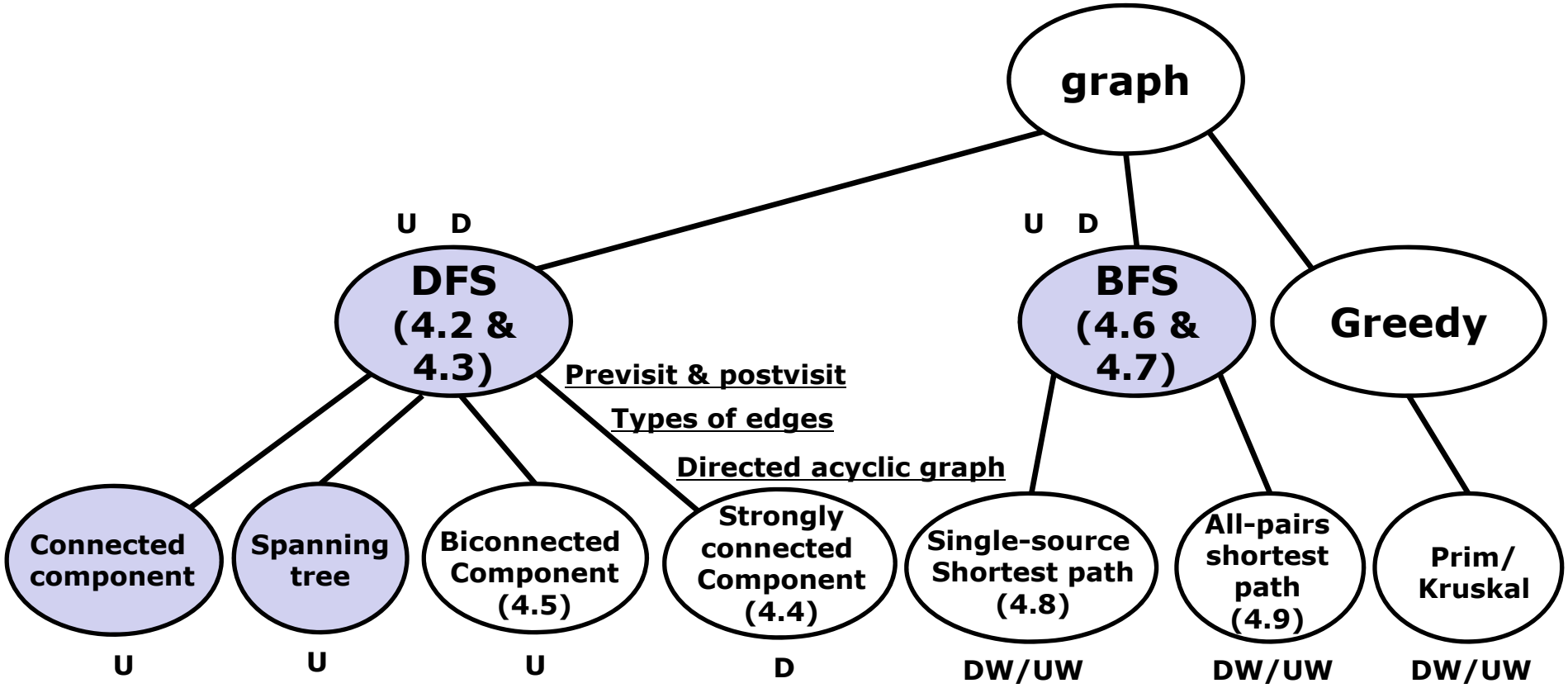
“본 강의 동영상 및 자료는 대한민국 저작권법을 준수합니다. 본 강의 동영상 및 자료는 상명대학교 재학생들의 수업목적으로 제작•배포되는 것이므로, 수업목적으로 내려받은 강의 동영상 및 자료는 수업목적 이외에 다른 용도로 사용할 수 없으며, 다른 장소 및 타인에게 복제, 전송하여 공유할 수 없습니다. 이를 위반해서 발생하는 모든 법적 책임은 행위 주체인 본인에게 있습니다.”

4.8 Single source shortest path

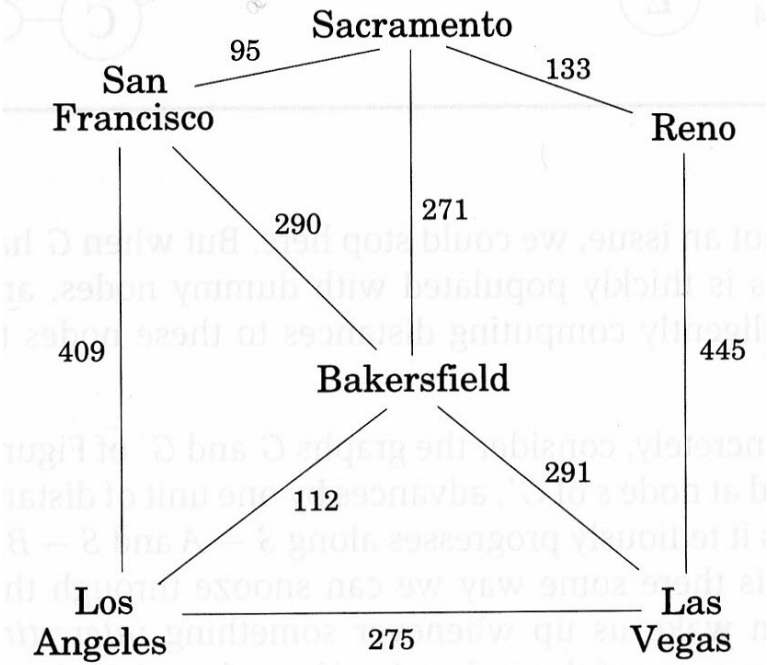
Classification of graph algorithms



4.8 Single source shortest path

(1) Basic concept (1)

- Find the shortest path from a node to all other nodes in a graph



4.8 Single source shortest path

(1) Basic concept (2)

- Single-source shortest path
  - A path from v_0 to u: v_0, v_1, …, v_k, u
  - Path
    - a set of edges composed of

      $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, u)$

  - Cost of path
    - sum of weights of the edges on the path

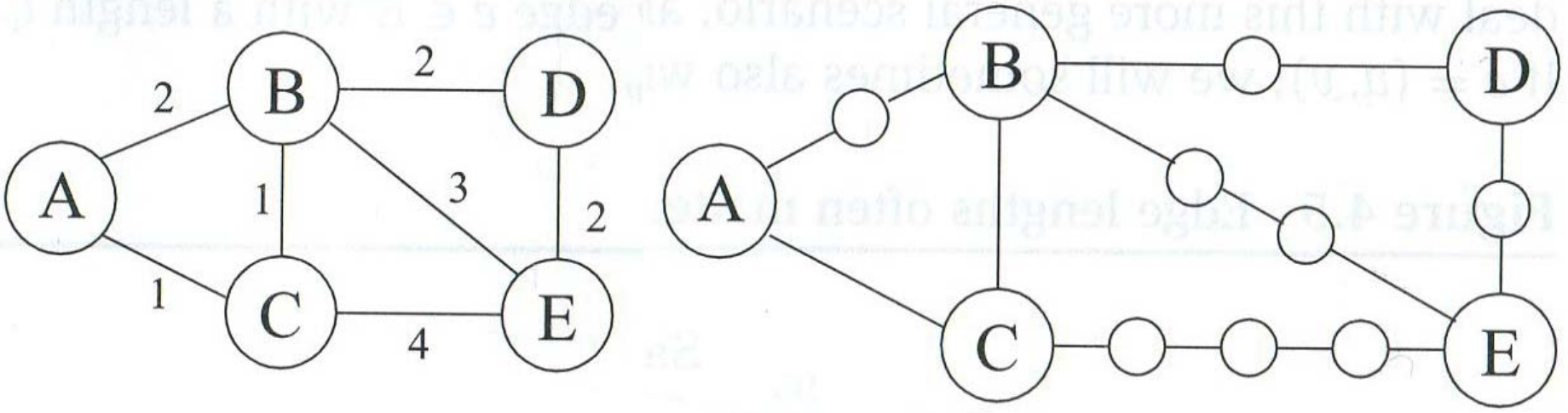
    $w(v_0, v_1)+w(v_1, v_2)+\ldots+w(v_{k-1}, v_k)+w(v_k, u)$

