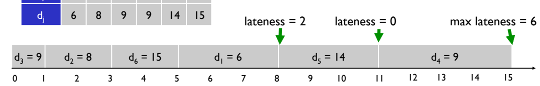
Objectives

- Problem: Minimizing Lateness
  - Greedy exchange
- Problem: Shortest Path

Review: Scheduling to Minimizing Lateness

- Single resource processes one job at a time
- Job j requires t_j units of processing time and is due at time d_j (its deadline)
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$
- **Goal**: schedule all jobs to *minimize* **maximum lateness** $L = \max \ell_j$

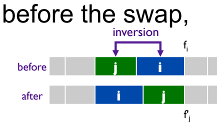
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Note: **not** a sum total

Minimizing Lateness: Inversions

- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does *not increase the max lateness*.
- Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards
  - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell_j \leq \ell_i$, $\ell'_i \leq \ell_i$
  - If job j is late:



$d_i < d_j$

$$\begin{aligned} \ell'_j &= f'_j - d_j && \text{(definition)} \\ &= f_i - d_j && (j \text{ finishes at time } f_i) \\ &\leq f_i - d_i && (i < j) \\ &\leq \ell_i && \text{(definition)} \end{aligned}$$

Shows that the lateness of jobs i and j do not increase from the original order

Minimizing Lateness: Analysis of Greedy Algorithm

- Theorem. Greedy schedule S is optimal
- Pf idea. Convert Opt to Greedy
  - Does opt schedule have idle time?
  - What if opt schedule has no inversions?
  - What if opt schedule has inversions?

Minimizing Lateness: Analysis of Greedy Algorithm

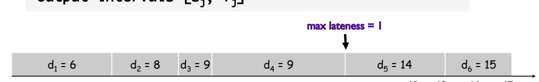
- Theorem. Greedy schedule S is optimal
- Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens
  - Can assume S* has no idle time
  - If S* has no inversions, then S = S*
  - If S* has an inversion, let i-j be an adjacent inversion
    - Swapping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

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?

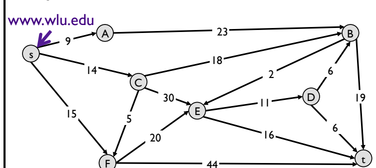
Greedy Exchange Proofs

1. Label your algorithm's solution and a general solution.
   - Example: let A = {$a_1, a_2, \ldots, a_k$} be the solution generated by your algorithm, and let O = {$o_1, o_2, \ldots, o_m$} be an arbitrary (or optimal) feasible solution.
2. Compare greedy with other solution.
   - Assume that your arbitrary/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 $e \in O$ that $\notin A$ and an element $f \in A$ that $\notin O$
     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 PROBLEMS

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

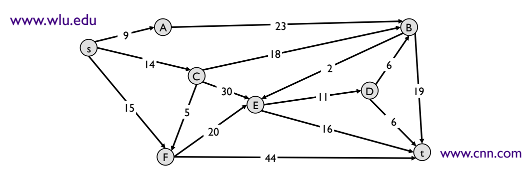


cost of path = sum of edge costs in path

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

Shortest Path Problem

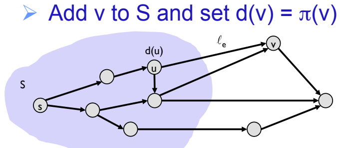
- Shortest path problem: find shortest directed path from s to t
- Towards algorithm ideas:
  - What is shortest path from s→A? s→C?
  - What is the shortest path from s→B? E? D?



