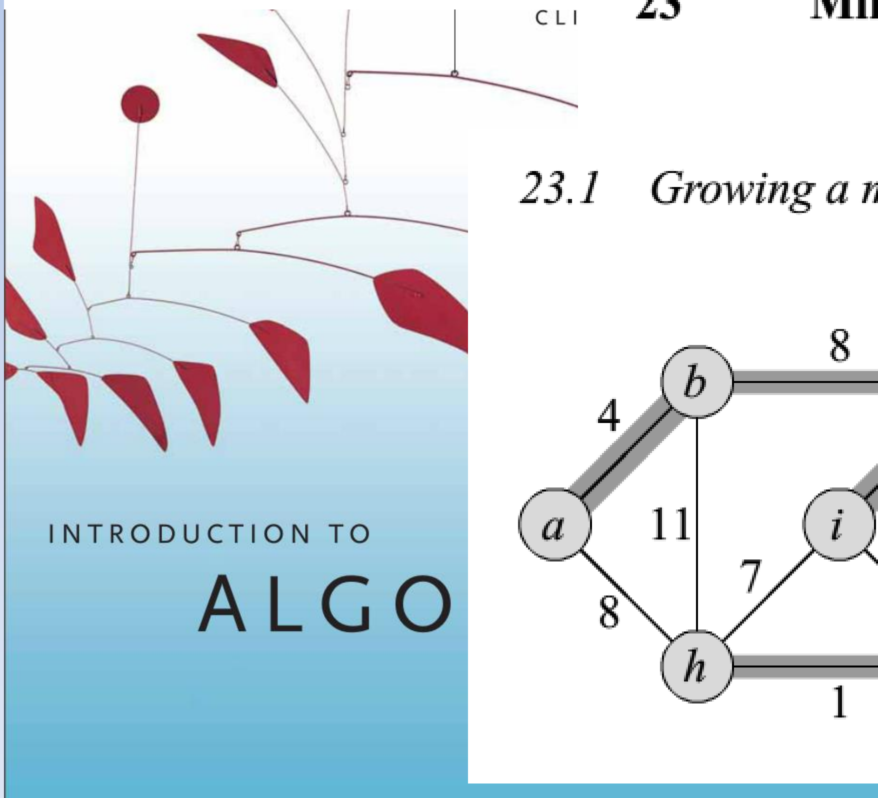
Design and Analysis of Algorithms

Section VI : Graph Algorithms

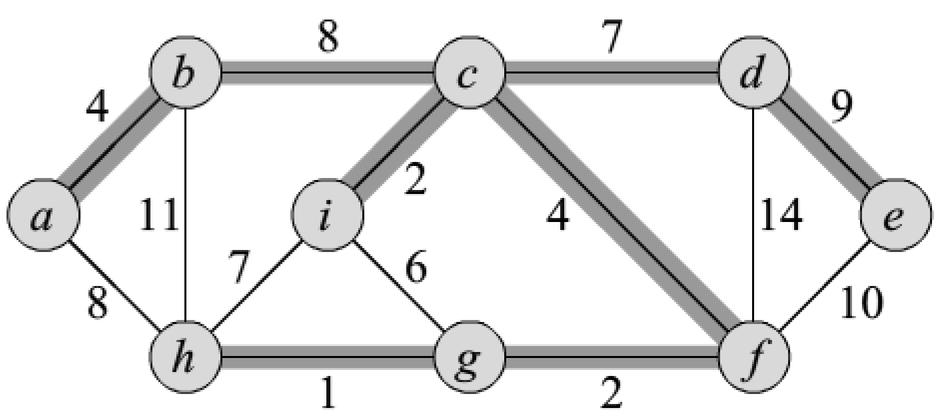
Chapter 23: Minimum Spanning Trees

VI Graph Algorithms



23 Minimum Spanning Trees

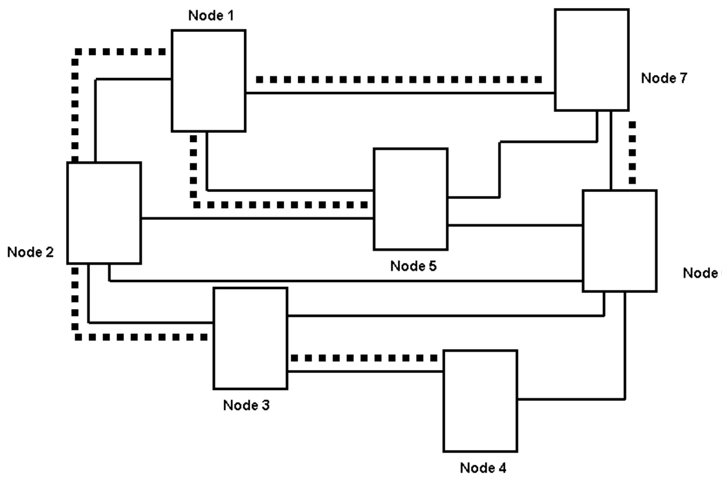
23.1 Growing a minimum spanning tree

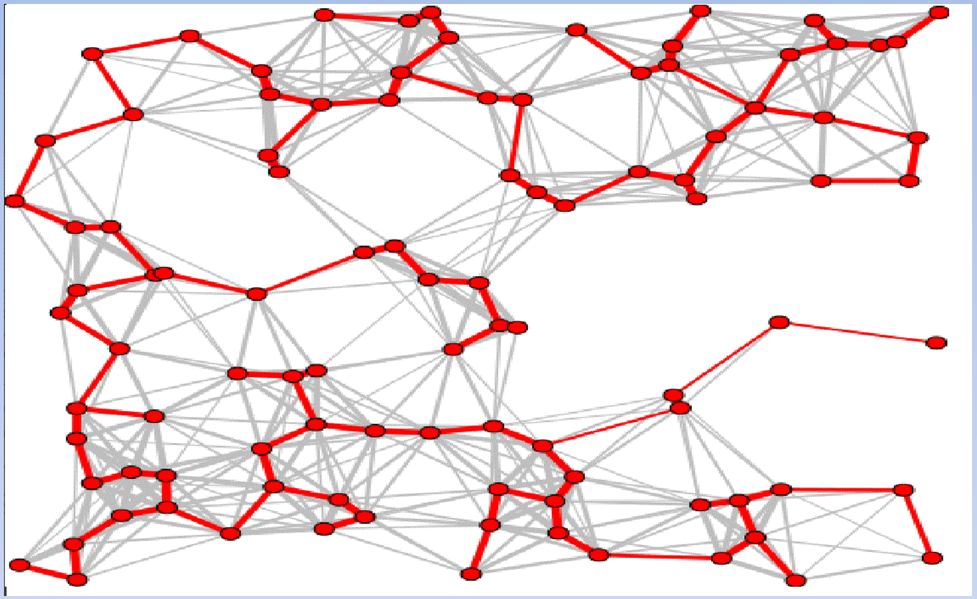


Minimum Spanning Trees: Why?

- Many Applications:
  - Circuit Wiring
  - Connect all pins with minimum amount of wire.

Figure 8 Minimum Spanning Tree





- Many Applications:
  - Minimizing power cabling needed to connect cities
  - Connect all cities for power with minimal amount of cabling.

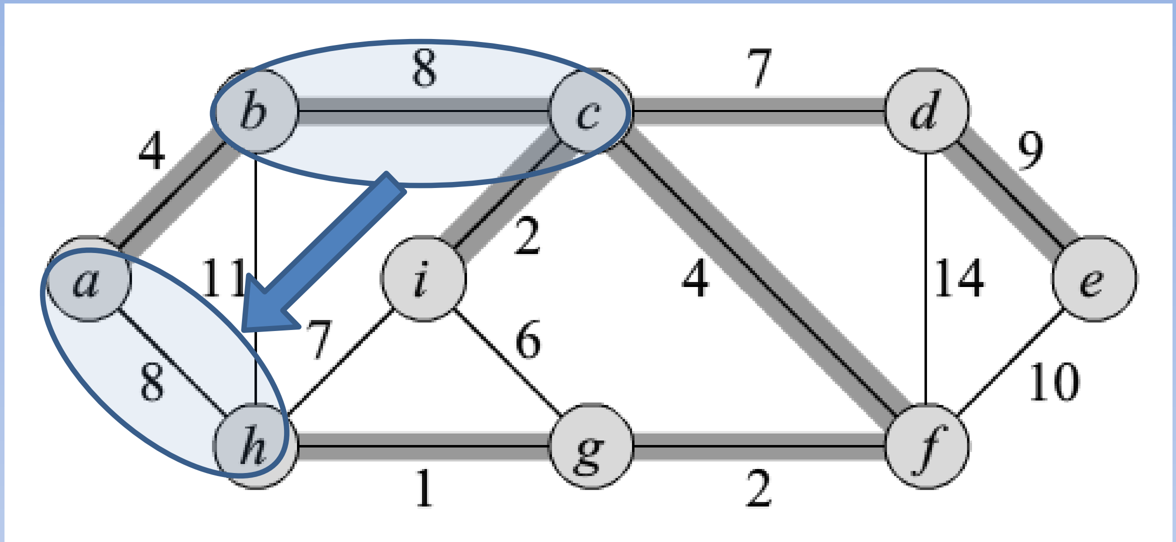
Model Problems w/ Graph

- Given Graph G=(V, E)
  - V is set of pints
  - E is possible interconnections
- Weight Function w(u,v) specifying cost to connect u & v.
- We wish to find an acyclic subset $T \subseteq E$ connecting all vertices and minimizing total weight:

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

- Since T is acyclic and connects all vertices, it must form a tree.
- Since it spans the graph, we call it a Spanning Tree.

MST



- MST cost of 37
- (b, c) => (a, h)
  - Another Minimum-Spanning Tree

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

Initialization: After line 1, the set A trivially satisfies the loop invariant.

Maintenance: The loop in lines 2–4 maintains the invariant by adding only safe edges.

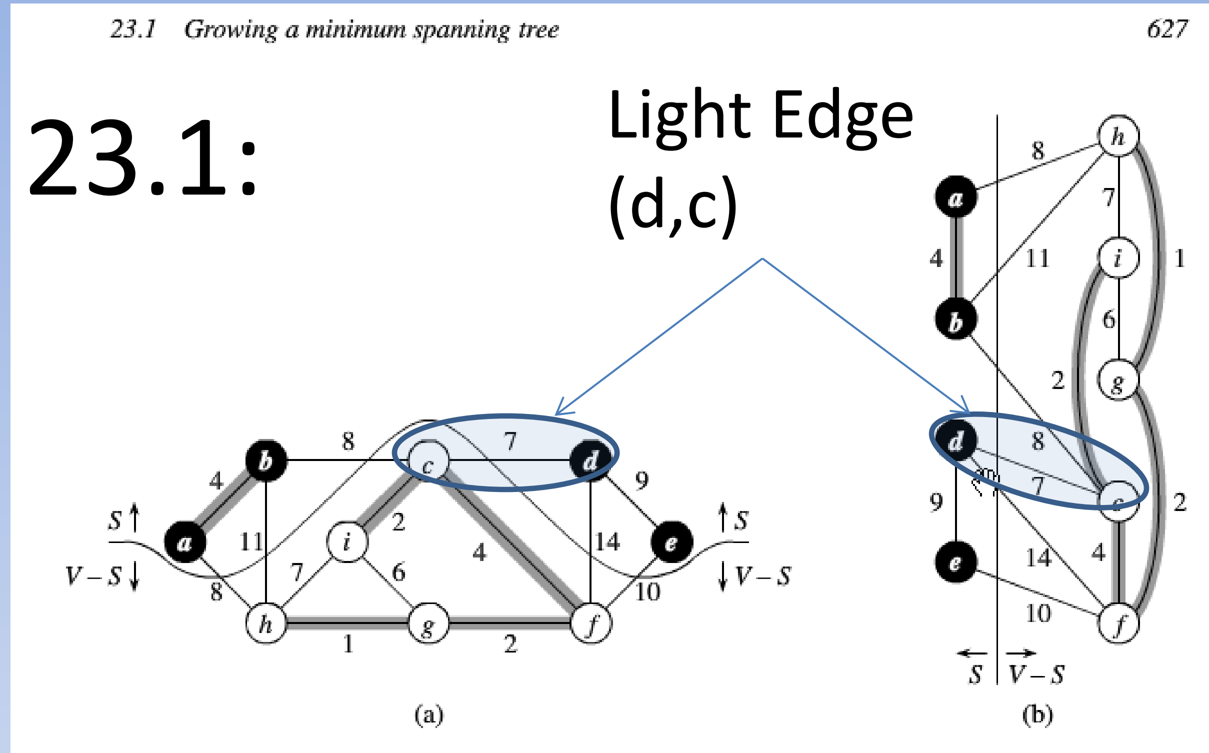
Termination: All edges added to A are in a minimum spanning tree, and so the set A returned in line 5 must be a minimum spanning tree.

- Problem is finding SAFE EDGE!

Finding Safe Edges
First: Some Definitions (Of Course!)

- A **cut** (S, V – S) of an undirected Graph G=(V,E) is a partition of V.
- Edge (u,v) ∈ E **crosses** the cut if one if its endpoints is in S and the other is in (V-S).
- A cut **respects** a 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 minimum of any edge crossing the cut!

Theorem 23.1:

