Objectives

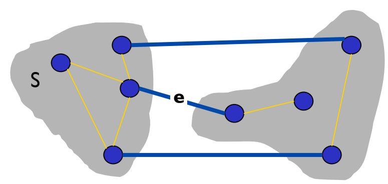
- Minimum Spanning Tree
- Union-Find Data Structure
- Clustering

Review

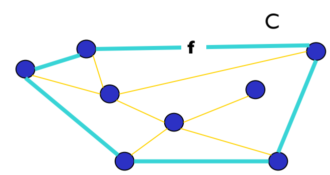
- What does the acronym MST stand for?
    - What is an MST?
- What are some algorithms to find the MST?

Summary of What We Proved

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



Cut Property: e is in MST



Cycle Property: f is **not** in MST

Prim's Algorithm

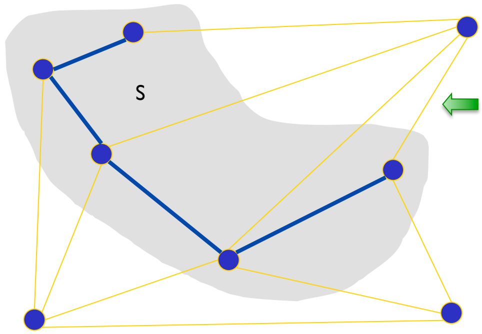
[Jarník 1930, Dijkstra 1957, Prim 1959]

- Start with some root node *s* 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.

How can we prove its correctness?

Prim's Algorithm: Proof of Correctness

- Initialize S to be any node
- Apply ***cut property*** to S
  - Add min cost edge (v, u) in *cutset* corresponding to S, and add one new explored node u to S



Ideas about implementation?

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*

Running Time?

```
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
```

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


Kruskal's Algorithm [1956]

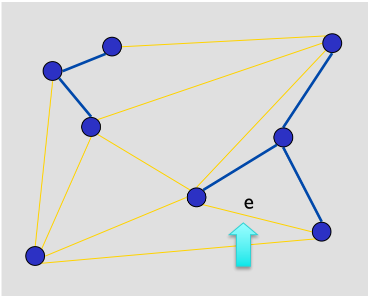
- Start with T = φ
- Consider edges in *ascending order of cost*
- Insert edge *e* in T unless doing so would create a cycle
  - Add edge as long as "compatible"

How can we prove algorithm's correctness?

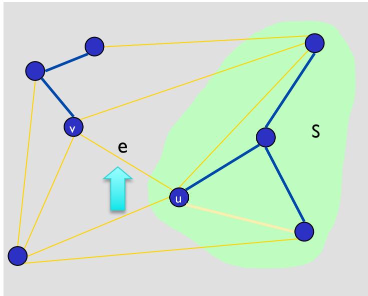
Kruskal's Algorithm: Proof of Correctness

What is tricky about implementing Kruskal's algorithm?

- Consider edges in ascending order of weight
- Case 1: If adding *e* to T creates a cycle, discard *e* according to ***cycle property*** (*e* must be max weight)
- 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?

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

UNION-FIND DATA STRUCTURE

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/API:
  - `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 Union-Find 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 edge weights so that c1 ≤ c2 ≤ ... ≤ cm
T = {}
foreach (u ∈ V) make a set containing singleton u

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

are u and v in different connected components?

