CSE 331: Algorithms & Complexity

"Dijkstra's Algorithm"

Prof. Charlie Anne Carlson (She/Her)

Lecture 18

Wednesday October 15, 2025

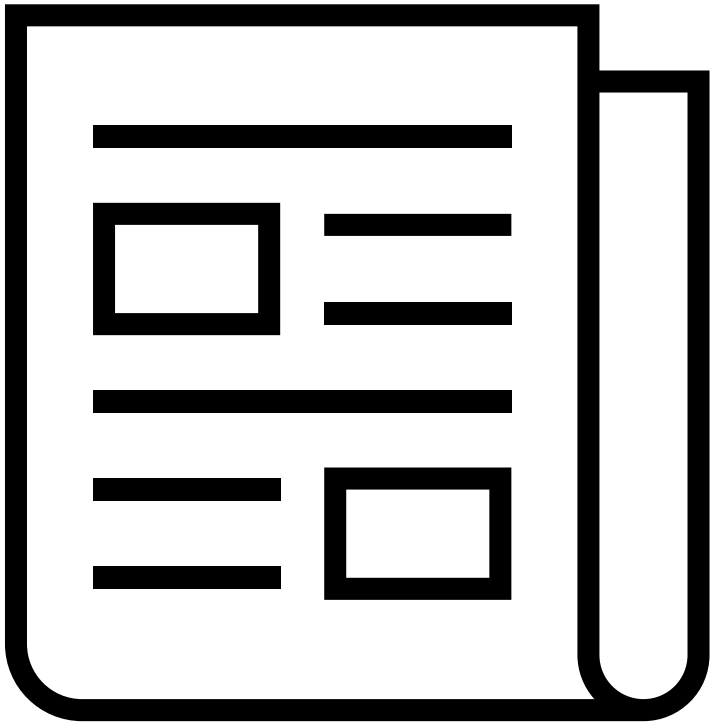
University at Buffalo®

Schedule

1. Course Updates
2. Dijkstra's Algorithm
3. Proof of Correctness
4. Min Spanning Trees
5. Kruskal's Algorithm
6. Prim's Algorithm

Course Updates

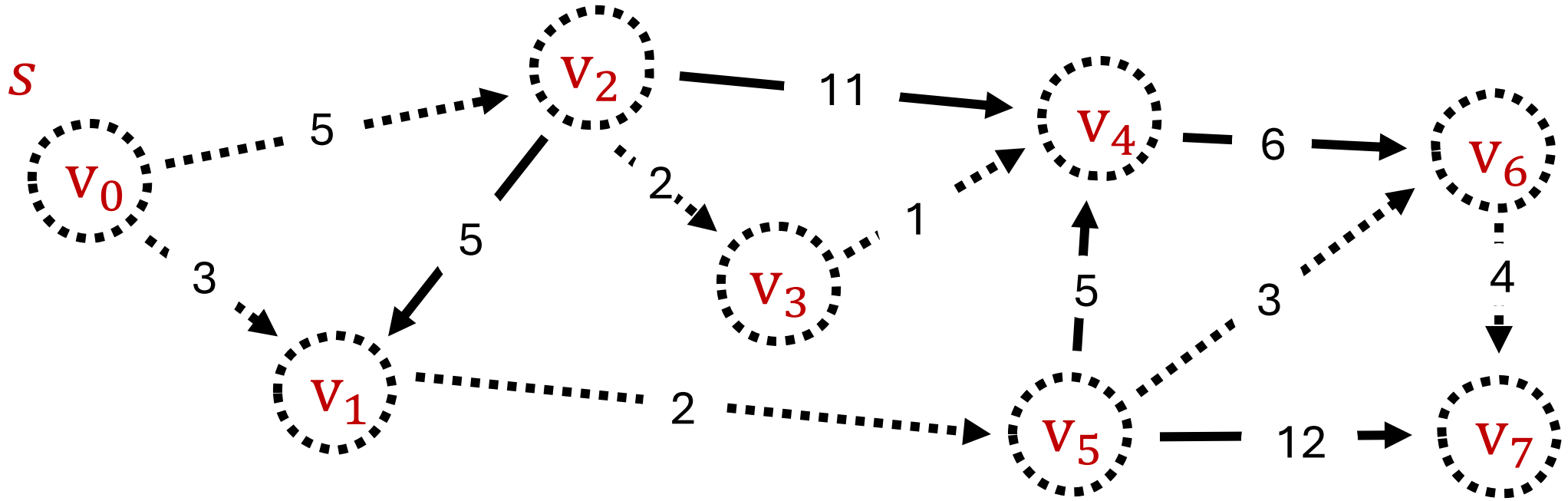
- Everything Besides Midterm Graded
  - Midterm Grades this week
- HW 4 Out
  - Due Next Week!
- Group Project
  - First Problems Oct 31st



Paths

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

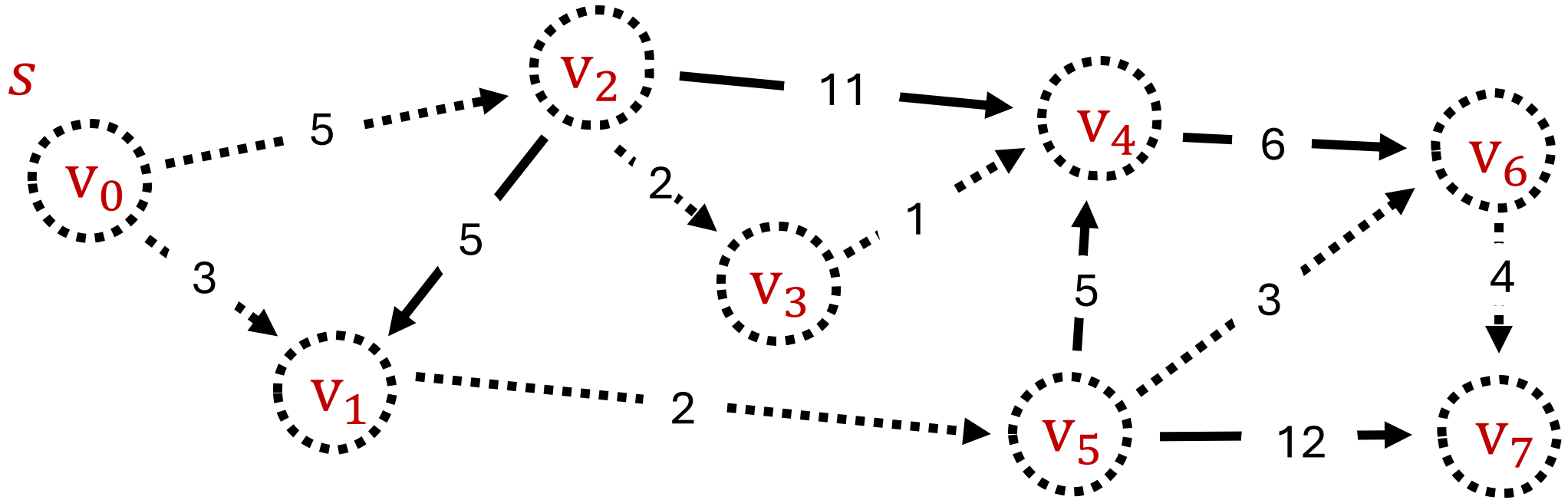
```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes   Let P a n length array of Null.
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is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

```
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
```

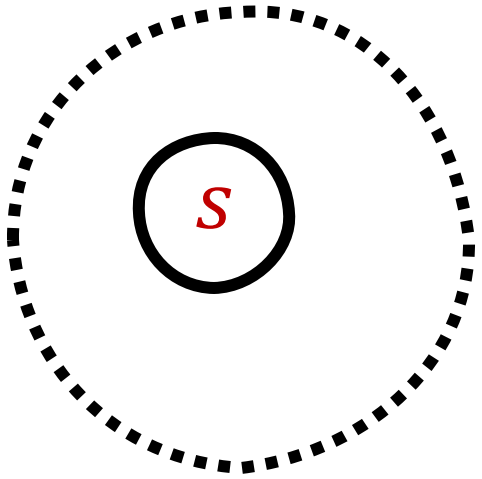


Proof Idea

- We proceed with induction on the size of S.
- If $|S| = 1$, then…

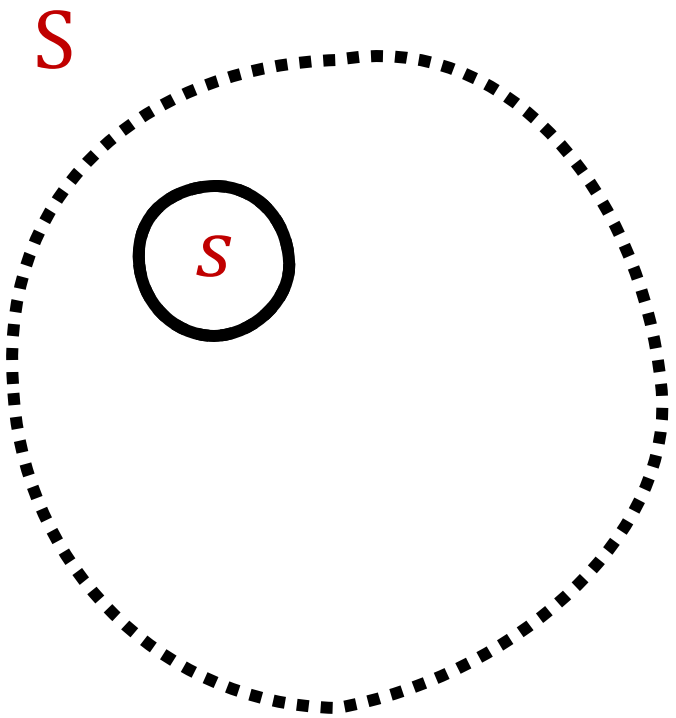
Proof Idea

- 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.



