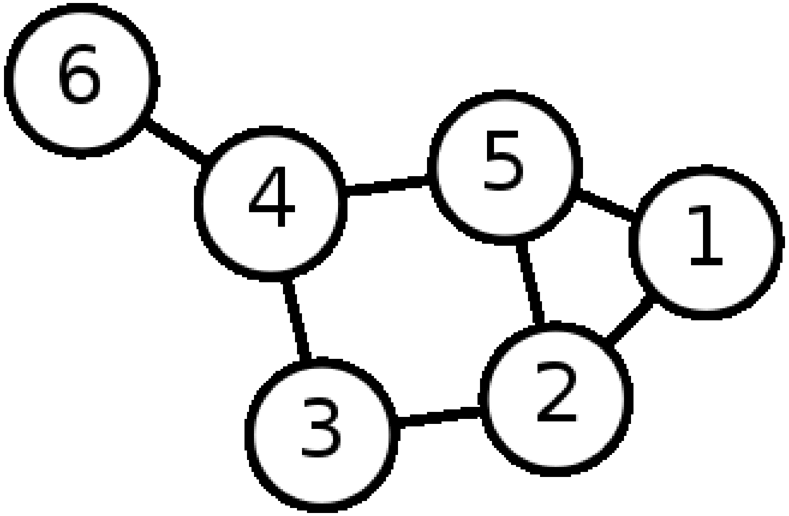
Djikstra's Algorithm

Slide Courtesy: Uwash, UT

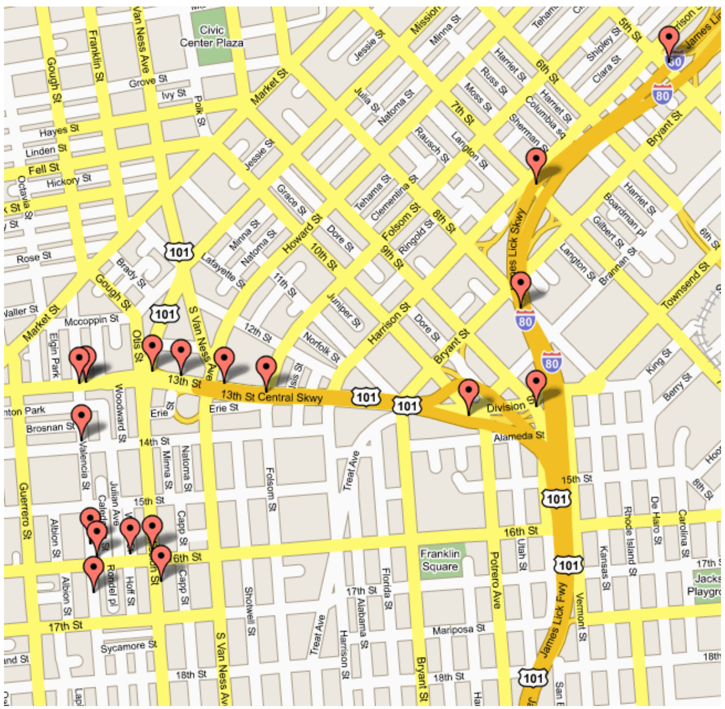
Single-Source Shortest Path Problem

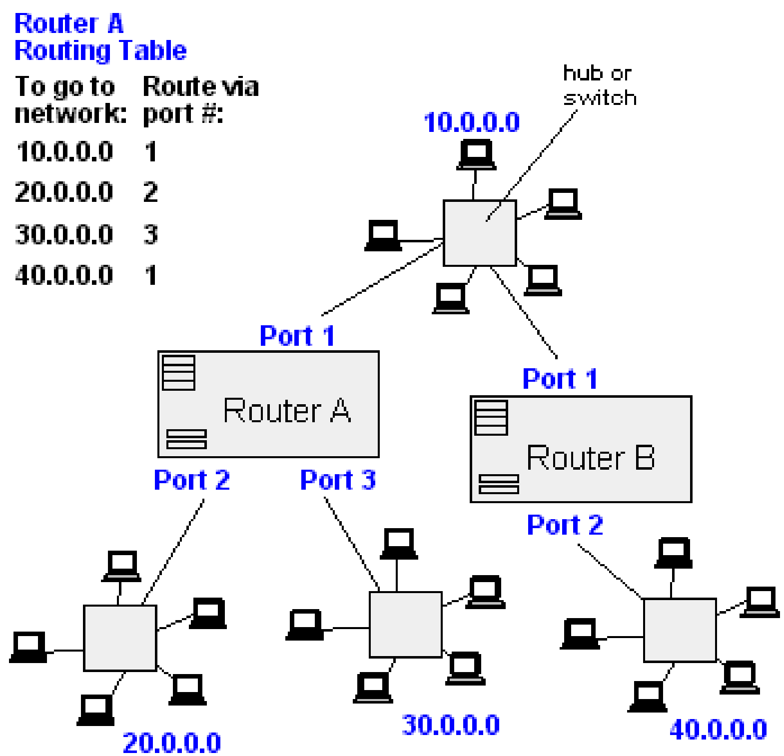
Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex *s* to all other vertices in the graph.



Applications

- Maps (Map Quest, Google Maps)
- Routing Systems





Dijkstra's algorithm

Dijkstra's algorithm - is 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.

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

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

Dijkstra's algorithm

Initial Approach: Can we adapt BFS by "sub-dividing each edge " ?

Dijkstra's algorithm: mimics BFS without explicitly subdivision.

$\delta(u)$: shortest path from s to u.

Order vertices from increasing distance from s to u:
$u_1, u_2, \dots, u_n$

The algorithm runs in n iterations: in iteration i, it finds u_i and $\delta(u_i)$.

Dijkstra's algorithm: motivation

- How to find u_2? u_3?

Dijkstra's algorithm

- Suppose after iteration i, we know the set $S_i = \{u_1, \dots, u_i\}$ and $\delta(u_1), \dots, \delta(u_i)$.

How to find u_{i+1} ?

Dijkstra's algorithm

- Suppose after iteration i, we know the set $S_i = \{u_1, \ldots, u_i\}$ and $\delta(u_1), \ldots, \delta(u_i)$.

How to find u_{i+1} ?

Idea: for each u not in S_i, find the shortest path from s to u which only uses vertices in S_i

Dijkstra's algorithm

Idea: for each u not in S_i, find the shortest path from s to u which only uses vertices in S_i

Call this $D_i[u] = \min_{v \in S_i}(\delta(v) + wt(v, u))$

For u_{i+1}, $\delta(u_{i+1}) = D_i[u_{i+1}]$

For other vertices $u \notin S_i, \delta(u) \geq D_i[u]$.

