Announcements

Exam Regrades

- Due by Wednesday's lecture.

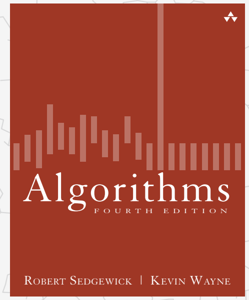
Teaching Experiment: Dynamic Deadlines (WordNet)

- Right now, WordNet is due at 11 PM on April 8th.
- Starting Tuesday at 11 PM:
  - Every submission that passes all Dropbox tests shortens the time limit by 30 minutes.
  - Maximum of 12 hours per day.
  - 3 hour grace period still applies.
- Email will be sent out every night at midnight with new deadline.
- I am lying.

"Dynamic Deadlines for Encouraging Earlier Participation on Assignments," Garcia, Dan. SIGCSE 2013
http://db.grinnell.edu/sigcse/sigcse2013/Program/viewAcceptedSession.asp?sessionID=7220

Algorithms

Robert Sedgewick | Kevin Wayne



http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *MST Basics, Kruskal, Prim*
- *Why Kruskal and Prim work*
- *Kruskal Implementation*
- *Prim Implementation*
- *Harder Problems*

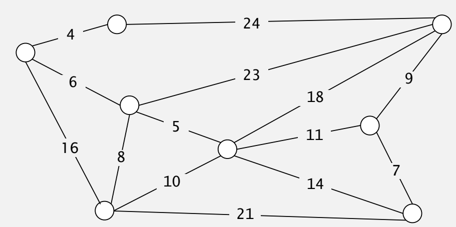
http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *MST Basics, Kruskal, Prim*
- *Why Kruskal and Prim work*
- *Kruskal Implementation*
- *Prim Implementation*
- *Harder Problems*

Minimum spanning tree and edge weighted graphs

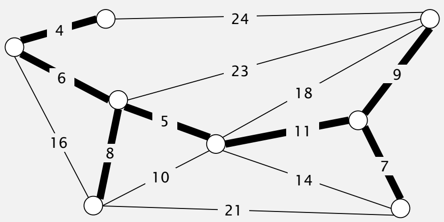
Given. Undirected graph G with positive edge weights (connected).
Def. A spanning tree of G is a subgraph T that is connected and acyclic.
Goal. Find a min weight spanning tree.



graph G

Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).
Def. A spanning tree of G is a subgraph T that is connected and acyclic.
Goal. Find a min weight spanning tree.

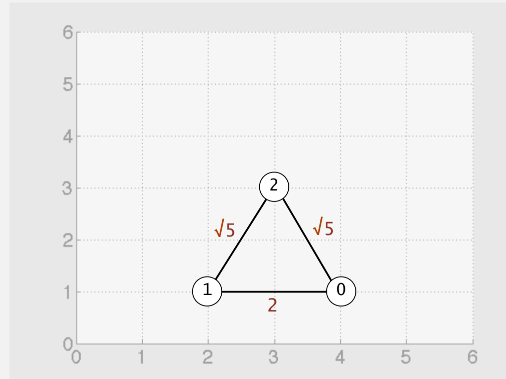


spanning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

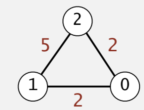
Brute force. Try all spanning trees? There are ~V^V of them.

Drawing conventions

Textbook Convention. Edges are drawn with length proportional to weight.
Constraint. This convention constrains the set of possible graphs.



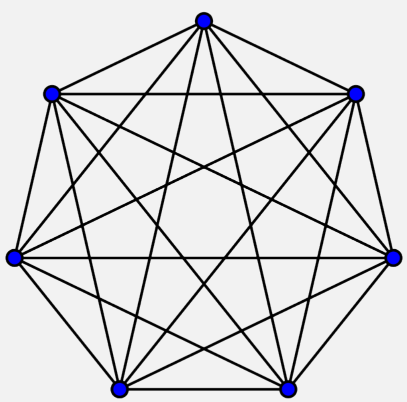
Allowable graph
Can be drawn with length = weight



Allowable graph
Cannot be drawn with length = weight

Drawing convention

Textbook Convention #2. Edges are straight lines and never cross.
Constraint. This convention constrains the set of possible graphs.

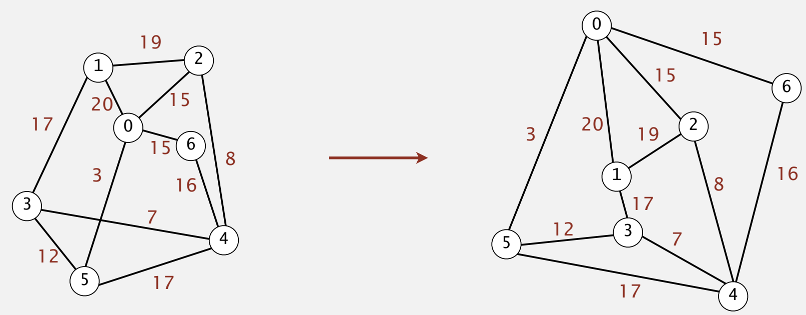


http://en.wikipedia.org/wiki/File:Complete_graph_K7.svg
Textbook graphs typically avoid crossings because they're hard to read

Drawing convention

Textbook Convention #2. Edges are straight lines and never cross.
Constraint. This convention constrains the set of possible graphs.

Q: How hard is it to determine whether a graph can be redrawn in a plane?

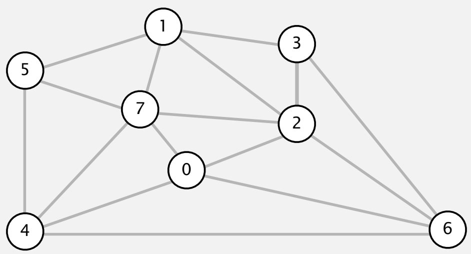


http://www.cs.princeton.edu/courses/archive/spring13/cos226/studyGuide.html

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

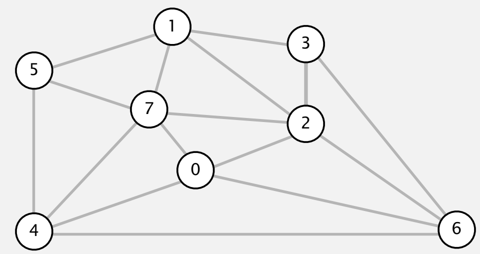
graph edges
sorted by weight

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.



