https://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

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

Last updated on 4/4/19 10:35 AM

https://algs4.cs.princeton.edu

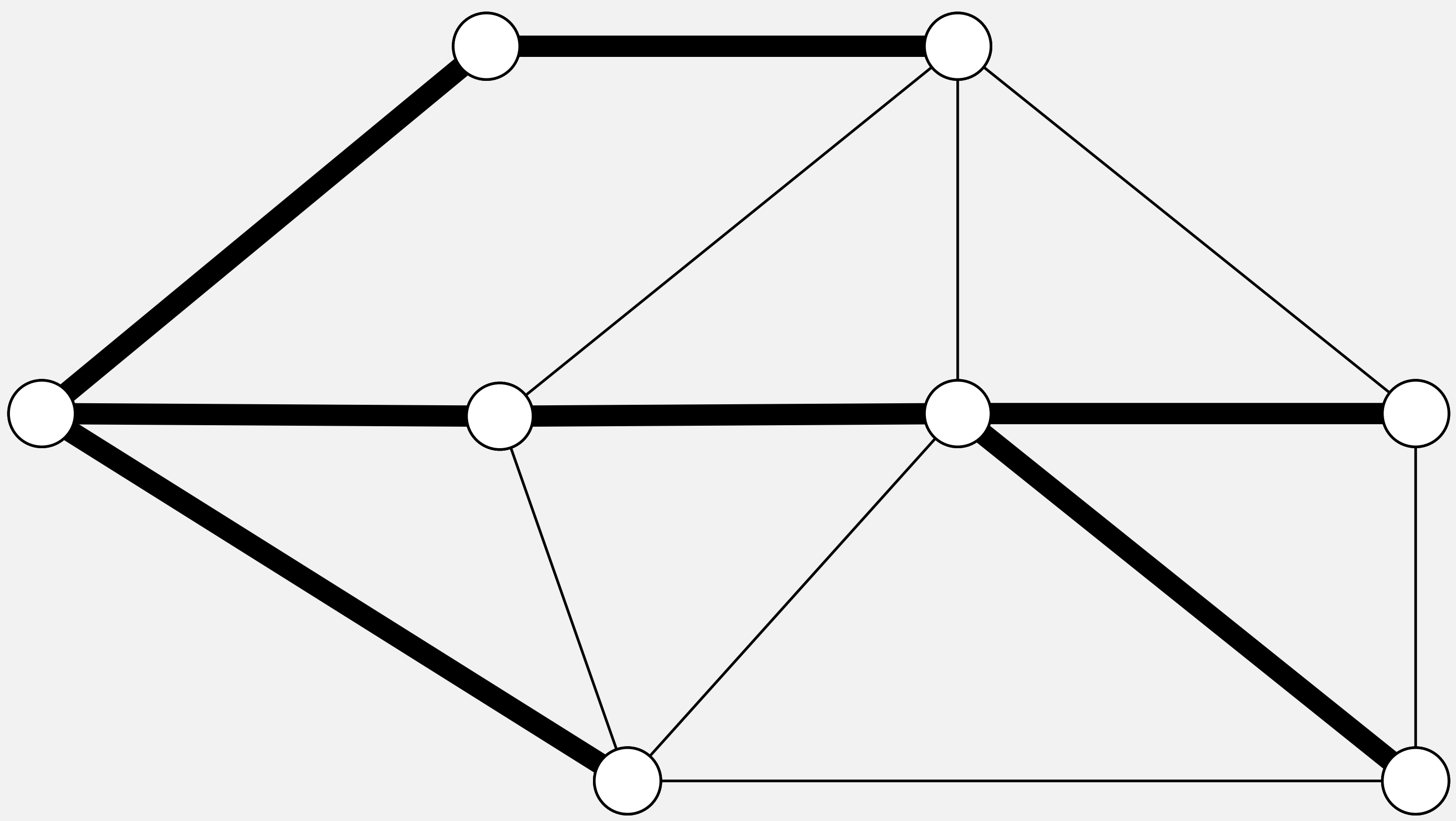
4.3 Minimum Spanning Trees

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

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.

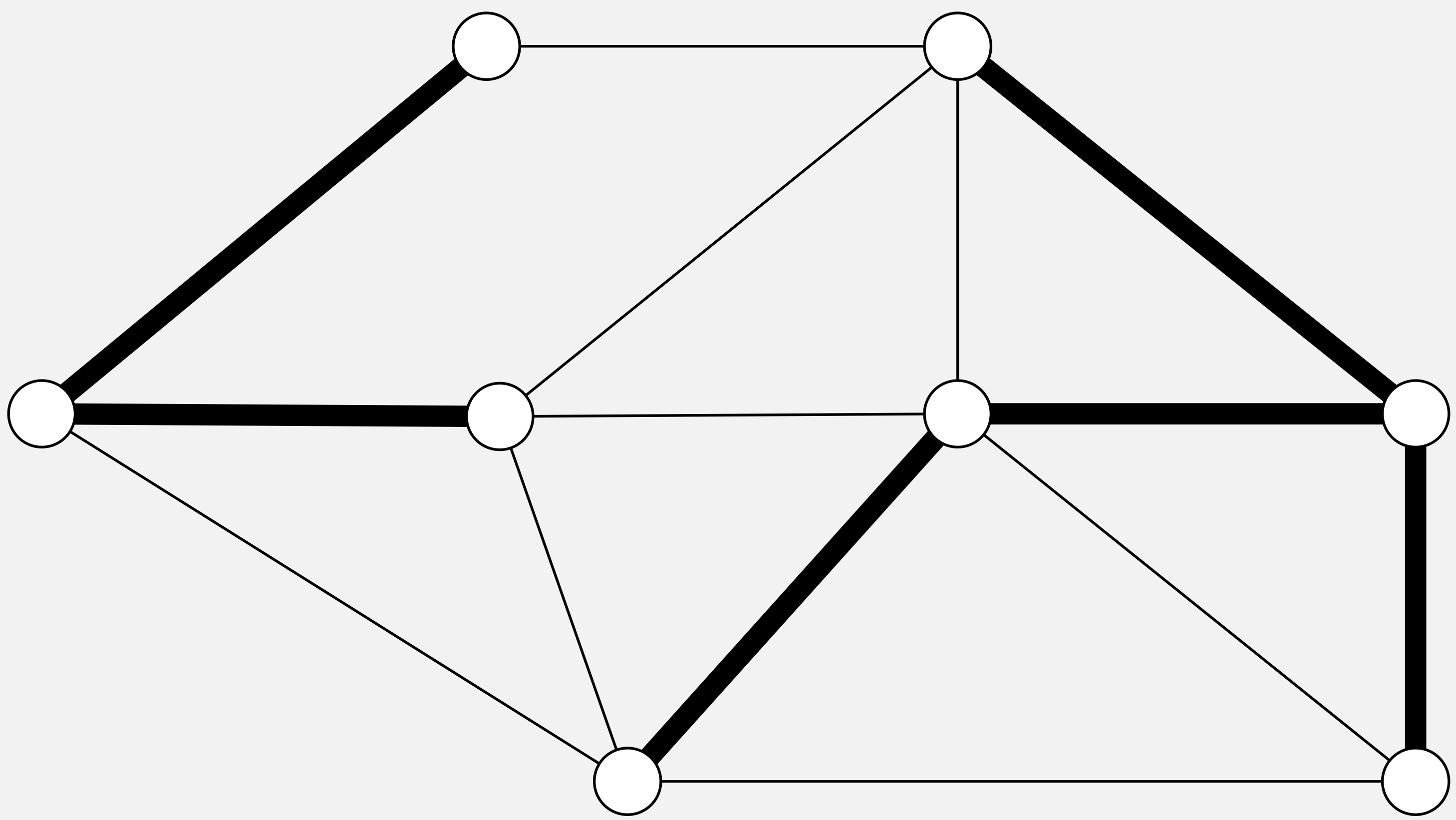


graph G
spanning tree T

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.

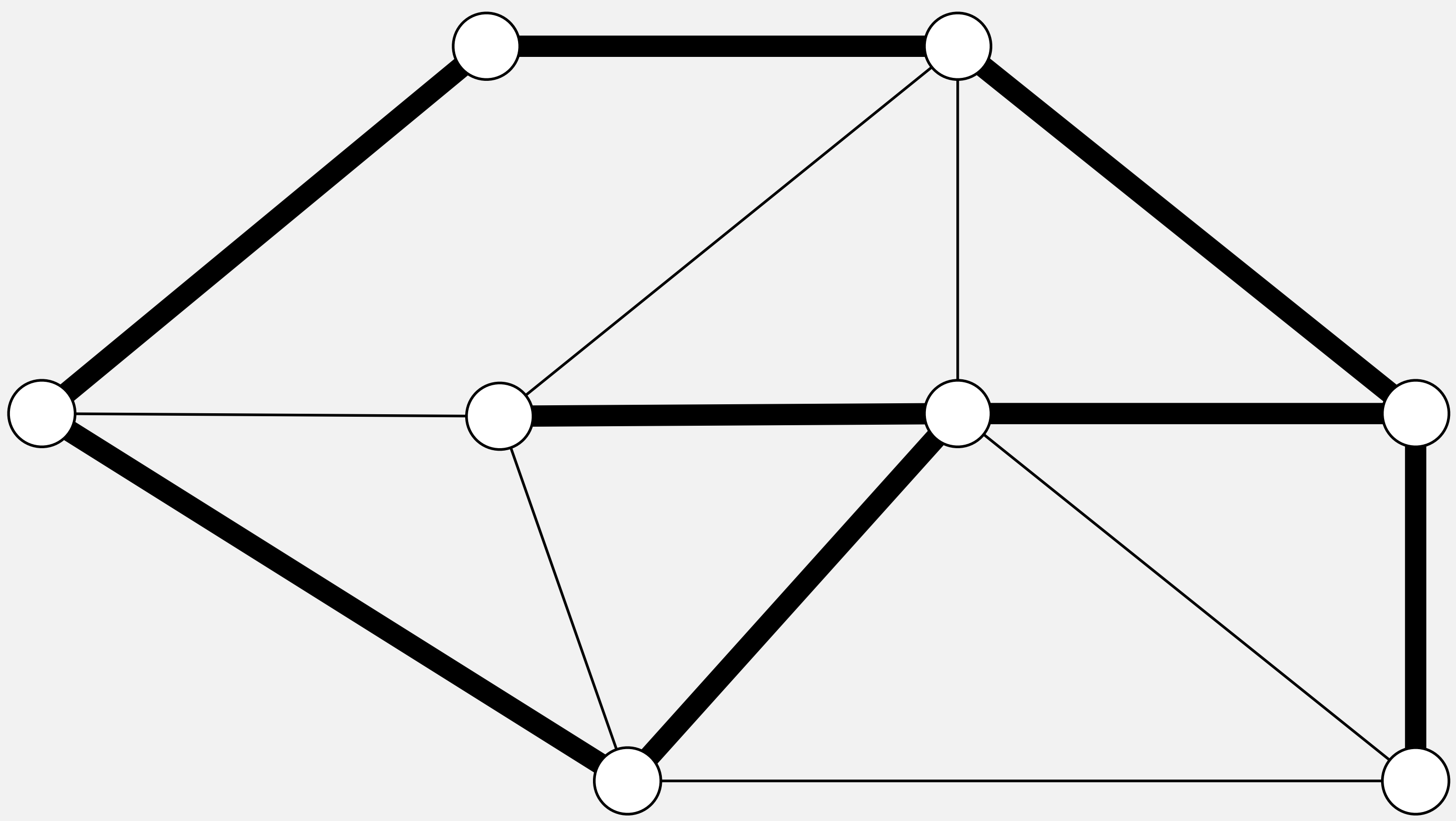


not connected

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.