So, $D_i[u_{i+1}] \leq D_i[u] \; \forall \; u \notin S_i$

Dijkstra's algorithm

Initialize $S_1 = \{s\}, \delta(s) = 0.$

For i=1, …, n-1

For every $u \notin S_i$,
$D_i[u] = \min_{v \in S_i}(\delta(v) + wt(v, u))$

Let u* be the vertex with min. $D_i[u]$

Set $\delta(u^*) = D_i[u^*]$ and $S_{i+1} = S_i \cup \{u^*\}$

Dijkstra's algorithm

- Correctness: consider iteration i and let u* be the vertex for which $D_i[u^*]$ is minimum.
- Need to show: $\delta(u^*) \leq \delta(u) \ \ \forall\, u \notin S_i$

Dijkstra's algorithm

Initialize $S_1 = \{s\}, \delta(s) = 0.$

For i=1, ..., n-1

For every $u \notin S_i$,
$D_i[u] = \min_{v \in S_i}(\delta(v) + wt(v,u))$

Let u* be the vertex with min. $D_i[u]$

Set $\delta(u^*) = D_i[u^*]$ and $S_{i+1} = S_i \cup \{u^*\}$

Dijkstra's algorithm

Initialize $S_1 = \{s\}, \delta(s) = 0$.

For i=1, ..., n-1

For every $u \notin S_i$,
$D_i[u] = \min_{v \in S_i}(\delta(v) + wt(v,u))$

Let u* be the vertex with min. $D_i[u]$

Set $\delta(u^*) = D_i[u^*]$ and $S_{i+1} = S_i \cup \{u^*\}$

Can we improve this code ?

$D_i[u]$
$= \min(D_{i-1}[u], \delta(u_i) + wt(v,u))$

Dijkstra's algorithm

Initialize $S = \emptyset, D[s] = 0, D[u] = \infty, u \neq s$

For i=1, …, n

Let u* be the vertex with min. $D[u]$

Add u* to S

For every $u \notin S, (u^*, u) \in E$
$D[u] = \min(D[u], D[u^*] + \ wt(u^*, u))$

Dijkstra's algorithm

Initialize $S = \emptyset, D[s] = 0, D[u] = \infty, u \neq s$

For i=1, …, n

Let u* be the vertex with min. $D[u]$

Add u* to S

For every $u \notin S, (u^*, u) \in E$

$D[u] = \min(D[u], D[u^*] + wt(u^*, u))$

: if min updated parent(u) = u*

How to find actual paths ?

Implementation Details

Initialize $S = \emptyset, D[s] = 0, D[u] = \infty, u \neq s$

For i=1, …, n

Let u* be the vertex with min. $D[u]$

Add u* to S

For every $u \notin S, (u^*, u) \in E$

$D[u] = \min(D[u], D[u^*] + wt(u^*, u))$

: if min updated parent(u) = u*

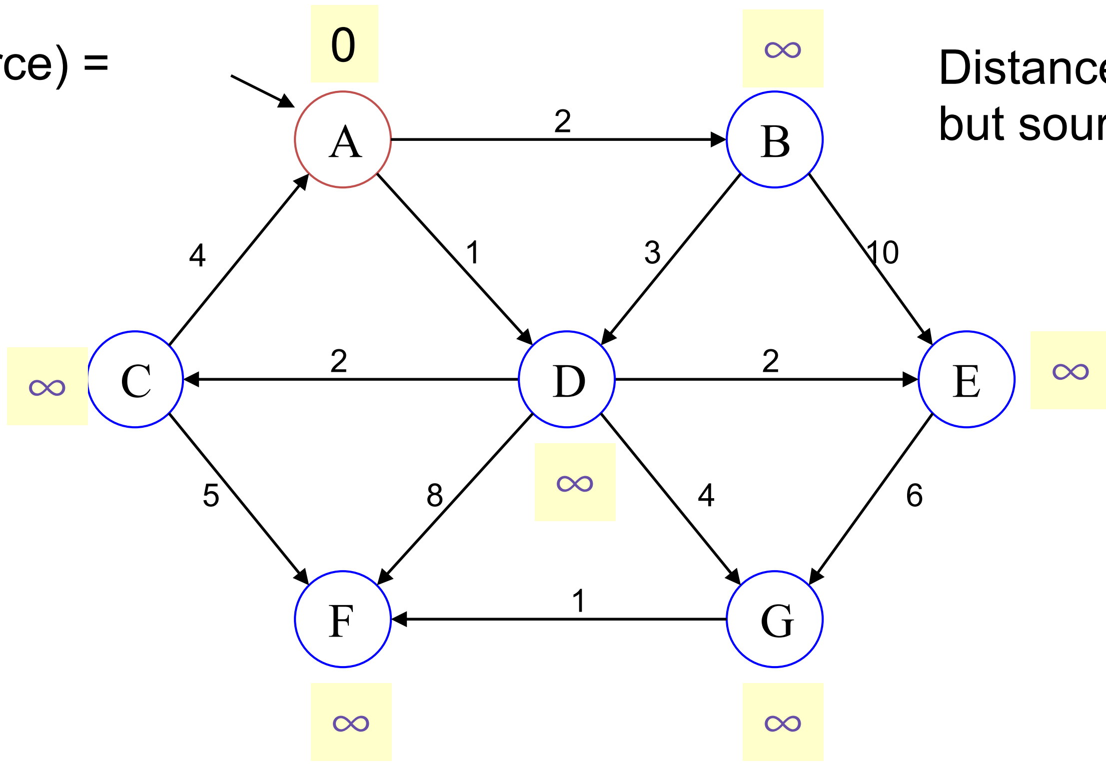
Store S (and D[u] values) in a heap, : deletemin and decrease key

Maintain whether a vertex is in S using a Boolean array.

Running time: O(m log n)