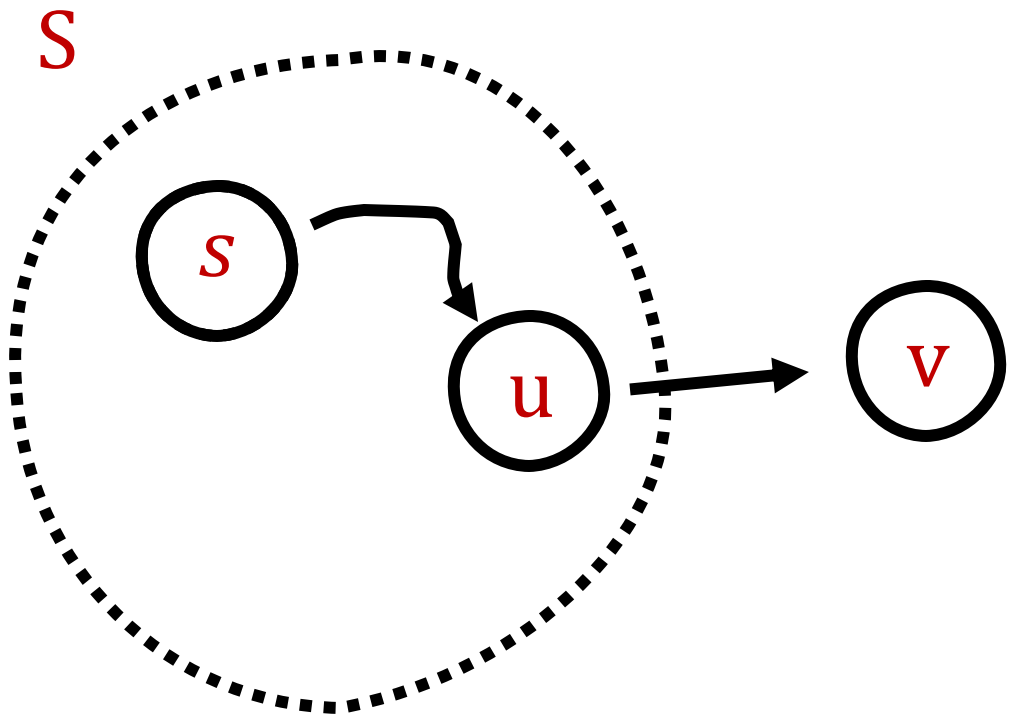
Proof Idea

- 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 $u \in S$.



Proof Idea

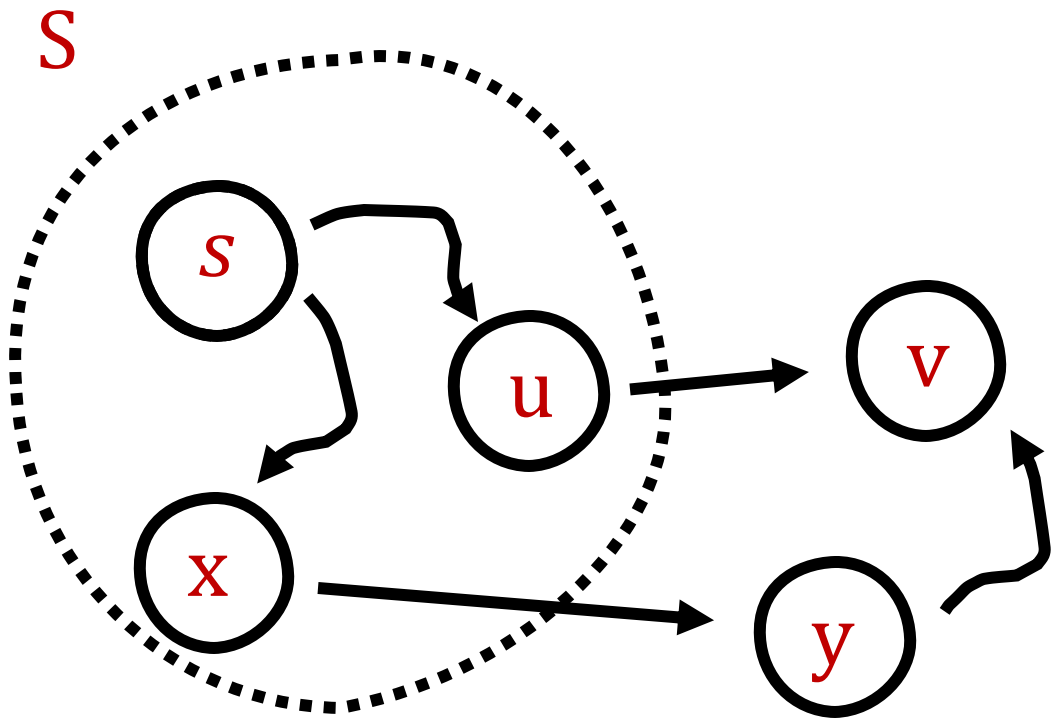
- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all $u \in S$.



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

Proof Idea

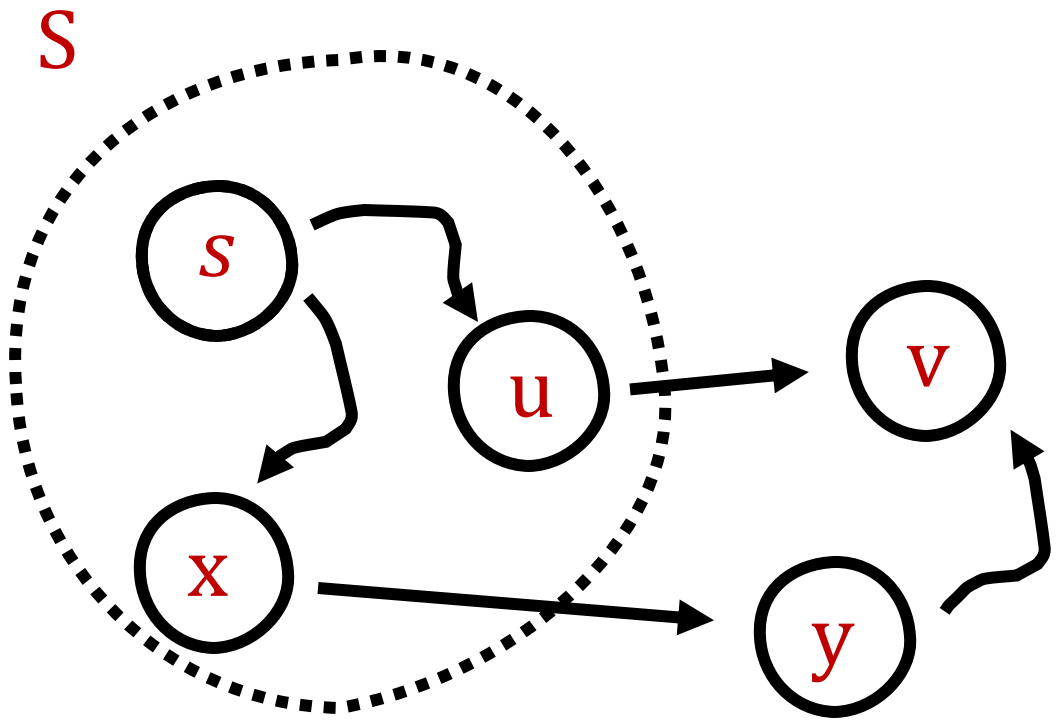
- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all $u \in S$.



- 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

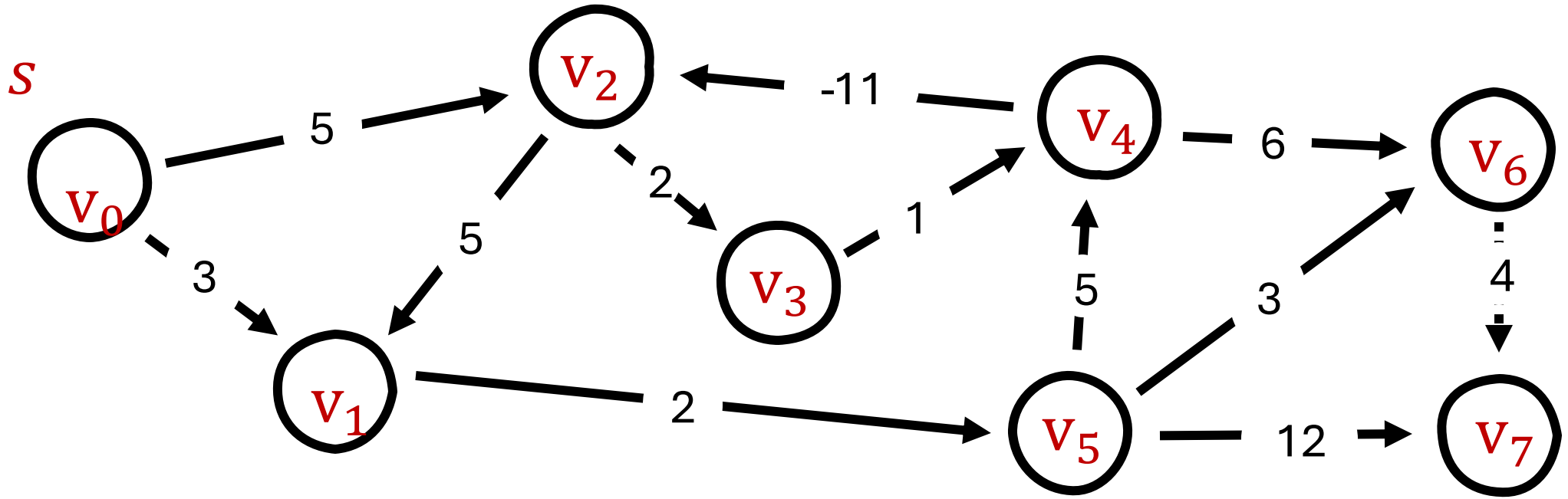
- We proceed with induction on the size of S.