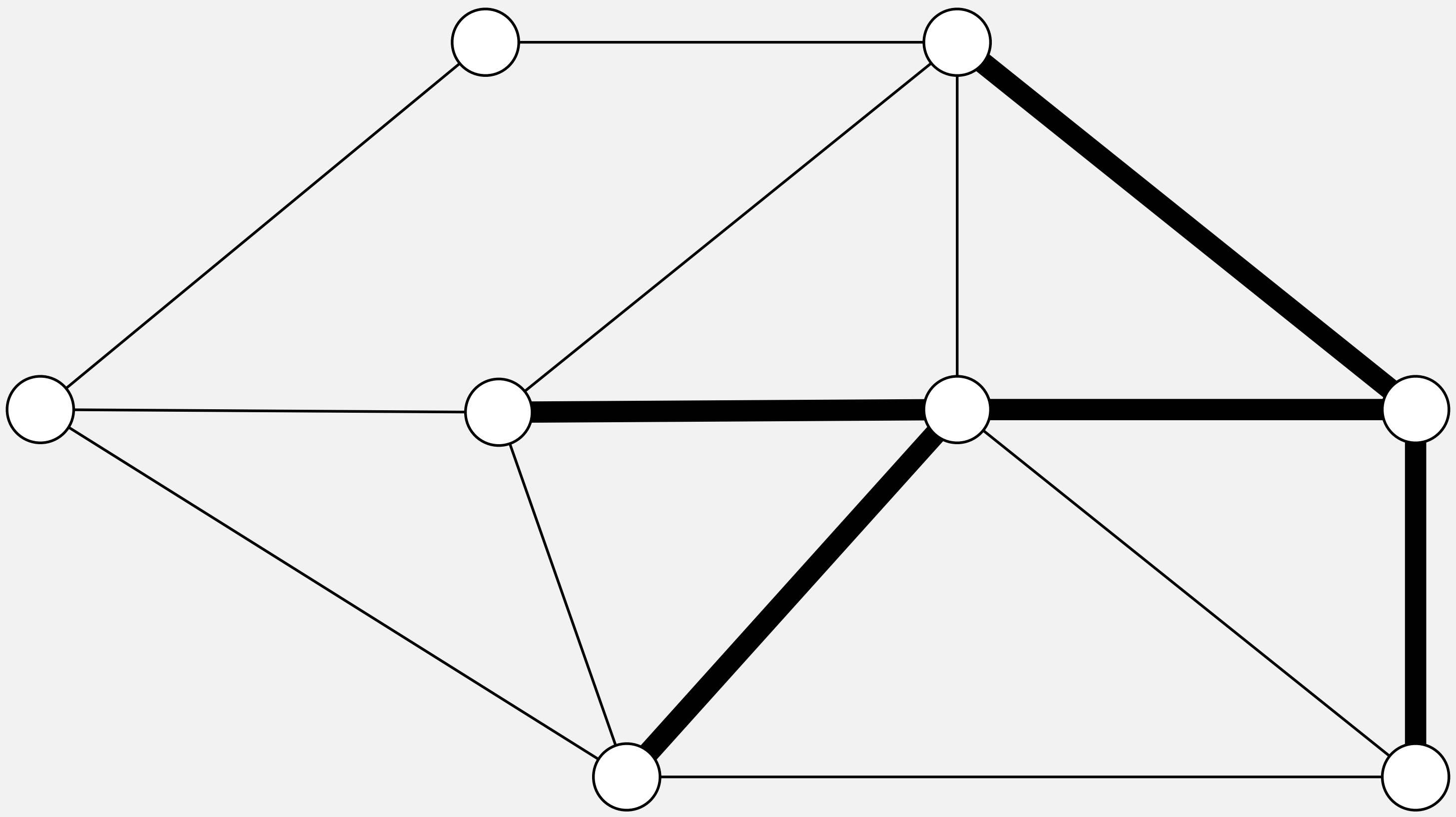


not acyclic

Spanning tree

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

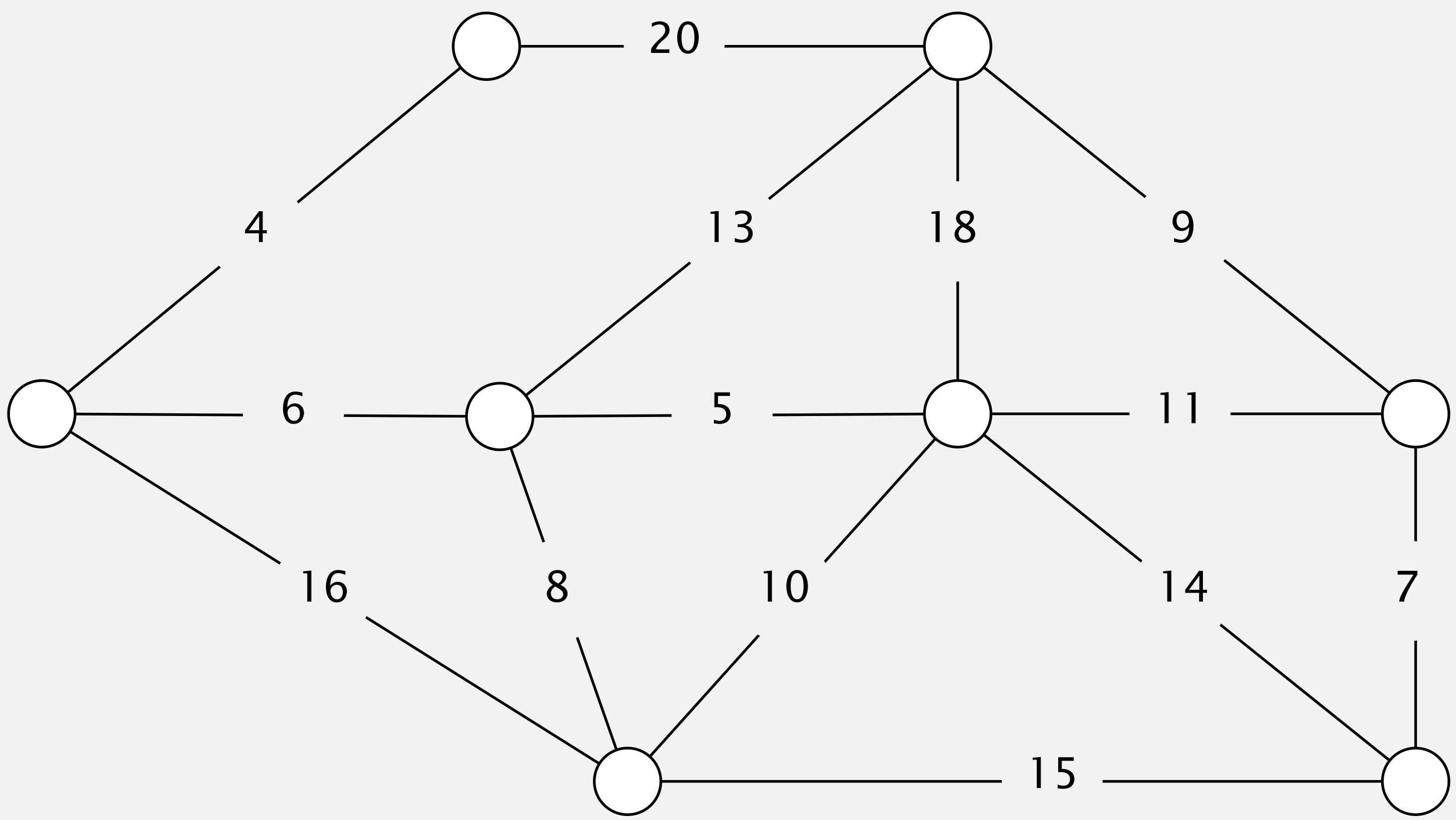
- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