Example: Initialization

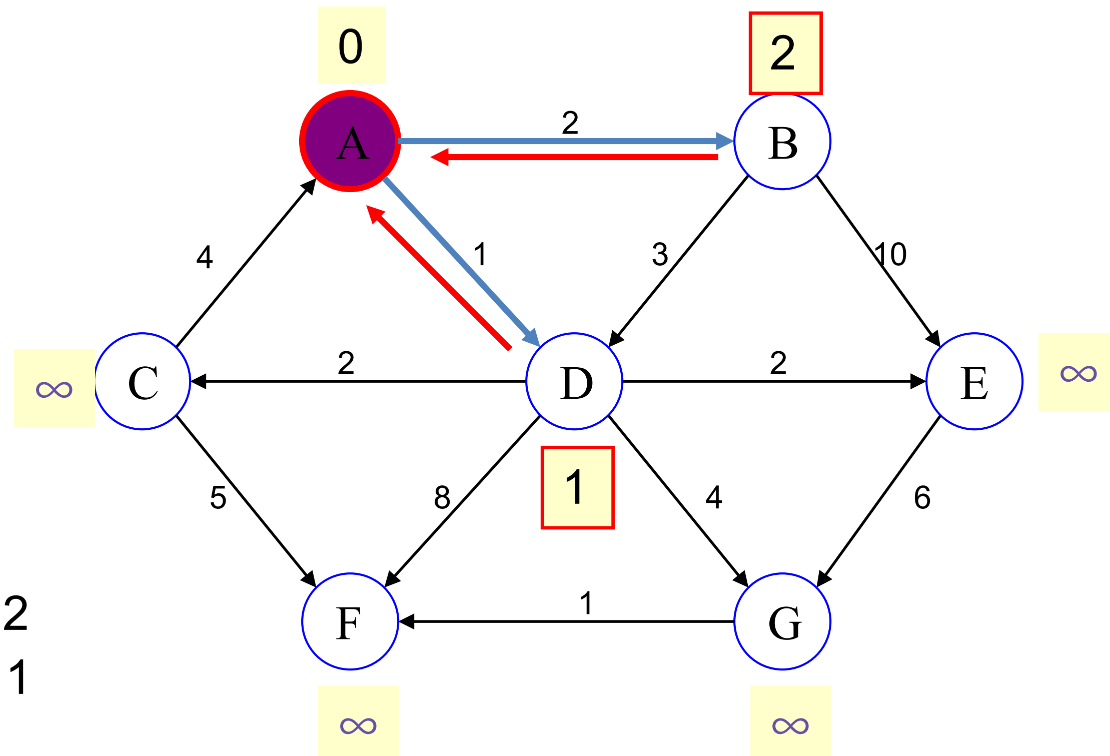
Distance(source) = 0



Distance (all vertices but source) = ∞

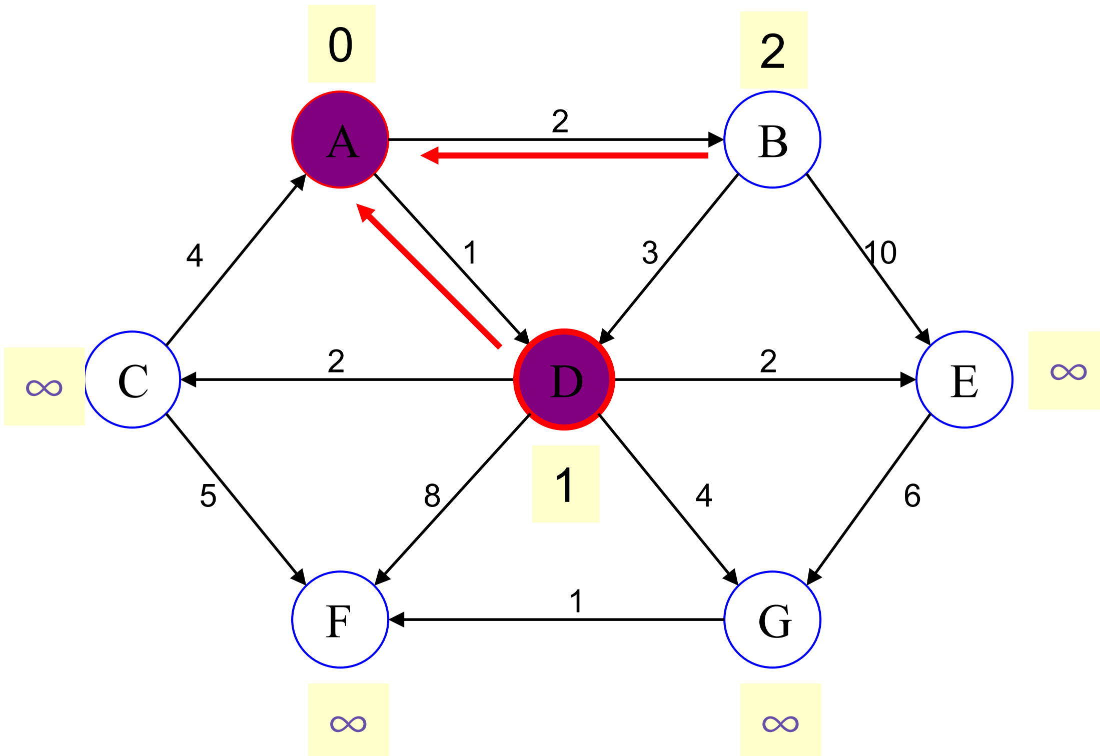
Pick vertex in List with minimum distance.