4.8 Single source shortest path

(2) Simple approach (1)

- Adaptation of breadth-first search
  - Modifying weighted graph to non-weighted graph

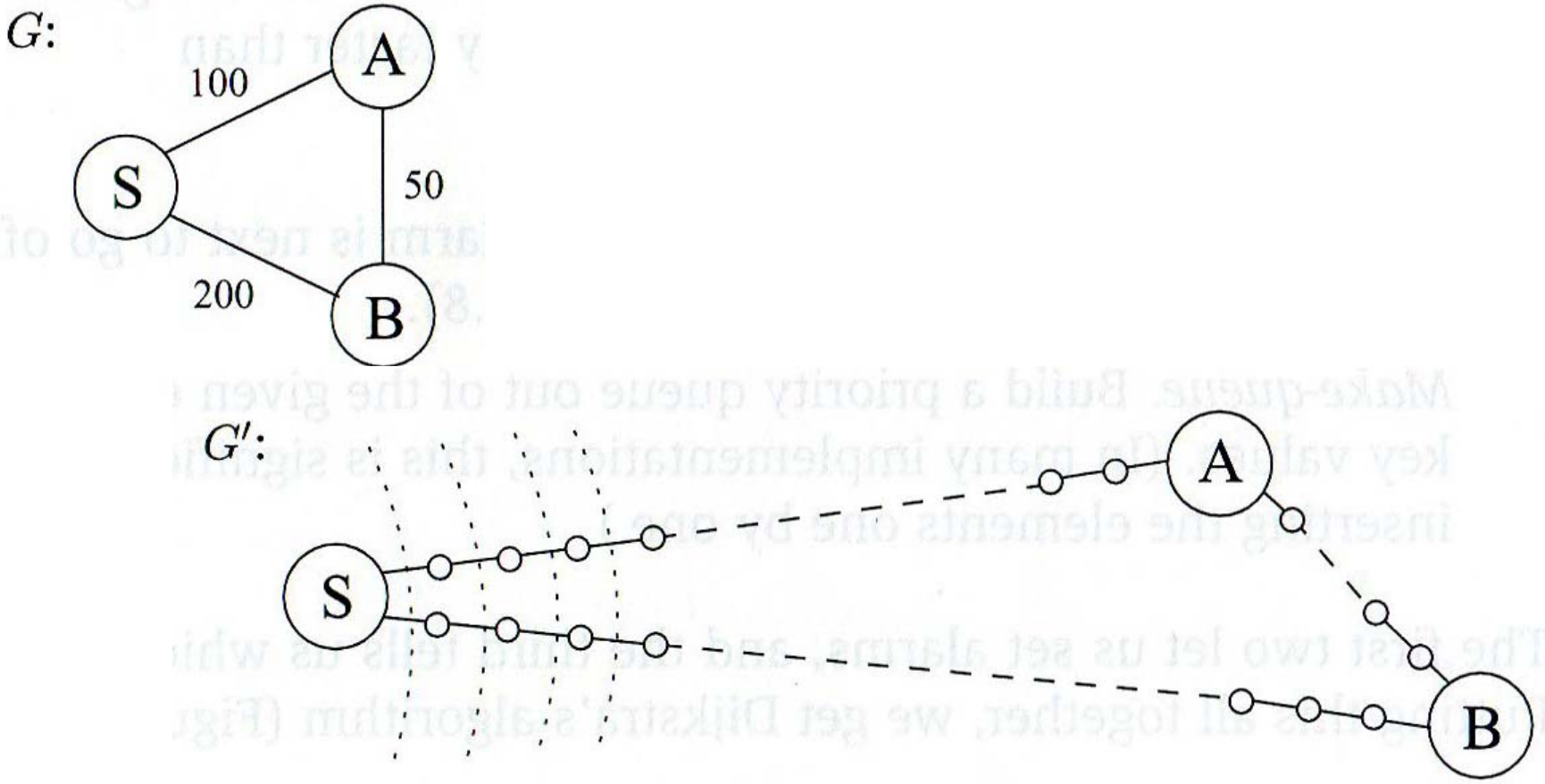
For any edge e = (u, v) of weight l_e, replace e with l_e edges of weight 1_e by adding l – 1 vertices between u & v



4.8 Single source shortest path

(2) Simple approach (2)

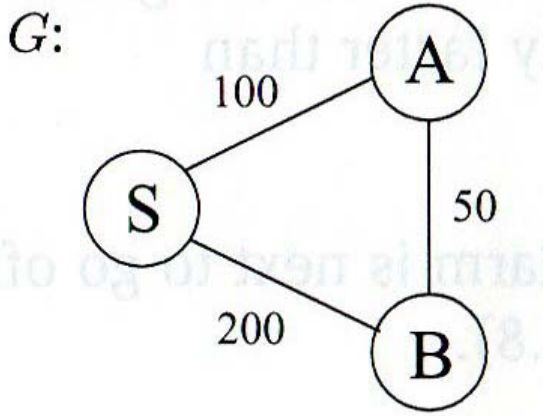
- Problem: Inefficiency



4.8 Single source shortest path

(3) Alarm clocks (1)

- A motivation to Dijkstra's algorithm
- Strategy
  - On leaving S (t = 0), we set two alarms for A (t = 100) and B (t = 200).
  - At arriving A at t = 100, we estimate the distance from A to B and reset the alarm for B (t = 150).



4.8 Single source shortest path

(3) Alarm clocks (2)

- General strategy

```
1. Set an alarm clock for node s at time 0.
2. Repeat until there are no more alarms.
3. If the next alarm goes off at time T for node u,
    3.1 The distance from s to u is T.
    3.2 For each neighbor v of u
        3.2.1 If no alarm set to v, then set as T + l(u, v).
        3.2.2 If v's alarm > T + l(u, v), then
              set v's alarm as T + l(u, v).
```

4.8 Single source shortest path

(3) Alarm clocks (2)

- General strategy

```
1. Set an alarm clock for node s at time 0.
2. Repeat until there are no more alarms.
3. If the next alarm goes off at time T for node u,
    3.1 The distance from s to u is T.
    3.2 For each neighbor v of u
        3.2.1 If no alarm set to v, then set as T + l(u, v).
        3.2.2 If v's alarm > T + l(u, v), then
              set v's alarm as T + l(u, v).
```

Edge relaxation
(principle of relaxation)

An approximation to the correct distance is gradually replaced
by more accurate values until eventually reaching the optimum solution

4.8 Single source shortest path

(4) Dijkstra's algorithm (1)

- Single source all destination shortest path
- Directed graph
- Non-negative edge

4.8 Single source shortest path

(4) Dijkstra's algorithm (2)