- Suppose for all S such that $|S| \leq k$, P_u is the shortest path from s to u for all $u \in S$.



- 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. ><

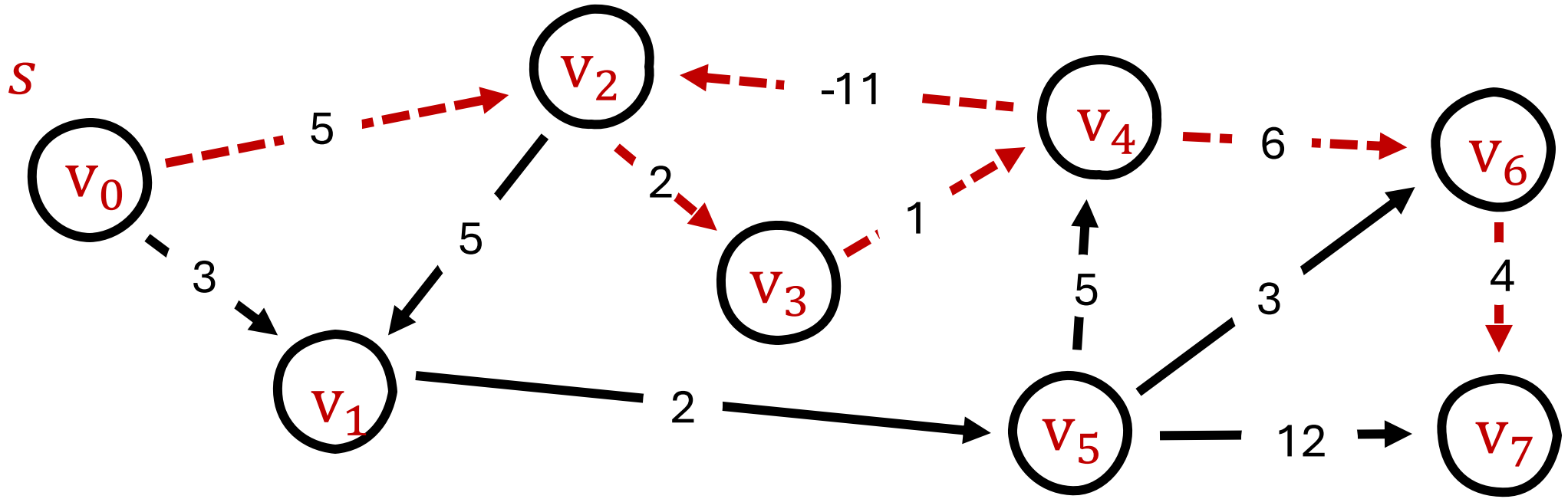
Q: What about negative edge weights?

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 $S \neq V$

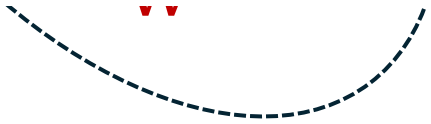
Select a node $v \notin S$ with at least one edge from S for which

$d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ 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 $S \neq V$

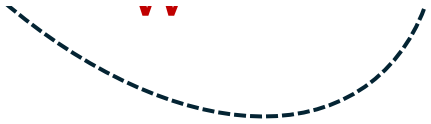
Select a node $v \notin S$ with at least one edge from S for which

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

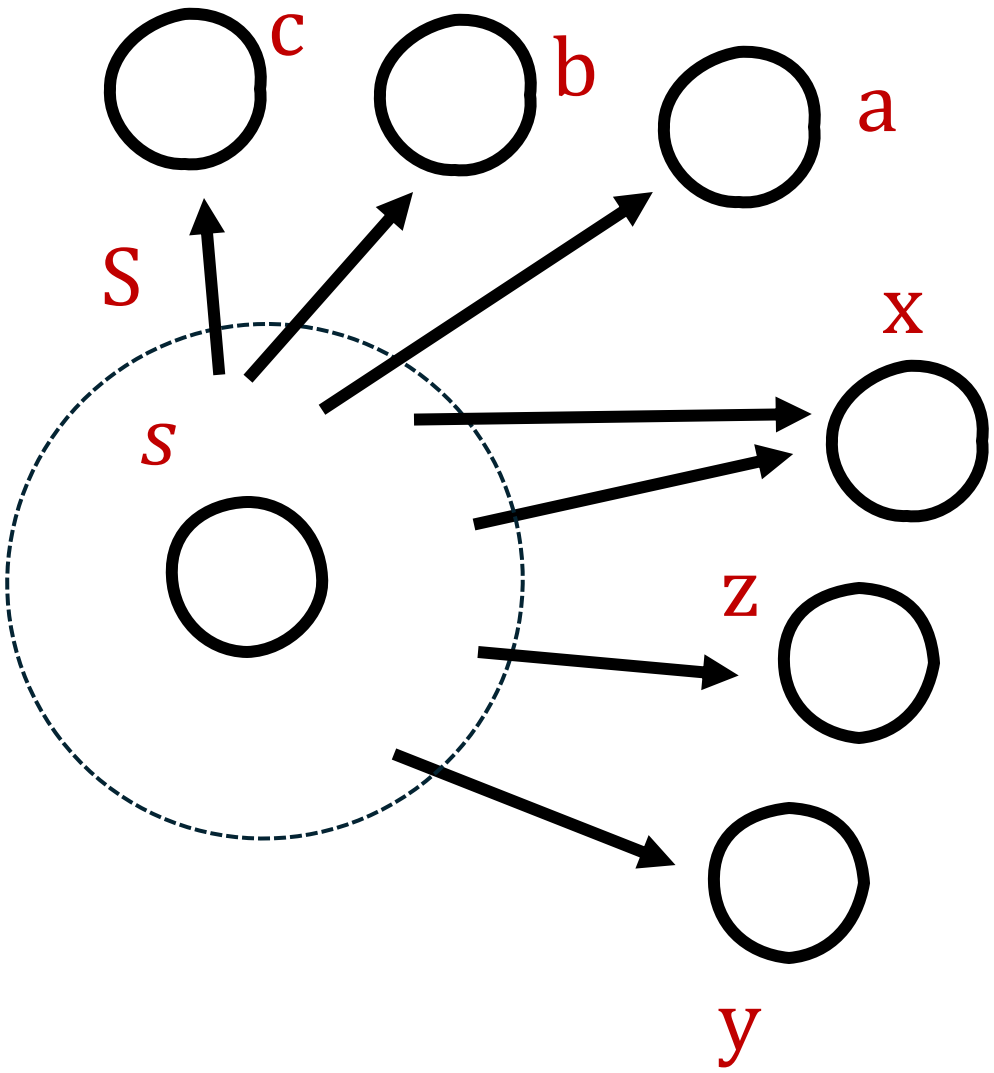
Add v to S and define $d(v) = d'(v)$

EndWhile

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



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 $d(w) = \ell_{(s,w)}$ 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 $d(w)$

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

Q: How long to find the min?

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

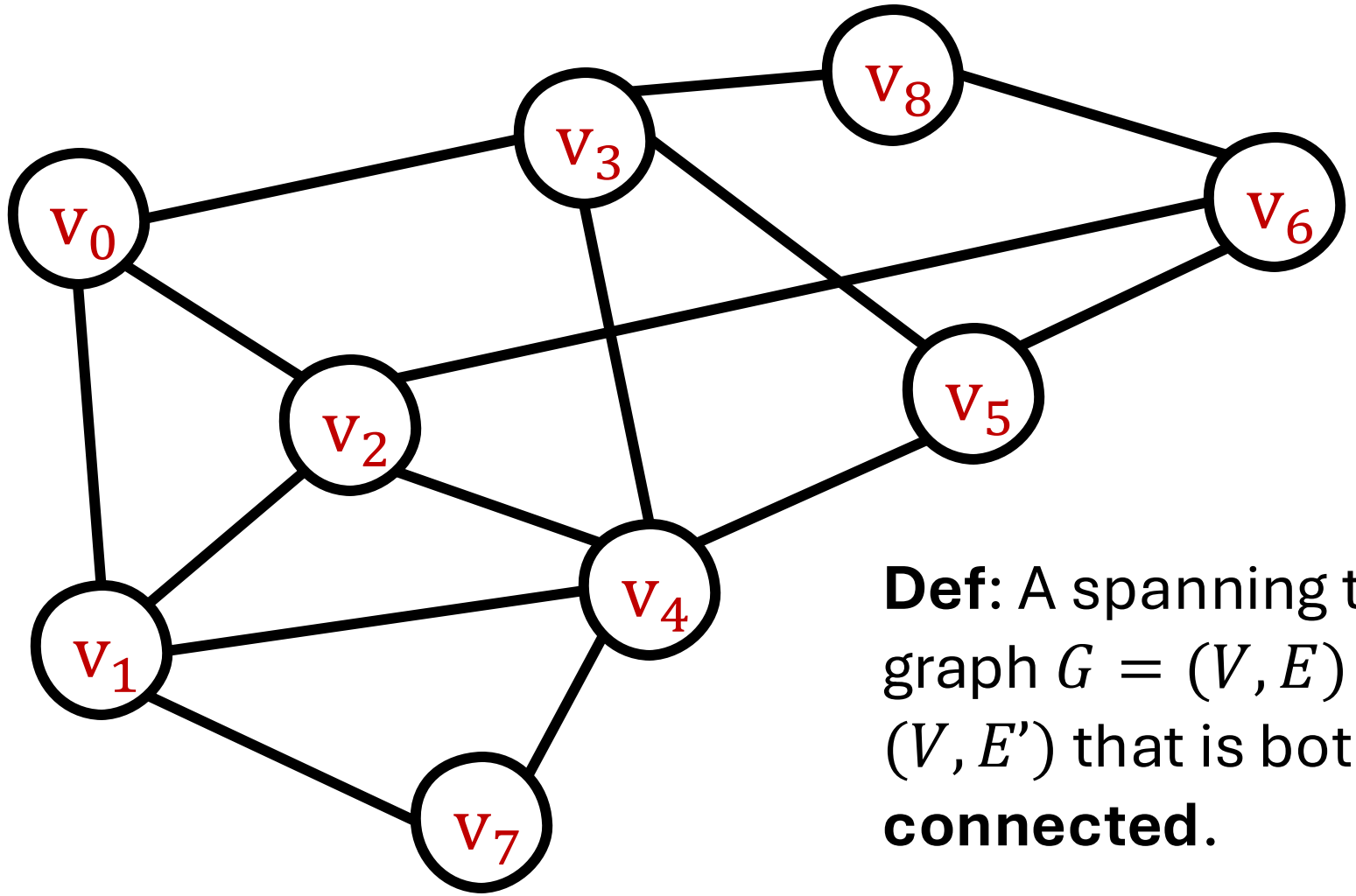
Ran Duan * Jiayi Mao * Xiao Mao † Xinkai Shu ‡ Longhui Yin *

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.

