Portland State University

W'21 CS 584/684
Algorithm Design & Analysis

Fang Song

Lecture 10

- Bellman-Ford algorithm, cont'd
- Dijkstra's algorithm

Credit: based on slides by K. Wayne

Recap: shortest path problem

Input: Graph G, nodes s and t .

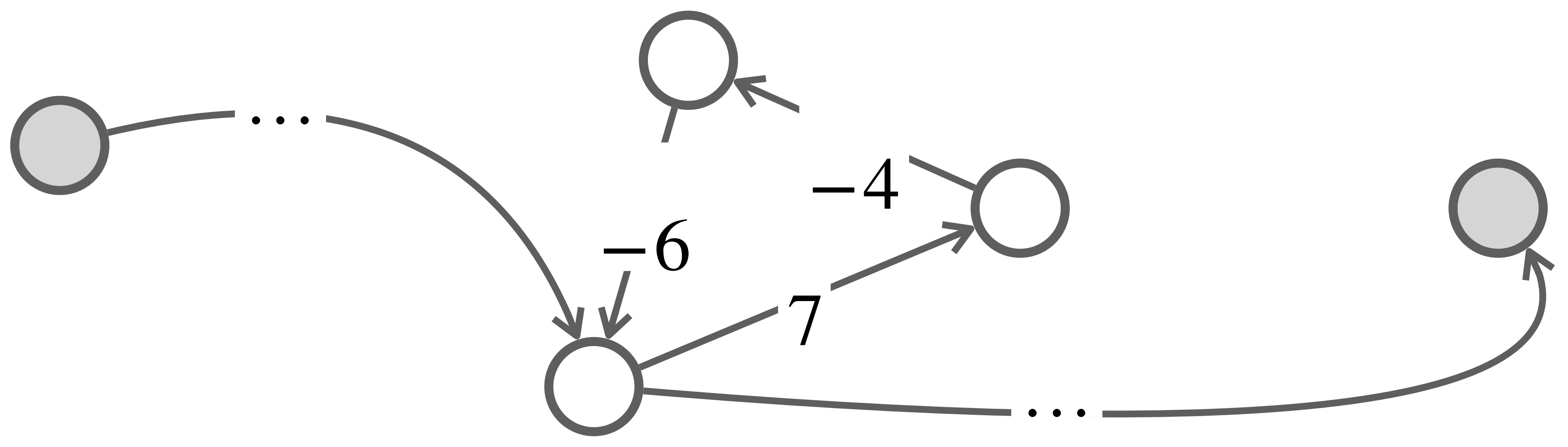
Output: $dist(s, t)$.

- Every edge has a length ℓ_e.
- Length of a path $\ell(P) = \sum_{e \in P} \ell_e$.
- Distance $dist(s, t) = \min_{P:u \rightsquigarrow v} \ell(P)$.

Special cases

- All edges of equal length: BFS $O(m + n)$.
- DAG: DP in topological order $O(m + n)$.

General case: Bellman-Ford algorithm by DP



- Assuming G has no negative length cycle.
- Obs. There exists a simple $s \rightsquigarrow t$ path $\leq n - 1$ edges.

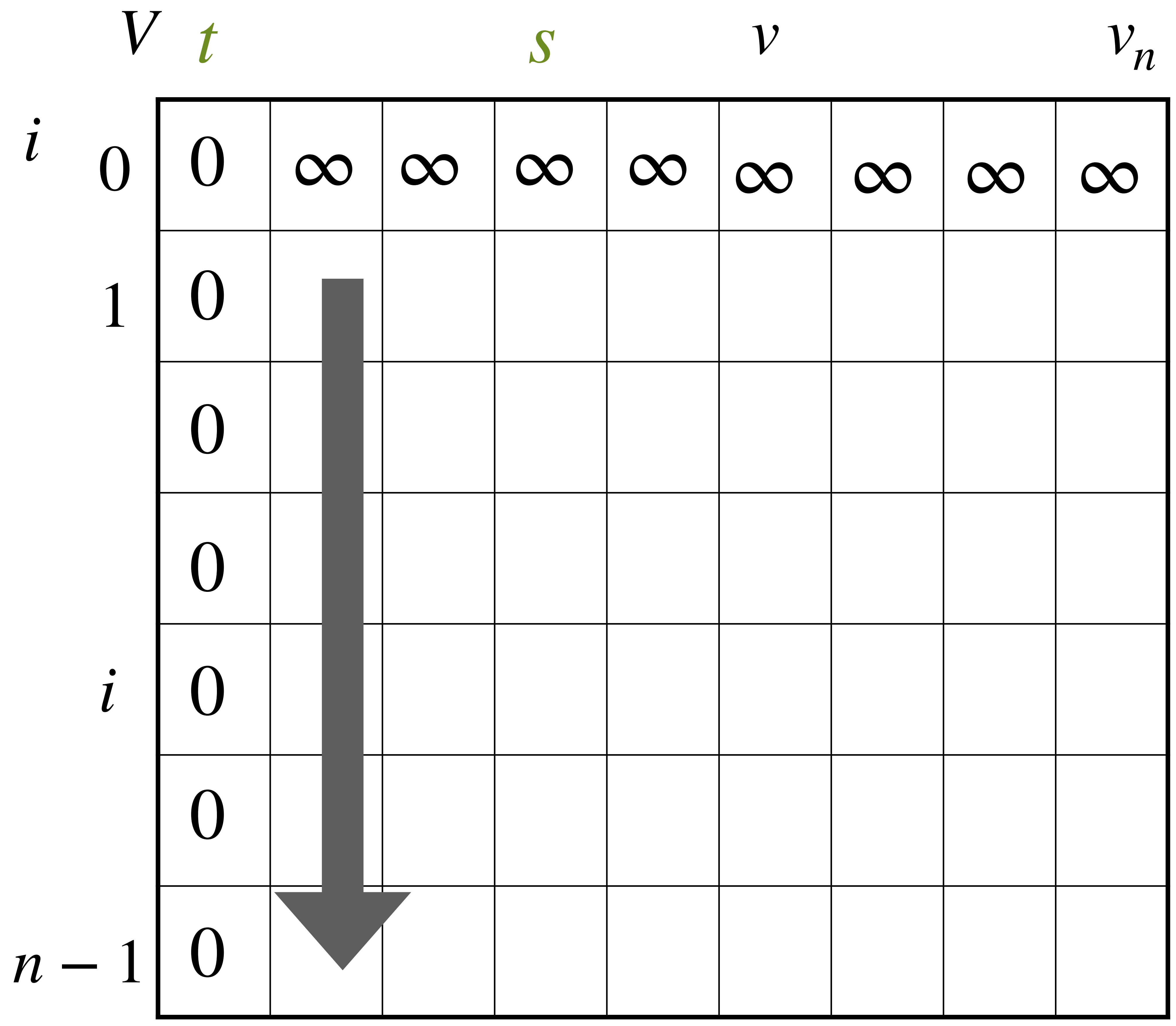
DP1: develop a recurrence

$OPT(i, v) :=$ length of shortest $v \rightsquigarrow t$ path P using $\leq i$ edges.