not spanning

Minimum spanning tree problem

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

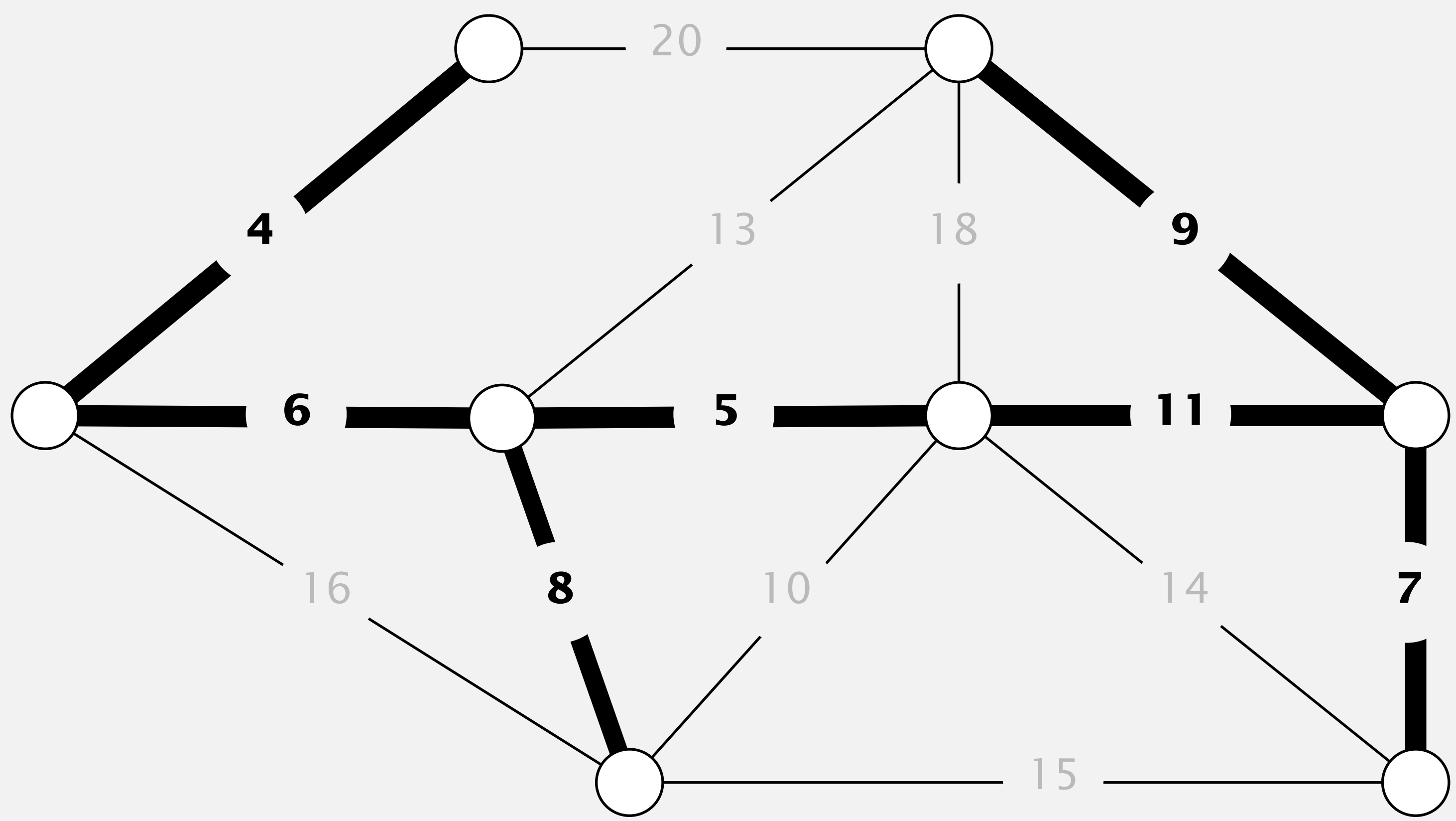


edge-weighted graph G

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 + 8 + 5 + 11 + 9 + 7)

Brute force. Try all spanning trees?

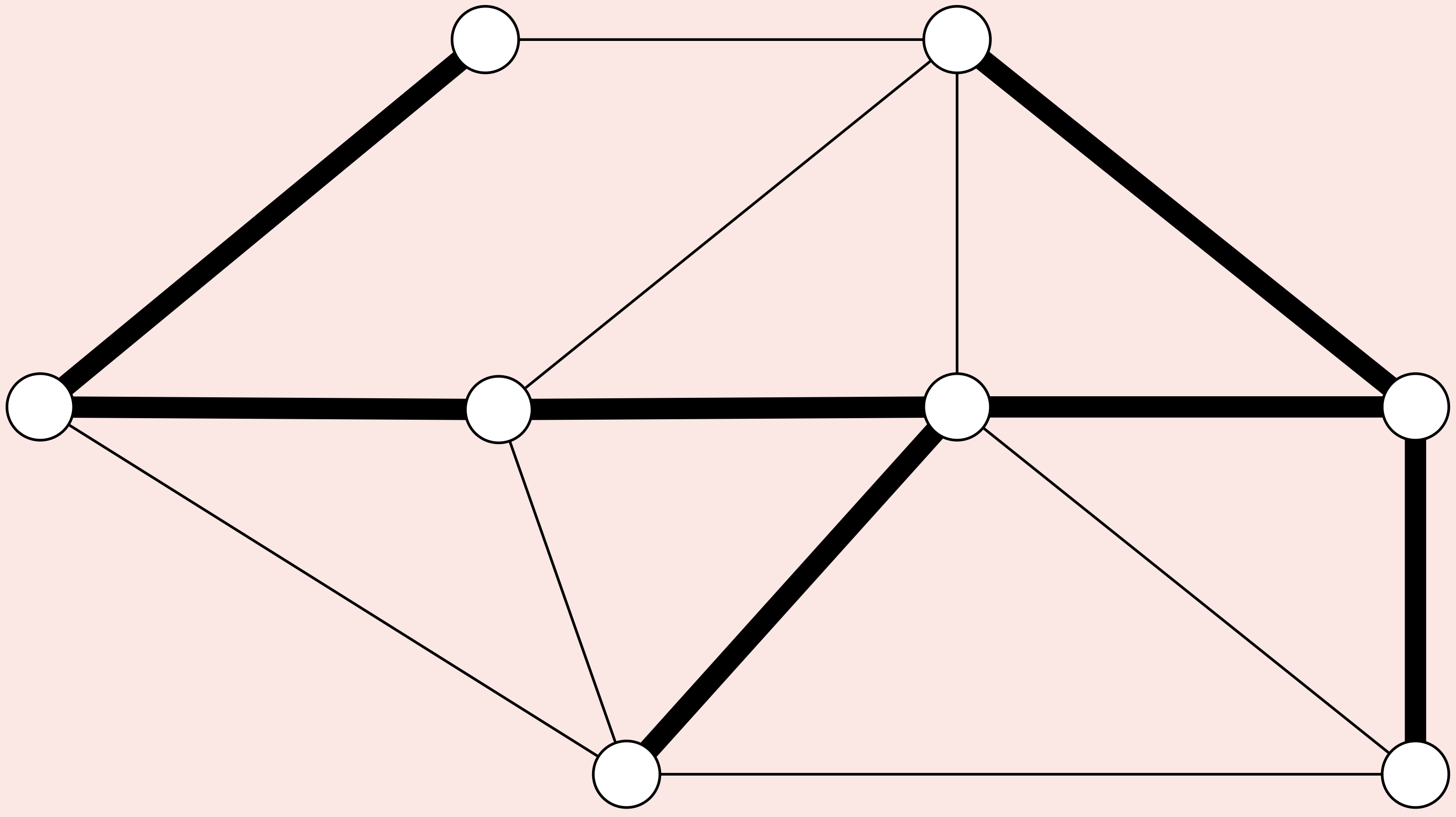
Let T be a spanning tree of a connected graph G with V vertices. Which of the following statements are true?

A. T contains exactly $V-1$ edges.

B. Removing any edge from T disconnects it.

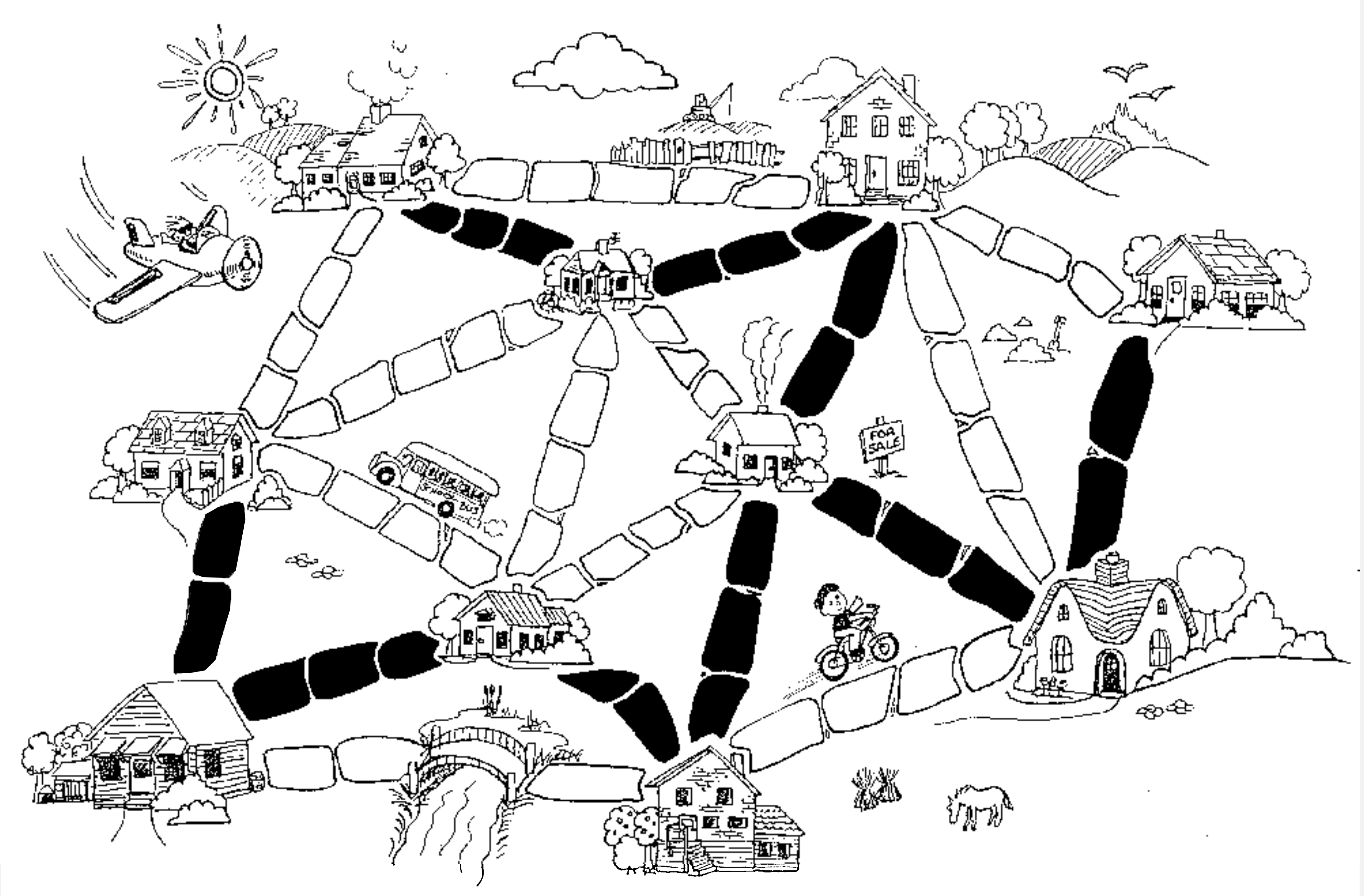
C. Adding any edge to T creates a cycle.