Example: Update neighbors' distance



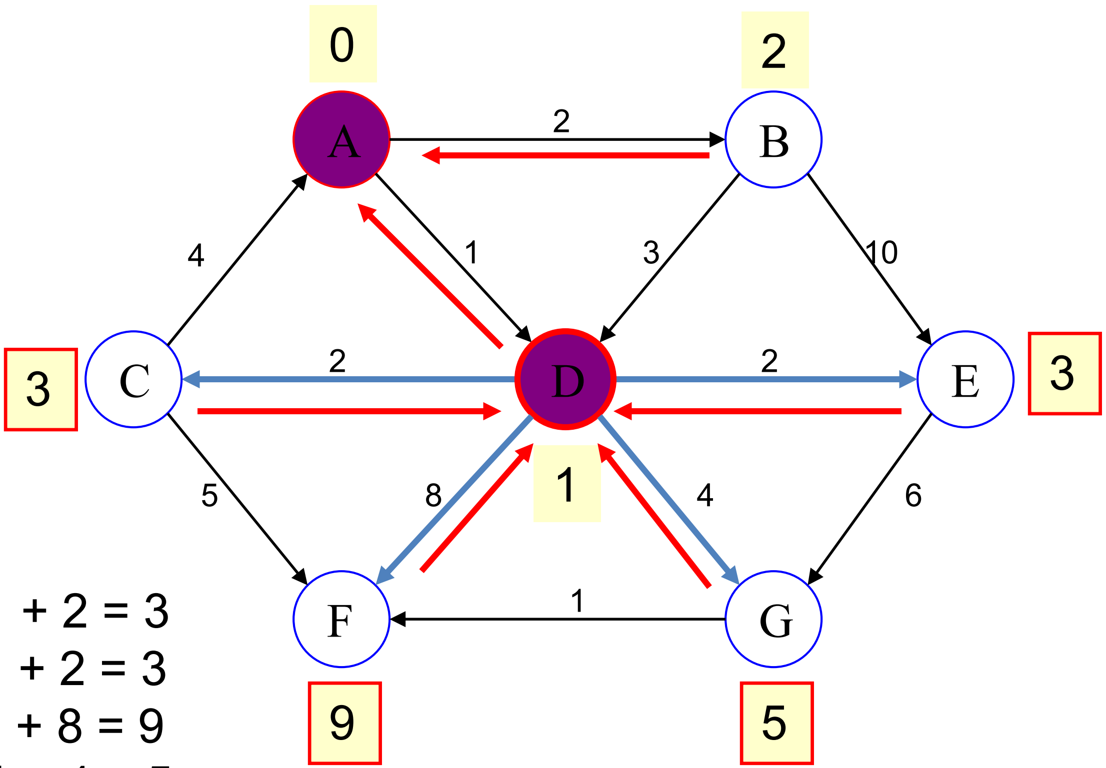
Distance(B) = 2
Distance(D) = 1

Example: Remove vertex with minimum distance



Pick vertex in List with minimum distance, i.e., D

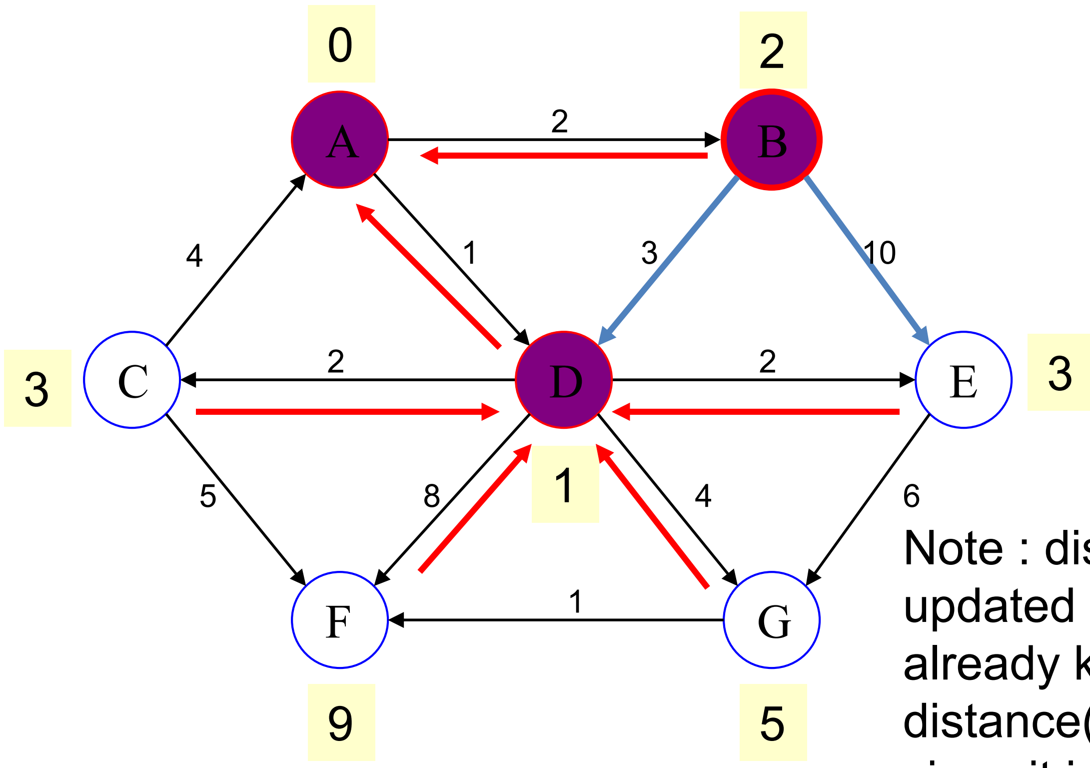
Example: Update neighbors



Distance(C) = 1 + 2 = 3
Distance(E) = 1 + 2 = 3
Distance(F) = 1 + 8 = 9
Distance(G) = 1 + 4 = 5

Example: Continued...

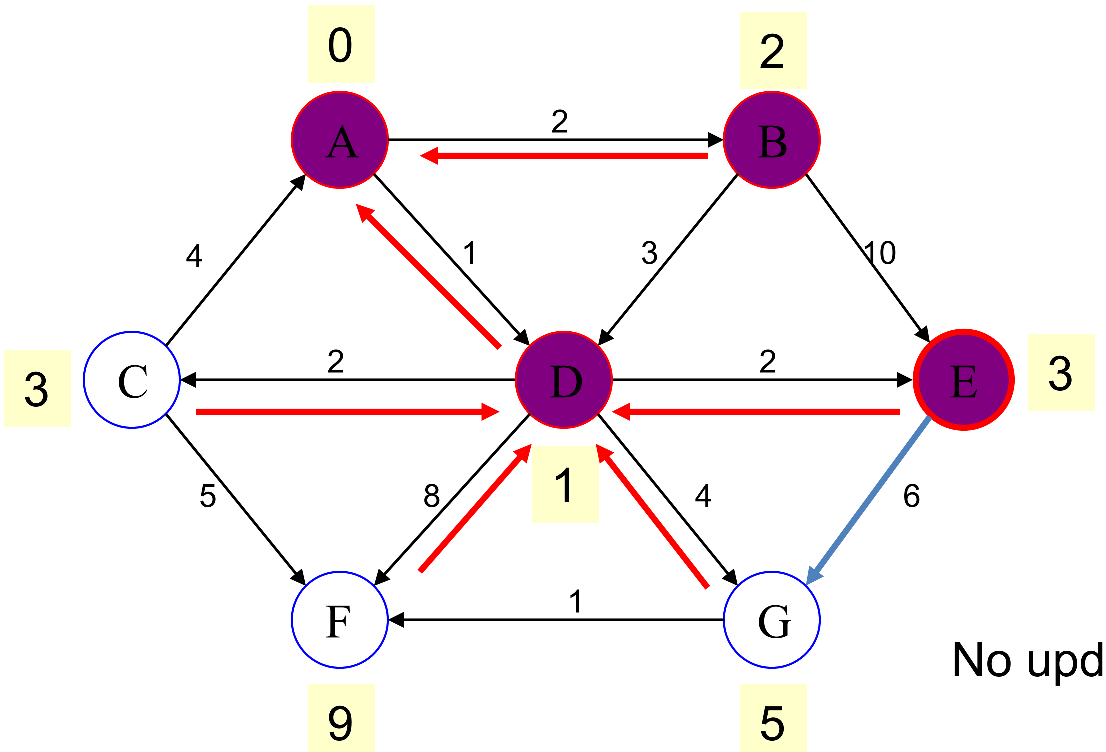
Pick vertex in List with minimum distance (B) and update neighbors



Note : distance(D) not updated since D is already known and distance(E) not updated since it is larger than previously computed

Example: Continued...

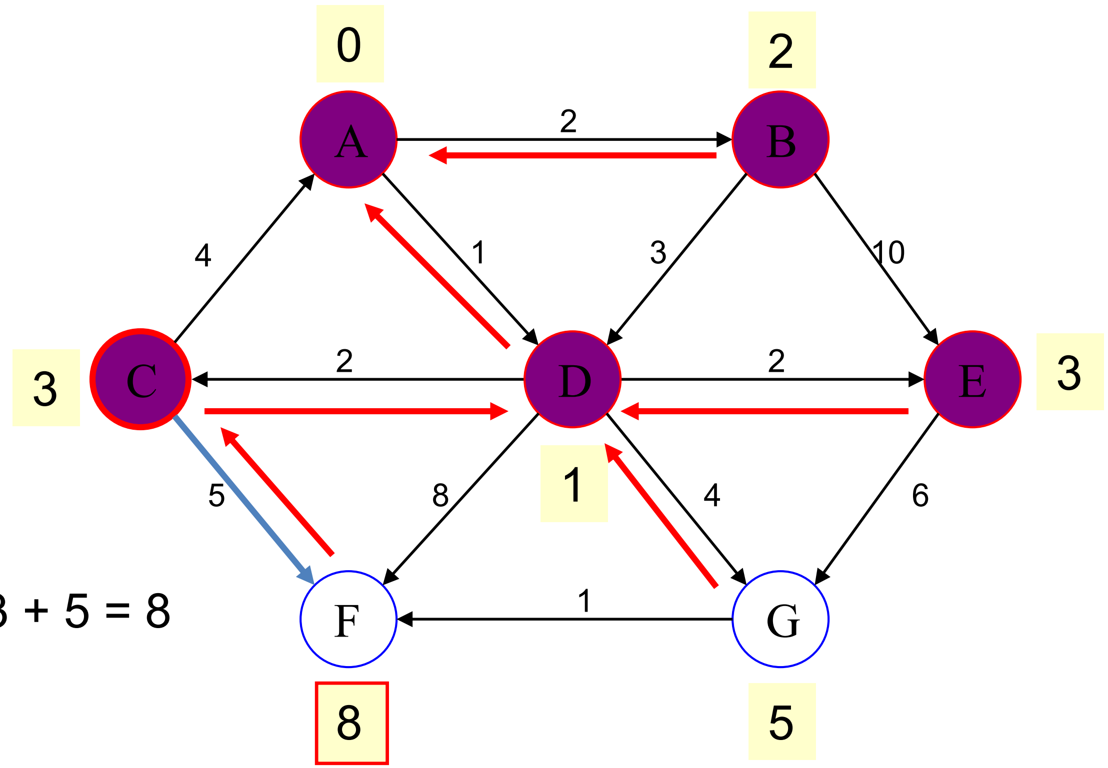
Pick vertex List with minimum distance (E) and update neighbors



No updating

Example: Continued...