- **Case 1.** P **uses at most** $i-1$ **edges.** $OPT(i, v) = OPT(i-1, v)$
- **Case 2.** P **uses exactly** i **edges.**
  - If (v, w) is the first edge, then OPT uses (v, w) and then select best $w \rightsquigarrow t$ path using $\leq i-1$ edges.
  - $OPT(i, v) = \min\limits_{v \to w \in E} \{OPT(i-1, w) + \ell_{v \to w}\}$

$$OPT(i, v) = \begin{cases} 0, & \text{if } v = t \\ \infty, & \text{if } i = 0 \\ \min\{OPT(i-1, v), \min_{v \to w \in E}\{OPT(i-1, w) + \ell_{v \to w}\}\}, & \text{otherwise} \end{cases}$$

DP2: build up solutions



- **Subproblems.** $O(n^2)$
- **Memoization data structure**
  - 2-D array $M[0,\ldots,n-1, v_1, \ldots, v_n]$.
- **Dependencies**
  - Each $OPT(i, v)$ depends on subproblems in the row above.
- **Evaluation order**
  - Row by row, arbitrary within a row.

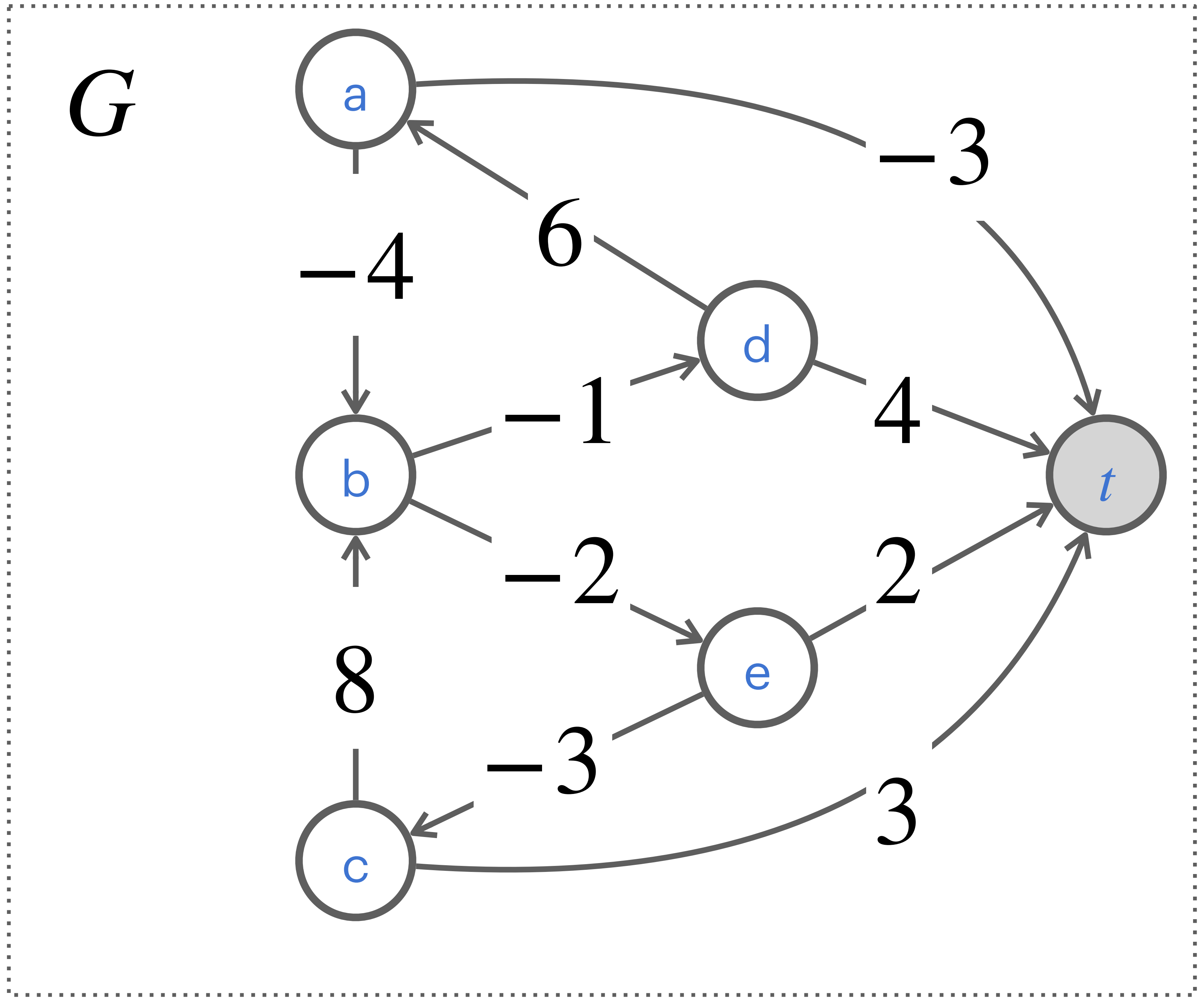
$$OPT(i,v) = \begin{cases} 0, & \text{if } v = t \\ \infty, & \text{if } i = 0 \\ \min\{OPT(i-1,v), \min_{v\to w\in E}\{OPT(i-1,w) + \ell_{v\to w}\}\}, & \text{otherwise} \end{cases}$$

DP2: build up solutions, cont'd

i \ V	t			s		v			v_n
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0								
	0								
	0								
i	0								
	0								
$n-1$	0								

SPLen(G, s, t):
// $M[i, v]$ store subproblem values
// $M[0,t] = 0$, $M[0,v] = \infty$ otherwise.
1. For $i = 1,\ldots,n-1$ // row by row
2. For $v \in V$ // arbitrary order
   $M[i, v] \leftarrow M[i-1,v]$ // case 1
   For edge $v \to w \in E$ // case 2
   $M[i, v] \leftarrow \min\{M[i, v], M[i-1,w] + \ell_{vw}\}$
3. Return $M[n-1,s]$

Example



i \ V	t	A	B	C	D	E
0	0	∞	∞	∞	∞	∞
1	0					
2	0					
3	0					
4	0					
5	0					

For $v \in V$ // arbitrary order

$M[i, v] \leftarrow M[i-1, v]$ // case 1

For edge $v \to w \in E$ // case 2

$M(i, j) \leftarrow \min\{M[i, v], M[i-1, w] + \ell_{vw}\}$

A simple but impactful improvement

Maintain only one array $M[v] =$ length of shortest $v \rightsquigarrow t$ path found so far.

