CSE 431/531: Algorithm Analysis and Design (Fall 2024)
Graph Algorithms

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm

Outline

Outline

Outline

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)

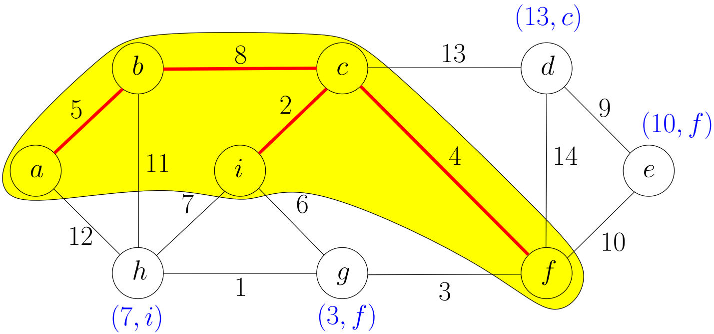
```
S ← {s}, where s is arbitrary vertex in V
F ← ∅
while S ≠ V do
    (u, v) ← lightest edge between S and V \ S,
                        where u ∈ S and v ∈ V \ S
    S ← S ∪ {v}
    F ← F ∪ {(u, v)}
return (V, F)
```

- 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

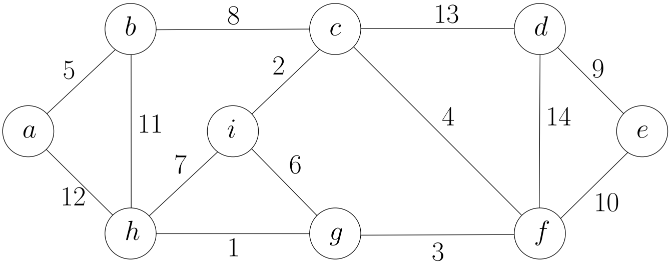
- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to F
- Add u to S, update d and π values.

Prim's Algorithm

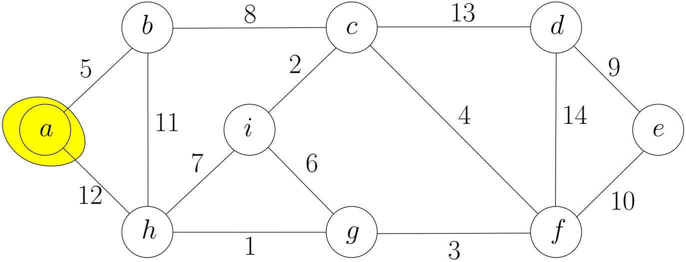
MST-Prim(G, w)

$s \leftarrow$ 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 $\big\{(u, \pi[u]) | u \in V \setminus \{s\}\big\}$

