CSE 331:
Algorithms & Complexity
"MSTs"

Prof. Charlie Anne Carlson (She/Her)

Lecture 20

Monday October 20, 2025

University at Buffalo

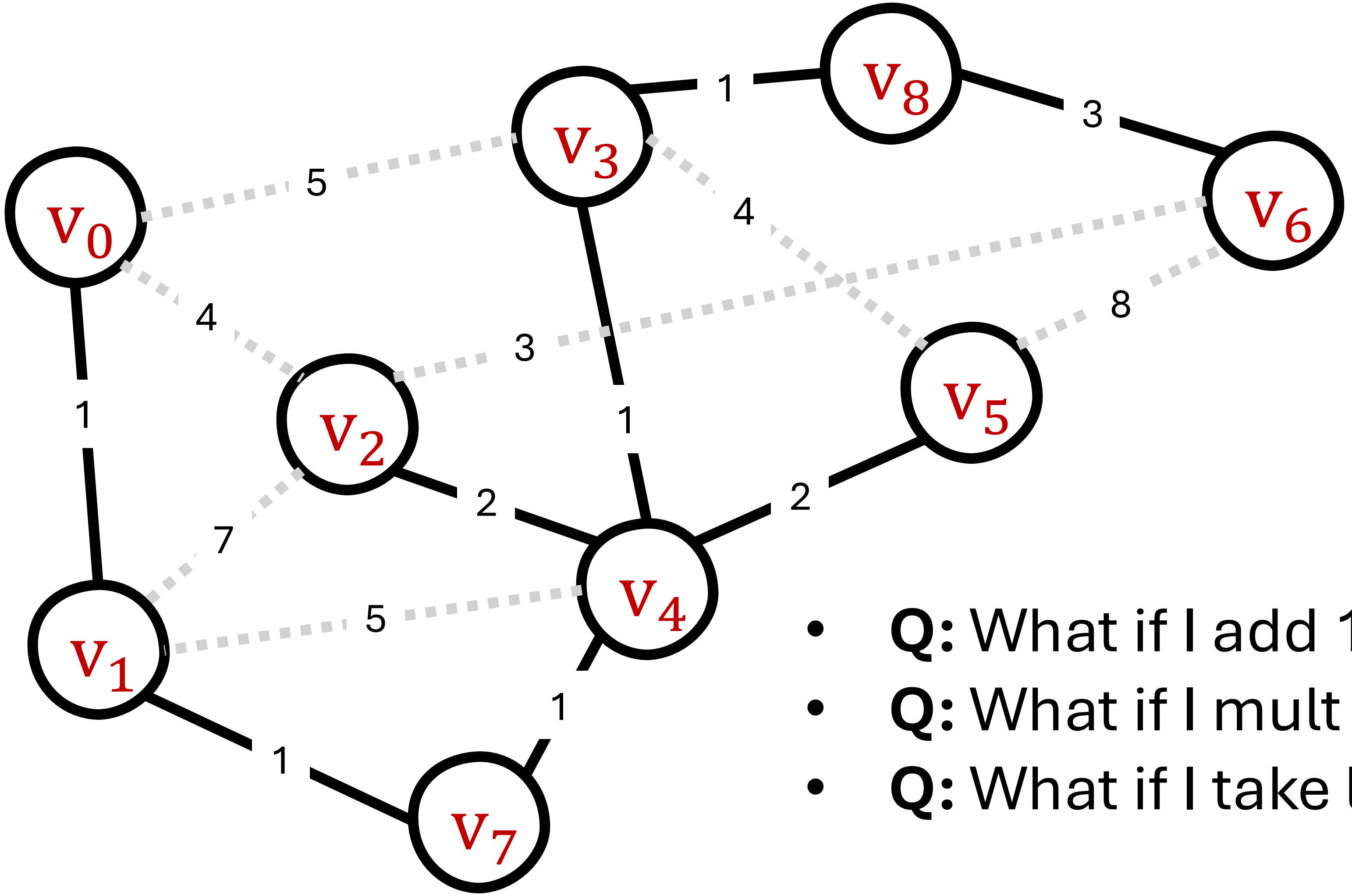
Schedule

1. Course Updates
2. Min Spanning Trees
3. Cut Property
4. Kruskal’s Algorithm
5. Prim’s Algorithm

Course Updates

- Midterm Part I Out
- Midterm Part II Out – Soonish
- Post Midterm Grades – Before Wednesday
- HW 4 Due Tomorrow
- Group Project
    - First Problems Oct 31st

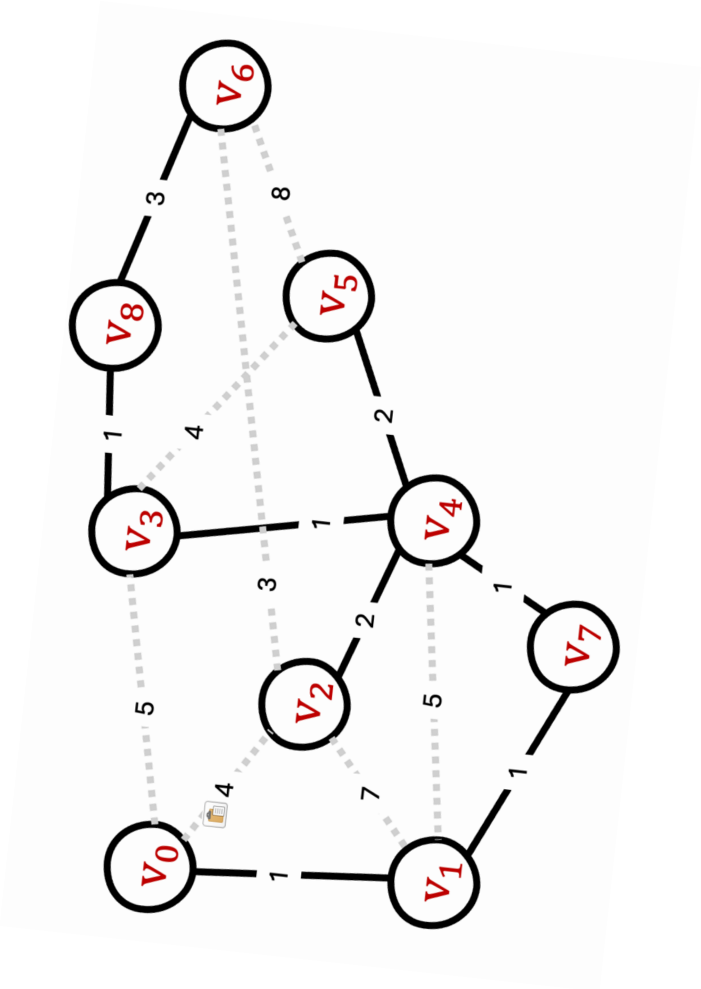
Minimum Spanning Trees (MST)



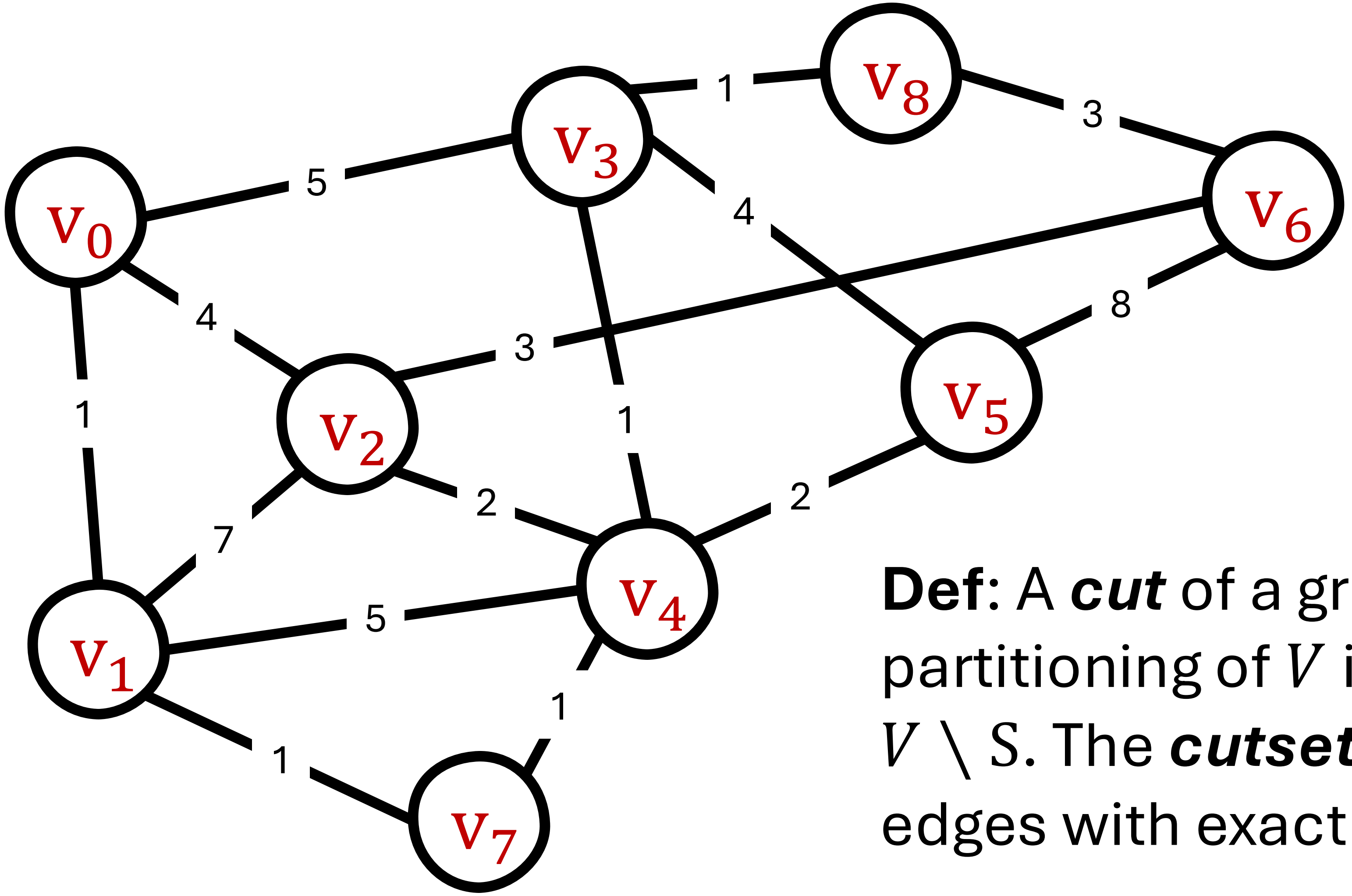
- **Q:** What if I add 100 to each edge?
- **Q:** What if I mult each edge by 23?
- **Q:** What if I take log of each edge?

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)

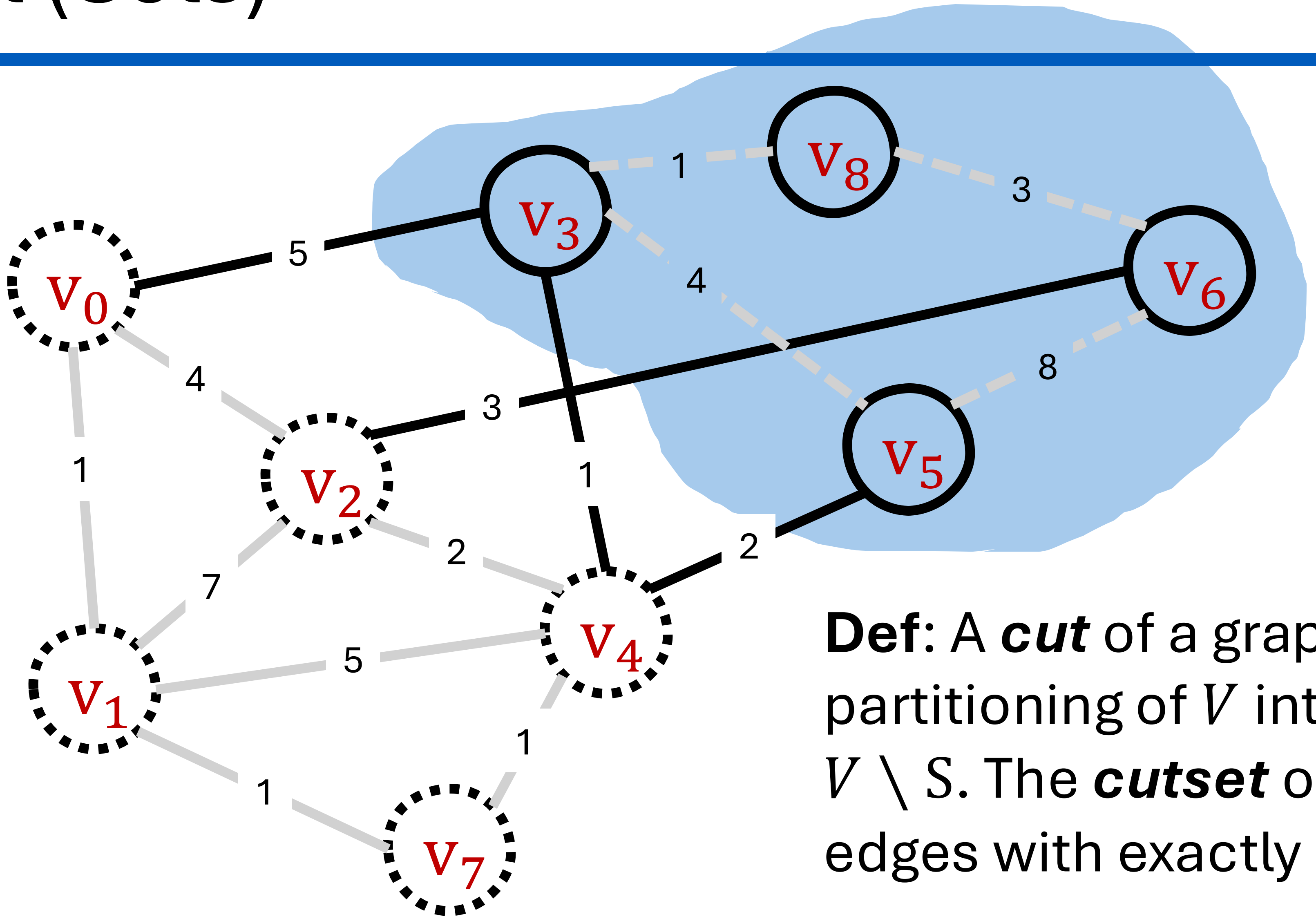


Super Graph

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)

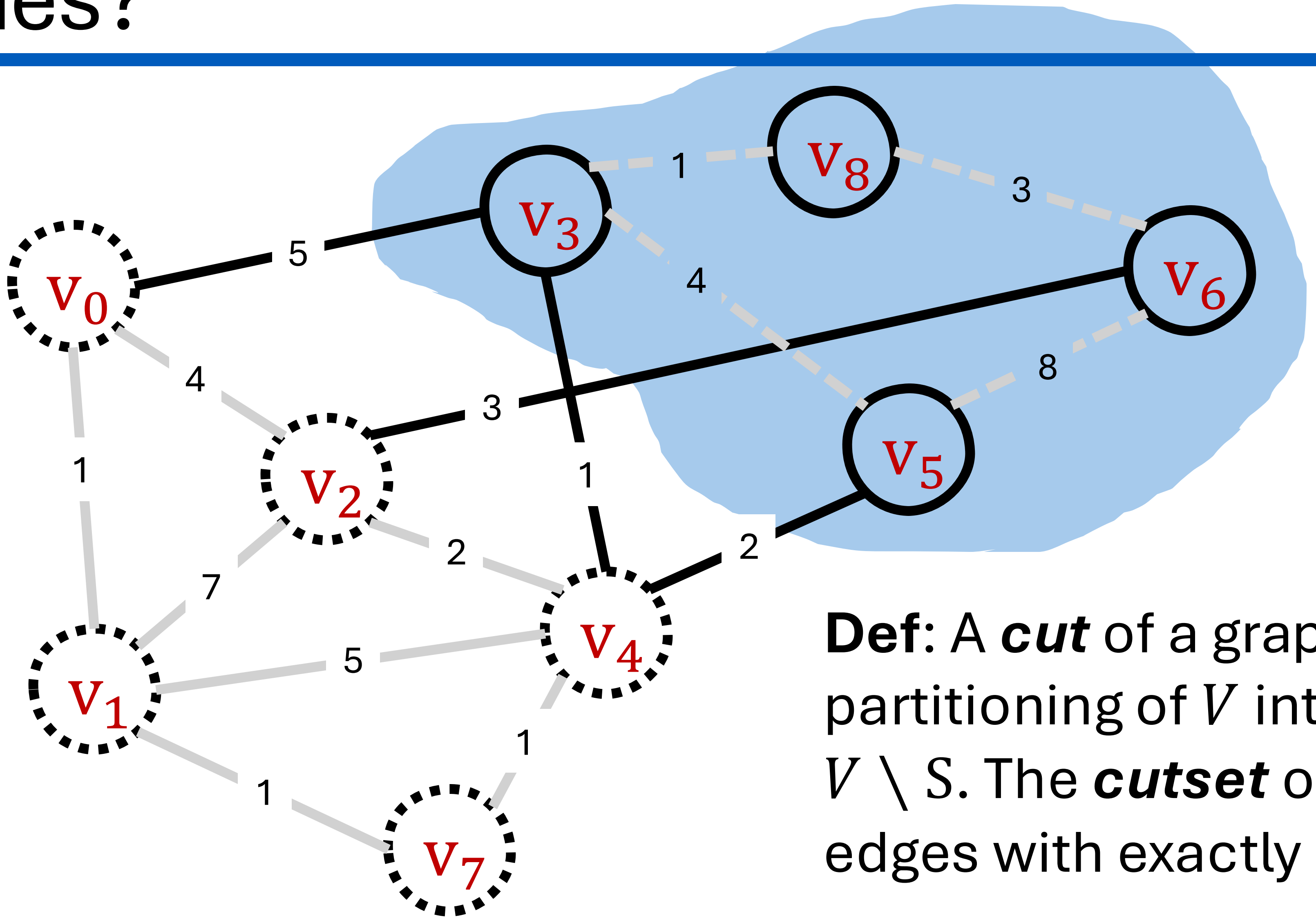
S



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.

