Chap 23 – Minimum Spanning Trees

23.1 Growing a minimum spanning tree

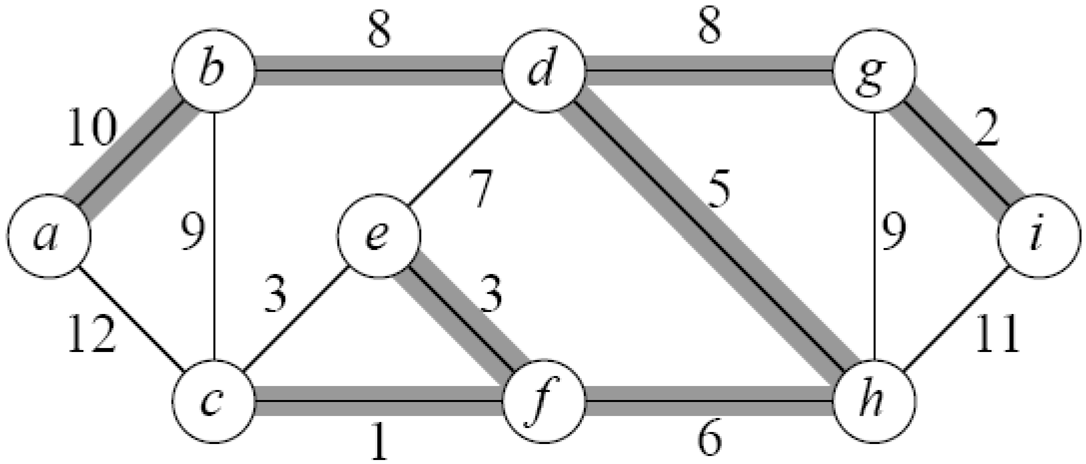
23.2 The algorithms of Kruskal and Prim

Minimum spanning tree

- Minimum spanning tree
  - An undirected graph $G = (V, E)$ with ***weight*** $w(u, v)$ on each edge $(u, v) \in E$
  - Find an MST $T \subseteq E$ such that
    1 T connects all vertices (T is a ***spanning tree***), and
    2 $w(T) = \sum_{(u,v)\in T} w(u, v)$ is minimized
  - A ***minimum spanning tree***, or ***MST***, is a spanning tree whose weight is minimum over all spanning trees.

Minimum spanning tree

- Minimum spanning tree
  - Some Properties of an MST
    - It has $|V| - 1$ edges.
    - It has no cycles
    - It might not be unique.
  - Example : Replacing (e, f) by (c, e) yields another MST.



23.1 Growing a minimum spanning tree

- Generic MST algorithm (Greedy strategy)
  - Initially, $A = \emptyset$ has no edges.
  - As we add edges to A, maintain a loop invariant

    Loop invariant

    A is a subset of some MST
  - Add only ***safe*** edges to maintain the loop invariant:

    If A is a subset of some MST, an edge (u, v) is safe iff $A \cup \{(u, v)\}$ is also a subset of some MST.

23.1 Growing a minimum spanning tree

- Generic MST algorithm
  - GENERIC-MST$(G = (V, E), w)$

    $A = \emptyset$

    while A is not a spanning tree

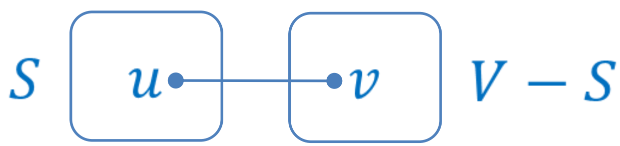
    find an edge $(u, v) \in E$ that is safe for A

    $A = A \cup \{(u, v)\}$

    return A
  - $G_A = (V, A)$ is a forest containing connected components. Each component is a tree. ($\because$ A is a subset of some MST.)
  - Initially, each component is a single vertex.
  - Any safe edge merges two components into one.
  - The loop iterates $|V| - 1$ times.