No need to check edge (v, w) unless $M[w]$ changed in previous iteration.

- **Theorem.** Throughout the algorithm, $M[v]$ is the length of some $v \rightsquigarrow t$ path, and after i rounds of updates, the value $M[v]$ is no larger than the length of shortest $v \rightsquigarrow t$ path using $\leq i$ edges.
- Memory: $O(m + n)$.
- Running time: $O(mn)$ worst case, but faster in practice.
- Bellman-Ford algorithm: efficient implementation

Single-source shortest path with negative weights

Year	Worst case	Discovered by
1955	$O(n^4)$	Shimbel
1956	$O(mn^2W)$	Ford
1958	$O(mn)$	Bellman, Moore
1983	$O(n^{3/4}m \log W)$	Gabow
1989	$O(mn^{1/2} \log(nW))$	Gabow-Tarjan
1993	$O(mn^{1/2} \log W)$	Goldberg
2005	$O(n^{2.38}W)$	Sankowsi, Yuster-Zwich
2016	$O(n^{10/7} \log W)$	Cohen-Madry-Sankowski-Vladu
20xx	???	you?

Weights between $[-W, W]$

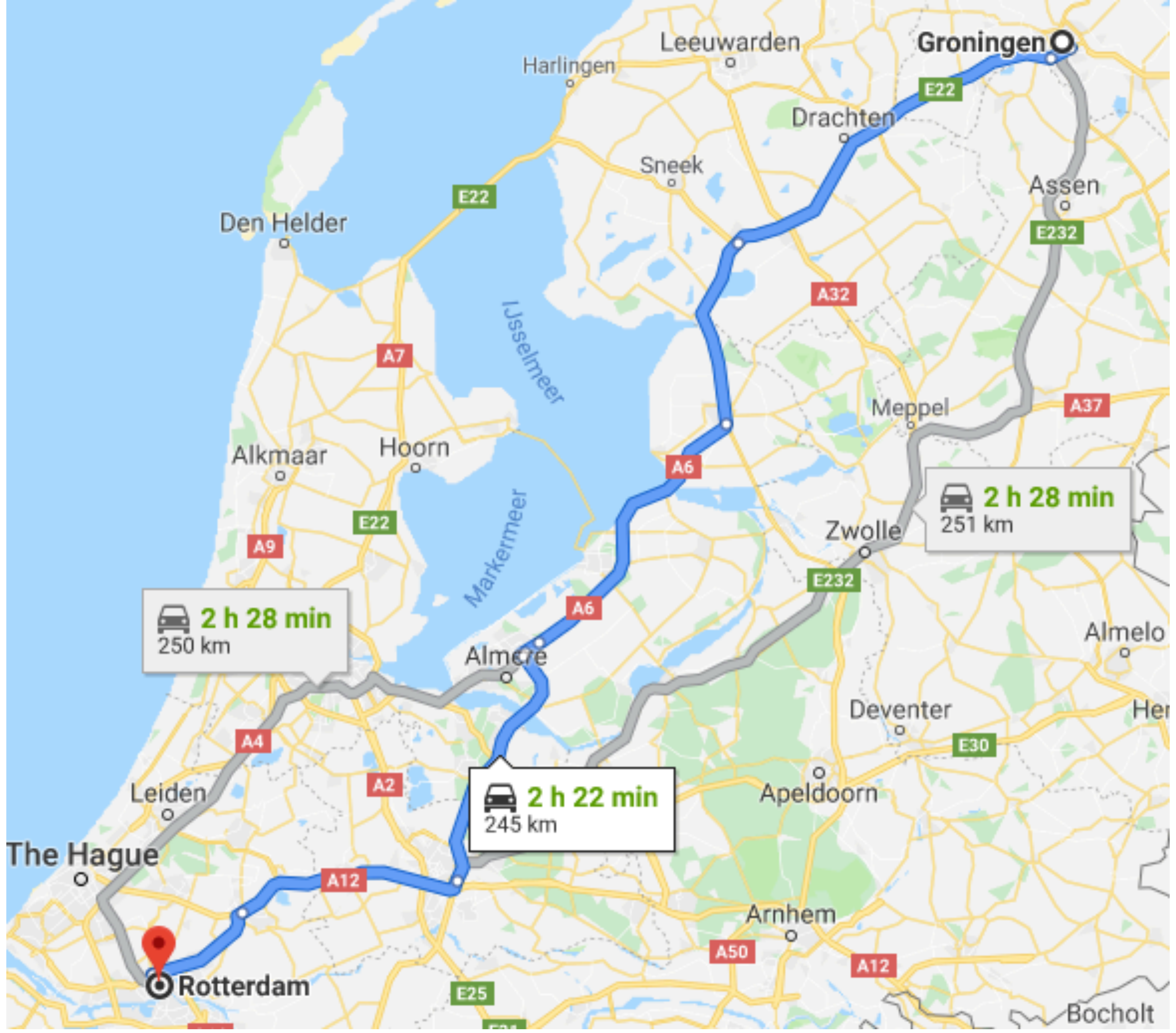
Dynamic programing re-recap

1. **Formulate the problem recursively (key step).**
   - Overlapping subproblems.
   - May be easier to first compute optimal value & then construct a solution.
2. **Build solutions to your recurrence (kinda routine).**
   - Top-down: smart recursion (i.e. without repetition) by momoization.
   - Bottom-up: determine dependencies & a right order (topo. order in DAG).

