Lecture 20: Minimum Spanning Trees

(CLRS 23)

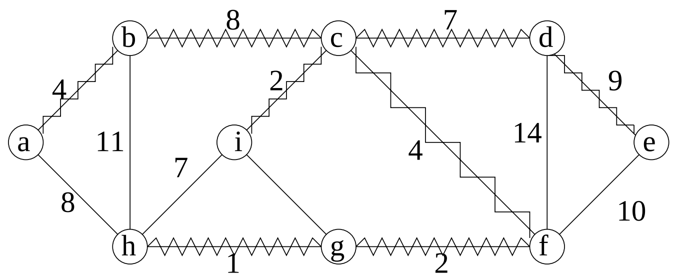
June 18, 2002

1 Graphs

- Last time we defined (weighted) graphs (undirected/directed) and introduced basic graph vocabulary (vertex, edge, degree, path, connected components, . . .)
- We also discussed adjacency list and adjacency matrix representation
  - We will use adjacency list representation unless stated otherwise ($O(|V|+|E|)$ space).
- We discussed $O(|V|+|E|)$ breadth-first (BFS) and depth-first search (DFS) algorithms and how they can be used to compute e.g. connected components, shortest path distances in unweighted graphs, and solve the topological sorting problem.
- We will now start discussing more complicated problems/algorithms on weighted graphs.

2 Minimum Spanning tree (MST)

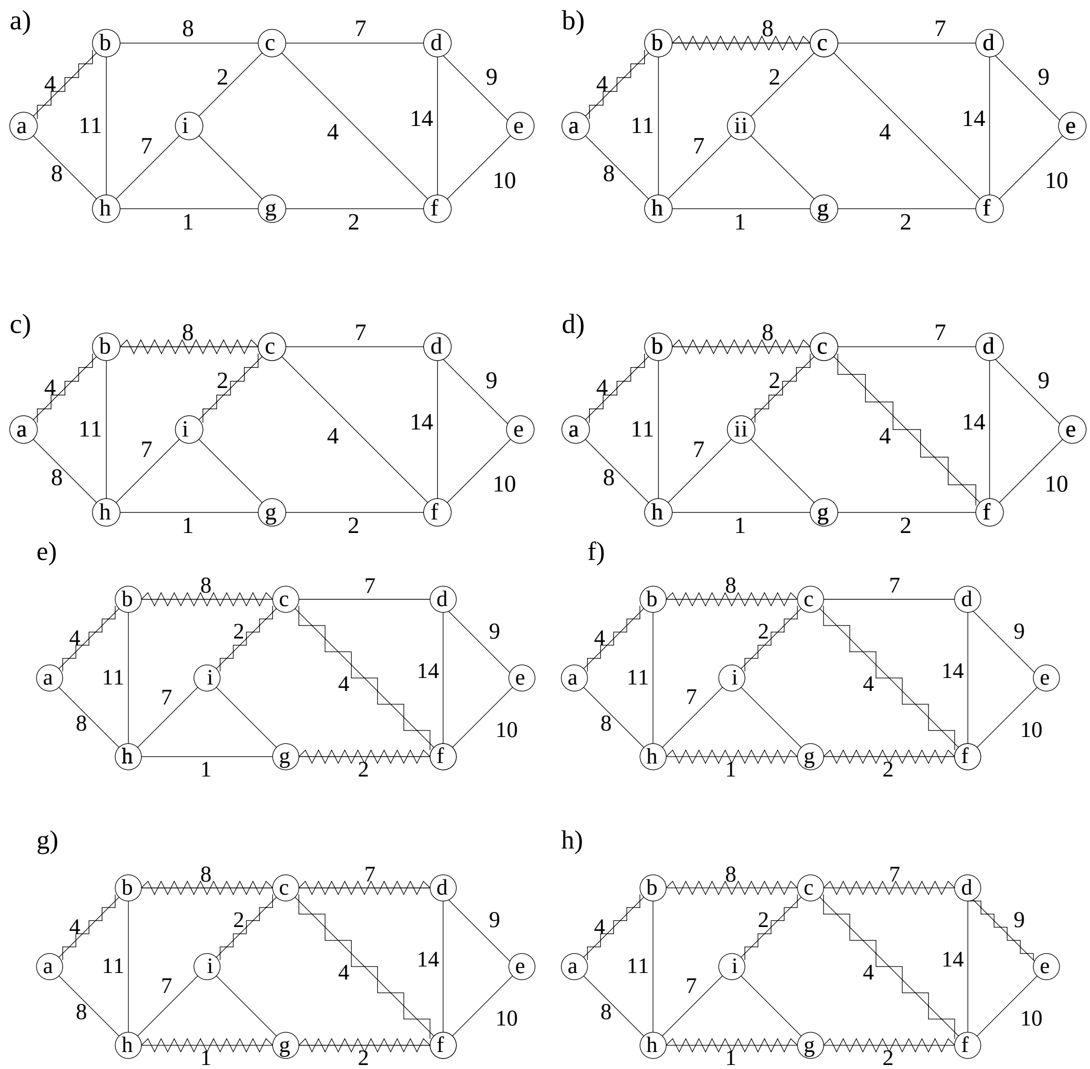
- Problem: Given connected, undirected graph $G = (V, E)$ where each edge (u, v) has weight $w(u, v)$. Find acyclic set $T \subseteq E$ connecting all vertices in V with minimal weight $w(T) = \sum_{(u,v)\in T} w(u, v)$
- Note: Problem is to find a *spanning tree* (acyclic set connecting all vertices) of *minimal weight.* (we use *minimum spanning tree* as short for *minimum weight spanning tree*).
- MST problem has many applications
  - For example, think about connecting cities with minimal amount of wire (cities are vertices, weight of edges are distances between city pairs).
- Example:



  - Weight of MST is $4+8+7+9+2+4+1+2 = 37$
  - MST is not unique: e.g. (b, c) can be exchanged with (a, h)

2.1 PRIM's algorithm

- *Greedy* algorithm for computing MST:
  - Start with spanning tree containing arbitrary vertex r and no edges
  - Grow spanning tree by repeatedly adding minimal weight edge connecting vertex in current spanning tree with a vertex not in the tree
- On the example graph, the greedy algorithm would work as follows (starting at vertex a):



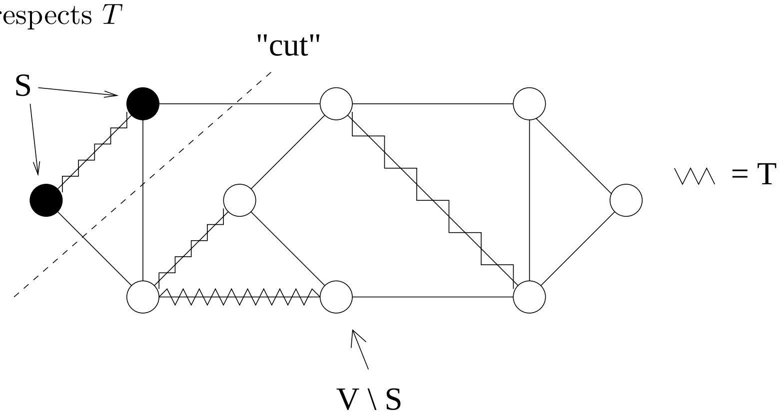
- Implementation:
  - To find minimal edge connected to current tree we maintain a priority queue on vertices not in the tree. The key/priority of a vertex is the weight of minimal weight edge connecting it to the tree. (We maintain pointer from adjacency list entry of v to v in the priority queue).

```
PRIM(r)

For each v ∈ V DO
    INSERT(Q, v, ∞)
OD
CHANGE(Q, r, 0)
WHILE Q not empty DO
    u = DELETEMIN(Q)
    For each (u, v) ∈ E DO
        IF v ∈ Q and w(u, v) < key(v) THEN
            visit[v] = u
            CHANGE(Q, v, w(u, v))
        FI
    OD
OD
```