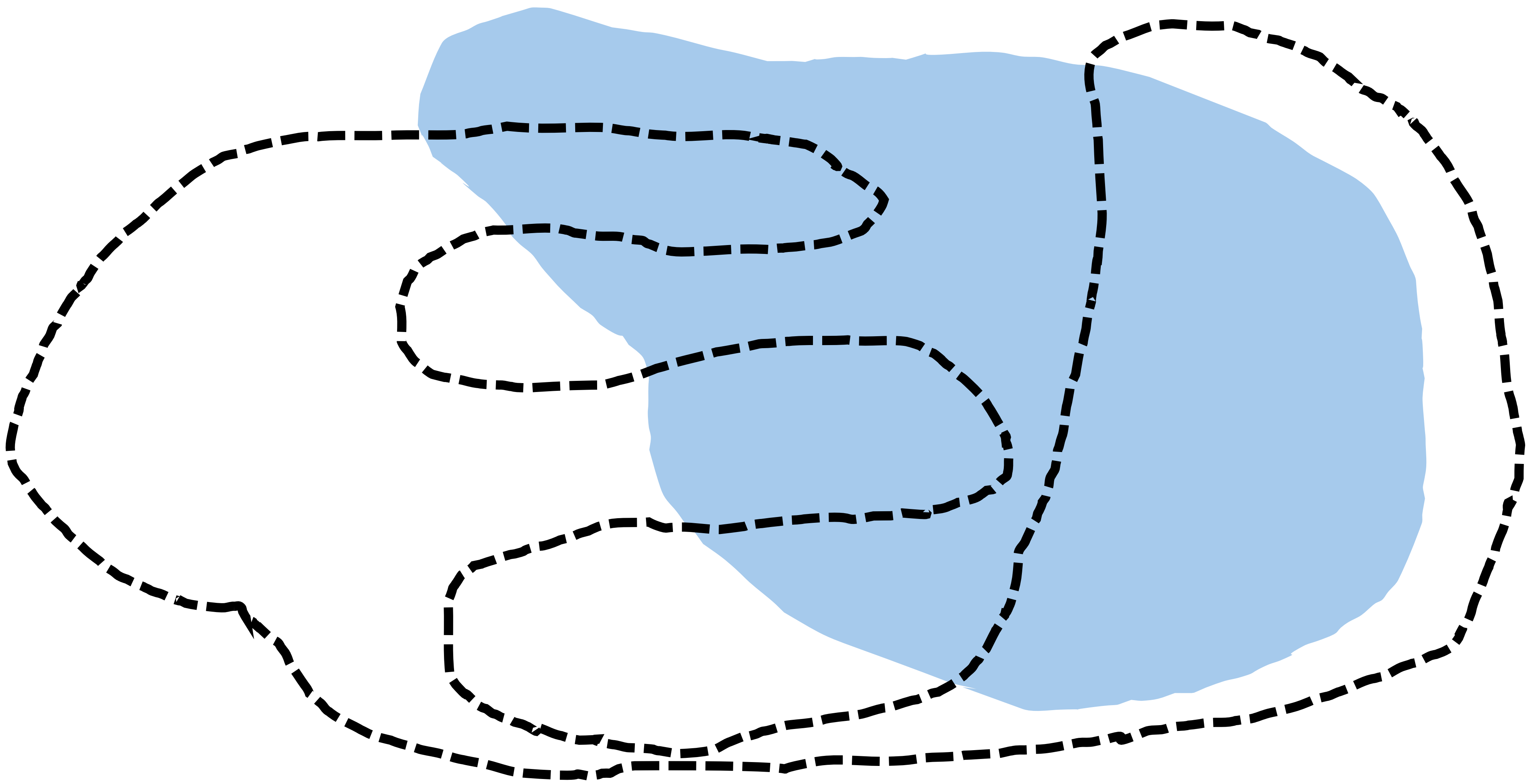
Q: Can a cycle intersect a cut set an odd number of times?



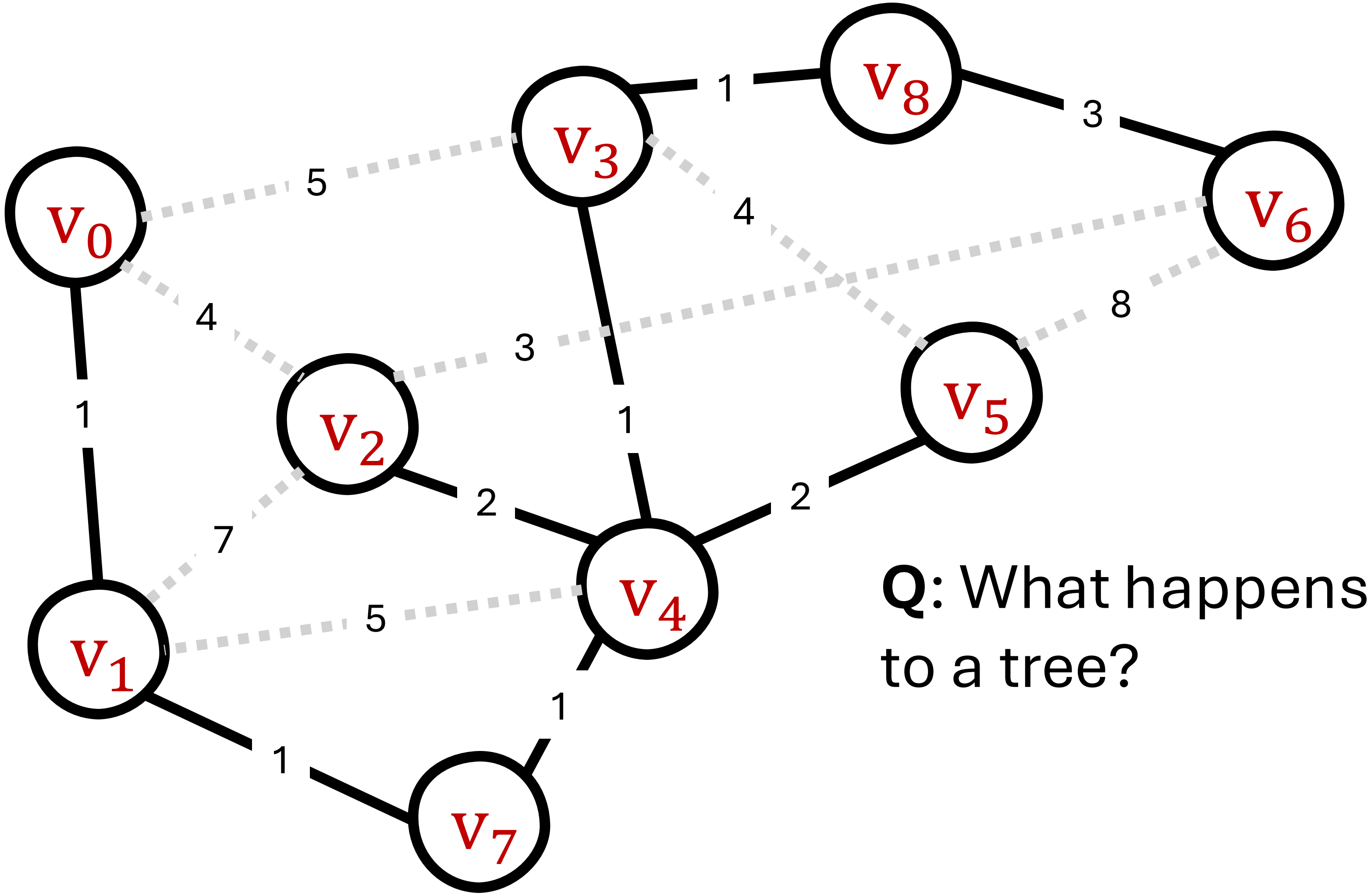
Cut Set of S

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.

A: No. “If it enters, it must leave.”



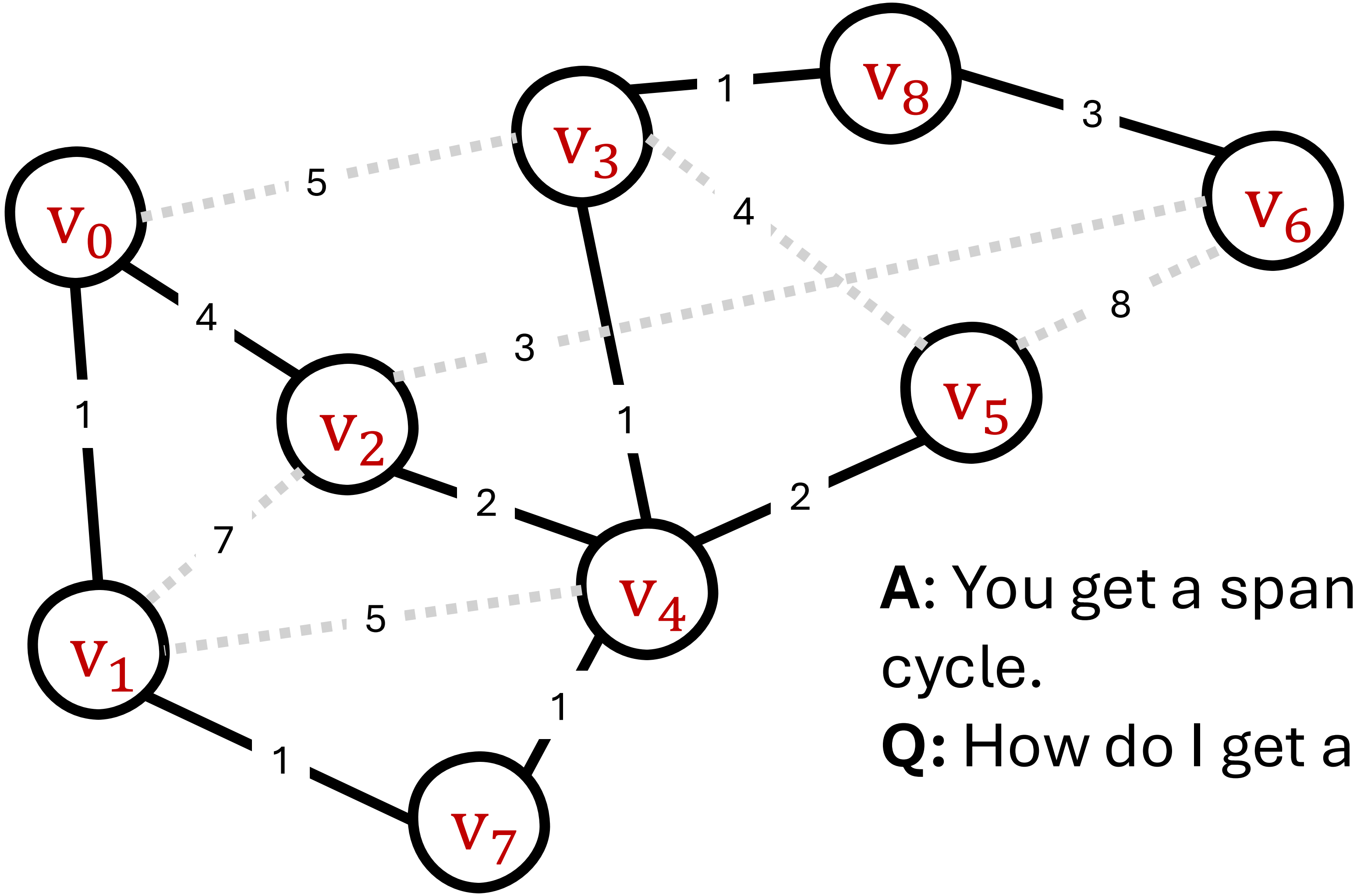
Spanning Trees



Tree

Q: What happens when I add an edge to a tree?

Spanning Not Trees

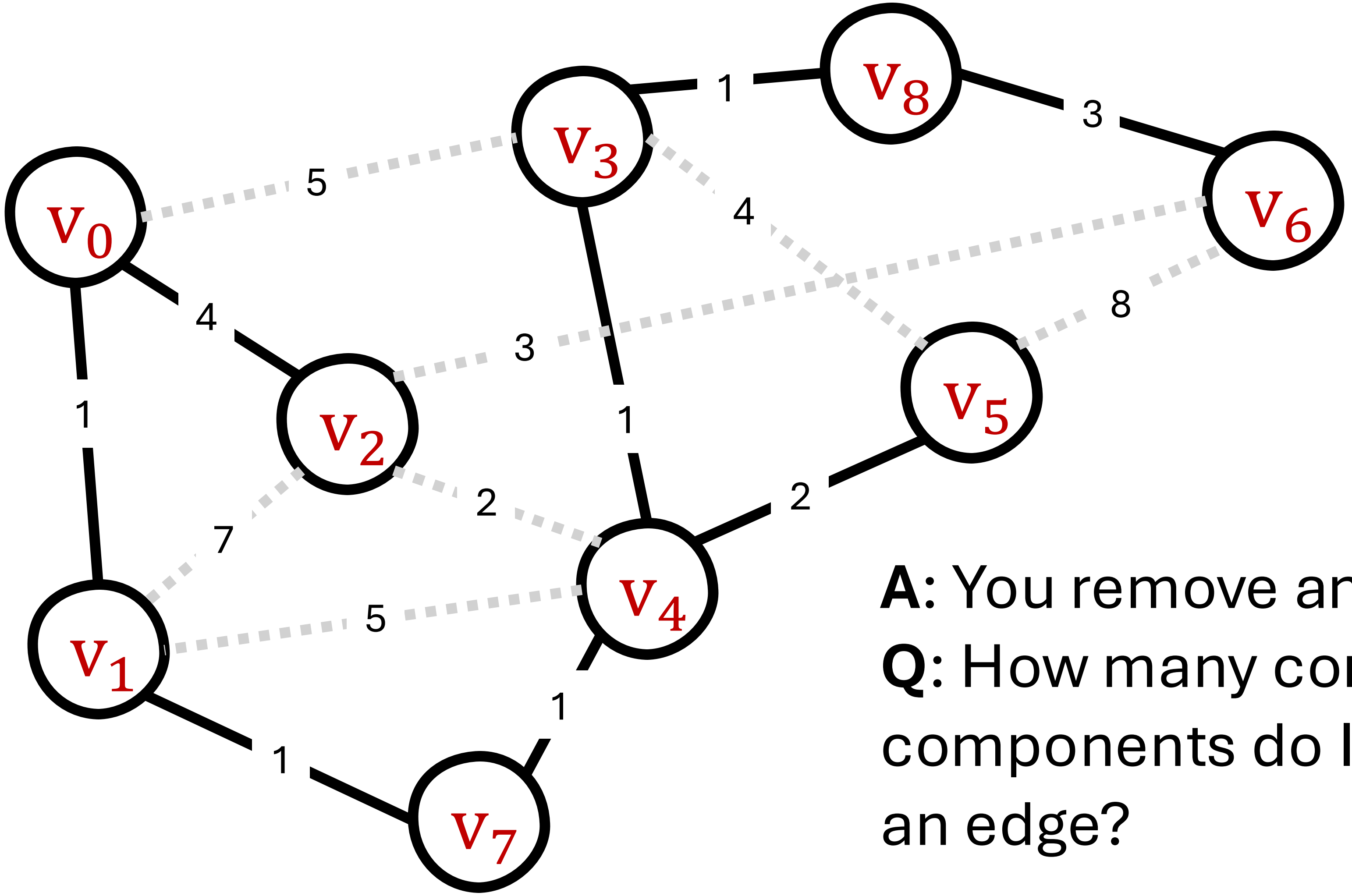


Not Tree

A: You get a spanning graph with one cycle.

Q: How do I get a tree again?

Spanning Trees

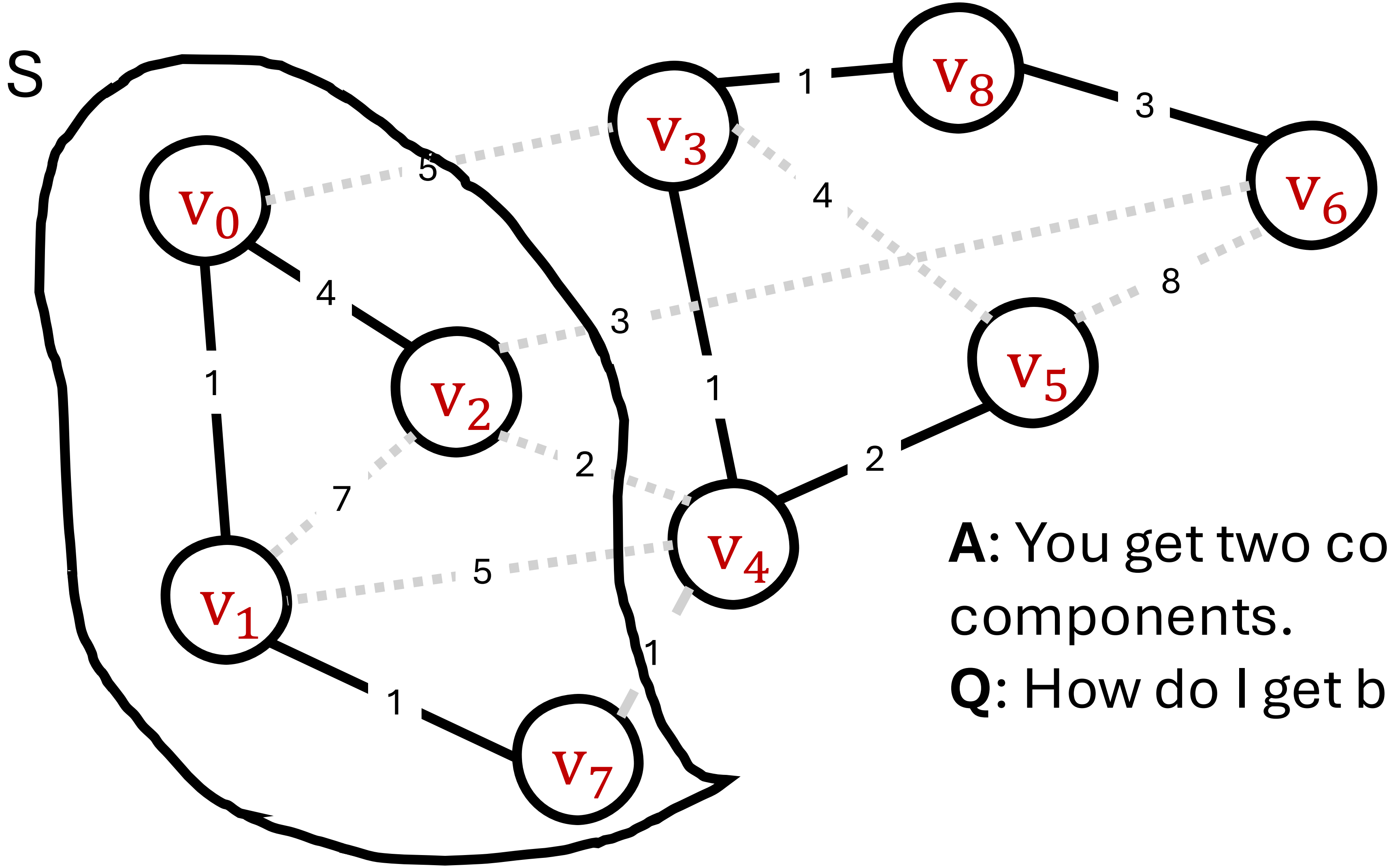


Tree

A: You remove any edge in the cycle!

