Lecture 12
Minimum Spanning Trees

Jianchen Shan
Department of Computer Science
Hofstra University

Lecture Goals

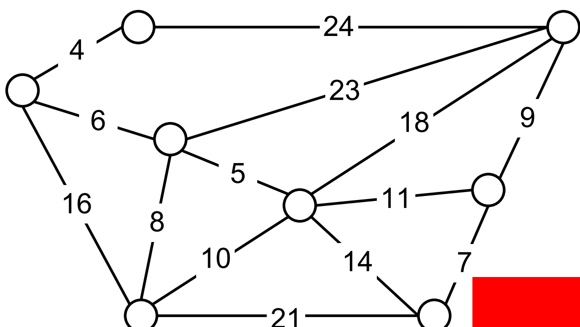
- In this lecture we study the minimum spanning tree problem.
- We begin by considering a generic greedy algorithm for the problem.
- Next, we consider and implement two classic algorithm for the problem—Kruskal's algorithm and Prim's algorithm.
- We conclude with some applications and open problems.

Minimum Spanning Tree (MST)

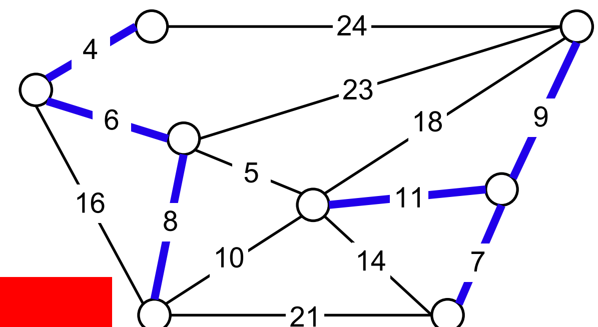
Given. Undirected graph G with positive edge weights (connected).

Def. A spanning tree of G is a subgraph T that is both a tree (connected and acyclic) and spanning (includes all of the vertices).

Goal. Find a min weight spanning tree.

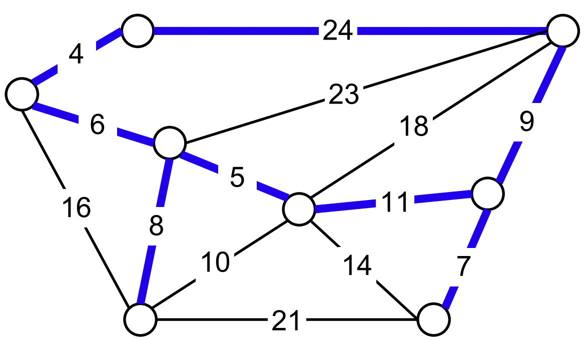


graph G

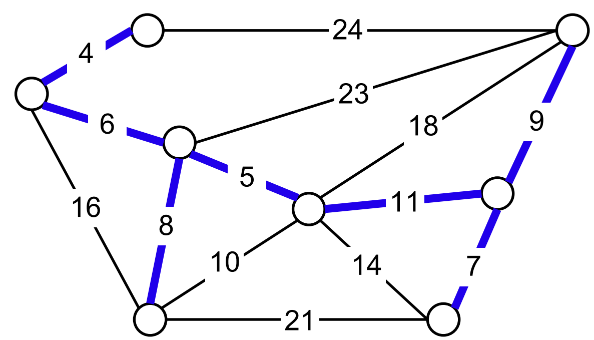


not connected

Brute force.
Try all spanning trees?



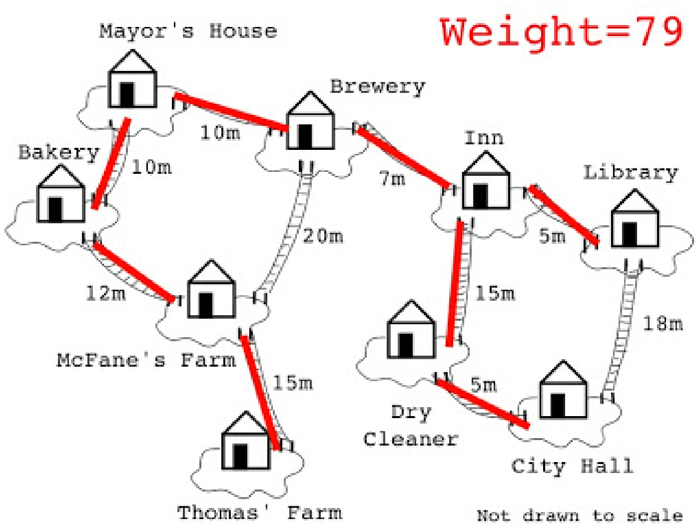
not acyclic



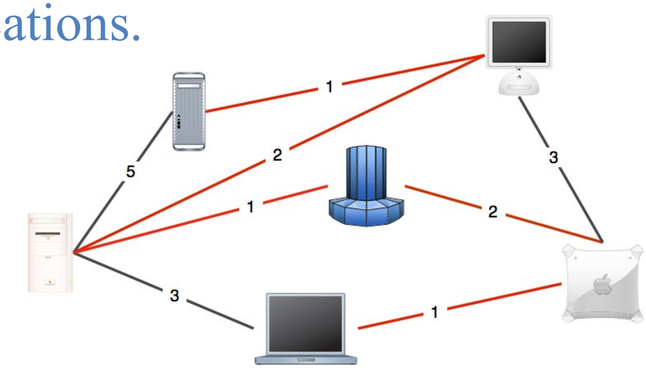
spnning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

MST Applications

- One example would be a telecommunications company trying to lay cable in a new neighborhood. If it is constrained to bury the cable only along certain paths (e.g. roads), then there would be a graph containing the points (e.g. houses) connected by those paths.
- Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights.
- A *MST* would be one with the lowest total cost, representing the least expensive path for laying the cable.



- Network design.
- Cluster analysis.
- Indirect applications.



Simplifying Assumptions and Cut Property

Assumptions. Edge weights are distinct; Graph is connected.

Consequence. MST exists and is unique.

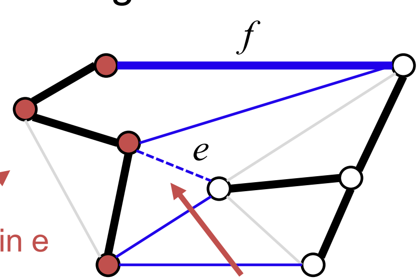
Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.
Def. A crossing edge connects a vertex in one set with a vertex in the other.

