

F, 10/04/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song
Texas A&M U

Lecture 14

- Dijkstra's algorithm: shortest path (non-negative weight)

Recall: shortest path problem

- **Input.** graph G , node s and t
- **Output.** $\text{dist}(s, t)$
- Every edge has a **length** l_e (can be negative)
- Length of a path $l(P) = \sum_{e \in P} l_e$
- **Distance** $\text{dist}(u, v) = \min_{P: u \rightsquigarrow v} l(P)$
- **Special cases**
 - All edge of equal length: **BFS** $O(m + n)$
 - DAG: **DP** in topological order $O(m + n)$
- Non-negative length: **Dijkstra** $O((m + n)\log n)$
- **General:** Bellman-Ford $O(mn)$

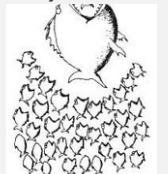
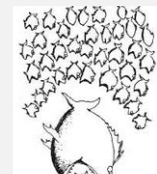
Dynamic Programming ReRecap

1. Formulate the problem recursively

- **Overlapping** subproblems
- May be easy to first compute optimal **value** & then construct an optimal solution

2. Build solutions to your recurrence

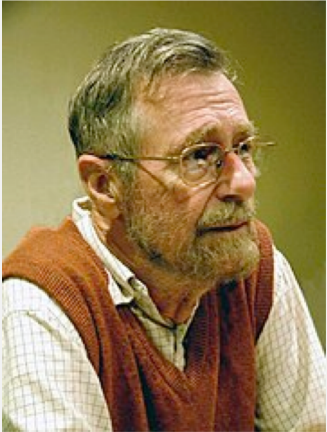
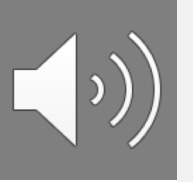
- Bottom-up: determine dependencies & find a right order (topo. order in DAG)
- Top-down: **smart recursion** (i.e. without repetition) by **memoization**



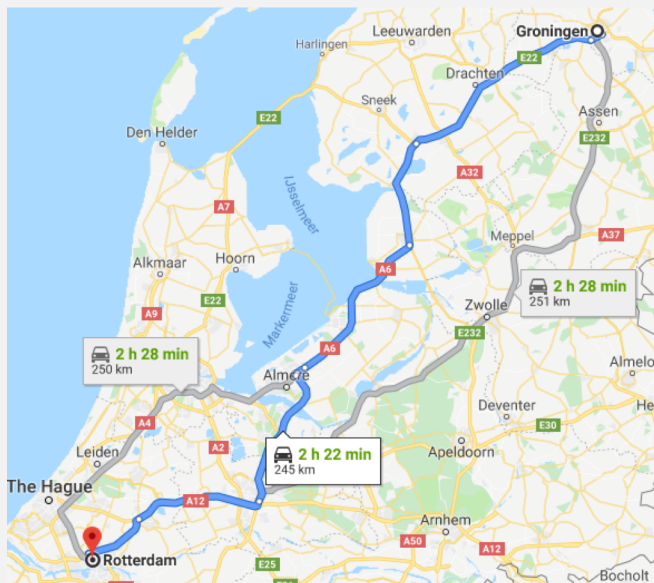
Examples

- **Explicit DAG**: shortest path in DAG; longest increasing subsequences (a.k.a. longest path in DAG)
- **Binary choice**: weighted interval scheduling
- **Multi-way choice**: matrix-chain mult.; longest common subsequence;
- **Adding a variable**: shortest path with negative length (Bellman-Ford)