an edge-weighted graph

graph edges
sorted by weight

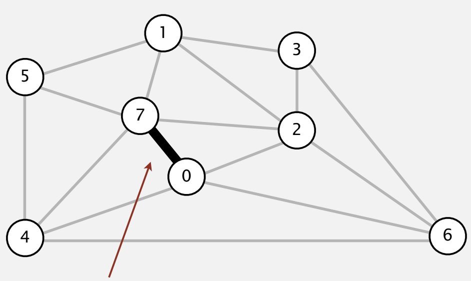
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

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



does not create a cycle

Kruskal's algorithm demo

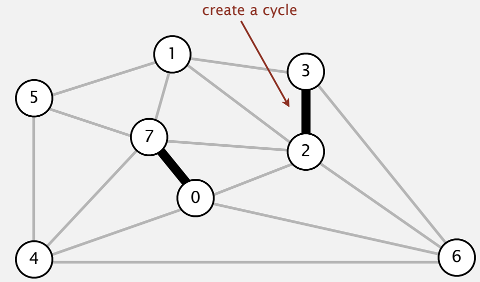
Consider edges in ascending order of weight.

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

does not
create a cycle

0-7	0.16
2-3	0.17

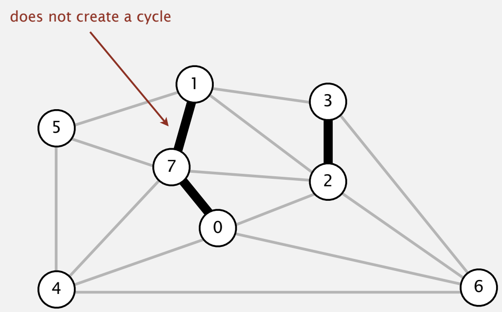
in MST → 2-3 0.17



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

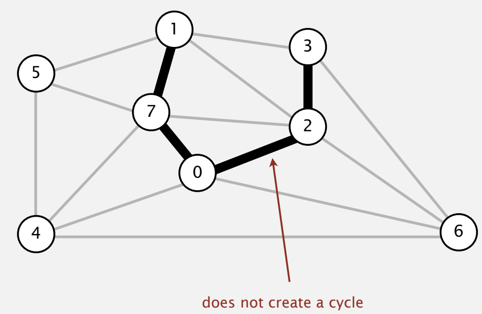
pollEv.com/jhug text to **37607**

Q: Which edge comes next?
A. 4-5 [127350]
B. 4-0 [127809]
C. 2-0 [127963]

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



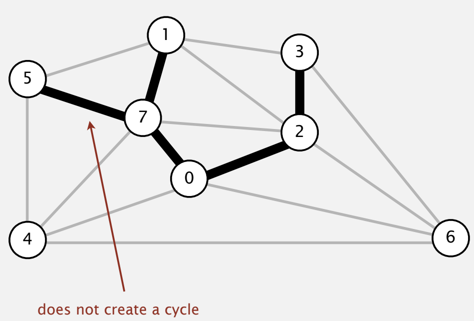
0-7	0.16
2-3	0.17
1-7	0.19
in MST → 0-2	0.26

Q: Which edge comes next?
C. 2-0 [127963]

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

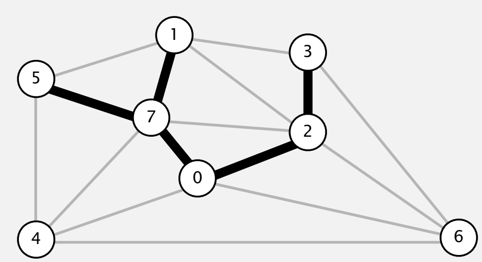


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
in MST → 5-7	0.28

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



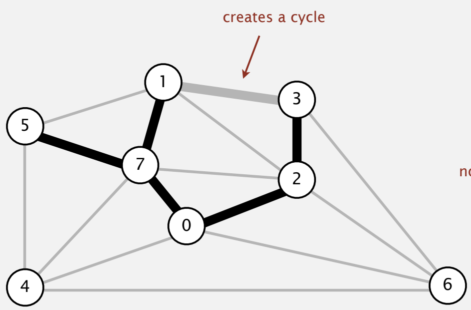
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35

Q: Which edge comes next?

Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

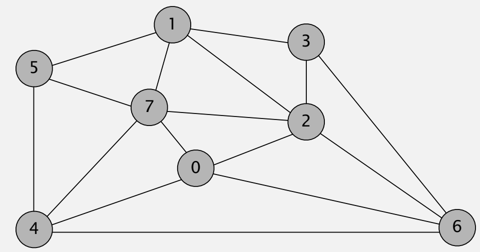


not in MST →

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

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.

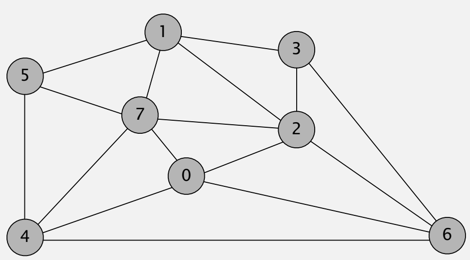


an edge-weighted graph

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.

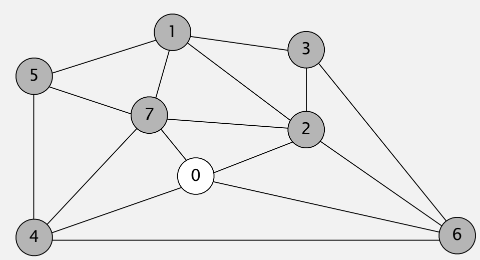


an edge-weighted graph

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

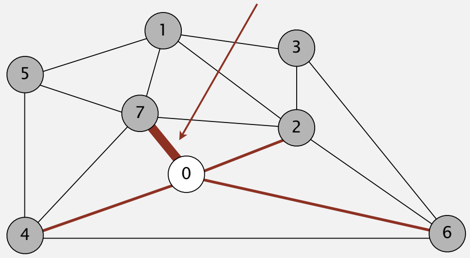
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



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.

min weight edge with exactly one endpoint in T

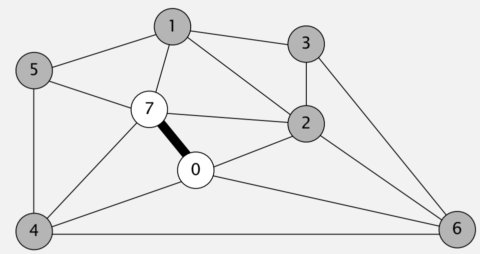


edges with exactly one endpoint in T (sorted by weight)

in MST → 0-7 0.16
0-2 0.26
0-4 0.38
6-0 0.58

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.

