CS2010: Data Structures and Algorithms II

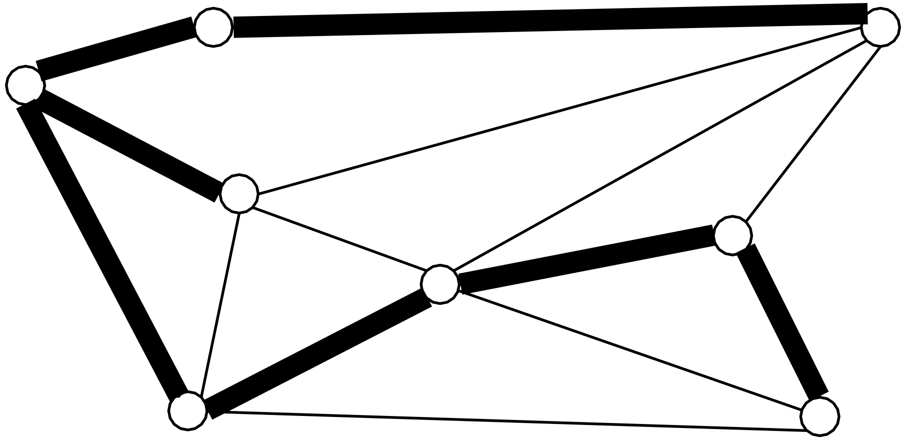
Minimum Spanning Trees

Ivana.Dusparic@scss.tcd.ie

Spanning tree

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

- Connected.
- Acyclic.
- Includes all of the vertices.

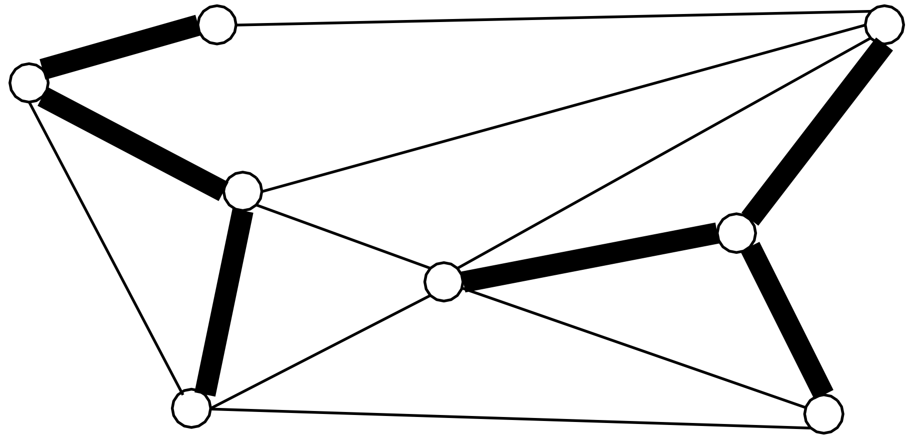


graph G

Spanning tree

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

- Connected.
- Acyclic.
- Includes all of the vertices.

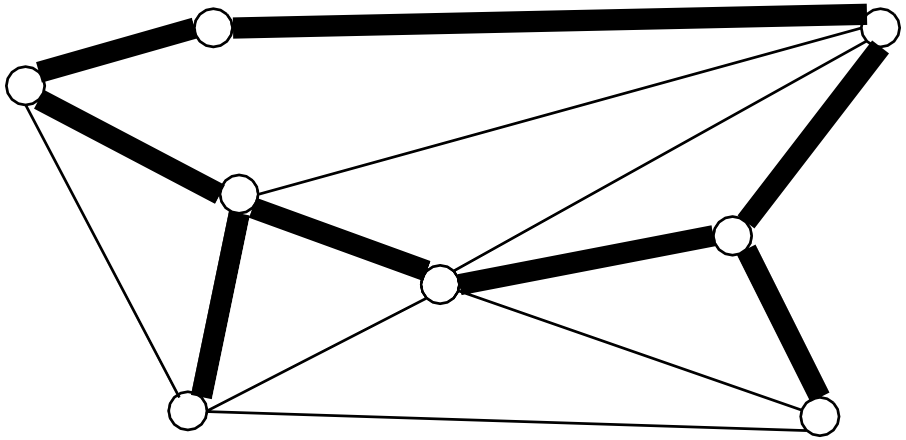


not connected

Spanning tree

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

- Connected.
- Acyclic.
- Includes all of the vertices.

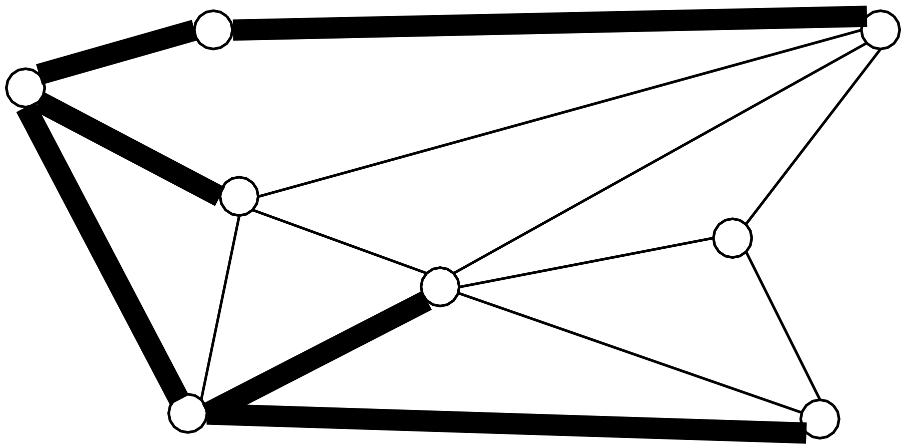


not acyclic

Spanning tree

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

- Connected.
- Acyclic.
- Includes all of the vertices.

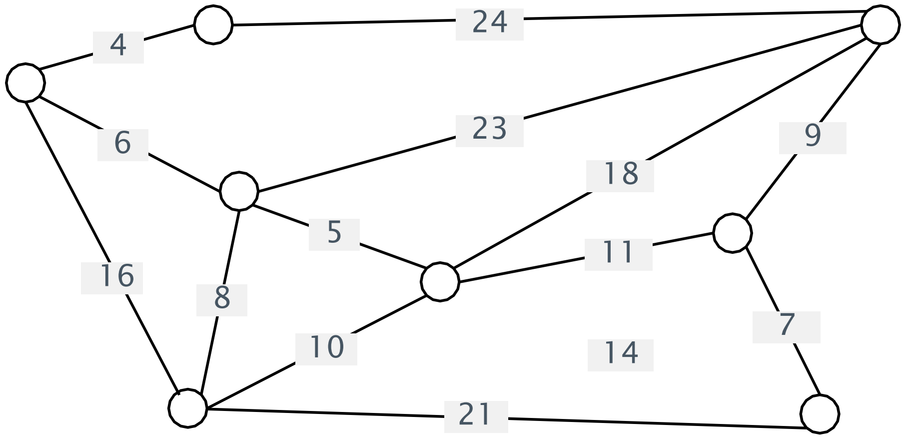


not spanning

Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Goal. Find a min weight spanning tree.

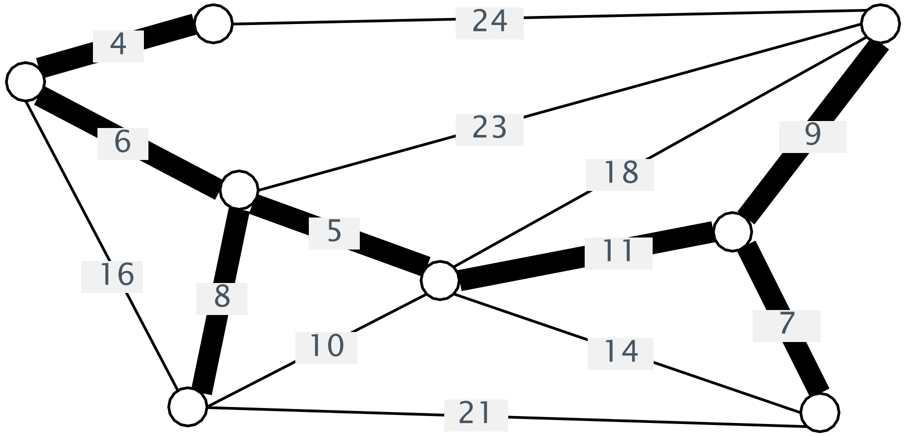


edge-weighted graph G

Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Goal. Find a min weight spanning tree.



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

Brute force. Try all spanning trees?

Applications

- Network design (communication, electrical, hydraulic, computer, road etc)
- Clustering
- Approximation algorithms (eg TSP)
- Many others – classification in biology, sociology, face verification, hand writing detection etc

MST question – Turning Point

- Let G be a connected, edge-weighted graph with V vertices and E edges. How many edges are in a minimum spanning tree of G?
  - V
  - V-1
  - E
  - E-1

Algorithms

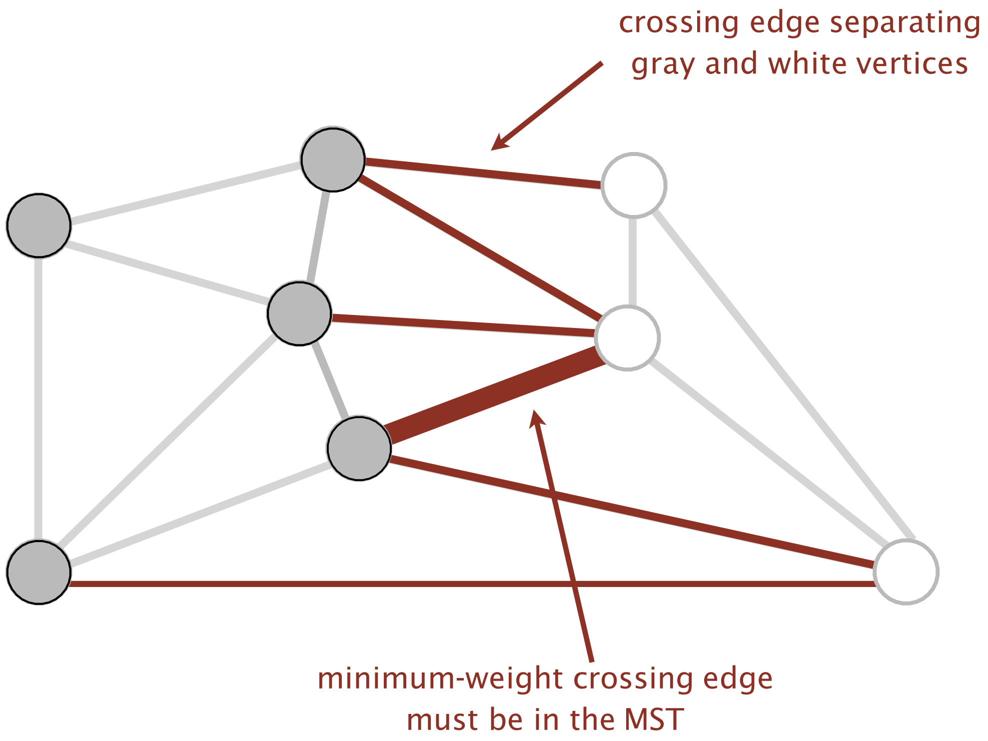
- Greedy algorithms
  - Prim's
  - Kruskal's

Cut property

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

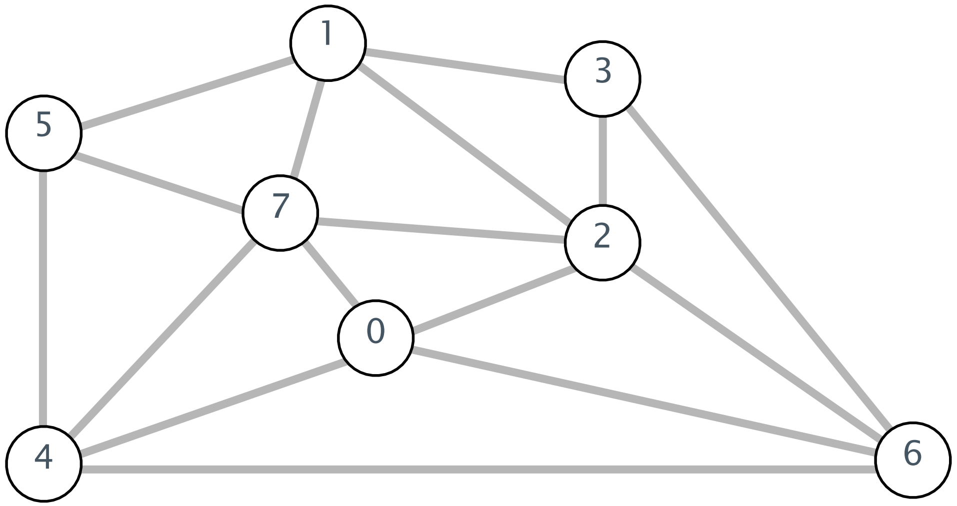
Def. A crossing edge connects a vertex in one set with a vertex in the other.

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



Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

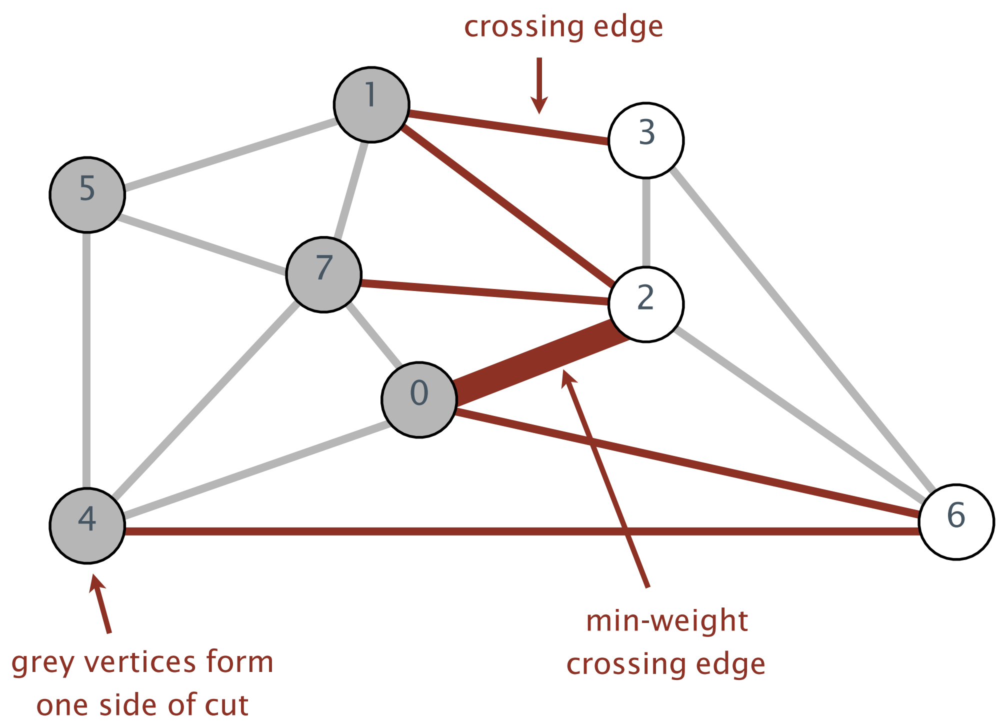


an edge-weighted graph

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

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



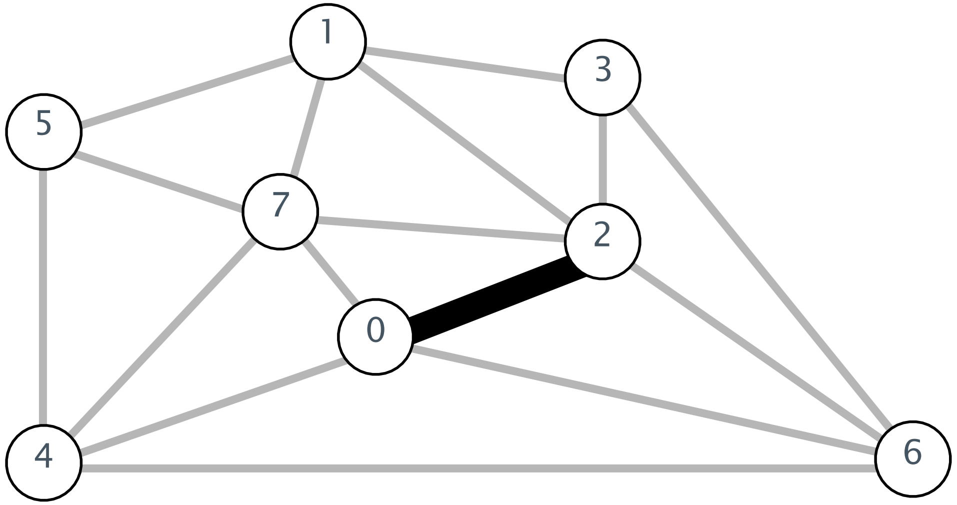
crossing edges
(sorted by weight)