Edsger W. Dijkstra



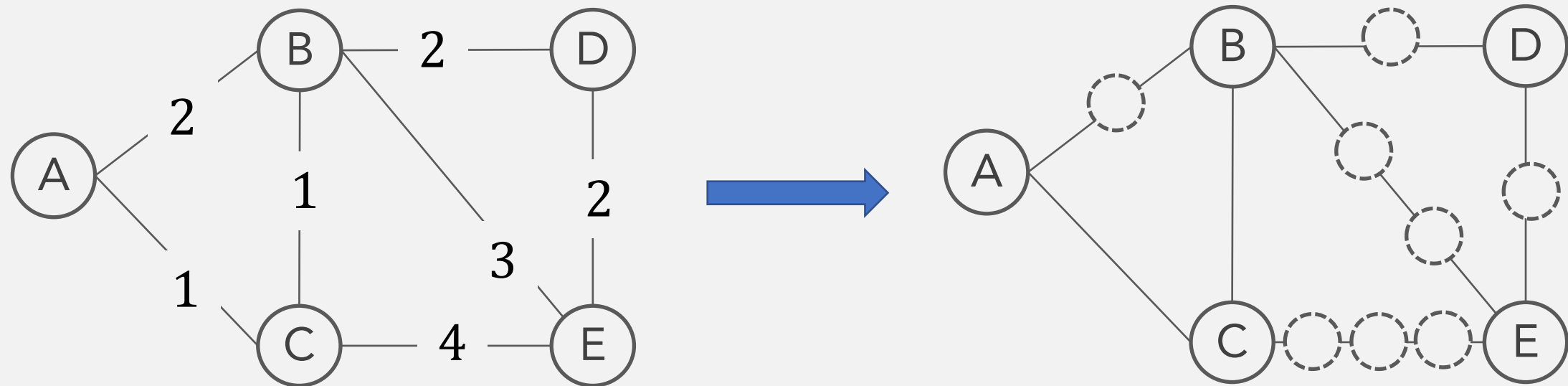
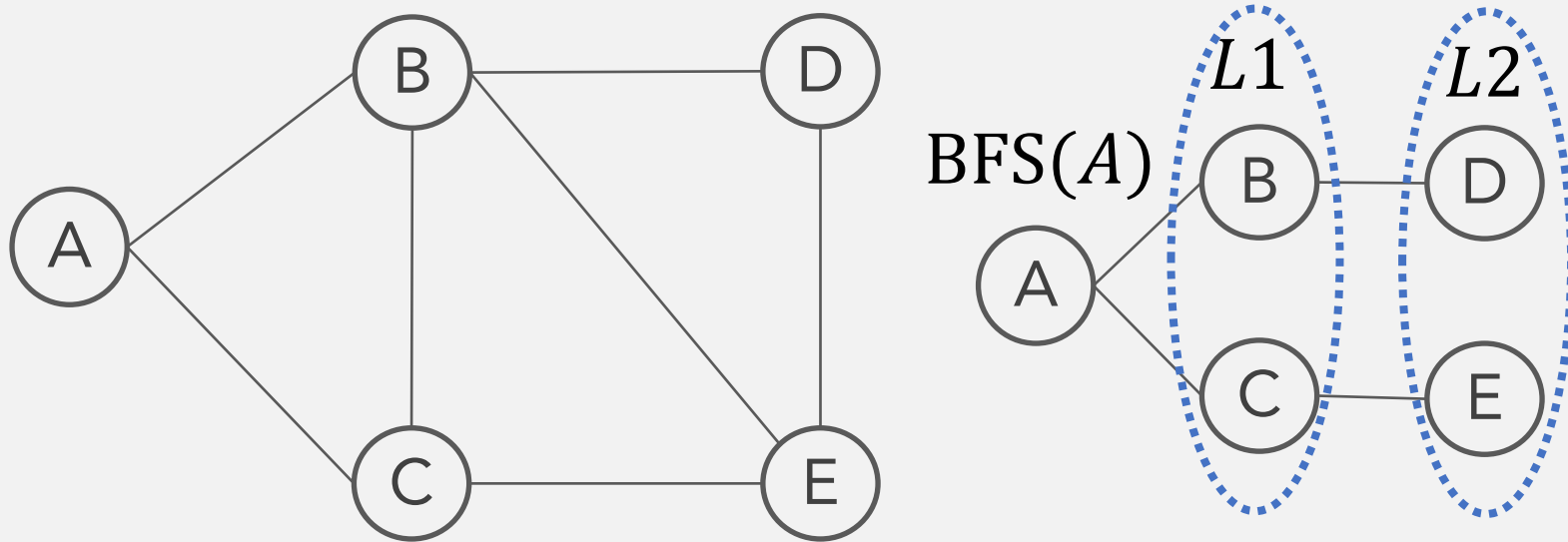
- Pioneer in graph algorithms, distributed computing, concurrent computing, programming ...
- 1984 - 1999 UT Austin, TX
- 2002 passed away in Nuenen, Netherlands



“What's the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about **20 minutes**. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.”

