Middlebury

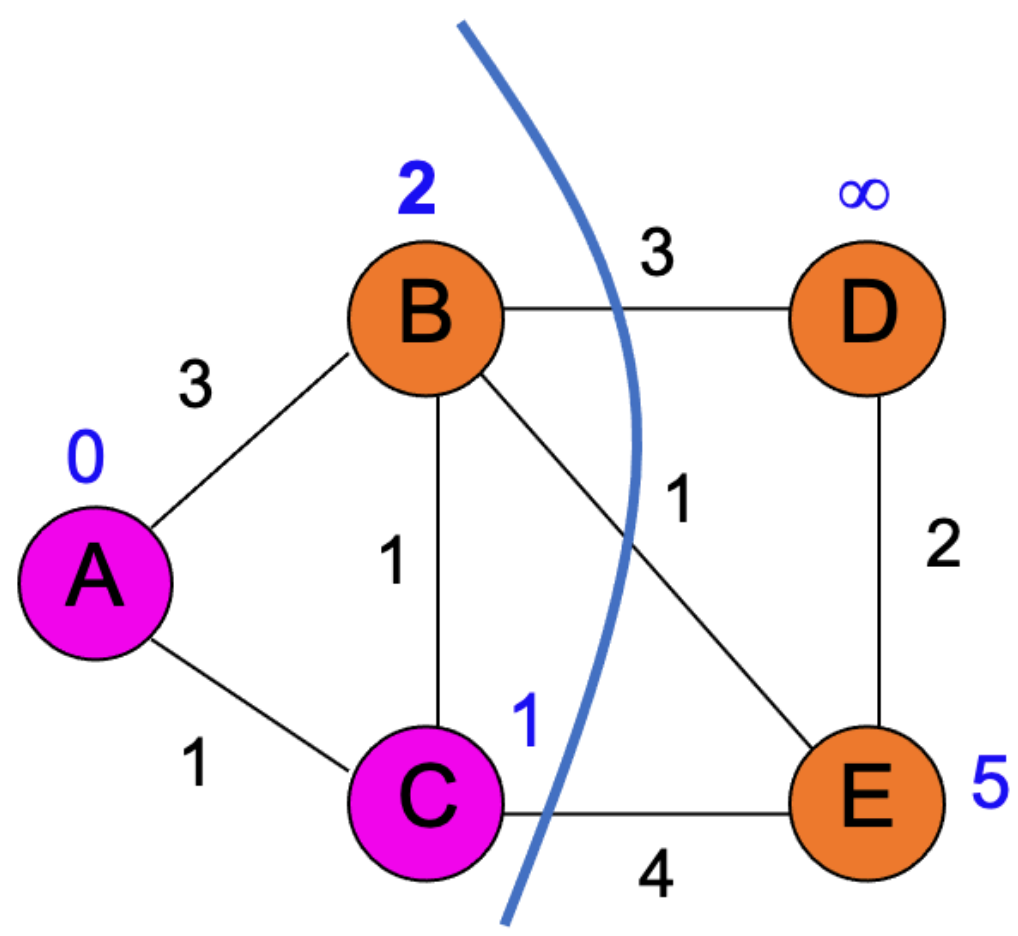
CSCI 201: Data Structures

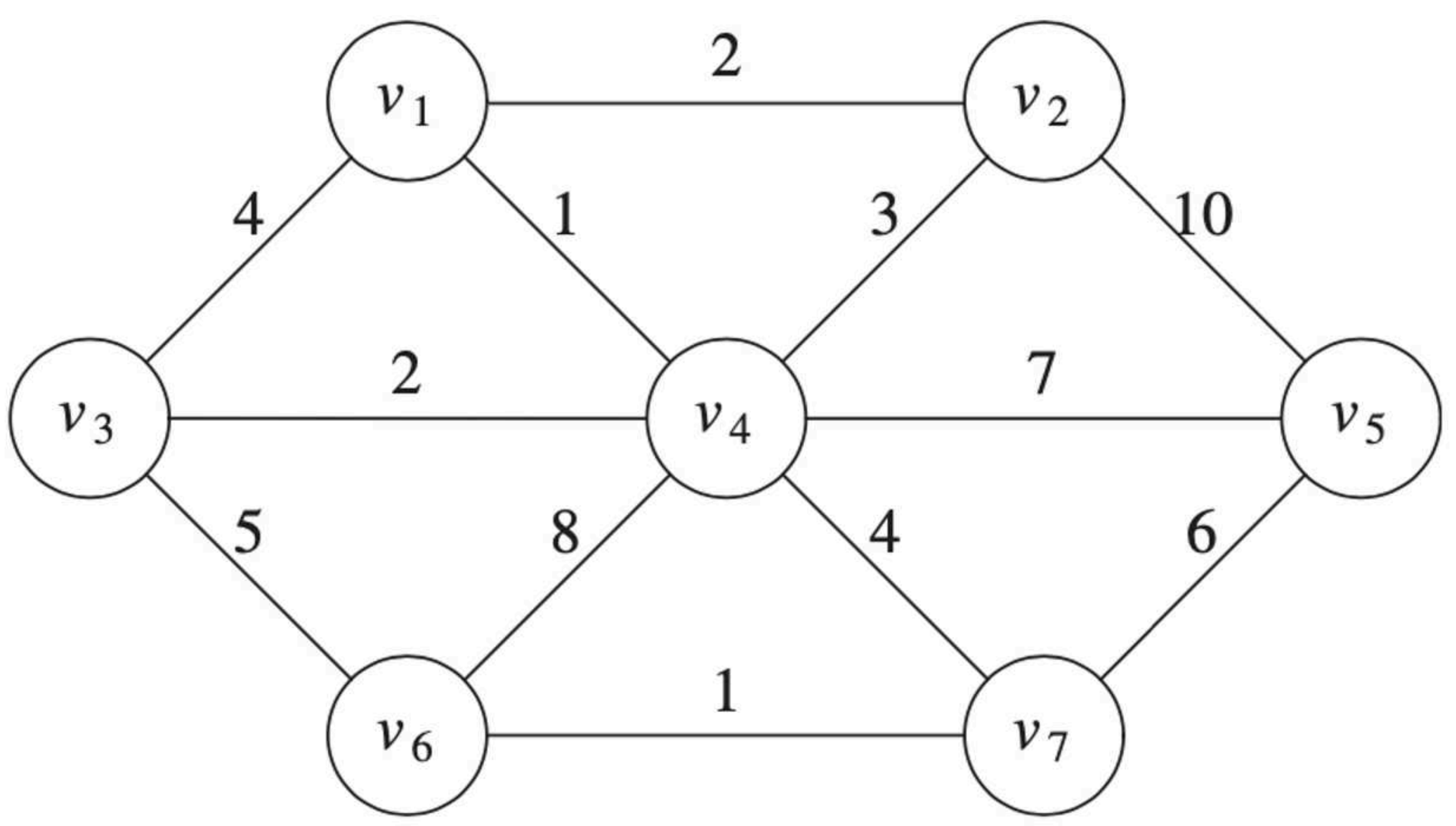
Spring 2025

Lecture 12M: Graph Algorithms

Today - Graph Algorithms

- **Single-source shortest paths:**
  - Dijkstra's algorithm
- **Minimum spanning trees:**
  - Kruskal's algorithm
  - Prim's algorithm





