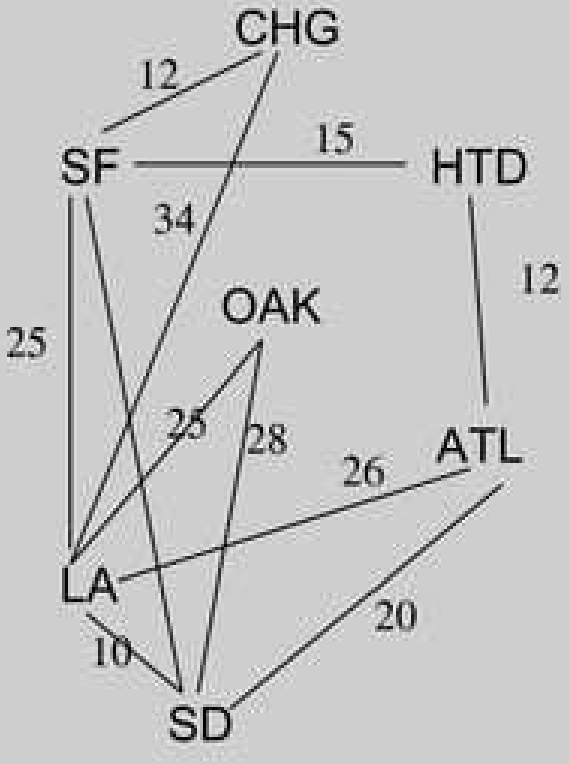
Weighted graphs

Example Consider the following graph, where nodes represent cities, and edges show if there is a direct flight between each pair of cities.



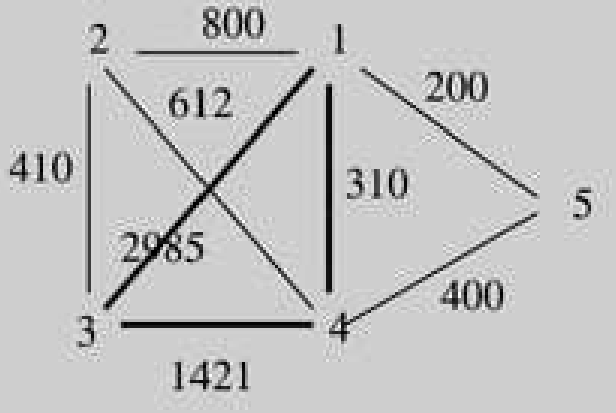
V = {SF, OAK, CHG, HTD, ATL, LA, SD}

E = {{SF, HTD}, {SF, CHG}, {SF, LA}, {SF, SD}, {SD, OAK}, {CHG, LA}, {LA, OAK}, {LA, ATL}, {LA, SD}, {ATL, HTD}, {SD, ATL}}

Problem formulation: find the "best" path between two vertices $v_1, v_2 \in V$ in graph G = (V, E). Depending on what the "best" path means, we have 2 types of problems:

- The **minimum spanning tree problem**, where the "best" path means the "lowest-cost" path.
- The **shortest path problem**, where the "best" path means the "shortest" path.

Note that here edge weights are not necessarily Euclidean distances. Example:

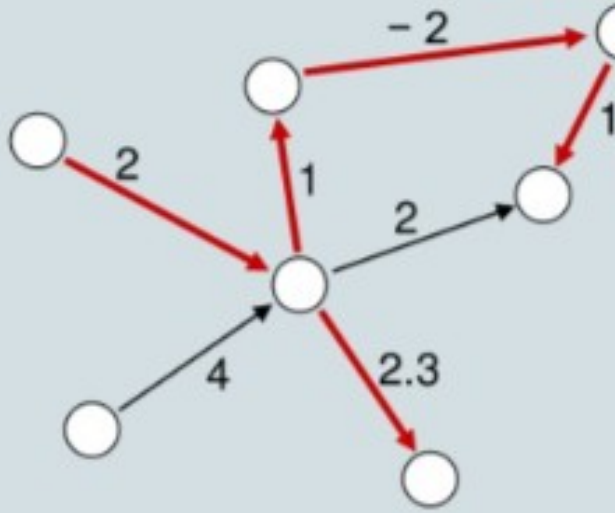


2985 > 1421 + 310, not the case here, however.

optimization problems on graphs

single-source shortest paths

Find shortest path from source vertex to all other vertices

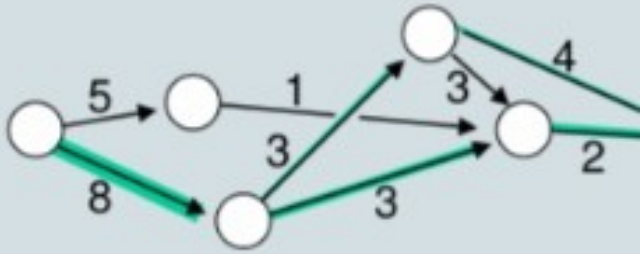


all-pairs shortest paths

Find shortest paths between all pairs of vertices

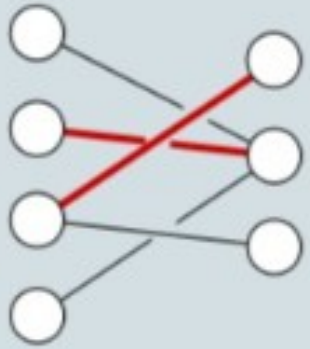
maximum flow

Find maximum flow from source vertex to target vertex



maximum bipartite matching

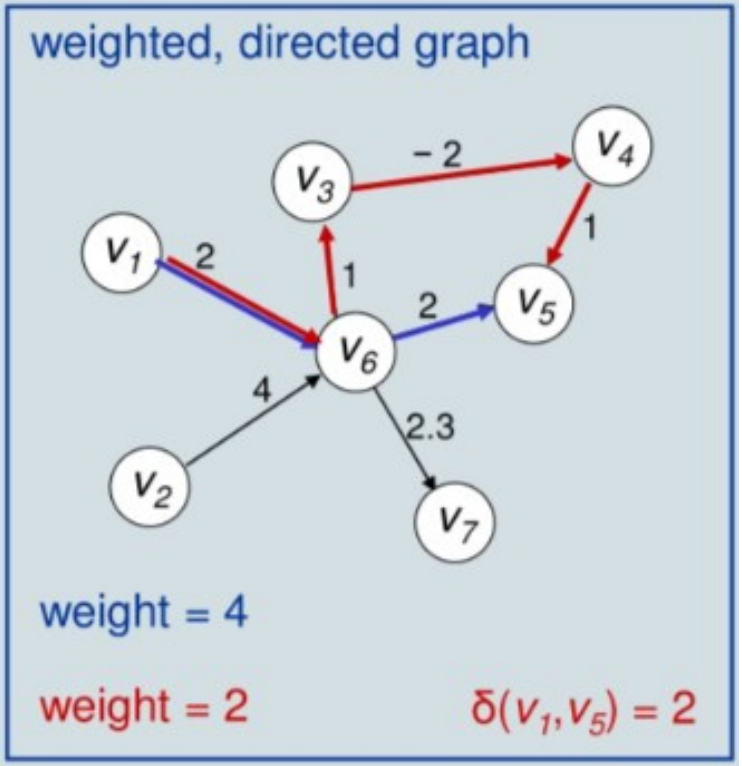
Find maximum number of disjoint pairs of vertices with edge between them



Shortest paths

weighted, directed graph $G = (V,E)$

- weight (or: length) of a path
  = sum of edge weights
- $\delta(u,v)$
  = distance from u to v
  = min weight of any path from u to v
- shortest path from u to v
  = any path from u to v of weight $\delta(u,v)$



Is $\delta(u,v)$ always well defined?

