University of Southern California

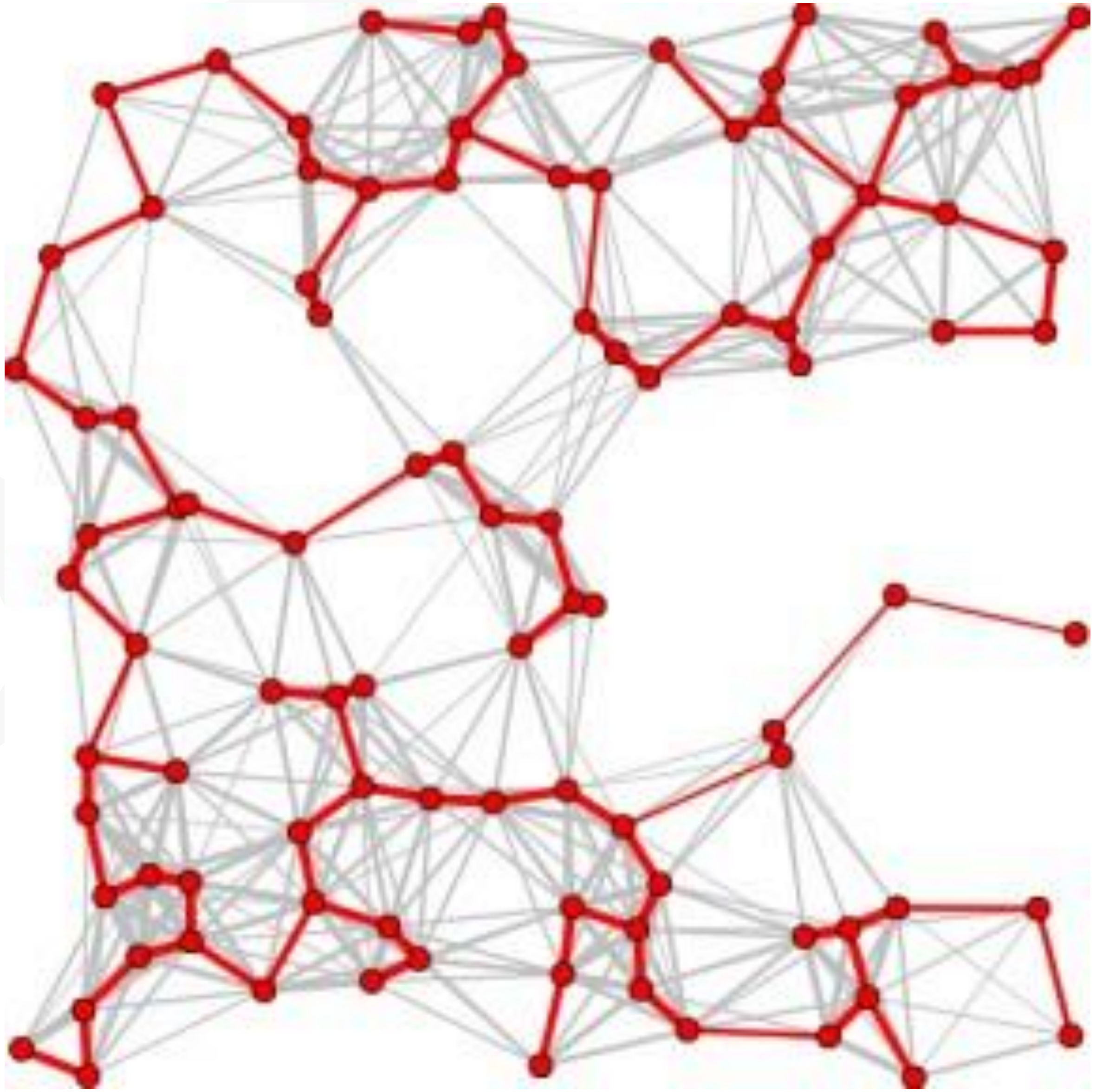
Viterbi School of Engineering

EE595: Software Design and Optimization

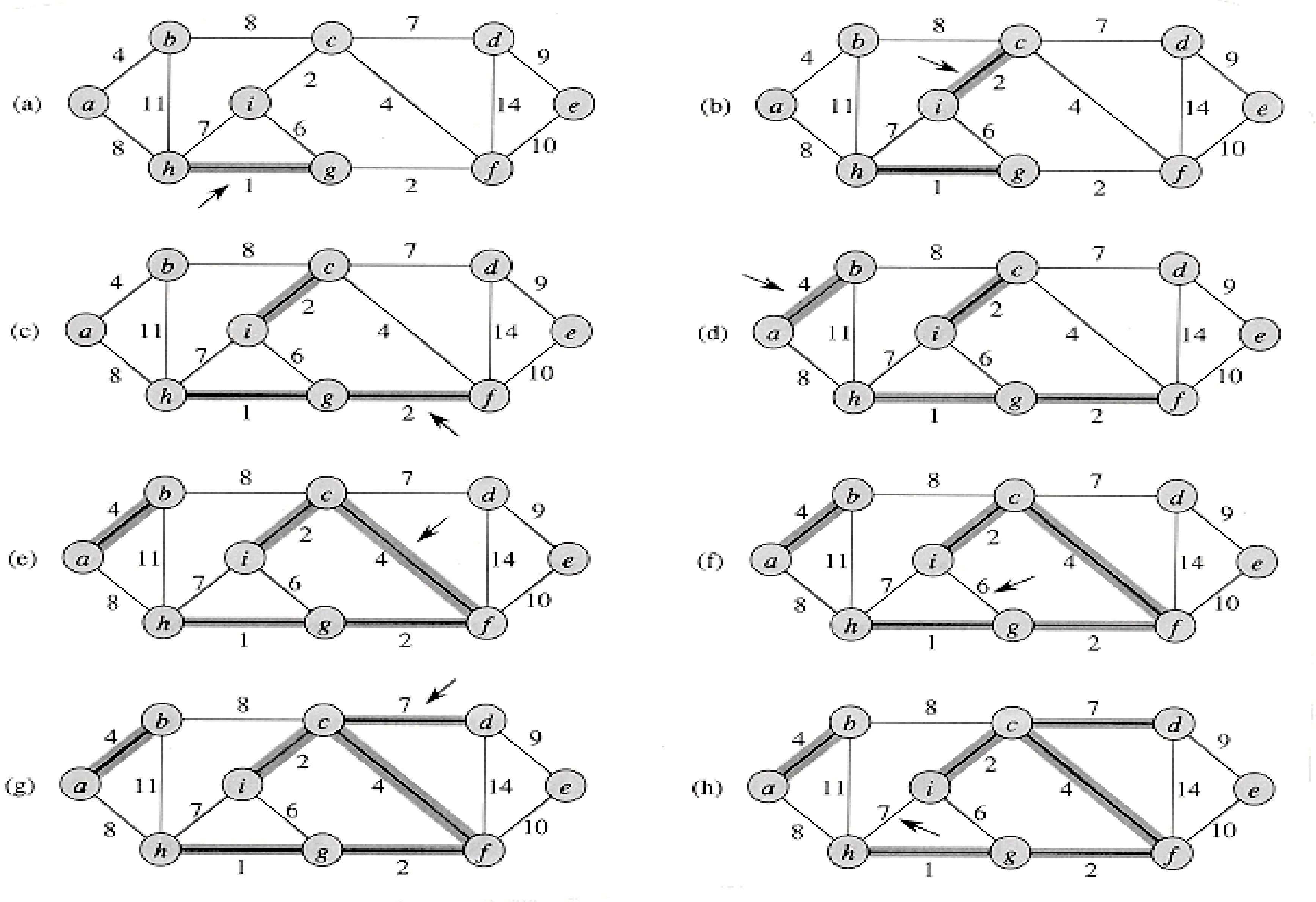
MST and Short Path

MSPT

