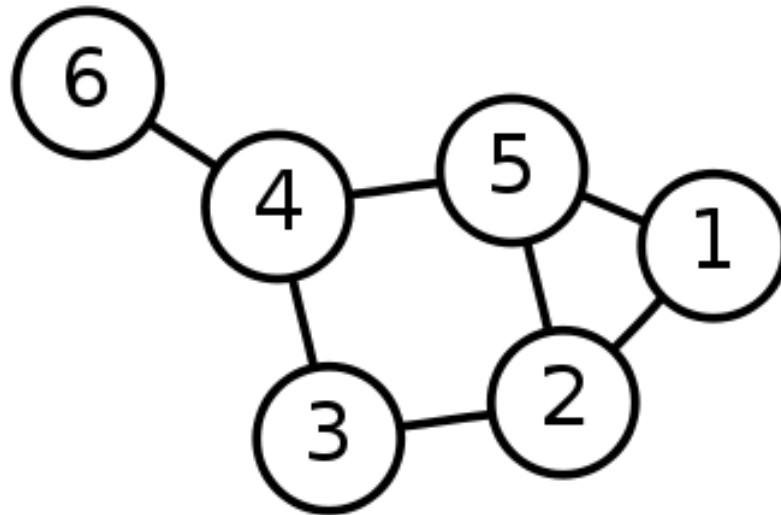


Single-Source Shortest Path Problem

The problem of finding shortest paths from a source vertex v to all other vertices in the graph.



Dijkstra's algorithm

a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs.
However, all edges must have nonnegative weights.

Approach: Greedy

Input: Weighted graph $G=\{E,V\}$ and source vertex $v \in V$,
such that all edge weights are nonnegative

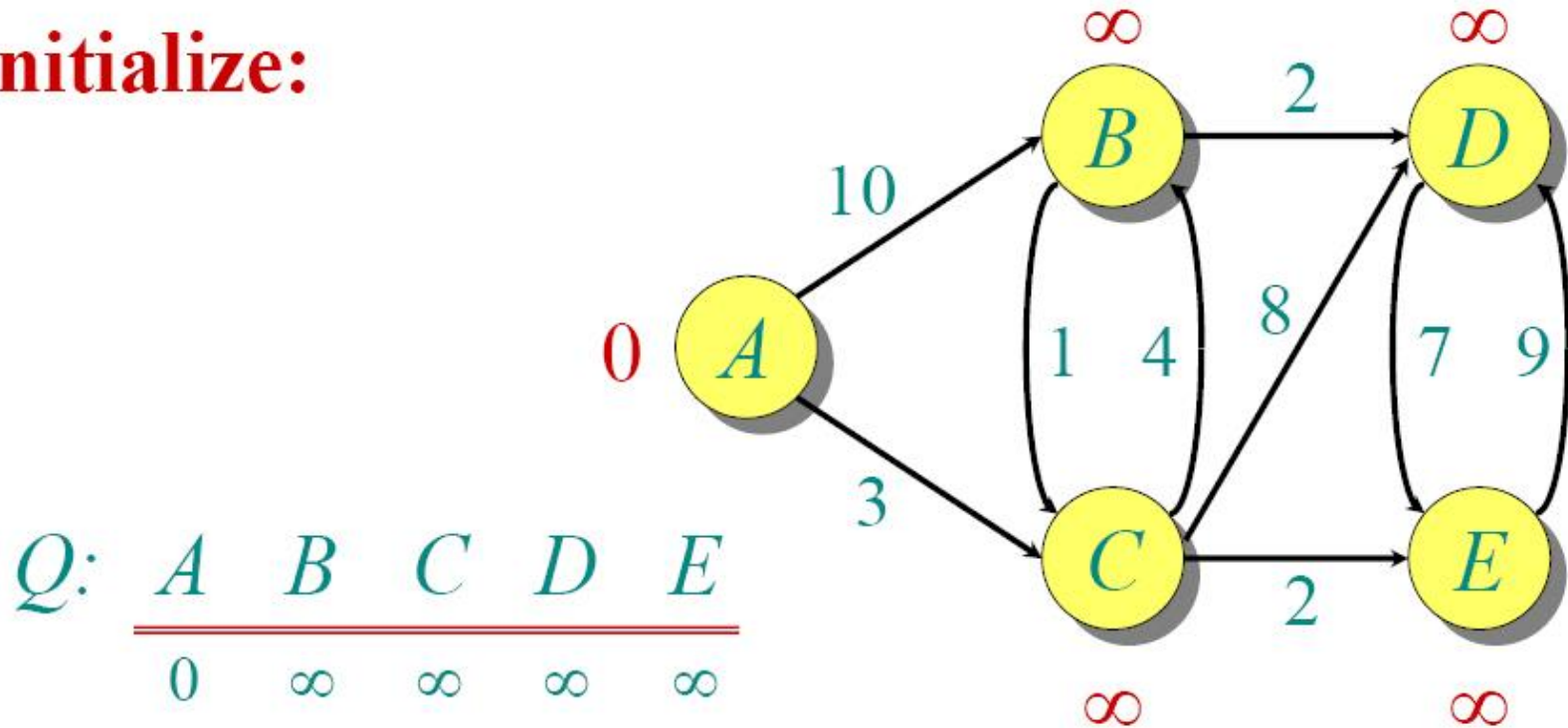
Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

Dijkstra's algorithm - Pseudocode

```
dist[s] ← 0                                (distance to source vertex is zero)
for all v ∈ V - {s}
    do dist[v] ← ∞                          (set all other distances to infinity)
S ← ∅                                        (S, the set of visited vertices is initially empty)
Q ← V                                        (Q, the queue initially contains all vertices)
while Q ≠ ∅                                (while the queue is not empty)
do u ← mindistance(Q, dist)                 (select the element of Q with the min. distance)
    S ← S ∪ {u}                             (add u to list of visited vertices)
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u, v)    (if new shortest path found)
            then d[v] ← d[u] + w(u, v)      (set new value of shortest path)
            (if desired, add traceback code)
return dist
```

Dijkstra Animated Example

Initialize:

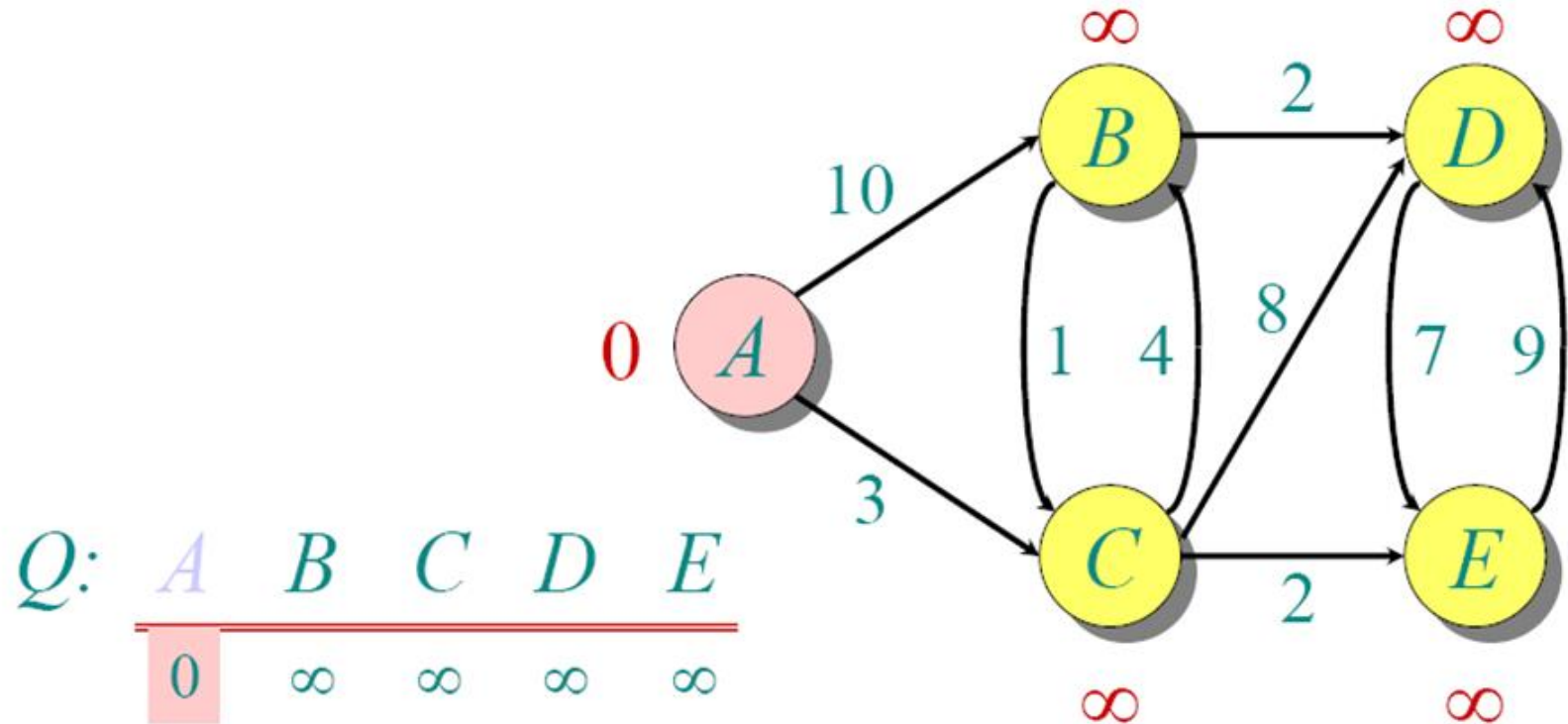


$Q:$

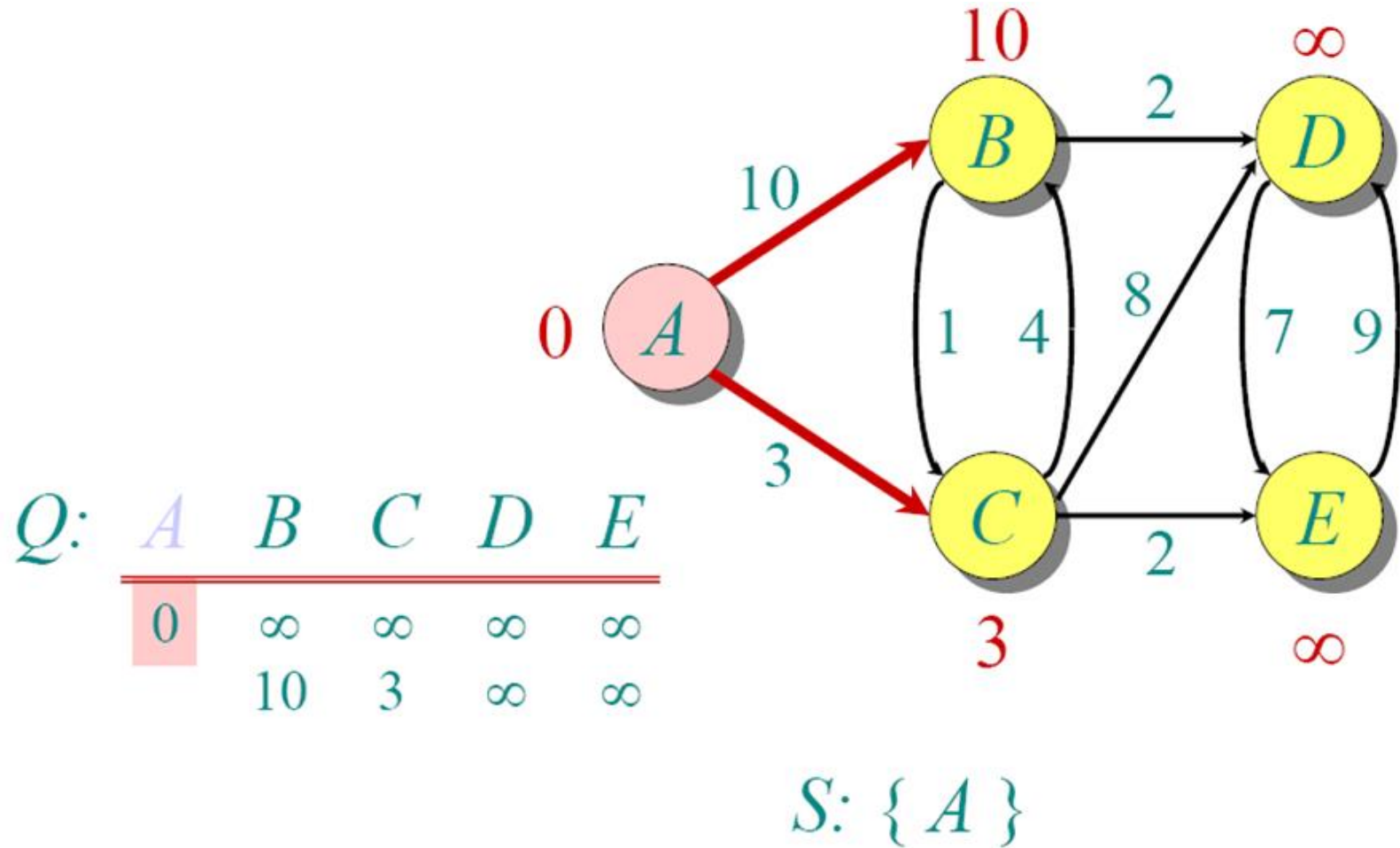
A	B	C	D	E
0	∞	∞	∞	∞

$S: \{\}$

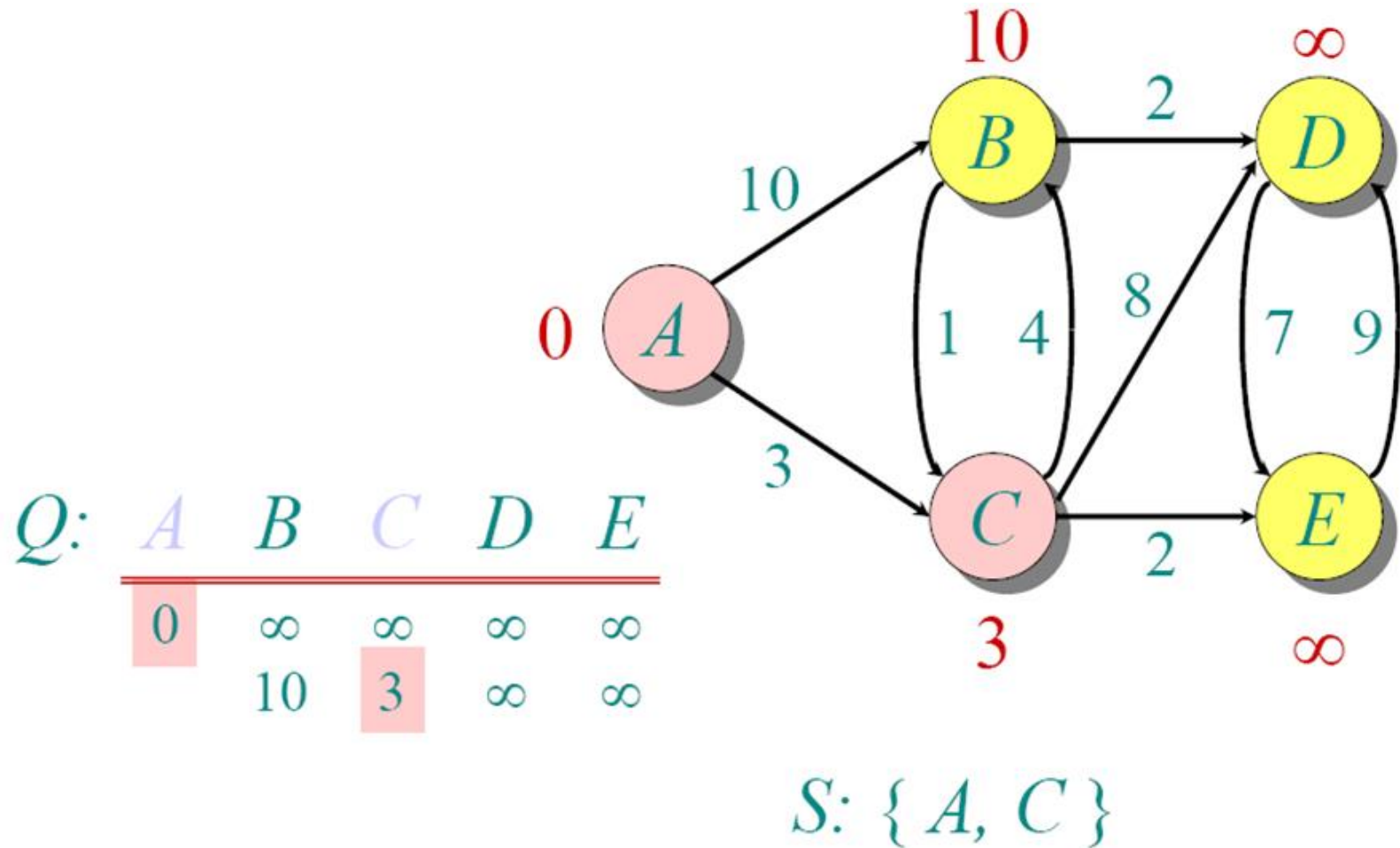
Dijkstra Animated Example



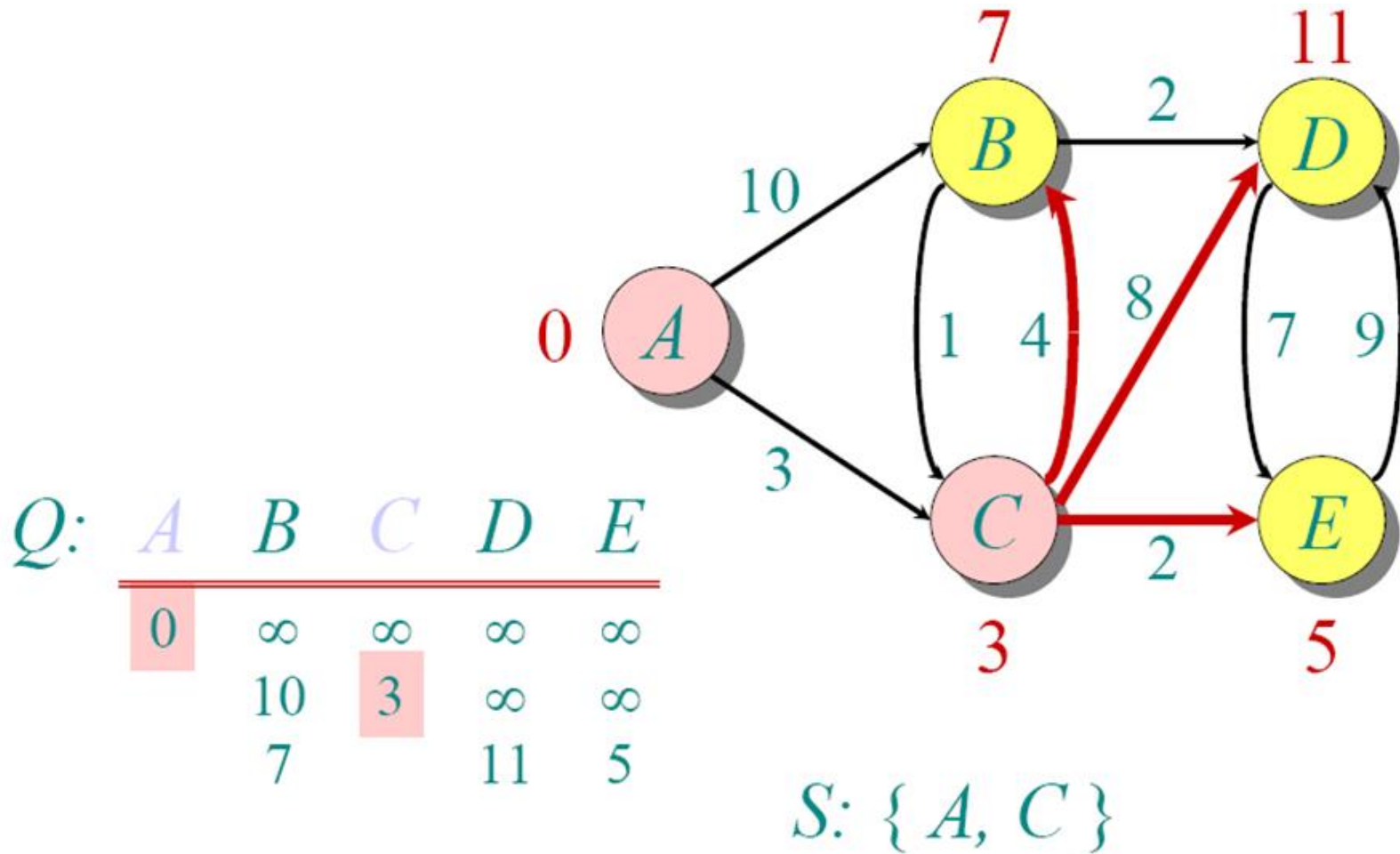
Dijkstra Animated Example



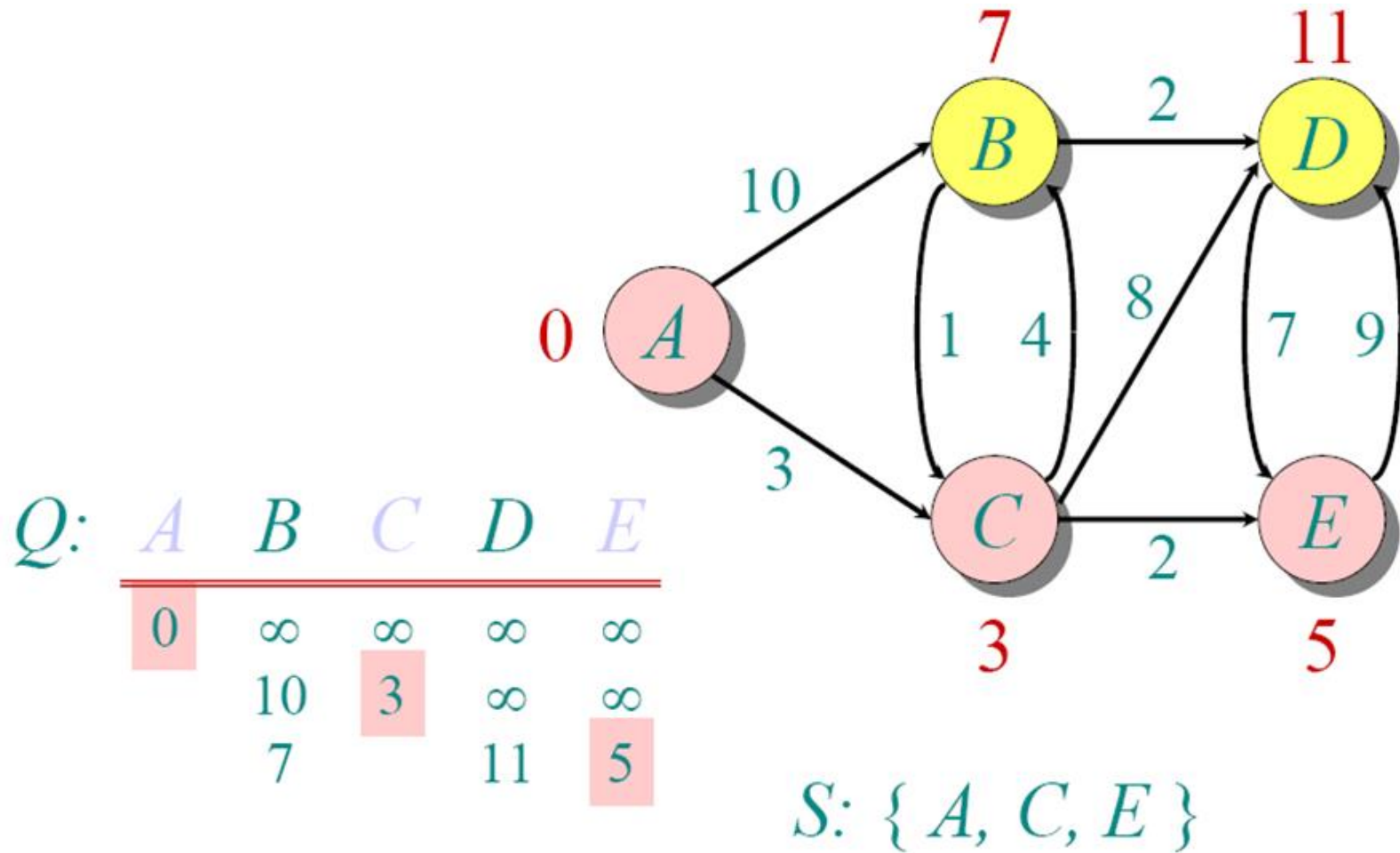
Dijkstra Animated Example



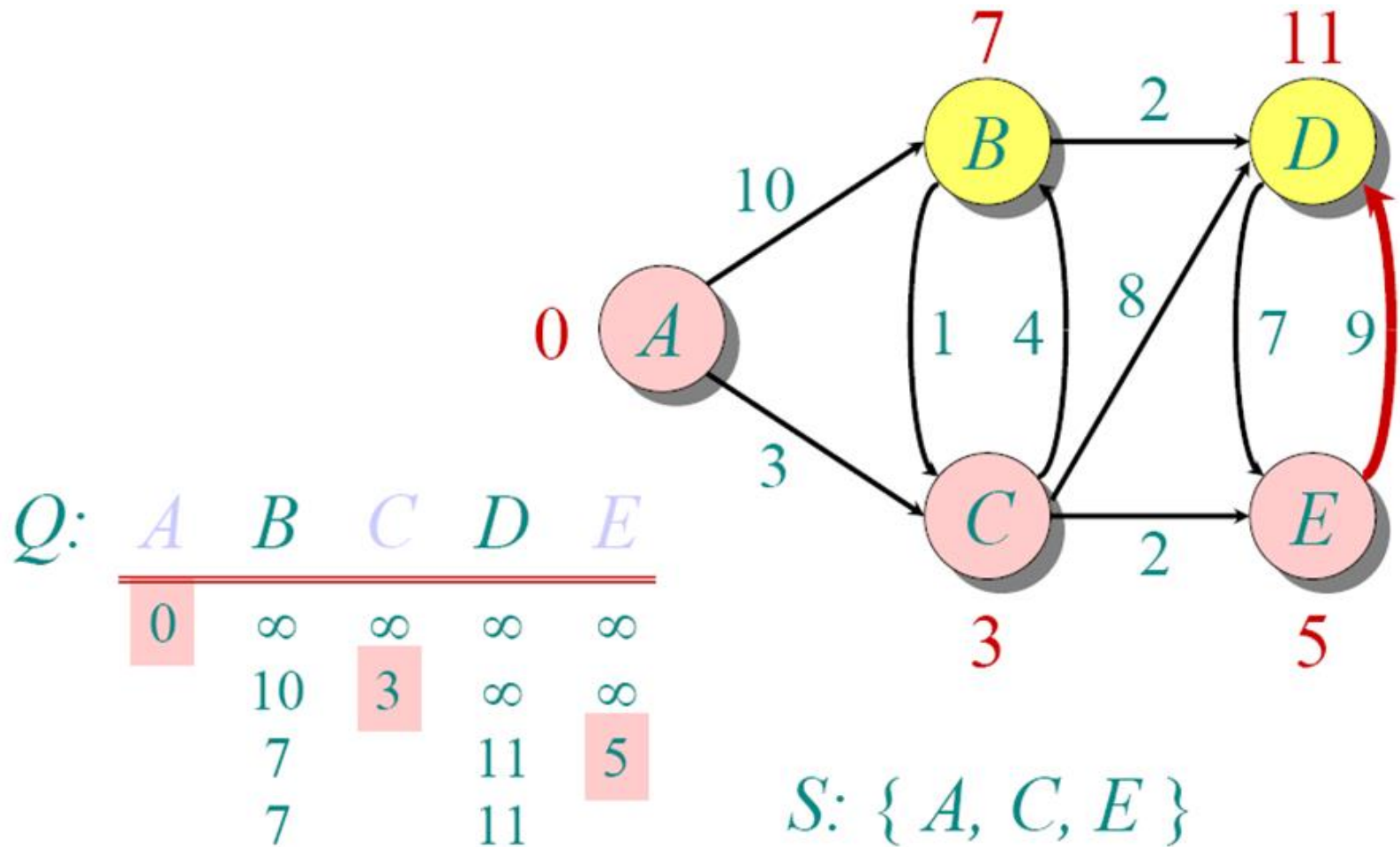
Dijkstra Animated Example



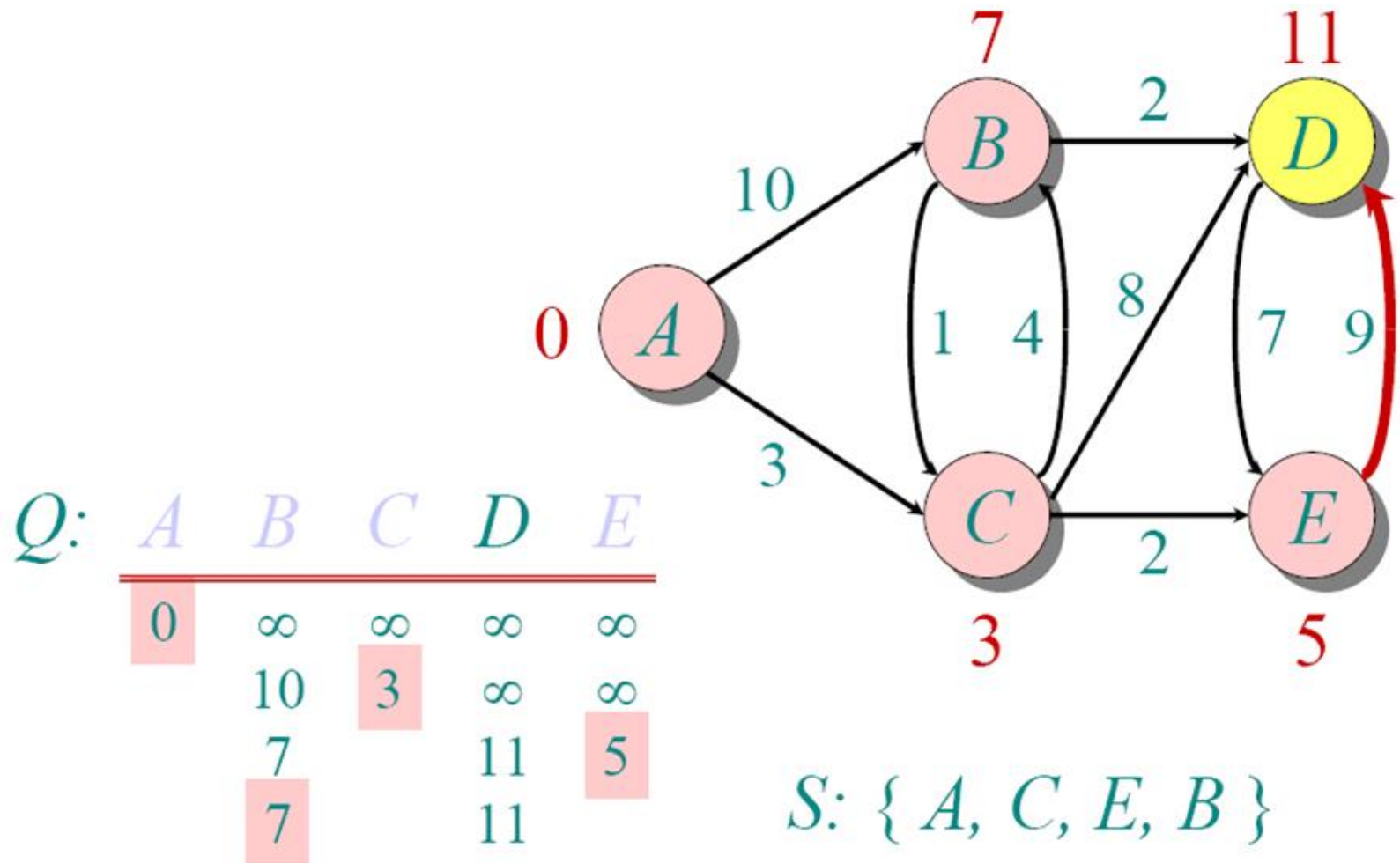
Dijkstra Animated Example



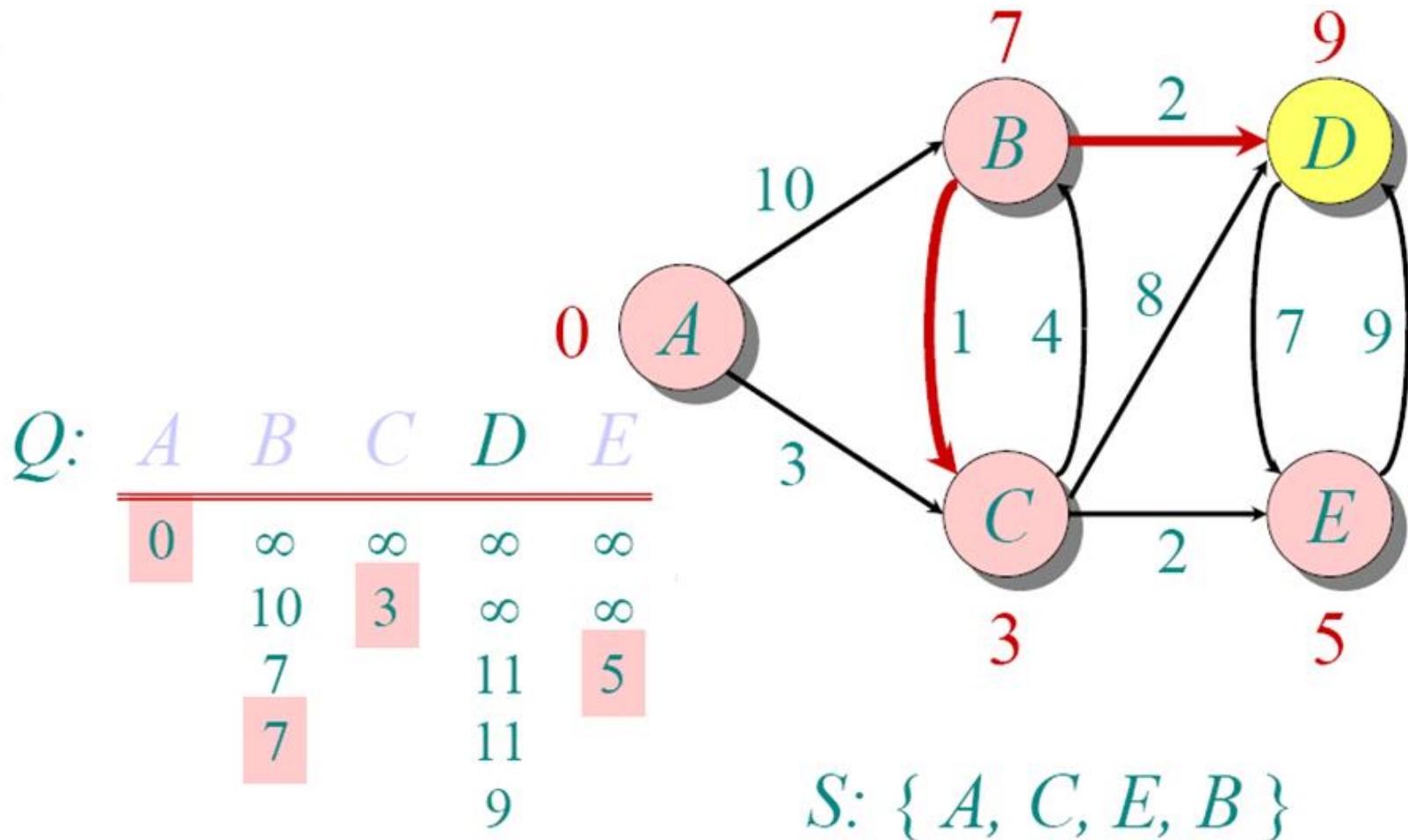
Dijkstra Animated Example



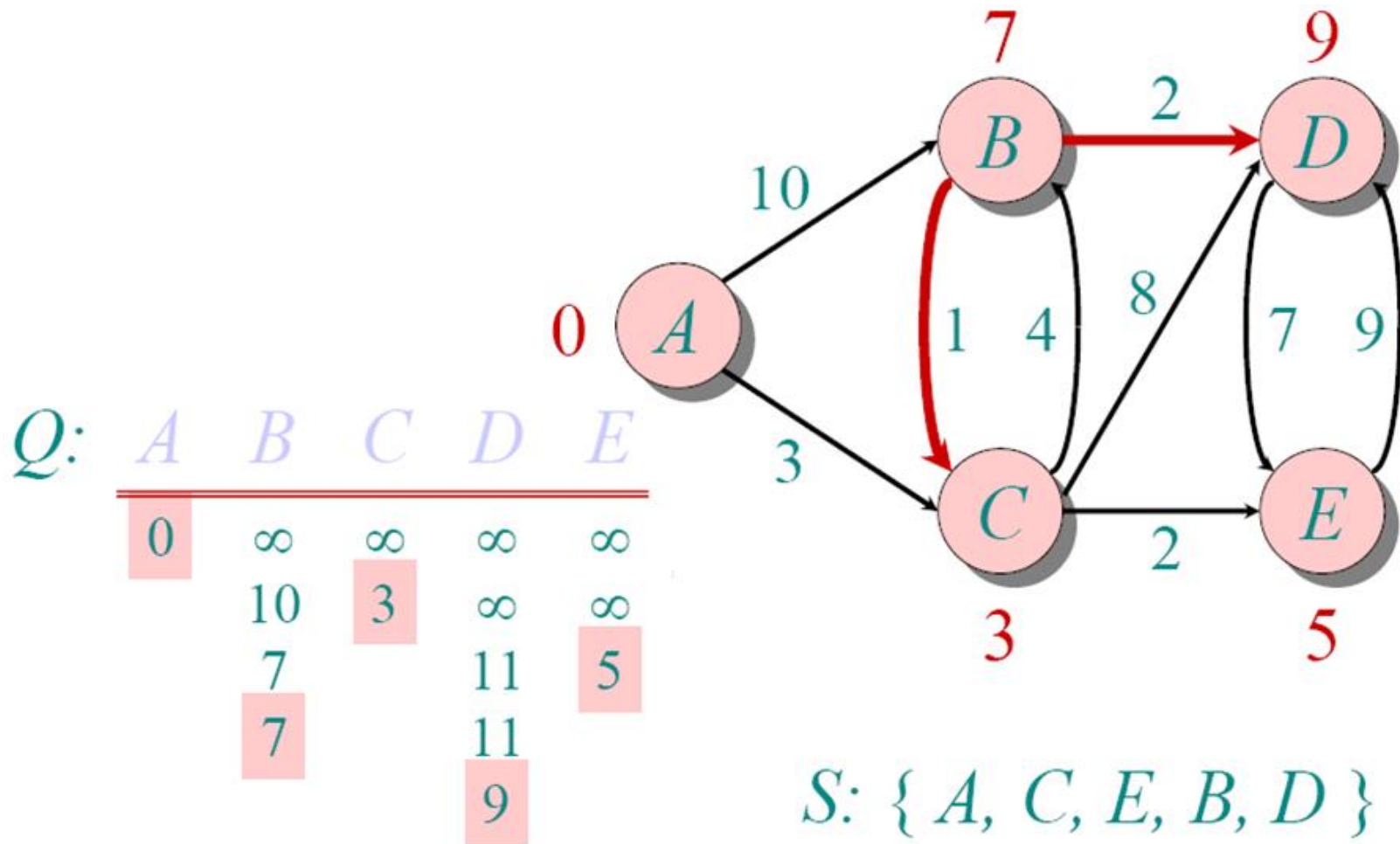
Dijkstra Animated Example



Dijkstra Animated Example



Dijkstra Animated Example



Implementations and Running Times

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of

$$O(|V|^2 + |E|)$$

For sparse graphs, or graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a binary heap or priority queue. This will produce a running time of

