CSE 331:
Algorithms & Complexity

"Dijkstra's Algorithm"

Prof. Charlie Anne Carlson (She/Her)

Lecture 18

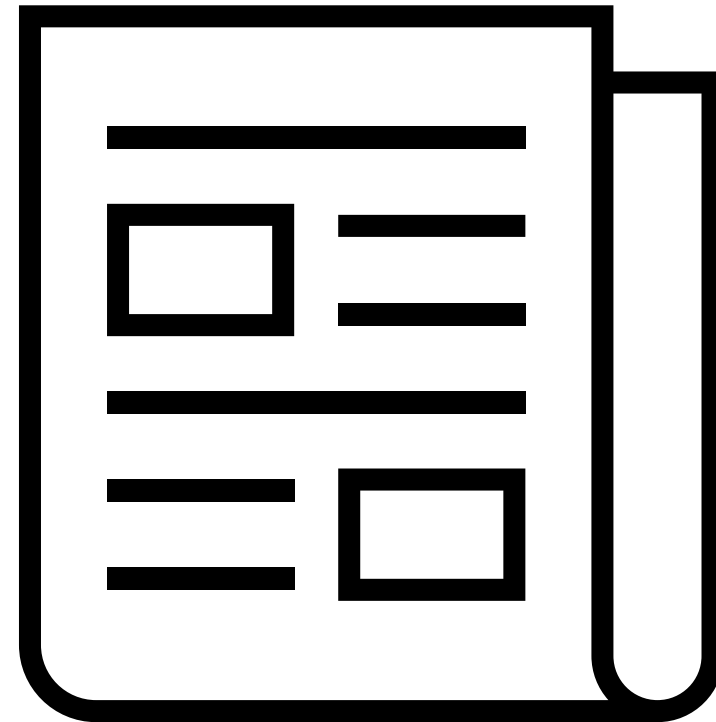Wednesday October 15, 2025

University at Buffalo

Schedule

1. Course Updates
2. Shortest Path
3. Dijkstra's Algorithm
4. Proof of Correctness
5. Runtime

Course Updates

- Everything Besides Midterm Graded
  - Midterm Grades this week
- HW 4 Out
  - Due Next Week!
- Group Project
 - First Problems Oct 31st

Mid Updates

- You got mid semester grades based only on HW1, HW2, HW3, and Quiz 1 (no drops).
  - I will give you more formal grades once midterms are graded.
- Midterms seemed to have been a stumble for a lot of people.
 - This is a chance to do better and not a chance to give up.
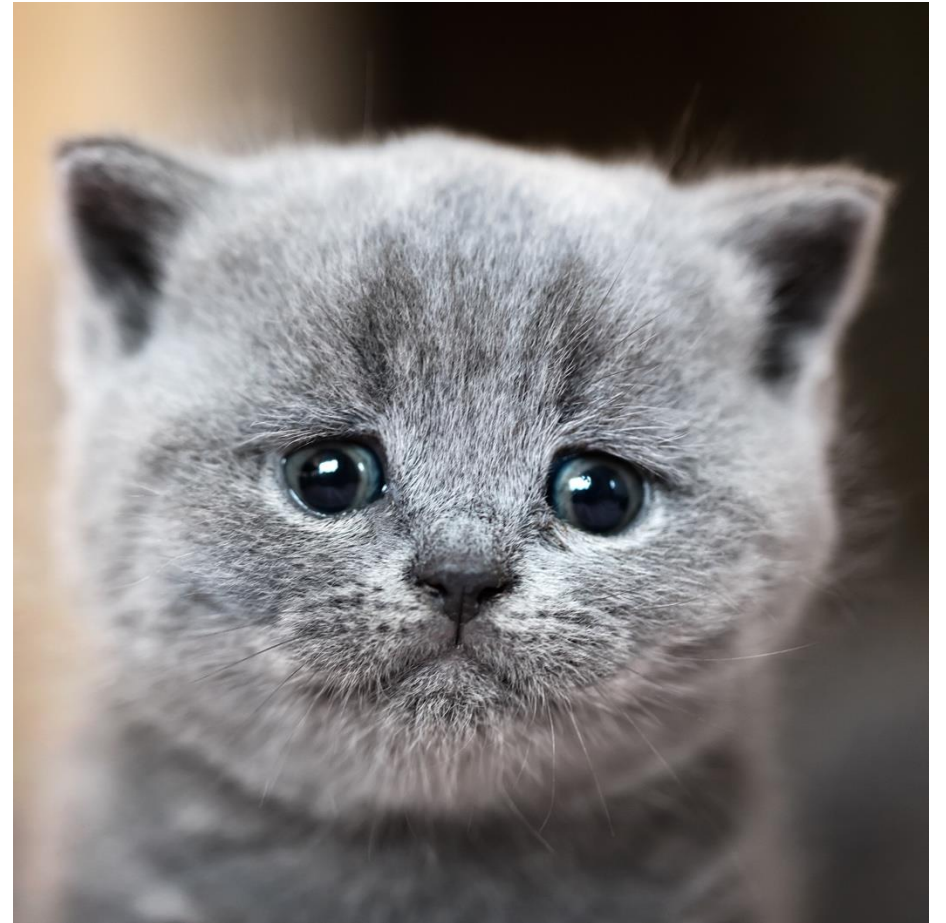
Cheating!

"If the **violation involves the use of ChatGPT (or other generative AI tools)**, irrespective of whether it is the student's first violation or not, then it will result in an automatic F letter grade in the course." <- Course Syllabus

- We know of a few people who are using generative AI tools on homework assignments.

