Objectives

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

Review: Greedy Algorithms and Dijkstra's Algorithm

- What are greedy algorithms?
- How are some strategies to prove that greedy algorithms are optimal?
- What was the greedy algorithm to find the shortest path in a weighted directed graph?

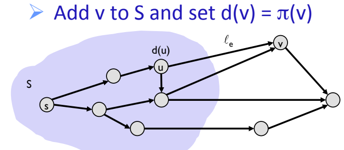
Review

- When we have a problem about finding the shortest path, what algorithm should we think about applying?
- BFS or Dijkstra's
  - Difference: Dijkstra's when edges have positive (and different) weights

Dijkstra's Algorithm

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, ∀ 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)

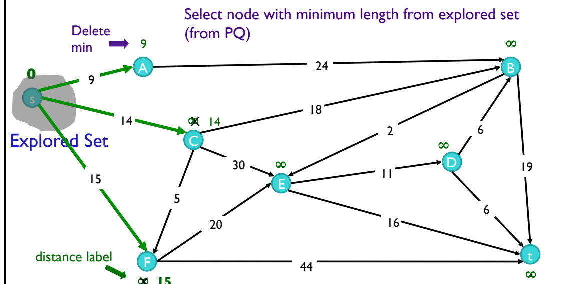
shortest path to some u in explored part followed by a single edge (u, v)



Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { A, C, F, B, D, E, t }

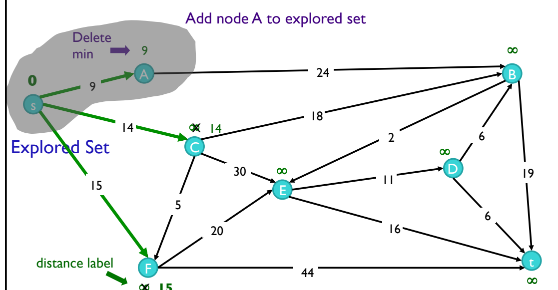
Select node with minimum length from explored set (from PQ)



Dijkstra's Shortest Path Algorithm

S = { s, A }
PQ = {C, F, B, D, E, t }

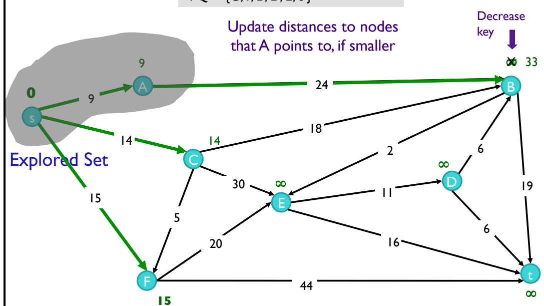
Add node A to explored set



Dijkstra’s Shortest Path Algorithm

S = { s, A }
PQ = {C, F, B, D, E, t }

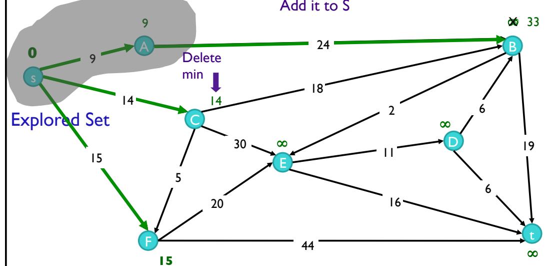
Update distances to nodes that A points to, if smaller



Dijkstra’s Shortest Path Algorithm

S = { s, A }
PQ = {C, F, B, D, E, t }

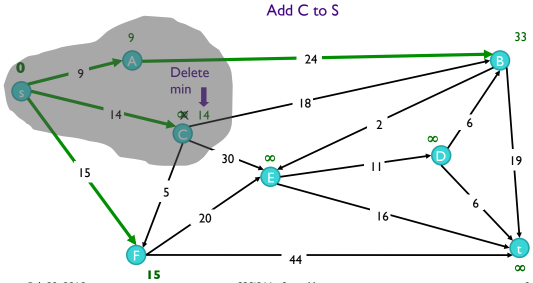
Delete the minimum node from PQ, Add it to S



