Objectives

Greedy Algorithms

- Minimum spanning tree
- Union-Find Data Structure
- Clustering
- Data Compression

Laying Cable

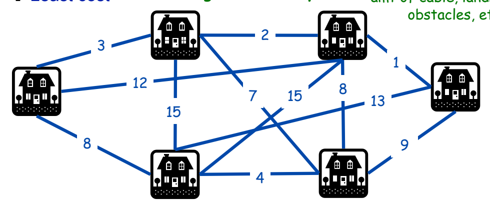
Comcast knows how to make money and how to save money

They want to lay cable in a neighborhood

- Reach all houses
- Least cost

Cost of laying cable between houses depends on amt of cable, landscaping, obstacles, etc.

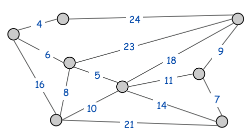
Neighborhood Layout



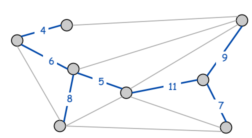
Minimum Spanning Tree

Given a connected graph G = (V, E) with positive edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a *spanning tree* whose sum of edge weights is *minimized*

- Spanning tree: spans all nodes in graph



G = (V, E)



T, $\Sigma_{e \in T}\, c_e = 50$

Minimal Spanning Tree: Why a Tree?

Proof by Contradiction.

Assume have a minimal solution *V* that is not a tree, i.e., it has a cycle

Contains edges to all nodes because solution must be connected (spanning)

Remove an edge from the cycle

Can still reach all nodes (could go "long way around")

But at lower cost

Contradiction to our minimal solution

Greedy Algorithms

All three algorithms produce a MST

Kruskal's algorithm. Start with $T = \phi$. Consider edges in ascending order of cost. Insert edge *e* in *T* unless doing so would create a cycle.

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge *e* from *T* unless doing so would disconnect *T*.

Prim's algorithm. Start with some root nodes and greedily grow a tree *T* from *s* outward. At each step, add the cheapest edge *e* to *T* that has exactly one endpoint in *T*.

- Similar to Dijkstra's (but simpler)

What do these algorithms have/do/check in common?

What Do These Algorithms Have in Common?

When is it safe to include an edge in the minimum spanning tree?

Cut Property

When is it safe to eliminate an edge from the minimum spanning tree?

Cycle Property

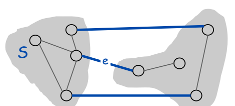
Cut and Cycle Properties

Simplifying assumption: All edge costs c_e are distinct

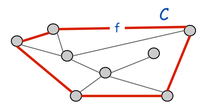
➡ MST is unique

Cut property. Let *S* be any subset of nodes, and let *e* be the min cost edge with exactly one endpoint in *S*. Then the MST contains *e*.

Cycle property. Let *C* be any cycle, and let *f* be the max cost edge belonging to *C*. Then the MST does not contain *f*.



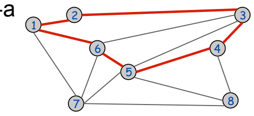
e is in the MST



f is not in the MST

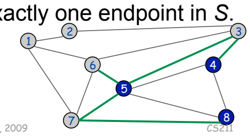
Cycles and Cuts

Cycle. Set of edges that form a-b, b-c, c-d, …, y-z, z-a



Cycle *C* = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

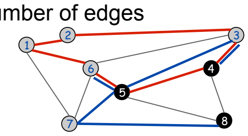
Cutset. A *cut* is a subset of nodes *S*. The corresponding *cutset* *D* is the subset of edges with exactly one endpoint in *S*.



Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

Cycle-Cut Intersection

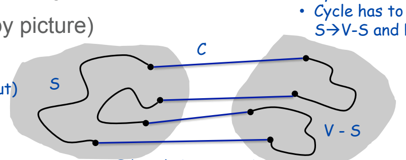
Claim. A cycle and a cutset intersect in an even number of edges



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

- Cycle all in S or V-S
- Cycle has to go from S→V-S and back

Pf. (by picture)



Edges link to not-Cut

Cut Property: OK to Include Edge

Simplifying assumption. All edge costs c_e are distinct

Cut property. Let S be any subset of nodes, and let *e* be the min cost edge with exactly one endpoint in *S*. Then the MST T* contains *e*.

Pf.

Cut Property: OK to Include Edge

Simplifying assumption. All edge costs c_e are distinct

Cut property. Let S be any subset of nodes, and let *e* be the min cost edge with exactly one endpoint in *S*. Then the MST T* contains *e*.

Pf. (exchange argument)

- Suppose there is an MST T* that does not contain *e*
  - What do we know about T?
  - What do we know about the nodes *e* connects?

Cut Property: OK to Include Edge

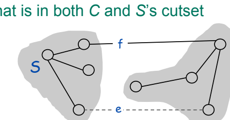
Simplifying assumption. All edge costs c_e are distinct

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST T* contains *e*

Pf. (exchange argument)

- Suppose there is an MST T* that does not contain *e*
- Adding *e* to T* creates a cycle *C* in T*
- Edge *e* is in cycle *C* and in cutset corresponding to *S*
  - ⇒ There exists another edge, say *f*, that is in both *C* and *S*'s cutset

AND ?!?



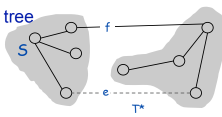
Cut Property: OK to Include Edge

Simplifying assumption. All edge costs c_e are distinct

Cut property. Let *S* be any subset of nodes, and let *e* be the min cost edge with exactly one endpoint in *S*. Then the MST T* contains *e*

Pf. (exchange argument)

- Suppose there is an MST T* that does not contain *e*
- Adding *e* to T* creates a cycle *C* in T*
- Edge *e* is in cycle *C* and in cutset corresponding to *S*
  - ⇒ there exists another edge, say *f*, that is in both *C* and S's cutset
- T' = T* ∪ { e } - { f } is also a spanning tree
- Since $c_e < c_f$, cost(T') < cost(T*)
- This is a contradiction. ▪



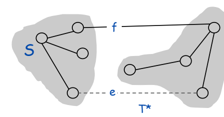
Cut Property: OK to Include Edge

Simplifying assumption. All edge costs c_e are distinct

Cut property. Let *S* be any subset of nodes, and let *e* be the min cost edge with exactly one endpoint in *S*. Then the MST T* contains *e*

Implication: Can always include an edge (meeting criteria) with minimum cost

- Many different configurations of *S*



Cycle Property: OK to Remove Edge

Simplifying assumption. All edge costs c_e are distinct

Cycle property. Let *C* be any cycle in *G*, and let *f* be the max cost edge belonging to *C*. Then the MST *T** does not contain *f*.

Ideas about approach?

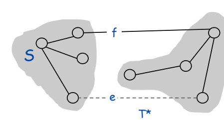
Cycle Property: OK to Remove Edge

Simplifying assumption. All edge costs c_e are distinct

Cycle property. Let *C* be any cycle in *G*, and let *f* be the max cost edge belonging to *C*. Then the MST T* does not contain *f*.

Pf. (exchange argument)

- Suppose *f* belongs to T*, and let's see what happens.
  - What happens if we deleted *f* from T*?



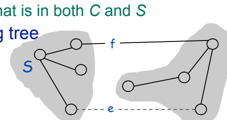
Cycle Property: OK to Remove Edge

Simplifying assumption. All edge costs c_e are distinct

Cycle property. Let *C* be any cycle in *G*, and let *f* be the max cost edge belonging to *C*. Then the MST T* does not contain *f*.

