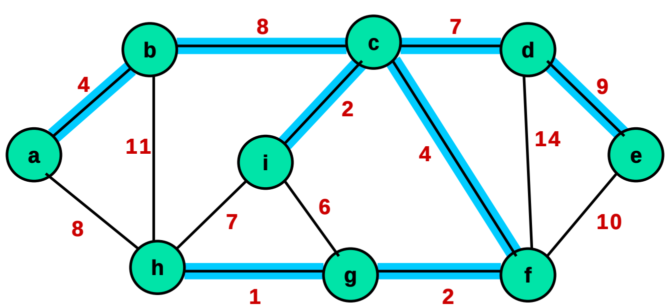
Minimum Spanning Tree Algorithms

Minimum Spanning Tree

- **G=(V,E): connected and undirected**
  w: E→R,   weight function



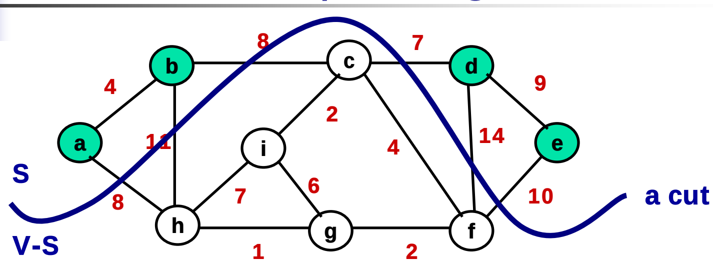
Minimum Spanning Tree

- **Let *A* be a subset of some minimum spanning tree if $A \cup \{(u,v)\}$ is also a subset of a MST, then we call *(u,v)* a safe edge of *A***

Generic-MST(G, w)
{
  $A \leftarrow \phi$
  while A does not form a spanning tree
    do find an edge (u, v) **that is safe**
      for A**;**
      $A \leftarrow A \cup \{(u,v)\}$;

  return A
}

Minimum Spanning Tree



- A **cut** (S, V-S) of an undirected graph G=(V, E) is a partition of V
- An edge $(u,v) \in E$ crosses the cut (S, V-S) if one of its endpoints is in S and the other is in V-S
- A cut respects the set A of edges if no edge in A crosses the cut
- An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut

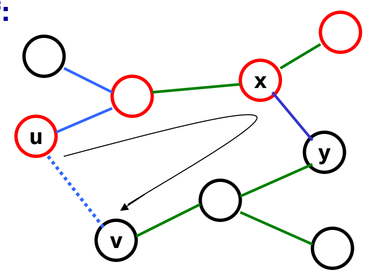
What is the light edge in the above graph ?

Minimum Spanning Tree

- **Thm1:**

  G=(V,E): connected, undirected
  w: real-valued weight function on E
  A: a subset of E and is included in some MST
  (S, V-S): any cut of G and respects A
  (u,v): a light edge crossing (S, V-S)
  Then (u,v) is safe for A.

pf:



S : {O}
V-S: {O}
A: { - }

T' = T – {(x,y)} U {(u,v)}
original MST

p5.

Minimum Spanning Tree

$w(T') = w(T) - w(x,y) + w(u,v)$
$\quad\quad = w(T)$

Thus, T' is a MST

$A \subseteq T$, and $(x,y) \notin A$
$A \cup \{(u,v)\} \subseteq T'$
$\Rightarrow$ **T' is a MST and (u, v) is safe for A**

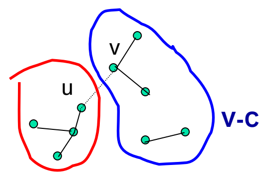
p6.

Minimum Spanning Tree

- **Cor2:**

  G=(V,E): connected, undirected
  w = real-valued weight function
  $\mathbf{A} \subseteq \mathbf{E}$ **and A is in some MST**
  C: a connected component in G_A=(V,A)
  if (u,v) is a light edge connecting C to some other component in G_A, then (u,v) is safe for A

  Pf: The cut(C, V-C) respects A, and (u,v) is therefore a light edge for this cut



C

p7.

- **Disjoint sets**

$$S = \{S_1, S_2, S_3, \ldots\ldots, S_n\}, S_i \cap S_j = \phi, \textbf{ if } i \neq j$$

Operations:

Make-Set(x) $S \leftarrow S \cup \{\{x\}\}$
Union(S_i, S_j) $S \leftarrow S - \{S_i, S_j\} \cup \{S_i \cup S_j\}$
Find-Set(x) **return** $S_i \in S$ **s.t.** $x \in S_i$

p8.