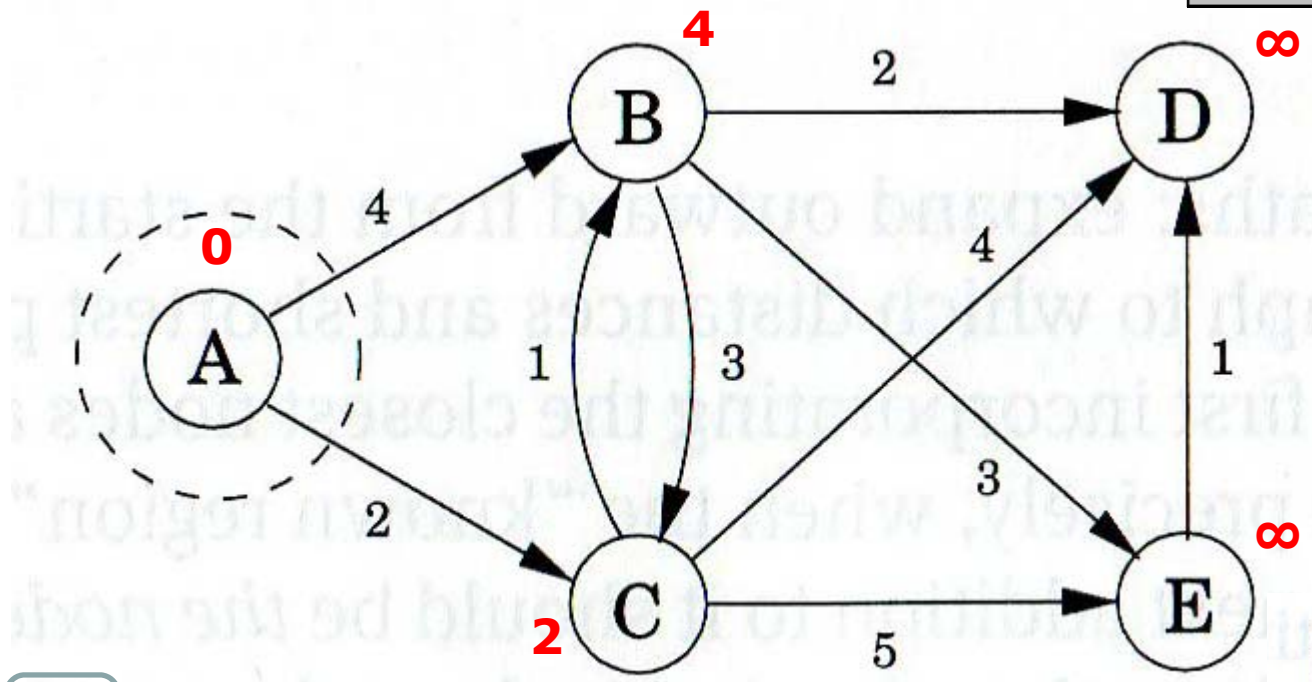
```
procedure Dijkstra ( G, s )

for all u ∈ V
    dist[u] = ∞;
    prev[u] = NULL;

dist[s] = 0;

H = makequeue ( V );
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

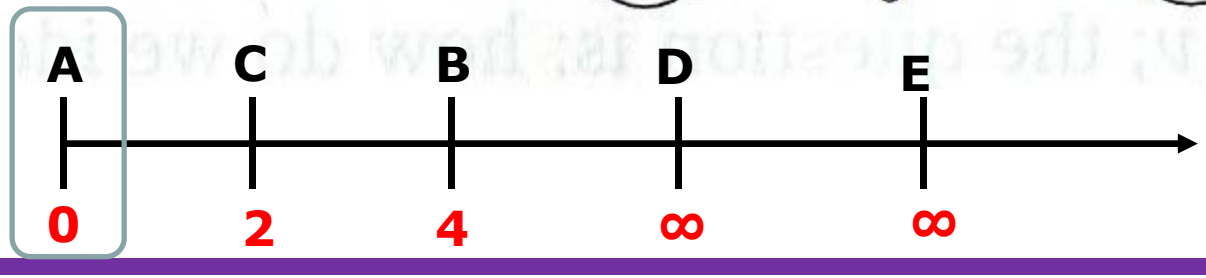
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Initial step





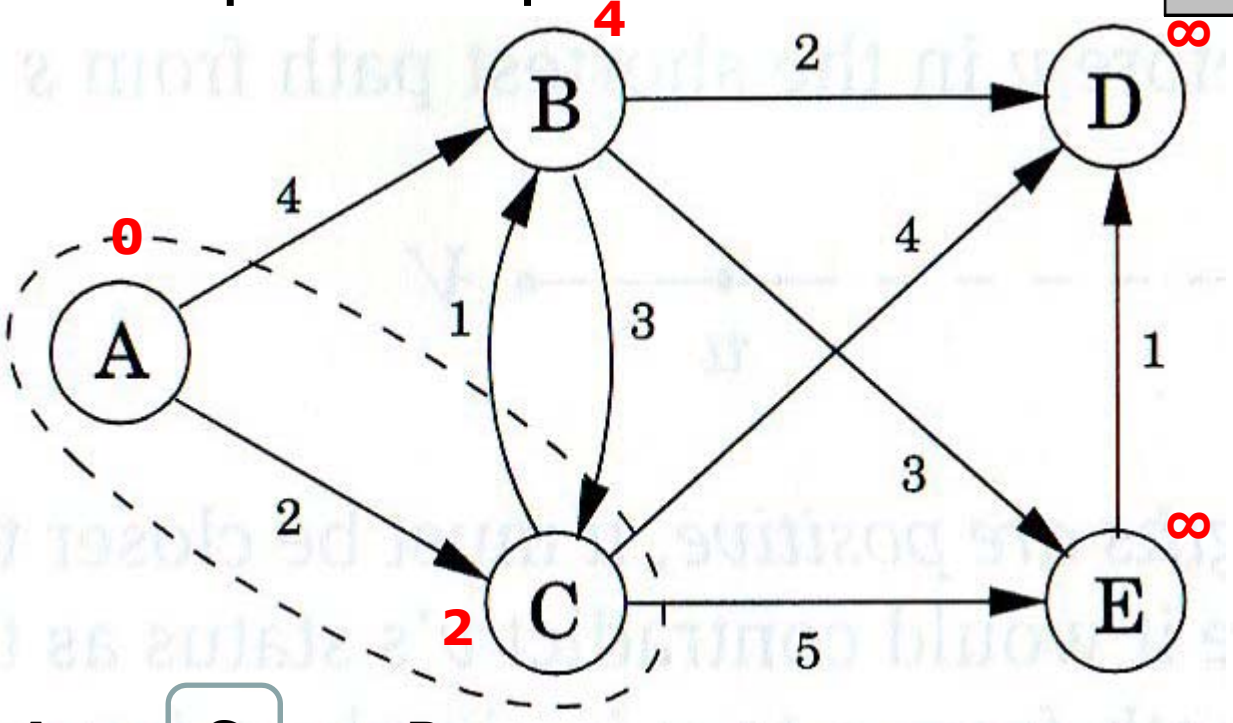
A: 0	D: ∞
B: 4	E: ∞