See Slido # 4086932

CS 201 Lecture 22

Which of the following are true? 26

Allowed answers: 2

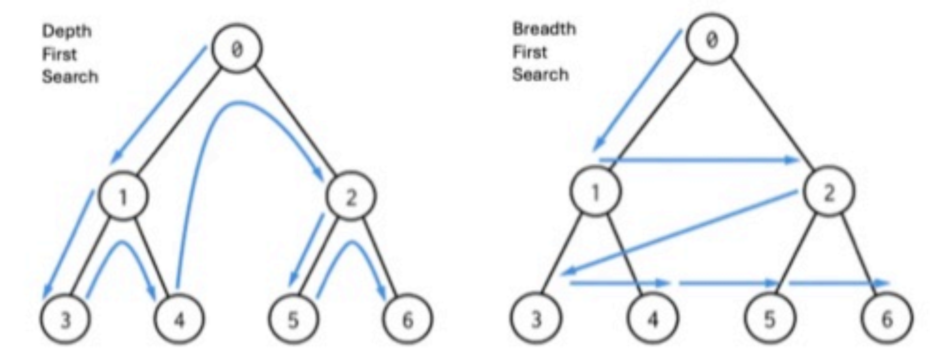
- [ ] In DFS, nodes are visited in FIFO order
- [x] In DFS, nodes are visited in LIFO order
- [x] In BFS, nodes are visited in FIFO order
- [ ] In BFS, nodes are visited in LIFO order

Send

Voting as Anonymous

slido

Acceptable Use - Slido Privacy - Cookie Settings

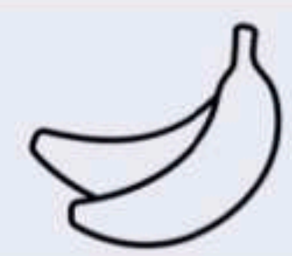


Why learn Greedy Algorithms?

- Sometimes a greedy algorithm is optimal
  - Huffman Compression
  - Prim's Minimum Spanning Tree
  - Kruskal's Minimum Spanning Tree
- Sometimes the greedy algorithm isn't provably optimal but works well in practice
- A greedy algorithm is typically easy to start with for optimization problems

Optimization

- Find the solution that maximizes or minimizes some objective
- Example: Knapsack
  - Find the collection of items with maximum value without exceeding a budget
  - What would you buy if you had $10?

Items	Value	Cost
	2	$1
	1	$1
	10	$10

Greedy algorithms

Greedily Searching for Optima

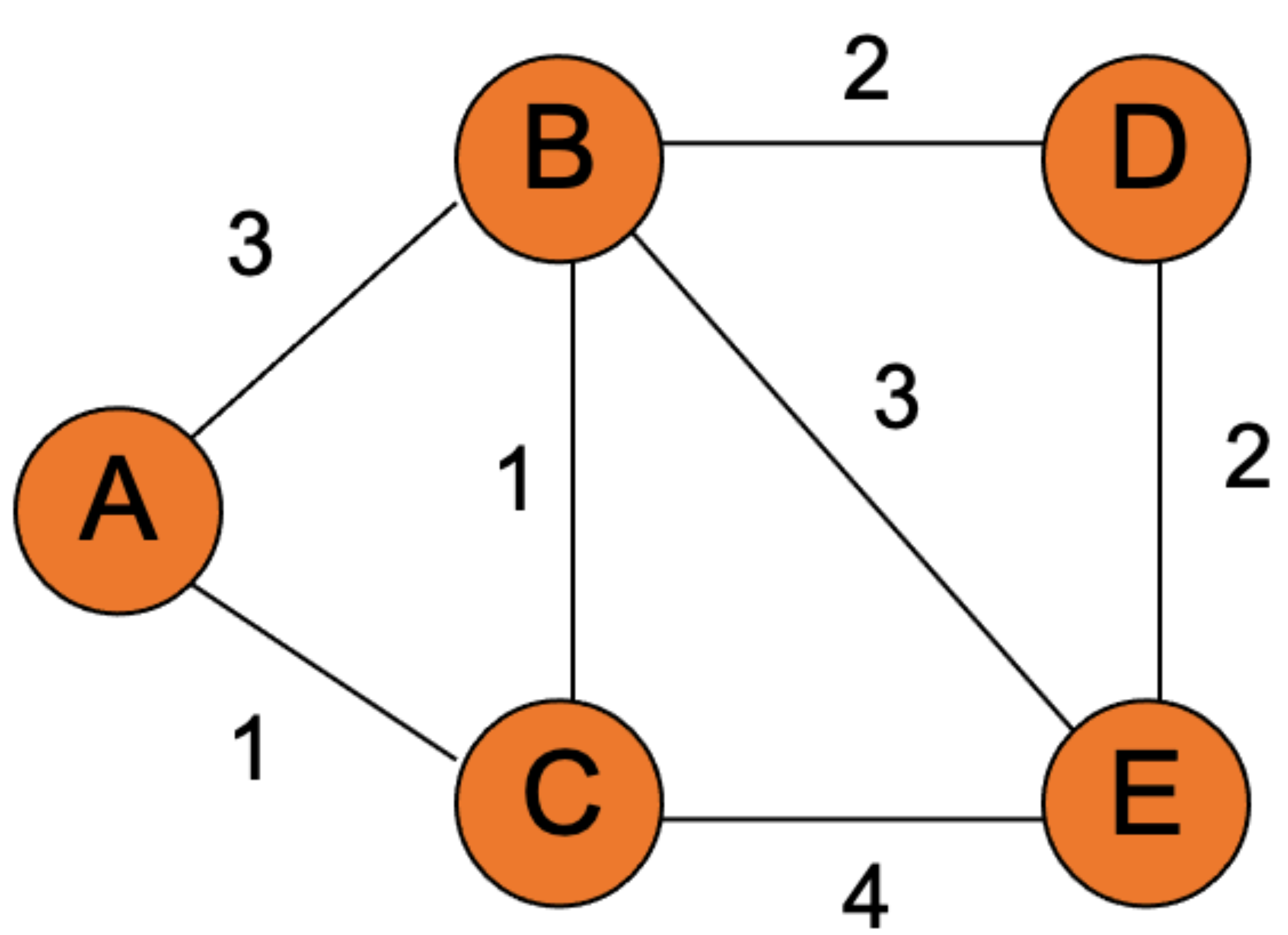
- Start with a partial solution. In each iteration make a step toward a complete solution

- Greedy principle: In each iteration, make the lowest cost or highest value step

- Knapsack:
  - Partial solution is a set of items you can afford
  - Greedy step: Add the next best value per cost item that you can afford

Shortest paths with weighted edges

What is the shortest path from a to d?



Key Idea

Explore the vertices in order of increasing distance from the starting vertex

Keep track of the distances to each vertex

If we find a better path, update that distance

Dijkstra’s algorithm - high level

Explore the vertices in order of increasing distance from the starting vertex

Use a priority queue to keep track of the shortest path found so far to a vertex

---

Initialize: distance to start = 0 and all others infinity

```
repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v → adj)
        if path including v → adj is shortest then is best path for adj so far
            update the distance for adj
            update the priority queue
```

