Kruskal's Algorithm: Efficient Implementation of Greedy Algorithm

MST-Kruskal(G, w)

```
F ← ∅
𝒮 ← {{v} : v ∈ V}
sort the edges of E in non-decreasing order of weights w
for each edge (u, v) ∈ E in the order do
    S_u ← the set in 𝒮 containing u
    S_v ← the set in 𝒮 containing v
    if S_u ≠ S_v then
        F ← F ∪ {(u, v)}
        𝒮 ← 𝒮 \ {S_u} \ {S_v} ∪ {S_u ∪ S_v}
return (V, F)
```

Running Time of Kruskal's Algorithm

MST-Kruskal(G, w)

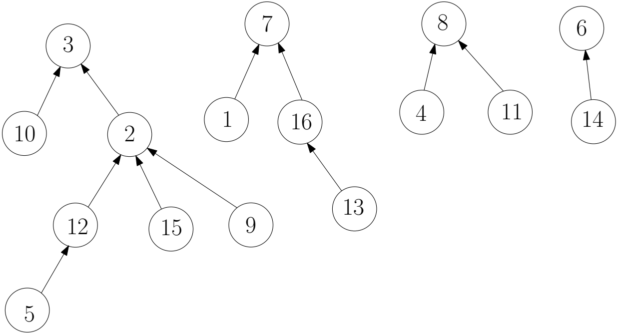
1: $F \leftarrow \emptyset$
2: $\mathcal{S} \leftarrow \{\{v\} : v \in V\}$
3: sort the edges of E in non-decreasing order of weights w
4: **for** each edge $(u, v) \in E$ in the order **do**
5: $\quad S_u \leftarrow$ the set in $\mathcal{S}$ containing u
6: $\quad S_v \leftarrow$ the set in $\mathcal{S}$ containing v
7: $\quad$**if** $S_u \neq S_v$ **then**
8: $\quad\quad F \leftarrow F \cup \{(u, v)\}$
9: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
10: **return** (V, F)

Use union-find data structure to support ❷, ❺, ❻, ❼, ❾.

Union-Find Data Structure

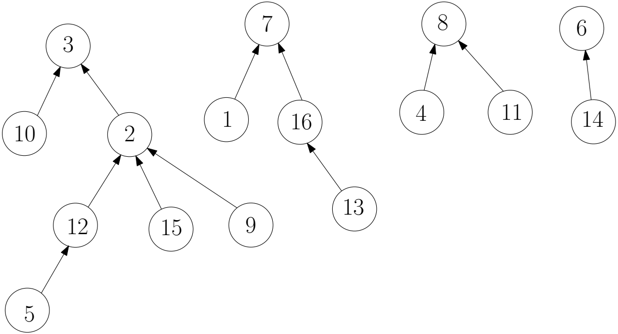
- V: ground set
- We need to maintain a partition of V and support following operations:
  - Check if u and v are in the same set of the partition
  - Merge two sets in partition

- $V = \{1, 2, 3, \cdots, 16\}$
- Partition: $\{2, 3, 5, 9, 10, 12, 15\}, \{1, 7, 13, 16\}, \{4, 8, 11\}, \{6, 14\}$

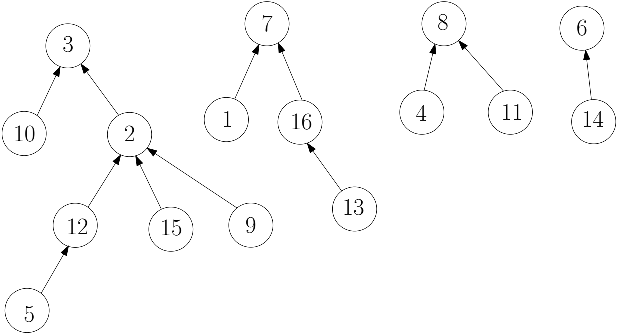


- $par[i]$: parent of i, ($par[i] = \perp$ if i is a root).

Union-Find Data Structure

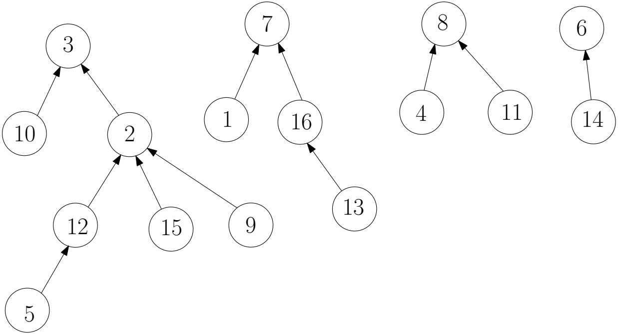


Union-Find Data Structure



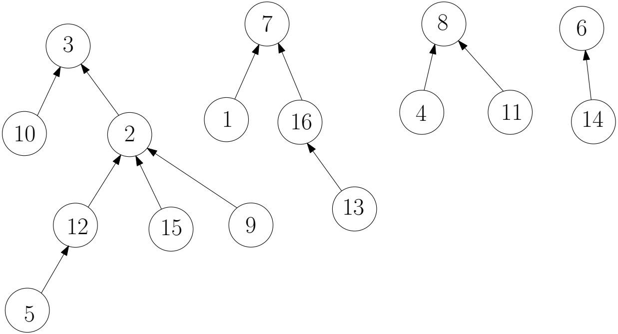
- Q: how can we check if u and v are in the same set?

Union-Find Data Structure



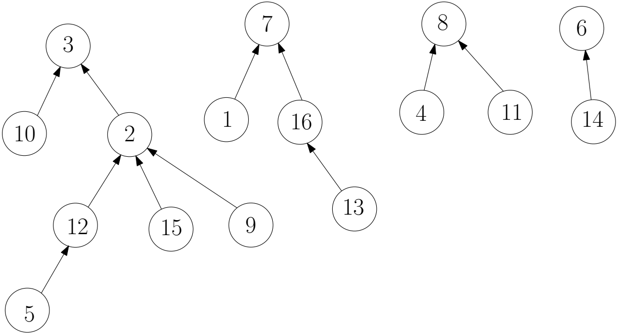
- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).

Union-Find Data Structure



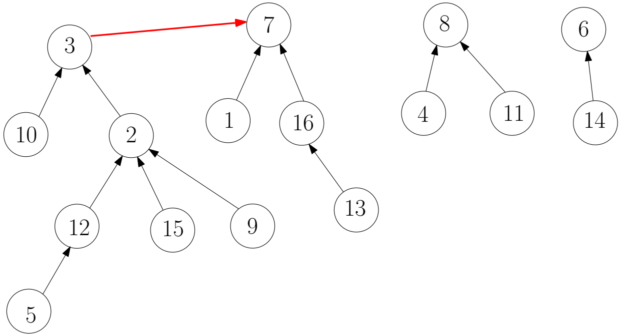
- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u

Union-Find Data Structure



- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u
- Merge the trees with root r and r': $par[r] \leftarrow r'$.

Union-Find Data Structure



- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u
- Merge the trees with root r and r': $par[r] \leftarrow r'$.

Union-Find Data Structure

root(v)

```
if par[v] = ⊥ then
    return v
else
    return root(par[v])
```

Union-Find Data Structure

root(v)

1: **if** $par[v] = \perp$ **then**
2:     **return** v
3: **else**
4:     **return** root($par[v]$)

- Problem: the tree might too deep; running time might be large

Union-Find Data Structure

root(v)

```
if par[v] = ⊥ then
    return v
else
    return root(par[v])
```

- Problem: the tree might too deep; running time might be large
- Improvement: all vertices in the path directly point to the root, saving time in the future.

Union-Find Data Structure

root(v)

```
if par[v] = ⊥ then
    return v
else
    return root(par[v])
```

root(v)

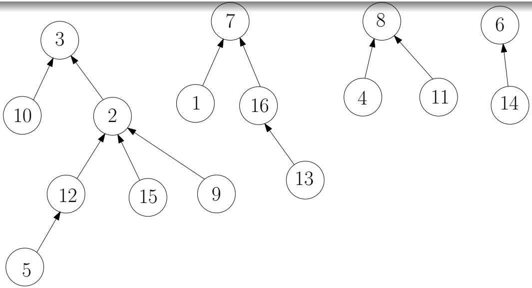
```
if par[v] = ⊥ then
    return v
else
    par[v] ← root(par[v])
return par[v]
```

- Problem: the tree might too deep; running time might be large
- Improvement: all vertices in the path directly point to the root, saving time in the future.

Union-Find Data Structure

root(v)

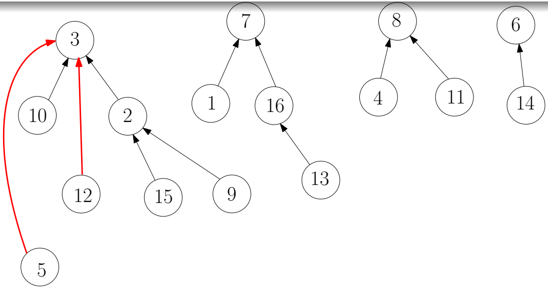
if $par[v] = \bot$ **then**
  return v
else
  $par[v] \leftarrow \text{root}(par[v])$
  return $par[v]$



Union-Find Data Structure

root(v)

```
if par[v] = ⊥ then
    return v
else
    par[v] ← root(par[v])
    return par[v]
```



MST-Kruskal(G, w)

$F \leftarrow \emptyset$
$\mathcal{S} \leftarrow \{\{v\} : v \in V\}$
sort the edges of E in non-decreasing order of weights w
for each edge $(u, v) \in E$ in the order **do**
    $S_u \leftarrow$ the set in $\mathcal{S}$ containing u
    $S_v \leftarrow$ the set in $\mathcal{S}$ containing v
    if $S_u \neq S_v$ **then**
        $F \leftarrow F \cup \{(u, v)\}$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
return (V, F)

MST-Kruskal(G, w)

$F \leftarrow \emptyset$
for every $v \in V$ **do**: $par[v] \leftarrow \bot$
sort the edges of E in non-decreasing order of weights w
for each edge $(u, v) \in E$ in the order **do**
    $u' \leftarrow \text{root}(u)$
    $v' \leftarrow \text{root}(v)$
    if $u' \neq v'$ **then**
        $F \leftarrow F \cup \{(u, v)\}$
        $par[u'] \leftarrow v'$
return (V, F)

MST-Kruskal(G, w)

```
1:  F ← ∅
2:  for every v ∈ V do: par[v] ← ⊥
3:  sort the edges of E in non-decreasing order of weights w
4:  for each edge (u, v) ∈ E in the order do
5:      u' ← root(u)
6:      v' ← root(v)
7:      if u' ≠ v' then
8:          F ← F ∪ {(u, v)}
9:          par[u'] ← v'
10: return (V, F)
```

- 2, 5, 6, 7, 9 takes time $O(m\alpha(n))$
- $\alpha(n)$ is very slow-growing: $\alpha(n) \le 4$ for $n \le 10^{80}$.

