Recap: Minimum Spanning Tree (MST)

Input: G is an undirected, connected graph, with a weight $w(u,v)$ associated to each edge $\{u,v\} \in E$.

Goal: Find a spanning tree $T \subseteq E$ (acyclic and connects all vertices) with smallest total weight

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

```
GenericMST(G, w):
    A = ∅
    While A is not a spanning tree
        find {u,v} ∈ E safe for A
        A = A ∪ {u,v}
    Return A
```

Loop Inv.: A is subset of some MST

{u,v} safe for A if A ∪ {u,v} also a subset of some MST

Kruskal's Algorithm

Idea: Put each vertex in its own connected component. Then repeatedly add the edge with lowest weight that connects two connected components to A.

```
MST-Kruskal(G=(V,E), w):
1.    A=∅
2.    For all v∈V:                    } |V| · T(MakeSet)
3.        MakeSet(v)                  } // each vertex in its own connected comp.
4.    Sort E in non-decreasing order  |E| log|E| ≤ 2|E|log|V|
5.    For {u,v}∈E:
6.        If FindSet(u)≠FindSet(v): // u and v in diff. comp.
7.            A=A∪{u,v}
8.            Union(u,v) // merge u's and v's components
9.    Return A
```

Runtime: $|V| \cdot T(\text{MakeSet}) + T(\text{Sort } E) + |E| \cdot T(\text{FindSet}) + |V| \cdot T(\text{Union})$

Disjoint Set (CLRS §19.3):
MakeSet $\Theta(1)$
FindSet/Union $\Theta(\log |V|)$

$\Rightarrow \Theta(|V| + |E| \log |V| + |E| \log |V| + |V| \log |V|) = \Theta(|E| \log |V|)$

1. Prim's Algorithm

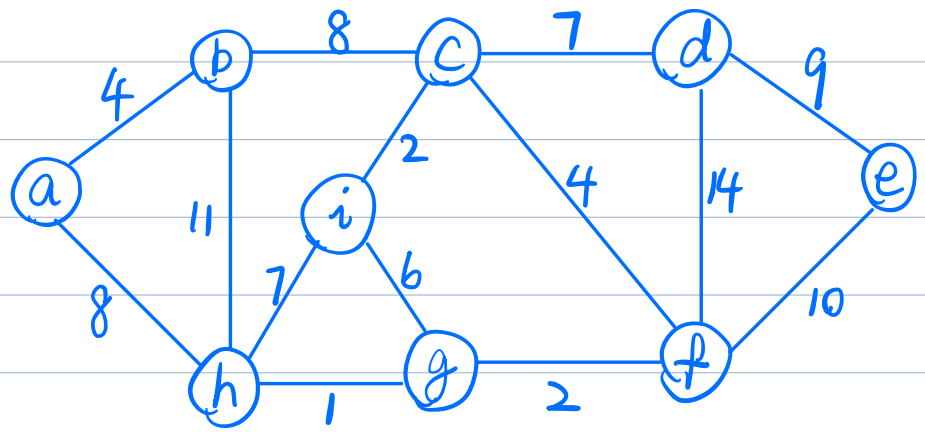
Idea: We grow a tree. Each time we add the vertex outside the tree that is connected by the lightest edge.

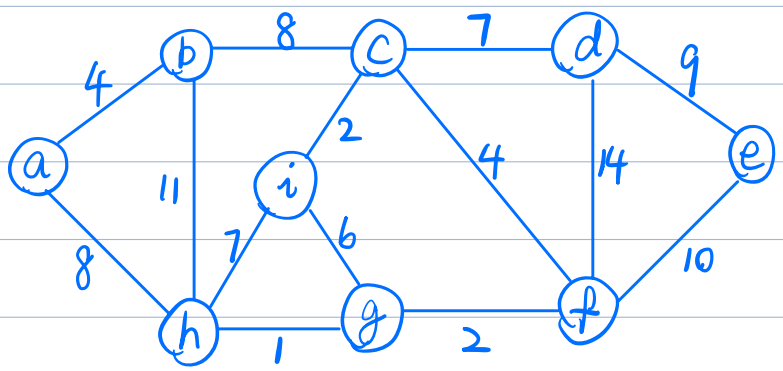
Correctness: Let the tree we grow be T, then the edge we added is a light edge for the cut (T, V-T), and therefore safe.

