https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

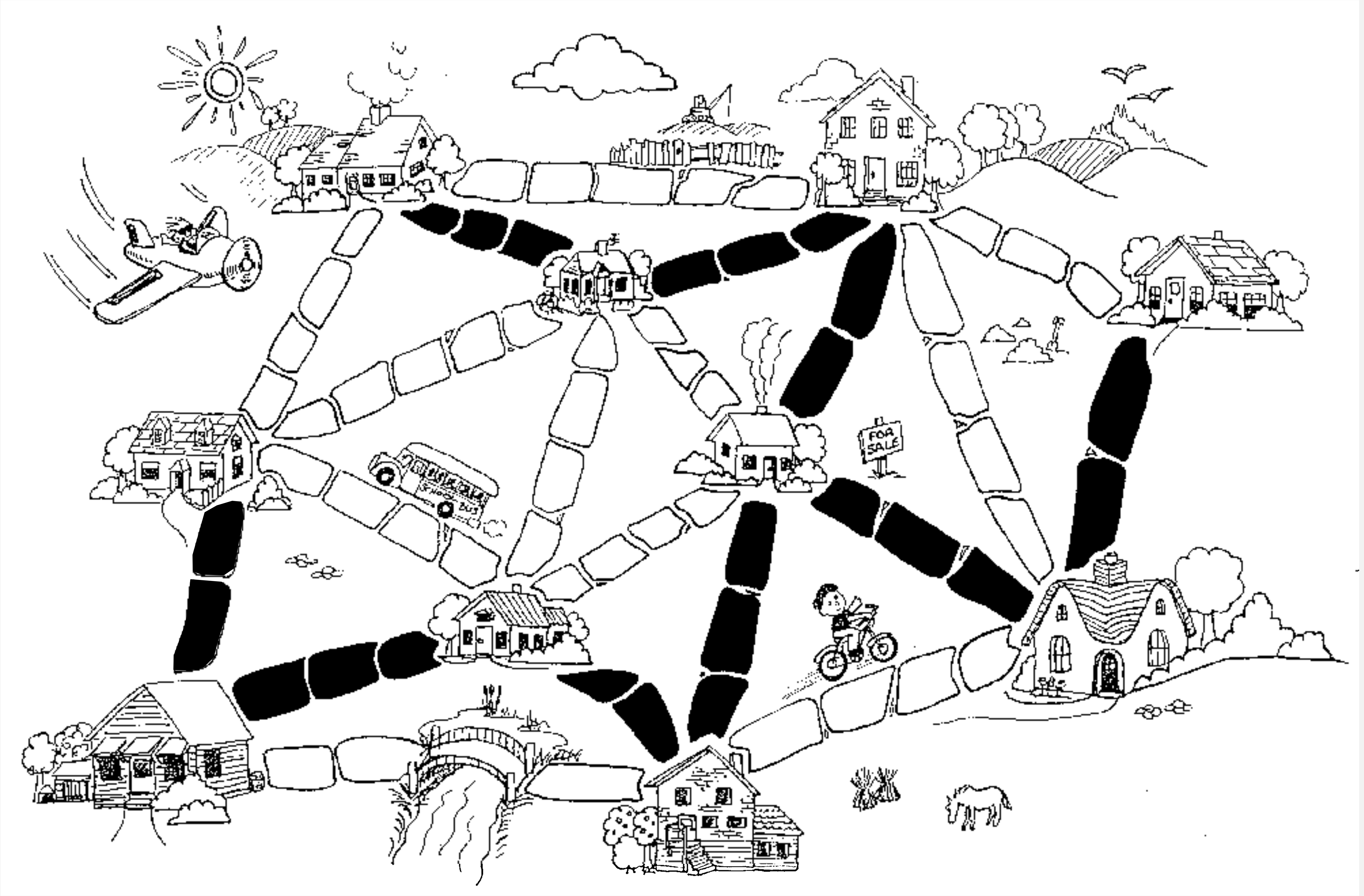
https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

A motivating example

Install minimum number of paving stones to connect all of the houses.

