Minimal Spanning Tree
(networks design problem)

Chapter 23

Cormen Leiserson Rivest&Stein:
Introduction to Algorithms

Minimal Spanning Trees

Weighted Graphs   G(V, E, w)

W: E → R

If w=1 for all edges BFS is the solution.

The MST is the way of connecting all the vertices at minimal cost of connections.

Two greedy algorithms : Kruskal

Jarnik-Prim

Minimal Spanning Trees

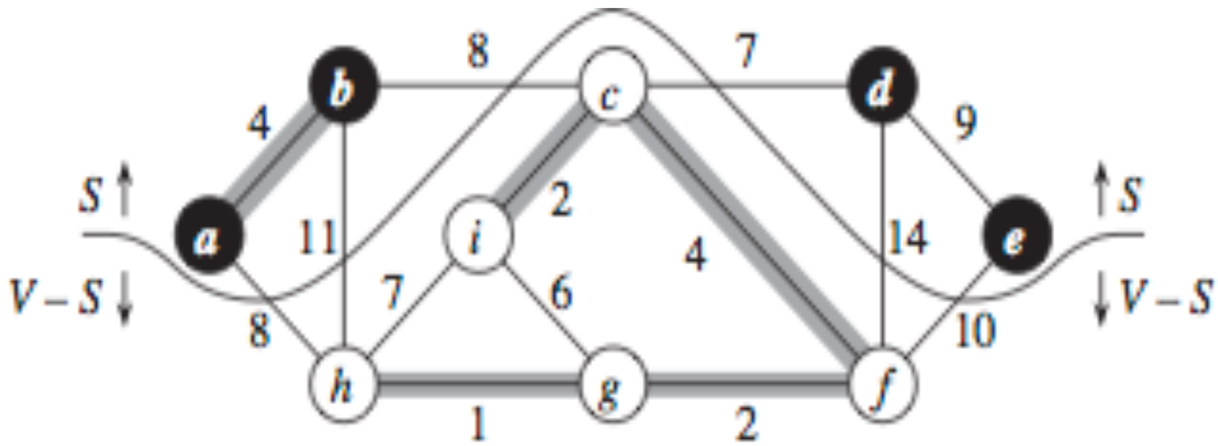
First a generic method utilized by the two algorithms:

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

Prior and after of each iteration, A is a subset of a MST

Determine a safe edge (u,v): A U (u,v) is still a subset of a MST.

Minimal Spanning Trees



(a)

S vertices in MST (black). V-S vertices to be selected. The line is the cut. Light edges crossing the cut can be selected for the MST.

They are safe!