Dijkstra’s Shortest Path Algorithm

S = { s, A, C }
PQ = { F, B, D, E, t }

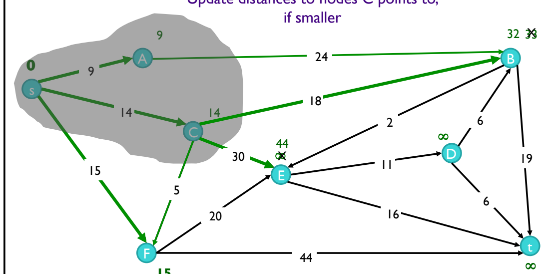
Add C to S



Dijkstra’s Shortest Path Algorithm

S = { s, A, C }
PQ = { F, B, **E**, D, t }

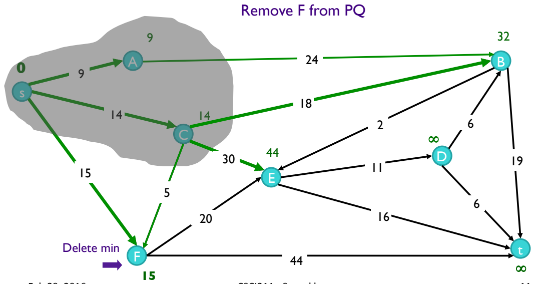
Update distances to nodes C points to, if smaller



Dijkstra’s Shortest Path Algorithm

S = { s, A, C }
PQ = { F, B, E, D, t }

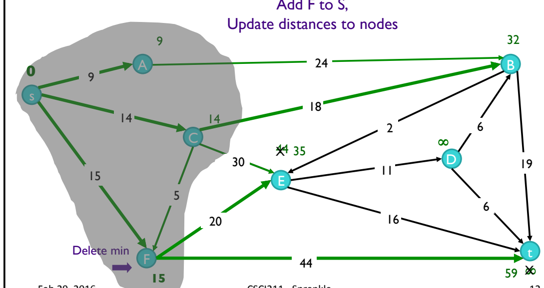
Remove F from PQ



Dijkstra’s Shortest Path Algorithm

S = { s, A, C, F }
PQ = { B, E, **t**, D }

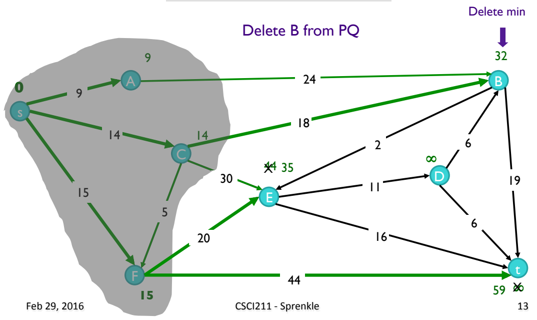
Add F to S, Update distances to nodes



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F }
PQ = { B, E, t, D }

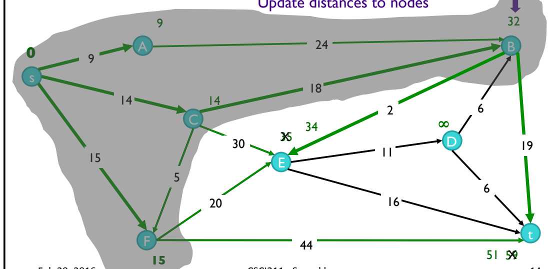
Delete B from PQ



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F, B }
PQ = { E, t, D }

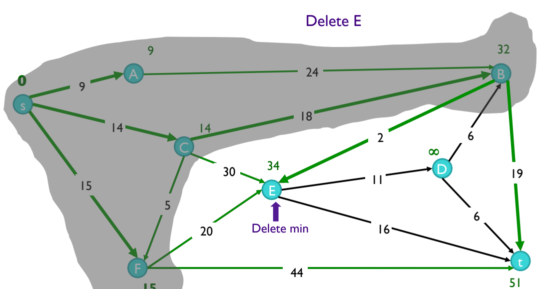
Add B to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F, B }
PQ = { E, t, D }