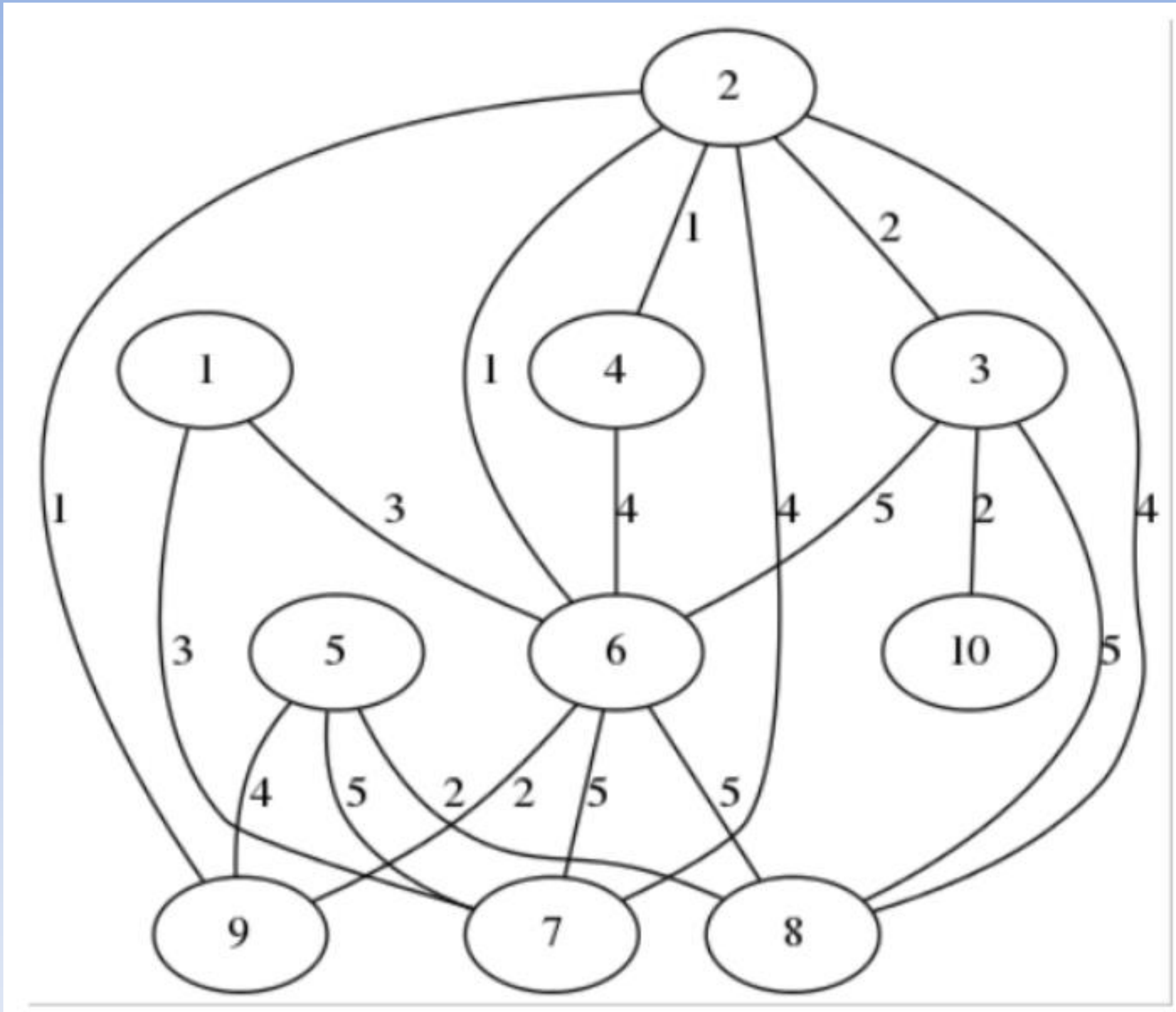


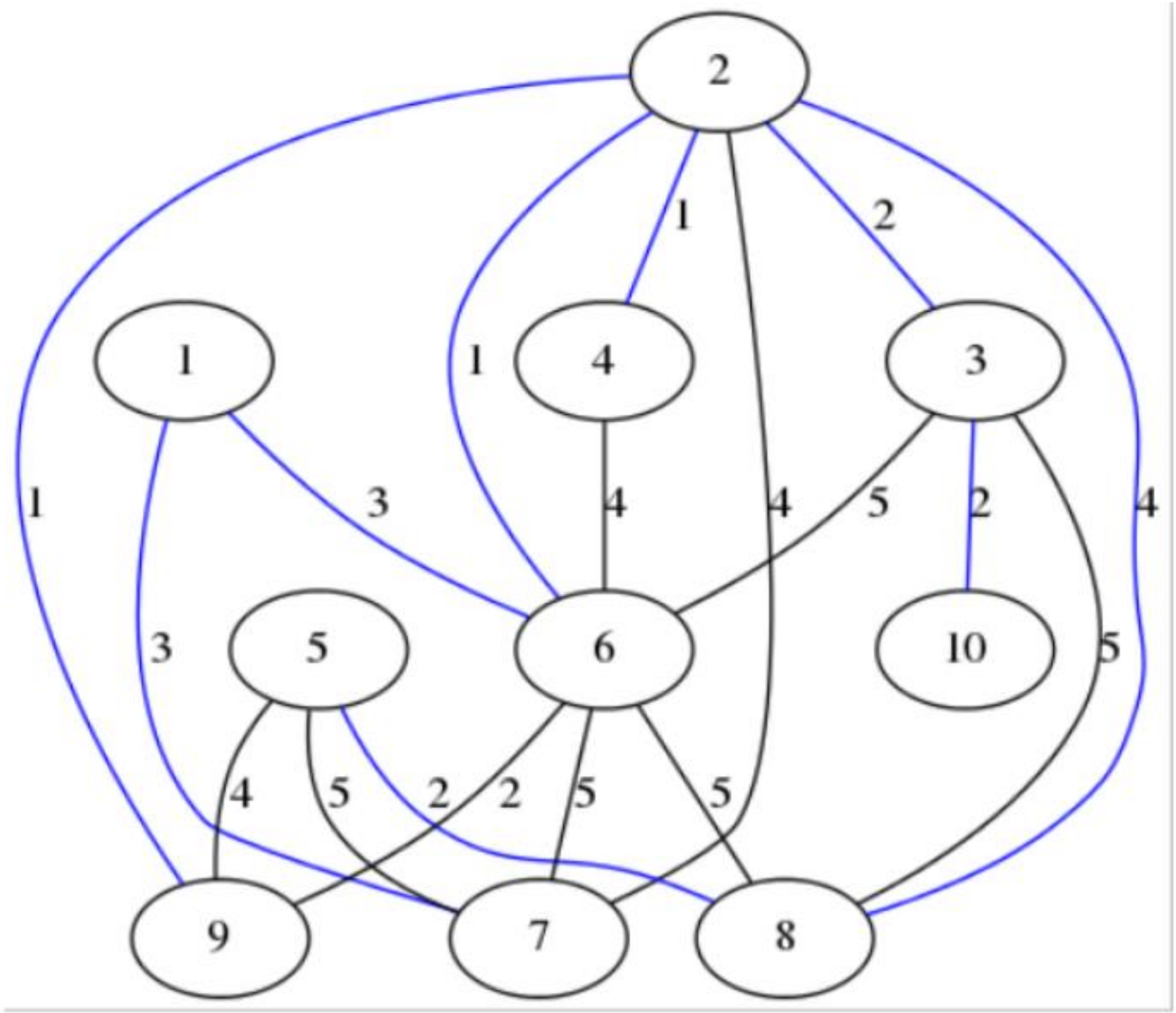
- Let G=(V,E) be a connected, undirected graph with real-valued weight function w defined on E.
- Let A be a subset of E that is included in some minimum spanning tree for G
- Let (S, V-S) be any cut of G that respects A
- Let (u,v) be a light edge crossing (S, V-S).
- THEN: (u,v) is safe for A.

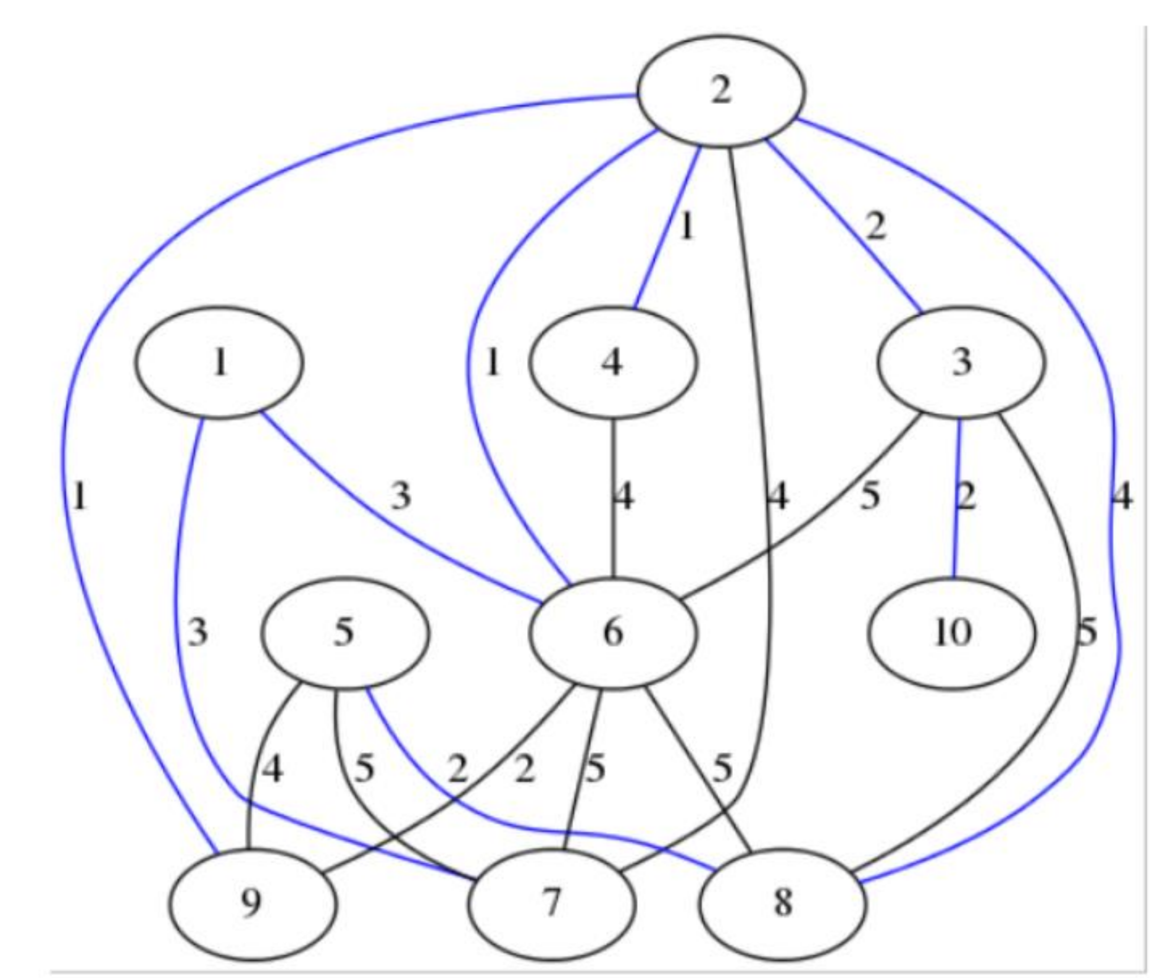
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

- Depends upon Union-Find
  - Chapter 21

21 Data Structures for Disjoint Sets


2
1
4
3
5
6
10
9
7
8
1
2
1
3
4
4
5
2
4
1
3
5
4
5
2
2
5
5


2
1
4
3
5
6
10
9
7
8




```
print "Kruskal: ", sorted([(e, W[e]) for e in AKruskal],key=lambda x:x[1]), np.sum([W[e] for e in
AKruskal])
print "Prim: ", sorted([(e, W[e]) for e in APrim],key=lambda x:x[1]), np.sum([W[e] for e in APrim])
```

```
Kruskal:  [((2, 4), 1), ((2, 6), 1), ((2, 9), 1), ((2, 3), 2), ((3, 10), 2), ((5, 8), 2), ((1, 6), 3),
 ((1, 7), 3), ((2, 8), 4)] 19
Prim:  [((2, 4), 1), ((2, 6), 1), ((2, 9), 1), ((2, 3), 2), ((3, 10), 2), ((5, 8), 2), ((1, 6), 3), ((1,
 7), 3), ((5, 9), 4)] 19
```

Kruskal's algorithm in words

- Procedure:
  - Sort all edges into non-decreasing order
  - Initially each node is in its own tree
  - For each edge in the sorted list
    - If the edge connects two separate trees, then
      - join the two trees together with that edge

Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

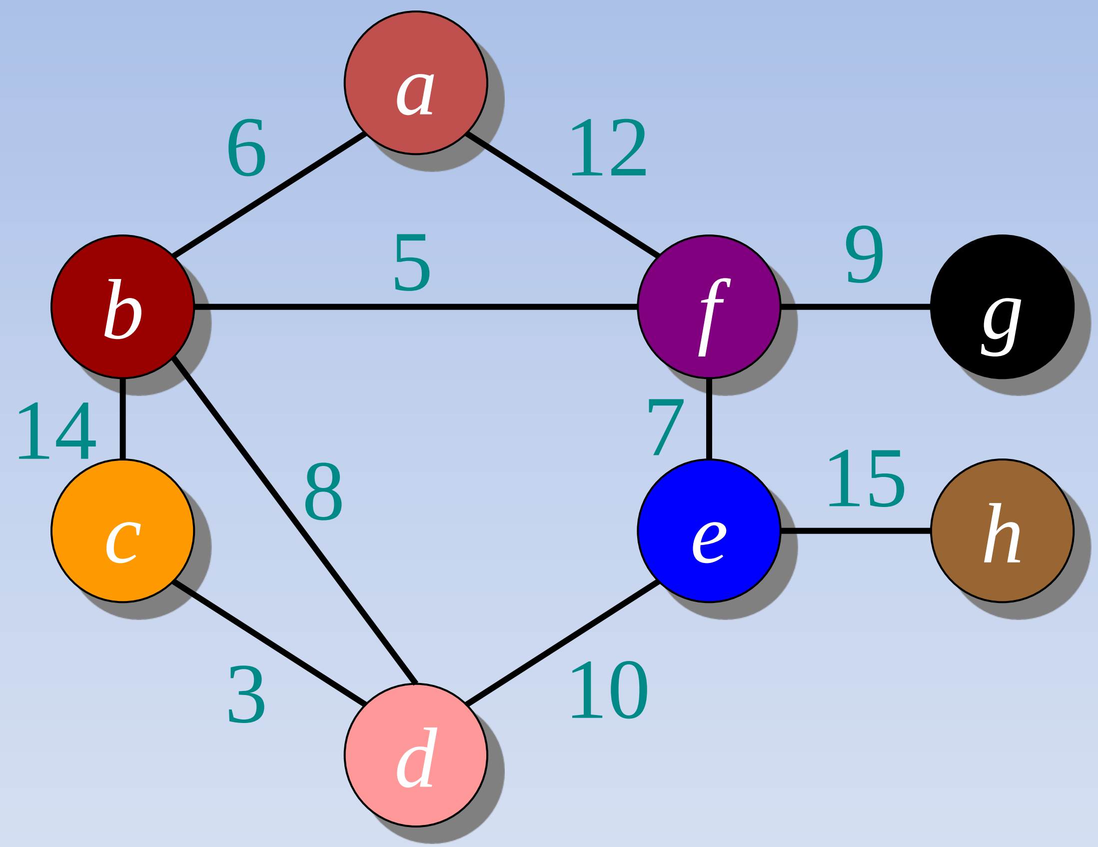
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

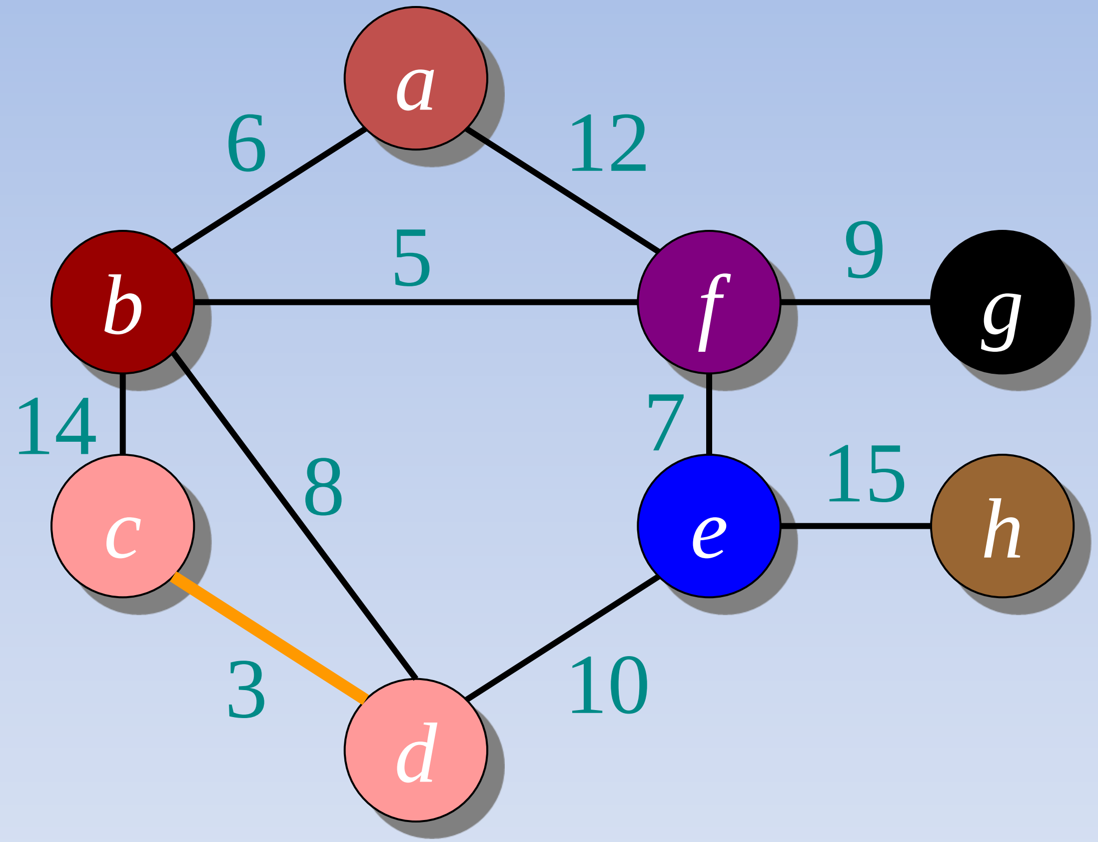
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

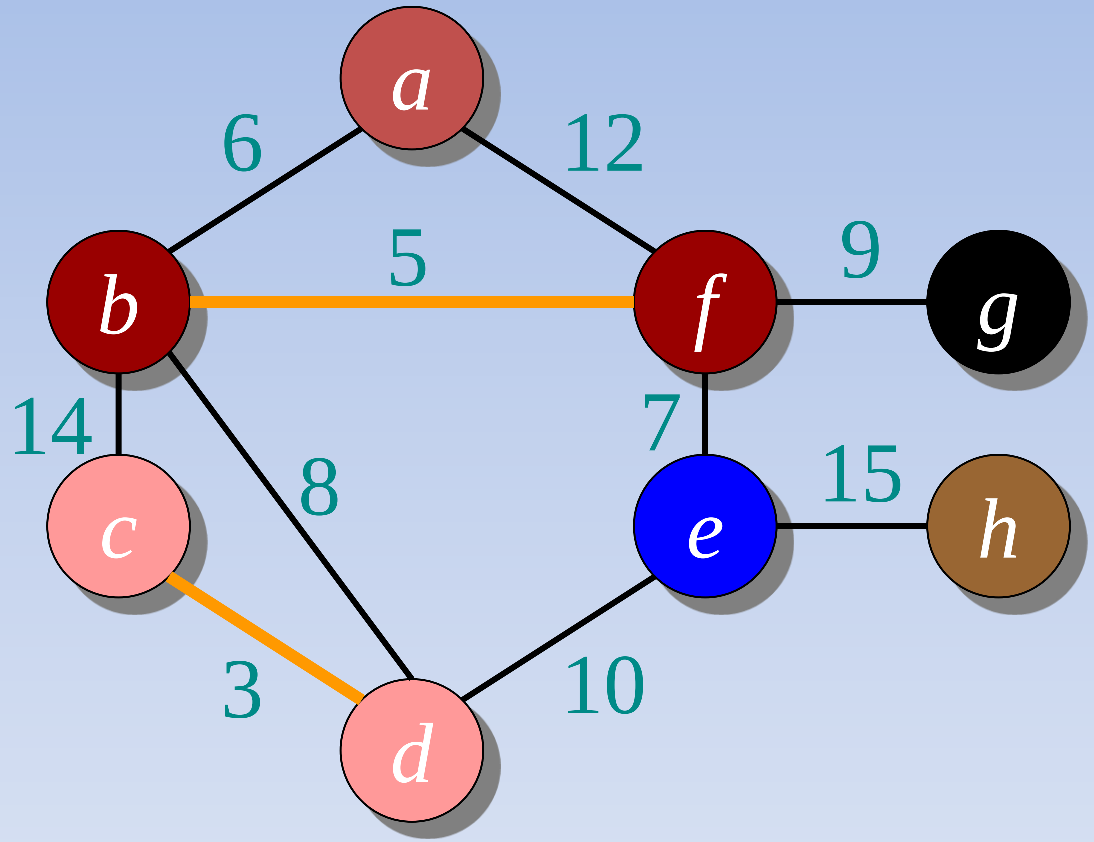
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