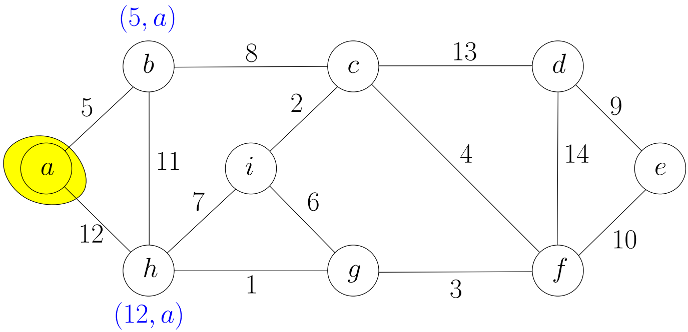
Example



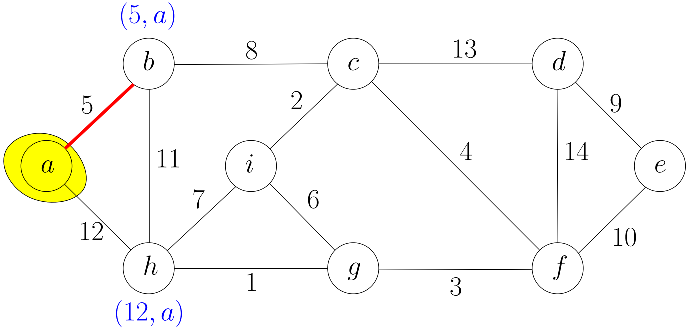
Example



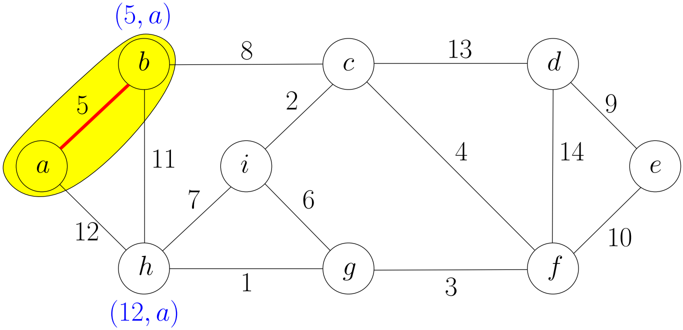
Example



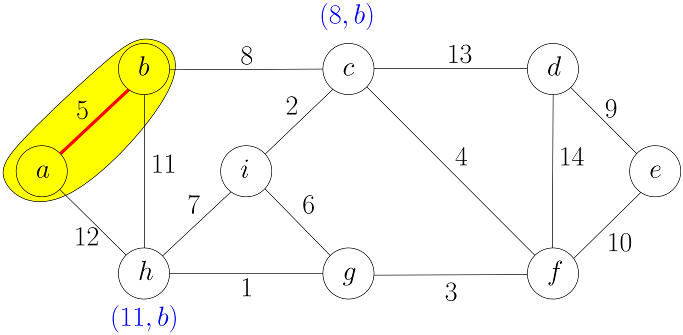
Example



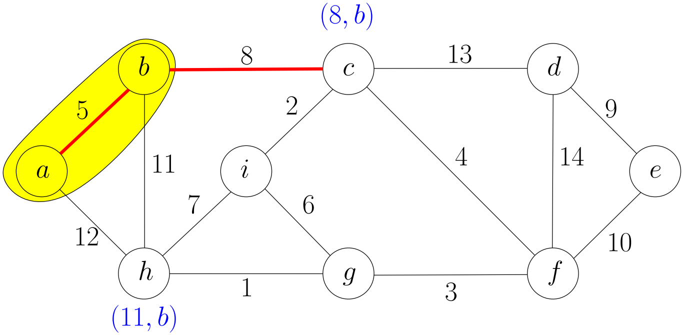
Example



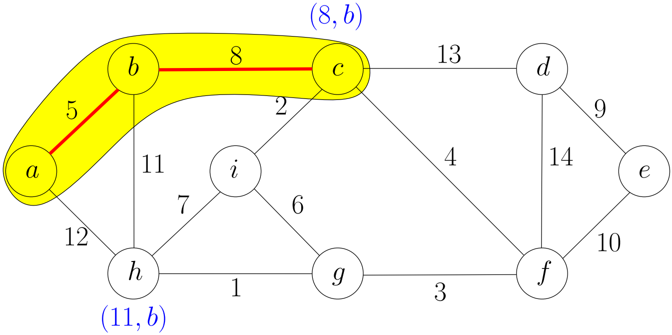
Example



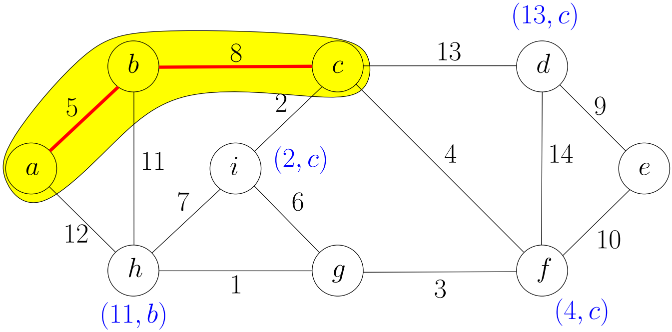
Example



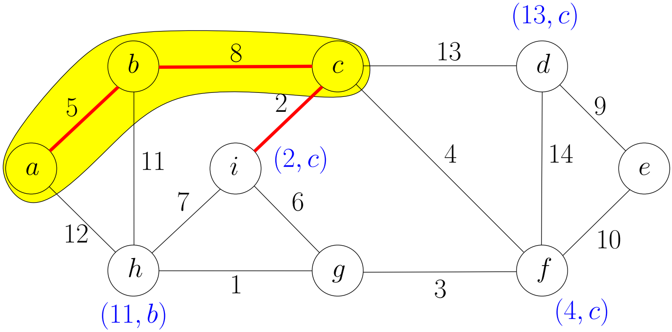
Example



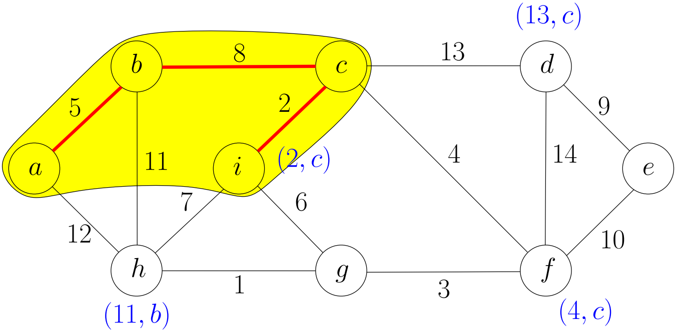
