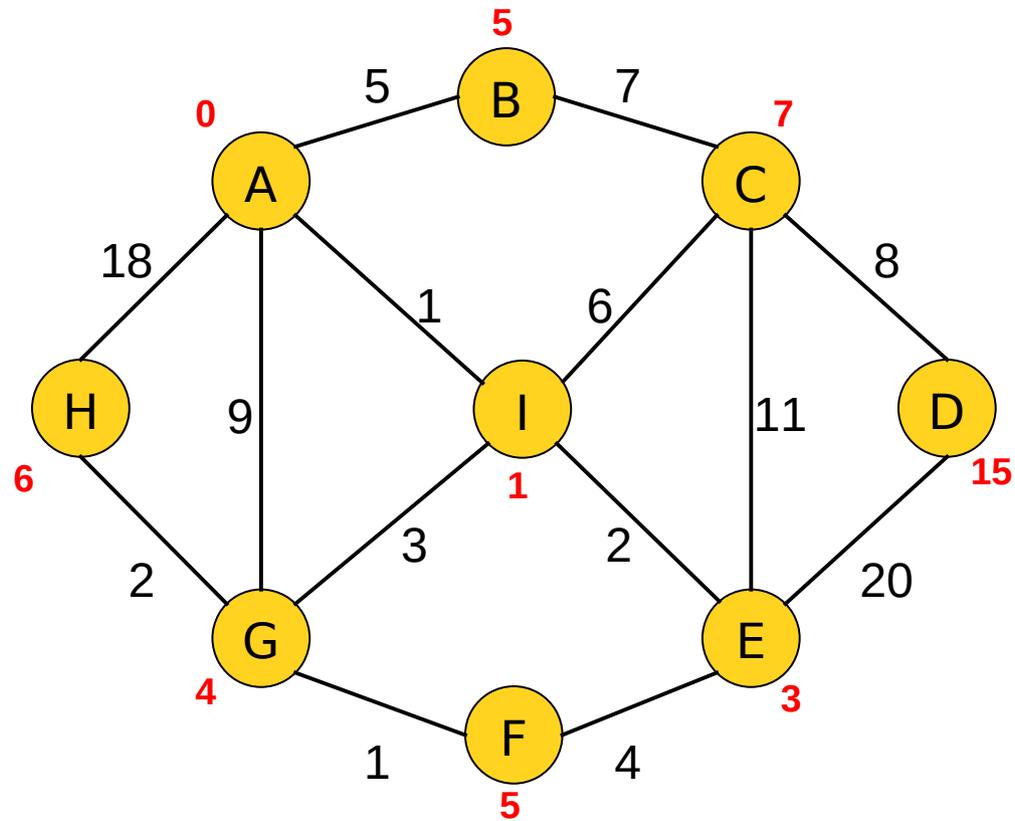


6: All vertices are visited, so terminate.

We now know the lowest cost to get from vertex A to each other vertex in the graph!