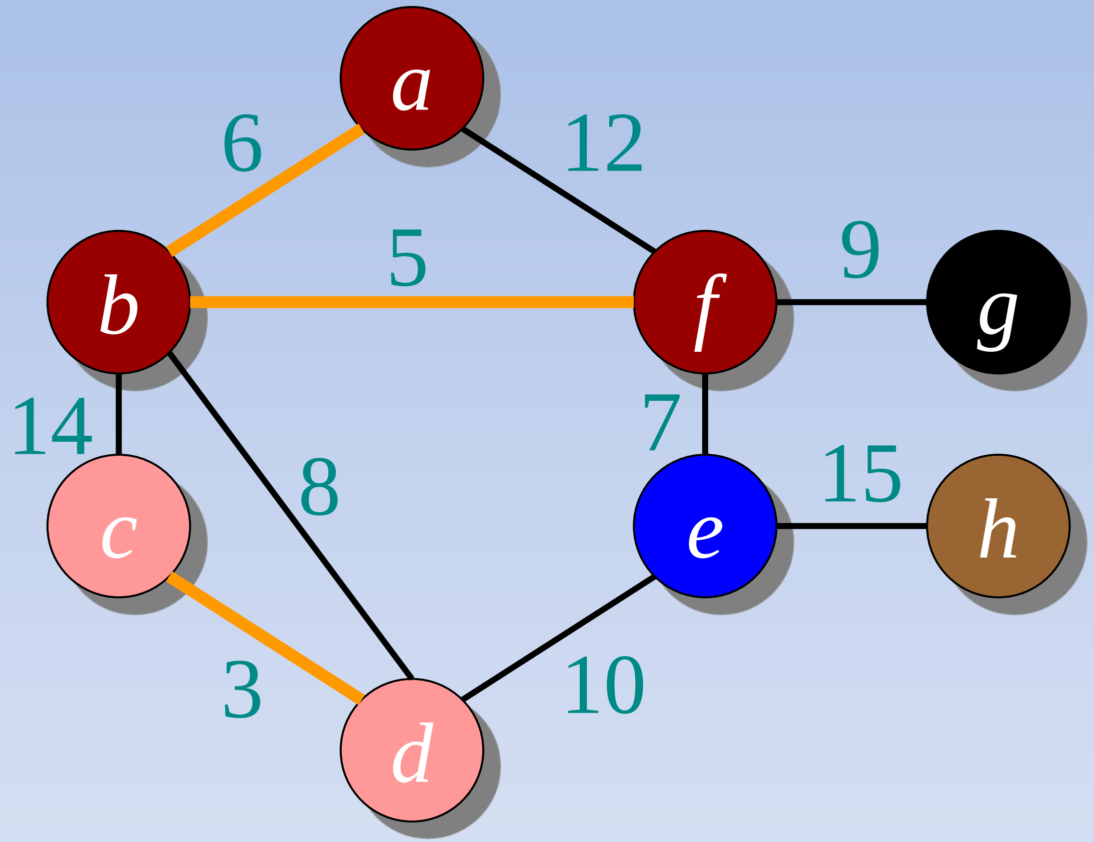
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

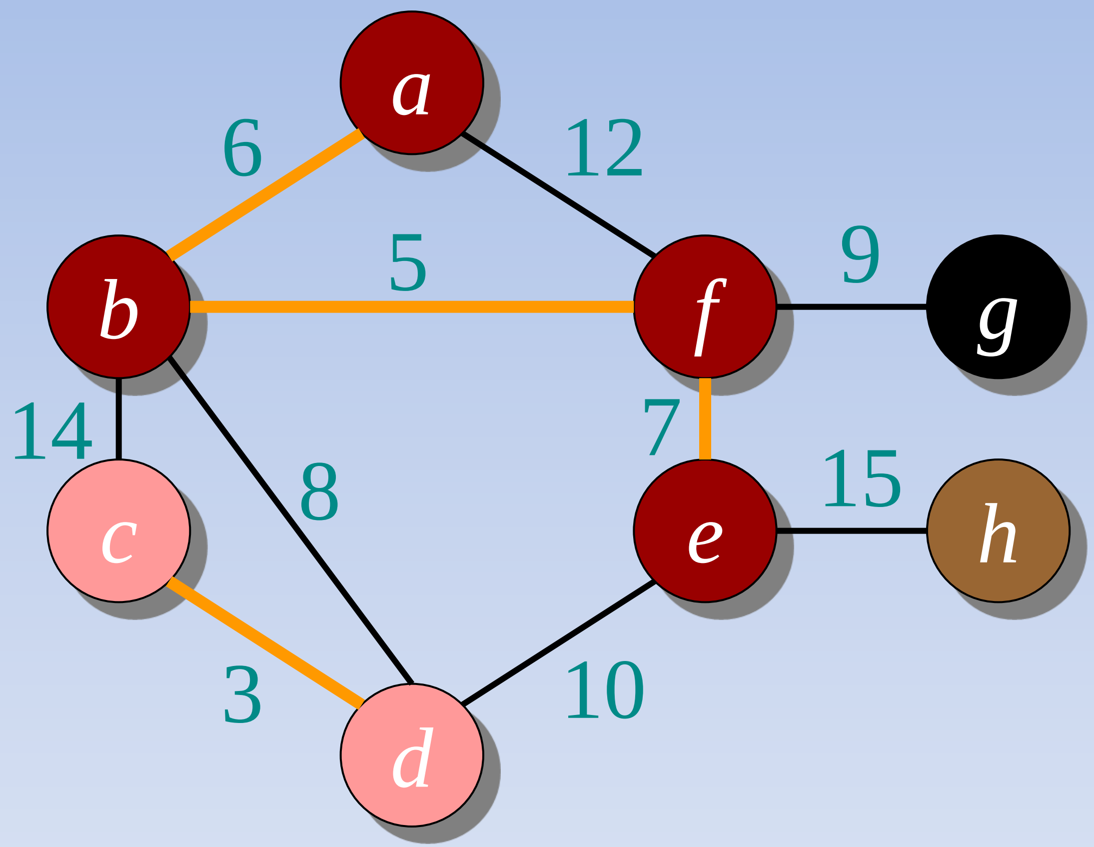
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

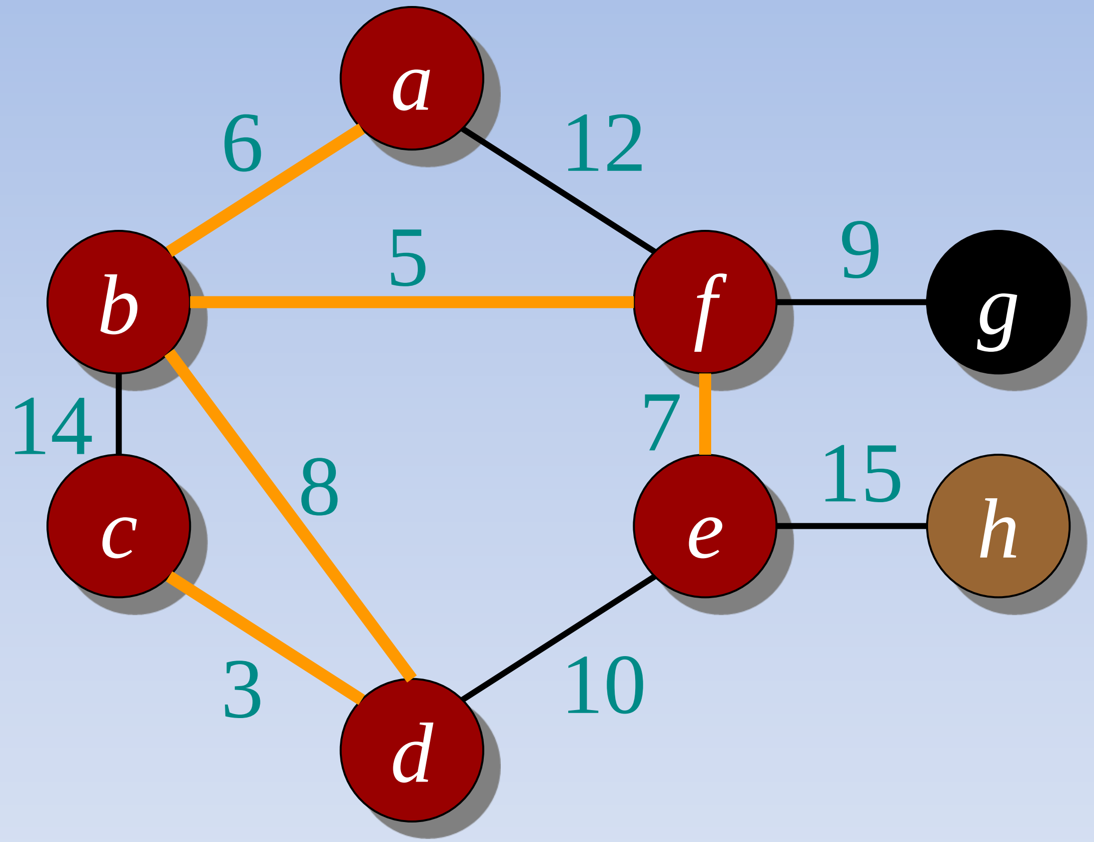
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

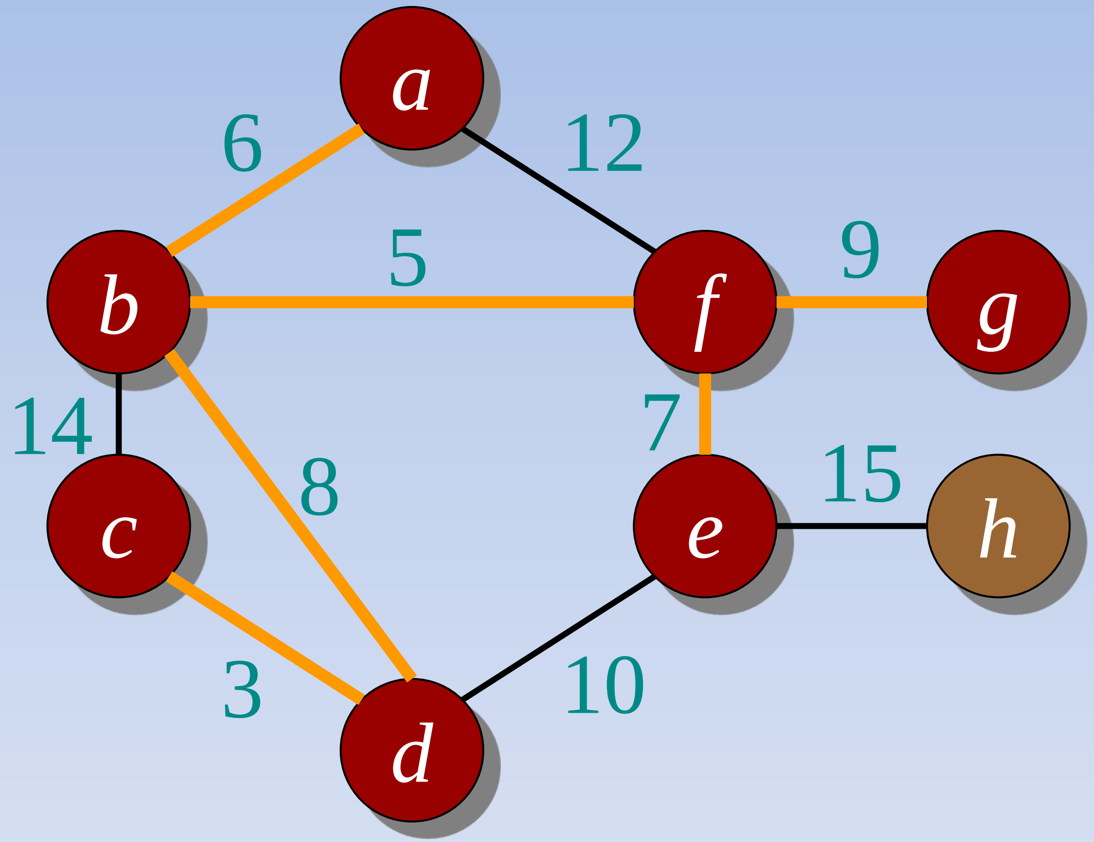
f-g: 9

d-e: 10

a-f: 12

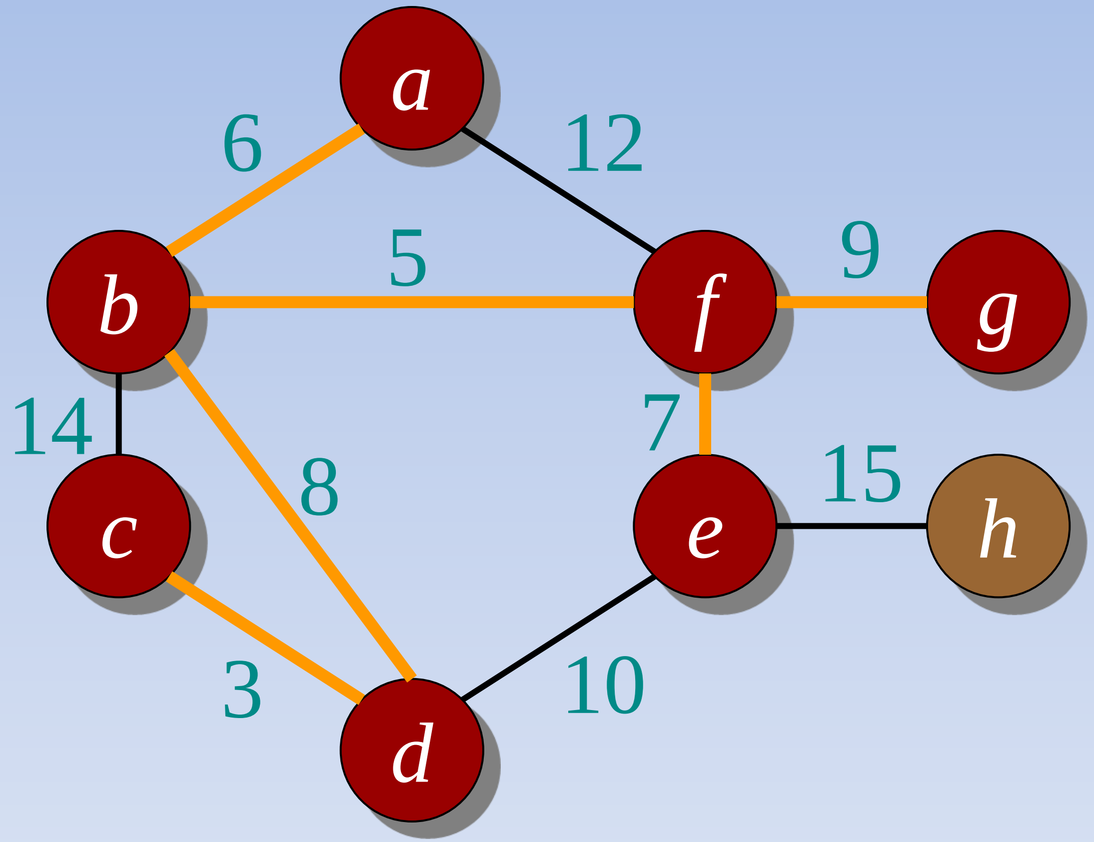
b-c: 14

e-h: 15



Example

c-d: 3
b-f: 5
b-a: 6
f-e: 7
b-d: 8
f-g: 9
d-e: 10
a-f: 12
b-c: 14
e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

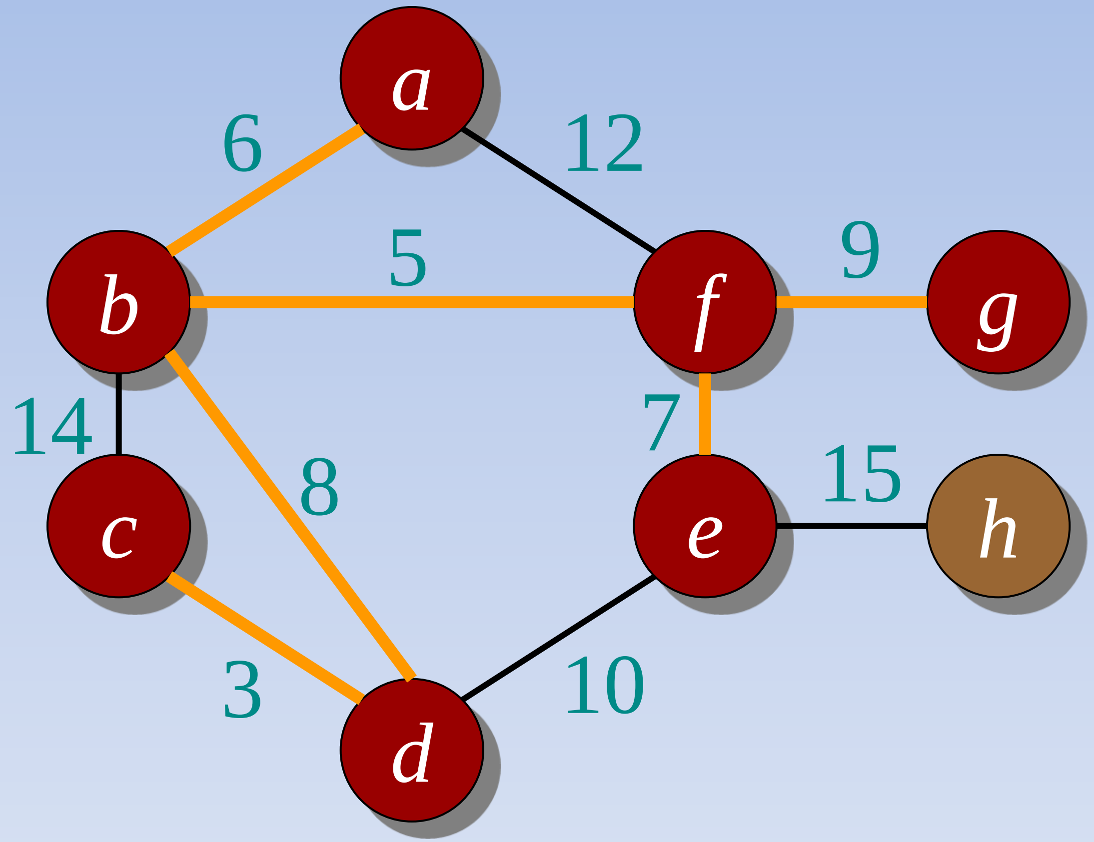
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