- Eg. Minimum spanning tree
  $G=(V,E)$: connected, undirected, edge-weighted graph
  $w: E \rightarrow R$

Kruskals' algorithm:

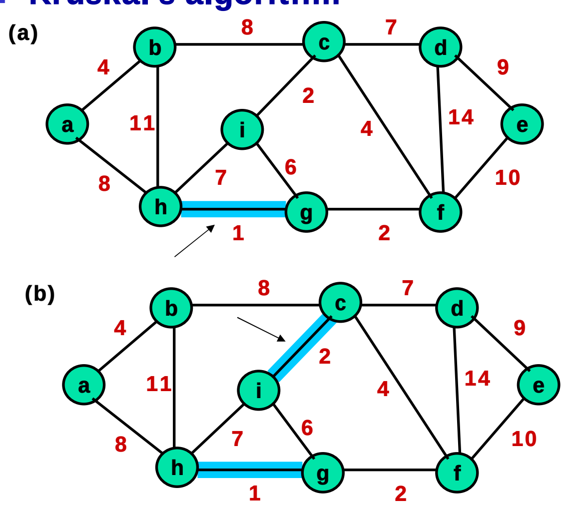
T=ϕ
for each $v \in V$
  do Make-Set(v**)**

sort E by increasing edge weight w

for each $(u,v) \in E$ **(in sorted order)**
  do if Find-Set(u**)** $\neq$ **Find-Set(**v**)**
    then T $\leftarrow$ **T**$\cup\{(u,v)\}$
      Union(Find-Set(u**), Find-Set(**v**))**

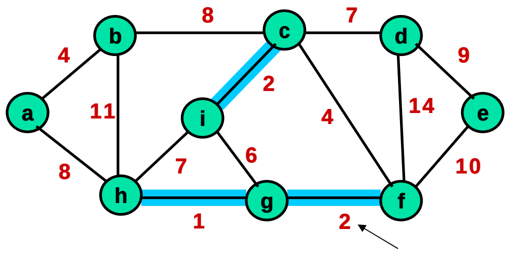
p9.

- **Kruskal's algorithm**

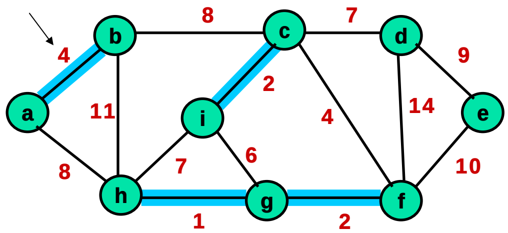


p10.

(c)

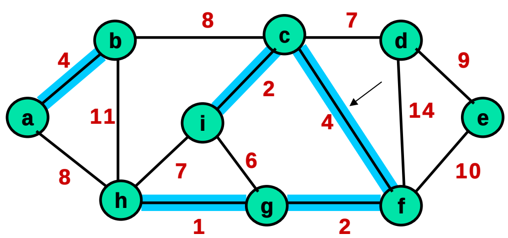

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

(d)

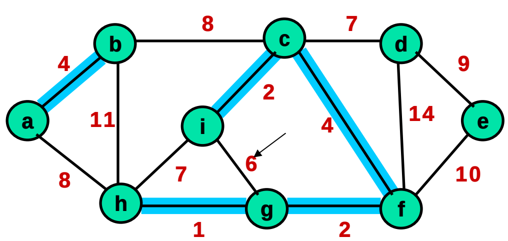

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

p11.

(e)

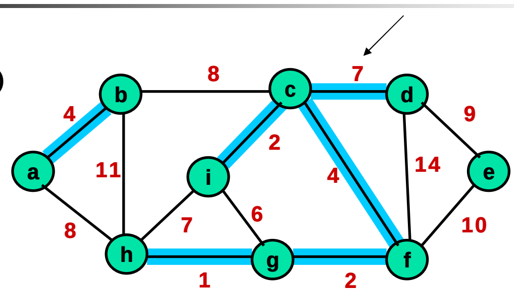

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

(f)

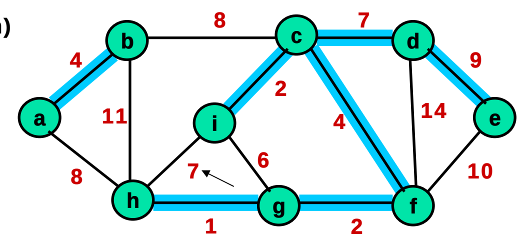

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

p12.