in MST →

0-2	0.26
1-3	0.29
2-7	0.34
1-2	0.36
6-0	0.58
6-4	0.93

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

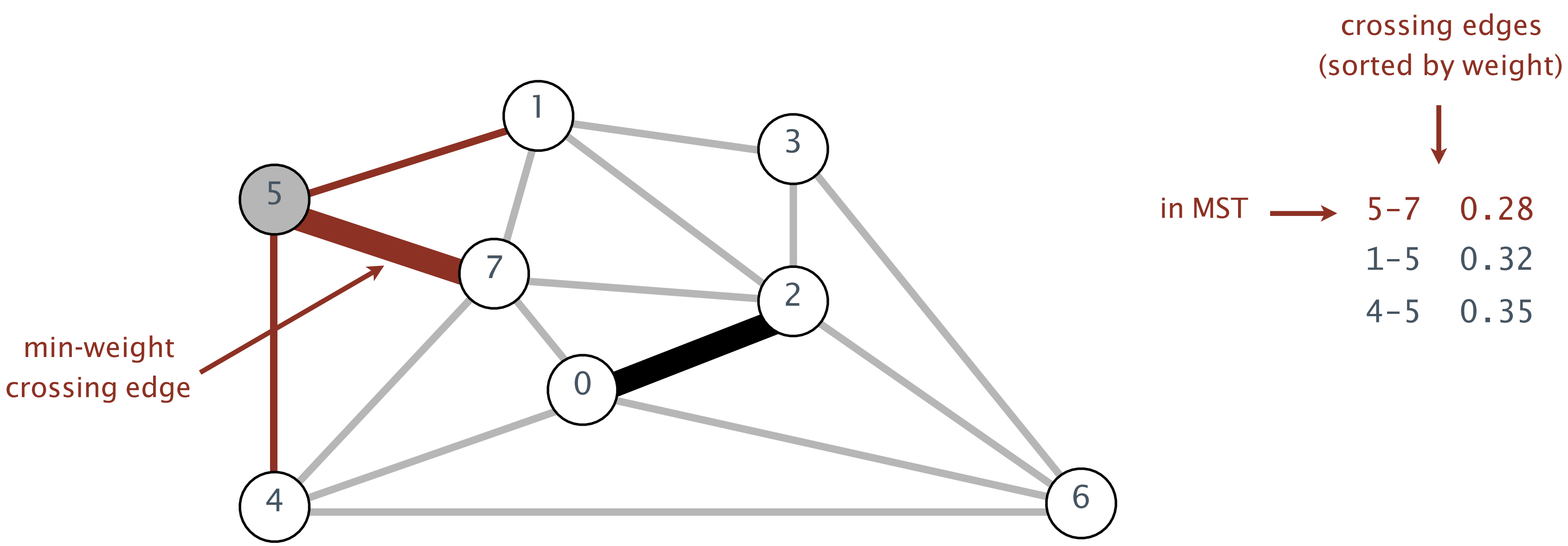


MST edges

0–2

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

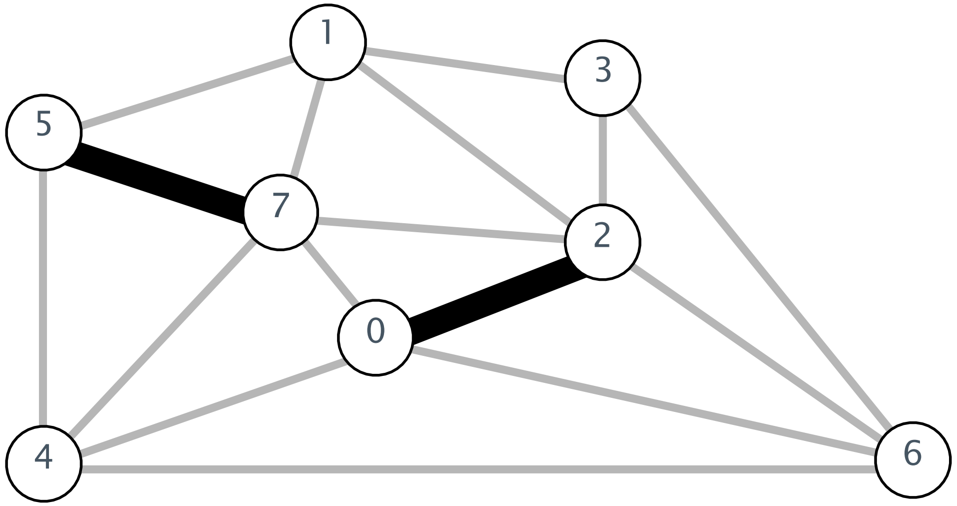


MST edges

0-2

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

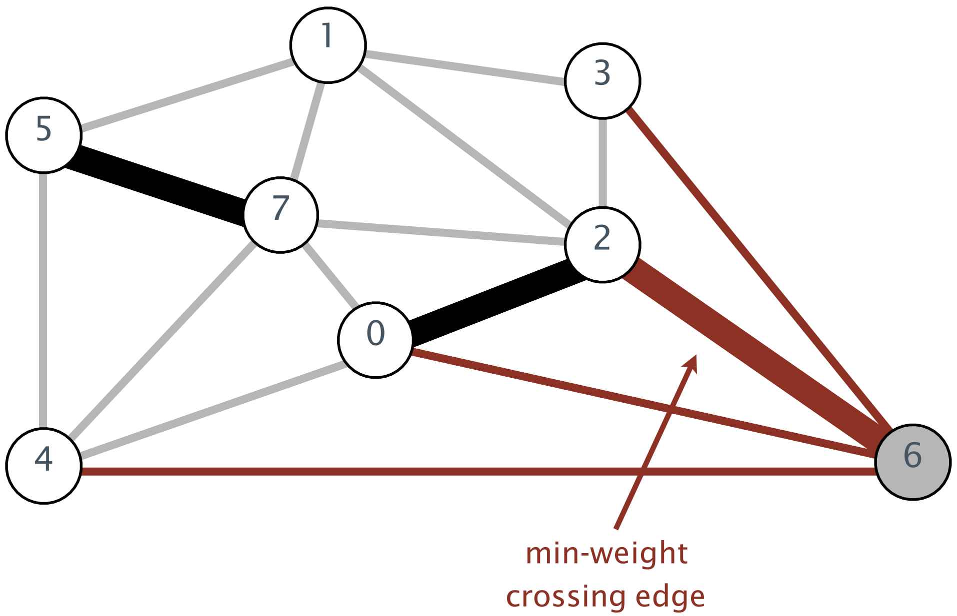


MST edges

0–2 5–7

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



crossing edges
(sorted by weight)

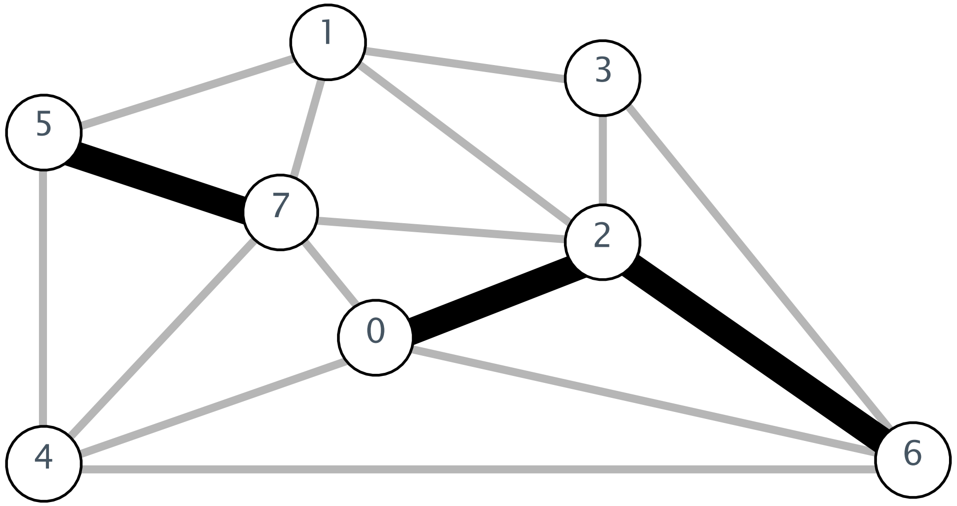
in MST →	6-2	0.40
	3-6	0.52
	6-0	0.58
	6-4	0.93

MST edges

0-2 5-7

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

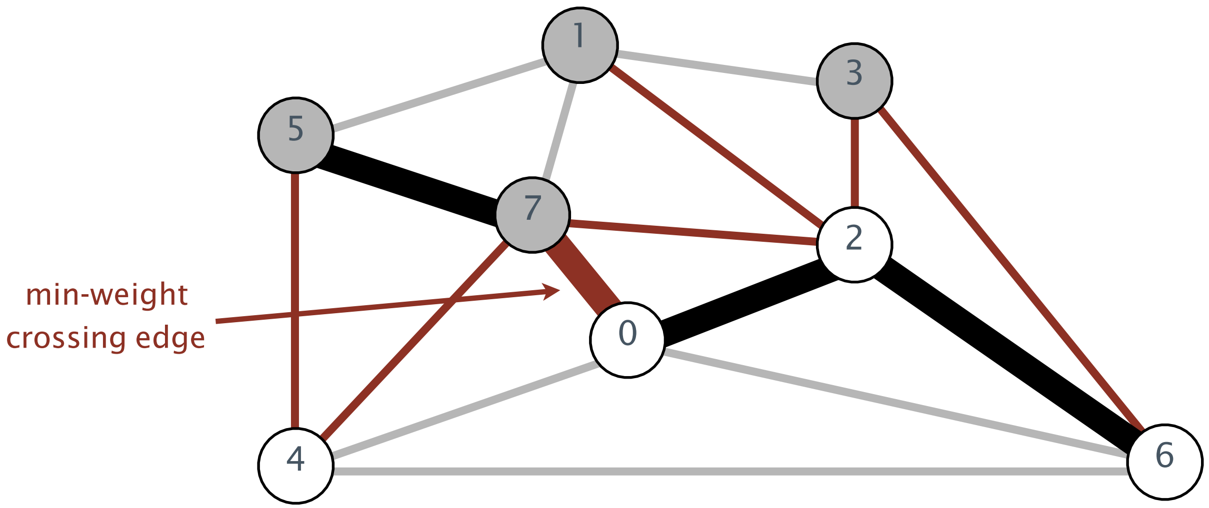


MST edges

0–2 5–7 6–2

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



crossing edges
(sorted by weight)

in MST

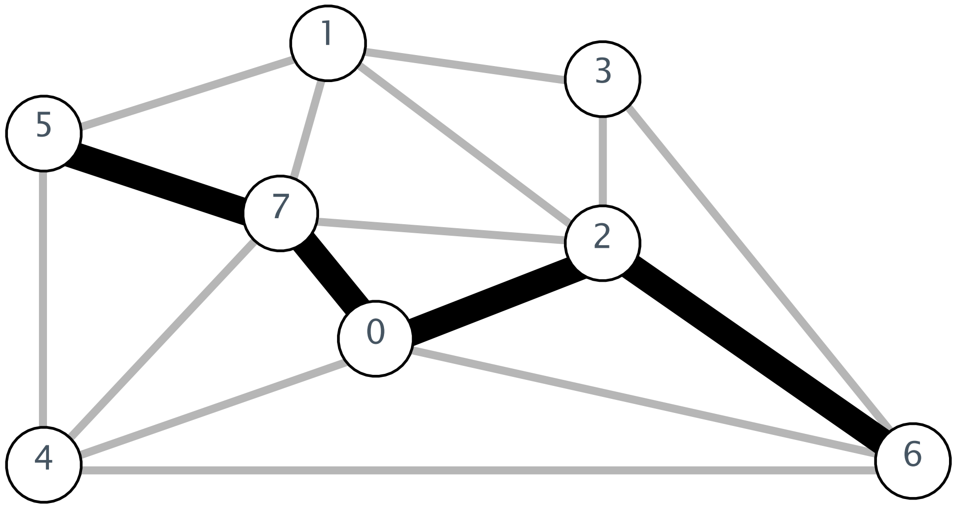
0-7	0.16
2-3	0.17
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
3-6	0.52

MST edges

0-2 5-7 6-2

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

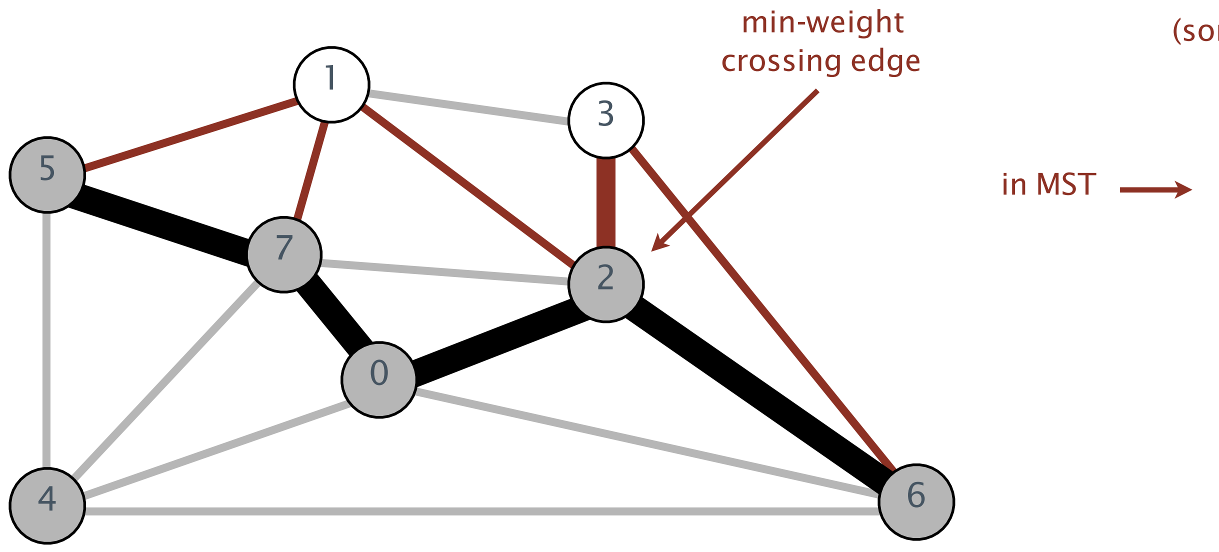


MST edges

0–2 5–7 6–2 0–7

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



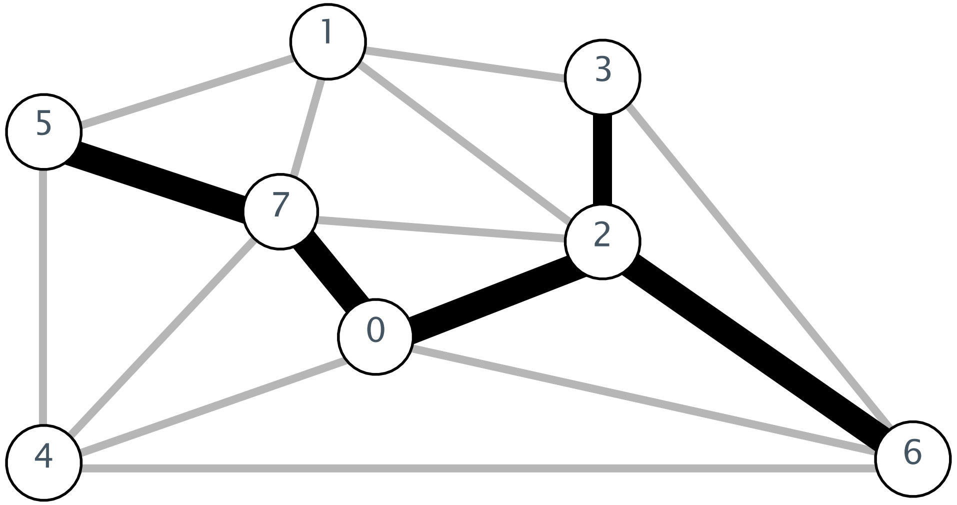
2-3	0.17
1-7	0.19
1-5	0.32
1-2	0.36

MST edges

0-2   5-7   6-2   0-7

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

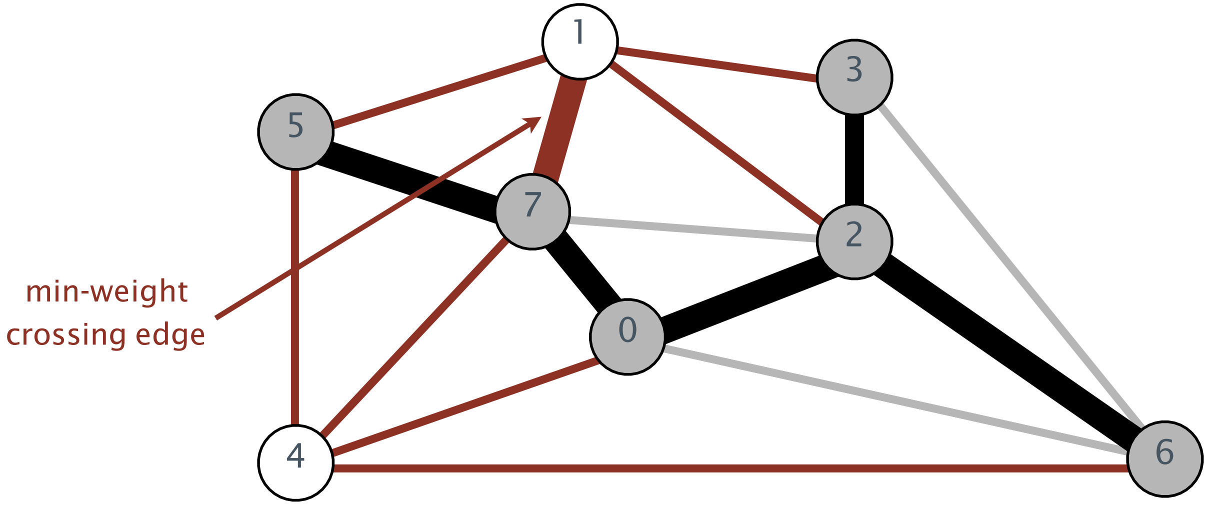


MST edges

0–2 5–7 6–2 0–7 2–3

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



crossing edges
(sorted by weight)

in MST

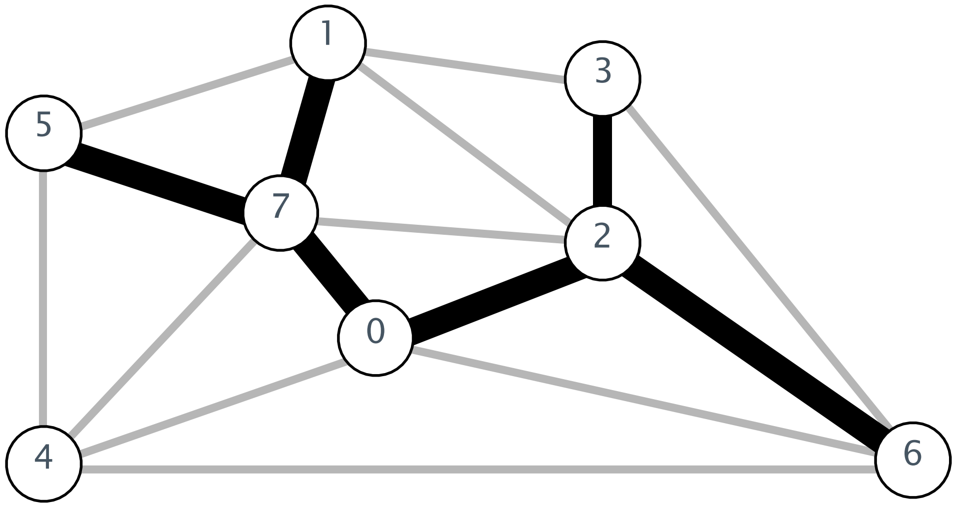
1-7	0.19
1-3	0.29
1-5	0.32
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-4	0.93

MST edges

0-2 5-7 6-2 0-7 2-3

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.