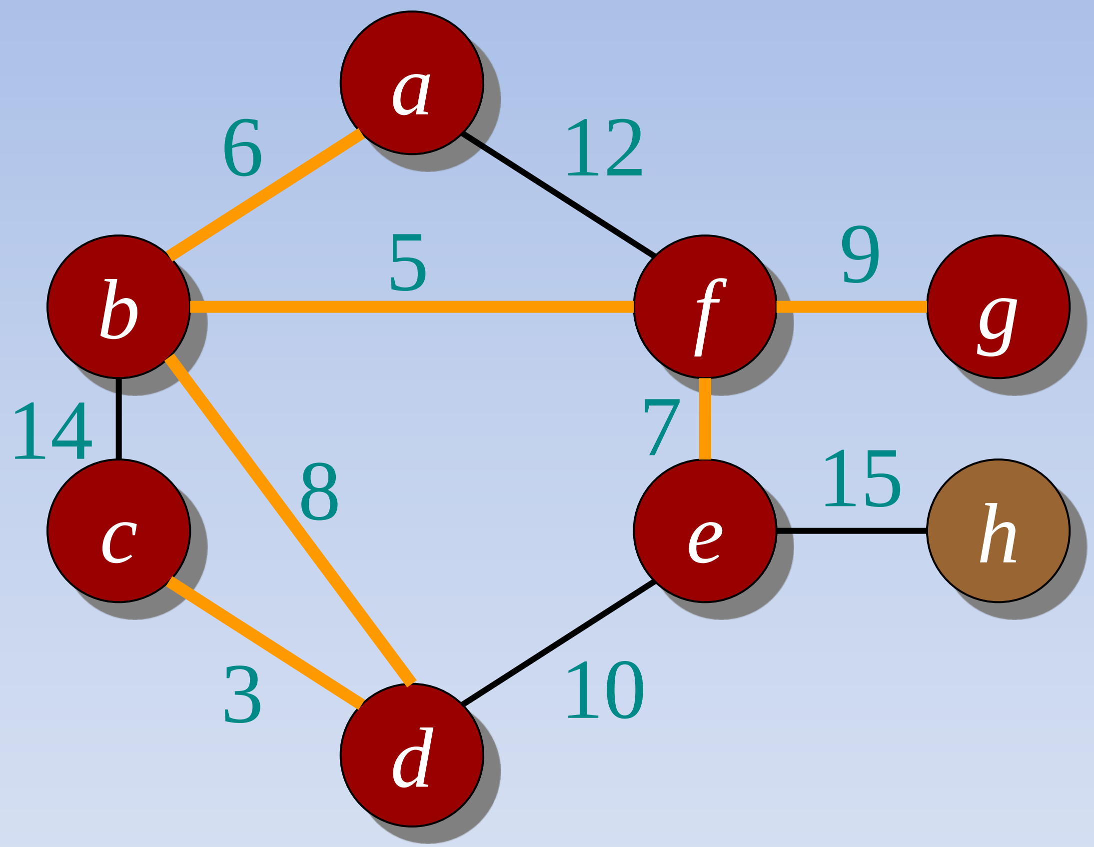
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

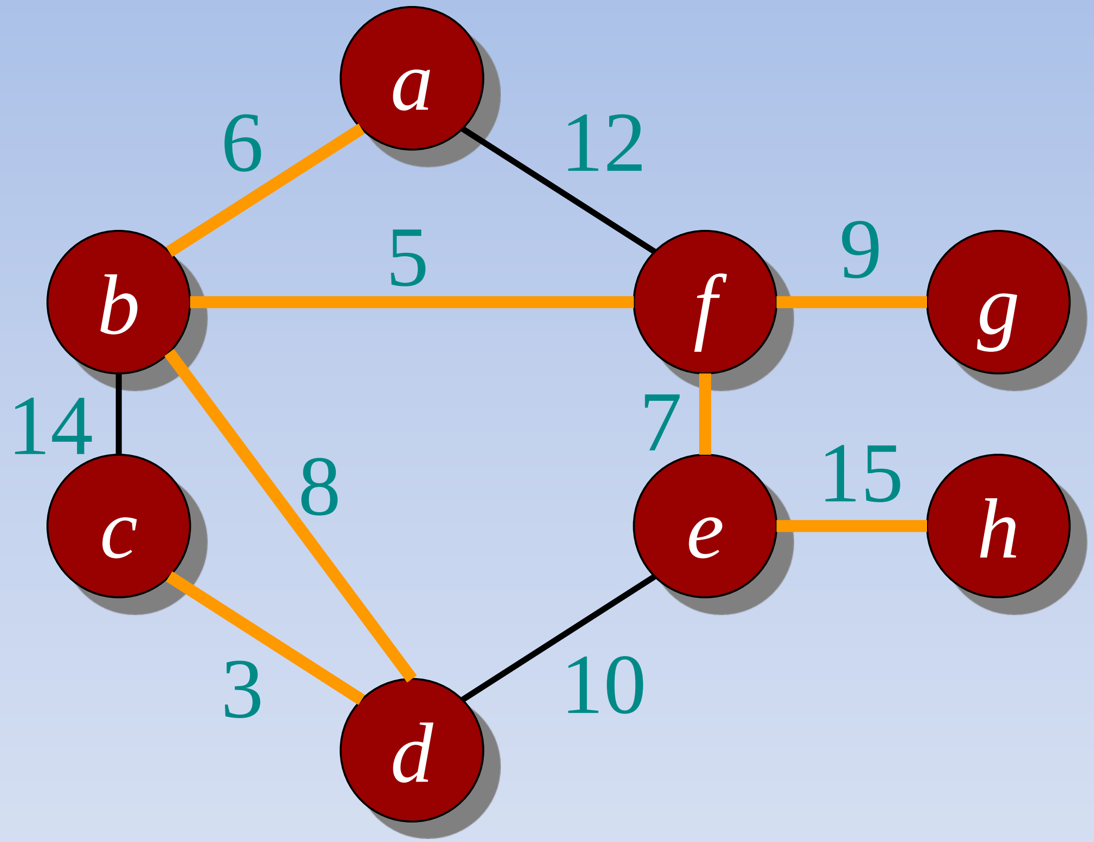
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Prim’s Algorithm

- Greedy Algorithm

- Operates like Dijkstra’s Shortest Path (later)
- Algorithm operates so edges in partial solution set of edges always form a single tree.

Prim’s Algorithm

- Starts from an arbitrary start node **r**
- Each step adds a LIGHT EDGE to current partial solution set A that connects a to an isolated vertex.

Prim Algorithm

- Q min-priority queue maintains list of all vertices not in tree.
- The MST edge set A is implicitly maintained
- A = { (v, v.p) : v in V – {r} – Q}
- At termination, Q is empty.
- Final MST A = { (v, v.p) for v in V – {r} }

Pseudocode

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

Pseudocode

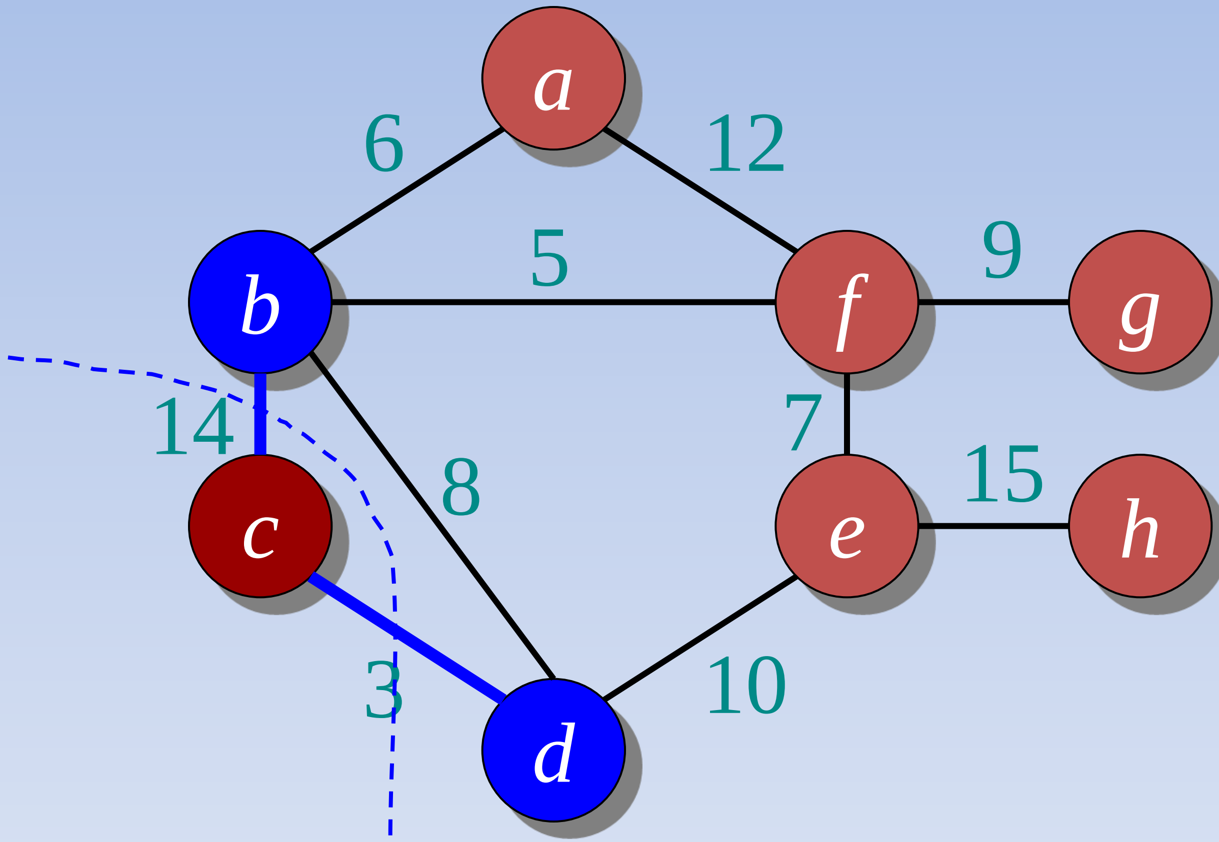
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Update Key

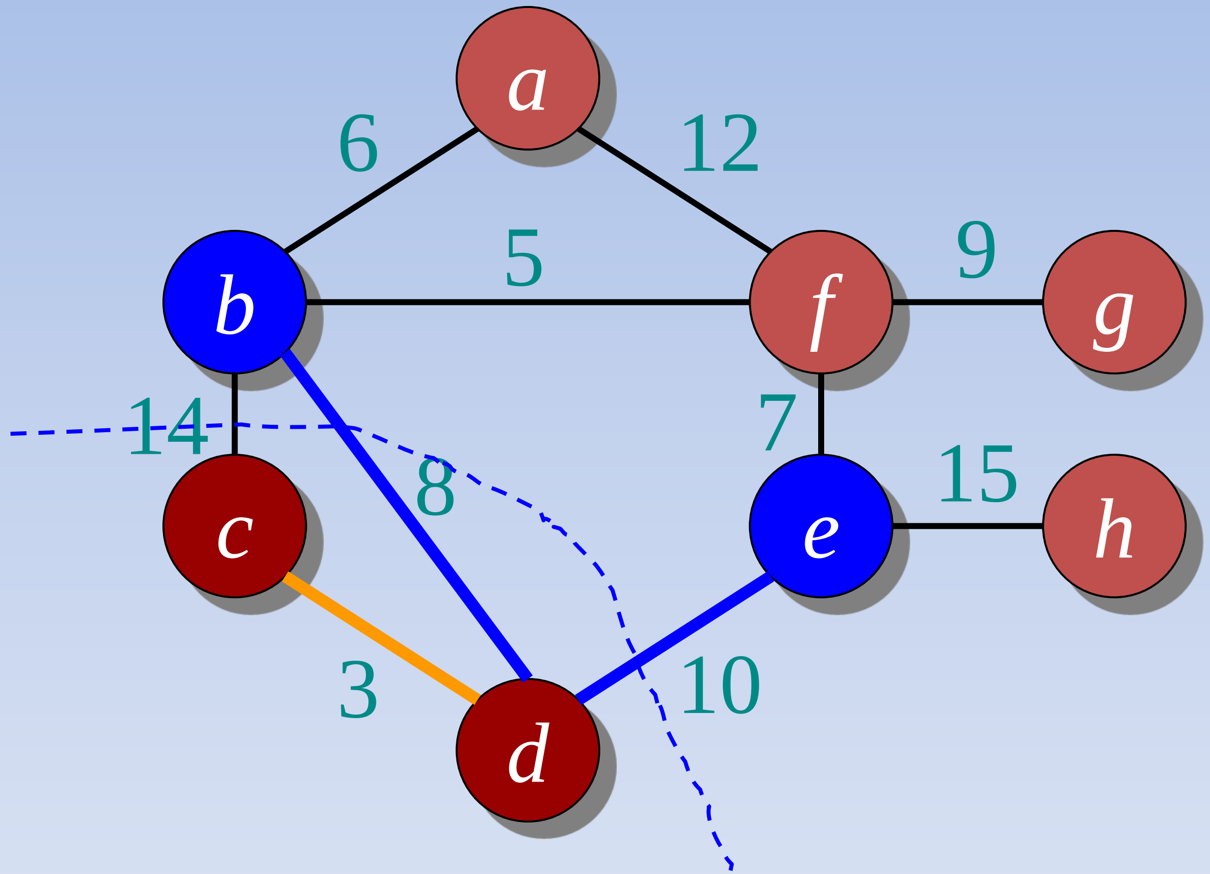
Prim's algorithm in words

- Randomly pick a vertex as the initial tree T
- Gradually expand into a MST:
  - For each vertex that is not in T but directly connected to some nodes in T
    - Compute its minimum distance to any vertex in T
  - Select the vertex that is closest to T
    - Add it to T