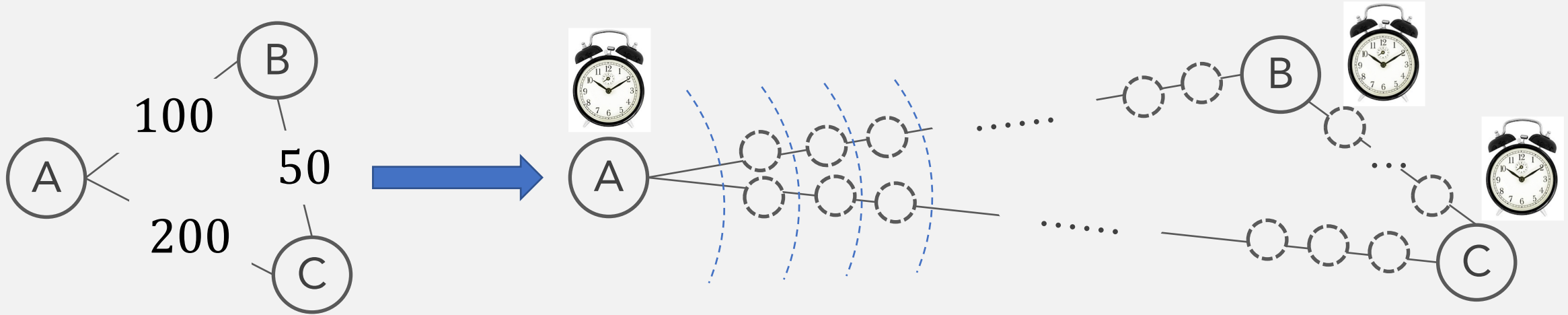
<https://cacm.acm.org/magazines/2010/8/96632-an-interview-with-edsger-w-dijkstra/fulltext>

Reducing to BFS



An alarm-clock algorithm

Idea. Convert G to G' by inserting dummy nodes. Run BFS on G'



AlarmSP(G, s) // set alarm clock for s at time 0

Repeat until no more alarms; Suppose next alarm goes off at T for node u
 $dist(s, u) \leftarrow T$

For each neighbor v of u

If no alarm for v , set one for time $T + l(u, v)$

Else If current alarm larger, reset it for time $T + l(u, v)$

Dijkstra's algorithm: priority queue for alarms

PriorityQueue Q: set of n elements w. associated key values (alarm)

- Change-key(x). change key value of an element
- Delete-min. Return the element with smallest key, and remove it.
- Can be done in $O(\log n)$ time (by a heap)

Dijkstra(G, s) // initialize $d(s) = 0$, others $d(u) = \infty$

Make Q from V using $d(\cdot)$ as key value

While Q not empty
 $u \leftarrow \text{Delete-min}(Q)$ } $O(n \log n)$

// pick node with shortest distance to s

For all edges $(u, v) \in E$
 If $d(v) > d(u) + l(u, v)$
 $d(v) \leftarrow d(u) + l(u, v)$ and Decrease-key(v) } $O(m \log n)$

