Objectives

- Weighted, directed graph shortest path

Review

- What are the three ways to prove the optimality of a greedy algorithm?
- Problem: minimizing maximum lateness
  - What was the problem?
  - What was our approach to solving it?
  - How did we prove the approach's optimality?
  - What is the algorithm's runtime?

Review: Greedy Analysis Strategies

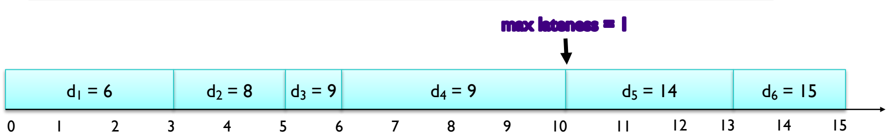
- Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
- Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Analyzing Running Time

- Earliest deadline first.

```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```

O(n logn)



What is the runtime of this algorithm?

Minimizing Lateness: Analysis of Greedy Algorithm

- Theorem. Greedy schedule S is optimal
- Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens
  - Can assume S* has no idle time
  - If S* has no inversions, then S = S*
  - If S* has an inversion, let i-j be an adjacent inversion
    - Swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions (as we proved separately)
    - This contradicts definition of S* ▪

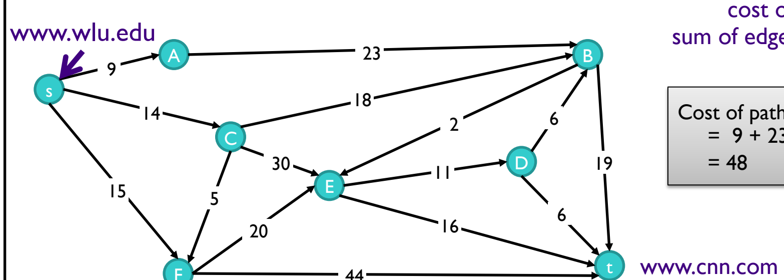
Greedy Exchange Proofs

1. Label your algorithm's solution and a general solution.
   - Example: let A = {a_1, a_2, ..., a_k} be the solution generated by your algorithm, and let O = {o_1, o_2, ..., o_m} be an optimal feasible solution.
2. Compare greedy with other solution.
   - Assume that the optimal solution is not the same as your greedy solution (since otherwise, you are done).
   - Typically, can isolate a simple example of this difference, such as:
     ① There is an element e ∈ O that ∉ A and an element f ∈ A that ∉ O
     ② 2 consecutive elements in O are in a different order than in A
       - i.e., there is an *inversion*
3. Exchange.
