Objectives

- Wrap-up Dijkstra's Algorithm
- Minimum Spanning Tree

Announcements, Discussion

- Wiki readings
  - Low risk, high reward assignments
  - Helpful feedback
  - Process: follow book closely
  - Next Wed: Chap 3.6, 4, 4.1, 4.2, 4.4,
- Jeopardy! Challenge
  - Monday – Wednesday
  - 7:30 on CBS
  - Answer questions on Sakai forum for 5 pts towards your problem set grade

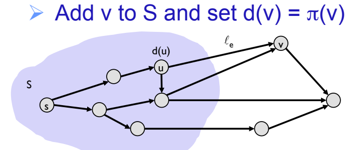
Review: Greedy Algorithms and Dijkstra's Algorithm

- What are greedy algorithms?
- What was the greedy algorithm to find the shortest path in a weighted directed graph?

Dijkstra's Algorithm: Analysis

1. Maintain a set of explored nodes S
   - Know the shortest path distance d(u) from *s* to *u*
2. Initialize S={s}, d(s)=0, ∀u≠s, d(u)=∞
3. Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e=(u,v):u\in S} d(u) + \ell_e,$
   - Add v to S and set d(v) = π(v)

shortest path to some u in explored part, followed by a single edge (u, v)



Running time?
Implementation?
Data structures?

Dijkstra's Algorithm: Analysis

1. Maintain a set of explored nodes S
   - Keep the shortest path distance d(u) from *s* to *u*
2. Initialize S={s}, d(s)=0
3. Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e=(u,v):u\in S} d(u) + \ell_e,$
   - Add v to S and set d(v) = π(v)

shortest path to some u in explored part, followed by a single edge (u, v)

PQ Operation	RT of Op	# in Dijkstra
Insert		
ExtractMin		
ChangeKey		
IsEmpty		
Total		

- How long does each operation take?
- How many of each operation?

Dijkstra's Algorithm: Implementation

- For each unexplored node, explicitly maintain $\pi(v) = \min_{e=(u,v):u\in S} d(u) + \ell_e .$
  - Next node to explore = node with minimum π(v).
  - When exploring v, for each incident edge e = (v, w), update $\pi(w) = \min\{\, \pi(w),\ \pi(v)+\ell_e \,\}.$
- Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by π(v)

PQ Operation	RT of Op	# in Dijkstra
Insert	log n	n
ExtractMin	log n	n
ChangeKey	log n	m
IsEmpty	1	n
Total		m log n

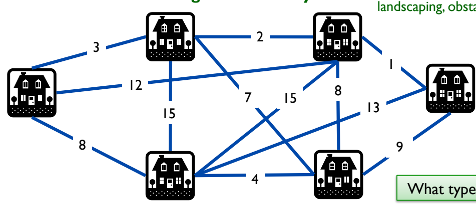
O(m log n)

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

Neighborhood Layout

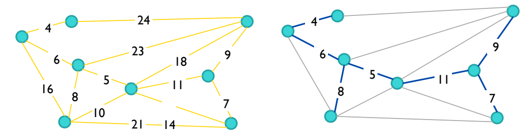
Cost of laying cable btw houses depends on amount of cable, landscaping, obstacles, etc.



What type of graph?

Minimum Spanning Tree (MST)

- **Spanning tree**: spans all nodes in graph
- 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*



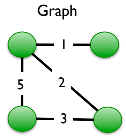
G = (V, E)

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

Examples

Identify spanning trees and which is the ***minimal*** spanning tree.

Graph



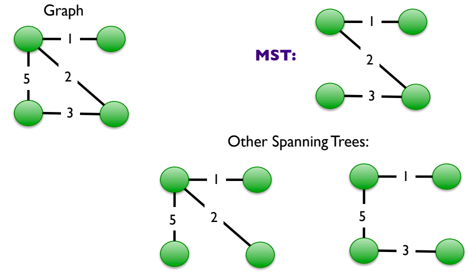
Examples

Identify spanning trees and which is the ***minimal*** spanning tree.