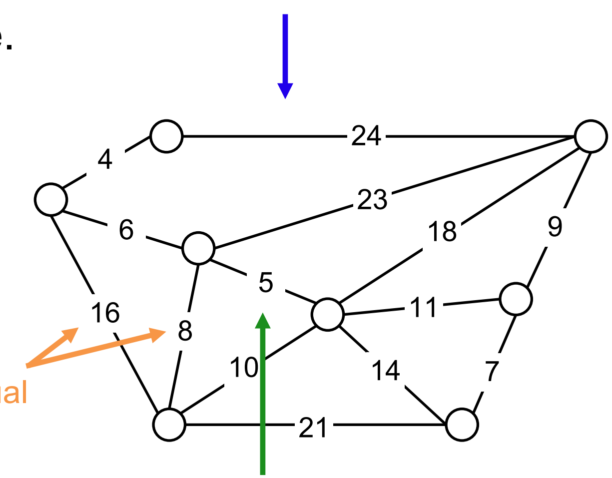
Property. Given any cut, the crossing edge of min weight is in the MST.

Pf. Suppose min-weight crossing edge e is not in the MST.

Contradiction

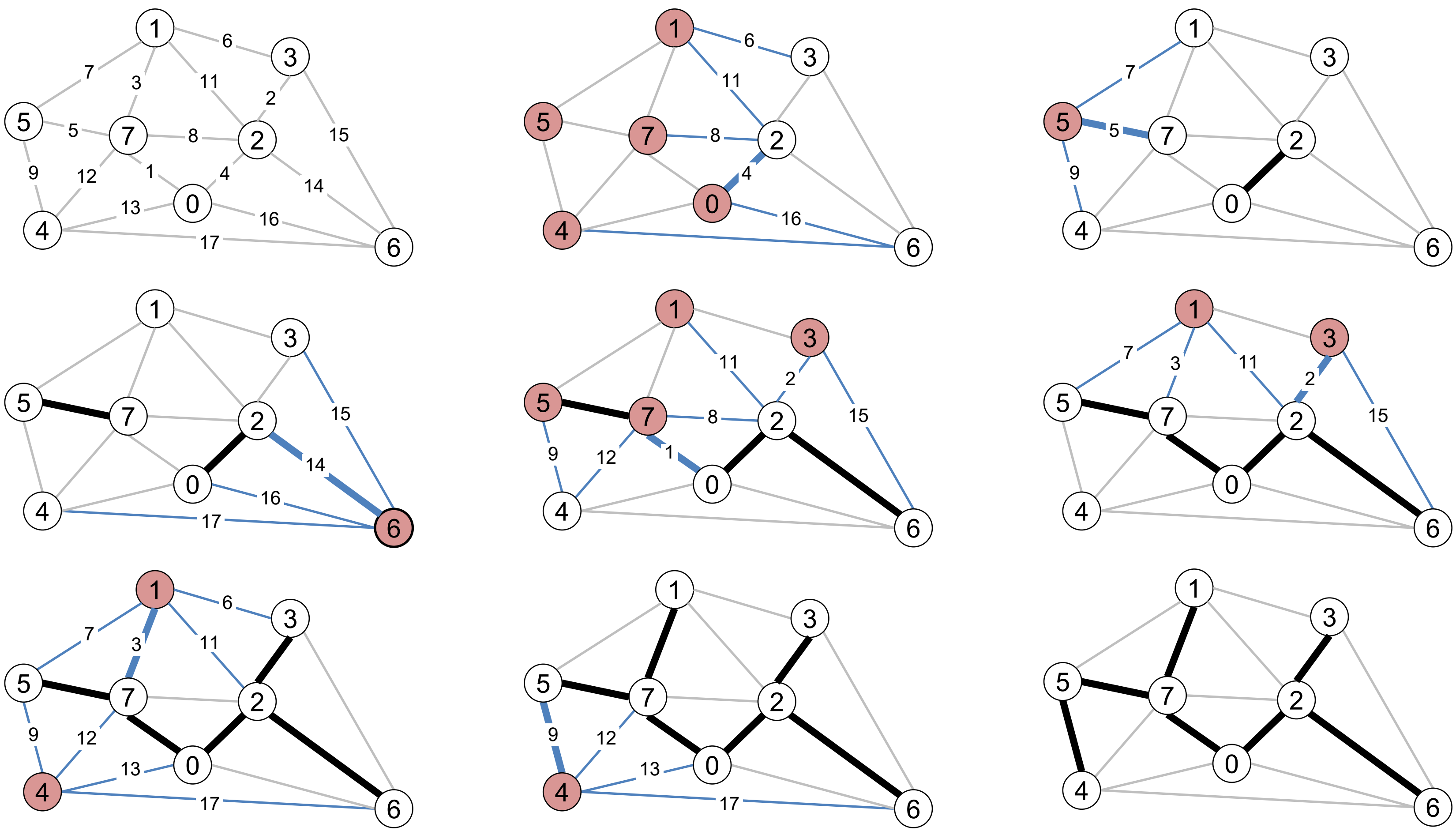
- Adding e to the MST creates a cycle.
- Some other edge f in cycle must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since weight of e is less than the weight of f, that spanning tree is lower weight.





Greedy MST Algorithm

- 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.

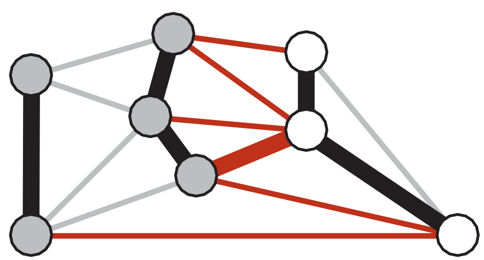


Greedy MST Algorithm: Correctness Proof

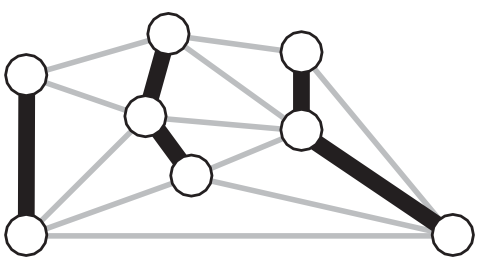
Proposition. The greedy algorithm computes the MST.