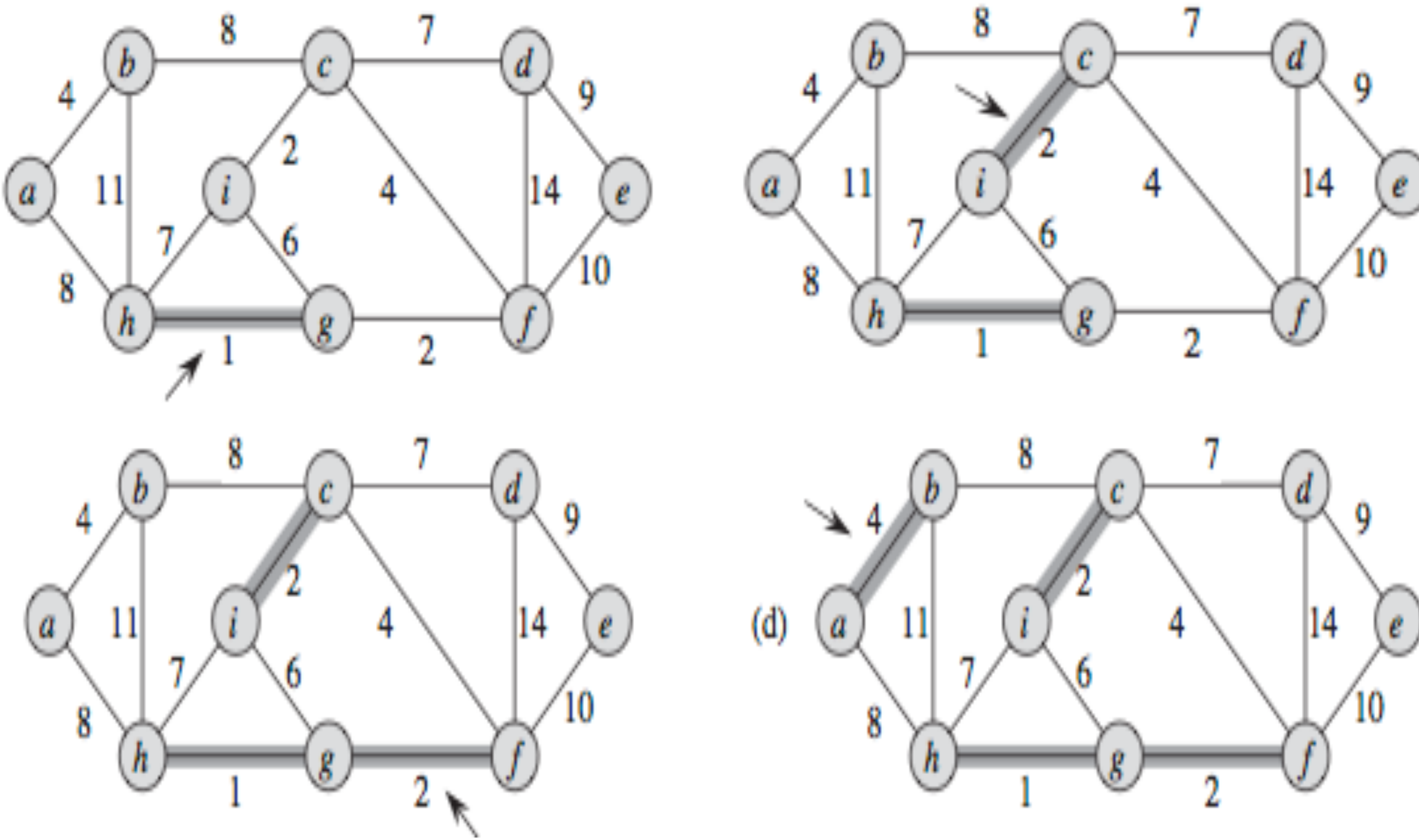
Kruskal Algorithm for MST

```
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
```