   - Swap the elements in question in O (either ① swap one element out and another in or ② swap the order of the elements) and argue that solution is no worse than before.
   - Argue that if you continue swapping, you eliminate all differences between O and A in a *finite* # of steps *without worsening the solution's quality*.
   - Thus, the greedy solution produced is just as good as any optimal solution, and hence is optimal itself.

SHORTEST PATH

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

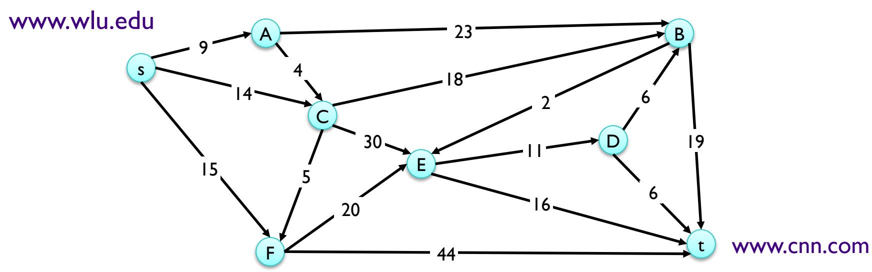


cost of path =
sum of edge costs in path

Cost of path s-A-B-E-t
= 9 + 23 + 2 + 16
= 48

Shortest Path Problem

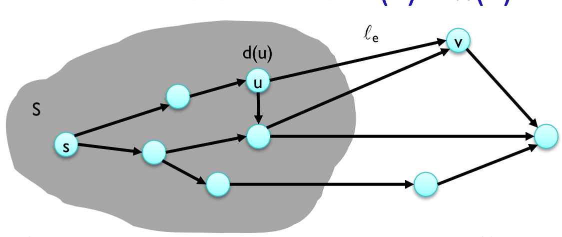
- Shortest path problem: find shortest directed path from s to t
- Brainstorming on solution …



Dijkstra's Algorithm

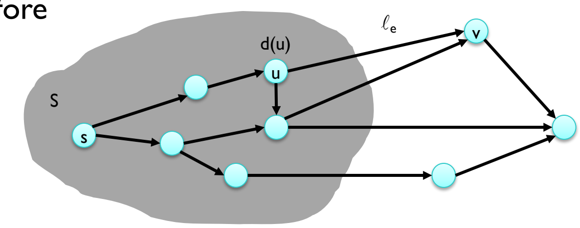
1. Maintain a set of **explored nodes** S
   - Keep the shortest path distance d(u) from *s* to *u*
2. Initialize S={s}, d(s)=0, ∀u≠s, d(u)=∞
3. Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$
   - Add v to S and set d(v) = π(v)

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

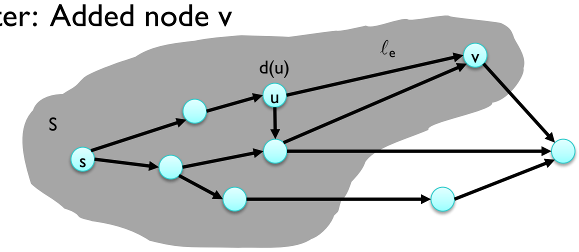


Dijkstra's Algorithm

Before



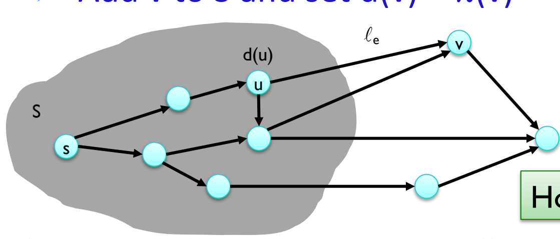
After: Added node v



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)



How is algorithm Greedy?

How is Algorithm Greedy?

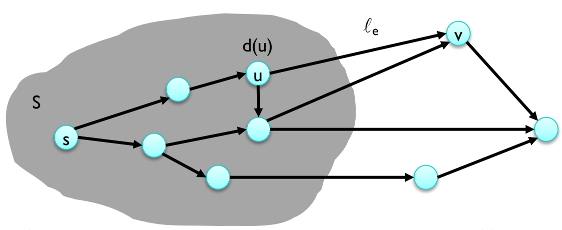
- We always form **shortest new *s->v* path** from a path in S followed by a *single* edge
- Proof of optimality: *Stays ahead* of all other solutions
  - Each time selects a path to a node *v*, that path is shorter than every other possible path to *v*

More on this later…

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)

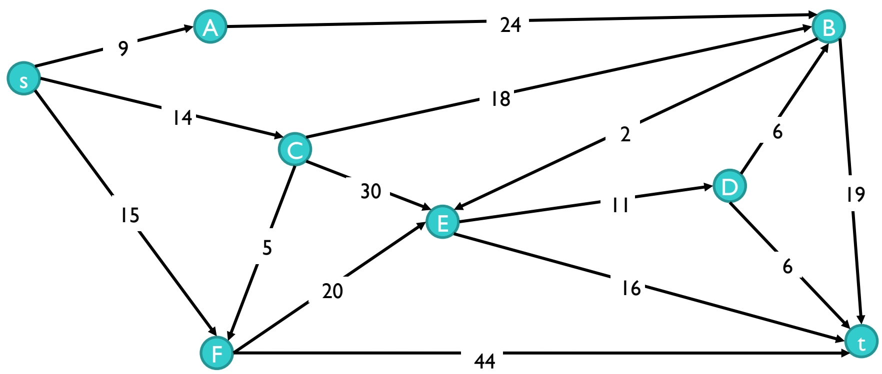
shortest path to (some u in explored part followed by a single edge (u, v))



Implementation Ideas
- What to represent?
- How to represent?