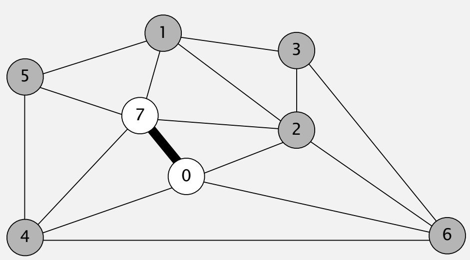


MST edges

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.



2-3 0.17
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

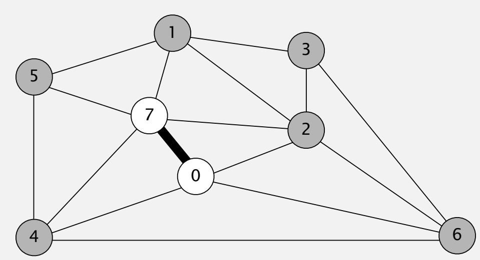
pollEv.com/jhug text to **37607**

Q: Which edge is added next to the MST?

A. 2-3 [149931]
B. 1-7 [149933]
C. 6-0 [149934]

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.



2-3 0.17
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

pollEv.com/jhug text to **37607**

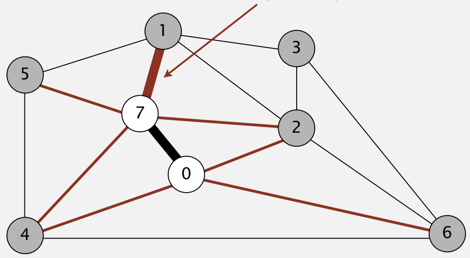
Q: Which edge is added next to the MST?

B. 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.

min weight edge with
exactly one endpoint in T



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

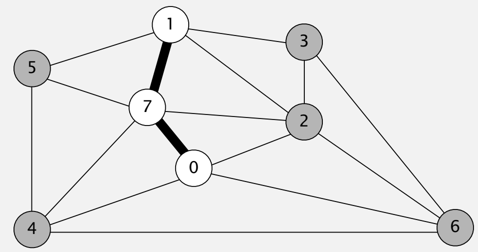
in MST → 1-7 0.19
0-2 0.26
5-7 0.28
2-7 0.34
4-7 0.37
0-4 0.38
6-0 0.58

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.



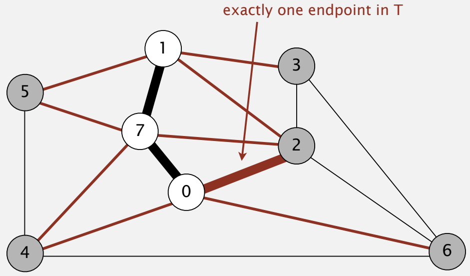
MST edges

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.

min weight edge with
exactly one endpoint in T



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

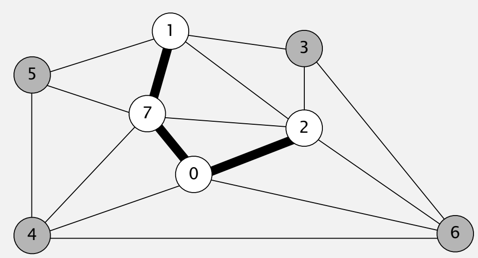
in MST → 0-2 0.26
5-7 0.28
1-3 0.29
1-5 0.32
2-7 0.34
1-2 0.36
4-7 0.37
0-4 0.38
6-0 0.58

MST edges

0-7 1-7

Prim's algorithm demo

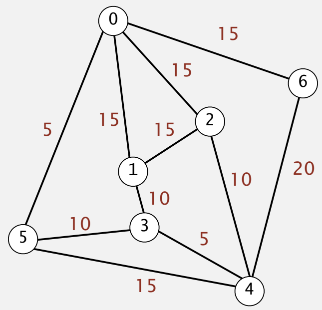
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



MST edges

0-7 1-7 0-2

MST

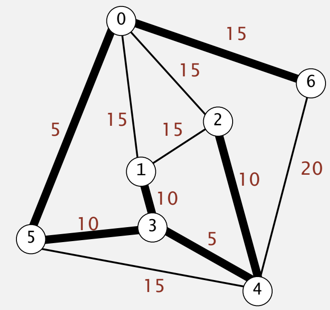


pollEv.com/jhug text to **37607**

Q: What is the weight of the MST?

A. 45	[540123]	D. 60	[520105]
B. 50	[540124]	E. 65	[370101]
C. 55	[520104]		

MST



pollEv.com/jhug text to **37607**

Q: What is the weight of the MST?

C. 55

http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

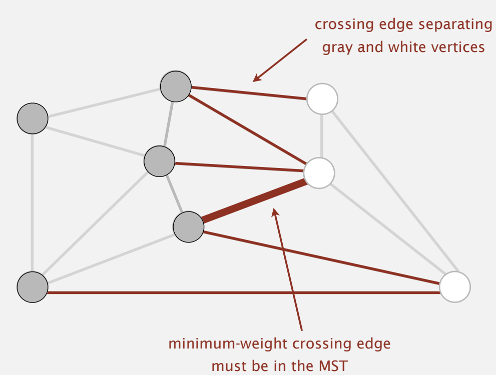
- What
- **Why Kruskal and Prim work**
- How - Kruskal's (data structures)
- How - Prim's (data structures)
- context

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.

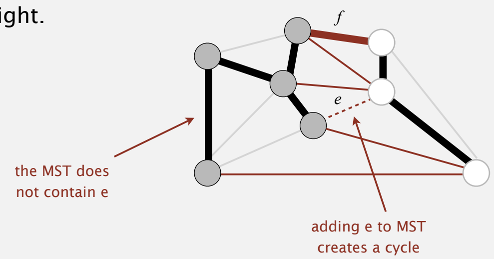


Cut property: correctness proof

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.

