## Objectives

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

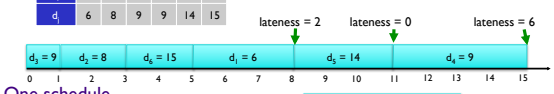


## Review: Scheduling to Minimizing Max Lateness

- 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 $\ell_j$

Input:

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




One schedule

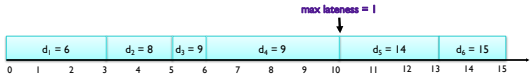
Note: **not** a sum total



## Minimizing Lateness: Greedy Algorithm

- 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]
```

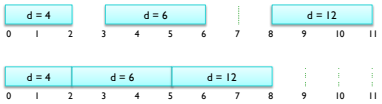



What can we say about this algorithm/its results?



## Minimizing Lateness: No Idle Time

- Observation. There exists an optimal schedule with no idle time




- Observation. The greedy schedule has no idle time



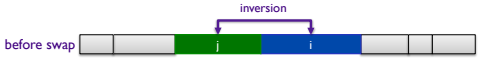
## Proving Optimality

- Goal: Prove greedy algorithm produces optimal solution
- Approach: **Exchange argument**
  - Start with an optimal schedule Opt
  - Gradually modify Opt, preserving its optimality
  - Transform into a schedule identical to greedy's schedule



## Minimizing Lateness: Inversions

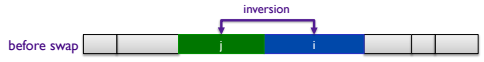
- Def. An ***inversion*** in schedule S is a pair of jobs *i* and *j* such that:
  $d_i < d_j$ (*i*'s deadline is before *j*)
  but *j* scheduled before *i*




Can Greedy's solution have any inversions?

## Minimizing Lateness: Inversions

- Def. An ***inversion*** in schedule S is a pair of jobs *i* and *j* such that:
  $d_i < d_j$ (*i*'s deadline is before *j*)
  but *j* scheduled before *i*




Greedy's schedule has no inversions!



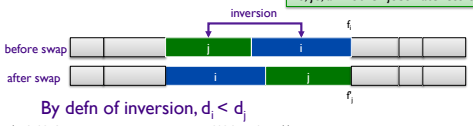
## Minimizing Lateness: Inversions

- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does *not increase the **max** lateness*