Dijkstra's Algorithm

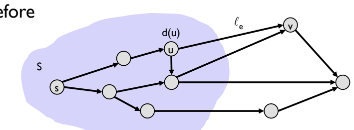
1. Maintain a set of **explored nodes** S
   - Keep the shortest path distance d(u) from *s* to *u*
2. Initialize S={s}, d(s)=0, ∀u≠s, d(u)=∞
3. Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e=(u,v)\,:\,u \in S} d(u) + \ell_e ,$
   - Add v to S and set d(v) = π(v)

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

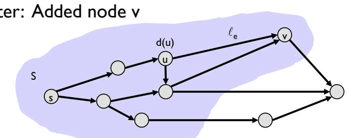


Dijkstra's Algorithm

Before

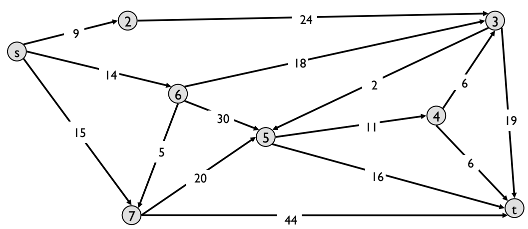


After: Added node v



Dijkstra's Shortest Path Algorithm

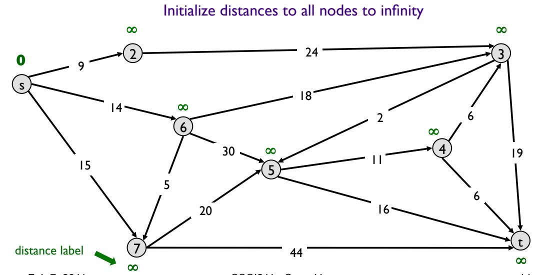
- Find shortest path from s to t.



Dijkstra's Shortest Path Algorithm

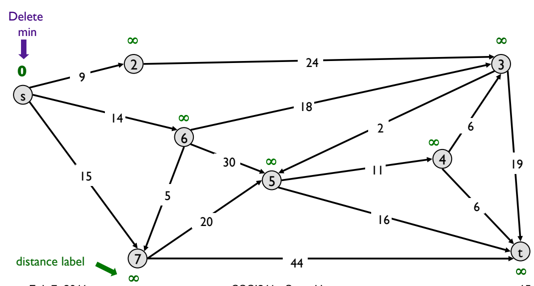
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }

Initialize distances to all nodes to infinity



Dijkstra's Shortest Path Algorithm

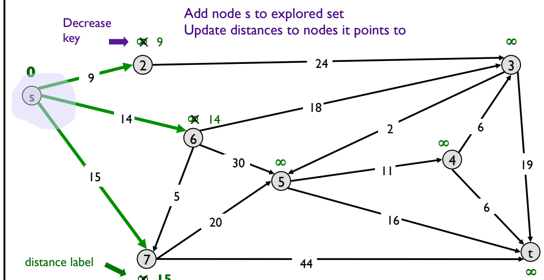
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }

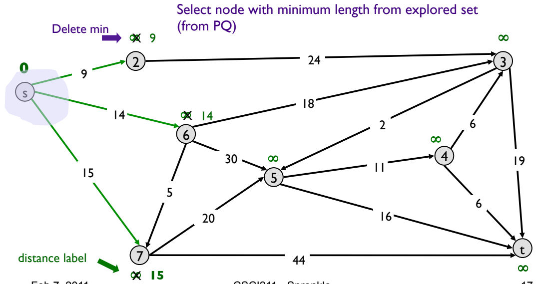
Add node s to explored set
Update distances to nodes it points to



Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 6, 7, 3, 4, 5, t }

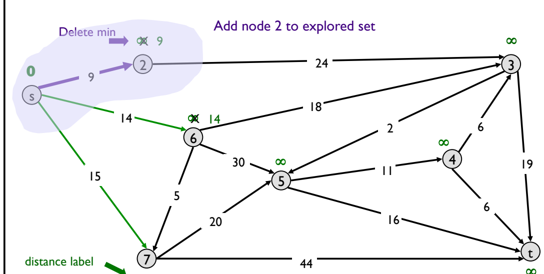
Select node with minimum length from explored set (from PQ)