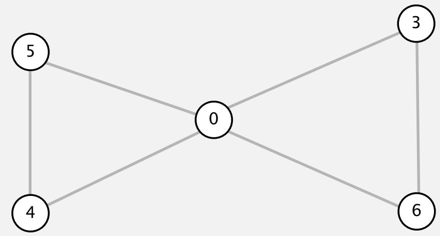
Cut property. Given any cut, the crossing edge of min weight is in the MST.
Pf. Suppose min-weight crossing edge e is not in the MST.

- Adding e to the MST creates a cycle.
- Some other edge f in cycle 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 is lower weight.
- Contradiction. ▪



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.



pollEv.com/jhug text to **37607**

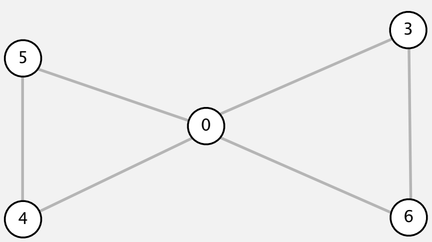
Q: How many distinct cuts are there for the graph above?

A. 7	[229703]	D. 16	[229801]
B. 14	[229704]	E. 30	[229802]
C. 15	[229705]	F. 32	[229803]

Extra: How does the number of distinct cuts grow with V for a general graph?

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.

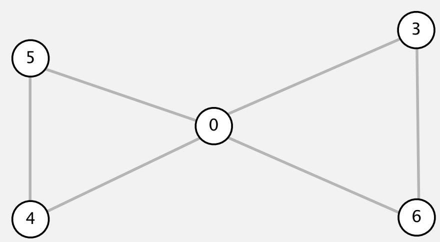


Q: How many distinct cuts are there for the graph above? C. 15

Choice of cut is basically a 5 bit binary number: 32 total choices.
Two of these involve an empty set. Total -> 30.
Half are redundant (e.g. 00100 is the same thing as 11011). Total -> 15.

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.



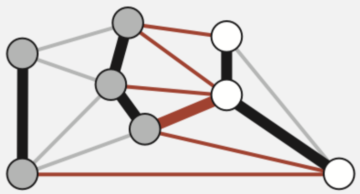
Q: How many distinct cuts are there for the graph above? C. 15

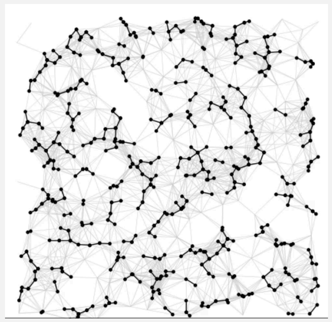
Extra: How does the number of distinct cuts grow with V for a general graph?
$2^{V-1}-1$

226 MST algorithms

Fundamental Idea

- Our algorithms grow an MSSapling until it becomes a full MST.
- The MSSapling starts as V disjoint components.
- Each step of the algorithm connects two MSSapling components.
  - Given 2 **cuts**, always connect by the smallest connecting edge.
  - This smallest edge belongs to MST by cut property.
  - Each connection reduces number of components by 1.
- Once the MSSapling has 1 component, it is the MST.



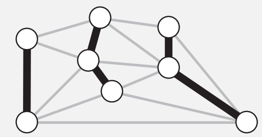


Greedy MST algorithm: correctness proof

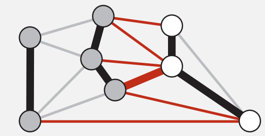
Proposition. Once the MSSapling has 1 component, it is the MST.

Pf.

- Any edge in the MSSapling is in the MST (via cut property).
- Fewer than $V-1$ black edges $\Rightarrow$ There is more than one component.



fewer than V-1 edges colored black



a cut with no black crossing edges

http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- *Why Kruskal and Prim work*
- Kruskal Implementation
- Prim Implementation
- Harder Problems

226 MST algorithms

Fundamental Idea

- Our algorithms grow an MSSapling until it becomes a full MST.
- The MSSapling starts as V disjoint components.
- Each step of the algorithm connects two MSSapling components.
  - Given 2 **cuts**, always connect by the smallest connecting edge.
  - This smallest edge belongs to MST by cut property.
  - Each connection reduces number of components by 1.
- Once the MSSapling has 1 component, it is the MST.

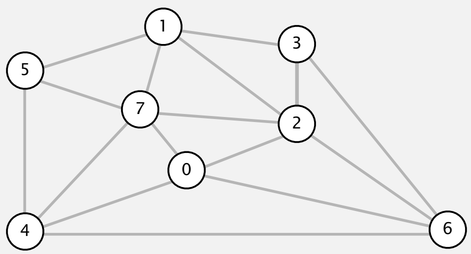
Kruskal's and Prim's

- Specific ways to pick our two MSSapling components.

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).



an edge-weighted graph

graph edges sorted by weight

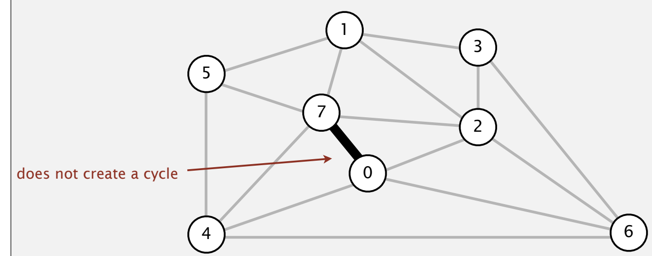
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

in MST → 0-7 0.16

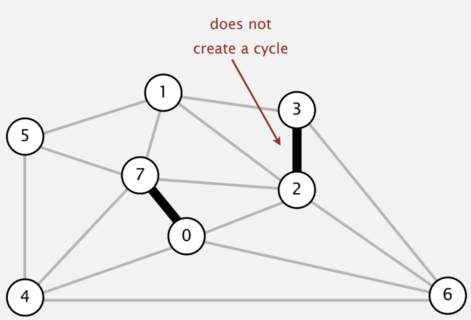


Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

0-7 0.16
in MST → 2-3 0.17

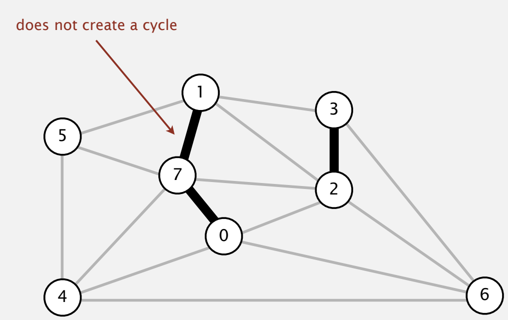


Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

0-7 0.16
2-3 0.17
in MST → 1-7 0.19



pollEv.com/jhug text to **37607**

Q: How many components are there?