How do we know inversions are adjacent?

- Pf Setup. Let $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards

What can we say about how i's, j's, and other jobs' lateness changes?




By defn of inversion, $d_i < d_j$



## Minimizing Lateness: Inversions

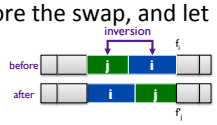
- Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness
- Pf. Let $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards
  - Lateness remains the same for all other jobs:
    - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell_j \leq \ell_i$ because $d_i < d_j$
  - Lateness of *i* before is $\ell_i = f_i - d_i = T_{i-1} + t_i + t_j - d_i$
  - Lateness of *j* after is $\ell'_j = f'_j - d_j = T_{i-1} + t_i + t_j - d_j$
    - But $d_i < d_j$

Put in terms of $\ell_i$




## 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 $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards
  - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell'_i \leq \ell_i$
  - 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 maximum lateness of jobs does not increase after swap

## 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 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 $e \in O$ 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.



## 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* ▪

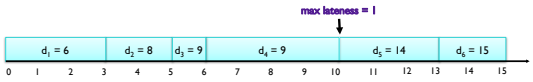


## Analyzing Running Time

- 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 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.

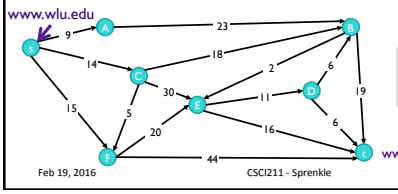


## SHORTEST PATH



## Shortest Path Problem

- Given
  - Directed graph G = (V, E)
  - Source s, destination t