- **Examples.** $O(n^2)$
  - Explicit DAG: shortest/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

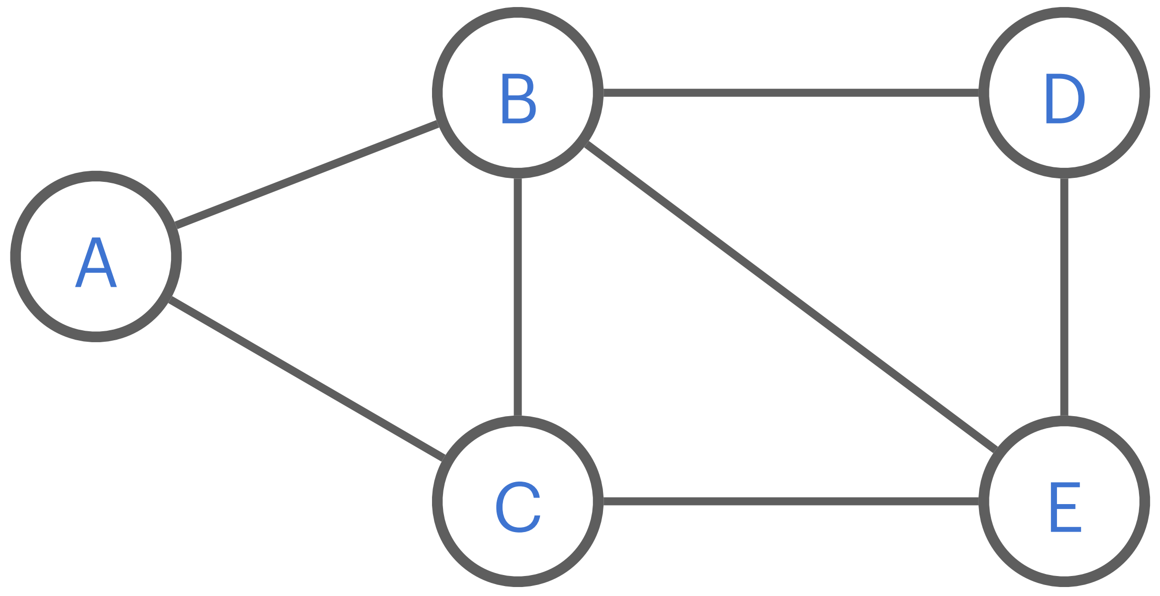
- Pioneer in graph algorithms, distributed computing, concurrent computing, programming ...