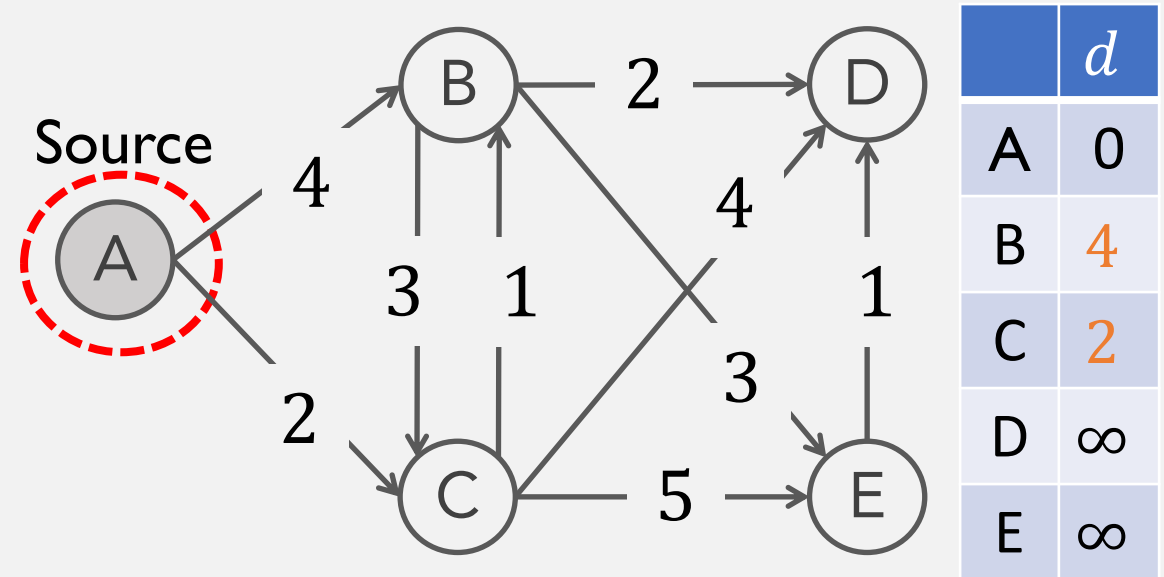
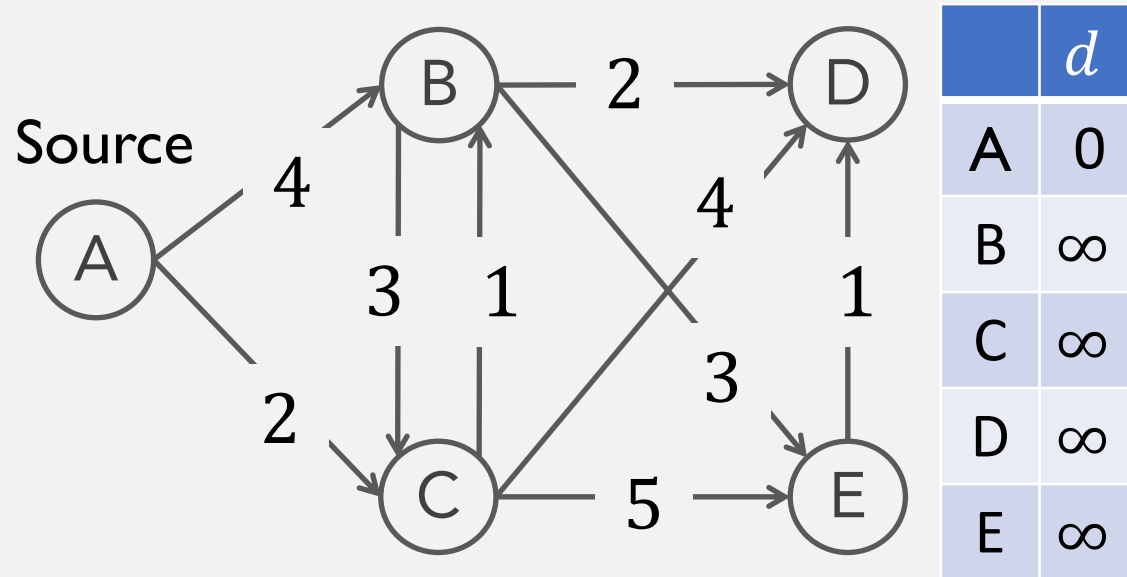
Dijkstra

$O((m + n) \log n)$

Further improvement
possible by Fibonacci
heap [More to come]