Proof.

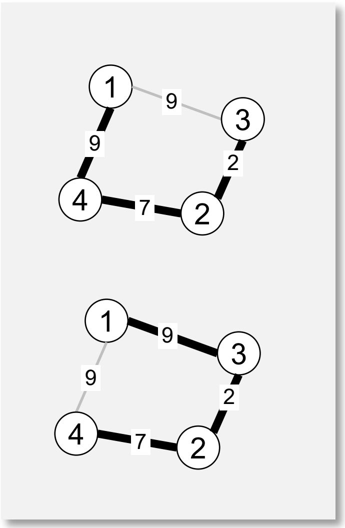
- Any edge colored black is in the MST (via cut property).
- Fewer than V - 1 black edges → cut with no black crossing edges.
  (consider cut whose vertices are one connected component)



a cut with no black crossing edges



fewer than V-1 edges colored black

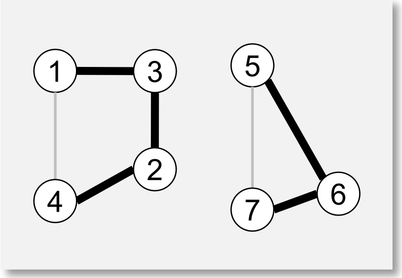


What if edge weights are not all distinct?

Greedy MST algorithm still correct if equal weights are present

What if graph is not connected?

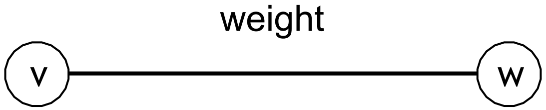
Compute minimum spanning forest = MST of each component.



Weighted Edge API

```
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
         String toString()                // string representation
```

- Class that implements Iterable interface, can be used in the for-each loop to retrieve elements one by one.
- Most collections either implement Iterable interface or have a view that returns one (such as Map's keySet() or values())

```
Iterable<String> myIterable
for (String str : myIterable) { ... }
```

```
public interface Iterable<T>
```

Implementing this interface allows an object to be the target of the "for-each loop" statement.

Edge-Weighted Graph: Adjacency-Lists Implementation

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

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

MST API

```
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
```

Greedy MST Algorithm: Efficient. Choose cut? Find min-weight edge?

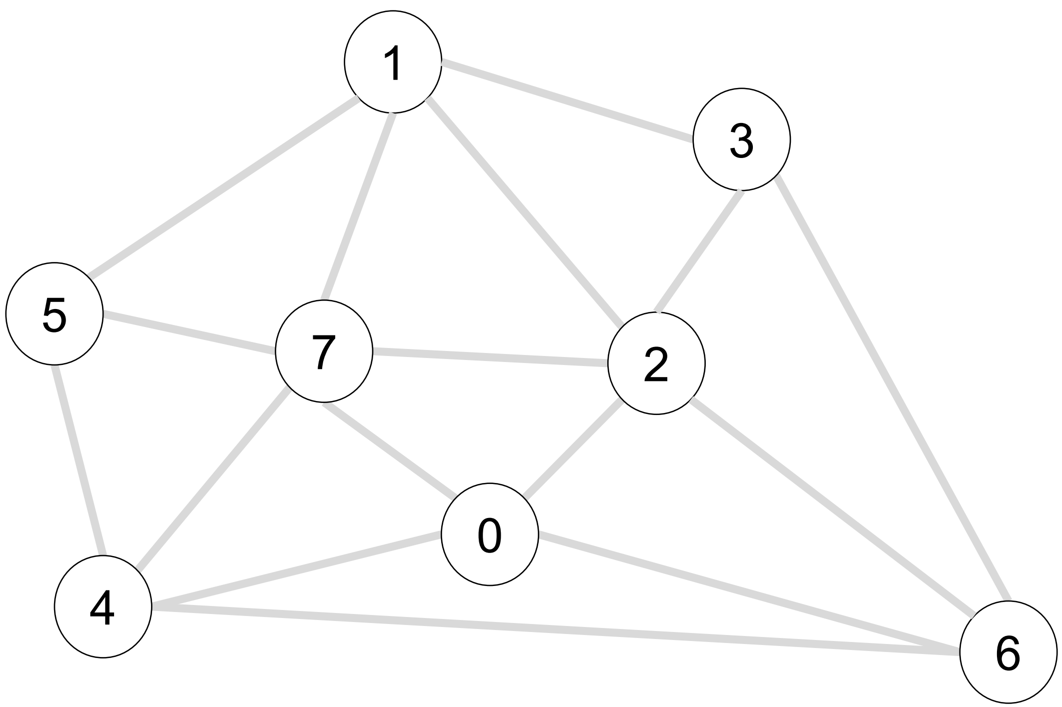
- Kruskal's algorithm.
- Prim's algorithm.

Kruskal's Algorithm

- 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

creates a cycle



an edge-weighted graph

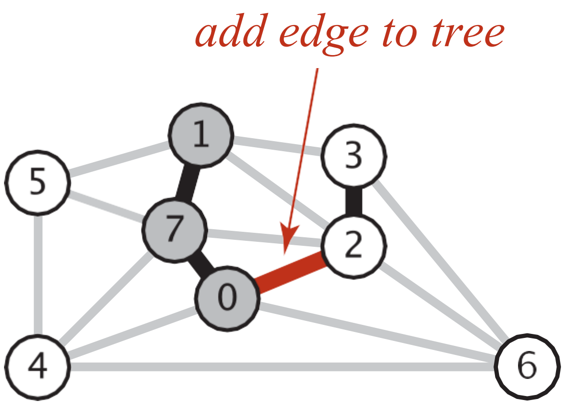
graph edges sorted by weight

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

Kruskal’s Algorithm: Correctness Proof

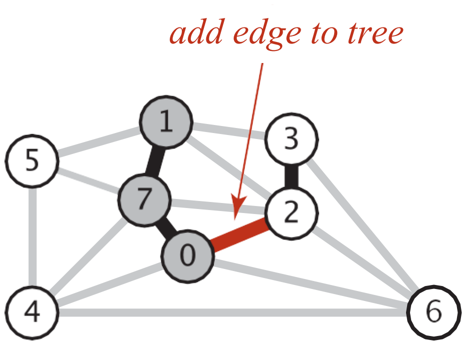
Pf. Kruskal's algorithm is a special case of the greedy MST algorithm.