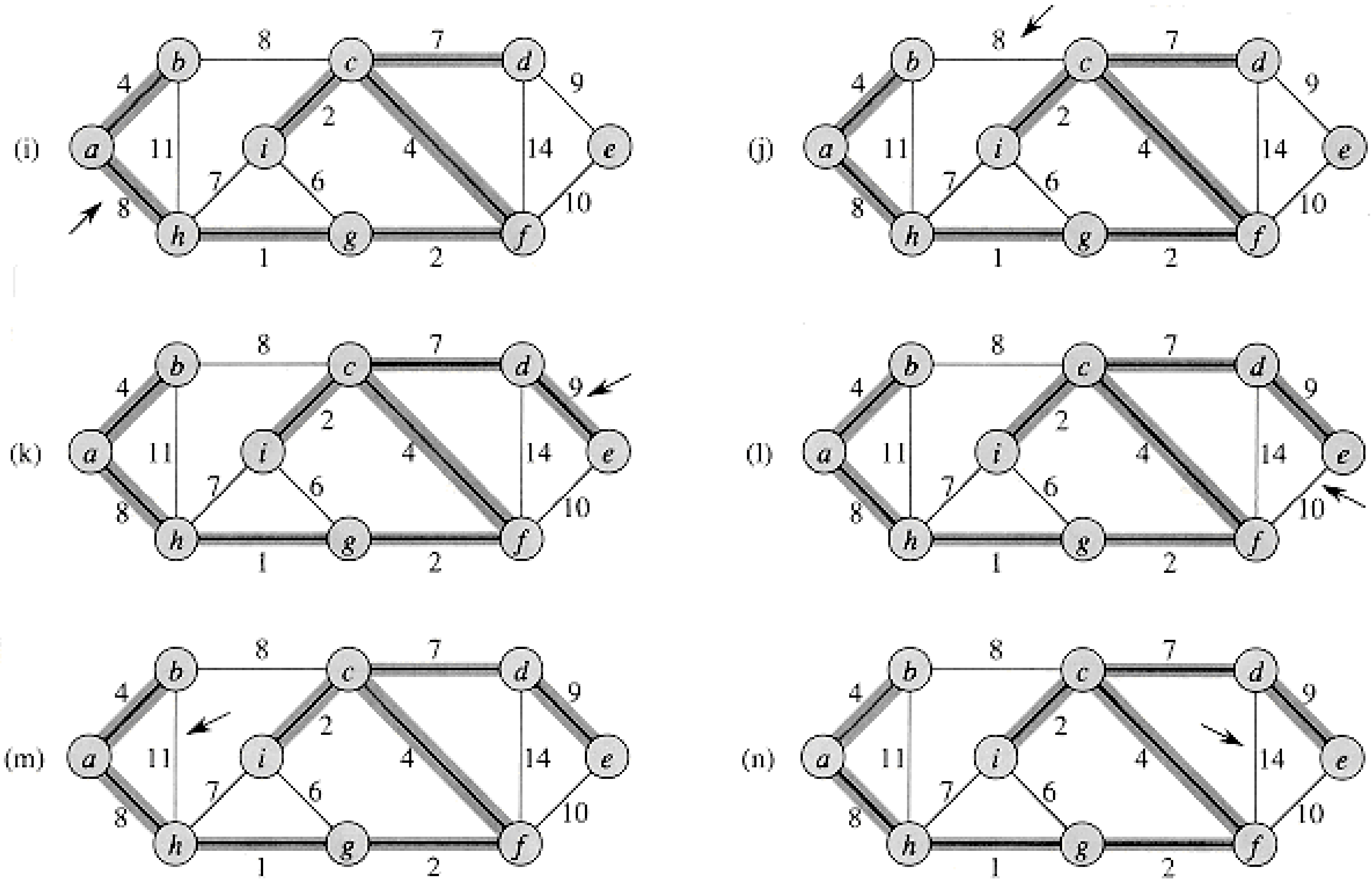
PRIM'S AND KRUSKAL'S



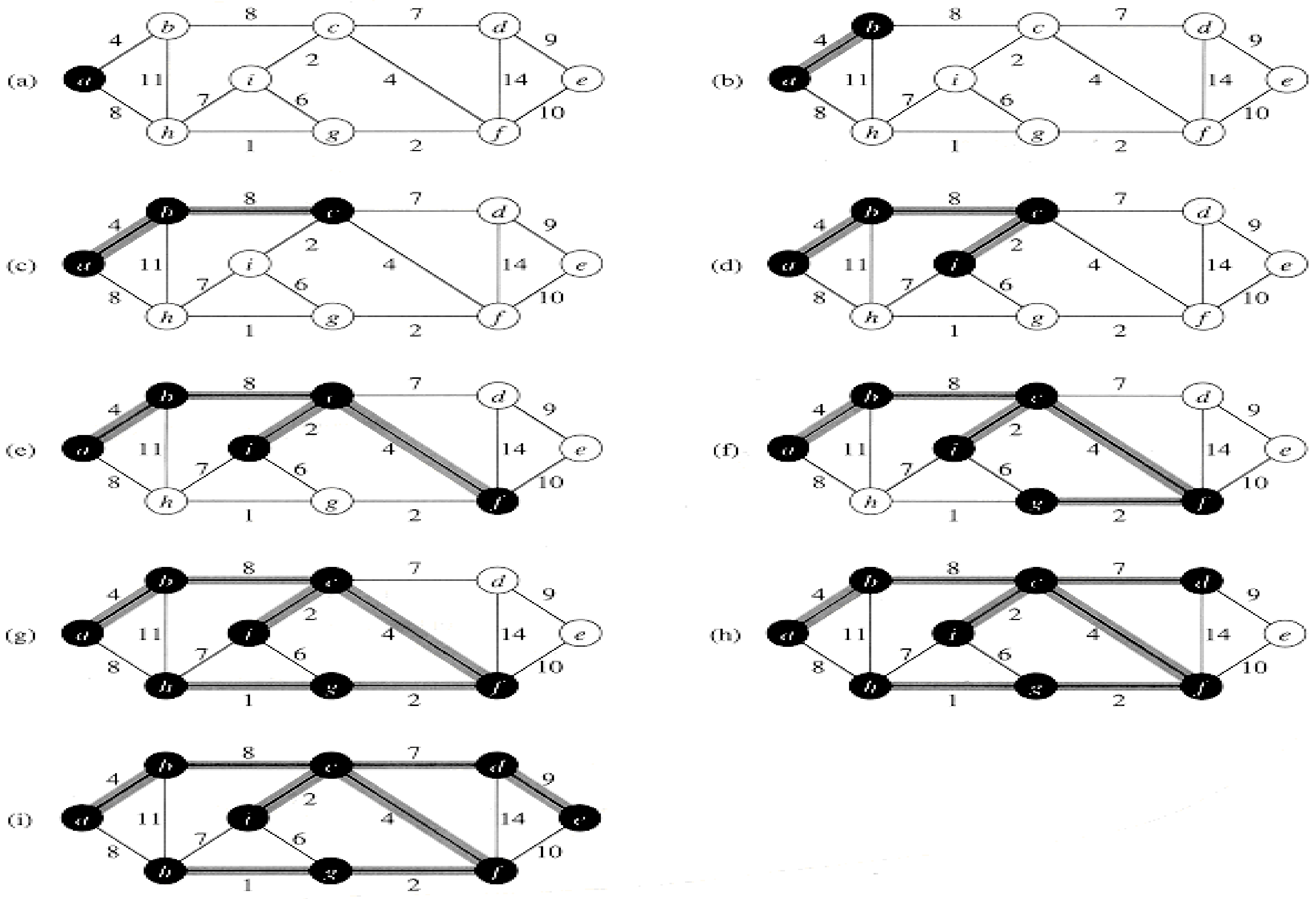
Kruskal’s Algorithm – Example I



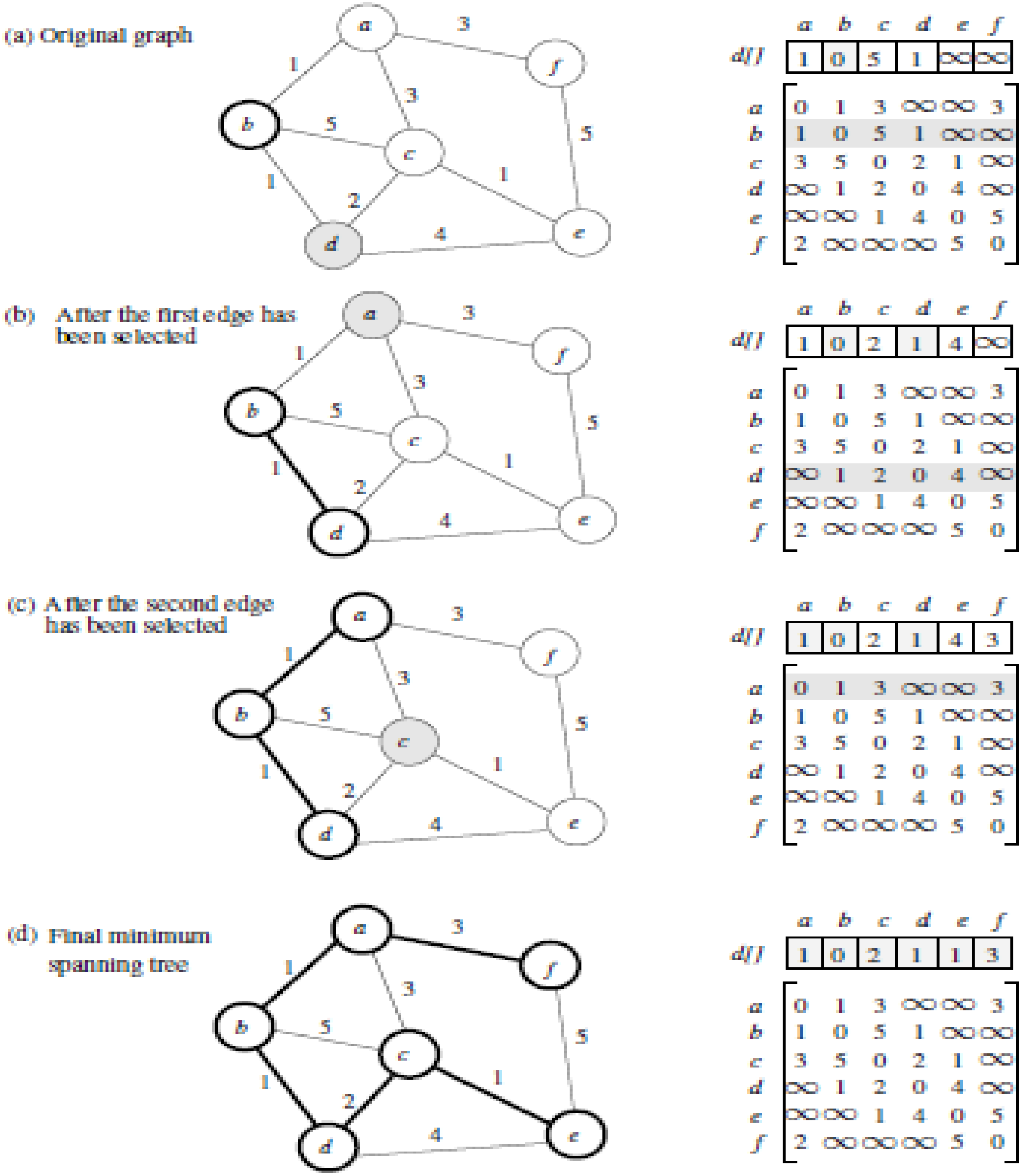
Kruskal's Algorithm – Example I (contd.)