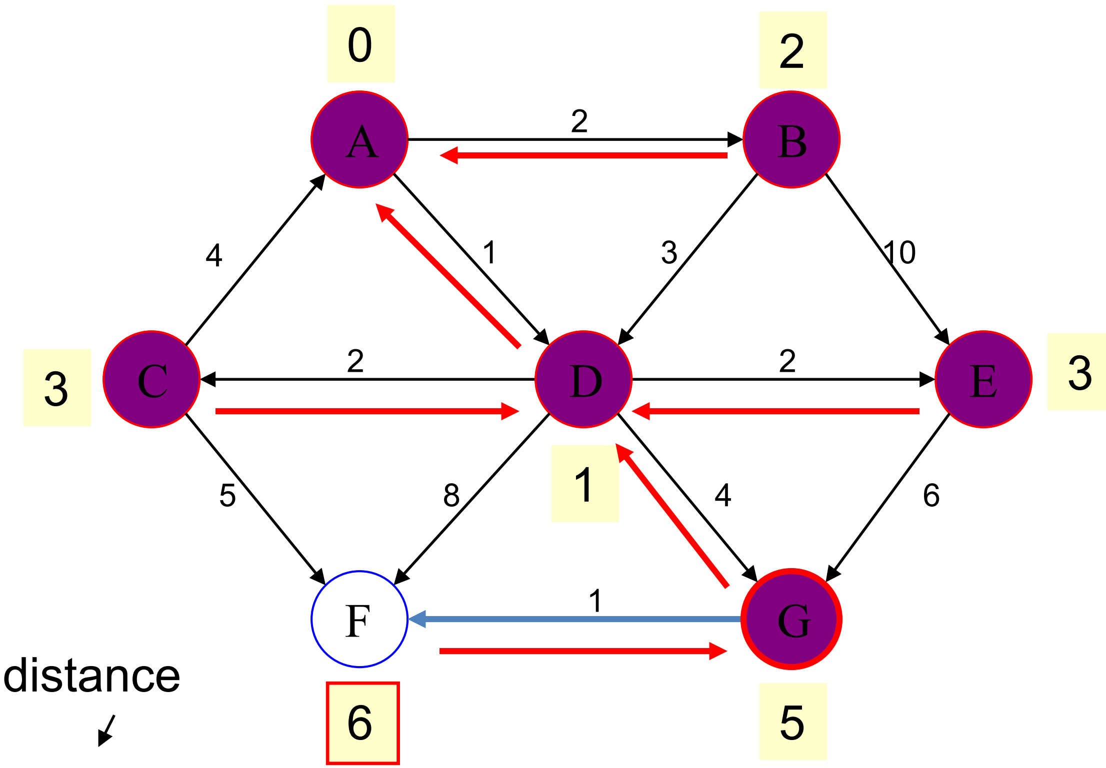
Pick vertex List with minimum distance (C) and update neighbors



Distance(F) = 3 + 5 = 8

Example: Continued...

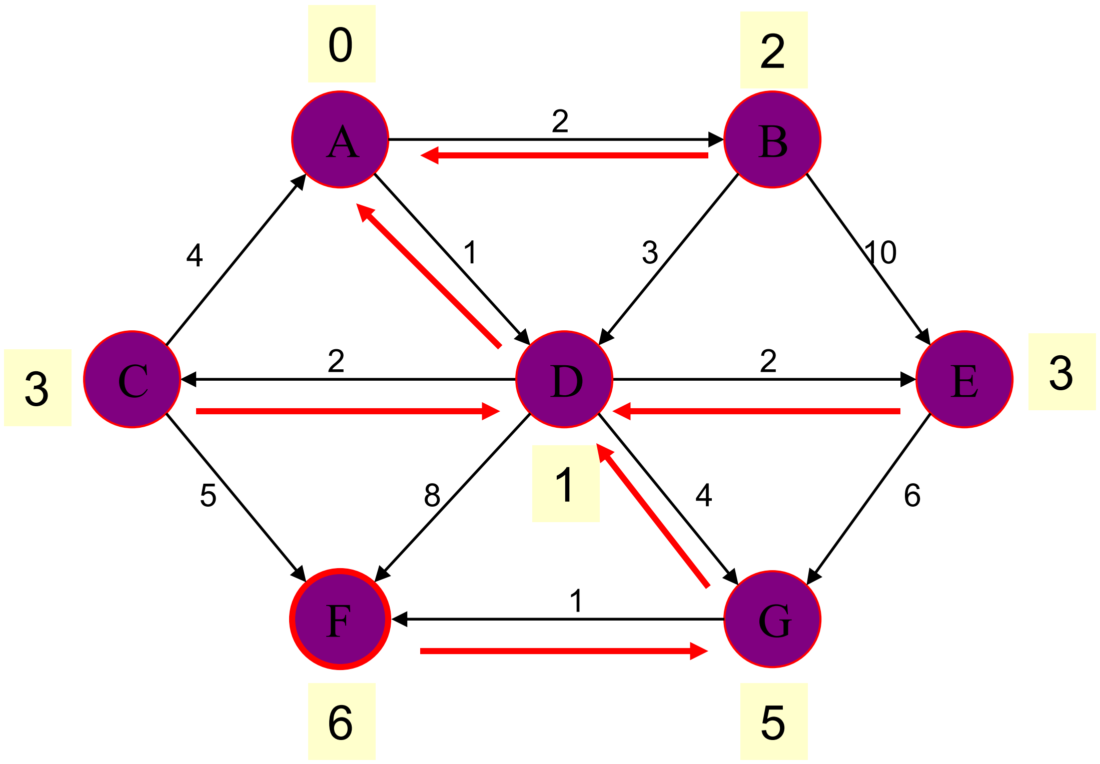
Pick vertex List with minimum distance (G) and update neighbors