Pf. (exchange argument)

- Suppose *f* belongs to T*, and let's see what happens.
- Deleting *f* from T* creates a cut *S* in T*.
- Edge *f* is both in the cycle *C* and in the cutset *S*
  - ⇒ There exists another edge, say *e*, that is in both *C* and *S*
- T' = T* ∪ { e } - { f } is also a spanning tree
- Since $c_e < c_f$, cost(T') < cost(T*).
- This is a contradiction. ▪



Prim's Algorithm

[Jarník 1930, Dijkstra 1957, Prim 1959]

Start with some root node *s*

Greedily grow a tree T from *s* outward

At each step, add the cheapest edge *e* to T that has exactly one endpoint in T.

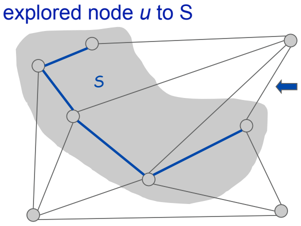
How can we prove its correctness?

Prim's Algorithm: Proof of Correctness

Initialize *S* = any node

Apply ***cut property*** to *S*

- Add min cost edge in *S*'s cutset to *T*
- Add one new explored node *u* to S



Implementation: Prim's Algorithm

Similar to Dijkstra's algorithm

Maintain set of explored nodes *S*

For each unexplored node *v*, maintain attachment cost a[v] = cost of cheapest edge *v* to a node in *S*

- O(m log n) with a heap

```
foreach (v ∈ V) a[v] = ∞
Initialize an empty priority queue Q
foreach (v ∈ V) insert v onto Q
Initialize set of explored nodes S = φ
while (Q is not empty)
   u = delete min element from Q
   S = S ∪ { u }
   foreach (edge e = (u, v) incident to u)
       if ((v ∉ S) and (c_e < a[v]))
           decrease priority a[v] to c_e
```

Update attachment cost

Kruskal's Algorithm [1956]

Start with *T* = φ

Consider edges in *ascending* order of cost

Insert edge *e* in *T* unless doing so would create a cycle

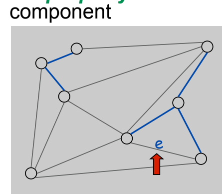
How can we prove its correctness?

Kruskal's Algorithm: Proof of Correctness

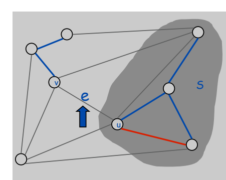
Consider edges in ascending order of weight

Case 1: If adding *e* to T creates a cycle, discard *e* according to ***cycle property***

Case 2: Otherwise, insert *e* = (*u*, *v*) into T according to ***cut property*** where S = set of nodes in *u*'s connected component



Case 1



Case 2

Implementing Kruskal's Algorithm

What is tricky about implementing Kruskal's algorithm?

Implementing Kruskal's Algorithm

What is tricky about implementing Kruskal's algorithm?

- How do we know when adding an edge will create a cycle?
  - What are the properties of an undirected /its nodes when adding an edge will create a cycle?

Union-Find Data Structure

Keeps track of a graph as edges are added

- Cannot handle when edges are deleted

Maintains disjoint sets

- E.g., graph's connected components

Operations:

- Find(u): returns name of set containing u
  - How utilized to see if two nodes are in the same set?
  - Goal implementation: O(log n)
- Union(A, B) : merge sets A and B into one set
  - Goal implementation: O(log n)

Best darn U-F Data Structure

Implementing Kruskal's Algorithm

Using the union-find data structure

- Build set T of edges in the MST
- Maintain set for each connected component

Costs?

```
Sort edges weights so that c_1 ≤ c_2 ≤ … ≤ c_m
T = {}
foreach (u ∈ V) make a set containing singleton u

for i = 1 to m                     are u and v in different connected components?
   (u,v) = e_i
   if (u and v are in different sets)
      T = T ∪ {e_i}
      merge the sets containing u and v      merge two components
return T
```


Implementing Kruskal's Algorithm

Using *best* implementation of union-find

- Sorting: O(m log n) ← $m \le n^2 \Rightarrow \log m$ is O(log n)
- Union-find: O(m $\underbrace{\alpha\ (m, n)}_{\text{essentially a constant}}$)