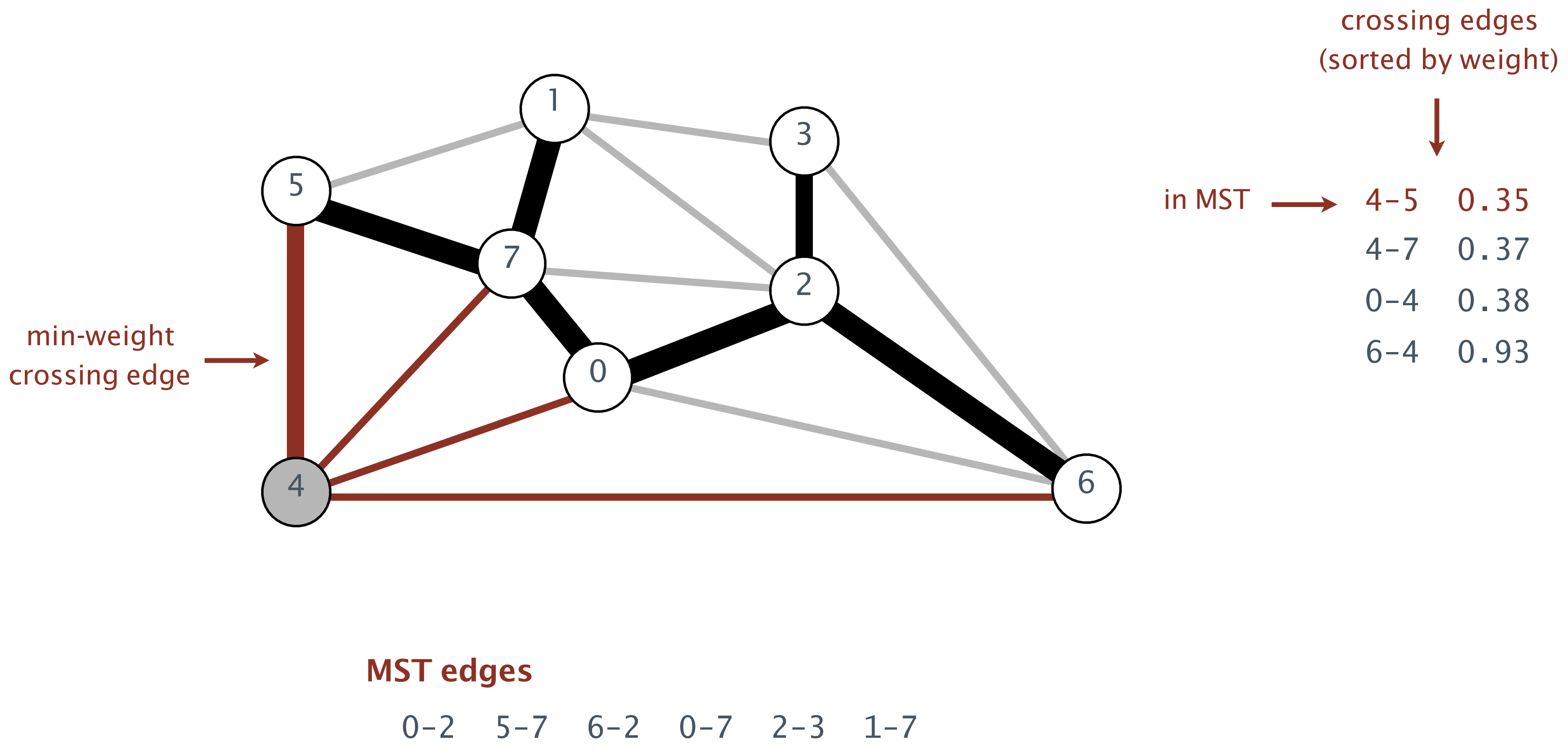


MST edges

0–2 5–7 6–2 0–7 2–3 1–7

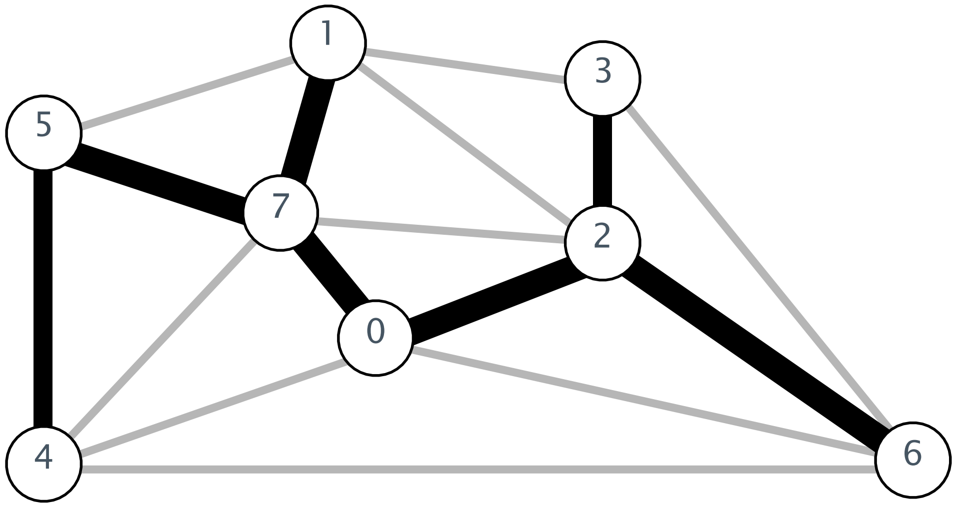
Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



MST edges

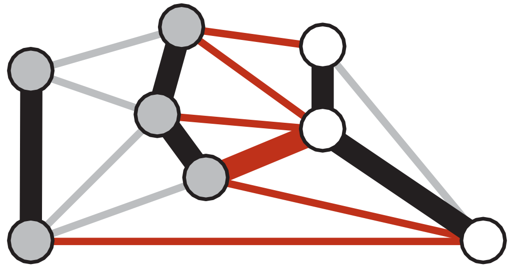
0–2 5–7 6–2 0–7 2–3 1–7 4–5

Greedy MST algorithm: correctness proof

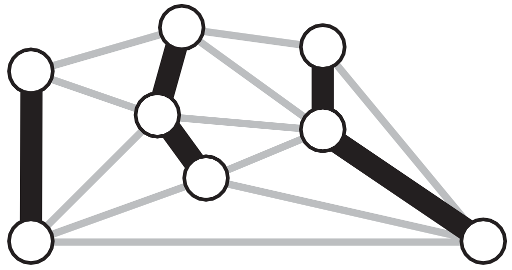
Proposition. The greedy algorithm computes the MST.

Pf.

- Any edge colored black is in the MST (via cut property).
- Fewer than $V-1$ black edges $\Rightarrow$ cut with no black crossing edges. (consider cut whose vertices are any one connected component)



a cut with no black crossing edges

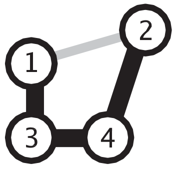


fewer than V-1 edges colored black

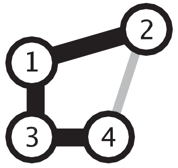
Q. What if edge weights are not all distinct?

A. Greedy MST algorithm still correct if equal weights are present!

(our correctness proof fails, but that can be fixed)



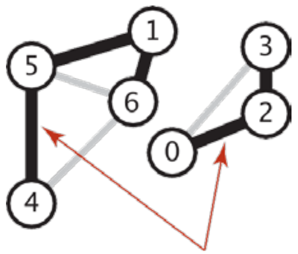
1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50



1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50

Q. What if graph is not connected?

A. Compute minimum spanning forest = MST of each component.



can independently compute MSTs of components

4	5	0.61
4	6	0.62
5	6	0.88
1	5	0.11
2	3	0.35
0	3	0.6
1	6	0.10
0	2	0.22

Greedy MST algorithm: efficient implementations

Efficient implementations.

How to choose cut?

How to find min-weight edge?

Ex 1. Kruskal's algorithm.

Ex 2. Prim's algorithm.

Ex 3. Borüvka's algorithm.

Weighted-edge graph API, Edge API

Adjacency-lists graph representation (review): Java implementation

```
public class Graph
{
   private final int V;
   private final Bag<Integer>[] adj;

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

   public void addEdge(int v, int w)
   {
      adj[v].add(w);
      adj[w].add(v);
   }

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

adjacency lists

create empty graph with V vertices

add edge v–w

iterator for vertices adjacent to v

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];  }
}
```

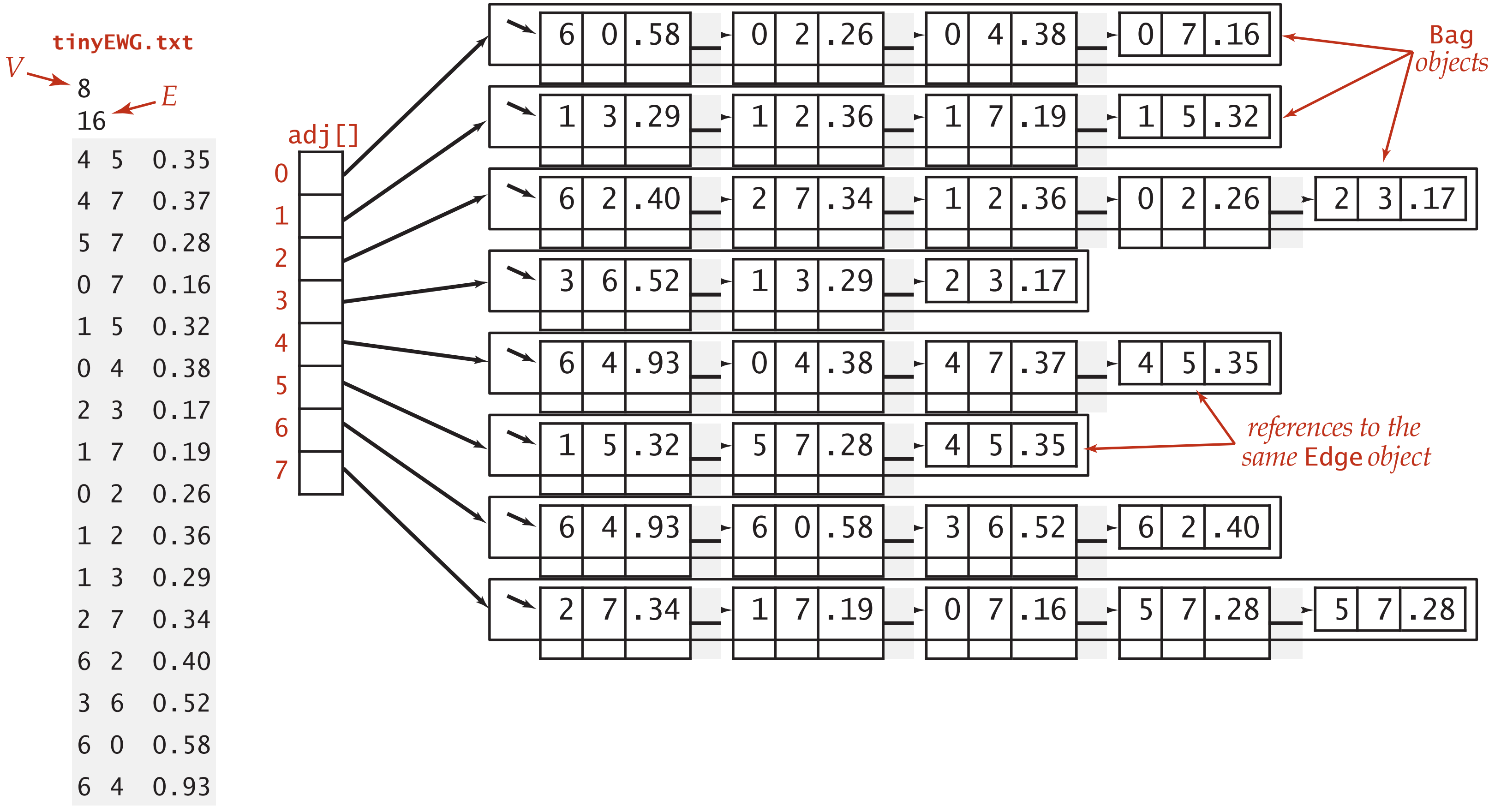
same as Graph, but adjacency lists of Edges instead of integers

constructor

add edge to both adjacency lists

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.



Edge-weighted graph API

public class	EdgeWeightedGraph	
	EdgeWeightedGraph(int V)	*create an empty graph with V vertices*
	EdgeWeightedGraph(In in)	*create a graph from input stream*
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*
String	toString()	*string representation*

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

MST API

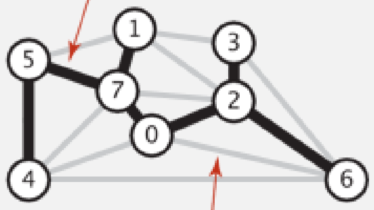
Minimum spanning tree API

Q. How to represent the MST?

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

tinyEWG.txt

```
V -> 8
     16 <- E
     4 5  0.35
     4 7  0.37
     5 7  0.28
     0 7  0.16
     1 5  0.32
     0 4  0.38
     2 3  0.17
     1 7  0.19
     0 2  0.26
     1 2  0.36
     1 3  0.29
     2 7  0.34
     6 2  0.40
     3 6  0.52
     6 0  0.58
     6 4  0.93
```




```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

Minimum spanning tree API

Q. How to represent the MST?

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

```
public static void main(String[] args)
{
   In in = new In(args[0]);
   EdgeWeightedGraph G = new EdgeWeightedGraph(in);
   MST mst = new MST(G);
   for (Edge e : mst.edges())
      StdOut.println(e);
   StdOut.printf("%.2f\n", mst.weight());
}
```

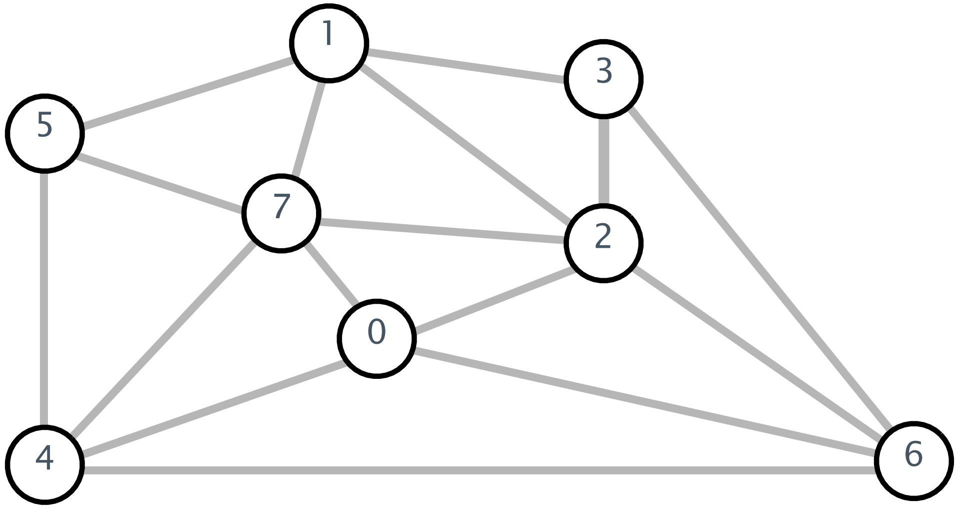
```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

Kruskal’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

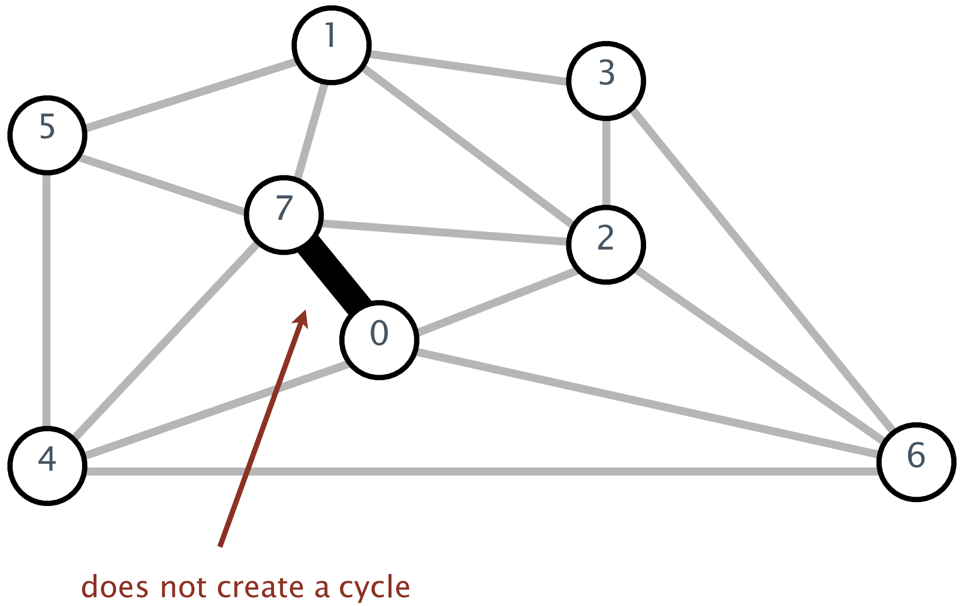
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