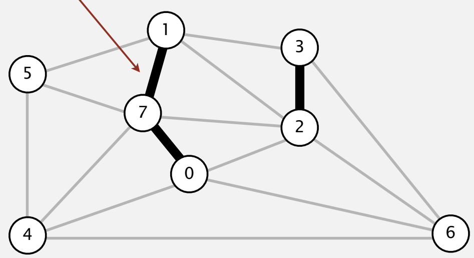
A. 1 [219103]
B. 2 [219104]
C. 3 [602201]
D. 4 [602202]
E. 5 [602302]

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

does not create a cycle



	0-7	0.16
	2-3	0.17
in MST →	1-7	0.19

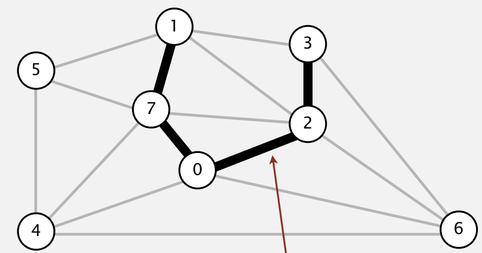
Q: How many components are there?

5

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle (cycle equivalent to having a black crossing edge).



does not create a cycle

	0-7	0.16
	2-3	0.17
	1-7	0.19
in MST →	0-2	0.26

http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- **Kruskal Implementation**
- Prim Implementation
- Harder Problems

Kruskal's algorithm implementation

Kruskal's algorithm

- Given a collection of all the edges in a graph:
  - Take out the minimum edge.
  - Add this edge to the MST as long as no cycle is created.

Challenges.

- What is the smallest weight edge that has not been considered?
- Would adding edge v–w to tree T create a cycle?

In Groups of 3.

- Choose appropriate data structures and algorithms to solve these two subproblems.
- Extra task: How much time does your scheme take to perform each task above? To build the entire MST?

```
private Queue<Edge> mst;
```

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

Debrief - which data structures should we use?

Challenges.

- What is the smallest weight edge that has not been considered?
  - MinPQ<Edge> - compared by weight
  - Edge[] - sorted (comparing by weight)
- Would adding edge v–w to tree T create a cycle?
  - [array that tracks connected components], a.k.a. Union find
  - DFS based graph search every time [very slow]
  - DYNAMIC CONNECTIVITY - UF is fast, DFS is slow

- Calls which interact with edges:
  - int v = e.either();
  - int w = e.other(v);
  - mst.enqueue(e);

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

```
private Queue<Edge> mst;
```


Kruskal's algorithm: Java implementation – live coding answer.

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

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

     for (Edge e : G.edges())
        pq.insert(e);

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

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


Kruskal's algorithm: Java implementation – (book implementation)

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

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

      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: Java implementation – (book implementation)


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

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

      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)

operation	frequency	time per op
build pq	1	E lg E could be E
delete-min	E	lg E
union	V	log* V
connected	E	log* V

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 log E
delete-min	E	log E
union	V	log* V †
connected	E	log* V †

How do we get time E?

Construct array of edges and pass to MinPQ constructor.

† amortized bound using weighted quick union with path compression

recall: log* V ≤ 5 in this universe

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

http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems

Prim's algorithm

- Starting with vertex 0.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.

Three flavors of Prim's

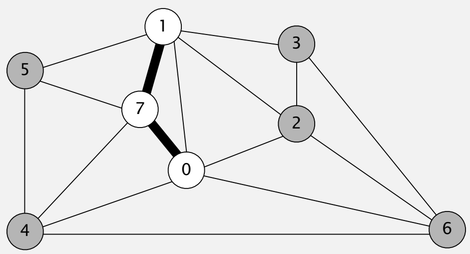
Prim's algorithm

- Intuitive - easy to discover
- Lazy - easy to code version of human
- Eager - optimized version of human

Prim's algorithm implementation

Prim's algorithm

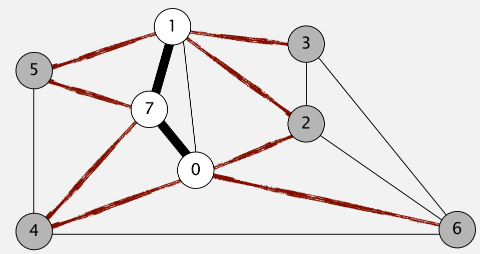
- In Kruskal's, picked MSSaplings by tracking all of the edges in the entire graph and selecting the smallest one.
- In Prim's, what is the most natural thing to track?



Prim's algorithm implementation

Prim's algorithm

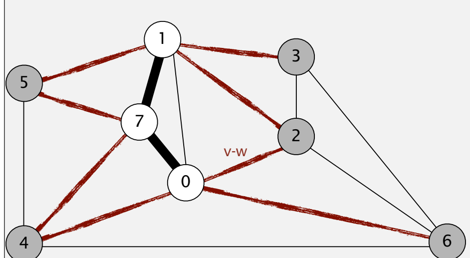
- In Kruskal's, picked MSSaplings by tracking all of the edges in the entire graph and selecting the smallest one.
- In Prim's, what is the most natural thing to track?
  - All outbound edges from core of the MSSapling.

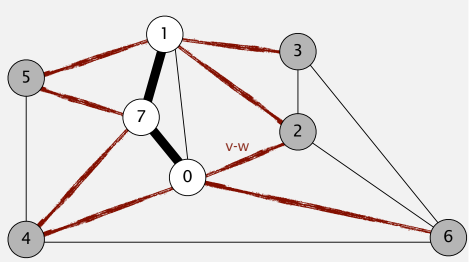


Prim's algorithm implementation

Intuitive Prim's algorithm

- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.

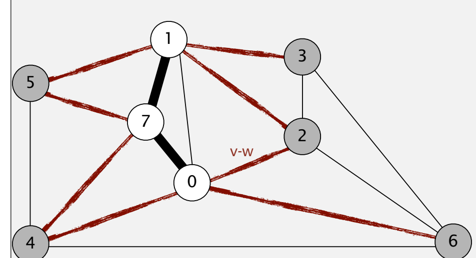


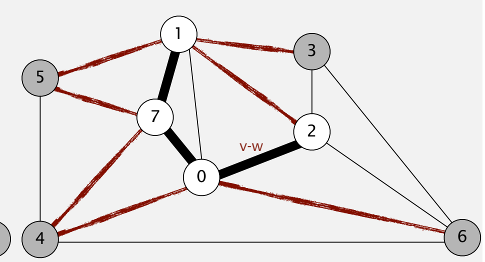


Prim's algorithm implementation

Intuitive Prim's algorithm

- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.