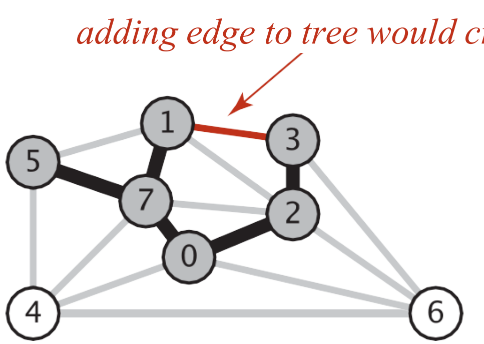
- Suppose Kruskal's algorithm colors the edge e = v–w black.
- Cut = set of vertices connected to v in tree T.
- No crossing edge is black.
- No crossing edge has lower weight.



Kruskal's Algorithm: Implementation Challenge

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



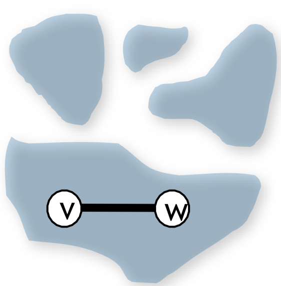


Solution 1. run DFS from v, check if w is reachable (T has at most V – 1 edges) O(V)

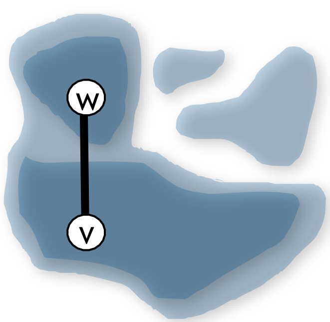
Solution 2. Use the union-find data structure. O(log V)

- Maintain a set for each connected component in T. ← more efficient
- 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.

A *union-find* data structure keeps track of a set of elements partitioned into a number of disjoint subsets. It performs two useful operations:
Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
Union: Join two subsets into a single subset.



Case 1: adding v–w creates a cycle



Case 2: 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 LinkedList<Edge>();

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

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

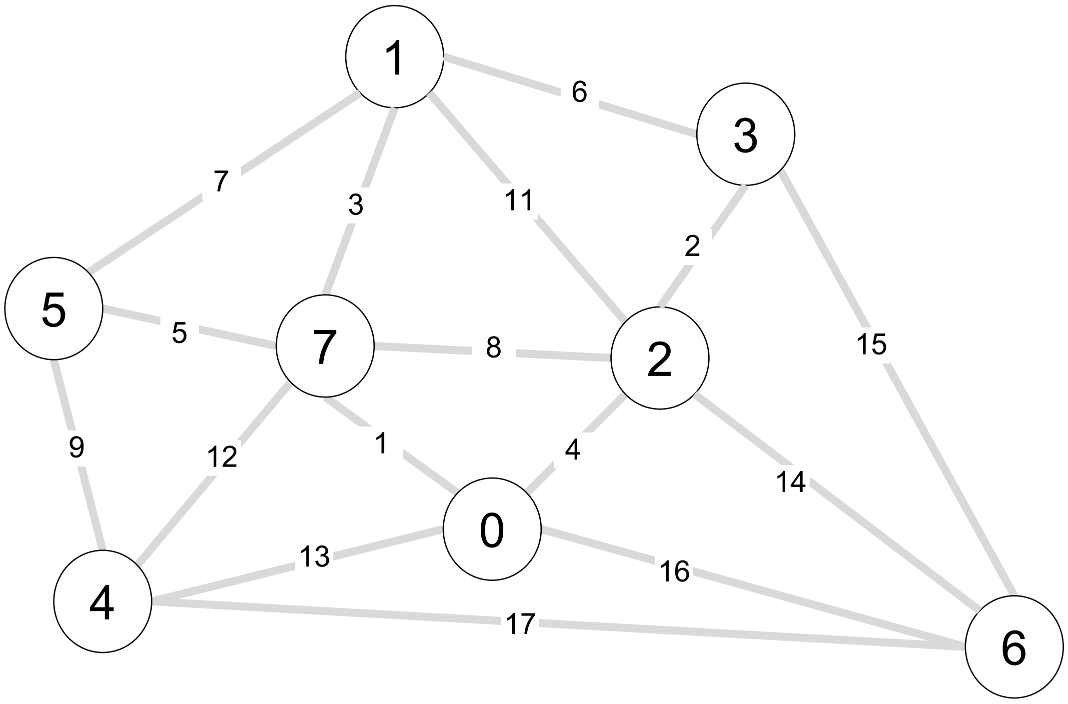
add edge e to MST

$O(E \log E)$

operation	frequency	time per op
sort	1	E log E
union	V - 1	log V
find	2 E	log V

Prim's Algorithm

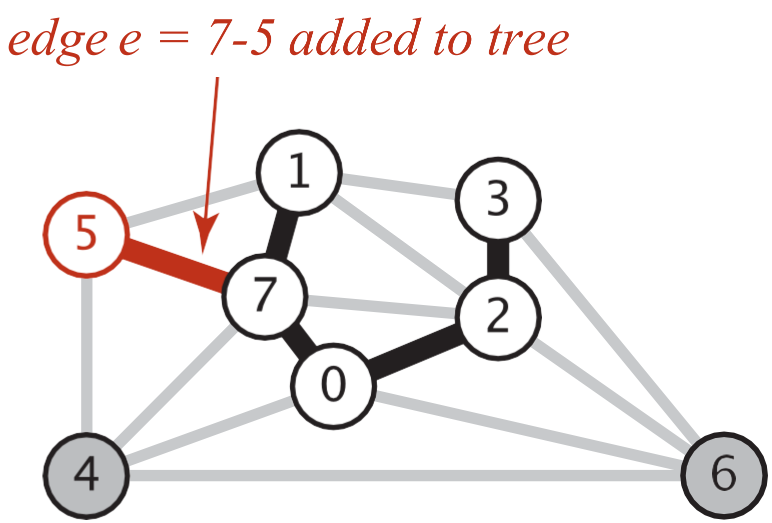
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until V – 1 edges.