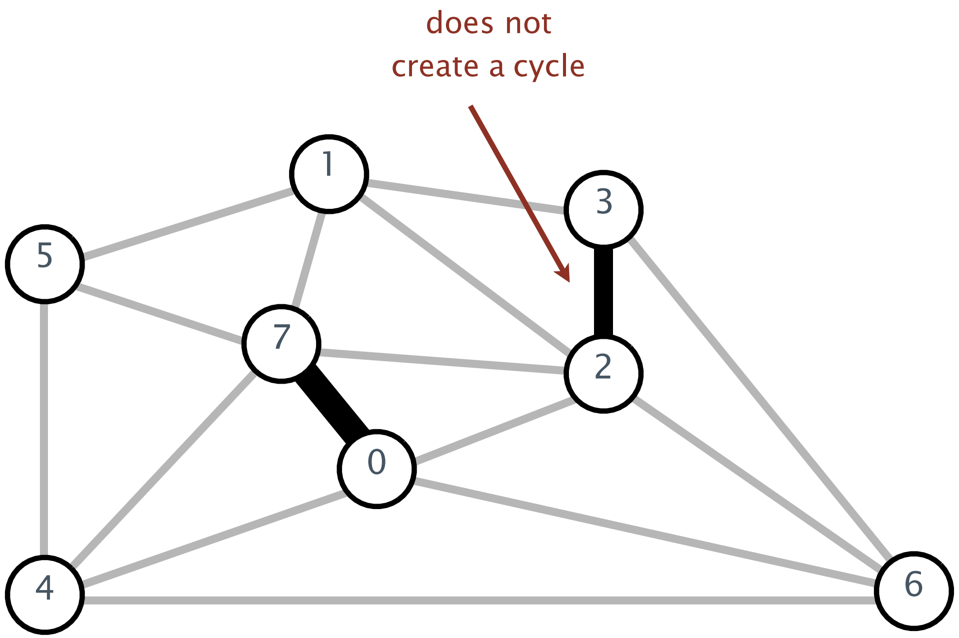
in MST → 0-7 0.16



Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

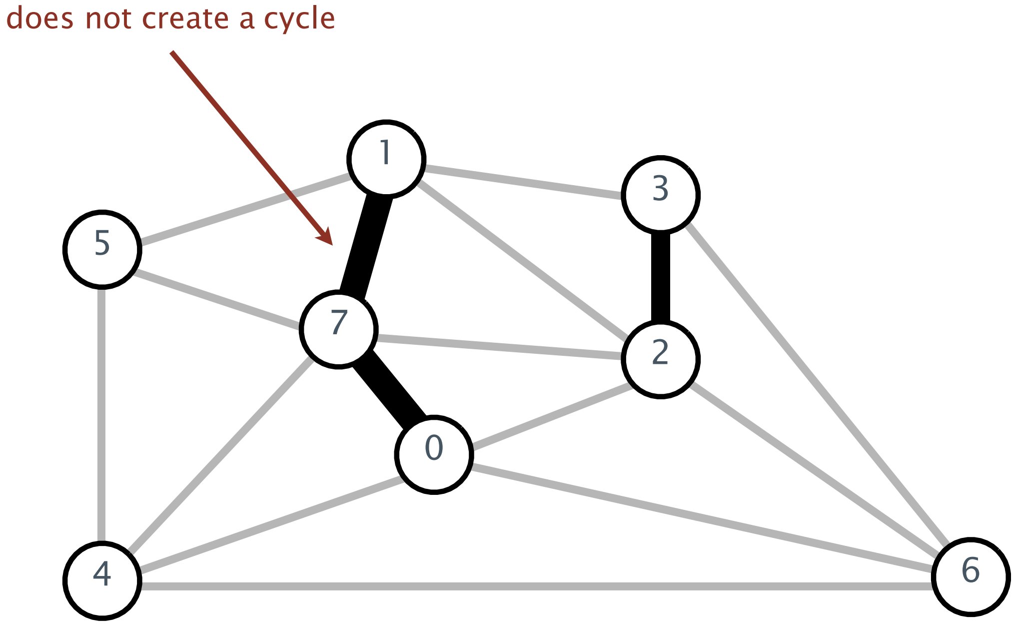


	0-7	0.16
in MST →	2-3	0.17

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

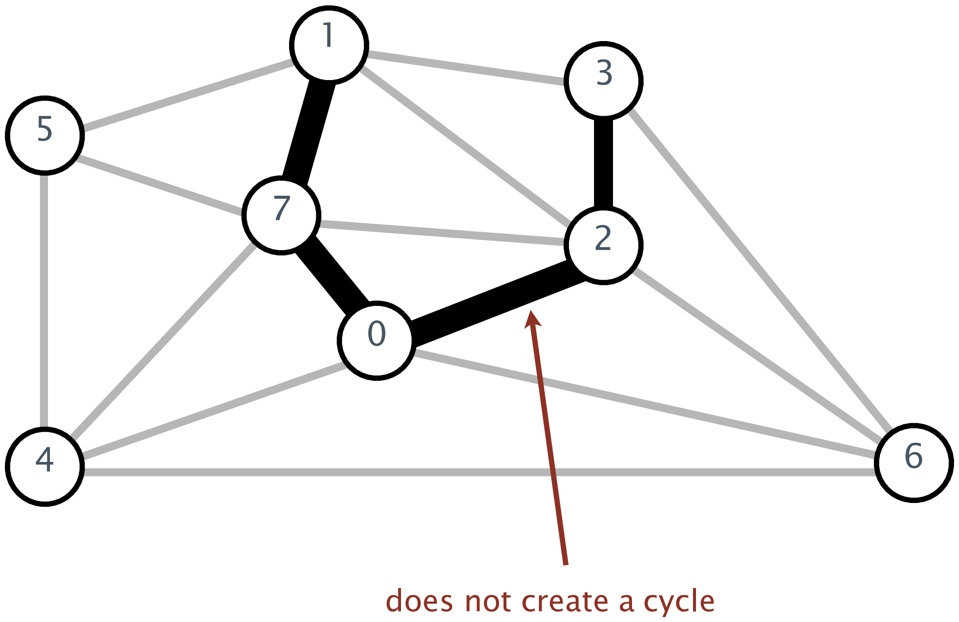


	0-7	0.16
	2-3	0.17
in MST →	1-7	0.19

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



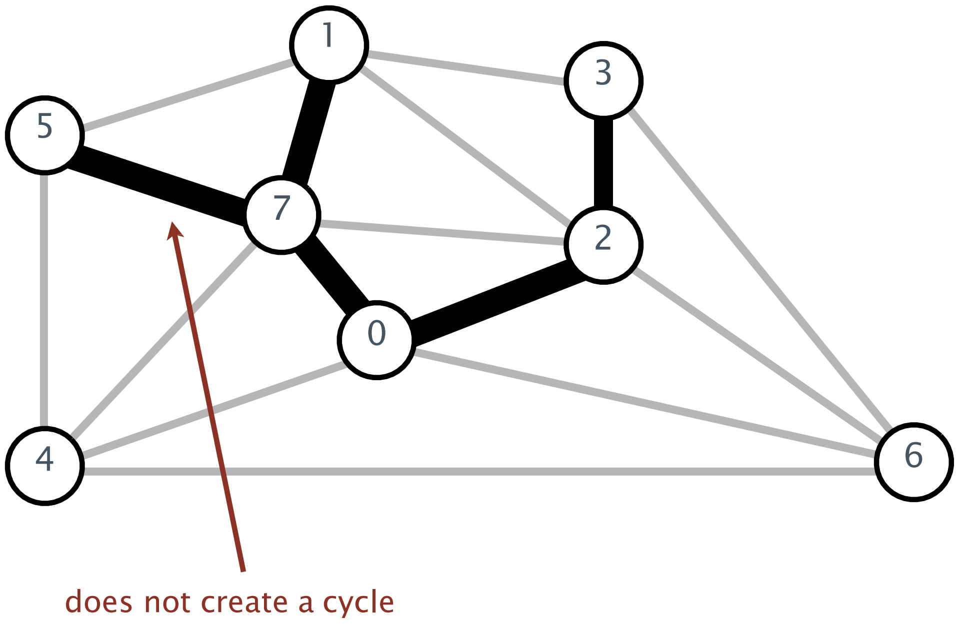
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26

in MST →

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



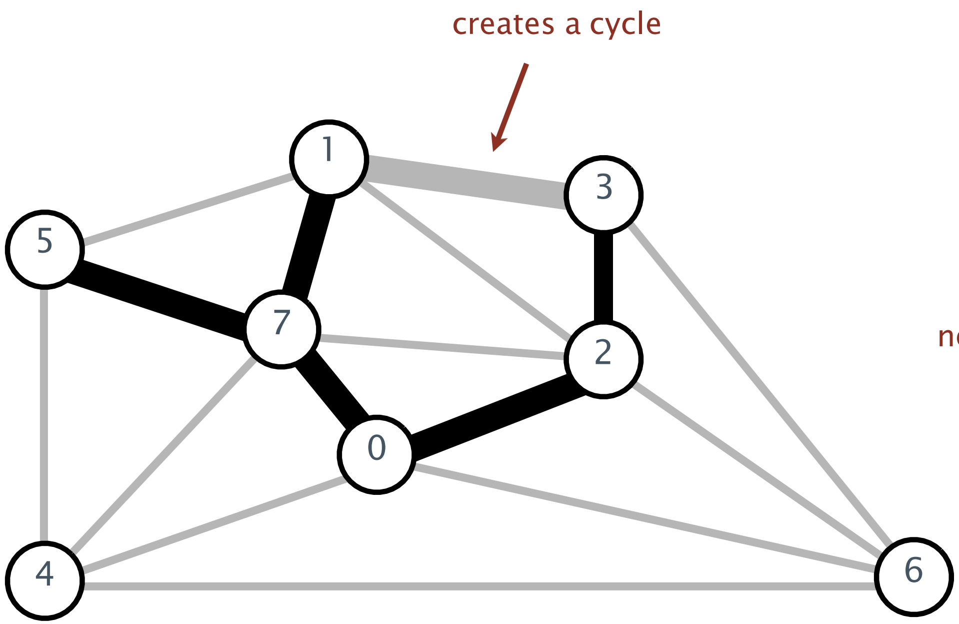
in MST →

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



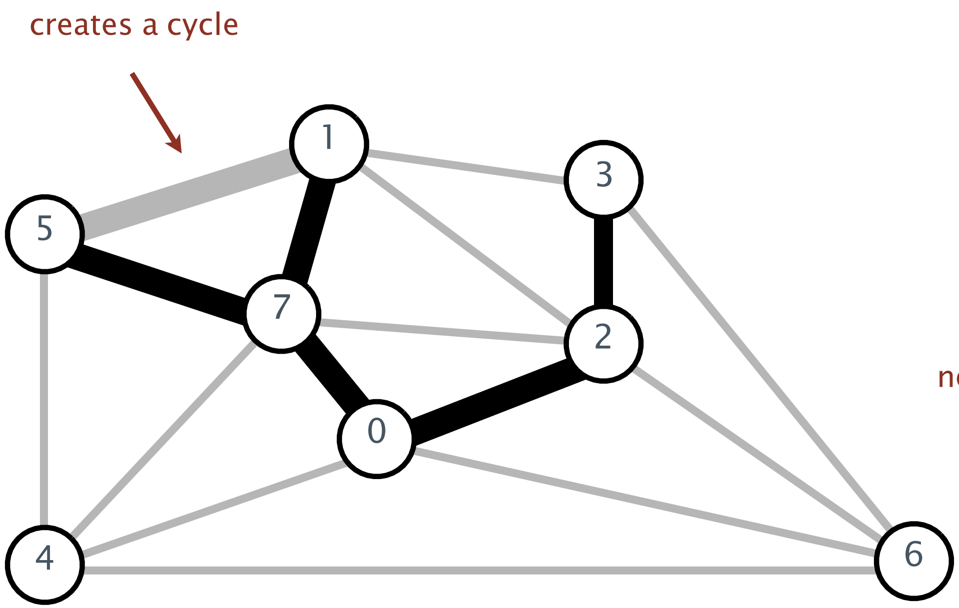
not in MST

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



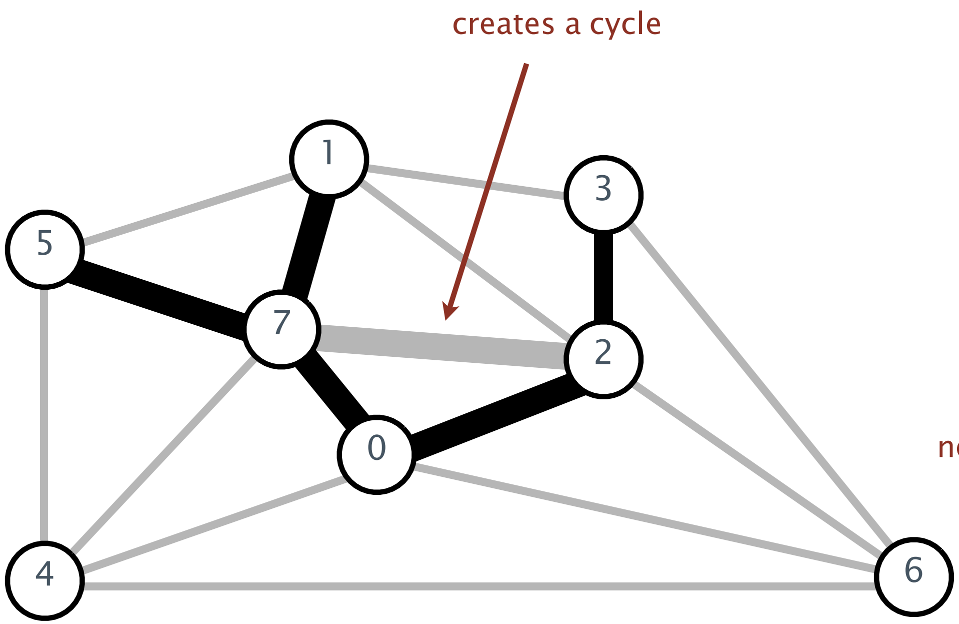
not in MST

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

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



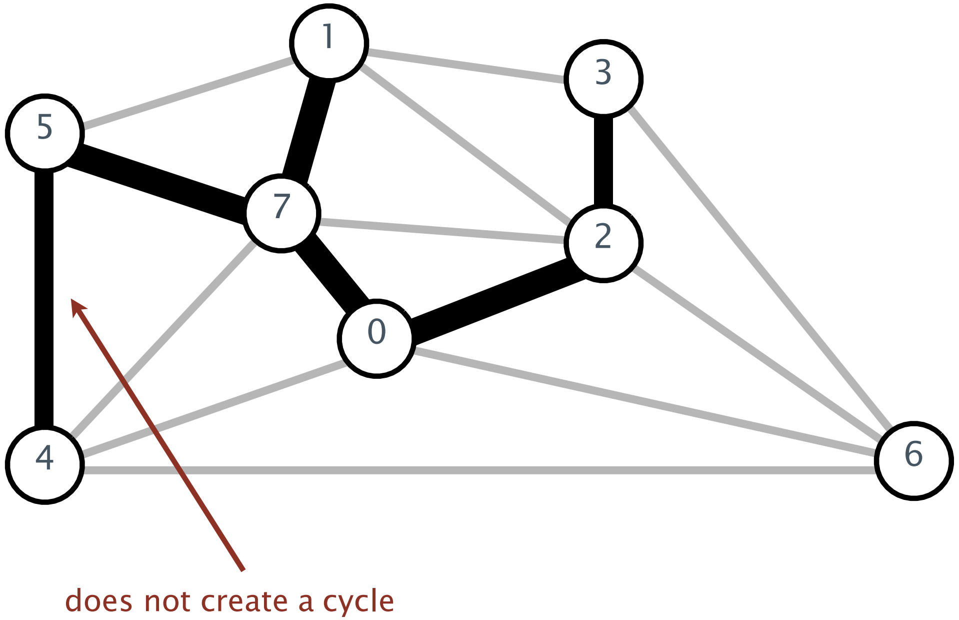
not in MST → 2-7 0.34

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

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

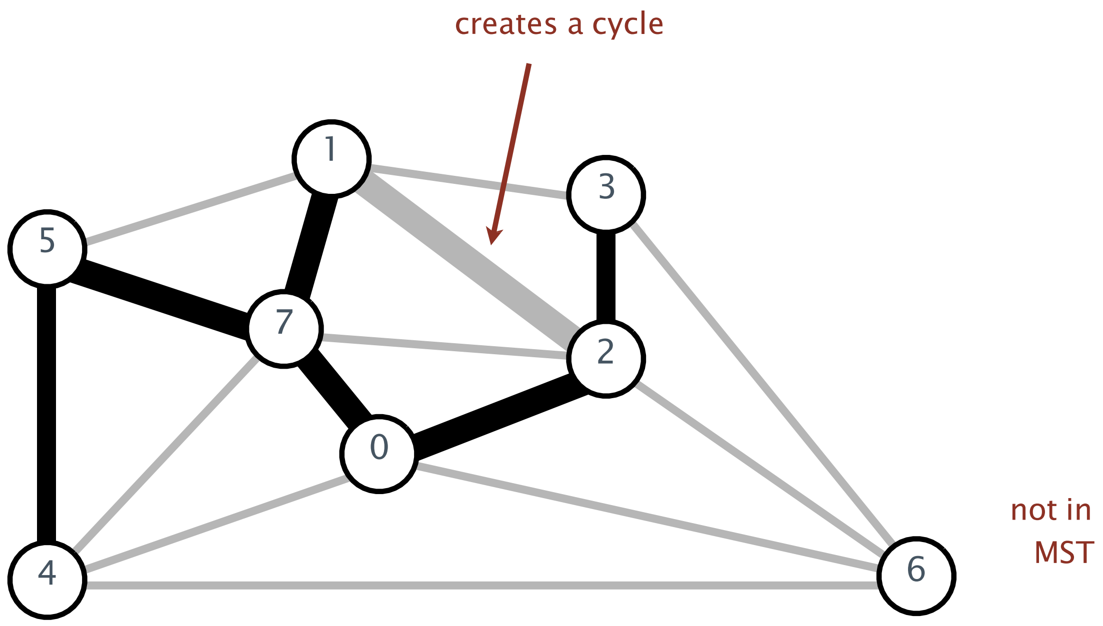


	0-7	0.16
	2-3	0.17
	1-7	0.19
	0-2	0.26
	5-7	0.28
	1-3	0.29
	1-5	0.32
	2-7	0.34
in MST →	4-5	0.35

