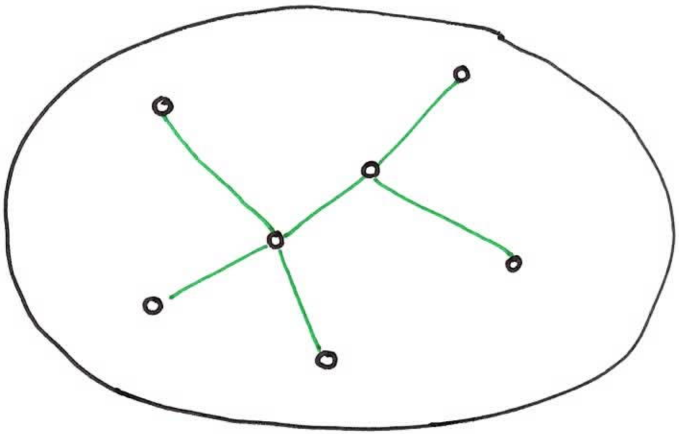
chang@umbc.edu

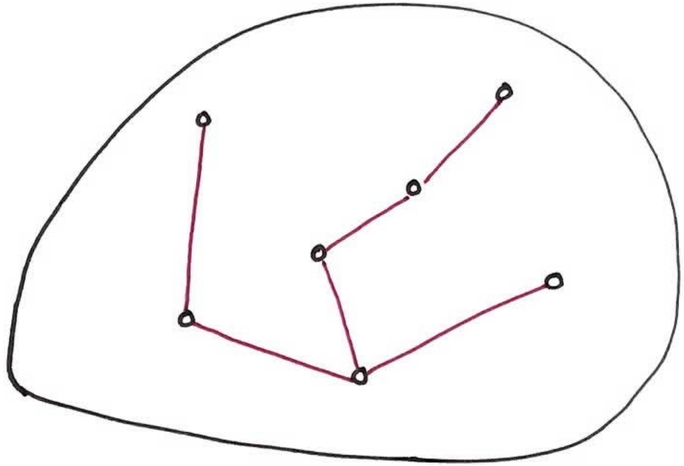
Minimum Spanning Tree = MST

Problem: n sites to be connected in a network.

cost proportional to length of wires.

don't care about network topology.





MST in a graph:

$G = (V, E)$ undirected graph with weights

$w: E \to \mathbb{R}$ ← "length" of the edge

Find an acyclic subset T of E that connects every vertex and minimizes total weight:

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

Note: MST not unique

Tree w/ n vertices has exactly $n-1$ edges

Two Greedy algorithms for MST

Prim's algorithm: grow 1 tree

Kruskal's algorithm: use lowest weight edge, if you can

Greedy $\Rightarrow$ needs proof.

General approach:

$A \leftarrow \emptyset$

Loop $n-1$ times:

- find an edge (u,v) s.t. $A \cup \{(u,v)\}$ is contained in some MST of G. ← safe edge
- $A \leftarrow A \cup \{(u,v)\}$