Prim’s Algorithm: Correctness Proof

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

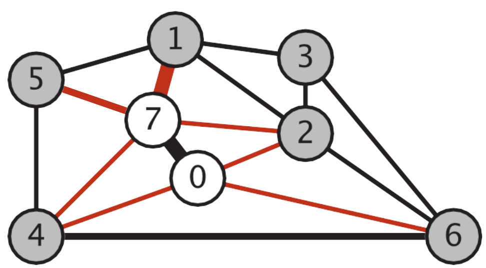
- Suppose edge e = min weight edge connecting a vertex on the tree to a vertex not on the tree.
- Cut = set of vertices connected on tree.
- No crossing edge is black.
- No crossing edge has lower weight.



Prim's Algorithm: Implementation Challenge

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

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

Solution 1. try all edges **O(E)**

A *Priority Queue* is an extension of queue with following properties.
1) Every item has a *priority* associated with it.
2) An element with high priority is *dequeued* before an element with low priority.

Solution 2. 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.
- 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

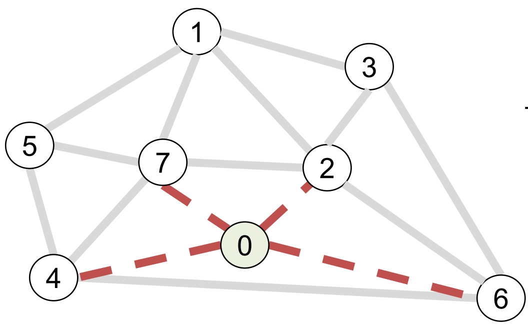
← more efficient

O(log E)

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.

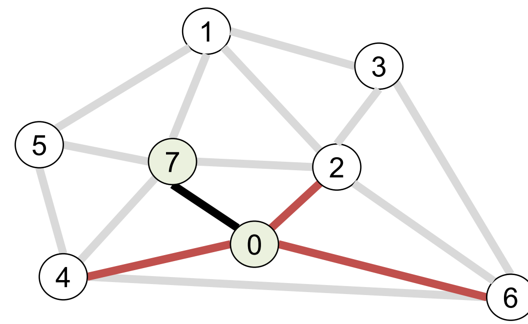
0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16



edges on PQ
(sorted by weight)

*	0 - 7	1
*	0 - 2	4
*	0 - 4	12
*	0 - 6	15

add to PQ all edges incident to 0

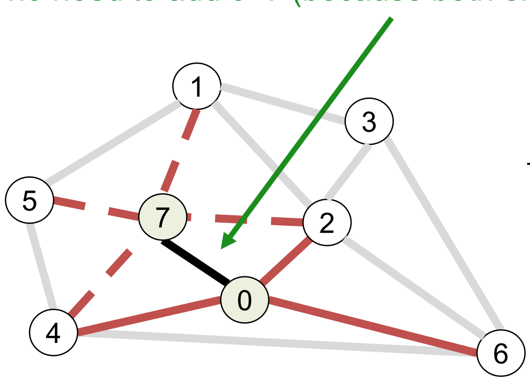


edges on PQ
(sorted by weight)

0 - 2	4
0 - 4	12
0 - 6	15

delete 0-7 and add to MST

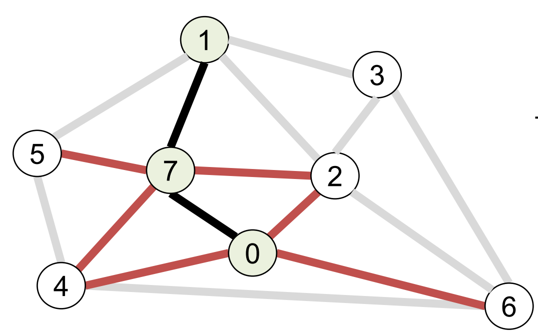
no need to add 0–7 (because both endpoints are in T)



edges on PQ
(sorted by weight)

*	1 - 7	3
	0 - 2	4
*	5 - 7	5
*	2 - 7	8
*	4 - 7	11
	0 - 4	12
	0 - 6	15

add to PQ all edges incident to 7



edges on PQ
(sorted by weight)

0 - 2	4
5 - 7	5
2 - 7	8
4 - 7	11
0 - 4	12
0 - 6	15

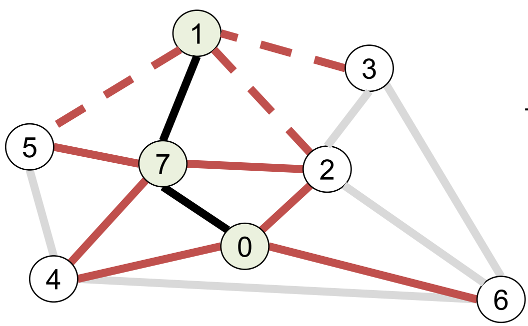
delete 1-7 and add to MST

MST edges:

0 - 7	1
1 - 7	3

Prim's Algorithm: Lazy Implementation (Contd.)

- 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.

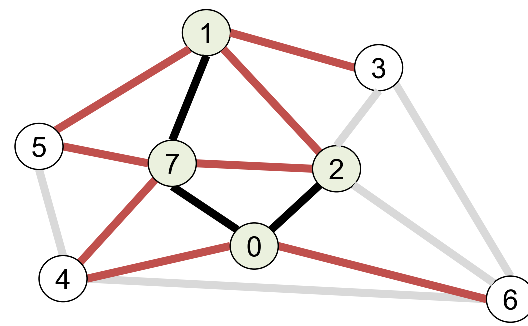


add to PQ all edges incident to 1

1-2 and 2-7 become obsolete
(lazy implementation leaves on PQ)

edges on PQ
(sorted by weight)

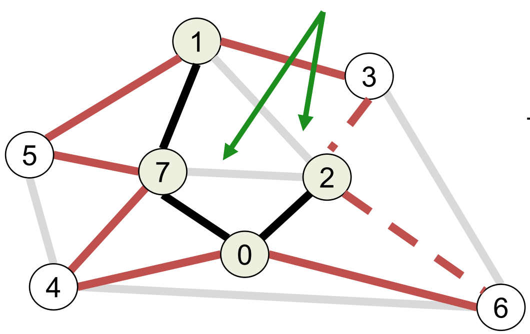
	0 - 2	4
	5 - 7	5
*	1 - 3	6
*	1 - 5	7
	2 - 7	8
*	1 - 2	10
	4 - 7	11
	0 - 4	12
	0 - 6	15



delete 0-2 and add to MST

edges on PQ
(sorted by weight)