Dijkstra's algorithm

Initialize: distance to start = 0 and all others infinity

repeat

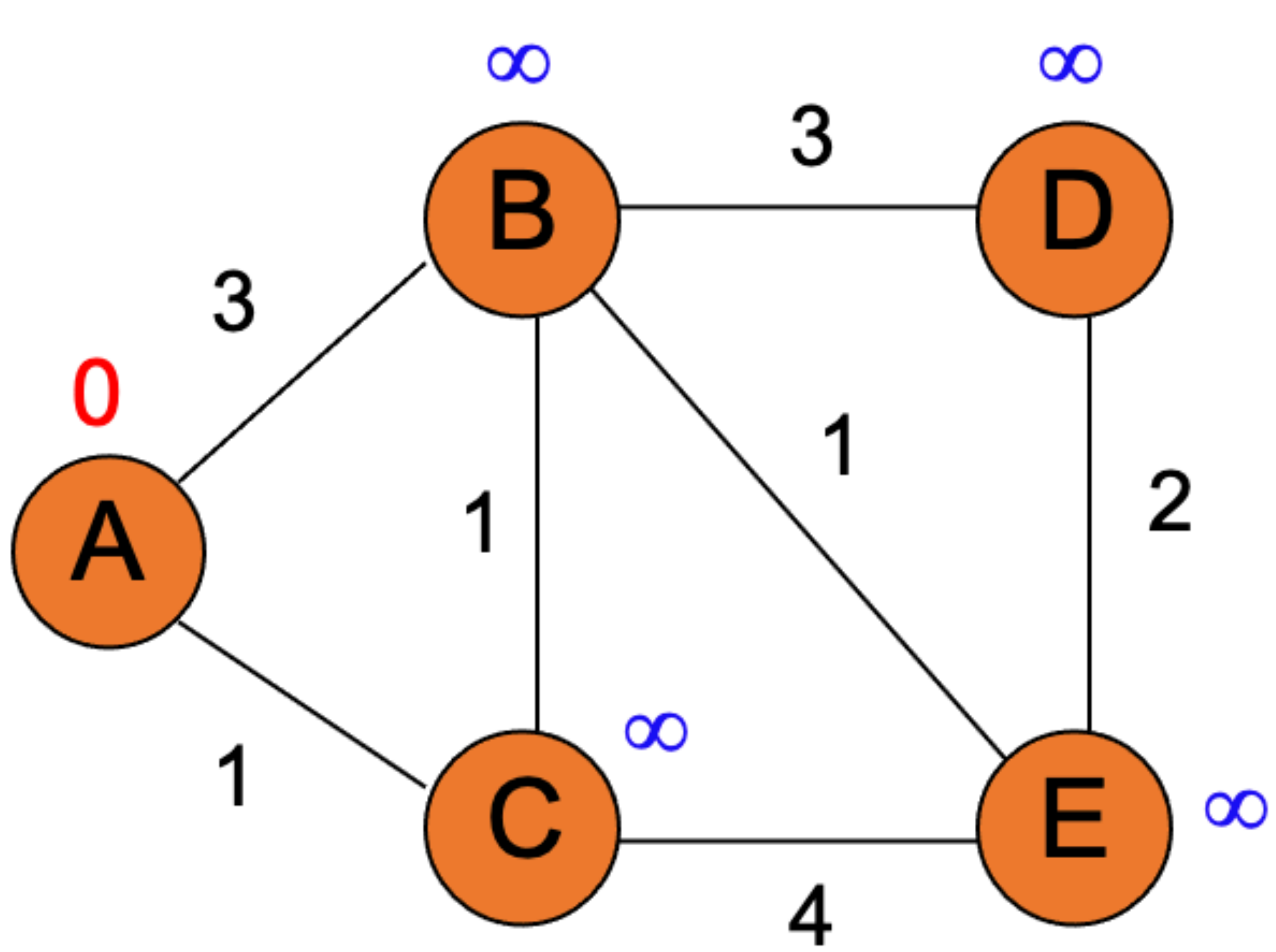
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v → adj)

    if path including v → adj is shortest then is best path for adj so far

      update the distance for adj

      update the priority queue



PQ

A 0
B ∞
C ∞
D ∞
E ∞

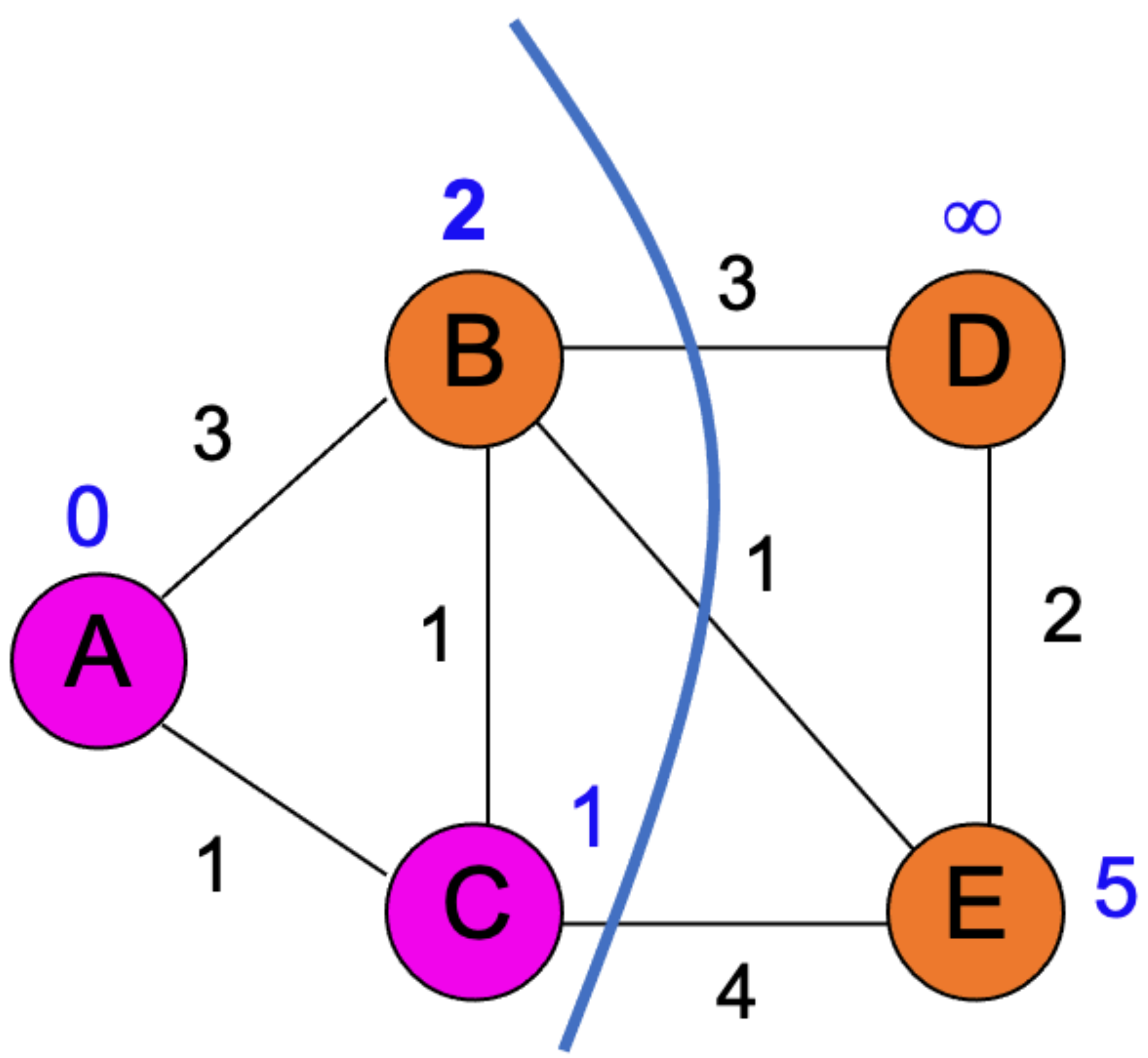
Dijkstra's algorithm

Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v → adj)
        if path including v → adj is shortest then is best path for adj so far
            update the distance for adj
            update the priority queue

PQ	
B	2
E	5
D	∞



Frontier:

All nodes reachable from starting node within a given distance

Dijkstra's algorithm

Initialize: distance to start = 0 and all others infinity

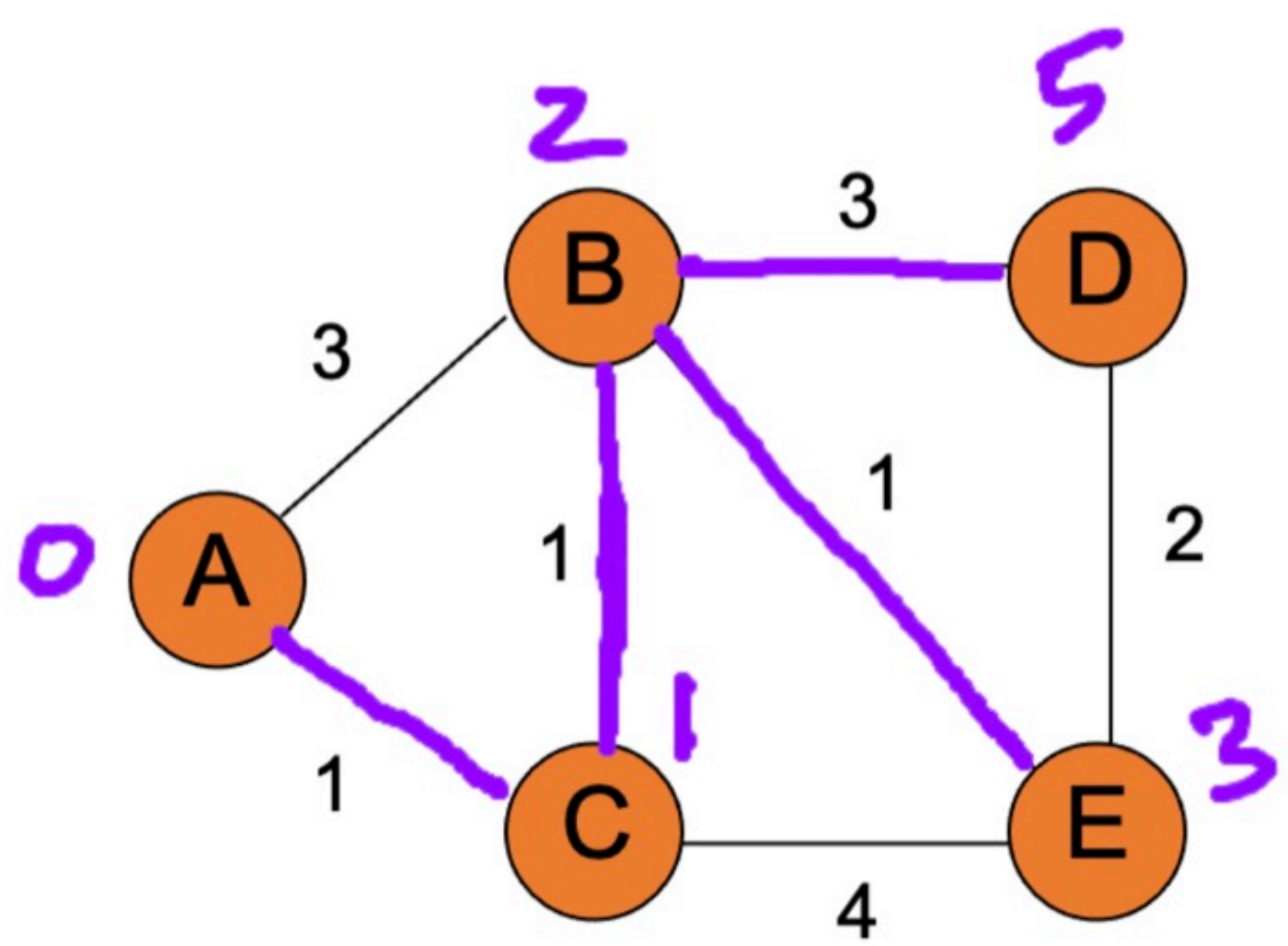
repeat
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v → adj)
    if path including v → adj is shortest then is best path for adj so far
      update the distance for adj
      update the priority queue

v	distTo(v)
A	0
B	~~3(A)~~
C	1(A)
D	5(B)
E	~~5(C)~~ 3(B)

PQ

~~A : 0~~
~~B : ∞ 3 2~~
~~C : ∞ 1~~
~~D : ∞ 5~~
~~E : ∞ 5 3~~



Why does it work?

When a vertex is removed from the priority queue, distTo[v] is the actual shortest distance from s to v

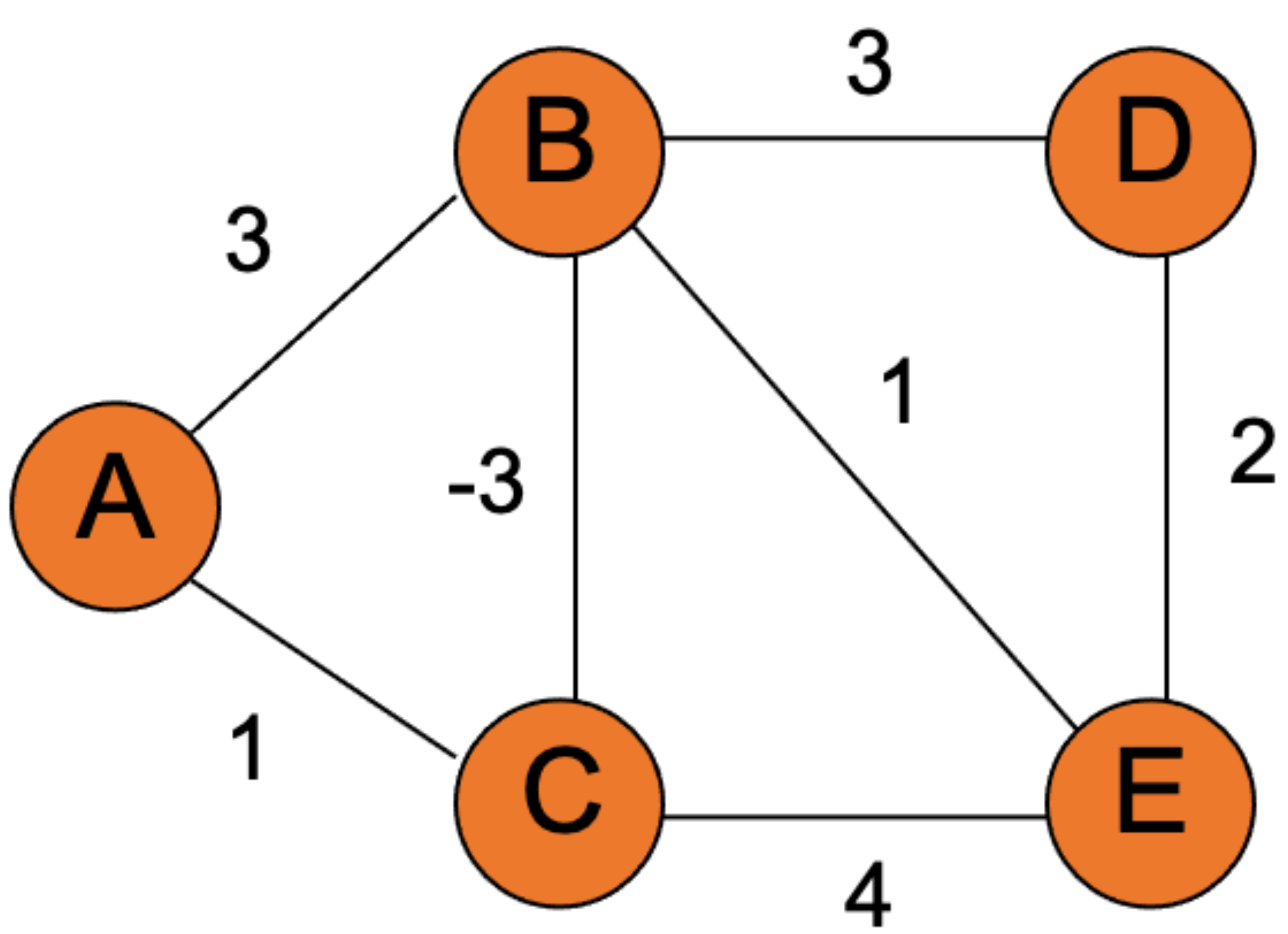
- The only time a vertex gets removed is when the distance from s to that vertex is smaller than the distance to any remaining vertex

- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

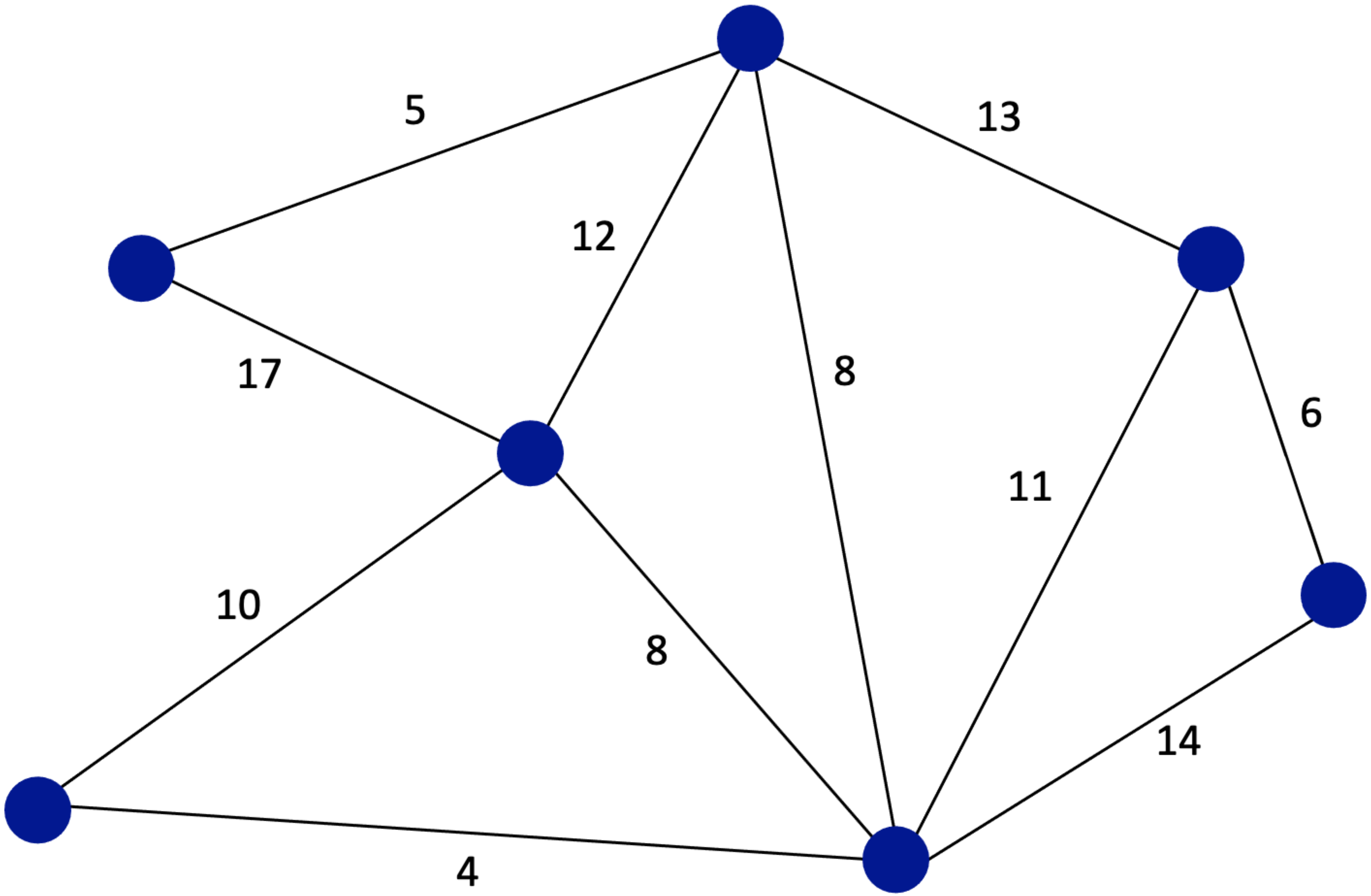
Assuming no negative edge weights!

Example graph with a negative edge weight

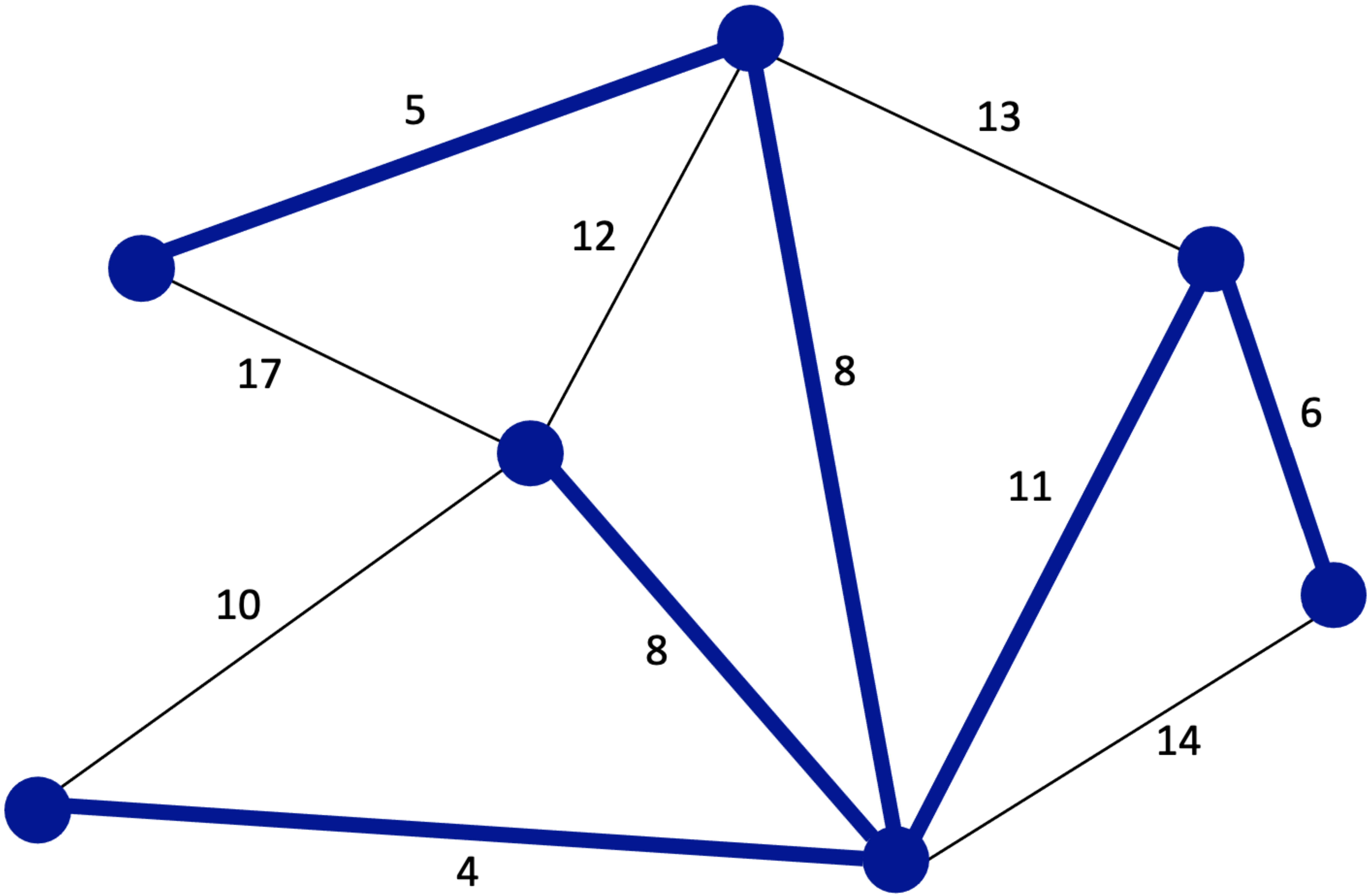
Dijkstra’s only works on graphs with positive edge weights