- Just because you got a grade, doesn't mean I don't know you cheated and you are in the clear.
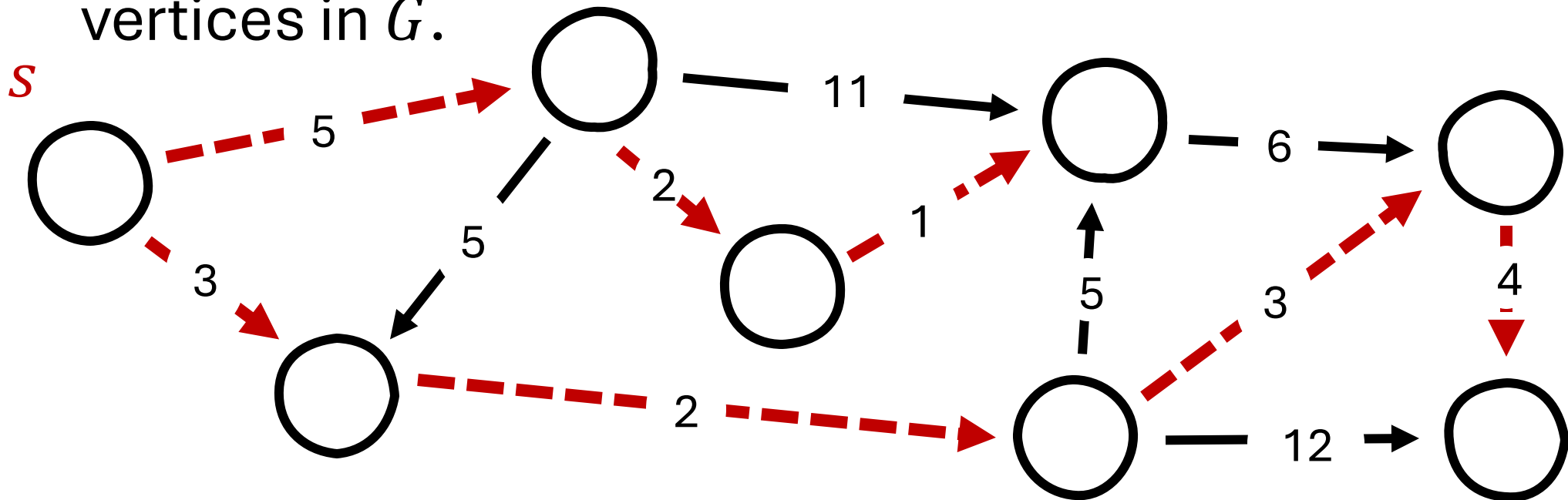
F
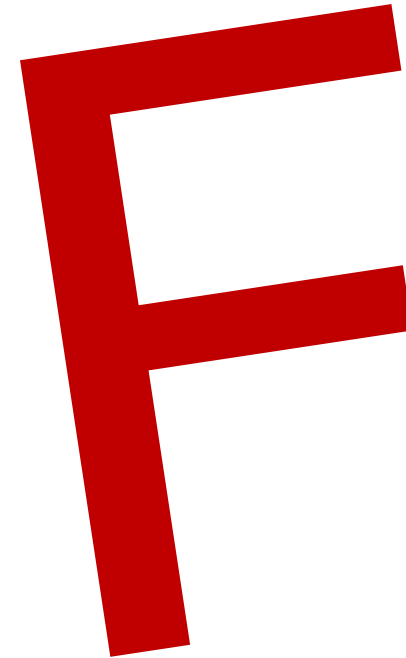
Single Pair Shortest Path
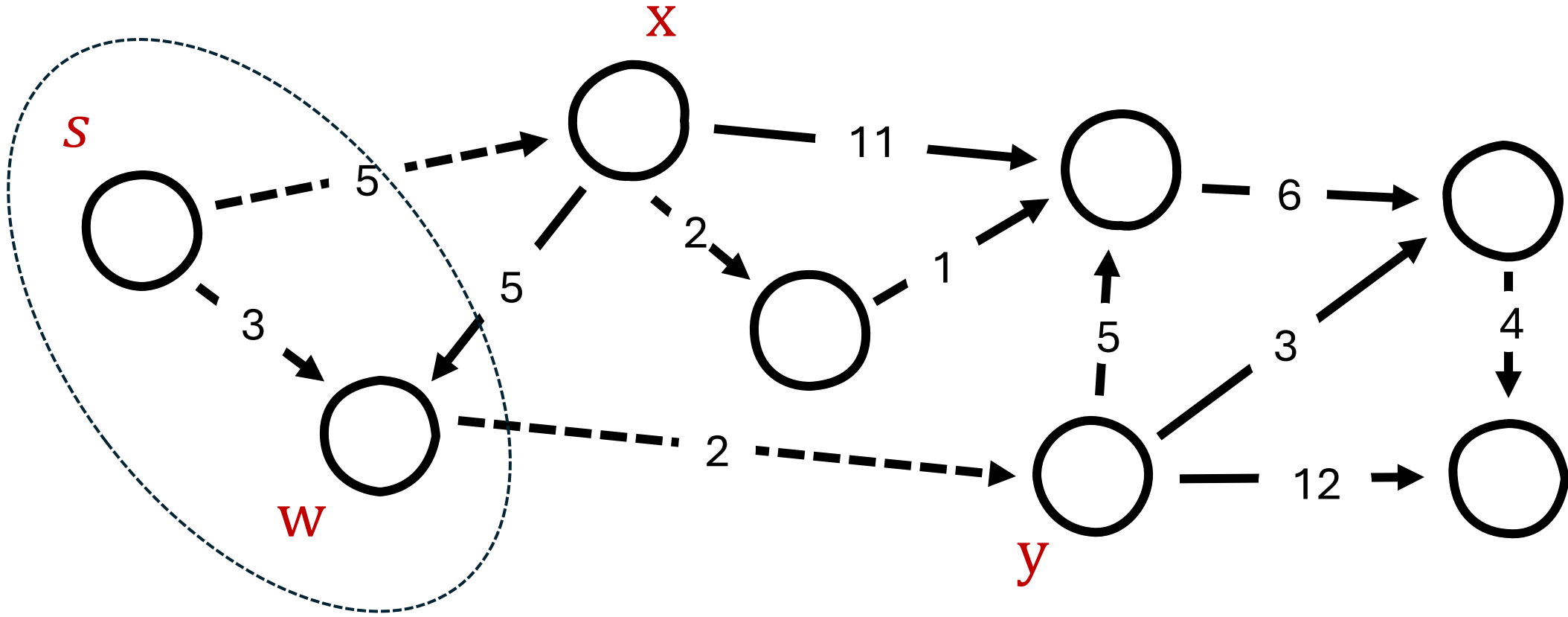
- **Input**: ?
- **Output**: ?
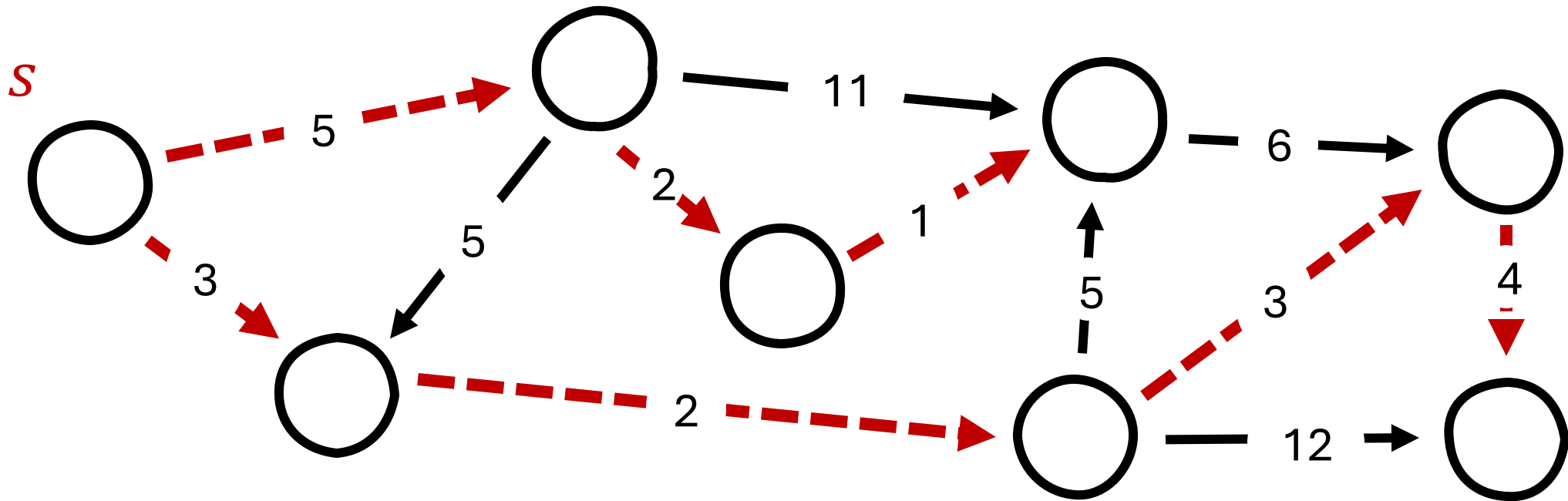
Single Pair Shortest Path

- **Input**: Directed graph $G = (V, E)$, edge lengths $\ell: E \to R_{\geq 0}$, and a source vertex $s \in V$.
- **Output**: Find the shortest directed path from s to all vertices in G.
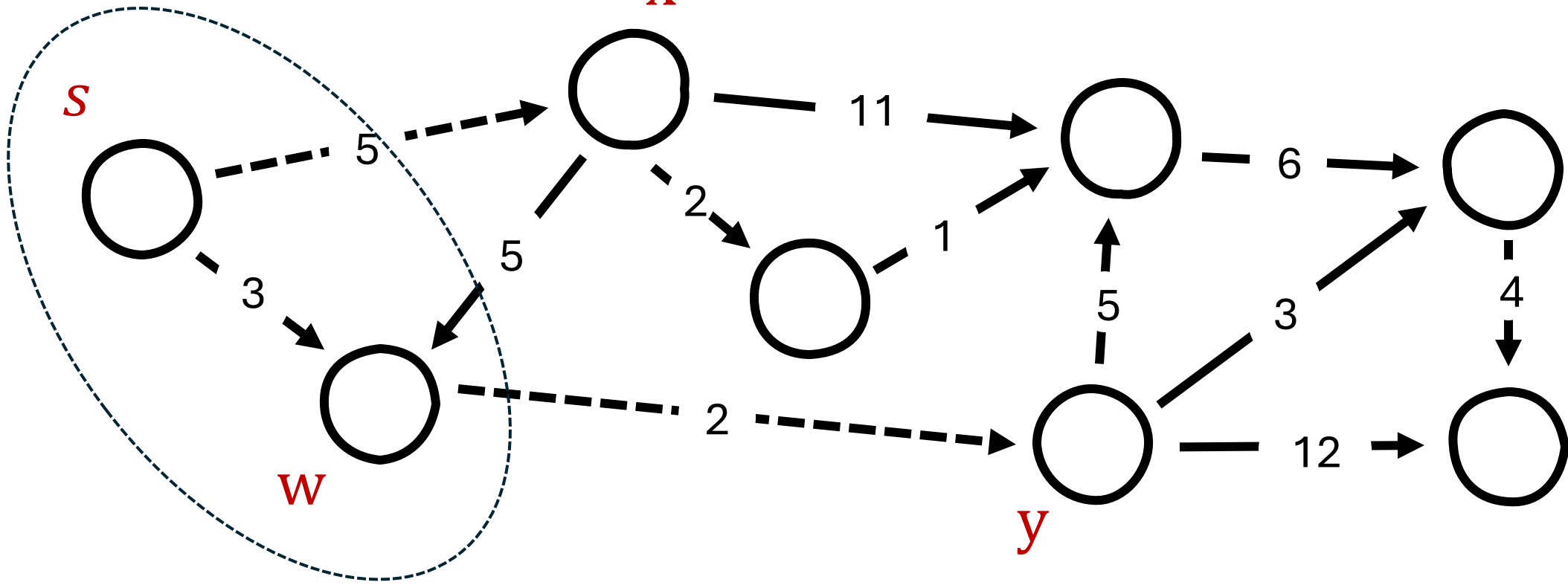

Shortest Path Tree

Dijkstra's Algorithm Idea

- Algorithm Idea: ?

Dijkstra's Algorithm Idea

- Algorithm Idea: In each iteration, pick an edge from my shortest path tree to the "nearest" vertex not yet in the tree. Nearest means distance to the root, s.

Dijkstra's Algorithm

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

$$d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e \text{ is as small as possible}$$

    Add v to S and define $d(v) = d'(v)$

EndWhile

y

Dijkstra's Algorithm

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a dist

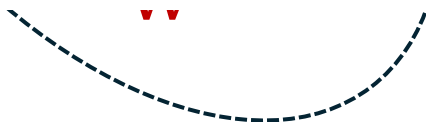Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

    $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and define $d(v) = d'(v)$

EndWhile

This is the distance from s to something on the "frontier".

y

Dijkstra's Algorithm Idea

- Look at all neighbors of S. Determine which has the shortest path from s. Add that to S. Repeat.