loop invariant: there exists an MST T of G s.t. $A \subseteq T$

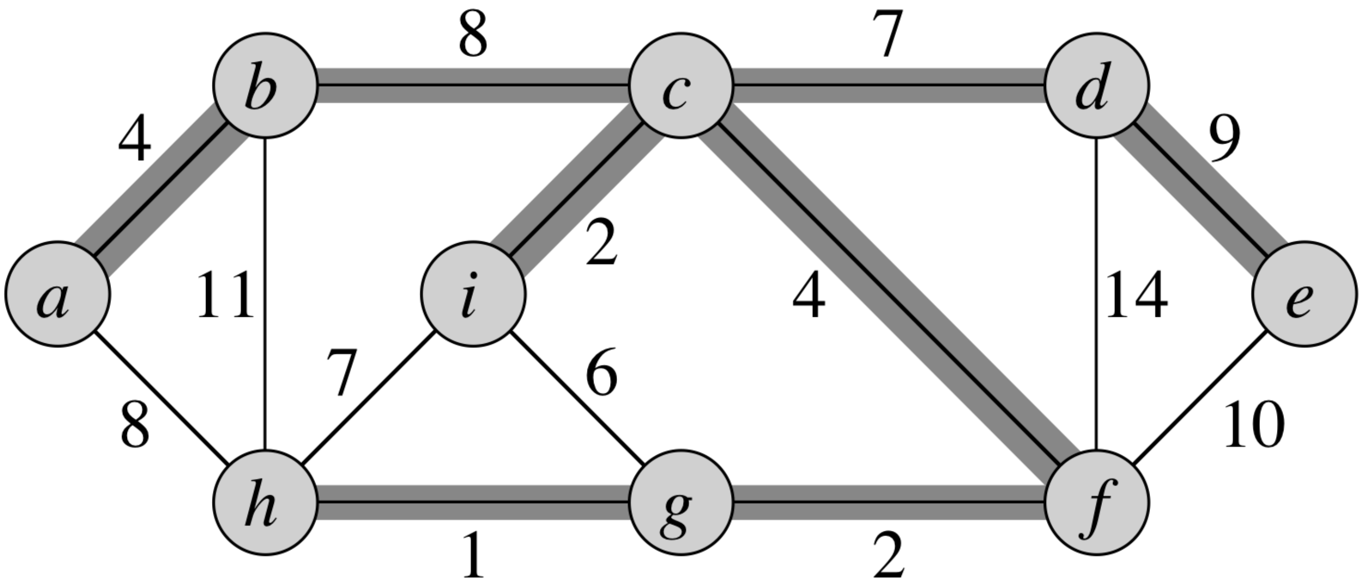
When loop terminates, A has $n-1$ edges. A must be an MST.

How? (find)

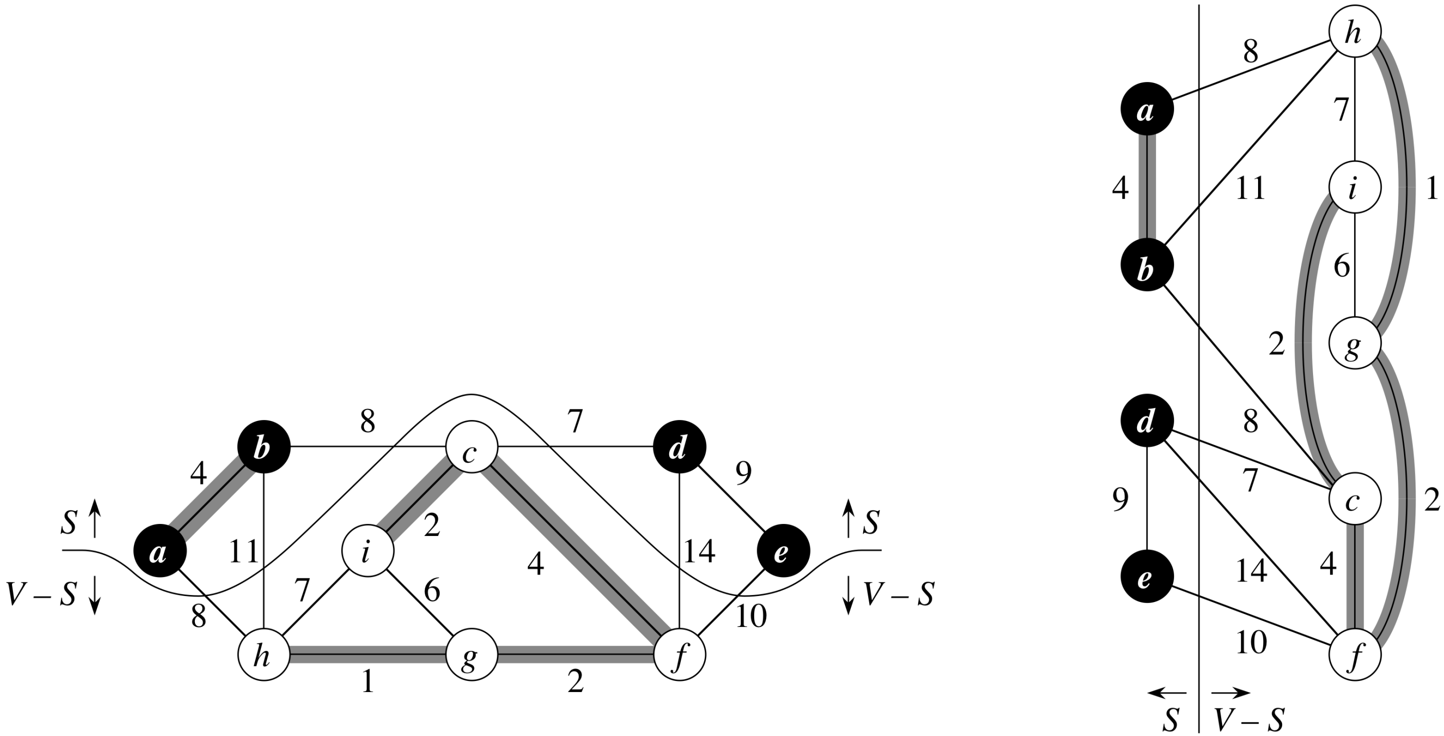
Defn: A cut $(S, V-S)$ in a graph G is a partition of the vertices V into S and $V-S$.

