Greedy Algorithms

- easy algorithms
- more difficult proofs

two primary proof ideas

- Greedy stays ahead - today
- Exchange argument - next week

Interval scheduling

set of intervals $[s(i), f(i))$

- sort by $f(i)$
- schedule the earliest $f(i)$
  - remove all conflicts
- output solution set A

Proof Idea: Greedy stays ahead of any optimal solution.

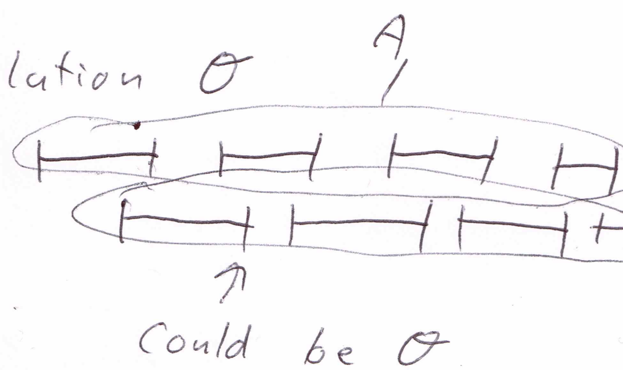
Proof: Assume an optimal solution $\mathcal{O}$

$A = i_1, i_2, \ldots, i_k$ $\quad |A| = k$

$\mathcal{O} = j_1, j_2, \ldots, j_m$ $\quad |\mathcal{O}| = m$

we want to show $k = m$

ex



assume (or sort) A and $\mathcal{O}$ by time (s or f is the same)

Greedy stays ahead is most times a proof by induction.

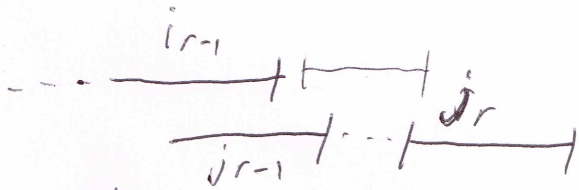
lemma / want to prove that $f(i_r) \leq f(j_r)$, by induction!

<u>base case</u>: $r=1$ $\quad f(i_1) \leq f(j_1)$

by definition of Algo, this true

<u>induction</u>: $r>1$ $\quad$ IH: assume true for ~~ff~~ $r-1$ s.t. $f(i_{r-1}) \leq f(j_{r-1})$

show true for r



Greedy algorithm always has the choice of j_r and maybe an earlier interval.

$\therefore f(i_r) \leq f(j_r)$ $\quad$ ∎ of lemma

---

finish the proof!!!... by contradiction

assume $m>k$ leads to a contradiction

at $r=k$ we know that $f(i_k) \leq f(j_k)$

Greedy will be able to pick j_{k+1}. By the definition of algorithm, it will pick j_{k+1} or better. Contradiction of the greedy algorithm.

$\therefore$ ~~$m \geq k$~~ $m \leq k$

Since O is optimal, $m \not< k$

$\therefore m = k$ ∎

n intervals

runtime of interval scheduling

- sort $O(n\log(n))$ (mer sort or your favorite n log n sort)
- iterate $O(n)$ times
  - pick ~~sm~~ earliest finish time $O(1)$
  - eliminate conflicts $O(n)$ ← can we do better?

- $O(n^2)$

Create array $S[i]$ where $S[i] = s[i]$
sorted by finist time $f(i)$

iterate through S

- take the first one.
- iterate through eliminating all intervals that conflict with the previous $f(i)$.
- take first interval without conflict.
- repeat until the end of S is reached.

new ~~algo~~ implementation

- sort $O(n\log n)$
- create S $O(n)$
- iterate through S as explained above. $O(n)$

$O(n\log n)$ runtime