Dijkstra's Shortest Path Algorithm

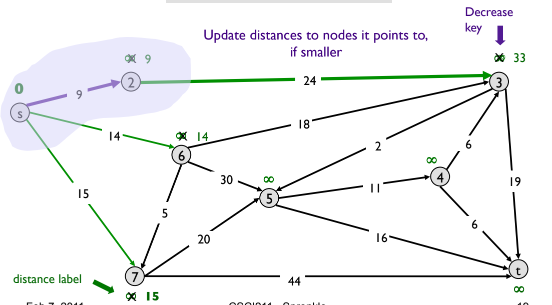
S = { s, 2 }
PQ = { 6, 7, 3, 4, 5, t }

Add node 2 to explored set



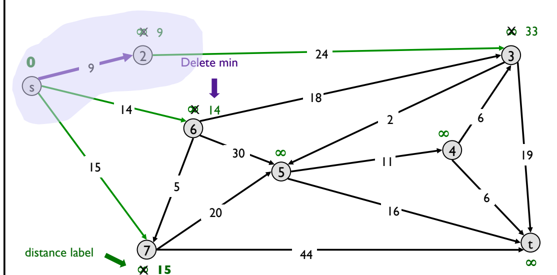
Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 6, 7, 3, 4, 5, t }



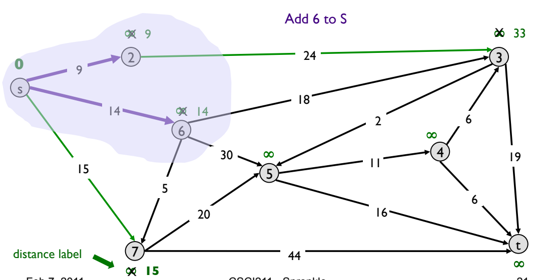
Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 6, 7, 3, 4, 5, t }



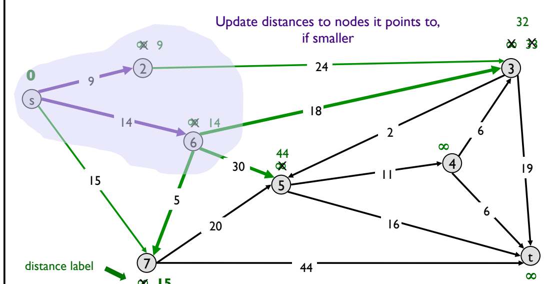
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 7, 3, 4, 5, t }



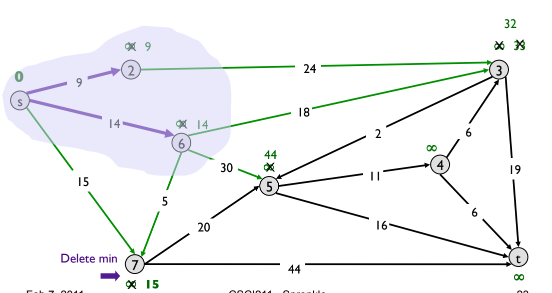
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 7, 3, 5, 4, t }



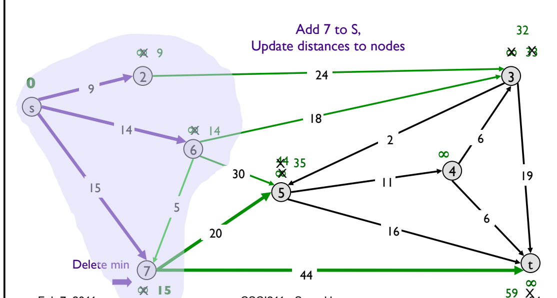
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 7, 3, 5, 4, t }



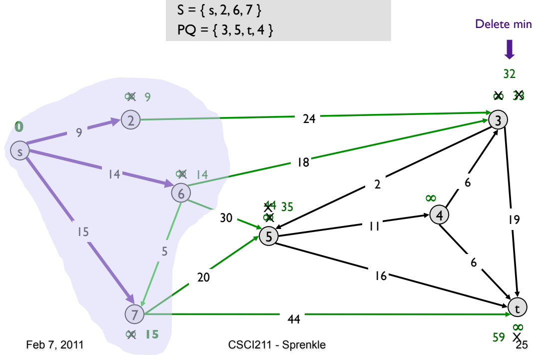
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }
PQ = { 3, 5, t, 4 }