Defn: An edge (u,v) crosses a cut $(S, V-S)$ if $u \in S$ and $v \in V-S$ or vice versa.

Defn: A cut $(S, V-S)$ respects a set of edges A, if no edge in A crosses $(S, V-S)$.



8
7
b
c
d
4
9
2
11
14
a
i
4
e
7
6
8
10
h
g
f
1
2



b
8
c
7
d
4
9
S
S
a
11
i
2
4
14
e
V – S
V – S
7
6
8
10
h
g
f
1
2
h
8
a
7
4
11
i
1
b
6
2
g
d
8
7
c
2
9
e
14
4
10
f
S
V – S

(a) (b)

Theorem If $G=(V,E)$ is a connected, undirected graph with weights and $A \subseteq E$ s.t. $A \subseteq T$ for some MST T.

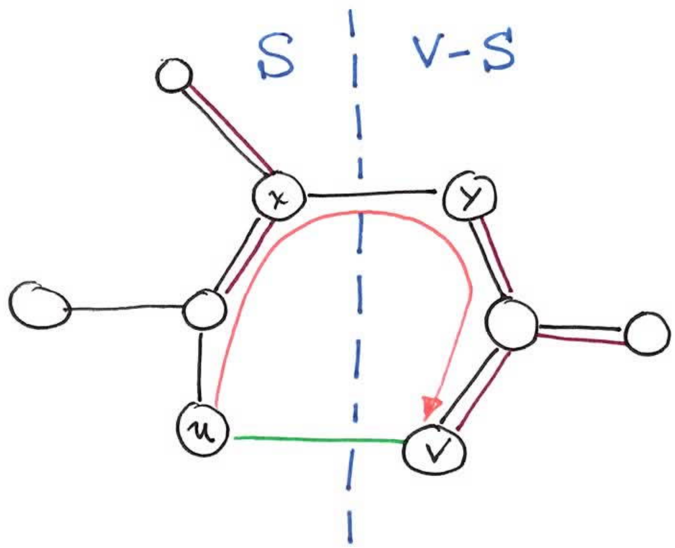
Let $(S, V-S)$ be a cut that respects A.

Let (u,v) be the lowest weight edge that crosses $(S, V-S)$.

light edge

Then, (u,v) is safe for A.

$A \cup \{(u,v)\}$ is still a subset of some MST T in G.



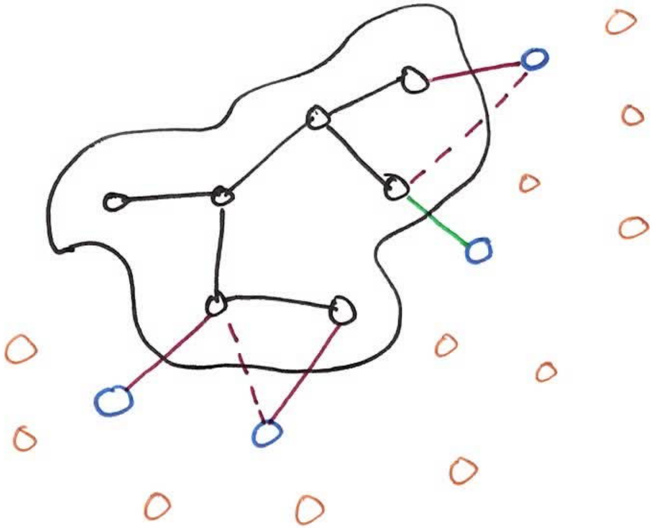
Claim: $T' = (T - \{(x,y)\}) \cup \{(u,v)\}$ is an MST