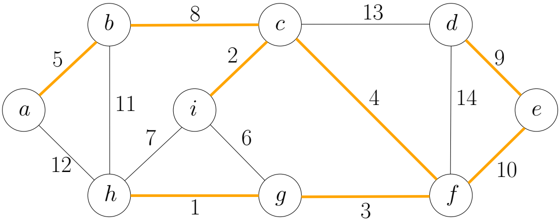
MST-Kruskal(G, w)

```
1:  F ← ∅
2:  for every v ∈ V do: par[v] ← ⊥
3:  sort the edges of E in non-decreasing order of weights w
4:  for each edge (u, v) ∈ E in the order do
5:      u' ← root(u)
6:      v' ← root(v)
7:      if u' ≠ v' then
8:          F ← F ∪ {(u, v)}
9:          par[u'] ← v'
10: return (V, F)
```

- 2, 5, 6, 7, 9 takes time $O(m\alpha(n))$
- $\alpha(n)$ is very slow-growing: $\alpha(n) \leq 4$ for $n \leq 10^{80}$.
- Running time $=$ time for 3 $= O(m \lg n)$.

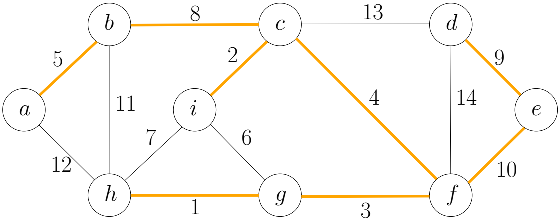
Assumption Assume all edge weights are different.

Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



Assumption Assume all edge weights are different.

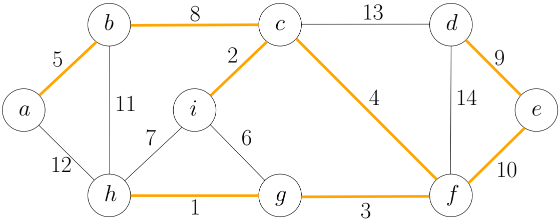
Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



- (i, g) is not in the MST because of cycle (i, c, f, g)

Assumption Assume all edge weights are different.

Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



- (i, g) is not in the MST because of cycle (i, c, f, g)
- (e, f) is in the MST because no such cycle exists

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm
2. Single Source Shortest Paths
   - Dijkstra's Algorithm
3. Shortest Paths in Graphs with Negative Weights
4. All-Pair Shortest Paths and Floyd-Warshall

Two Methods to Build a MST

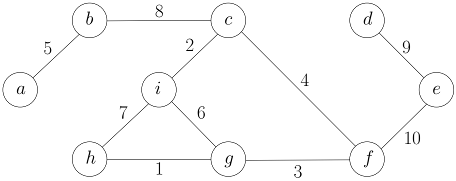
1. Start from $F \leftarrow \emptyset$, and add edges to F one by one until we obtain a spanning tree

Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to F one by one until we obtain a spanning tree
2. Start from $F \leftarrow E$, and remove edges from F one by one until we obtain a spanning tree

Two Methods to Build a MST

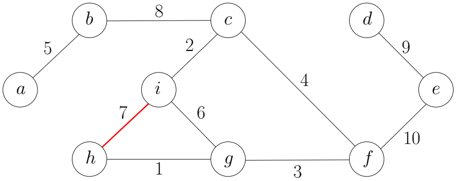
1. Start from $F \leftarrow \emptyset$, and add edges to F one by one until we obtain a spanning tree
2. Start from $F \leftarrow E$, and remove edges from F one by one until we obtain a spanning tree



Q: Which edge can be safely excluded from the MST?

Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to F one by one until we obtain a spanning tree
2. Start from $F \leftarrow E$, and remove edges from F one by one until we obtain a spanning tree

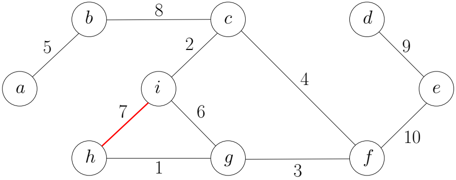


Q: Which edge can be safely excluded from the MST?

A: The heaviest non-bridge edge.

Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to F one by one until we obtain a spanning tree
2. Start from $F \leftarrow E$, and remove edges from F one by one until we obtain a spanning tree



Q: Which edge can be safely excluded from the MST?

A: The heaviest non-bridge edge.

Def. A bridge is an edge whose removal disconnects the graph.

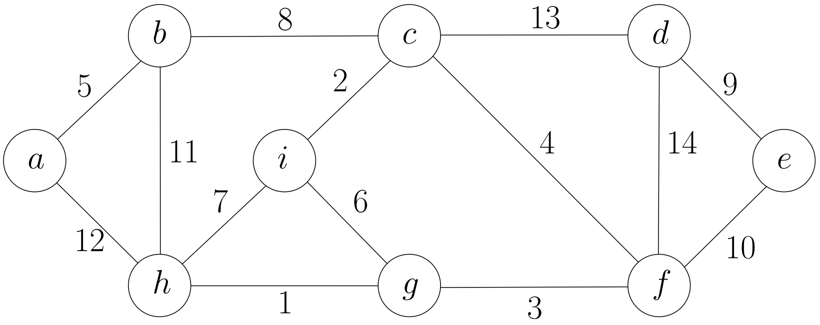
Lemma It is safe to exclude the heaviest non-bridge edge: there is a MST that does not contain the heaviest non-bridge edge.

Reverse Kruskal's Algorithm

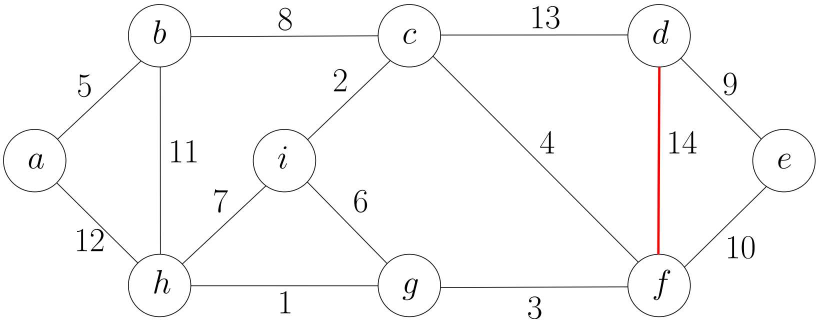
MST-Greedy(G, w)

$F \leftarrow E$
sort E in non-increasing order of weights
for every e in this order **do**
    if $(V, F \setminus \{e\})$ is connected **then**
        $F \leftarrow F \setminus \{e\}$
return (V, F)