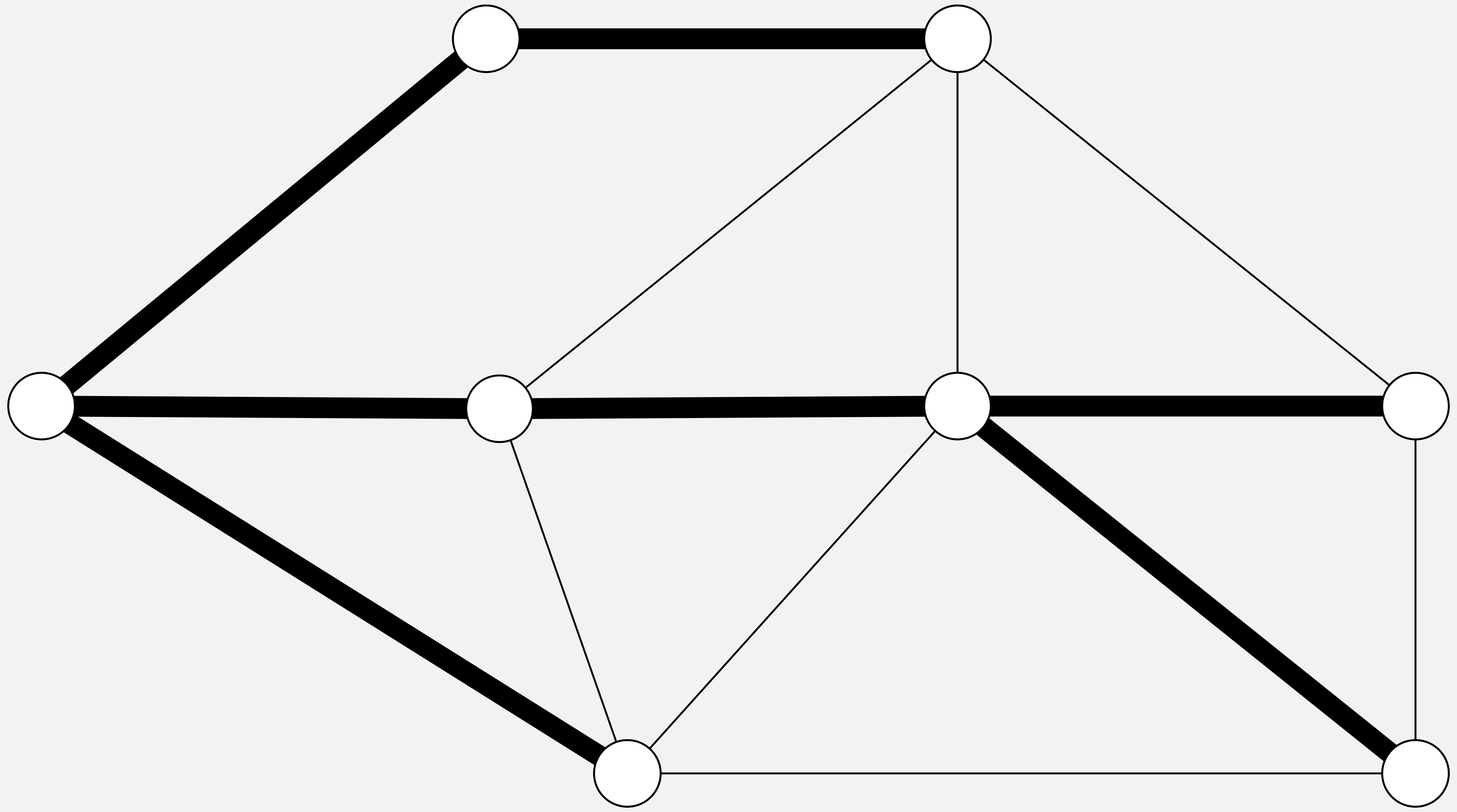


https://www.utdallas.edu/~besp/teaching/mst-applications.pdf

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



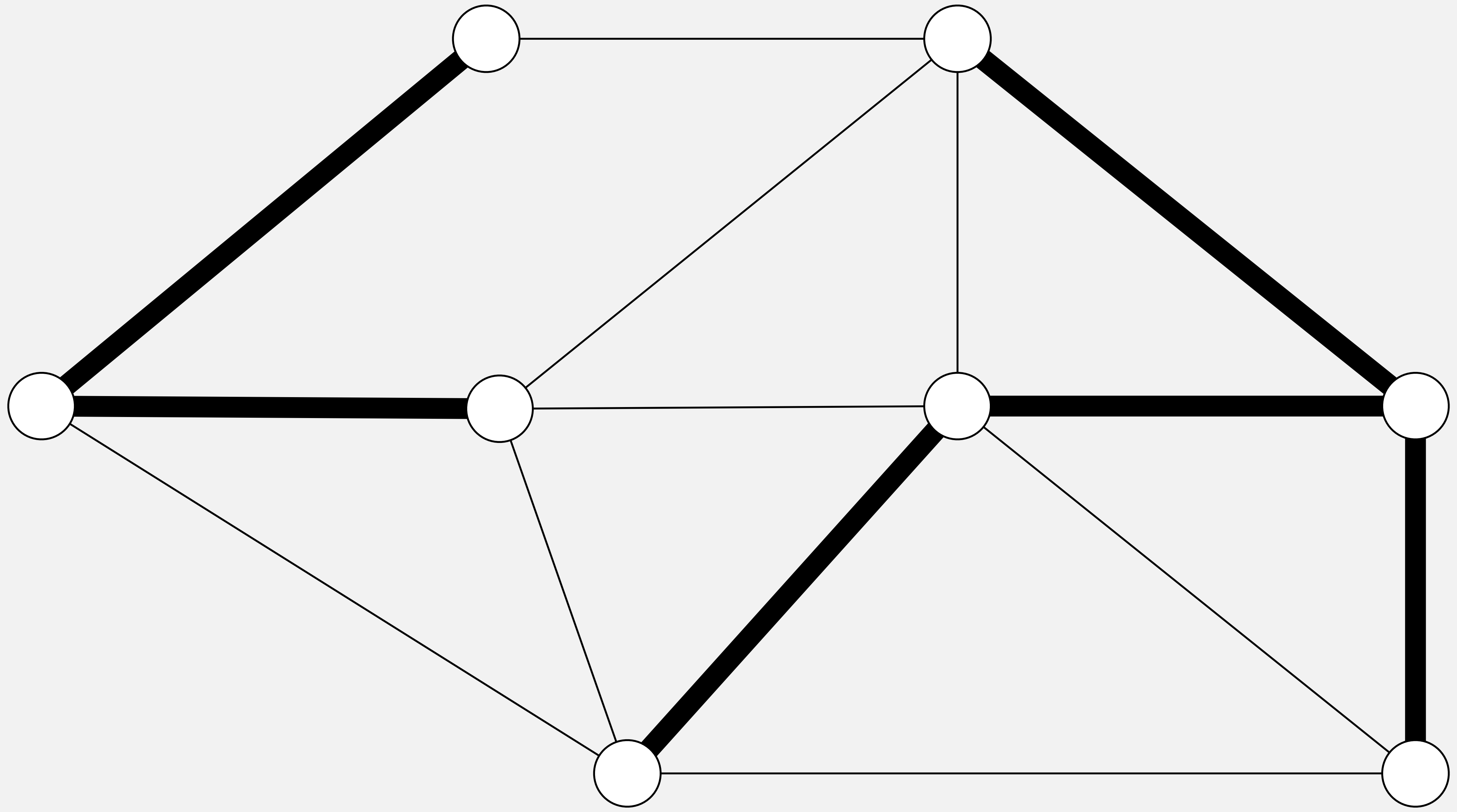
graph G

spanning tree T

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

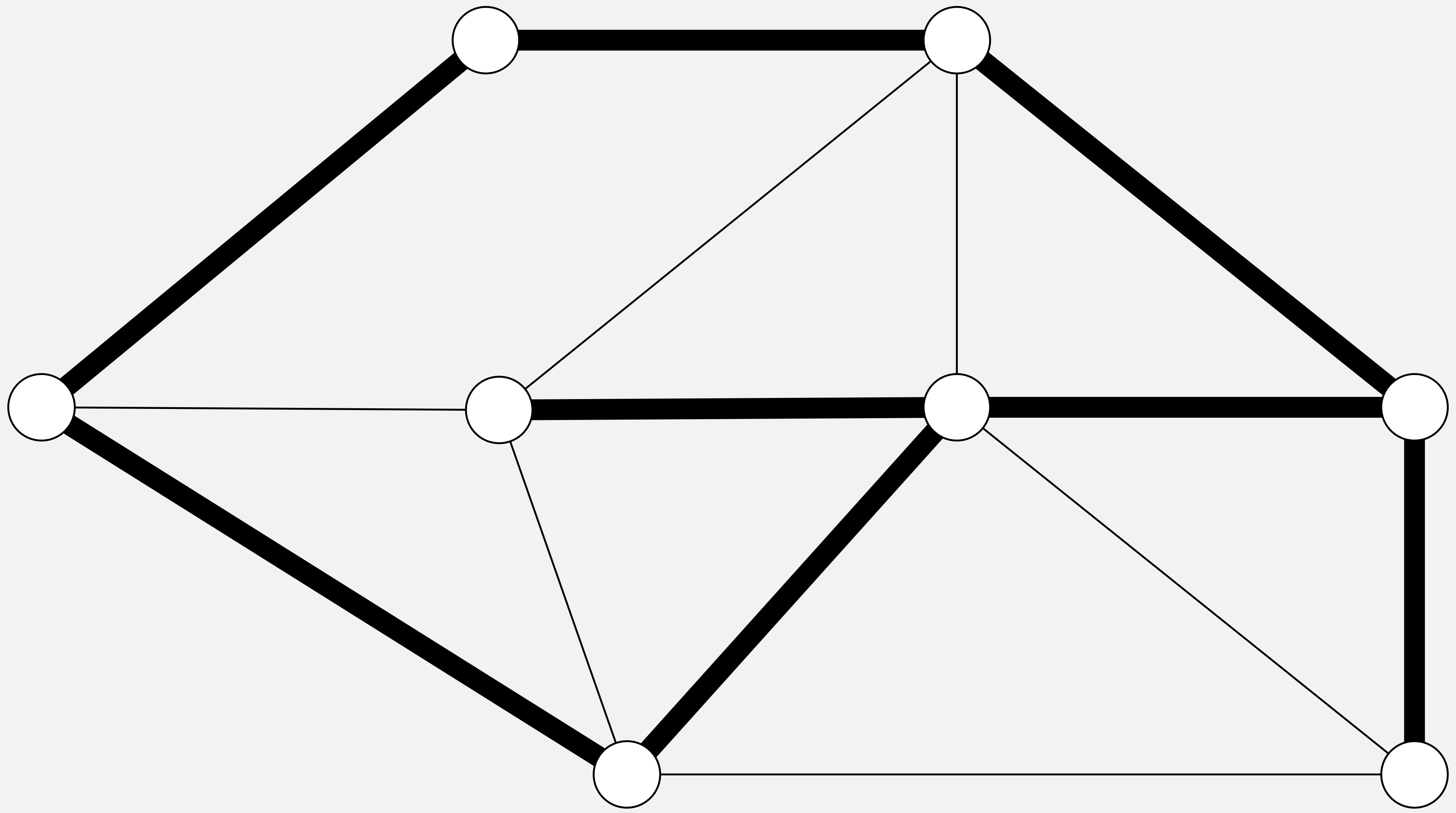


not connected

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

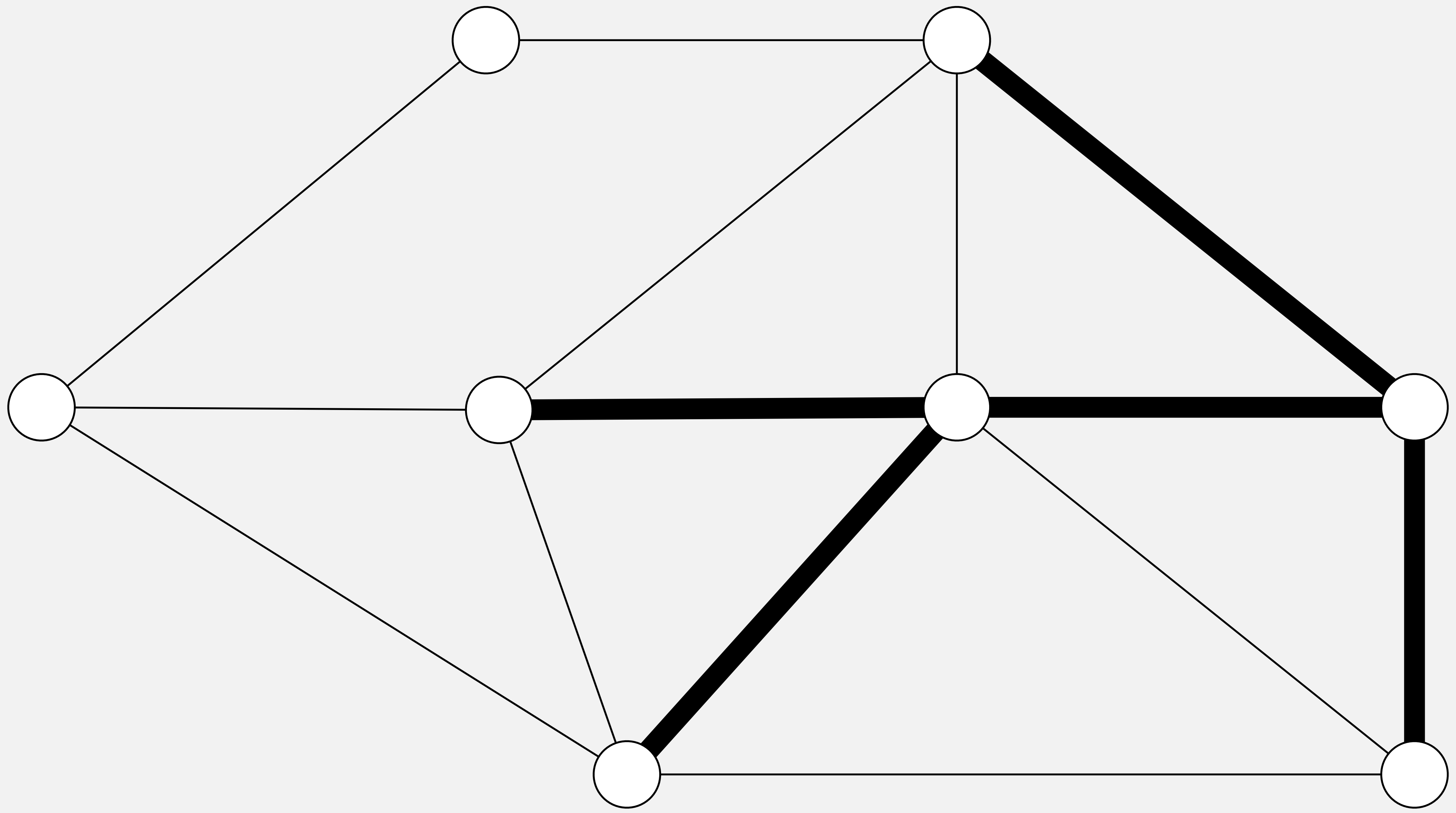


not acyclic

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

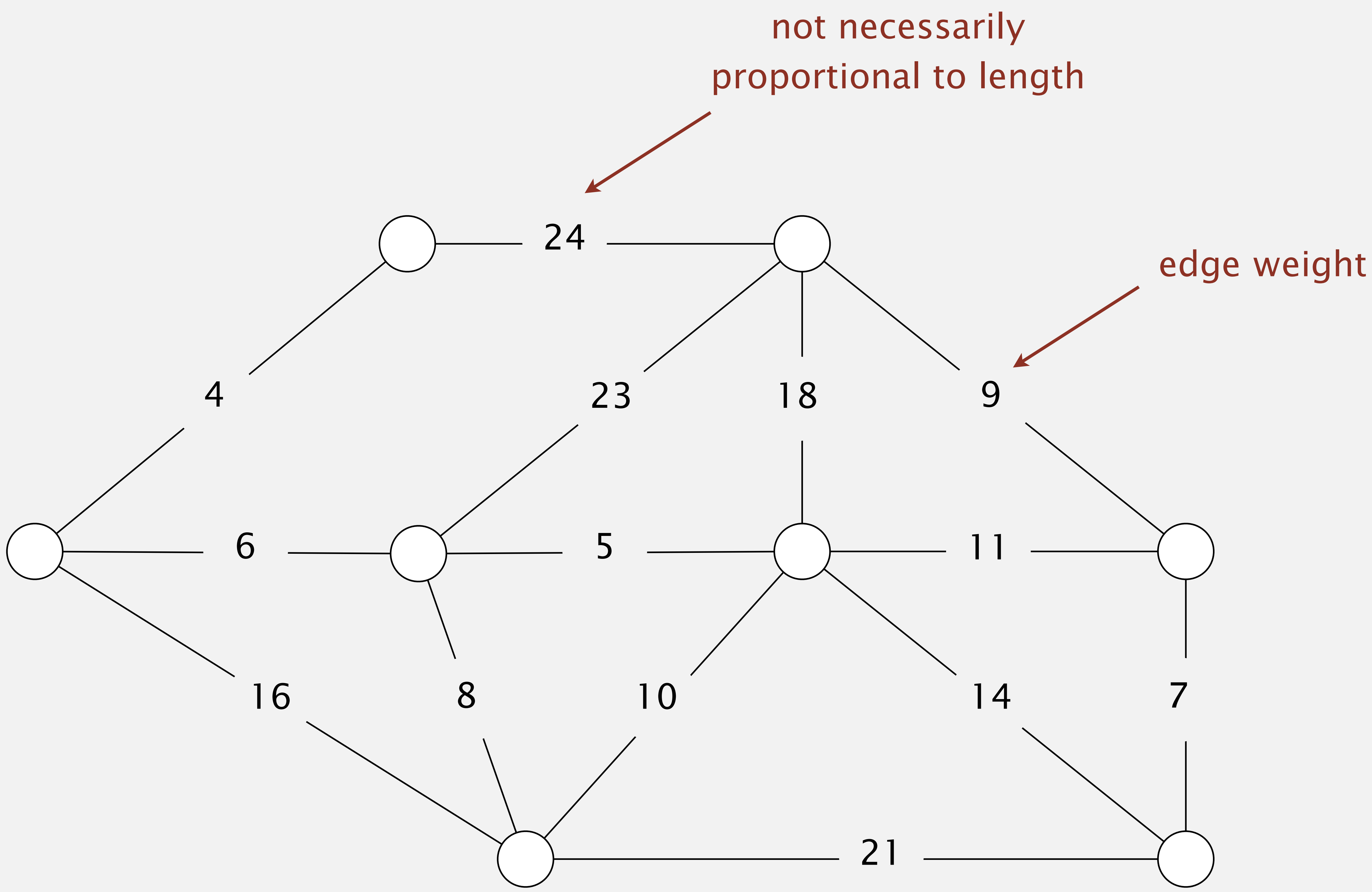
- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



not spanning

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

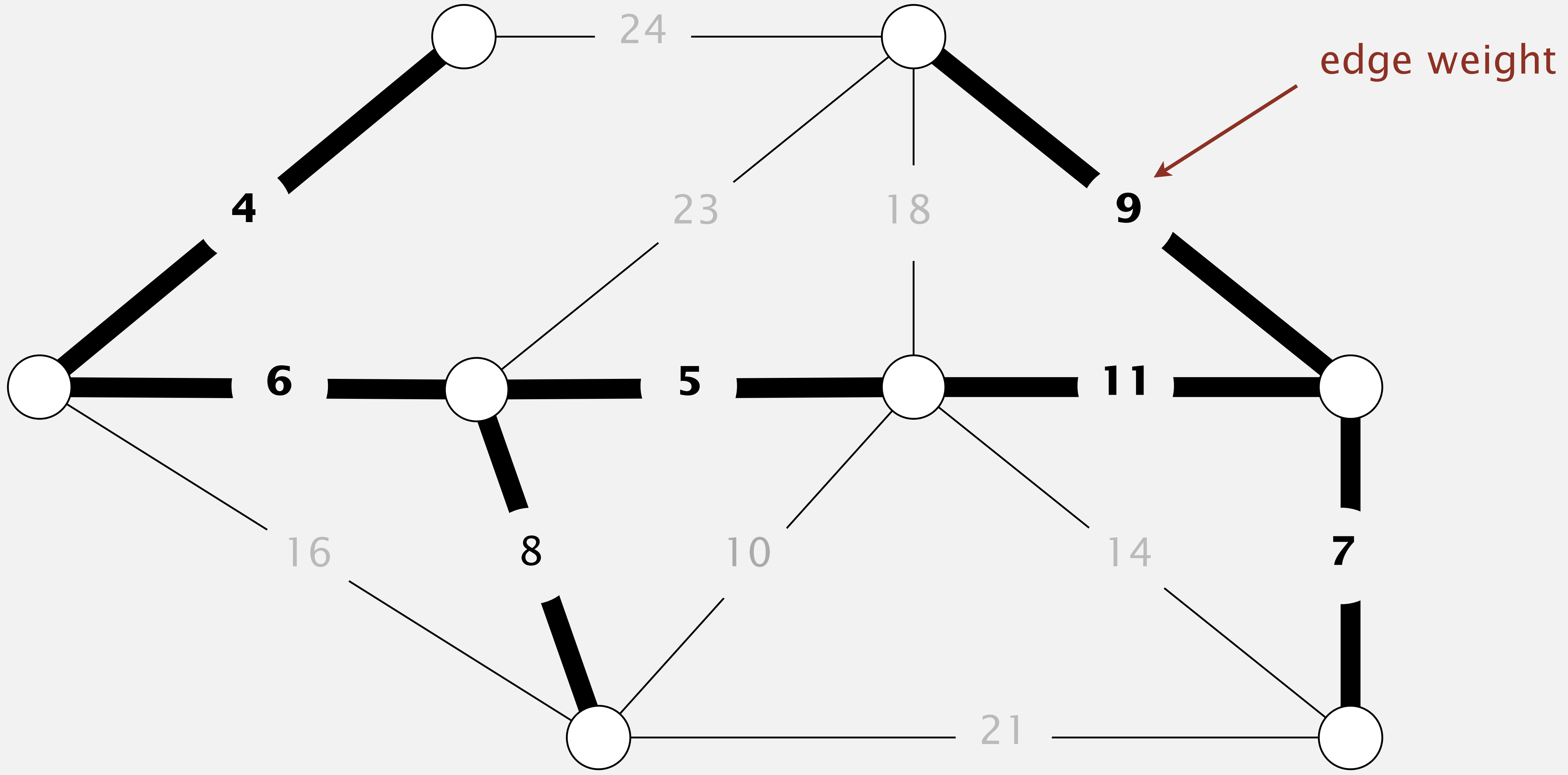


edge-weighted graph G

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

Output. A spanning tree of minimum weight.

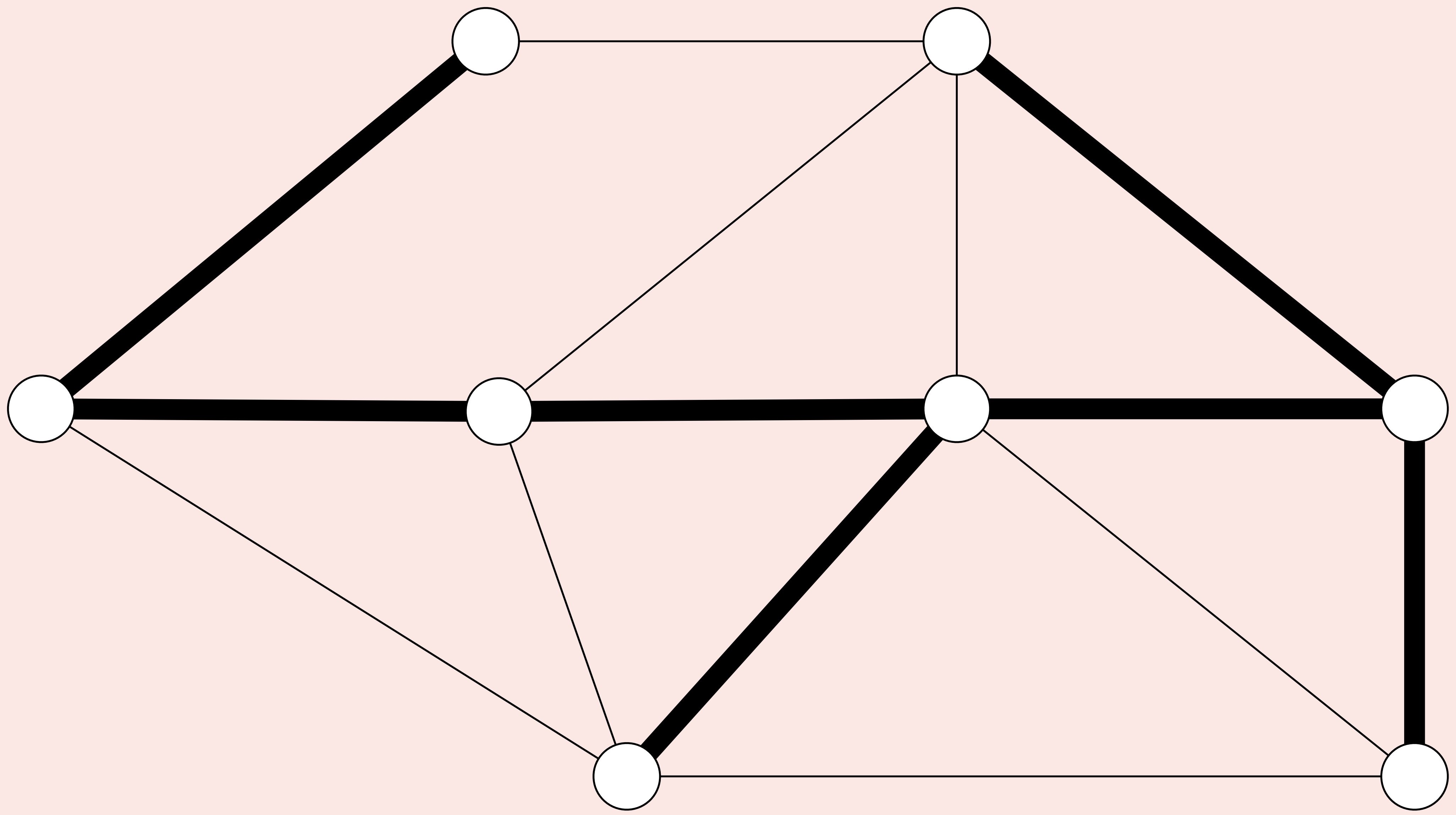


minimum spanning tree T
(weight = 50 = 4 + 6 + 5 + 8 + 9 + 11 + 7)

Brute force. Try all spanning trees?

Let T be any spanning tree of a connected graph G with V vertices.
Which of the following properties must hold?

- **A.** T contains exactly $V - 1$ edges.
- **B.** Removing any edge from T disconnects it.
- **C.** Adding any edge to T creates a cycle.
- **D.** All of the above.

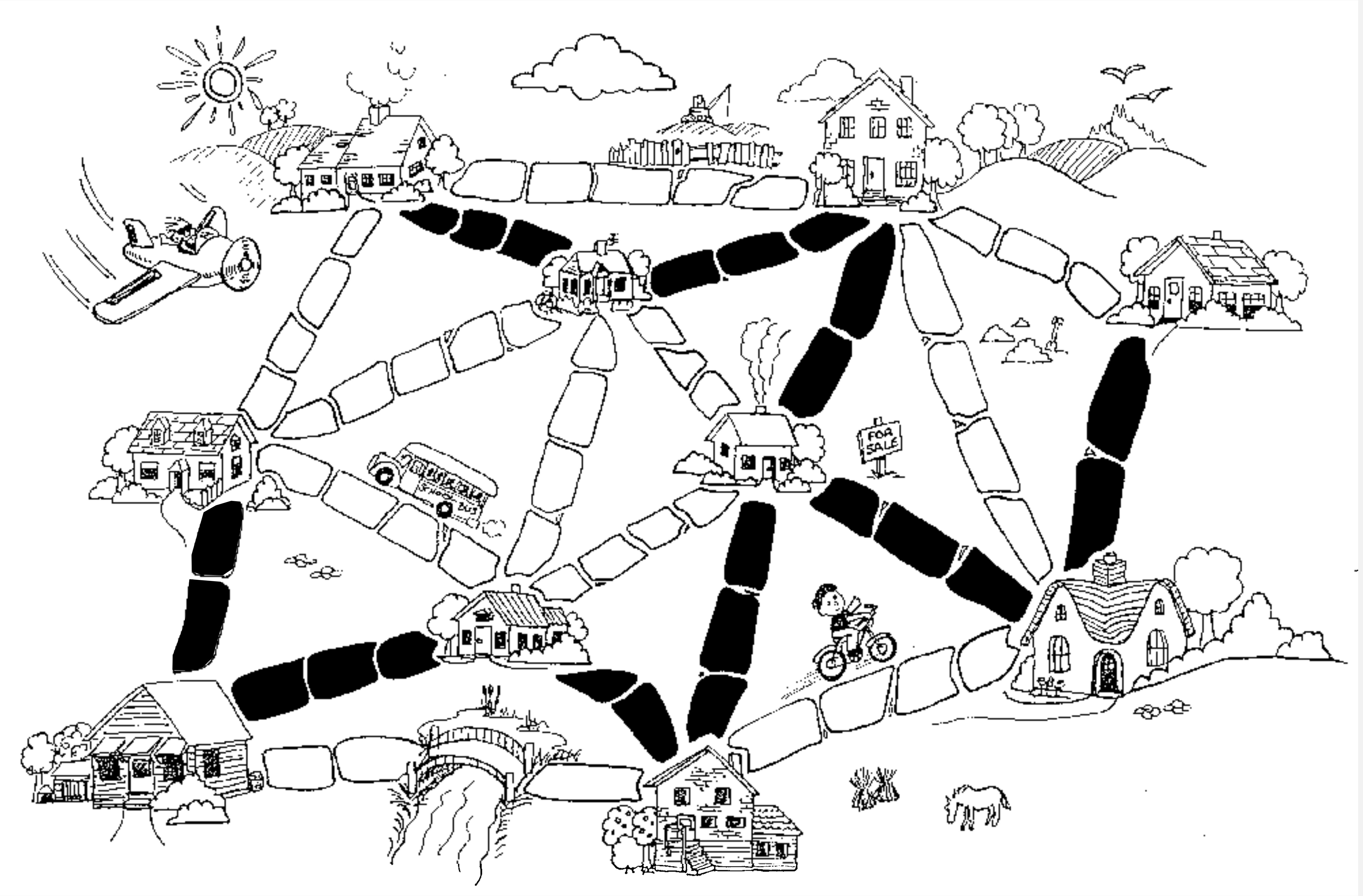


spanning tree T of graph G

Network design

Network. Vertex = network component; edge = potential connection; edge weight = cost.