No, not if there are negative-weight cycles.

If $\delta(u,v)$ well defined, then there is a shortest path with at most $|V|-1$ edges

The Single-Source Shortest-Paths Problem

Input:

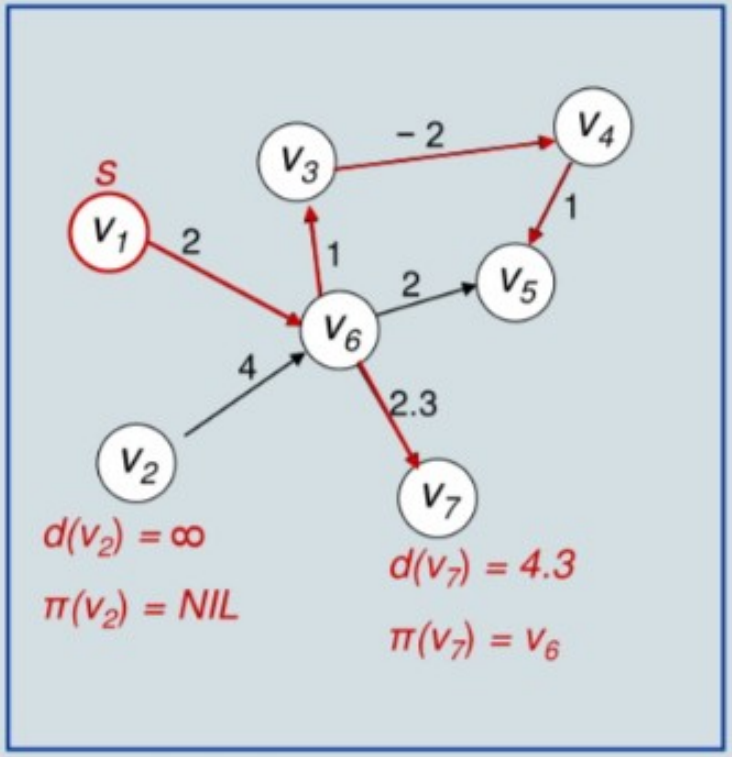
- graph G = (V,E)
- source vertex $s \in V$

output

- for each $v \in V$: distance $\delta(s,v)$
  store distance in attribute $d(v)$
  if v not reachable: $\delta(s,v) = \infty$

- a shortest-path tree that encodes all shortest paths from s:

  for $v \neq s$ the predecessor $\pi(v)$ on a shortest path from s to v



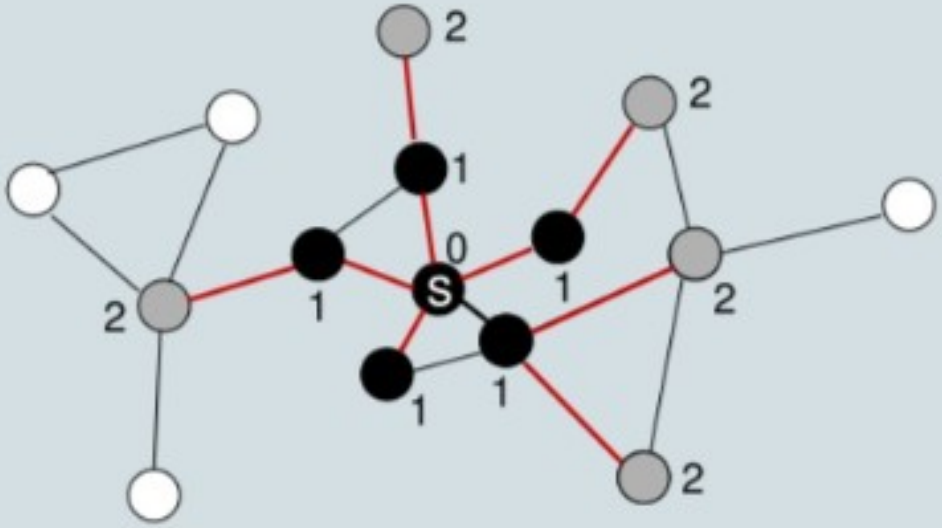
book (3rd ed) uses $v.d$ and $v.\pi$ instead of $d(v)$ and $\pi(v)$

NB if negative-weight cycle is reachable from s, then we only need to report this fact

The SSSP problem on unweighted graphs

Breadth-First Search (BFS)

running time:
$\Theta (|V| + |E|)$



round *i*:

for each vertex *u* at distance *i-1* from *s* (that is, with *d(v)=i-1*)

do for all neighbors *v* of *u*

do if *v* unvisited **then** set *d(v)=i*

implementation: maintain queue *Q*

initial part of *Q:* nodes with *d(v)=i-1* that have not been handled yet

rest of Q: nodes with *d(v)=i* with a neighbor that has been handled

Theorem: Dijkstra's algorithm solves the Single-Source Shortest-Path Problem for graphs with non-negative edge weights in $O(|V| \log |V| + |E|)$ time.

The SSSP problem on weighted graphs

Dijkstra's algorithm

- works only for non-negative edge weights
- running time $\Theta (|V| \log |V| + |E|)$

Bellman-Ford algorithm

- can handle negative edge weights, works even for negative-weig
- running time $\Theta (|V| \cdot |E|)$

Special case: directed acyclic graph (DAG)

- can handle negative edge weights
- running time $\Theta (|V| + |E|)$

Introduction

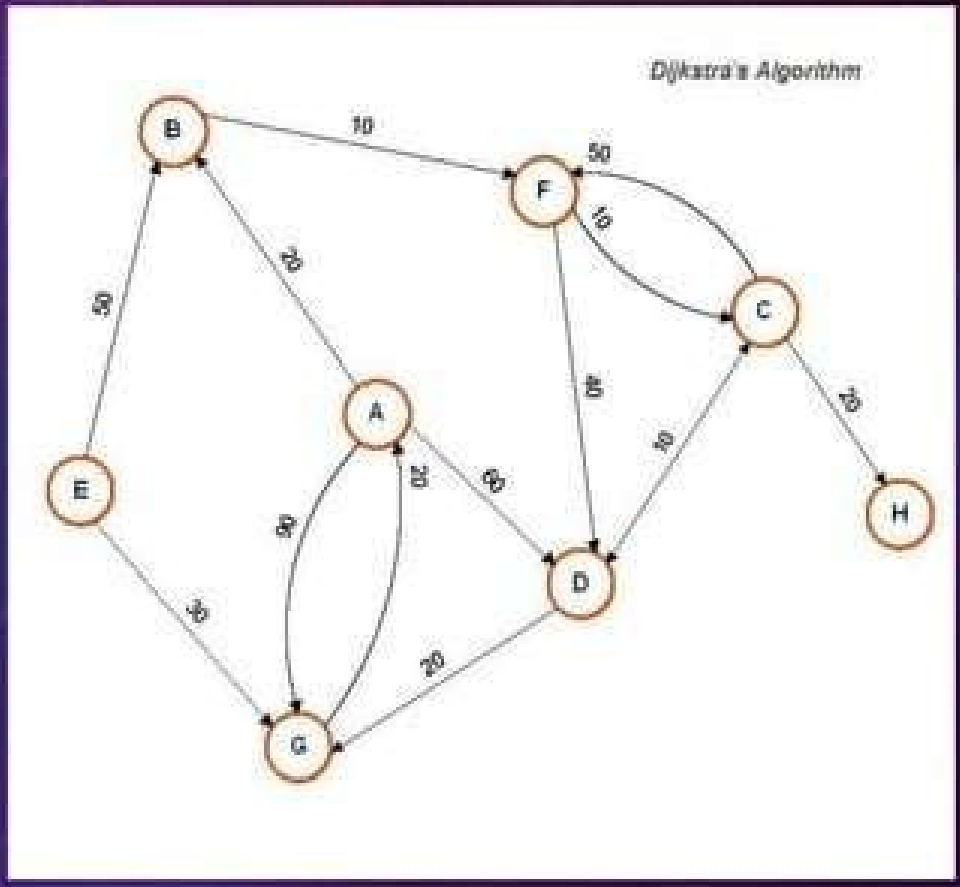
Dijkstra's Algorithm derived by a Dutch computer scientist **'Edsger Wybe Dijkstra'** in 1956 and published in 1959