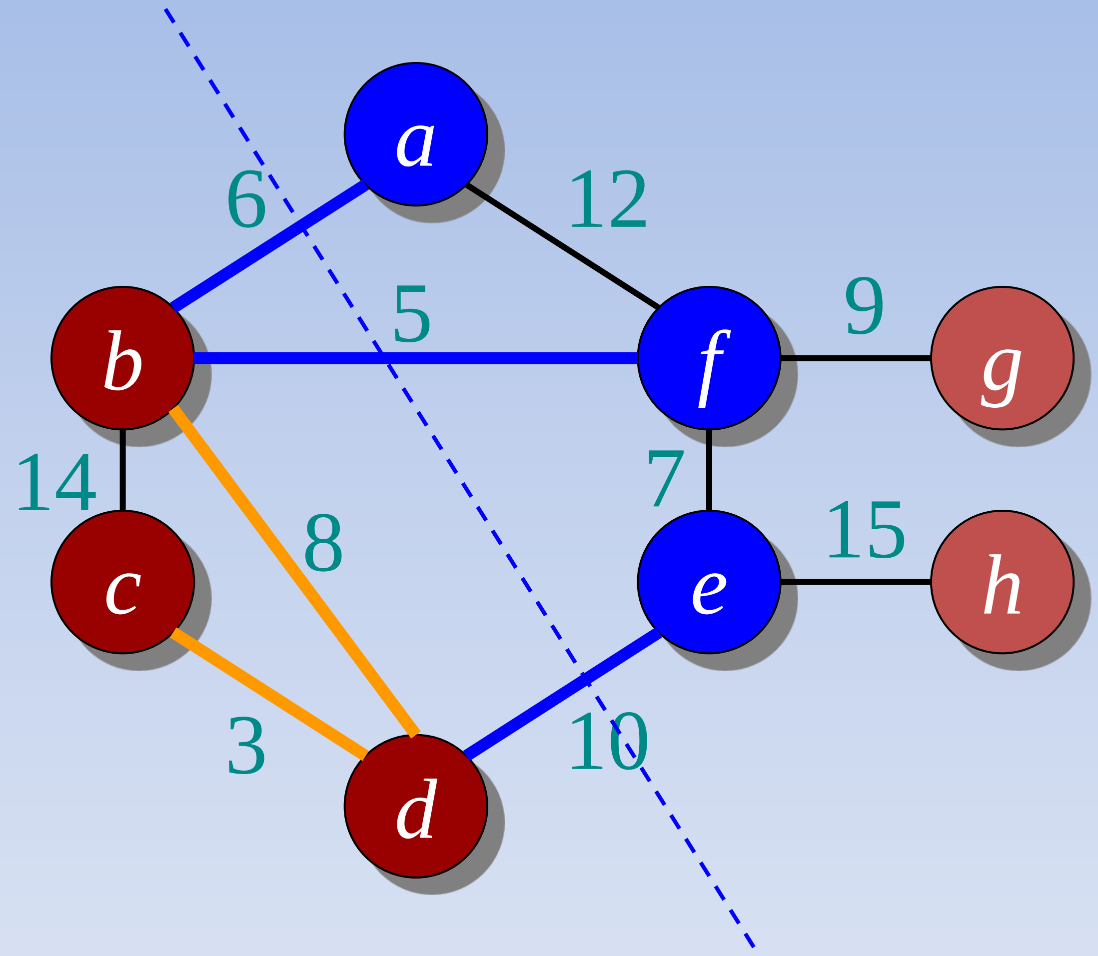
Example



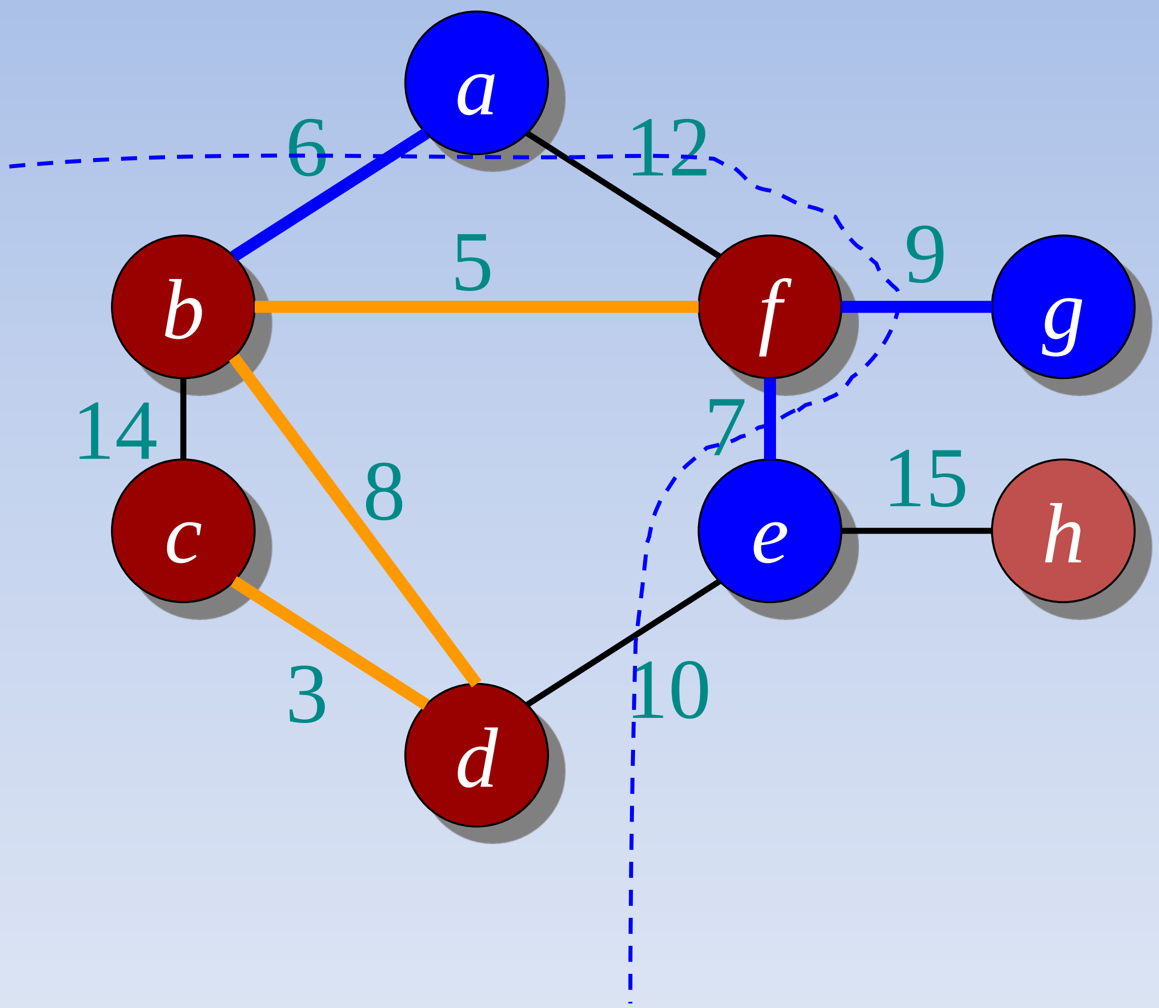
Example



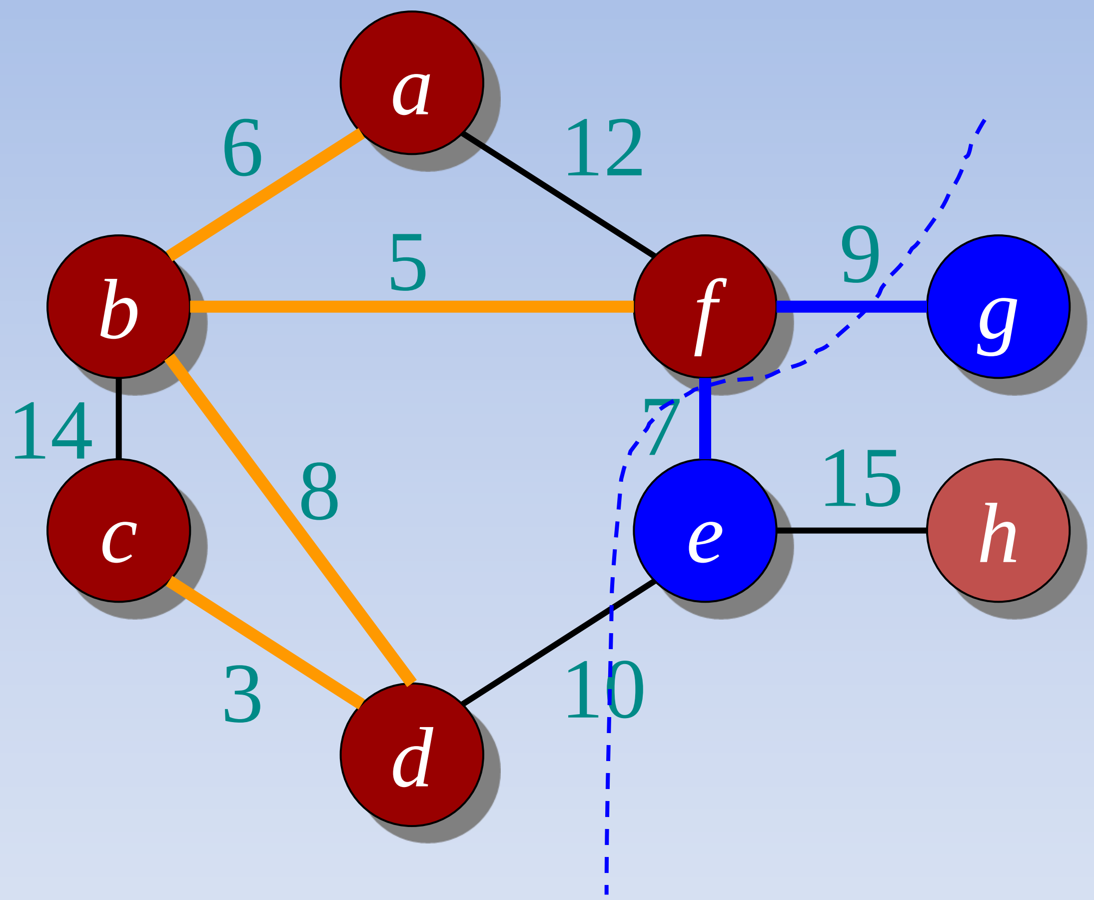
Example



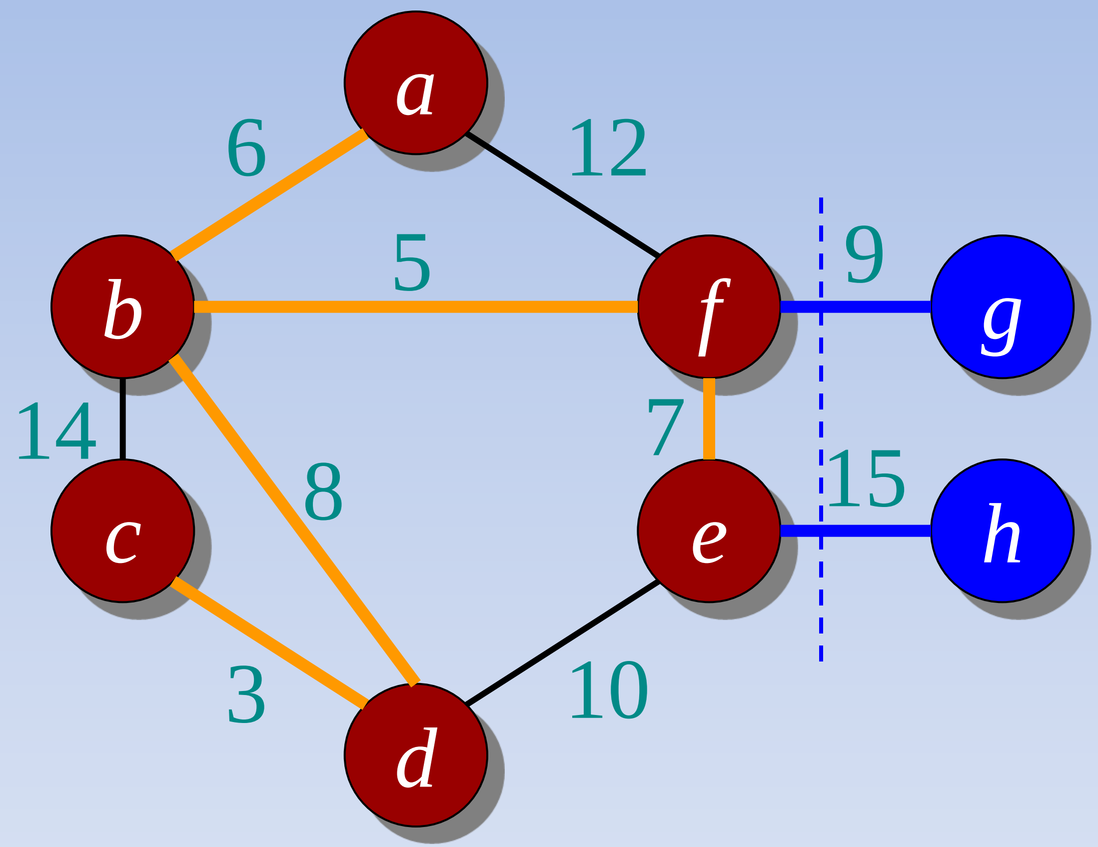
Example



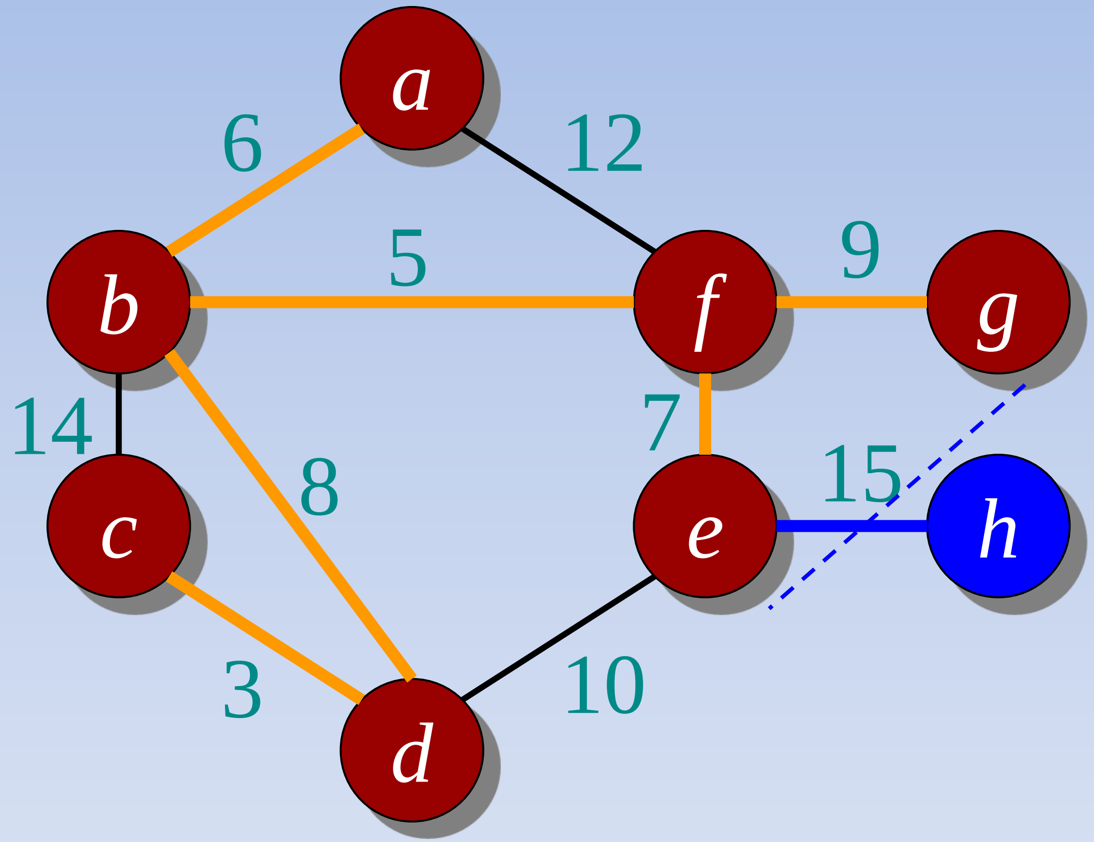
Example



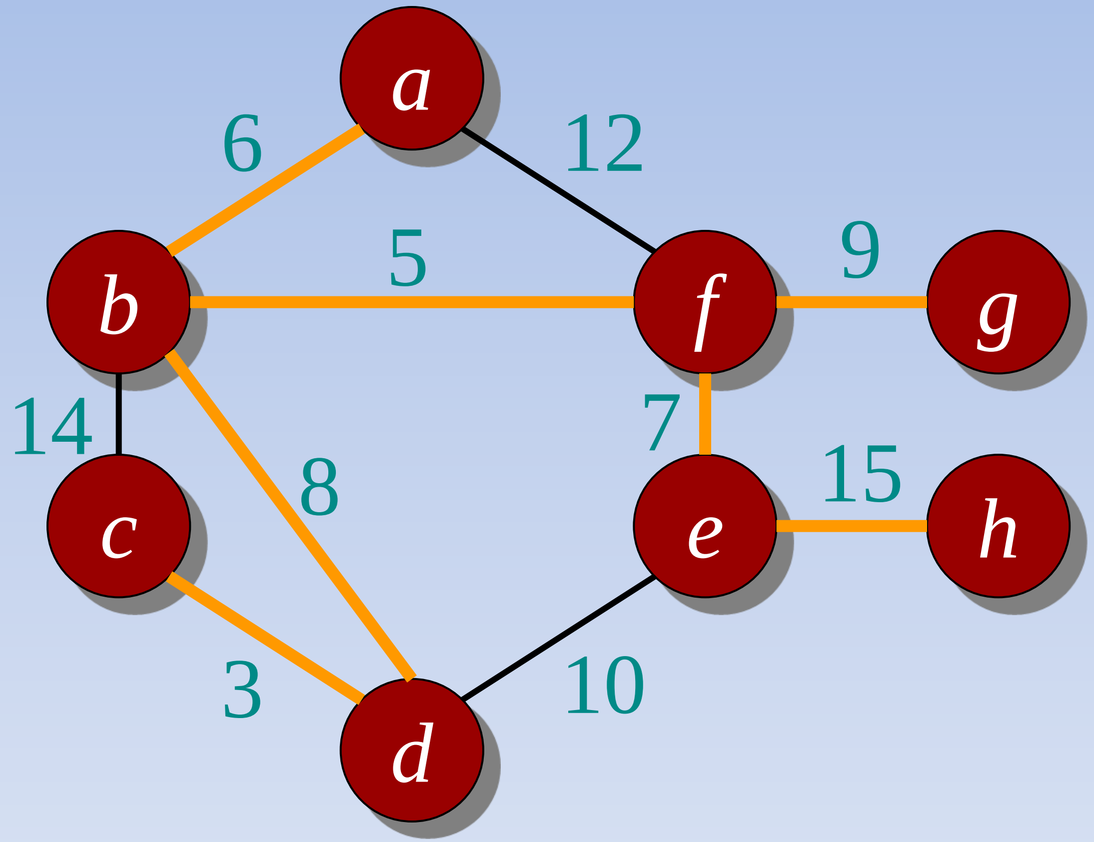
Example



Example



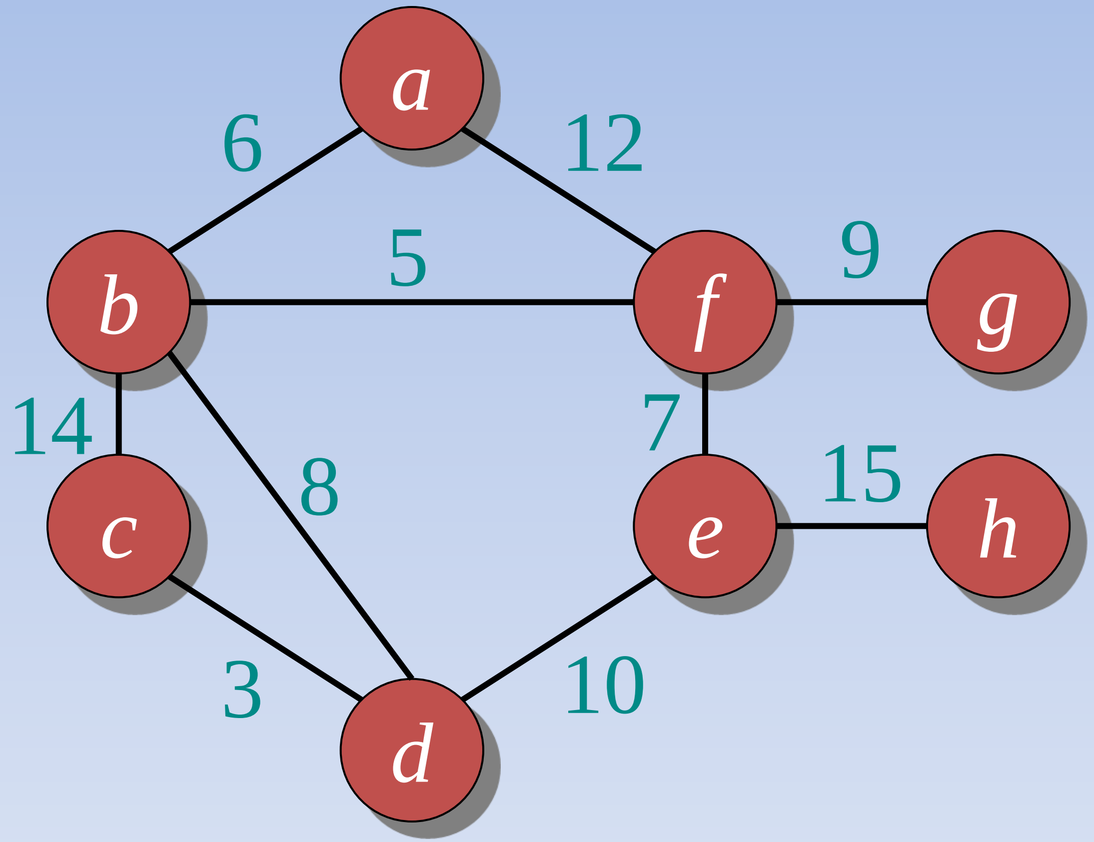
Example



Total weight = 3 + 8 + 6 + 5 + 7 + 9 + 15 = 53

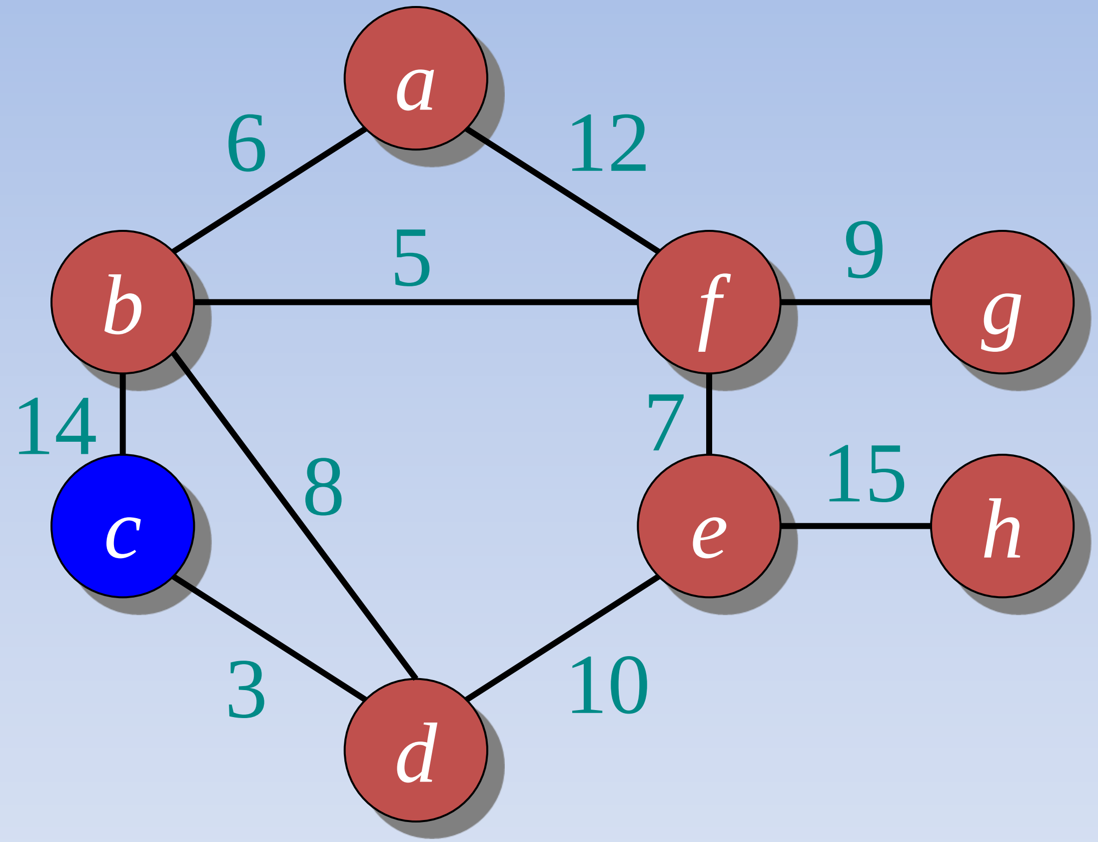
Prim Example w/ Priority Queue

Example



a	b	c	d	e	f	g	h
∞	∞	∞	∞	∞	∞	∞	∞

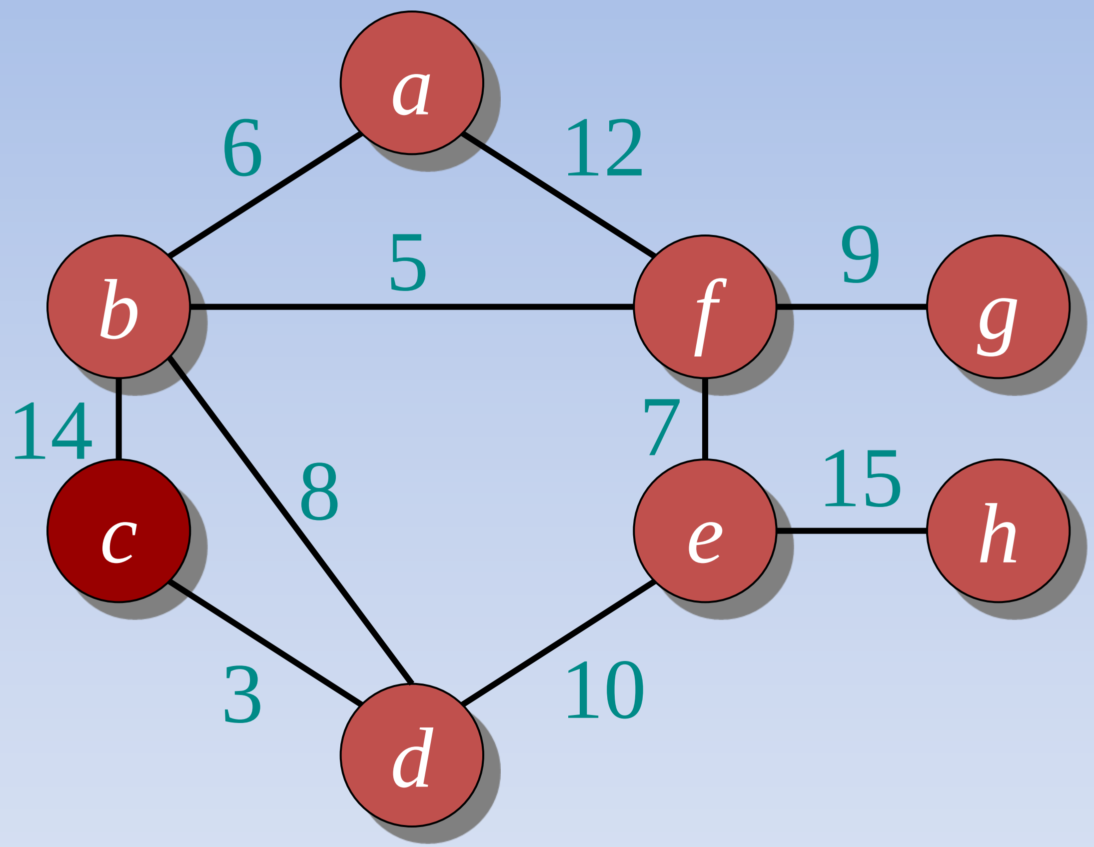
Example



ChangeKey

c	b	a	d	e	f	g	h
0	∞	∞	∞	∞	∞	∞	∞

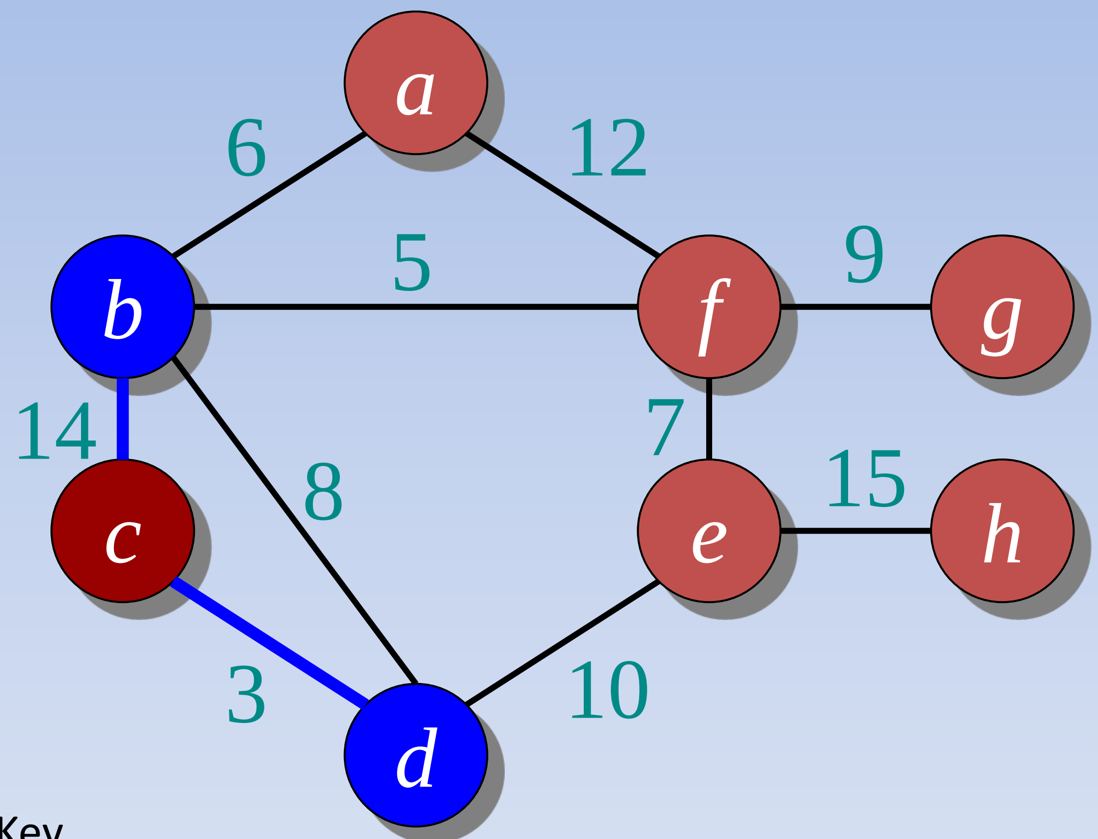
Example



ExctractMin

h	b	a	d	e	f	g
∞	∞	∞	∞	∞	∞	∞

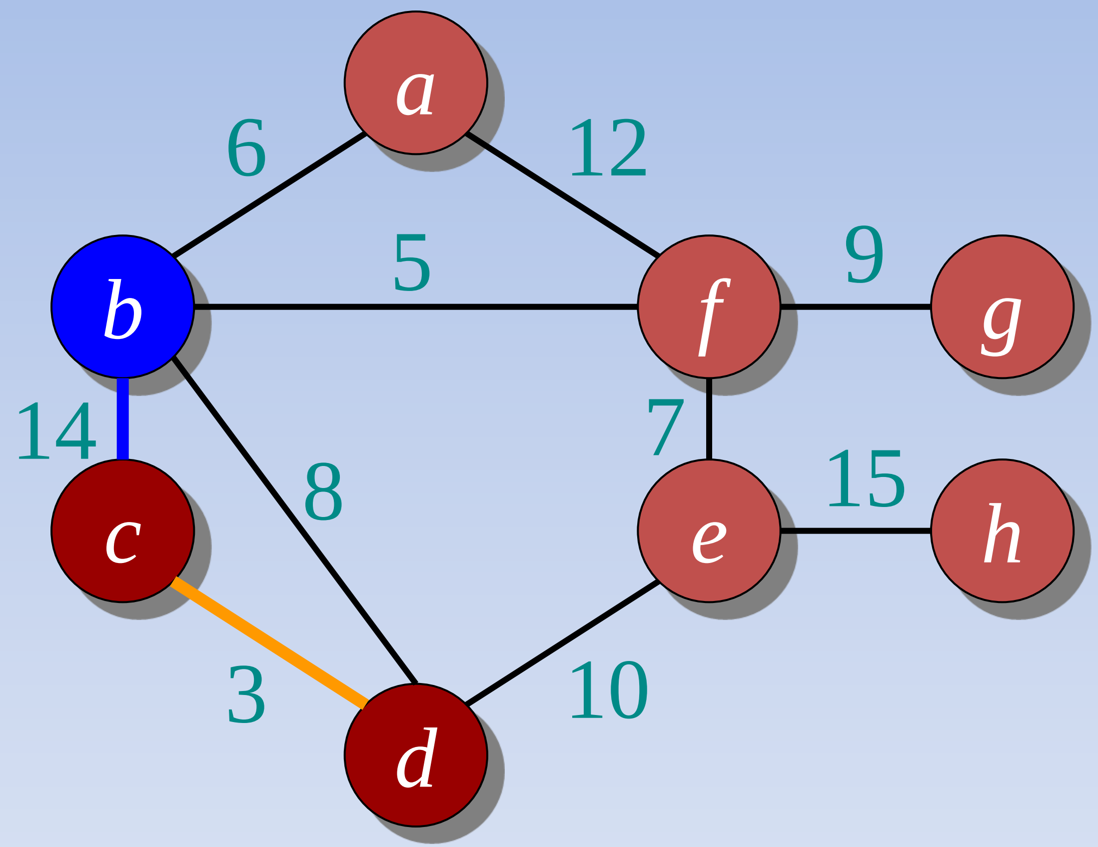
Example



ChangeKey

d	b	a	h	e	f	g
3	14	∞	∞	∞	∞	∞

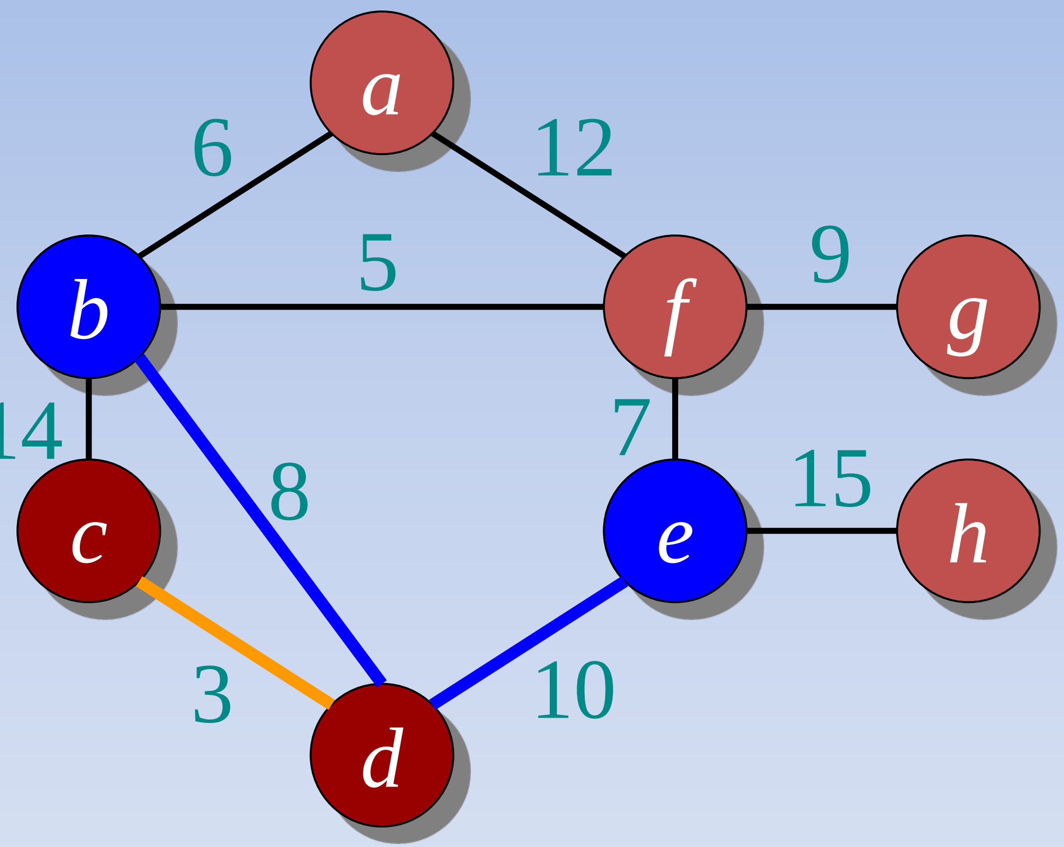
Example



ExctractMin

b	g	a	h	e	f
14	∞	∞	∞	∞	∞

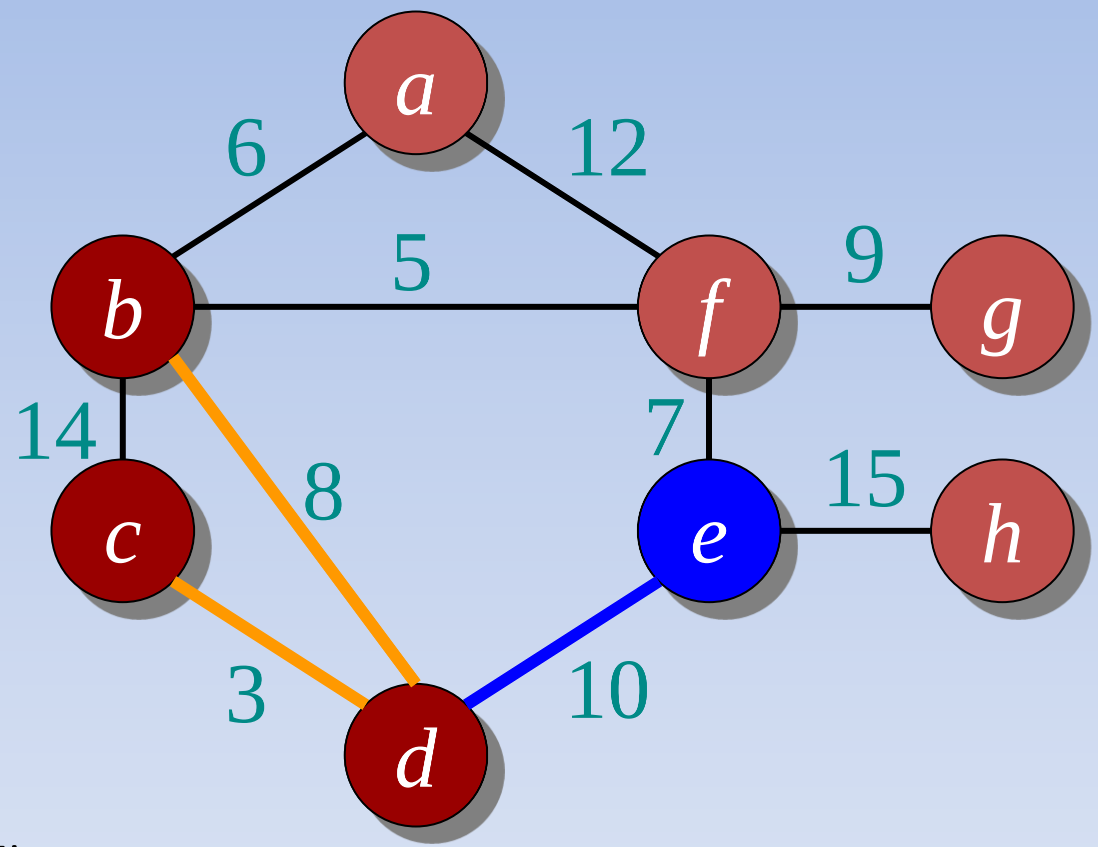
Example



Changekey

b	e	a	h	g	f
8	10	∞	∞	∞	∞

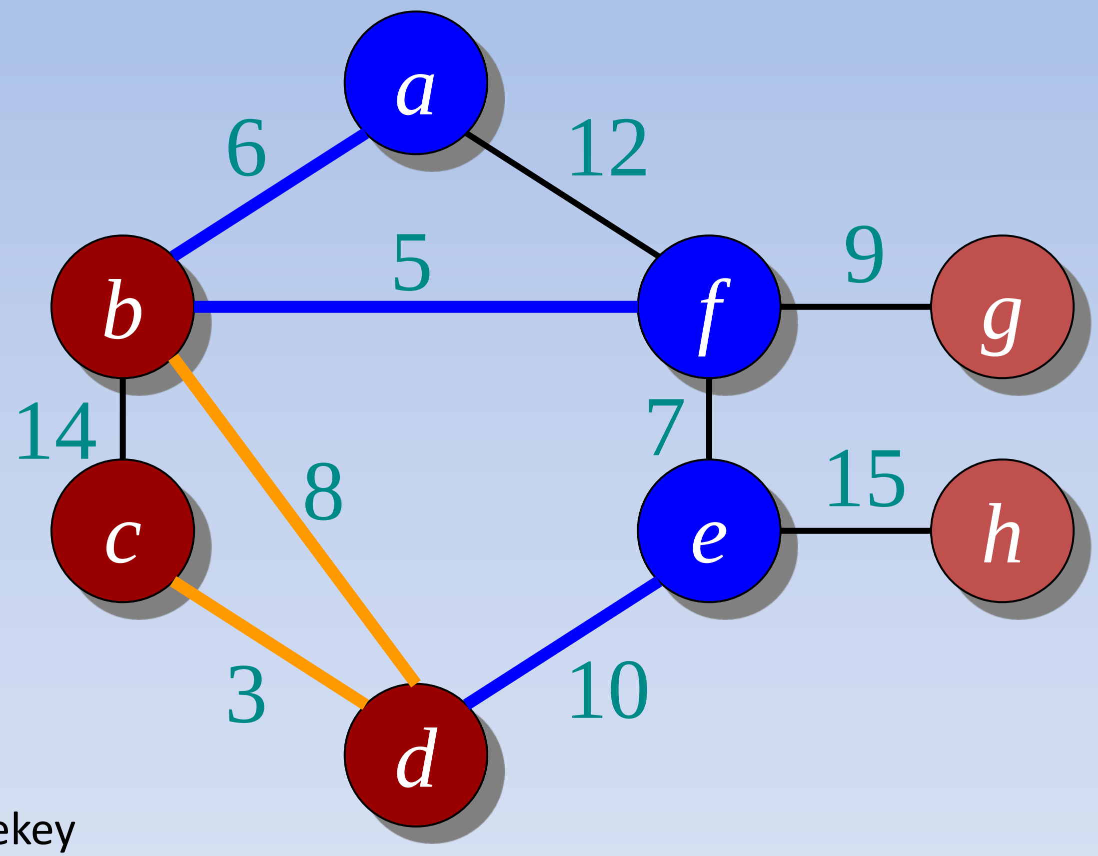
Example



ExtractMin

e	f	a	h	g
10	∞	∞	∞	∞

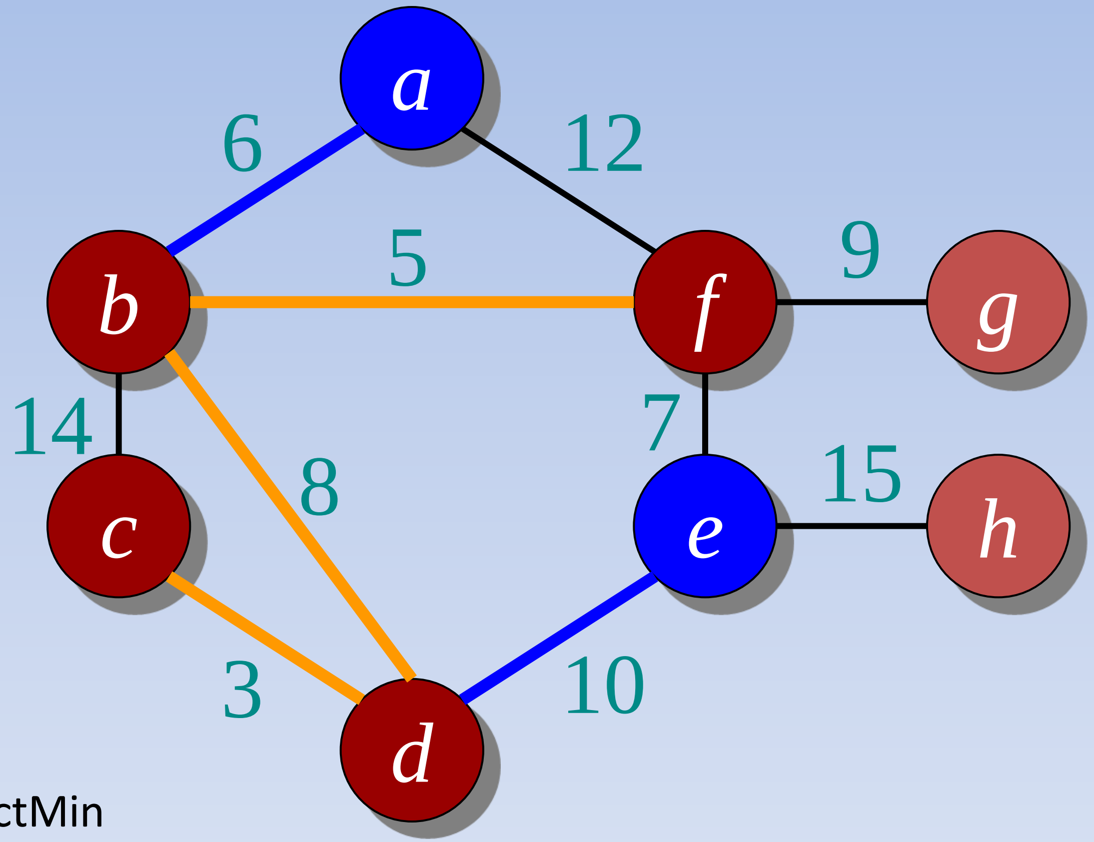
Example



Changekey

f	e	a	h	g
5	10	6	∞	∞

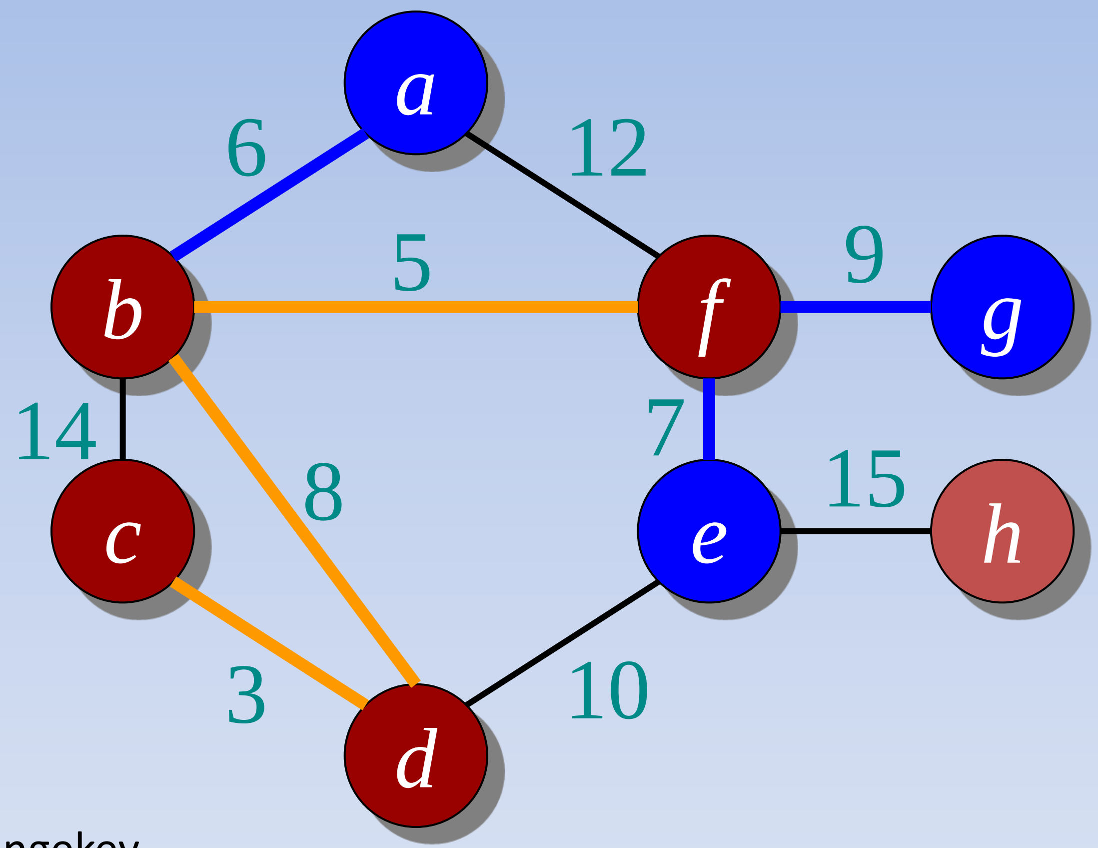
Example



ExtractMin

a	e	g	h
6	10	∞	∞

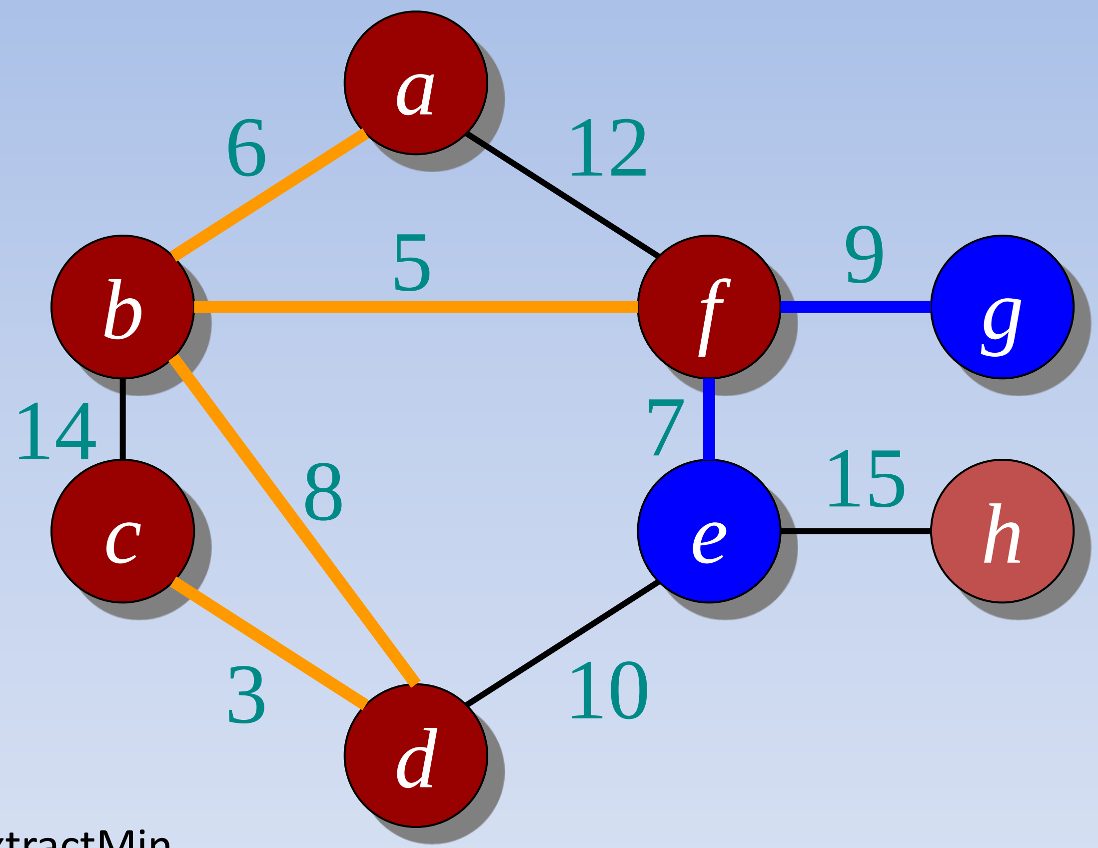
Example



Changekey

a	e	g	h
6	7	9	∞

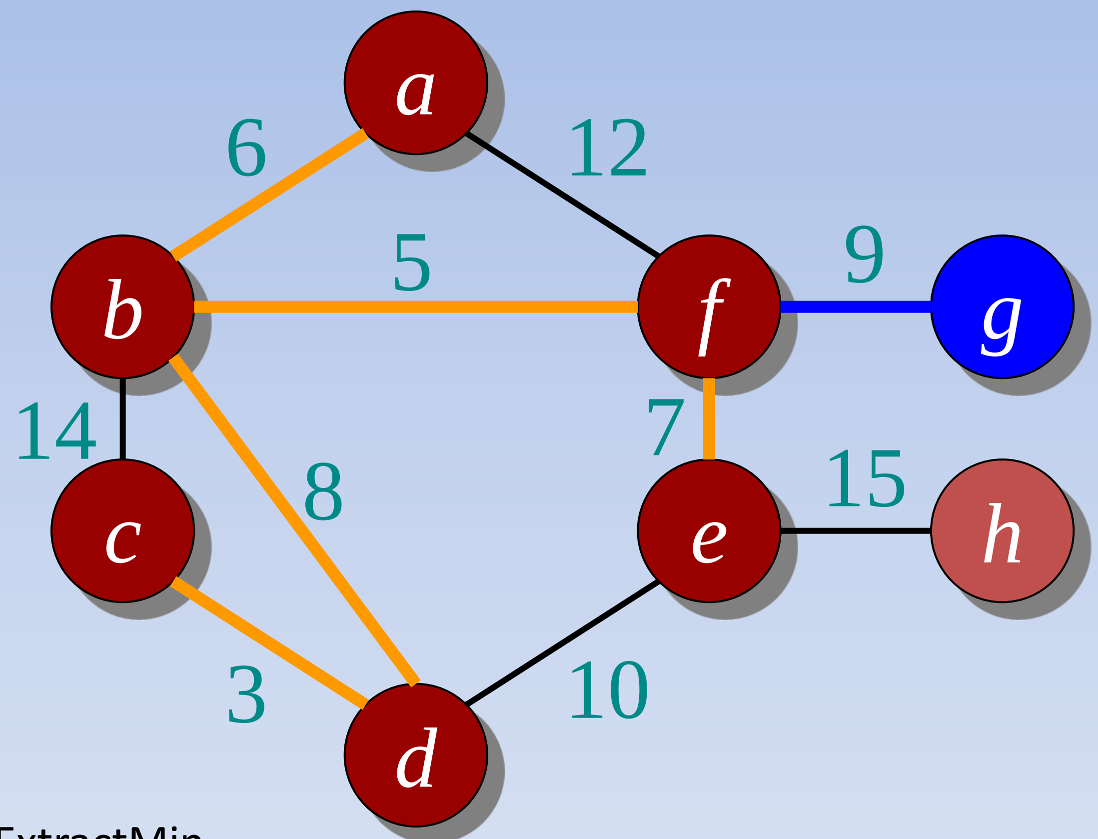