(g)

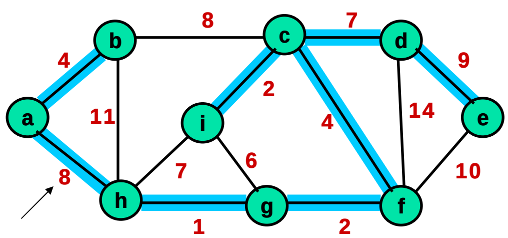

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

(h)

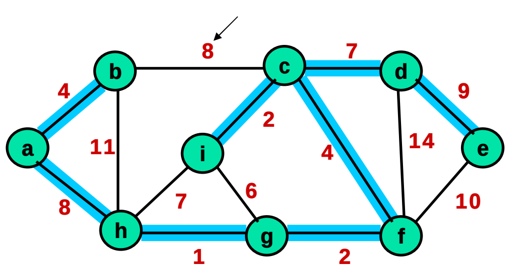

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

p13.

(i)

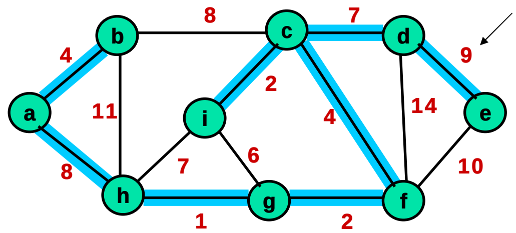

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

(j)

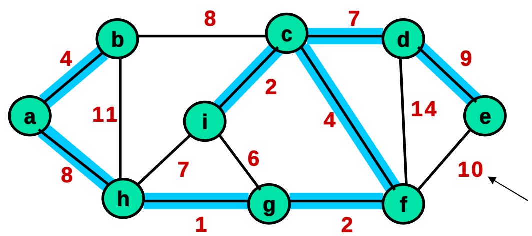

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

p14.

(k)

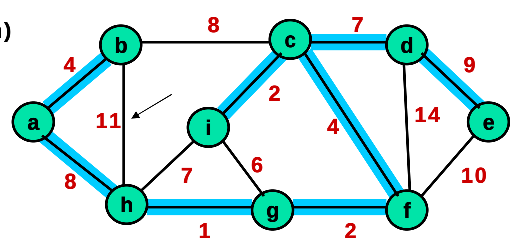

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

(l)

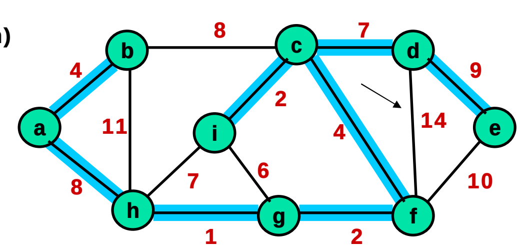

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

p15.

(m)


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

(n)


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

p16.

- **Prim's algorithm:**

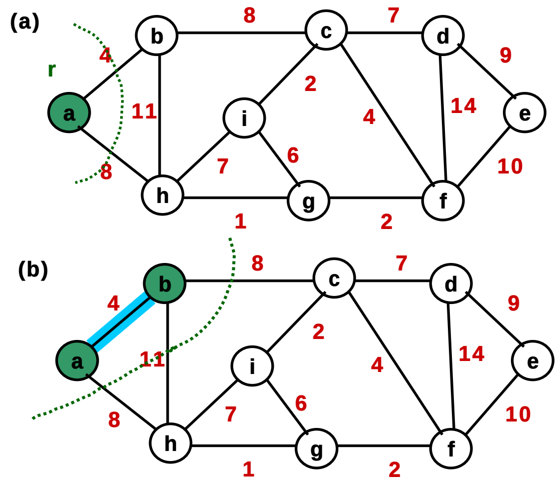
```
MTS-Prim(G, w, r)
{
         Q ← V[G]          /* Q: priority queue */
         for each u ∈ Q
O(V)         do key[u] ← ∞
                π[u] ← NIL
         key[r] ← 0

         while Q ≠ ϕ
                do u ← Extract-Min(Q)            lg V
O(V lg V)           for each v ∈ adj[u]
                        do if v ∈ Q and w(u,v) < key[v]
O(E)                        then π[v] ← u
                                 key[v] ← w(u,v)
}
                              O(lg V), Decrease-key involves
```

p17.