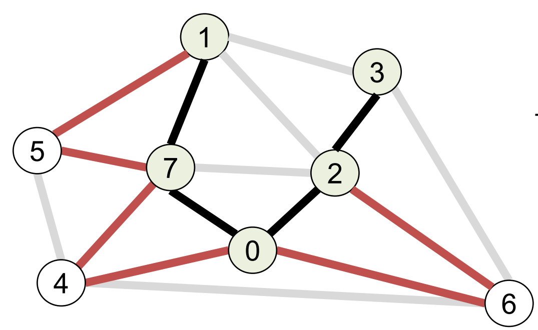
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
1 - 2	10
4 - 7	11
0 - 4	12
0 - 6	15



add to PQ all edges incident to 2

edges on PQ
(sorted by weight)

*	2 - 3	2
	5 - 7	5
	1 - 3	6
	1 - 5	7
	2 - 7	8
	1 - 2	10
	4 - 7	11
	0 - 4	12
*	2 - 6	13
	0 - 6	15



delete 2-3 and add to MST

edges on PQ
(sorted by weight)

5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
0 - 6	15

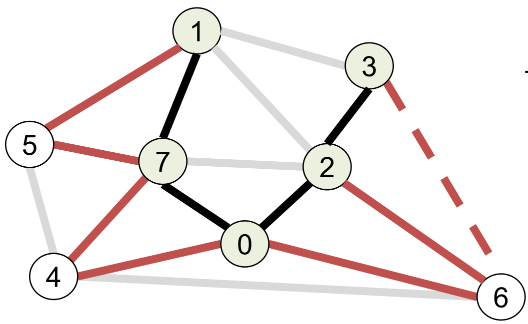
0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

MST edges:

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

Prim's Algorithm: Lazy Implementation (Contd.)

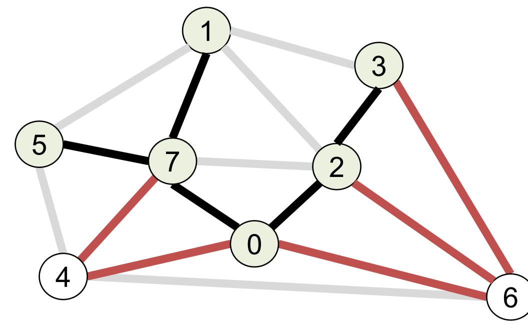
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until V – 1 edges.



add to PQ all edges incident to 3

edges on PQ (sorted by weight)

5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
* 3 - 6	14
0 - 6	15

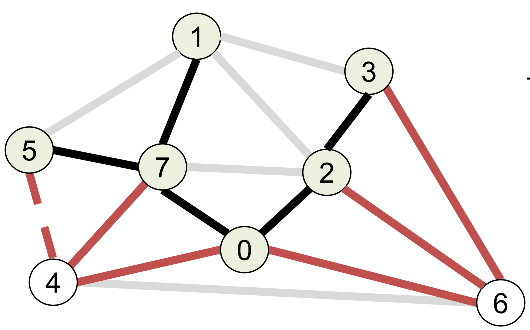


delete 5-7 and add to MST

edges on PQ (sorted by weight)

1 - 3	6
1 - 5	7
2 - 7	8
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15

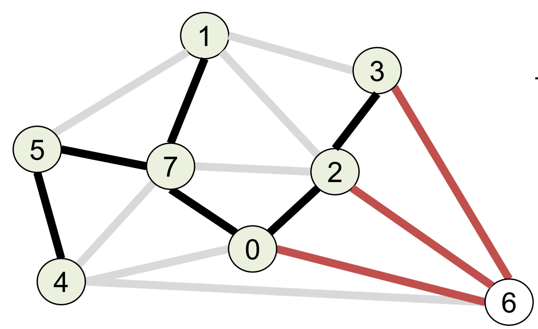
0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16



add to PQ all edges incident to 5

edges on PQ (sorted by weight)

1 - 3	6
1 - 5	7
2 - 7	8
* 4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15



delete 1-3, 1-5, and 2-7
and discard obsolete edge
delete 4-5 and add to MST

edges on PQ (sorted by weight)

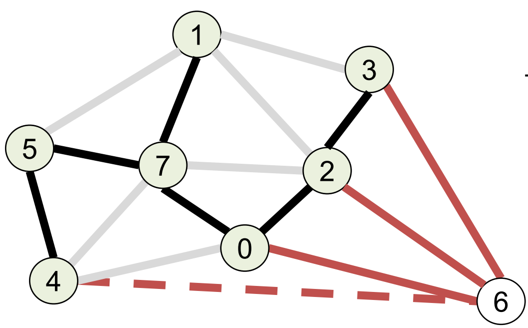
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15

MST edges:

0 - 7	1
1 - 7	3
0 - 2	4
2 - 3	2
5 - 7	5
4 - 5	9

Prim’s Algorithm: Lazy Implementation (Contd.)

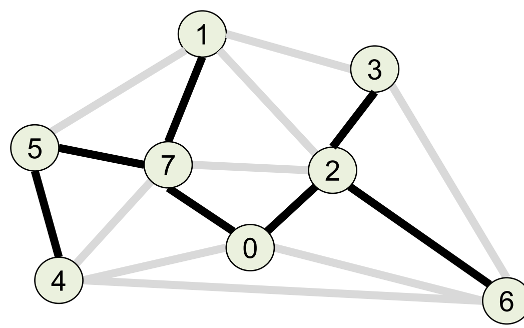
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until V – 1 edges.



add to PQ all edges incident to 4

edges on PQ
(sorted by weight)

	1 - 2	10
	4 - 7	11
	0 - 4	12
	2 - 6	13
	3 - 6	14
	0 - 6	15
*	4 - 6	16



delete 1-2, 4-7, and 0-4
and discard obsolete edge
delete 2-6 and add to MST

edges on PQ
(sorted by weight)

3 - 6	14
0 - 6	15
4 - 6	16

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

MST edges:

0 - 7	1
1 - 7	3
0 - 2	4
2 - 3	2
5 - 7	5
4 - 5	9
2 - 6	13

stop since V-1 edges

Prim's Algorithm: Lazy Implementation in Java

```
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);
        }
    }
    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;  }
```