A weighted graph



A minimum spanning tree



Kruskal's algorithm for minimum spanning trees

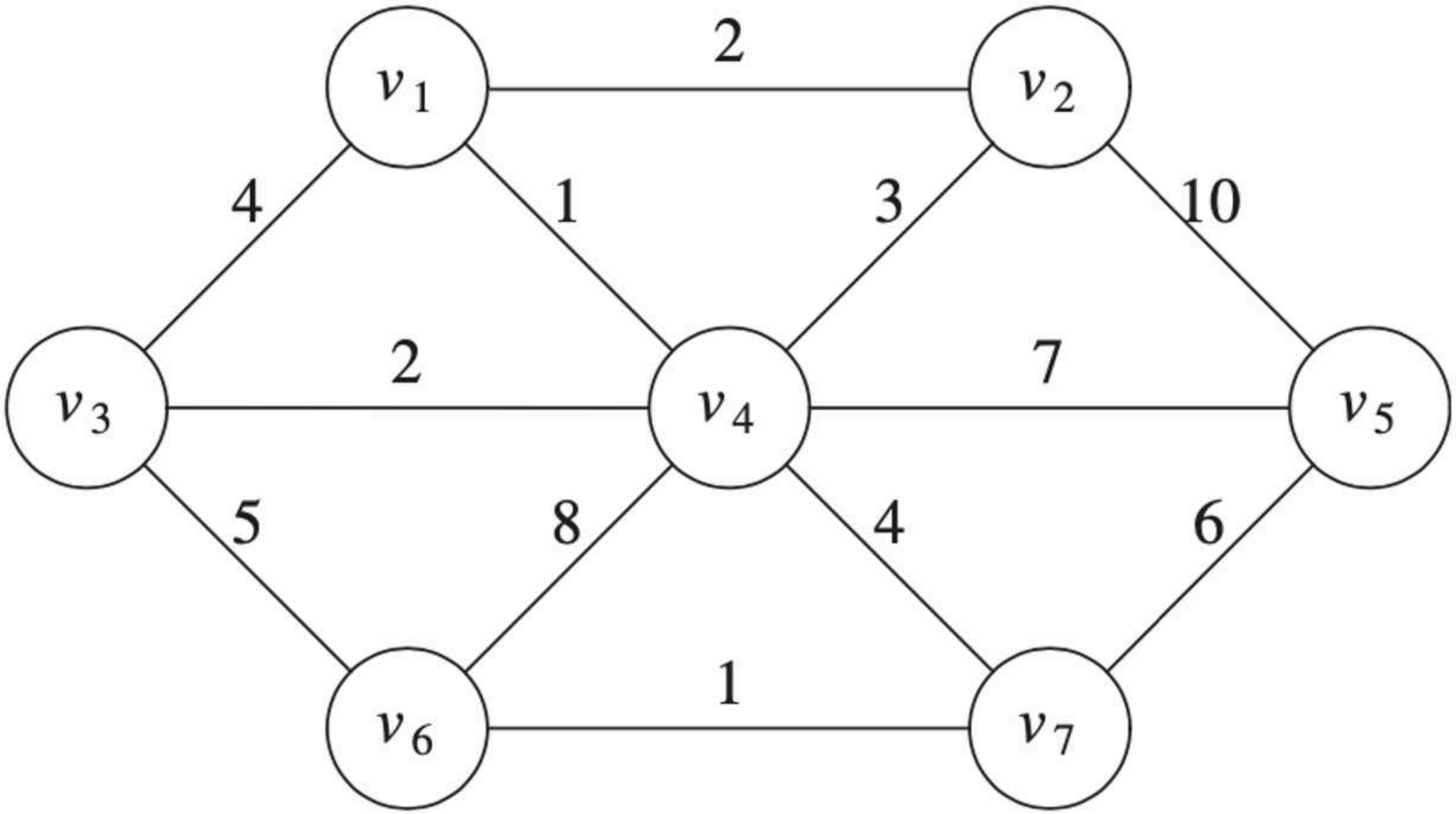
Minimum Spanning Tree

Kruskal’s Algorithm

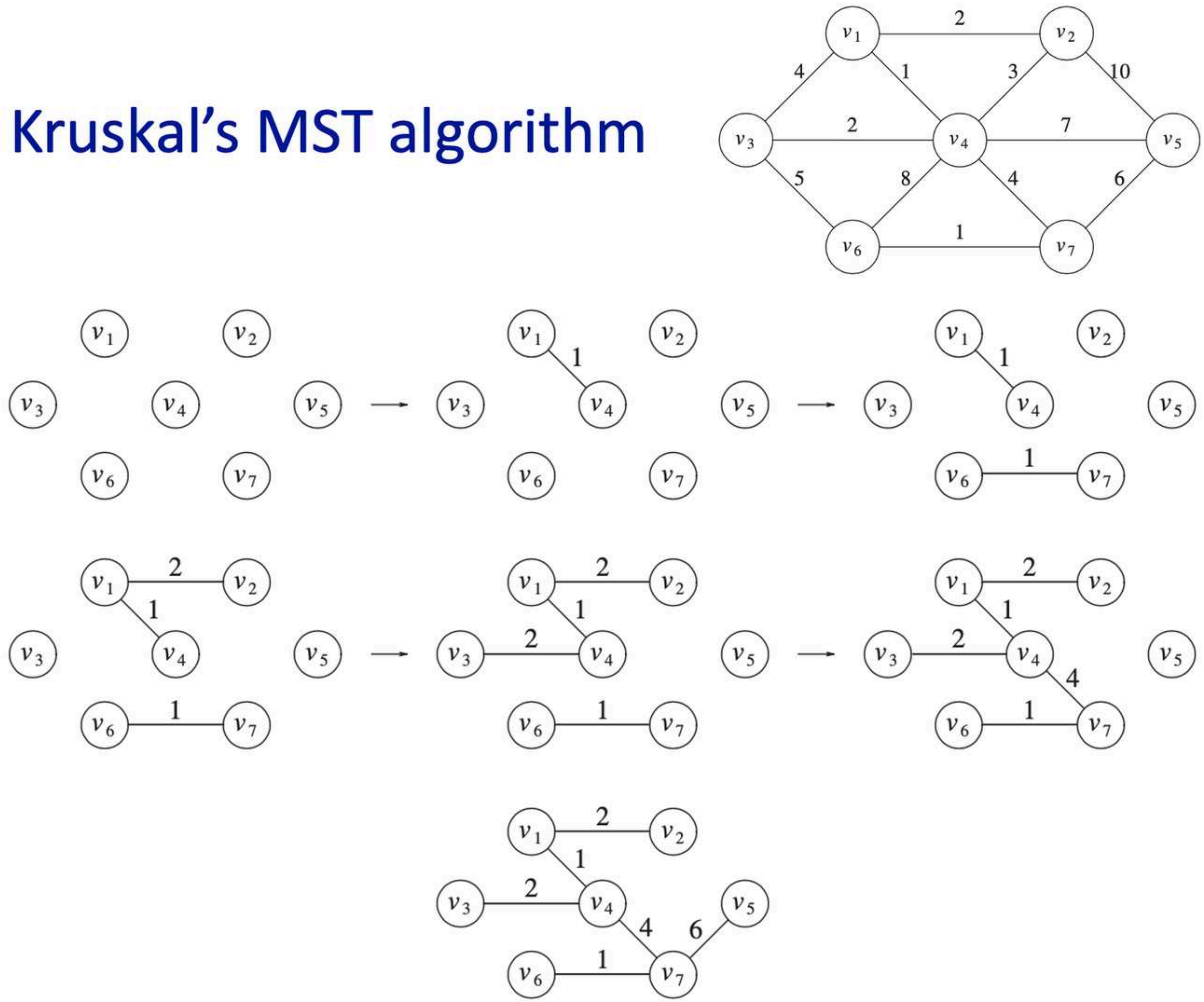
- Maintain a forest, ie, a collection of trees
- Initially there are |V| single-node trees
- Select edges in order of smallest weight and accept an edge if it does not cause a cycle
- Accepting an edge merges two trees into one