Example



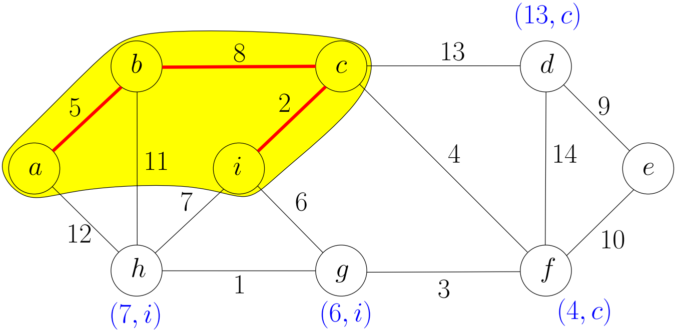
Example



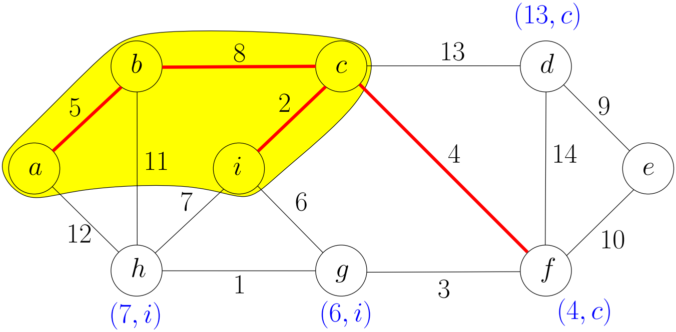
Example



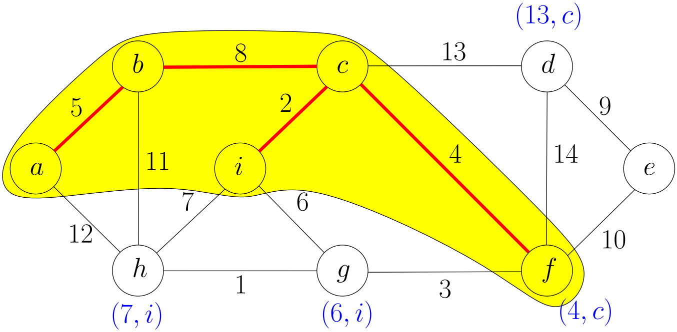
Example



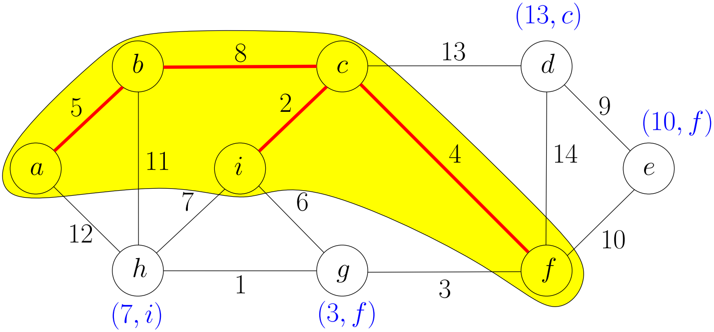
Example



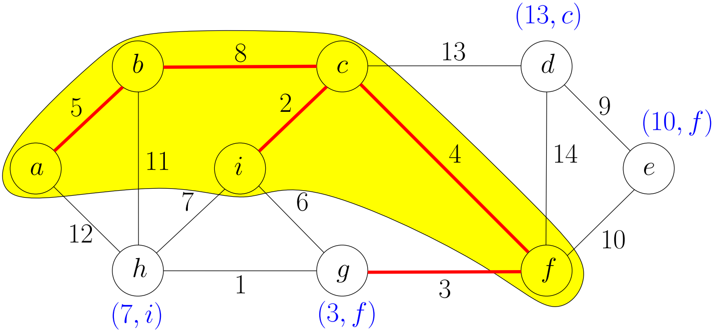
Example



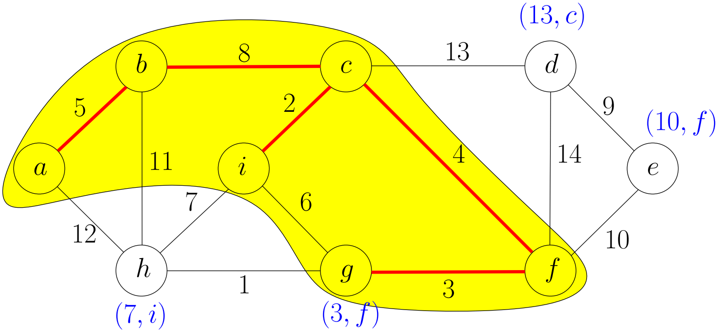
Example



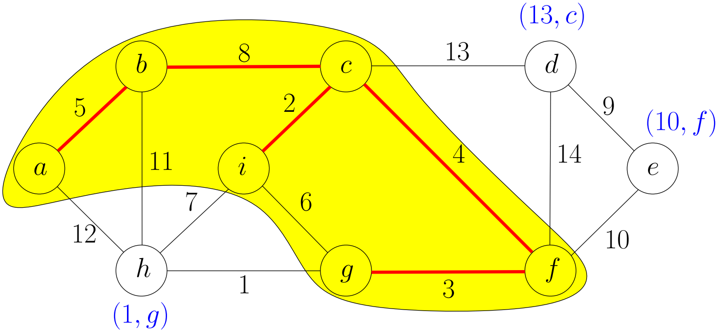
Example



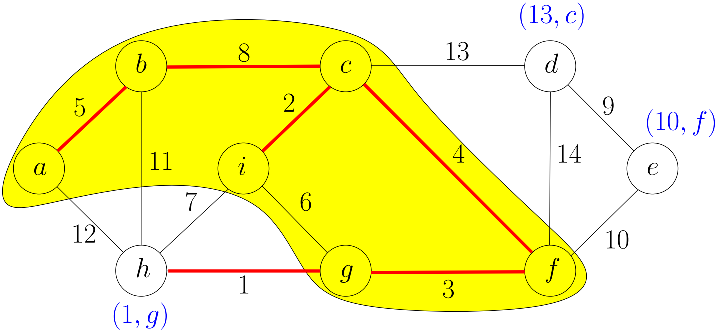
Example



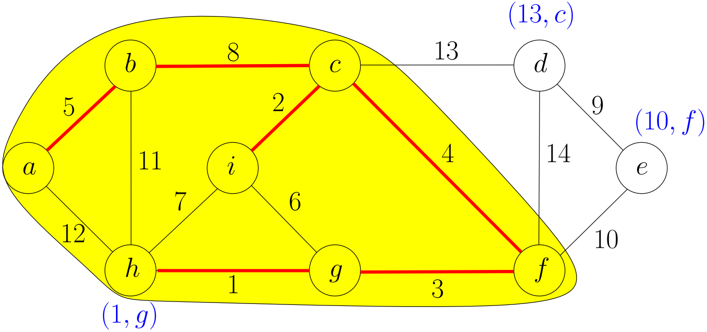