Prim’s Algorithm – Example I



Prim's Algorithm – Example II



Prim's Algorithm – Pseudocode

procedure PRIM_MST(V, E, w, r)
begin
    $V_T := \{r\};$
    $d[r] := 0;$
    for all $v \in (V - V_T)$ **do**
        if edge (r, v) exists set $d[v] := w(r, v);$
        else set $d[v] := \infty;$
    while $V_T \neq V$ **do**
    begin
        find a vertex u such that $d[u] := \min\{d[v] | v \in (V - V_T)\};$
        $V_T := V_T \cup \{u\};$
        for all $v \in (V - V_T)$ **do**
            $d[v] := \min\{d[v], w(u, v)\};$
    endwhile
end PRIM_MST

Prim's Complexity

- For a sparse graph with $E \approx kV$ (k some constant), the complexity is $O(E \log_2 V)) = O(V \log_2 V))$

- For a dense graph $E \approx V^2$, the complexity is $O(E \log_2 V)) = O(V^2 \log_2 V))$ if adjacency list is used. However with adjacency matrix it can be reduced to $O(V^2)$

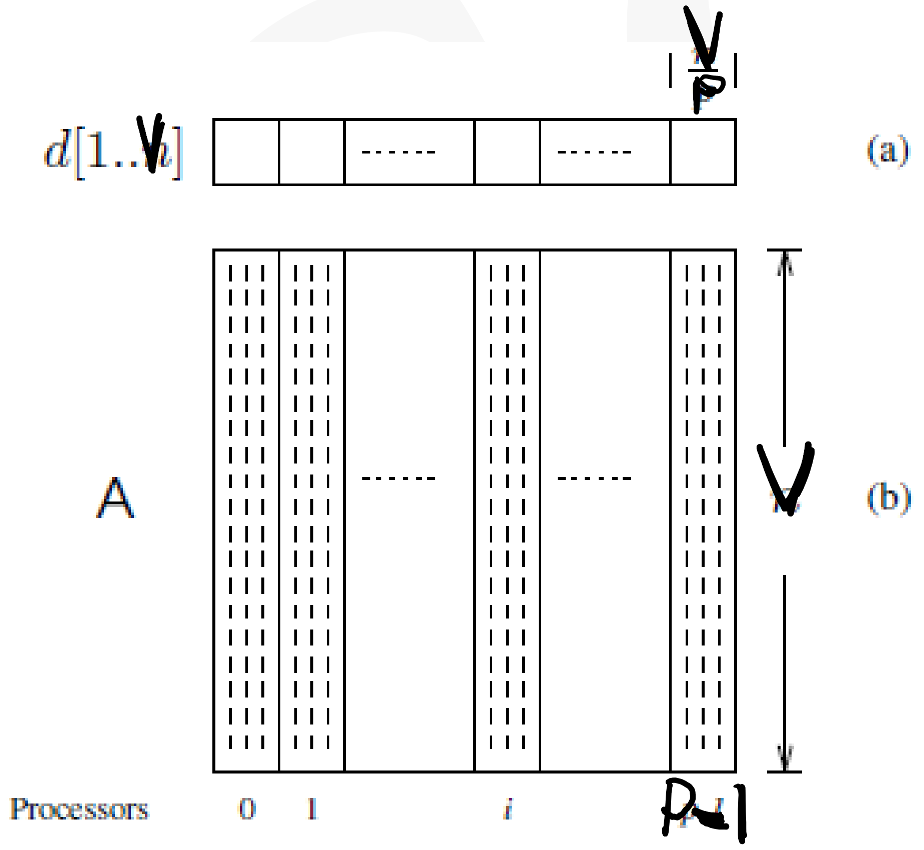
- Note: Adjacency matrix would result in $O(V^2)$ even for sparse graphs, therefore for sparse graphs we better use adjacency list

Prim's Complexity – Parallelism

- The algorithm works in n outer iterations . It is hard to execute these iterations concurrently
- The inner loop is relatively easy to parallelize. Let P be the number of processes, and let V be the number of vertices
- The adjacency matrix is partitioned in a 1-D block fashion, with distance vector d partitioned accordingly
- In each step, a processor selects the locally closest node, followed by a global reduction to select globally closest node
- This node is inserted into MST, and the choice broadcast to all processors
- Each processor updates its part of the d vector locally

Prim's Complexity – Parallelism (contd.)

- Parallel processing in Prime's Algorithm:
  - The partitioning of the distance array d and the adjacency matrix A among p processes

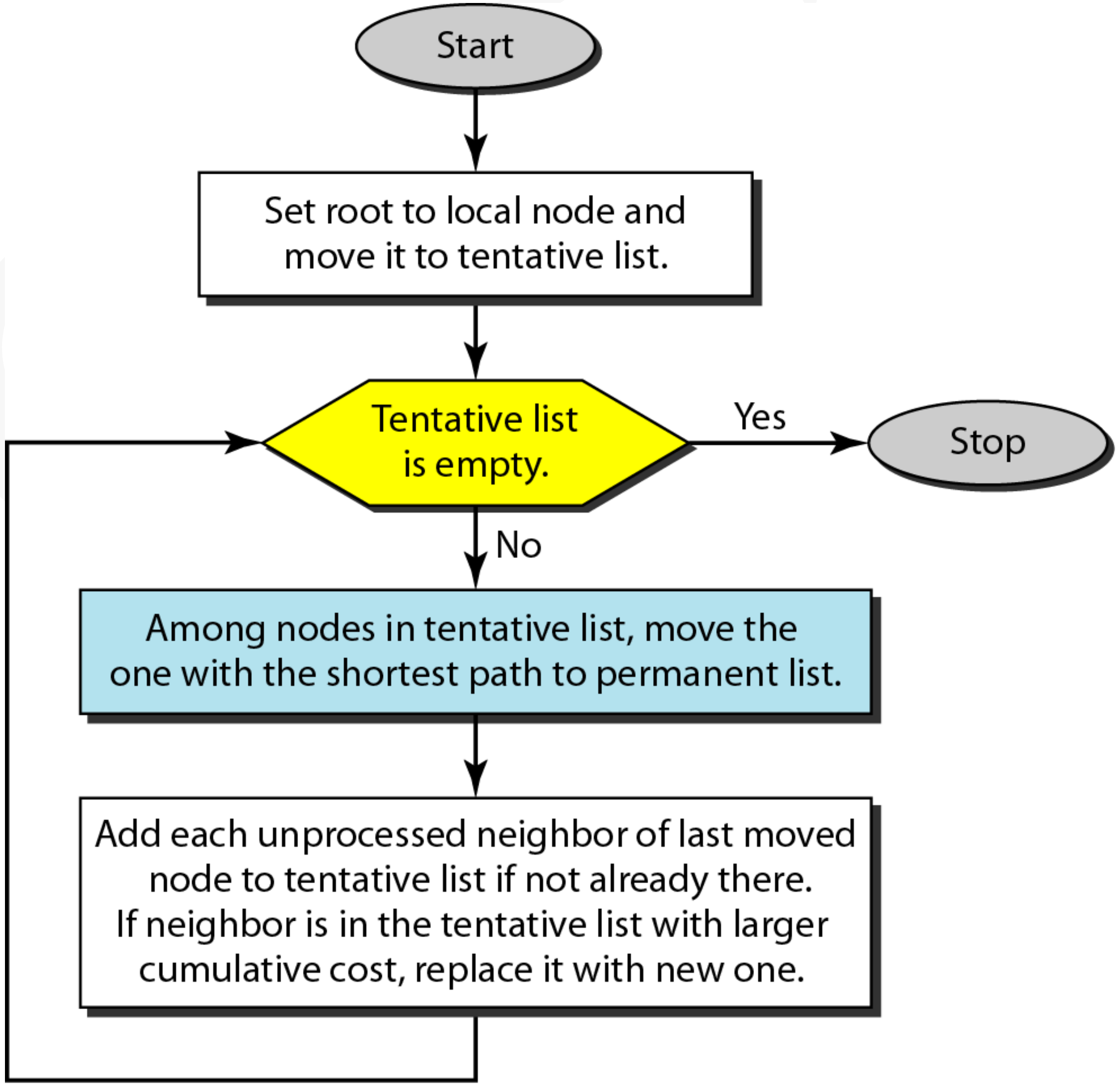


Prim's Complexity – Parallelism (contd.)

- The cost to select the minimum entry is O(V/P + log P)
- The cost of a broadcast is O(log P)
- The cost of local updating of the d vector is O(V/P)
- The parallel time per iteration is O(V/P + log P)
- The total parallel time is given by $O(V^2/P + V \log P)$
- The corresponding isoefficiency is O(p2 log2 p)

Dijkstra's Algorithm – Flowchart

- The flowchart is short and simple which means Dijkstra's algorithm is simple



Dijkstra's Algorithm – Notation

Dijkstra's algorithm

- Net topology, link costs known to all nodes
  - Accomplished via "link state broadcast"
  - All nodes have same info
- Computes least cost paths from one node ('source") to all other nodes
  - Gives **forwarding table** for that node
- Iterative: after k iterations, know least cost path to k dest.'s

Notation:

- **c(x,y):** link cost from node x to y; = ∞ if not direct neighbors
- **D(v):** current value of cost of path from source to dest. v
- **p(v):** predecessor node along path from source to v
- **N':** set of nodes whose least cost path definitively known

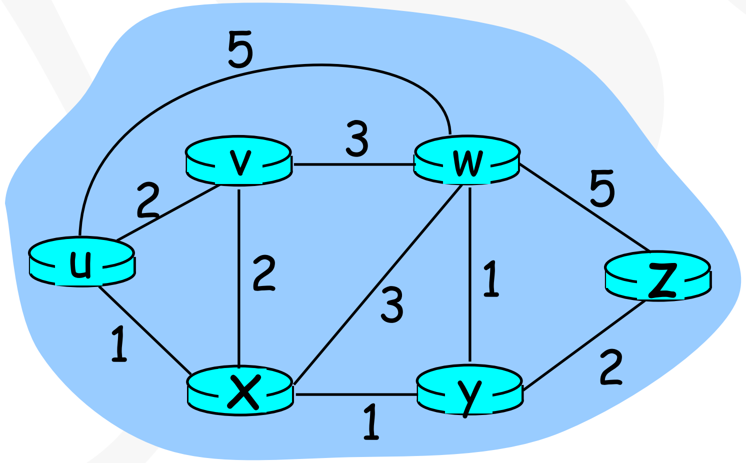
Dijkstra's Algorithm – Pseudocode

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
     D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