D. All of the above.



spanning tree T of graph G

Network design

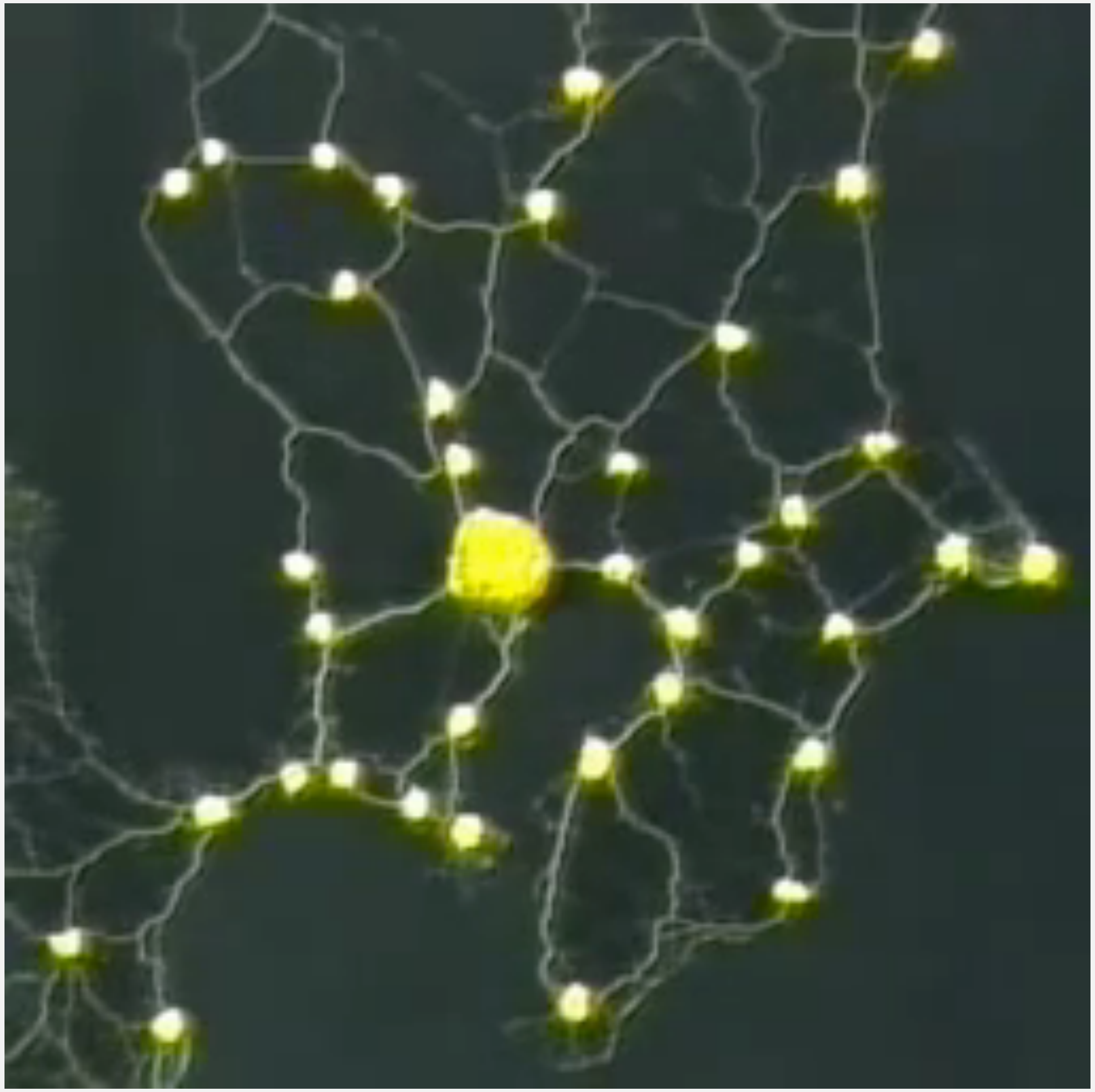


http://www.utdallas.edu/~besp/teaching/mst-applications.pdf

Slime mold grows network just like Tokyo rail system

Rules for Biologically Inspired Adaptive Network Design

Atsushi Tero,[1,2] Seiji Takagi,[1] Tetsu Saigusa,[3] Kentaro Ito,[1] Dan P. Bebber,[4] Mark D. Fricker,[4] Kenji Yumiki,[5] Ryo Kobayashi,[5,6] Toshiyuki Nakagaki[1,6]*



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

Applications

MST is fundamental problem with diverse applications.

- Cluster analysis.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Curvilinear feature extraction in computer vision.
- Find road networks in satellite and aerial imagery.
- Handwriting recognition of mathematical expressions.
- Measuring homogeneity of two-dimensional materials.

  Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Network design (communication, electrical, hydraulic, computer, road).
- Approximation algorithms for **NP**-hard problems (e.g., TSP, Steiner tree).

http://www.ics.uci.edu/~eppstein/gina/mst.html

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

https://algs4.cs.princeton.edu

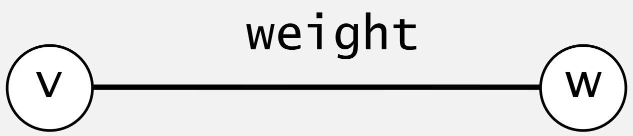
4.3 Minimum Spanning Trees

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

Weighted edge API

Edge abstraction needed 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 this edge to that edge*
`double`	`weight()`	*the weight*
`String`	`toString()`	*string representation*