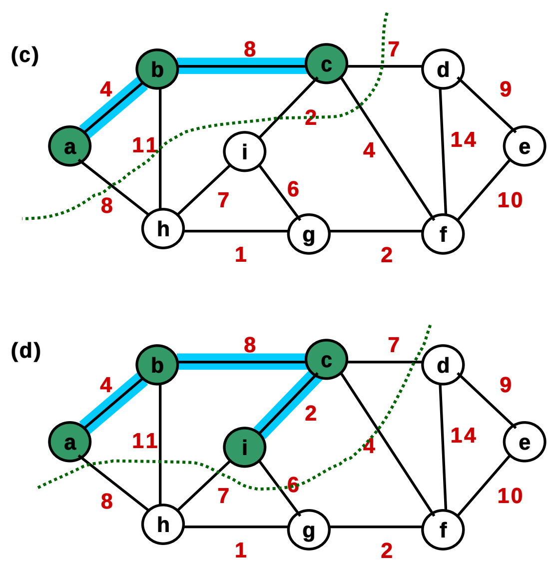
- Analysis
  - **Binary heap: O(V lg V + E lg V)**
    = O(E lg V)
  - **Fibonacci heap:**
    Decrease-key: O(1) amortized time
    O(V lg V + E)

p18.

- **Prim's algorithm**

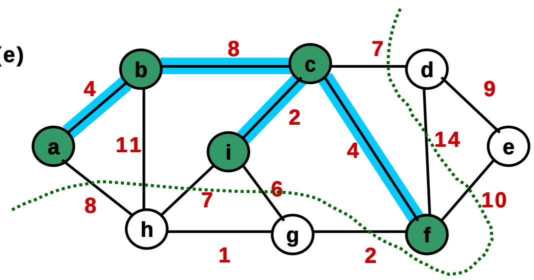


p19.



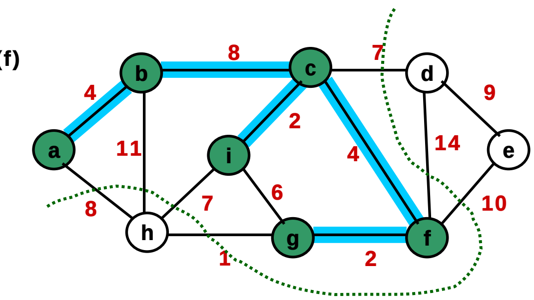
p20.

(e)



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

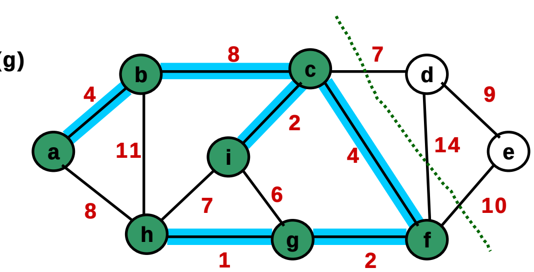
(f)



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

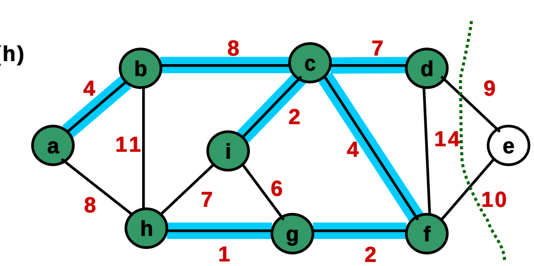
p21.

(g)



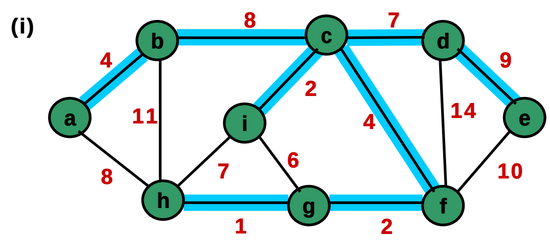
b
8
c
7
d
4
9
2
a
11
i
4
14
e
6
7
8
10
h
g
f
1
2

(h)



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

p22.



p23.