$O(E \log E)$

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

add v to T

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

implement priority queue

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

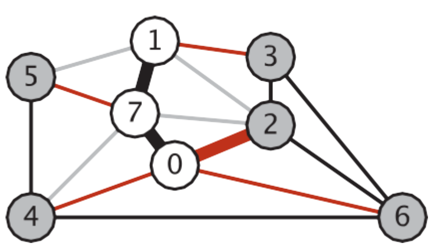
Prim's Algorithm: Eager Implementation

Challenge. Find the 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
  - ✓ if already on PQ, 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

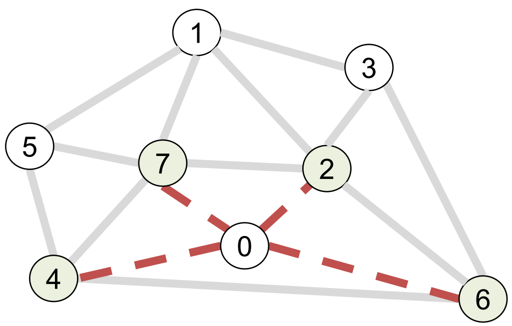
Motivation: the height of the binary-heap PQ would be reduced → O(log V)

Prim's Algorithm: Eager 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.

MST edges:

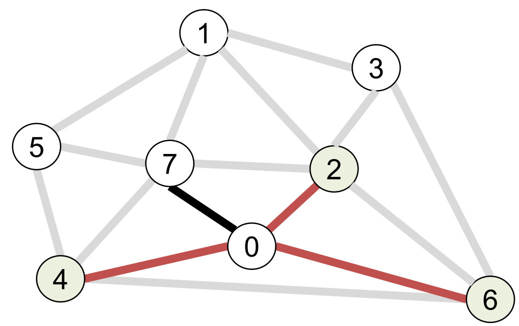
0 - 7	1



add to PQ the vertices incident to 0

vertices on PQ
(sorted by weight)

	v	edgeTo[ ]	distTo[ ]
*	7	0 - 7	1
*	2	0 - 2	4
*	4	0 - 4	12
*	6	0 - 6	15



Delete 7 and add 0-7 to T

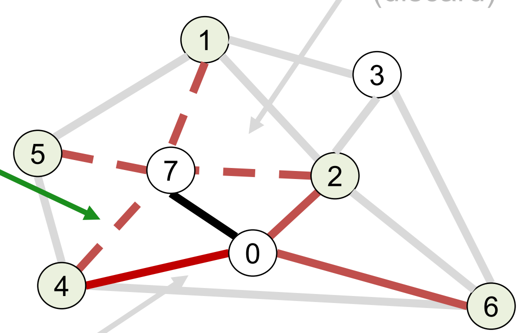
vertices on PQ
(sorted by weight)

v	edgeTo[ ]	distTo[ ]
2	0 - 2	4
4	0 - 4	12
6	0 - 6	15

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

a better connection to 4 (decrease key of 4)

already a better connection to 2 (discard)



not the best connection to 4 (discard)

add to PQ the vertices incident to 7

vertices on PQ
(sorted by weight)

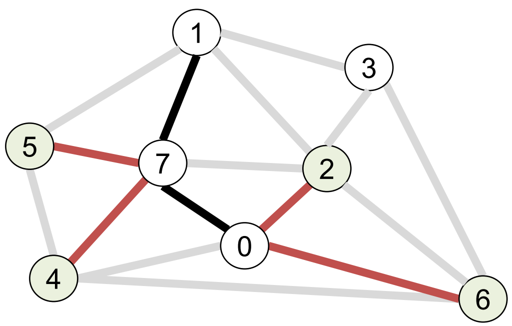
	v	edgeTo[ ]	distTo[ ]
*	1	1 - 7	3
	2	0 - 2	4
*	5	5 - 7	5
	4	~~0 - 4~~ 4 - 7	~~12~~ 11
	6	0 - 6	15

Prim's Algorithm: Eager 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.

MST edges:

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

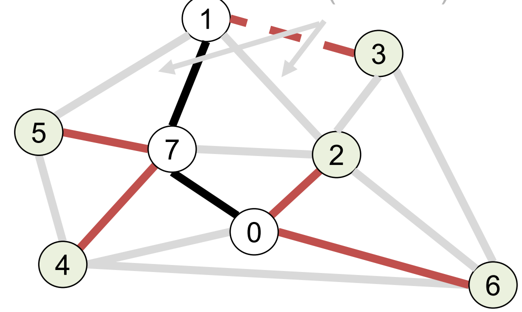


vertices on PQ
(sorted by weight)

v	edgeTo[ ]	distTo[ ]
2	0 - 2	4
5	5 - 7	5
4	4 - 7	11
6	0 - 6	15

Delete 1 and add 1-7 to T

already a better connection to 5 and 2 (discard)

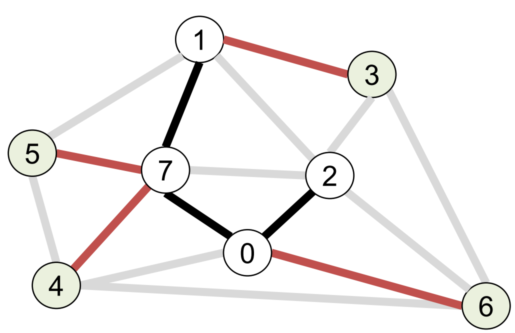


add to PQ the vertices incident to 1

vertices on PQ
(sorted by weight)

	v	edgeTo[ ]	distTo[ ]
	2	0 - 2	4
	5	5 - 7	5
*	3	1 - 3	6
	4	4 - 7	11
	6	0 - 6	15

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16



vertices on PQ
(sorted by weight)

v	edgeTo[ ]	distTo[ ]
5	5 - 7	5
3	1 - 3	6
4	4 - 7	11
6	0 - 6	15

Delete 2 and add 0-2 to T

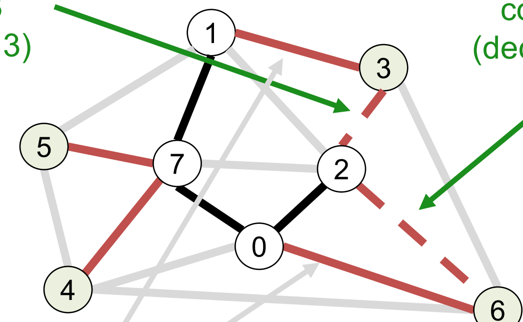
Prim's Algorithm: Eager Implementation

- 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.