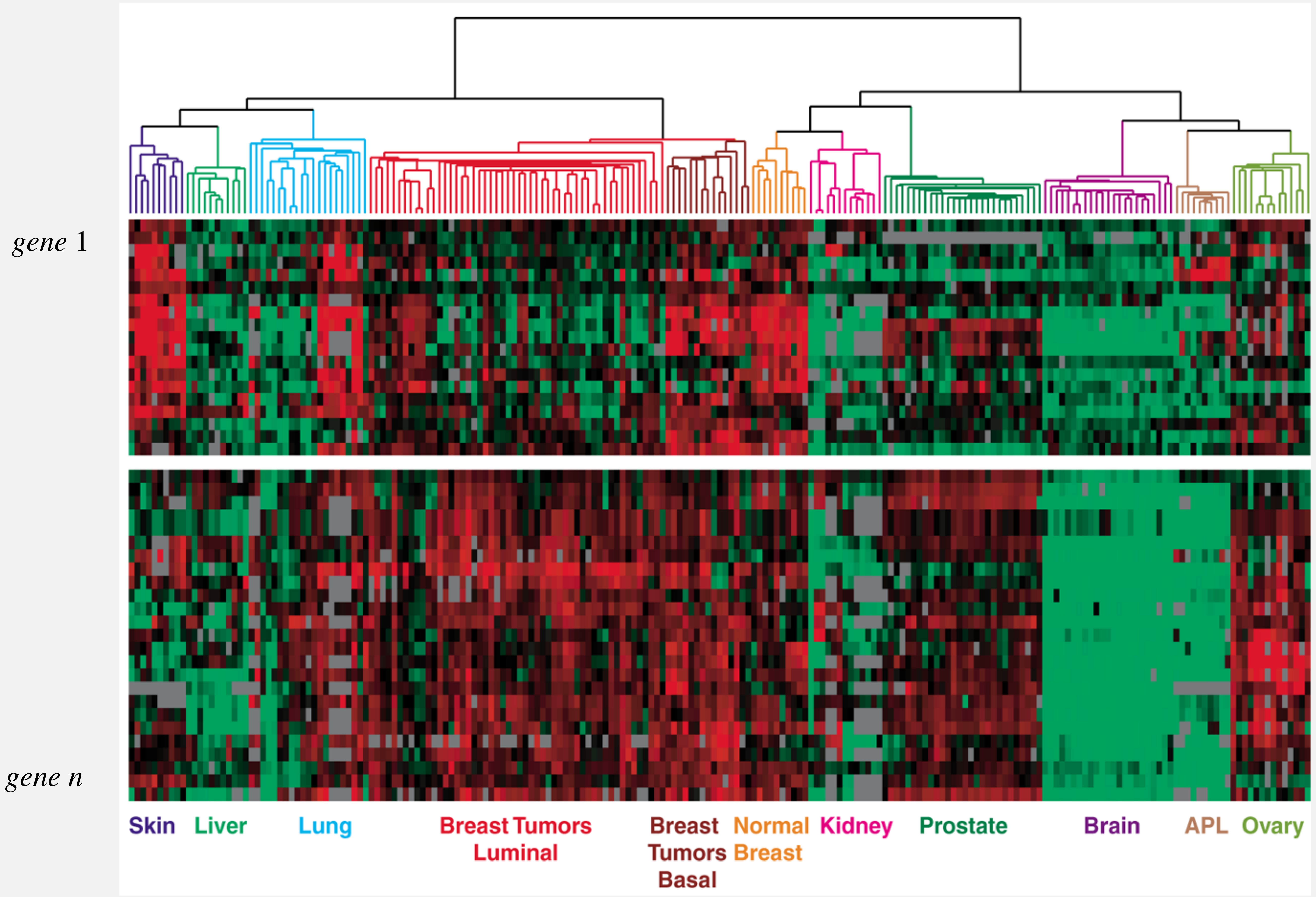
electrical, computer, telecommunication, transportation



https://www.utdallas.edu/~besp/teaching/mst-applications.pdf

Hierarchical clustering

Microarray graph. Vertex = cancer tissue; edge = all pairs; edge weight = dissimilarity.

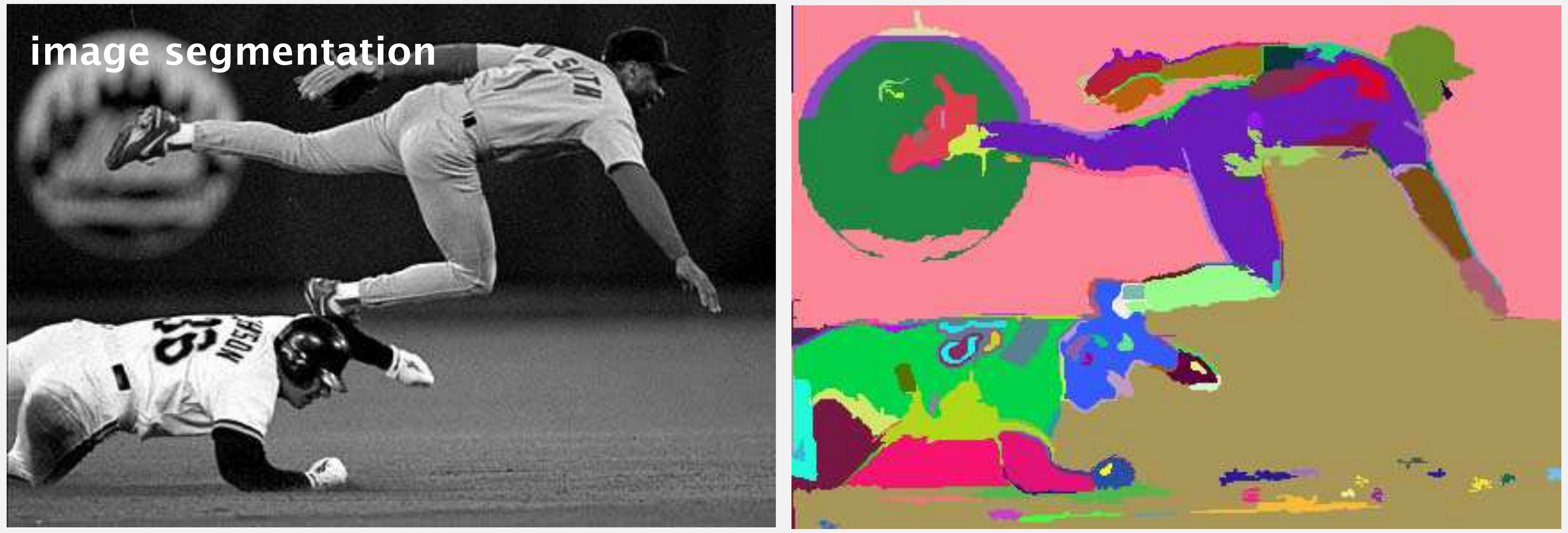


Reference: Botstein & Brown group

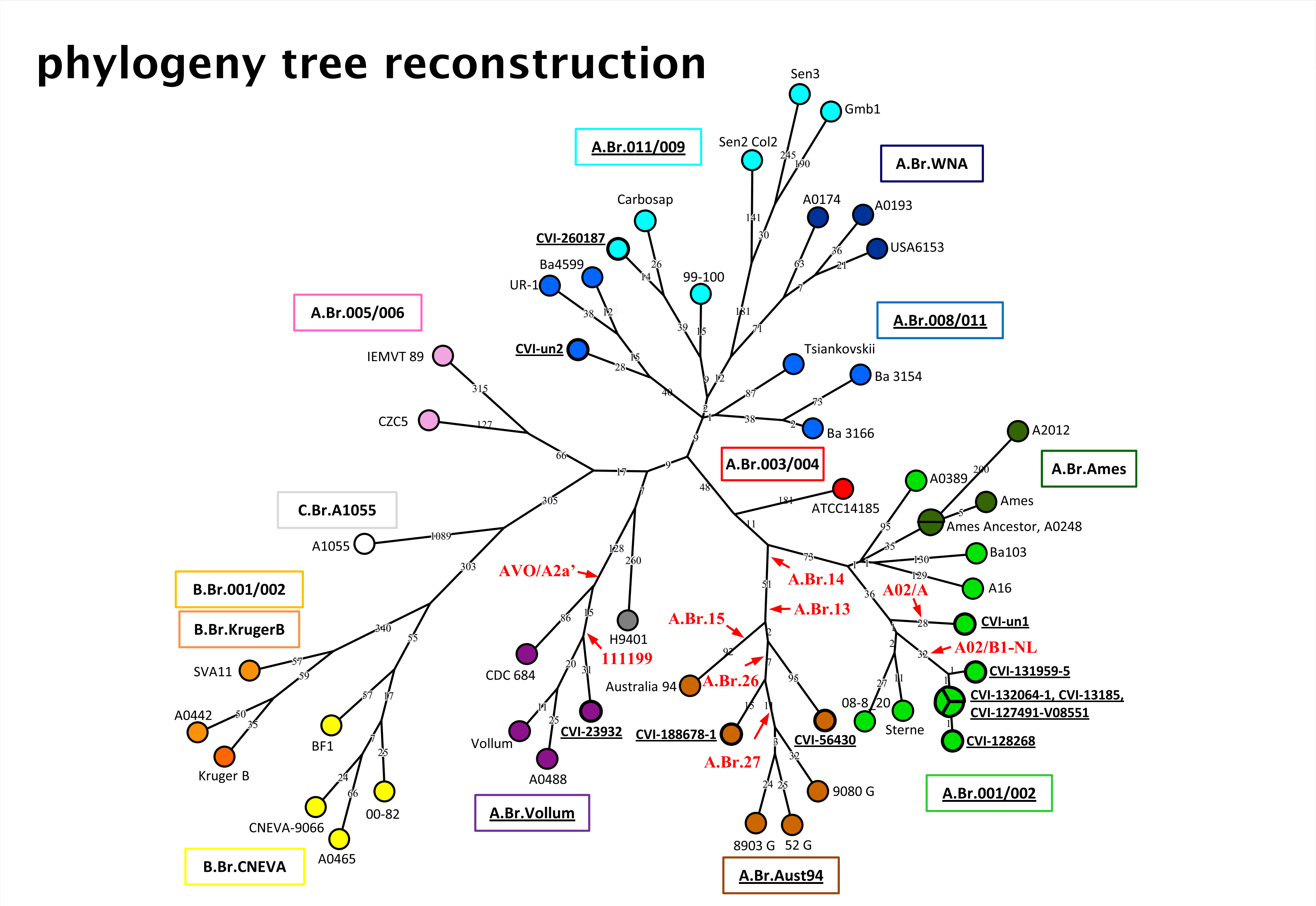
gene expressed

gene not expressed

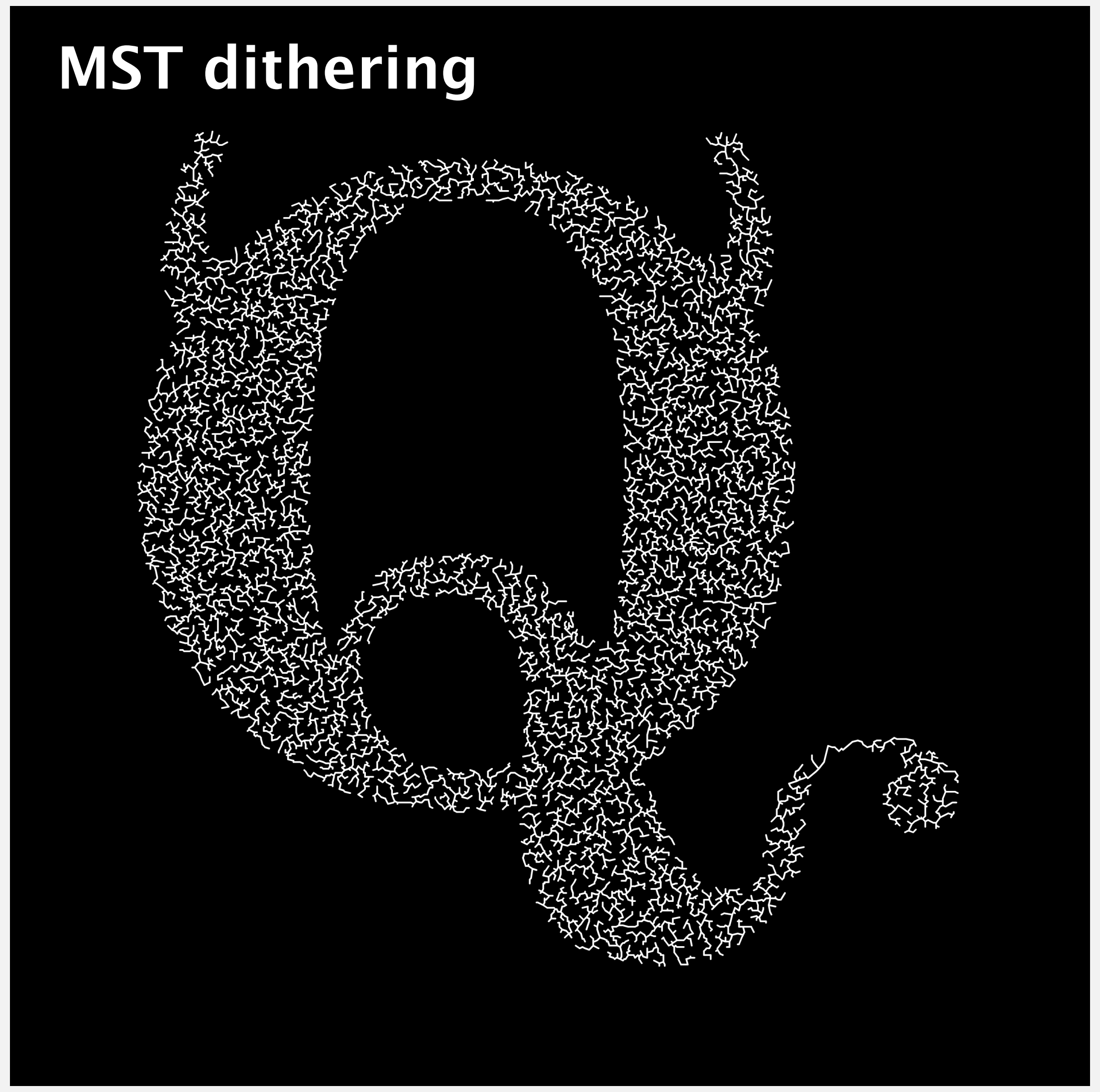
More MST applications



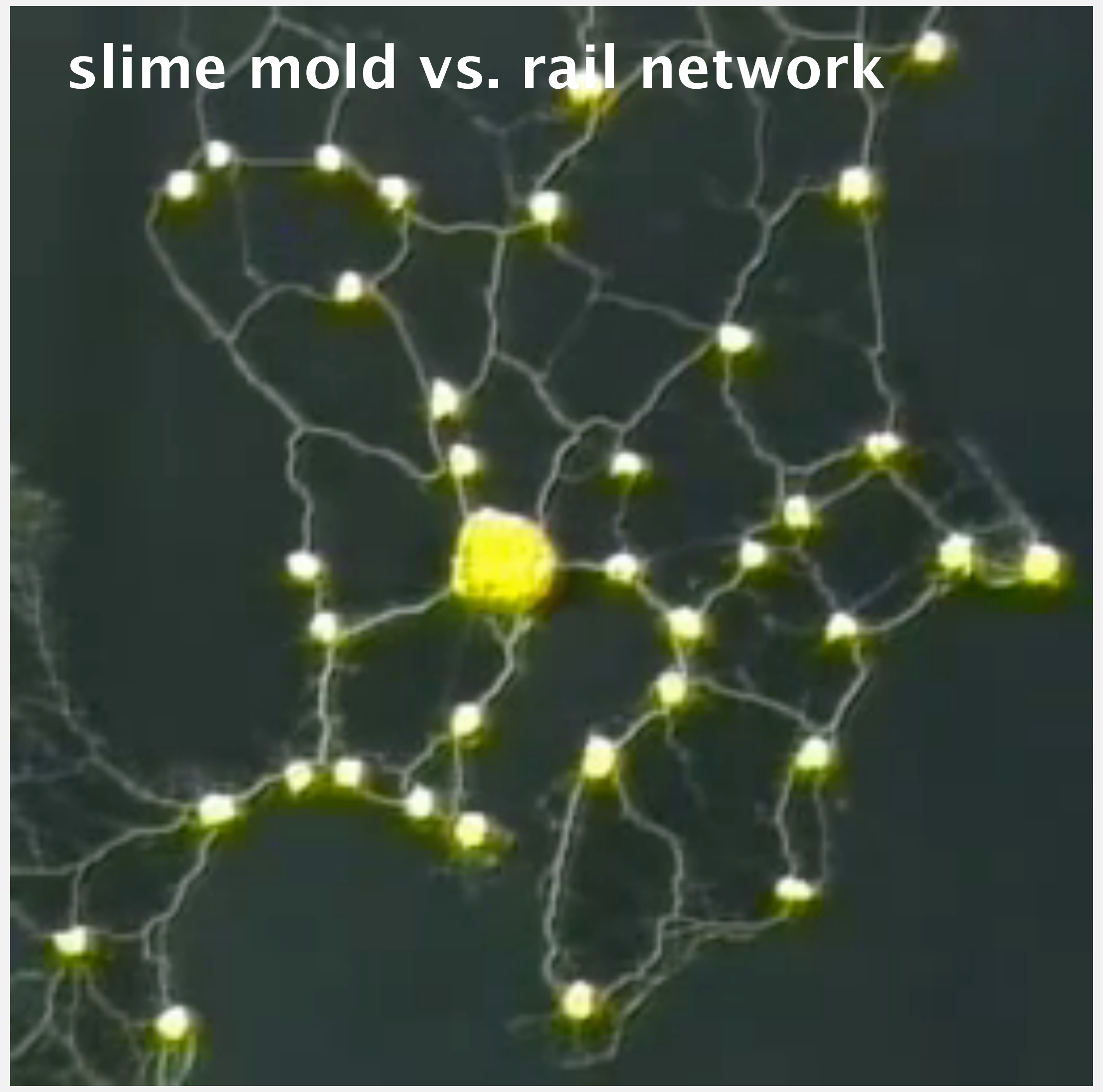
https://link.springer.com/article/10.1023/B:VISI.0000022288.19776.77