Q: How many connected components do I get when I remove an edge?

Spanning Forest

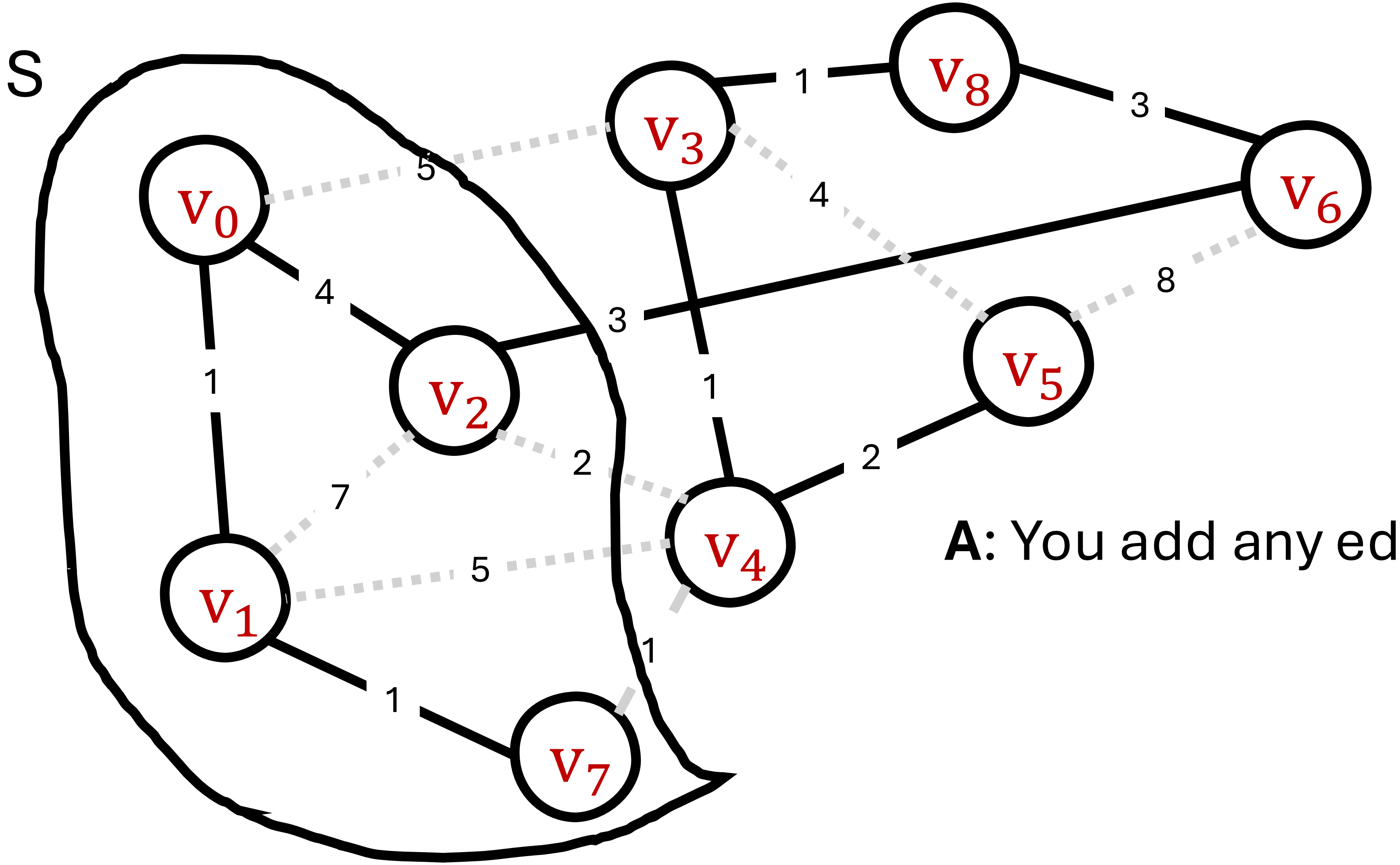


Forest/Cut

A: You get two connected components.

Q: How do I get back a tree?

Spanning Tree

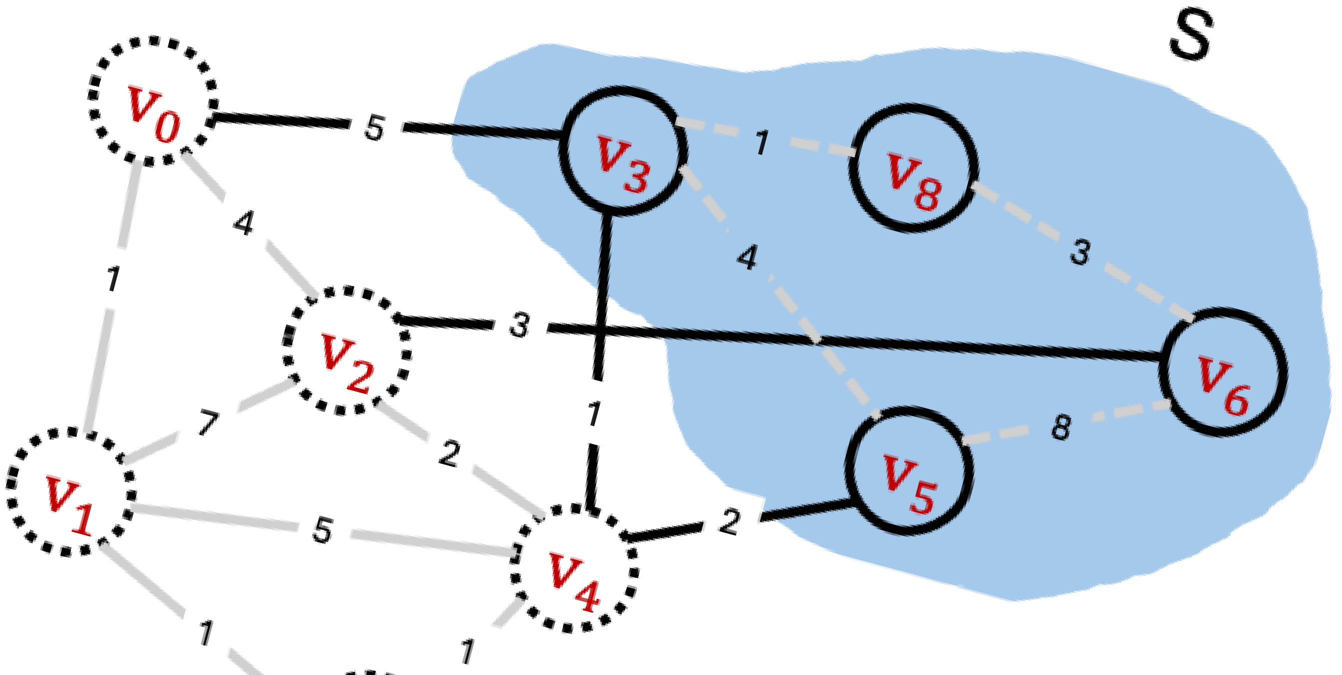


Forest/Cut

A: You add any edge in the cut set!

Cut Property

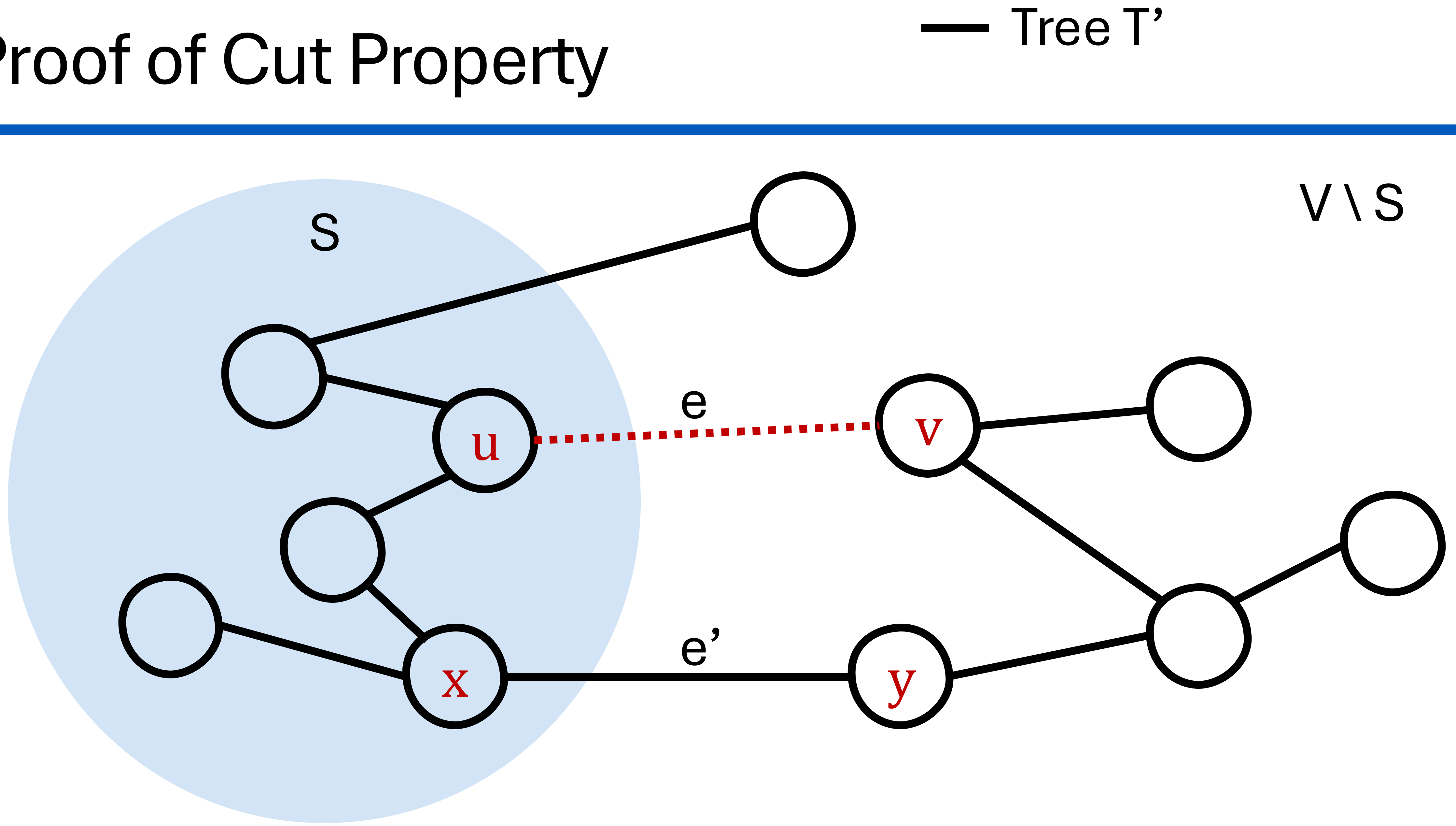
Lemma: Fix a graph $G = (V, E)$ with edge weights ℓ. Assume that all edges are distinct. Let S be any subset of nodes that is neither empty or equal to all of V, and let $e = (u, v)$ be the minimum-cost edge with on end in S and the other in $V \setminus S$. Then every minimum spanning tree contains the edge e.



Proof of Cut Property

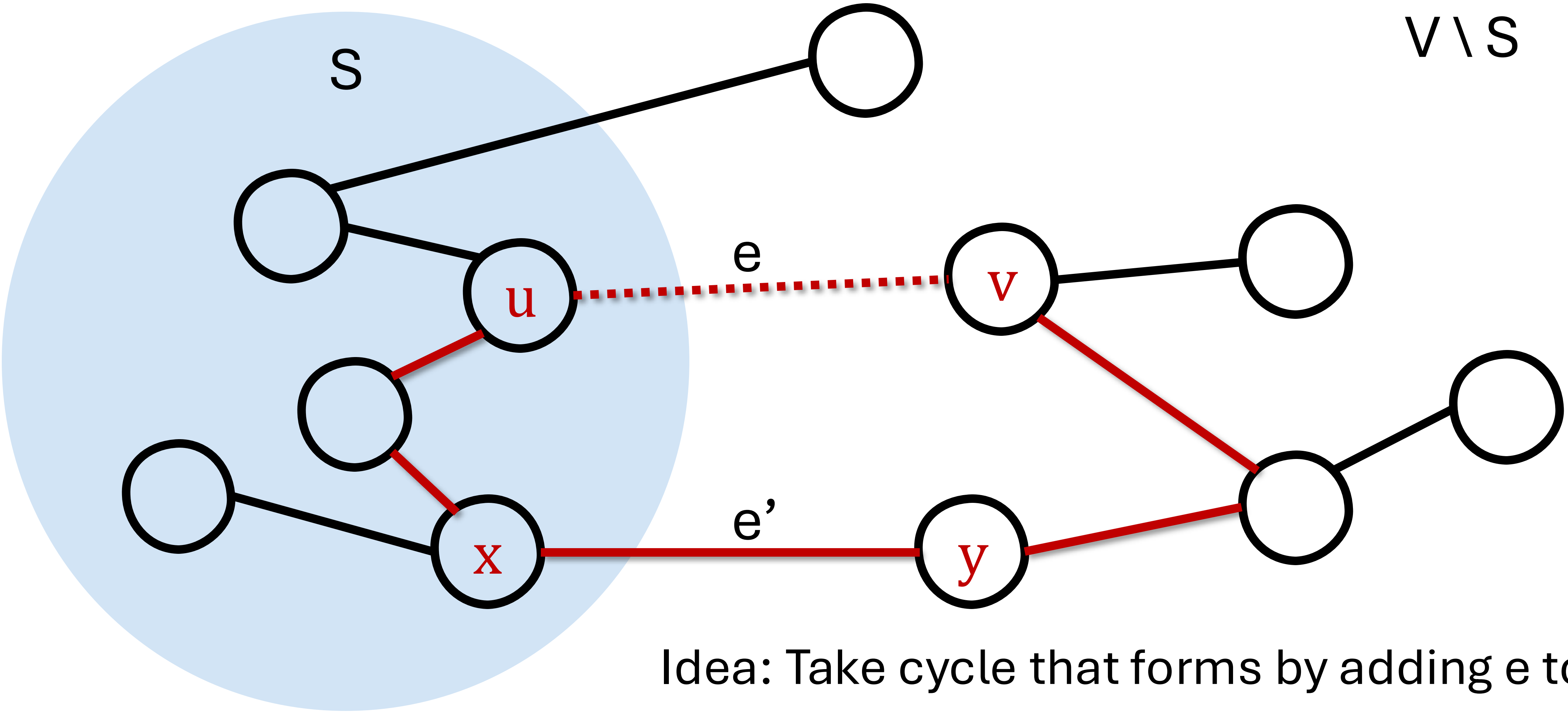
- We will do an exchange argument.
  - Let T be a spanning tree that doesn't contain e = (u,v).
  - We will show that we can construct a tree T' that does include e that has strictly less total weight.
    - To this end, we will identify another edge e' = (x,y) that is in T and be be "exchanged" with e.