Dijkstra's Shortest Path Algorithm

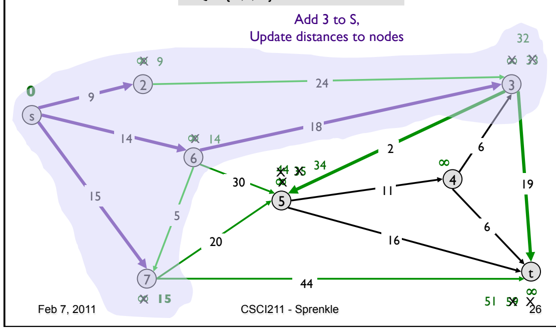
S = { s, 2, 6, 7 }
PQ = { 3, 5, t, 4 }



Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

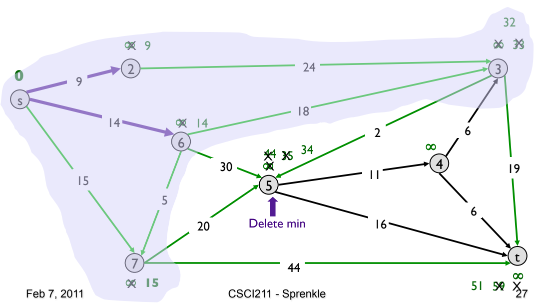
S = { s, 2, 6, 7, 3 }
PQ = { 5, t, 4 }



Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

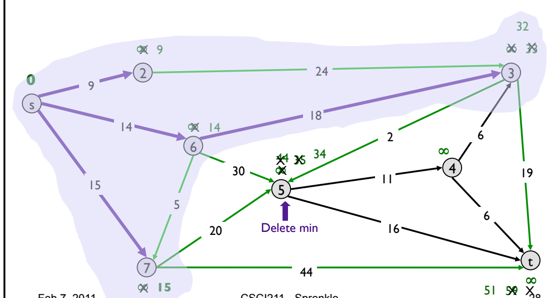
S = { s, 2, 6, 7, 3 }
PQ = { 5, t, 4 }



Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

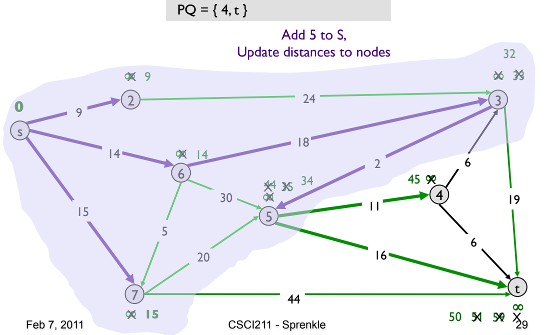
S = { s, 2, 6, 7, 3 }
PQ = { 5, t, 4 }



Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

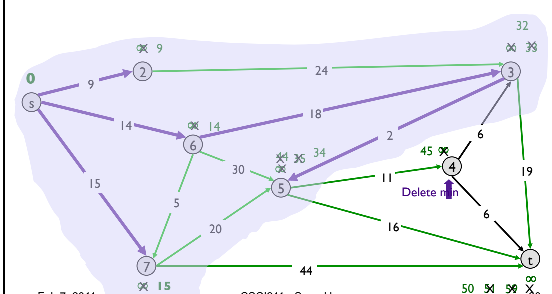
S = { s, 2, 6, 7, 3, 5 }
PQ = { 4, t }



Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7, 3, 5 }
PQ = { 4, t }

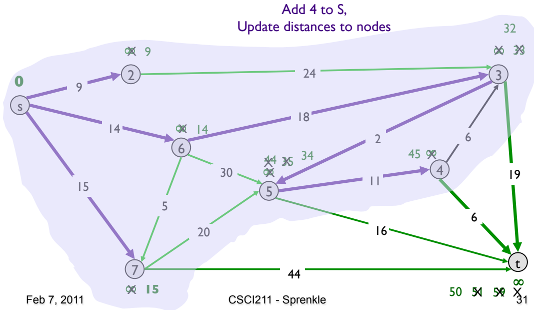


Feb 7, 2011 CSCI211 - Sprenkle

Dijkstra's Shortest Path Algorithm

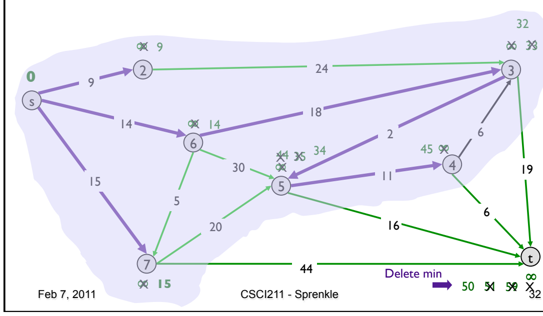
S = { s, 2, 6, 7, 3, 5, 4 }
PQ = { t }

Add 4 to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

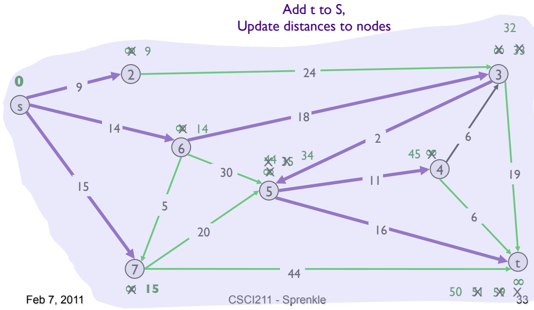
S = { s, 2, 6, 7, 3, 5, 4 }
PQ = { t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7, 3, 5, 4, t }
PQ = { }

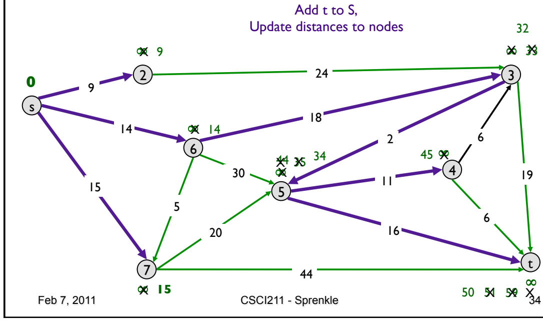
Add t to S,
Update distances to nodes



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7, 3, 5, 4, t }
PQ = { }

Add t to S,