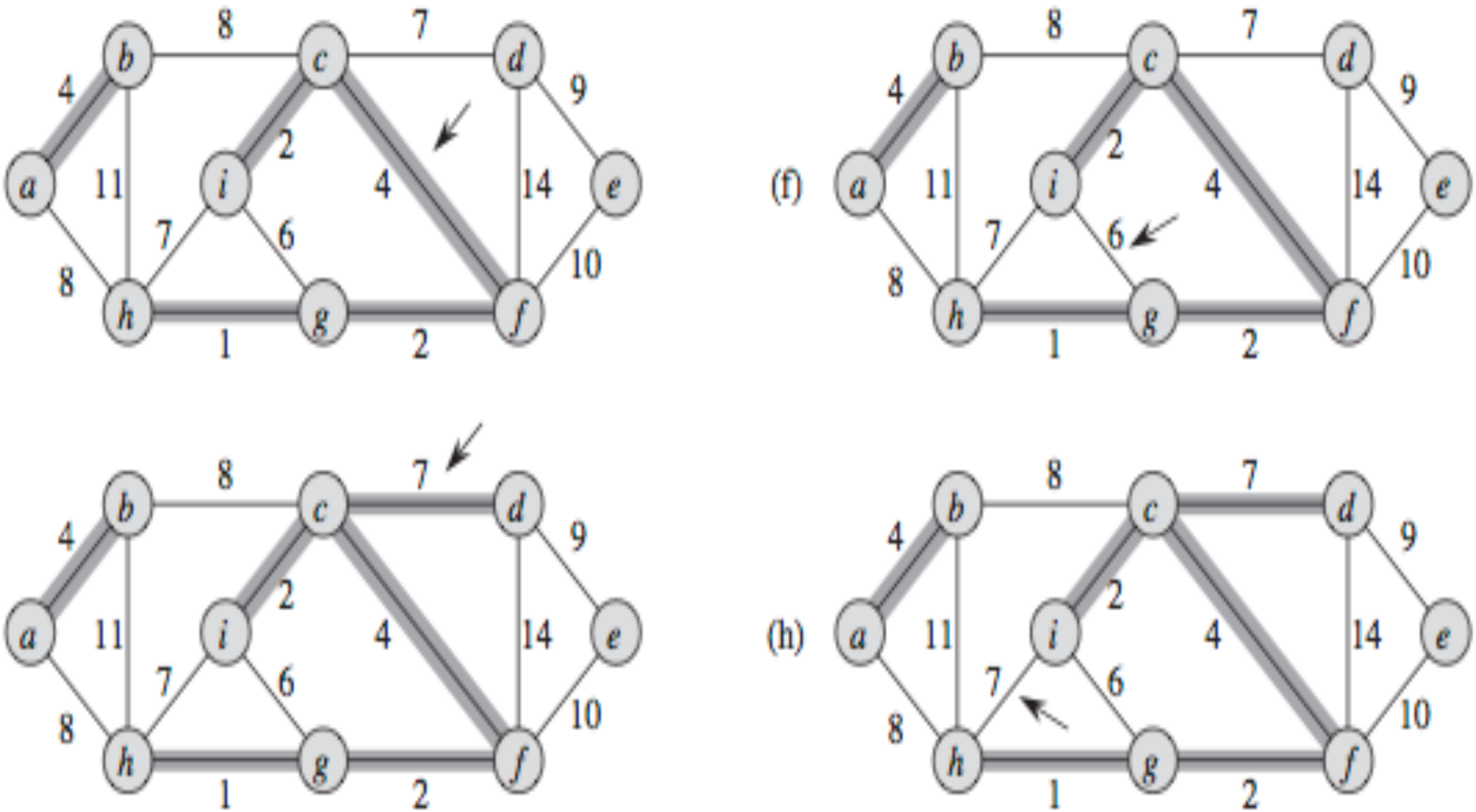
Uses a disjoint-set data structure to maintain several disjoint sets of elements. FIND-SET(u) returns a representative element of the set containing u.

(Disjoint set forest chapter. 21)