Example



ExtractMin

e	h	g
7	∞	9

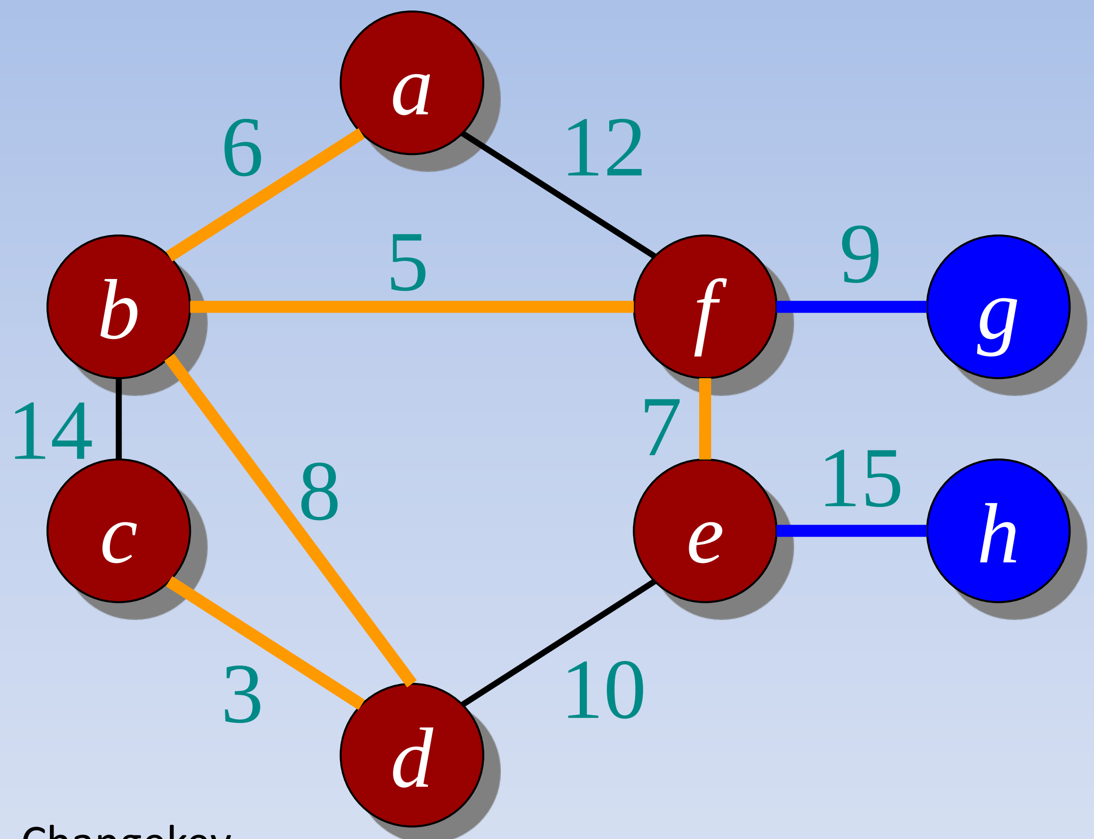
Example



ExtractMin

g	h
9	∞

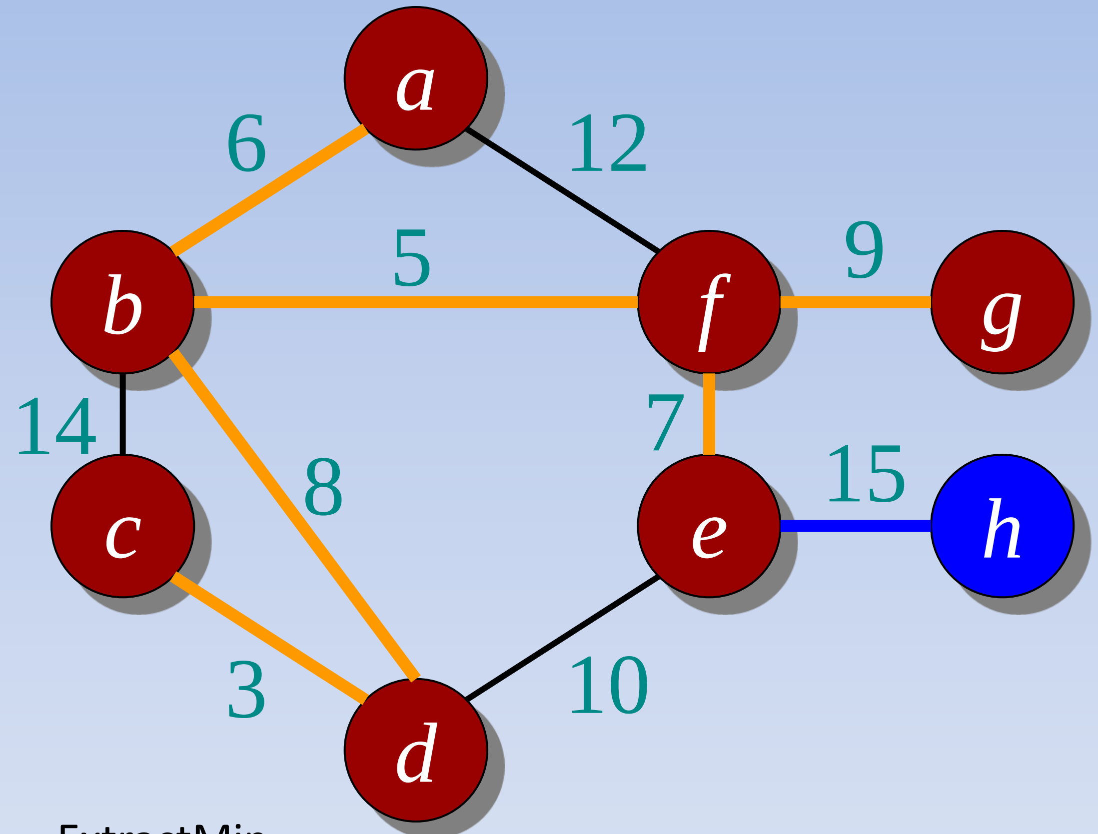
Example



Changekey

g	h
9	15

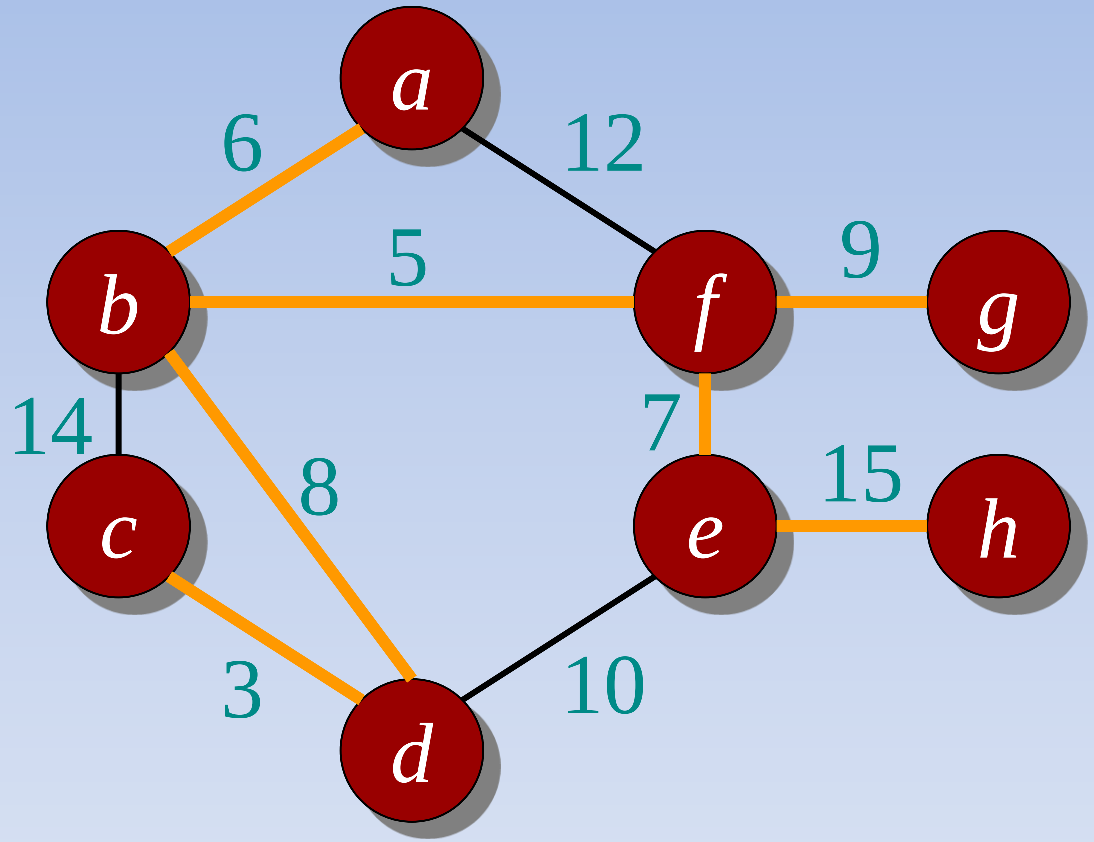
Example



ExtractMin

h
15

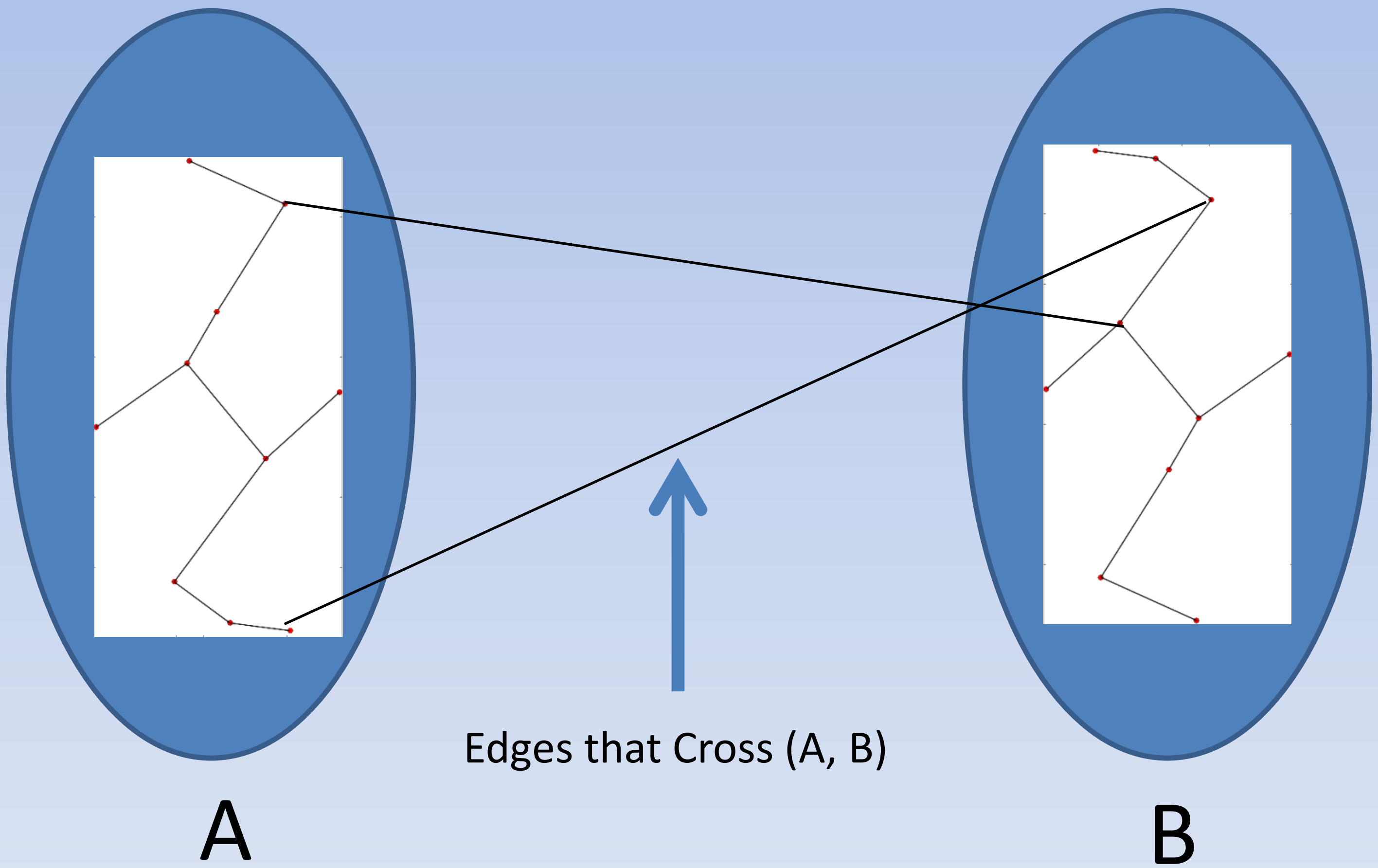
Example



Prim’s Proof – Part 1

- Prove: Prim’s Algorithm outputs a spanning tree!
- Cuts – Important Concept for proof.
- A cut of a graph G=(V,E) is a partition of V into 2 non-empty sets!

Prim's Proof:
Cut (A, B)



Empty Cuts

- Show:
  - A graph is not connected IF-AND-ONLY-IF
  - Exists a cut (A, B) with NO edges that Cross It!

- Part A:
  - IF a graph is not connected
  - THEN exists a cut (A, B) with no edges that Cross It!
- Part B:
  - IF there exists a cut (A, B) with no edges that Cross It
  - THEN the graph is not connected!

Empty Cuts

- Part B: If exists a cut (A, B) with no edges that cross it THEN the graph is not connected.
- Assume Graph G has a cut (A, B) with no edges that cross it.
- Pick any vertices u∈A and v∈B
- Since there are no edges that cross A, B, there can be no path between u—v.
  - THEREFORE the graph is not connected!

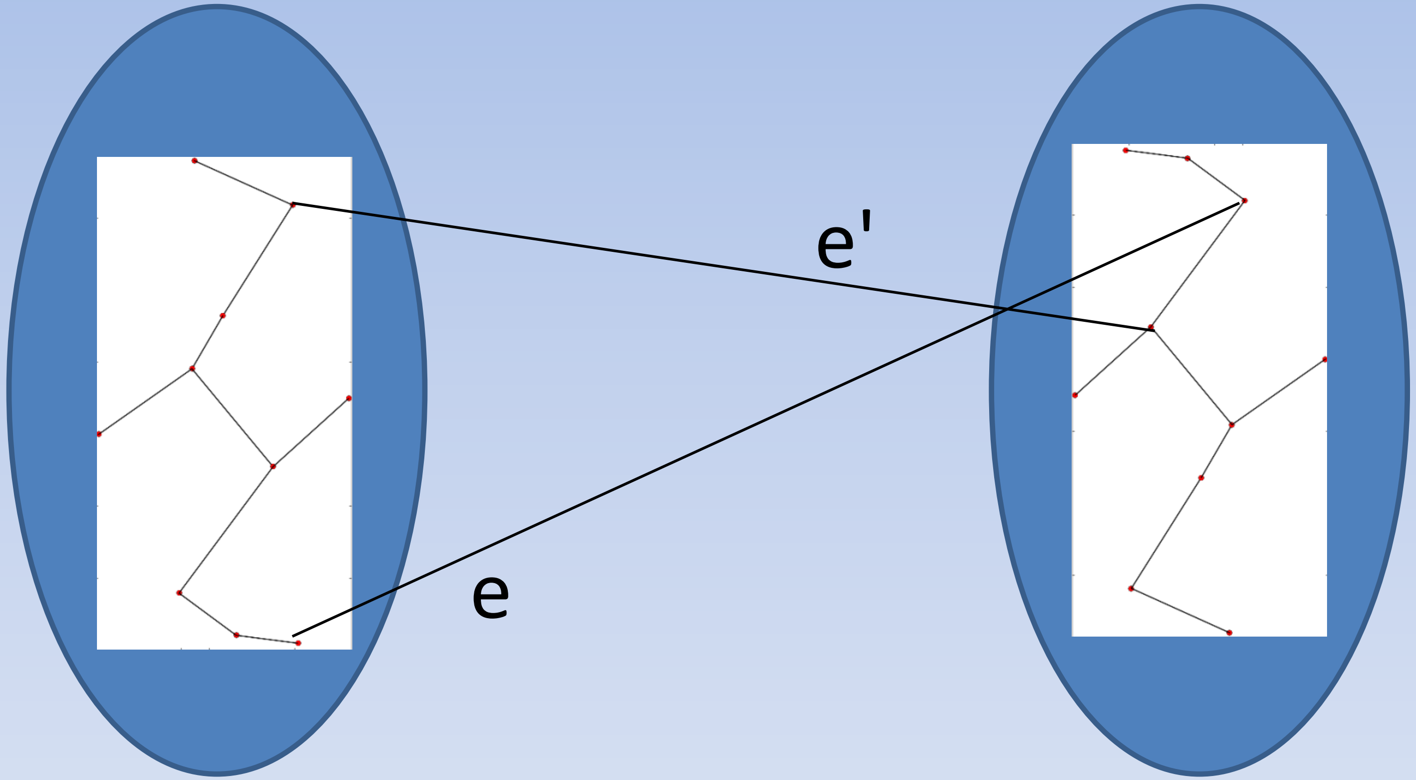
Empty Cuts

- Part A: If a graph is not connected, THEN exists a cut (A, B) with not edge that crosses it!
- Assume graph is not connected,
  - so exists vertices u, v with no path between them!