"root" of the tree

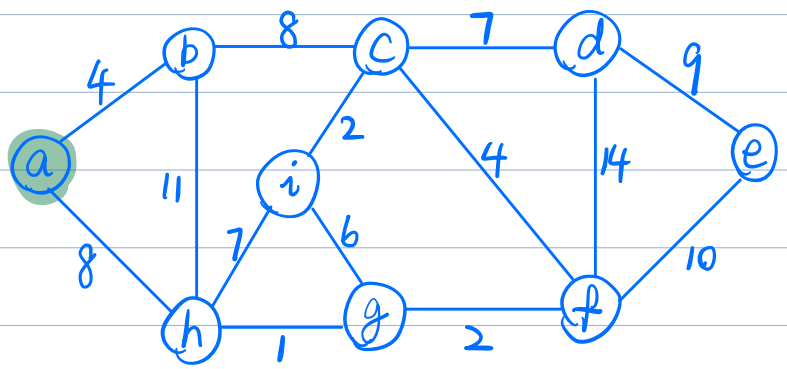
```
MST-Prim(G=(V,E), w, r)
1.   For u ∈ V:
2.       key(u) = ∞   // maintains weight of the lightest
3.       parent(u) = ⊥ // edge b/w u and the tree
4.   key(r) = 0
5.   For u ∈ V:
6.       PQ.Push(u, key(u))
7.   While !PQ.IsEmpty():
8.       u = PQ.ExtractMin()
9.       For v ∈ Adj[u]:
10.          If v ∈ PQ and w(u,v) < key(v)
11.              parent(v) = u
12.              key(v) = w(u,v)
13.              PQ.DecreaseKey(v, w(u,v))
```

e.g.

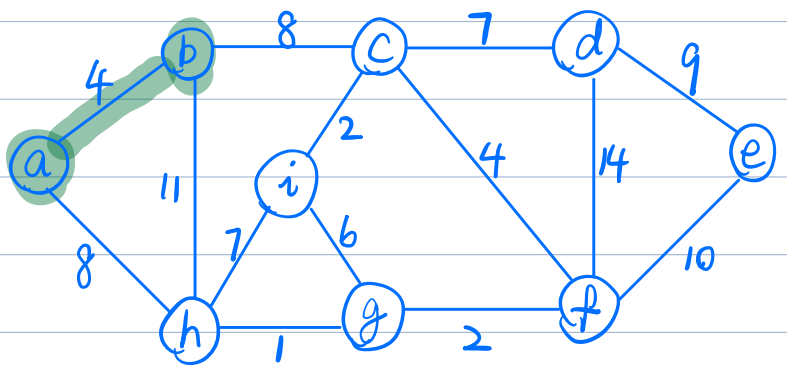




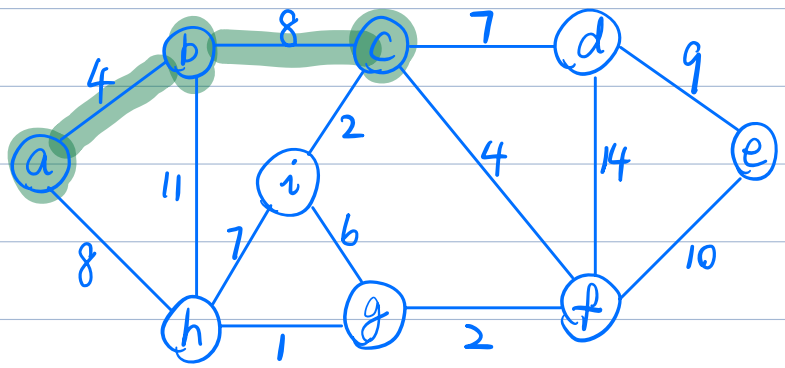
		parent	key
r →	a	⊥	0
	b	⊥	∞
	c	⊥	∞
	d	⊥	∞
	e	⊥	∞
	f	⊥	∞
	g	⊥	∞
	h	⊥	∞
	i	⊥	∞