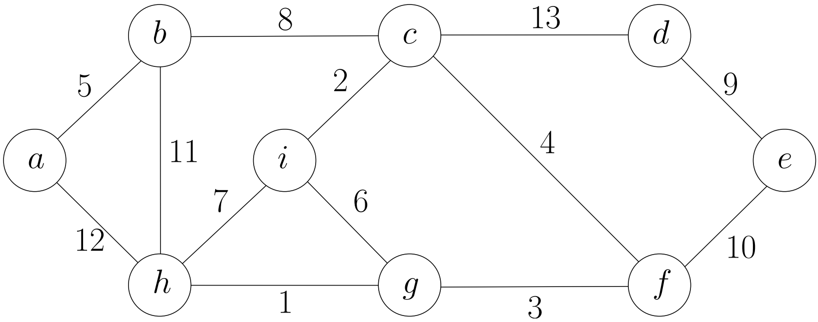
Reverse Kruskal's Algorithm: Example



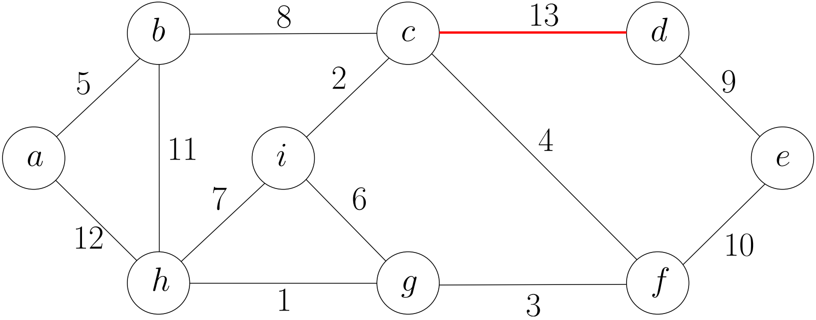
Reverse Kruskal's Algorithm: Example



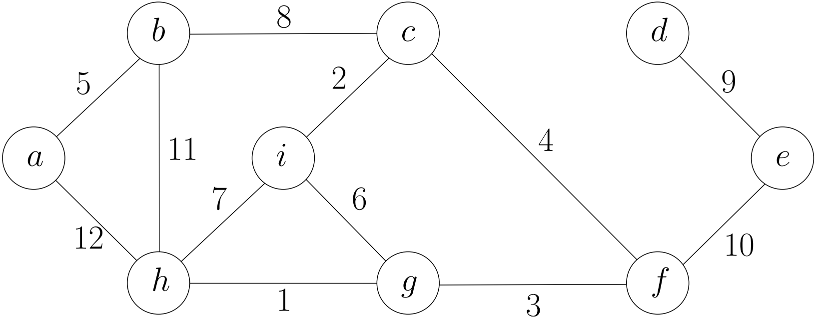
Reverse Kruskal's Algorithm: Example



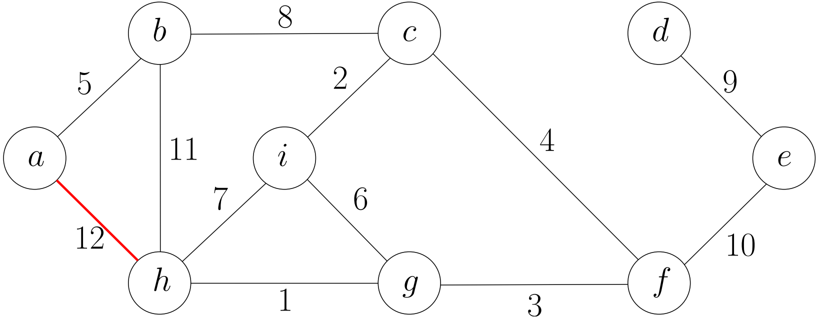
Reverse Kruskal's Algorithm: Example



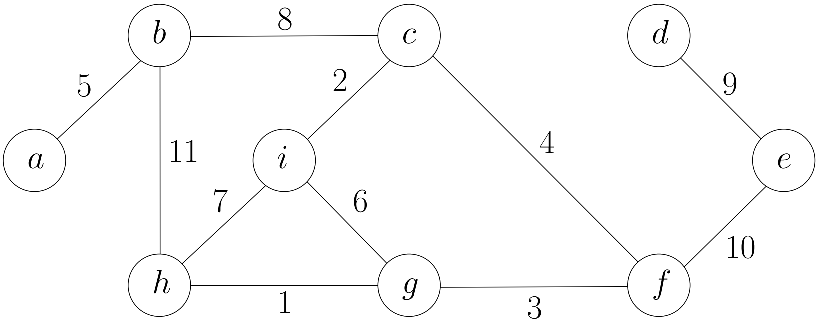
Reverse Kruskal's Algorithm: Example



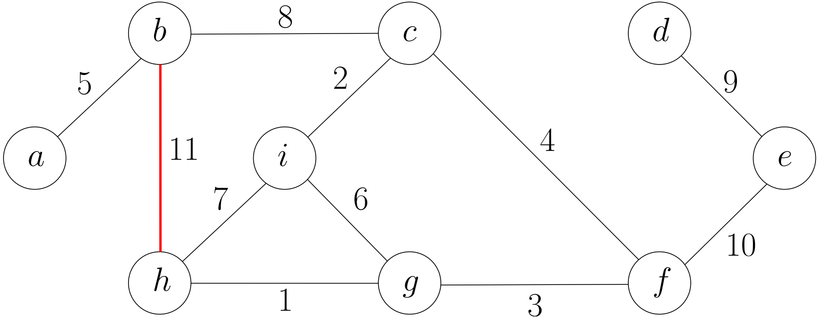
Reverse Kruskal's Algorithm: Example



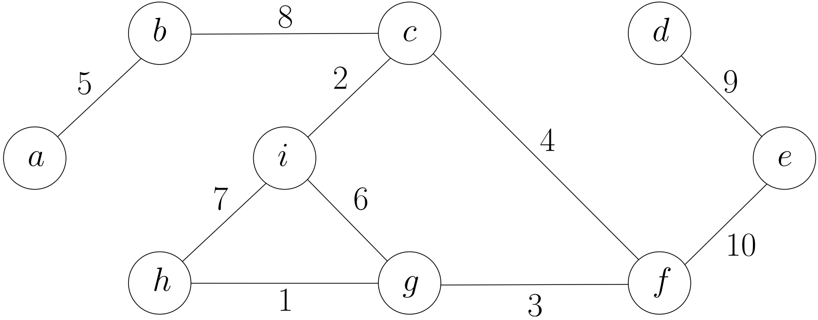
Reverse Kruskal's Algorithm: Example



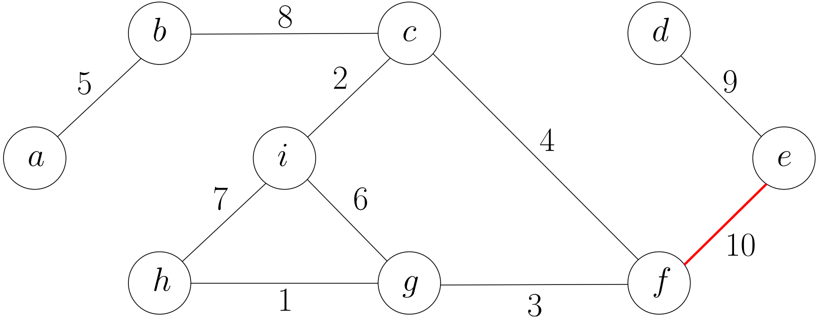
Reverse Kruskal's Algorithm: Example



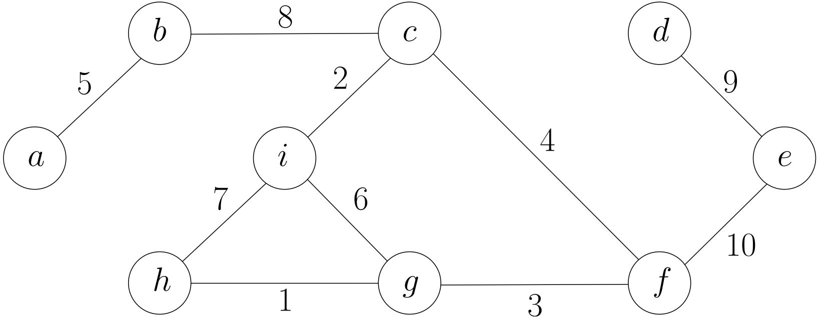
Reverse Kruskal's Algorithm: Example



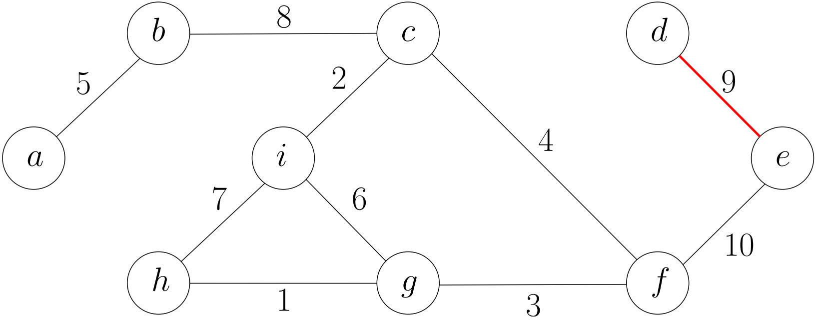
Reverse Kruskal's Algorithm: Example



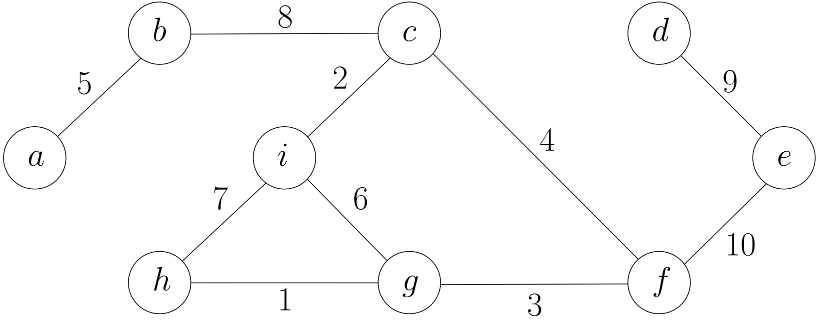
Reverse Kruskal's Algorithm: Example



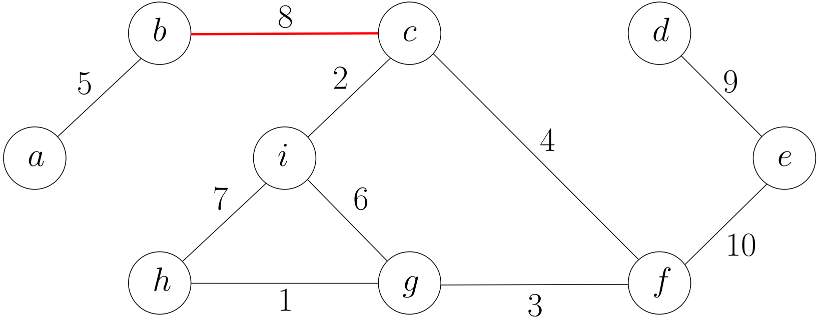
Reverse Kruskal's Algorithm: Example



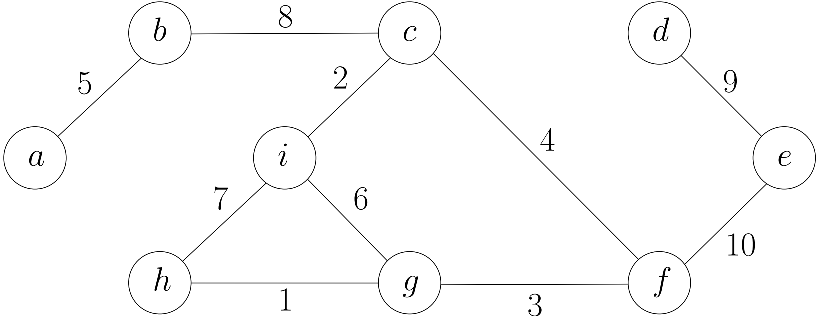
Reverse Kruskal's Algorithm: Example



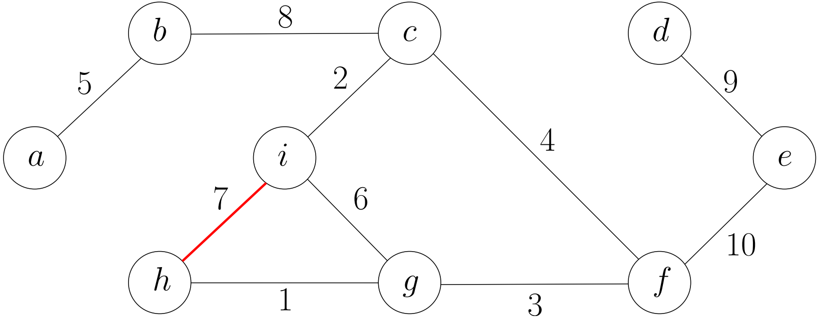
Reverse Kruskal's Algorithm: Example



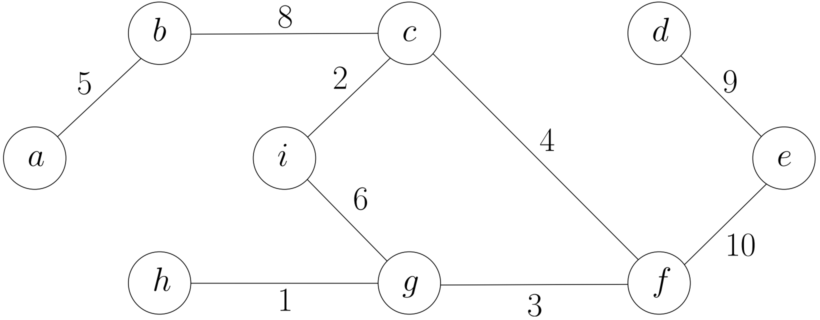
Reverse Kruskal's Algorithm: Example



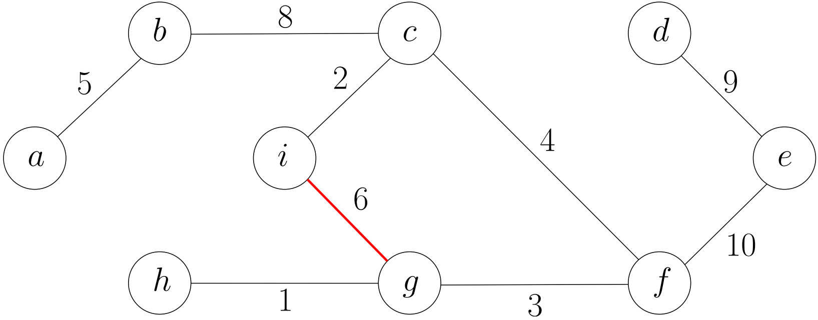
Reverse Kruskal's Algorithm: Example



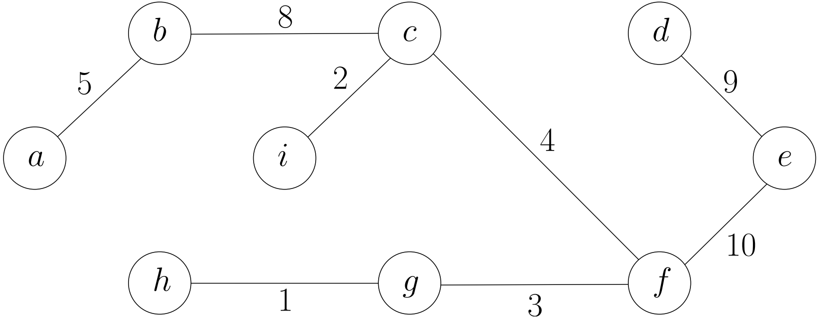
Reverse Kruskal's Algorithm: Example



Reverse Kruskal's Algorithm: Example

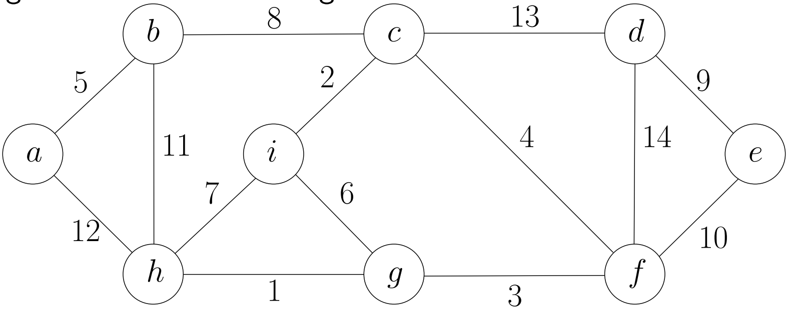


Reverse Kruskal's Algorithm: Example



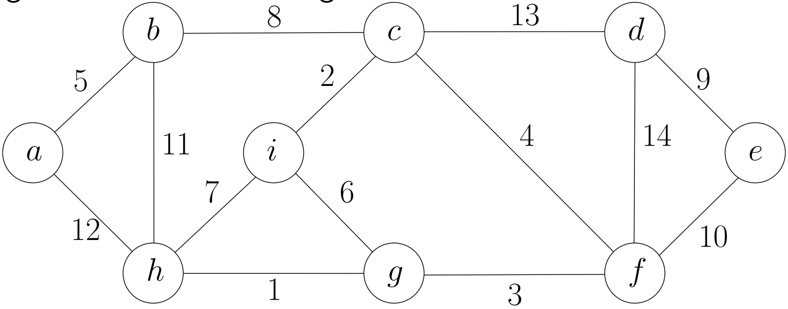
Outline