MST edges:

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

a better connection to 3 (decrease key of 3)



a better connection to 6 (decrease key of 6)

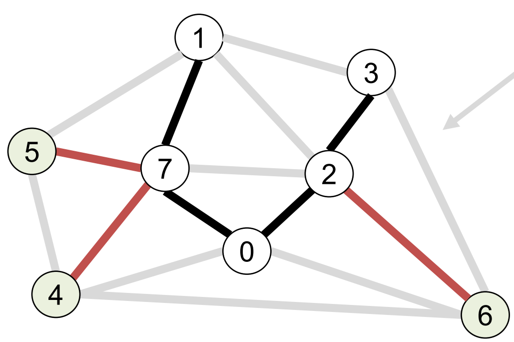
add to PQ the vertices incident to 2

no longer the best connection to 3 and 6 (discard)

vertices on PQ (sorted by weight)

v	edgeTo[ ]	distTo[ ]
5	5 - 7	5
3	~~1 - 3~~ 2 - 3	~~6~~ 2
4	4 - 7	11
6	~~0 - 6~~ 2 - 6	~~15~~ 13

already a better connection to 6 (discard)



Delete 3 and add 2-3 to T

add to PQ the vertices incident to 3

vertices on PQ (sorted by weight)

v	edgeTo[ ]	distTo[ ]
5	5 - 7	5
4	4 - 7	11
6	2 - 6	13

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

Prim's Algorithm: Eager 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.

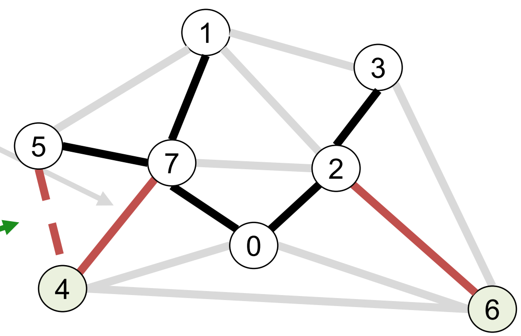
MST edges:

0 - 7	1
1 - 7	3
0 - 2	4
2 - 3	2
5 - 7	5
4 - 5	9
2 - 6	13

0 - 7	1
2 - 3	2
1 - 7	3
0 - 2	4
5 - 7	5
1 - 3	6
1 - 5	7
2 - 7	8
4 - 5	9
1 - 2	10
4 - 7	11
0 - 4	12
2 - 6	13
3 - 6	14
0 - 6	15
4 - 6	16

no longer the best connection to 4 (discard)

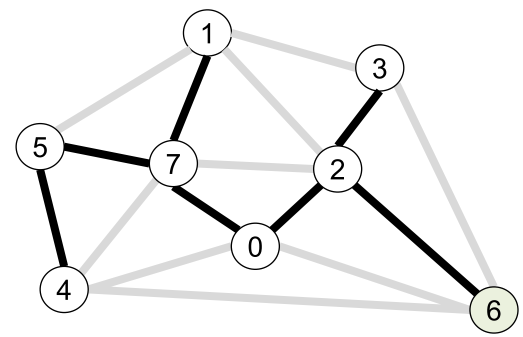
a better connection to 4 (decrease key of 6)



Delete 5 and add 5-7 to T
add to PQ the vertices incident to 5

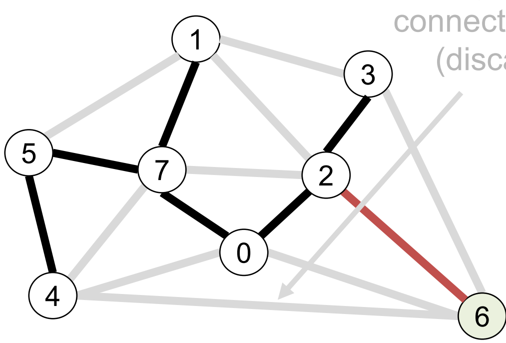
vertices on PQ (sorted by weight)

v	edgeTo[ ]	distTo[ ]
4	~~4 - 7~~ 4 - 5	~~11~~ 9
6	2 - 6	13



Delete 2 and add 2-6 to T

already a better connection to 6 (discard)



Delete 4 and add 4-5 to T
add to PQ the vertices incident to 4

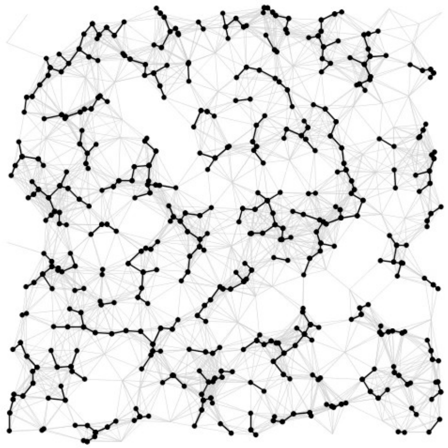
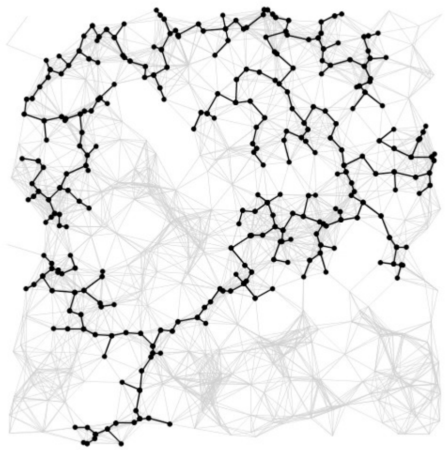
vertices on PQ (sorted by weight)

v	edgeTo[ ]	distTo[ ]
6	2 - 6	13

O(E log V)

operation	frequency	time per op
Insert	E	log V
delete min	E	log V
decrease key	E	log V

Summary

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

https://www.youtube.com/watch?v=vmWSnkBVvQ0

Additional Resources

- Union-Find
  - https://www.geeksforgeeks.org/union-find/
- Priority Queue
  - https://www.geeksforgeeks.org/priority-queue-set-1-introduction/
- Binary Heap
  - https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html