C: 2	

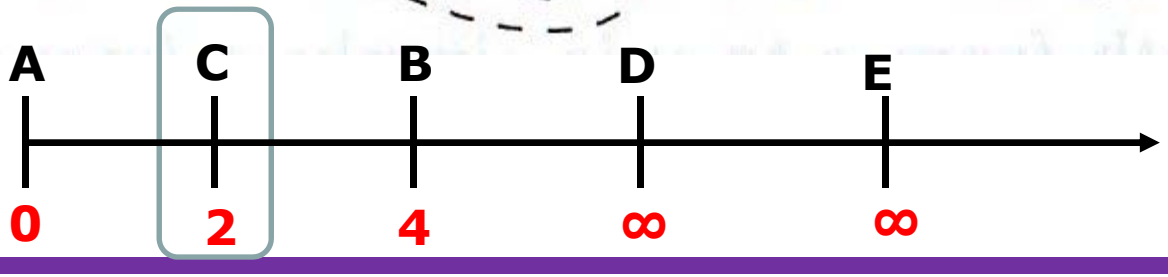
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 1





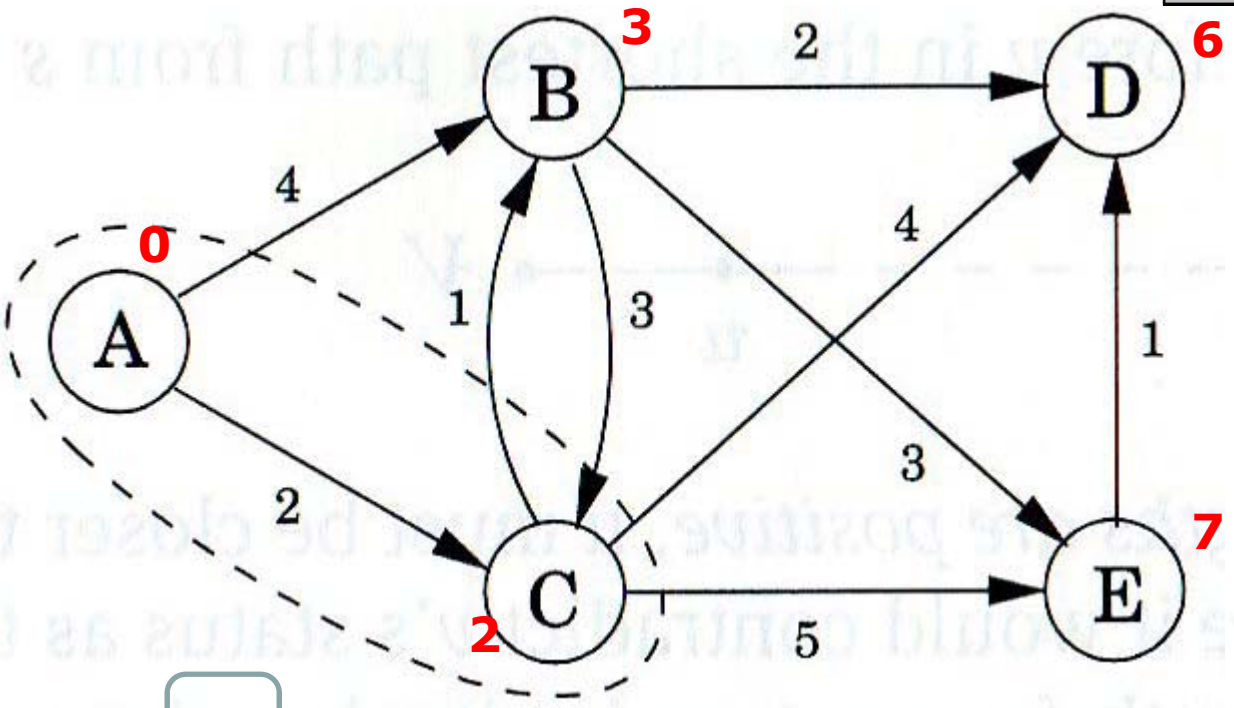
A: 0	D: 6
B: 3	E: 7
C: 2	

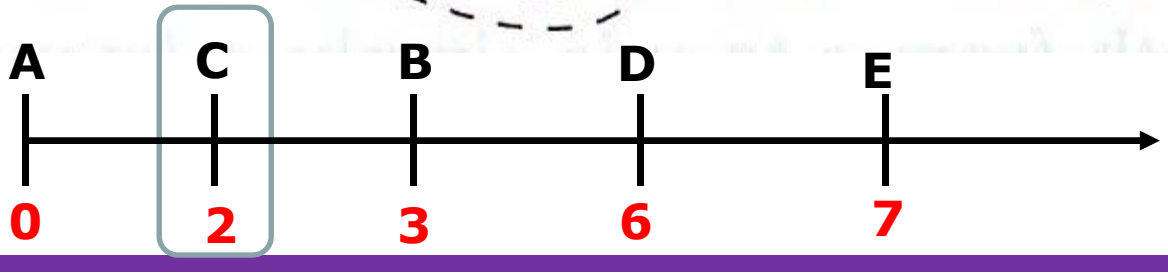
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 1





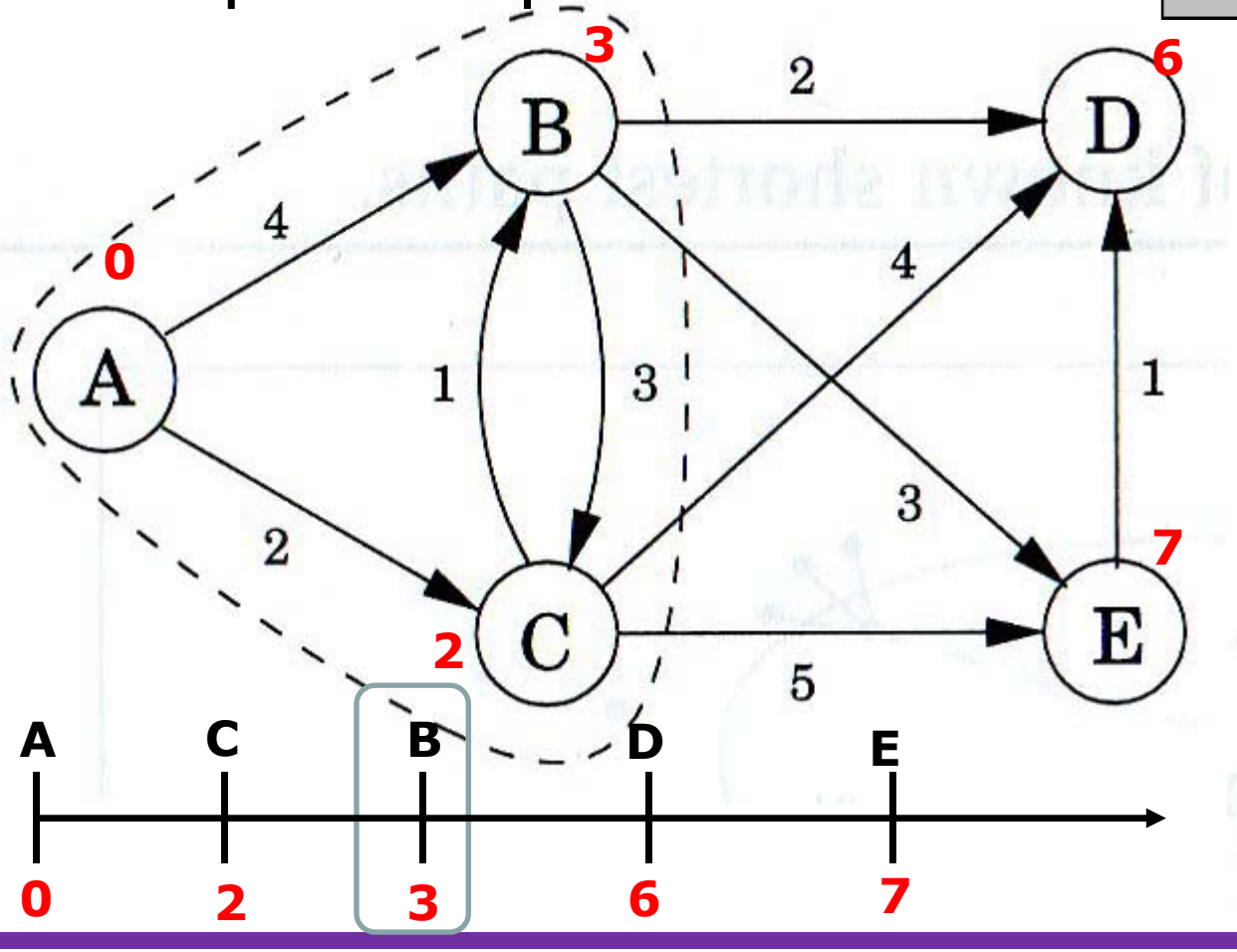
A: 0	D: 6
B: 3	E: 7
C: 2	

4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 2