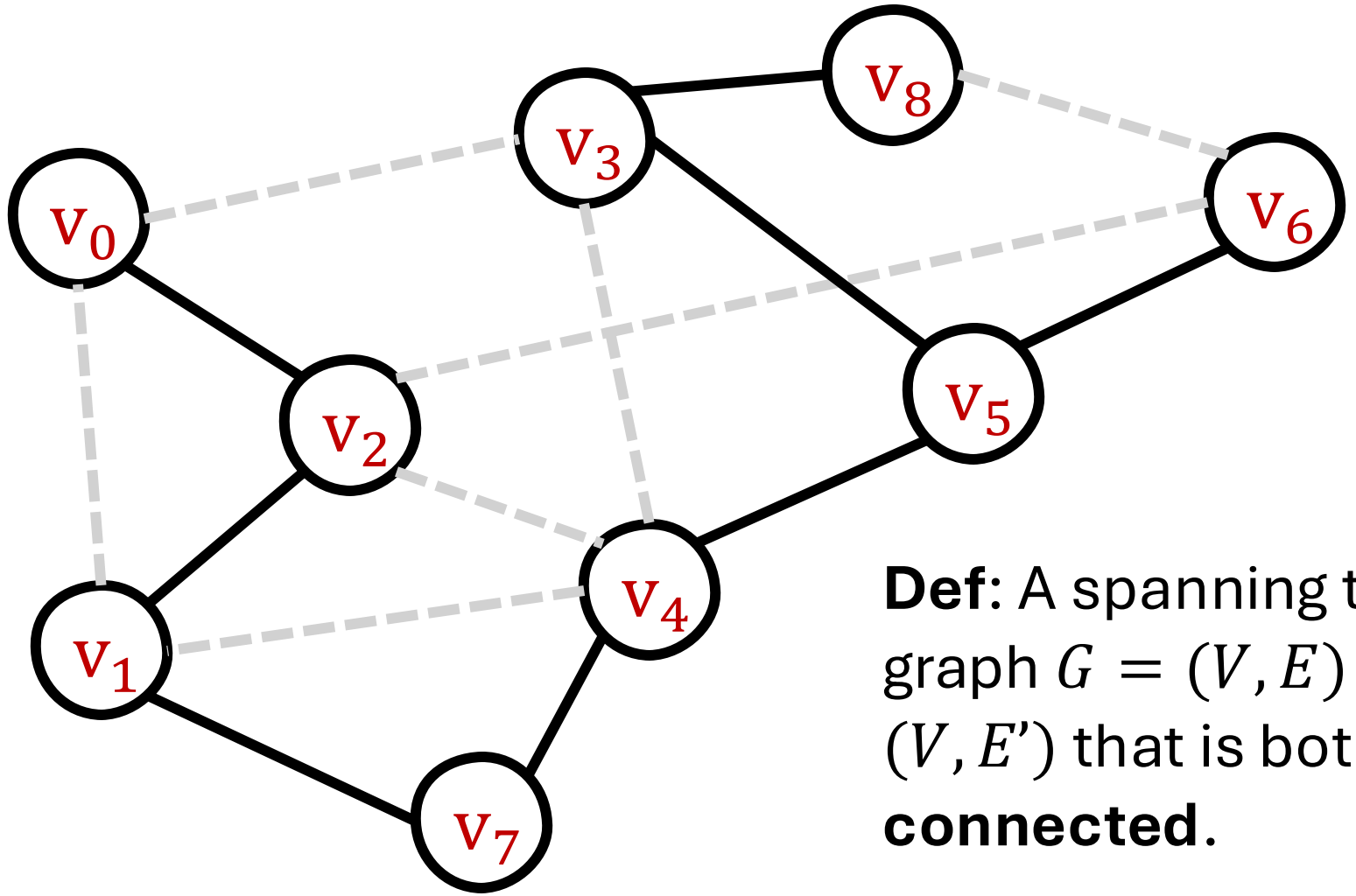
Spanning Trees



Super Graph

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

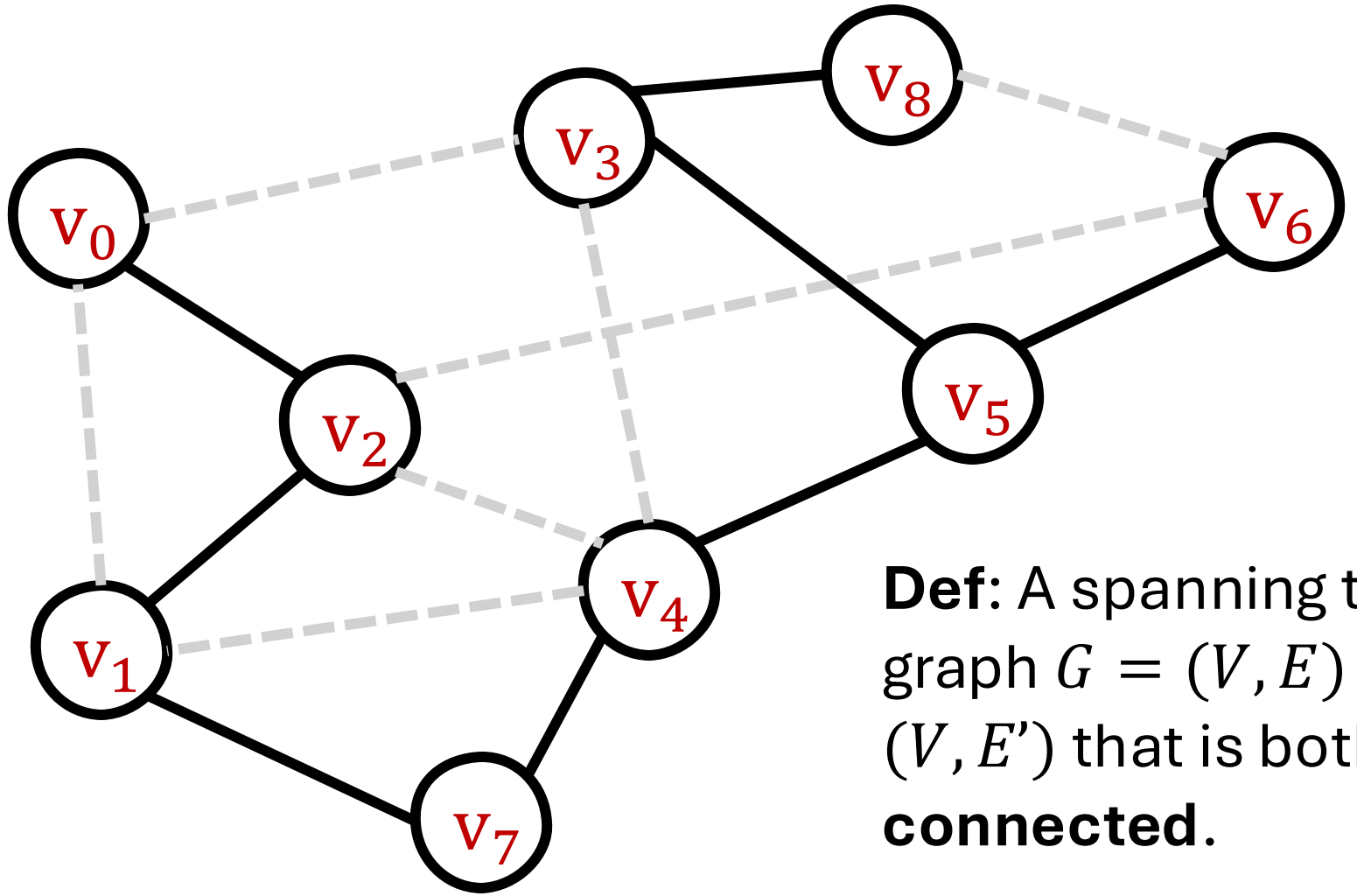
Spanning Trees



Spanning Tree

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

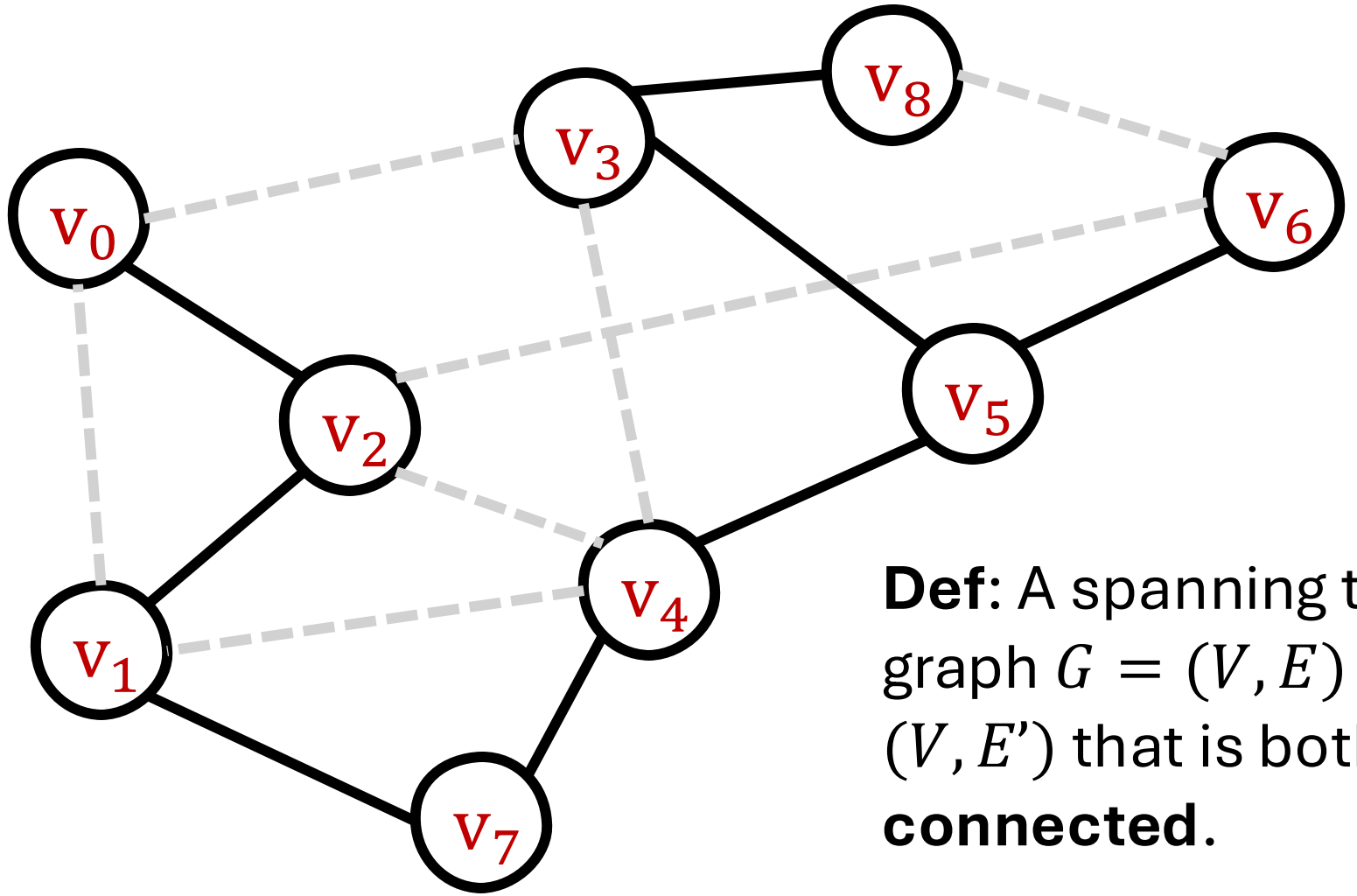
Q: How many edges does a spanning tree have?