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

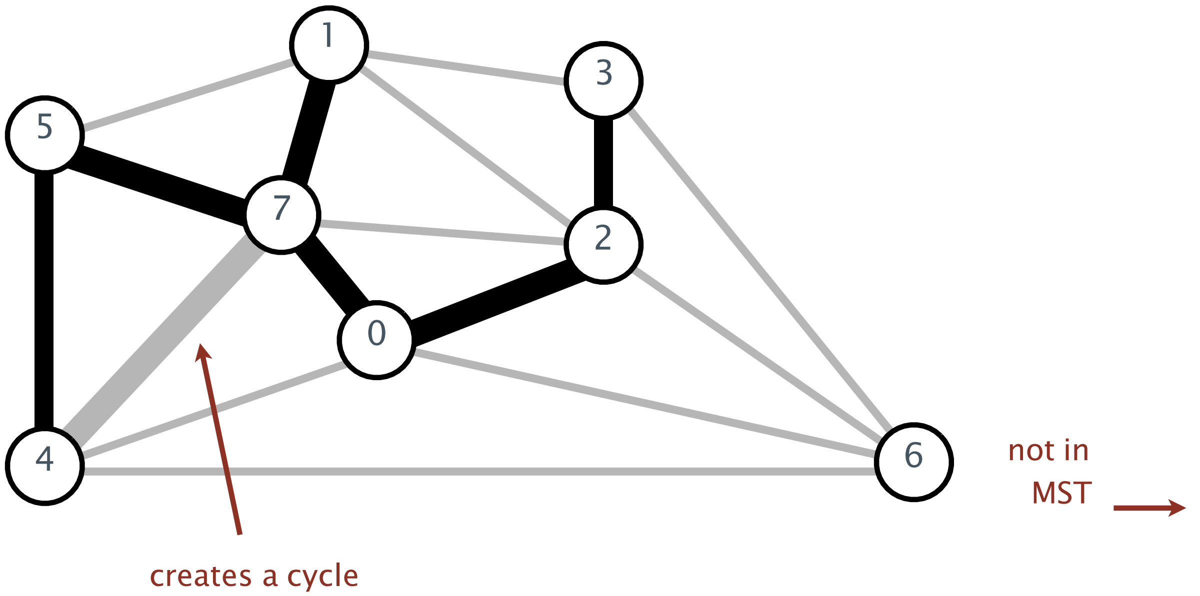


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

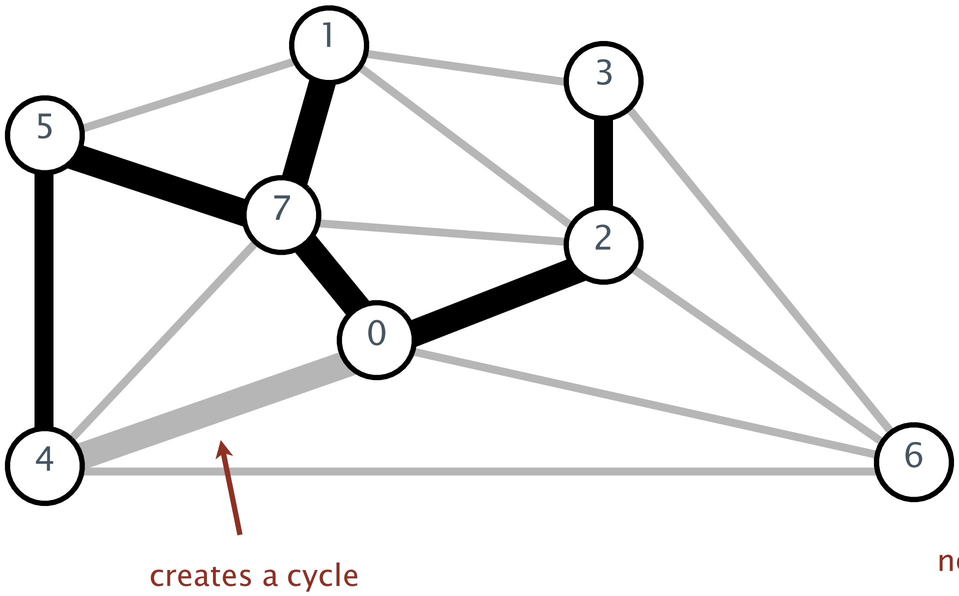


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



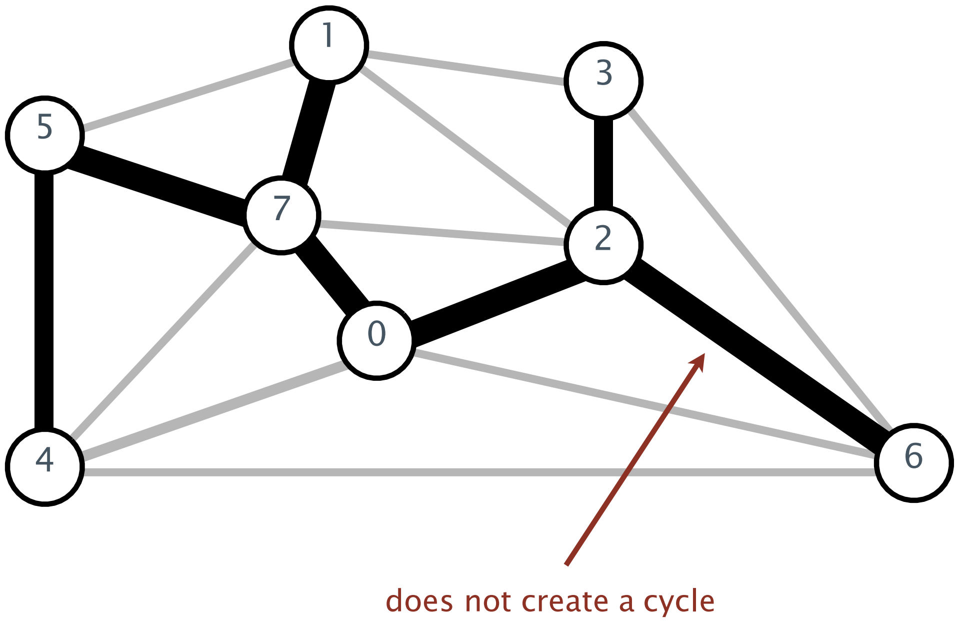
not in MST →

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

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



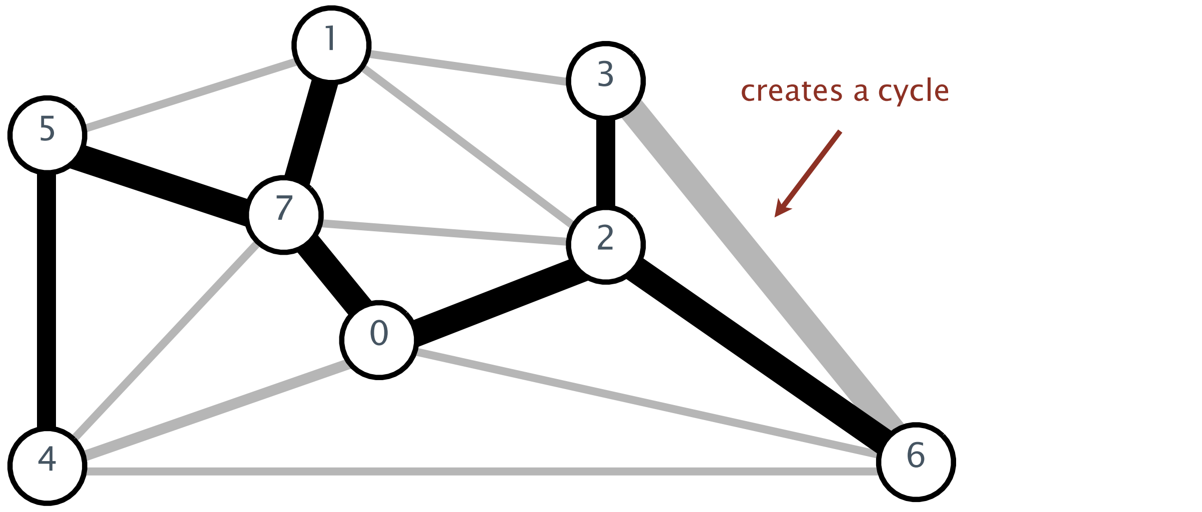
in MST →

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

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

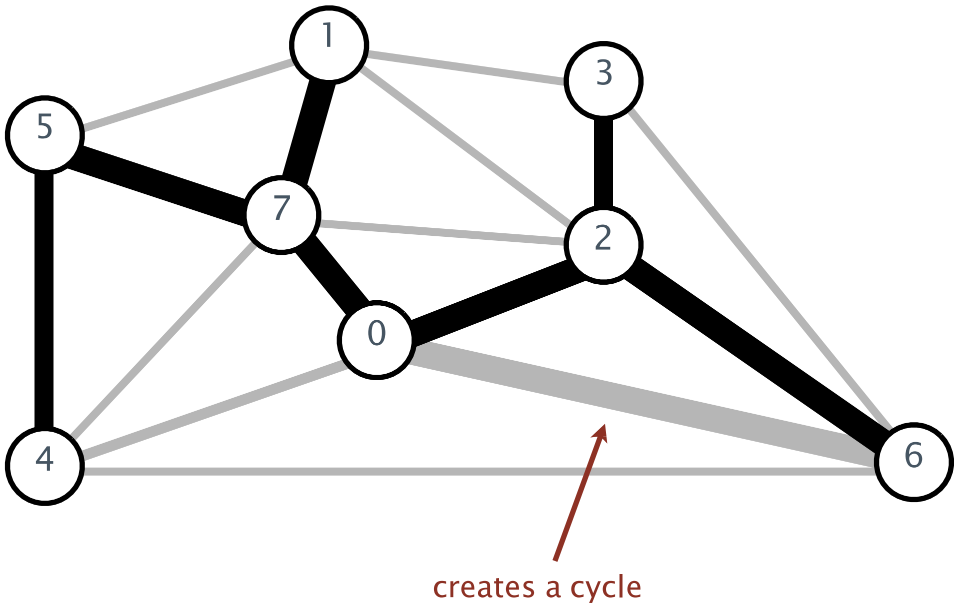


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



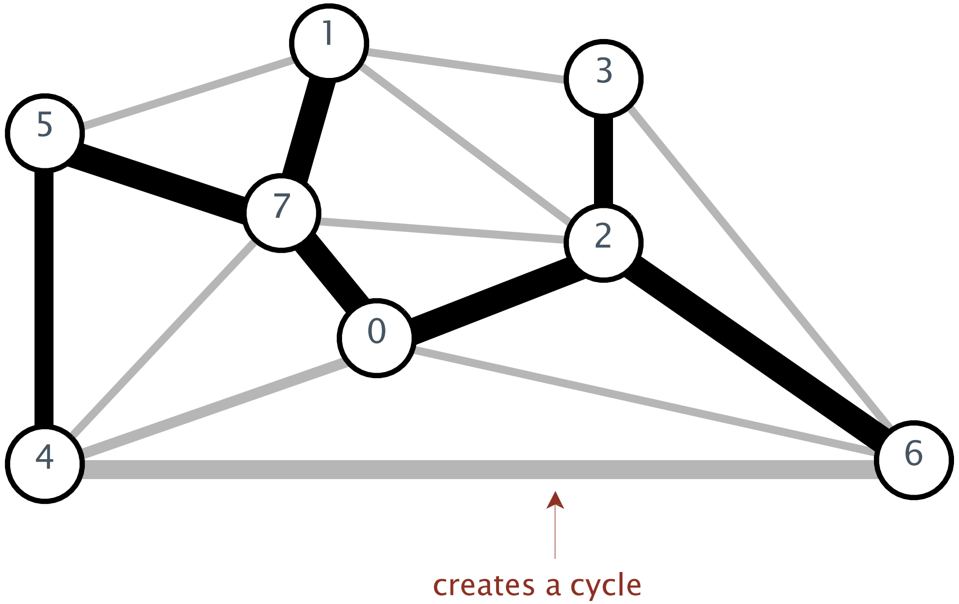
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58

not in MST

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



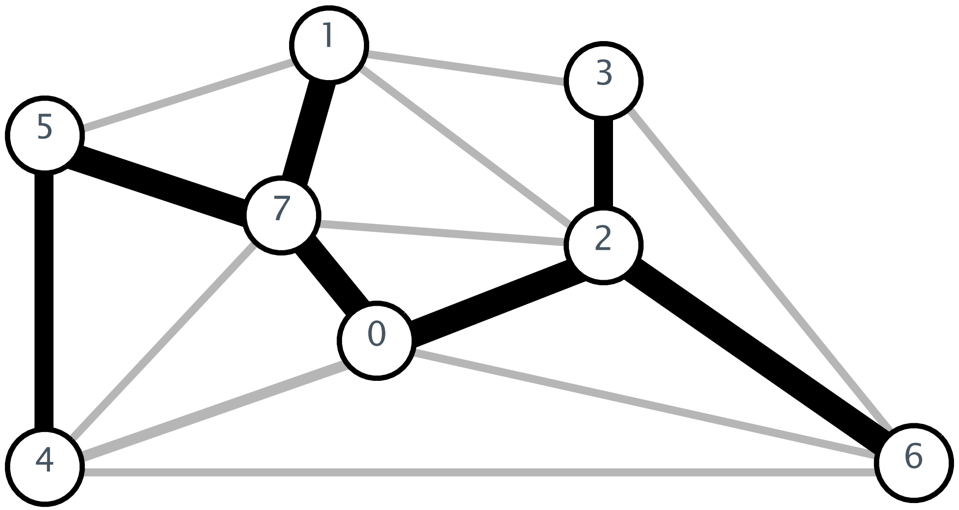
not in MST

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 demo

Consider edges in ascending order of weight.

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



a minimum spanning tree

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

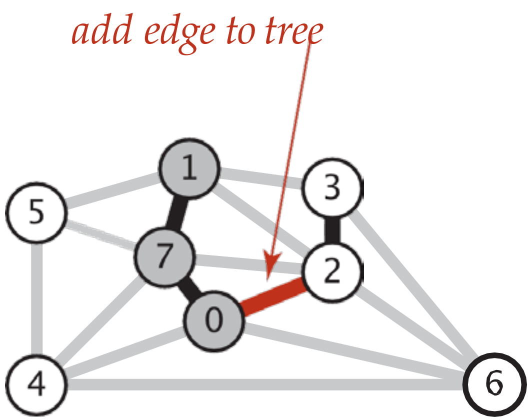
Kruskal’s progression visualisation

Kruskal's algorithm: correctness proof

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

Pf. Kruskal's algorithm is a special case of the greedy MST algorithm.

- Suppose Kruskal's algorithm colors the edge $e = v\text{–}w$ black.
- Cut = set of vertices connected to v in tree T.
- No crossing edge is black.
- No crossing edge has lower weight

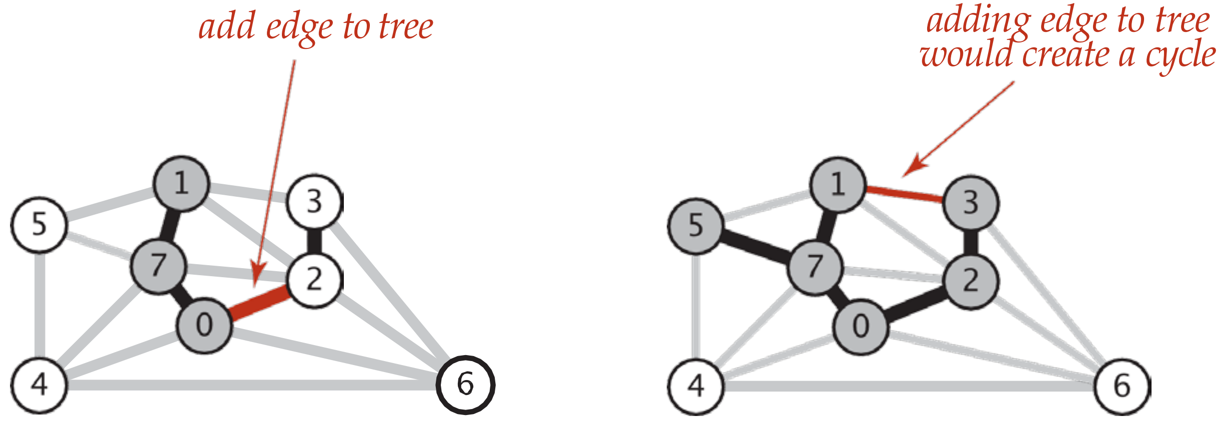


Kruskal's algorithm: implementation challenge

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

How difficult?

- $E + V$
- V ← run DFS from v, check if w is reachable (T has at most V – 1 edges)
- $\log V$
- $\log^* V$ ← use the union-find data structure !
- 1

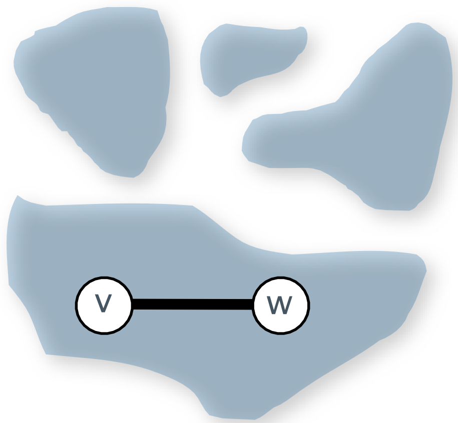


Kruskal's algorithm: implementation challenge

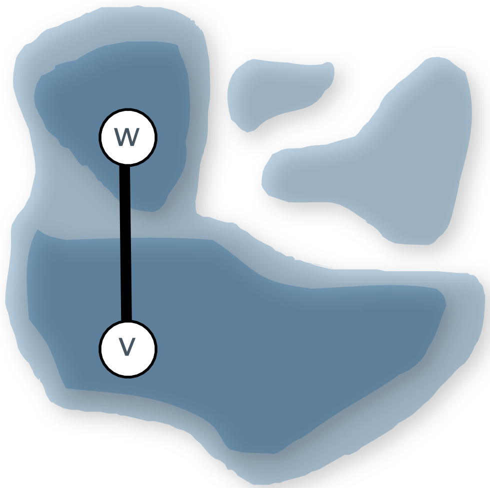
Challenge. Would adding edge $v–w$ to tree T create a cycle? If not, add it.

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 1: adding v–w creates a cycle



Case 2: add v–w to T and merge sets containing v and w

Kruskal implementation

- So what other data structures do we need?
  - Maintain the list of edges, ordered by weight, removing the lowest-weight edge when we add it to MST
  - List od edges and their weights added to the MST, to represent the MST (we'll need to iterate through them, and sum up their weight – to provide API required by MST)

Kruskal's algorithm: Java implementation

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

   public KruskalMST(EdgeWeightedGraph G)
   {
      MinPQ<Edge> pq = new MinPQ<Edge>(G.edges());

      UF uf = new UF(G.V());
      while (!pq.isEmpty() && mst.size() < G.V()-1)
      {
         Edge e = pq.delMin();
         int v = e.either(), w = e.other(v);
         if (!uf.connected(v, w))
         {
            uf.union(v, w);
            mst.enqueue(e);
         }
      }
   }

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

build priority queue (or sort)

greedily add edges to MST

edge v–w does not create cycle

merge sets

add edge to MST

Kruskal's algorithm: running time

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

Pf.

operation	frequency	time per op
build pq	1	E
delete-min	E	$\log E$
union	V	$\log^* V$ †
connected	E	$\log^* V$ †

† amortized bound using weighted quick union with path compression

recall: $\log^* V \le 5$ in this universe

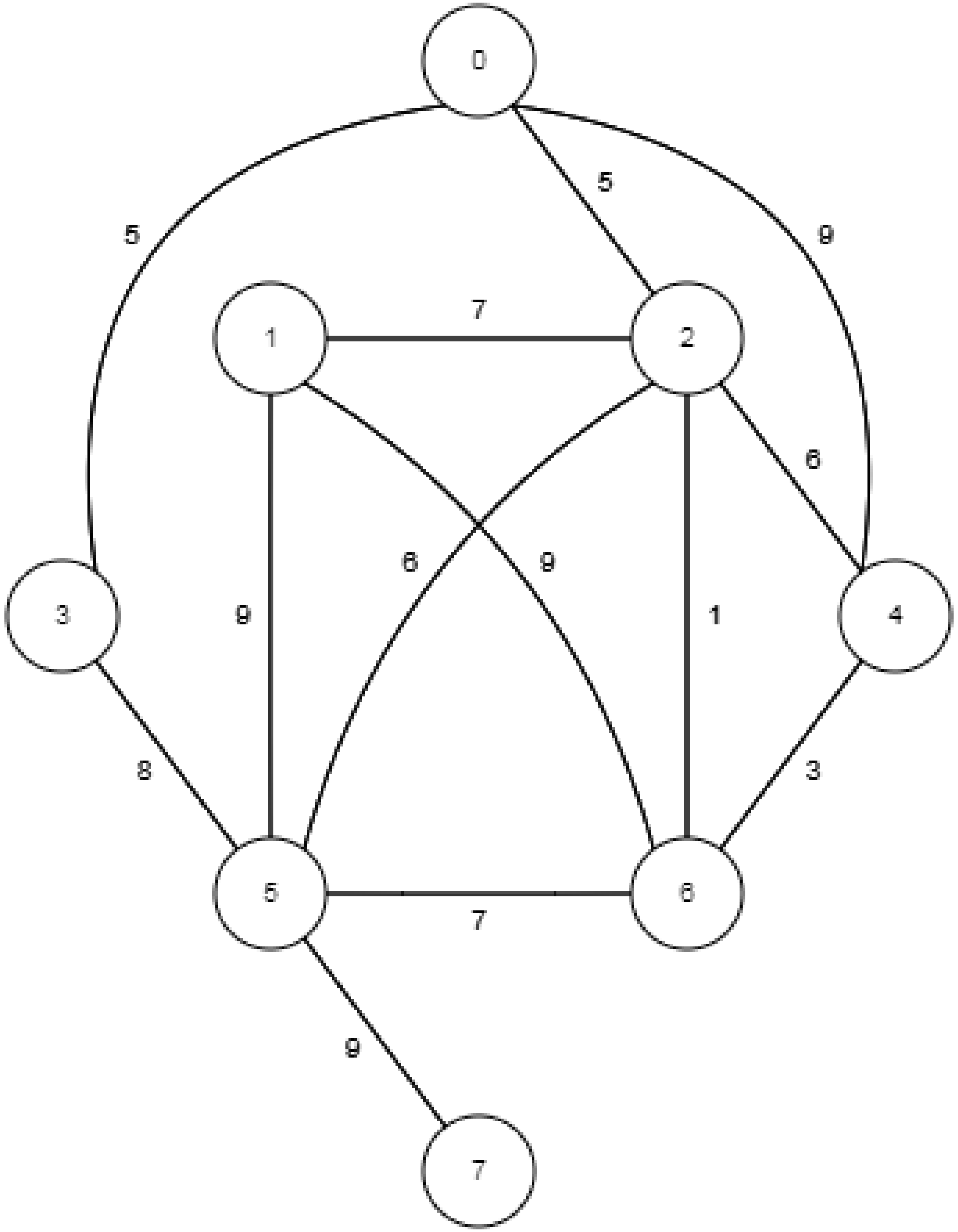
Remark. If edges are already sorted, order of growth is $E \log^* V$.