How it works ?

This algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

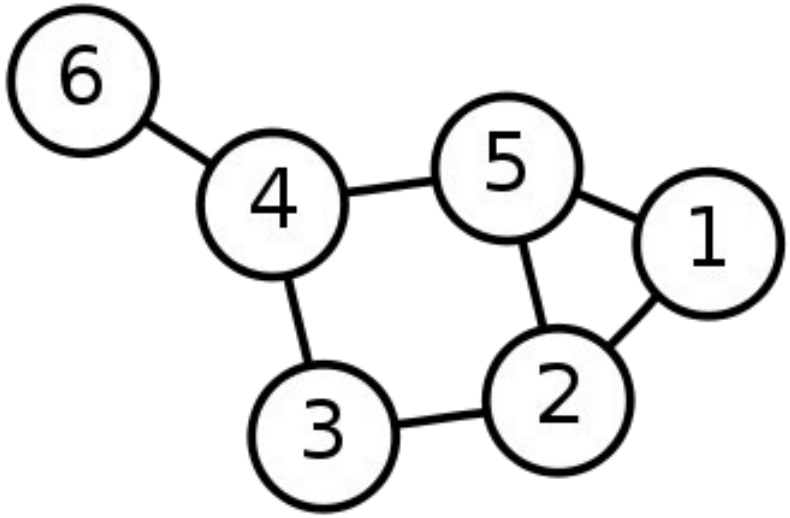
Graph Algorithm

- In this interconnected ‘Vertex’ we’ll use ‘Dijkstra’s Algorithm’.
- To use this algorithm in this network we have to start from a decided vertex and then continue to others.



Single-Source Shortest Path Problem

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

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.

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

Input:

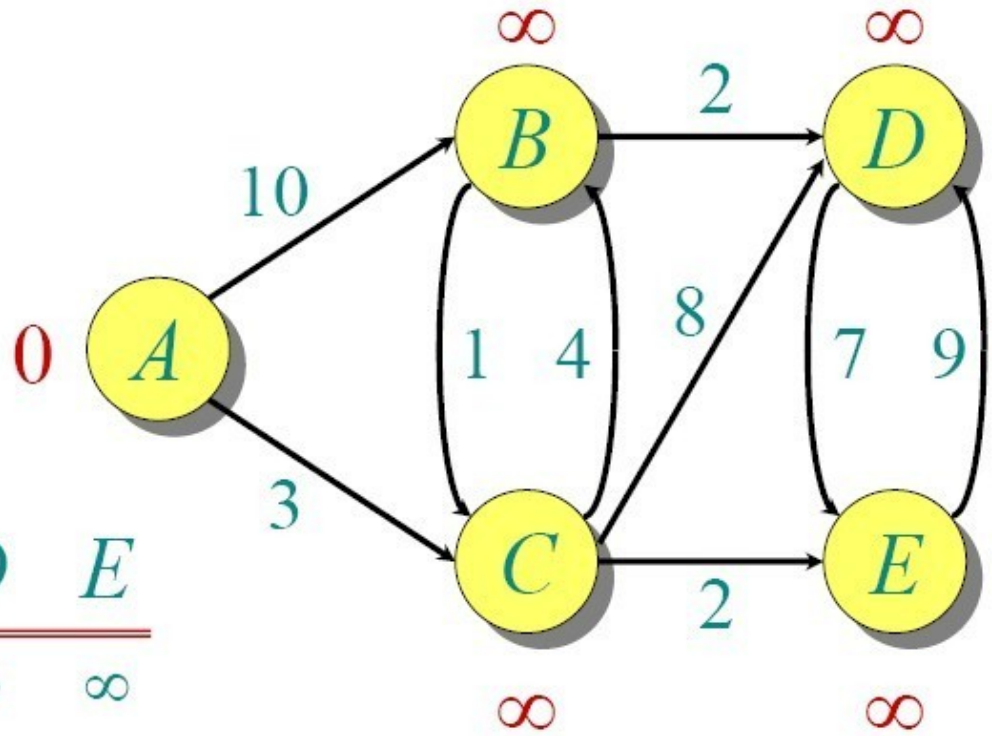
- weighted graph G(V,E) with non-negative edge weights
- source node s

Required output:

- for each vertex *v*:
  - value $d(v) = \delta(s,v)$
  - pointer $\pi(v)$ to parent in shortest-path tree