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)
   {
     if      (this.weight < that.weight) return -1;
     else if (this.weight > that.weight) return +1;
     else                                return  0;
   }
}
```

constructor

either endpoint

other endpoint

compare edges by weight

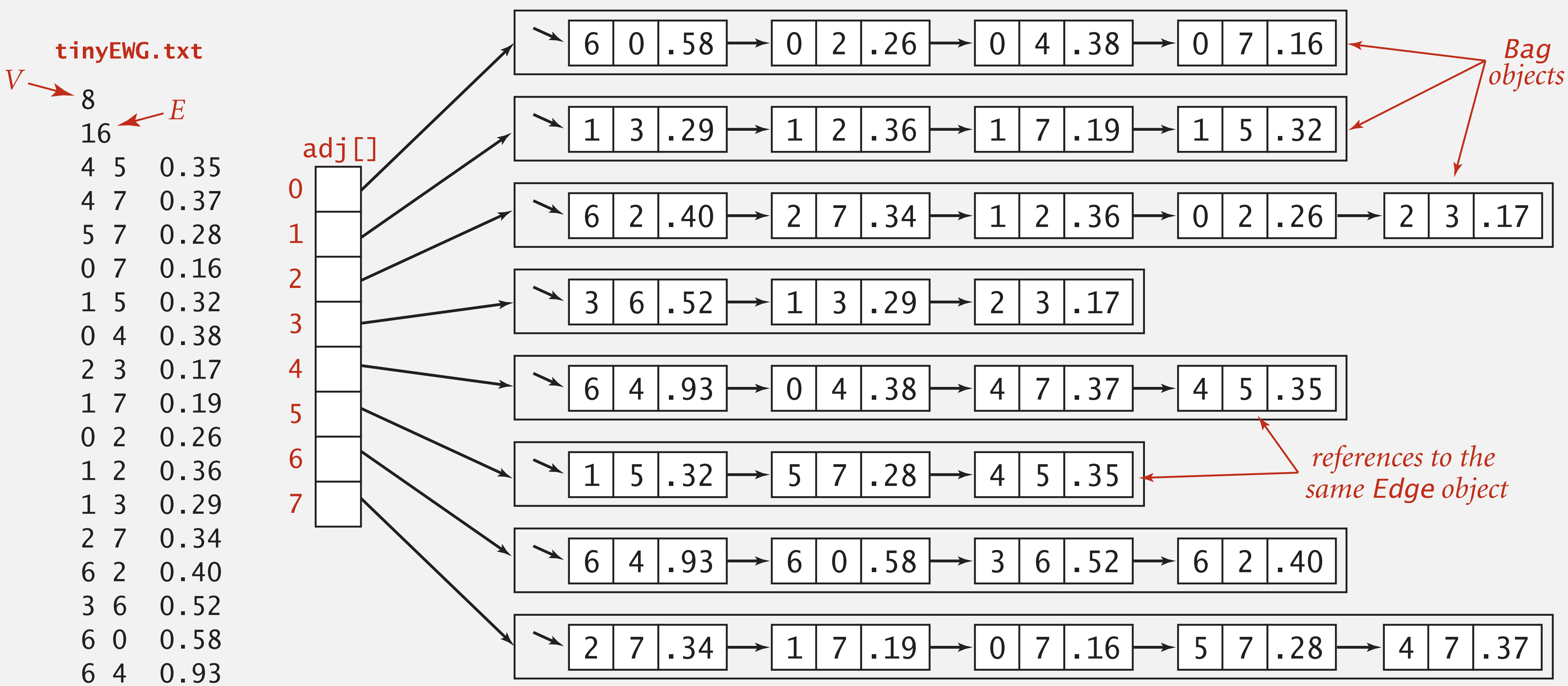
Edge-weighted graph API

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*
Iterable<Edge>	edges()	*all edges in this graph*
int	V()	*number of vertices*
int	E()	*number of edges*

Conventions. Allow self-loops and parallel edges.

Edge-weighted graph: adjacency-lists 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<Edge>();
   }

   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 Edges instead of integers

constructor

add edge to both adjacency lists

https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

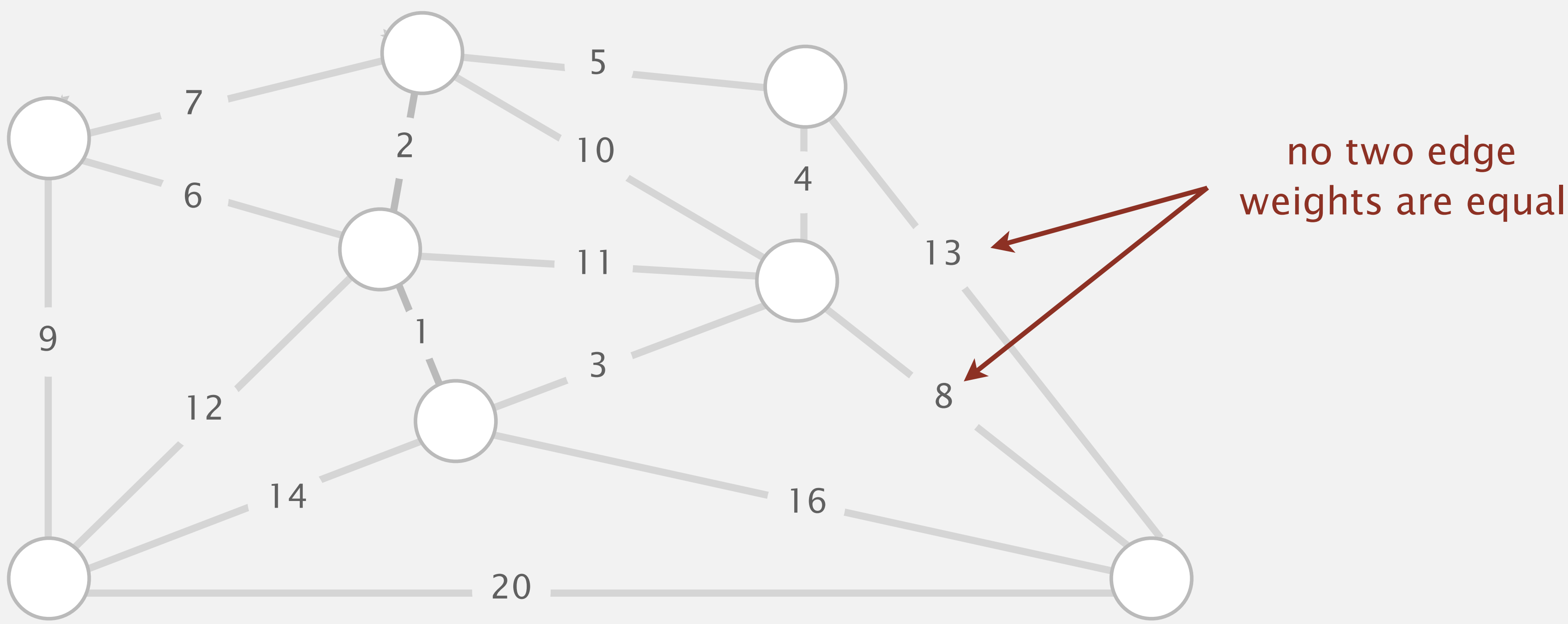
- *introduction*
- *edge-weighted graph API*
- *cut property*
- *Kruskal's algorithm*
- *Prim's algorithm*

Simplifying assumptions

For simplicity, we assume:

- No parallel edges.
- The graph is connected. $\Rightarrow$ MST exists.
- The edge weights are distinct. $\Rightarrow$ MST is unique. ← see Exercise 4.3.3

Note. Algorithms still work even if parallel edges or duplicate edge weights.

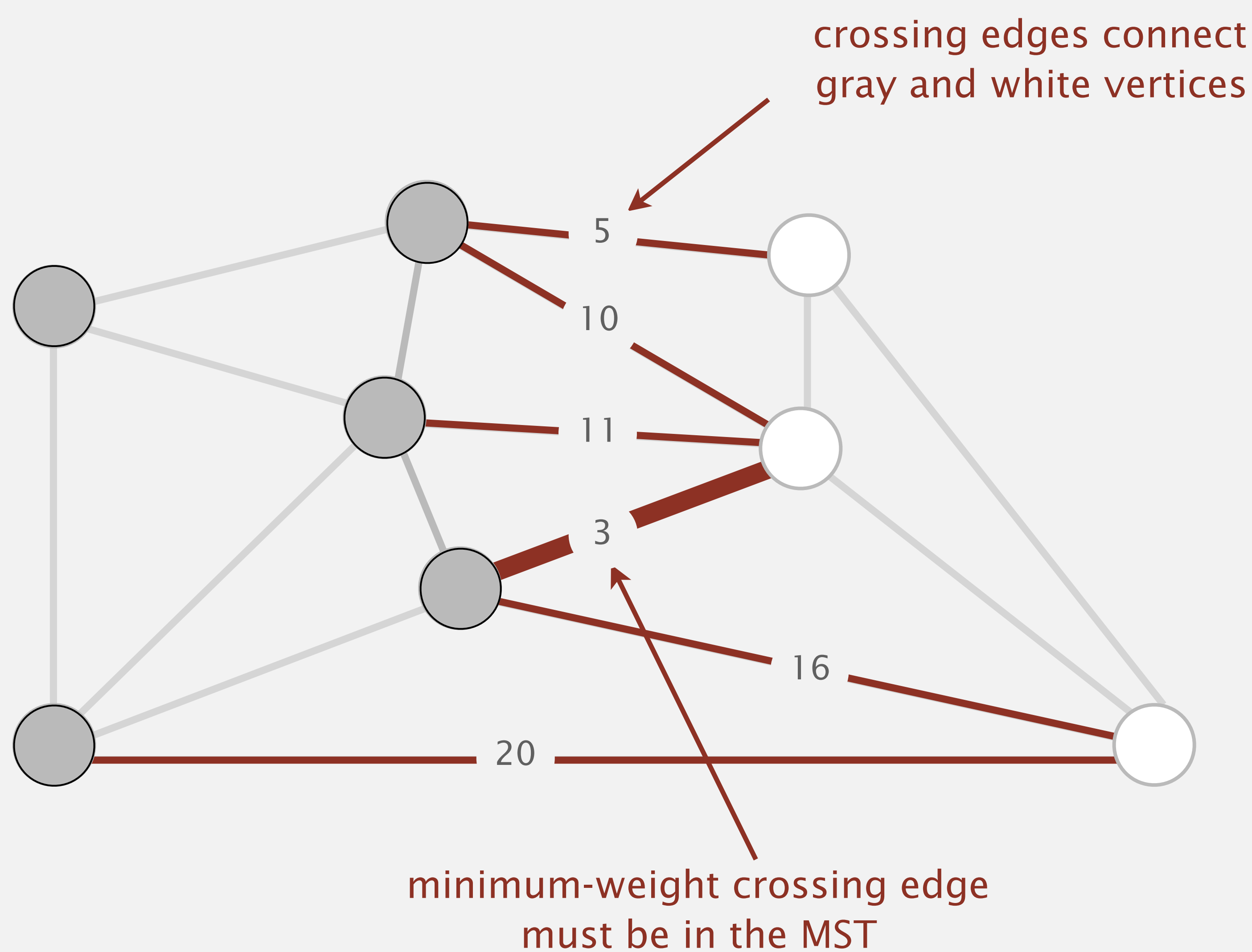


Cut property

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

Def. A crossing edge connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.



Minimum spanning trees: quiz 2

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

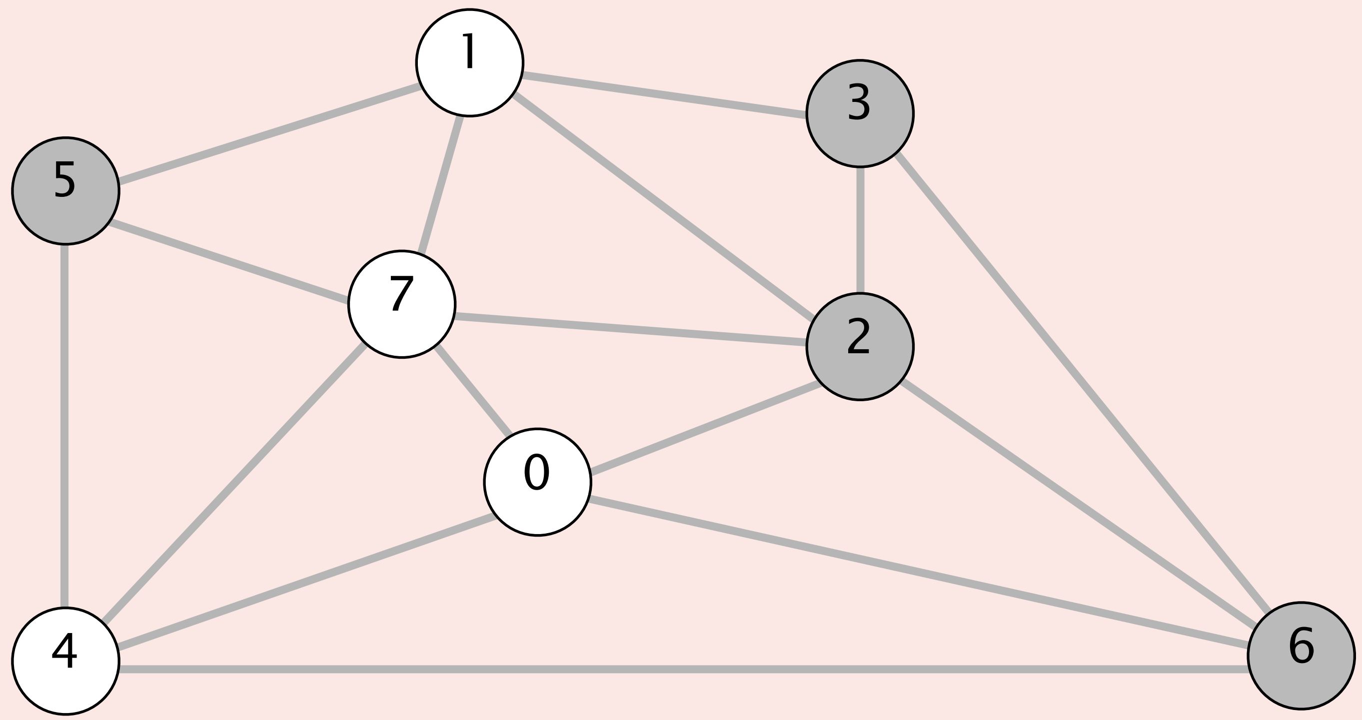
A. 0–7 (0.16)

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

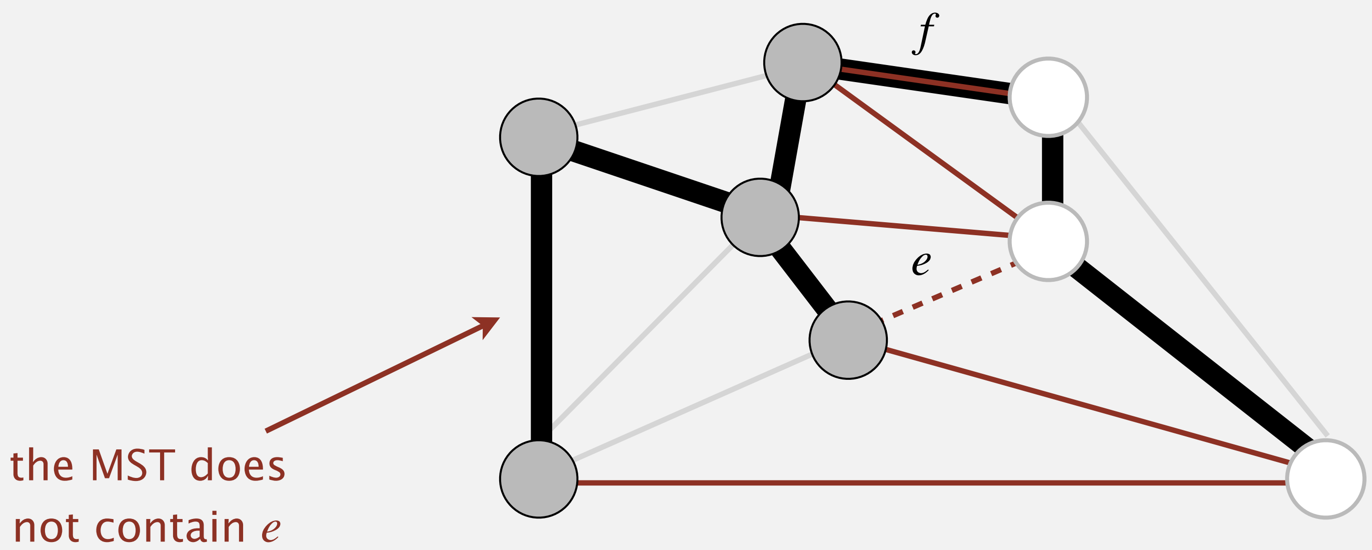
Def. A cut is a partition of a graph's vertices into two (nonempty) sets.

Def. A crossing edge connects two vertices in different sets.

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

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

- Some other edge f in the MST must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since weight of e is less than the weight of f, that spanning tree has lower weight.
- Contradiction. ▪



Application of cut property [warmup for Kruskal's algorithm]

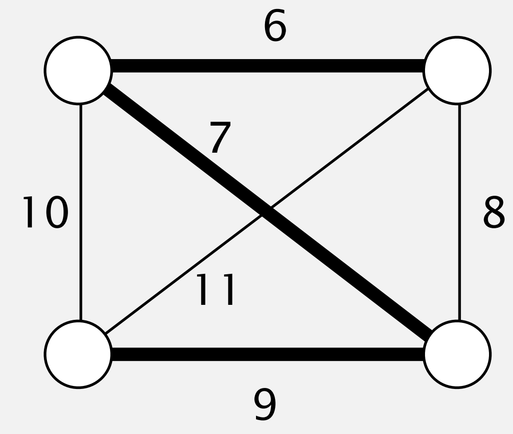
Def. A cut is a partition of a graph's vertices into two (nonempty) sets.

Def. A crossing edge connects two vertices in different sets.

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

Exercise. In any connected graph of ≥ 3 vertices (distinct edge weights; no parallel edges):

- Show that the edge with lowest weight is in the MST.
- Show that the edge with second lowest weight is in the MST.
- Note that the edge with third lowest weight may not be in the MST.



https://algs4.cs.princeton.edu

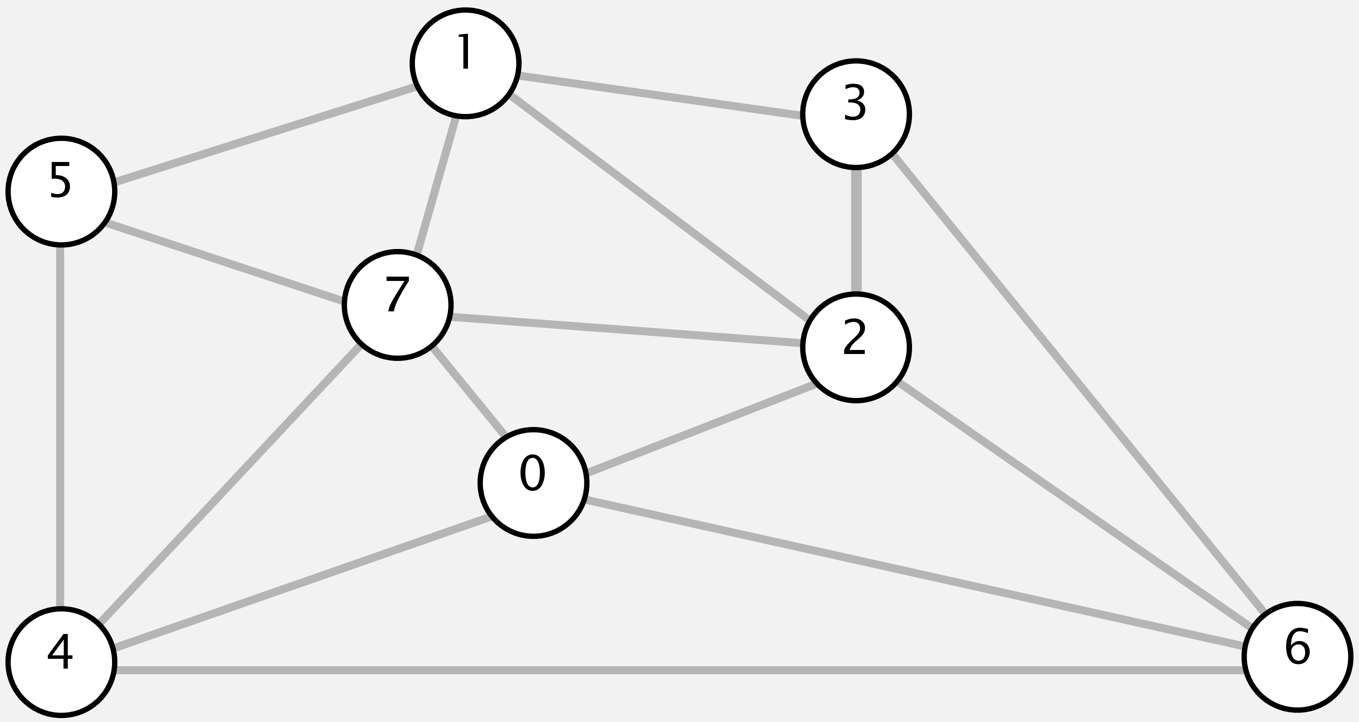
4.3 Minimum Spanning Trees

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

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree 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

Kruskal's algorithm: visualization

Kruskal's algorithm: correctness proof

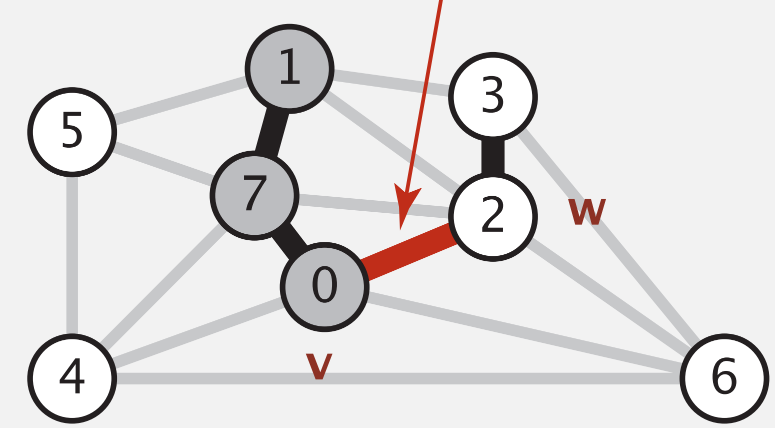
Proposition. Kruskal's algorithm computes the MST.

Recall: increasing order of edge weights