Example



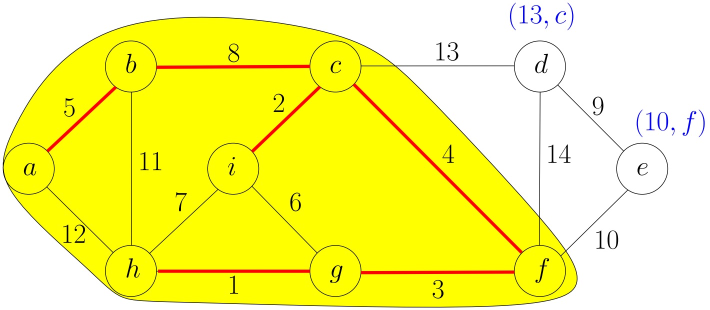
Example



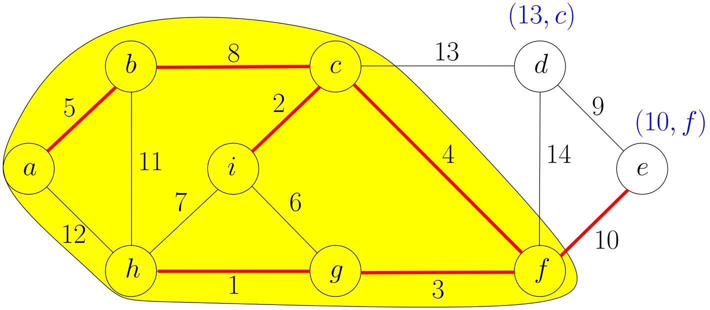
Example



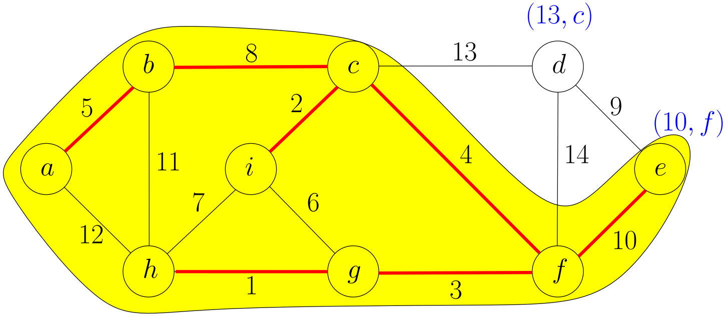
Example



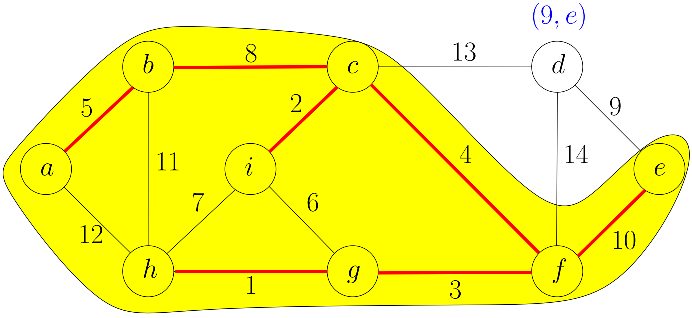
Example



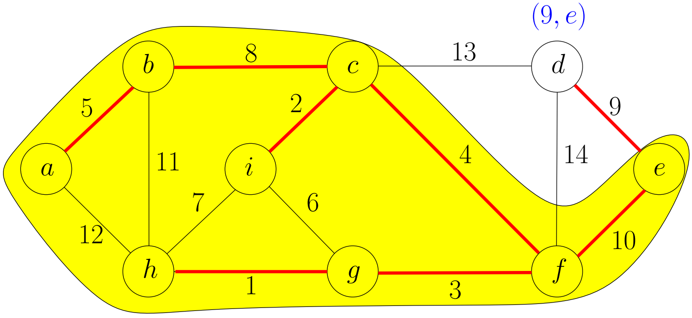
Example



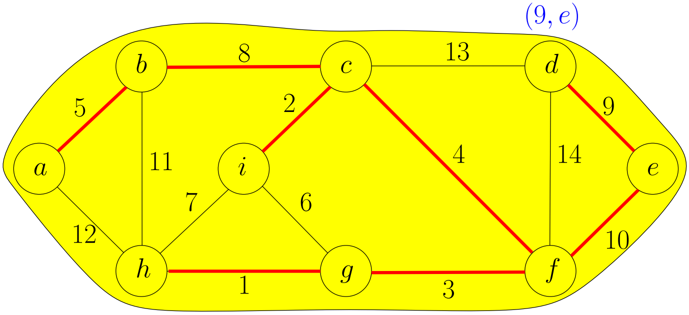
Example



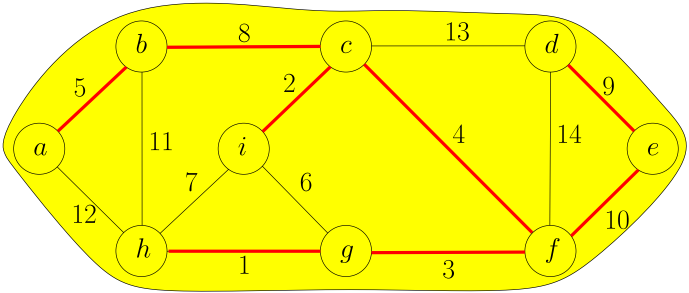
Example



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
- $\cdots$