- Analysis:
  - While loop runs $|V|$ times $\Rightarrow$ we perform $|V|$ DELETEMIN's
  - We perform at most one CHANGE for each of the $|E|$ edges
    $\Downarrow$
    $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$ running time.
- Correctness:
  - As discussed previously, when designing a greedy algorithm the hard part is often to prove that it works correctly.
  - We will prove a Theorem that allows us to prove the correctness of a general class of greedy MST algorithms:
    Some definitions
    * A *cut* S is a partition of V into sets S and $V \setminus S$
    * A *edge* (u, v) *crosses a cut* S if $u \in S$ and $v \in V \setminus S$ *or* $v \in S$ and $u \in V \setminus S$
    * A *cut* S *respects a set* $T \subseteq E$ if no edge in T crosses the cut

    Example: Cut S respects T

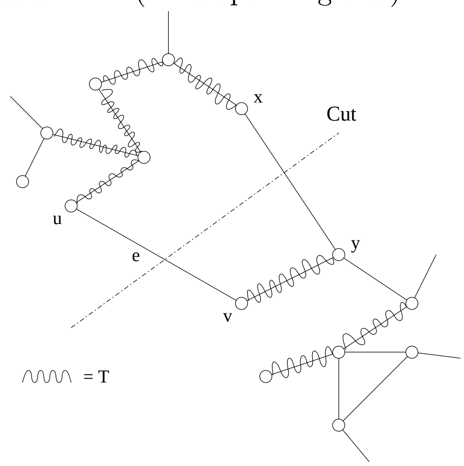


- *Theorem*: If $G = (V, E)$ is a graph such that $T \subseteq E$ is subset of some MST of G, and S is a cut respecting T **then** there is a MST for G containing T and the minimum weight edge $e = (u, v)$ crossing S.

- Note: Correctness of Prim's algorithm follows from the Theorem by induction—cut consist of current spanning tree.

- Proof:

  - Let T^* be MST containing T
  - If $e \in T^*$ we are done
  - If $e \notin T^*$:
    * There got to be (at least) one other edge $(x, y) \in T^*$ crossing the cut S such that there is a unique path from u to v in T^* (T^* is spanning tree)



    * This path together with e forms a cycle
    * If we remove edge (x, y) from T^* and add e instead, we still have spanning tree
    * New spanning tree must have same weight as T^* since $w(u, v) \leq w(x, y)$
      $\Downarrow$
      There is a MST containing T and e.

- The Theorem allows us to describe a very abstract greedy algorithm for MST:

```
T = ∅
While |T| ≤ |V| − 1 DO
    Find cut S respecting T
    Find minimal edge e crossing S
    T = T ∪ {e}
OD
```

  - Prim's algorithm follows this abstract algorithm.