Proof of Cut Property



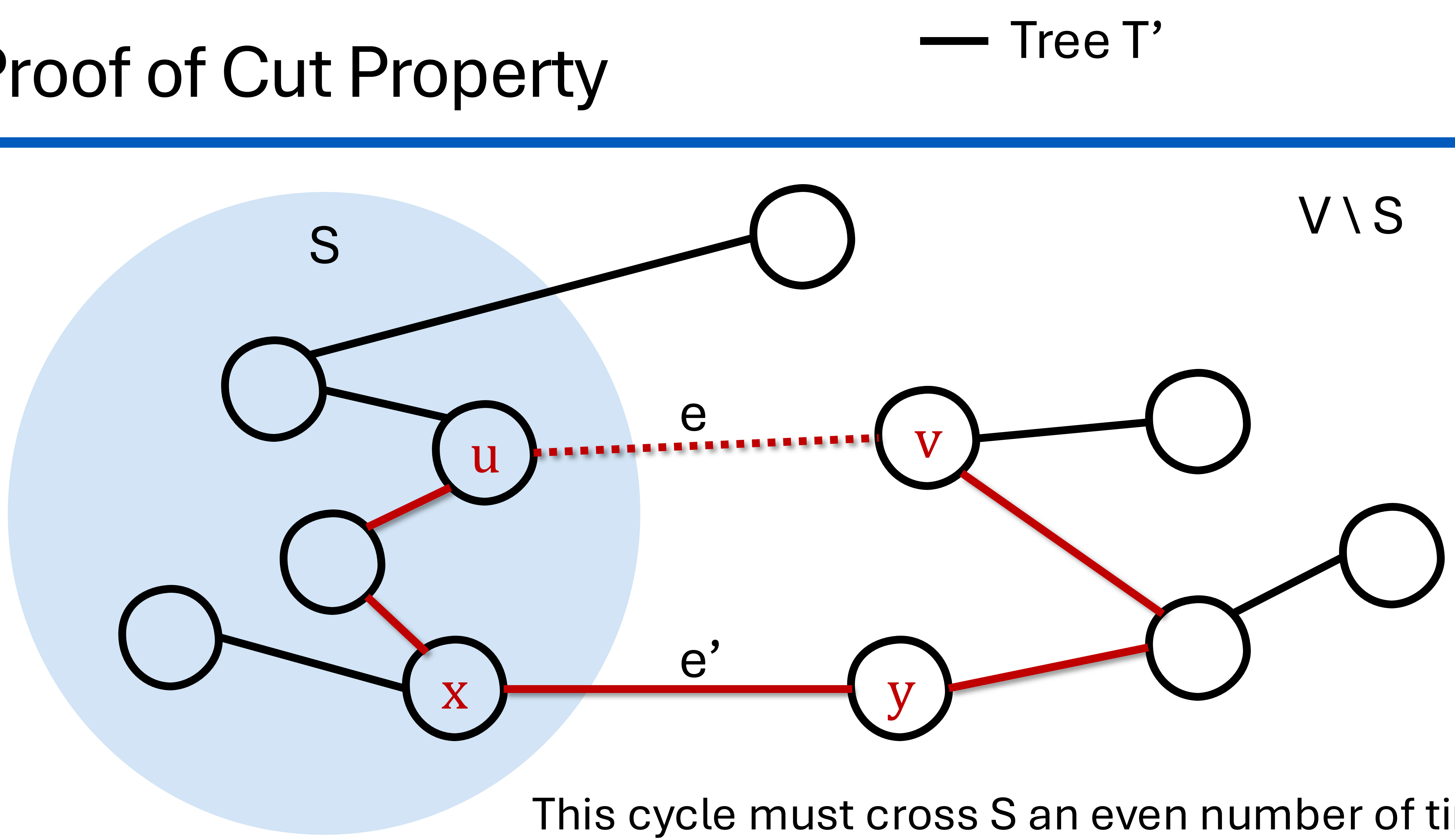
Proof of Cut Property

— Tree T'



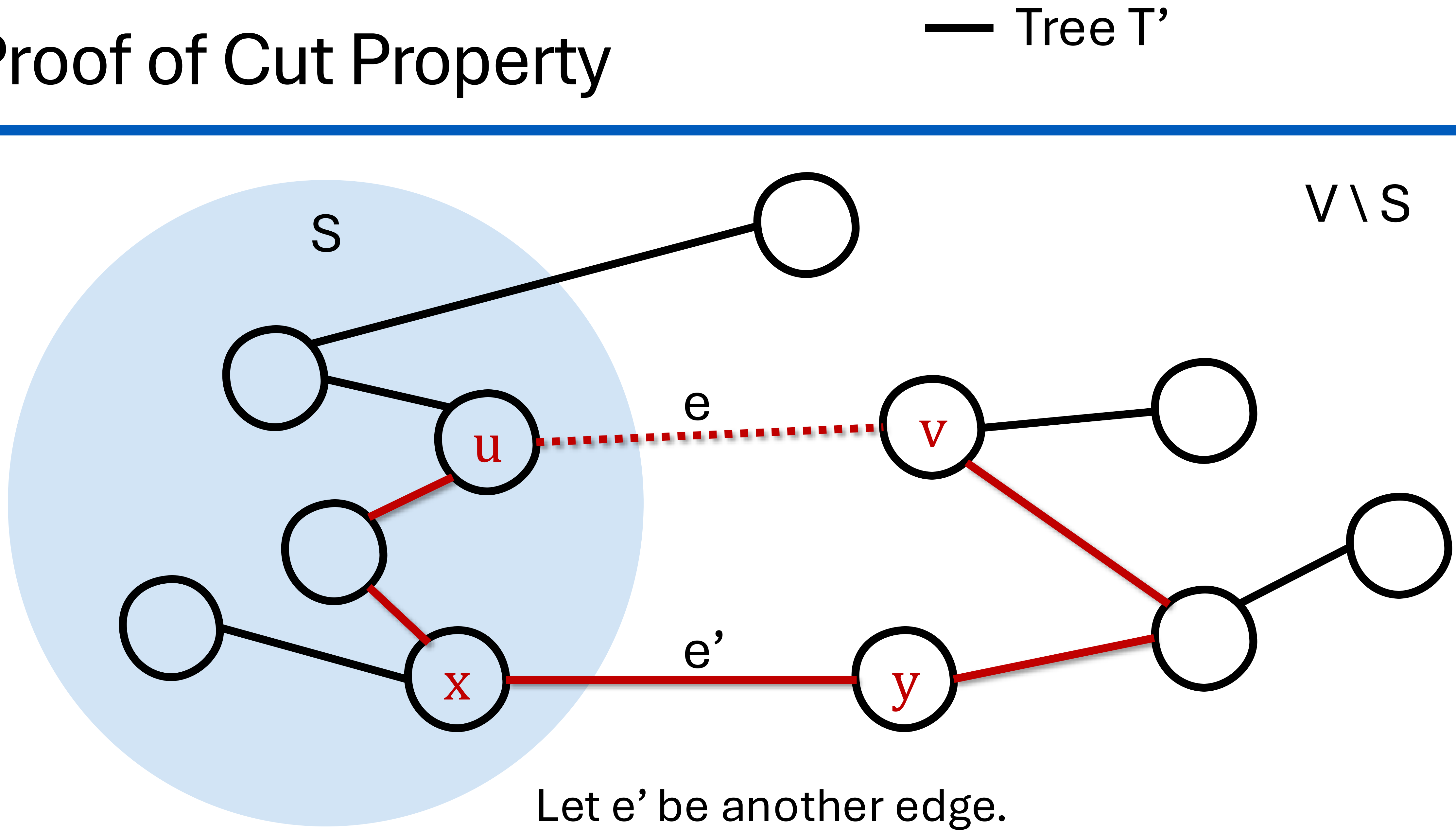
Idea: Take cycle that forms by adding e to T.

Proof of Cut Property



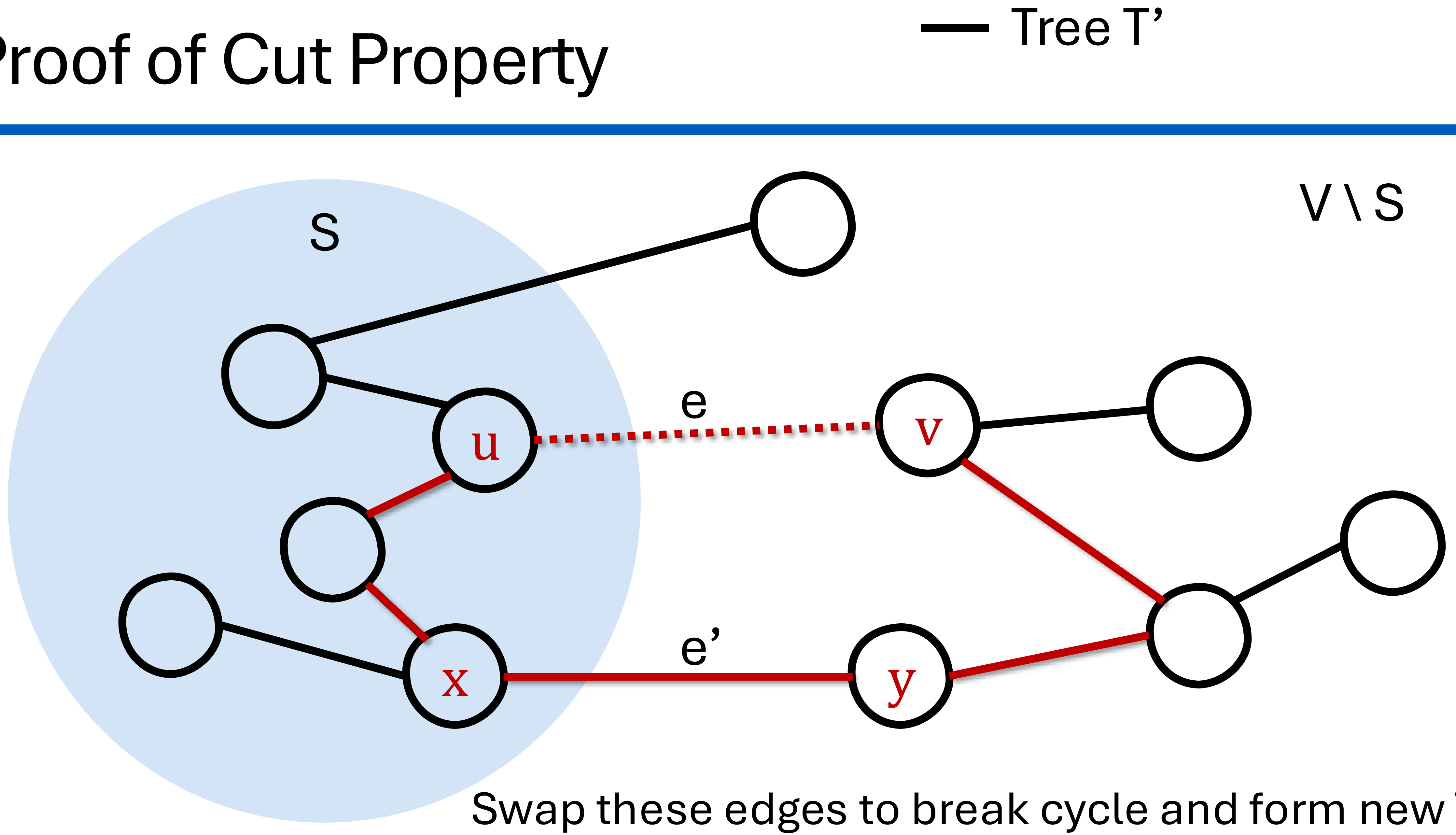
This cycle must cross S an even number of times.

Proof of Cut Property



Let e' be another edge.

Proof of Cut Property



Swap these edges to break cycle and form new Tree.

Proof of Cut Property

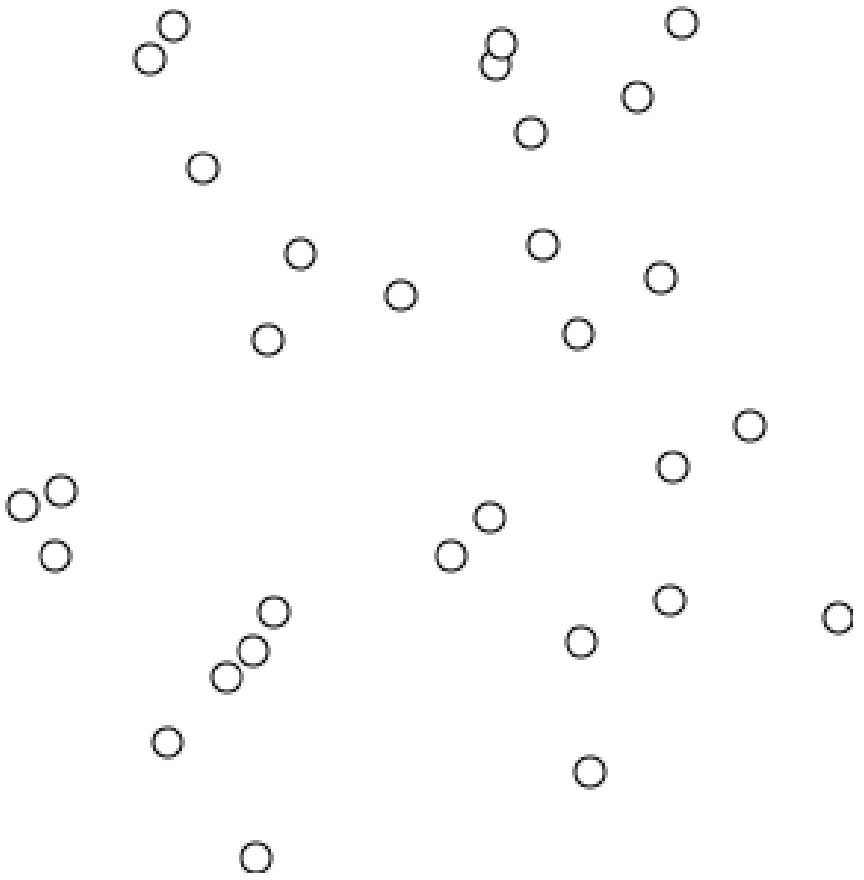
- We will do an exchange argument.
  - Let T be a spanning tree that doesn't contain e = (u,v).
  - We will show that we can construct a tree T' that does include e that has strictly less total weight.
    - To this end, we will identify another edge e' = (x,y) that is in T and be be "exchanged" with e.
  - Since T is a spanning tree there must be a path from u to v.
    - Take this path and let e'=(x,y) be first edge to leave S.
    - By lemma assumption, we know that $\ell_e < \ell_{e'}$.
    - Swap e and e' to make T'.
  - T' is connected and acyclic and total weight went down.

Proof of Cut Property

- We will do an exchange argument.
  - …
  - T’ is connected and acyclic and total weight went down.
    - To see T’ is connected, take any pair of vertices (a,b) and their path in T.
      - If this path used e then ”reroute” to use e’.
      - Otherwise, path still exists.
    - To see T’ is acyclic, note that the only cycle in T with e must have been the cycle that contained e and it is no longer a cycle since e’ was removed.
    - To see that the weight went down, recall $\ell_e < \ell_{\mathrm{e}'}$.

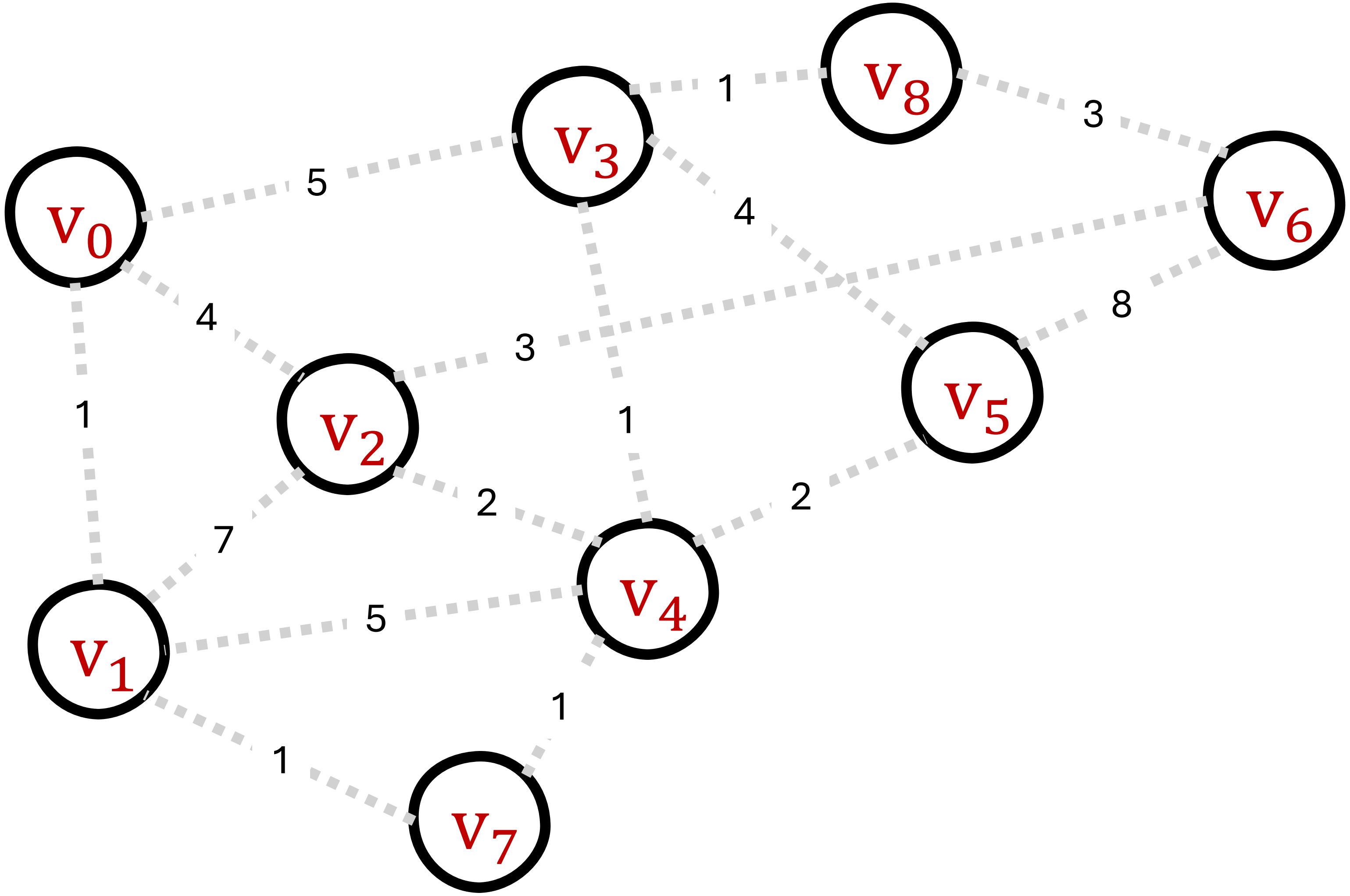
Kruskal's Algorithm

- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Sort E using values in L
    - Break ties arbitrarily
  - Let T be an empty graph
  - For e in E:
    - If adding e to T doesn't case a cycle, add it.