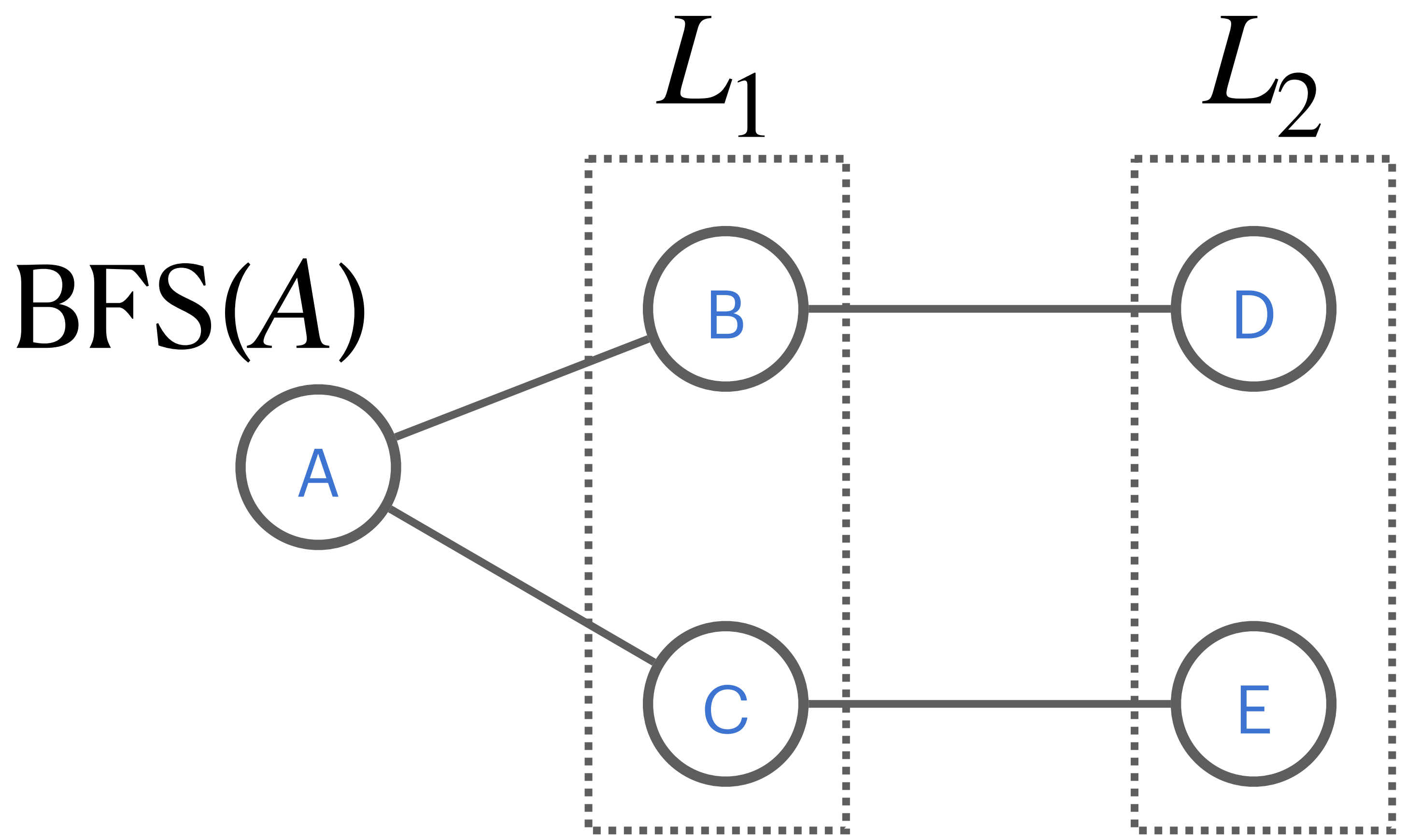


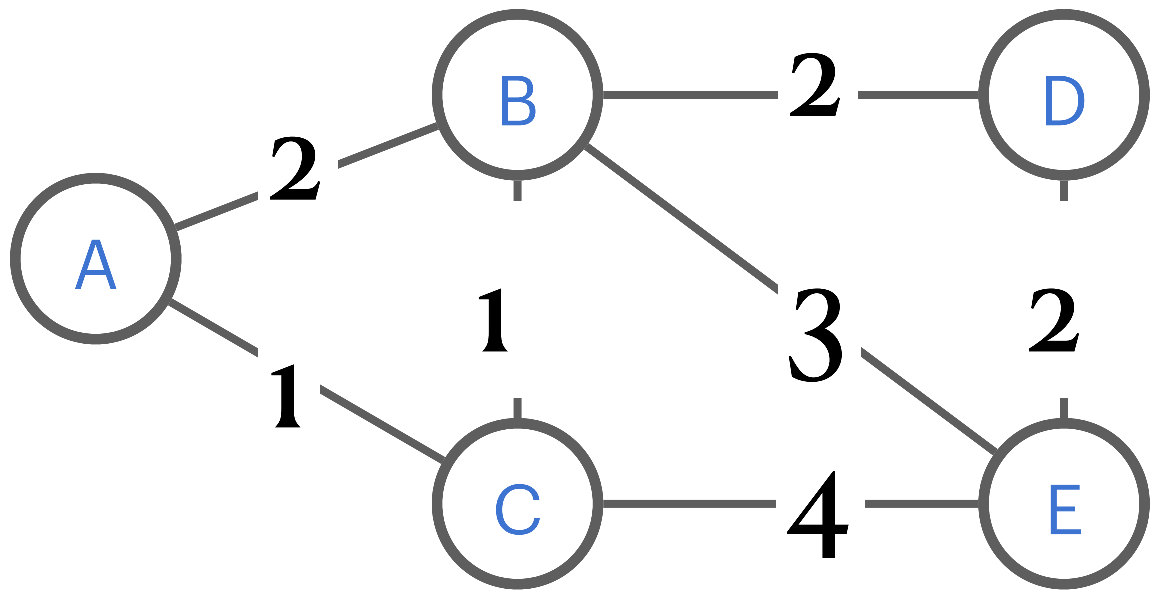
"What's the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes.
One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about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

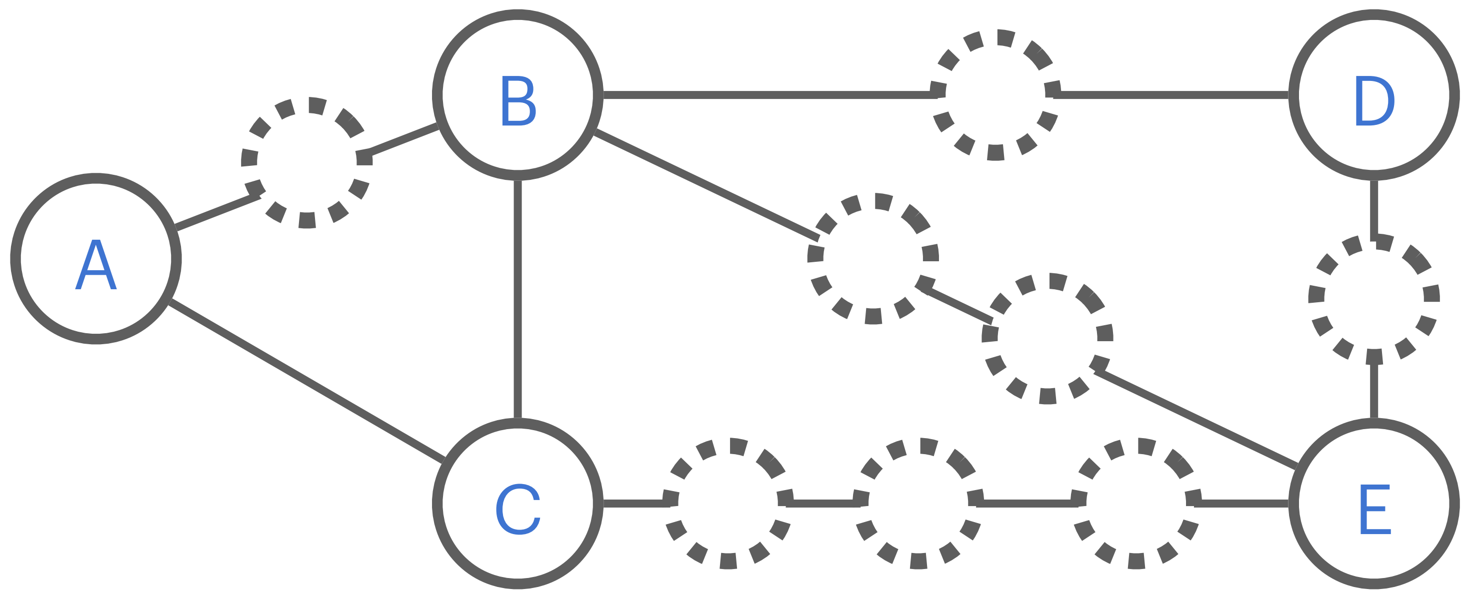
https://cacm.acm.org/magazines/2010/8/96632-an-interview-with-edsger-w-dijkstra/fulltext