# An idea for path recovery

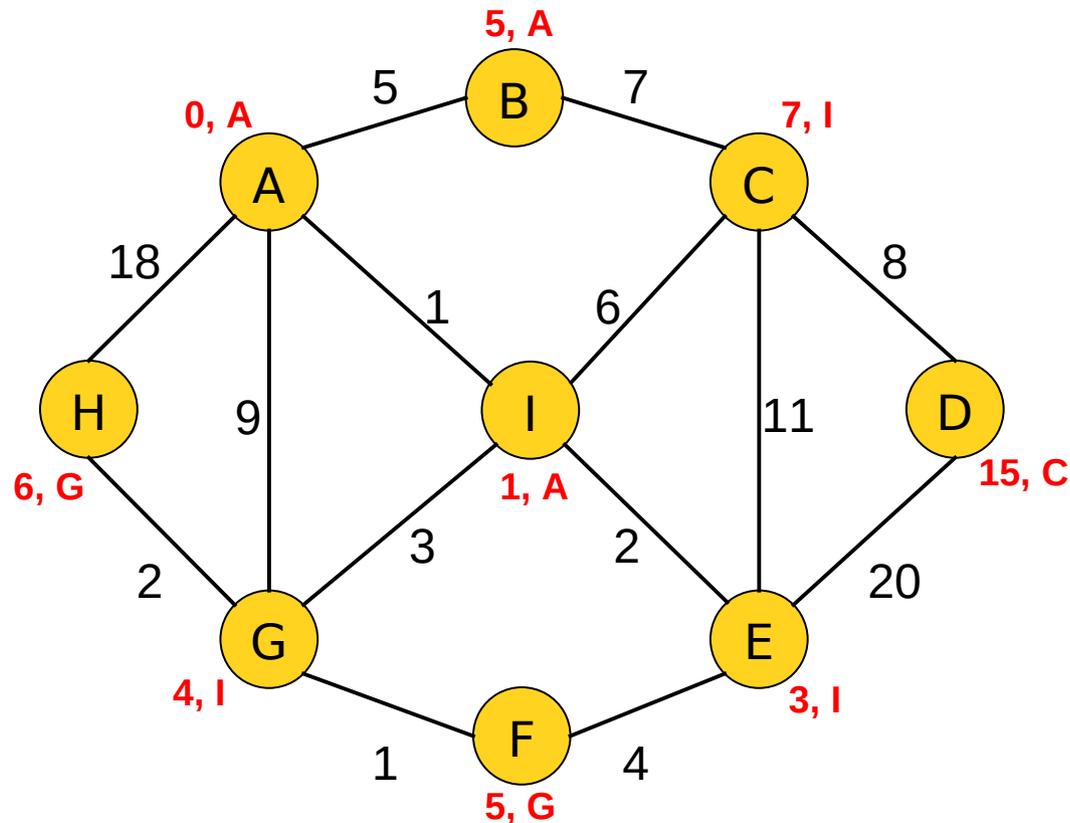
(with Dijkstra's Algorithm)



Idea: Keep track of the vertex we take every time we update a  $\text{dist}[i]$  value.

# An idea for path recovery

(with Dijkstra's Algorithm)



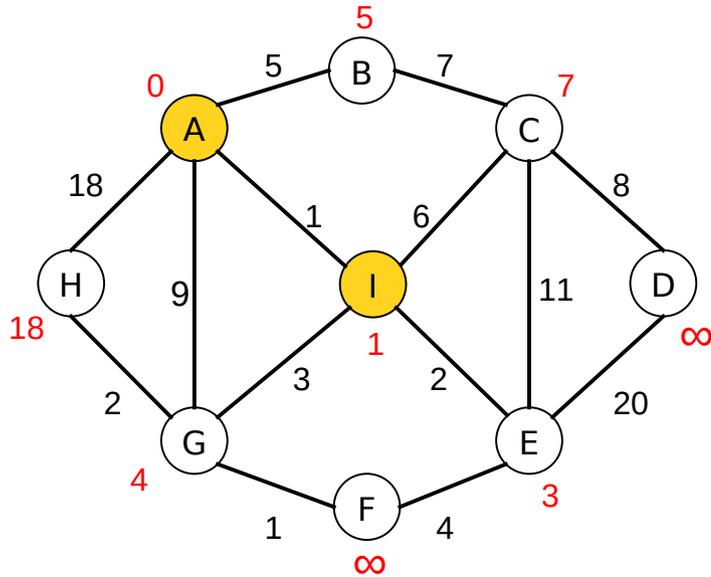
Idea: Keep track of the vertex we take every time we update a  $\text{dist}[i]$  value.

Now follow the vertices backward to the source to reconstruct the path.

For example, the path to D is  $D \leftarrow C \leftarrow I \leftarrow A$  (aka  $A \rightarrow I \rightarrow C \rightarrow D$ )

# The Algorithm

(with Dijkstra's Algorithm)



Initialize  $\text{dist}[i]$  to  $\infty$ .

Initialize  $\text{dist}[\text{source}]$  to 0.

while there are unvisited vertices:

Find the unvisited vertex with minimum  $\text{dist}[i]$  value  
Visit that vertex.

Update  $\text{dist}[i]$  for its unvisited neighbors.

We might want to halt if the minimum  $\text{dist}[i]$  value is  $\infty$ .

What is the runtime?