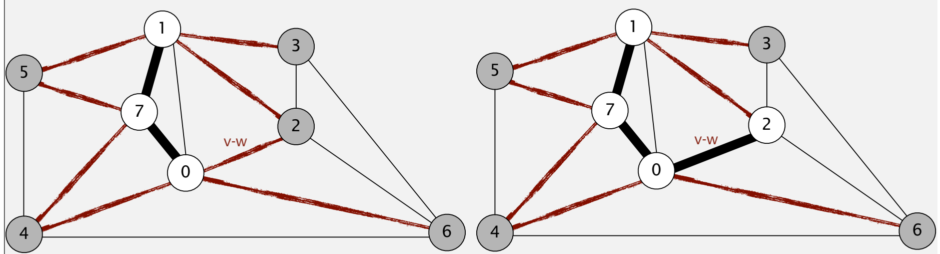




Prim's algorithm implementation

Intuitive Prim's algorithm

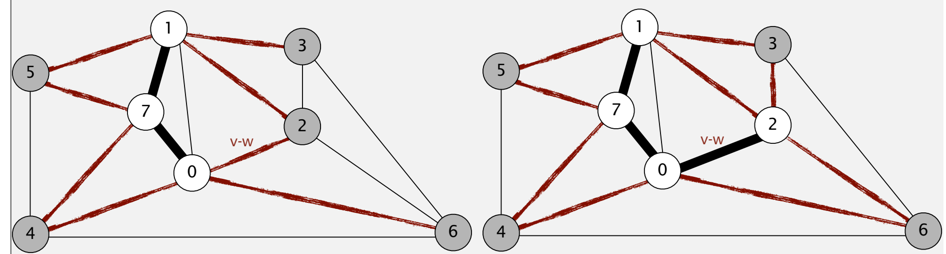
- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.



Prim's algorithm implementation

Intuitive Prim's algorithm

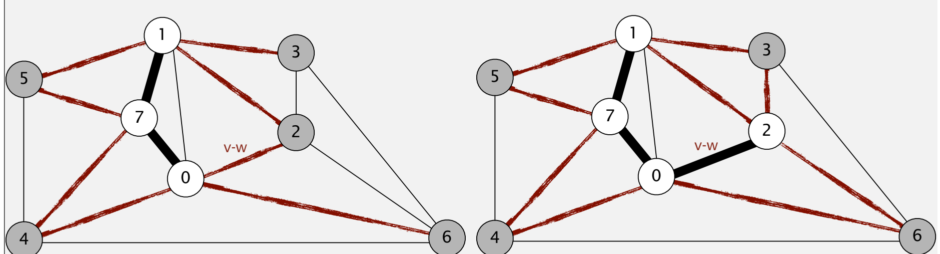
- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.



Prim's algorithm implementation

Intuitive Prim's algorithm

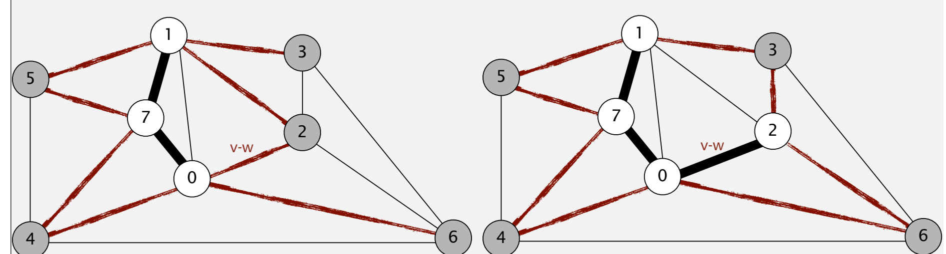
- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
  - Remove from C any edges v-x, where x is also in the core.



Prim's algorithm implementation

Intuitive Prim's algorithm

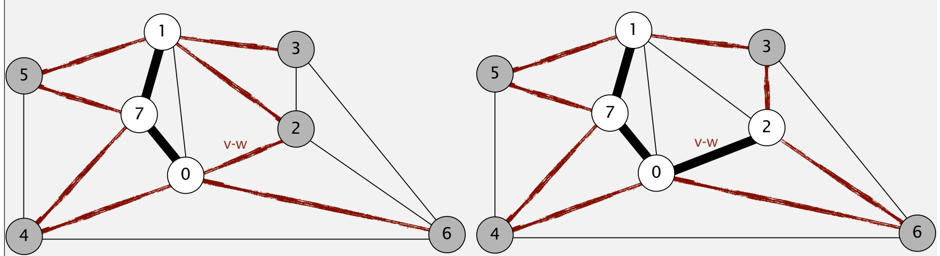
- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
  - Remove from C any edges v-x, where x is also in the core.



Prim's algorithm implementation

Intuitive Prim's algorithm

- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
  - Remove from C any edges v-x, where x is also in the core.

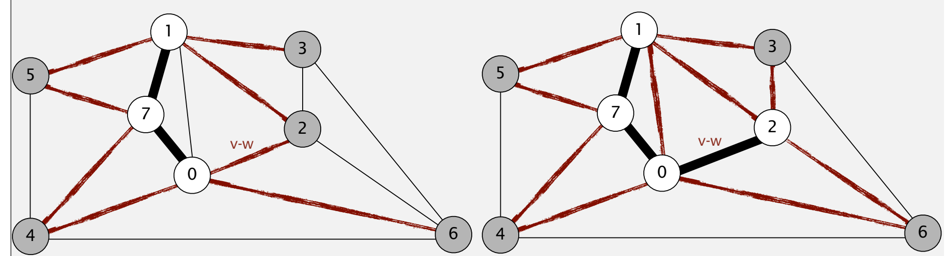


- Turns out this algorithm is a pain to implement (not in textbook).