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
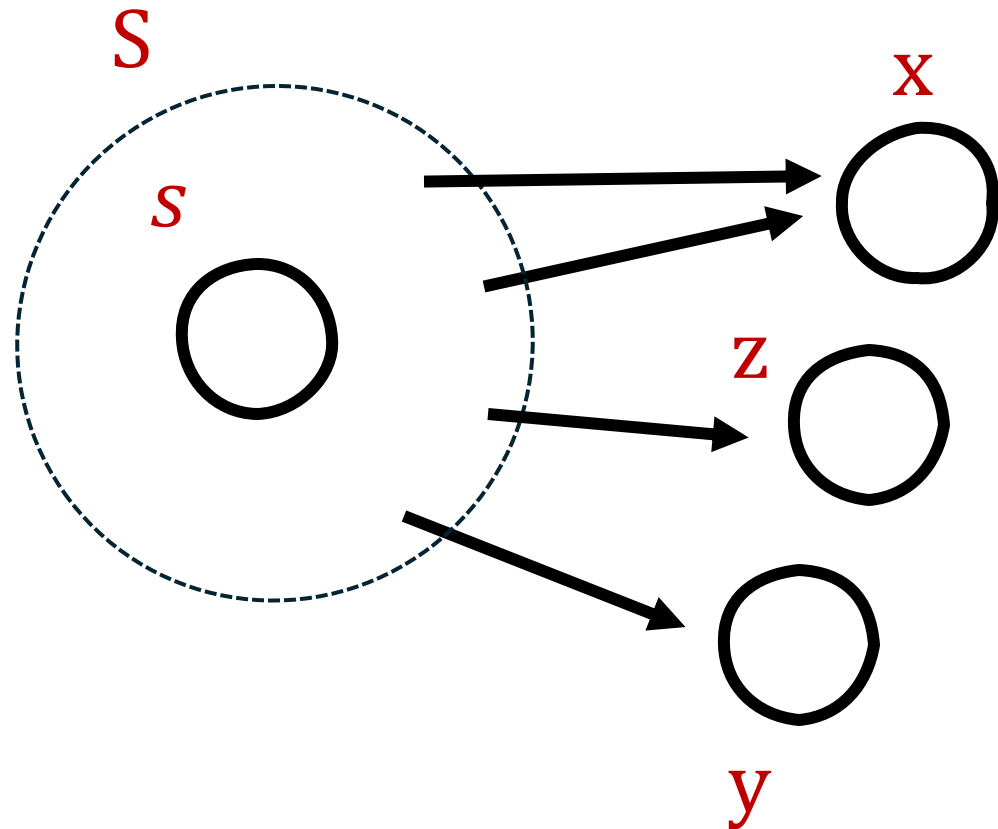 $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

Dijkstra's Algorithm

S = {s}

d = [0, ∞, ∞, ∞, ∞, ∞, ∞, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, ∞, ∞, ∞, ∞, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
      For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
      Select a node v ∉ S with at least one edge from S for which
            d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
      Add v to S and define d(v) = d'(v)
EndWhile
```
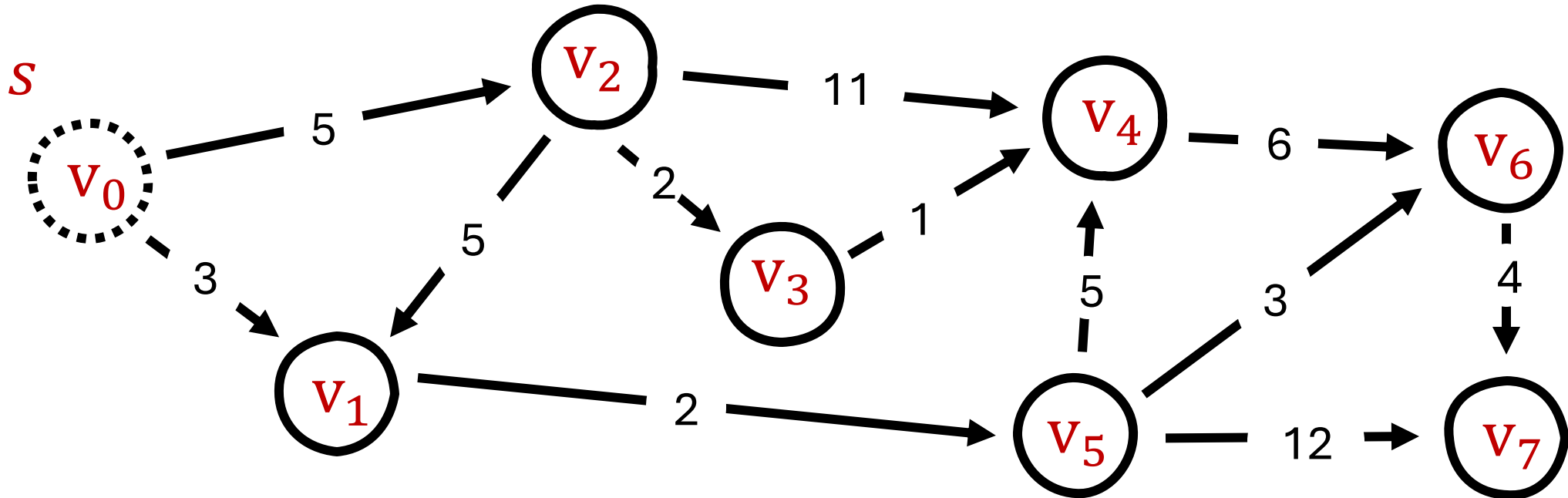
Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, ∞, ∞, 5, ∞, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, 7, ∞, 5, ∞, ∞]

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
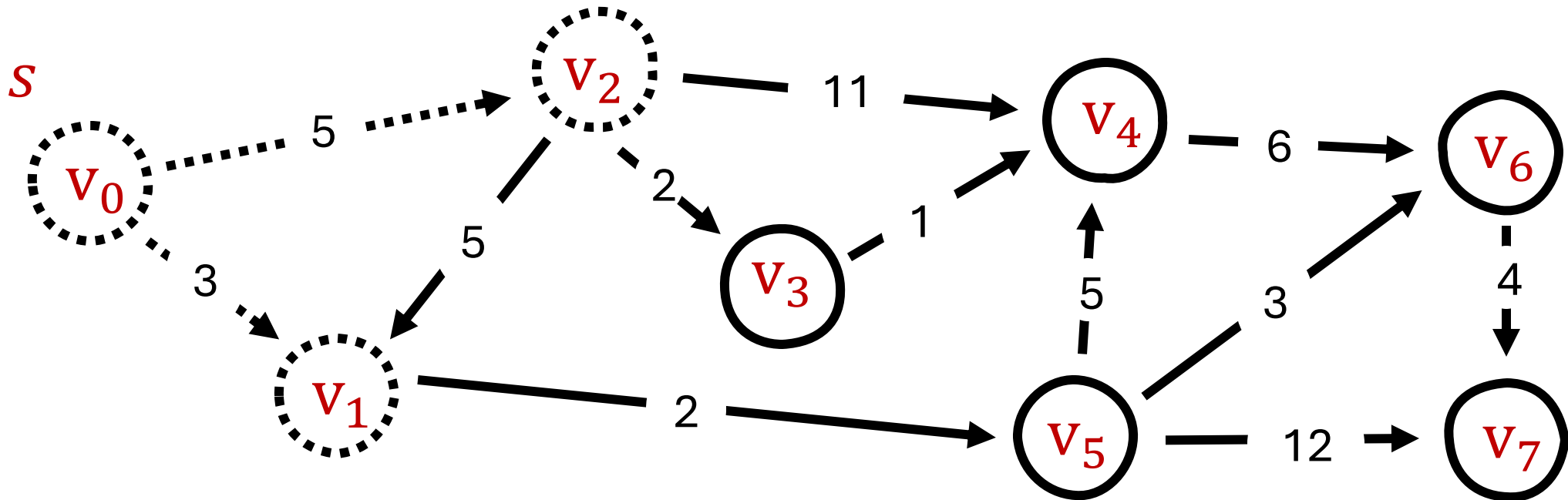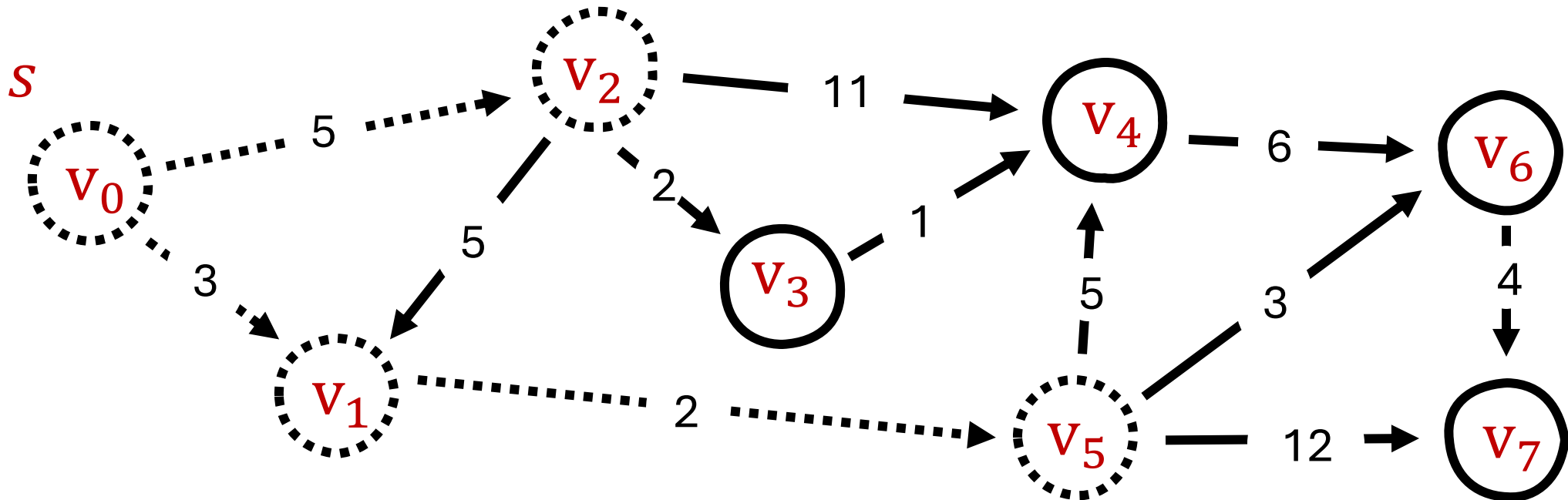Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, 7, 8, 5, ∞, ∞]

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
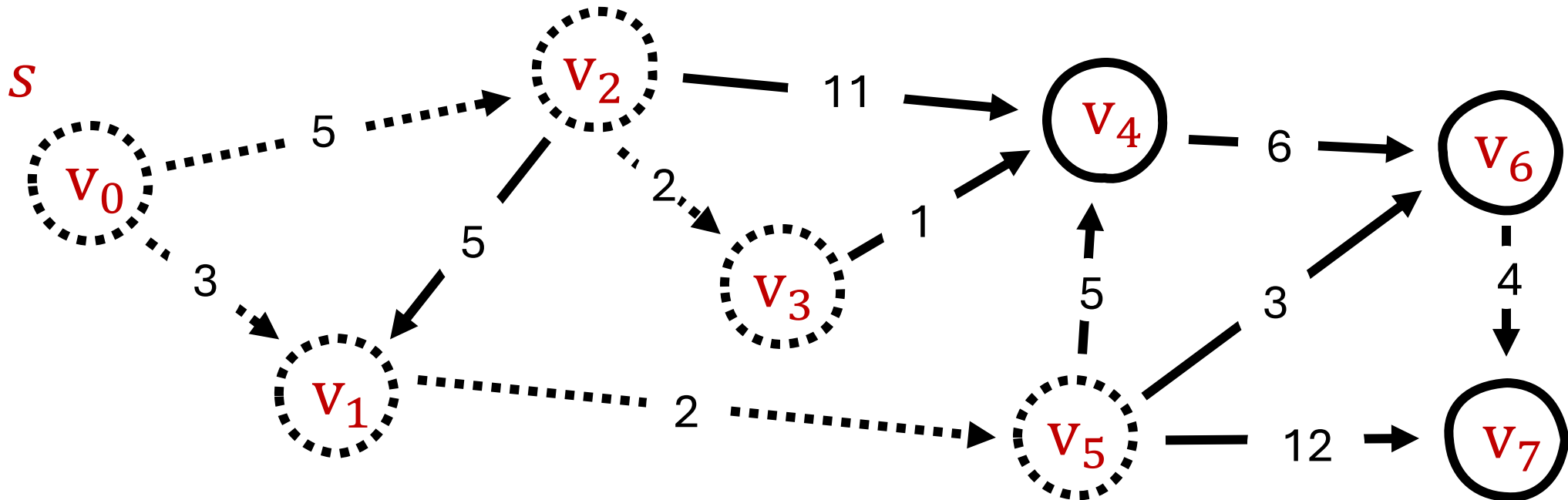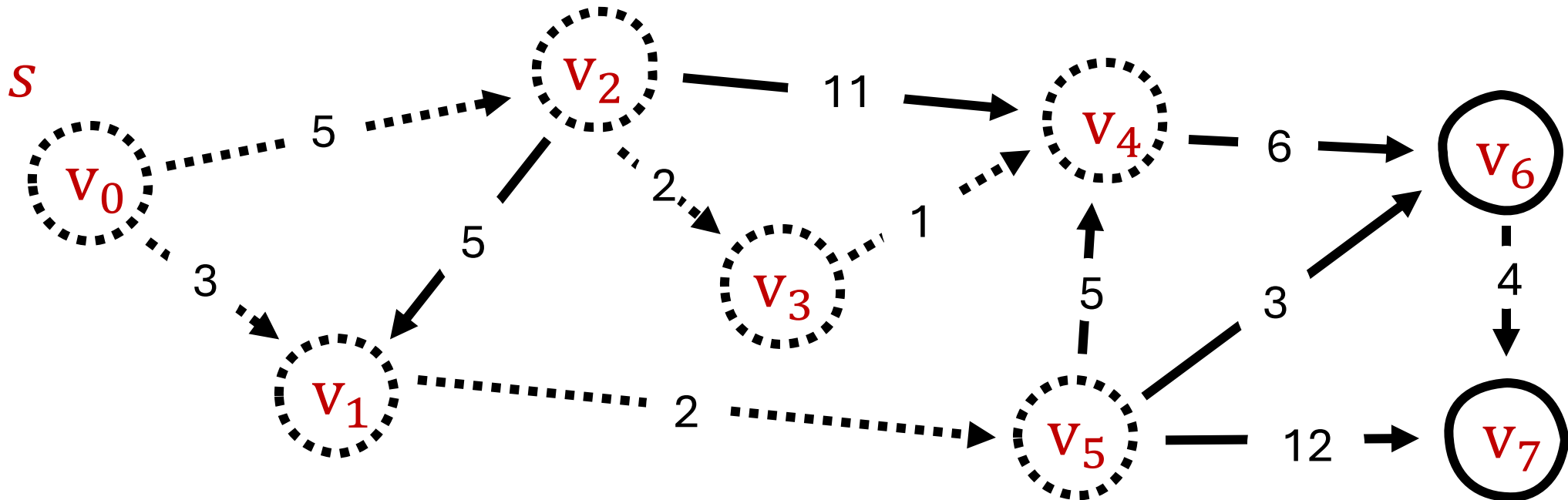Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

Dijkstra's Algorithm

S = {s}
d = [0, 3, 5, 7, 8, 5, 8, ∞]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
     For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
     Select a node v ∉ S with at least one edge from S for which
         d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
     Add v to S and define d(v) = d'(v)
EndWhile
```