https://www.sciencedirect.com/science/article/pii/S156713481500115X



http://www.flickr.com/photos/quasimondo/2695389651



https://www.youtube.com/watch?v=GwKuFREOgmo

https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

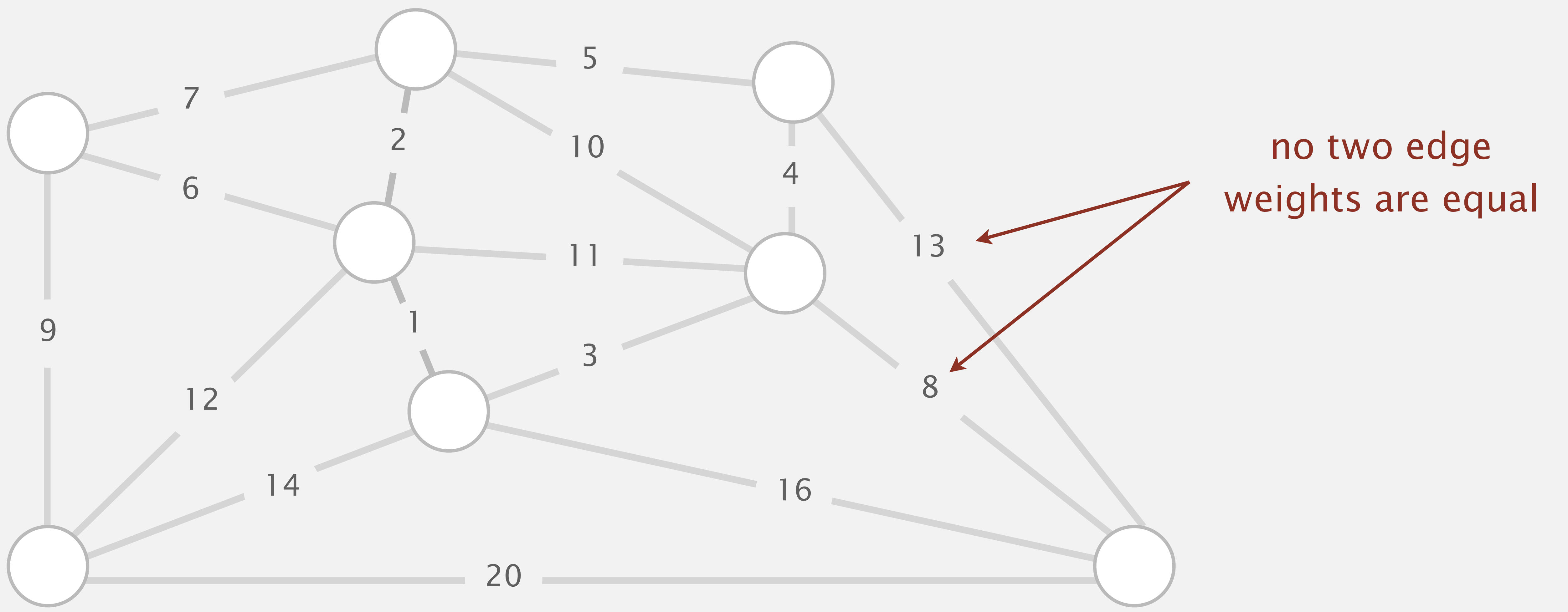
Simplifying assumptions

For simplicity, we assume:

- The graph is connected. ⇒ MST exists.
- The edge weights are distinct. ⇒ MST is unique. ← see Exercise 4.3.3 (solved on booksite)

Note. Today's algorithms all work fine with duplicate edge weights.

↖ assumption simplifies the analysis

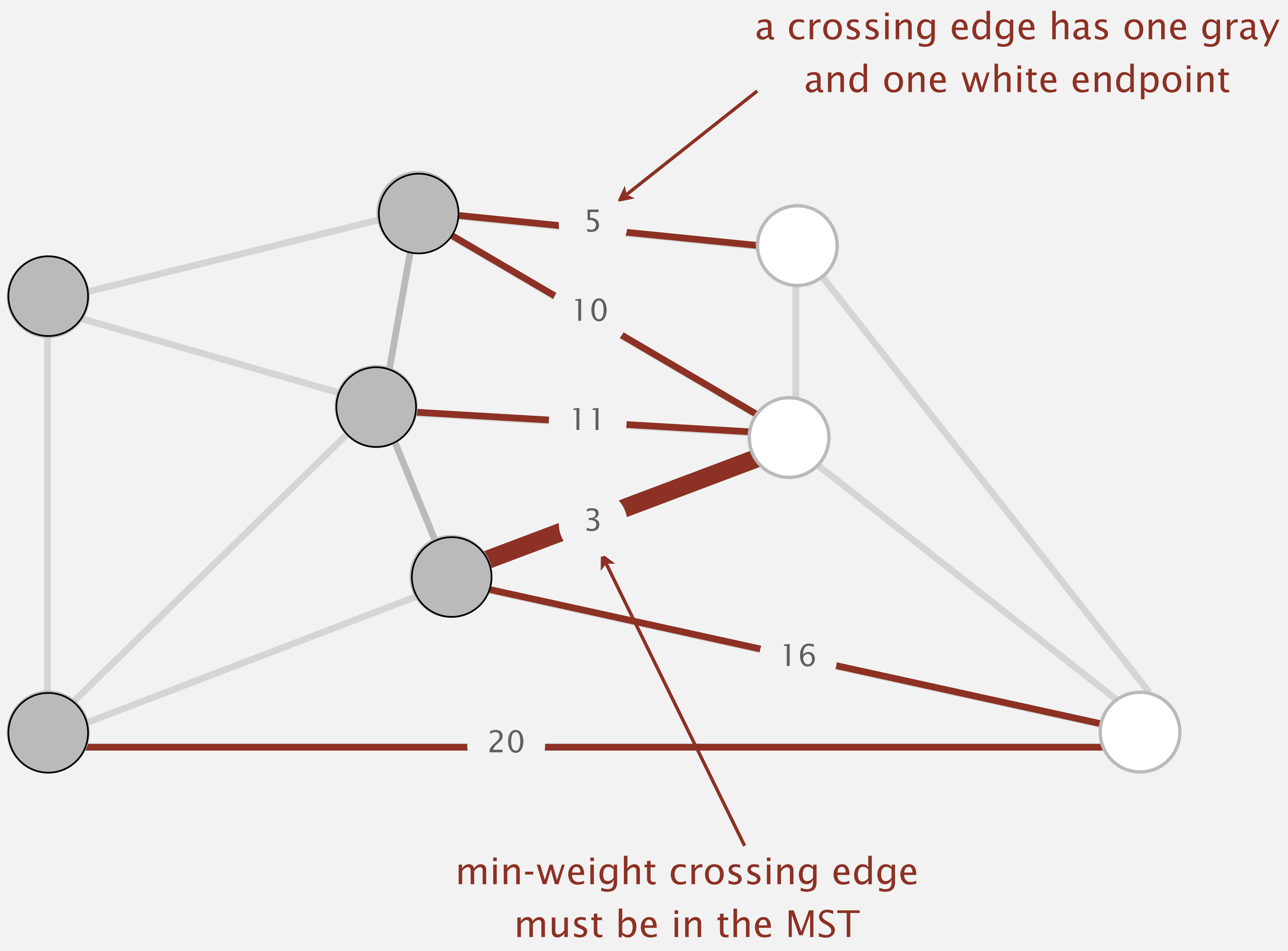


Cut property

Def. A cut in a graph is a partition of its vertices into two nonempty sets.

Def. A crossing edge of a cut is an edge that has one endpoint in each set.

Cut property. For any cut, its min-weight crossing edge is in the MST.



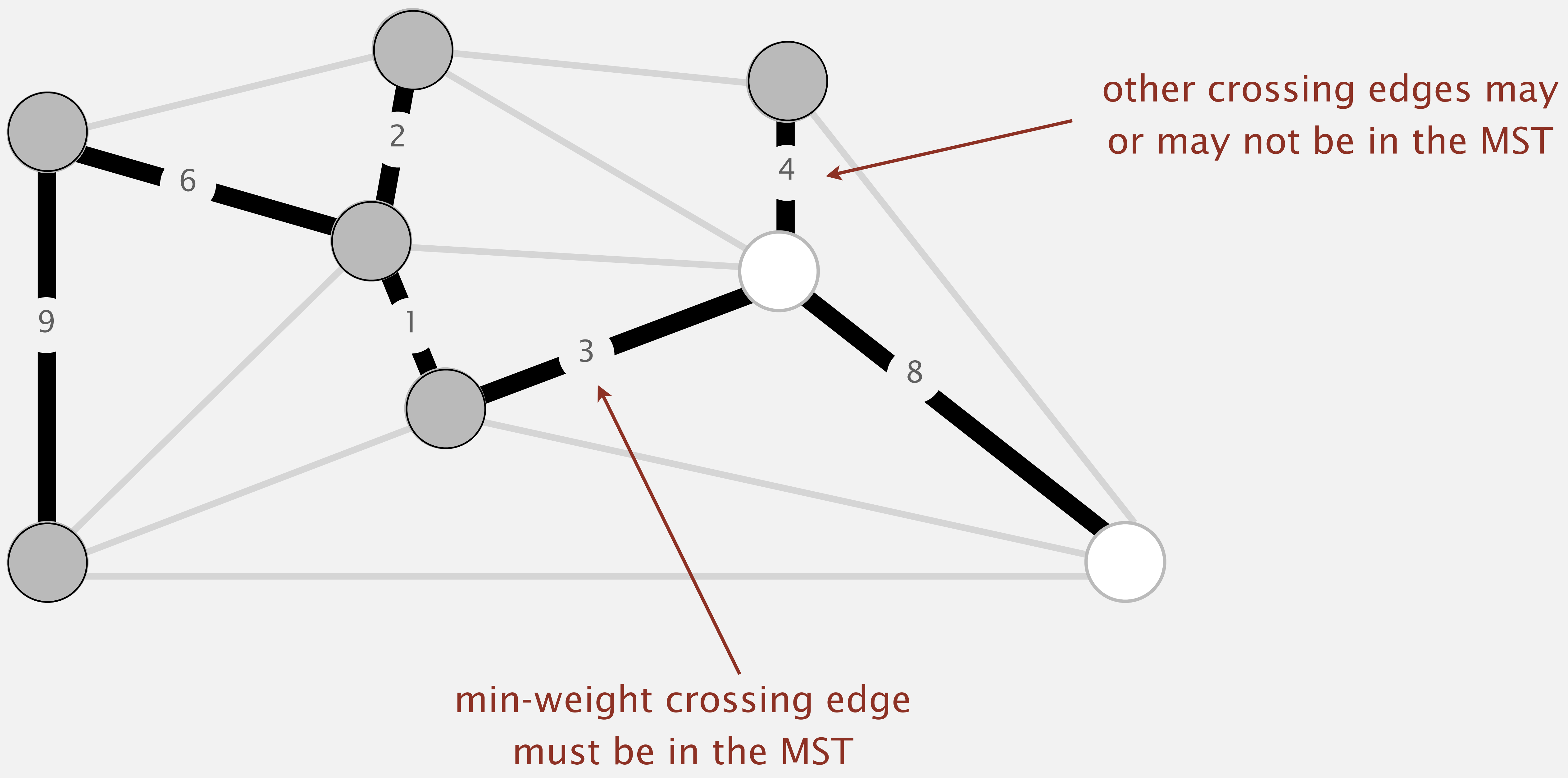
Cut property

Def. A cut in a graph is a partition of its vertices into two nonempty sets.

Def. A crossing edge of a cut is an edge that has one endpoint in each set.

Cut property. For any cut, its min-weight crossing edge is in the MST.

Note. A cut may have multiple edges in the MST.



Minimum spanning trees: quiz 2

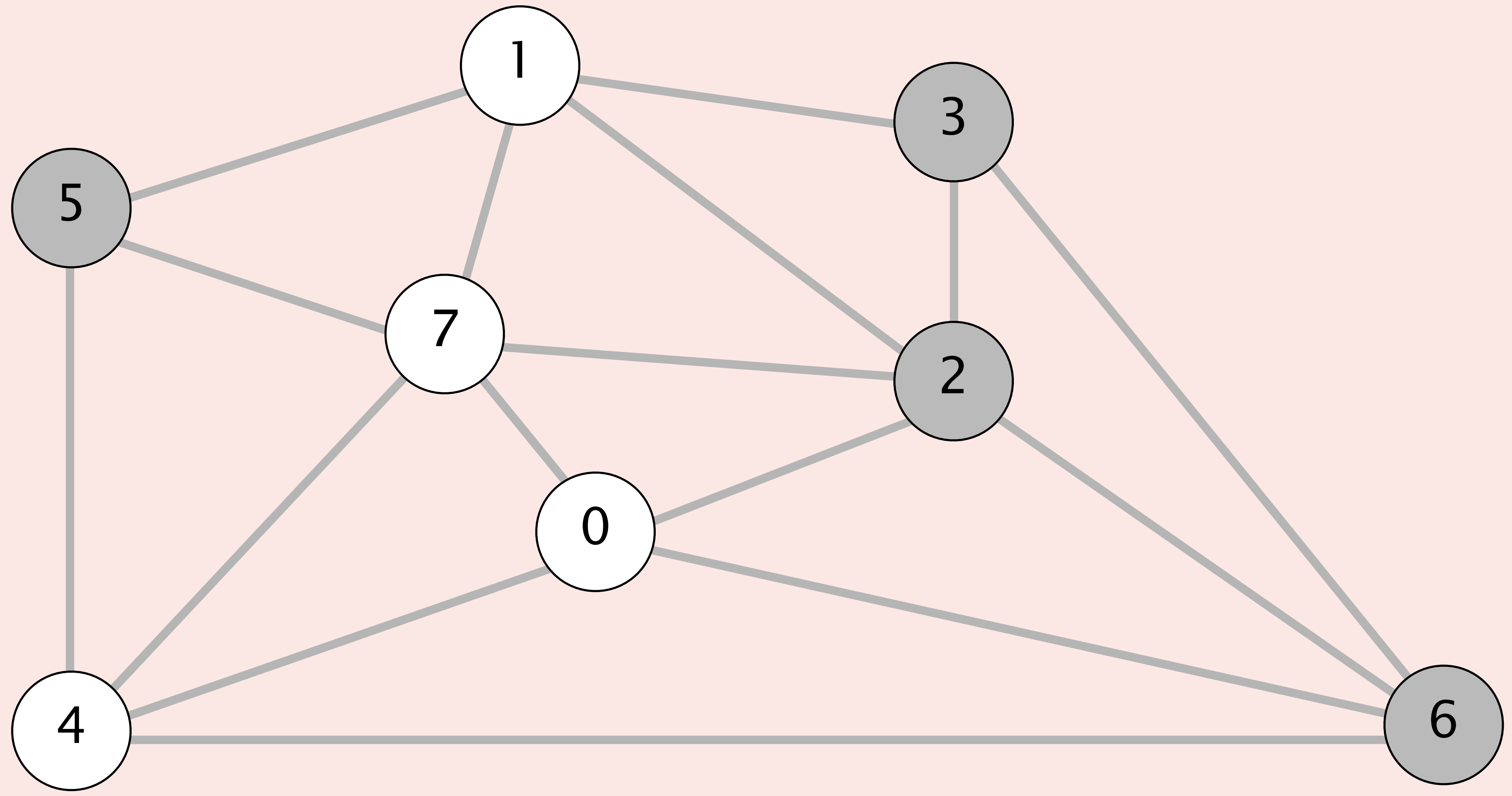
Which is the min-weight edge crossing the cut { 2, 3, 5, 6 } ?