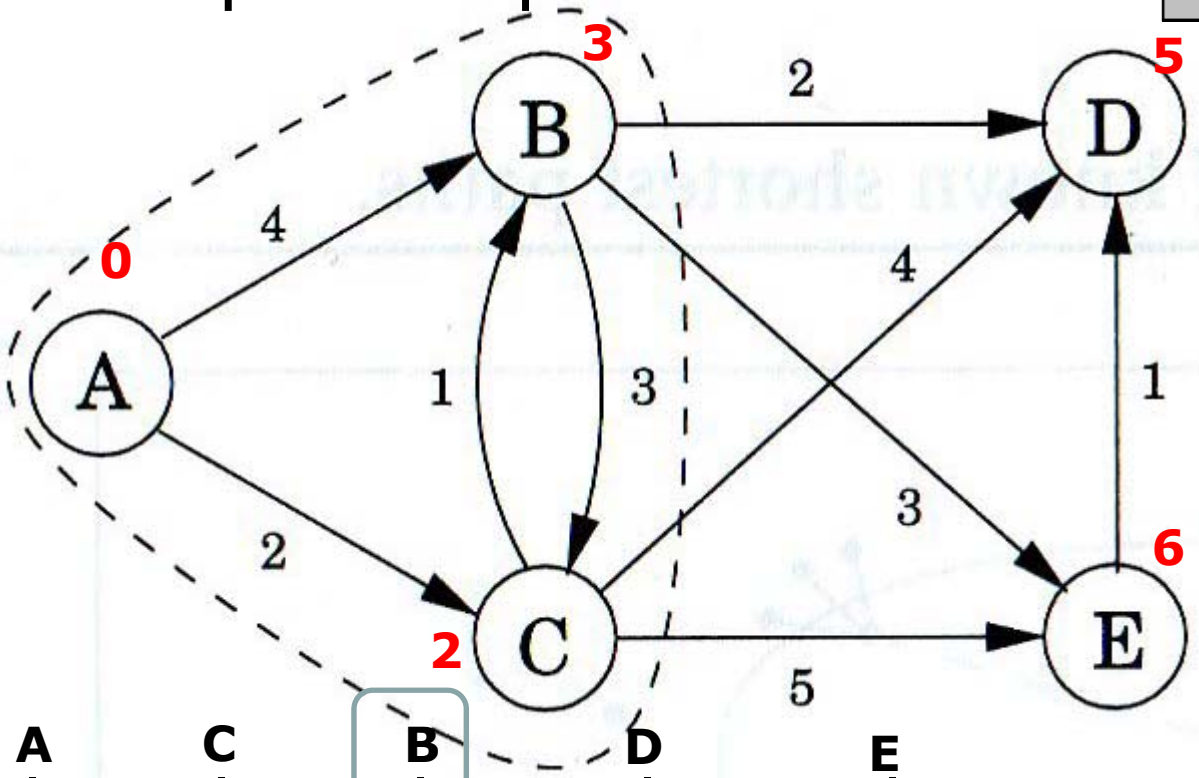
A: 0	D: 5
B: 3	E: 6
C: 2	

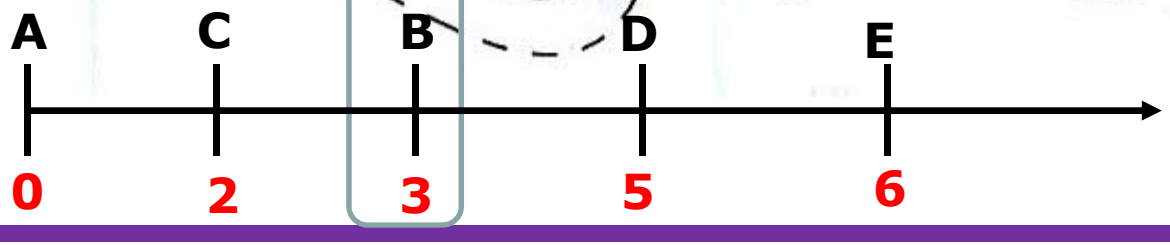
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 2





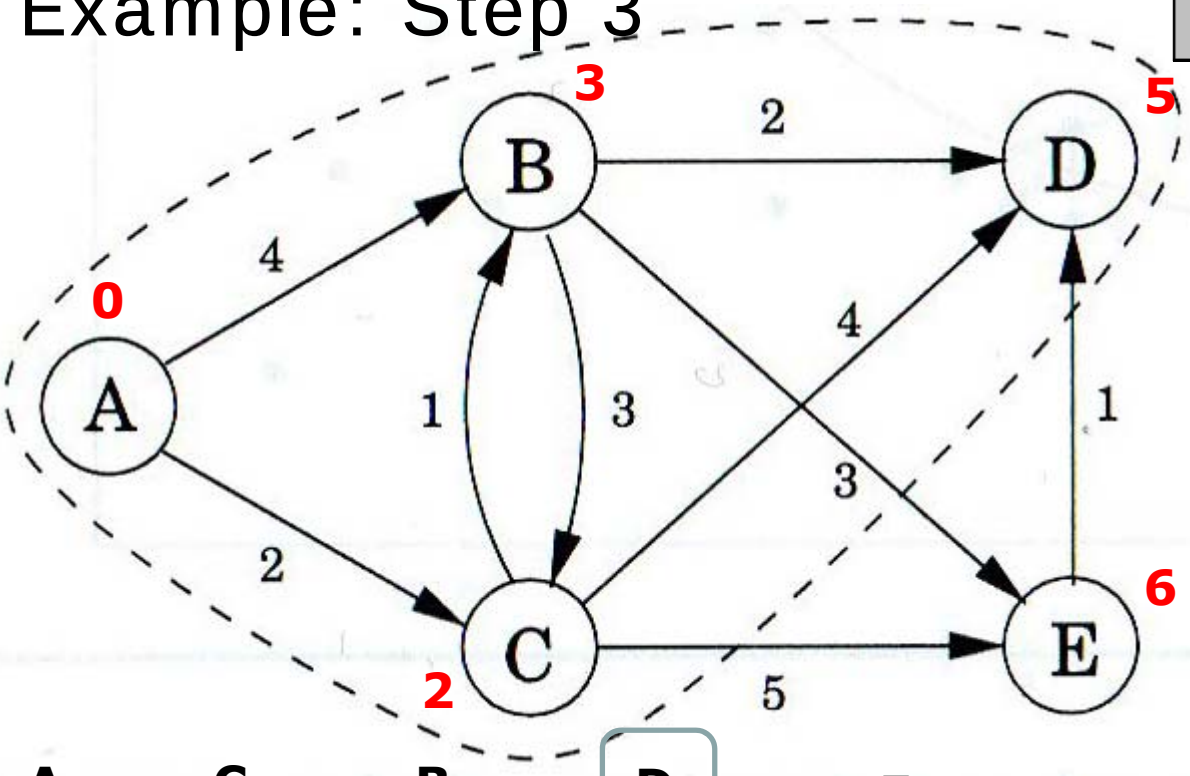
A: 0	D: 5
B: 3	E: 6
C: 2	

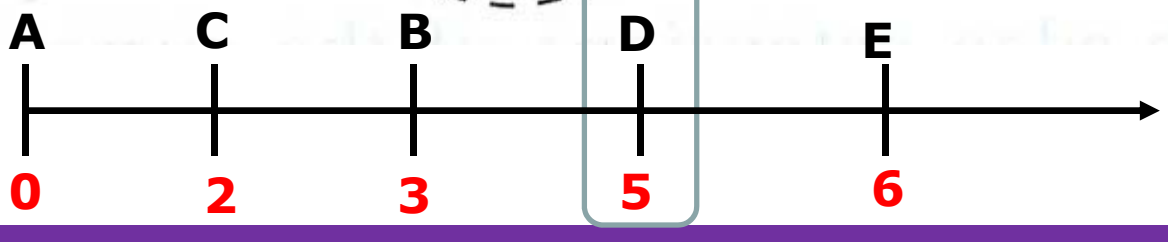
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 3





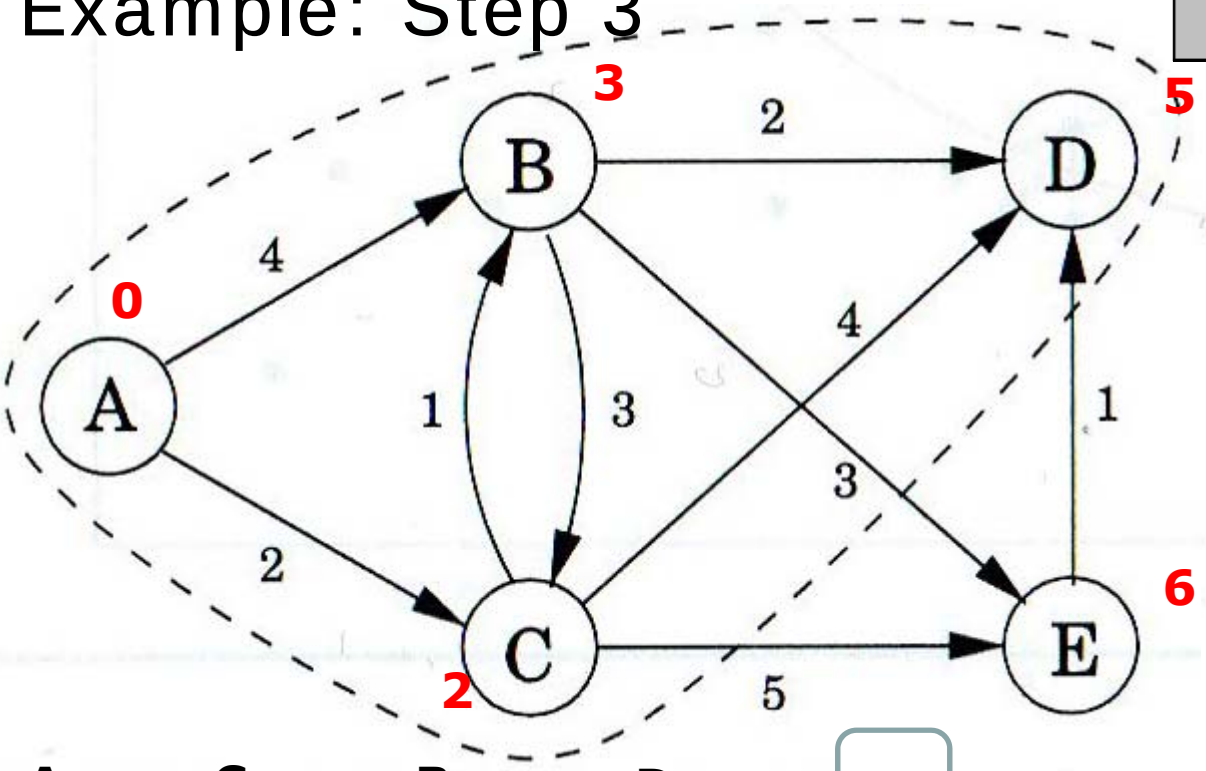