  - Length $\ell_e$ = length of edge e (non-negative)
- Shortest path problem: find shortest directed path from s to t



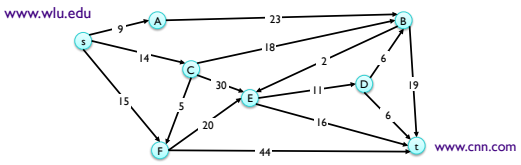

cost of path = sum of edge costs in path

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



## Shortest Path Problem

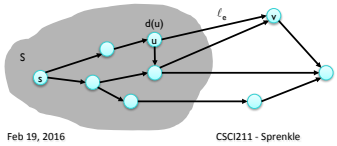
- Shortest path problem: find shortest directed path from s to t
- Brainstorming on solution …



## 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)

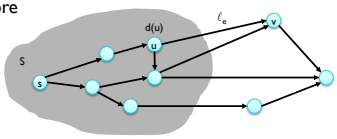
shortest path to some u in explored part followed by a single edge (u, v)



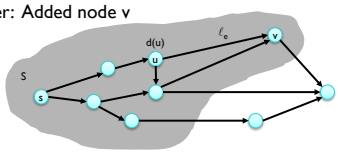



## Dijkstra's Algorithm

Before




After: Added node v

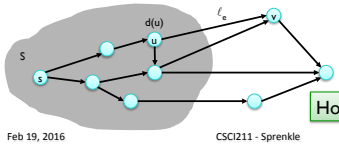





## Dijkstra's Algorithm

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?

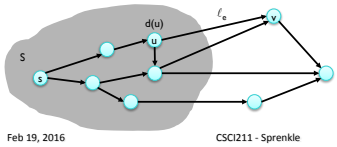
- We always form **shortest new *s-v* path** from a path in S followed by a *single* edge
- Proof of optimality: *Stays ahead* of all other solutions
  - Each time selects a path to a node *v*, that path is shorter than every other possible path to *v*



## 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)

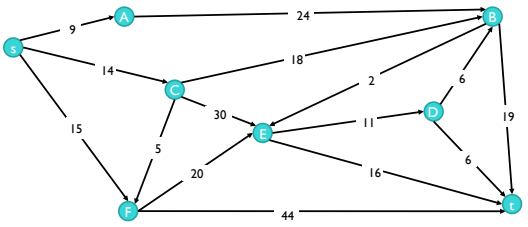



Implementation Ideas
- What to represent?
- How to represent?



## Dijkstra's Shortest Path Algorithm

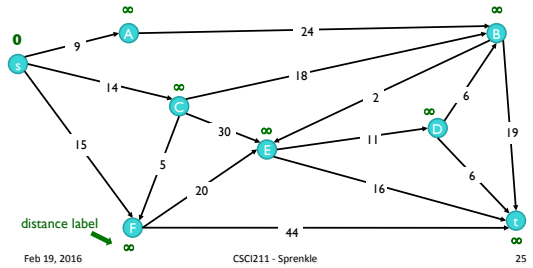
- Find shortest path from s to t



## Dijkstra's Shortest Path Algorithm

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

Initialize distances to all nodes to infinity

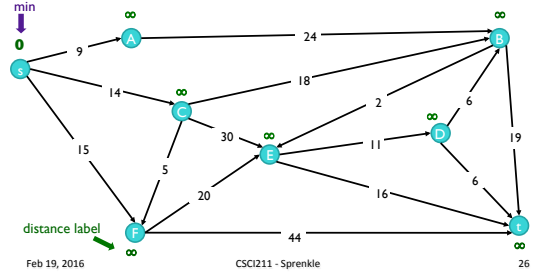





## Dijkstra's Shortest Path Algorithm

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

Delete min



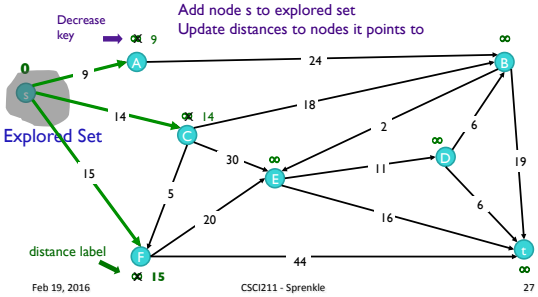