Dijkstra's Algorithm

S = {s}

d = [0, 3, 5, 7, 8, 5, 8, 12]

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```
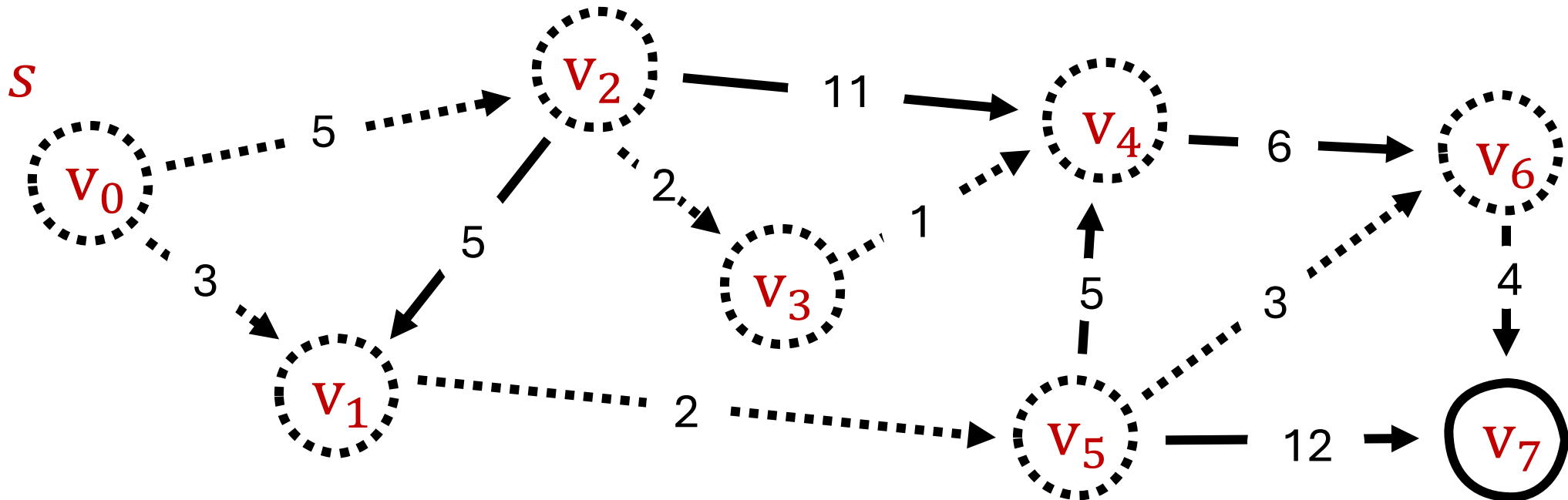
Paths?

Q: How do we return the paths?

Paths?

A: When we add v to S, we can also keep track of what edge was used.

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes   Let P a n length array of Null.
    For each $u \in S$, we store a distance $d(u)$
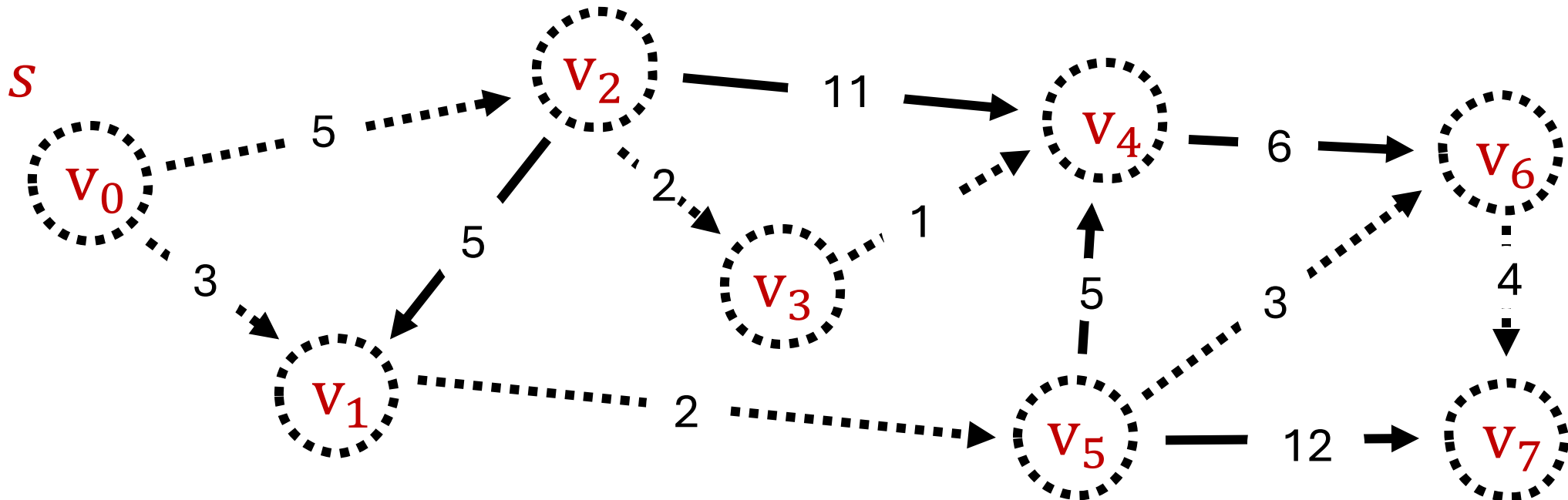Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$ Set P[v] = u such that (u,v) was min.
EndWhile

Paths?