23.1 Growing a minimum spanning tree

- Generic MST algorithm: some definitions
  - Let $G = (V, E), S \subseteq V$ *and* $A \subseteq E$
  - A ***cut*** $(S, V - S)$ is a partition of V.
  - An edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$ if one endpoint $\in S$ and the other $\in V - S$.



  - A cut ***respects*** A if and only if no edge in A crosses the cut.
  - An edge is a ***light edge*** crossing the cut iff its weight is minimum over all edges crossing the cut.
    For a given cut, there can be > 1 light edge crossing it.

23.1 Growing a minimum spanning tree

- Generic MST algorithm
  - **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 the cut. Then, (u, v) is safe for A.

    Proof

    Let T be an MST that includes A.

    If T contains (u, v), done.

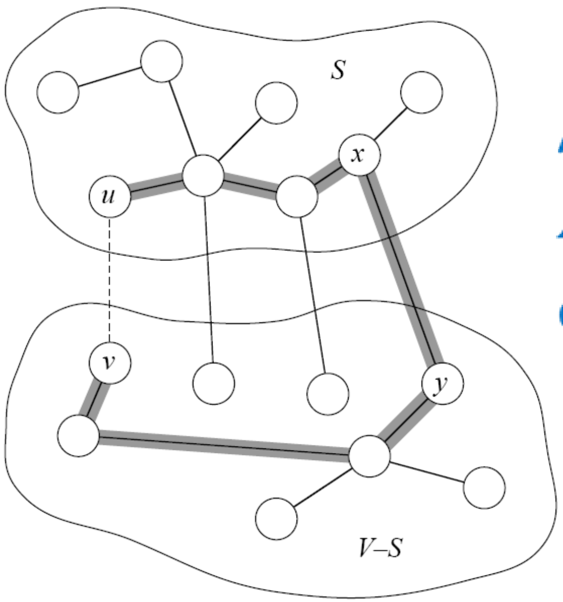
    If T doesn't contain (u, v), we'll construct a different MST T' that includes $A \cup \{(u, v)\}$.

    Recall: A tree has unique path between each pair of vertices

23.1 Growing a minimum spanning tree

- Generic MST algorithm
  - **THEOREM** (Cont'd)

    Since T is an MST, it contains a unique path p between u and v. Path p must cross the cut $(S, V - S)$ at least once. Let (x, y) be an edge of p that crosses the cut.



All edges shown are in T.

A is some subset of T, but it doesn't contain any edges that cross the cut.

23.1 Growing a minimum spanning tree

- Generic MST algorithm
  - **THEOREM** (Cont'd)

    Then, $w(u, v) \leq w(x, y) \because (u, v)$ is a light edge crossing the cut.

    Also, $(x, y) \notin A, \because$ the cut respects A.

    Now, let

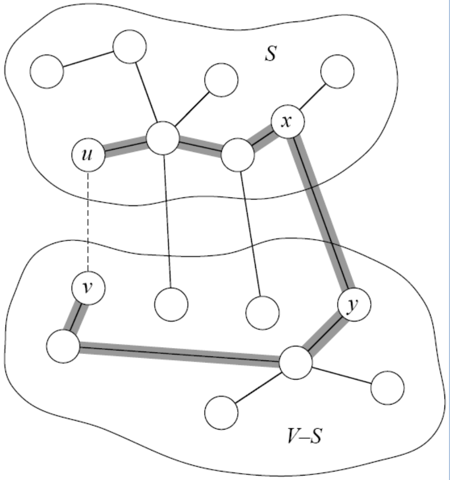
    $T' = T - \{(x, y)\} \cup \{(u, v)\}$

    Then, T' is a spanning tree.

    Since T is an MST and

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

    It follows that T' is an MST, too.



23.1 Growing a minimum spanning tree

- Generic MST algorithm
  - **Theorem** (Cont'd)

    We now show that (u, v) is safe for A.
    - $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T'$
    - $A \cup \{(u, v)\} \subseteq T'$
    - Since T' is an MST, (u, v) is safe for A.
  - **Corollary**

    If $C = (V_c, E_c)$ is a connected component in the forest $G_A = (V, A)$ and (u, v) is a light edge connecting C to some other component in G_A, i.e. (u, v) crosses the cut $(V_c, V - V_c)$, then (u, v) is safe for A.