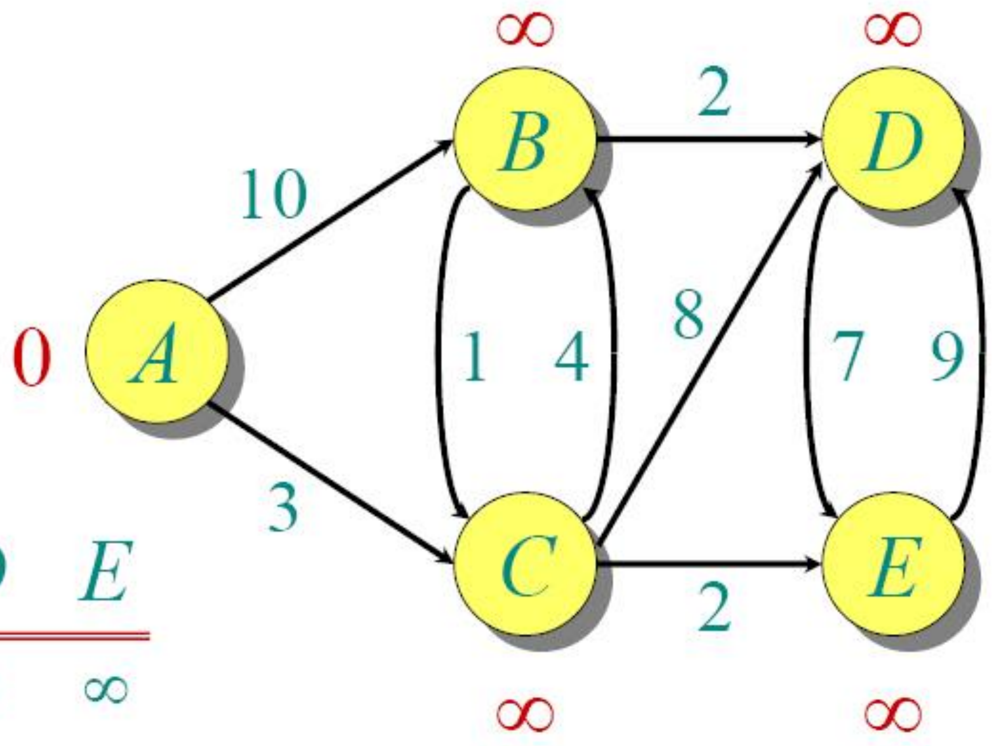
Dijkstra’s Algorithm: Example I

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Dijkstra's Algorithm: Example II

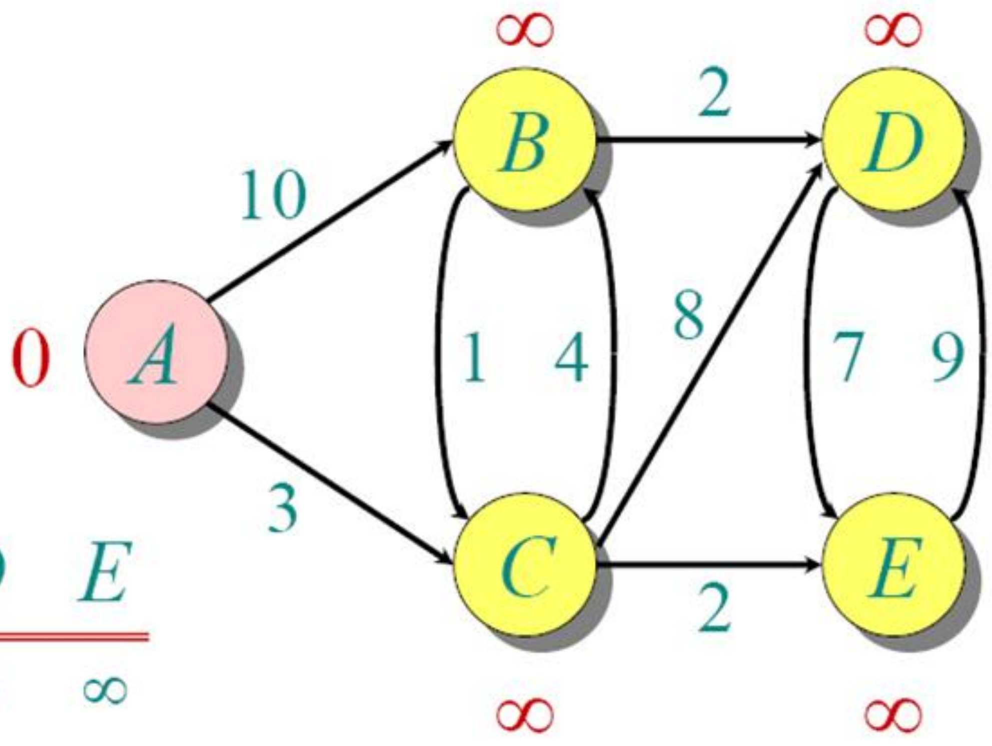
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

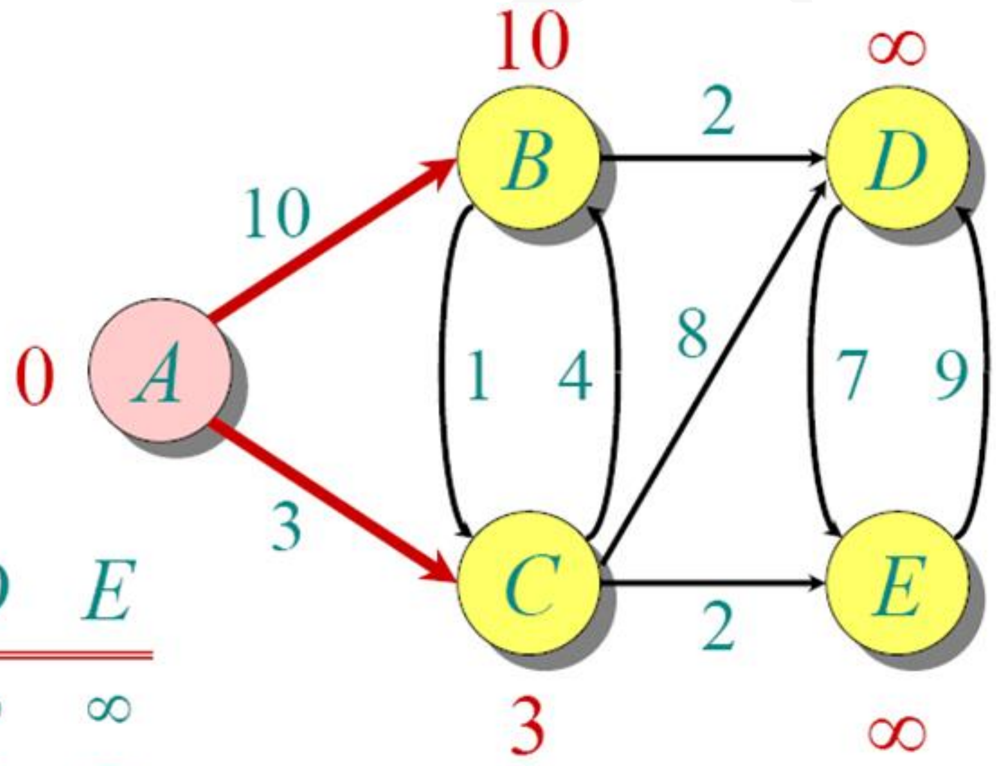
S: {}

Dijkstra's Algorithm: Example II (contd.)



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

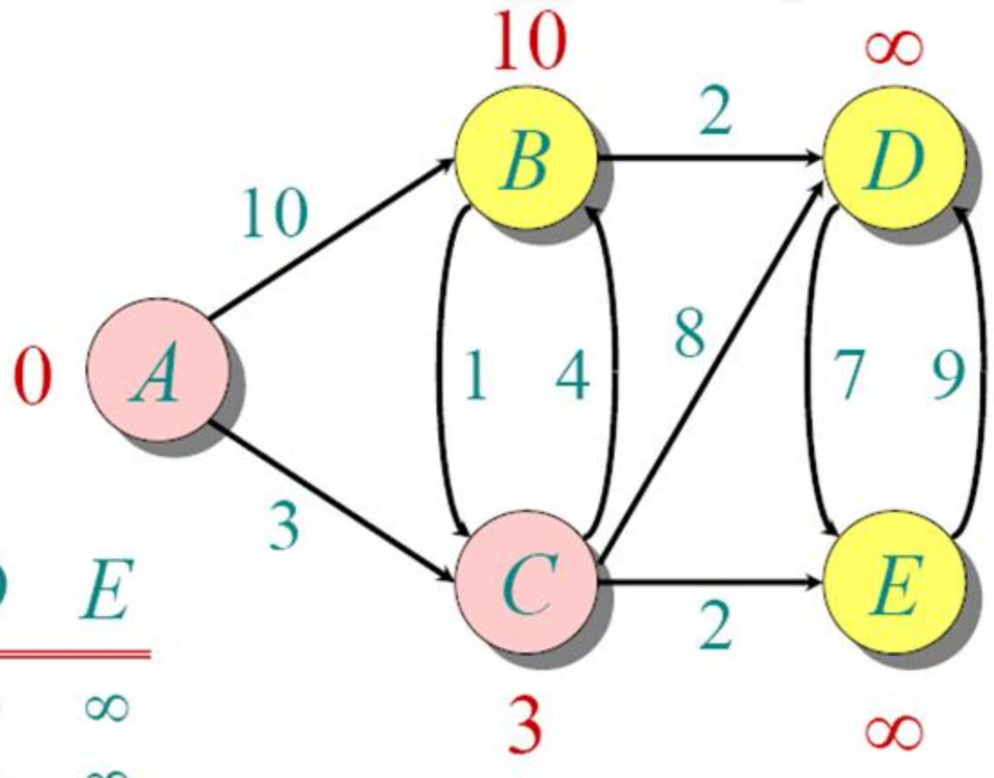
Dijkstra's Algorithm: Example II (contd.)