Pf. Let T be the "tree" at some point during execution, and e the next edge considered.

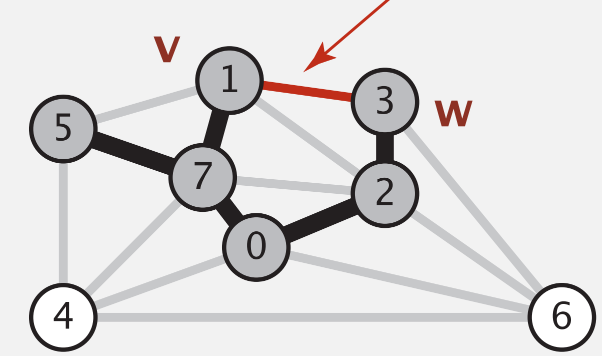
[Case 1] Kruskal's algorithm adds edge $e = v–w$ to T.

add edge to tree



- 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 edge crossing cut is in T.
- No edge crossing cut has lower weight. Why?
- Cut property $\Rightarrow$ edge e is in the MST.
- $\Rightarrow$ Kruskal's algorithm correctly adds e to T.

adding edge to tree would create a cycle



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

- From Case 1, all edges in T are in the MST.
- The MST can't contain a cycle.
- $\Rightarrow$ Kruskal's algorithm correctly discards e.

Minimum spanning trees: quiz 3

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

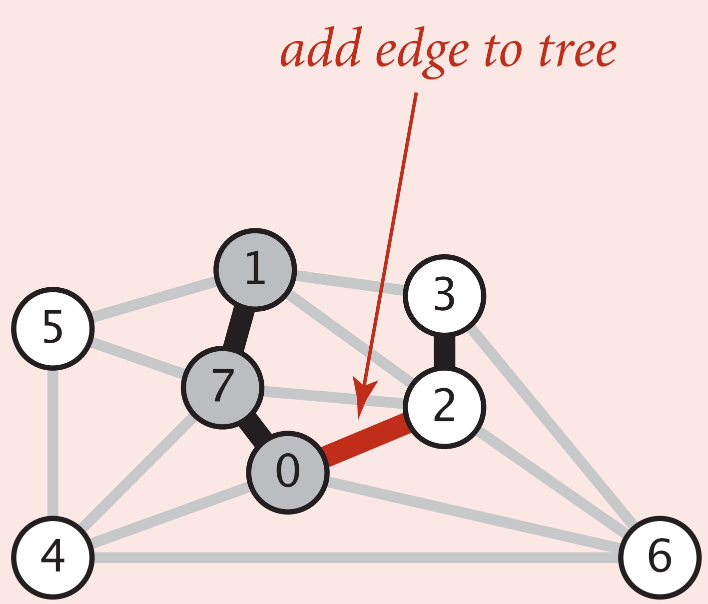
How difficult to implement? (Worst case order of growth of best impl.)

A. 1

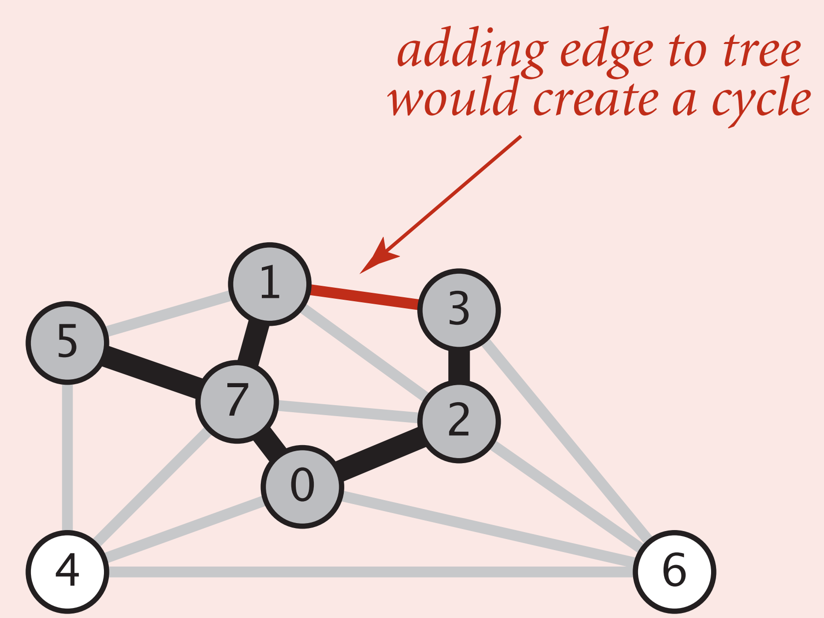
B. $\log V$

C. V

D. $E + V$



Case 1: v and w in same component



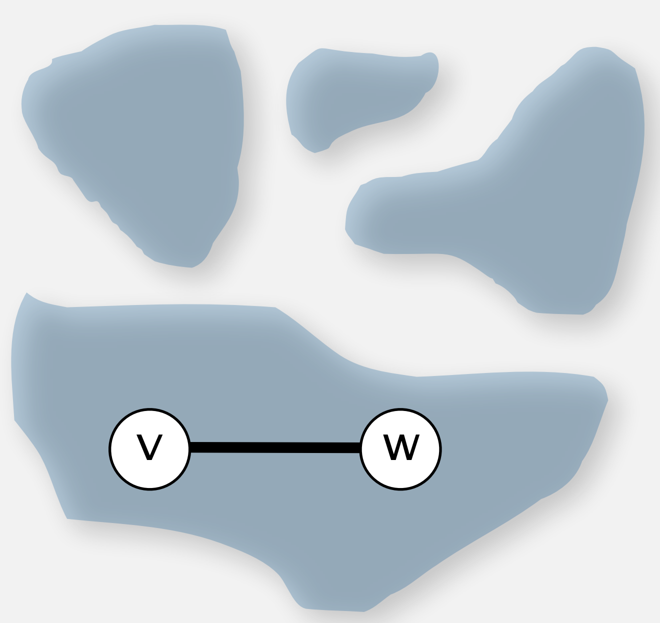
Case 2: v and w in different components

Kruskal's algorithm: implementation challenge

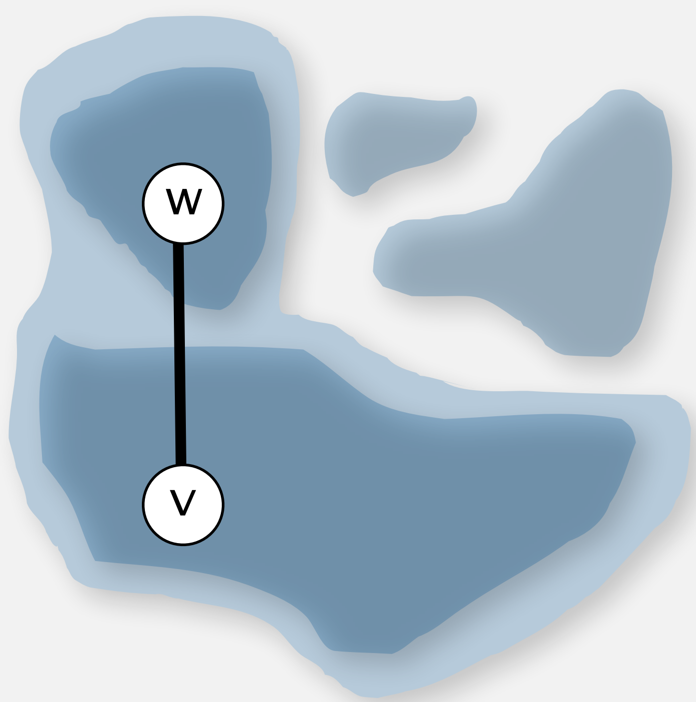
Challenge. Would adding edge *v*–*w* to tree *T* create a cycle? If not, add it.

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* would create a cycle.
- To add *v*–*w* to *T*, merge sets containing *v* and *w*.



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<Edge>();

   public KruskalMST(EdgeWeightedGraph G)
   {
      DirectedEdge[] 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))
         {
            uf.union(v, w);
            mst.enqueue(e);
         }
      }
   }

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

edges in the MST

sort edges by weight

maintain connected components

greedily add edges to MST

edge v–w does not create cycle

merge connected components

add edge e to MST

Kruskal's algorithm: running time

Proposition. Kruskal's algorithm computes MST in time proportional to $E \log V$ (in the worst case).

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

same as $E \log V$
if no parallel edges

† using weighted quick union

See Piazza post @519 for a detailed explanation
https://piazza.com/class/jrp35q44vo35p2?cid=519

https://algs4.cs.princeton.edu

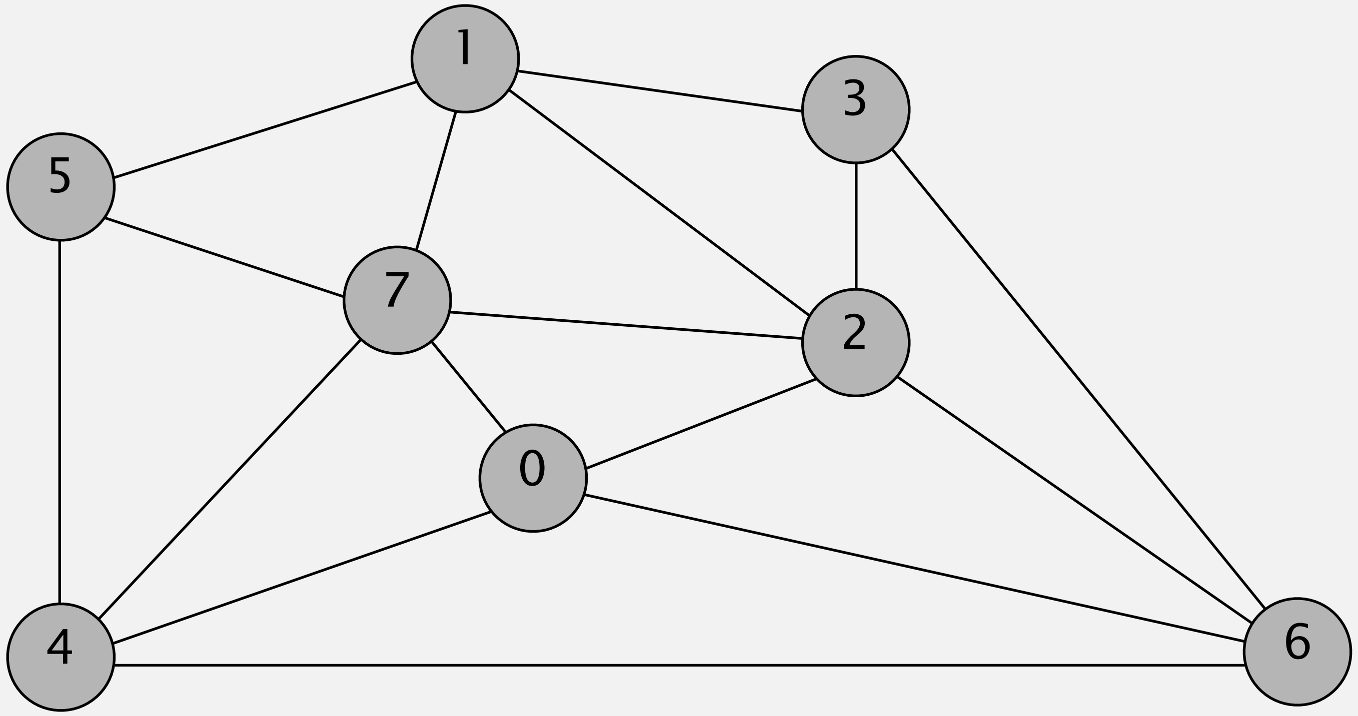
4.3 Minimum Spanning Trees

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

Only lazy implementation covered;
see textbook / videos for eager implementation

Prim's algorithm demo

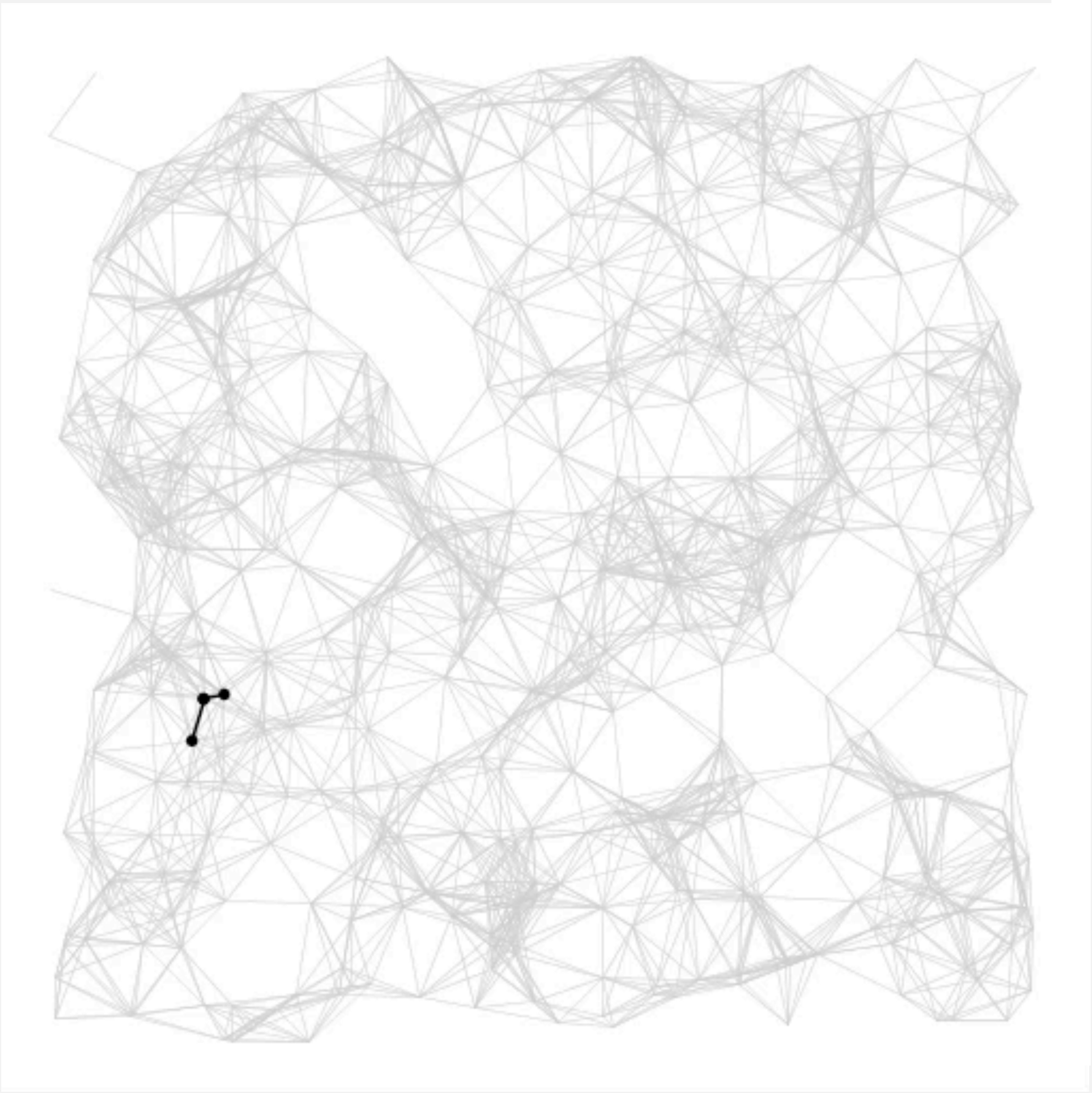
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V-1$ edges.



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: visualization



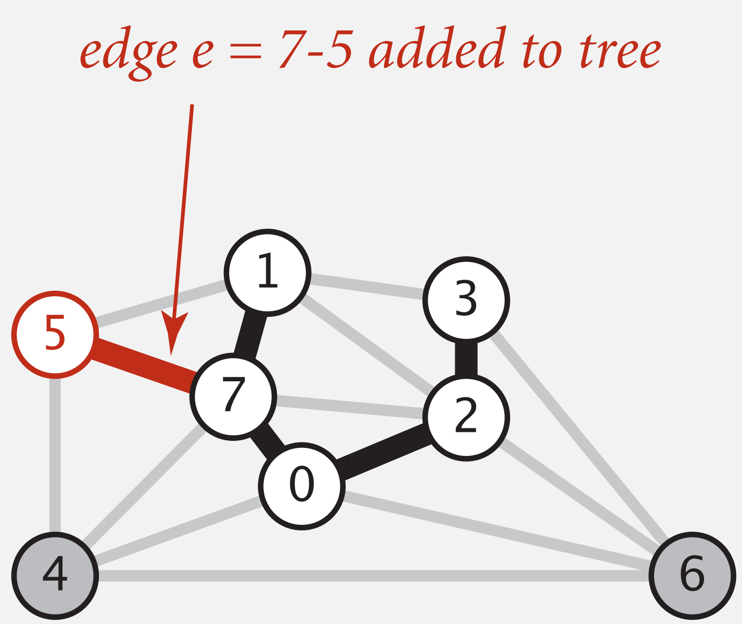
Prim's algorithm: proof of correctness

Proposition.

Prim's algorithm computes the MST.

Pf.

- Cut = set of vertices in T.
- The edges crossing this cut are precisely those considered by Prim's algorithm (edges with exactly one endpoint in T).
- Cut property $\Rightarrow$ edge added by Prim's algorithm must be in the MST.



Minimum spanning trees: quiz 4

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

How difficult to implement?

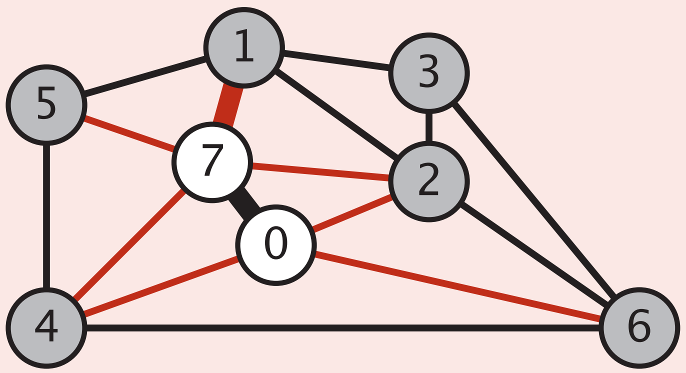
A. 1

B. $\log E$

C. V

D. E

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