    Proof Set $S = V_c$ in the theorem

23.2 The algorithms of Kruskal and Prim

- Kruskal's algorithm
  - Kruskal's algorithm immediately follows the corollary.
    - start with each vertex being its own component
    - repeatedly

      merge two components into one by choosing the light edge that connects them, i.e. the light edge crossing the cut between them
  - For the light edge, scan the set of edges in monotonically increasing order by weight.
  - Use a disjoint-set data structure to determine whether an edge connects vertices in different components

23.2 The algorithms of Kruskal and Prim

- Kruskal's algorithm
  - MST-KRUSKAL(G, w)

    $A = \emptyset$

    for each vertex $v \in G.V$ $\quad\cdots$ $|V|$ MAKE-SETS

    MAKE-SET(v)

    sort the edges of $G.E$ into nondecreasing order by w

    for each (u, v) taken from the sorted list

    if FIND-SET$(u) \neq$ FIND-SET(v) **then**

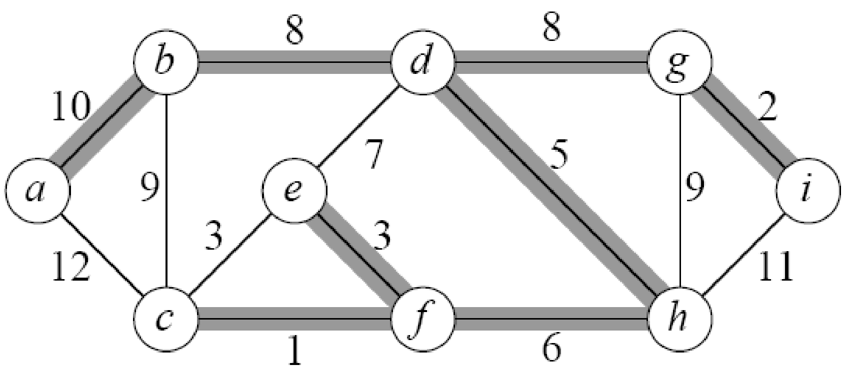
    $A = A \cup \{(u, v)\}$

    UNION(u, v) $\quad\cdots$ $O(E)$ FIND-SETS and UNIONS

    return A

23.2 The algorithms of Kruskal and Prim

- Kruskal's algorithm
  - Example



(c, f) : safe
(g, i) : safe
(e, f) : safe
(c, e) : reject
(d, h) : safe
(f, h) : safe
(d, e) : reject
(b, d) : safe
(d, g) : safe

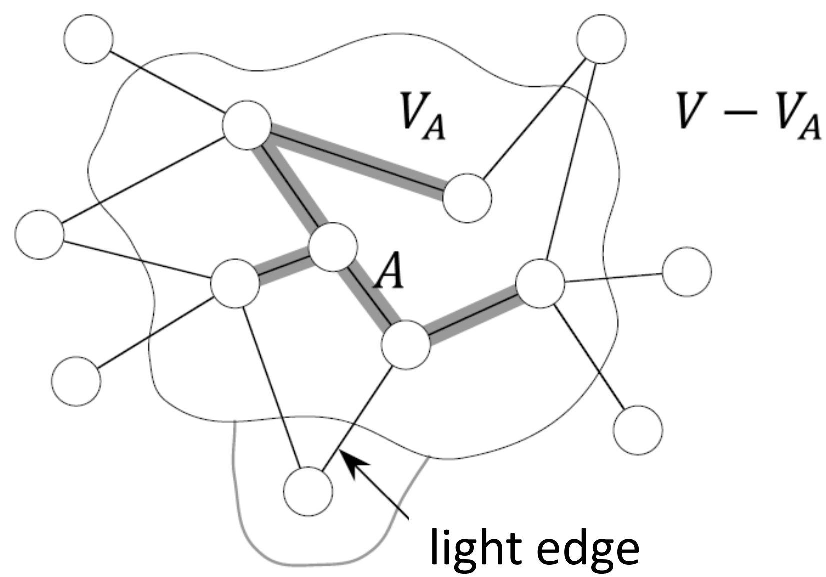
(b, c) : reject
(g, h) : reject
(a, b) : safe