Dijkstra's Shortest Path Algorithm

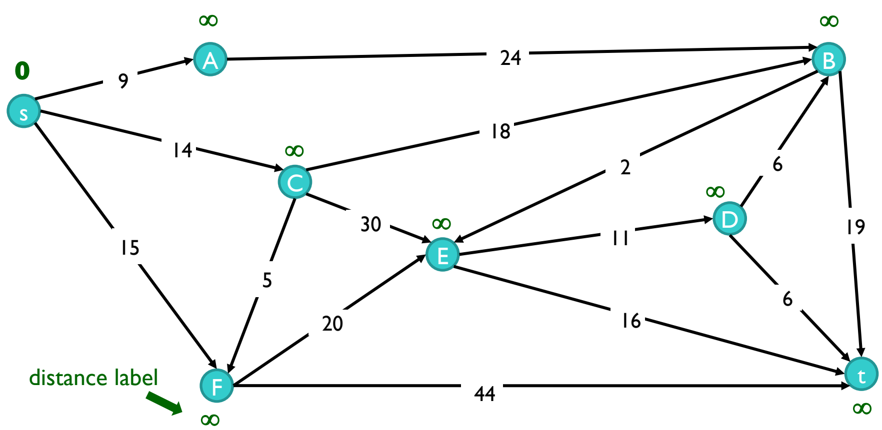
- Find shortest path from s to t



Dijkstra's Shortest Path Algorithm

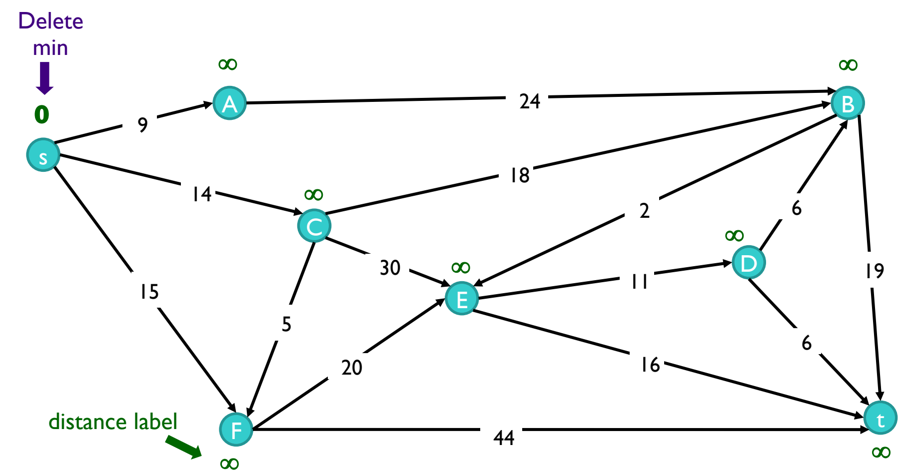
S = { }
PQ = { s, A, B, C, D, E, F, t }

Initialize distances to all nodes to infinity



Dijkstra's Shortest Path Algorithm

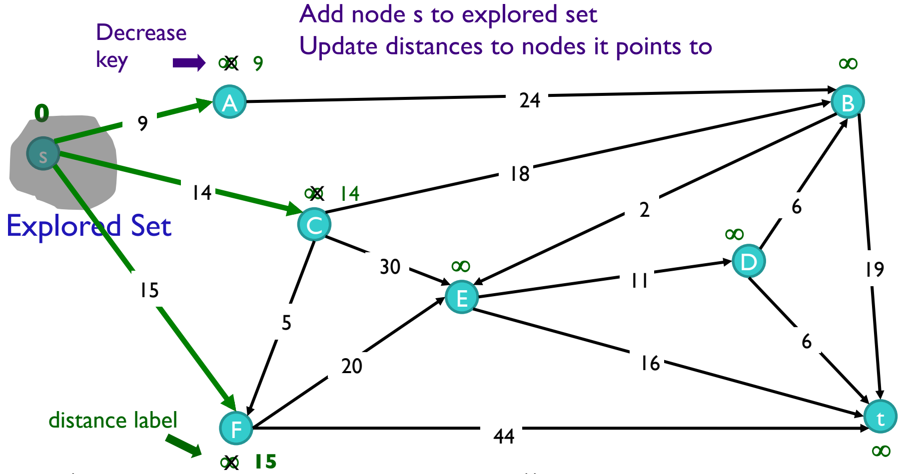
S = { }
PQ = { s, A, B, C, D, E, F, t }



Dijkstra's Shortest Path Algorithm

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

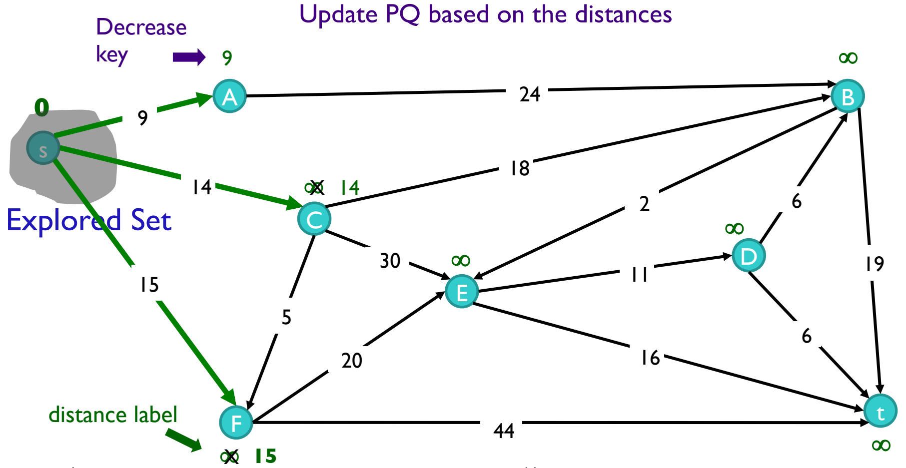
Add node s to explored set
Update distances to nodes it points to



Dijkstra's Shortest Path Algorithm

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