- Define A as vertices reachable from u
- Define B as all other vertices.
- A includes at least u, and B includes at least v
  - So A & B are non-empty.
- There can be no edge between A and B
  - Since this edge would create a path between u—v, introducing contradiction!

Additional Facts

- Double-Crossing Lemma:
  - Suppose there is a cycle $C \subseteq E$
  - Suppose that for cut (A, B), C has edge **e** that crosses it!
  - Then C must have some other edge **e’** that also crosses it!

Double-Crossing Lemma:



- e and e’ both needed to create cycle!

Lonely-Cut Corollary

- Given cut (A, B) with edge e as the only edge to cross it, THEN e is not in any cycle!
  - Since if e were in cycle, then cut (A, B) would have another edge to cross it!

Prove Prim Produces Spanning Tree

- Not necessarily minimum spanning tree (yet)!
- (1): Introduce Invariant: partial tree T constructed spans vertices X.
  - Given: X, T
  - Vertices in : V-X
    - Those vertices from V not yet spanned by Tree T.
  - Add edge between X and V-X.
  - Move edge vertex from V-X to X!
    - Maintains invariant.

Prove Prim Produces Spanning Tree

- (2) Prim can't stop with $X \neq V$
  - Otherwise the cut $(X, V-X)$ must be empty
  - This implies the graph must not be connected!
    - Can't construct a spanning tree with a disconnected graph!

Prove Prim Produces Spanning Tree