23.2 The algorithms of Kruskal and Prim

- Kruskal's algorithm
  - With union by rank and path compression, the sequence of $O(V+E)$ disjoint-set operations on $|V|$ elements takes a time in $O((V+E)\alpha(V))$.
  - Total time: $O((V+E)\alpha(V)) + O(E \lg E)$
  - Since the graph is connected, $|E| \geq |V| - 1$
    Also, $\alpha(V) = O(\lg V) = O(\lg E)$
    Total time: $O(E \lg E) + O(E \lg E) = O(E \lg E)$
  - Another result
    $|E| \ \leq |V|^2 \Rightarrow \lg E = O(\lg V)$
    Total time: $O(E \lg V)$

23.2 The algorithms of Kruskal and Prim

- Prim's algorithm
  - Build one tree, so A is always a tree
  - Start 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



23.2 The algorithms of Kruskal and Prim

- Prim's algorithm
  - How to find the light edge quickly?

    Use a priority queue Q
    - Each object is a vertex u in $V - V_A$ (i.e. not in tree A)
    - $u.key$ is minimum weight of any edge $(u, v), v \in V_A$
    - Then, the vertex returned by EXTRACT-MIN is u such that there exists $v \in V_A$ and (u, v) is the light edge crossing $(V_A, V - V_A)$
    - $u.key = \infty$ if u is not adjacent to any vertices in V_A

23.2 The algorithms of Kruskal and Prim

- Prim's algorithm
  - The edges of A will form a rooted tree with root r:
    - r is given; it can by any vertex.
    - $v.\pi =$ parent of v
    - As algorithm progress,
      $A = \{(v, v.\pi : v \in V - \{r\} - Q)\}$
    - At termination, $Q = \emptyset$
      So, MST is $A = \{(v, v.\pi : v \in V - \{r\})\}$

23.2 The algorithms of Kruskal and Prim

- Prim's algorithm
  - MST-PRIM(G, w, r)

