Minimum Spanning Trees

Chapter 23

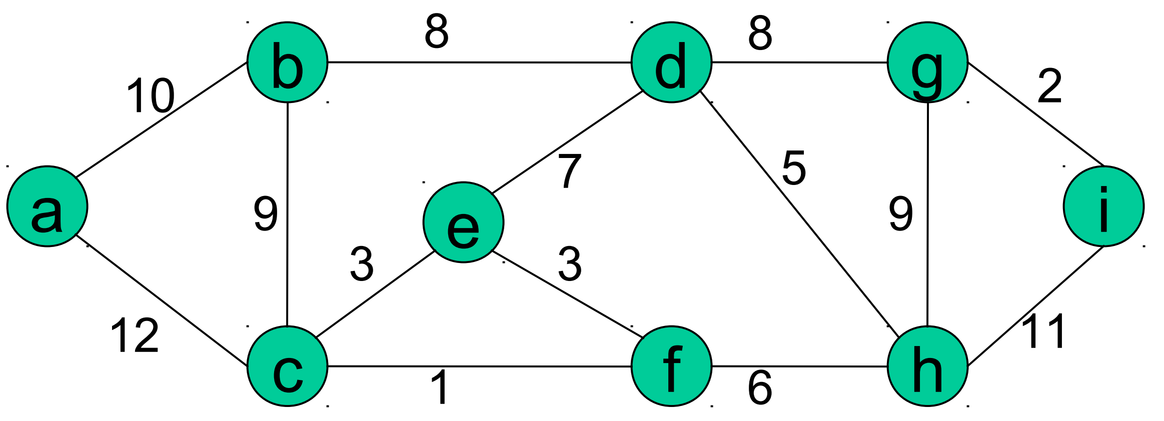
Spanning Tree

- What are the edges you need to keep the graph connected?
  - If you remove any edge, the graph becomes disconnected
- Minimum Spanning Tree
  - minimize the total weight of the edges
- Problem: Minimal set of roads needed to connect cities

Minimum Spanning Tree

- Undirected graph G = (V, E)
  - Weight w(u, v) on each edge (u, v) in E
  - Find T that is a subset of E such that
    - T connects all vertices, and
    - $w(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized

Minimum Spanning Tree



Growing an MST

- Properties of an MST:

- Building up a Solution
  -

Generic MST Algorithm

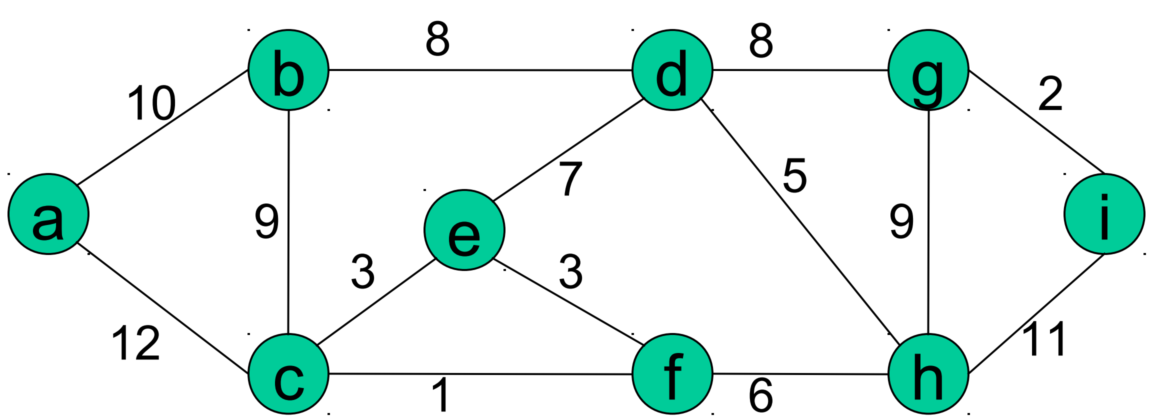
```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Proof via loop invariant (p 18-19)

- Initialization
- Maintenance
- Termination

Finding a Safe Edge

- How do we find safe edges?
- Edge (c,f) - Is it safe for A?



Finding a Safe Edge

Definitions

- Let S be a subset of V and A be a subset of E
  - A **cut** (S, V-S) is a partition of vertices into disjoint sets V and S-V
  - Edge (u,v) in E **crosses** cut (S,V-S) if one endpoint is in S and the other is in V-S
  - A cut **respects** A (A is a set of edges) if and only if no edge in A crosses the cut
  - An edge is a **light edge** crossing a cut if and only if its weight is minimum over all edges crossing the cut

Theorem

- Let A be a subset of some MST, (S,V-S) be a cut that respects A, and *(u,v) be a light edge crossing (S,V-S)*.
- Then….

Generic-MST

- So, in a generic MST
  - A is a *forest (set of trees. haha)* containing connected components.
  - Any safe edge merges two of these components into one.
  - Since an MST has exactly |V|-1 edges, the for loop iterates |V|-1 times.

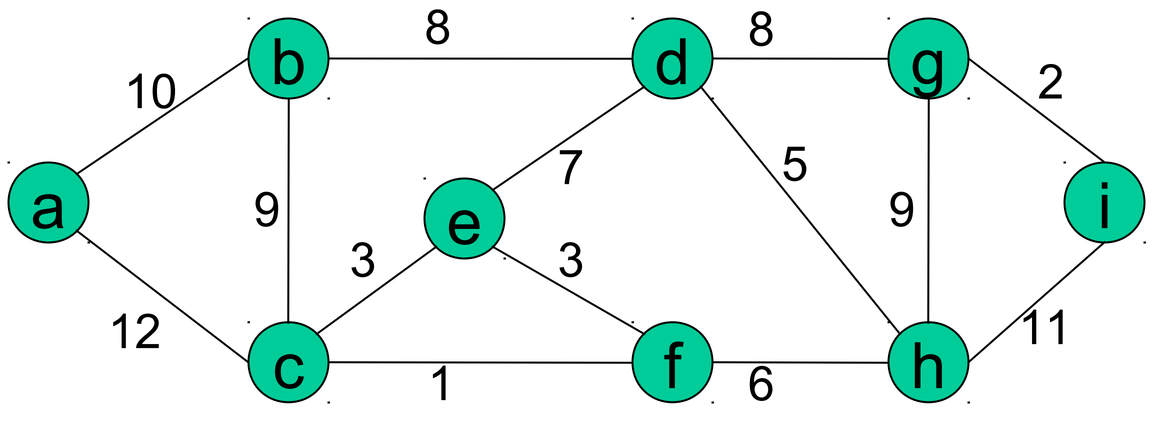
Kruskal's Algorithm

- G = (V,E) is a connected, undirected, weighted graph. w:E->R
  - Starts with each vertex being its own component
  - Repeatedly merges two components into one by choosing the light edge that connects them
  - Scans the set of edges in monotonically increasing order by weight
  - Uses a disjoint-set data structure to determine whether an edge connects vertices in different components

Kruskal(V,E,w)

```
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 (u, v) taken from the sorted list
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Example



Prim's Algorithm

- Builds one tree, so A is always a tree
- Starts from an arbitrary "root" r
- At each step, find a light edge crossing cut (V_A, $V-V_A$), where V_A = vertices that A is incident on. Add this edge to A

How to Find a Light Edge Quickly

- Use a priority queue Q:
  - Each object is a vertex in $V-V_A$
  - Key of v is minimum weight of any edge (u,v), where u is in V_A
  - Then the vertex returned by EXTRACT-MIN is v such that there exists u in V_A and (u,v) is a light edge crossing (V_A , $V-V_A$)
  - Key of v is infinity if v is not adjacent to any vertices in V_A

Prim's Algorithm

- The edges of A will form a rooted tree with root r:
  - r is given as an input to the algorithm, but it can be any vertex
  - Each vertex knows its parent in the tree by the attribute $\pi[v]$ = parent of v. $\pi[v]$ = NIL if v = r or v has no parent

PRIM(G,w,r)

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

Example