Previous distance

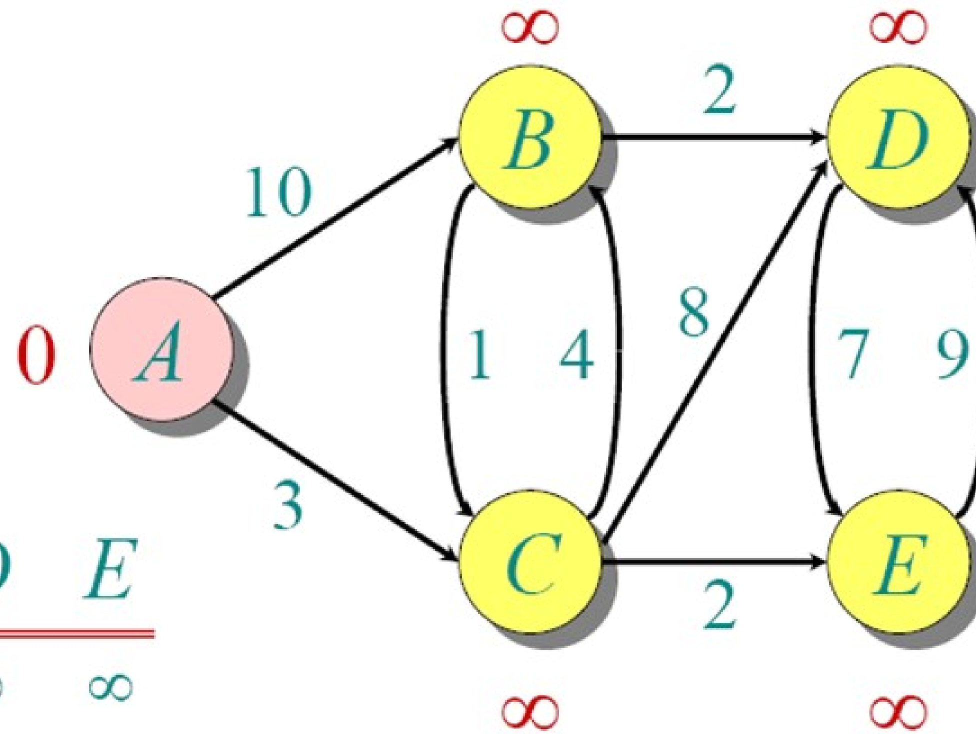
Distance(F) = min (8, 5+1) = 6

Example (end)



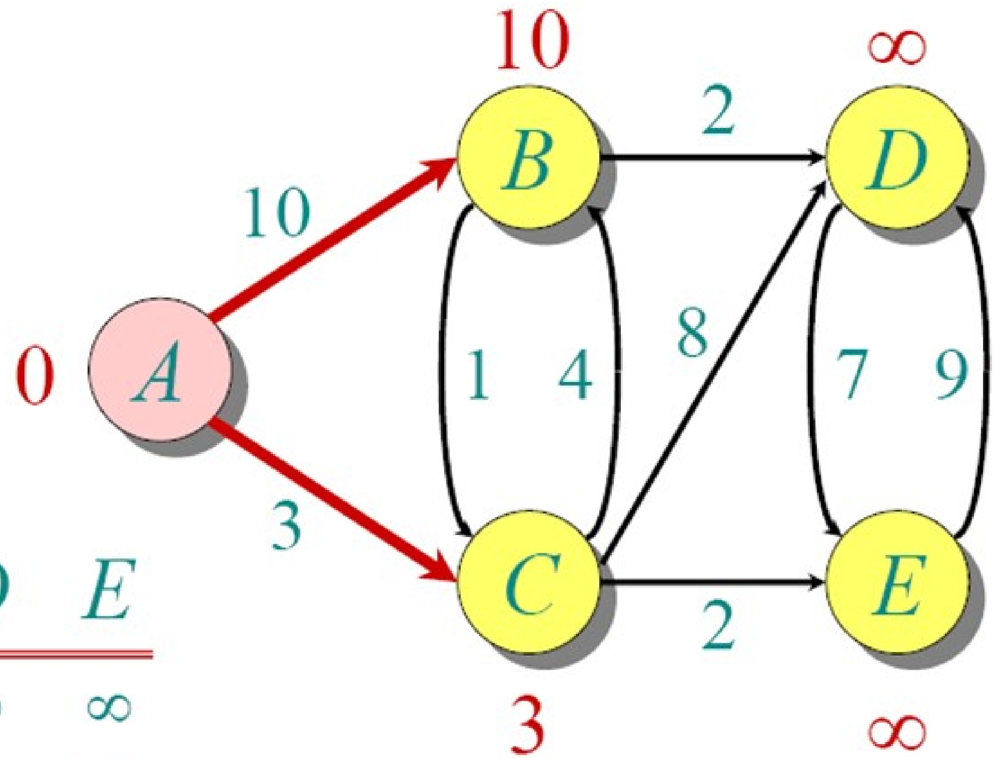
Pick vertex not in S with lowest cost (F) and update neighbors

Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

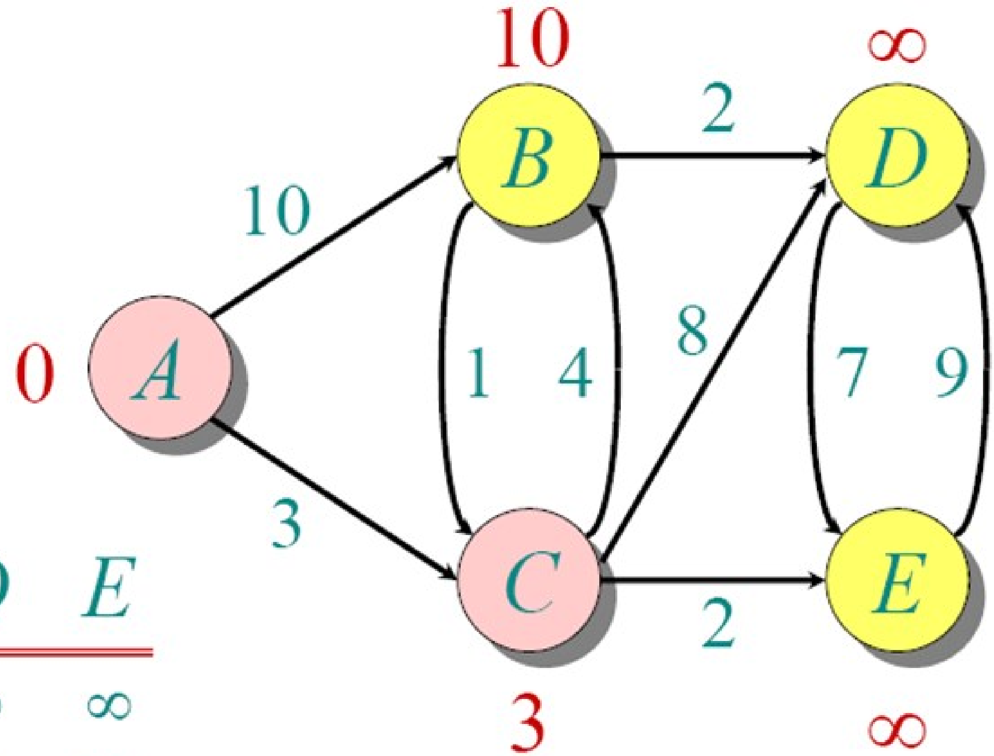
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