```
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
```

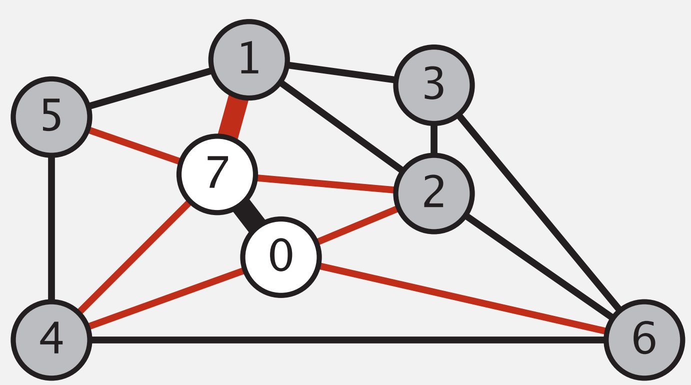
Prim's algorithm: lazy implementation

Challenge. Find the 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 (assuming other endpoint not in T)

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

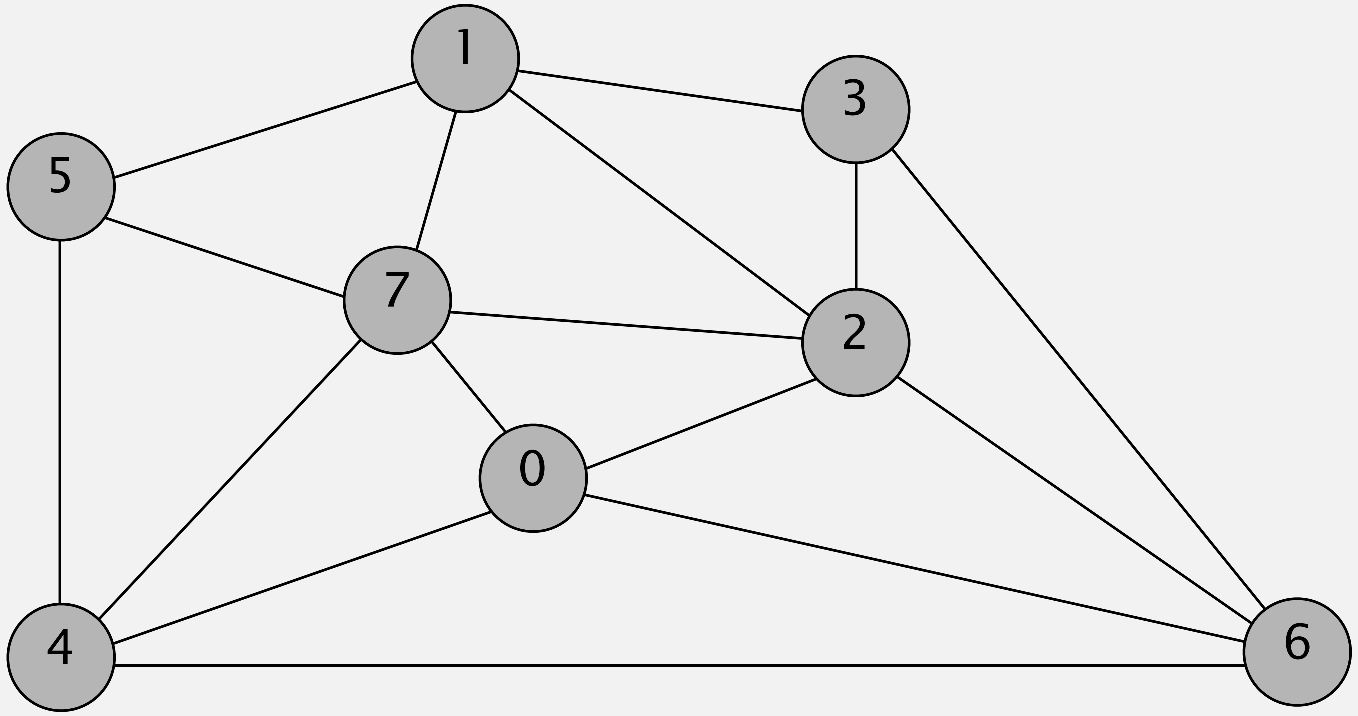


priority queue of crossing edges