Prim's algorithm implementation

Lazy Prim's algorithm

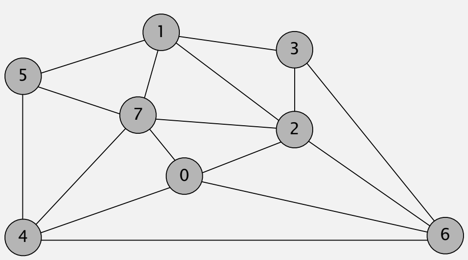
- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling
    - If it doesn't create a cycle, otherwise delete v-w.
  - Add to C any outward pointing edges from w.
  - ~~Remove from C any edges v-x, where x is also in the core.~~



- Much easier to implement.

Prim's algorithm (lazy) 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 implementation

Lazy Prim's algorithm

- Given a collection C of all edges outbound from core:
  - Add C's minimum edge v-w to the MSSapling
    - If it doesn't create a cycle, otherwise delete v-w.
  - Add to C any outward pointing edges from w.
  - ~~Remove from C any edges v-x, where x is also in the core.~~

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: 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

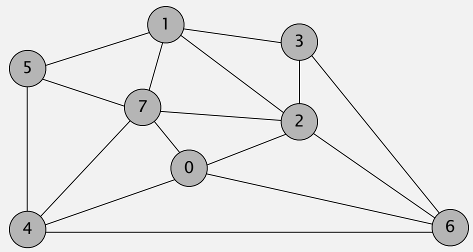
Prim's algorithm demo

Eager Prim's algorithm

- Given a collection C of all ~~edges outbound from~~ vertices adjacent to core:
  - ~~Add C's minimum edge v-w to the MSSapling.~~
  - Remove vertex w that is closest to core, and add edge ?-w.
  - ~~Add to C any outward pointing edges from w.~~
  - ~~Remove from C any edges v-x, where x is also in the core.~~
  - Update distance to each vertex adjacent to core.

Prim's algorithm (eager) 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.

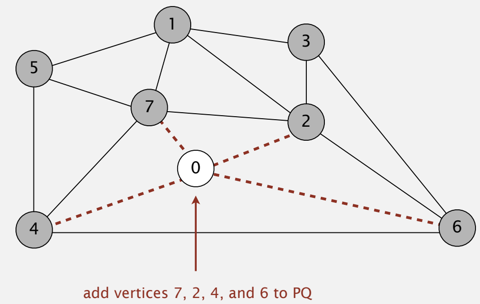


an edge-weighted graph

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 (eager) 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.



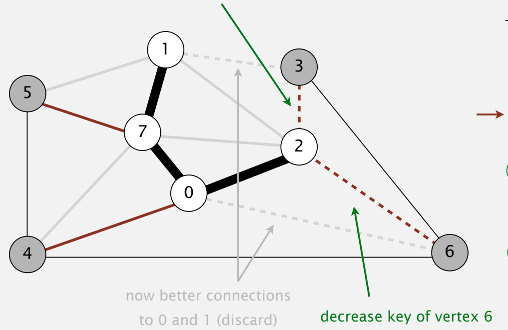
v	edgeTo[]	distTo[]
0	–	–
7	0-7	0.16
2	0-2	0.26
4	0-4	0.38
6	6-0	0.58

vertices on PQ (sorted by weight)

```
IndexMinPQ<Double> pq = new IndexMinPQ<Double>(G.V());
pq.insert(7, 0.16); pq.insert(2, 0.26); ...
```


Prim's algorithm (eager) 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
3	~~1-3~~ 2-3	~~0.29~~ 0.17
5	5-7	0.28
4	0-4	0.38
6	~~6-0~~ 6-2	~~0.58~~ 0.40

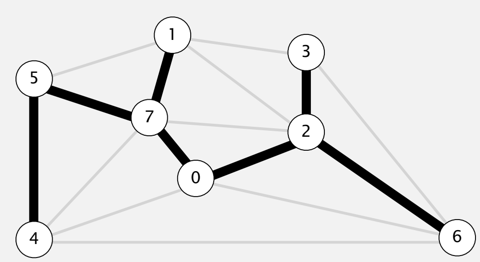
MST edges

0-7 1-7 0-2

```
pq.change(3, 0.17); pq.change(6, 0.4);
```


Prim's algorithm (eager) 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
3	2-3	0.17
5	5-7	0.28
4	4-5	0.35
6	6-2	0.40

MST edges

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

Eager Prim's algorithm: which priority queue?

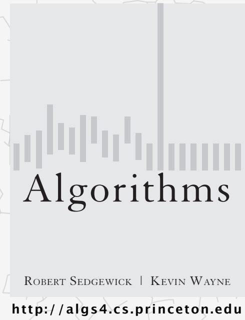
Depends on PQ implementation: V insert, V delete-min, E decrease-key.

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 (Johnson 1975)	$d \log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap (Fredman-Tarjan 1984)	1 †	log V †	1 †	E + V log V

† amortized

Bottom line.

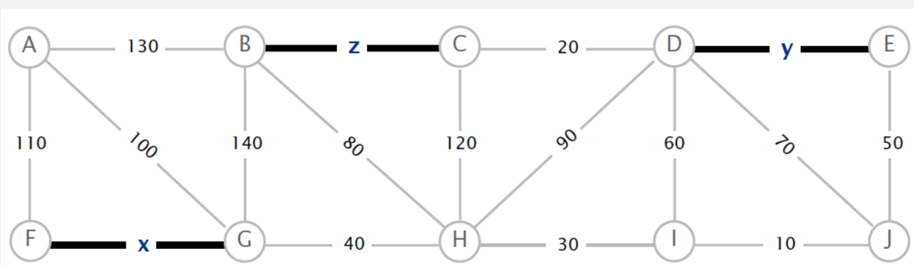
- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.



4.3 Minimum Spanning Trees

B level problems

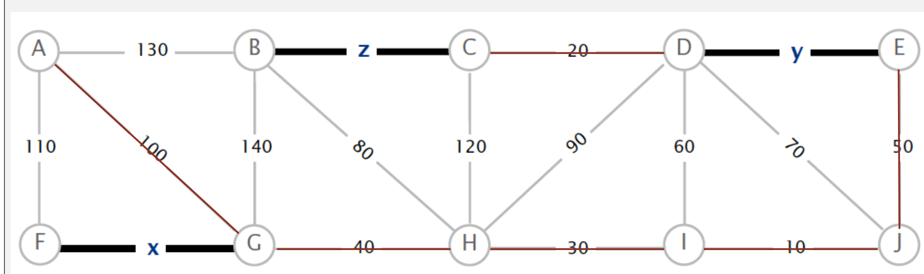
Suppose the that the MST of the graph below contains the edges with weights x, y, and z.