Graph

MST:

Other Spanning Trees:



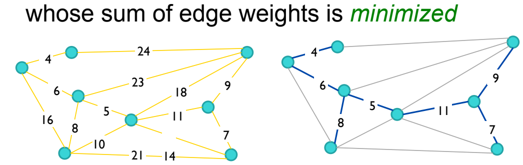
MST Applications

- Network design
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems
  - traveling salesperson problem, Steiner tree
- Indirect applications
  - max bottleneck paths
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
- **Cluster analysis**

`http://www.ics.uci.edu/~eppstein/gina/mst.html`

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*



G = (V, E)

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

Why *must* the solution be a tree?

Minimum Spanning Tree

- Assume have a minimal solution that is not a tree, i.e., it has a cycle
- What could we do?
  - ➢ What do we know about the edges?
  - ➢ How does that change the cost of the solution?

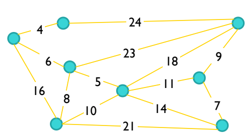
Minimal Spanning Tree

- 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 total cost
  - ➢ Contradiction to our minimal solution

Ideas for Solutions?

- Cayley's Theorem. There are n^{n-2} spanning trees of K_n

  ↑ can't solve by brute force
- Towards a solution…
  - ➢ Where to start?



G = (V, E)

Greedy Algorithms

All three algorithms produce a MST

- Prim's algorithm. 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*.
  - ➢ Similar to Dijkstra's (but simpler)
- 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*.

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

What Do These Algorithms Have in Common?

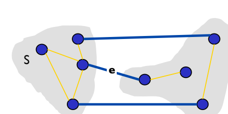
- When is it safe to include an edge in the minimum spanning tree?

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

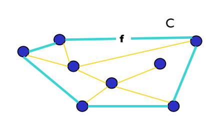
  Cycle Property

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 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

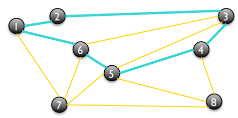


Cycle Property: f is **not** in MST

Let's try to prove these …

Cycles and Cuts

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

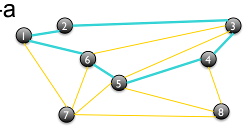


Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1

Feb 11, 2011 CSCI211 - Sprenkle 19

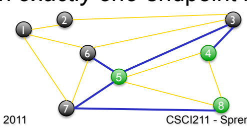
Cycles and Cuts

- Cycle. Set of edges in the 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*.

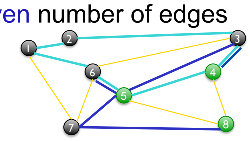


Cut S = { 4, 5, 8 }
Cutset D = 5-6, 5-7, 3-4, 3-5, 7-8

Feb 11, 2011 CSCI211 - Sprenkle 20

Cycle-Cut Intersection

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



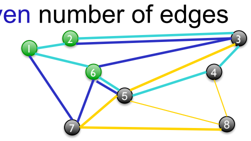
Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cut S = { 4, 5, 8 }
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

What are the possibilities for the cycle?

Feb 11, 2011 CSCI211 - Sprenkle 21

Cycle-Cut Intersection

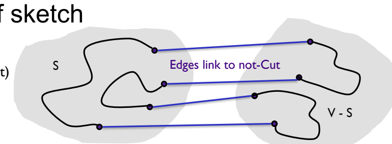
- Claim. A *cycle* and a *cutset* intersect in an even number of edges



Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cut S = { 1, 2, 6 }
Cutset D = 1-7, 2-3, 6-3, 6-5, 6-7
Intersection = 2-3, 6-5

- Proof sketch

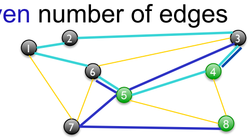
(Cut)



Feb 11, 2011 CSCI211 - Sprenkle 22

Cycle-Cut Intersection

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

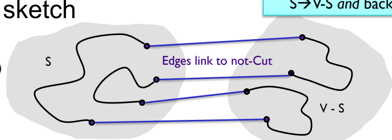


Cycle C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
Cut S = { 4, 5, 8 }
Cutset D = 3-4, 3-5, 5-6, 5-7, 7-8
Intersection = 3-4, 5-6

1. Cycle all in S
2. Cycle not in S
3. Cycle has to go from S→V-S *and* back

- Proof sketch

(Cut)



Feb 11, 2011 CSCI211 - Sprenkle 23