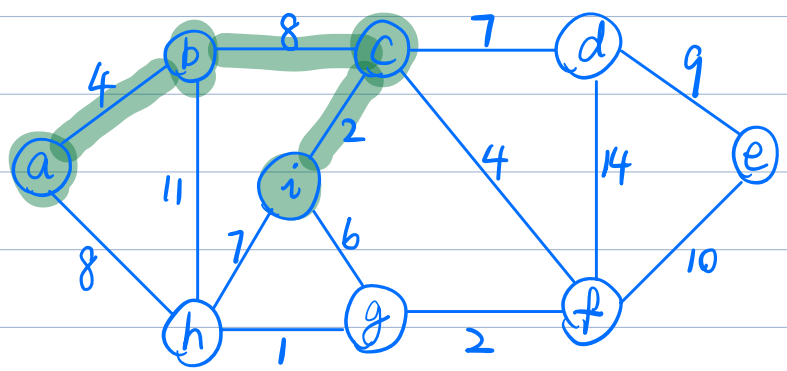
	parent	key
r → a	⊥	0
b	a	4
c	⊥	∞
d	⊥	∞
e	⊥	∞
f	⊥	∞
g	⊥	∞
h	a	8
i	⊥	∞



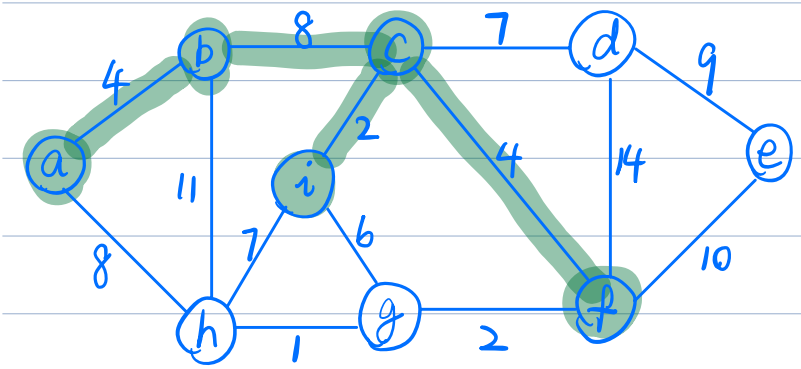
	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	⊥	∞
e	⊥	∞
f	⊥	∞
g	⊥	∞
h	a	8
i	⊥	∞



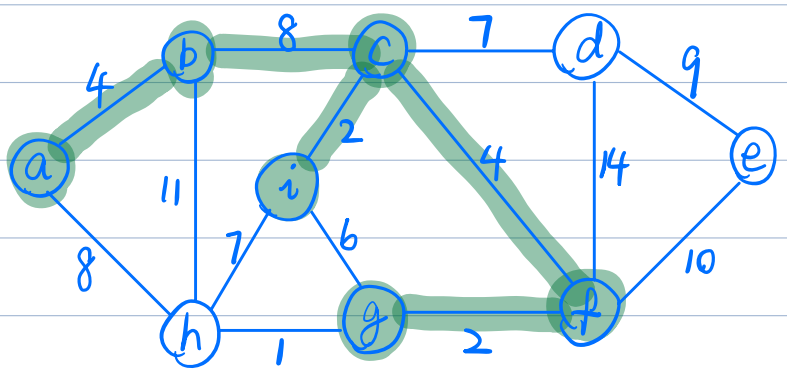
	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	⊥	∞
f	c	4
g	⊥	∞
h	a	8
i	c	2