Subclaims:

1. (x,y) exists
2. $w(x,y) \geq w(u,v)$
3. T' is a tree (spanning, too)
4. $w(T') \leq w(T)$

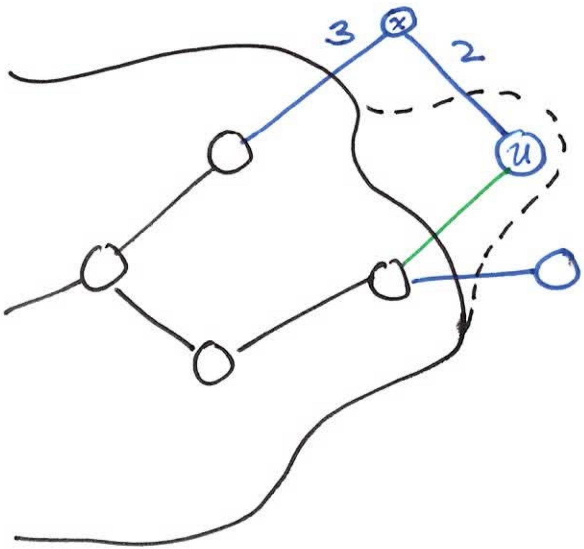
Note: remove an edge from a tree breaks the tree into exactly 2 pieces.

Prim's algorithm: Cut around current tree A



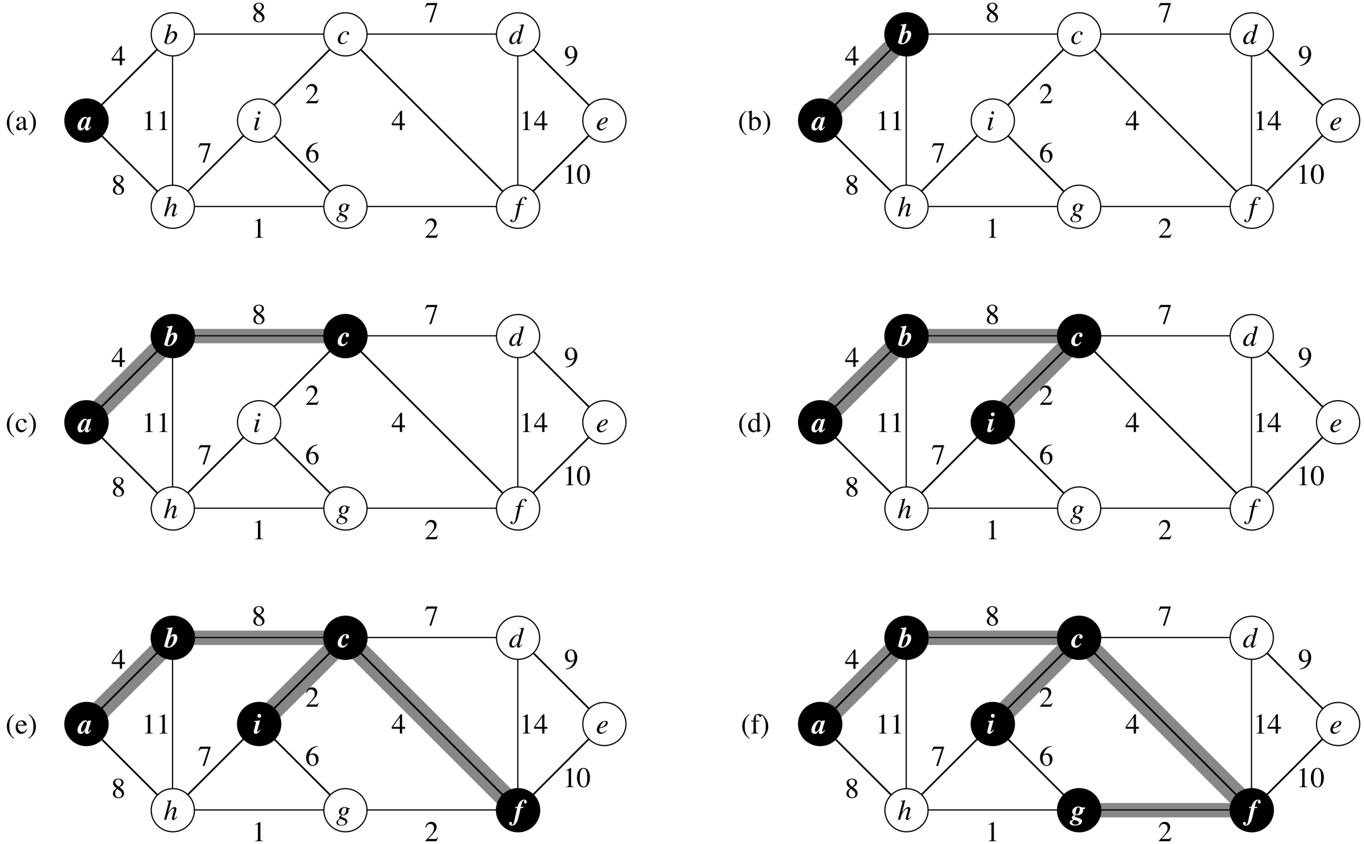
Keep track of vertices not in A, but adjacent to some vertex in A.