A. 0–7 (0.16)

B. 2–3 (0.17)

C. 0–2 (0.26)

D. 5–7 (0.28)



0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Cut property: correctness proof

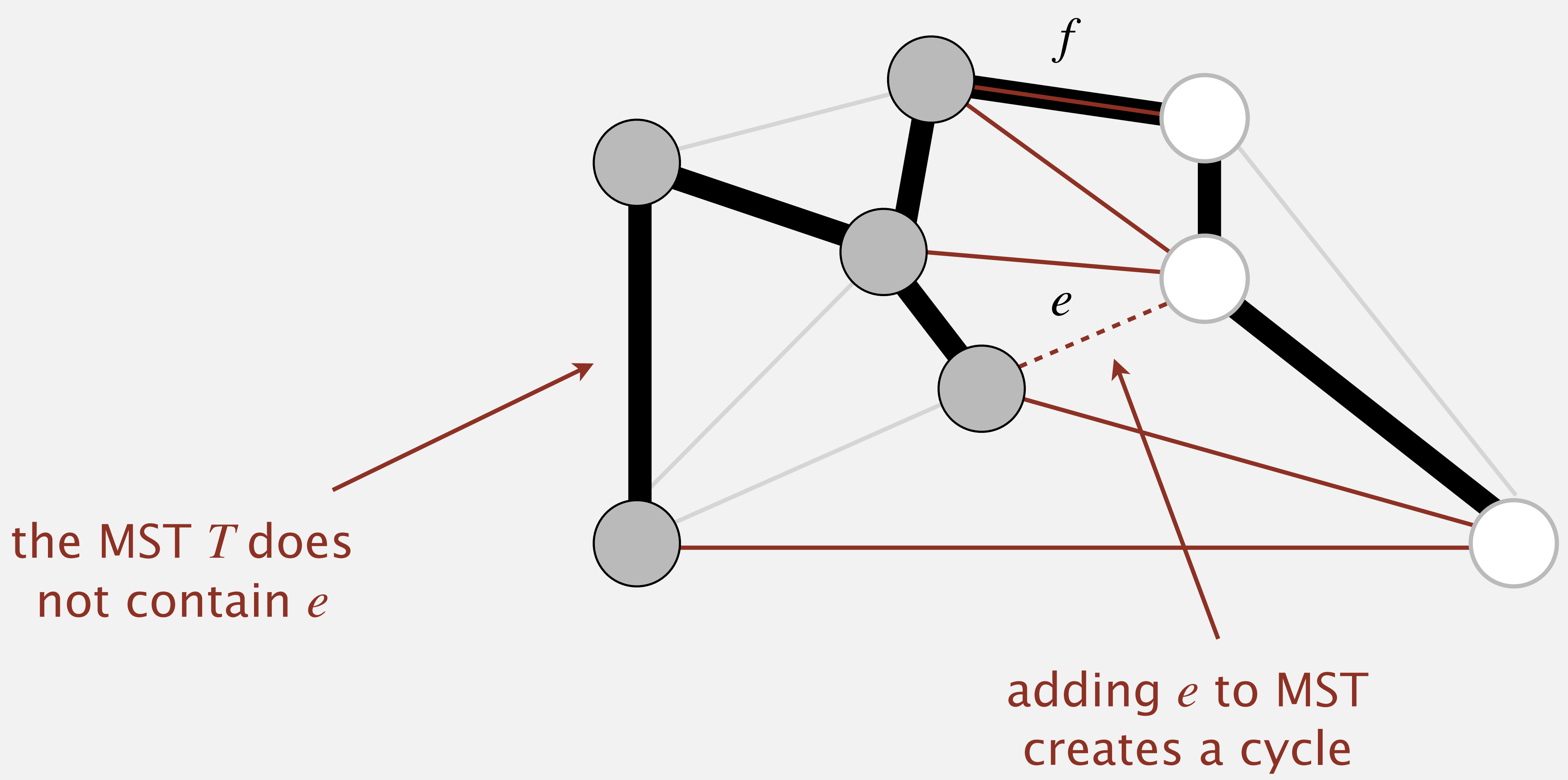
Def. A cut in a graph is a partition of its vertices into two nonempty sets.

Def. A crossing edge of a cut is an edge that has one endpoint in each set.

Cut property. For any cut, its min-weight crossing edge e is in the MST.

Pf. [by contradiction] Suppose e is not in the MST T.

- Adding e to T creates a cycle.
- Some other edge f in cycle must also be a crossing edge.
- Removing f and adding e yields a different spanning tree T'.
- Since $weight(e) < weight(f)$, we have $weight(T') < weight(T)$.
- Contradiction. ▪



Framework for minimum spanning tree algorithms

Generic algorithm (to compute MST in G)

T = ∅.

Repeat until T is a spanning tree: ⟵ $V-1$ edges

- **Find a cut in G.**
- **e ← min-weight crossing edge.**
- **T ← T ∪ { e }.**

Efficient implementations.

- Which cut? ⟵ 2^{V-2} distinct cuts
- How to compute min-weight crossing edge?

Ex 1. Kruskal's algorithm.

Ex 2. Prim's algorithm.

Ex 3. Borüvka's algorithm.

https://algs4.cs.princeton.edu

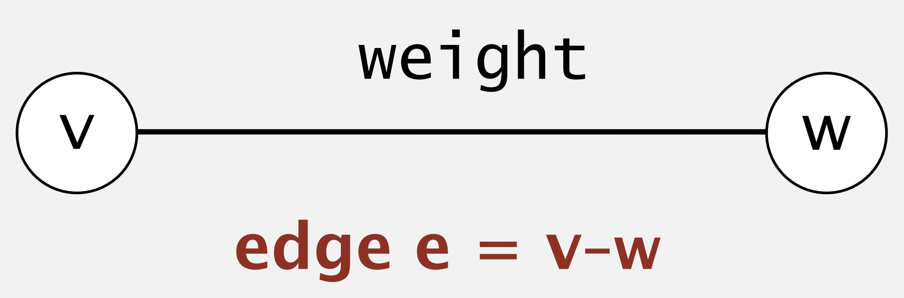
4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Weighted edge API

API. Edge abstraction for weighted edges.

public class Edge implements Comparable<Edge>		
	`Edge(int v, int w, double weight)`	*create a weighted edge v–w*
`int`	`either()`	*either endpoint*
`int`	`other(int v)`	*the endpoint that's not v*
`int`	`compareTo(Edge that)`	*compare edges by weight*
	⋮	⋮



Idiom for processing an edge e. `int v = e.either(), w = e.other(v);`

Weighted edge: Java implementation

```
public class Edge implements Comparable<Edge>
{
   private final int v, w;
   private final double weight;

   public Edge(int v, int w, double weight)
   {
      this.v = v;
      this.w = w;
      this.weight = weight;
   }

   public int either()
   {  return v;  }

   public int other(int vertex)
   {
      if (vertex == v) return w;
      else return v;
   }

   public int compareTo(Edge that)
   {  return Double.compare(this.weight, that.weight);  }

}
```

constructor

either endpoint

other endpoint

compare edges by weight

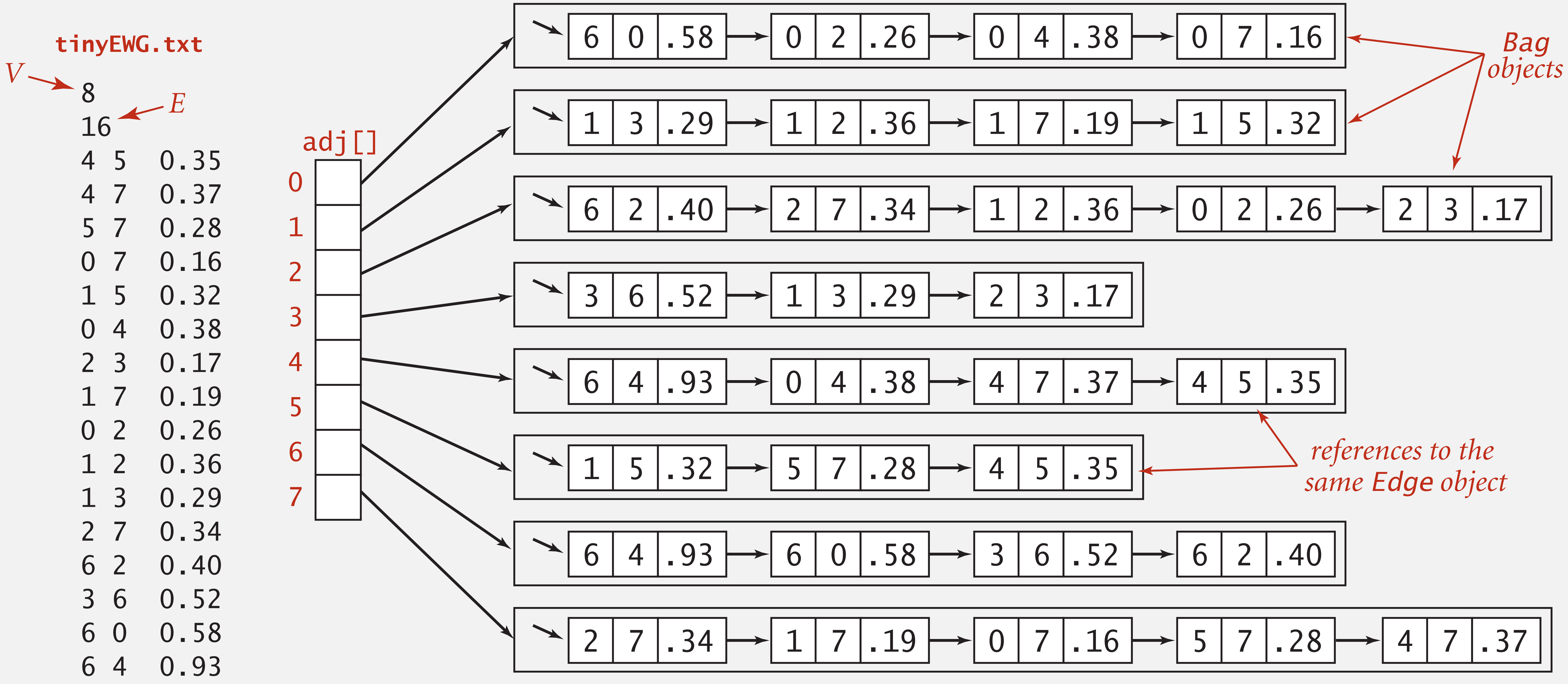
Edge-weighted graph API

API. Same as `Graph` and `Digraph`, except with explicit `Edge` objects.

public class EdgeWeightedGraph		
	`EdgeWeightedGraph(int V)`	*create an empty graph with V vertices*
`void`	`addEdge(Edge e)`	*add weighted edge e to this graph*
`Iterable<Edge>`	`adj(int v)`	*edges incident to v*
	⋮	⋮

Edge-weighted graph: adjacency-lists representation

Representation. Maintain vertex-indexed array of Edge lists.



Edge-weighted graph: adjacency-lists implementation

```
public class EdgeWeightedGraph
{
   private final int V;
   private final Bag<Edge>[] adj;

   public EdgeWeightedGraph(int V)
   {
     this.V = V;
     adj = (Bag<Edge>[]) new Bag[V];
     for (int v = 0; v < V; v++)
        adj[v] = new Bag<>();
   }

   public void addEdge(Edge e)
   {
     int v = e.either(), w = e.other(v);
     adj[v].add(e);
     adj[w].add(e);
   }

   public Iterable<Edge> adj(int v)
   {  return adj[v];   }
}
```

← same as Graph (but adjacency lists of Edge objects)

← constructor

← add same Edge object to both adjacency lists

Minimum spanning tree API

Q. How to represent the MST?

A. Technically, an MST is an edge-weighted graph.

For convenience, we represent it as a set of edges.

public class MST		
	`MST(EdgeWeightedGraph G)`	*constructor*
`Iterable<Edge>`	`edges()`	*edges in MST*
`double`	`weight()`	*weight of MST*

https://algs4.cs.princeton.edu

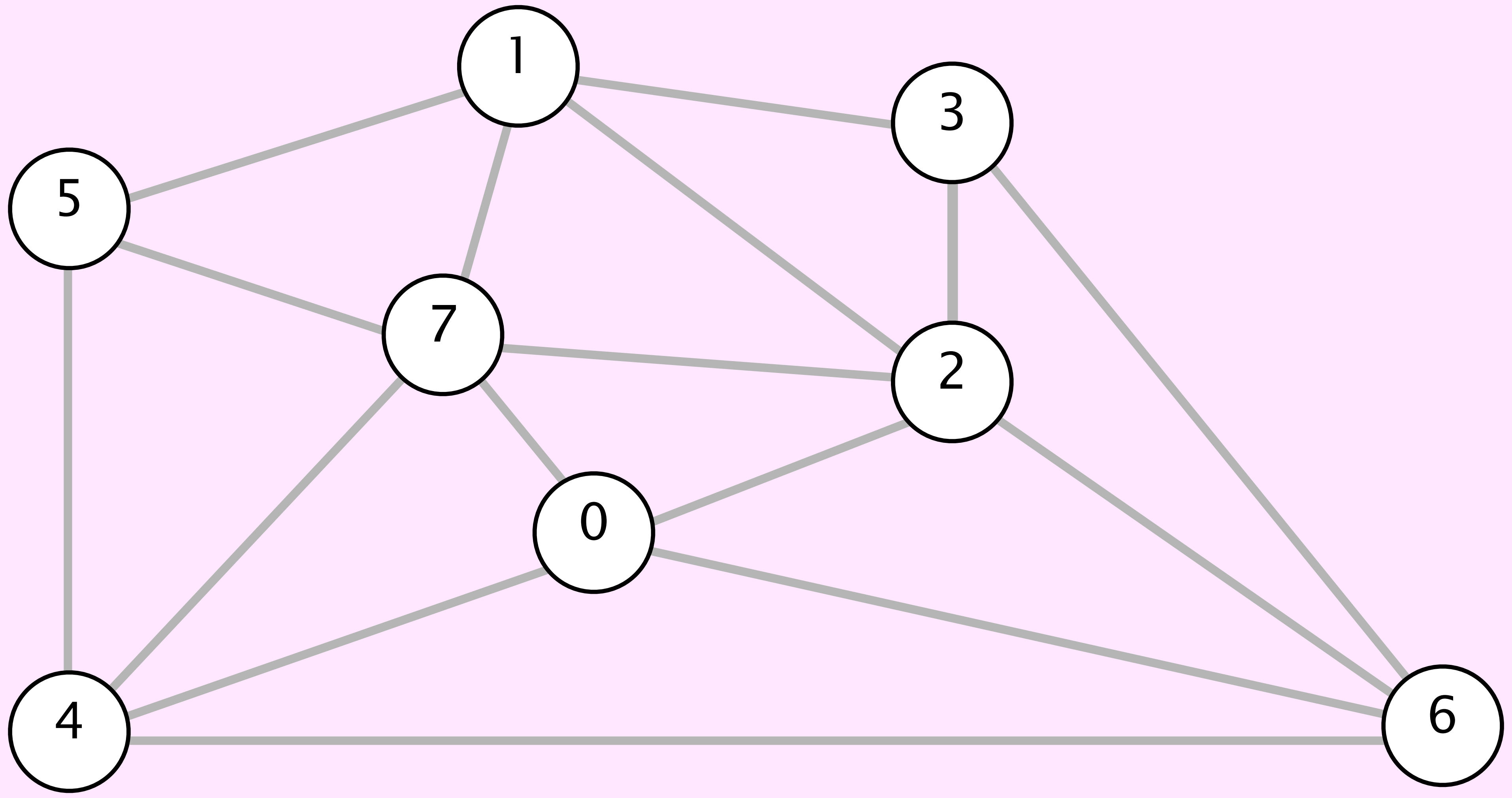
4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.



an edge-weighted graph

graph edges
sorted by weight

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

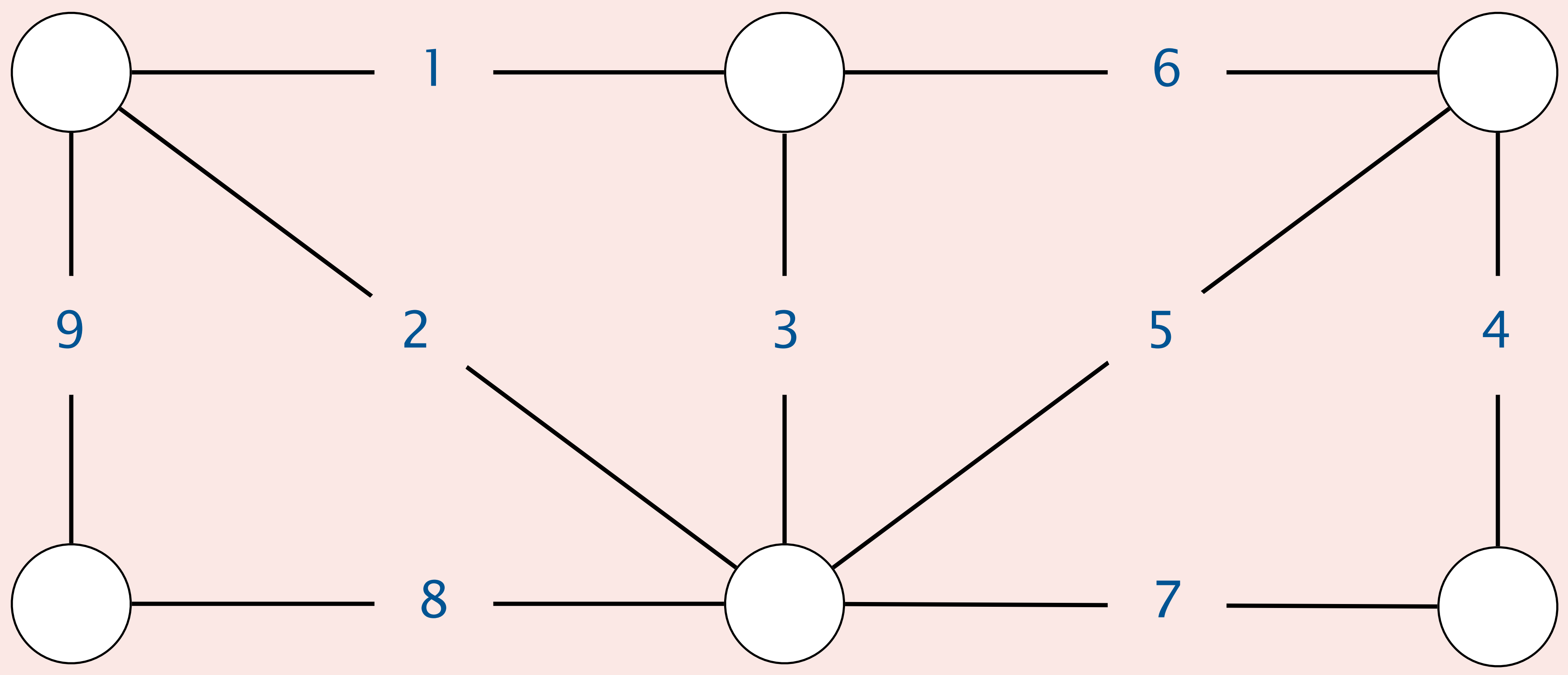
In which order does Kruskal's algorithm select edges in MST?