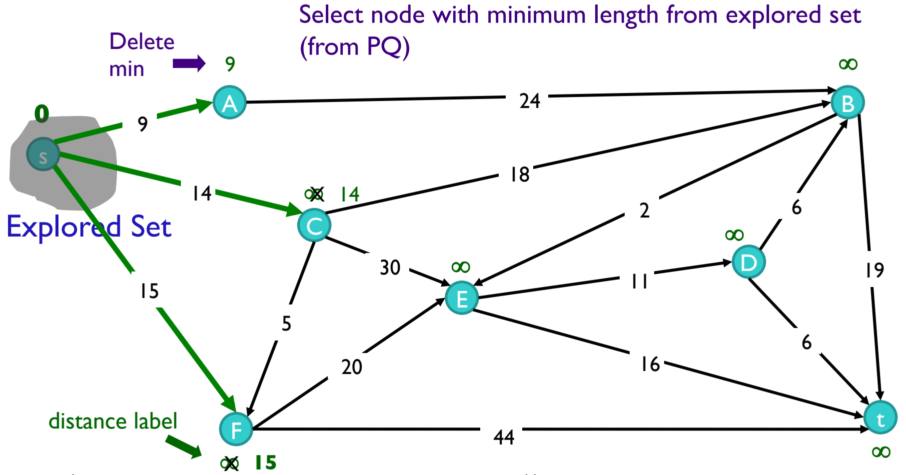
Update PQ based on the distances



Dijkstra's Shortest Path Algorithm

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

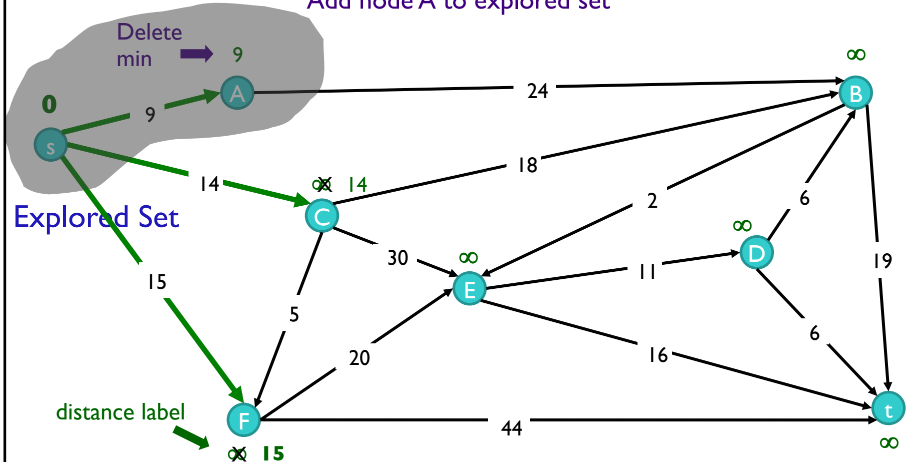
Select node with minimum length from explored set (from PQ)



Dijkstra's Shortest Path Algorithm

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

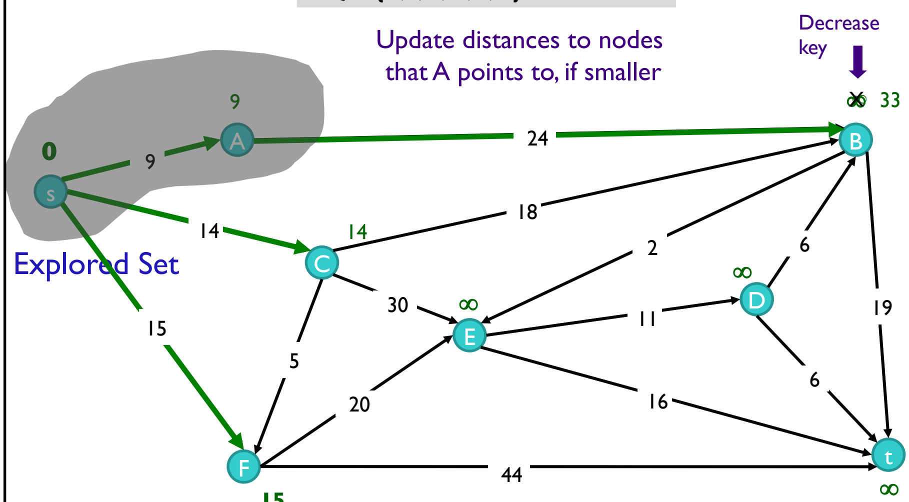
Add node A to explored set



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



Looking Ahead

- Wiki due Monday, after break
  - "Front matter" of Chapter 4
  - 4.1, 4.2, 4.4
- Problem Set 5 due Friday, after break

Feb 15, 2019 CSCI211 - Sprenkle 23