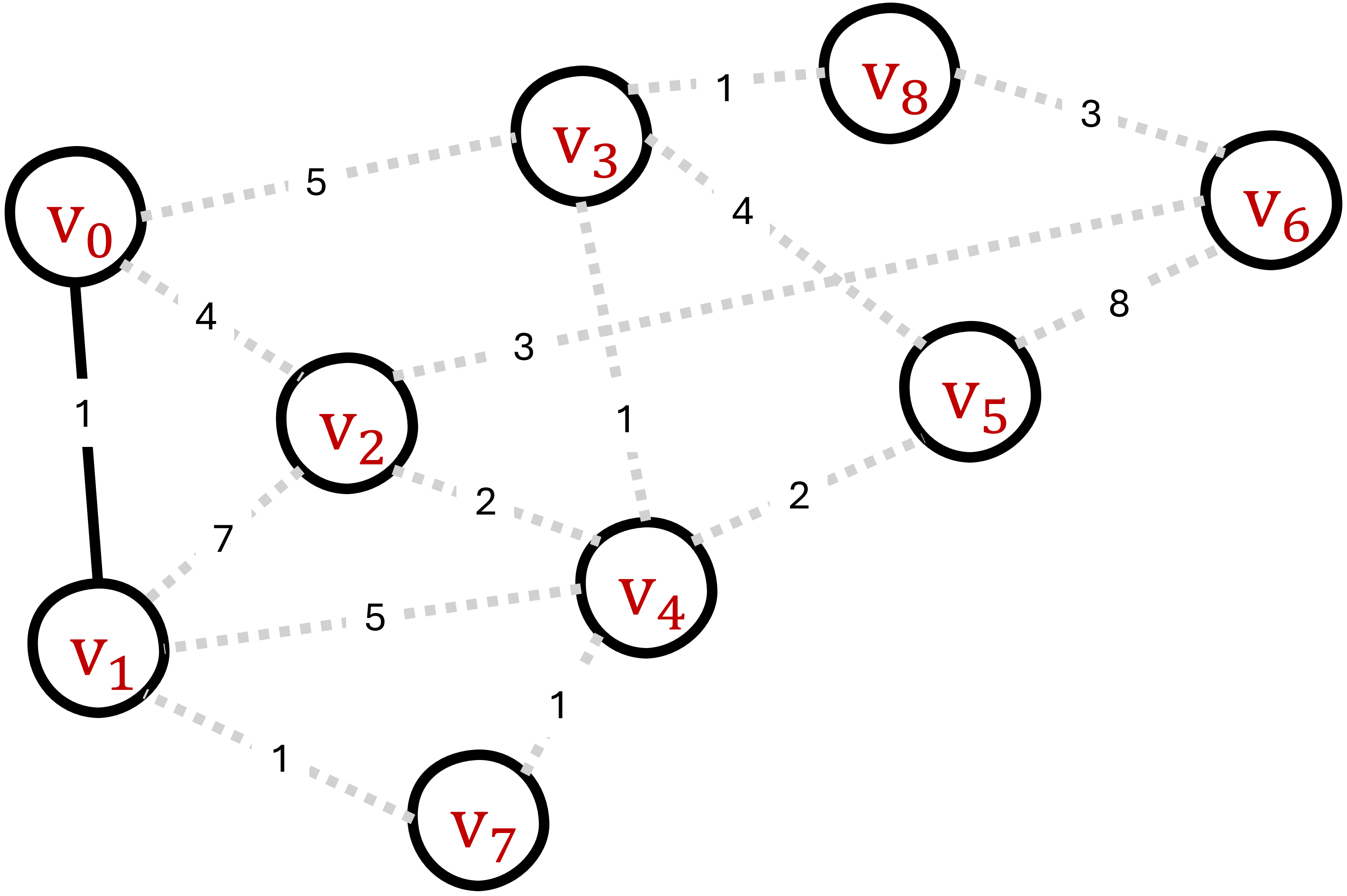


https://en.wikipedia.org/wiki/File:KruskalDemo.gif

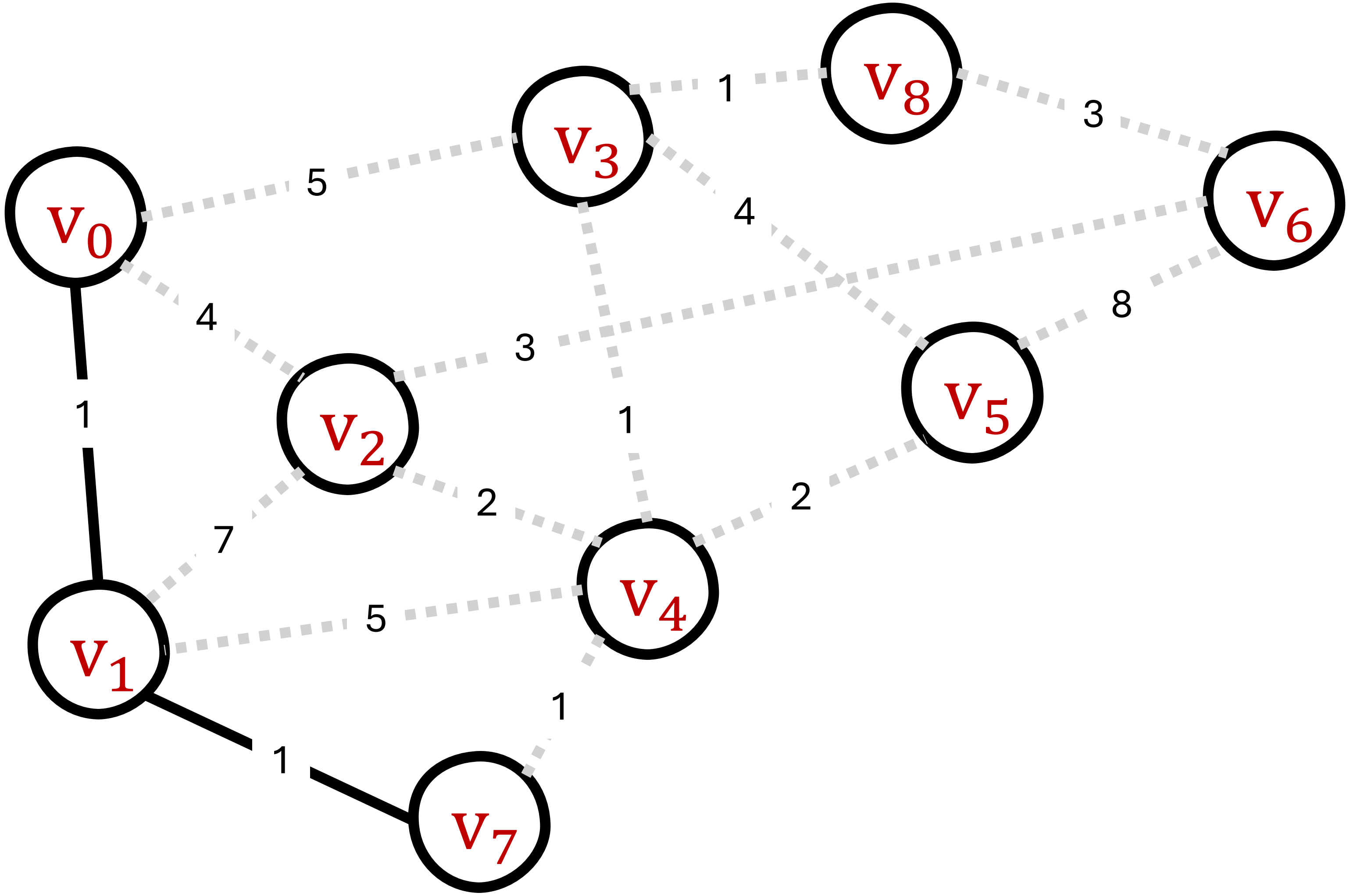
Kruskal's Algorithm



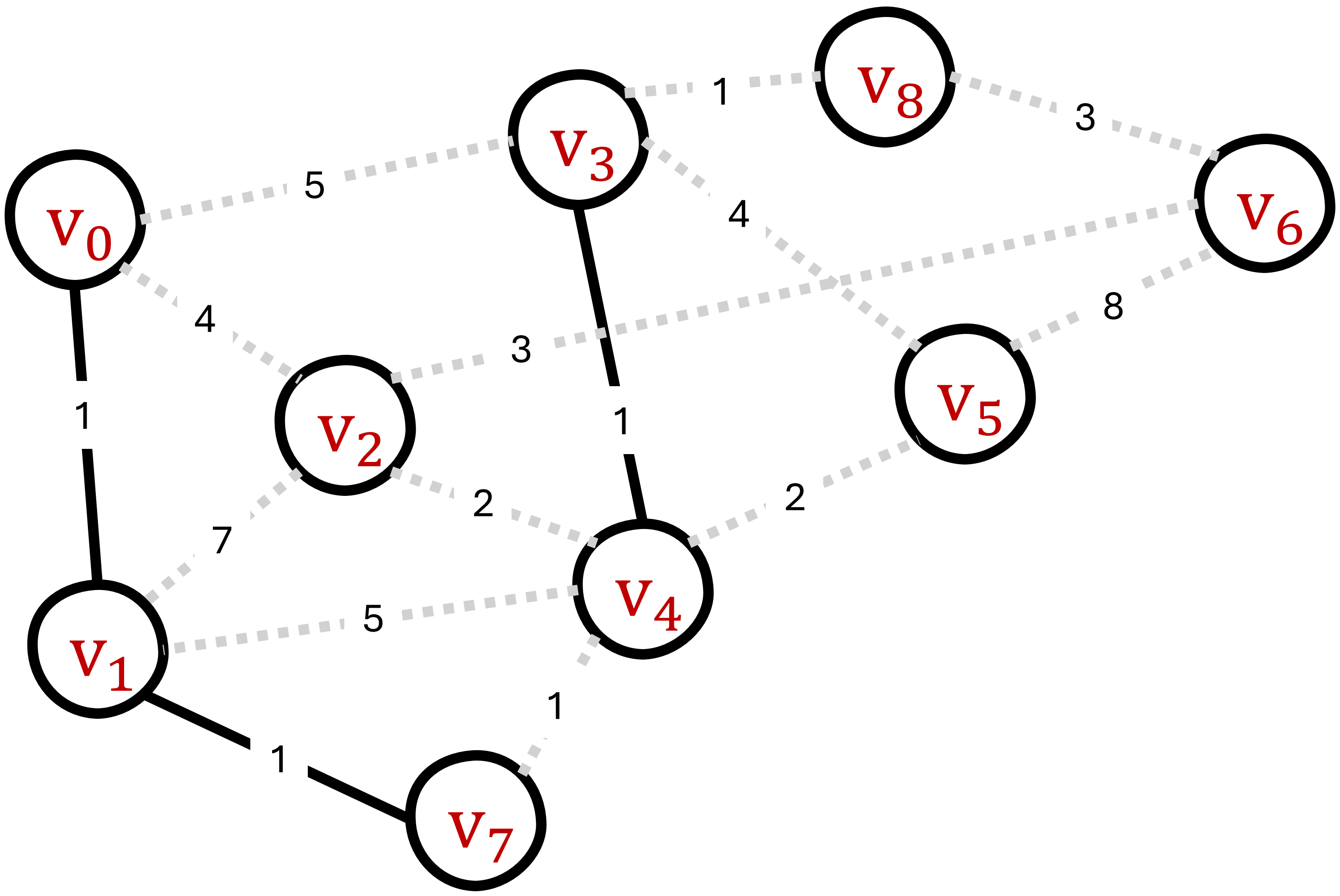
Kruskal’s Algorithm



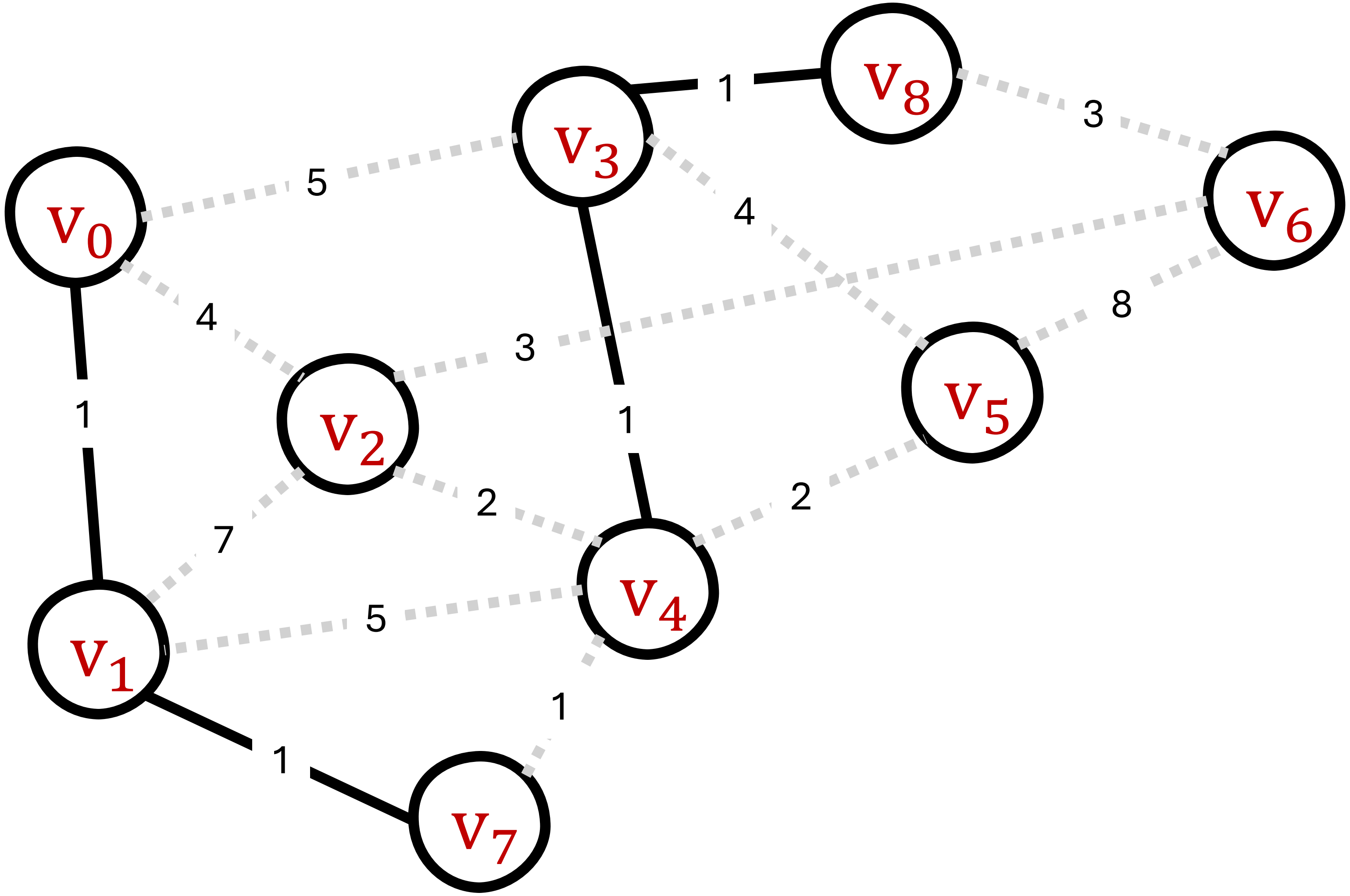
Kruskal's Algorithm



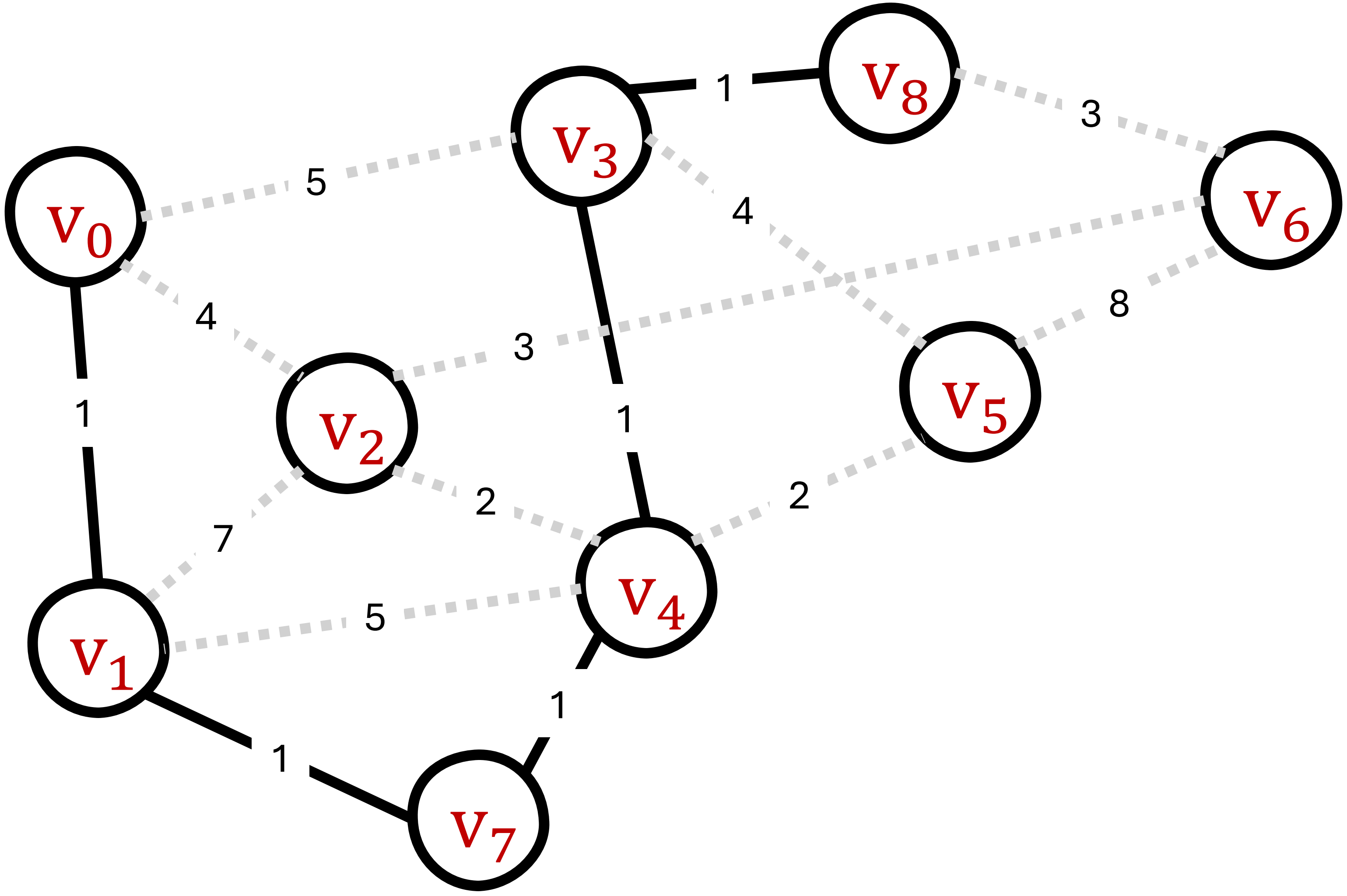
Kruskal’s Algorithm



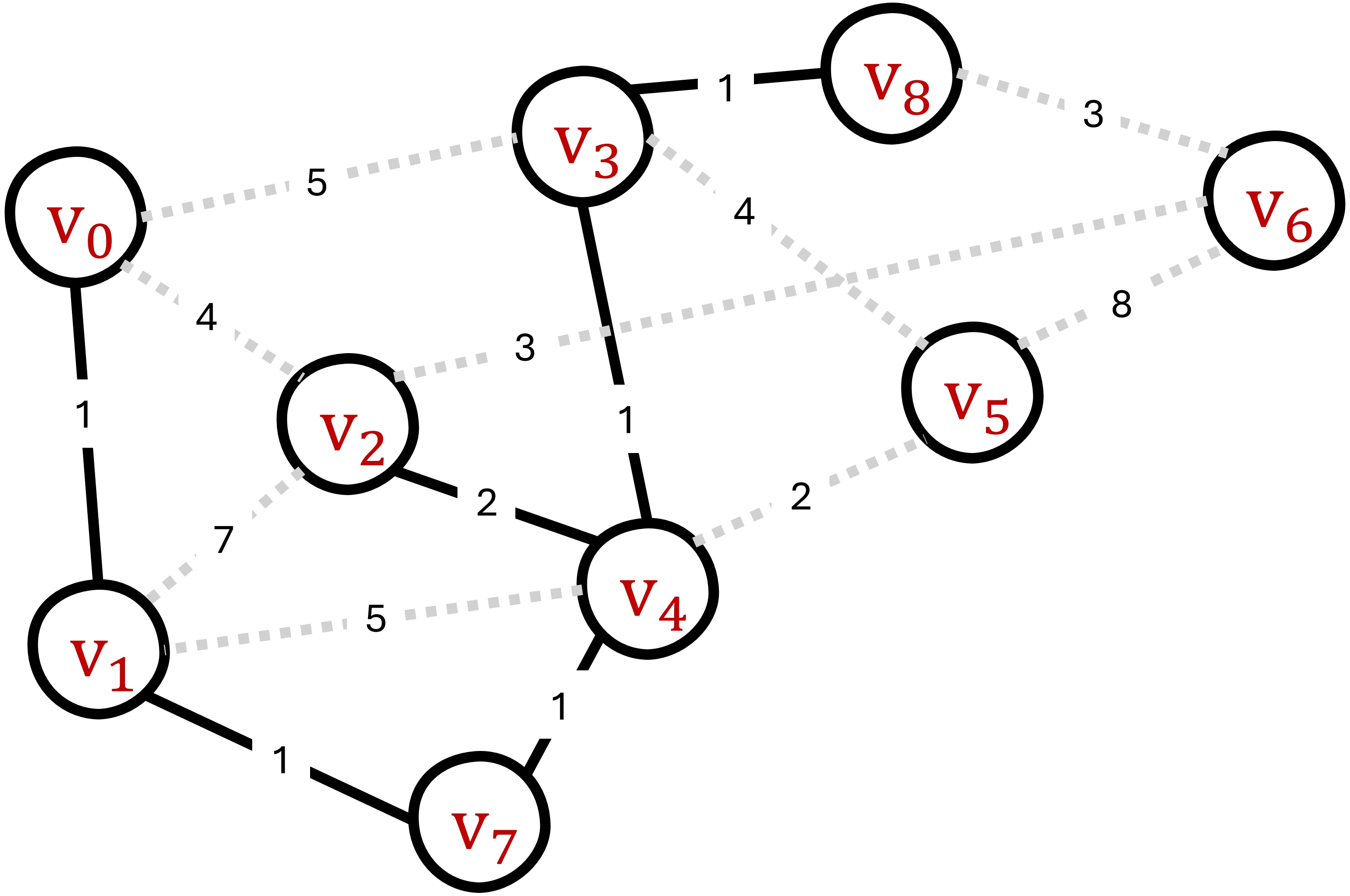
Kruskal’s Algorithm



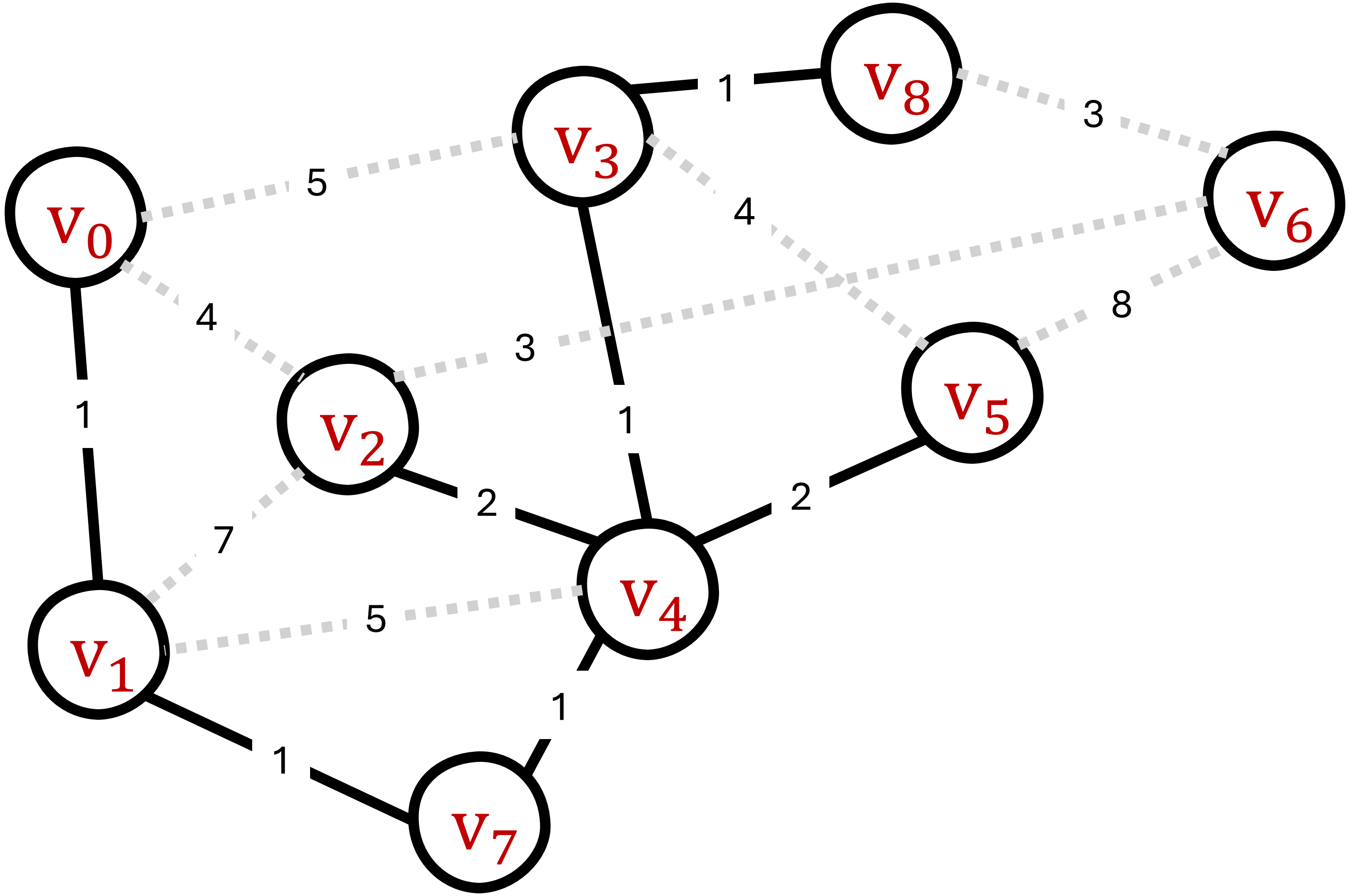
Kruskal's Algorithm