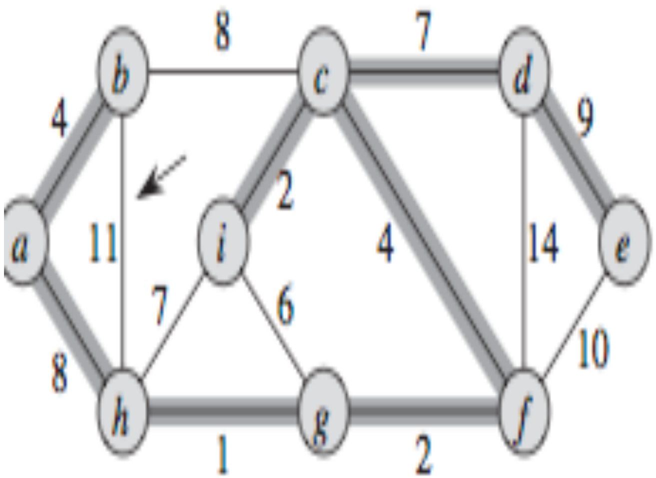
Kruskal Algorithm for MST



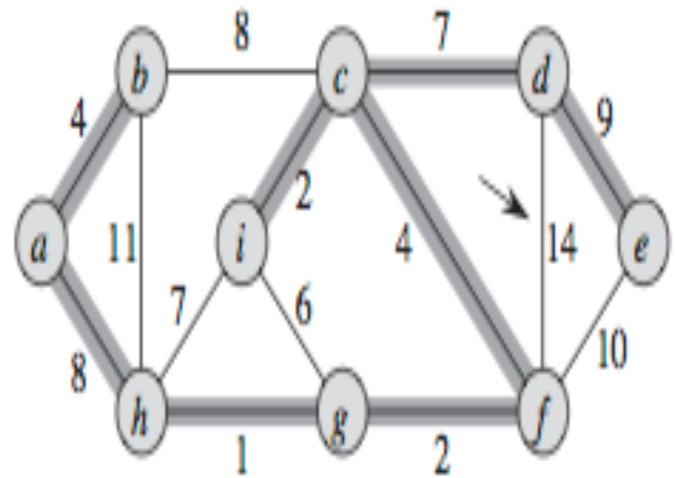
Kruskal Algorithm for MST



Kruskal Algorithm for MST



(n)



Kruskal Algorithm complexity

Complexity depends on how we implement the disjoint-set data structure.

(Disjoint set forest chapter. 21.3)

Sorting of the edges $O(E \log E)$

Loop 5-8: $O(E)$ FIND-SET operations on the disjoint set forest.

$|V|$ operations of MAKE-SET

In total the loop takes $O((V+E)\,\alpha(V))$ time, where α is avery slowly growing function ($\log^* V$). Since $E \geq V-1$ and $\alpha(V)=O(\log V) = O(\log E)$ the loop takes $O(E \log E)$.

In total the Kruskal alg. takes $O(E \log E)$

Jarnik-Prim algorithm for MST

Starts from an arbitrary vertex r the root.

The set A forms a single tree at any step.

All vertices v not in the spanning tree form a min Heap Q ordered on the weight of any edge connecting v and the tree. (value v.key in the alg.)

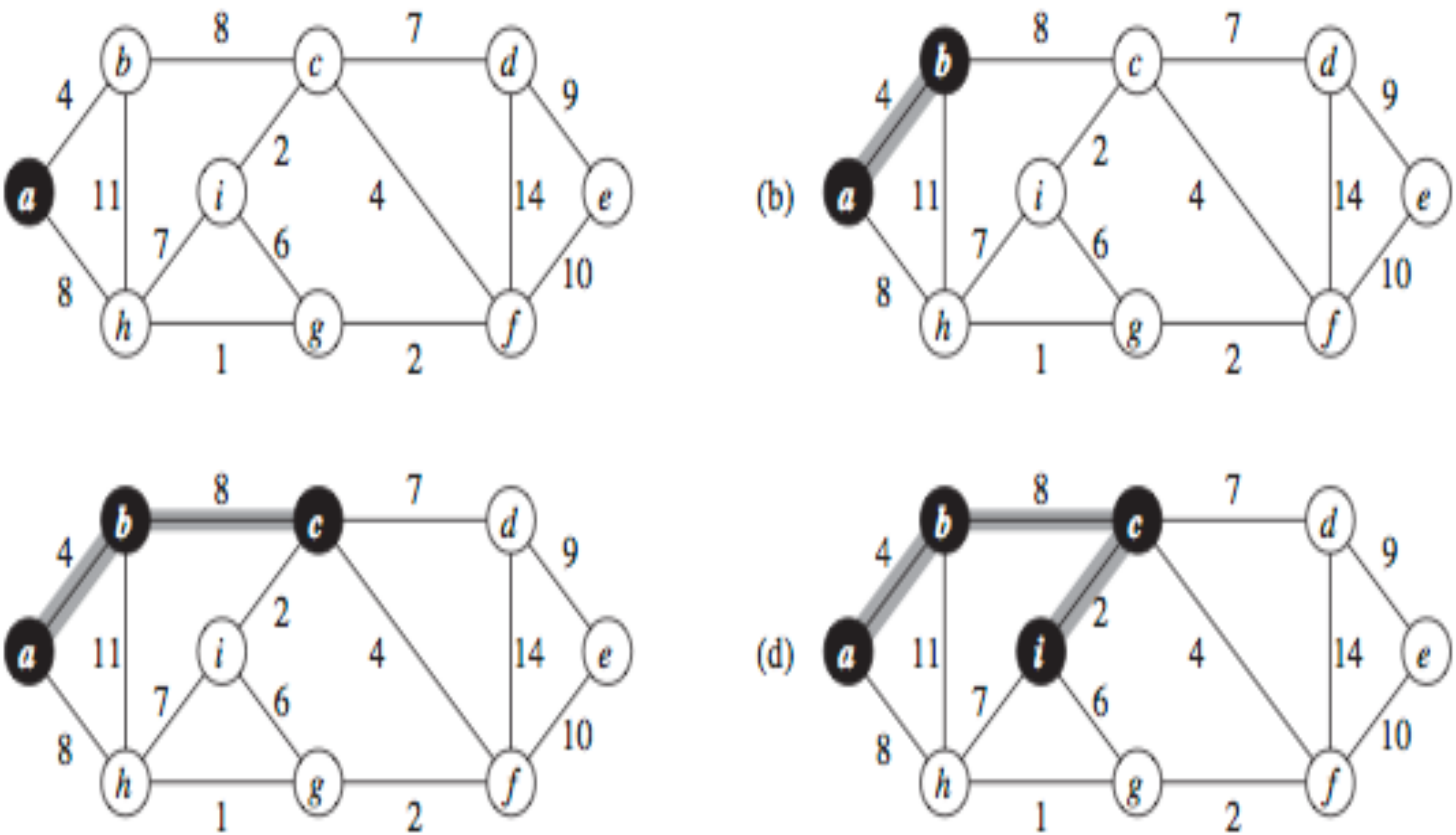
The termination condition is that the heap Q is empty, that means that all vertices have been connected to the MST.