$$O((|E| + |V|) \log |V|)$$

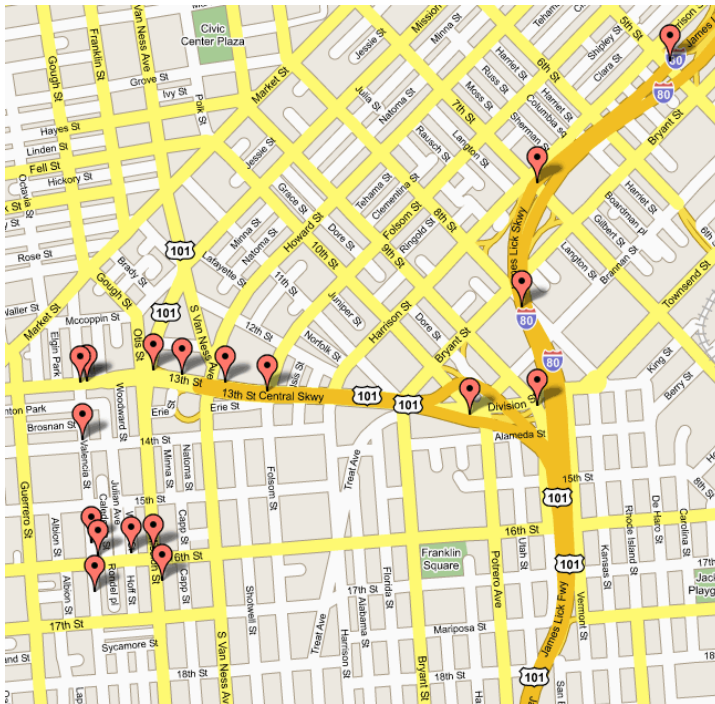
• DIJKSTRA'S ALGORITHM - WHY USE IT?

- As mentioned, Dijkstra's algorithm calculates the shortest path to every vertex.
- However, it is about as computationally **expensive** to calculate the shortest path from vertex u to every vertex using Dijkstra's as it is to calculate the shortest path to some particular vertex v .
- Therefore, anytime we want to know the optimal path to some other vertex from a determined origin, we can use Dijkstra's algorithm.

Applications of Dijkstra's Algorithm

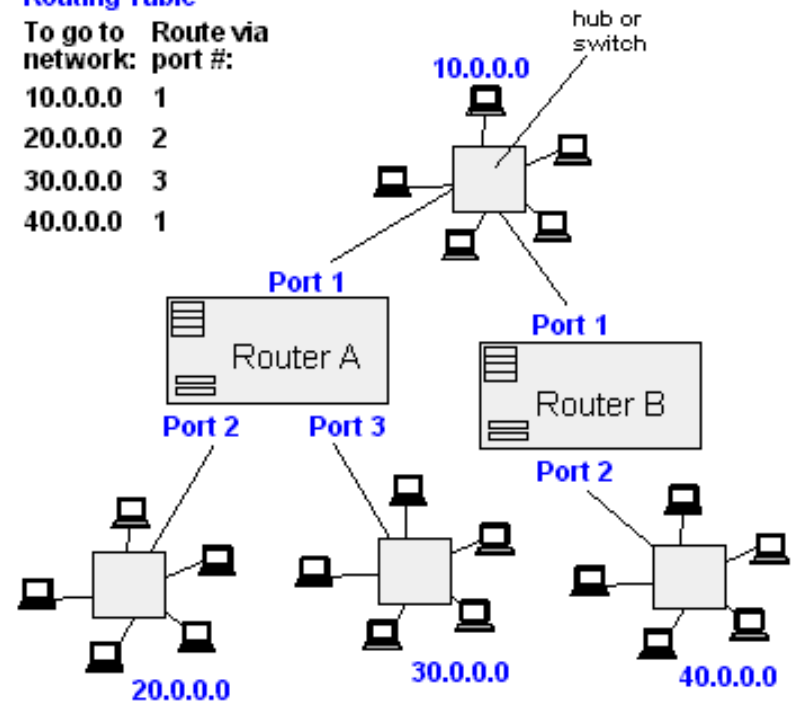
- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



Router A Routing Table

To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1



Exercise

- Implement **Dijkstra's algorithm**

- **Inputfile**

บรรทัดแรก เป็น จำนวน **node**, จำนวน **edge**, **source**

บรรทัดต่อๆ มาคือ **edge** ต่าง ๆ ประกอบด้วยข้อมูล

nodeต้นทาง **node**ปลายทาง ระยะทาง

- **Output** ให้พิมพ์ระยะทางจาก **Source** ไปยัง **node** ต่าง ๆ

A Minimum Spanning Tree (MST)

- a subgraph of an undirected graph such that
 - the subgraph spans (includes) all nodes,
 - is connected,
 - is acyclic,
 - and has minimum total edge weight

Minimum Spanning Trees

Prim's Algorithm

- Similar to Dijkstra's Algorithm

Kruskal's Algorithm

- Focuses on edges, rather than nodes

Algorithm Characteristics

- Both **Prim**'s and **Kruskal**'s Algorithms work with undirected graphs
- Both work with weighted and unweighted graphs but are more interesting when edges are weighted
- Both are greedy algorithms that produce optimal solutions

Prim's Algorithm

- Similar to **Dijkstra's** Algorithm except that d_v records edge weights, not path lengths