- (3) Prim can’t introduce cycles
  - Consider Iteration with T, X
  - Assume e gets added
  - e is the first edge introduced crossing (X, V-X)
    - e’s introduction cannot introduce cycle!
      - Lonely Cut Corollary!

Part II: Prim's Algorithm Outputs Minimum Cost Spanning Tree

- When is it "safe" to include an edge in tree so far?

The Cut Property

- Consider an edge e of G.
- Suppose there is a cut of G (A, B) such that e is the cheapest edge of G that crosses it!
- Then e belongs to the MST of G.
  - Turns out MST is unique IF edge costs are distinct!

Prim Analysis

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

Prim Analysis

- Line 7 identifies a vertex v in Q incident on a light edge that crosses the cut (V-Q, Q)
  - Except for the first iteration, in which u=r due to line 4.
- Removing u from Q adds it to V-Q
  - vertices in Tree
  - (u, u.p) added to A
- Looping Lines 8-11 update Key and P attributes for every vertex v adjacent to u that is not in the tree.

Prim Analysis

- Build-Min-Heap = O(V)
- While Loop executes |V| times with each Extract-Min taking lgV total = O(V lg V)
- Looping Lines 8-11 go though all Edges, and each Decrease-Key takes lgV total= O(E lg V)
- O(V lg V + E lg V) = O(E lg V)
  - Same and Kruskal
- Fibonacci Heaps can improve by changing Descrease-Key time to O(1)
  - O(V lg V + E*1) = O(V lg V)