Kruskal's exercise

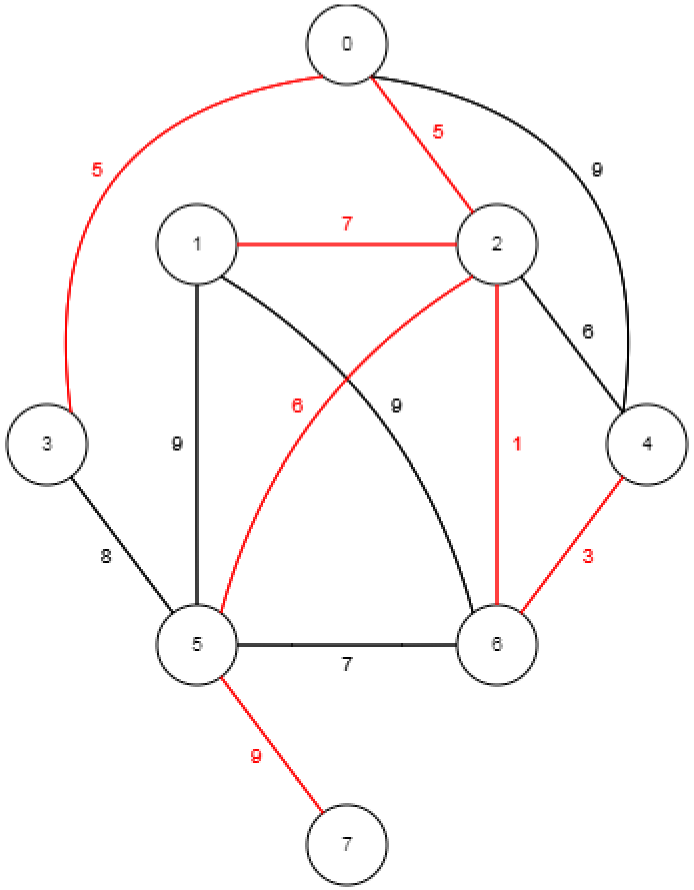
› Apply Kruskal's algorithm to find an MST of the following graph

› Provide trace of order in which edges are considered and added/discarded from being added to an MST

v	w	Weight	Added to MST?
2	6	1	yes
4	6	3	yes



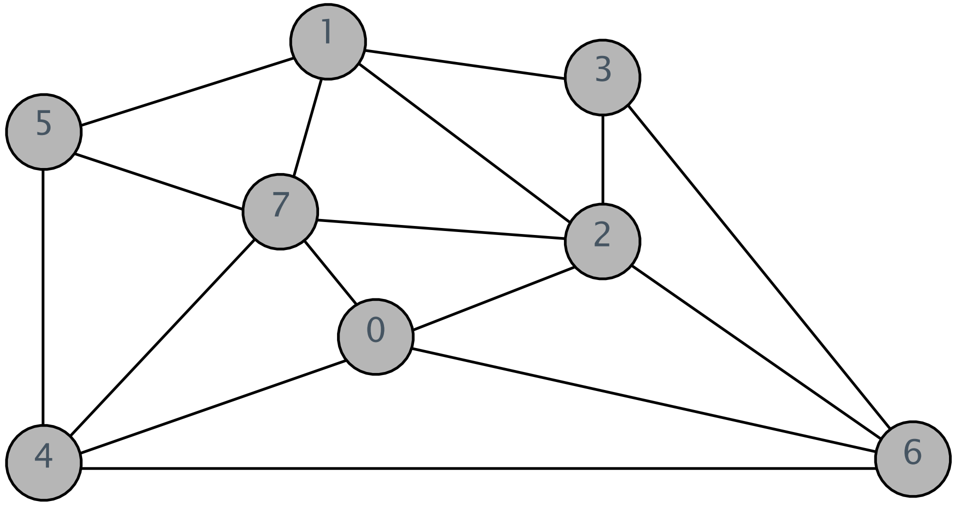
Kruskal's algorithm exercise solution



Prim’s algorithm

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.

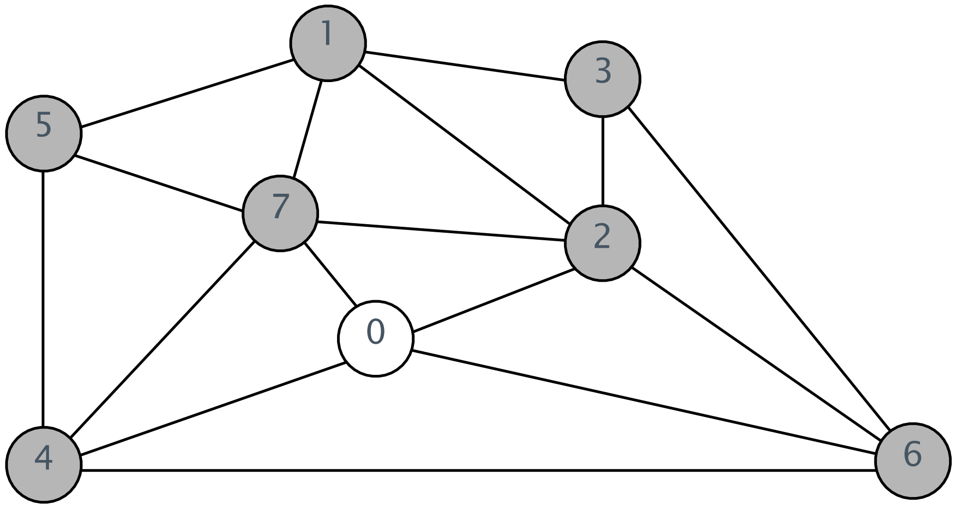


an edge-weighted graph

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

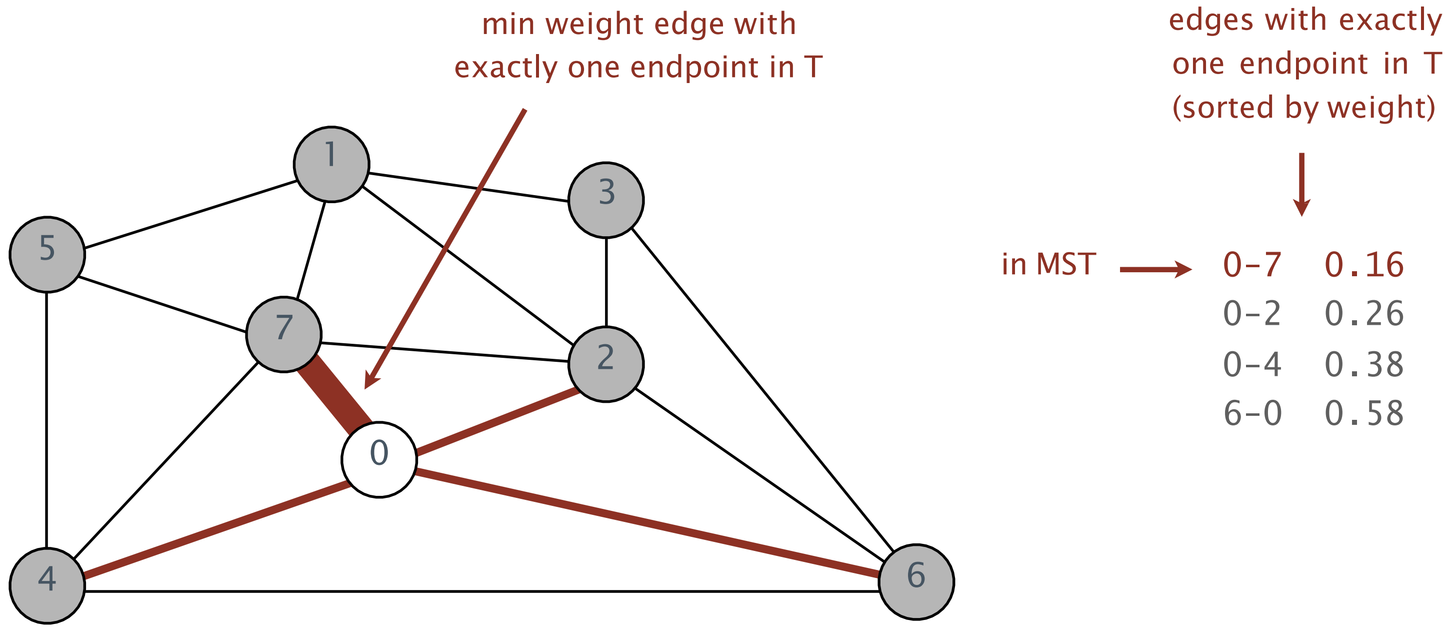
Prim's algorithm demo

- 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.



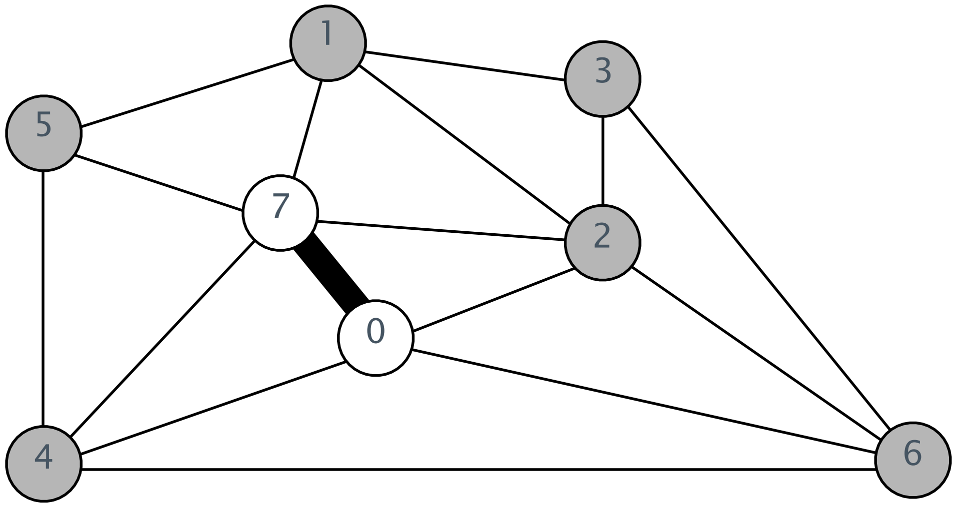
Prim's algorithm demo

- 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.



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.

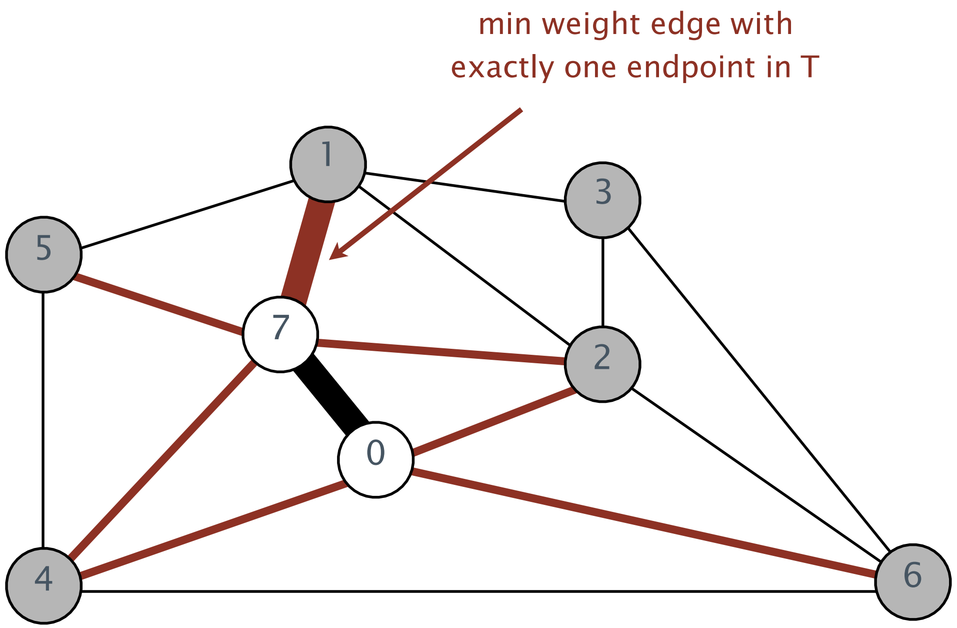


MST edges

`0-7`

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.



edges with exactly one endpoint in T (sorted by weight)

in MST →

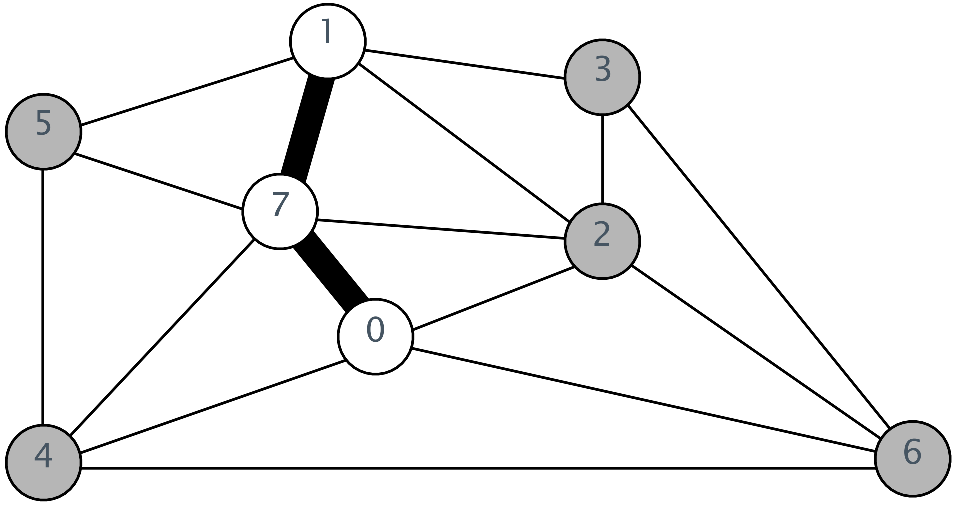
1-7	0.19
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

MST edges

0-7

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.

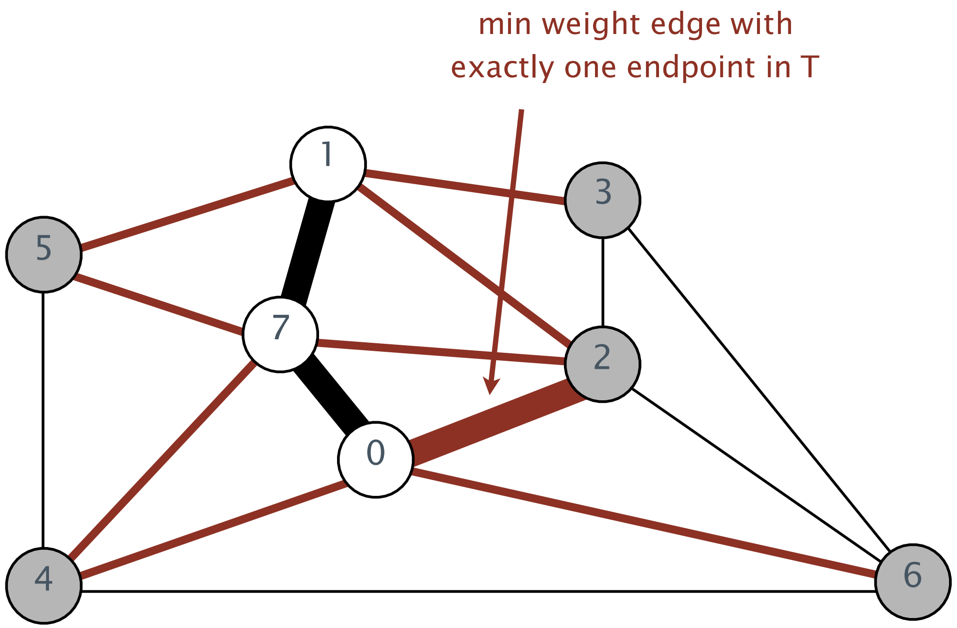


MST edges

`0-7  1-7`

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.



edges with exactly one endpoint in T (sorted by weight)

in MST →

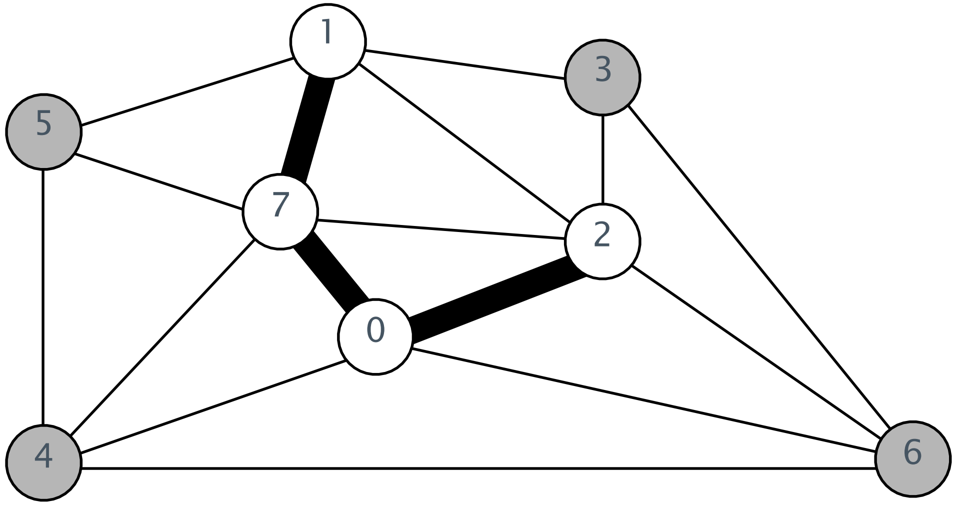
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
1-2	0.36
4-7	0.37
0-4	0.38
6-0	0.58

MST edges

0-7 1-7

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.

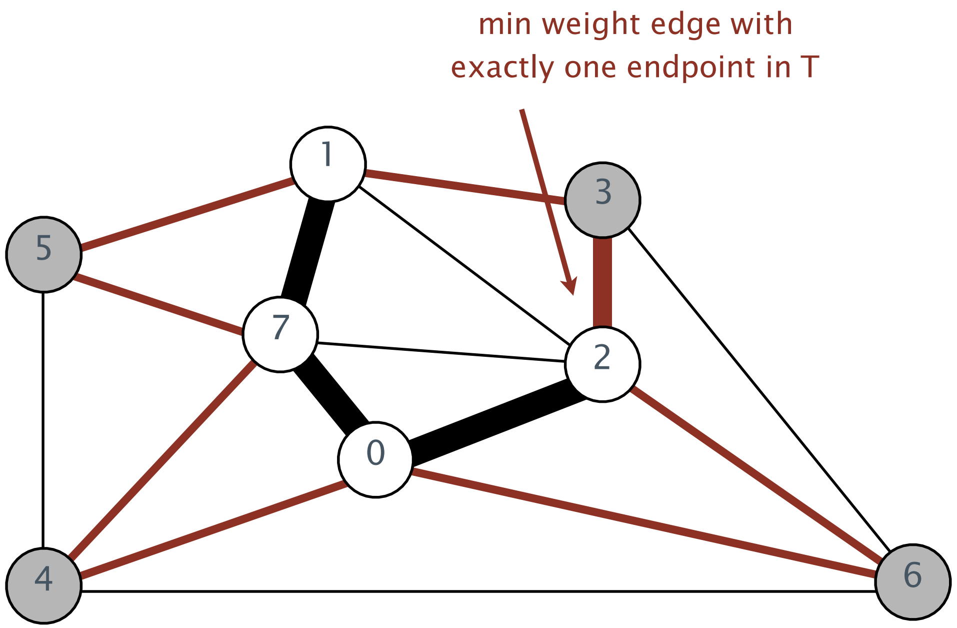


MST edges

0-7 1-7 0-2

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.



edges with exactly
one endpoint in T
(sorted by weight)

in MST

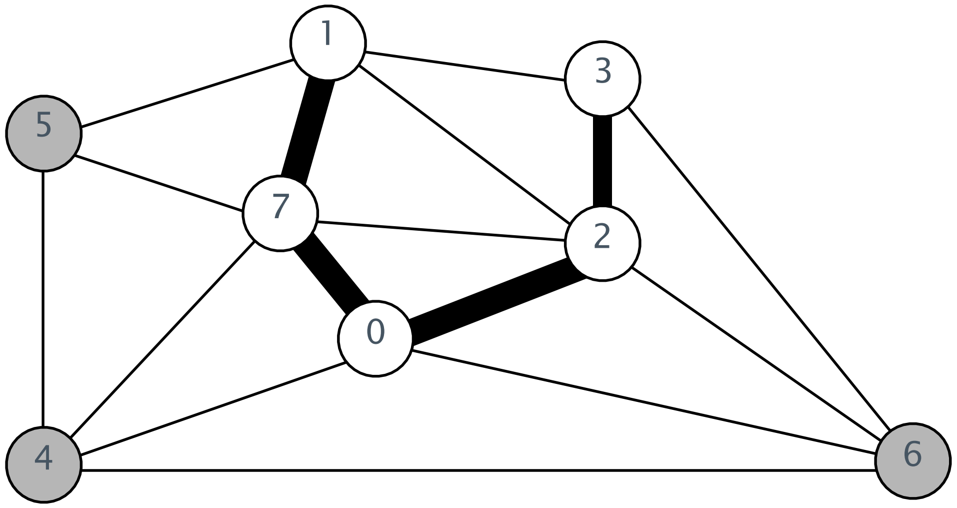
2-3	0.17
5-7	0.28
1-3	0.29
1-5	0.32
4-7	0.37
0-4	0.38
6-2	0.40
6-0	0.58

MST edges

0-7 1-7 0-2

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.