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

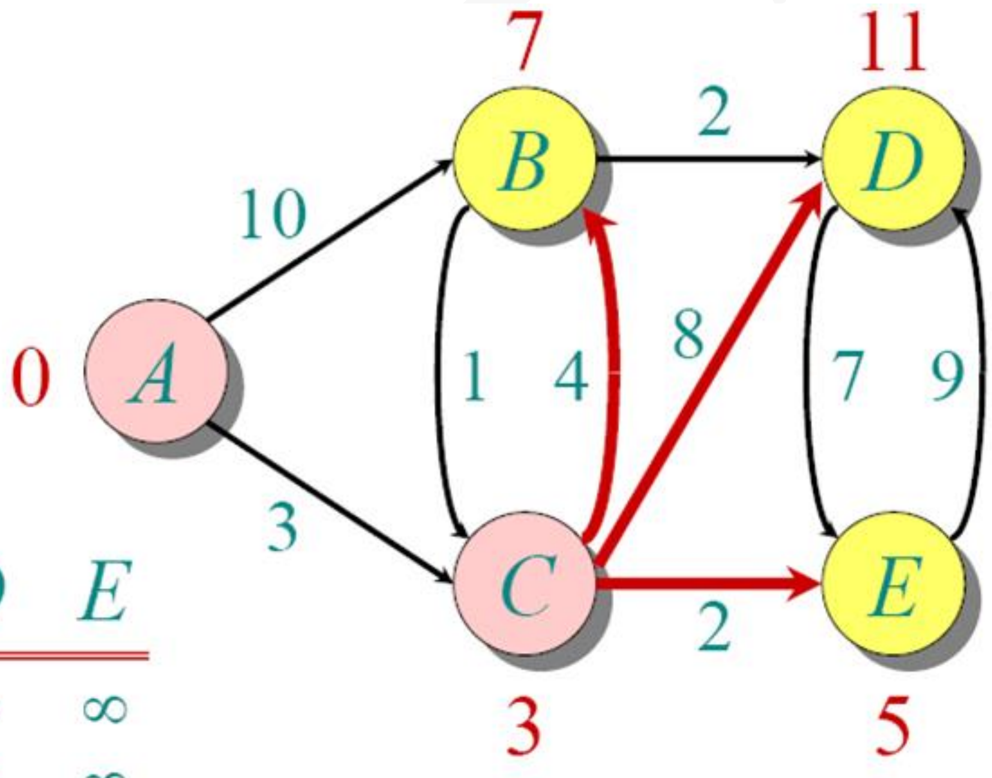
Example II (contd.)



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

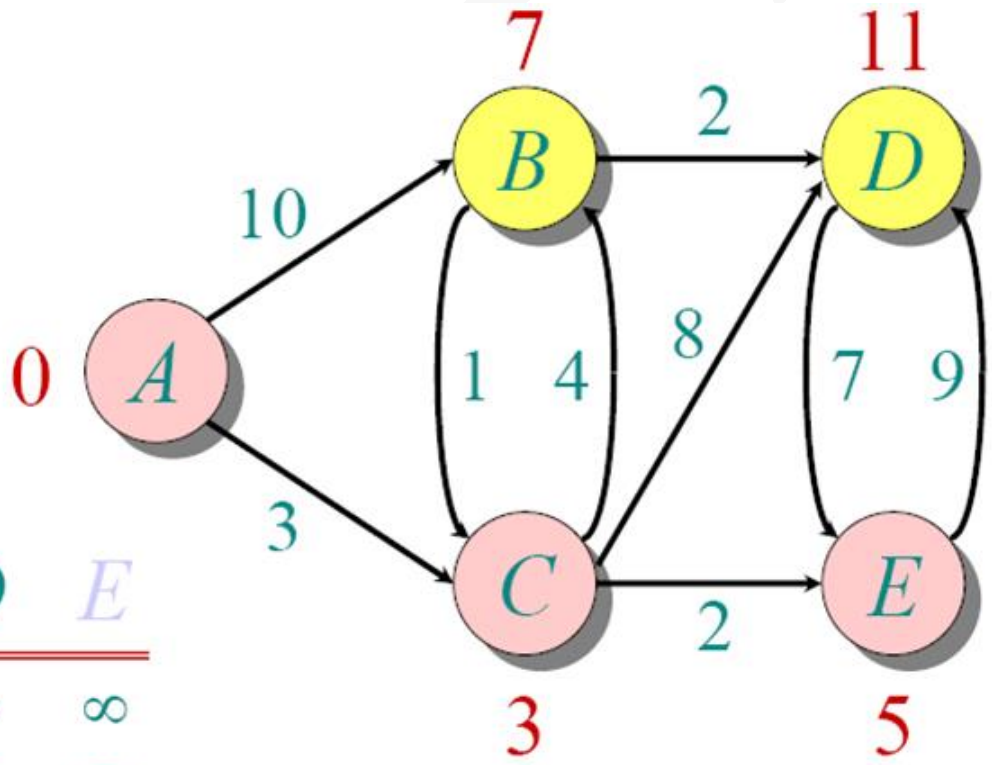
Example II (contd.)



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	5

S: { A, C }

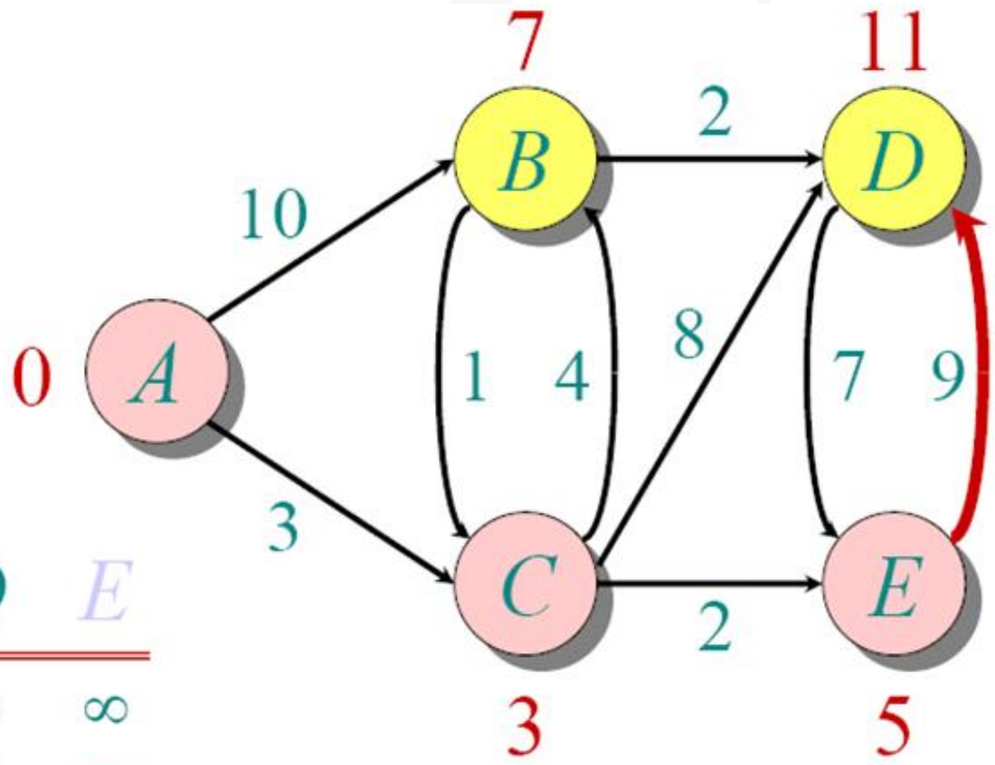
Example II (contd.)



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

S: { A, C, E }

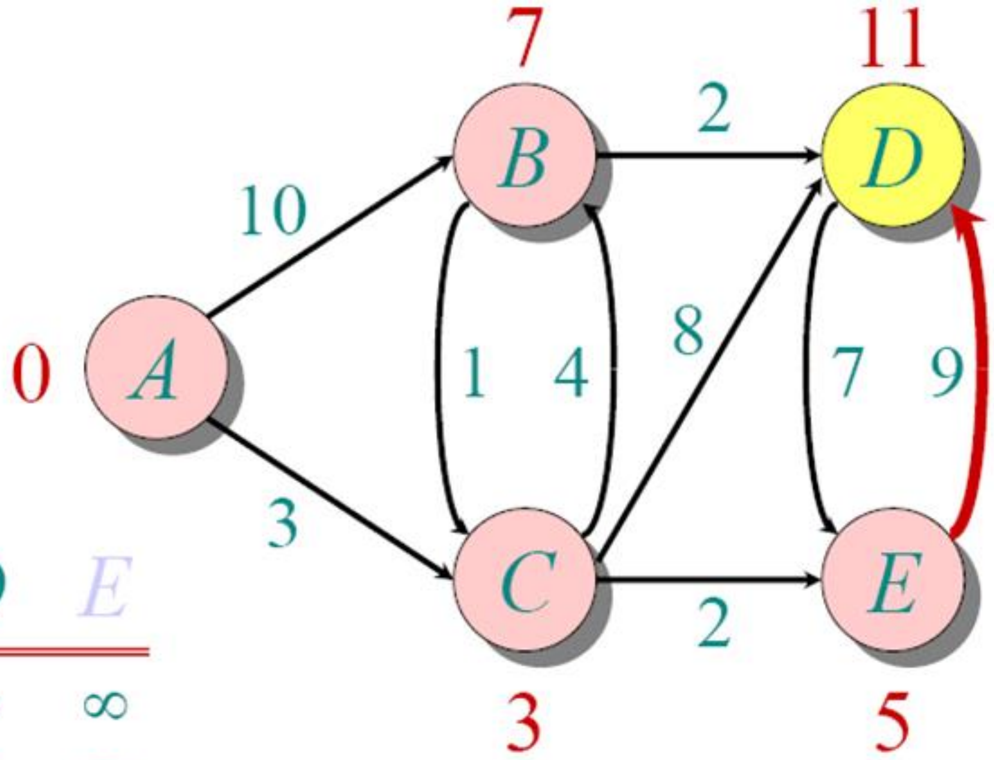
Example II (contd.)