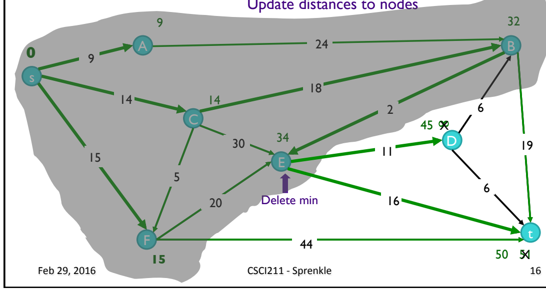
Delete E



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F, B, E }
PQ = { **D**, t }

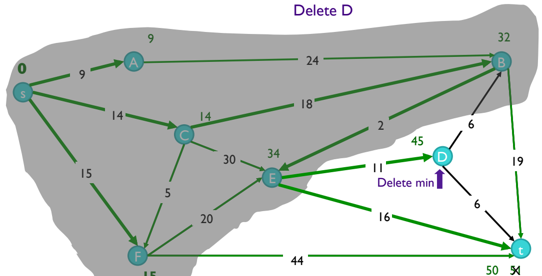
Add E to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F, B, E }
PQ = { D, t }

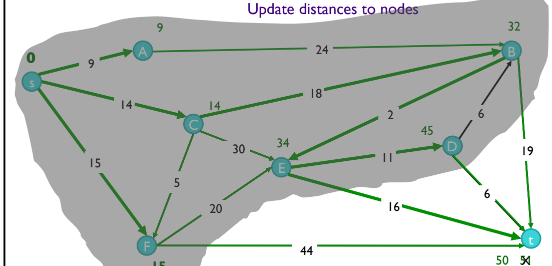
Delete D



Dijkstra's Shortest Path Algorithm

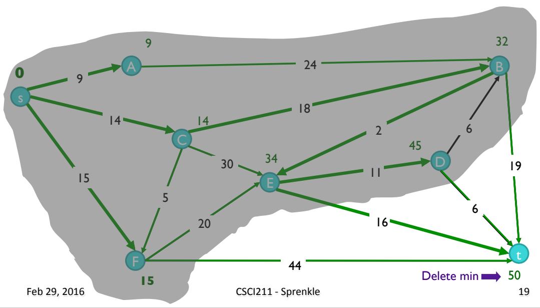
S = { s, A, C, F, B, E, D }
PQ = { t }

Add D to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

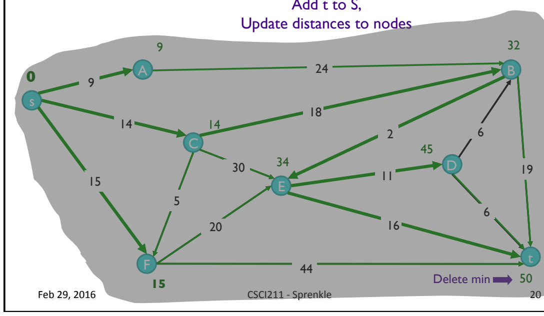
S = { s, A, C, F, B, E, D }
PQ = { t }



Dijkstra's Shortest Path Algorithm

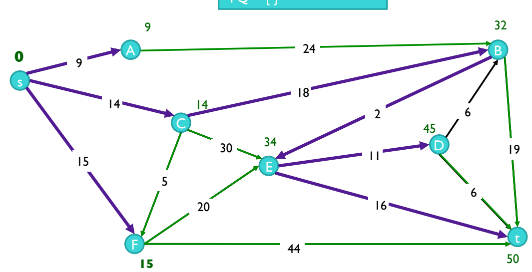
S = { s, A, C, F, B, E, D, t }
PQ = { }

Add t to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

S = { s, A, C, F, B, E, D, t }
PQ = { }

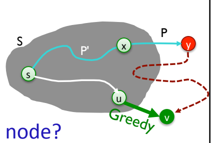


Dijkstra's Algorithm: Proof of Correctness

- Invariant. For each node u ∈ S, d(u) is the length of the shortest s-u path
- Pf. (by induction on |S|)
- Base case: |S|=1 …
- Inductive hypothesis?
- Next step?