Update distances to nodes



How Greedy?

How Greedy?

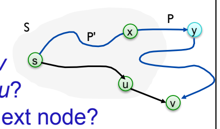
- We always form **shortest new *s-v* path** from a path in S followed by a *single* edge
- Proof of optimality: *Stays ahead* of all other solutions
  - Each time selects a path to a node *v*, that path is shorter than every other possible path to *v*

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: |S|=1 …
- Inductive hypothesis?
- Next step?

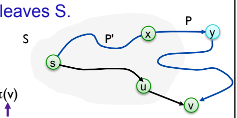
Dijkstra's Algorithm: Proof of Correctness

- 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
  - 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

- Invariant. 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 $\pi(v)$
  - 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.

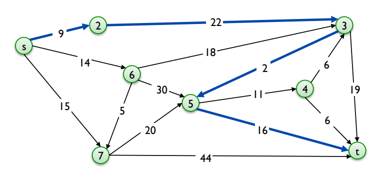


In terms of inequalities:

$$\underbrace{\ell(P) \geq \ell(P')}_{\text{nonnegative weights}} + \ell(x,y) \underbrace{=}_{\text{inductive hypothesis}} d(x) + \ell(x,y) \underbrace{\geq}_{\text{defn of } \pi(y)} \pi(y) \underbrace{\geq}_{\text{Dijkstra chose v instead of y}} \pi(v)$$

Discussion: Dijstra's Algorithm

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



Looking Ahead

- Read 3.6, 4, 4.1, 4.2, 4.4
  - Wiki due next Wednesday
- Exam due Friday
- Wednesday: Exam work day
  - I'll be available for questions