## Dijkstra's Shortest Path Algorithm

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

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

Decrease key



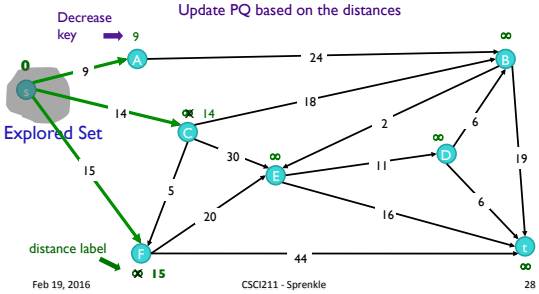



## Dijkstra's Shortest Path Algorithm

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

Update PQ based on the distances

Decrease key



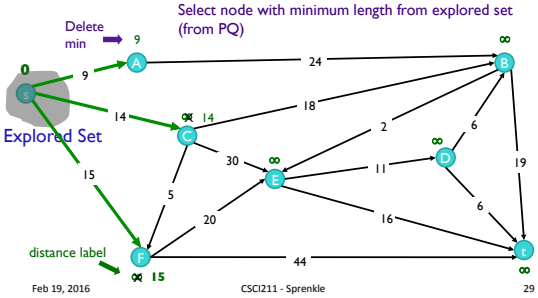



## Dijkstra's Shortest Path Algorithm

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

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

Delete min



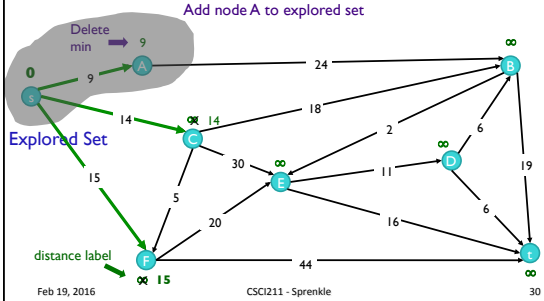



## Dijkstra's Shortest Path Algorithm

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

Add node A to explored set

Delete min



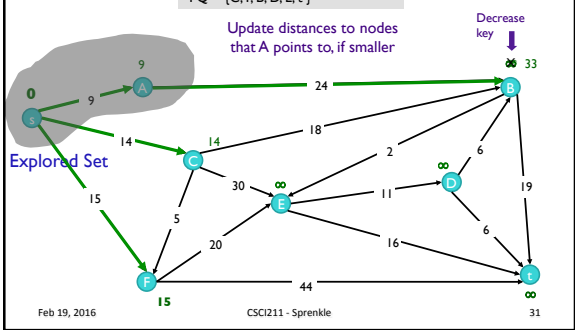
## Dijkstra's Shortest Path Algorithm

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

Update distances to nodes that A points to, if smaller

Decrease key

Explored Set





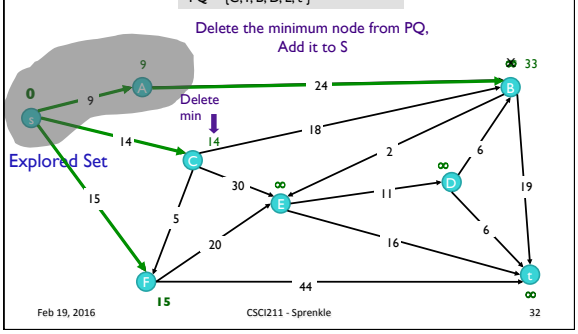

## Dijkstra's Shortest Path Algorithm

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

Delete the minimum node from PQ, Add it to S

Delete min

Explored Set



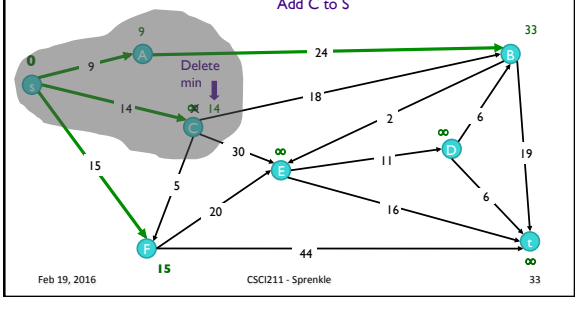



## Dijkstra's Shortest Path Algorithm

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

Add C to S

Delete min

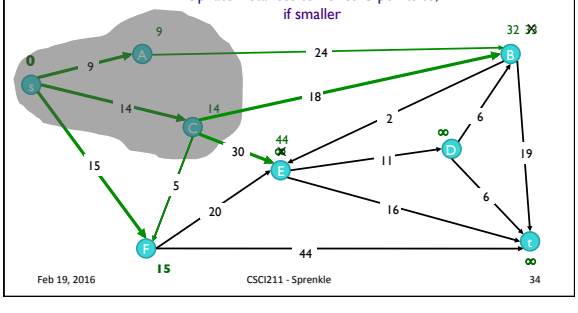





## Dijkstra's Shortest Path Algorithm

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

Update distances to nodes C points to, if smaller



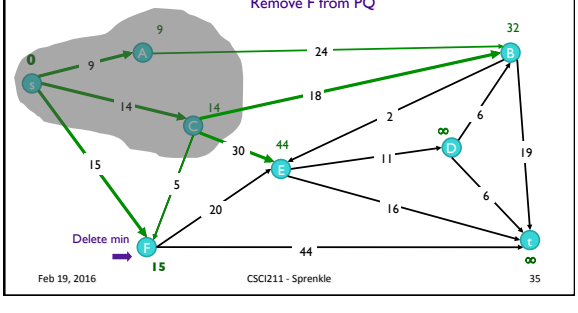



## Dijkstra's Shortest Path Algorithm

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

Remove F from PQ

Delete min