A: 0	D: 5
B: 3	E: 6
C: 2	

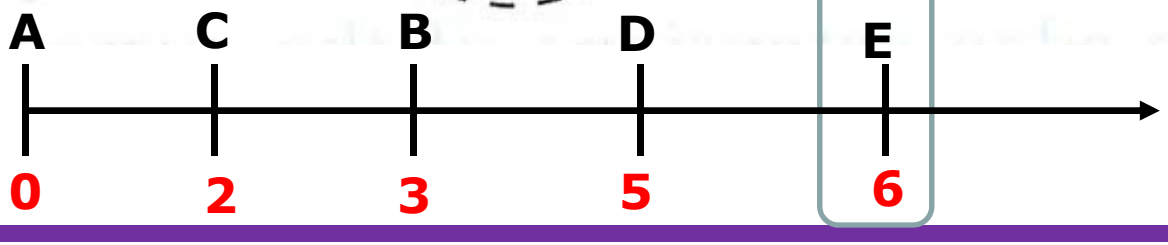
4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example: Step 3



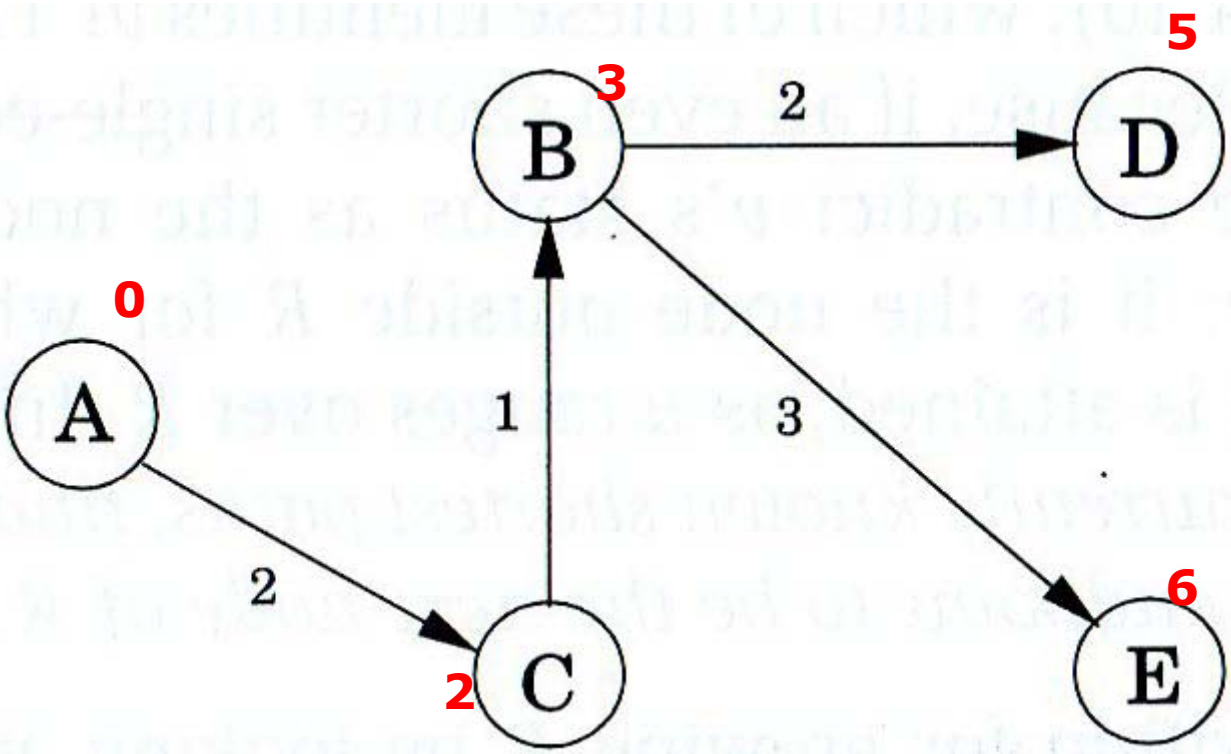


A: 0	D: 5
B: 3	E: 6
C: 2	

4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

- Example: Result

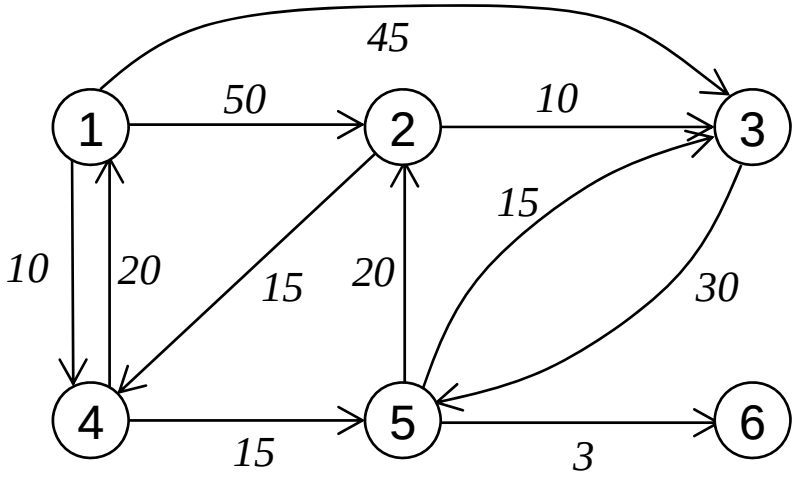


4.8 Single source shortest path

(4) Dijkstra's algorithm (3)