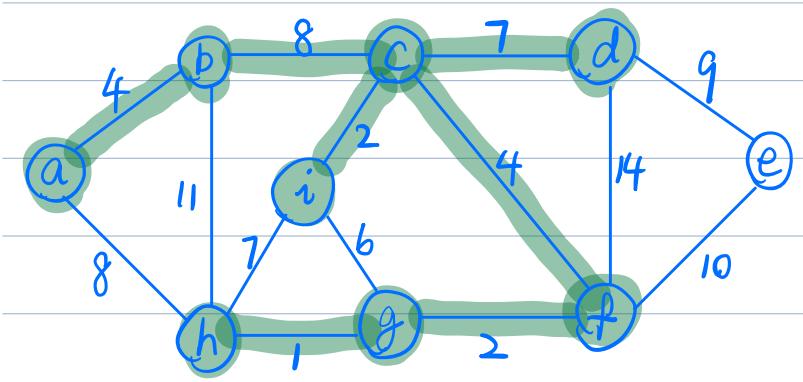
	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	⊥	∞
f	c	4
g	i	6
h	i	7
i	c	2



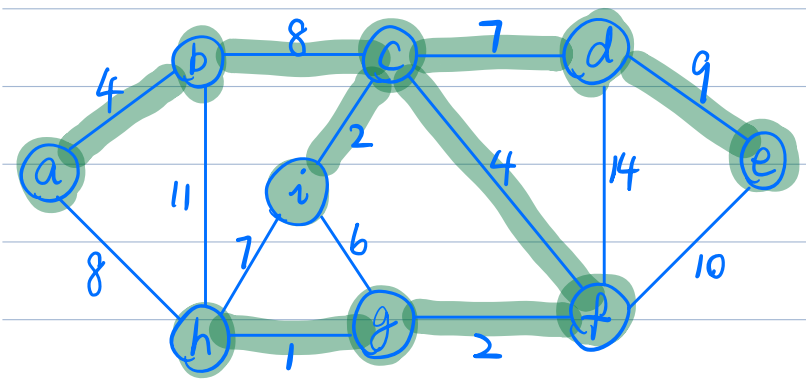
	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	f	10
f	c	4
g	f	2
h	i	7
i	c	2



	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	f	10
f	c	4
g	f	2
h	g	1
i	c	2

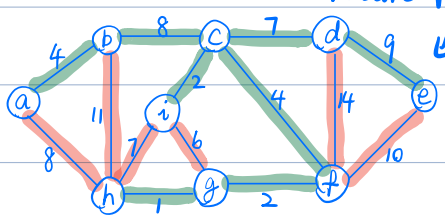


	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	d	9
f	c	4
g	f	2
h	g	1
i	c	2



	parent	key
r → a	⊥	0
b	a	4
c	b	8
d	c	7
e	d	9
f	c	4
g	f	2
h	g	1
i	c	2

Result from Kruskal ↙



Runtime:

```
MST-Prim(G=(V,E), w, r)
1.    For u ∈ V:                                  } |V|
2.        key(u) = ∞   // maintains weight of the lightest
3.        parent(u) = ⊥ // edge b/w u and the tree
4.    key(r) = 0                                  ← Θ(1)
5.    For u ∈ V:                                  } |V| × T(PQ.Push)
6.        PQ.Push(u, key(u))
7.    While !PQ.IsEmpty():   ← Iterates |V| times
8.        u = PQ.ExtractMin()
9.        For v ∈ Adj[u]:   ← In total |E| times
10.           If v ∈ PQ and w(u,v) < key(v)
11.               parent(v) = u
12.               key(v) = w(u,v)
13.               PQ.DecreaseKey(v, w(u,v))
```

Runtime is: $|V| + \Theta(1) + |V| \times T(PQ.Push) + |V| \times T(PQ.ExtMin) + |E| \cdot T(PQ.DecKey)$

Use PQ w/ binary heap, all "T"s are $\log|V| \Rightarrow |V| + \Theta(1) + |V|\log|V| + |E|\log|V| = \Theta((|E|+|V|)\log|V|)$.

w/ Fibonacci Heaps, we can reduce $T(PQ.DecKey)$ to $\Theta(1)$ (amortized).

Then, we have runtime $\Theta(|E| + |V|\log|V|)$.