Jarnik Prim algorithm for MST

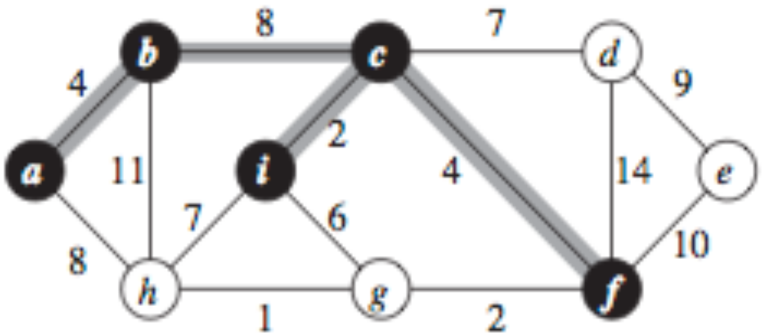
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

The parent in the MST

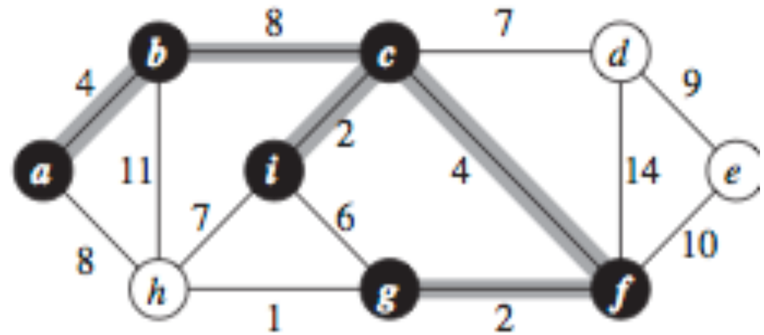
PRIM algorithm for MST

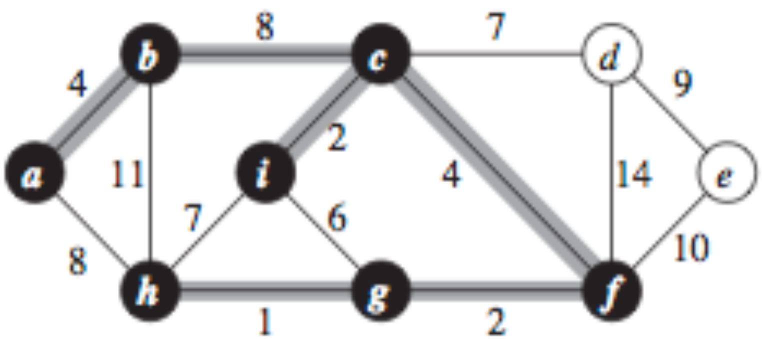


PRIM algorithm for MST

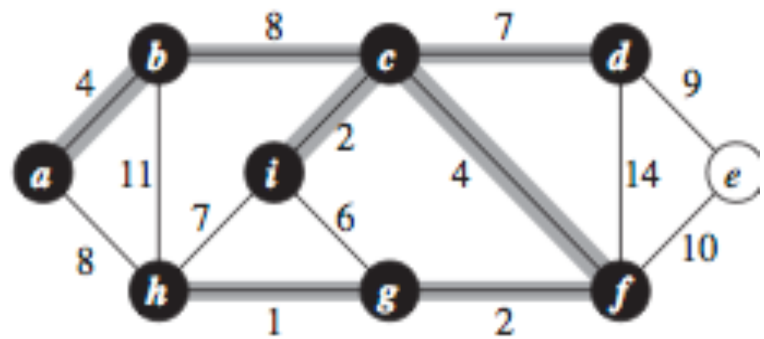


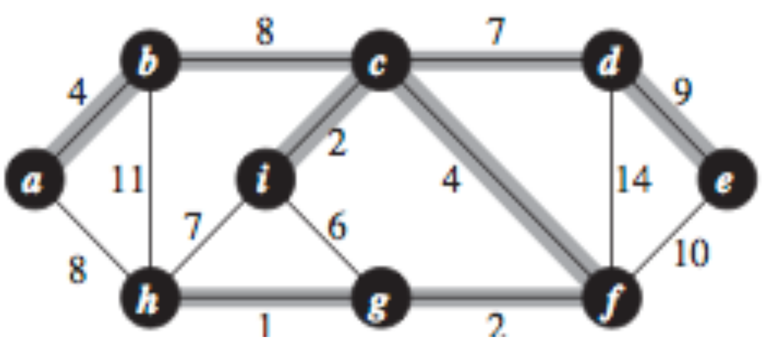
(f)





(h)





Loop 6-11 invariant

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the minimum spanning tree are those in $V - Q$.
3. For all vertices $v \in Q$, if $v.\pi \neq$ NIL, then $v.key < \infty$ and $v.key$ is the weight of a light edge $(v, v.\pi)$ connecting v to some vertex already placed into the minimum spanning tree.

Complexity PRIM algorithm

Row 1-5 : Build min heap takes $O(V)$

The while loop is executed $O(V)$ times and EXTRACT-MIN take $O(\log V)$ time. In total $O(V\log V)$.

The for loop is executed $O(E)$ in total, and the last operation is a DECREASE-KEY operation $O(\log V)$.

Prims algorithm takes $O(V\log V + E\log V) = O(E \log V)$ the same as Kruskal algorithm!

It can be improved to:

$O(E + V \log V)$

using for Q the data structure Fibonacci Heap

Semi external algorithm for MST-Kruskal

Semi external graph algorithms:
Requiring O(n) internal memory.

Semi external MST- Kruskal : the internal memory is big enough to contain the Union-Find (Find-Set) data structure but not the whole graph.

Semi external algorithm for MST-Kruskal

- Sort the edges using any optimal external (2-level model) sorting algorithm.
- Scan the edges in order of increasing weight, as for the normal algorithm.
- If an edge is selected for the MST, output it.
- Number of I/O's operations as Sorting.