```
while ( H is not empty )
    u = get_min ( H );
    for all edges (u, v) ∈ E and v ∈ H
        if ( dist[v] > dist[u] + l(u, v) )
            dist[v] = dist[u] + l(u, v);
            prev[v] = u;
            modify_H ( H, v );
```

- Example:



4.8 Single source shortest path

(4) Dijkstra's algorithm (4)

- Performance analysis (|V| = n, |E| = m)

```
procedure Dijkstra ( G, s )
for all u ∈ V                                   } O(n)
    dist[u] = ∞;
    prev[u] = NULL;

dist[s] = 0;

H = makequeue ( V );                            ——— O(?) …… A
while ( H is not empty )
    u = get_min ( H );                          ——— n x O(?) … B
    for all edges (u, v) ∈ E and v ∈ H           }
        if ( dist[v] > dist[u] + l(u, v) )       } n x O(?) →
            dist[v] = dist[u] + l(u, v);         } m x O(?) … C
            prev[v] = u;                         }
            modify_H ( H, v );                   }
```

4.8 Single source shortest path

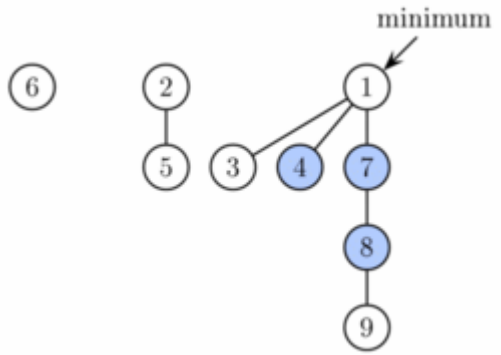
(4) Dijkstra's algorithm (4)

- Performance analysis ($|V| = n$, $|E| = m$)
  - Performance = A + B + C
- Original Dijkstra's algorithm (1956) doesn't use a priority queue
  - A: inserting n elements to a queue
    - Array-based queue: O(1) → O(n)
  - B: finding/deleting minimum from a queue
    - Array-based queue: O(n) → $O(n^2)$
  - C: modifying the queue
    - Array-based queue: O(1) → O(m)
- Time complexity: $O(n^2)$

4.8 Single source shortest path

(4) Dijkstra's algorithm (4)

- Fredman and Tarjan proposed a Fibonacci heap to improve Dijkstra's algorithm (1984)
  - Fibonacci heap: a kind of min binary heap



	Binary heap	Fibonacci heap
find-min	O(1)	O(1)
delete-min	O(log n)	O(log n)
insert	O(log n)	**O(1)**
heapify	O(log n)	**O(1)**
merge	O(m log n)	**O(1)**

