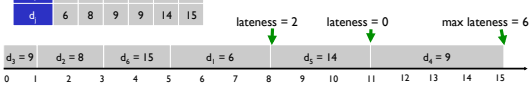
Objectives

- Wrap Up: 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** $L = \max \ell_j$

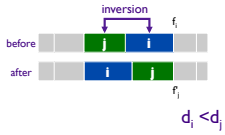
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Note: **not** a sum total

Minimizing Lateness: Inversions

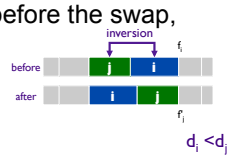
- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does *not increase the max lateness*.
- How to prove?



$d_i < d_j$

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



$d_i < d_j$

Minimizing Lateness: Inversions

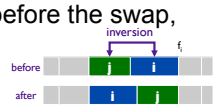
- Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness
- Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards
  - Lateness remains the same for all other jobs:
    - $\ell'_k = \ell_k$ for all $k \neq i, j$
  - $\ell_j \leq \ell_i$ because $d_i < d_j$
  - 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 ℓ_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 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:

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



$d_i < d_j$

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 = $\{a_1, a_2, ..., a_k\}$ 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 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 ∈ 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.

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

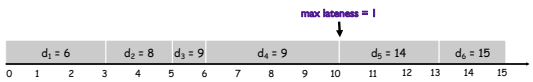
- Theorem. Greedy schedule S is optimal
- Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens
  - Can assume S* has no idle time
  - If S* has no inversions (and no idle time), then S = S*
  - If S* has an inversion, let i-j be an adjacent inversion
    - 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 d_1 ≤ d_2 ≤ … ≤ d_n
t = 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j = t
   f_j = t + t_j
   t = t + t_j
output intervals [s_j, f_j]
```

O(n logn)



What is the runtime of this algorithm?

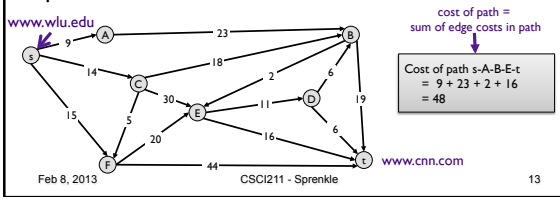
Greedy Analysis Strategies

- 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

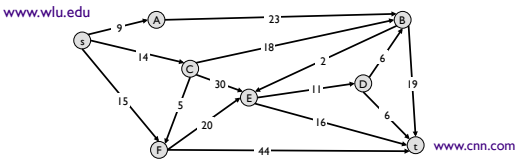
Shortest Path Problem

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



Shortest Path Problem

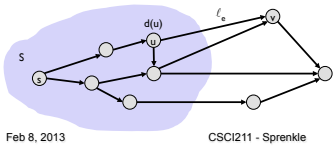
- Shortest path problem: find shortest directed path from s to t
- Brainstorming on solution …



Dijkstra's Algorithm

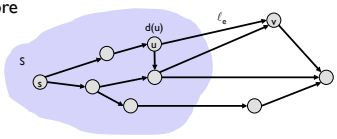
1. Maintain a set of **explored nodes** S
   - Keep the shortest path distance d(u) from *s* to *u*
2. Initialize S={s}, d(s)=0, ∀u≠s, d(u)=∞
3. Repeatedly choose unexplored node *v* which minimizes $\pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$,
   - Add v to S and set d(v) = π(v)

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

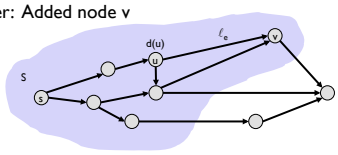


Dijkstra's Algorithm

Before



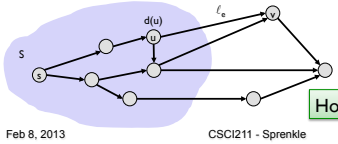
After: Added node v



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?

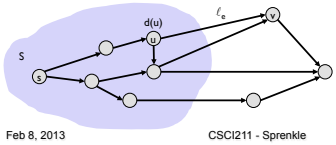
How is Algorithm Greedy?

- 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?

Looking Ahead

- Exam due today at 5 p.m.
- Wiki due Tuesday for sections 3.4-3.6; chapter 4 (front matter), 4.1
  - Directed graphs, topological order
  - Greedy algorithms
- PS4 due Friday