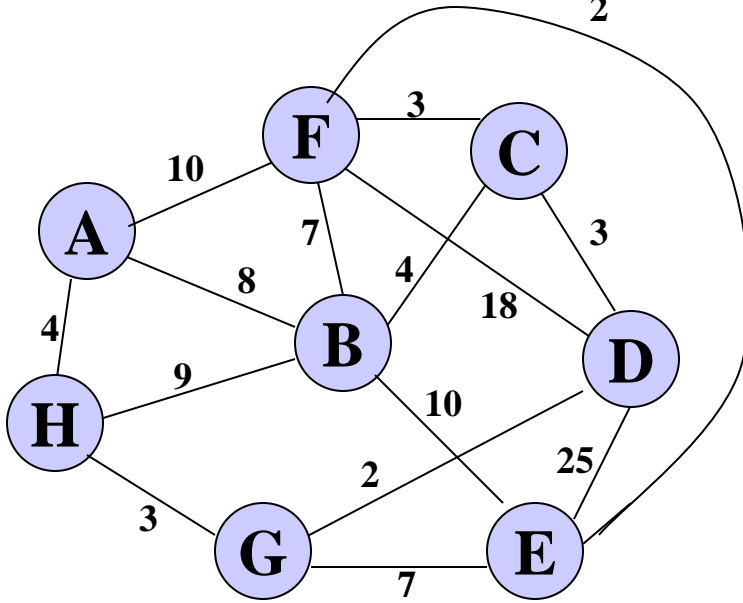
Prim's algorithm

```
{  T =  $\phi$ ;  
  U = { 1 };  
  while ( $U \neq V$ ) {  
    let ( $u, v$ ) be the lowest cost edge  
    such that  $u \in U$  and  $v \in V - U$ ;  
     $T = T \cup \{(u, v)\}$   
     $U = U \cup \{v\}$   
  }  
}
```

From <http://lcm.csa.iisc.ernet.in/dsa/node183.html>

Walk-Through₂

Initialize array

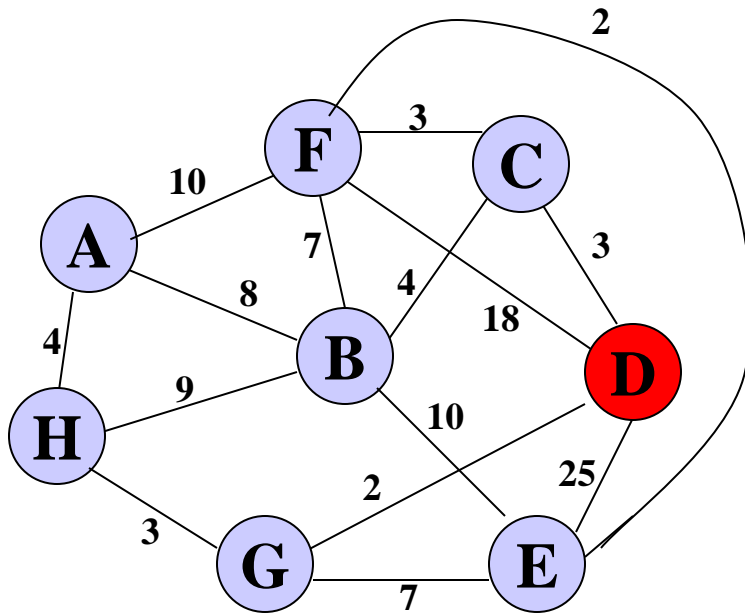


	K	d_v	p_v
A	F	∞	—
B	F	∞	—
C	F	∞	—
D	F	∞	—
E	F	∞	—
F	F	∞	—
G	F	∞	—
H	F	∞	—

d_v records edge weights

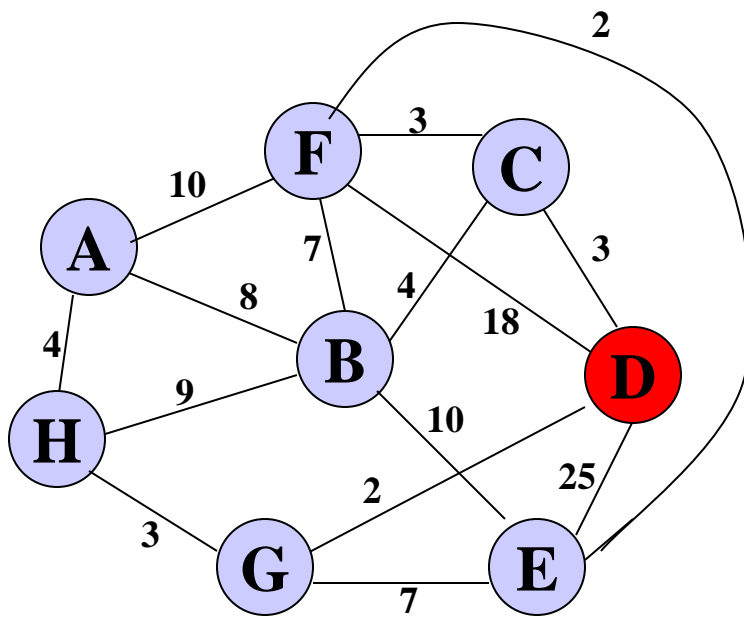
P_v : Parant node

K : visited



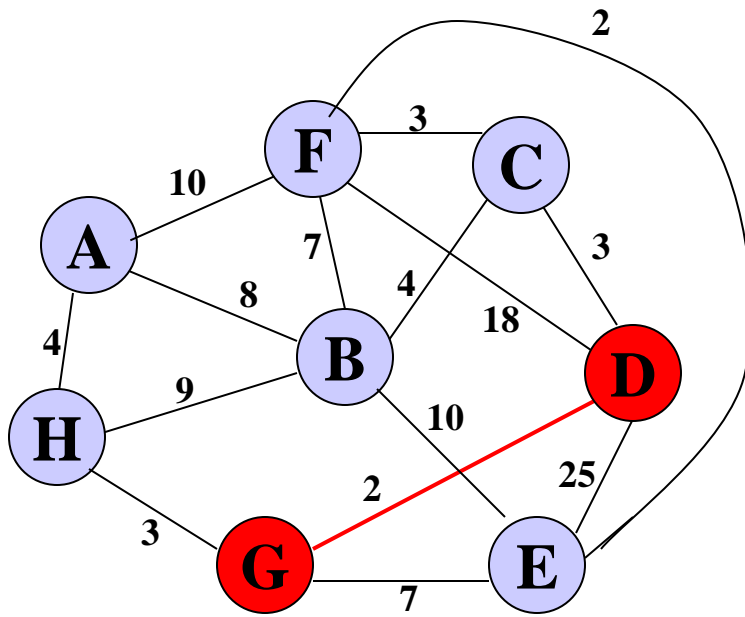
Start with any node, say D

	K	d_v	p_v
A			
B			
C			
D	T	0	—
E			
F			
G			
H			



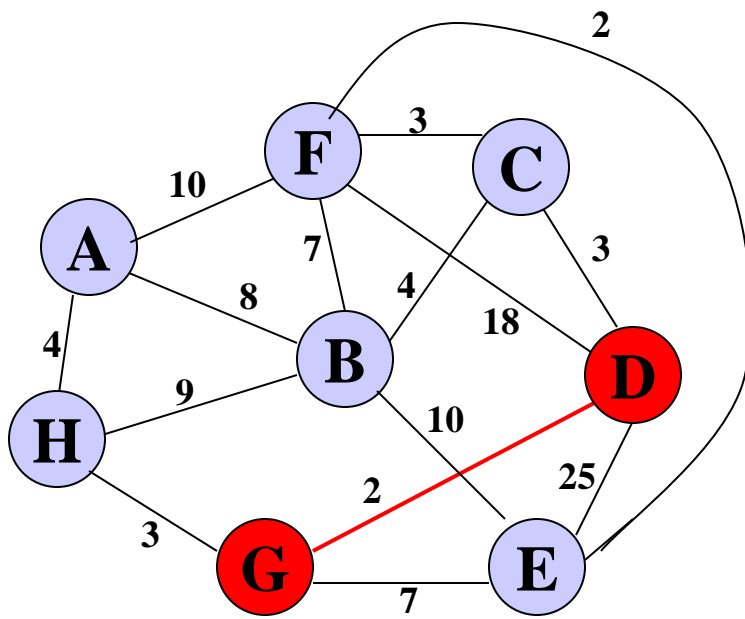
Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	—
E		25	D
F		18	D
G		2	D
H			



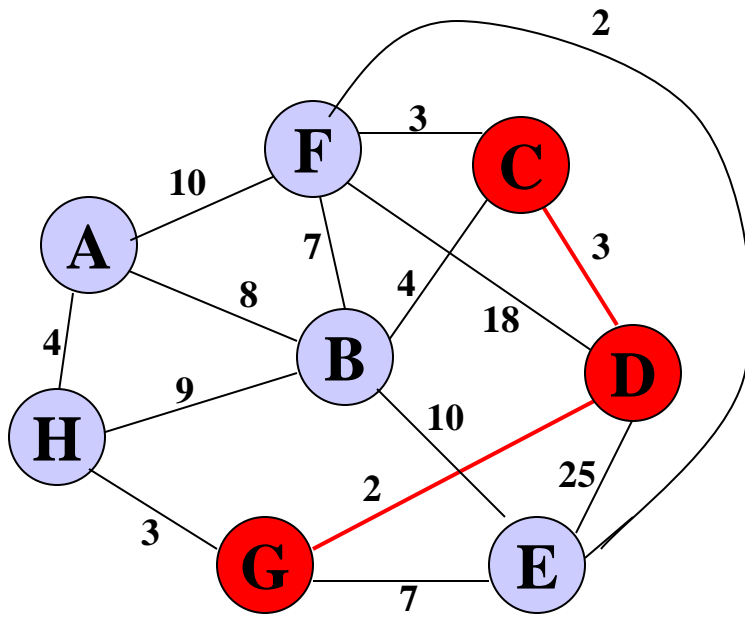
Select node with
minimum distance

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G	T	2	D
H			



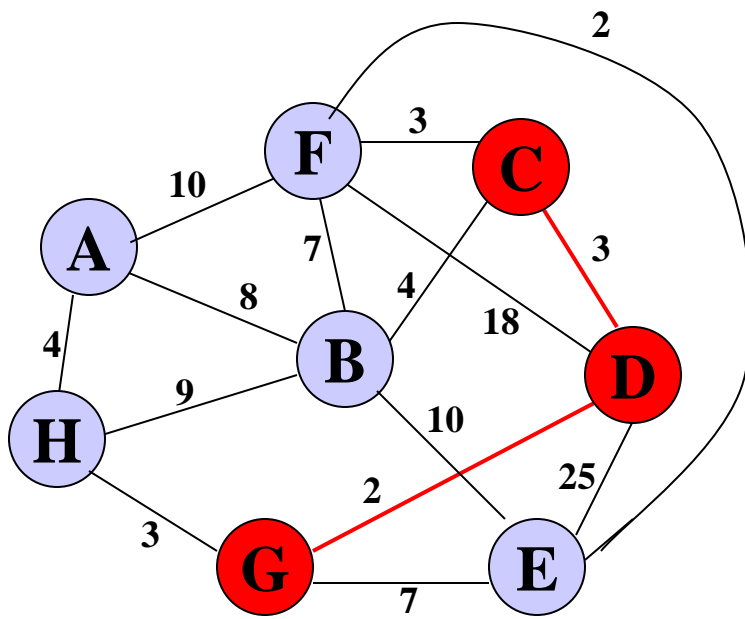
Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	—
E		7	G
F		18	D
G	T	2	D
H		3	G