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

S: { A, C, E }

Example II (contd.)

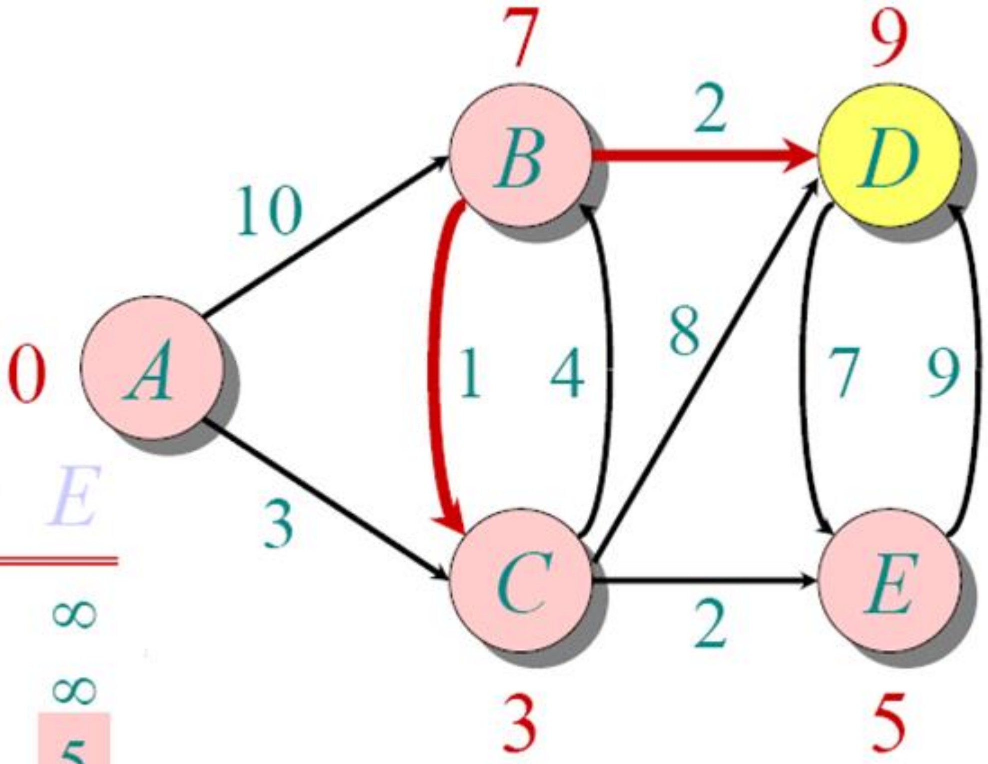


Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	

S: { A, C, E, B }

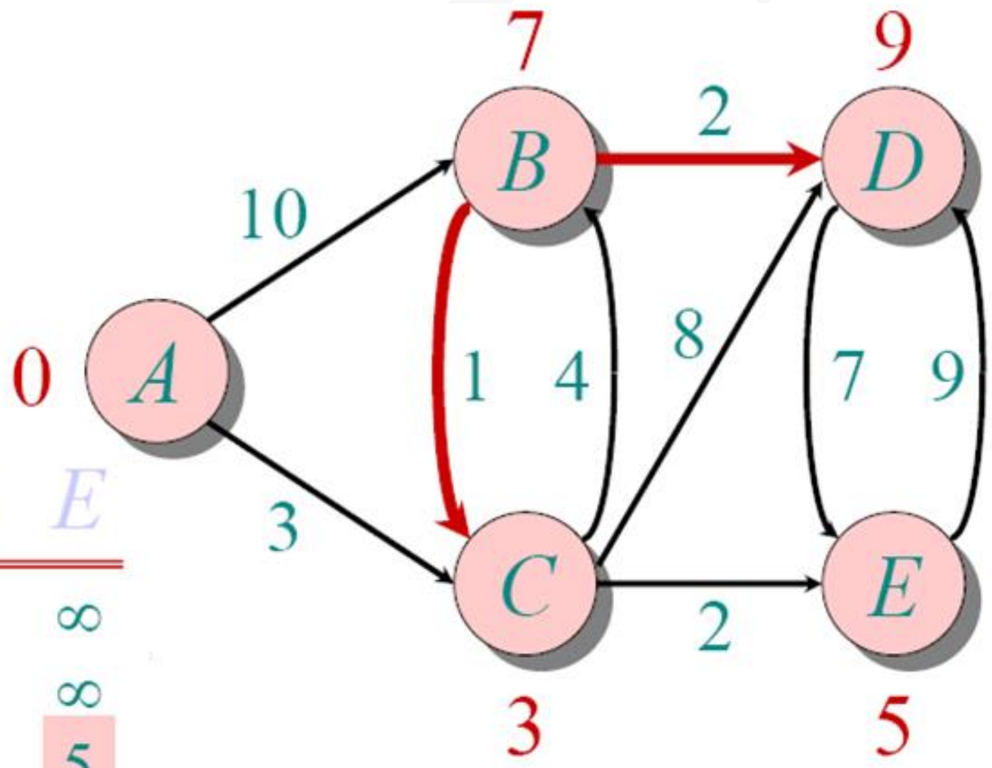
Example II (contd.)

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



S: { A, C, E, B }

Example II (contd.)

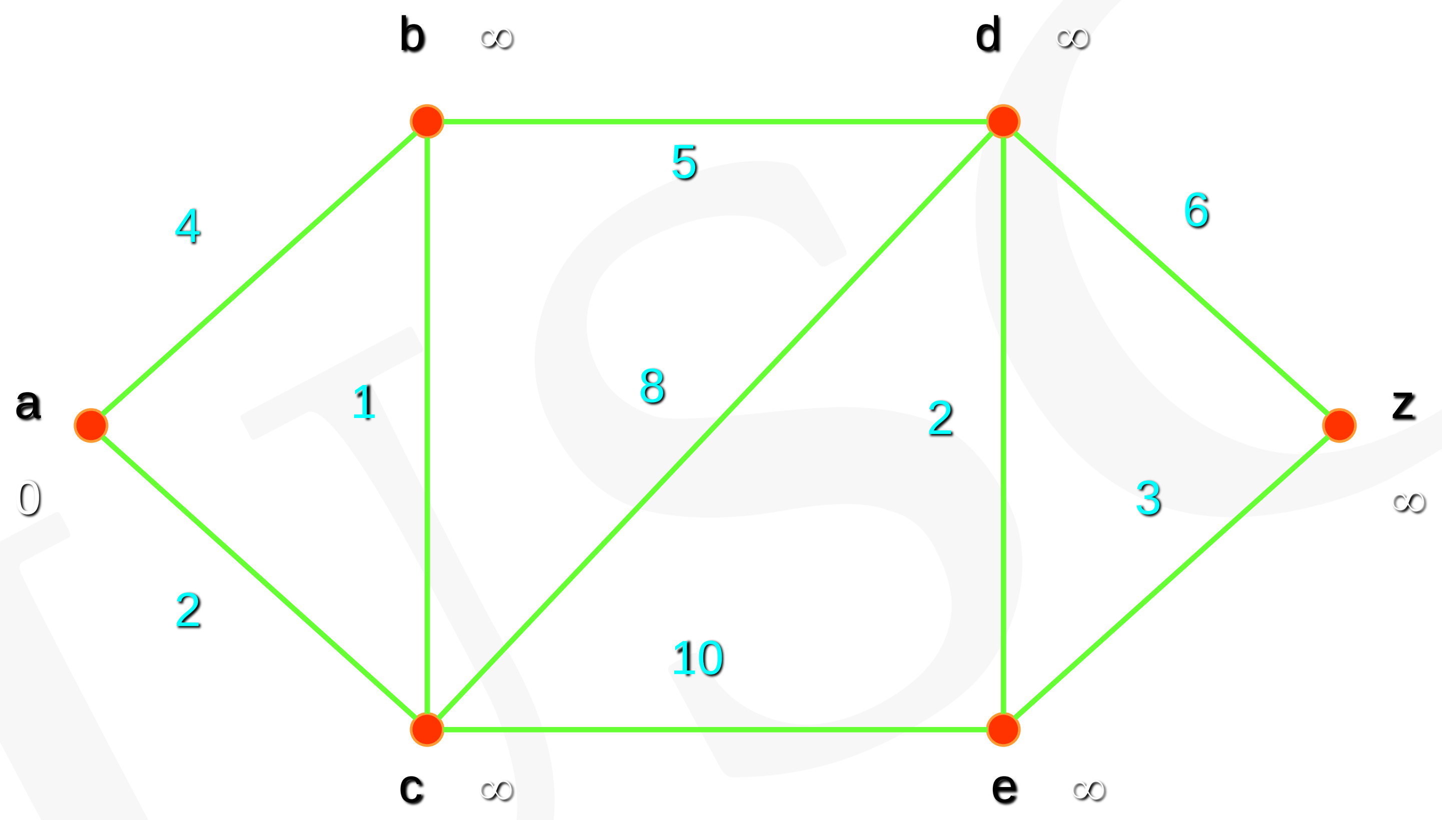


Q:

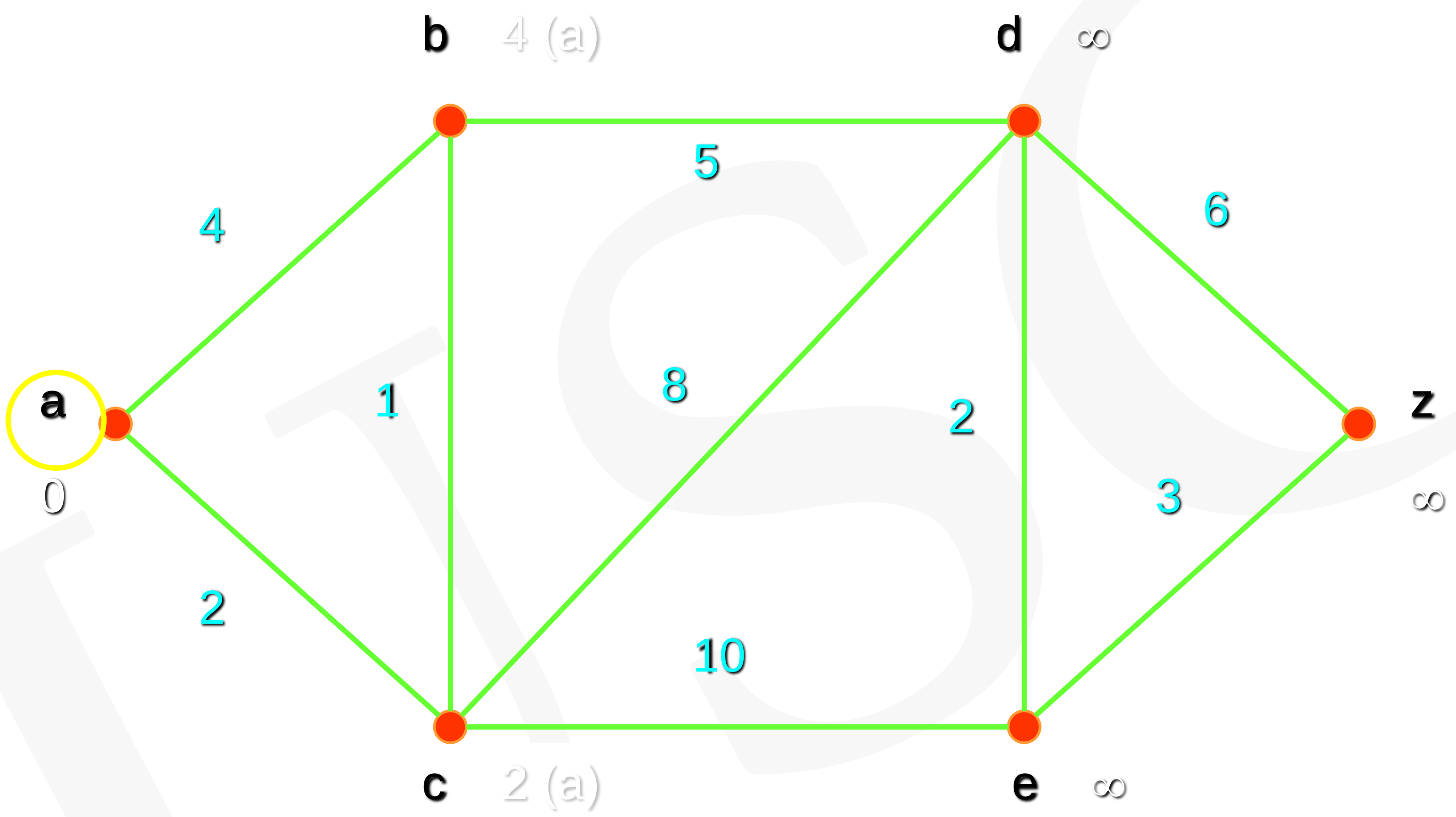
A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