N.B. BFS uses ordinary Queue. Dijkstra = BFS+Priority Queue

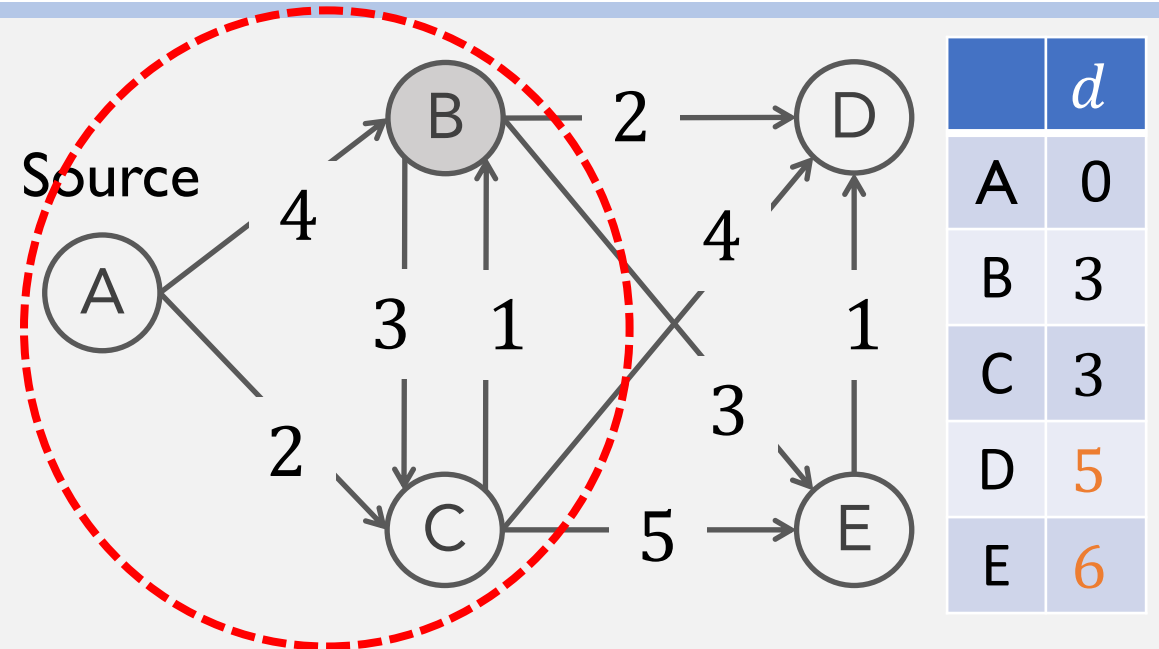
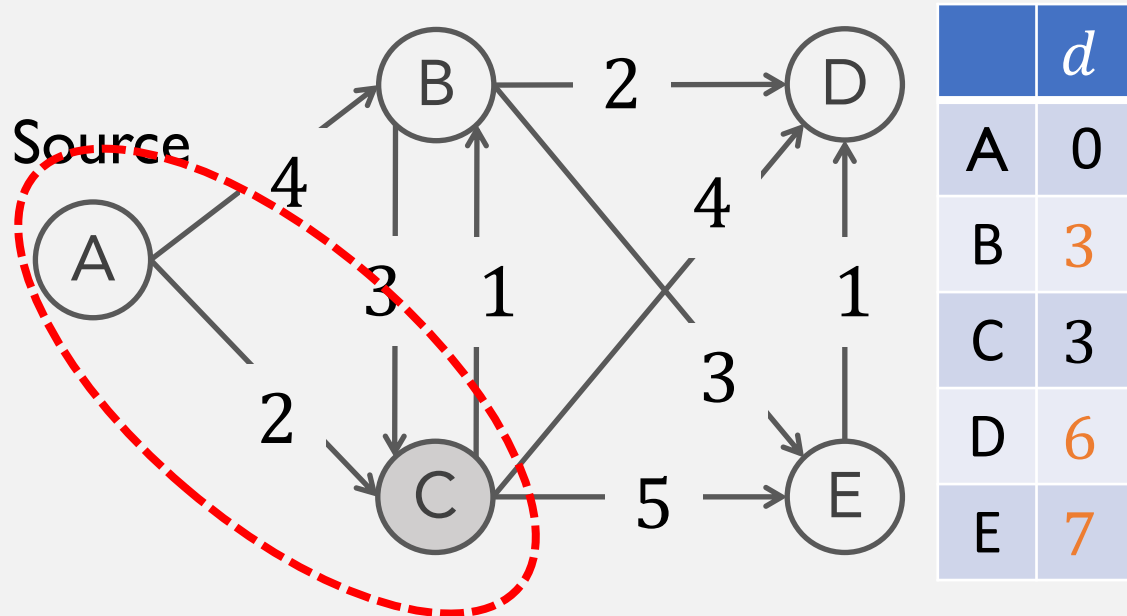
Demo



```

While Q not empty
   $u \leftarrow \text{Delete-min}(Q)$ 
  // pick node with shortest distance to s
  For all edges  $(u, v) \in E$ 
    If  $d(v) > d(u) + l(u, v)$ 
       $d(v) \leftarrow d(u) + l(u, v)$  and Decrease-key(v)
  