Q: How do we get the shortest path from s to w?

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes   Let P a n length array of Null.
    For each $u \in S$, we store a distance $d(u)$
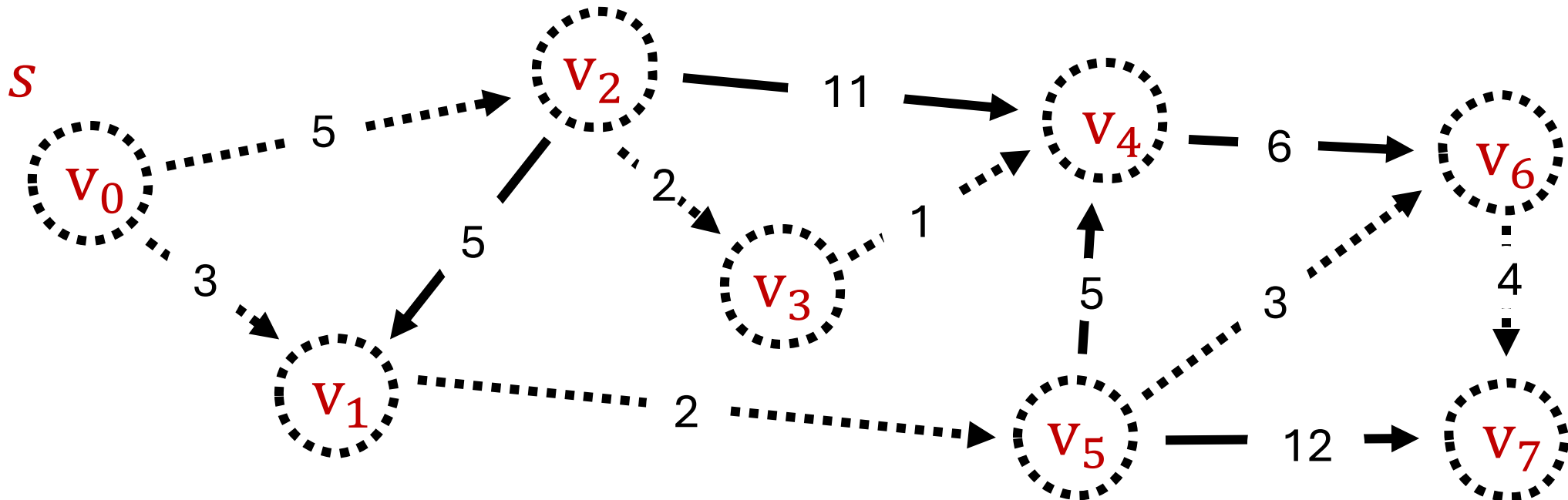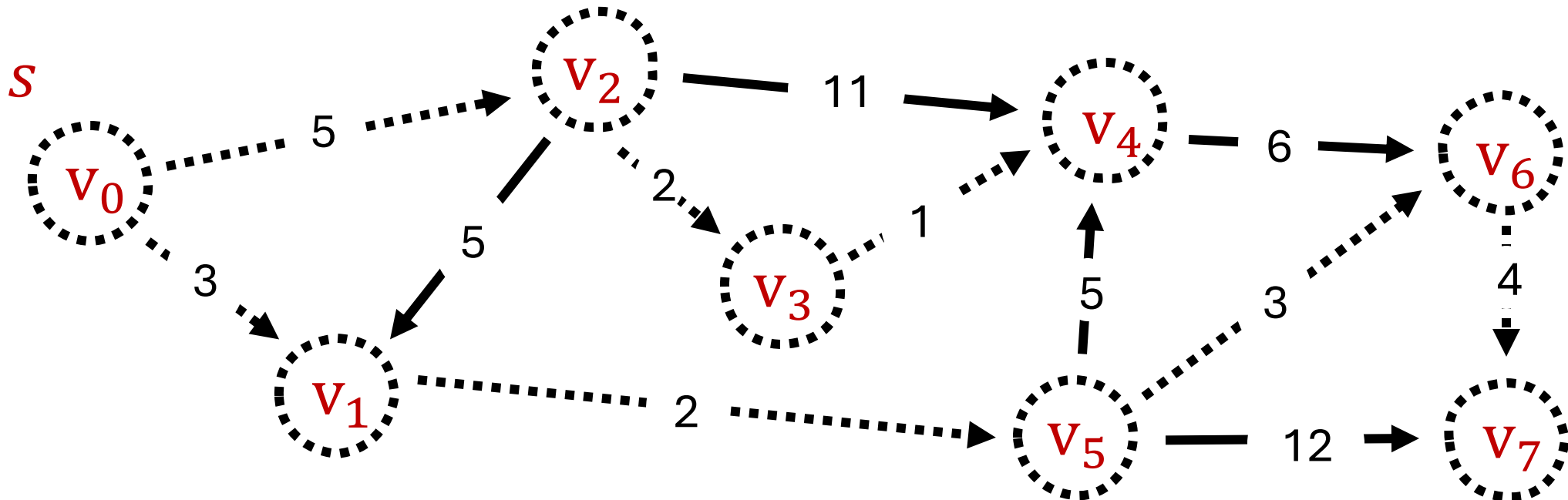Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$ Set P[v] = u such that (u,v) was min.
EndWhile

Paths?

A: We can recursively construct it by looking at taking the edge (P[w],w) and the shortest path from s to P[w].

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes Let P a n length array of Null.

    For each $u \in S$, we store a distance $d(u)$

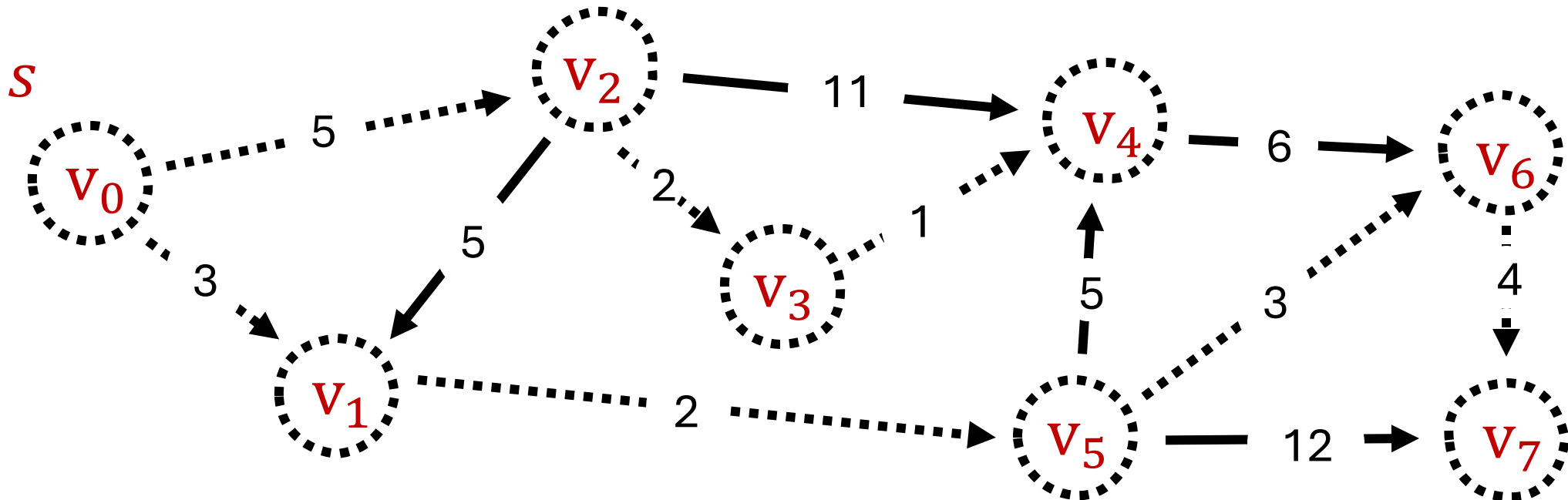Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and define $d(v) = d'(v)$ Set P[v] = u such that (u,v) was min.

EndWhile

Paths

Let P_u be the path found from s to u using these modifications.

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes   Let P a n length array of Null.