Kruskal's algorithm for minimum spanning trees

Kruskal's MST algorithm



Kruskal's algorithm for minimum spanning trees



Prim's algorithm for minimum spanning trees

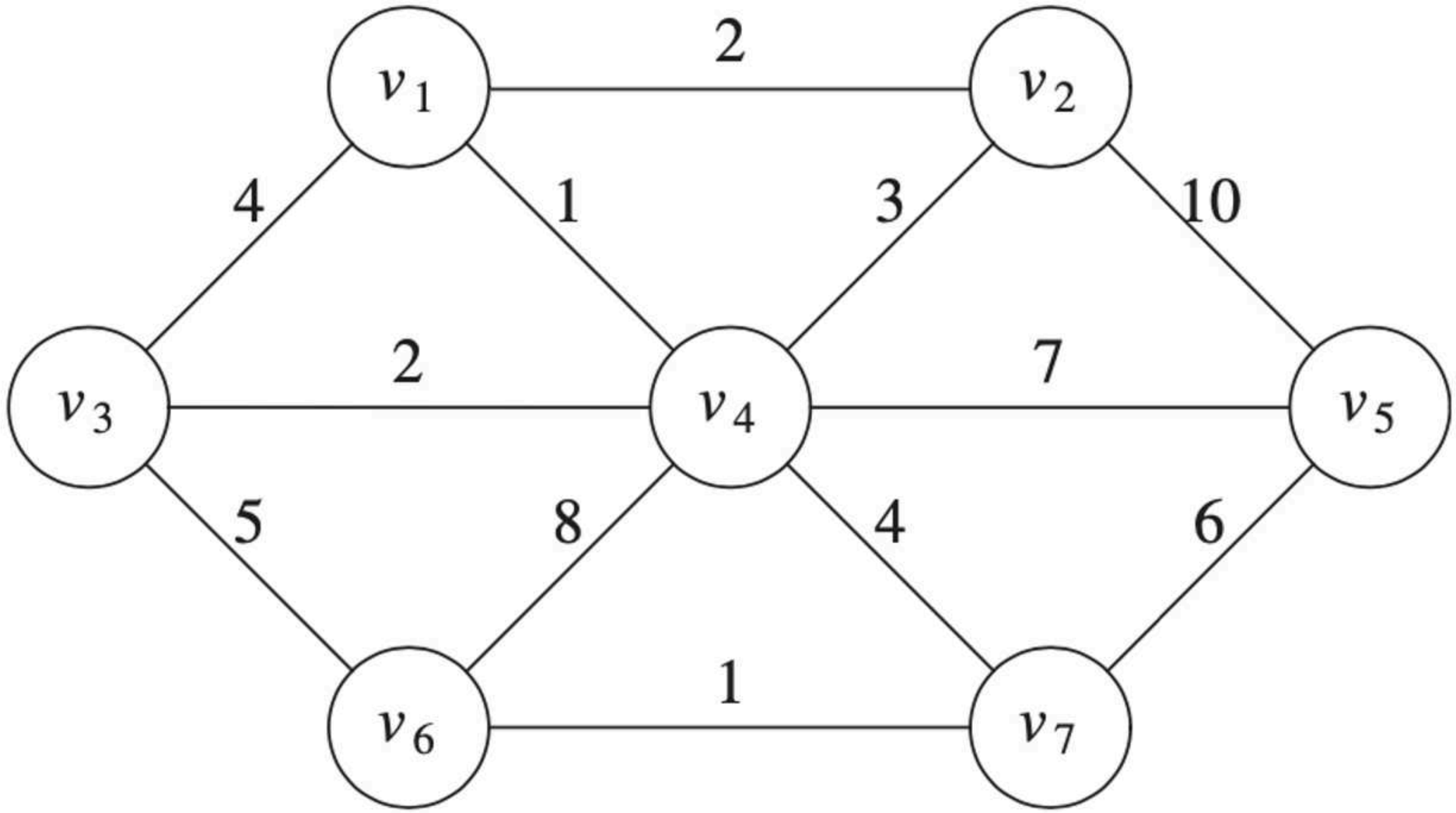
Minimum Spanning Tree

Prim’s Algorithm

- Maintain a single tree
- At each stage add an edge and a vertex
- Select edge (u, v) such that cost of (u, v) is smallest among all edges where u is in tree and v is not

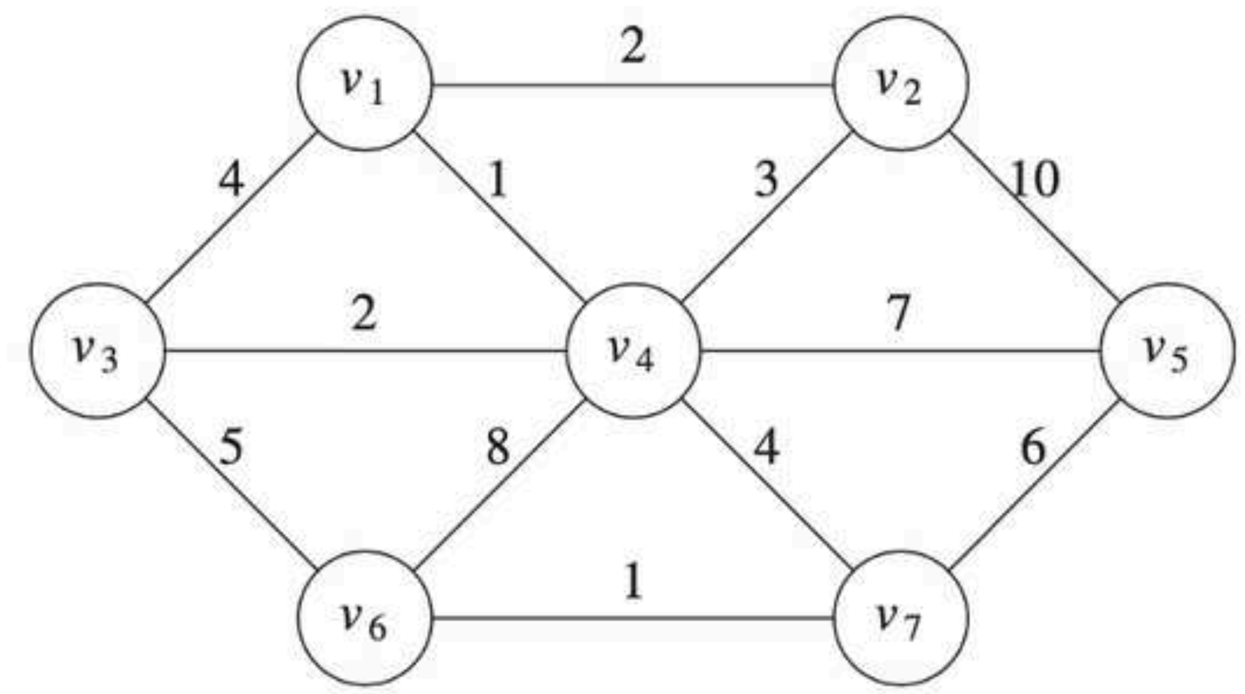
Prim's algorithm for minimum spanning trees