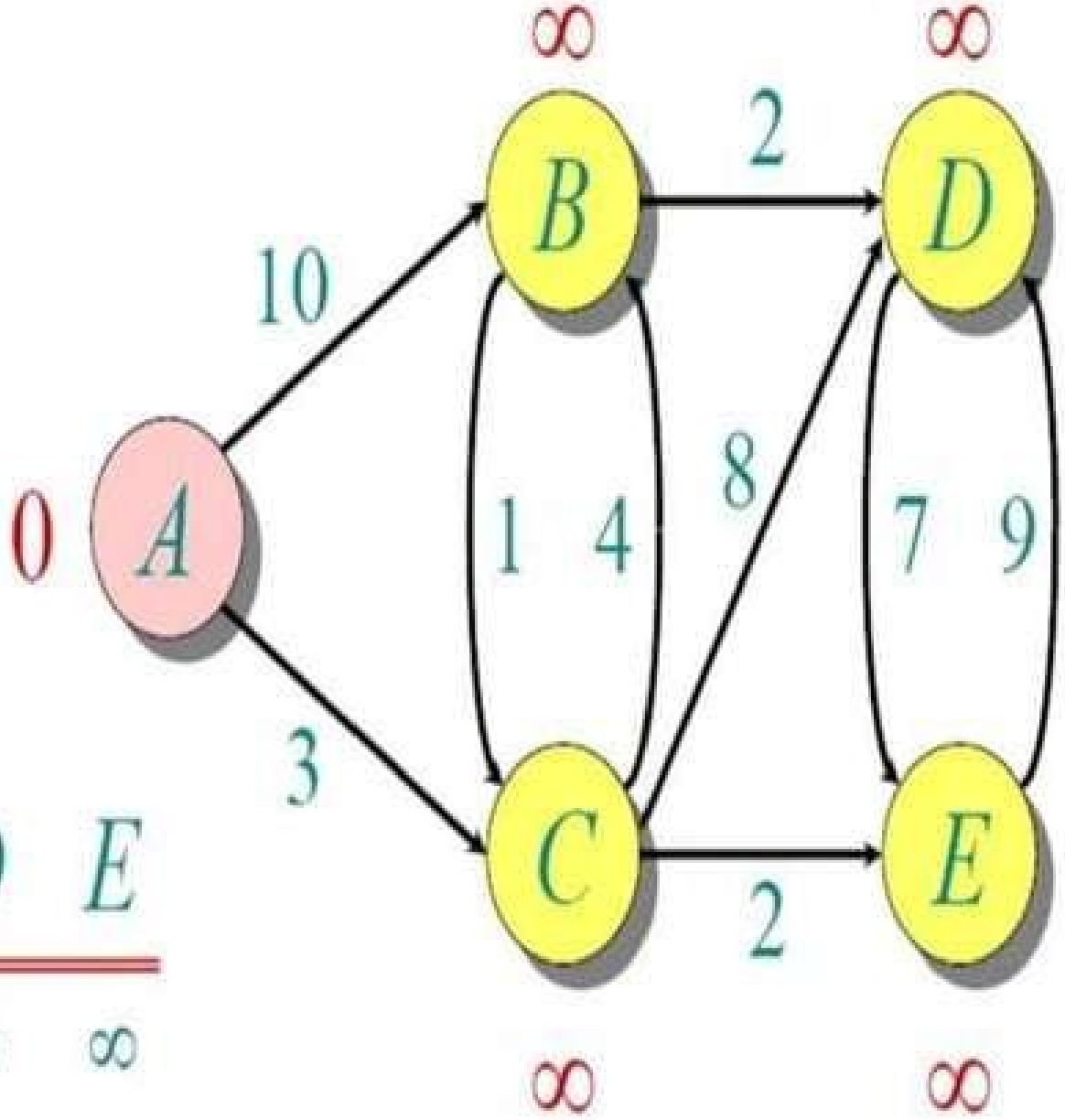
Dijkstra Animated Example

Initialize:

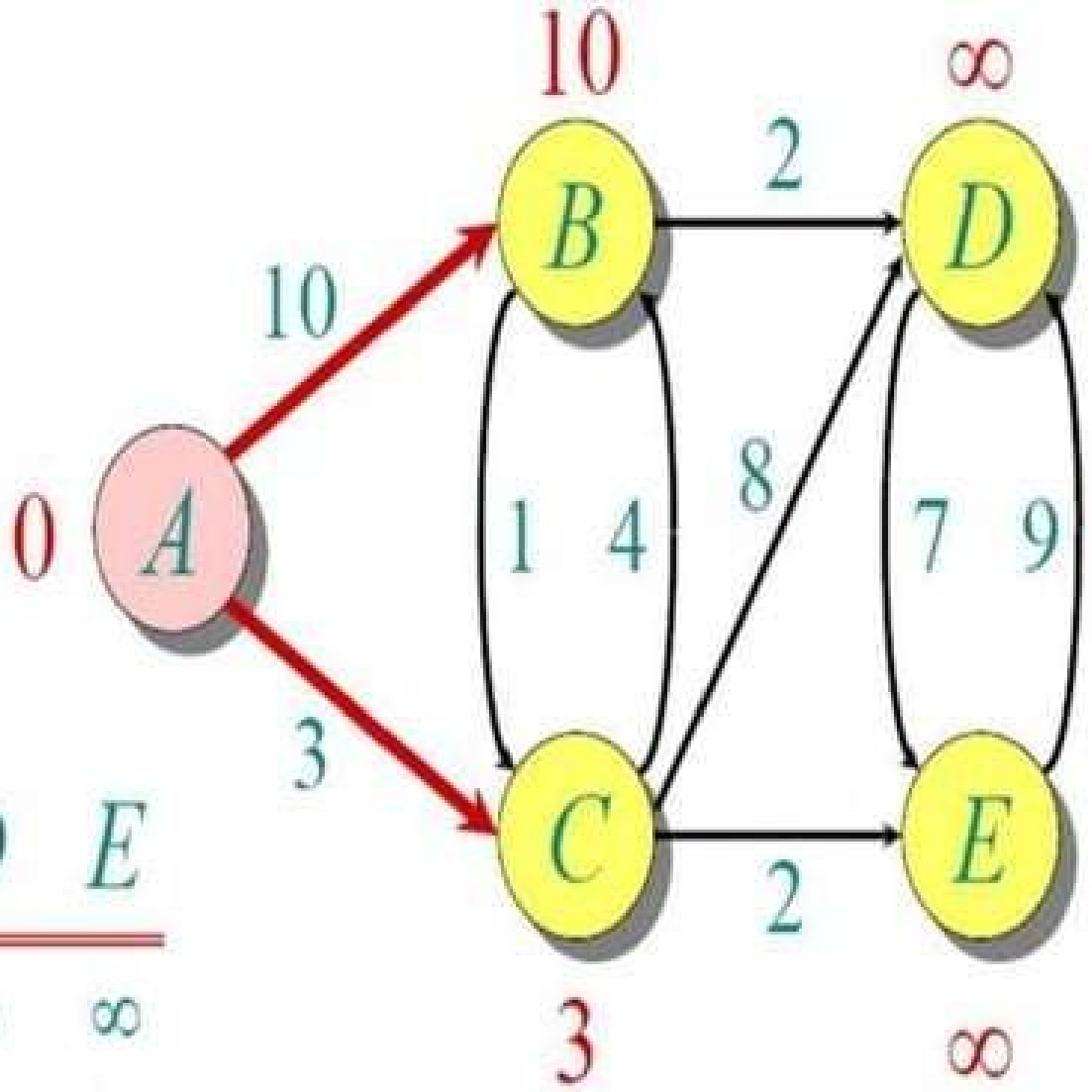


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

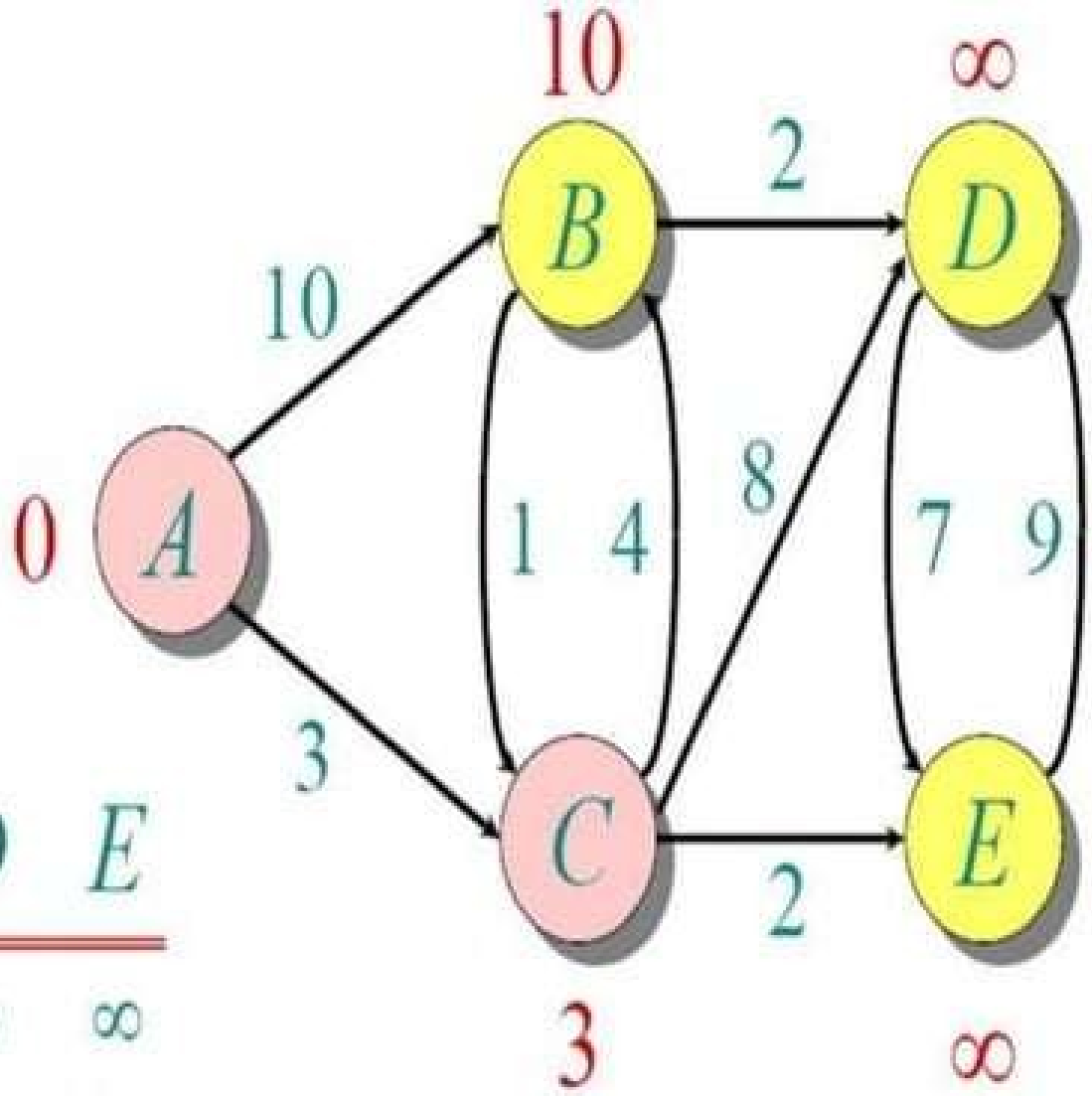


Q:	A	B	C	D	E
	0	∞	∞	∞	∞



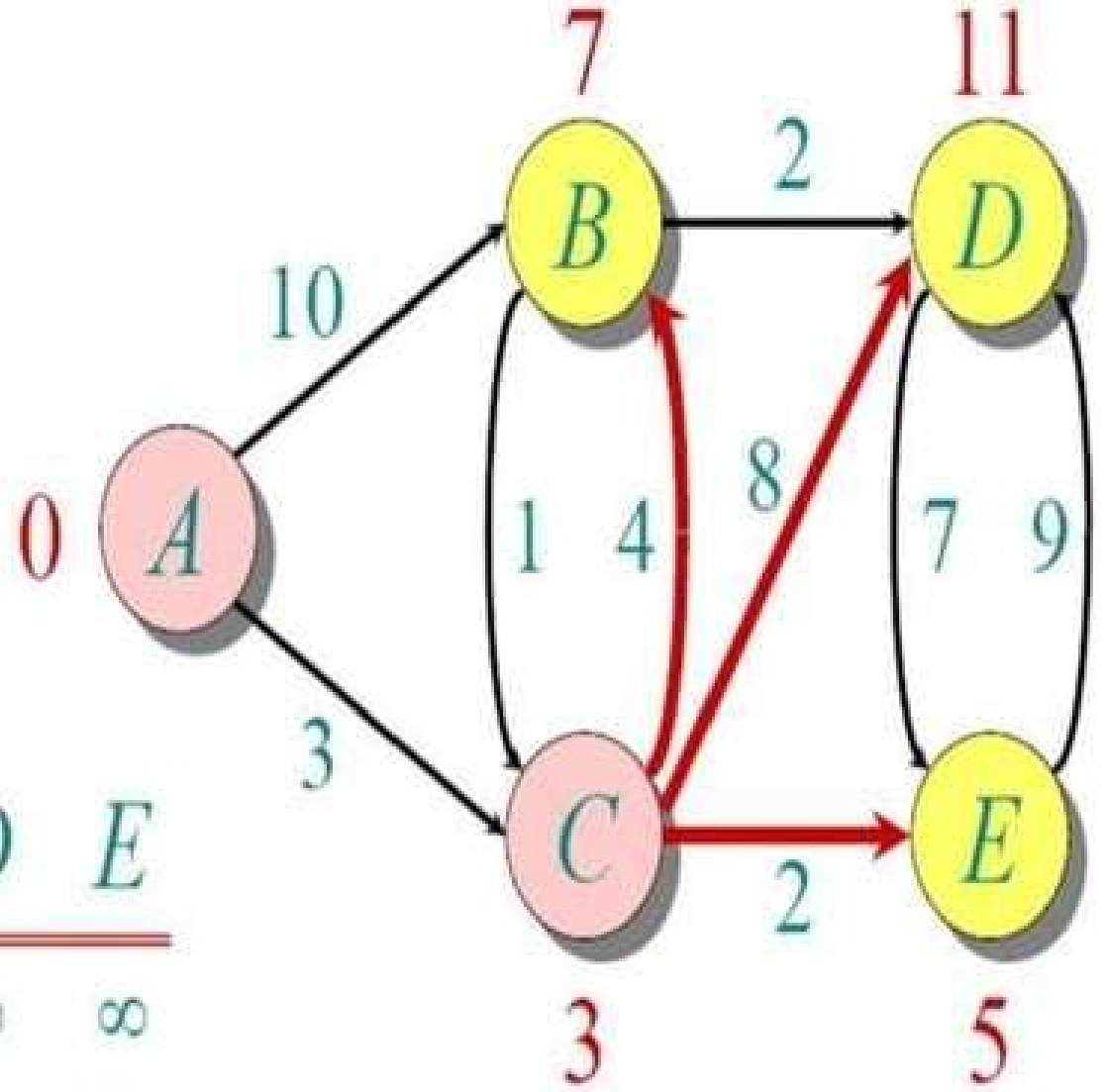
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }



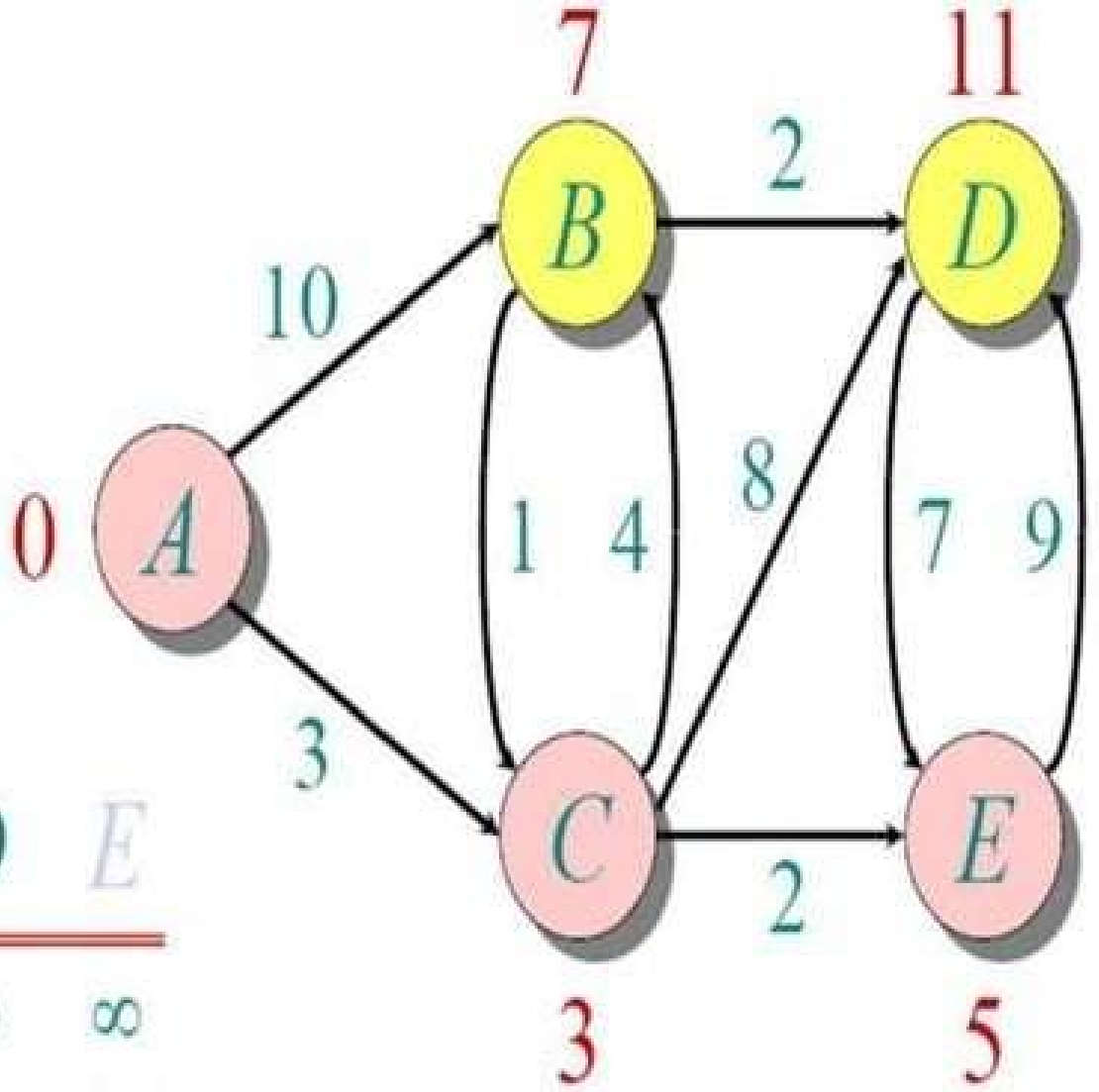
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S: \{A, C\}$



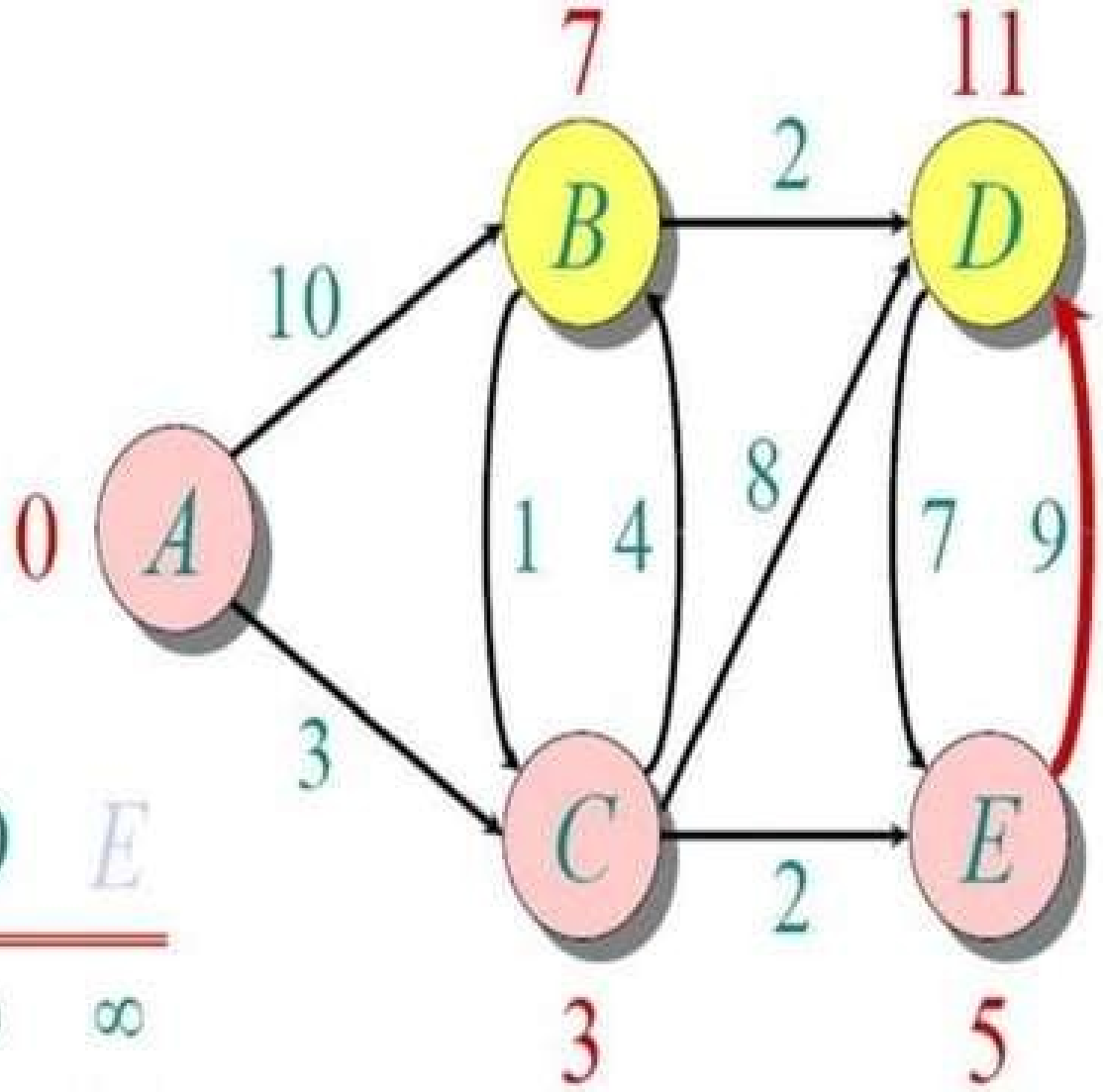
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{A, C\}$