- Recall the greedy strategy for Kruskal's algorithm: choose the edge with the smallest weight.



Design Greedy Strategy for MST

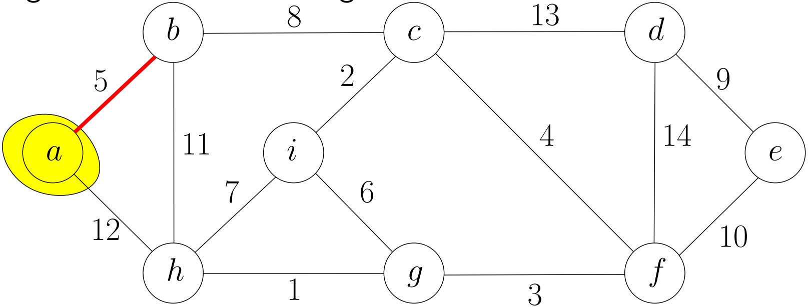
- Recall the greedy strategy for Kruskal's algorithm: choose the edge with the smallest weight.



- Greedy strategy for Prim's algorithm: choose the lightest edge incident to a.

Design Greedy Strategy for MST

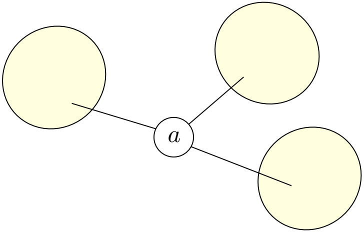
- Recall the greedy strategy for Kruskal's algorithm: choose the edge with the smallest weight.



- Greedy strategy for Prim's algorithm: choose the lightest edge incident to a.

Lemma It is safe to include the lightest edge incident to a.

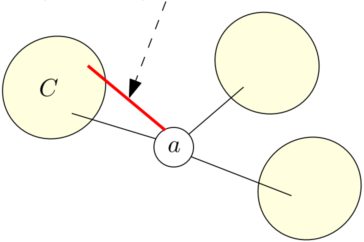
Lemma It is safe to include the lightest edge incident to a.



Proof.

- Let T be a MST
- Consider all components obtained by removing a from T

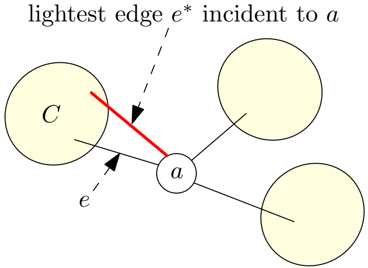
Lemma It is safe to include the lightest edge incident to a.



Proof.

- Let T be a MST
- Consider all components obtained by removing a from T
- Let e^* be the lightest edge incident to a and e^* connects a to component C

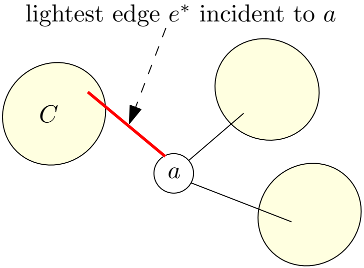
Lemma It is safe to include the lightest edge incident to a.



Proof.

- Let T be a MST
- Consider all components obtained by removing a from T
- Let e^* be the lightest edge incident to a and e^* connects a to component C
- Let e be the edge in T connecting a to C

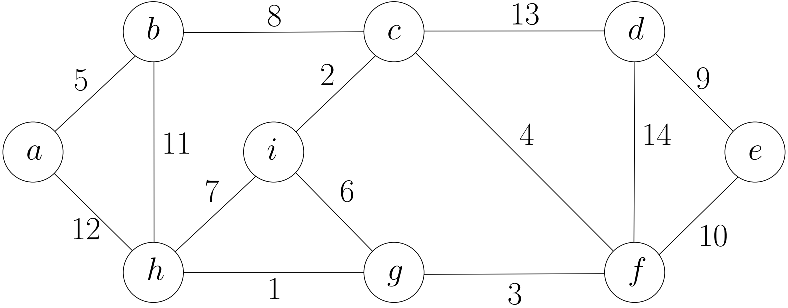
Lemma It is safe to include the lightest edge incident to a.



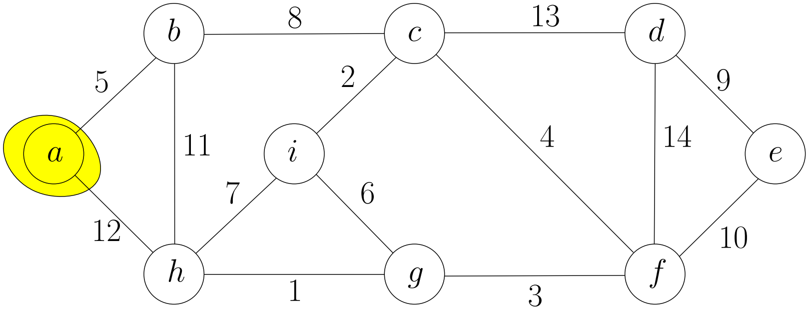
Proof.