```

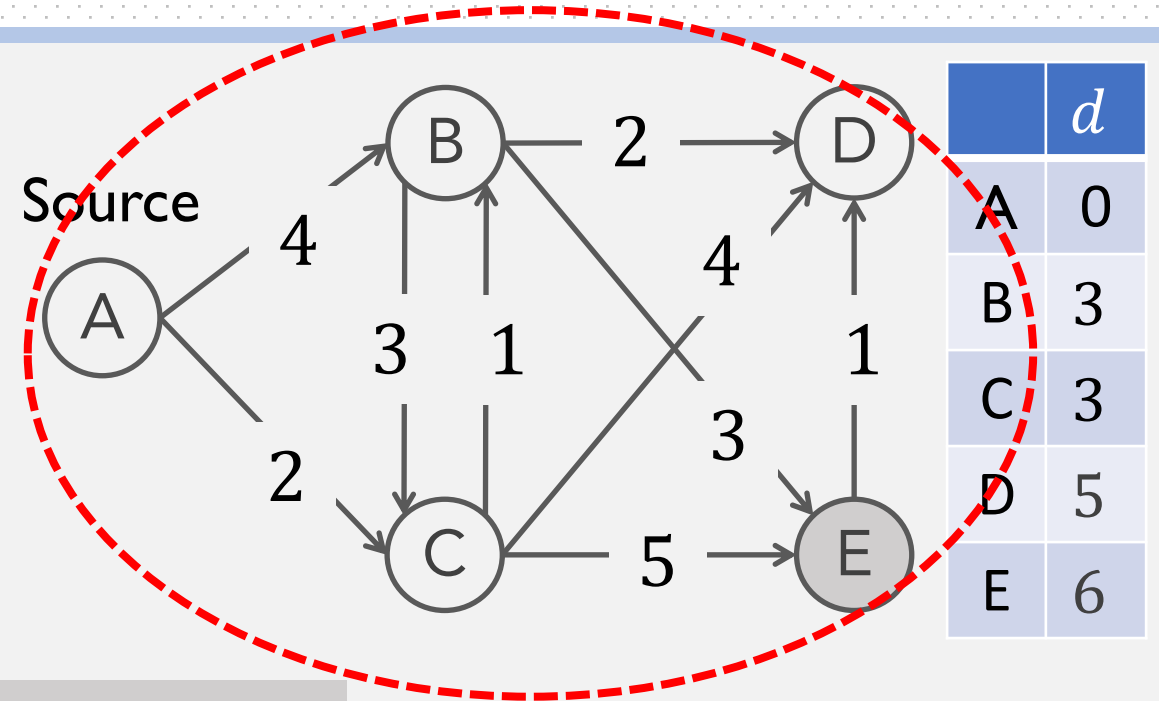
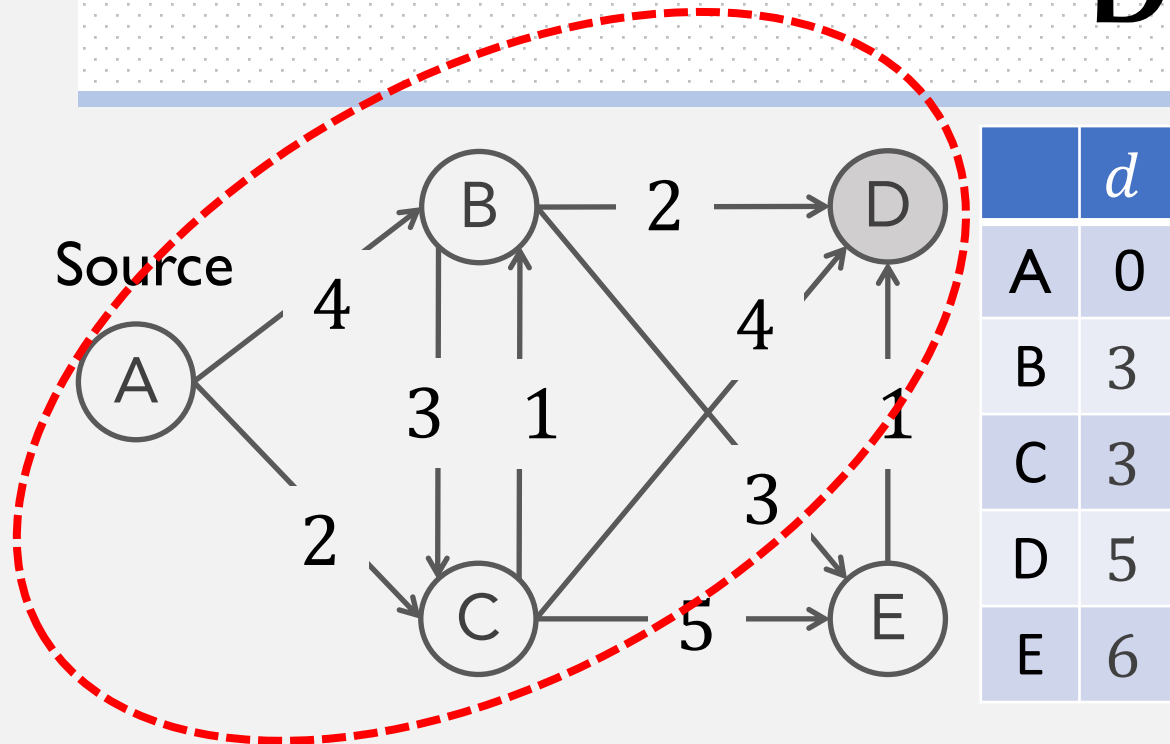