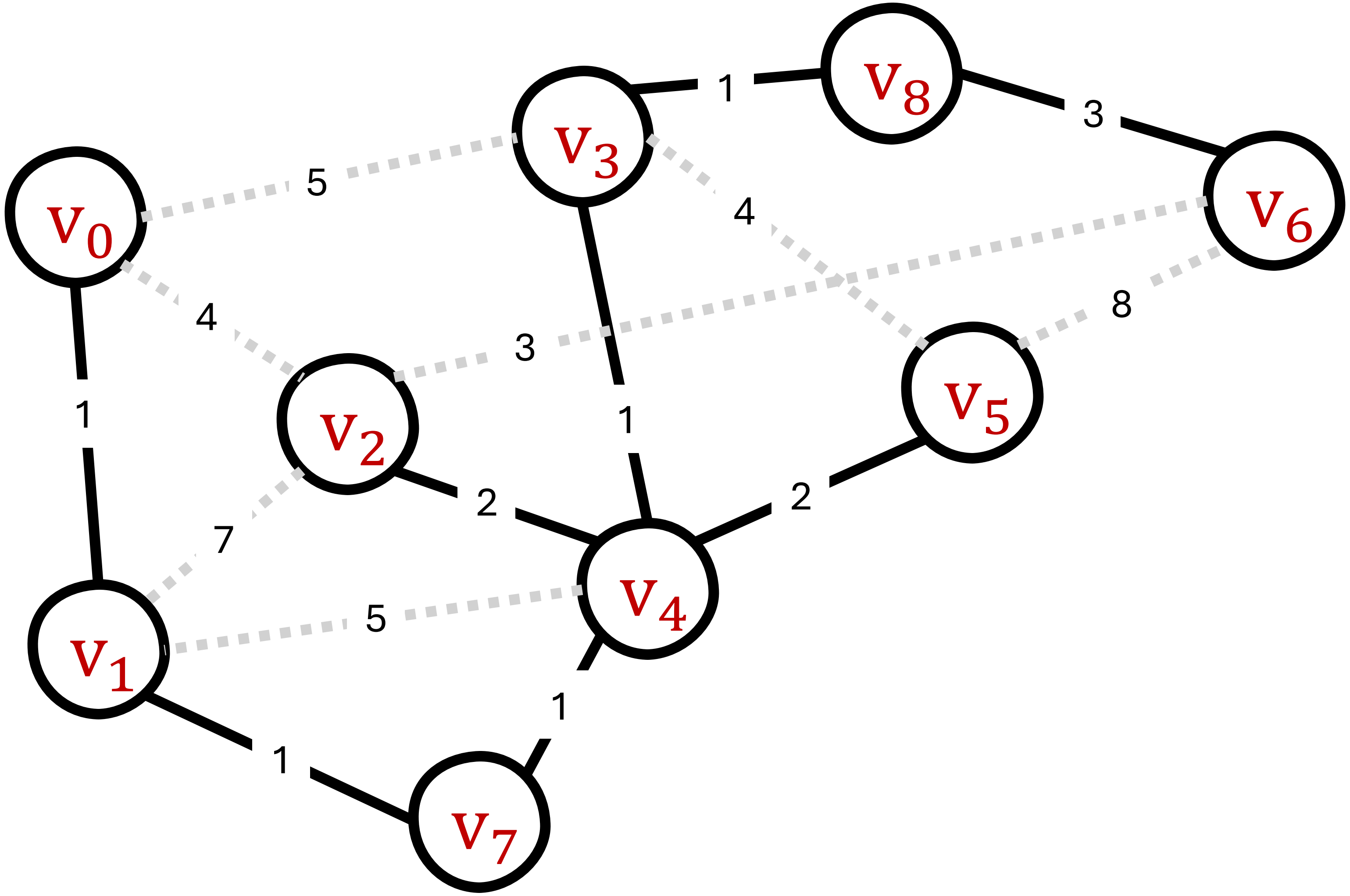
Kruskal's Algorithm



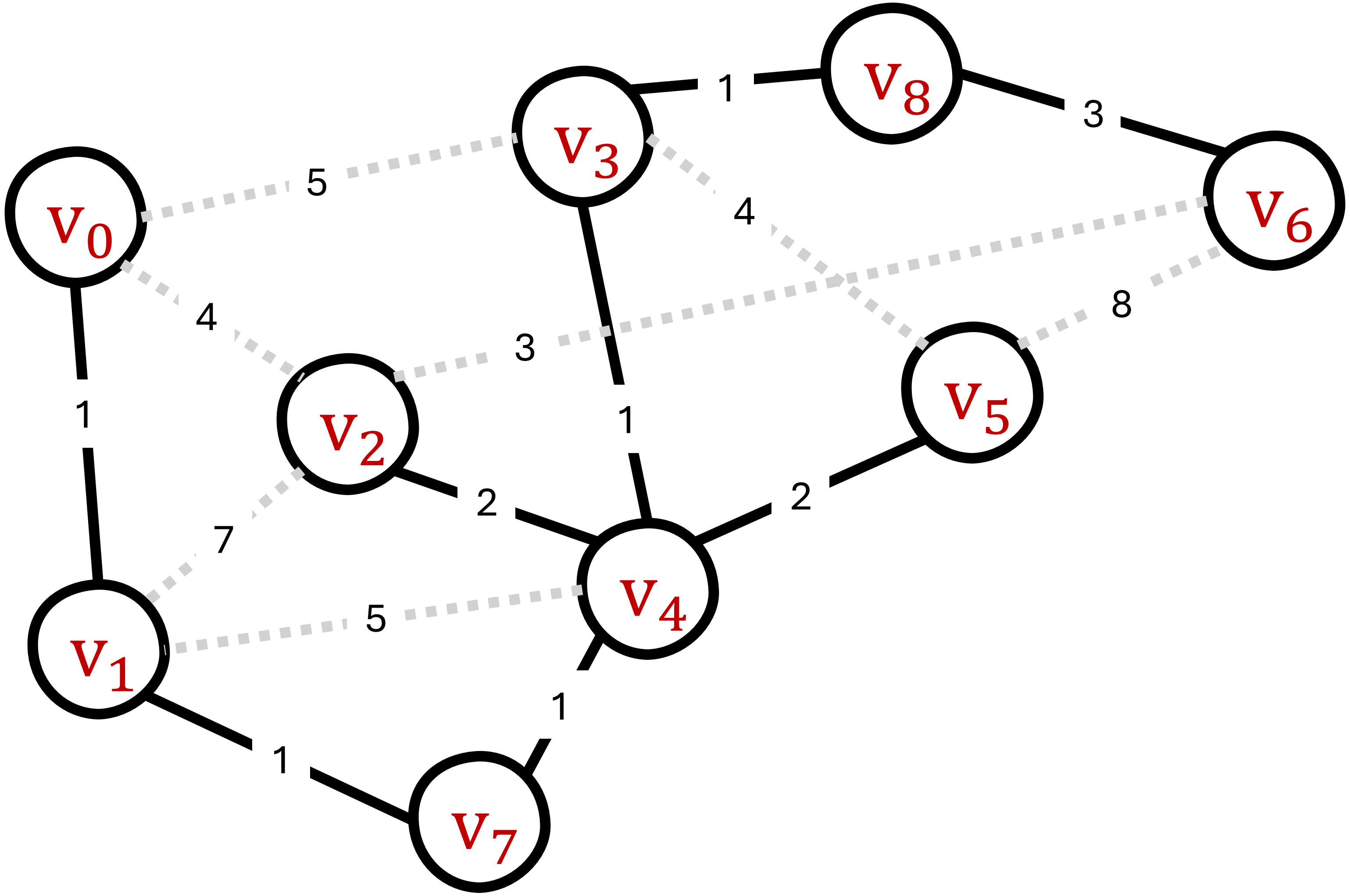
Kruskal's Algorithm



Kruskal's Algorithm

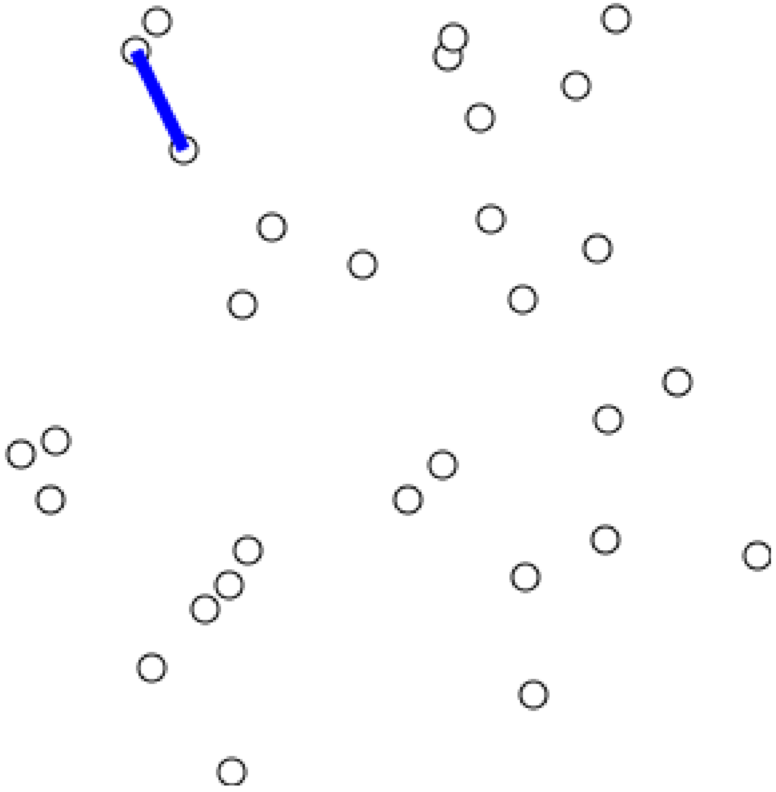


Kruskal’s Algorithm



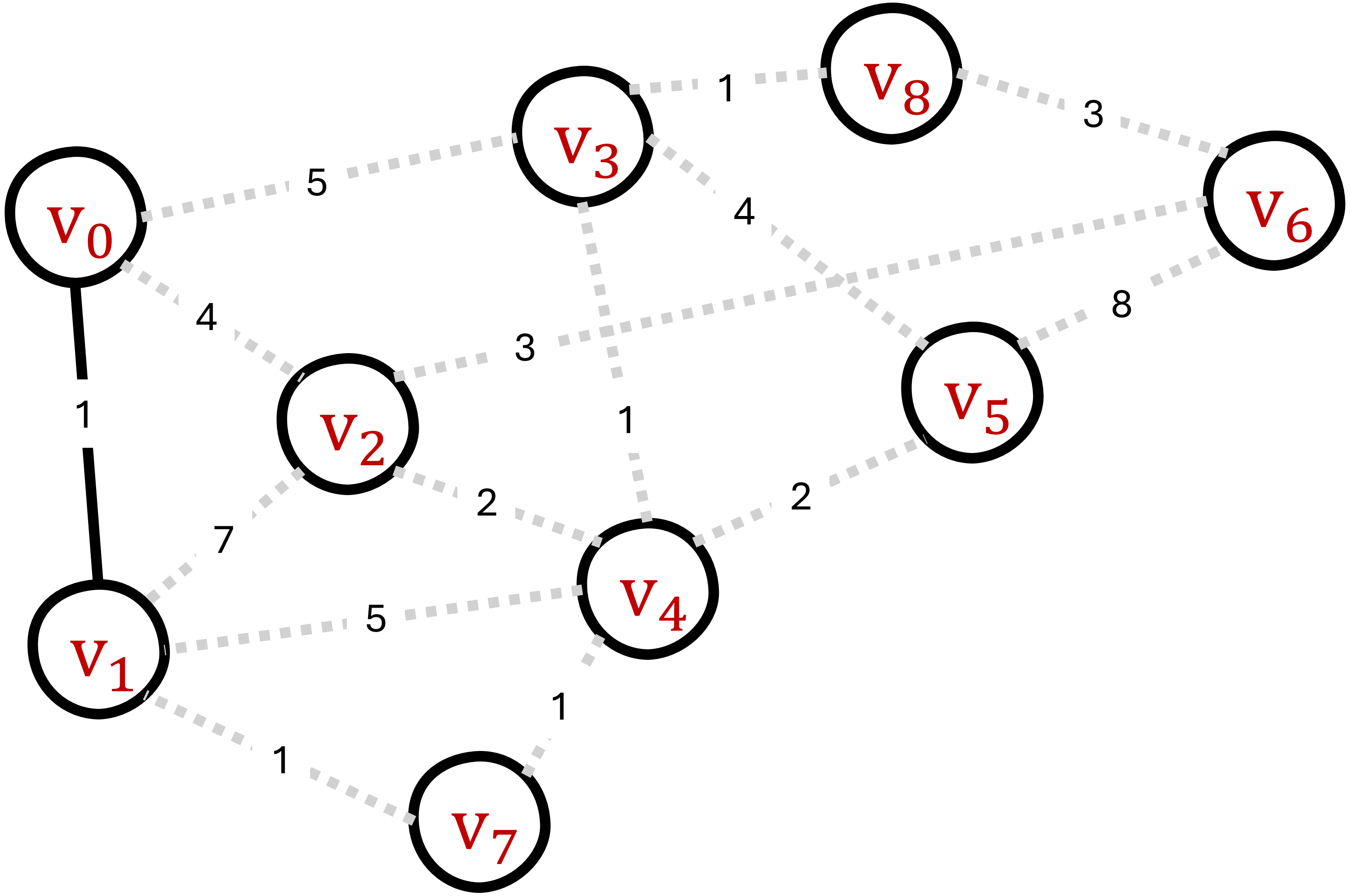
Prim's Algorithm

- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Pick s in V arbitrarily
  - Let S = {s}
  - While S != V:
    - Find minimum weight edge e = (u,v) where u is in S but v is not.
    - Add v to S