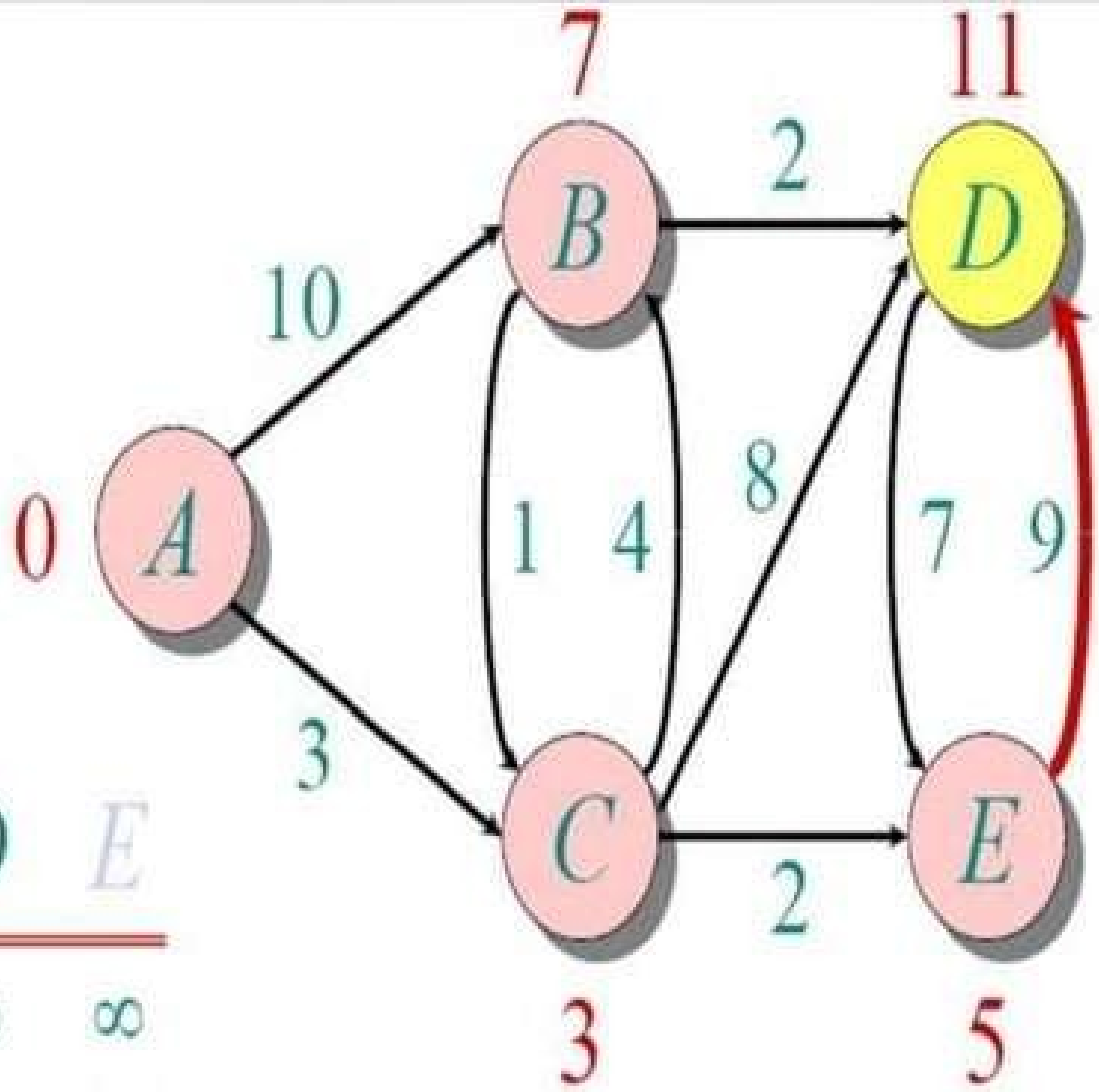
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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



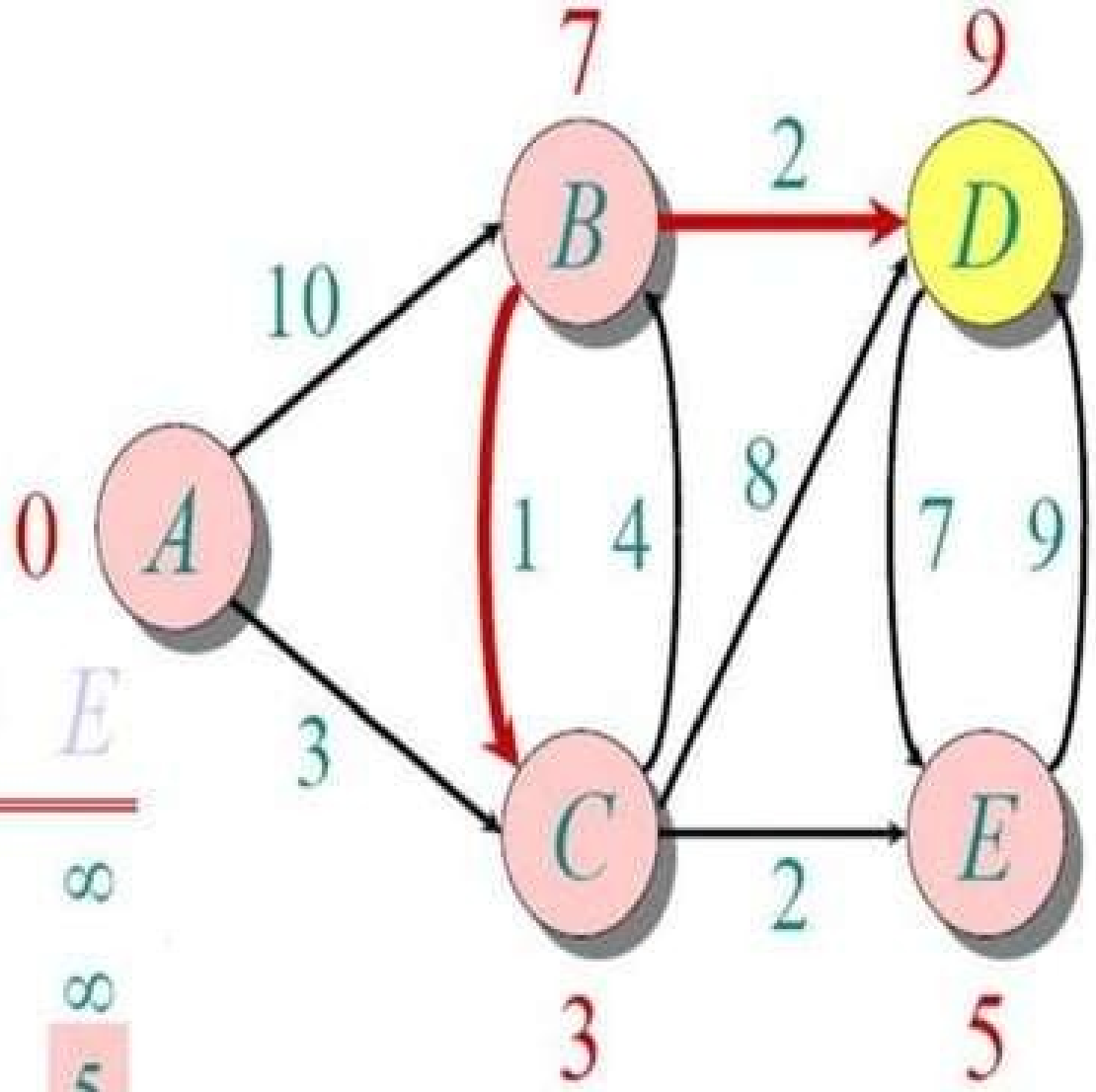
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }



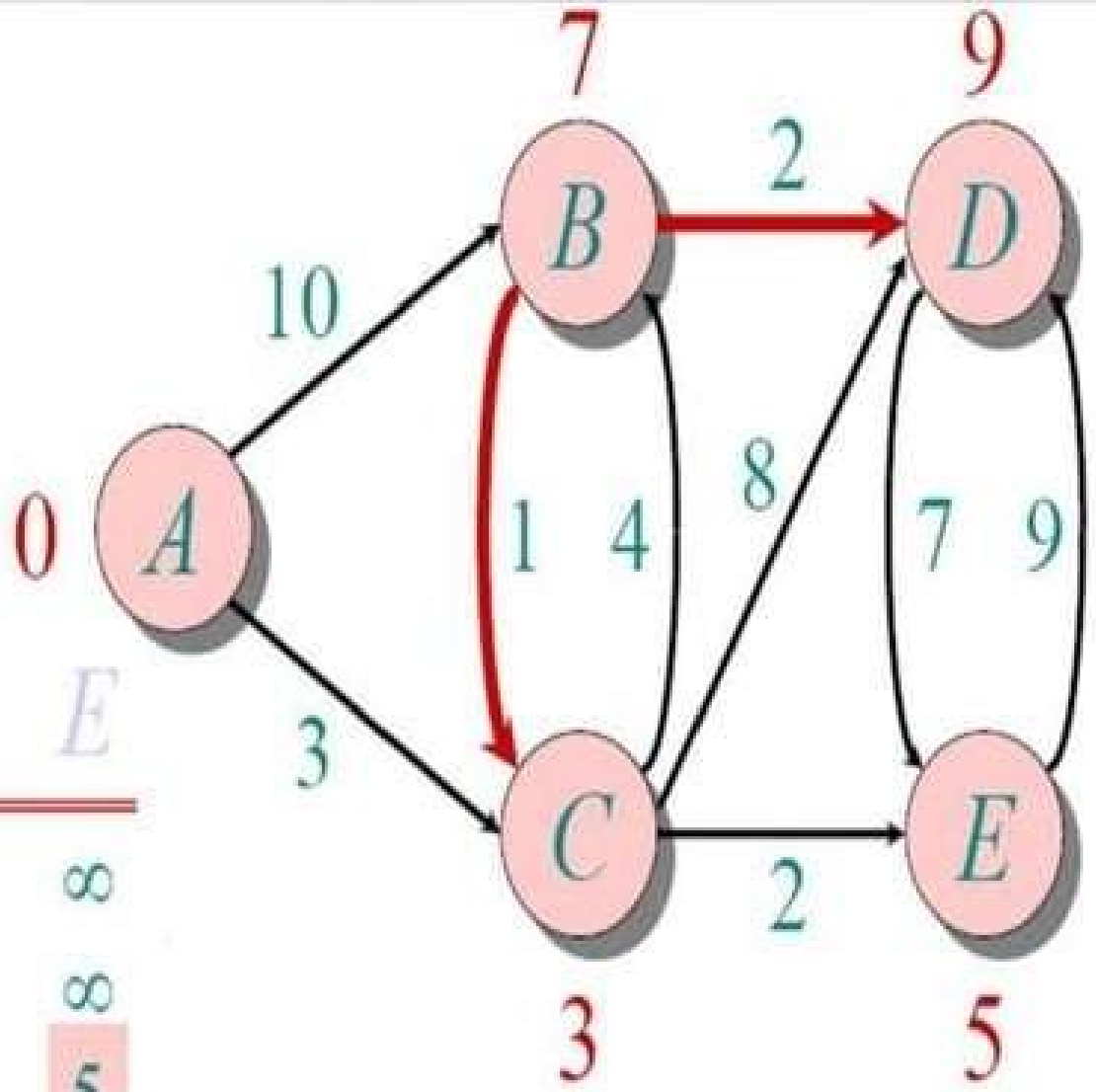
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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



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

S: { A, C, E, B }



Q:

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

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

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}                                   .
    for all v ∈ neighbors[u]              (add u to list of visited vertices)
        do if dist[v] > dist[u] + w(u, v) then d[v] ←d[u] + w(u, v)   (if new shortest path found) (set new value of shortest path)
            (if desired, add traceback code)
return dist
```

https://www.youtube.com/watch?v=EFg3u_E6eHU

Dijkstra (*G*,*s*)

// *G* = (*V*,*E*) is a weighted directed graph with no negative edge weights

1. Initialize-Single-Source (*G*,*s*)
2. *S* ← empty set // S = set of vertices that have been handled
3. Put all vertices *v* in *V* in min-priority queue *Q*, with *d(v)* as key
4. **while** *Q* not empty
5. **do** *u* ← Extract-Min(*Q*)
6. *S* ← *S* U { *u* }
7. **for** each outgoing edge (*u*,*v*)
8. **do** Relax(*u*,*v*)

Running time:

- number of Extract-Min operations = |V|
- number of Relax-operations = |*E*|

Relax involves Decrease-Key-operation

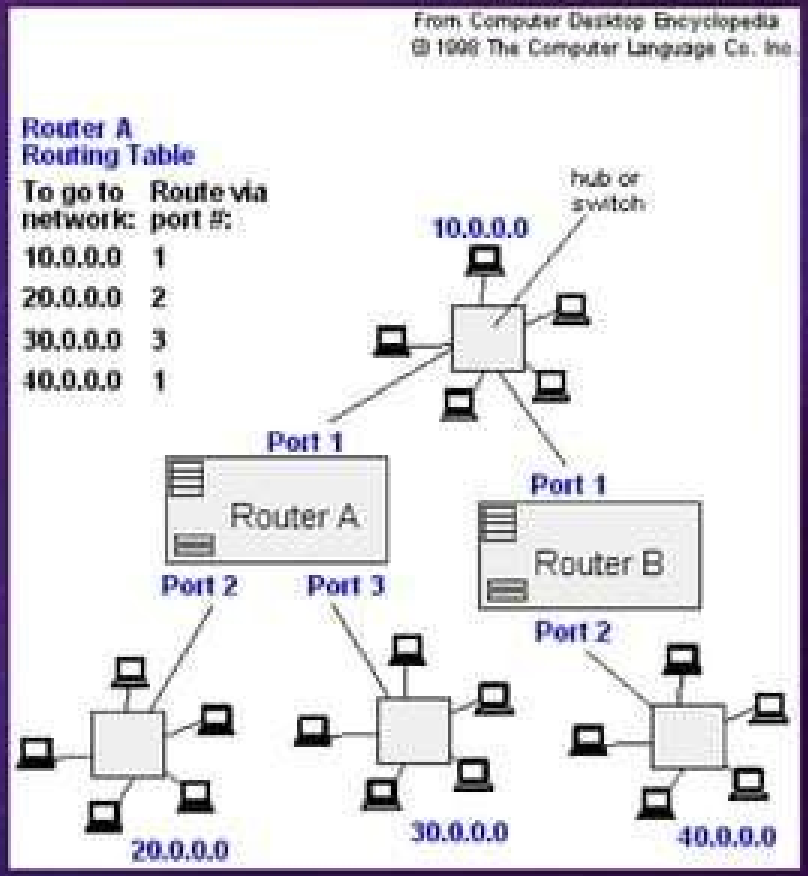
normal heap: Extract-Min and Decrease-Key both O(log *n*) O((*V*+*E*) log *V*

Fibonacci heap: Extract-Min O(log *n*), Decrease-Key O(1) on average

O(*V* log *V* + *E*

Application :

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



References

- Dijkstra's original paper:
  E. W. Dijkstra. (1959) *A Note on Two Problems in Connection with Graphs.* Numerische Mathematik, 1. 269-271.
- MIT OpenCourseware, 6.046J Introduction to Algorithms.
- wikipedia.org


THANK
YOU!