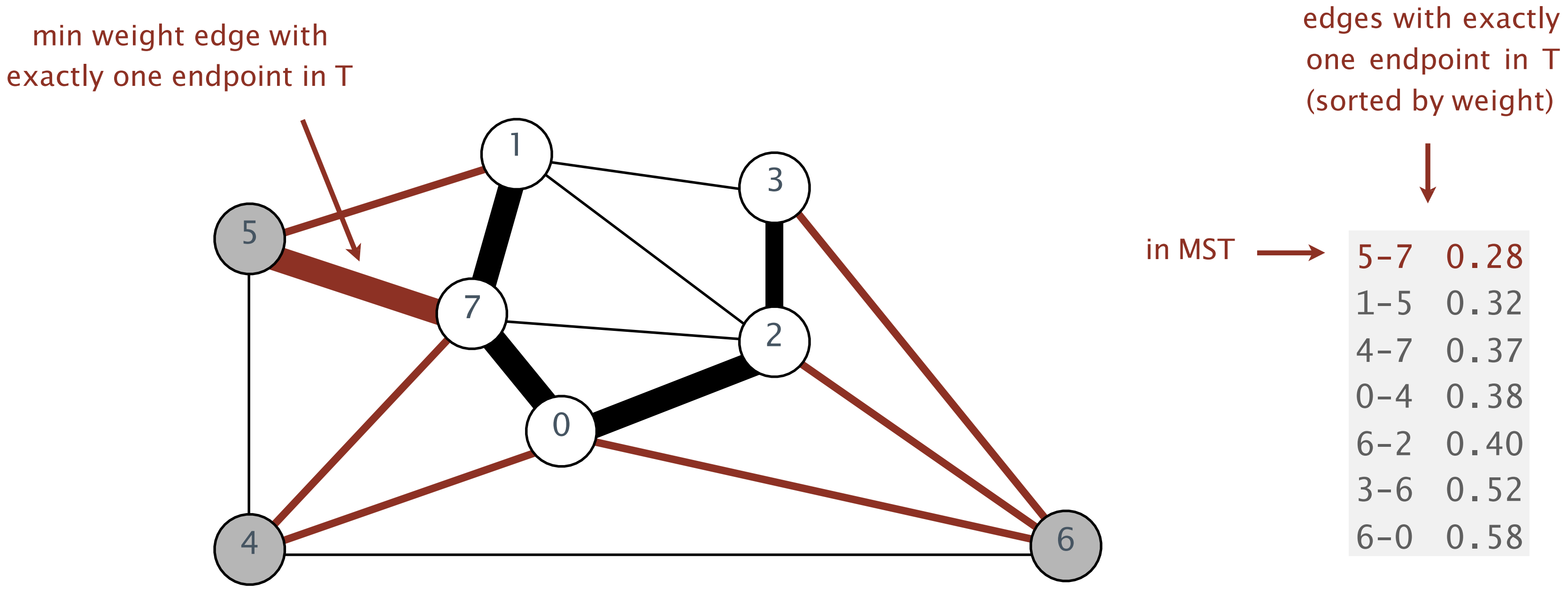


MST edges

`0-7  1-7  0-2  2-3`

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.

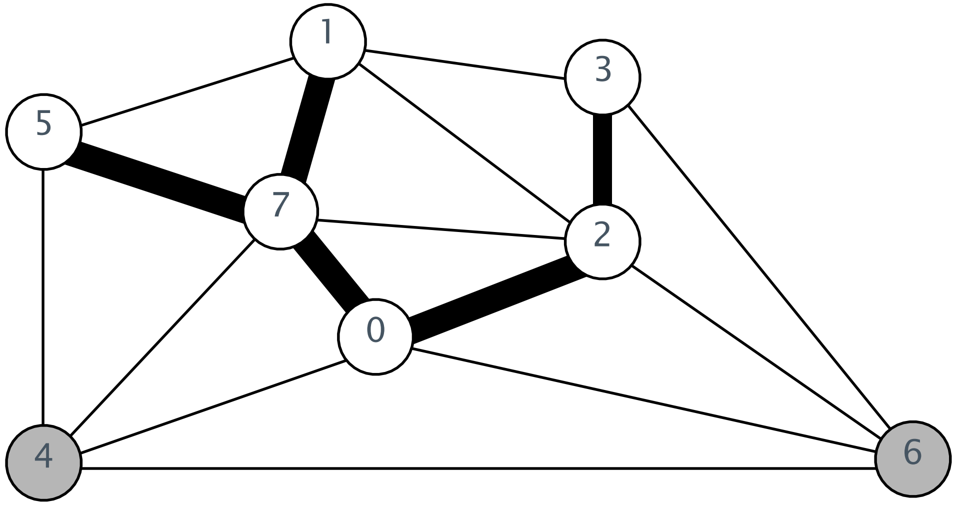


MST edges

0-7 1-7 0-2 2-3

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.



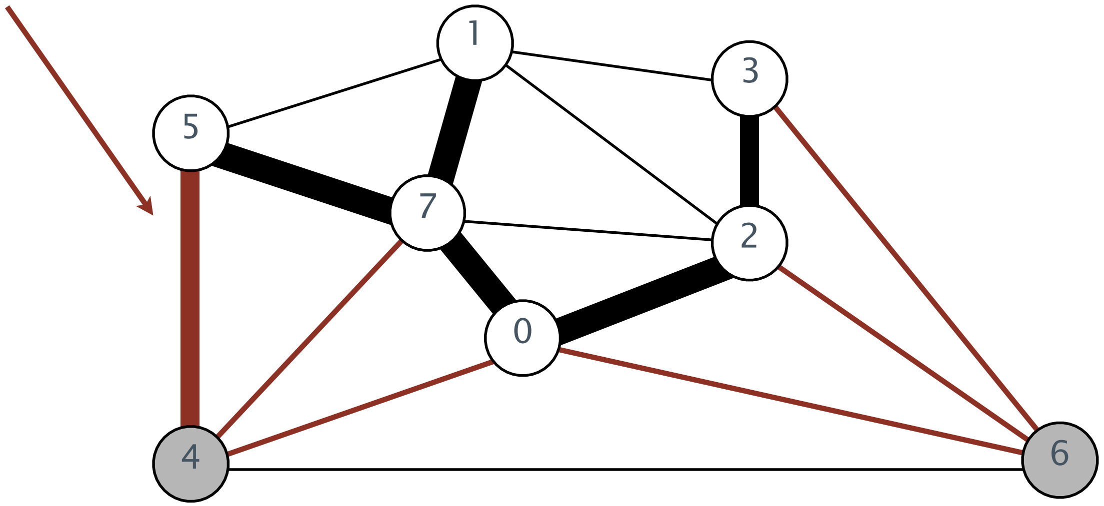
MST edges

`0-7   1-7   0-2   2-3   5-7`

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.

min weight edge with
exactly one endpoint in T



edges with exactly
one endpoint in T
(sorted by weight)

in MST → 4-5 0.35

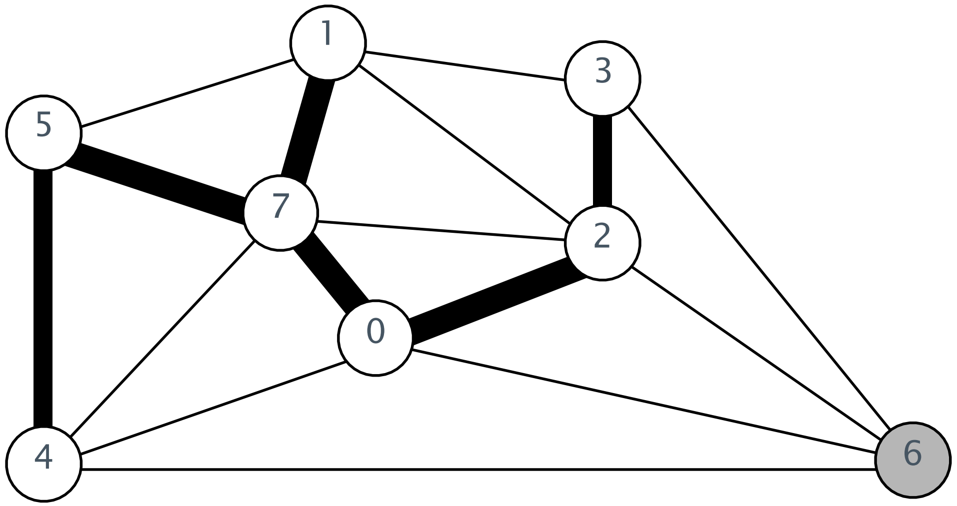
4-5	0.35
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58

MST edges

0-7 1-7 0-2 2-3 5-7

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.

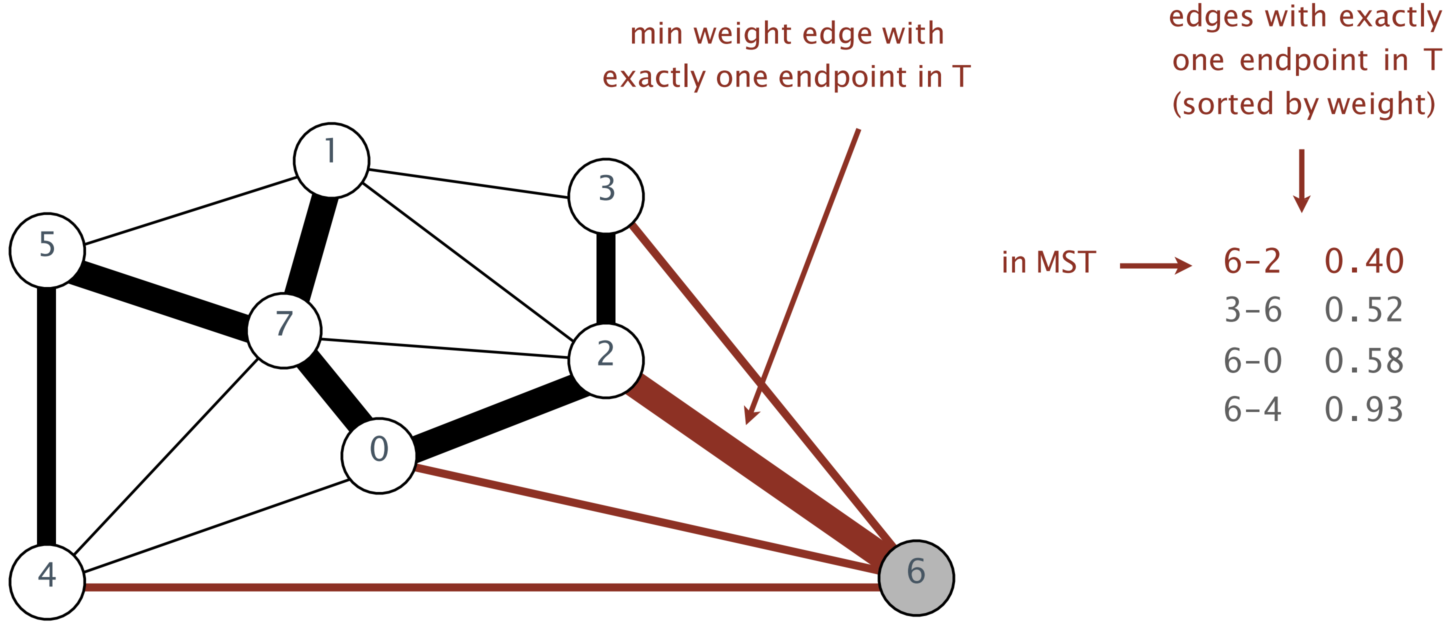


MST edges

`0-7   1-7   0-2   2-3   5-7   4-5`

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.

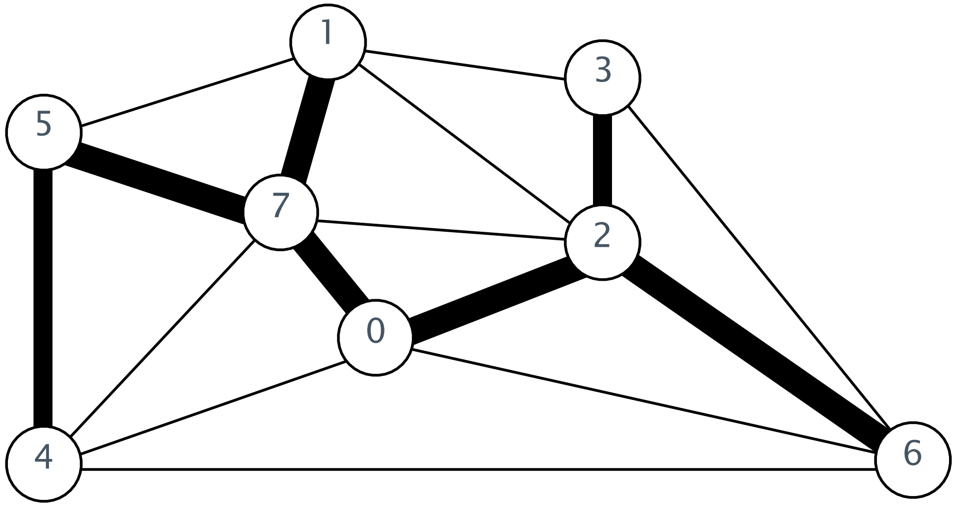


MST edges

0-7 1-7 0-2 2-3 5-7 4-5

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.



MST edges

0-7 1-7 0-2 2-3 5-7 4-5 6-2

Prim’s algorithm visualisation

Prim implementation

- So what other data structures do we need?
  - Maintain the list of edges, ordered by weight, removing the lowest-weight edge when we add it to MST
    - This list will be used slightly differently than in Kruskal – not list of all edges, just those with exactly one end-point in current MST
    - Lazy vs Eager implementation
  - List od edges and their weights added to the MST, to represent the MST (we’ll need to iterate through them, and sum up their weight – to provide API required by MST)

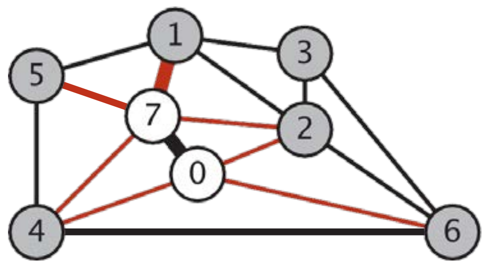
Prim's algorithm: lazy implementation

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 $e = v\text{–}w$ to add to T.
- Disregard if both endpoints v and w are marked (both in T).
- Otherwise, let w be the unmarked vertex (not in T):
  - add to PQ any edge incident to w (assuming other endpoint not in T)
  - add e to T and mark w

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

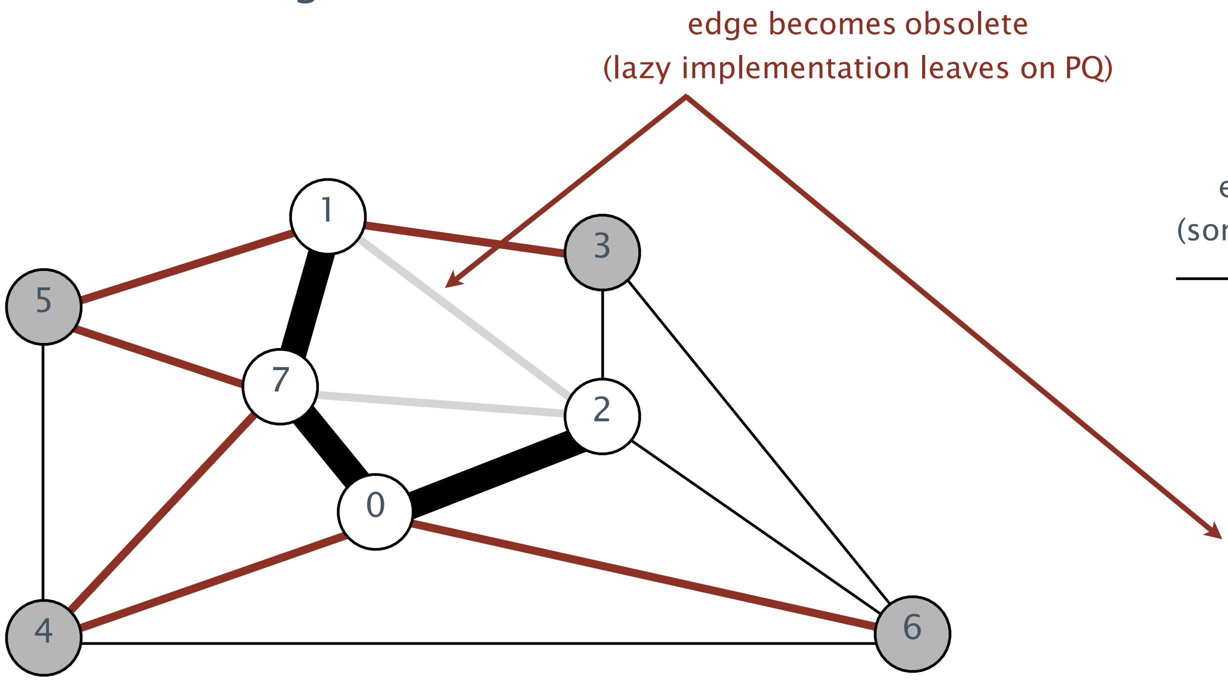


priority queue of crossing edges

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

- 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.



edges on PQ
(sorted by weight)

5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
1-2	0.36
4-7	0.37
0-4	0.38
6-0	0.58

MST edges

0-7 1-7 0-2

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 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).