- Let T be a MST
- Consider all components obtained by removing a from T
- Let e^* be the lightest edge incident to a and e^* connects a to component C
- Let e be the edge in T connecting a to C
- $T' = T \setminus \{e\} \cup \{e^*\}$ is a spanning tree with $w(T') \leq w(T)$ $\square$

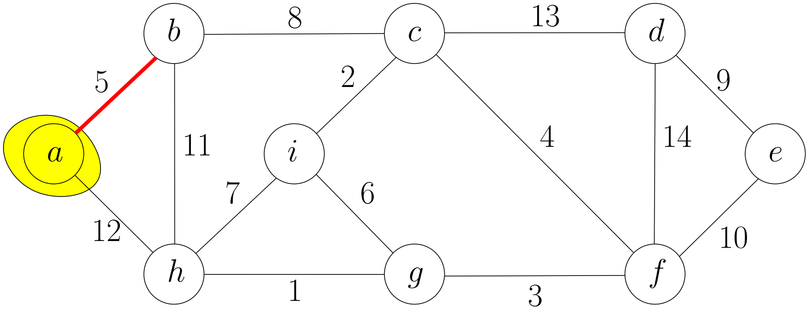
Prim's Algorithm: Example



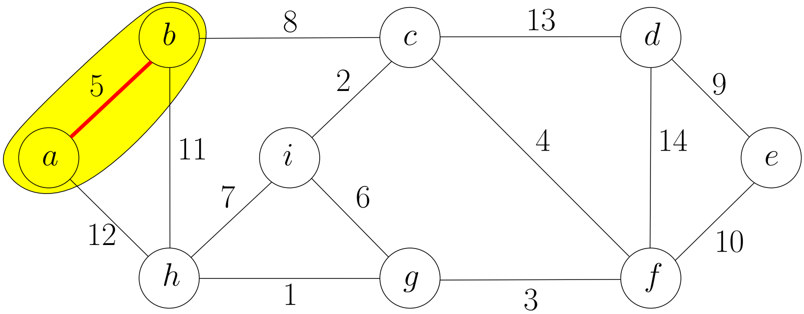
Prim's Algorithm: Example



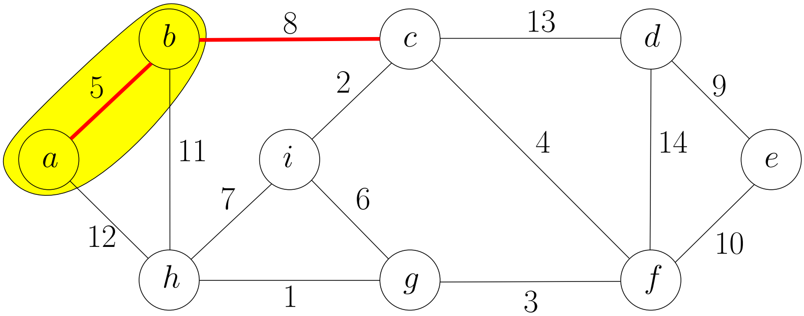
Prim's Algorithm: Example



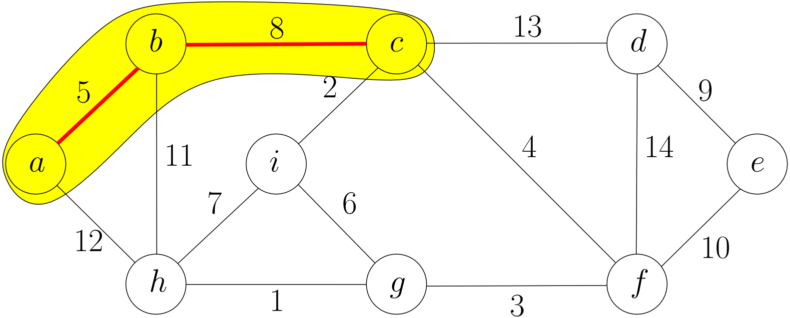
Prim's Algorithm: Example



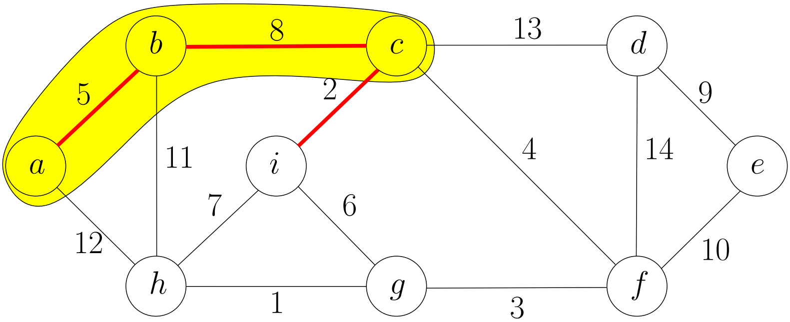
Prim's Algorithm: Example



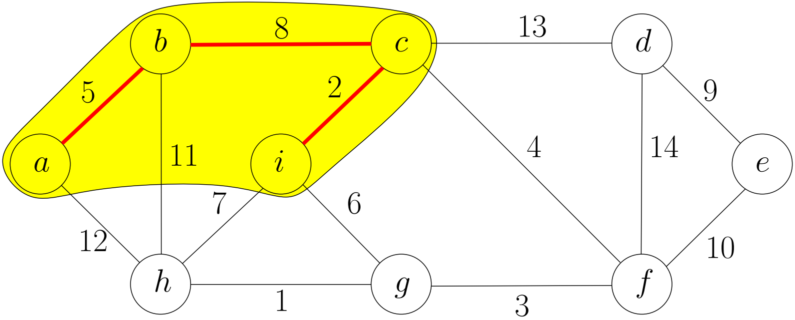
Prim's Algorithm: Example



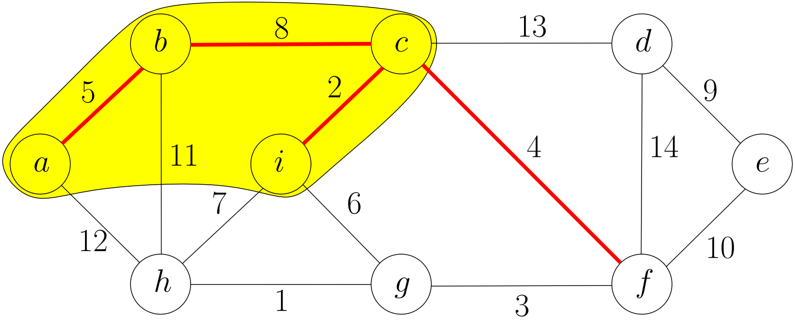
Prim's Algorithm: Example



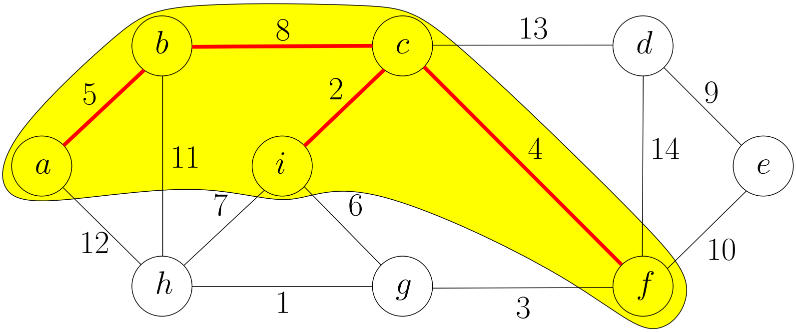
Prim's Algorithm: Example



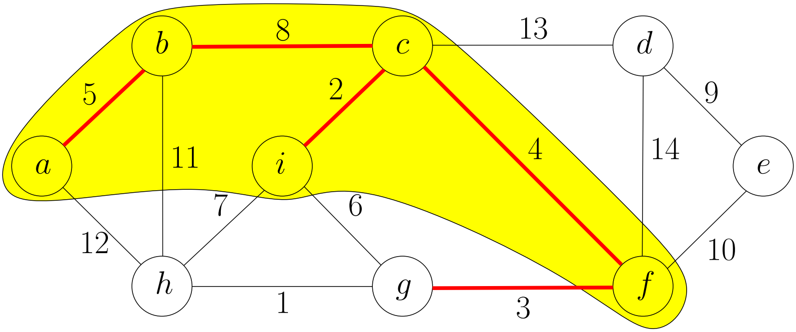
Prim's Algorithm: Example



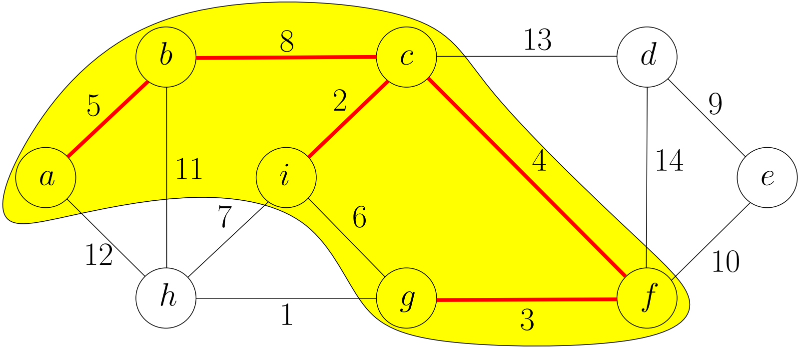
Prim's Algorithm: Example



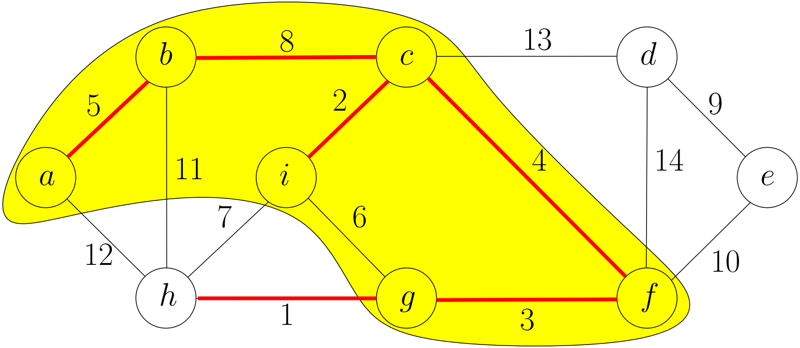
Prim's Algorithm: Example



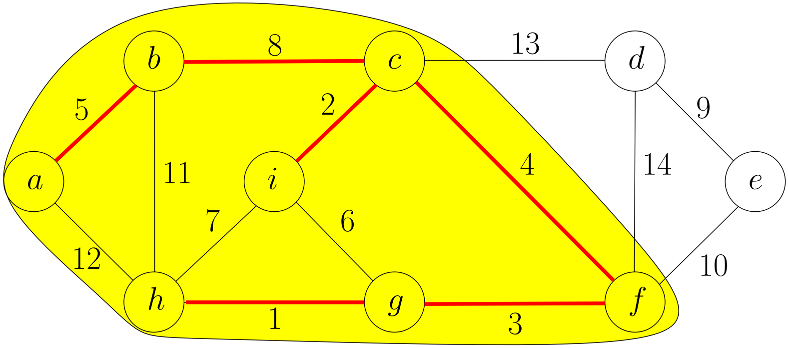
Prim's Algorithm: Example



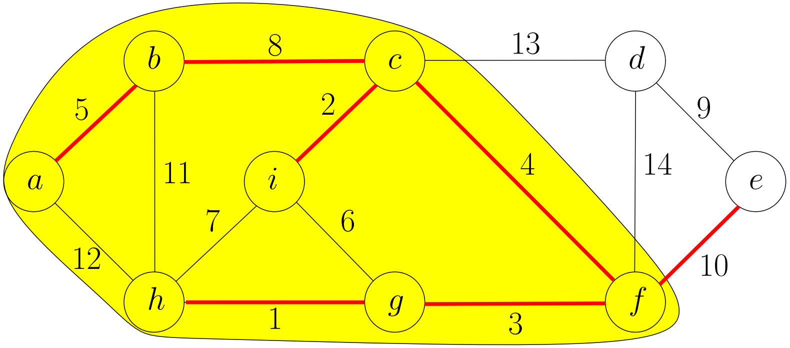
Prim's Algorithm: Example



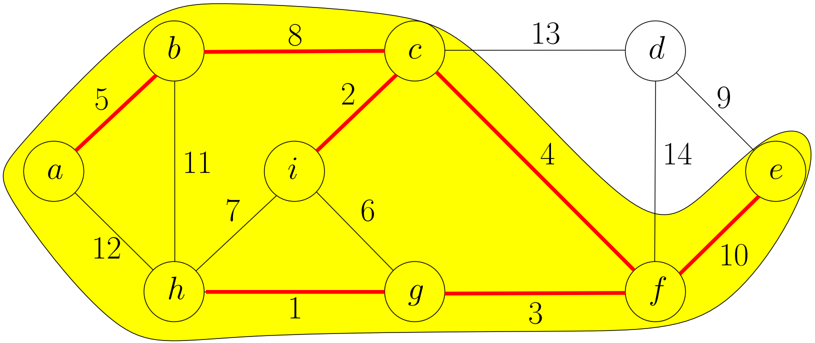
Prim's Algorithm: Example



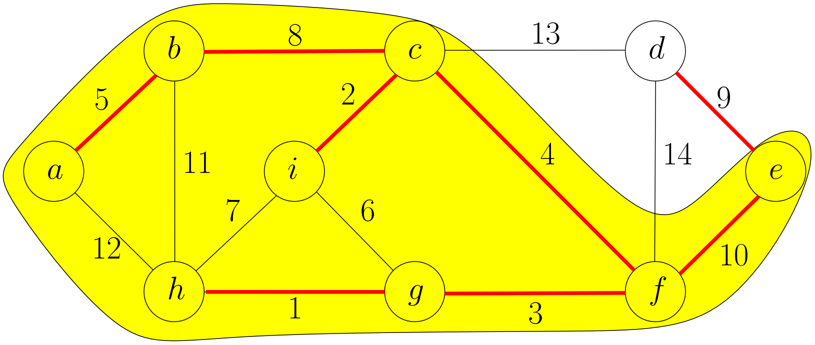
Prim's Algorithm: Example



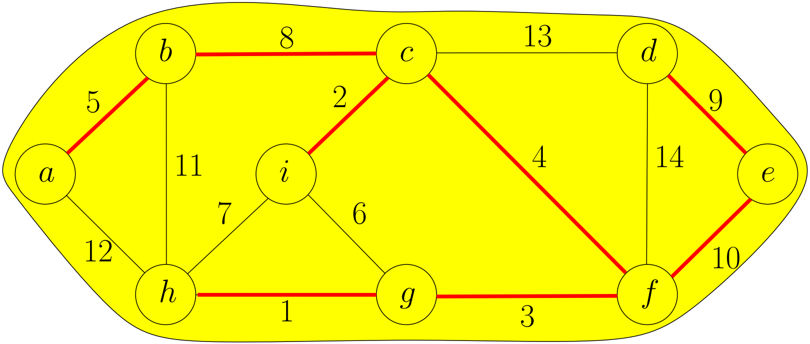
Prim's Algorithm: Example



Prim's Algorithm: Example



Prim's Algorithm: Example



Greedy Algorithm

MST-Greedy1(G, w)

$S \leftarrow \{s\}$, where s is arbitrary vertex in V