Spanning Tree

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

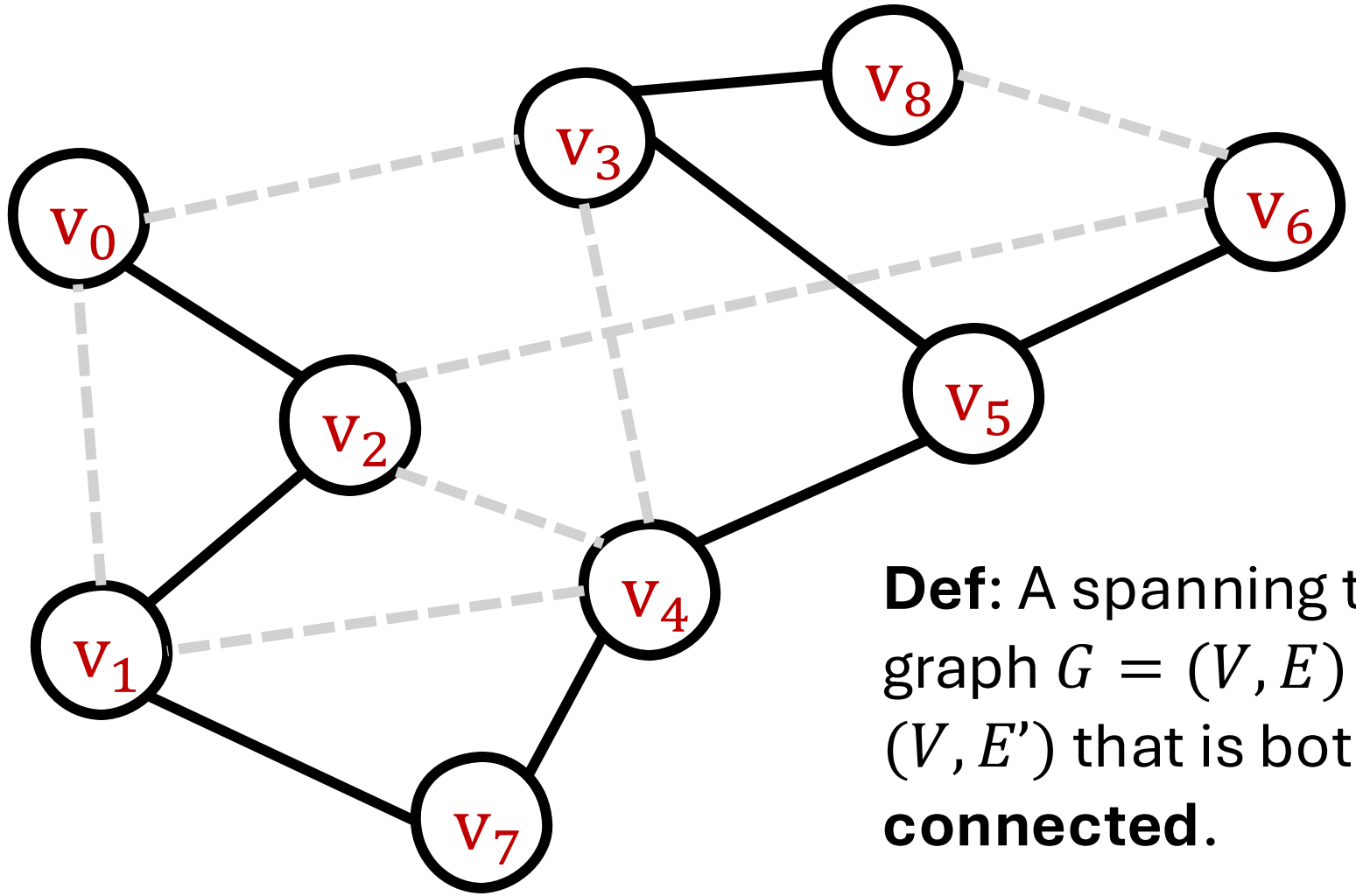
A: It has |V| - 1 because it is a tree!



Spanning Tree

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

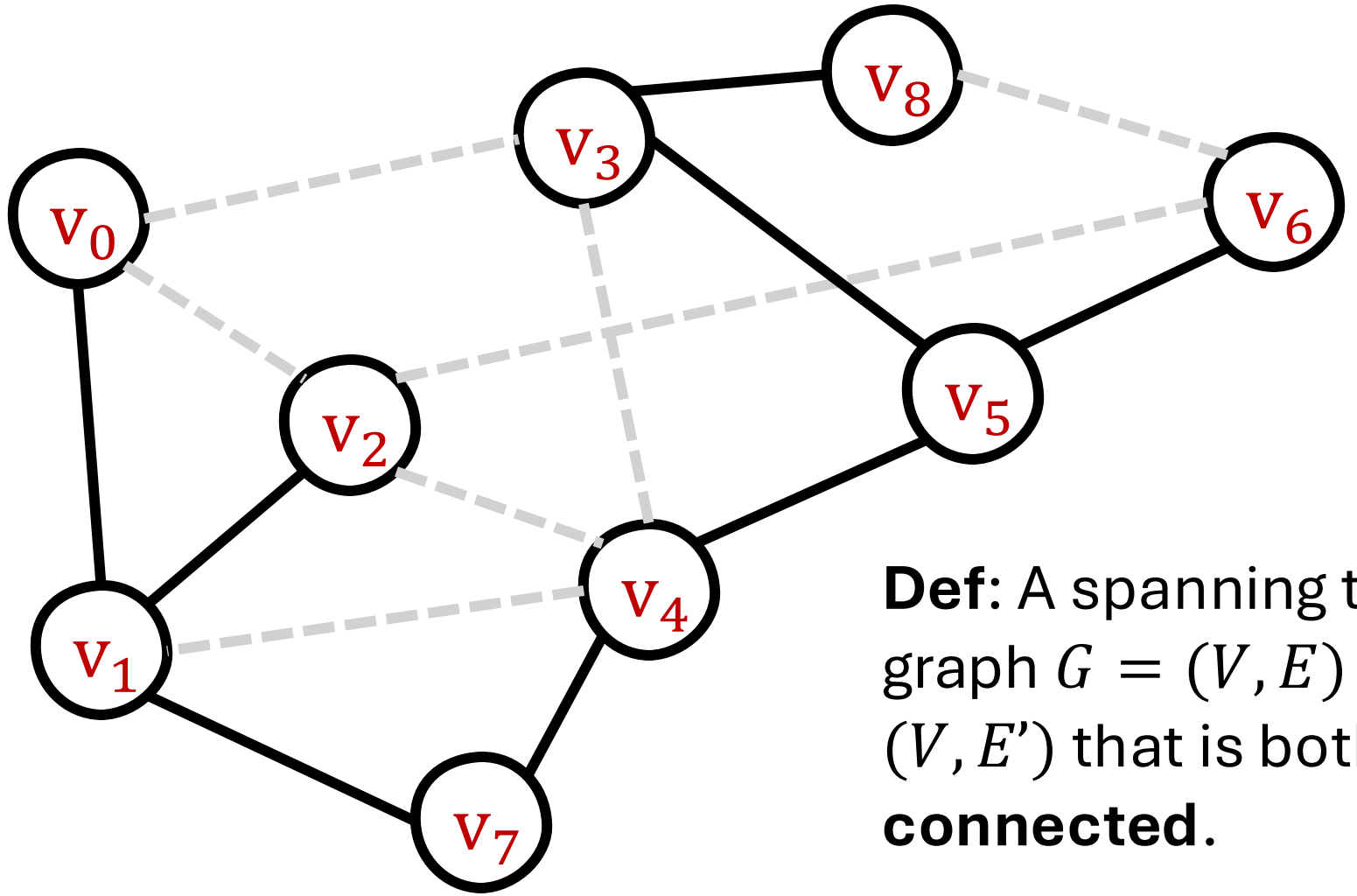
Q: Can you have more than one spanning tree?