    For each $u \in S$, we store a distance $d(u)$
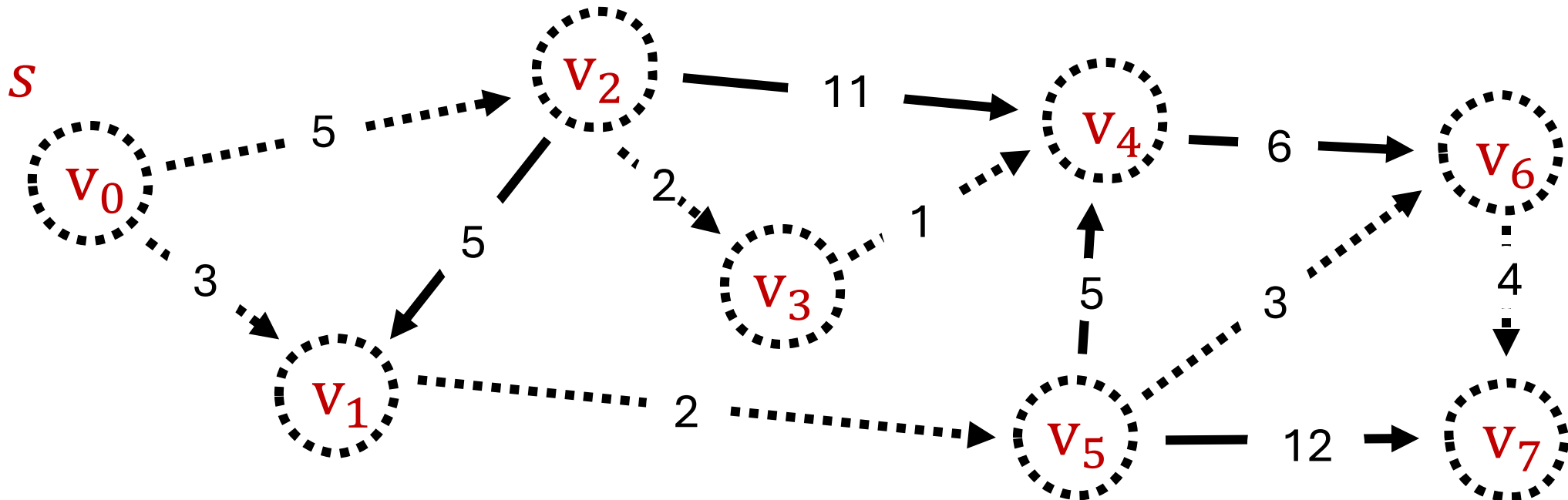
Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and define $d(v) = d'(v)$ Set P[v] = u such that (u,v) was min.

EndWhile

Correctness

Claim: At the start of each loop, P_u is the shortest path from s to u for all $u \in S$.

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
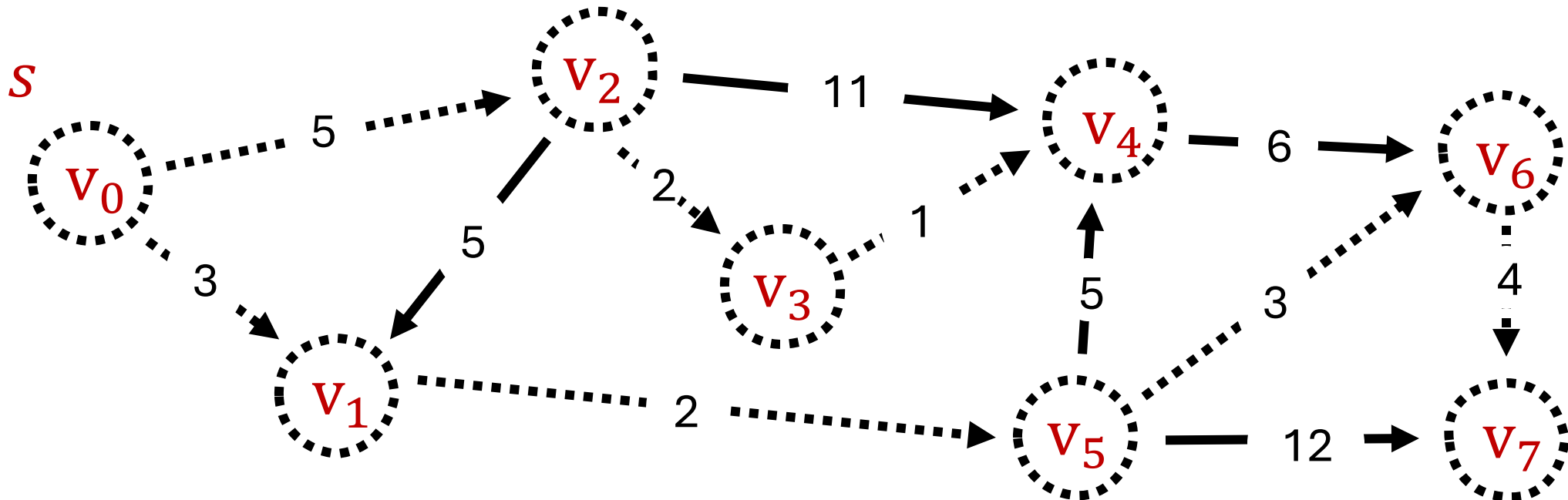Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile
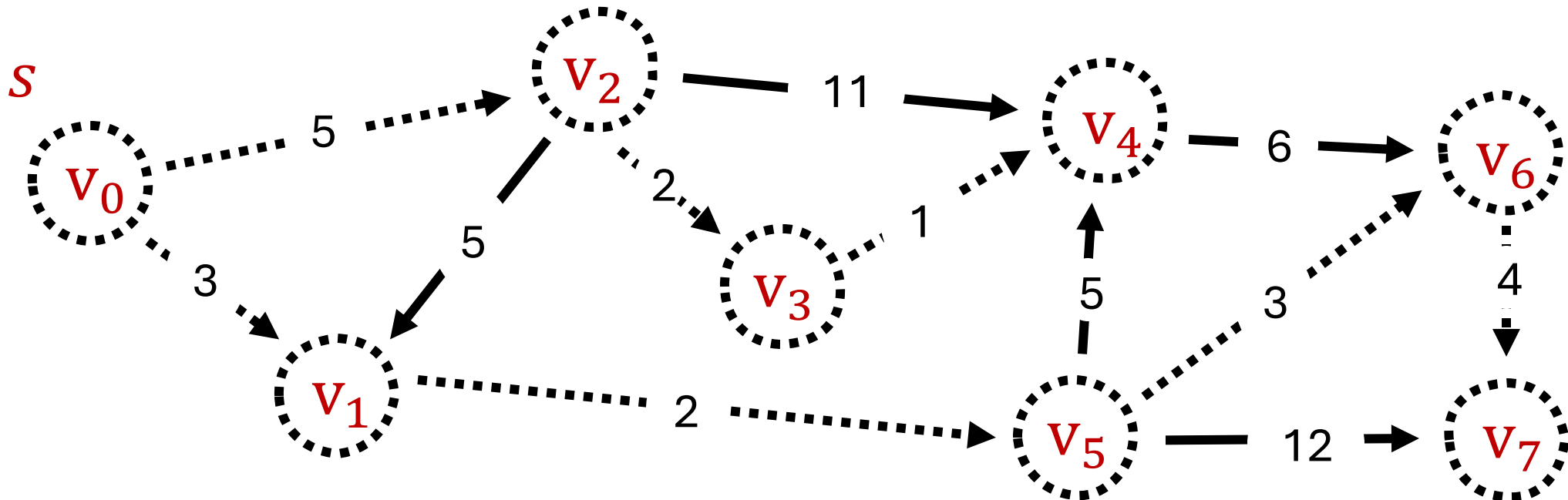
Proof Idea
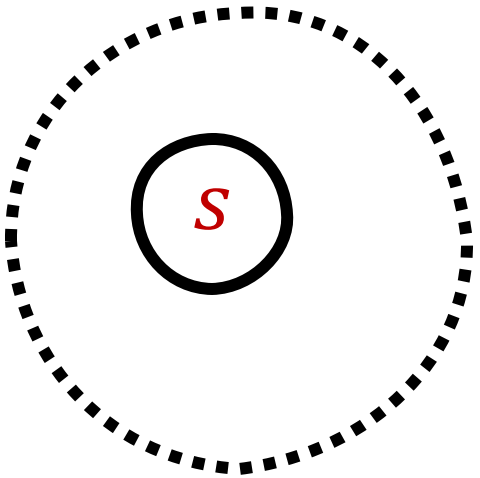
- We proceed with induction on the size of S.
- If $|S| = 1$, then...

Proof Idea

- We proceed with induction on the size of S.
- If $|S| = 1$, then it is true because the shortest path from s to s is empty set.

Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all u $\in$ S.

S

s

Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all u $\in$ S.

S

- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u \in S} d(u) + \ell_e$.

s

u

v

Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all u $\in$ S.

