

# Minimum Spanning Trees

... or how to bring the world  
together on a budget

# Remembering Graphs

- *Relationships* between entities
- Weighted edges *quantify* relationships

# Carving Trees from Graphs

- BFS creates tree (or forest)
  - BF-tree contains shortest paths
- DFS creates tree (or forest)

# Spanning Tree

- Derived from *connected* graph
- Contains all vertices
- Is a tree (no cycles)
- Obviously, there could be many

# The Math View

- $\mathbf{G}=(V,E)$  is a connected graph
- $\mathbf{T}=(V,E)$  is a spanning tree on  $\mathbf{G}$  iff
  - $V_T = V_G, E_T \subseteq E_G$
  - $\mathbf{T}$  contains no cycles

# RETURN OF THE SEARCH!

- BF tree is a spanning tree
- DF tree is a spanning tree

# Philosophical View

- *A tree on a graph is*
  - **filter** on the set of relationships
  - enable discovery of new relationships
  - or meta-relationships
- For example, BF and DF trees

# Spanning Trees and Weights

- Tree in weighted graph has an associated weight
- Weight is just sum of weights of edges in the tree

# Minimum Spanning Tree

- An MST on **G**
  - is a spanning tree
  - weight  $\leq$  every other ST on **G**
- NOTE: any connected subgraph with min weight **must** be a tree
  - Removing a cyclic edge
    - Preserves connectedness
    - Reduces weight

# Obvious application

- Running electrical wires or water pipes
  - Assuming shorter wires/pipes are cheaper
- Car / airplane repair
  - Vertices = states, edges = repairs  
(open hood, remove air filter, remove carburator...)
  - Eliminating edges simplifies the design

# A “Cut” of $G$

- A *cut* partitions vertices  $(S, V-S)$
- Edge  $(u, v)$  crosses the cut if
  - $u$  in  $S$
  - $v$  in  $V-S$
- A cut respects  $A \subseteq E$  if none of the edges in  $A$  cross the cut

# “Light” Edges

- A light edge satisfying a property
  - Has min weight for all edges that satisfy the property

# Generic MST Algorithm (setup)

- Let **A** be a subset of **E**
- **A** is a subset of some MST
- $(u,v)$  is a *safe edge* for **A** iff
  - $A \cup \{(u,v)\} \subseteq \text{MST}$

# Generic-MST( $G, w$ )

1.  $A \leftarrow \emptyset$
2. **while**  $A$  is not a spanning tree
3.     **do** find edge  $(u,v)$  that is safe for  **$A$**
4.      $A \leftarrow A \cup \{(u,v)\}$
5. **return**  $A$

# Generic MST Loop Invariant

- Initialization:
  - Line 1
  - Trivially satisfies
- Maintenance:
  - Lines 2-4
  - Only add safe edges
- Termination:
  - All edges are in MST
  - Therefore, A (line 5) must be an MST

# Theorem 1

## (recognizing safe edges)

- Let  $\mathbf{G}=(\mathbf{V},\mathbf{E})$  is connected, undirected, and weighted
- Let  $\mathbf{A}$  be a subset of  $\mathbf{E}$  (in some MST)
- Let  $(\mathbf{S},\mathbf{V}-\mathbf{S})$  be a cut of  $\mathbf{G}$  that respects  $\mathbf{A}$
- Let  $(u,v)$  be a light edge crossing  $(\mathbf{S}, \mathbf{V}-\mathbf{S})$
- $(u,v)$  is safe for  $\mathbf{A}$

# Proof of Theorem 1

- See CLRS 23.1 (p 563-565)

# Generic Uselessness

- Of course, the generic algorithm isn't very useful
- The hard part is finding a safe edge.
- Two *greedy* algorithms
  - Kruskal's algorithm
  - Prim's algorithm

# MST-Kruskal( $G, W$ )

1.  $A = \{\}$
2. **for** each  $v$  in  $V$
3.     MAKE-SET( $v$ )
4. sort  $E$  in nondecreasing order by weight
5. **for**  $(u,v)$  in sorted- $E$
6.     **if** FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7.          $A = A + \{(u,v)\}$
8.     UNION( $u,v$ )
9. **return**  $A$

# Kruskal: Intuition

- Create disjoint sets for each  $v$
- In the loop
  - Find light edge (using sort)
  - The disjoint sets represent the cut (partition)
  - The cut  $(A, V-A)$  respects  $A$
  - $(u,v)$  is light edge crossing cut
  - $(u,v)$  is safe for  $A$

# Kruskal's Running Time

- MAKE-SET takes  $O(v)$  time
- Sorting edges takes  $O(E \log E)$
- Find set for each  $e$   $O(E \alpha(V))$
- Union is  $O(1)$
- Total time is  $O(E \log E)$
- (or  $O(E \log V)$  )

# Prim's Algorithm

- Starts with arbitrary root
- In loop
  - Adds a light edge  $(u,v)$ 
    - $u$  in the tree
    - $v$  out of the tree
  - Uses a min priority queue for  $v$
- $\text{MST-PRIM}(G,W,r)$  on next slide

1. **for** each  $u$  in  $V$
2.      $\text{key}[u] = \text{INFINITY}$
3.      $\text{p}[u] = \text{NULL}$
4.      $\text{key}[r] = 0$
5.      $Q = \text{CREATE-MIN-QUEUE}(V)$
6.     **while**  $Q \neq \{\}$
7.          $u = \text{EXTRACT-MIN}(Q)$
8.         **for** each  $v$  adjacent to  $u$
9.             **if**  $v$  is in  $Q$  and  $w(u,v) < \text{key}[v]$
10.                  $\text{p}[v] = u$
11.                  $\text{key}[v] = w(u,v)$
12.      $A = \{ (v, \text{p}[v]) : v \text{ is in } V - \{r\} \}$
13.     **return**  $A$

# Prim's Running Time

- If we use a binary heap
  - $O(E \lg V)$
- Fibonacci heap
  - $O(E + V \lg V)$
  - (See CLRS 20 and end of 23.2)