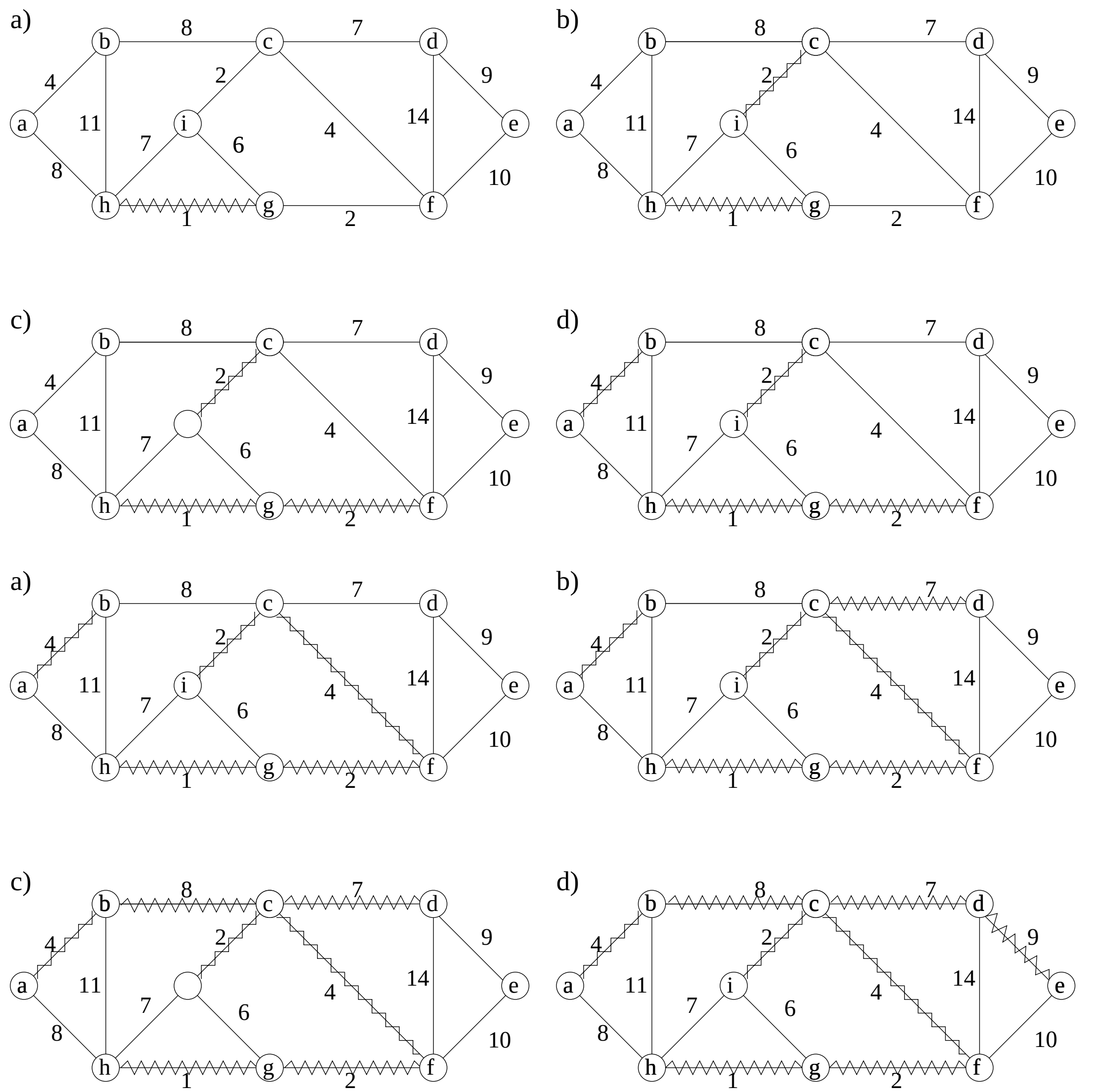
3 Kruskal's Algorithm

- Kruskal's algorithm is another implementation of the abstract algorithm.

- Idea in Kruskal's algorithm:

  - Start with $|V|$ trees (one for each vertex)
  - Consider edges E in increasing order; add edge if it connects two trees

- Example:



- Correctness of Kruskal's algorithm follows from Theorem: If minimal edge connects two trees then a cut respecting the current set of edges exists (cut consisting of vertices in one of the trees)

- Implementation:

```
KRUSKAL

T = ∅
FOR each vertex v ∈ V DO
    Make-Set(v)
OD
Sort edges of E in increasing order by weight
FOR each edge e = (u, v) ∈ E in order DO
    IF Find-Set(u) ≠ Find-Set(v) THEN
        T = T ∪ {e}
        Union-Set(u, v)
    FI
OD
```

  - We need (Union-Find) data structure that supports:
    * Make-set(v): Create set consisting of v
    * Union-set(u, v): Unite set containing u and set containing v
    * Find-set(u): Return unique representative for set containing u
  - We use $O(|E| \log |E|)$ time to sort edges and we perform $|V|$ Make-Set, $|V| - 1$ Union-Set, and $2|E|$ Find-Set operations.
  - Next time we will discuss a simple solution to the *Union-Find problem* (maintain set system under Find-Set and Union-Set) such that Make-Set and Find-Set take $O(1)$ time and Union-Set takes $O(\log V)$ time amortized.

    $\Downarrow$

    Kruskal's algorithm runs in time $O(|E| \log |E| + |V| \log |V|) = O((|E| + |V|) \log |E|) = O(|E| \log |V|)$ like Prim's algorithm.

- Note:
  - Prim's algorithm can be improved to $O(|V| \log |V| + |E|)$ using another heap (Fibonacci heap)
  - Very recently an $O(|V| + |E|)$ *randomized* minimum spanning tree algorithm has been developed.