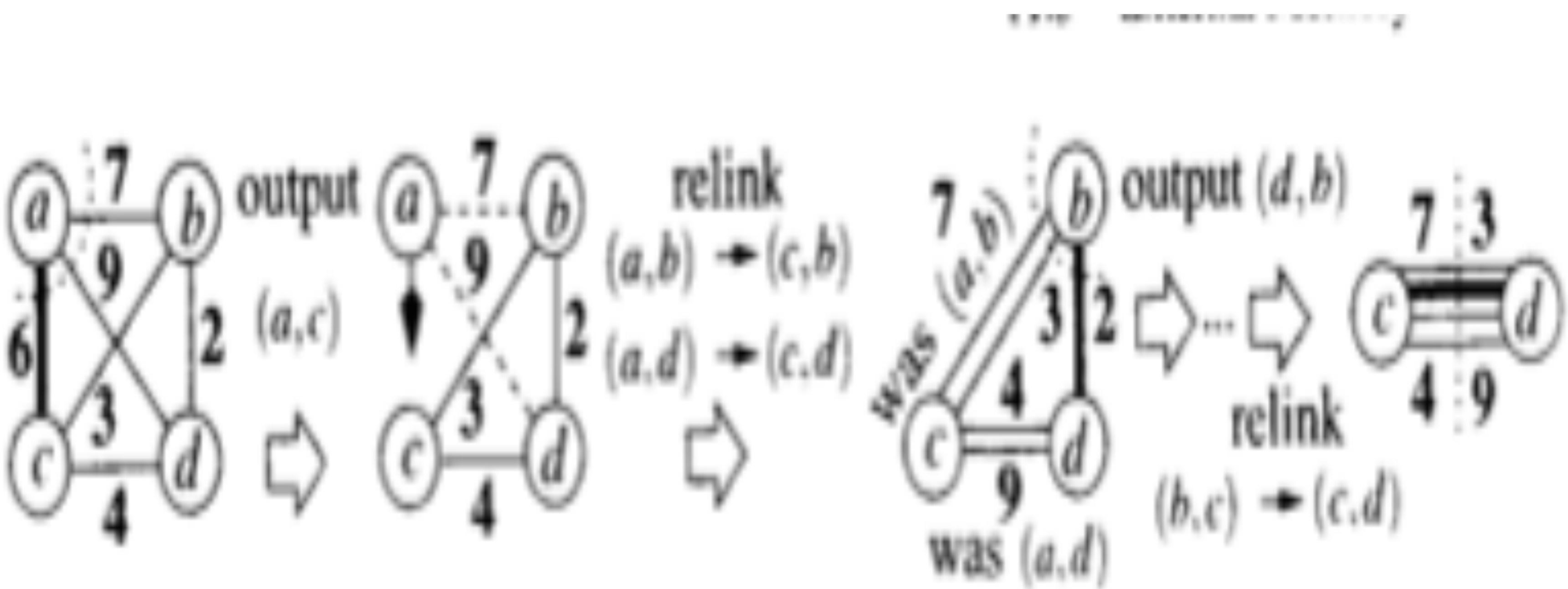
External algorithm for MST - Kruskal

Try to reduce the number of nodes!
Edge contraction:
Reduce the vertices from n to n'.

For v=1 **to** n-n'
find the lightest edge (u, v) incident to v and contract it

When, after contraction, n becomes n' adopt semi-external algorithm.

Contraction



Edge (c, a, 6) is the cheapest incident in a. Contract it: merge a and c to c. Relink a: (a, b, 7) becomes (c,b, 7), (a,d,9) becomes (c,d,9). Output (b,d,2) …

Contraction

Contraction can be implemented using a priority queue both to find the cheapest edge incident to v and to relink the other edges incident to v.

For each edge e=(u,v) we have to store the additional information:

{min(u,v), max(u,v), w(e), original e}

Edges are sorted according to weight and according to the lower endpoint.

Contraction is proportional to sum of the degrees of the nodes encountered.

Complexity: worst case $O(n^2)$ I/O's   n=|V|, m=|E|

Expected : O(sort(m) ln n/n') I/O's