    $Q = \emptyset$

    for each $u \in G.V$

    $\quad u.key = \infty$; $u.\pi =$ NIL

    $\quad$ INSERT(Q, u) $\qquad \cdots |V|$ INSERTS

    DECREASE-KEY$(Q, r, 0)$ $\qquad // r.key = 0$

    while $Q \neq \emptyset$

    $\quad u =$ EXTRACT-MIN(Q) $\qquad \cdots |V|$ EXTRACT-MINS

    $\quad$ **for** each $v \in G.Adj[u]$

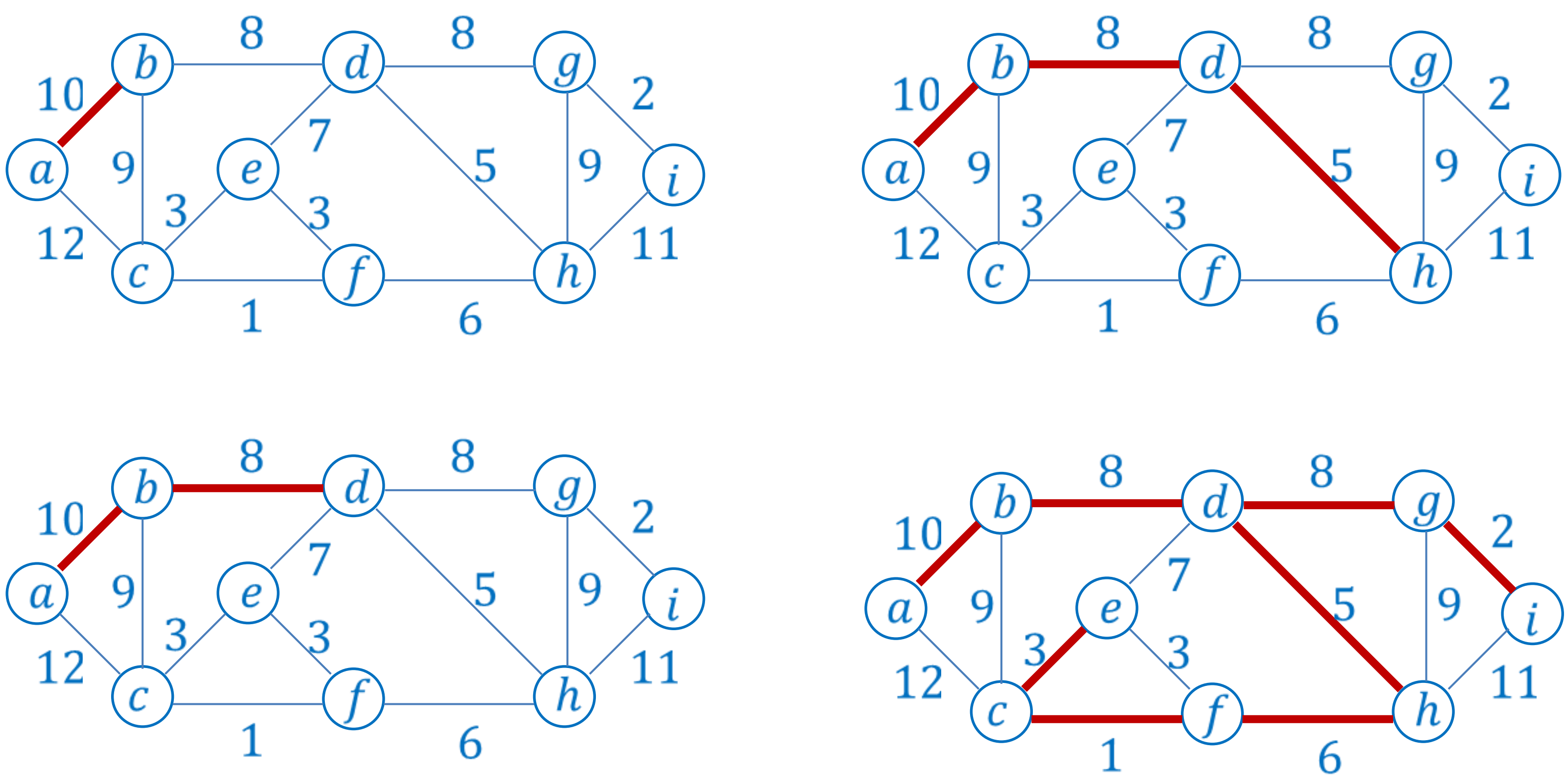
    $\quad\quad$ **if** $v \in Q$ and $w(u, v) < v.key$

    $\quad\quad\quad v.\pi = u$

    $\quad\quad\quad$ DECREASE-KEY$(Q, v, w(u, v)) \cdots \leq |E|$ DECREASE-KEY

23.2 The algorithms of Kruskal and Prim

- Example on Prim's algorithm



23.2 The algorithms of Kruskal and Prim

- Analysis of Prim's algorithm
  - Depend on how the priority queue Q is implemented
  - Suppose Q is a binary heap

    $|V|$ INSERTS + $|V|$ EXTRACT-MINS + ($\leq |E|$ DECREASE-KEYS)

    take a time in

    $O(V \lg V) + O(V \lg V) + O(E \lg V) = O(E \lg V)$

    $\because$ the graph is connected $\Rightarrow |E| \geq |V| - 1$
  - Suppose Q is a Fibonacci heap (Chap. 19)

    DECREASE-KEY can be done in $O(1)$ amortized time.

    Then, the operations take a time in

    $O(V \lg V) + O(V \lg V) + O(E) = O(E + V \lg V)$