S

- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u \in S} d(u) + \ell_e$.
- Suppose $P_u \cup \{(u,v)\}$ is not the shortest path. Then there exists a path that must leave S via some edge (x,y).
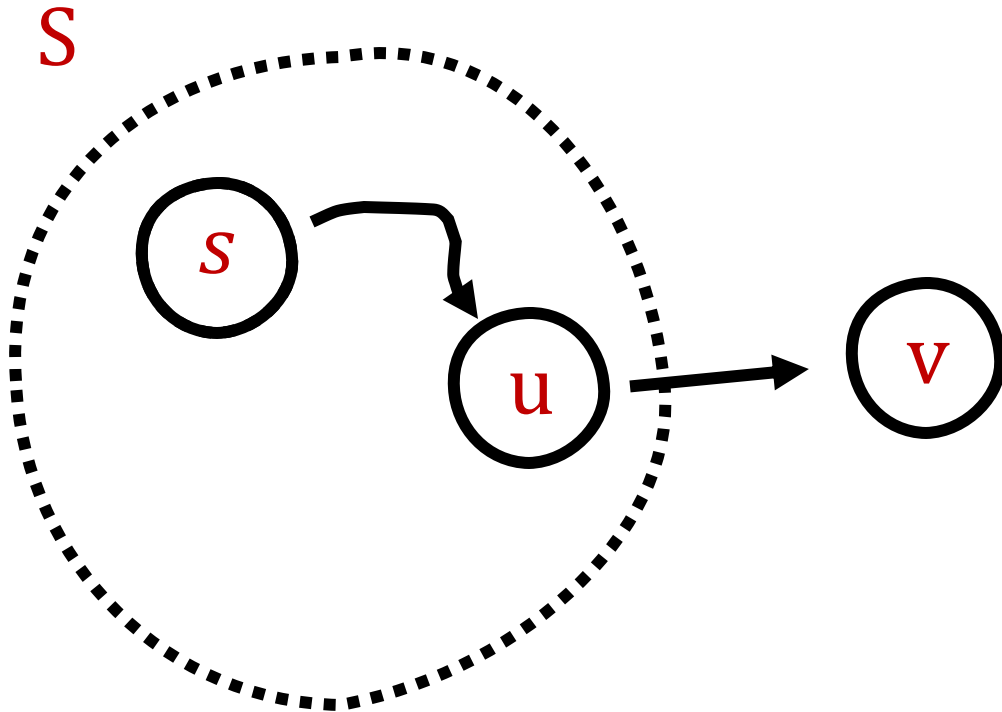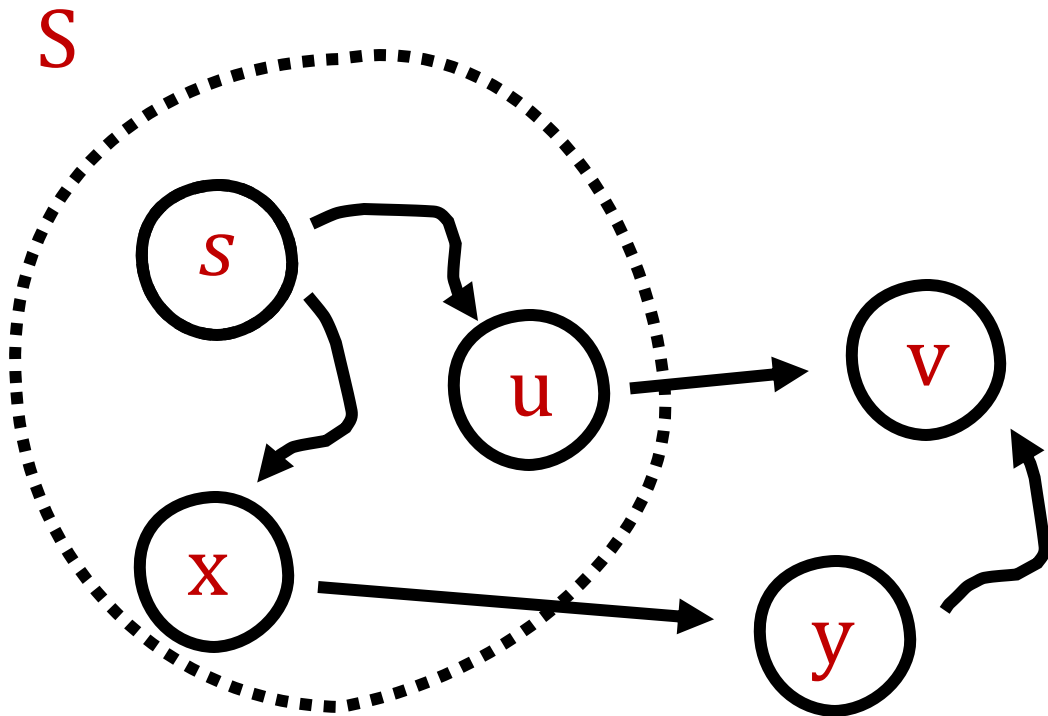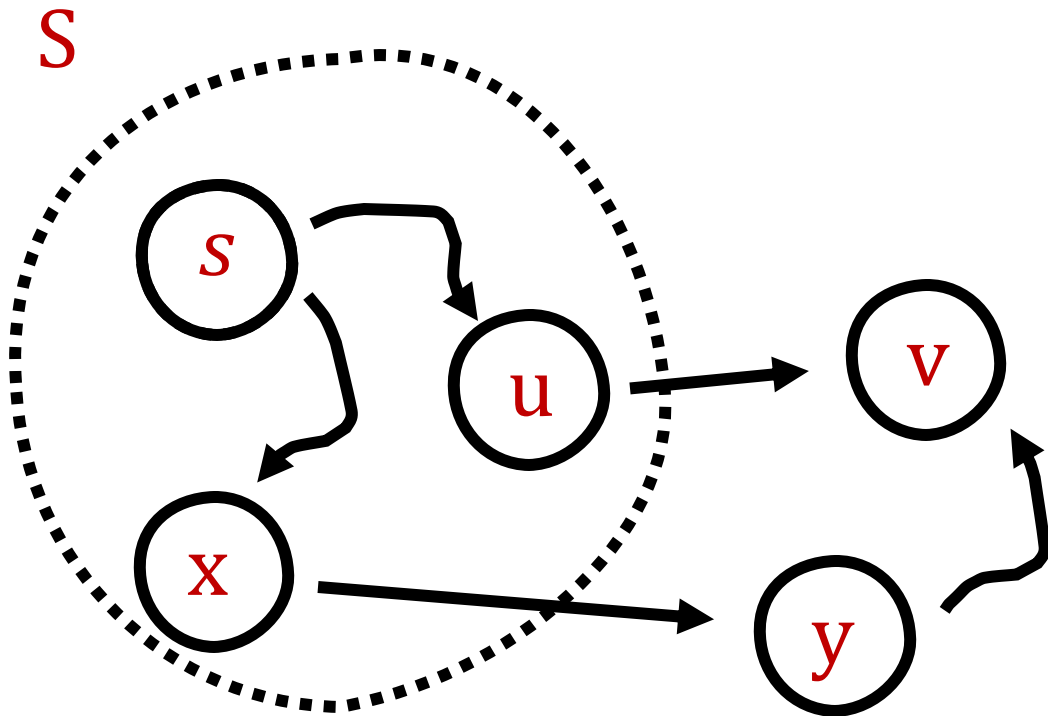
Proof Idea

- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all u ∈ S.

S

- Let v be the k+1 vertex added to S and let (u,v) be the edge minimized $min_{e=(u,v):u \in S} d(u) + \ell_e$.
- Suppose $P_u \cup \{(u,v)\}$ is not the shortest path. Then there exists a path that must leave S via some edge (x,y).
- However, the algo picked v and so the path from s to y must be just as long. ><

Q: What about negative edge weights?

A: If you have negative edge weights then you can have negative cycles and the minimum distance becomes "weird or hard to find."
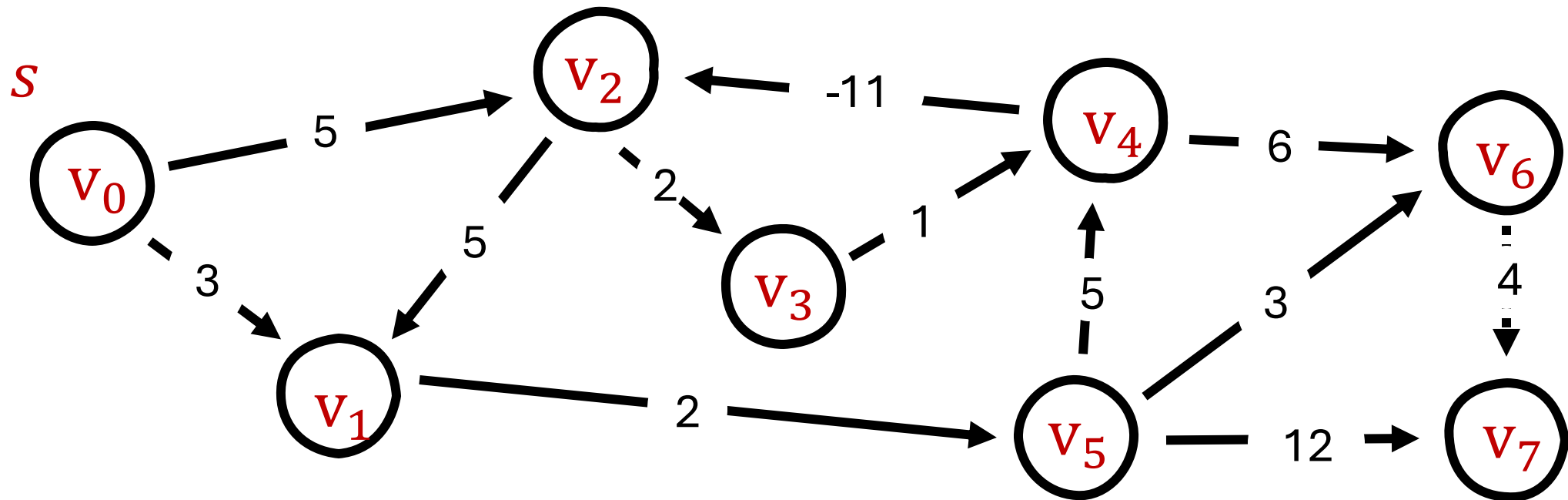
Q: What about negative edge weights?