4.8 Single source shortest path

(4) Dijkstra's algorithm (4)

- Performance analysis (|V| = n, |E| = m)
  - Performance = A + B + C
- Fredman and Tarjan's improvement using Fibonacci heap
  - A: inserting n elements to a queue
    - Fibonacci heap-based queue: O(1) → O(n)
  - B: finding/deleting minimum from a queue
    - Fibonacci heap-based queue : O(**log n**) → O(**n log n**)
  - C: modifying the queue
    - Fibonacci heap-based queue : O(1) → O(m)
- Time complexity: O(m) + O(n log n)

4.8 Single source shortest path

(4) Dijkstra's algorithm (5)

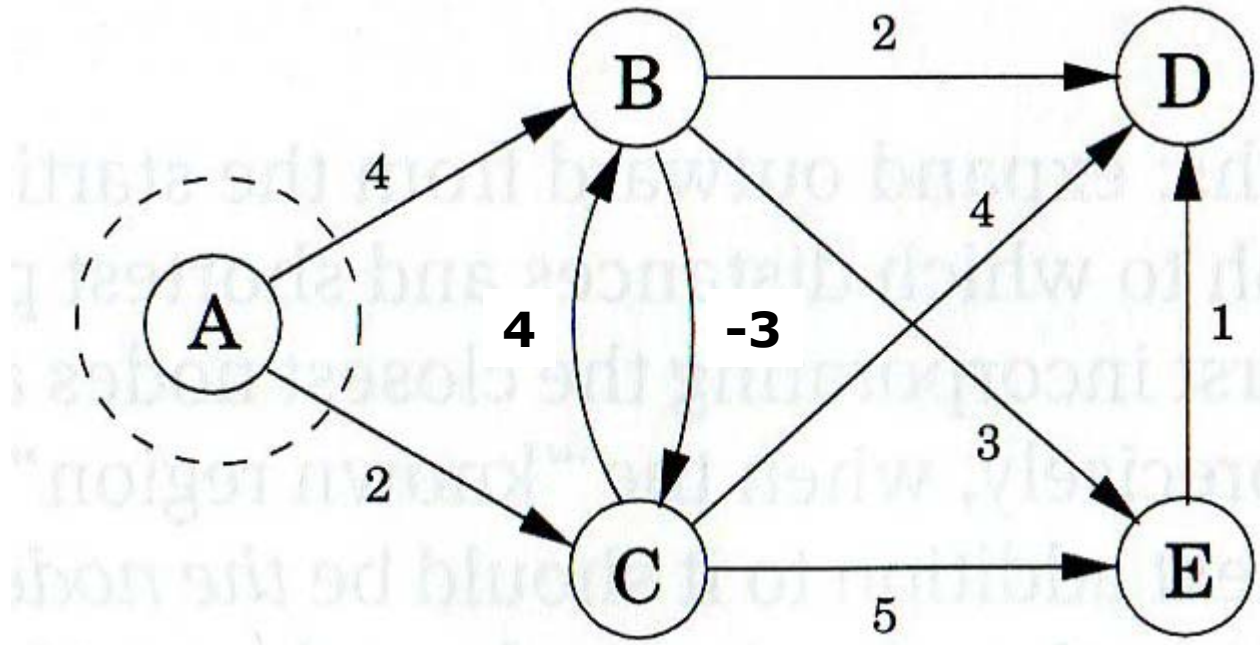
- Performance depends on queue & complexity

complexity \ queue	array	Priority queue
$\|E\| = O(\|V\|^2)$ $m = O(n^2)$	A: O(n)	A: O(n log n)
	B: $O(n^2)$	B: O(n log n)
	C: $O(\|E\|) = O(n^2)$	C: $O(\|E\|) = O(n^2)$
	Total: **$O(n^2)$**	Total: **$O(n^2)$**
$\|E\| = O(\|V\|)$ $m = O(n)$	A: O(n)	A: O(n log n)
	B: $O(n^2)$	B: O(n log n)
	C: $O(\|E\|) = O(n)$	C: $O(\|E\|) = O(n)$
	Total: **$O(n^2)$**	Total: **O(n log n)**

4.8 Single source shortest path

(4) Dijkstra's algorithm (6)

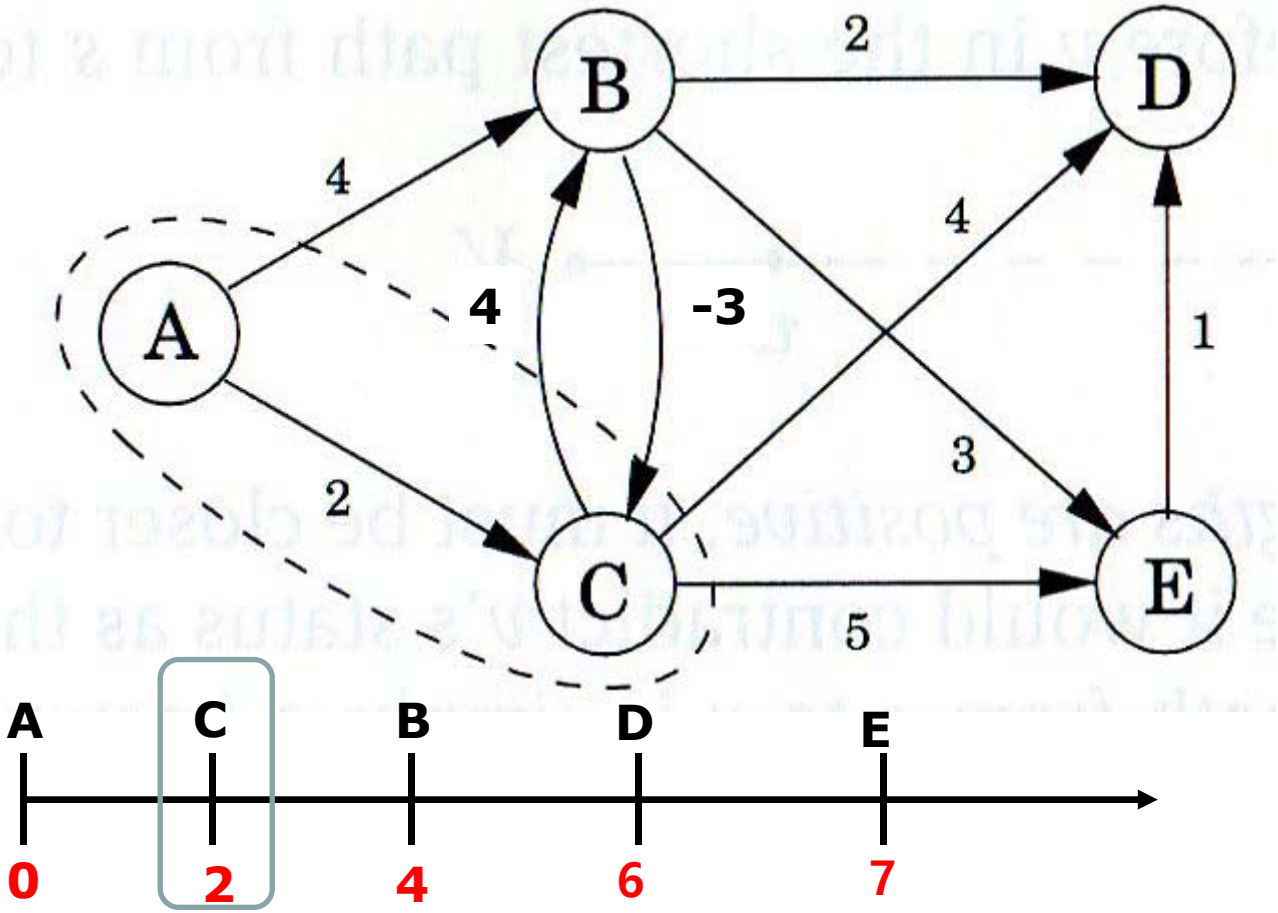
- Dijkstra's algorithm with a negative edge?



4.8 Single source shortest path

(4) Dijkstra's algorithm (6)

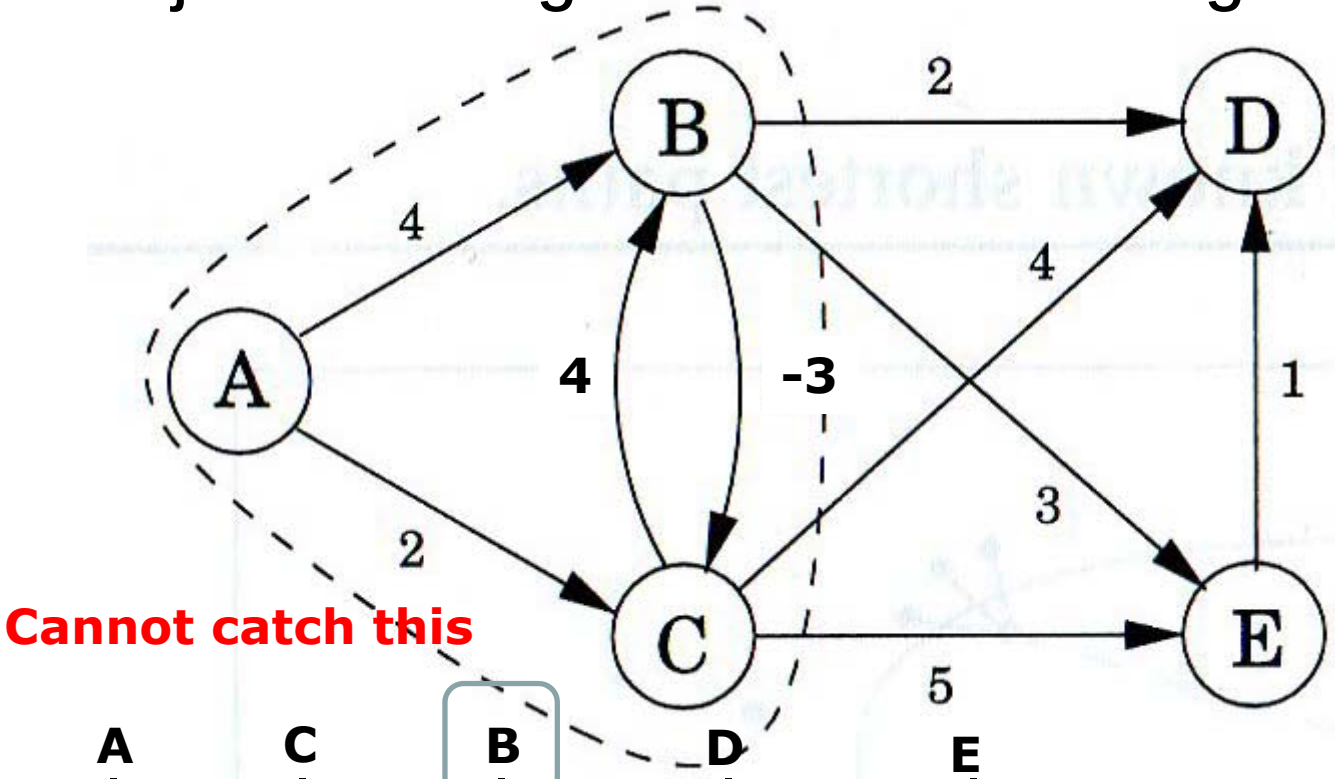
- Dijkstra's algorithm with a negative edge?



4.8 Single source shortest path

(4) Dijkstra's algorithm (6)

- Dijkstra's algorithm with a negative edge?



4.8 Single source shortest path

- Dijkstra's algorithm에 대한 설명이다. 올바른 것을 모두 고르시오.

(a) 하나의 출발 vertex에서 모든 vertex로 가는 최단 거리를 계산하는 알고리즘이다.

(b) 임의의 두 vertex 사이의 최단 거리를 계산하는 알고리즘이다.

(c) negative edge를 가진 graph에서는 정확한 거리를 계산할 수 없다.

(d) Dijkstra algorithm에서 일반적인 queue를 이용할 경우, 연산 시간은 O(n^2)이 요구된다.

(e) Dense graph에 대한 Dijkstra algorithm의 시간 복잡도는 O(n^2)이다.