Objectives

Greedy Algorithms

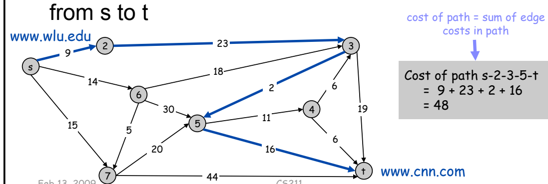
- Shortest path
- Minimum spanning tree

Shortest Path Problem

Given

- Directed graph G = (V, E)
- Source s, destination t
- Length ℓ_e = length of edge e (non-negative)

Shortest path problem: find shortest directed path from s to t



Dijkstra's Algorithm

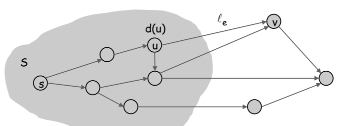
Maintain a set of explored nodes S

- Know the shortest path distance d(u) from *s* to *u*

Initialize S={s}, d(s)=0

Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e=(u,v):u\in S} d(u) + \ell_e$, ← shortest path to some u in explored part, followed by a single edge (u, v)

- add v to S and set d(v) = π(v)



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	Dijkstra	Binary heap
Insert	n	log n
ExtractMin	n	log n
ChangeKey	m	log n
IsEmpty	n	1
Total		m log n

How Greedy?

We always form shortest new *s-v* path from a path in S followed by a *single* edge

Proof of optimality: *Stays ahead* of all other solutions

- Each time selects a path to a node *v*, that path is shorter than every other possible path to *v*

Dijkstra's Algorithm: Proof of Correctness

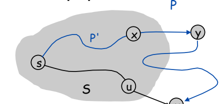
Invariant. For each node u ∈ S, d(u) is the length of the shortest s-u path

Pf. (by induction on |S|)

Base case: For |S| = 1, S={s}; d(s) = 0

Inductive hypothesis: Assume true for |S| = k, k ≥ 1

- Grow |S| to k+1
- Adding next node *v* by *u*→ *v*
- What do we know about *s*→ *u*?
- What can we say about other *s*→ *v* paths?
- Why didn't we pick *y* as the next node?



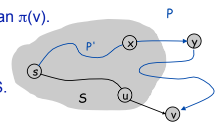
Dijkstra's Algorithm: Proof of Correctness

Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest s-u path

Pf. (by induction on $|S|$)

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let v be next node added to S, and let u-v be the chosen edge
- The shortest s-u path plus (u, v) is an s-v path of length $\pi(v)$
- Consider any s-v path P. It's no shorter than $\pi(v)$.
- Let x-y be the first edge in P that leaves S, and let P' be the subpath to x.
- P is already too long as soon as it leaves S.

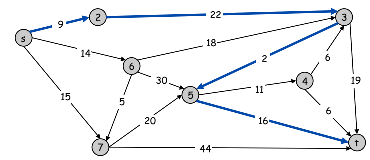


$\ell(P) \geq \ell(P') + \ell(x,y) = d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)$

nonnegative weights; inductive hypothesis; defn of $\pi(y)$; Dijkstra chose v instead of y

Discussion: Dijstra's Algorithm

Why does the algorithm break down if we allow negative weights/costs on edges?



MINIMUM SPANNING TREE

Laying Cable

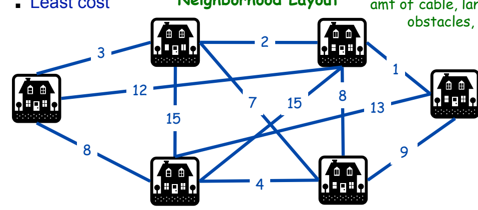
Comcast knows how to make money and how to save money

They want to lay cable in a neighborhood

- Reach all houses
- Least cost

Neighborhood Layout

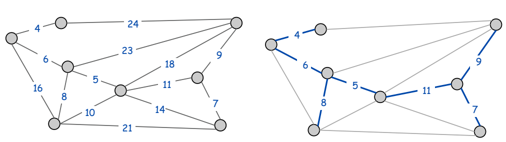
Cost of laying cable between houses depends on amt of cable, landscaping, obstacles, etc.



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

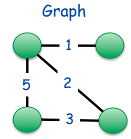


$G = (V, E)$ $\qquad$ $T, \ \Sigma_{e \in T} c_e = 50$

Examples

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

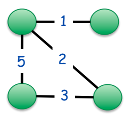
Graph



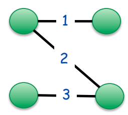
Examples

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

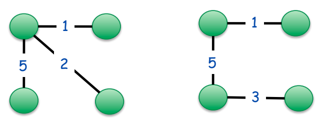
Graph



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

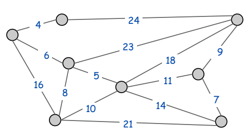
Indirect applications
- max bottleneck paths
- image registration with Renyi entropy
- learning salient features for real-time face verification
- reducing data storage in sequencing amino acids in a protein
- model locality of particle interactions in turbulent fluid flows

Cluster analysis

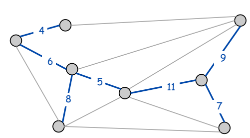
Minimum Spanning Tree

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*.

Why must the solution be a tree?



$G = (V, E)$



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

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 cost
- Contradiction to our minimal solution

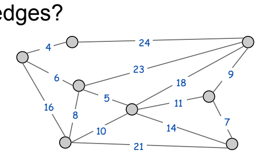
Ideas for Solutions?

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

can't solve by brute force

Where to start?

Orders to add/remove edges?



$G = (V, E)$

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

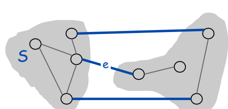
Greedy Algorithms

Simplifying assumption: All edge costs c_e are distinct

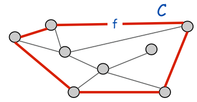
➡ MST is unique

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.



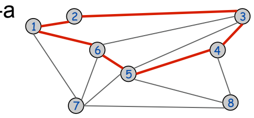
e is in the MST



f is not in the MST

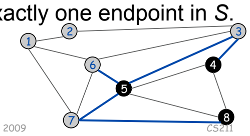
Cycles and Cuts

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

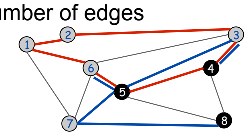
Cutset. A *cut* is a subset of nodes S. The corresponding *cutset* D is the subset of edges with exactly one endpoint in S.



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

Cycle-Cut Intersection

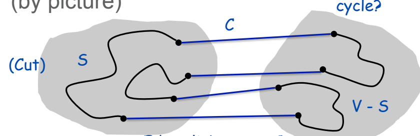
Claim. A cycle and a cutset intersect in an even number of edges



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

- What are the possibilities for the cycle?

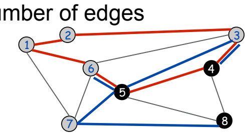
Pf. (by picture)



Edges link to not-Cut

Cycle-Cut Intersection

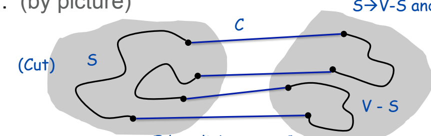
Claim. A cycle and a cutset intersect in an even number of edges



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 ?!?