Reducing to BFS



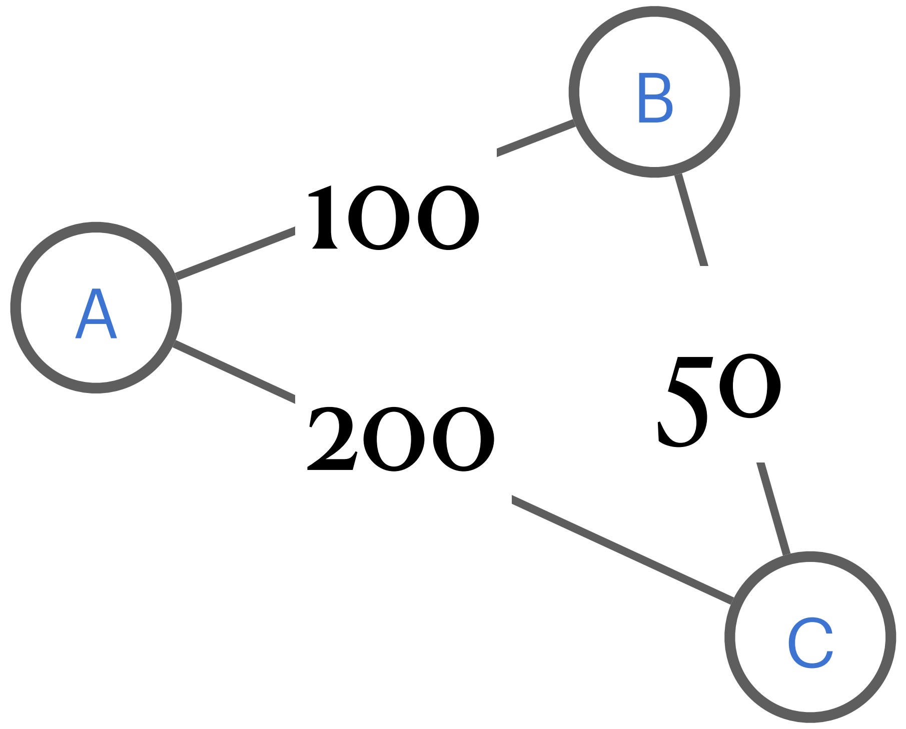


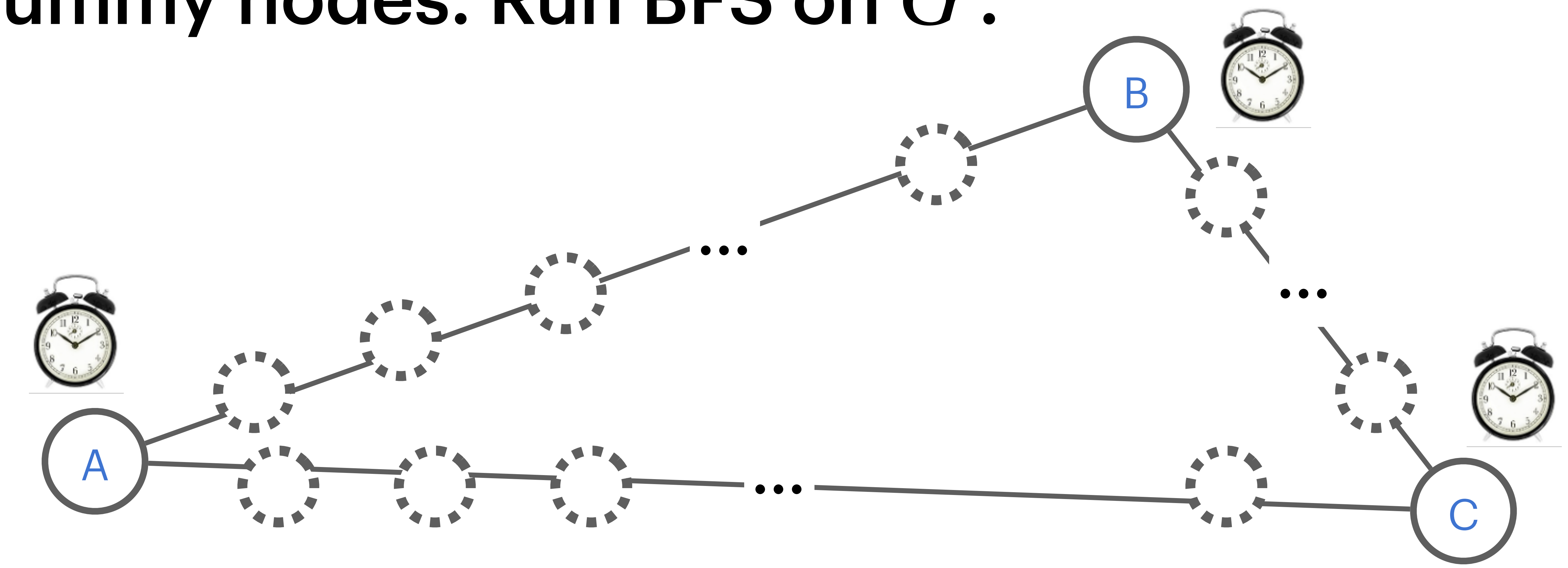




An alarm-clock algorithm

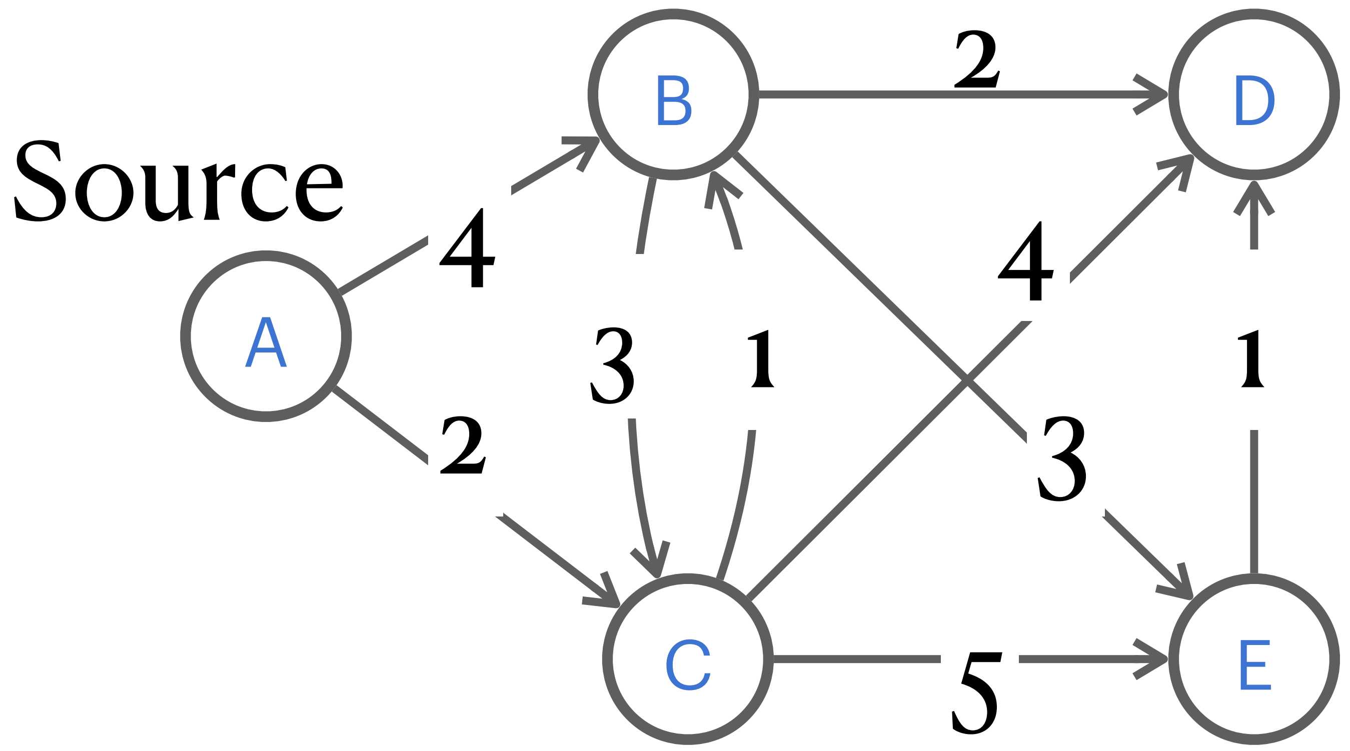
- **Idea.** convert G to G' by inserting dummy nodes. Run BFS on G'.