$F \leftarrow \emptyset$
while $S \neq V$ **do**
    $(u, v) \leftarrow$ lightest edge between S and $V \setminus S$,
        where $u \in S$ and $v \in V \setminus S$
    $S \leftarrow S \cup \{v\}$
    $F \leftarrow F \cup \{(u, v)\}$
return (V, F)

Greedy Algorithm

MST-Greedy1(G, w)

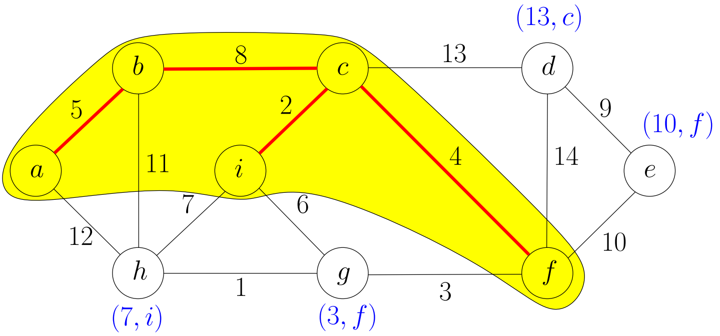
```
S ← {s}, where s is arbitrary vertex in V
F ← ∅
while S ≠ V do
    (u, v) ← lightest edge between S and V \ S,
                              where u ∈ S and v ∈ V \ S
    S ← S ∪ {v}
    F ← F ∪ {(u, v)}
return (V, F)
```

- Running time of naive implementation: $O(nm)$

Prim's Algorithm: Efficient Implementation of Greedy Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v) \in E} w(u,v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v) \in E} w(u,v)$:
  $(\pi[v], v)$ is the lightest edge between v and S



Prim's Algorithm: Efficient Implementation of Greedy Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v) \in E} w(u, v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v) \in E} w(u, v)$:
  $(\pi[v], v)$ is the lightest edge between v and S

In every iteration

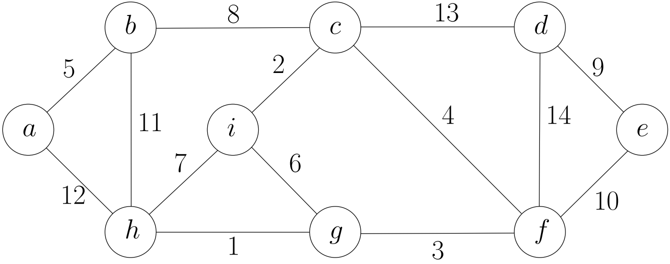
- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values.

Prim's Algorithm

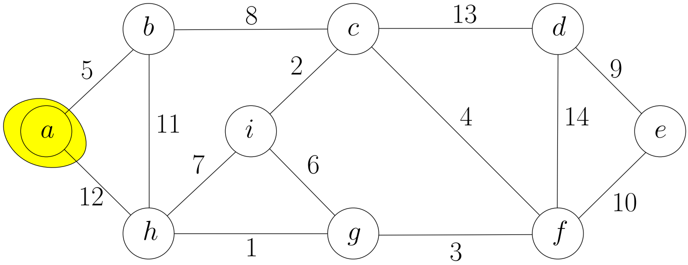
MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
while $S \neq V$ **do**
    $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$
            $\pi[v] \leftarrow u$
return $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$