Demo



```

While Q not empty
  u ← Delete-min(Q)
  // pick node with shortest distance to s
  For all edges (u, v) ∈ E
    If d(v) > d(u) + l(u, v)
      d(v) ← d(u) + l(u, v) and Decrease-key(v)
  
```

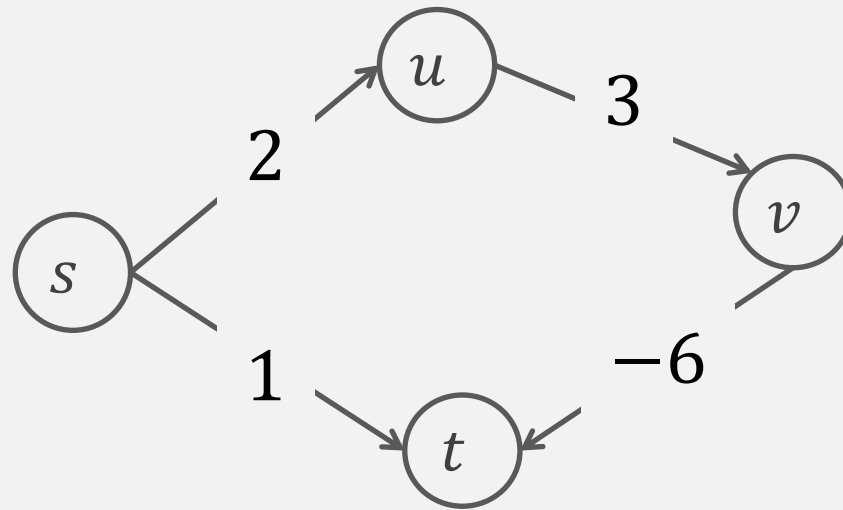
Demo



```

While Q not empty
   $u \leftarrow \text{Delete-min}(Q)$ 
  // pick node with shortest distance to s
  For all edges  $(u, v) \in E$ 
    If  $d(v) > d(u) + l(u, v)$ 
       $d(v) \leftarrow d(u) + l(u, v)$  and Decrease-key(v)
  
```

How it fails on negative lengths?