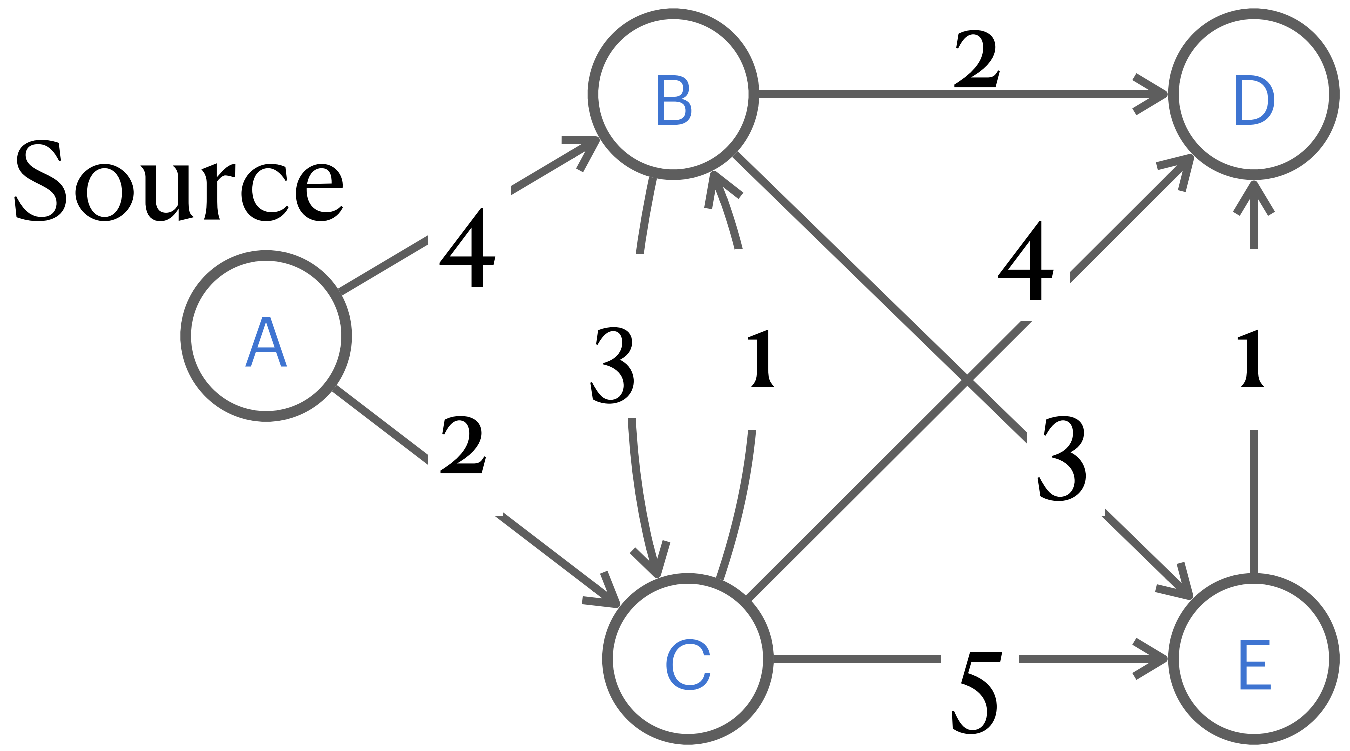


AlarmSP(G, s):
// set alarm clock for s at time 0
Repeat until no more alarms
// Suppose next alarm goes off at T for node u
    $dist(s, u) \leftarrow T$
    For each neighbor v of u
        If no alarm for v, set one for time $T + \ell(u, v)$
        Else If current alarm larger, reset it for time $T + \ell(u, v)$

Demo



Demo



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).