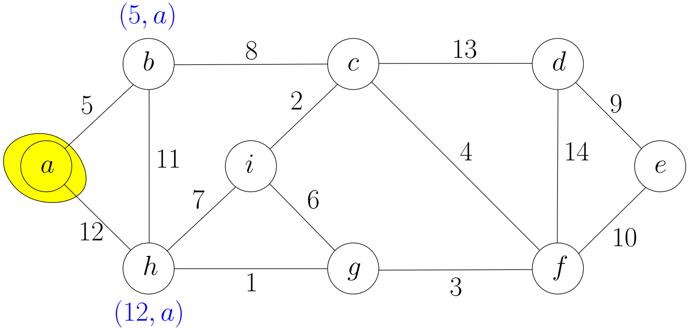
Example



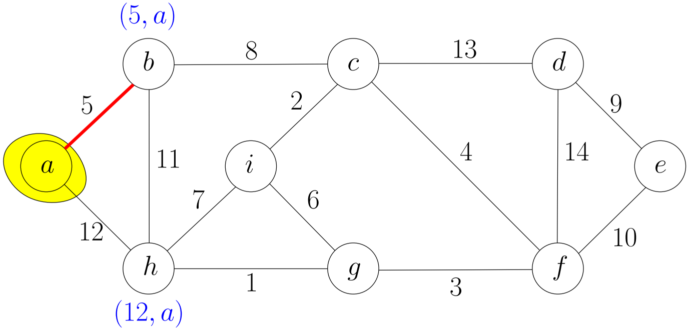
Example



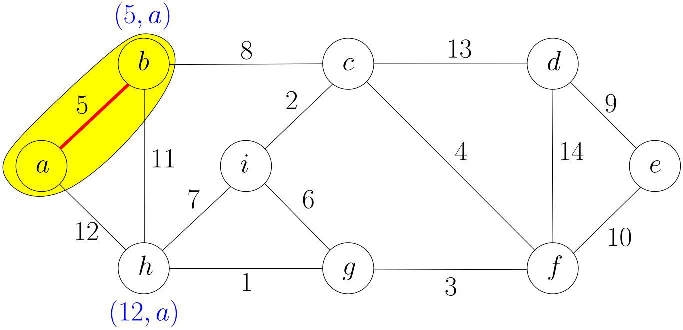
Example



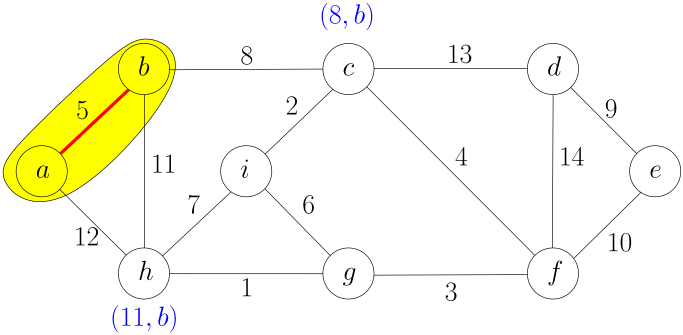
Example



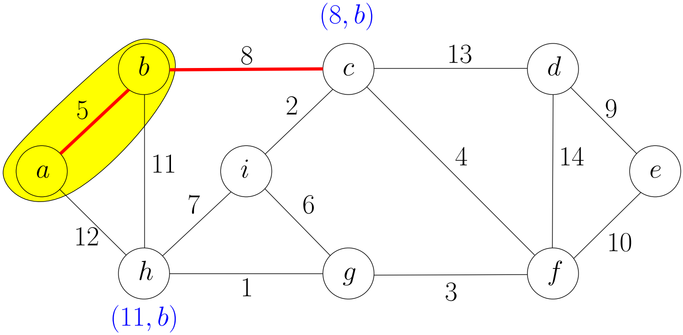
Example



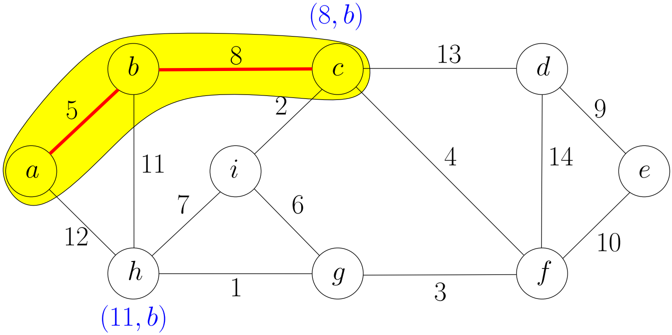
Example



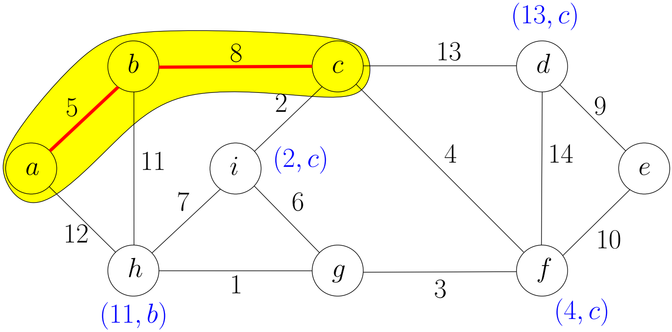
Example



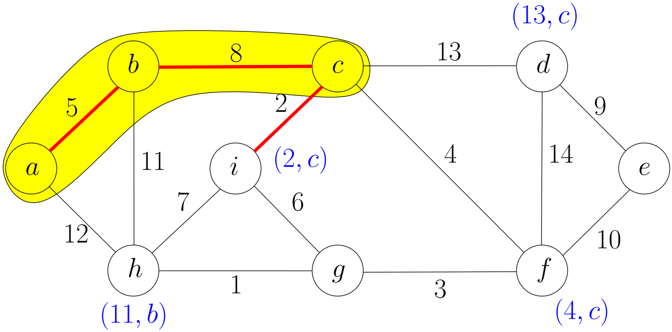
Example



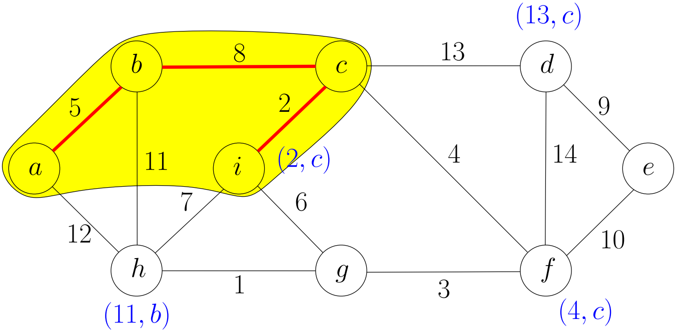
Example



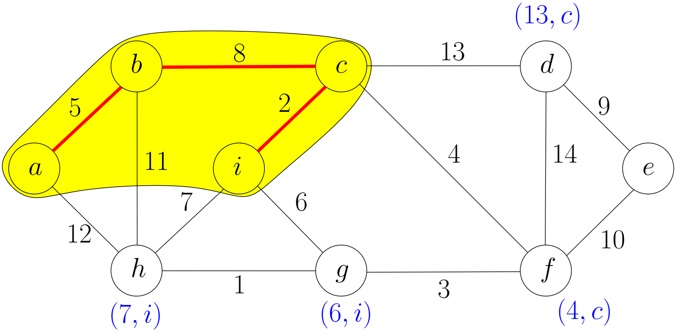
Example



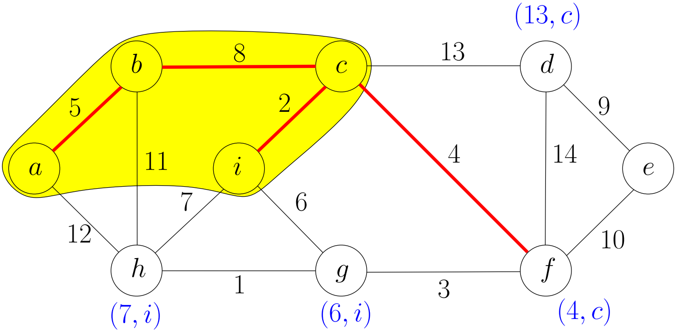
Example



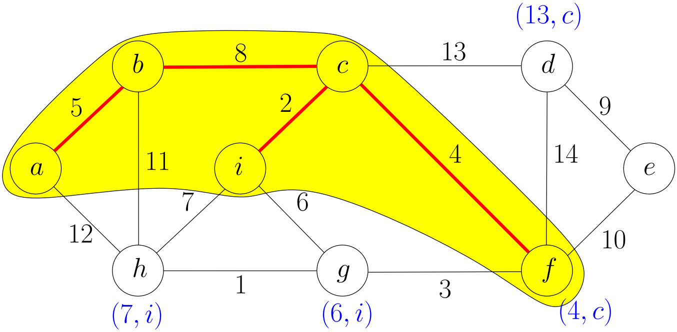
Example



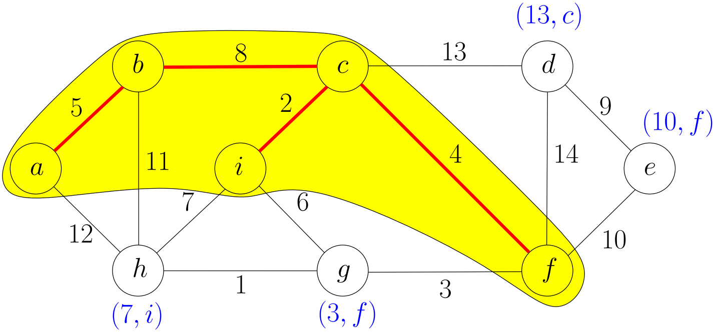
Example



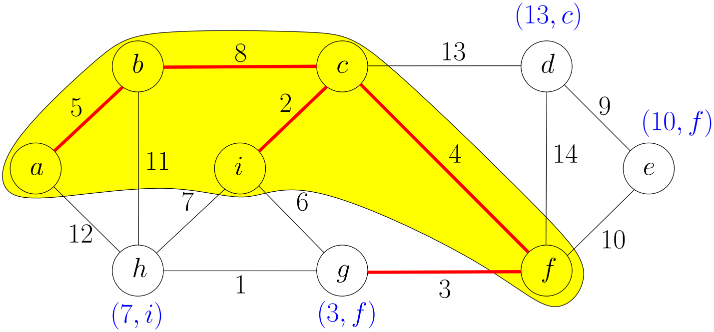
Example



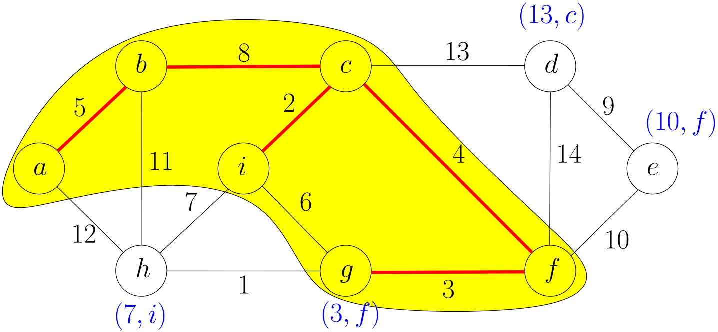
Example



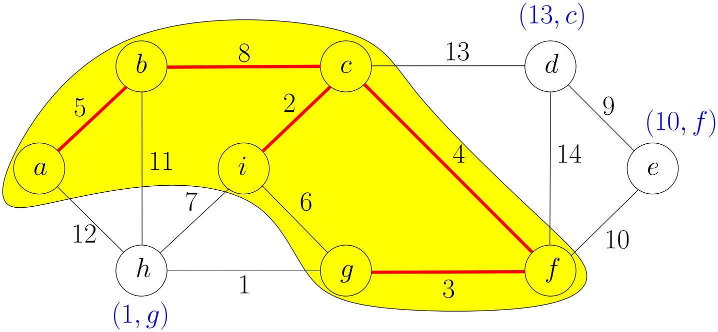
Example



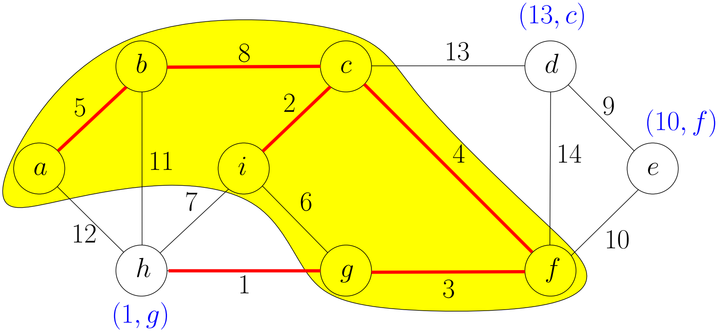
Example



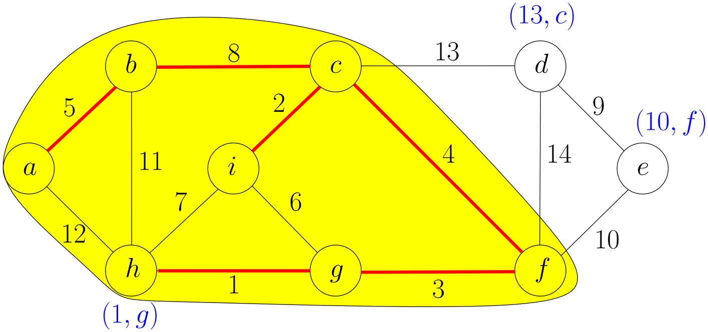
Example



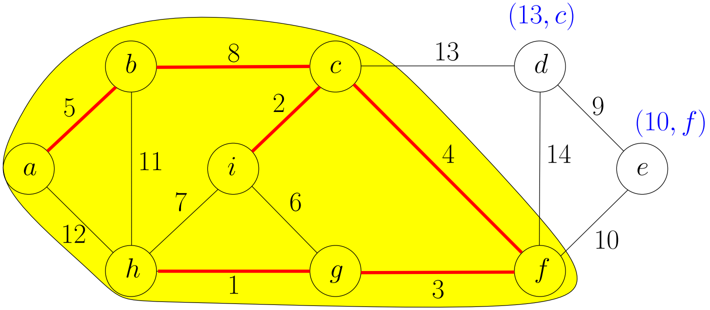
Example



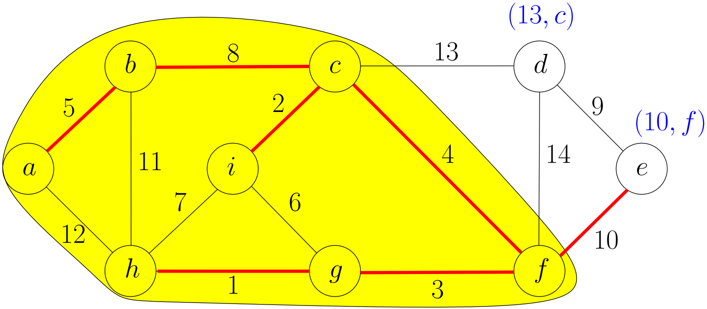
Example



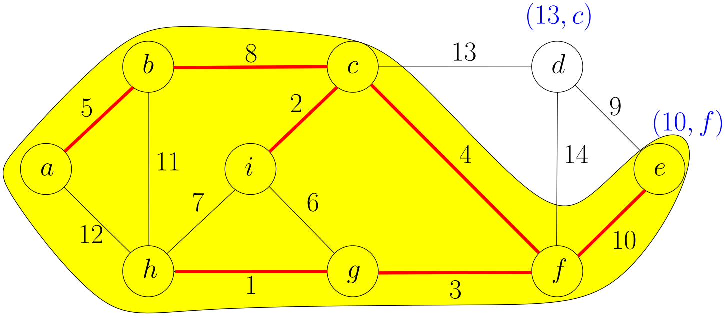
Example



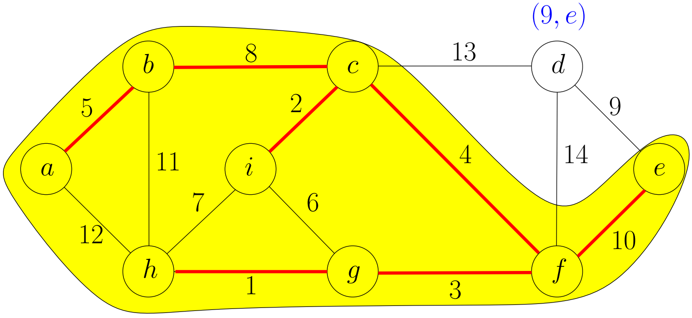
Example



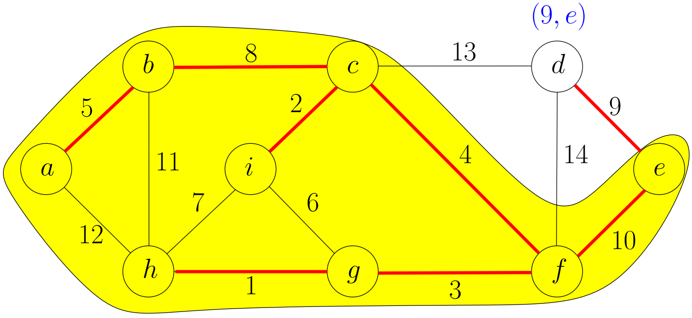
Example



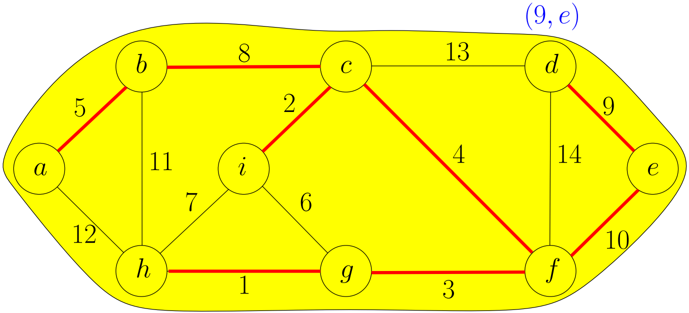
Example



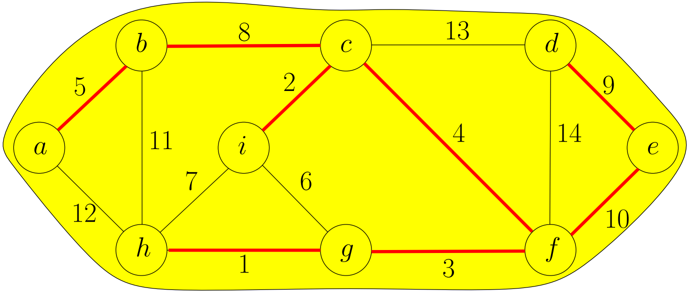
Example



Example



Example



Prim's Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v) \in E} w(u, v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v) \in E} w(u, v)$:
  $(\pi[v], v)$ is the lightest edge between v and S

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values.

Prim's Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S:(u,v) \in E} w(u,v)$:
  the weight of the lightest edge between v and S
- $\pi[v] = \arg\min_{u \in S:(u,v) \in E} w(u,v)$:
  $(\pi[v], v)$ is the lightest edge between v and S

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value — extract_min
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values. — decrease_key

Use a priority queue to support the operations

Def. A priority queue is an abstract data structure that maintains a set U of elements, each with an associated key value, and supports the following operations:

- insert(v, key_value): insert an element v, whose associated key value is key_value.
- decrease_key(v, new_key_value): decrease the key value of an element v in queue to new_key_value
- extract_min$()$: return and remove the element in queue with the smallest key value
- $\cdots$

Prim's Algorithm

MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$

while $S \neq V$ **do**
    $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$
            $\pi[v] \leftarrow u$
return $\big\{(u, \pi[u]) | u \in V \setminus \{s\}\big\}$

Prim's Algorithm Using Priority Queue

MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
$Q \leftarrow$ empty queue, for each $v \in V$: Q.insert$(v, d[v])$
while $S \neq V$ **do**
    $u \leftarrow Q$.extract_min()
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$, Q.decrease_key$(v, d[v])$
            $\pi[v] \leftarrow u$
return $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+ O(m)\times$ (time for decrease_key)

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+ O(m)\times$ (time for decrease_key)

concrete DS	extract_min	decrease_key	overall time
heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Running Time of Prim's Algorithm Using Priority Queue

$O(n)\times$ (time for extract_min) $+\ O(m)\times$ (time for decrease_key)

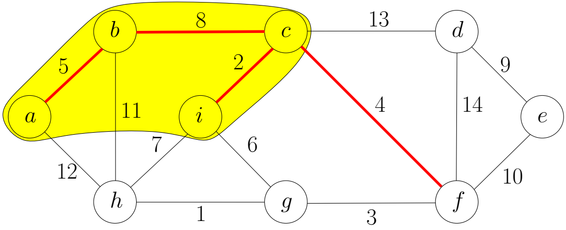
concrete DS	extract_min	decrease_key	overall time
heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Assumption Assume all edge weights are different.

Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.

Assumption Assume all edge weights are different.

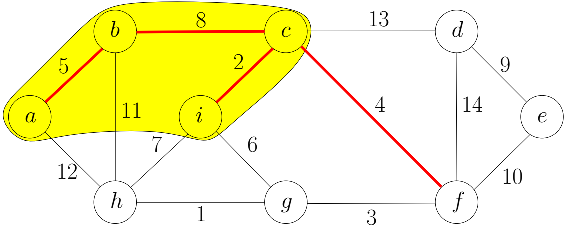
Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.



- (c, f) is in MST because of cut $\big(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\big)$

Assumption Assume all edge weights are different.

Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.



- (c, f) is in MST because of cut $\big(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\big)$
- (i, g) is not in MST because no such cut exists

"Evidence" for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

"Evidence" for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

Exactly one of the following is true:

- There is a cut in which e is the lightest edge
- There is a cycle in which e is the heaviest edge

“Evidence” for $e \in$ MST or $e \notin$ MST

Assumption Assume all edge weights are different.

- $e \in$ MST $\leftrightarrow$ there is a cut in which e is the lightest edge
- $e \notin$ MST $\leftrightarrow$ there is a cycle in which e is the heaviest edge

Exactly one of the following is true:

- There is a cut in which e is the lightest edge
- There is a cycle in which e is the heaviest edge

Thus, the minimum spanning tree is unique with assumption.