A. 1, 2, 4, 5, 6

B. 1, 2, 4, 5, 8

C. 1, 2, 5, 4, 8

D. 8, 2, 1, 5, 4



Kruskal's algorithm: correctness proof

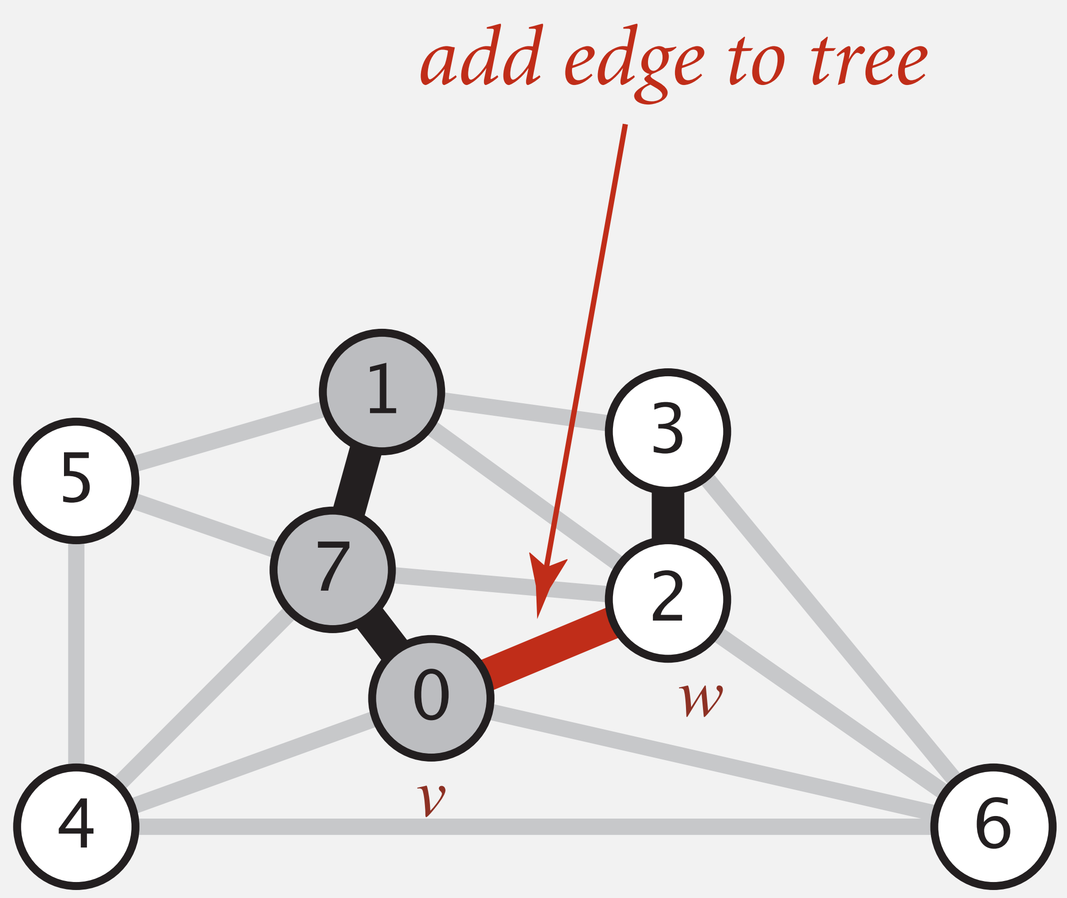
Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. Kruskal's algorithm adds edge e to T if and only if e is in the MST.

[Case 1 $\Rightarrow$] Kruskal's algorithm adds edge $e = v–w$ to T.

- Vertices v and w are in different connected components of T.
- Cut = set of vertices connected to v in T.
- By construction of cut, no crossing edge
  - is currently in T
  - was considered by Kruskal before e
- Thus, e is a min weight crossing edge.
- Cut property $\Rightarrow$ e is in the MST.

Kruskal considers edges in ascending order by weight



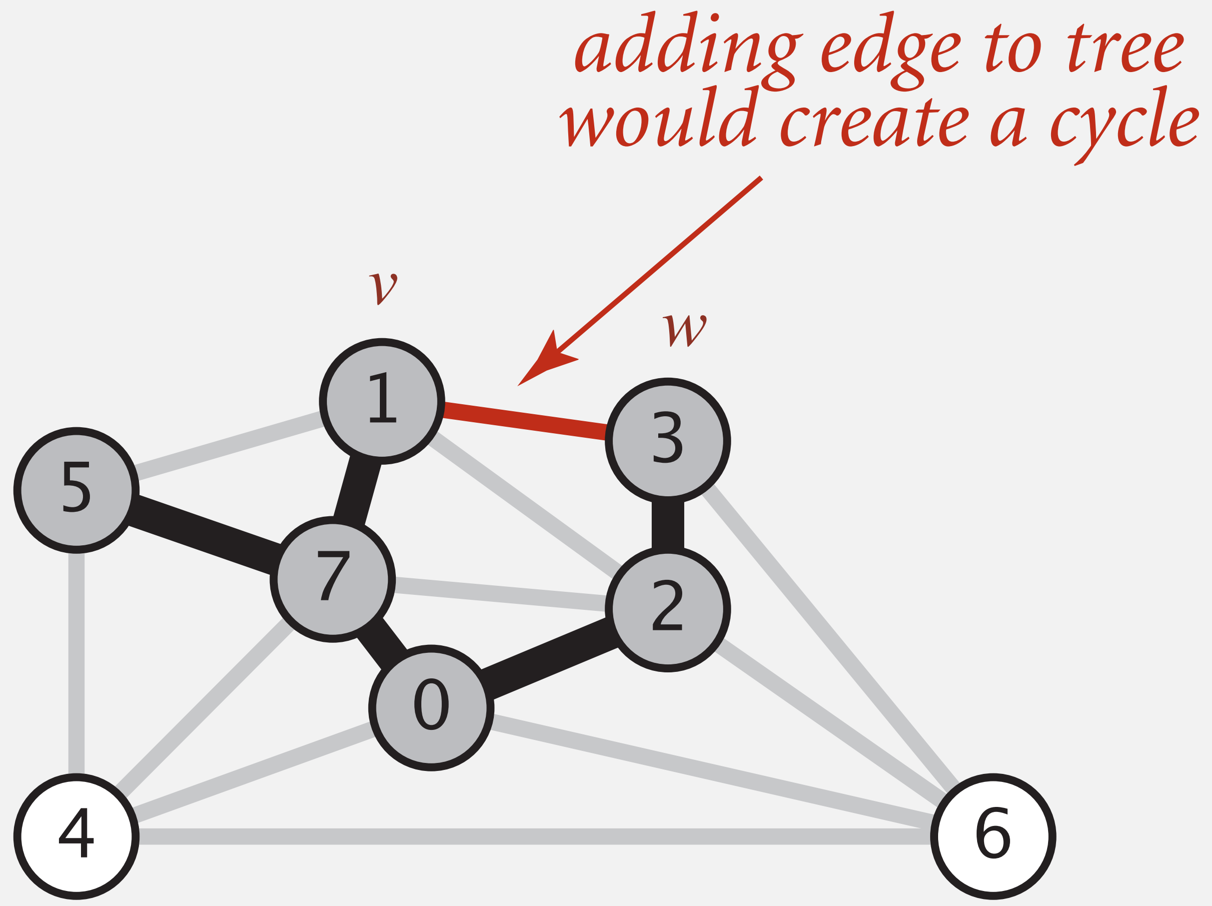
Kruskal's algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. Kruskal's algorithm adds edge e to T if and only if e is in the MST.

[Case 2 $\Leftarrow$] Kruskal's algorithm discards edge $e = v–w$.

- From Case 1, all edges currently in T are in the MST.
- The MST can't contain a cycle, so it can't also contain e. ▪

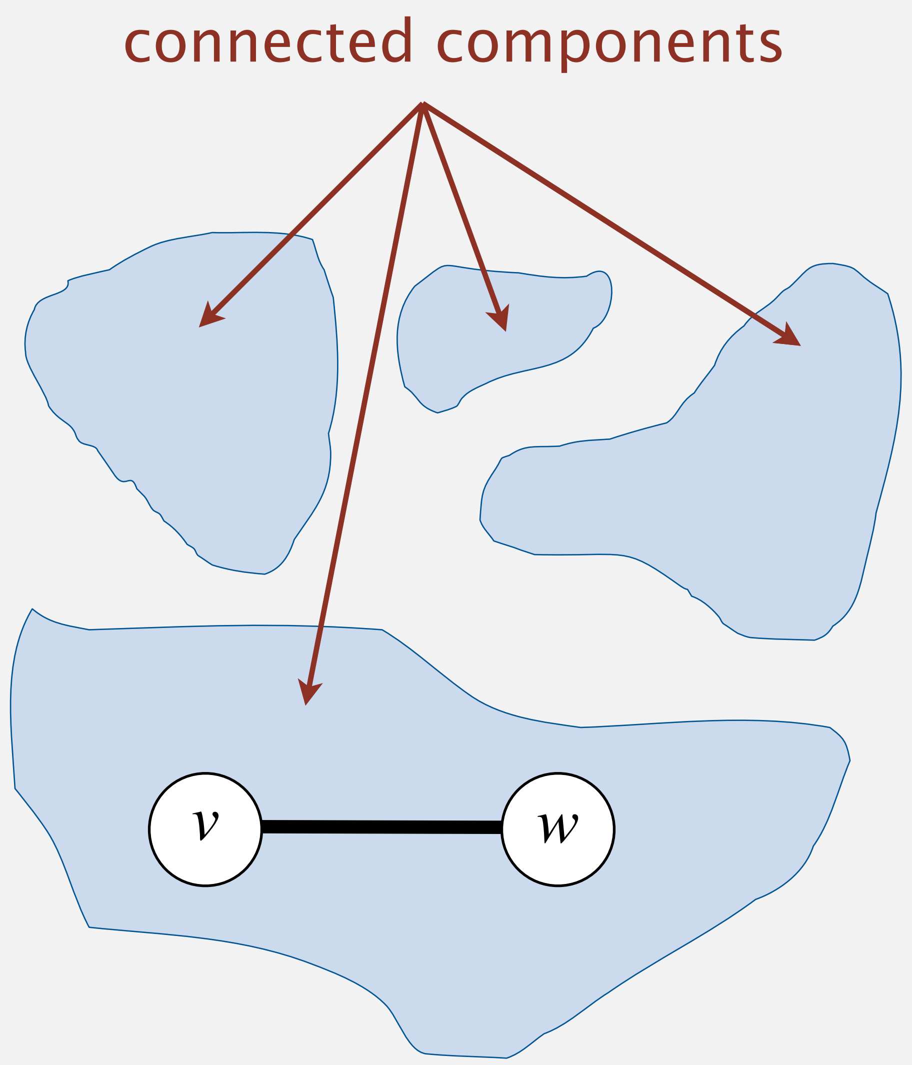


Kruskal's algorithm: implementation challenge

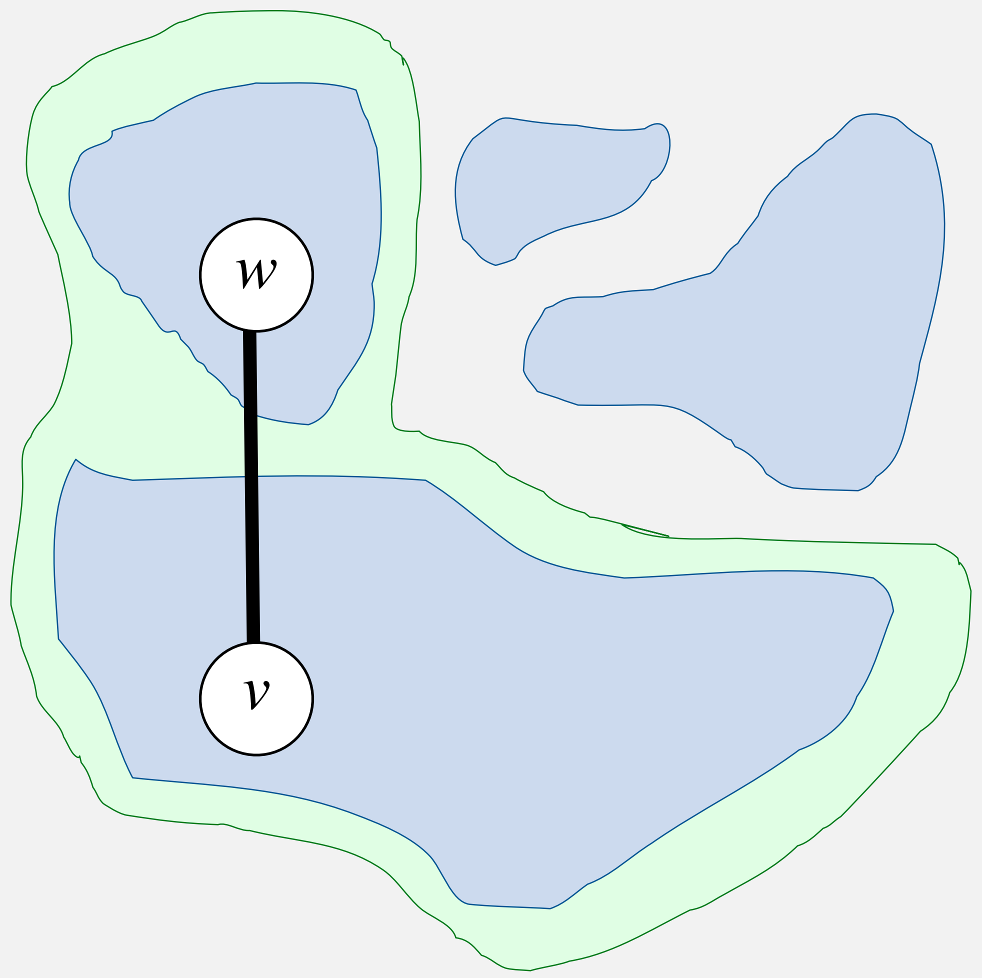
Challenge. Would adding edge v–w to T create a cycle? If not, add it.

Efficient solution. Use the union–find data structure.

- Maintain a set for each connected component in T.
- If v and w are in same set, then adding v–w to T would create a cycle. [Case 2]
- Otherwise, add v–w to T and merge sets containing v and w. [Case 1]



Case 2: adding v–w creates a cycle



Case 1: add v–w to T and merge sets containing v and w

Kruskal's algorithm: Java implementation

```
public class KruskalMST
{
   private Queue<Edge> mst = new Queue<>();

   public KruskalMST(EdgeWeightedGraph G)
   {
      Edge[] edges = G.edges();
      Arrays.sort(edges);
      UF uf = new UF(G.V());

      for (int i = 0; i < G.E(); i++)
      {
         Edge e = edges[i];
         int v = e.either(), w = e.other(v);
         if (uf.find(v) != uf.find(w))
         {
            mst.enqueue(e);
            uf.union(v, w);
         }
      }
   }

   public Iterable<Edge> edges()
   {  return mst;  }
}
```

- edges in the MST
- sort edges by weight
- maintain connected components
- optimization: stop as soon as $V-1$ edges in T
- greedily add edges to MST
- edge v–w does not create cycle
- add edge e to MST
- merge connected components

Kruskal's algorithm: running time

Proposition. In the worst case, Kruskal's algorithm computes the MST in an edge-weighted graph in $\Theta(E \log E)$ time and $\Theta(E)$ extra space.

Pf.