- True or false: The minimum weight edge from every node must be part of the MST.
- List the weights of the **other** edges in the MST:
  _10 ___ ___ ___ ___ ___
- What are the possible values for the weights of x, y, and z?

B level problems

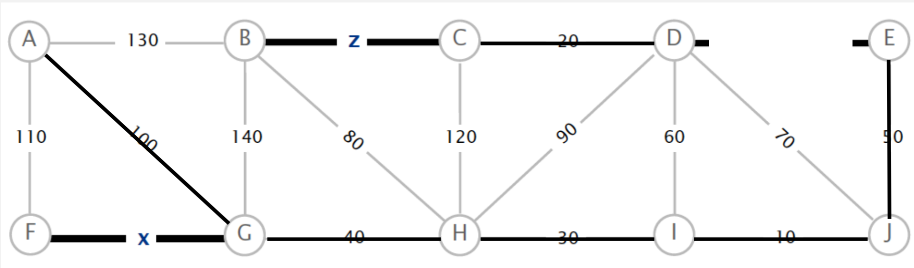
Suppose the that the MST of the graph below contains the edges with weights x, y, and z.



- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the **other** edges in the MST:
  _10 _30 _50 _20 _40 100
- What are the possible values for the weights of x, y, and z?

B level problems

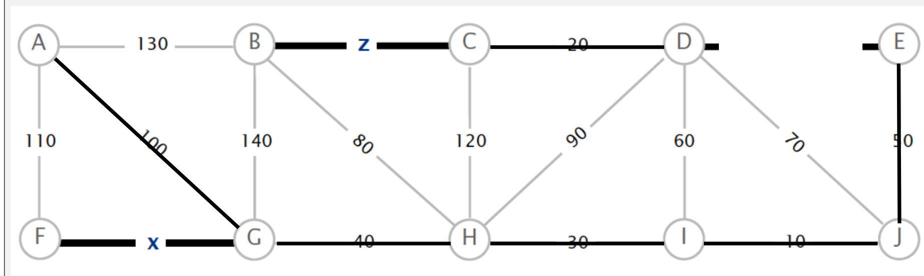
Suppose the that the MST of the graph below contains the edges with weights x, y, and z.



- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the **other** edges in the MST:
  _10 _30 _50 _20 _40 100
- What are the possible values for the weights of x, y, and z?
  - x <= 110, y <= ?

B level problems

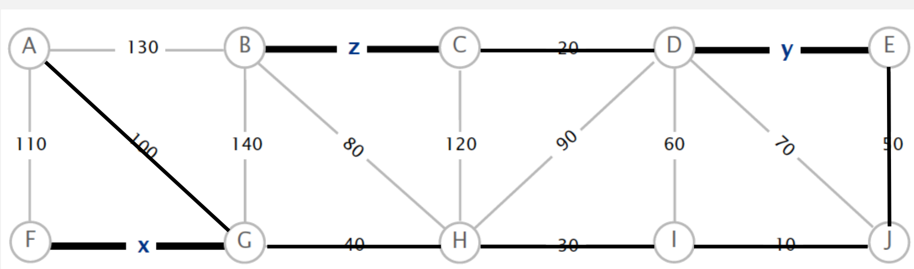
Suppose the that the MST of the graph below contains the edges with weights x, y, and z.



- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the **other** edges in the MST:
  _10 _30 _50 _20 _40 100
- What are the possible values for the weights of x, y, and z?
  - x <= 110, y <= 60,

B level problems

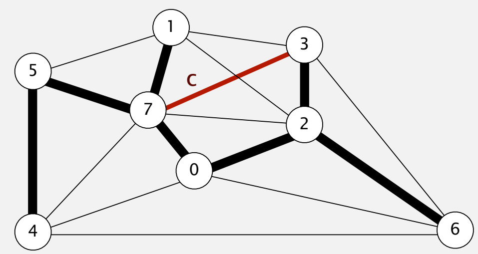
Suppose the that the MST of the graph below contains the edges with weights x, y, and z.



- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the **other** edges in the MST:
  _10 _30 _50 _20 _40 100
- What are the possible values for the weights of x, y, and z?
  - x <= 110, y <= 60, z <= 80

A level problems

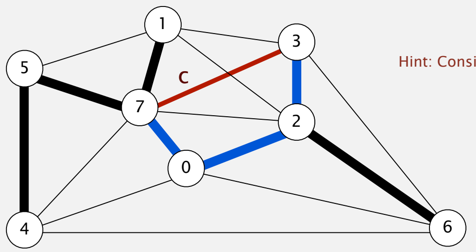
- Suppose you know the MST of G. Now a new edge v-w of weight c is added to G, resulting in a new graph G'. Design a O(V) algorithm to determine if the MST for G is also an MST for G'.



- Bonus: Given a graph G and its MST, if we remove an edge from G that is part of the MST, how do we find the new MST in O(E) time?

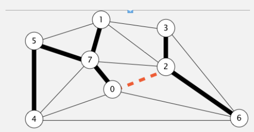
A level problems

- Suppose you know the MST of G. Now a new edge v-w of weight c is added to G, resulting in a new graph G'. Design a O(V) algorithm to determine if the MST for G is also an MST for G'.



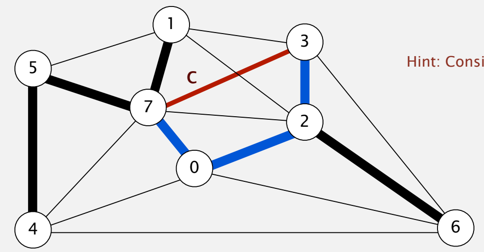
Hint: Consider the blue path.

- Bonus: Given a graph G and its MST, if we remove an edge from G that is part of the MST, how do we find the new MST in O(E) time?



A level problems

- Suppose you know the MST of G. Now a new edge v-w of weight c is added to G, resulting in a new graph G'. Design a O(V) algorithm to determine if the MST for G is also an MST for G'.



Hint: Consider the blue path.

- If any edge on the blue path is longer than c:
  - Replace that edge with c - you get a new MST with shorter distance.
- If every edge on the blue path is shorter than c:
  - Then we know original MST was the best.
- Finding the blue path: Run DFS from one of c's vertices to the other, only taking steps along the MST.

A level problems

- Given a graph G and its MST, if we remove an edge from G that is part of the MST, how do we find the new MST in O(E) time?