A: If you have negative edge weights then you can have negative cycles, and the minimum distance becomes "weird or hard to find."
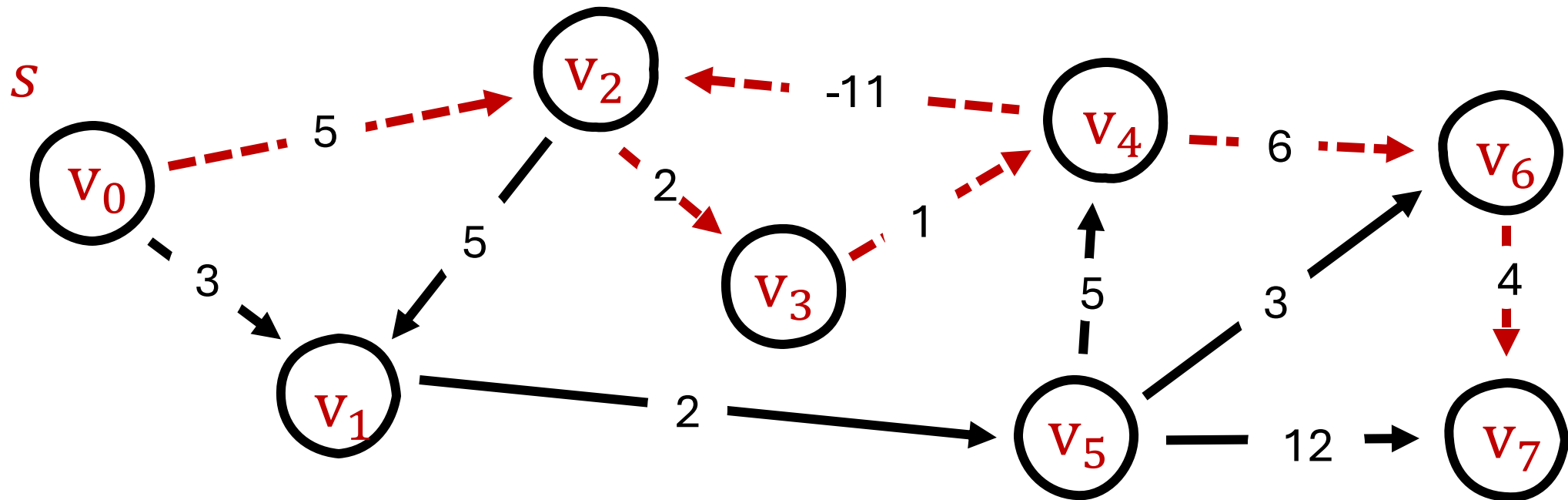
Dijkstra's Algorithm

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a dist

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

$$d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e \text{ is as small as possible}$$

    Add v to S and define $d(v) = d'(v)$

EndWhile

Q: How many times might we compute this?

y

Dijkstra's Algorithm

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a dist

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

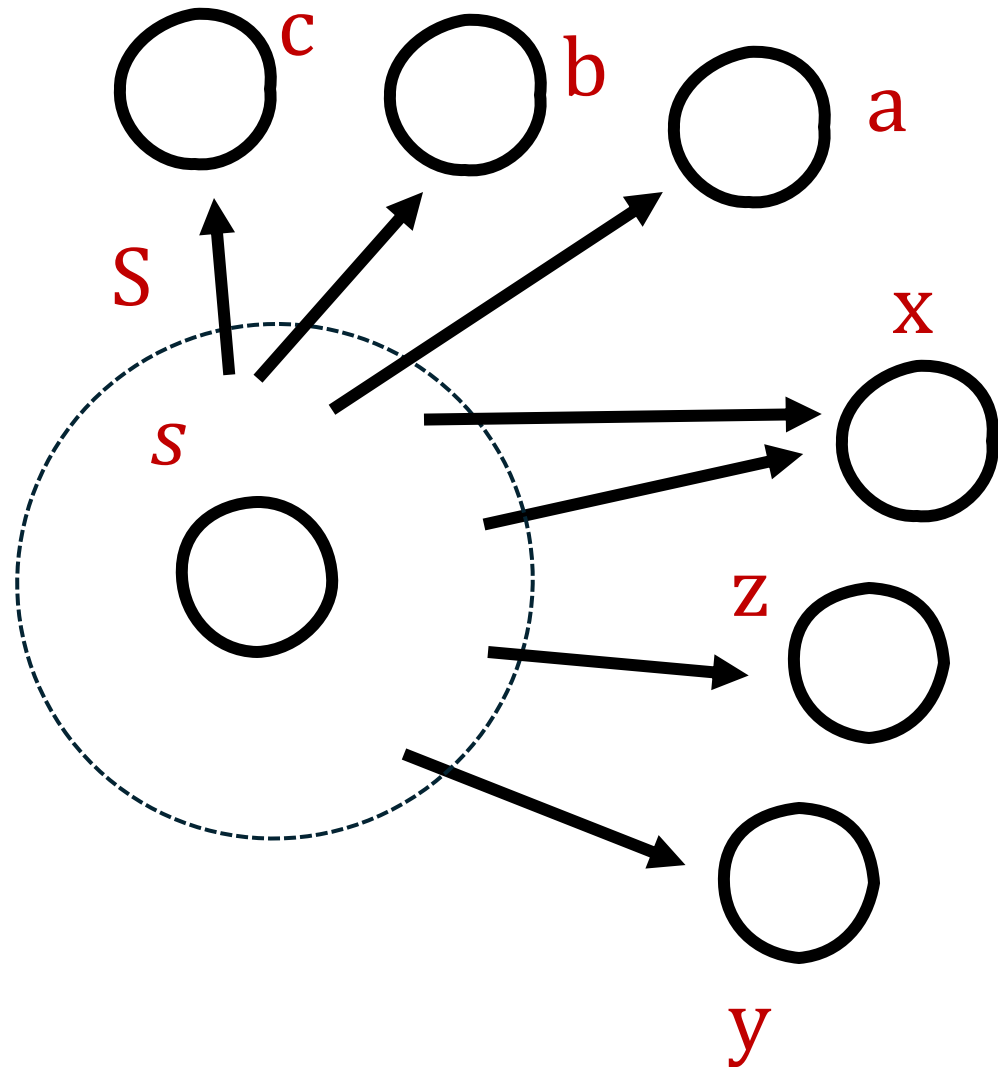   Select a node $v \notin S$ with at least one edge from S for which

    $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and define $d(v) = d'(v)$

EndWhile

A: We may do it several times because of large frontier.

y

Dijkstra's Algorithm Idea

Observation: A vertex could be in the frontier a lot!

Idea: Don't recompute every time!

Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

Dijkstra's Algorithm Optimization

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$ and $d(\text{w}) = \ell_{(s,w)}$ for everyone neighbors of s

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and for each neighbor w, check if $d(v) + \ell_{(v,w)}$ is smaller than current guess and update!

EndWhile

Dijkstra's Algorithm Optimization

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a dist[...]

Initially $S = \{s\}$ and $d(s) = 0$ and $d(w)$[...]
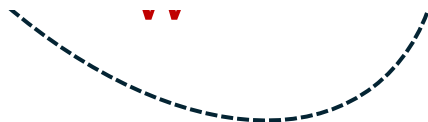
Q: How long to find the min?

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

    $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and for each neighbor w, check if $d(v) + \ell_{(v,w)}$ is smaller than current guess and update!

EndWhile

Dijkstra's Algorithm Optimization

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a dist

Initially $S = \{s\}$ and $d(s) = 0$ and $d(w)$

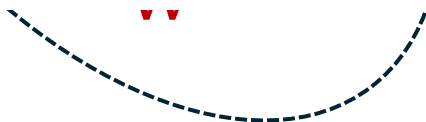While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and for each neighbor w, check if $d(v) + \ell_{(v,w)}$
            is smaller than current guess and update!

EndWhile

A: Still loop over everything in frontier (O(n)).

Priority Queue

- A priority queue is a data structure that supports the following operations:
  - `Insert(PQ, v, p)`: Inserts element v into PQ with priority p.
  - `Pop-Min(PQ)`: Returns element with minimum priority and removes it from PQ.
  - `Decrease-Key(PQ,v,p)`: Updates the priority of v in PQ to be p.
- **Recall**: You can implement this using a Heap data structure.

Input: graph G = **(V,E)**, lengths **L,** and start **s**

- Initialize Priority Queue **PQ,** arrays **P** and **d**
- For Each **v** in **V**:
  - Let **P**[**v**] = Null and **d**[**v**] = INFINITY
  - Insert **v** into **PQ** with priority **d**[**v**]
- Let **d**[**s**] = 0
- While **PQ** is not empty:
  - **u** = Pop-Min(**PQ**)
  - For each edge **e** = **(u,v)** in **E** leaving **u**:
    - If **d**[**v**] > **d**[**u**] + **L**[**e**]:
      - Decrease-Key(**PQ, v, d**[**u**] + **L**[**e**])
      - **d**[**v**] = **d**[**u**] + **L**[**e**]
      - **P**[**v**] = **u**

Dijkstra's Algorithm Runtime

- The runtime depends on the priority queue used.
- Using a priority queue, Dijkstra's Algorithm can be implemented on a graph with n nodes and m edges to run in O(m) time plus the time for n Pop-Min and m Decrease-Priority operations.
  - Using the heap-based priority queue, each priority queue operation can be implemented in log(n) time giving a total runtime of O(mlog(n)).

Input: graph G = (**V,E**), lengths **L,** and start **s**

- Initialize Priority Queue **PQ,** arrays **P** and **d**
- For Each **v** in **V:**
  - Let **P**[**v**] = Null and **d**[**v**] = INFINITY
  - Insert **v** into **PQ** with priority **d**[**v**]
- Let **d**[**s**] = 0
- While **PQ** is not empty:
  - **u** = Pop-Min(**PQ**)
  - For each edge **e** = (**u,v**) in **E** leaving **u:**
    - If **d**[**v**] > **d**[**u**] + **L**[**e**]:
      - Decrease-Key(**PQ, v, d**[**u**] + **L**[**e**])
      - **d**[**v**] = **d**[**u**] + **L**[**e**]
      - **P**[**v**] = **u**

Even Better?

Breaking the Sorting Barrier for Directed Single-Source Shortest Paths

Ran Duan [*]   Jiayi Mao [*]   Xiao Mao [†]   Xinkai Shu [‡]   Longhui Yin [*]

July 31, 2025

Abstract

We give a deterministic $O(m \log^{2/3} n)$-time algorithm for single-source shortest paths (SSSP) on directed graphs with real non-negative edge weights in the comparison-addition model. This is the first result to break the $O(m + n \log n)$ time bound of Dijkstra's algorithm on sparse graphs, showing that Dijkstra's algorithm is not optimal for SSSP.