- Bottlenecks are sort and union–find operations.

operation	frequency	time per op
SORT	1	$E \log E$
UNION	$V - 1$	$\log V$ †
FIND	$2\,E$	$\log V$ †

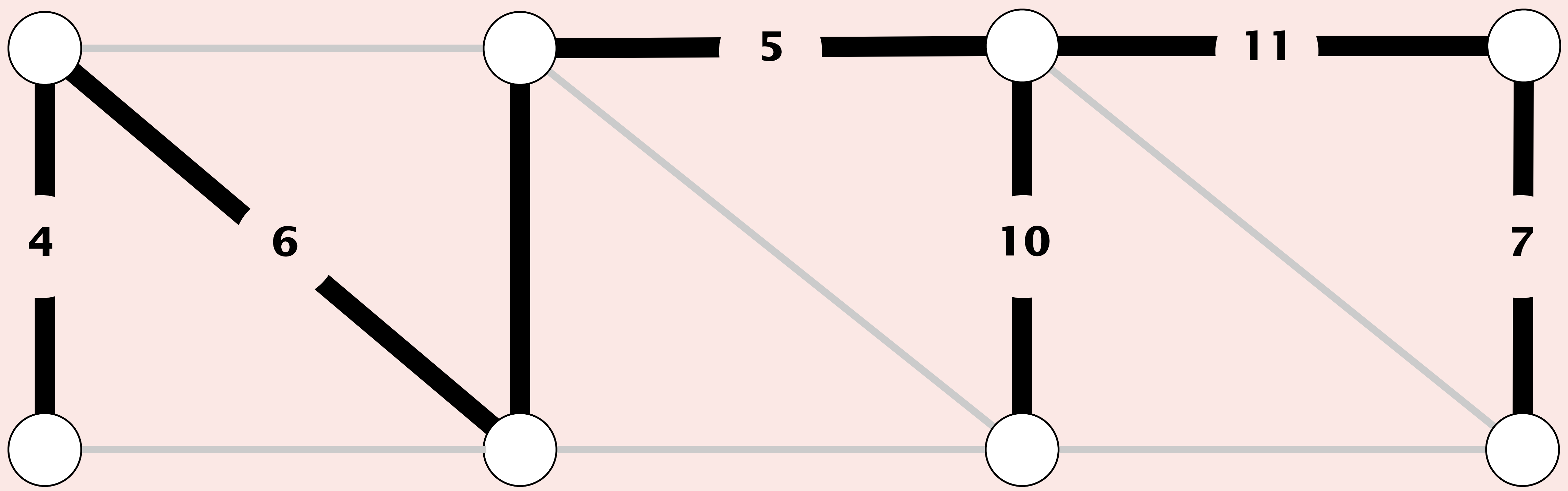
† using weighted quick union

- Total. $\Theta(V \log V)$ + $\Theta(E \log V)$ + $\Theta(E \log E)$.

dominated by $\Theta(E \log E)$ since graph is connected

Given a graph with positive edge weights, how to find a spanning tree that minimizes the sum of the squares of the edge weights?

- **A.** Run Kruskal's algorithm using the original edge weights.
- **B.** Run Kruskal's algorithm using the squares of the edge weights.
- **C.** Run Kruskal's algorithm using the square roots of the edge weights.
- **D.** All of the above.

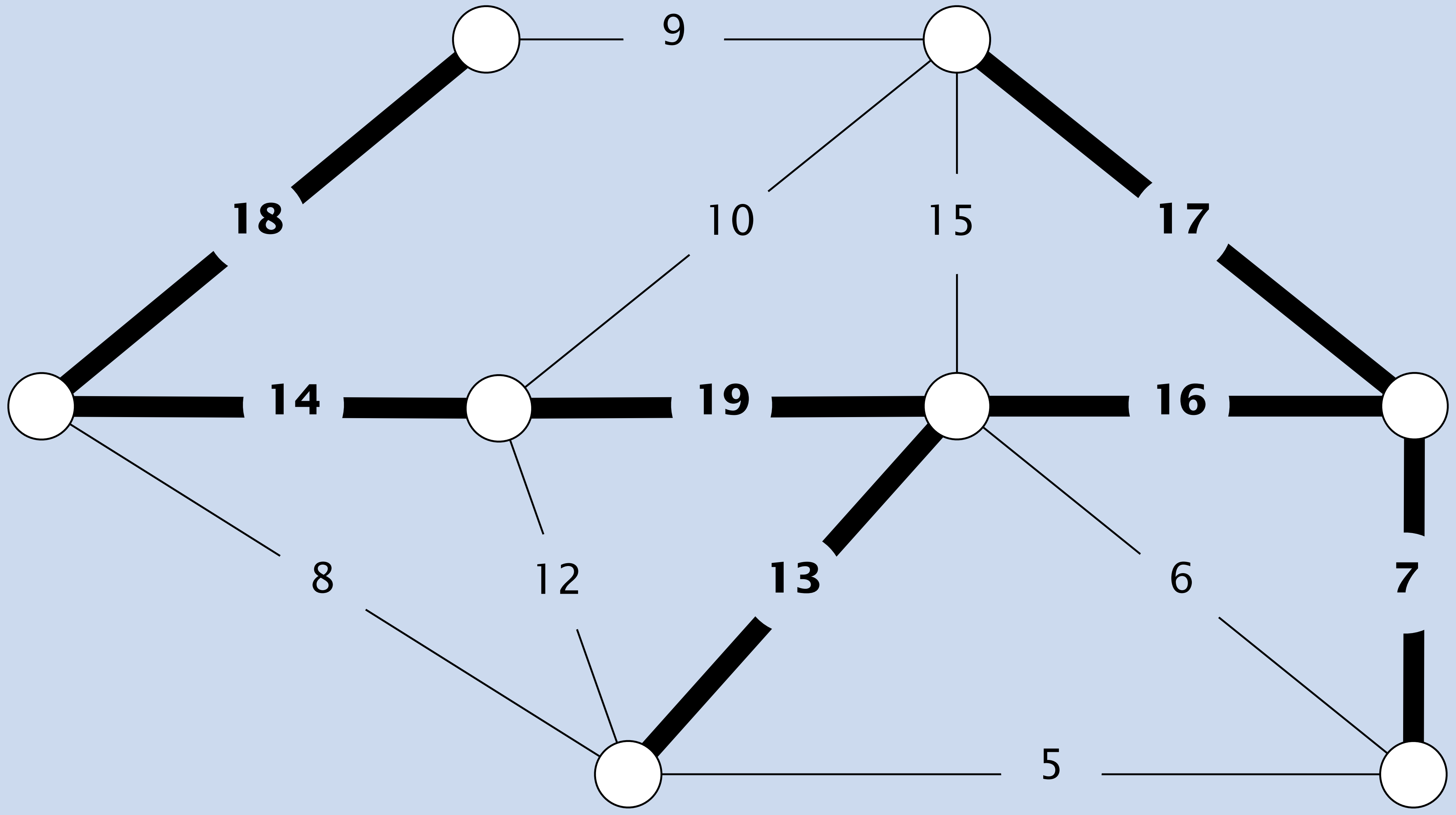


sum of squares $= 4^2 + 6^2 + 5^2 + 10^2 + 11^2 + 7^2 = 347$

Maximum Spanning Tree

Problem. Given an undirected graph G with positive edge weights, find a spanning tree that maximizes the sum of the edge weights.

Goal. Design algorithm that takes $\Theta(E \log E)$ time in the worst case.



maximum spanning tree T (weight = 104)

Greed is good

Gordon Gecko (Michael Douglas) evangelizing the importance of greed

Wall Street (1986)

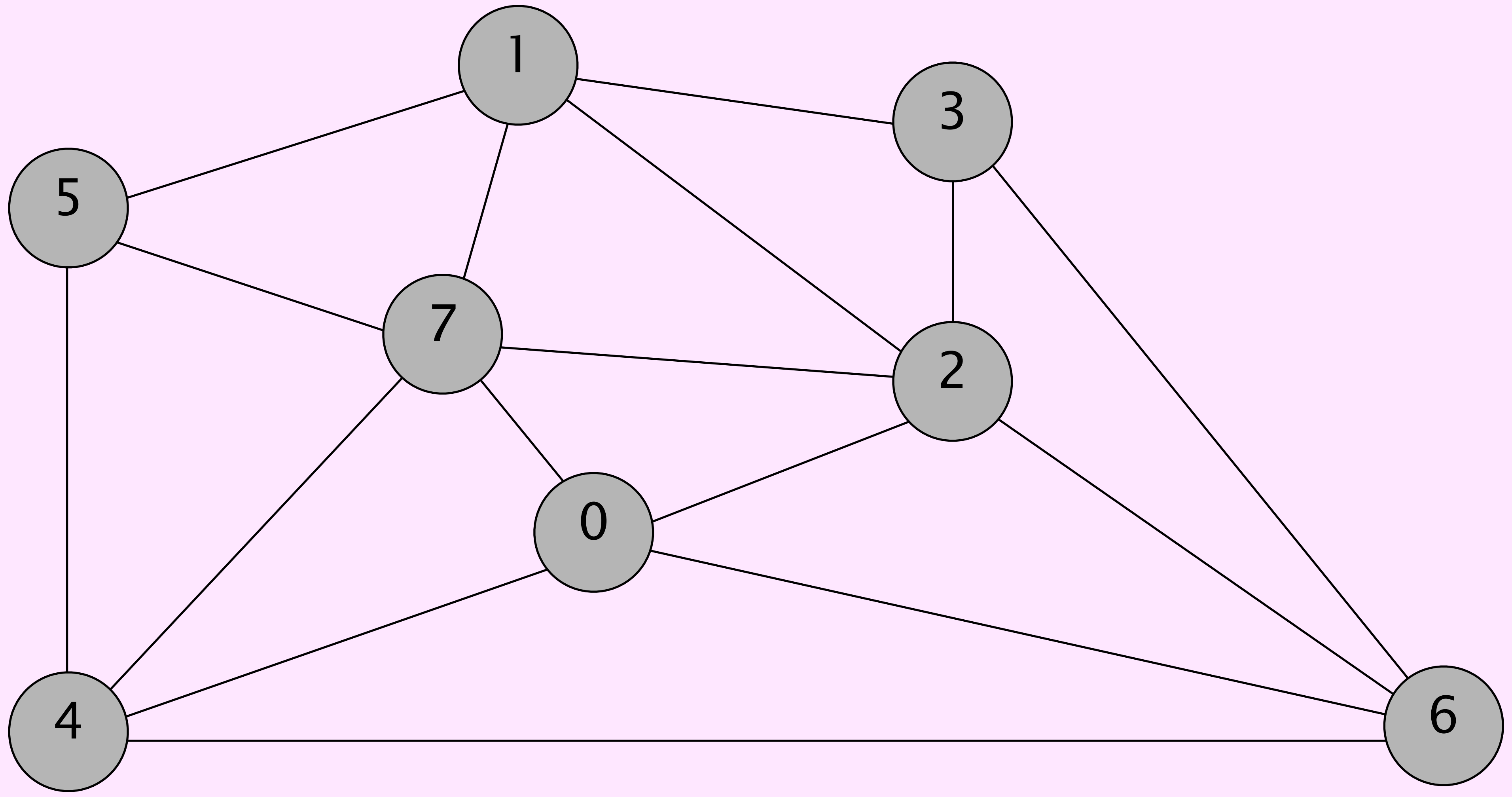
https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Prim's algorithm demo

- Start with vertex 0 and grow tree T.
- Repeat until $V - 1$ edges:
  - add to T the min-weight edge with exactly one endpoint in T



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

In which order does Prim's algorithm select edges in the MST?

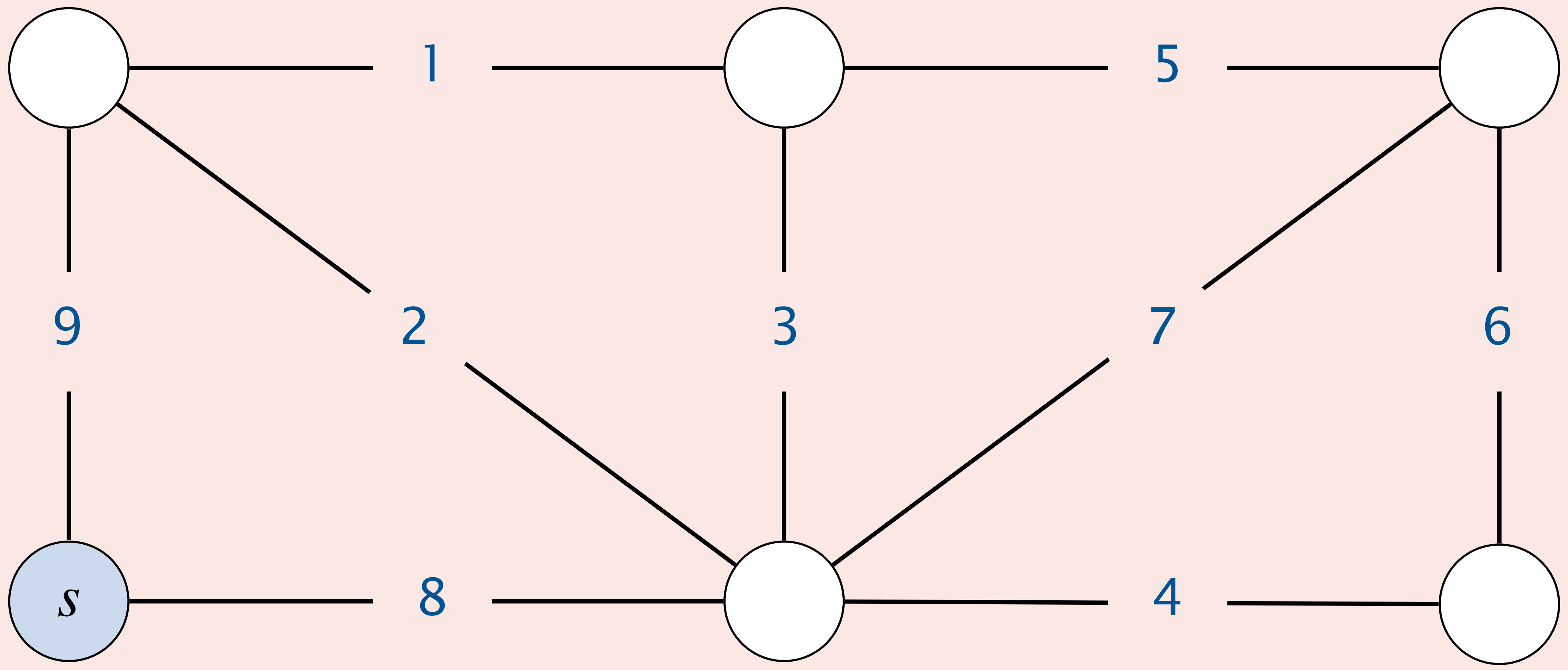
Assume it starts from vertex s.

A. 8, 2, 1, 4, 5

B. 8, 2, 1, 5, 4

C. 8, 2, 1, 5, 6

D. 8, 2, 3, 4, 5



Prim's algorithm: proof of correctness

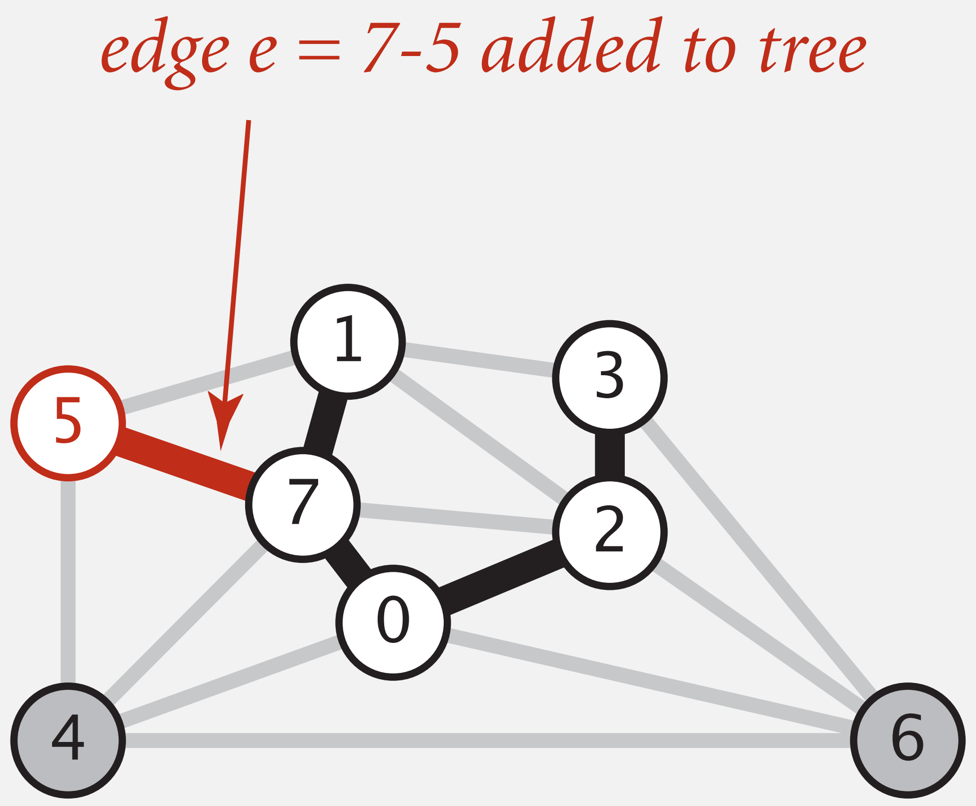
Proposition. [Jarník 1930, Dijkstra 1957, Prim 1959]

Prim's algorithm computes the MST.

Pf. Let e = min-weight edge with exactly one endpoint in T.