Dijkstra's Algorithm: Proof of Correctness

- Prove: 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
  - Greedy: Add node *v* by *u*→*v*
  - What do we know about *s*→*u*?
  - Why didn't we pick *y* as the next node?
  - What can we say about other *s*→*v* paths?



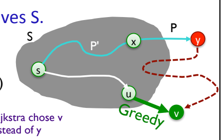
Dijkstra's Algorithm: Proof of Correctness

- Prove: For each node u ∈ S, d(u) is the length of the shortest s-u path
- Pf. (by induction on |S|)
- Inductive hypothesis: Assume true for |S| = k, k ≥ 1
  - Let *v* be the next node added to *S* by Greedy, and let *u*→*v* be the chosen edge
  - The shortest *s*→*u* path plus *u*→*v* is an *s*→*v* path of length π(v)
  - Consider any *s*→*v* path P. It's no shorter than π(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.

In terms of inequalities:

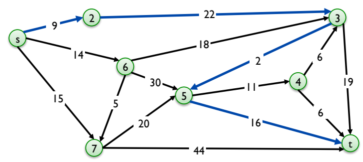
$$\ell(P) \geq \ell(P') + \ell(x,y) = d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)$$

(nonnegative weights; inductive hypothesis; defn of π(y); Dijkstra chose v instead of y)



Discussion: Dijstra's Algorithm

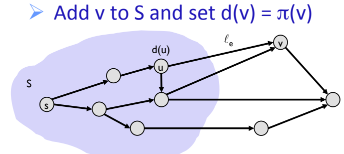
- Why does the algorithm break down if we allow negative weights/costs on edges?



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, ∀ 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)

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

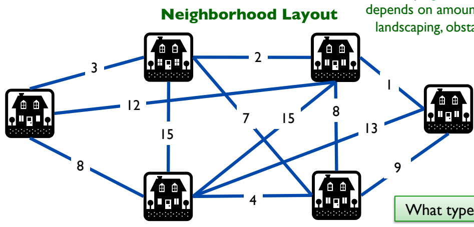
O(m log n)

Laying Cable

- Comcast wants to lay cable in a neighborhood
  - Reach all houses
  - Least cost

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

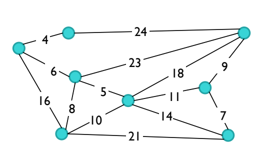
Neighborhood Layout



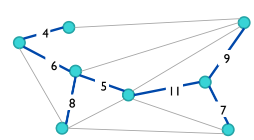
What type of graph?

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 ⊆ E such that T is a ***spanning tree*** whose **sum of edge weights** is ***minimized***



G = (V, E)

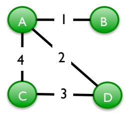


T, $\Sigma_{e \in T}\, c_e = 50$

Examples

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

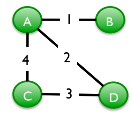
Graph



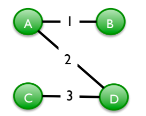
Examples

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

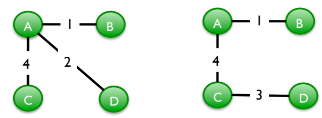
Graph



MST:



Other Spanning Trees:



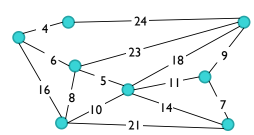
MST Applications

- 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

Ideas for Solutions?

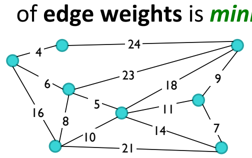
- Cayley's Theorem. There are n^{n-2} spanning trees
  - can't solve by brute force
- Towards a solution…
  - Where to start?



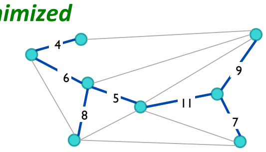
G = (V, E)

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?

Looking ahead

- Wiki – "front matter" of 4, 4.1, 4.2, 4.4
- Problem Set 5 due Friday