Add light edge

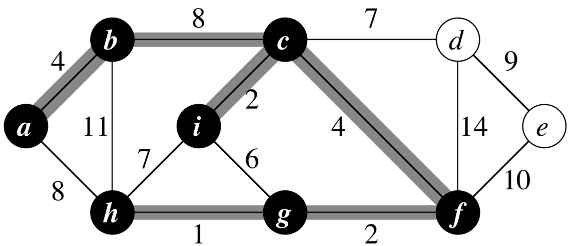


After u is added to A, distance of x to some vertex in A changes.

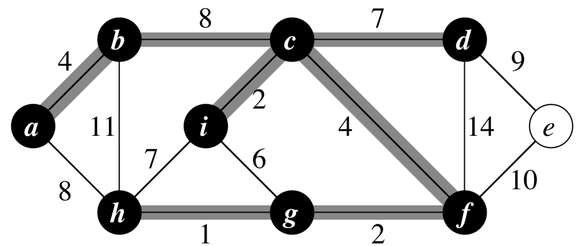
Use priority queue to keep track of distances.


(a)
(b)
(c)
(d)
(e)
(f)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2

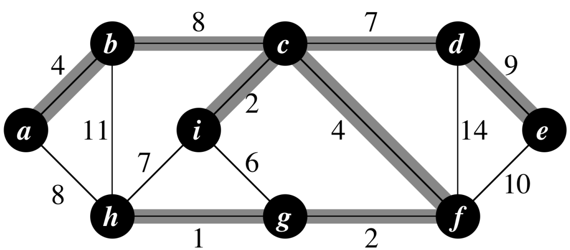
(g)


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2

(h)


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2

(i)


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
8
10
1
2

```
PRIM(G, w, r)
Q = ∅
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
    INSERT(Q, u)
DECREASE-KEY(Q, r, 0)        // r.key = 0
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            DECREASE-KEY(Q, v, w(u, v))
```

Running time of Prim's algorithm.

$V-1$ Extract-Min

E Decrease-Key

Arrays:		Heaps:	
$O(V)$	Extract-Min	$O(\log V)$	Extract-Min
$O(1)$	Decrease-Key	$O(\log V)$	DecreaseKey
$O(V^2)$	Total	$O(E \log V)$	

which is better?