⇒O(m log n)

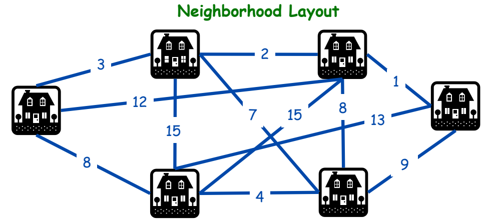
```
Sort edges weights so that c_1 ≤ c_2 ≤ … ≤ c_m
T = {}
foreach (u ∈ V) make a set containing singleton u

for i = 1 to m                     are u and v in different connected components?
   (u,v) = e_i
   if (u and v are in different sets)
      T = T ∪ {e_i}
      merge the sets containing u and v      merge two components
return T
```

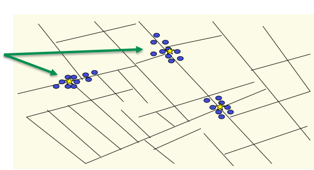

Limitations to Applying MST?

Motivating Example: Comcast laying cable

Neighborhood Layout



Intersections with polluted wells



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

CLUSTERING

Clustering

Given a set U of n objects labeled $p_1, \ldots, p_n$, classify into coherent groups

- Example objects: photos, documents, micro-organisms

Distance function. Numeric value specifying "closeness" of two objects

Clustering

Given a set U of n objects labeled $p_1, \ldots, p_n$, classify into coherent groups
- Example objects: photos, documents, micro-organisms

Distance function. Numeric value specifying "closeness" of two objects

Fundamental problem. Divide into clusters so that points in different clusters are far apart
- Routing in mobile ad hoc networks
- Identify patterns in gene expression
- Identifying patterns in web application use cases
  - Sets of URLs
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies

Clustering

k-clustering. Divide objects into k non-empty groups

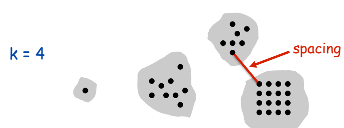
Distance function. Assume it satisfies several natural properties
- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

Clustering of Maximum Spacing

k-clustering. Divide objects into k non-empty groups

Spacing. Min distance between any pair of points in different clusters

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing



Ideas about Solving?

Greedy algorithm?

How relates to the minimum spanning tree?

Greedy Clustering Algorithm

Single-link k-clustering algorithm
- Form a graph on the vertex set U, corresponding to n clusters
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them
- Repeat n-k times until there are exactly k clusters

Key observation. Same as Kruskal's algorithm
- Except we stop when there are k connected components

Remark. Equivalent to finding an MST and deleting the k-1 most expensive edges

Problem Set 2

Solutions not online

See me to discuss your solution/write up or best solution

Common mistakes
- Not stating and/or discussing algorithm's runtime
- Not backing up claims
  - Ex: why has to have only one node in a layer
- Not using "algorithm terms", e.g., topological ordering, DAG, etc.
  - Not clear if following material, know how to apply solutions
- Not explaining intuition or model
  - Ex: what nodes and edges represent in last problem

Problem 3: Good Solution Sketch

Describe how modeling information:

- Let G be a directed graph with two nodes for each person
  - One representing person's birth, person's death
- A directed edge between nodes i and j means "i happened before j"
- How can use this model for data collected…

Data is consistent if G is a DAG

- Topological ordering is relative birth and death dates
- If cycle, inconsistent
  - Explain how can find a cycle

Our Plan

Wednesday: Finish up Chapter 4: Huffman Codes

Friday:

- Problem Set 3 due
- SSA – Extra credit opportunities
  - Added to homework grade

Monday: Divide and conquer algorithms (Chap 5)

Tue-Fri: Open-book midterm

- Turned into my mailbox in CS office by Friday
- I'll be at a conference Tuesday through Saturday
  - Available by email