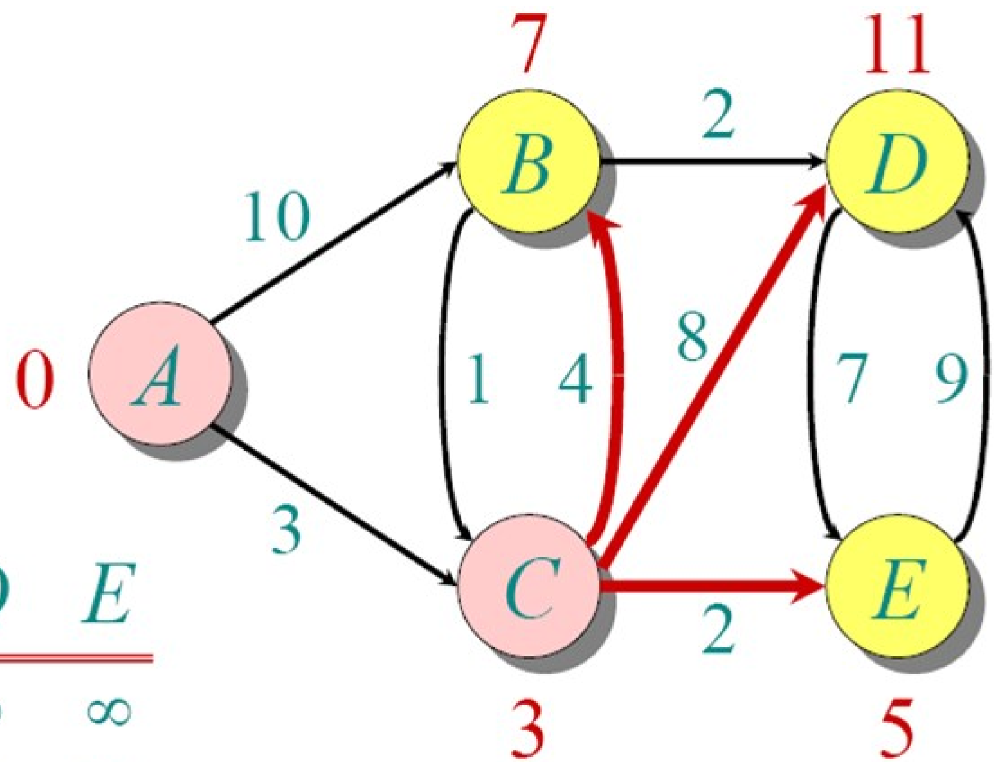
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