```
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 demo

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



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

```
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<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty() && 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);
        }
   }
}
```

assume G is connected

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

ignore if both endpoints in T

add edge e to tree

add either v or w to tree

Prim's algorithm: lazy implementation

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

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

add v to T

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

Lazy Prim's algorithm: running time

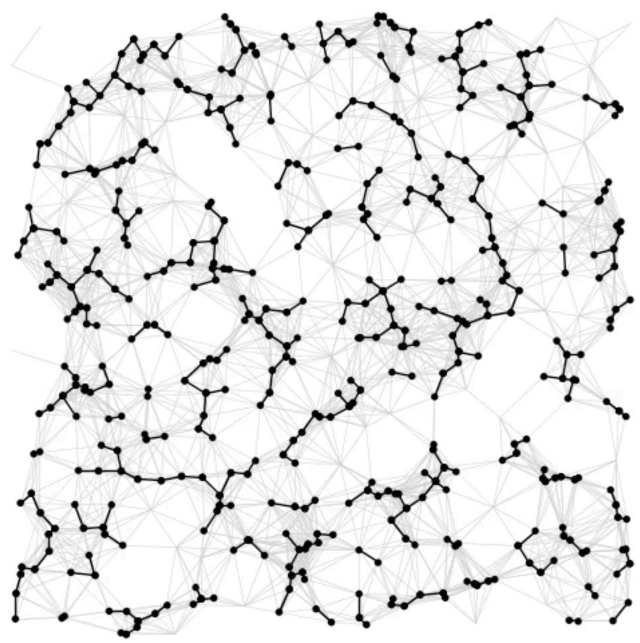
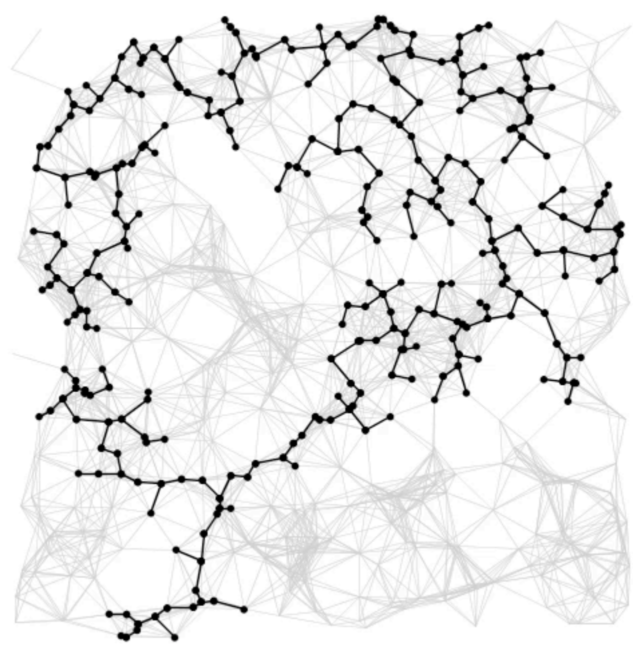
Proposition. Lazy Prim's algorithm computes the MST in time proportional to $E \log E$ and extra space proportional to E (in the worst case).

minor defect

Pf.

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

MST: algorithms of the day

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