Fibonacci Heaps:

$O(\log V)$ Extract-Min

$O(1)$ Decrease-Key (Amortized running time)

$O(V\log V + E)$

beats Array implementation $O(V^2)$

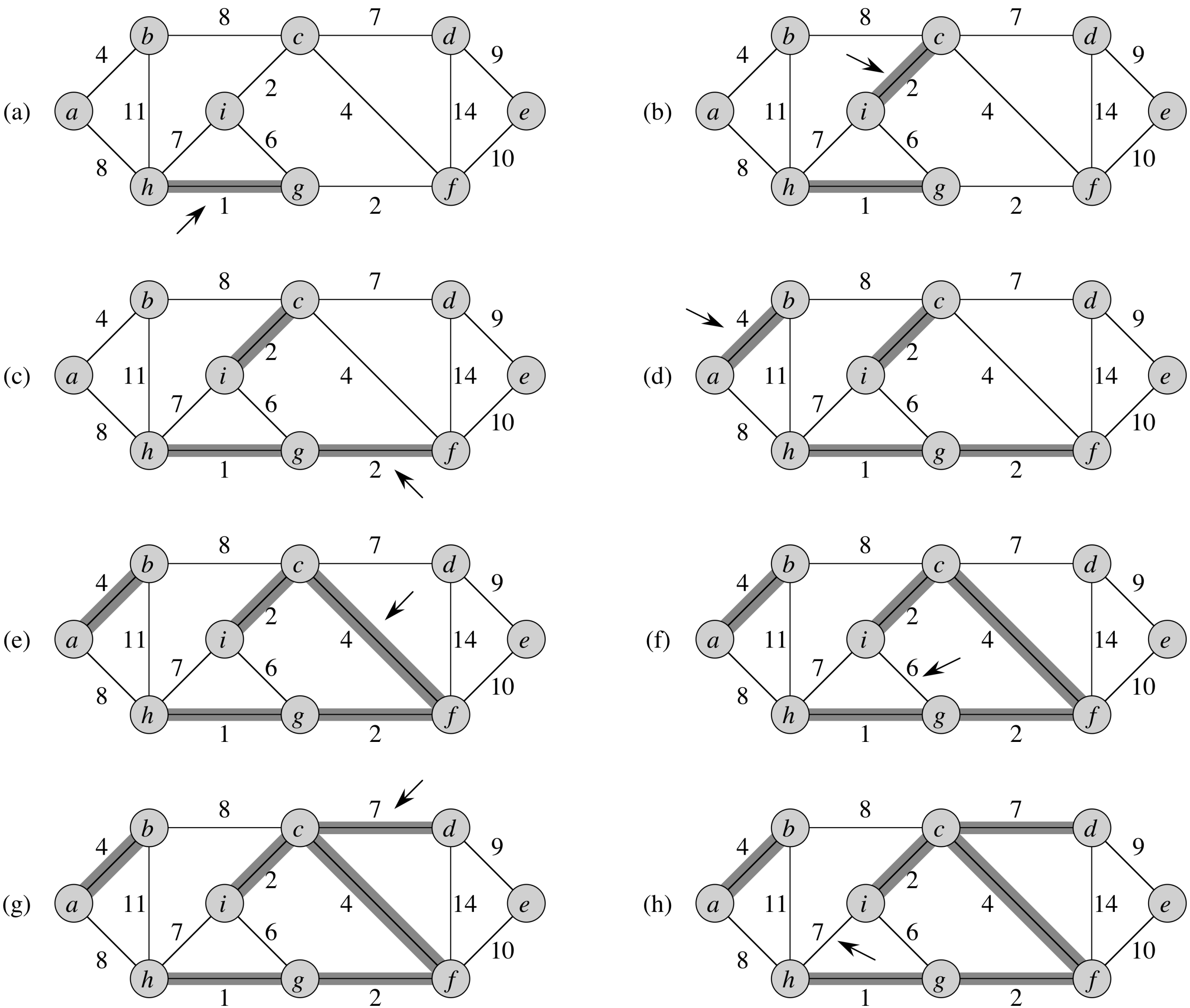
beats Heap implementation $O(E\log V)$

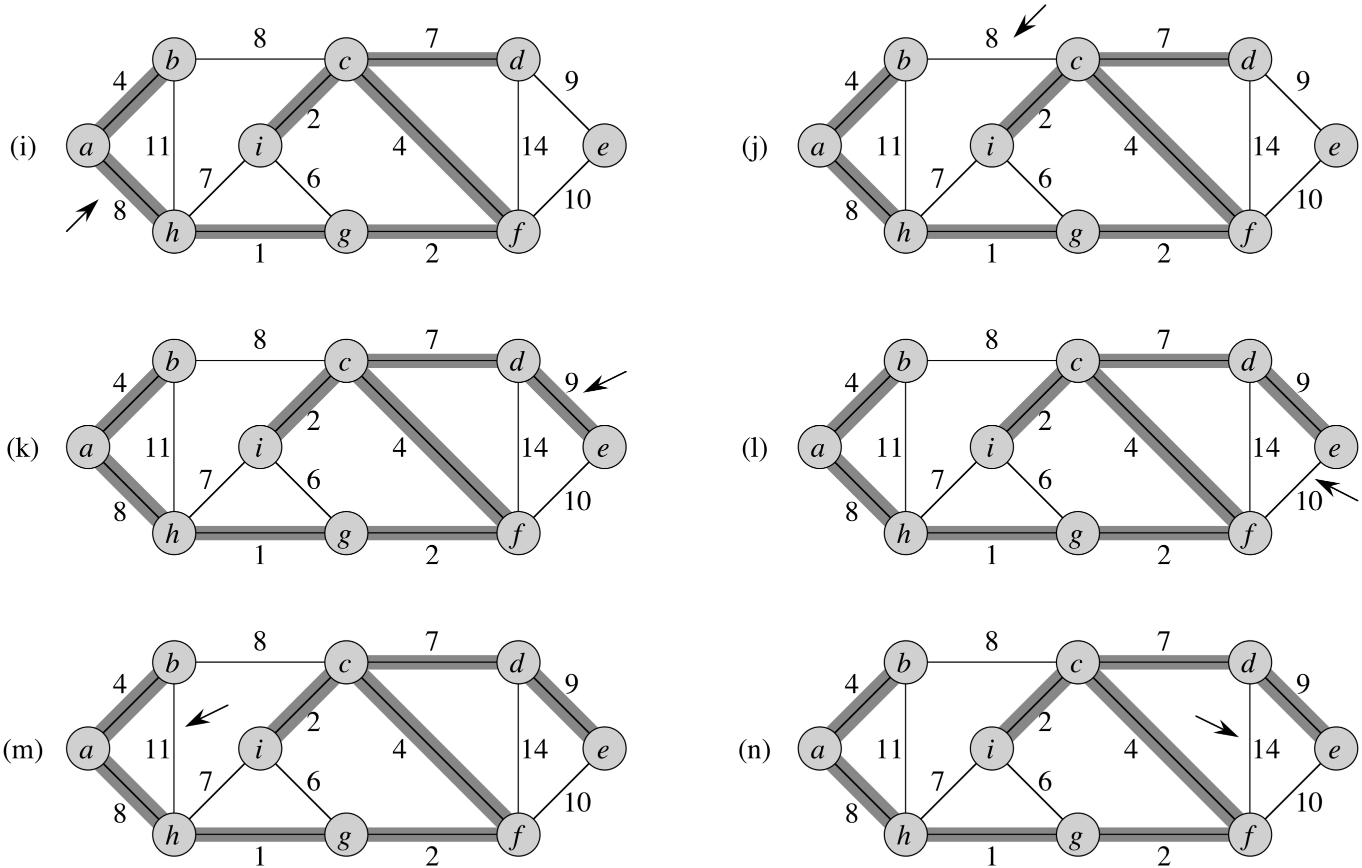
Kruskal's algorithm

Grow several trees.

Lowest-weight edge might belong to some MST.

Check if edge creates a cycle, if so discard.


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 10 8 1 2


(i)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
(j)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
(k)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
(l)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
(m)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
(n)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2

```
KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each (u, v) taken from the sorted list
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Need Disjoint Set Union operations

V Make-Sets $= V \times O(1)$

$2E$ Find-Sets $= 2E \times O(\log^* V)$

$V-1$ Unions $= \underline{(V-1) \times O(1)}$

$$O(V) + O(E \log^* V)$$

Total Running Time $= \underbrace{O(E \log E)}_{\text{sorting edges by weight}} + \underbrace{O(E \log^* V)}_{\text{Disjoint Set Union ops}} = O(E \log E)$