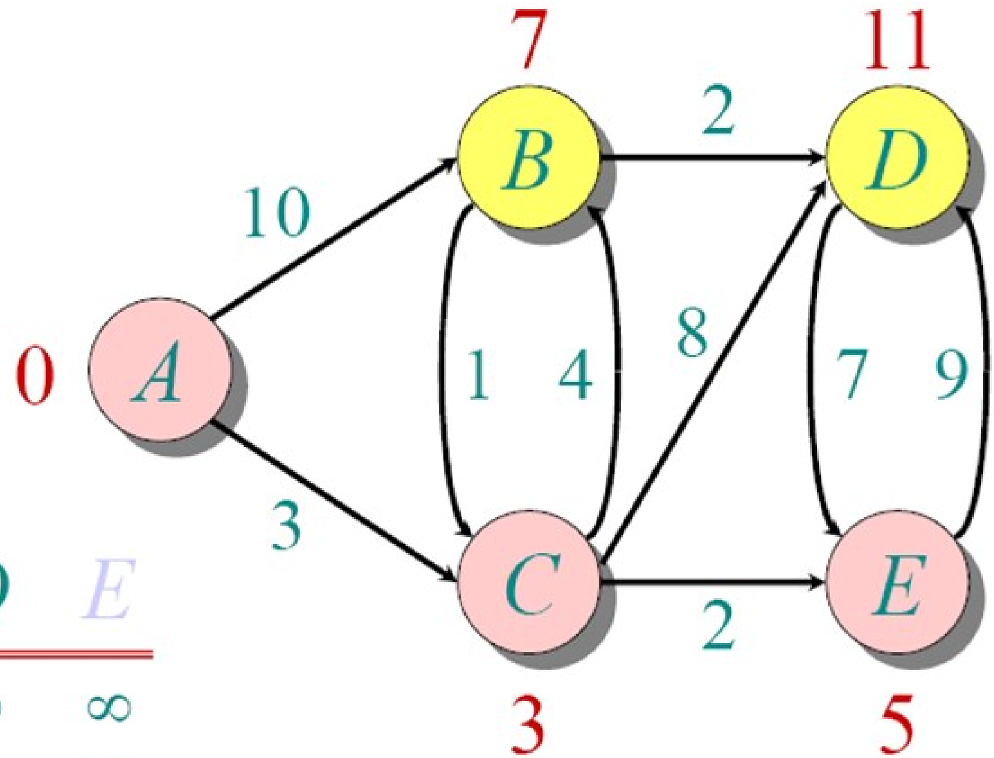
Another Example



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

$S: \{ A, C \}$

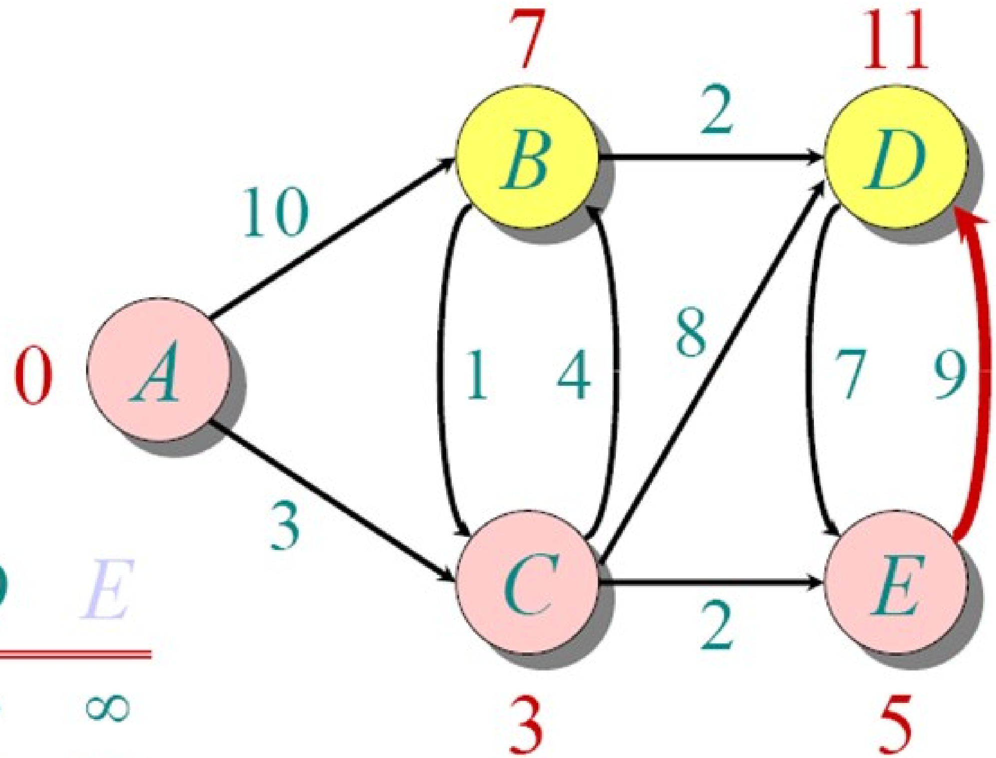
Another Example



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

S: { *A*, *C*, *E* }

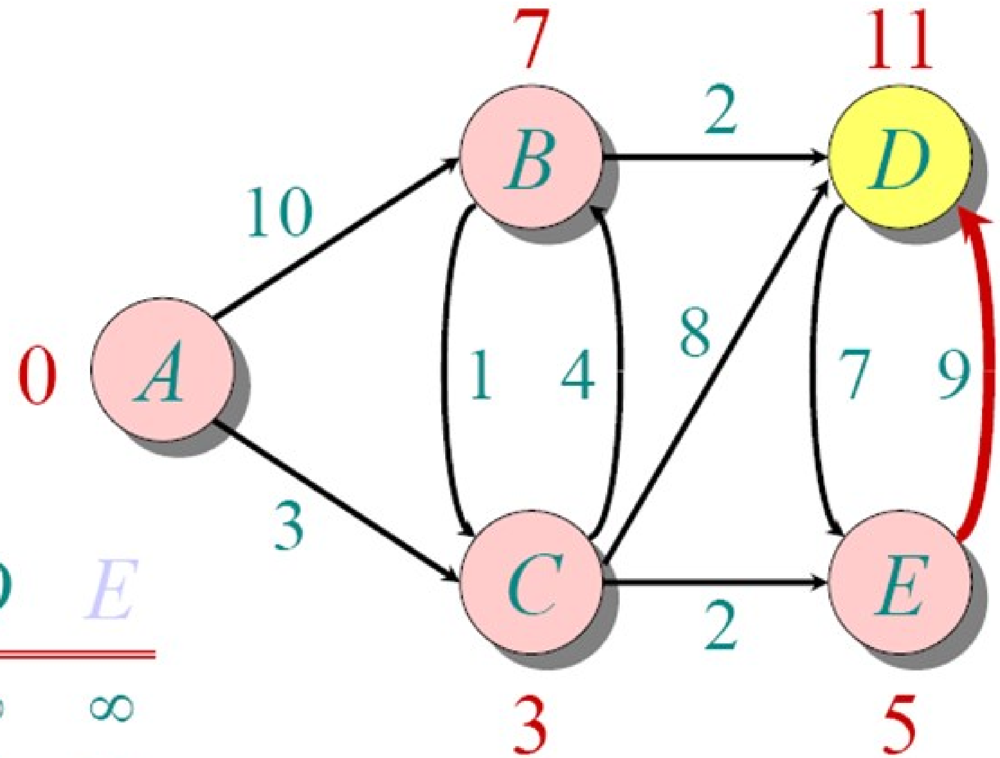
Another Example



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

S: { *A*, *C*, *E* }

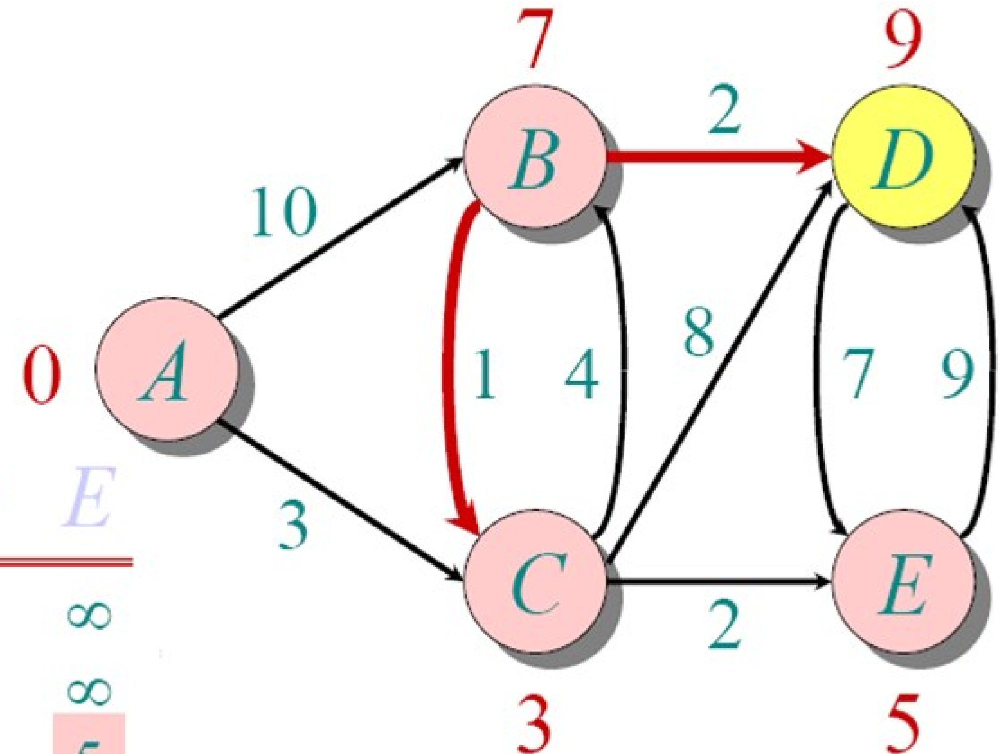
Another Example



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

S: { A, C, E, B }

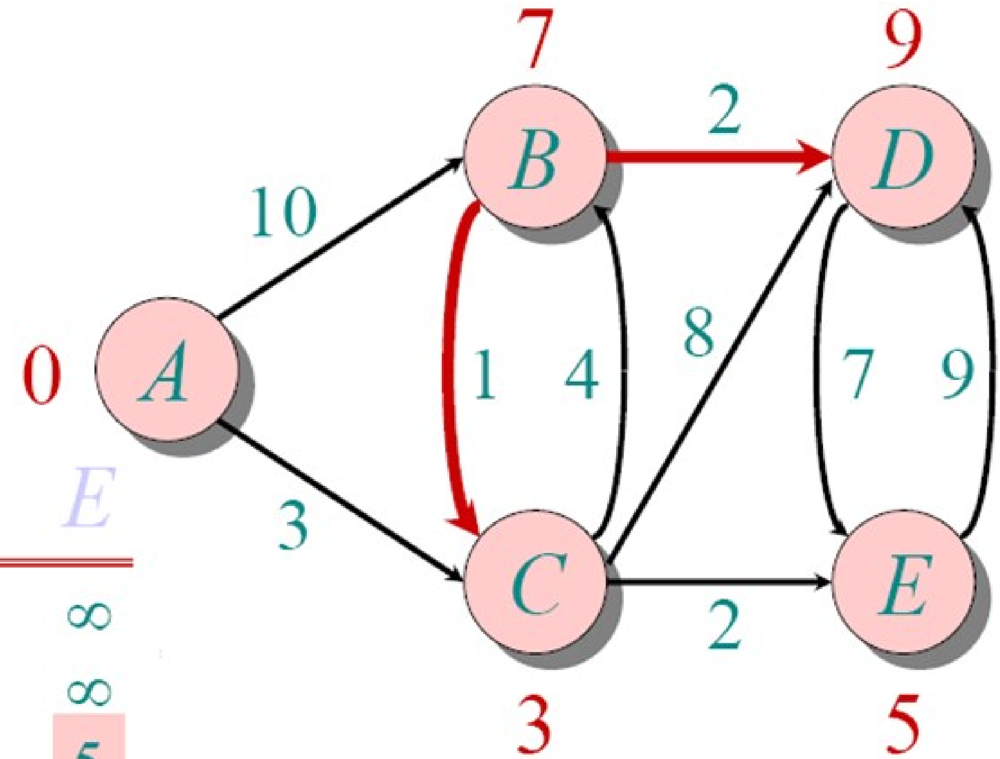
Another Example



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

S: { A, C, E, B }

Another Example



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

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