Spanning Tree

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

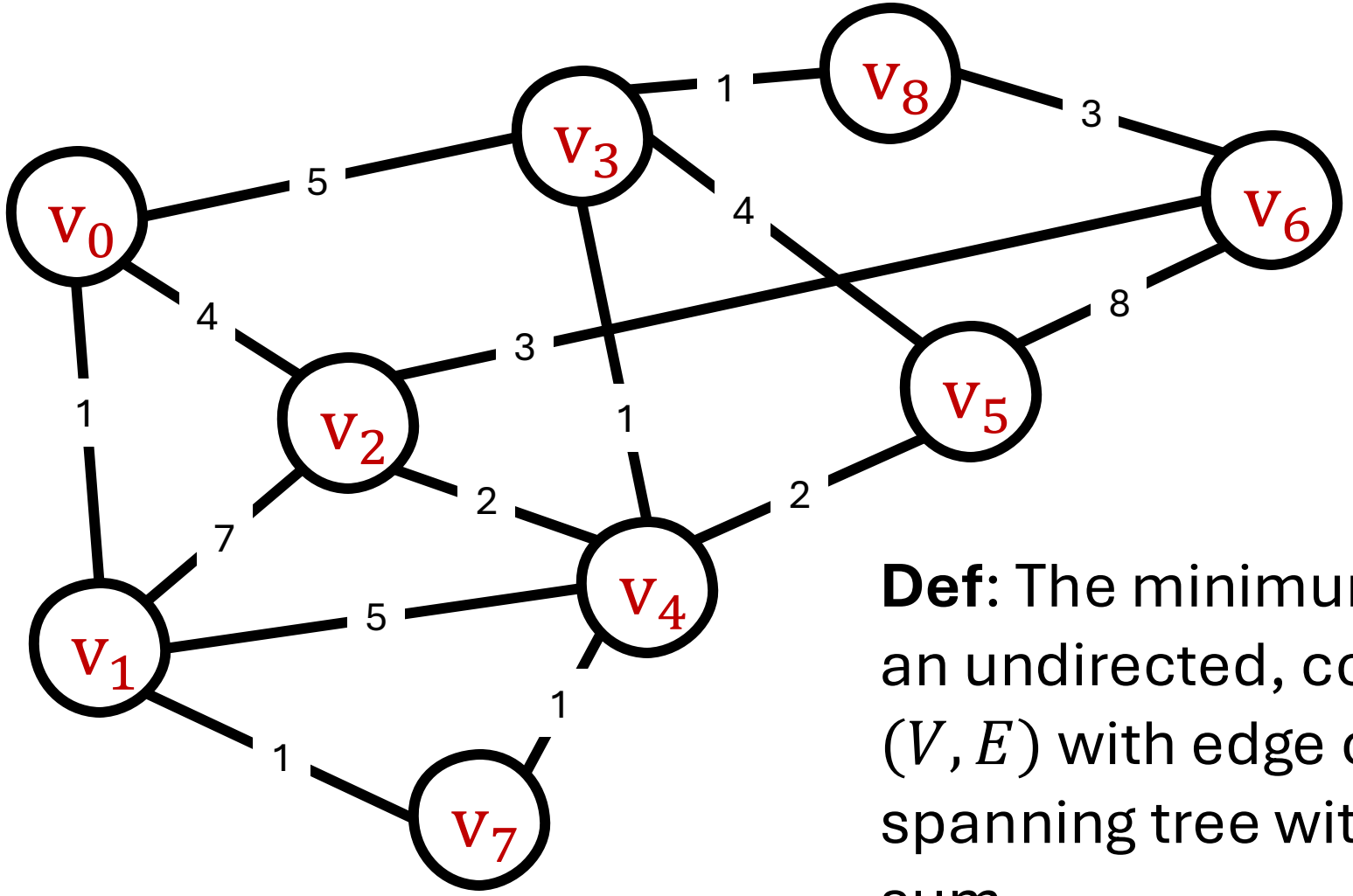
A: Can you have more than one spanning tree?



Spanning Tree

Def: A spanning tree of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ that is both **acyclic** and **connected**.

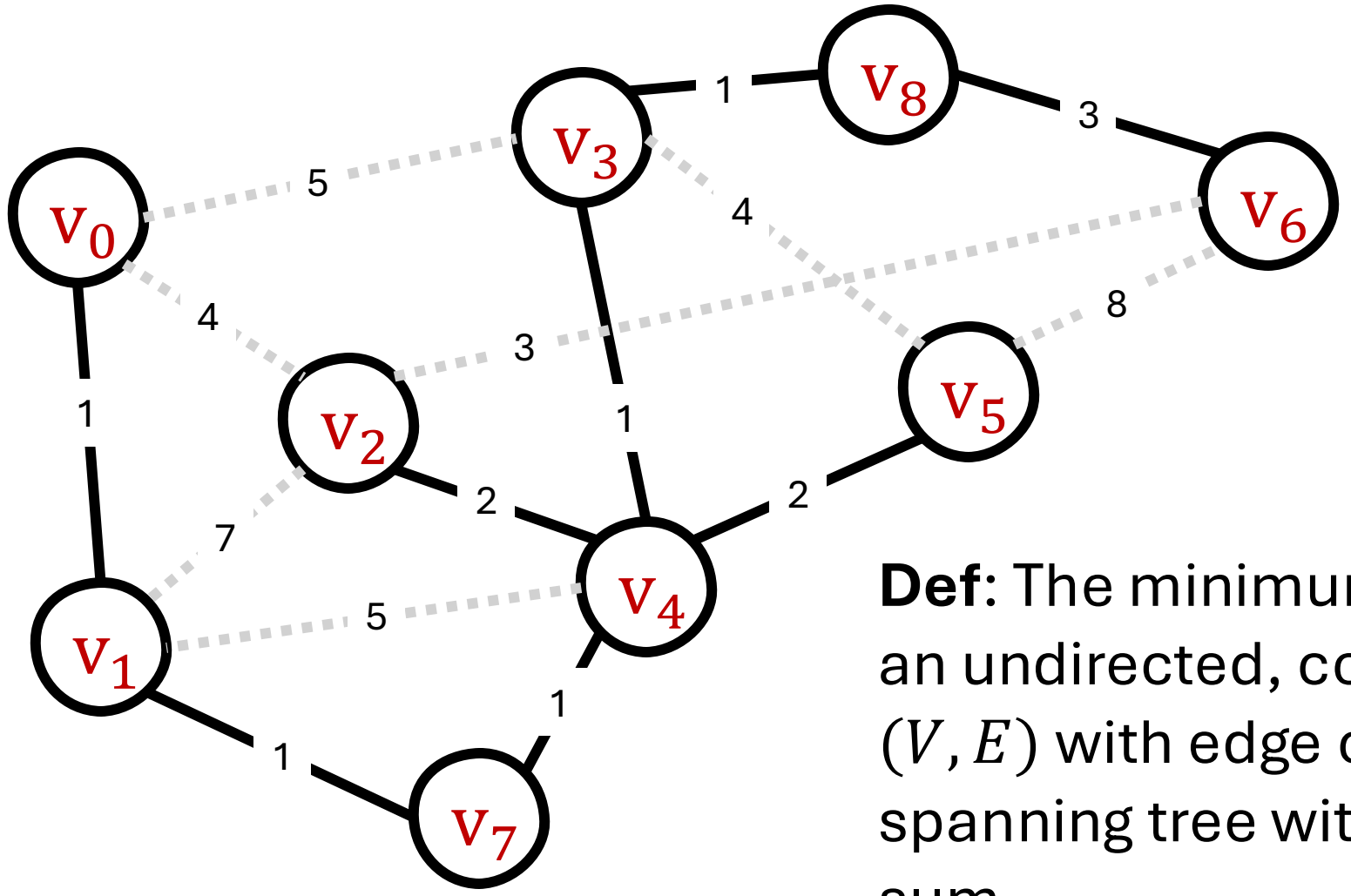
Minimum Spanning Trees



Super Graph

Def: The minimum spanning tree of an undirected, connected graph $G = (V, E)$ with edge costs c_e is the spanning tree with the min total edge sum.

