# CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

## Business

- SA6 due 11:59pm, Dec. 1
  - SA6 involves working through an example of the algo we covered last Monday
- PS4 due already
- PS5 out, due Dec. 9
  - PS5 will be the final PS for the semester
- PS3, SA4, SA5 grading update
- If you have any questions or comments, *please see me about feedback on your graded PS3's*
- Project 4 due 11:59pm, Monday, Dec. 12
  - Tell me your team by end of day *today*
  - Intended team size: 4 (but talk to me if you'd prefer to work with a smaller team size)

See CLRS, Sec. B.5.1 and B.5.2

See also the Rush album ***Hemispheres***, which many people may find even more dense and inaccessible than the CLRS textbook.

# The Trees

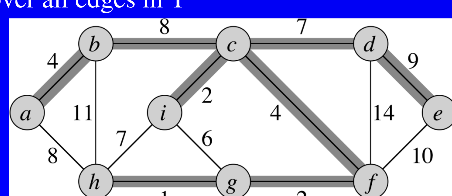
- A *tree* (sometimes called a *free tree*) is an acyclic, connected, undirected graph
  - We've seen *rooted trees* such as binary trees before, but from a more general graph-oriented perspective, trees do not need to have roots

**A collection of (possibly) disconnected trees is called a *forest*. Really.**

- Important properties of (free) trees—the below statements are all equivalent for undirected graph G = (V, E):
  - G is a free tree
  - Any two vertices in G are connected by a unique simple path
  - G is connected, but if any edge is removed, the resulting graph is disconnected
  - G is connected and |E| = |V|-1
  - G is acyclic and |E| = |V|-1
  - G is acyclic, but if any edge is added to G, the resulting graph has a cycle

# Minimum Spanning Trees (MSTs)

- Given a connected undirected graph G = (V,E), an acyclic subgraph that connects all the vertices in V is a *spanning tree* of G
  - It's a tree; and it covers ("spans") all the vertices of G
  - For network G, represents unique connections / paths between each pair of nodes in G
- Consider the *minimum spanning tree* (MST) problem: given weighted, undirected, connected graph G, find a spanning tree T with minimal total weight over all edges in T



"In the not too distant future…"
-- MST 3K

# A Generic MST Algorithm

- Minimum spanning trees can be grown one edge at a time

  - *Safe* here means an edge that can be added without violating the property that A is a subgraph of an MST.
  - Digression: How do we argue correctness of the 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
```

- Some vocabulary
  - A *cut* (S,V-S) of an undirected graph G=(V,E) is a partition of V
  - An edge (u,v) *crosses* a cut if u is in S and v is in V-S
  - A cut *respects* a set of edges if no edge in the set crosses the cut
  - A *light edge* is a minimum-weight edge satisfying a property (e.g., a light edge that crosses a cut)

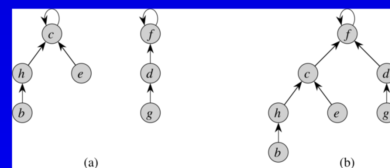
How can this vocab be used to describe an MST algorithm?

# Greedy MST Algorithms

- Greedy strategy for building MSTs: Add the best edge (from edge set E of graph G); repeat until an MST is built
  - Overall structure: Turn a forest (some trees have only 1 node) into a tree by adding light edges connecting separate components
- Question: What is the best edge (the greedy choice) to add?
  - A possibility: Pick the least-weight edge from E that connects two separate components
    - Possibly results in multiple trees growing in the forest, but all will be connected by the end of the algorithm
    - I.e., maintains a *disjoint set* of sub-trees

Invariant: All subgraphs are trees. How do we know?

# Data Structures Flashback: Disjoint Sets / Union-Find




- A *union-find* data structure is used to maintain a collection of disjoint sets
- Operations on disjoint sets:
  - *Find*: Given *v*, find component (set) containing *v*
  - *Union*: Given components *A*, *B*, replace them by their union $A \cup B$
- Representation: a disjoint-set forest of rooted trees
  - Union operation joins trees *A* and *B* into a new rooted tree
  - Find operation gives the root of the tree containing *v*
- Use this representation with *union-by-rank* and *path-compression* heuristics as data structure for MST algorithm

**See CLRS Chapter 21.3 for details of implementation and run-time complexity.**

# Kruskal's Algorithm

- MST possibility: Pick the least-weight edge from E that connects two separate components
  - Disjoint-set data structure maintains forest of MST sub-trees
  - Efficient to determine whether two vertices are already connected
  - Start by sorting edges, least-weight first, and take edges in that order, as long as they connect separate components

**… i.e., add light edges that connect sub-trees**

```
KRUSKAL(G, w)
  A = Ø
  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
```

# Kruskal's Algorithm, kontinued

- What's a *correctness* argument for Kruskal's algorithm?

```
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
```

- What's a complexity argument for it?

# Kruskal's Algorithm, kontinued

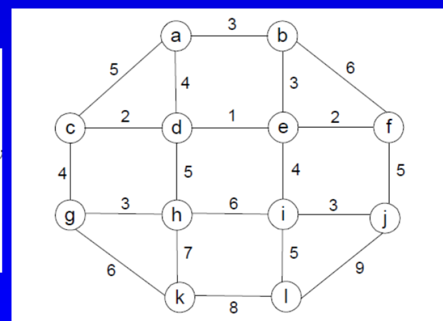
- What's a *complexity* argument for Kruskal's algorithm?

```
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
```

  - Depends on the complexity of Find-Set and Union, from previous slide: O((V + E) \alpha(E))
  - And O(E lg E) to sort E, so [O(E lg E), thus…] O(E lg V), total

# Example Exercise: Greedy Algorithms & MSTs

```
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
```




- Apply Kruskal's algorithm to this graph, to find a minimum spanning tree.
- (Break ties, where applicable, by alphabetical ordering on the endpoints of edges.)