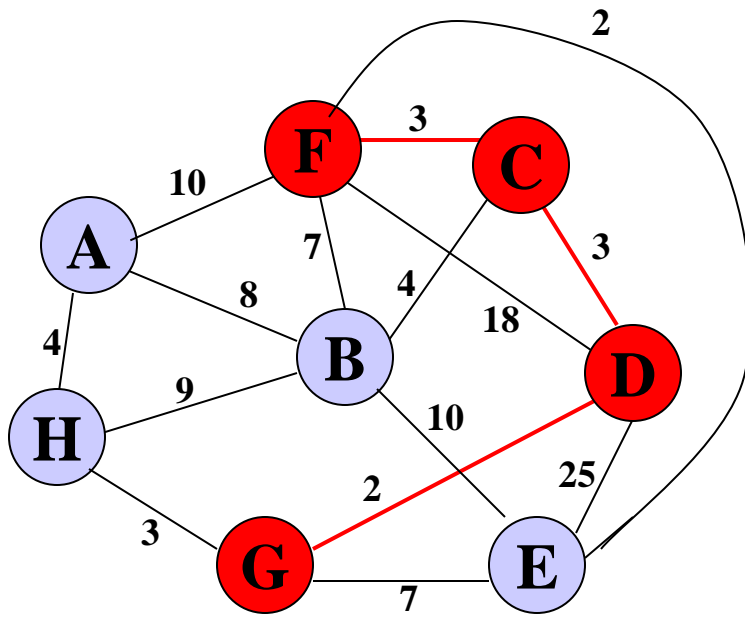
Select node with
minimum distance

	K	d_v	p_v
A			
B			
C	T	3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G



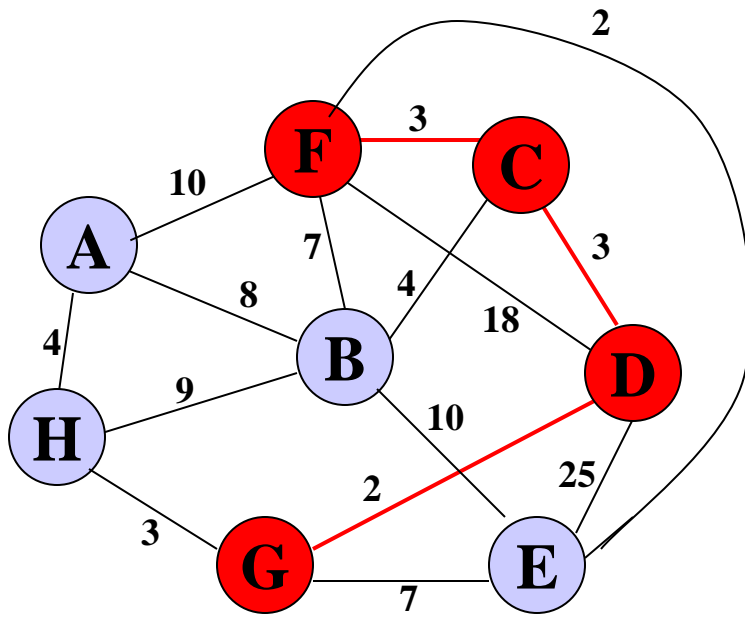
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	—
E		7	G
F		3	C
G	T	2	D
H		3	G



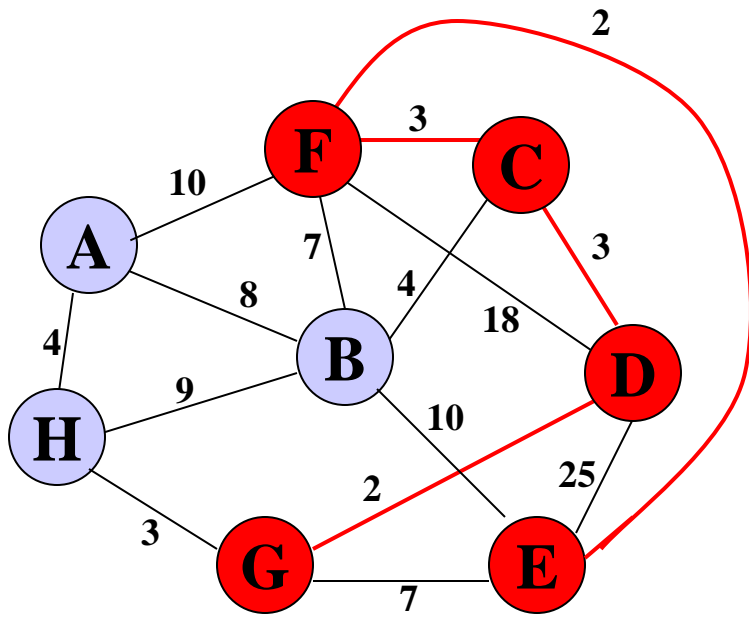
Select node with
minimum distance

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F	T	3	C
G	T	2	D
H		3	G



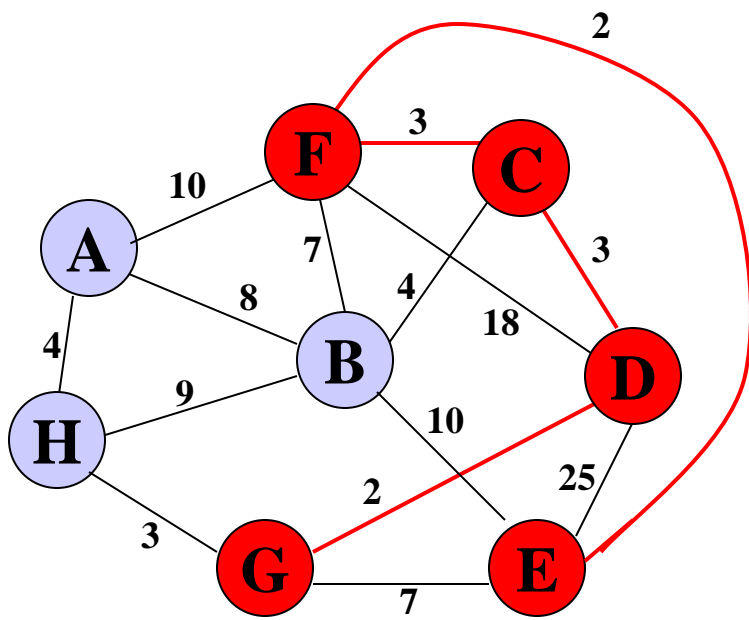
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	—
E		2	F
F	T	3	C
G	T	2	D
H		3	G



Select node with
minimum distance

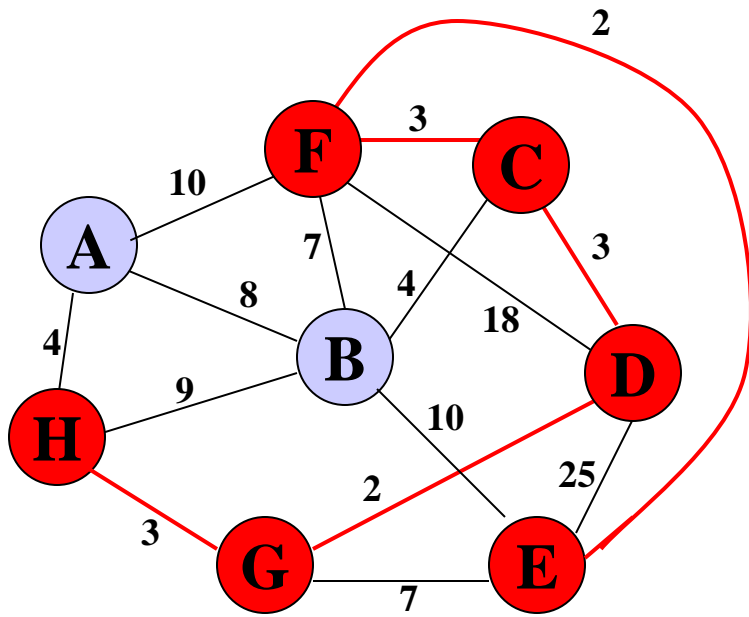
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G



Update distances of
adjacent, unselected nodes

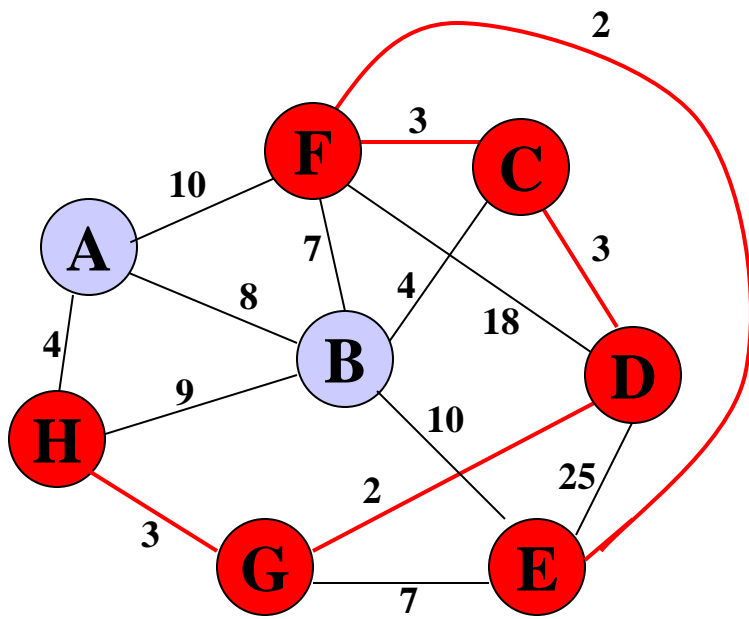
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	—
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Table entries unchanged



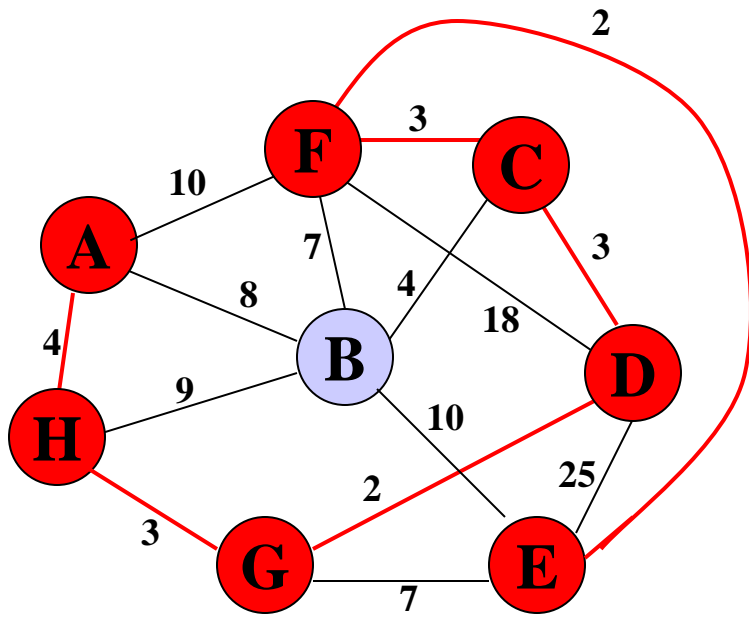
Select node with
minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



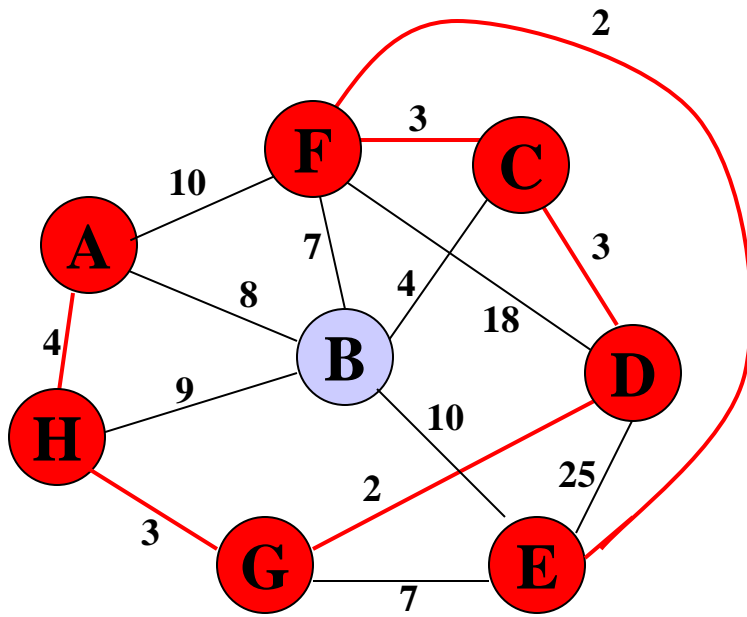
Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	—
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