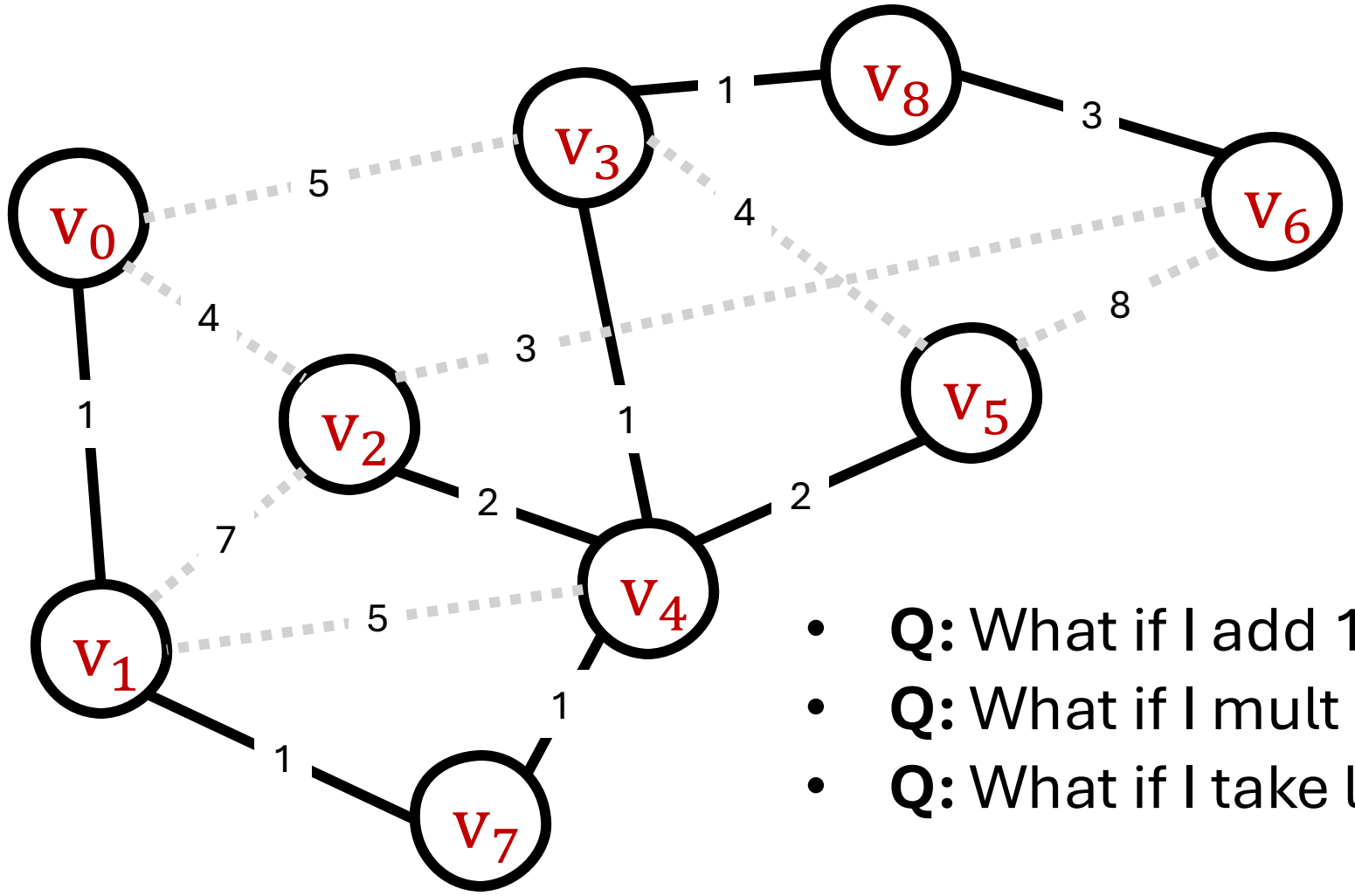
Minimum Spanning Trees (MST)



MST

Def: The minimum spanning tree of an undirected, connected graph $G = (V, E)$ with edge costs c_e is the spanning tree with the min total edge sum.

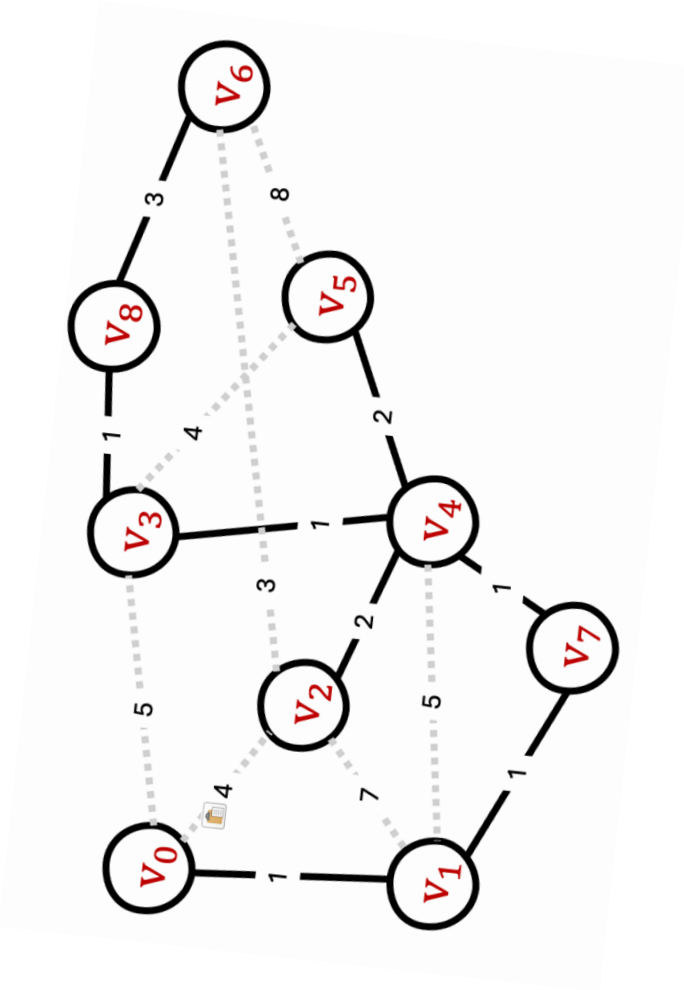
Minimum Spanning Trees (MST)



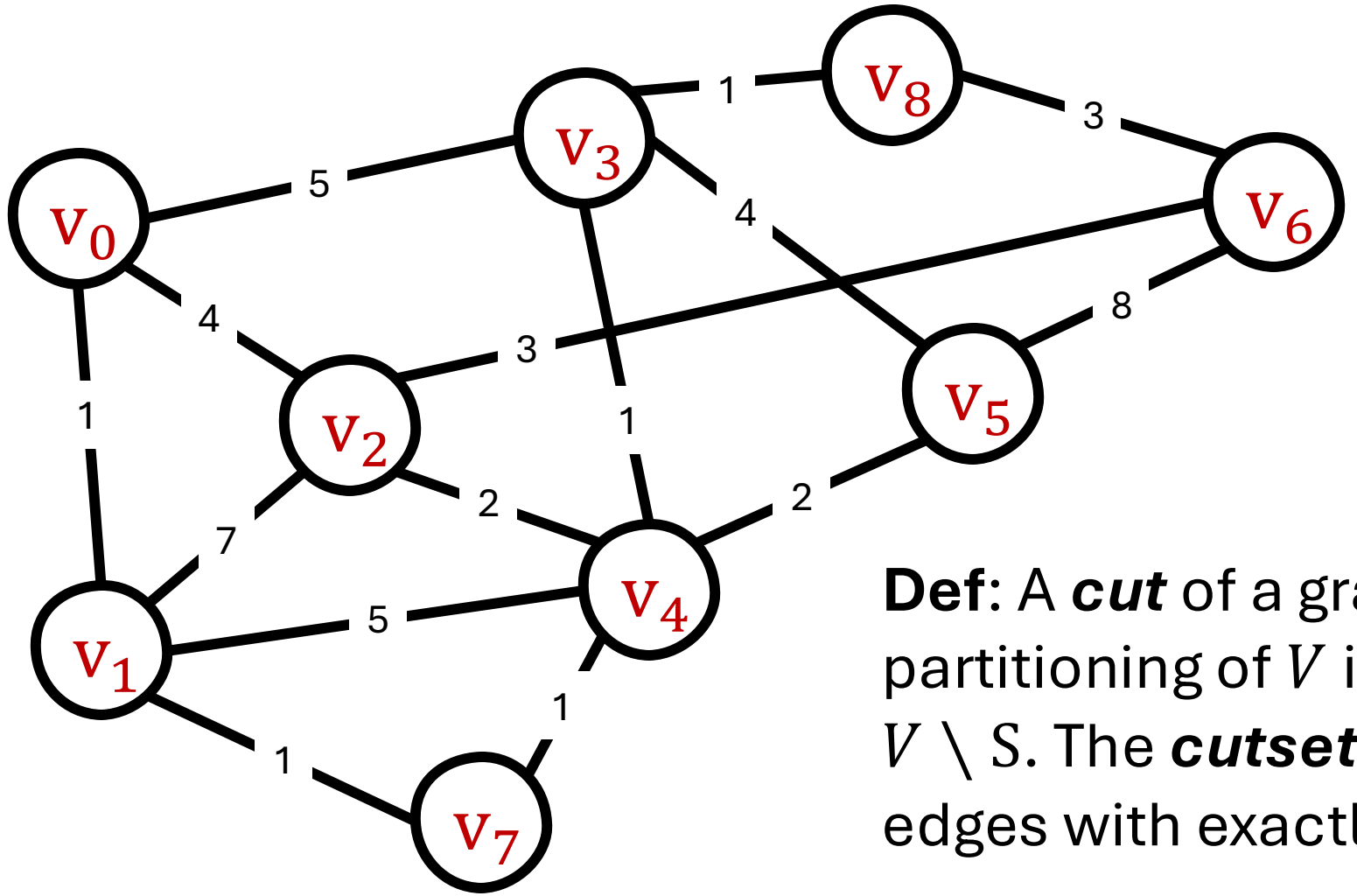
- **Q:** What if I add 100 to each edge?