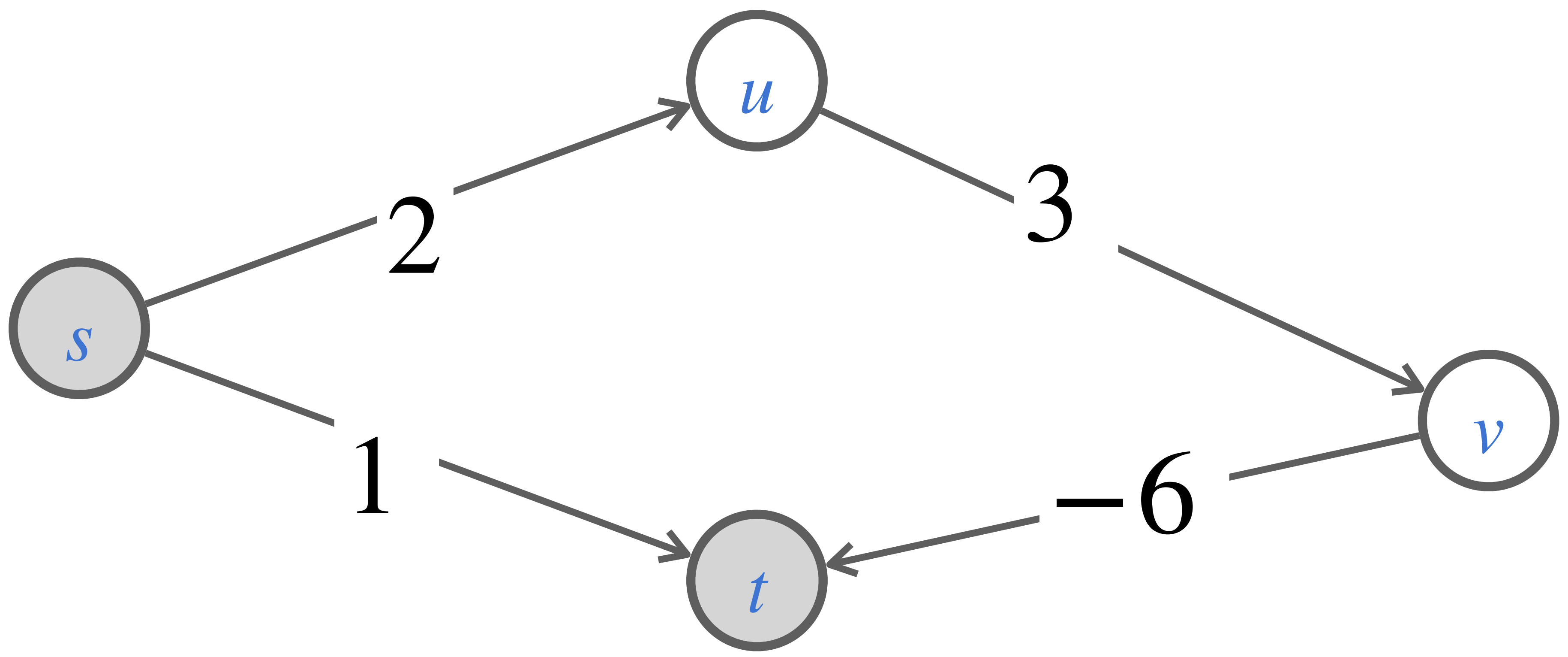
Dijkstra(G, s):
// initialize $d(s) = 0$, others $d(u) = \infty$
1. Make Q from V using $d(\cdot)$ as key value
2. While Q not empty
   $u \leftarrow$ Delete-min(Q) — $O(n \log n)$
   // pick node with shortest distance to s
   For all edges $(u, v) \in E$
     If $d(v) > d(u) + \ell(u, v)$
       $d(v) \leftarrow d(u) + \ell(u, v)$ and Change-key(v) — $O(m \log n)$

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

Further improvement possible by Fibonacci heap

NB. BFS uses ordinary Queue.
Dijkstra = BFS w/ priority queue

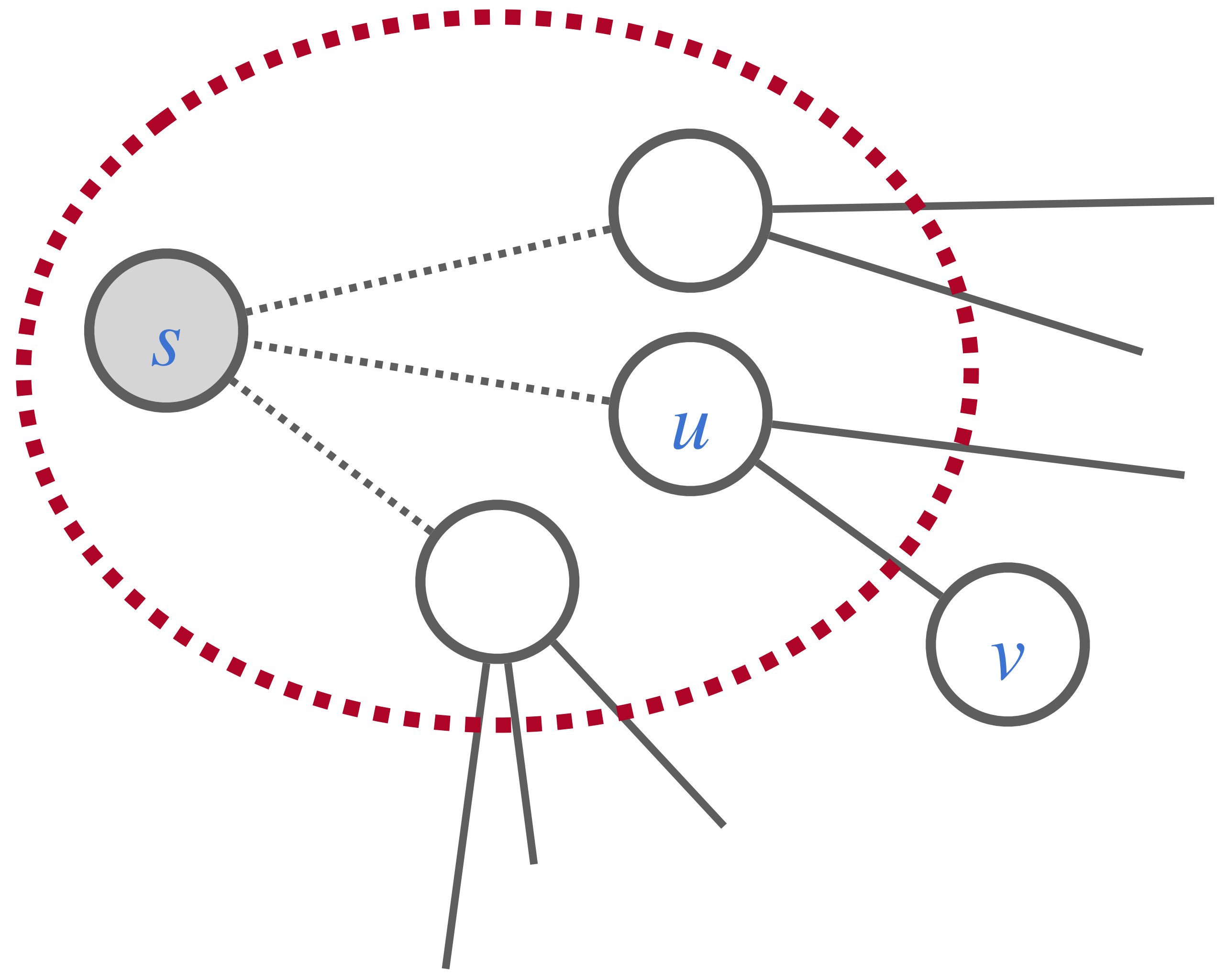
How it fails on negative lengths



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 $d(u) + \ell(u, v)$
  - Shortest path to some u in known region, followed by a single edge (u, v).



Dijkstra(G, s):
// initialize $d(s) = 0$, $d(u) = \infty$, $R = \emptyset$
1. 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 $d(v) > d(u) + \ell(u, v)$
       $d(v) \leftarrow d(u) + \ell(u, v)$

Scratch