Select node with
minimum distance

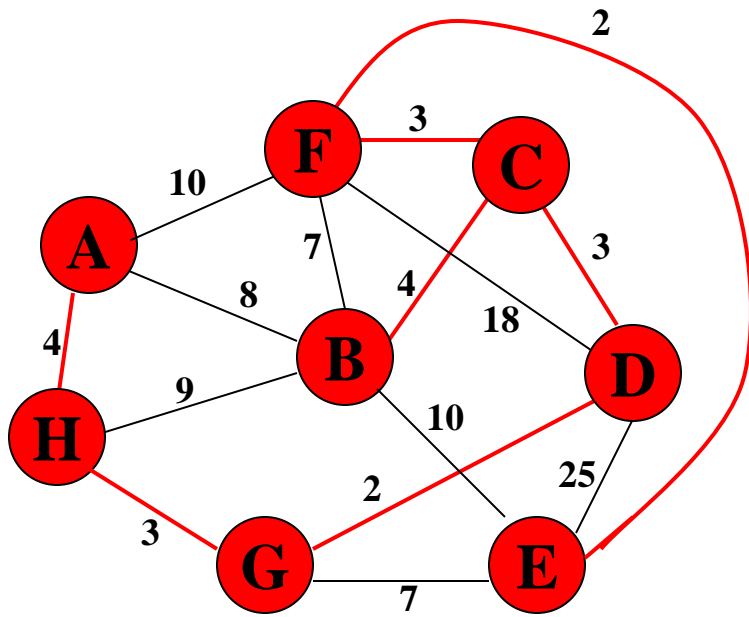
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



Update distances of
adjacent, unselected nodes

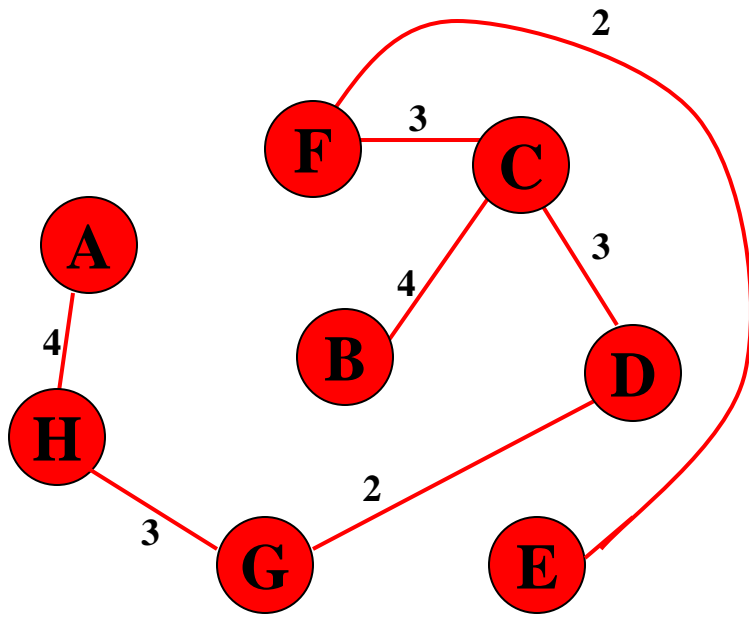
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	—
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Table entries unchanged



Select node with
minimum distance

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



Cost of Minimum
Spanning Tree = $\sum d_v = \mathbf{21}$

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	—
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Done

Exercise

- Implement **Prim's algorithm**
- **Input file : in_MST.txt**
บรรทัดแรก เป็น จำนวน node, จำนวน edge
บรรทัดต่อๆ มาคือ edge ต่าง ๆ ประกอบด้วยข้อมูล
nodeต้นทาง nodeปลายทาง ระยะทาง
- **Output** ให้พิมพ์ผลรวมของระยะทางที่สั้นที่สุด

Kruskal's Algorithm

- Work with edges, rather than nodes
- Two steps:
 - Sort edges by increasing edge weight
 - Select the first $|V| - 1$ edges that do not generate a cycle

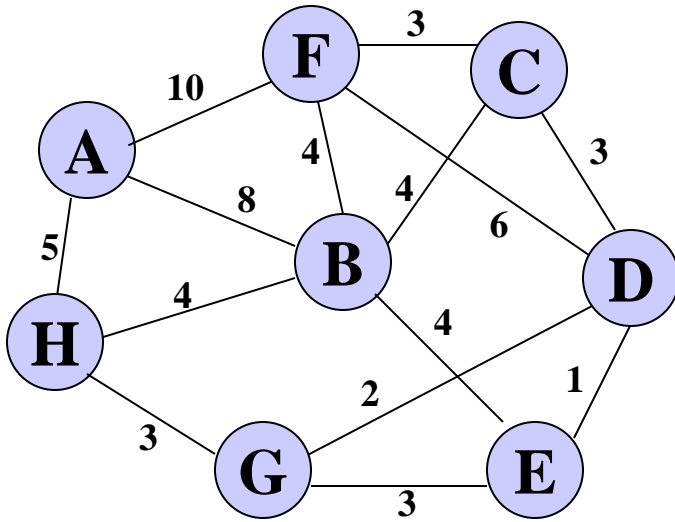
Kruskal's Algorithm

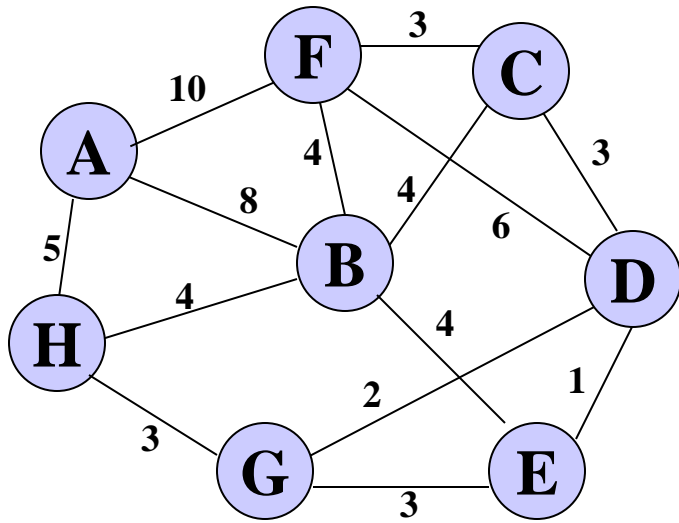
```
Let  $G = (V, E)$  be the given graph, with  $|V| = n$  {  
    Start with a graph  $T = (V, \phi)$   
    consisting of only the vertices of  $G$  and no edges;  
    /* This can be viewed as  $n$  connected components,  
       each vertex being one connected component */  
    Arrange  $E$  in the order of increasing costs;  
    for ( $i = 1, I \leq n - 1, i++$ ) {  
        Select the next smallest cost edge;  
        if (the edge connects two different connected components)  
            add the edge to  $T$ ;  
    }  
}
```

from <http://lcm.csa.iisc.ernet.in/dsa/node184.html>

Walk-Through

Consider an undirected, weight graph

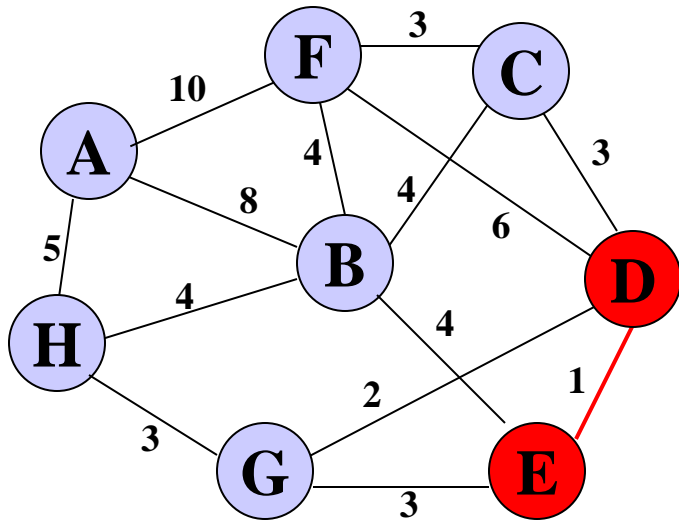




Sort the edges by increasing edge weight

<i>edge</i>	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

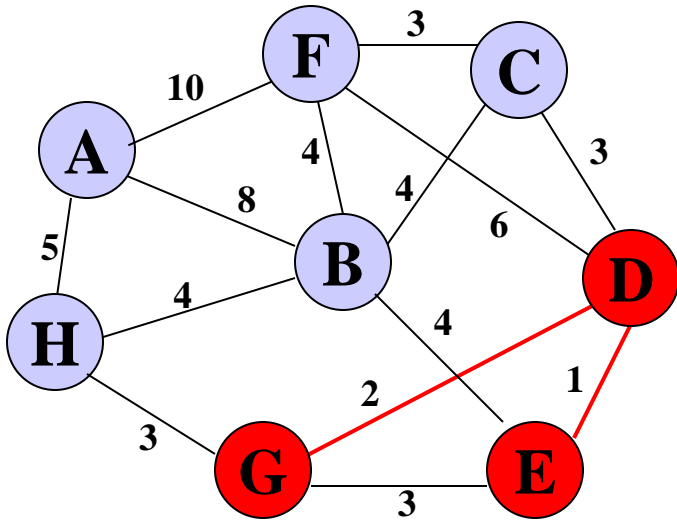


Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

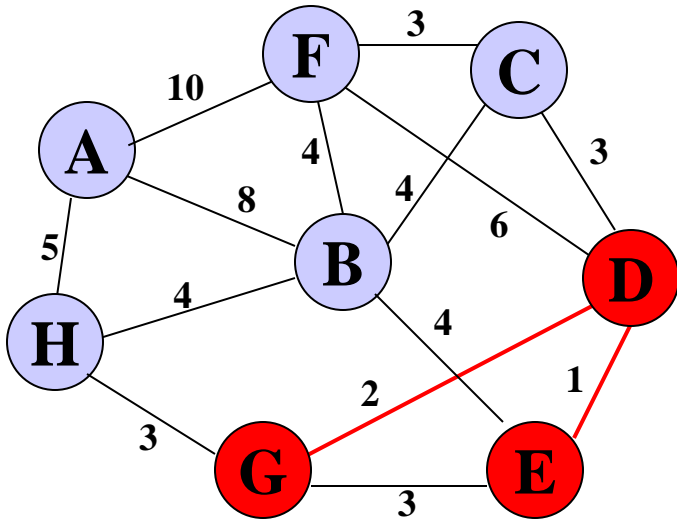
Select first $|V|-1$ edges which do not
generate a cycle