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

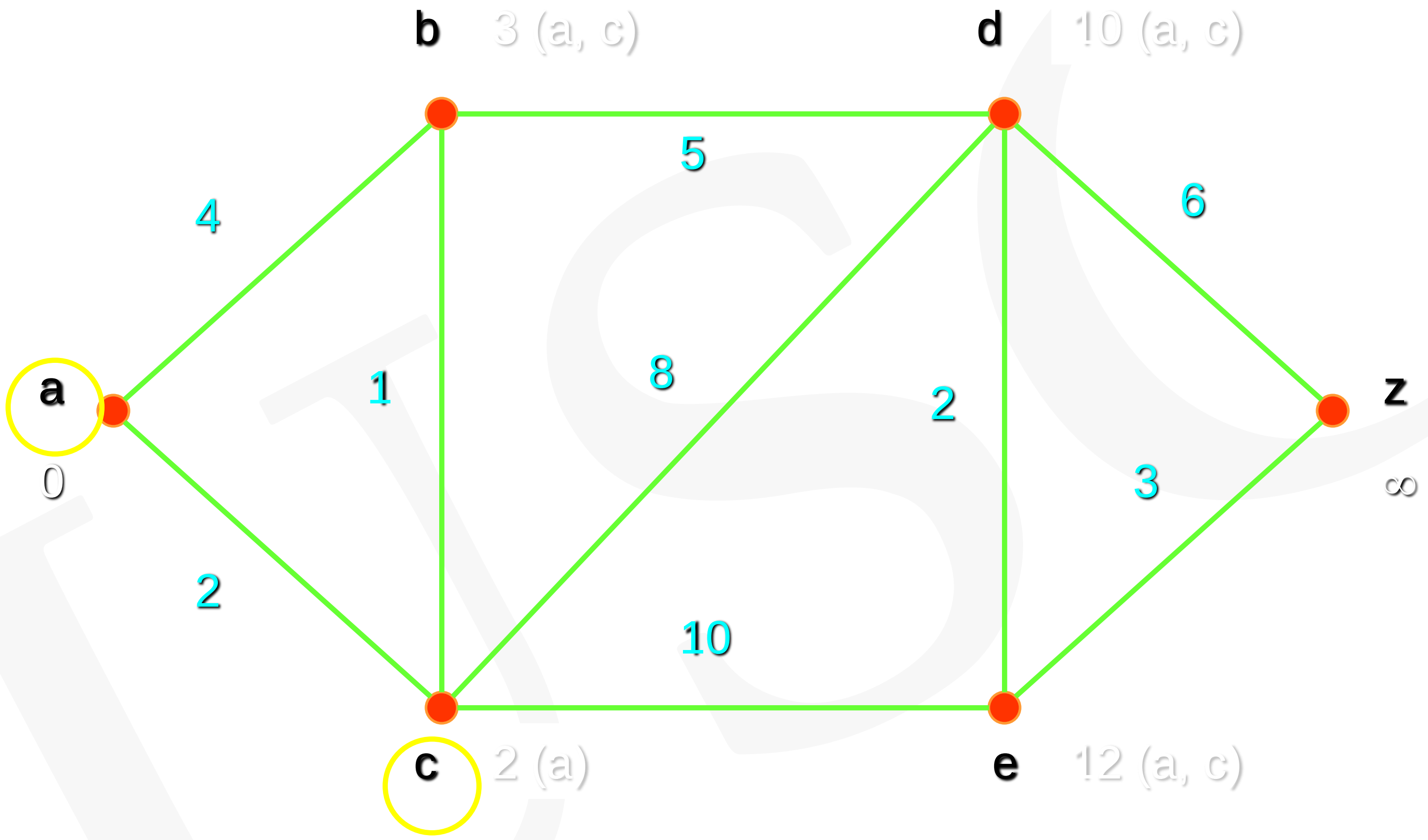
Example III



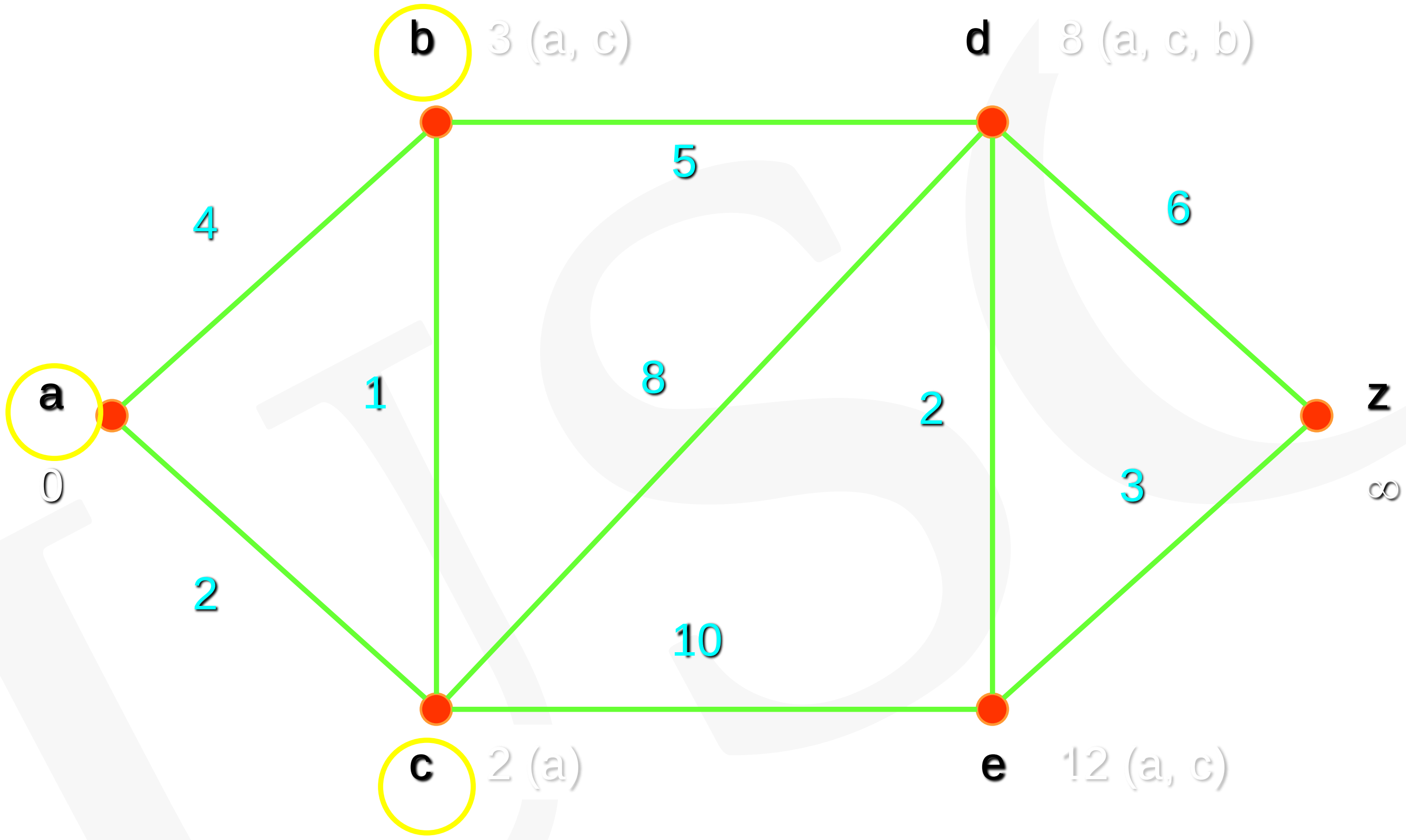
Example III (contd.)



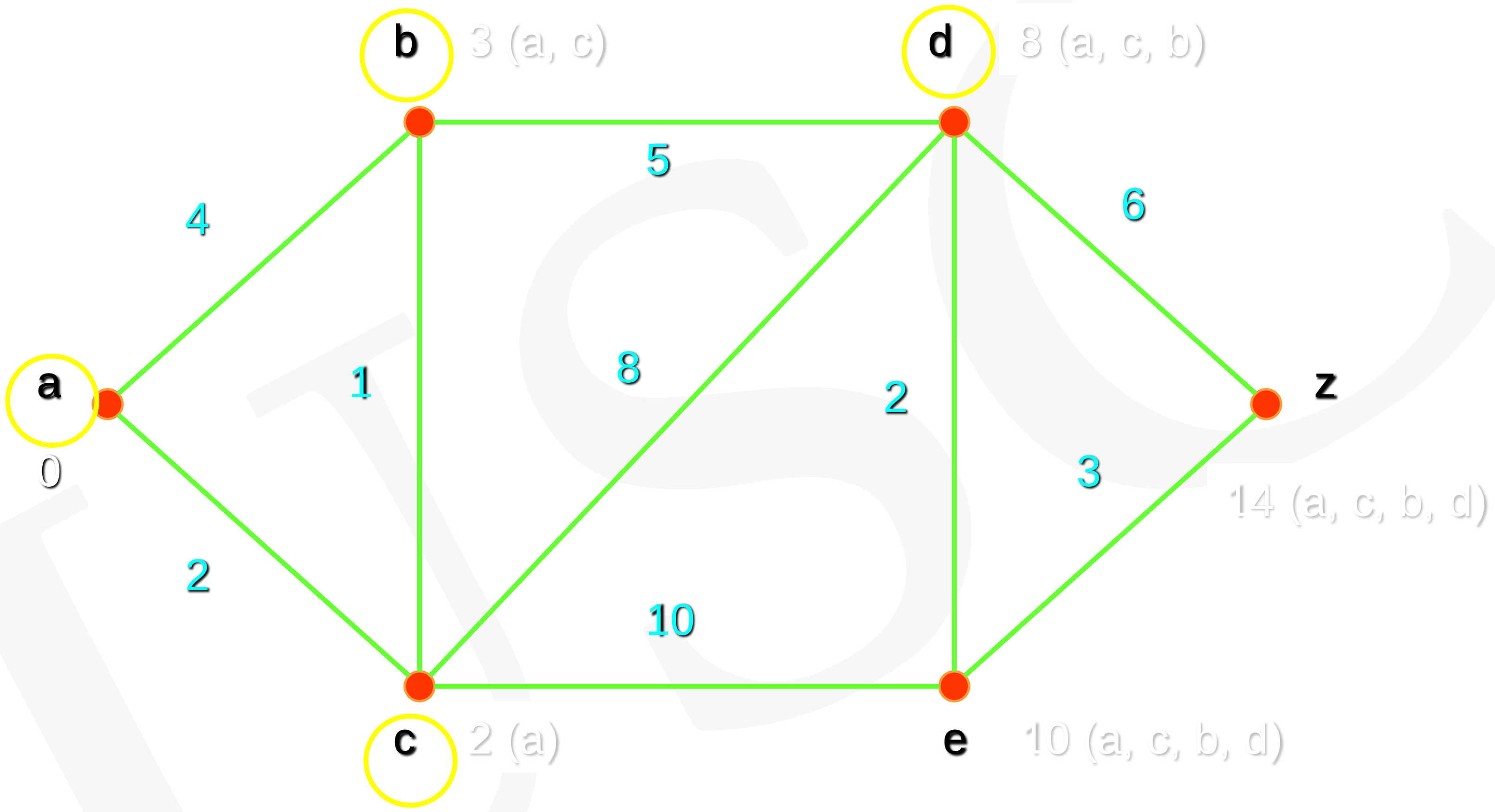
Example III (contd.)



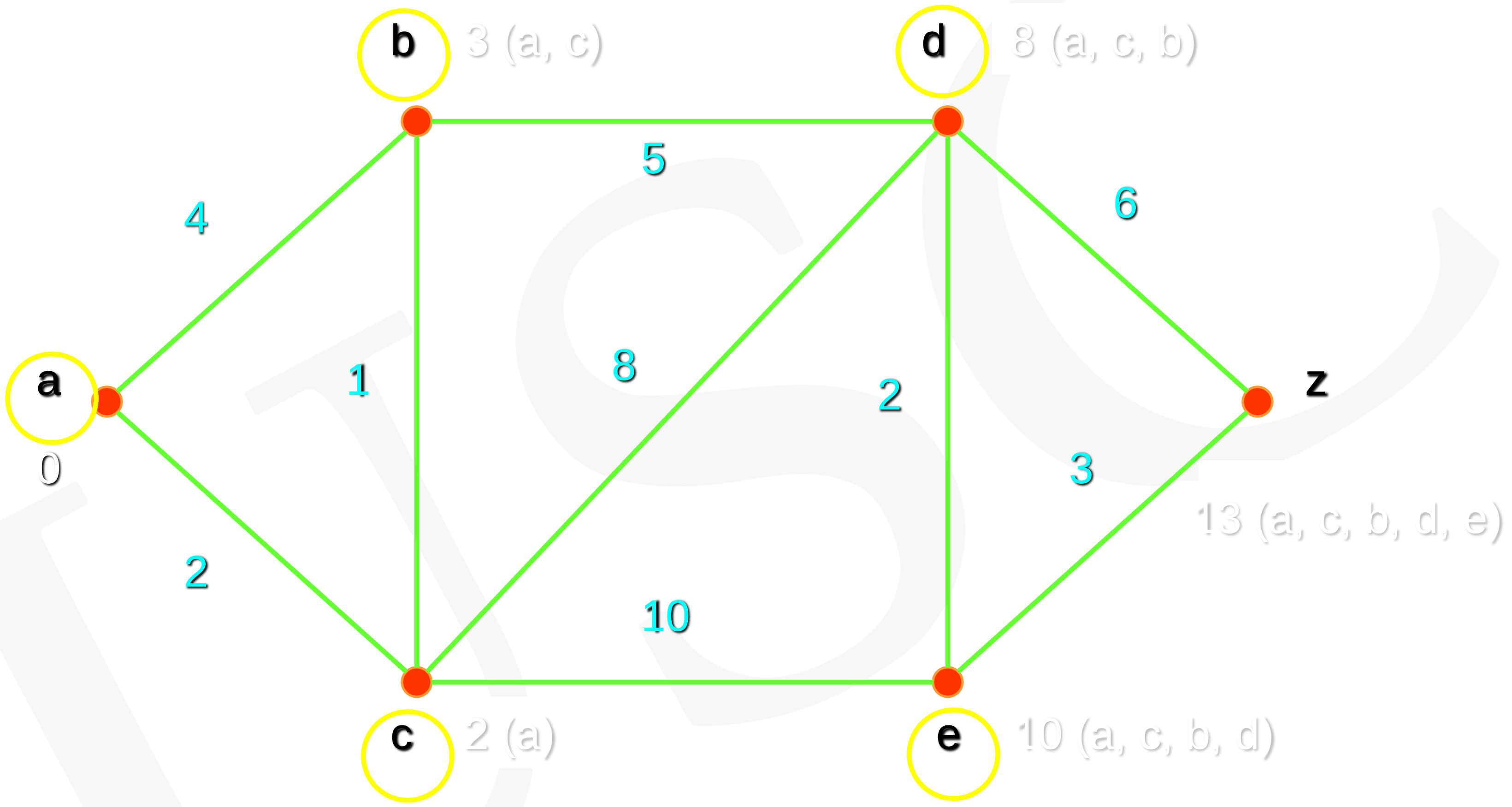
Example III (contd.)



Example III (contd.)



Example III (contd.)



Example III (contd.)