- **Q:** What if I mult each edge by 23?
- **Q:** What if I take log of each edge?

MST Algorithms (Greedy) Ideas:

- Start with empty graph:
  - **Kruskal:** Add small edge that connect components.
  - **Prim:** Grow a component by taking smallest edge leaving it.
  - **Borůvka**: Add min weight edge leaving each component.
- Start With G:
  - **Reverse Kruskal**: Remove big edges that aren't needed.



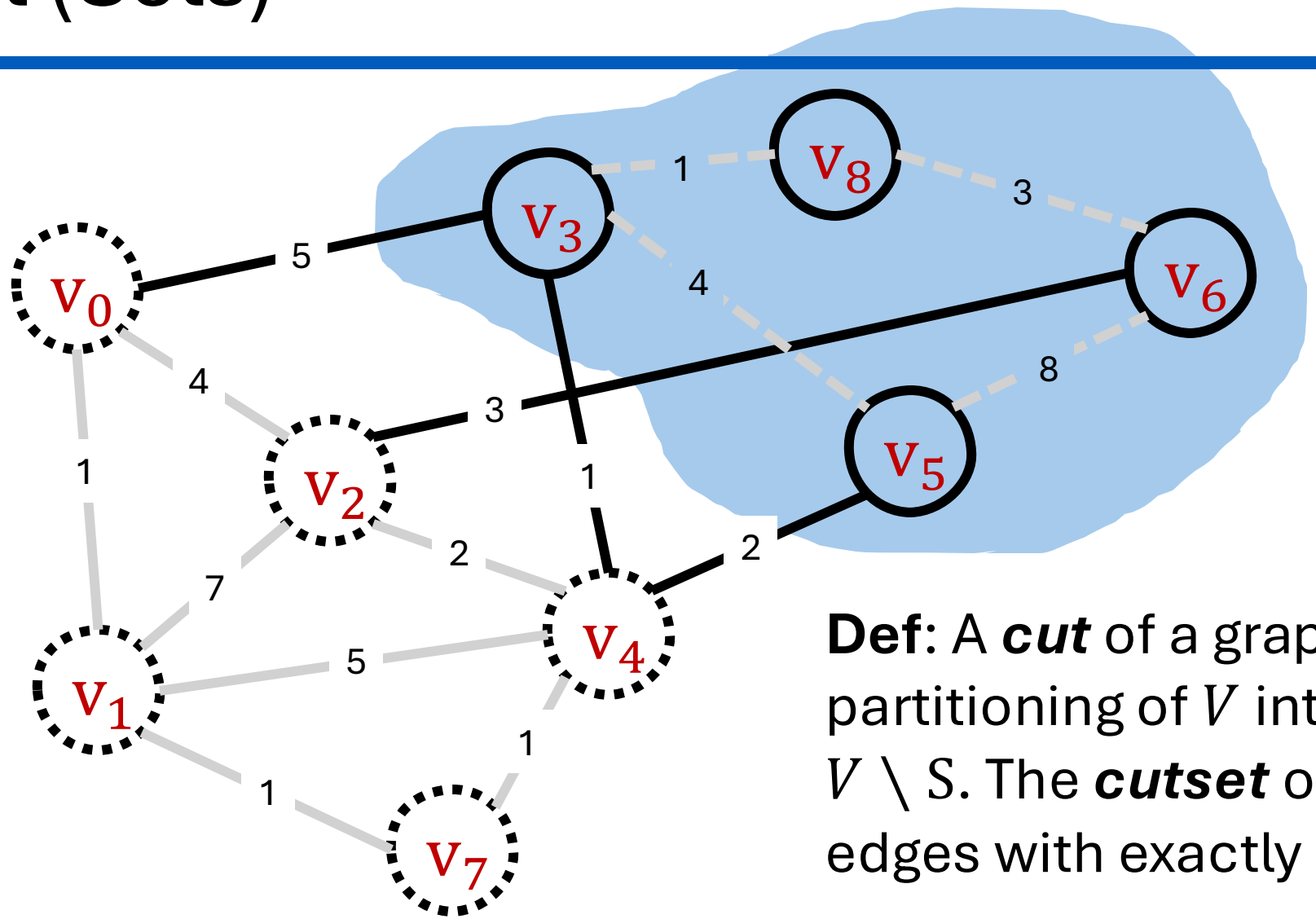
Cut (Sets)



Super Graph

Def: A ***cut*** of a graph $G = (V, E)$ is a partitioning of V into two sets S and $V \setminus S$. The ***cutset*** of S is the set of edges with exactly on endpoint in S.

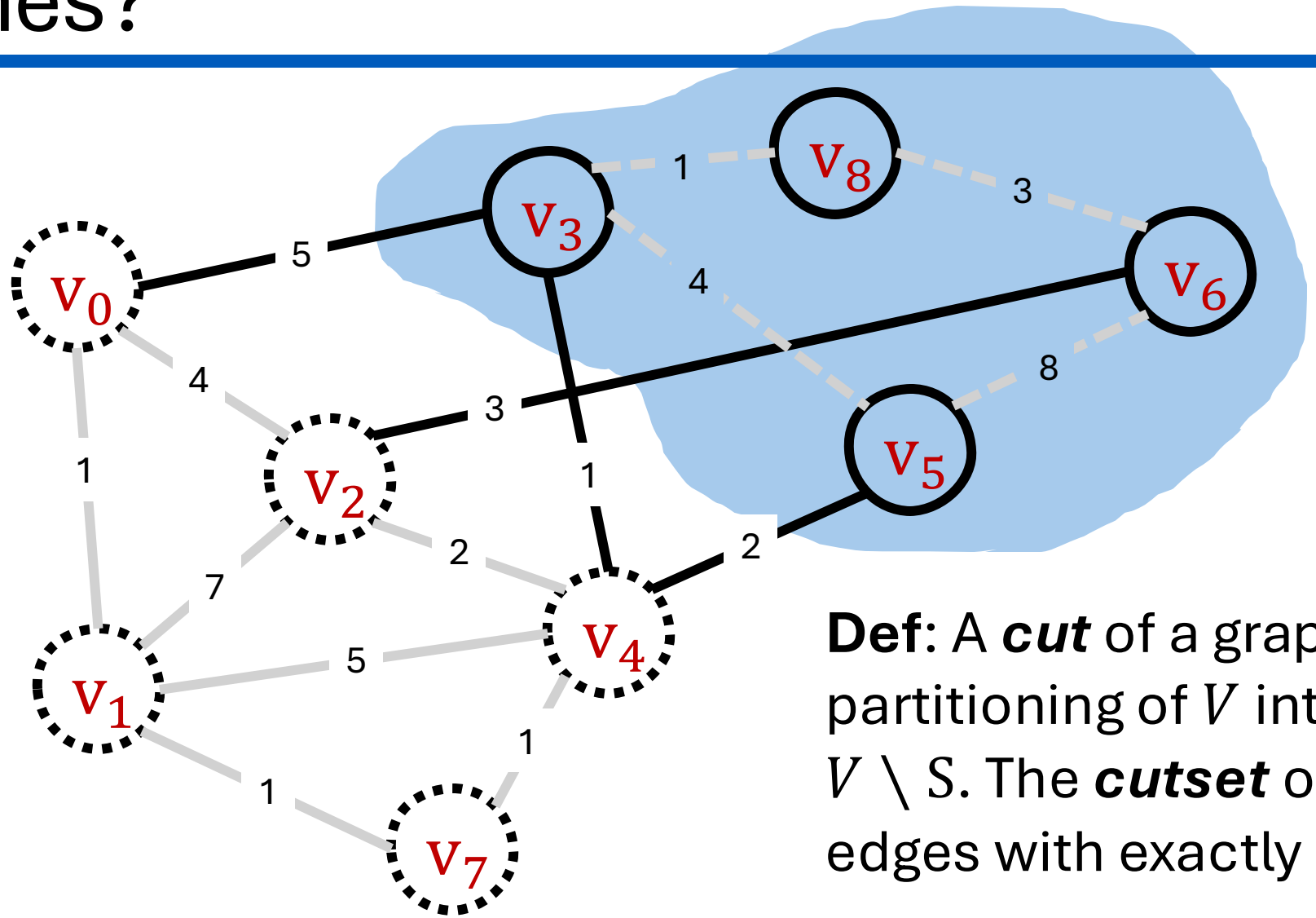
Cut (Sets)



Cut Set of S

Def: A ***cut*** of a graph $G = (V, E)$ is a partitioning of V into two sets S and $V \setminus S$. The ***cutset*** of S is the set of edges with exactly on endpoint in S.

Q: Can a cycle intersect a cut set an odd number of times?



Cut Set of S

Def: A ***cut*** of a graph $G = (V, E)$ is a partitioning of V into two sets S and $V \setminus S$. The ***cutset*** of S is the set of edges with exactly on endpoint in S.