<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

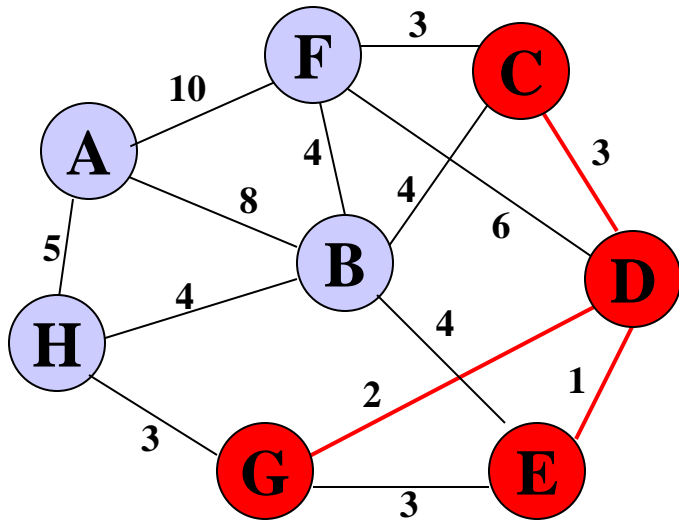
Select first $|V|-1$ edges which do not
generate a cycle



<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

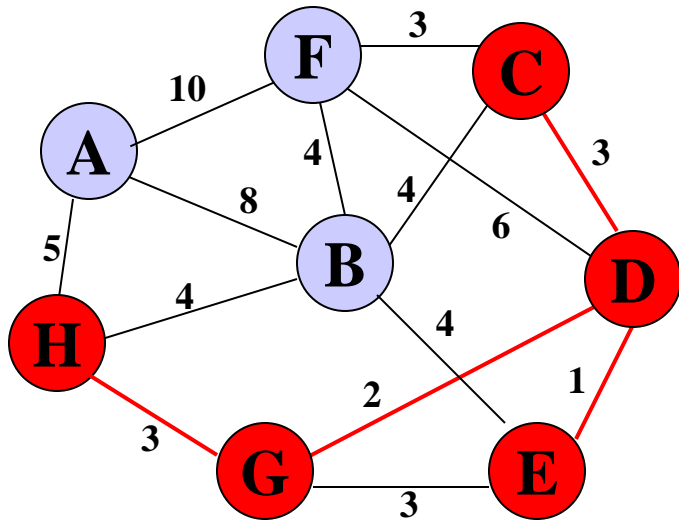
Accepting edge (E,G) would create a cycle



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	
(C,F)	3	
(B,C)	4	

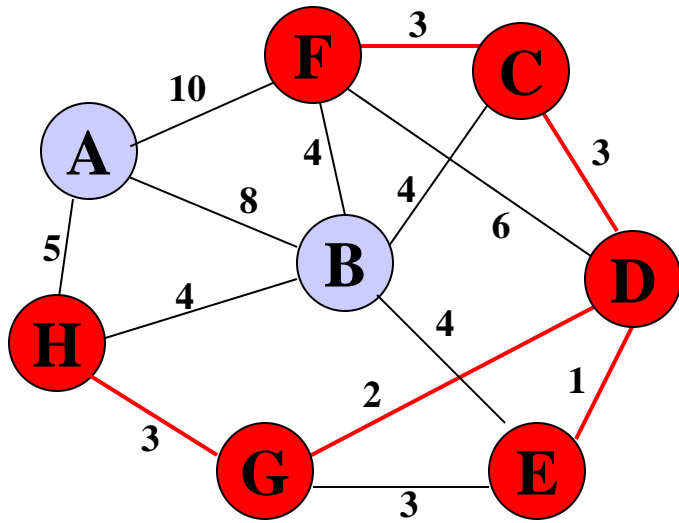
<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	
(B,C)	4	

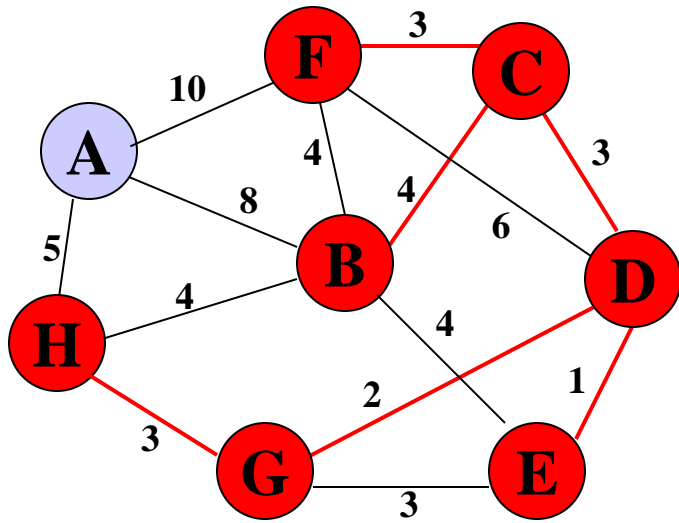
<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	

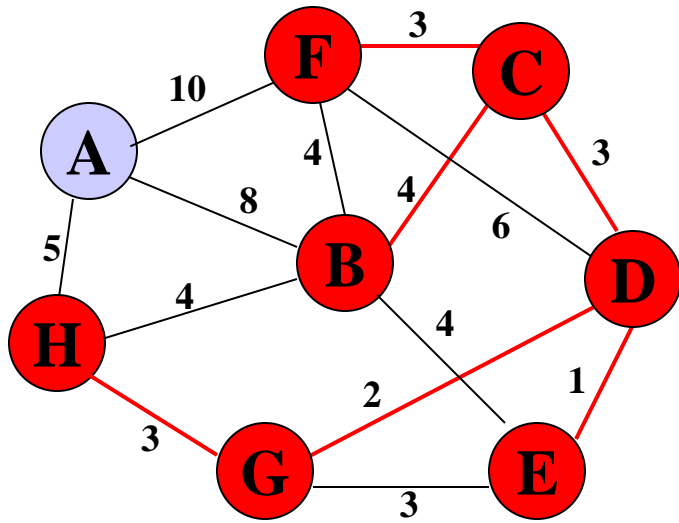
<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

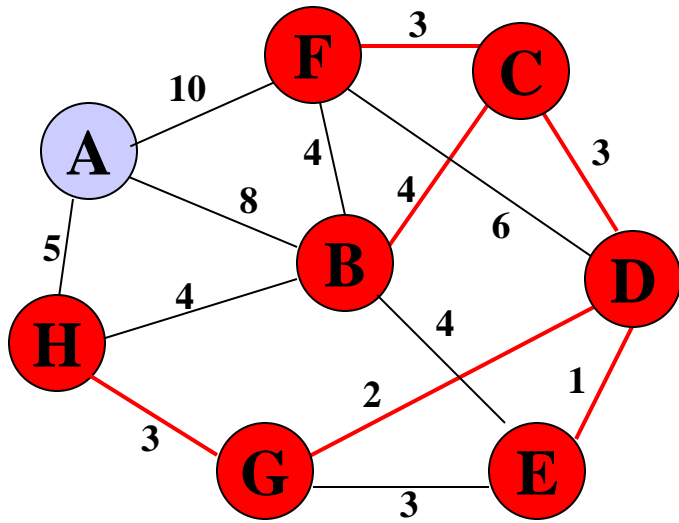
<i>edge</i>	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

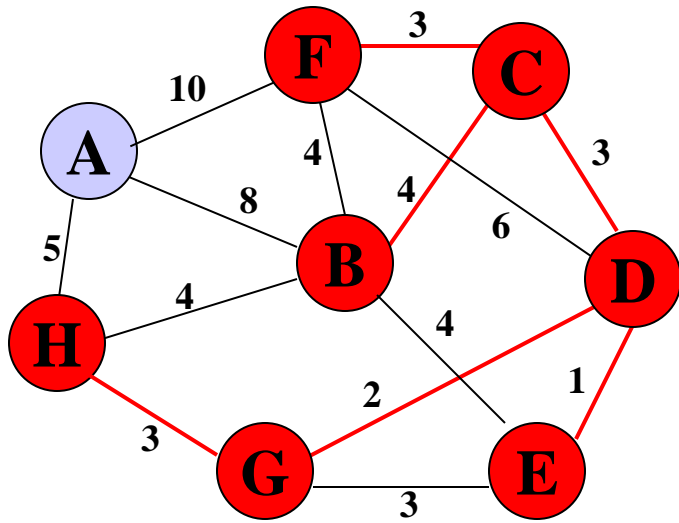
<i>edge</i>	d_v	
(B,E)	4	✗
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

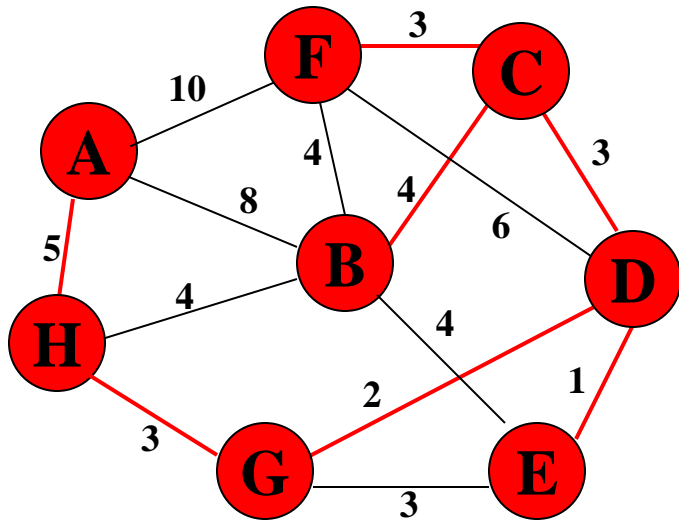
<i>edge</i>	d_v	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

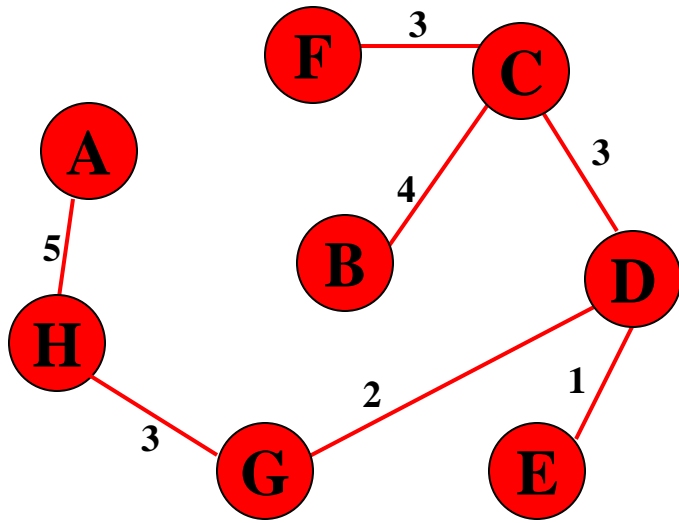
<i>edge</i>	d_v	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

<i>edge</i>	d_v	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	✓
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not
generate a cycle

<i>edge</i>	d_v	
(D,E)	1	✓
(D,G)	2	✓
(E,G)	3	✗
(C,D)	3	✓
(G,H)	3	✓
(C,F)	3	✓
(B,C)	4	✓

<i>edge</i>	d_v	
(B,E)	4	✗
(B,F)	4	✗
(B,H)	4	✗
(A,H)	5	✓
(D,F)	6	
(A,B)	8	
(A,F)	10	

} not
considered

Done

$$\text{Total Cost} = \sum d_v = 21$$

Exercise

- Implement Kruskal's algorithm
- Input file : in_MST.txt
บรรทัดแรก เป็น จำนวน node, จำนวน edge
บรรทัดต่อๆ มาคือ edge ต่าง ๆ ประกอบด้วยข้อมูล
nodeต้นทาง nodeปลายทาง ระยะทาง
- Output ให้พิมพ์ผลรวมของระยะทางที่สั้นที่สุด