merge two components

Implementing Kruskal's Algorithm

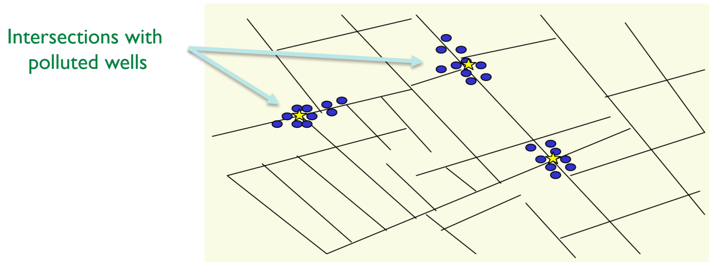
- Using best implementation of union-find
  - Sorting: O(m log n) ← $m \le n^2 \Rightarrow \log m$ is O(log n)
  - Union-find: $O(m \underbrace{\alpha(m, n)}_{\text{essentially a constant}})$
  - O(m log n)

```
Sort edges weights so that c1 ≤ c2 ≤ ... ≤ cm
T = {}
foreach (u ∈ V) make a set containing singleton u

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

are u and v in different connected components?

merge two components



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

CLUSTERING

Clustering

- Given a set *U* of *n* objects (or points) labeled $p_1, \ldots, p_n$, classify into coherent groups
  - Problem: Divide objects into clusters so that points in different clusters are far apart
    - Requires quantification of distance
- Applications
  - 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

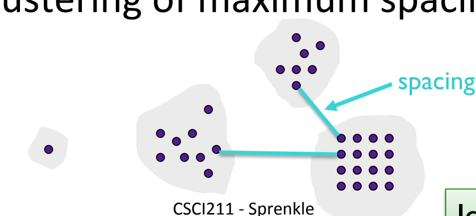
Clustering: Distance Function

- Numeric value specifying "closeness" of two objects
- Assume distance function 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)

Our Problem: k-Clustering of Maximum Spacing

- k-clustering. Divide objects into *k* non-empty groups
- Spacing. Min distance between any pair of points in different clusters
- k-clustering of maximum spacing.
  Given an integer *k*,
  find a *k*-clustering of maximum spacing

k = 4



Ideas about solving?

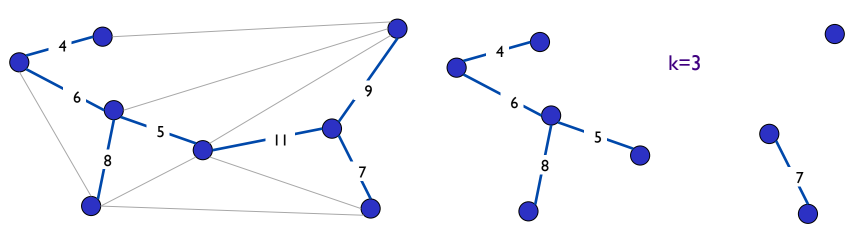
Greedy Clustering Algorithm

- 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

How is this related to the MST?

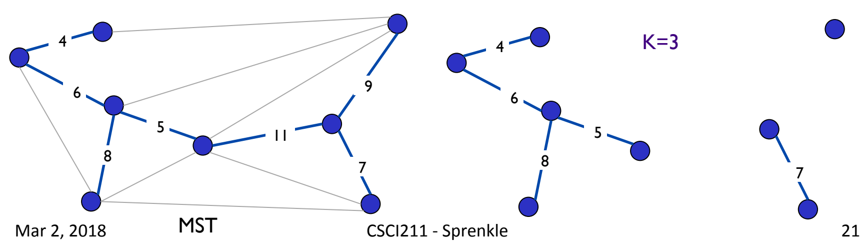
Greedy Clustering Algorithm

- Key observation: Same as Kruskal's algorithm
  - Except we stop when there are *k* connected components
- Remark. Equivalent to finding MST and deleting the *k-1* most expensive edges



Greedy Clustering Algorithm: Analysis

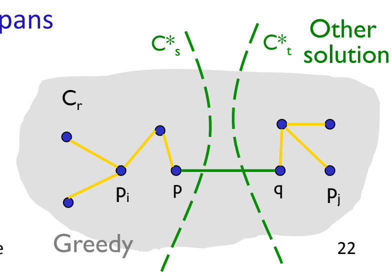
- Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *max spacing*.
- Pf Intuition:
  - ➢ What can we say about C's spacing?
    - Within clusters and between clusters
  - ➢ What if C isn't optimal?
    - What does that mean about C's clusters vs (optimal) C*'s clusters?



Greedy Clustering Algorithm: Analysis

- Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *maximum spacing*.
- Pf Sketch. Let C* denote some other clustering $C^*_1, \ldots, C^*_k$. C* and C must be different; otherwise we're done.
  - ➢ The spacing of *C* is length *d* of $(k-1)^{st}$ most expensive edge
  - ➢ Let p_i, p_j be in the same cluster in Greedy solution *C* (say C_r) but different clusters in other solution C*, say C^*_s and C^*_t
  - ➢ Some edge (*p*, *q*) on p_i-p_j path in C_r spans two different clusters in *C**

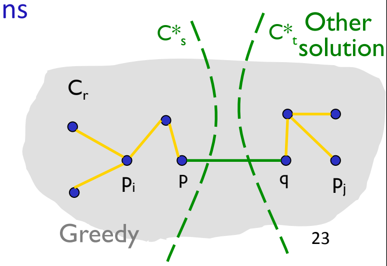
What do we know about *(p, q)*?



Greedy Clustering Algorithm: Analysis

- Theorem. Let C denote the clustering $C_1, \ldots, C_k$ formed by deleting the *k-1* most expensive edges of a MST. C is a *k*-clustering of *maximum spacing*.

- Pf. Let C* denote some other clustering $C^*_1, \ldots, C^*_k$. C* and C must be different; otherwise we're done.
  - The spacing of *C* is length *d* of (k-1)st most expensive edge
  - Let p_i, p_j be in the same cluster in *C* (say C_r) but different clusters in C*, say C^*_s and C^*_t
  - Some edge (*p*, *q*) on p_i-p_j path in C_r spans two different clusters in *C**
  - All edges on p_i-p_j path have length ≤ *d* since Kruskal chose them
  - Spacing of C* is at most ≤ d since *p* and *q* are in different clusters



Looking Ahead

- Wiki: 4.5-4.7