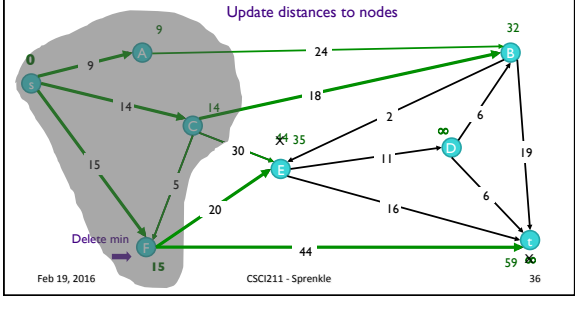



## Dijkstra's Shortest Path Algorithm

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

Add F to S, Update distances to nodes

Delete min

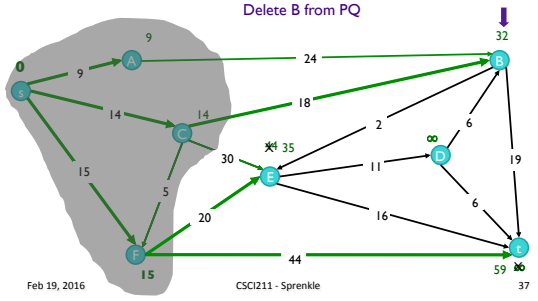


## Dijkstra's Shortest Path Algorithm

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

Delete B from PQ

Delete min



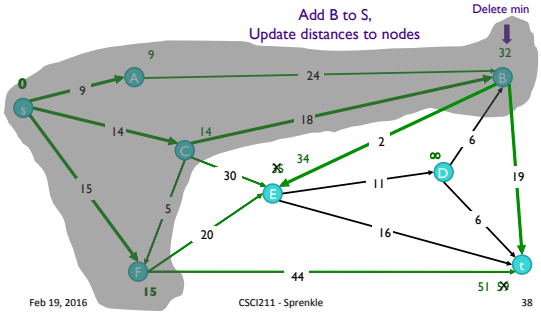



## Dijkstra's Shortest Path Algorithm

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

Add B to S,
Update distances to nodes

Delete min



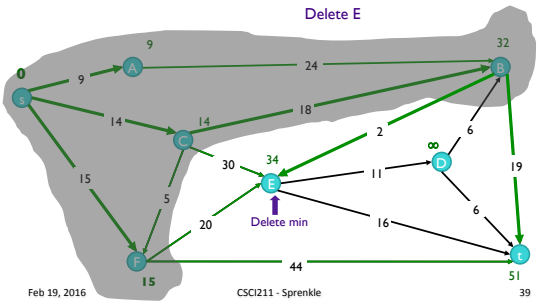



## Dijkstra's Shortest Path Algorithm

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

Delete E

Delete min



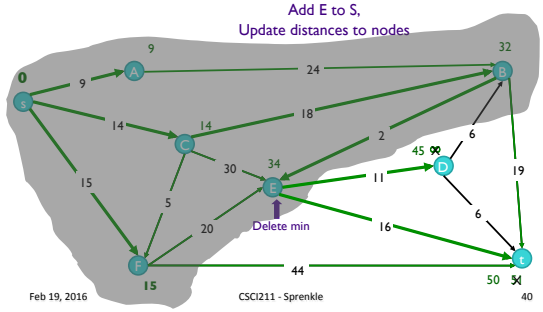



## Dijkstra's Shortest Path Algorithm

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

Add E to S,
Update distances to nodes

Delete min



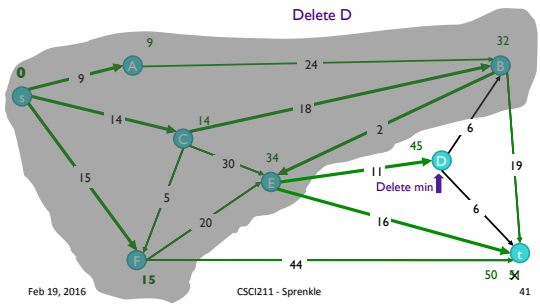



## Dijkstra's Shortest Path Algorithm

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

Delete D

Delete min

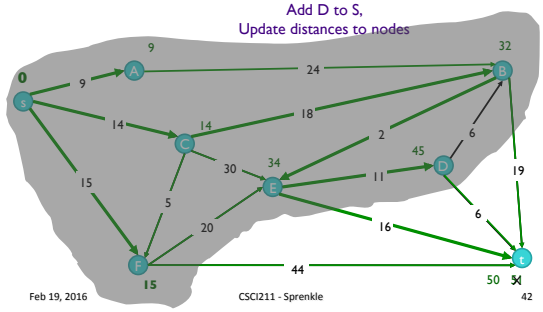





## Dijkstra's Shortest Path Algorithm

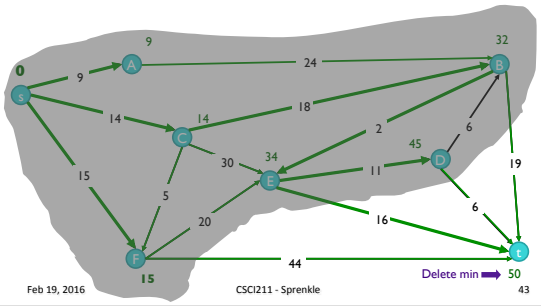
S = { s, A, C, F, B, E, D }
PQ = { t }

Add D to S,
Update distances to nodes



## Dijkstra's Shortest Path Algorithm

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

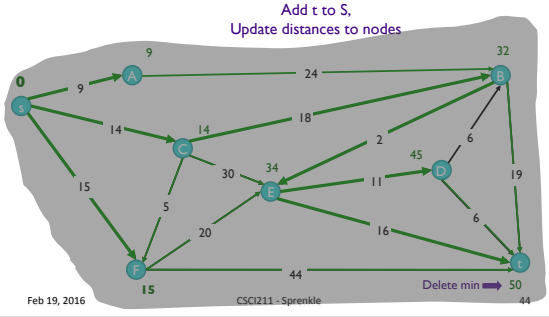





## Dijkstra's Shortest Path Algorithm

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

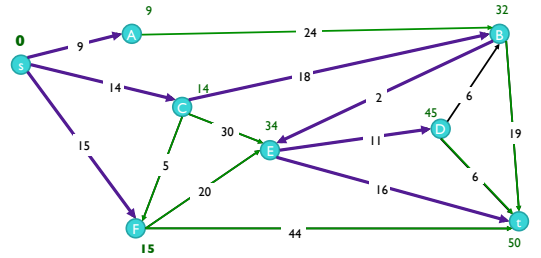
Add t to S,
Update distances to nodes






## Dijkstra's Shortest Path Algorithm

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






## 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