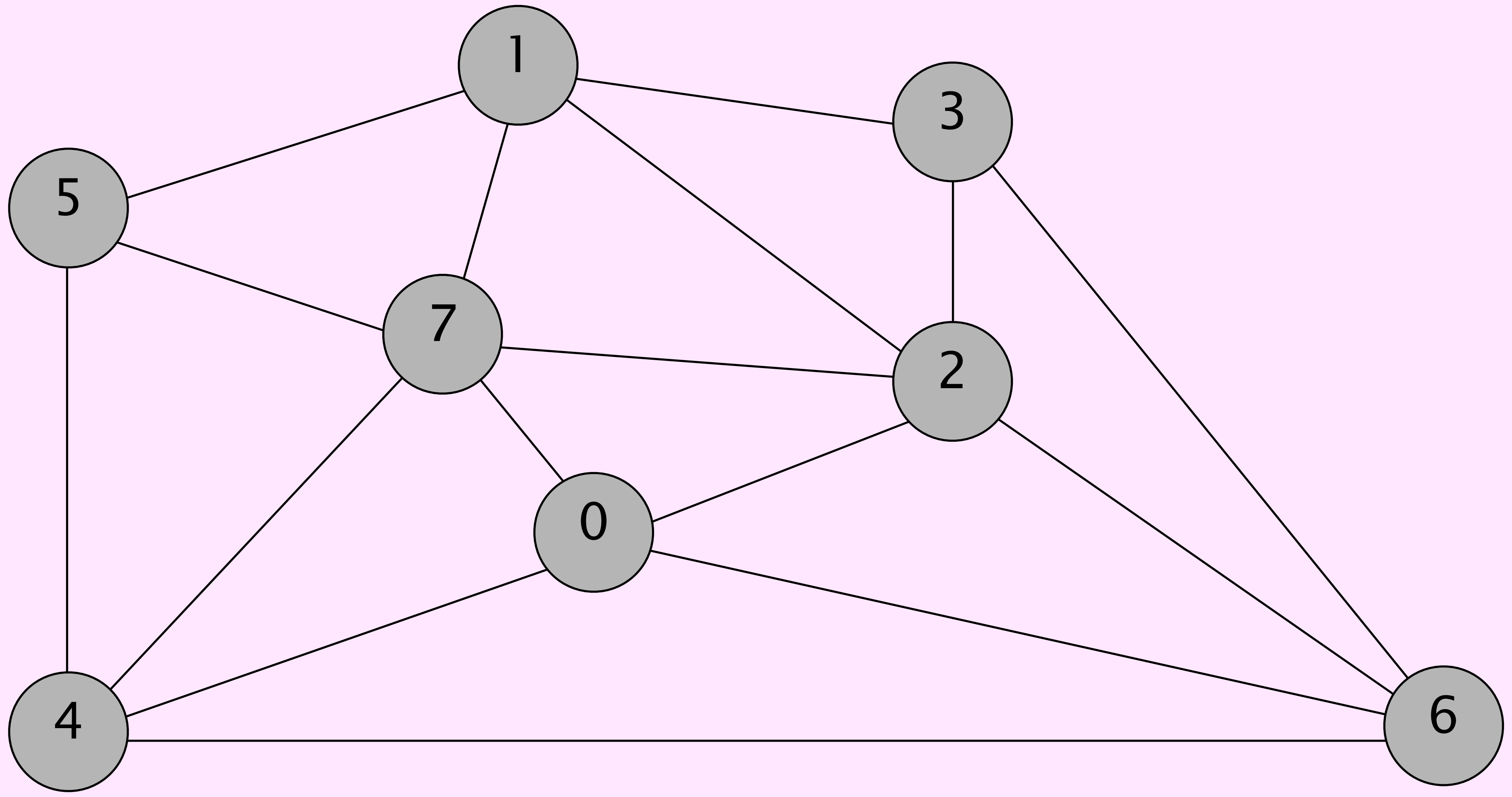
- Cut = set of vertices in T.
- Cut property $\Rightarrow$ edge e is in the MST. ▪

Challenge. How to efficiently find min-weight edge with exactly one endpoint in T?



Prim's algorithm: lazy implementation demo

- Start with vertex 0 and grow tree T.
- Repeat until $V - 1$ edges:
  - add to T the min-weight edge with exactly one endpoint in T



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

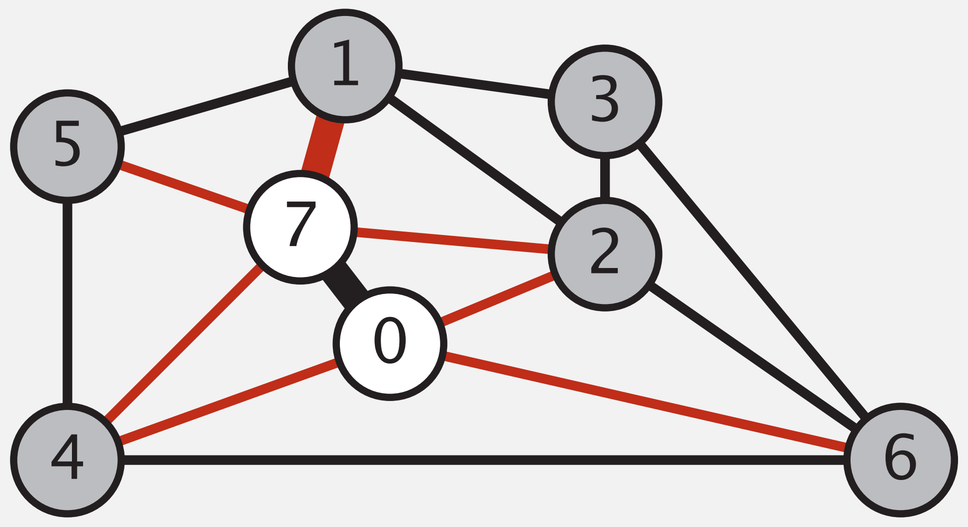
Prim's algorithm: lazy implementation

Challenge. How to efficiently find min-weight edge with exactly one endpoint in T?

Lazy solution. Maintain a PQ of edges with (at least) one endpoint in T.

- Key = edge; priority = weight of edge.
- DELETE-MIN to determine next edge $e = v–w$ to add to T.
- If both endpoints v and w are marked (both in T), disregard.
- Otherwise, let w be the unmarked vertex (not in T):
  - add e to T and mark w
  - add to PQ any edge incident to w ← but don't bother if other endpoint is in T

1-7 is min weight edge with exactly one endpoint in T



priority queue of crossing edges

```
1-7 0.19
0-2 0.26
5-7 0.28
2-7 0.34
4-7 0.37
0-4 0.38
6-0 0.58
```

Prim's algorithm: lazy implementation

```
public class LazyPrimMST
{
   private boolean[] marked;   // MST vertices
   private Queue<Edge> mst;    // MST edges
   private MinPQ<Edge> pq;     // PQ of edges

    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<>();
        mst = new Queue<>();
        marked = new boolean[G.V()];
        visit(G, 0);
```

assume graph G is connected

```
        while (mst.size() < G.V() - 1)
        {
           Edge e = pq.delMin();
           int v = e.either(), w = e.other(v);
           if (marked[v] && marked[w]) continue;
           mst.enqueue(e);
           if (!marked[v]) visit(G, v);
           if (!marked[w]) visit(G, w);
        }
    }
}
```

repeatedly delete the min-weight edge $e = v–w$ from PQ

ignore if both endpoints in tree T

add edge e to tree T

add either v or w to tree T

```
private void visit(WeightedGraph G, int v)
{
  marked[v] = true;
  for (Edge e : G.adj(v))
     if (!marked[e.other(v)])
        pq.insert(e);
}
```

add v to tree T

for each edge $e = v–w$: add e to PQ if w not already in T

```
public Iterable<Edge> mst()
{  return mst;  }
```

Lazy Prim's algorithm: running time

Proposition. In the worst case, lazy Prim's algorithm computes the MST in $\Theta(E \log E)$ time and $\Theta(E)$ extra space.

Pf.

- Bottlenecks are PQ operations.
- Each edge is added to PQ at most once.
- Each edge is deleted from PQ at most once.

operation	frequency	binary heap
INSERT	E	$\log E$
DELETE-MIN	E	$\log E$

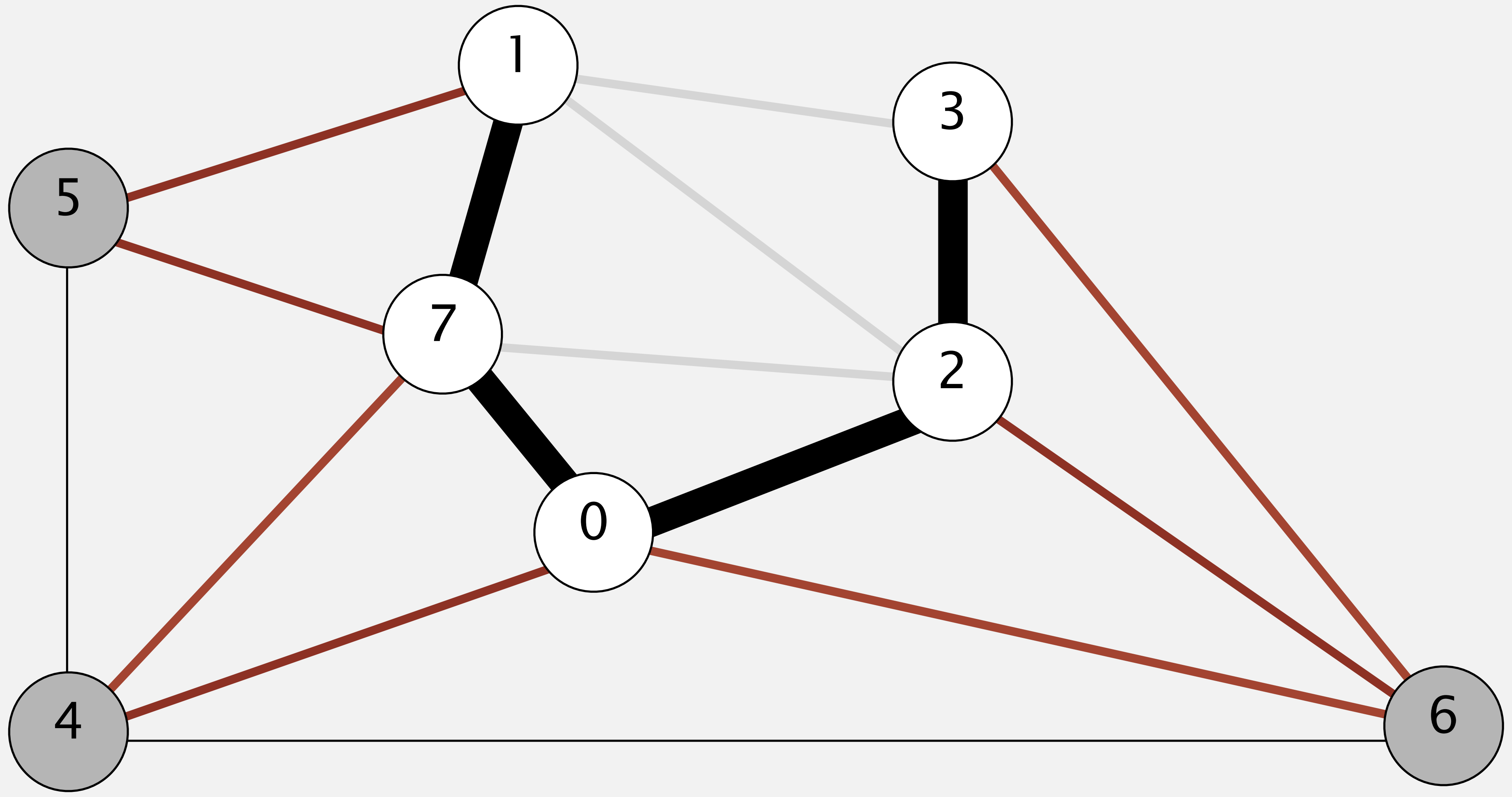
Prim's algorithm: eager implementation

Challenge. Find min-weight edge with exactly one endpoint in T.

Observation. For each vertex v, need only min-weight edge connecting v to T.

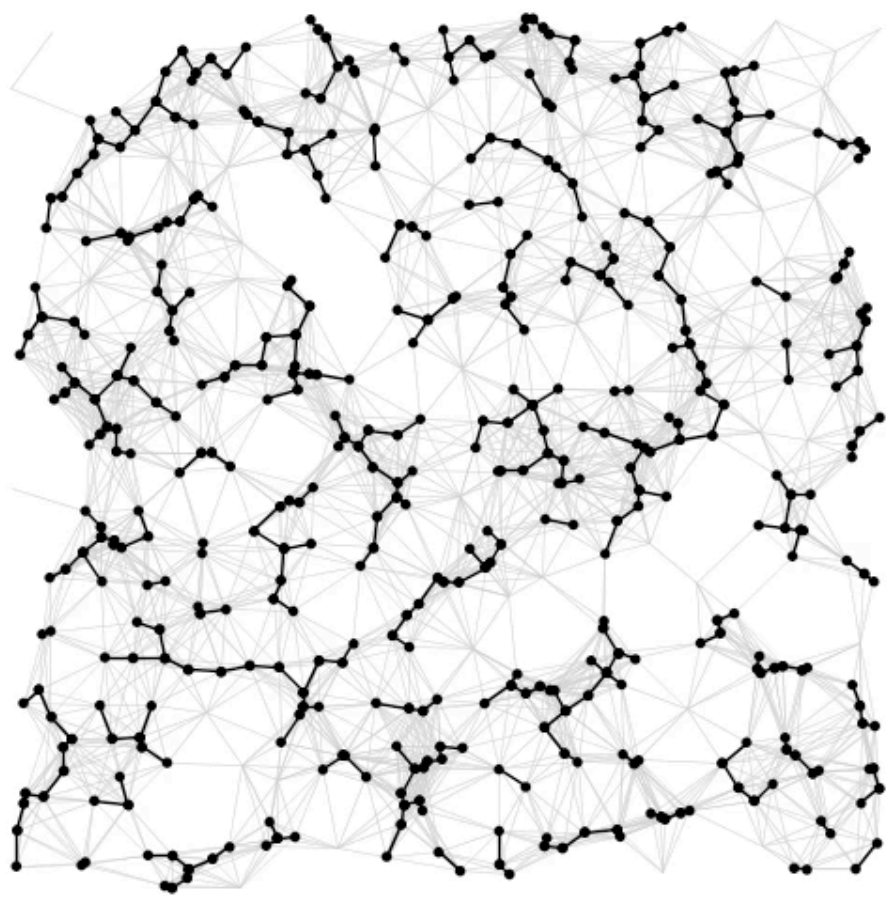
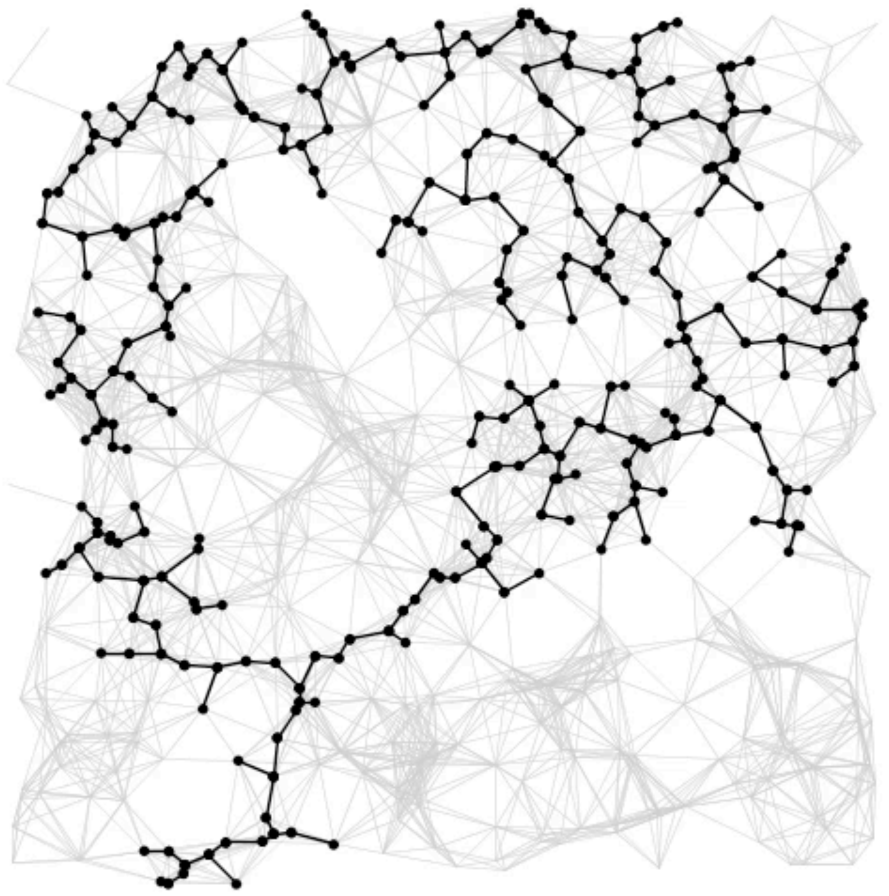
- MST includes at most one edge connecting v to T. Why?
- If MST includes such an edge, it must take lightest such edge. Why?

Impact. PQ of vertices; $\Theta(V)$ extra space; $\Theta(E \log V)$ running time in worst case.



see textbook for details

MST: algorithms of the day

algorithm	visualization	bottleneck	running time
Kruskal		*sorting* *union–find*	$E \log E$
Prim		*priority queue*	$E \log V$

© Copyright 2021 Robert Sedgewick and Kevin Wayne