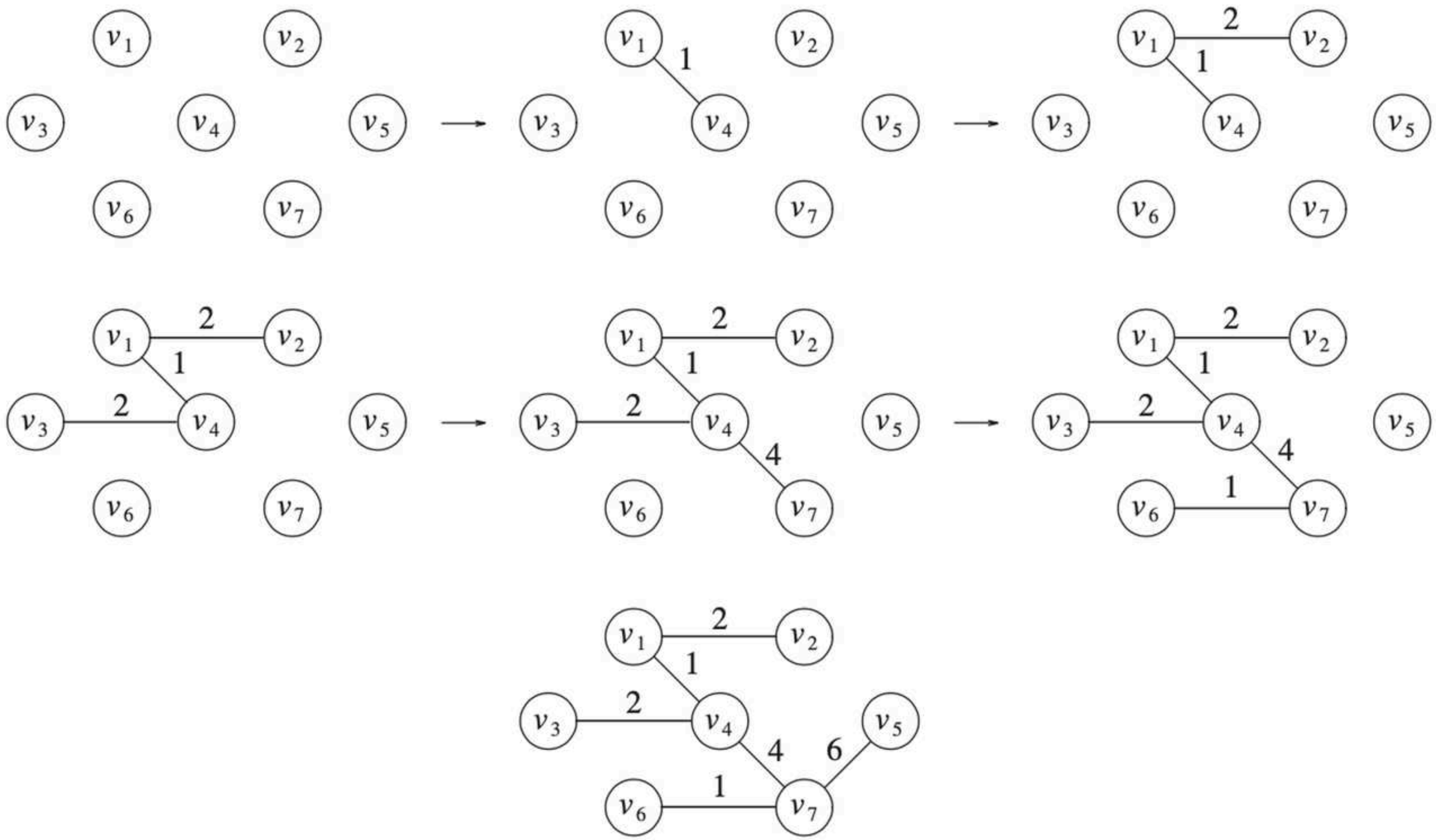
Prim’s MST algorithm



Prim's algorithm for minimum spanning trees

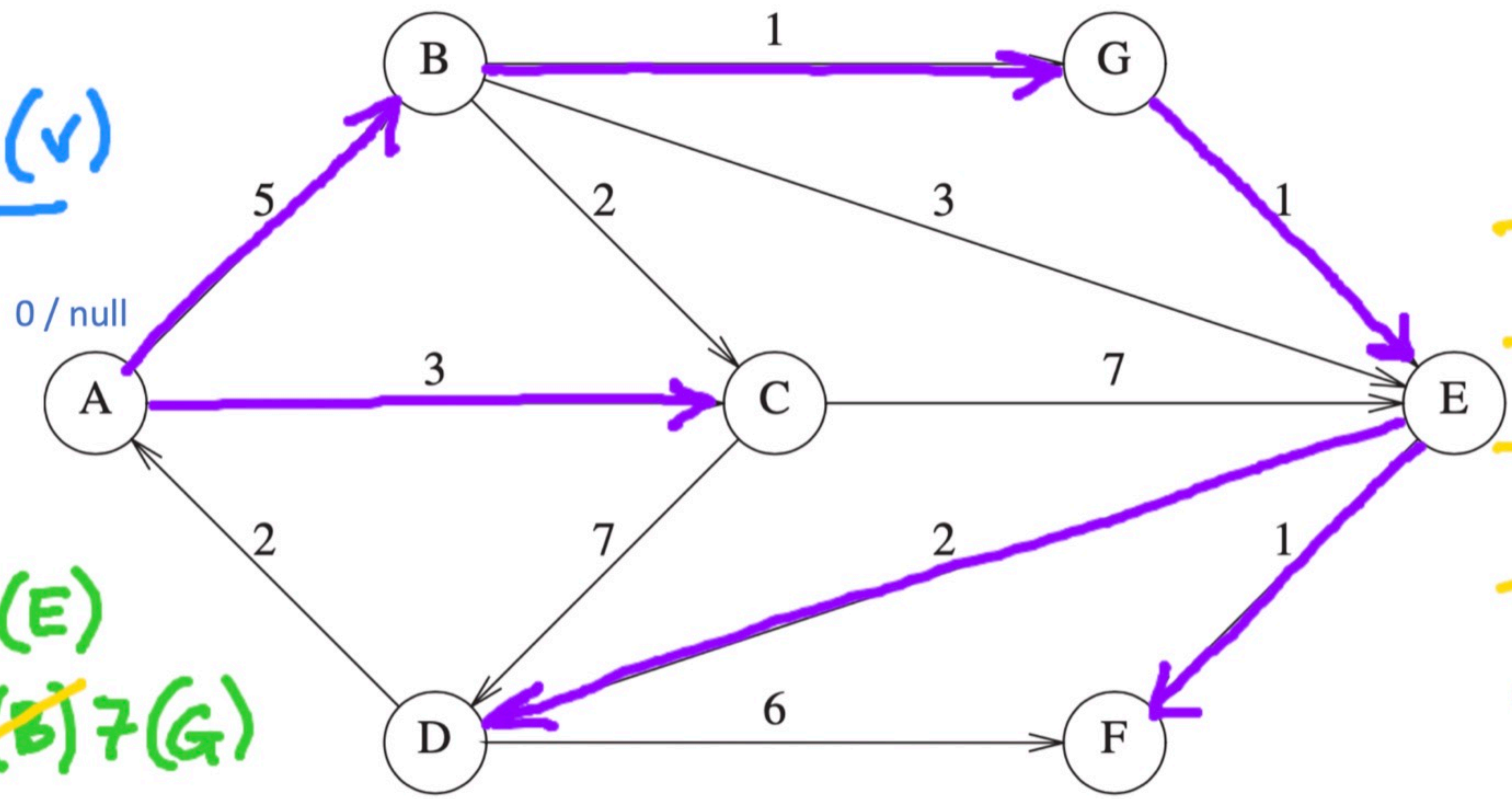
Prim’s MST algorithm





Practice with Dijkstra's algorithm

v	distTo(v)
✓A	0
✓B	5(A)
✓C	3(A)
✓D	~~10(C)~~ 9(E)
✓E	~~10(C)~~ ~~8(B)~~ 7(G)
✓F	8(E)
✓G	6(B)



PQ

~~A: 0~~
~~B: ∞ 5~~
~~C: ∞ 3~~
~~D: ∞ 10 9~~
~~E: ∞ 10 8 7~~
~~F: ∞ 8~~
~~G: ∞ 6~~

Coming up:

- Continue working on Homework 10, due Thursday 5/8
- Lab 9 assignment due tonight: if you haven't already, submit to Gradescope what you have so far for Homework 10 (even if no tests pass)
- Lab 10 on Friday 5/9: practice and surveys
- Monday 5/12: office hours, no lab meetings