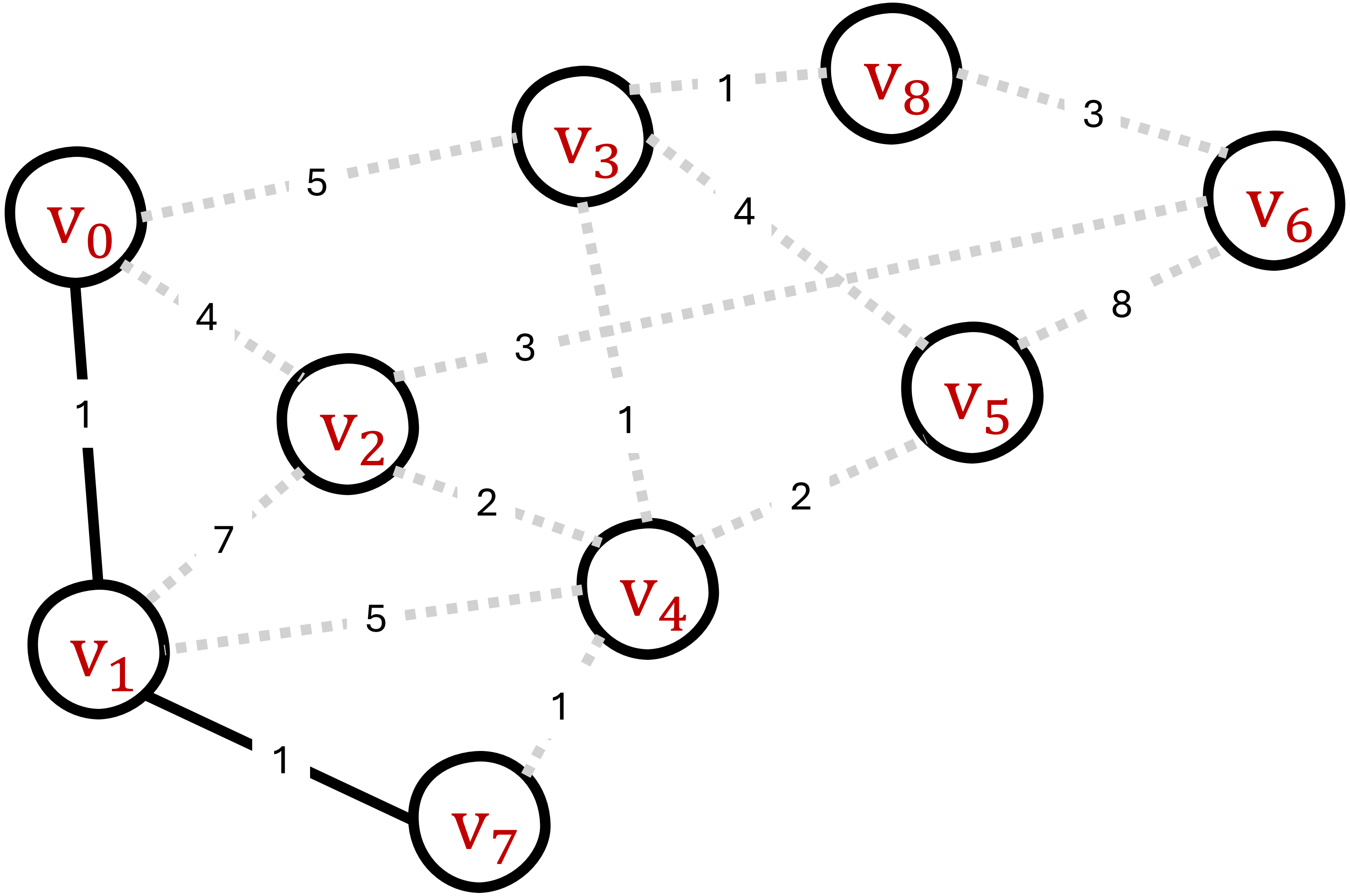


https://en.wikipedia.org/wiki/File:PrimAlgDemo.gif

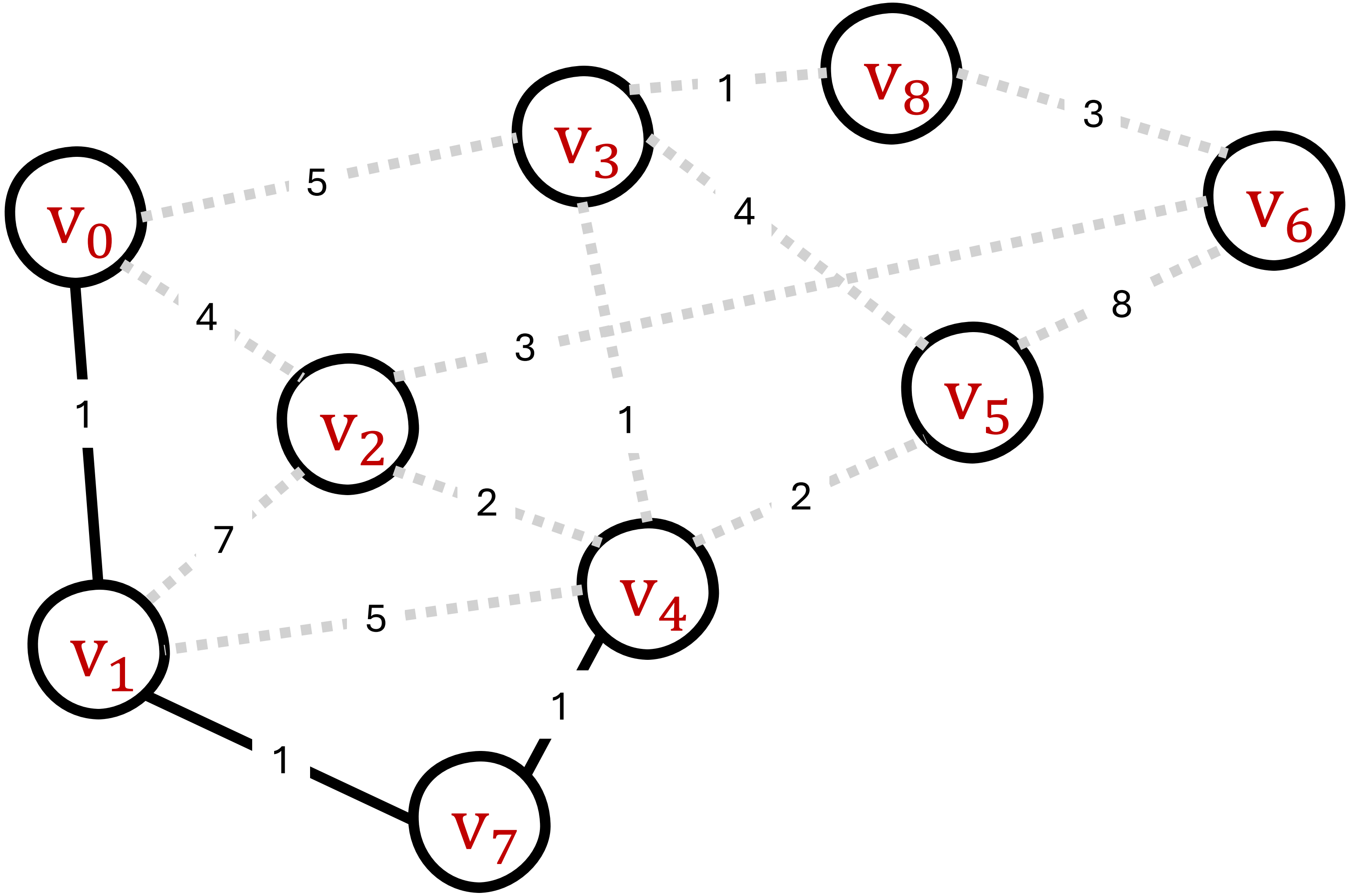
Prim's Algorithm



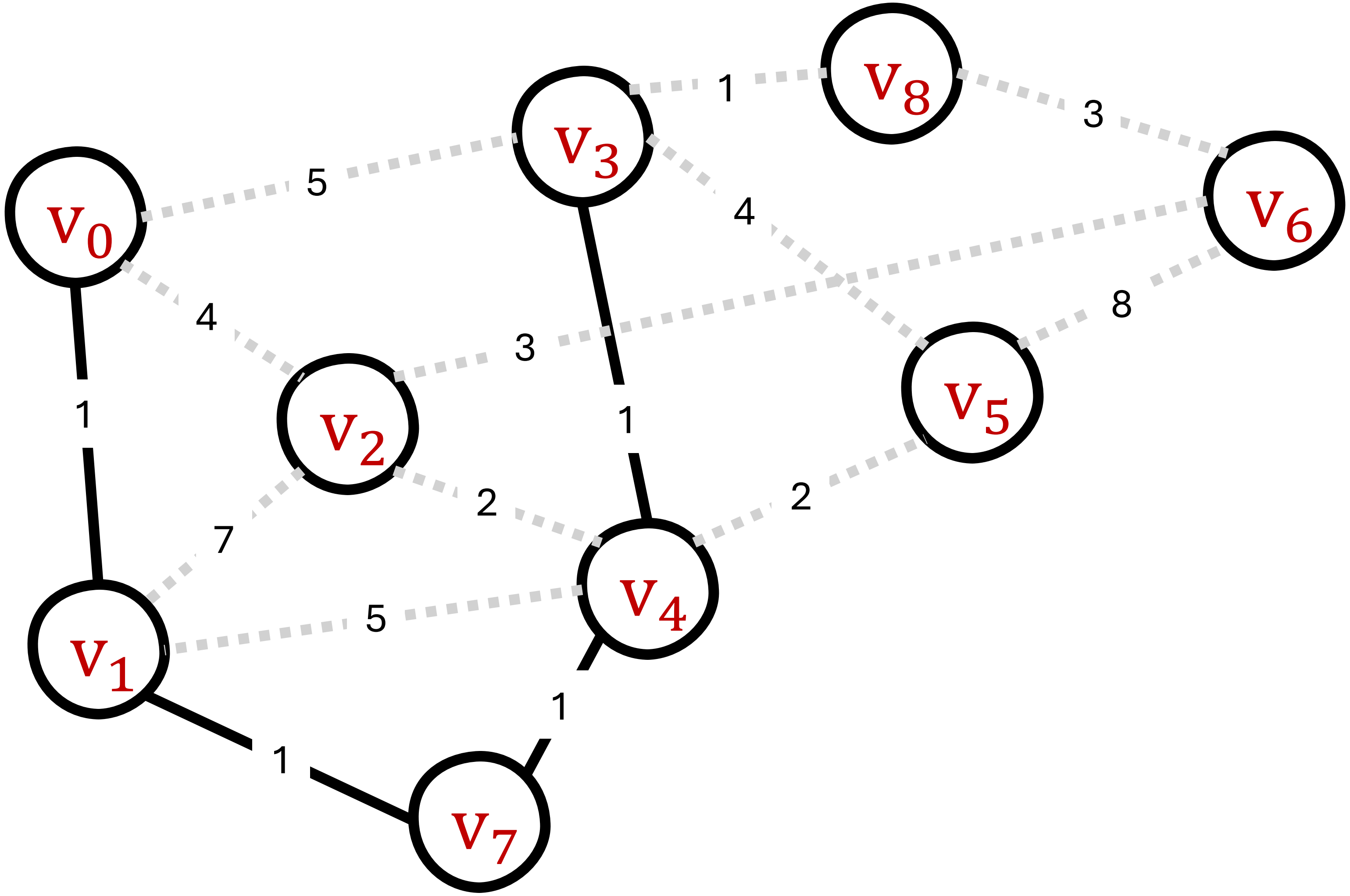
Prim’s Algorithm



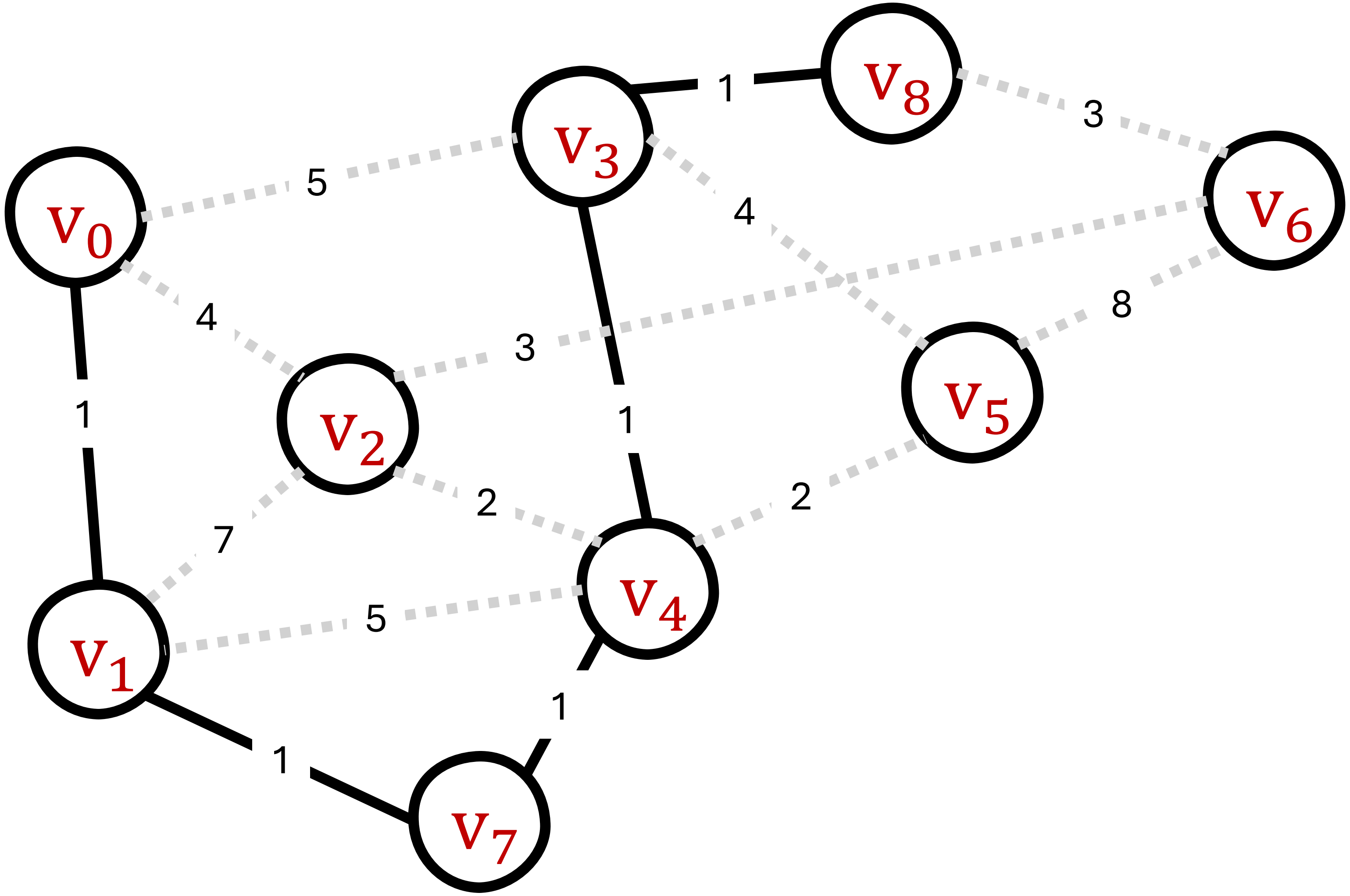
Prim's Algorithm



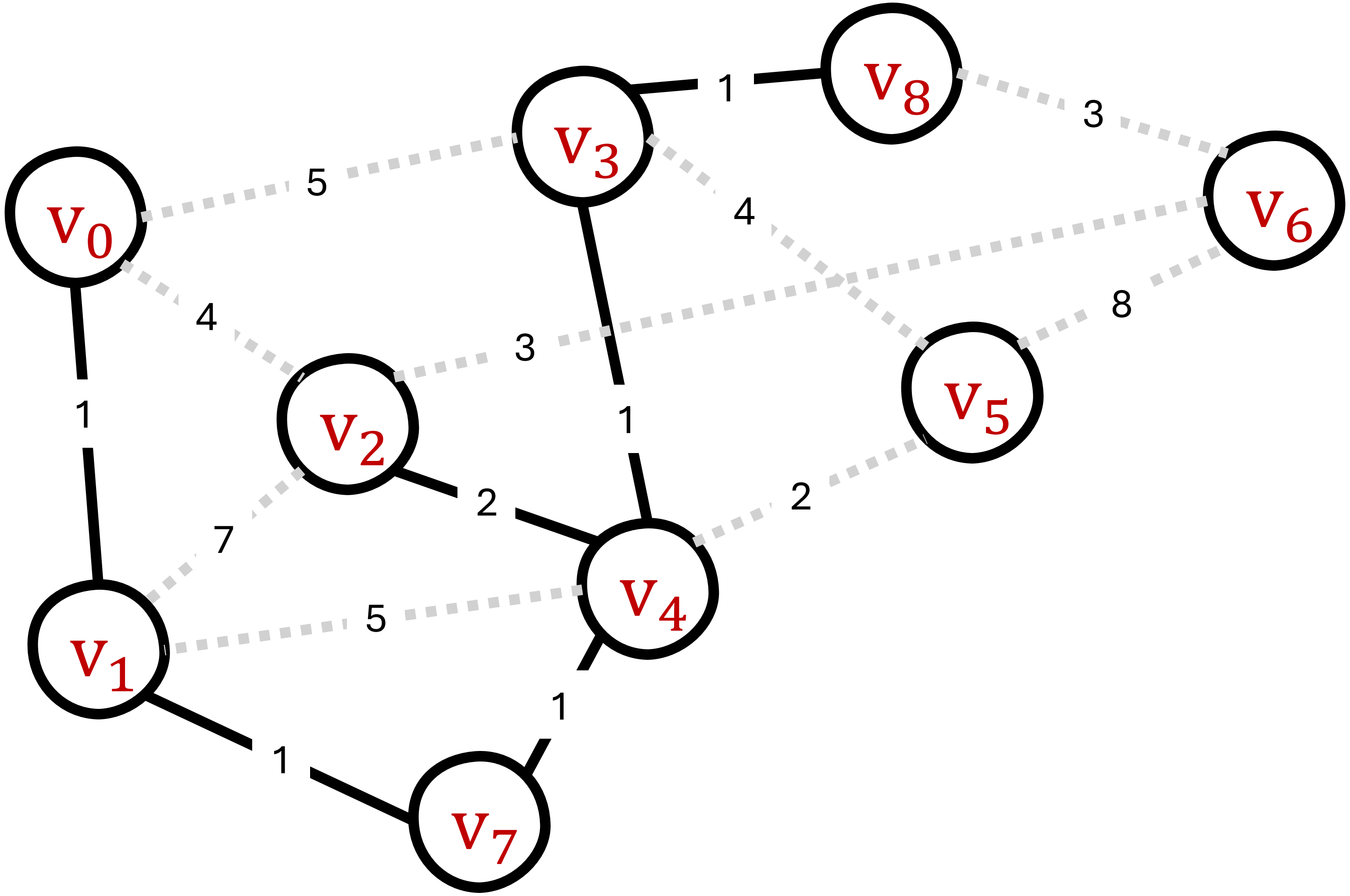
Prim's Algorithm



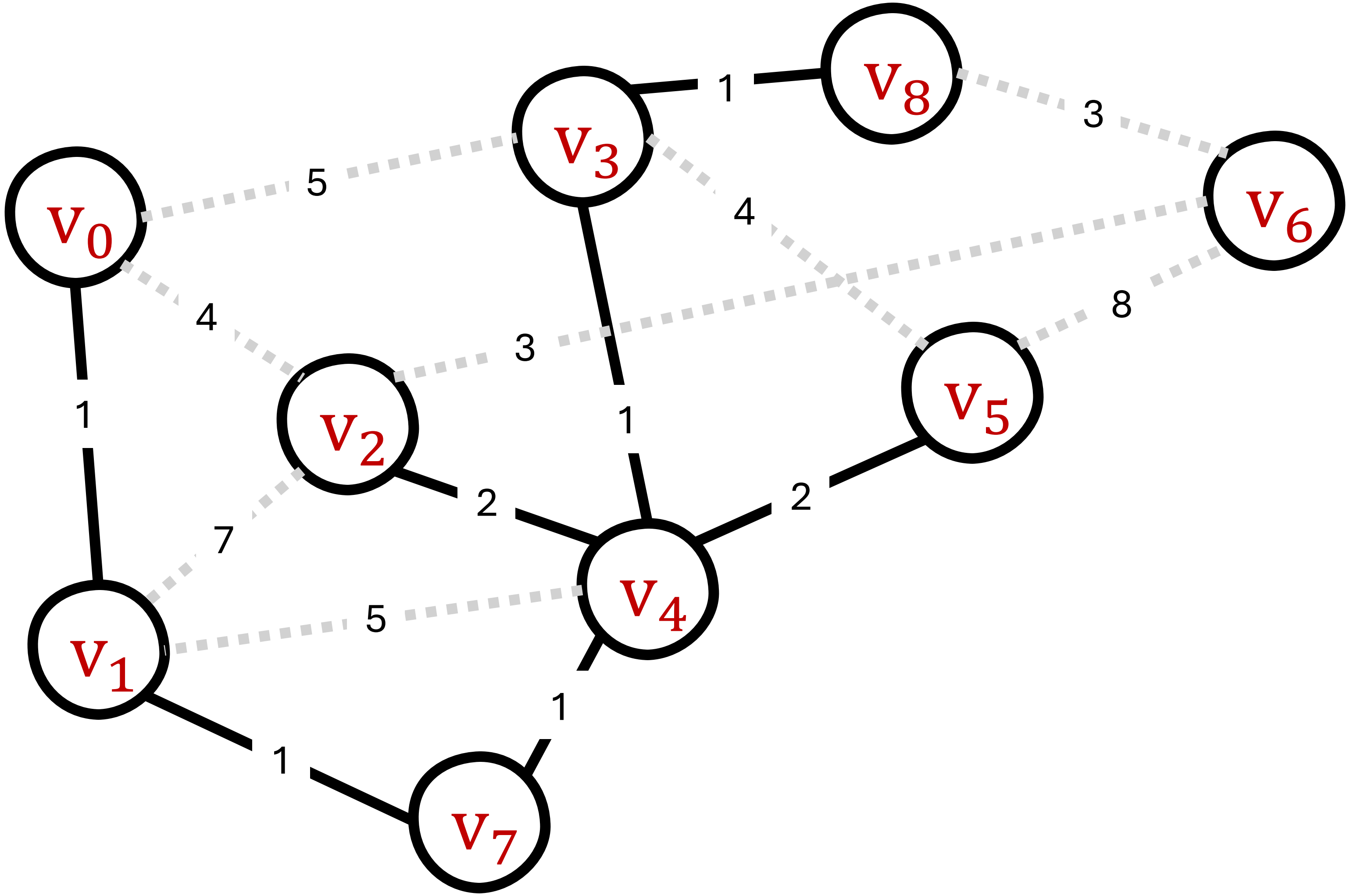
Prim's Algorithm



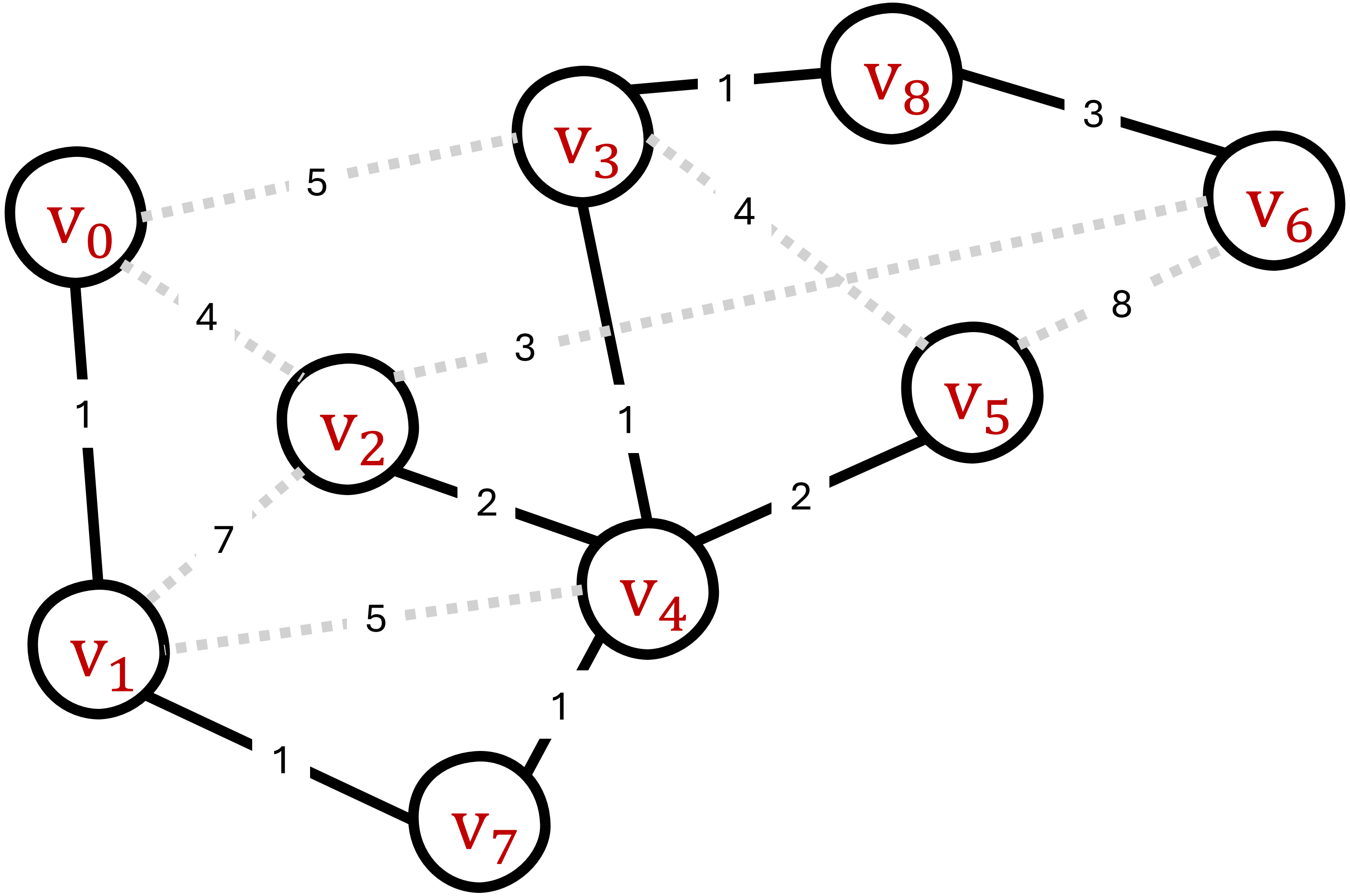
Prim's Algorithm



Prim's Algorithm



Prim’s Algorithm



Other Algorithms

Reverse Kruskal's

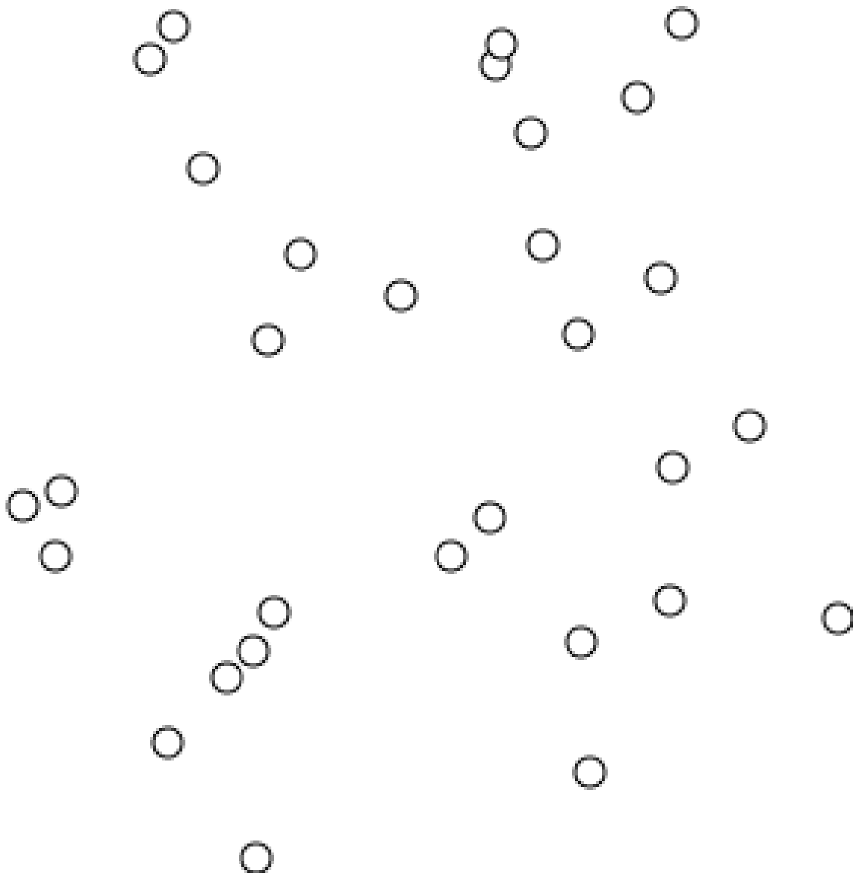
- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
    - Sort E using values in L
        - Break ties arbitrarily
    - Let T be a copy of G
    - For e in E backwards:
        - If removing e from T doesn't disconnect the graph, remove it.

Borůvka's

- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
    - Let T be an empty graph
    - For c in CC(T):
        - Find edge e leaving c with smallest weight
        - Add e to T

Kruskal's Algorithm

- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Sort E using values in L
    - Break ties arbitrarily
  - Let T be an empty graph
  - For e in E:
    - If adding e to T doesn't case a cycle, add it.



https://en.wikipedia.org/wiki/File:KruskalDemo.gif

Claim: Kruskal's Algorithm is Correct

Proof:

- Let e = (u,v) be an edge added by Kruskal's algorithm
- Consider the T just before adding e.
  - Let S be the connected component of T that contains u.
- Then e was the minimum weight edge leaving S and by the Cut Property it must be in the MST.
- Hence, Kruskal's algorithm only adds edges that must be in the MST.

Claim: Kruskal’s Algorithm is Correct

Proof:

- Let e = (u,v) be an edge added by Kruskal’s algorithm
- Consider the T just before adding e.
  - Let S be the connected component of T that contains u.
- Then e was the minimum weight edge leaving S and by the Cut Property it must be in the MST.
- Hence, Kruskal’s algorithm only adds edges that must be in the MST.
- Finally, we note that if T was not connected then there would have been edge that could have been added without forming a cycle.
- It follows that T is the MST at the end of the algorithm.

Claim: Prim's Algorithm is Correct

Proof:

- Let e = (u,v) be an edge added by Kruskal's algorithm
- Consider the T just before adding e.
  - Let S be the connected component of T that contains u.
- Then e was the minimum weight edge leaving S and by the Cut Property it must be in the MST.
- Hence, Prim's algorithm only adds edges that must be in the MST.
- Finally, we note that if T was not connected then there would have been edge that could have been added without forming a cycle.
- It follows that T is the MST at the end of the algorithm.