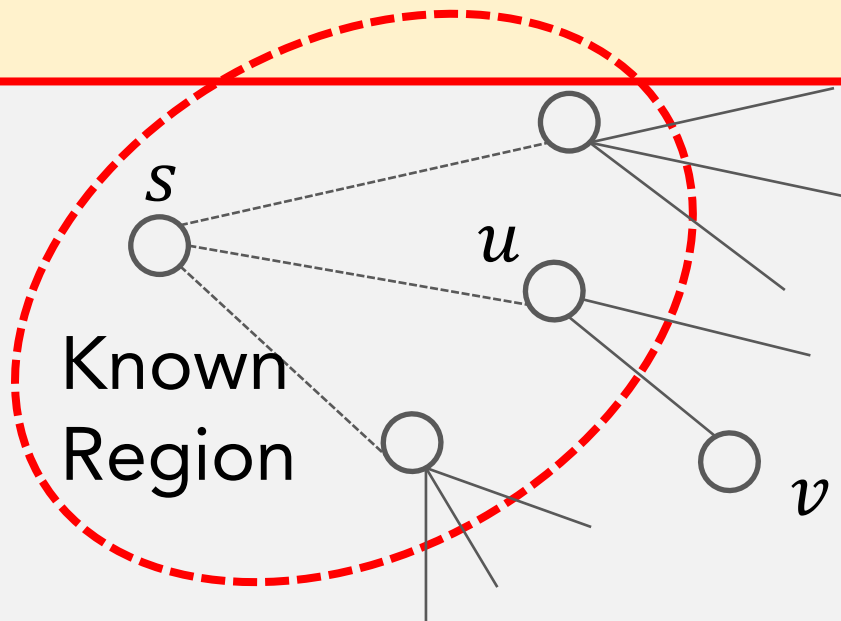


Jumping to a short one too early!

Reflection on Dijkstra: **greedy** stays ahead

- Known region R : in which the shortest distance to s is known
- Growing R : adding v that has the **shortest** distance to s
- How to Identify v ? The one that **minimizes** $d(u) + l(u, v)$ for $u \in R$

Shortest path to some u in known region, followed by a single edge (u, v)



```
Dijk( $G, s$ ) // initialize  $d(s) = 0, d(u) = \infty, R = \emptyset$   
While  $R \neq V$   
    Pick  $v \notin R$  w. smallest  $d(u)$  // by Priority Q  
    Add  $v$  to  $R$   
    For all edges  $(v, w) \in E$   
        If  $d(v) > d(u) + l(u, v)$   
             $d(v) \leftarrow d(u) + l(u, v)$ 
```

Contrast with Bellman-Ford

■ Dijkstra (Greedy)

$$d(v) = \min_{u \in R} d(u) + l(u, v)$$

- Positive weight: no need to wait; more edges in a path do not help

■ Bellman-Ford (Dynamic programming)

$$OPT(i, v) = \min \left\{ OPT(i-1, v), \min_{v \rightarrow w \in E} \{ OPT(i-1, w) + l_{v \rightarrow w} \} \right\}$$

❖ Global vs. Local

- Dijkstra's requires **global** information: known region & which to add
- Bellman-Ford uses only **local** knowledge of neighbors, suits **distributed** setting

Network routing: distance-vector protocol

■ Communication network

- Nodes: routers
- Edges: direct communication links
- Cost of edge: delay on link.

naturally nonnegative, but
Bellman-Ford used anyway!

■ Distance-vector protocol ["routing by rumor"]

- Each router maintains a vector of shortest-path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs n separate computations, one for each potential destination node.

■ Path-vector protocol: coping with dynamic costs