Pf.

operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

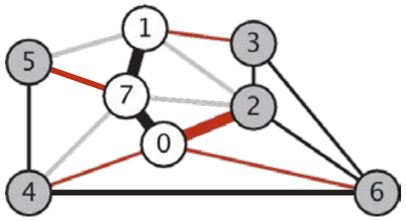
Prim's algorithm: eager implementation

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

pq has at most one entry per vertex

Eager solution. Maintain a PQ of vertices connected by an edge to T, where priority of vertex v = weight of shortest edge connecting v to T.

- Delete min vertex v and add its associated edge $e = v–w$ to T.
- Update PQ by considering all edges $e = v–x$ incident to v
  - ignore if x is already in T
  - add x to PQ if not already on it
  - decrease priority of x if $v–x$ becomes shortest edge connecting x to T



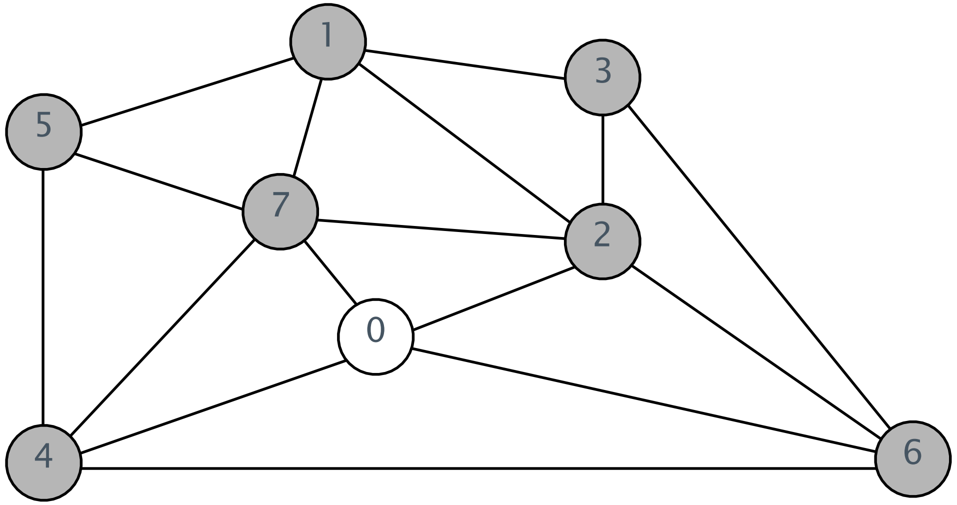
0		
1	1-7	0.19
2	0-2	0.26
3	1-3	0.29
4	0-4	0.38
5	5-7	0.28
6	6-0	0.58
7	0-7	0.16

red: on PQ

black: on MST

Prim's algorithm: eager implementation demo

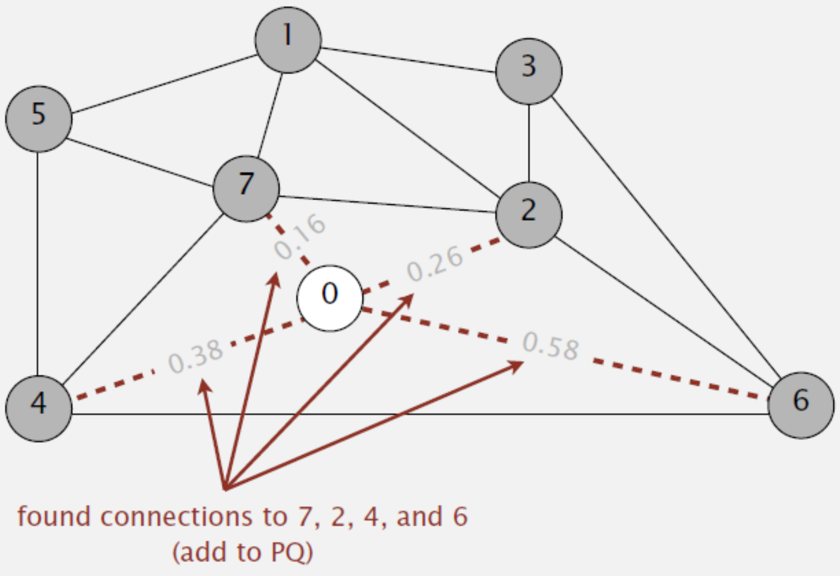
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



v	edgeTo[]	distTo[]
0	-	-

Prim's algorithm: eager 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.

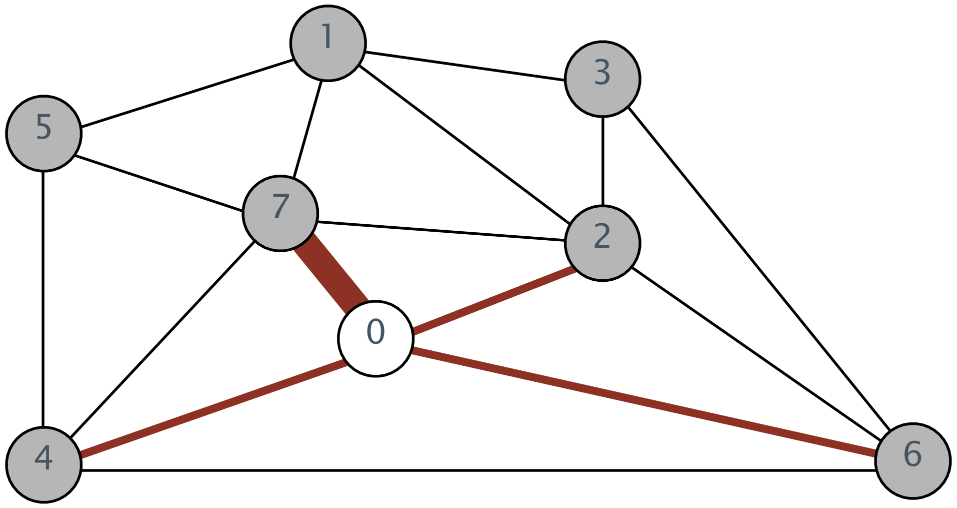


v	edgeTo[]	distTo[]
0	–	–
7	0–7	0.16
2	0–2	0.26
4	0–4	0.38
6	6–0	0.58

vertices on PQ
(sorted by weight)

Prim's algorithm: eager 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.

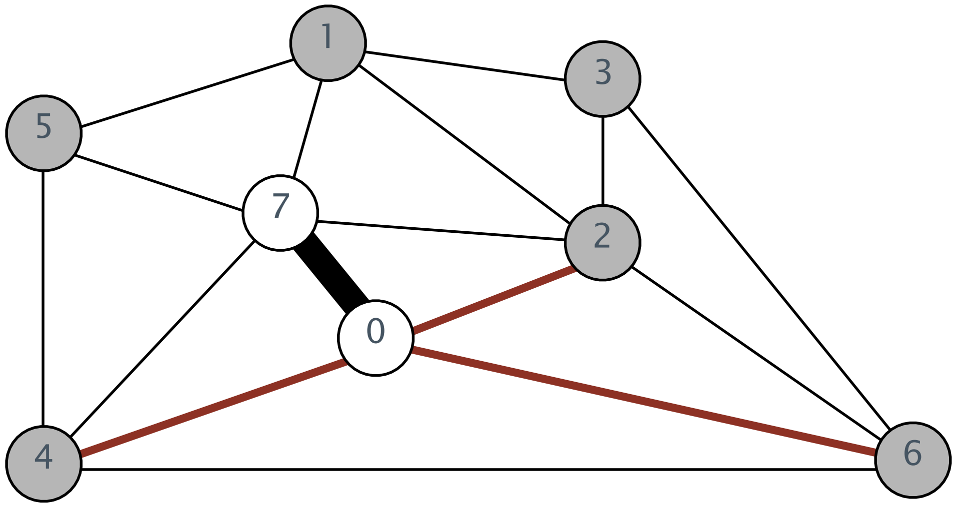


v	edgeTo[]	distTo[]
0	–	–
7	0–7	0.16
2	0–2	0.26
4	0–4	0.38
6	6–0	0.58

vertices on PQ
(sorted by weight)

Prim's algorithm: eager 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.



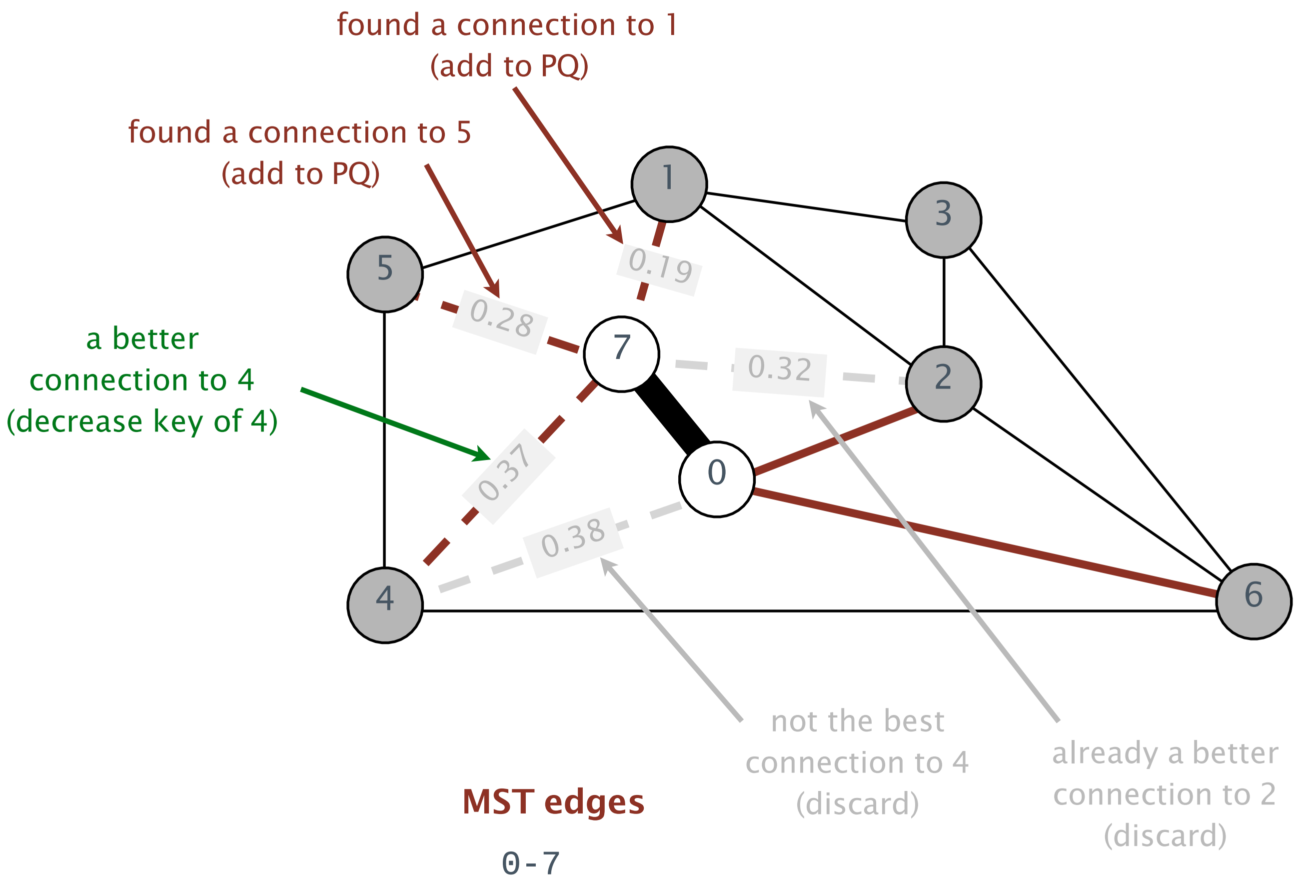
v	edgeTo[]	distTo[]
0	–	–
7	0–7	0.16
2	0–2	0.26
4	0–4	0.38
6	6–0	0.58

MST edges

0-7

Prim's algorithm: eager 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.



v	edgeTo[]	distTo[]
0	–	–
→ 7	0–7	0.16
1	1–7	0.19
2	0–2	0.26
5	5–7	0.28
4	~~0–4~~ 4–7	~~0.38~~ 0.37
6	6–0	0.58

vertices on PQ
(sorted by weight)

Indexed priority queue

Associate an index between 0 and $N - 1$ with each key in a priority queue.

- Supports insert and delete-the-minimum.
- Supports decrease-key given the index of the key.

`public class IndexMinPQ<Key extends Comparable<Key>>`

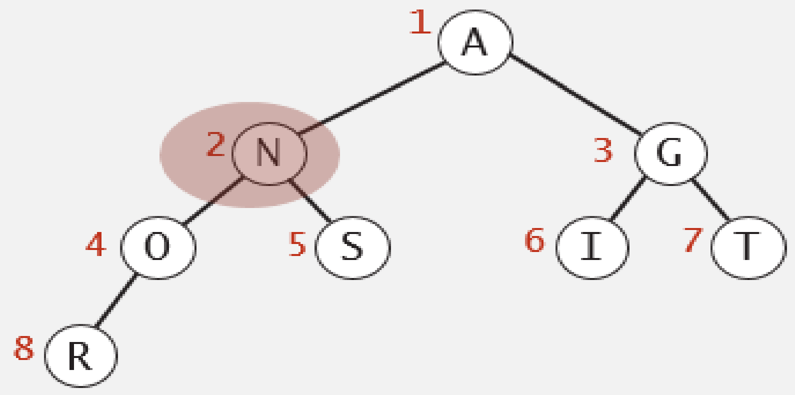
	`IndexMinPQ(int N)`	*create indexed priority queue with indices 0, 1, …, N – 1*
`void`	`insert(int i, Key key)`	*associate key with index i*
`void`	`decreaseKey(int i, Key key)`	*decrease the key associated with index i*
`boolean`	`contains(int i)`	*is i an index on the priority queue?*
`int`	`delMin()`	*remove a minimal key and return its associated index*
`boolean`	`isEmpty()`	*is the priority queue empty?*
`int`	`size()`	*number of keys in the priority queue*

Indexed priority queue implementation

Binary heap implementation. [see Section 2.4 of textbook]

- Start with same code as `MinPQ`.
- Maintain parallel arrays `keys[]`, `pq[]`, and `qp[]` so that:
  - `keys[i]` is the priority of `i`
  - `pq[i]` is the index of the key in heap position `i`
  - `qp[i]` is the heap position of the key with index `i`
- Use `swim(qp[i])` to implement `decreaseKey(i, key)`.

i	0	1	2	3	4	5	6	7	8
keys[i]	A	S	O	R	T	I	N	G	-
pq[i]	-	0	6	7	2	1	5	4	3
qp[i]	1	5	4	8	7	6	2	3	-



Prim's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

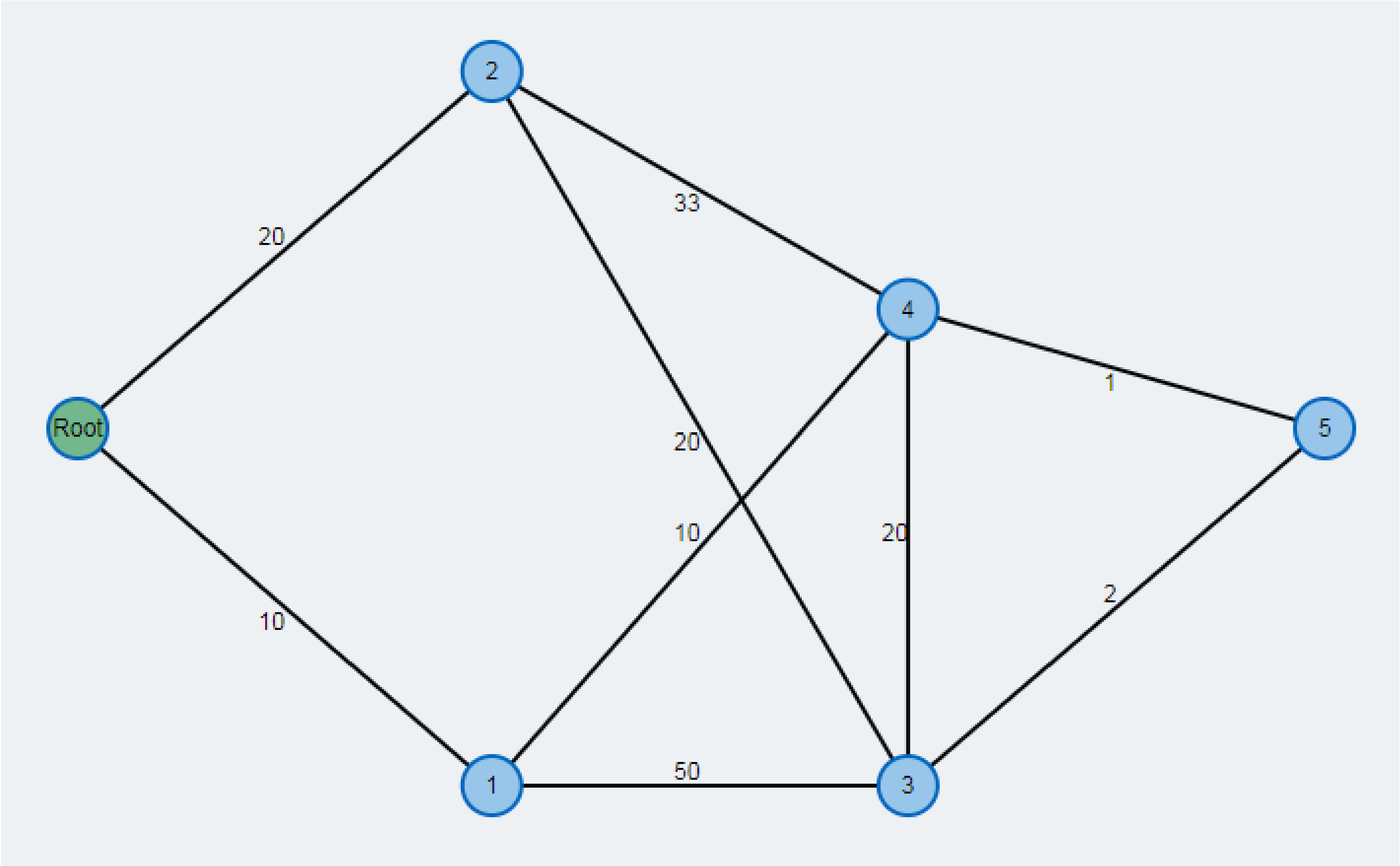
† amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.

Prim’s algorithm exercise- Turning Point

- Starting from root (0)
- show order in which edges are added to mst
- Status of queue of vertices with edgeTo and distance values



MST for Digraphs?

› Equivalent for digraphs is "minimum spanning arborescence" which will produce a tree where every vertex can be reached from a single vertex (panning arborescence of minimum weight, optimum branching)

Greedy Algorithms

- Applicable to optimisation problems
- Constructs a solution through a sequence of steps, each expanding on the partially constructed solution obtained so far, until a complete solution is built
- On each step, a choice is made which is:
  1. Feasible – has to satisfy problem constraints
  2. Locally optimal – it has to be the best local choice among all feasible choices
  3. Irrevocable – once made, it cannot be changed on subsequent steps of the algorithm
- Greedily takes the best current option, in the hope it will add up to the overall best option
  - in some problems it does, in some it doesn't, but approximation might be good enough
- Dijsktra's shortest path algorithm is also greedy