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 ← arbitrary vertex in G
S ← ∅, d(s) ← 0 and d[v] ← ∞ for every v ∈ V \ {s}
Q ← empty queue, for each v ∈ V: Q.insert(v, d[v])
while S ≠ V do
    u ← Q.extract_min()
    S ← S ∪ {u}
    for each v ∈ V \ S such that (u, v) ∈ E do
        if w(u, v) < d[v] then
            d[v] ← w(u, v), Q.decrease_key(v, d[v])
            π[v] ← u
return {(u, π[u]) | u ∈ V \ {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)

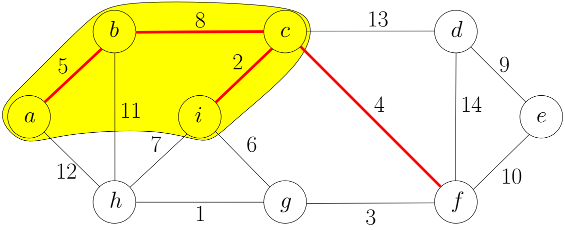
concrete DS	extract_min	decrease_key	overall time
heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

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.

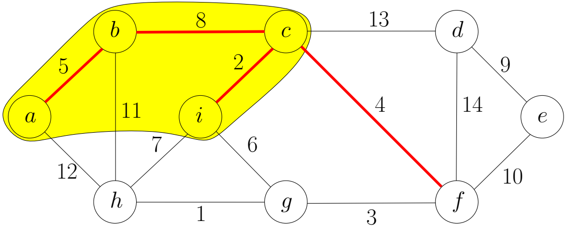
Lemma (u, v) is in MST, if and only if there exists a cut $(U, V \setminus U)$, such that (u, v) is the lightest edge between U and $V \setminus U$.



- (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.