Greedy Algorithms

A greedy algorithm you all know:

Make change for \$3.47.

$$\begin{array}{rl} 1 & \times \$2 \\ 1 & \times \$1 \\ 1 & \times 25¢ \\ 2 & \times 10¢ \\ 2 & \times 1¢ \\ \hline & 7 \text{ coins} \end{array}$$

Claim: This is the minimum number of coins.

Exercise: (not easy) Prove that the greedy method of making change works for the Canadian coin system.

Does the greedy method work for every possible coin system?

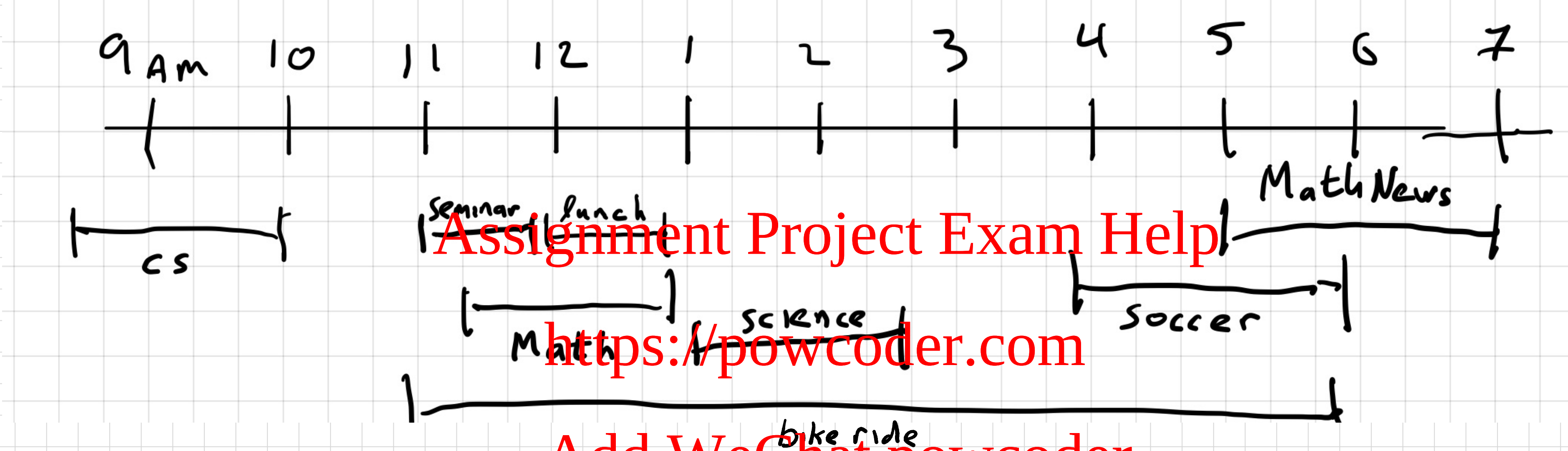
1¢ 6¢ 7¢ coins. Make change for 12¢.

Greedy: 7¢ + 5 × 1¢ Better: 2 × 6¢

Claim: The greedy change algorithm can be implemented in polynomial time using quotients and remainders.

Interval Scheduling or "Activity Selection"

Given a set of activities, each with a specified time interval,
select a maximum set of disjoint (= non-intersecting) intervals.

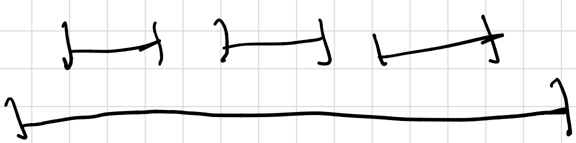


Greedy Approach:

- pick one activity greedily
- remove conflicts
- repeat

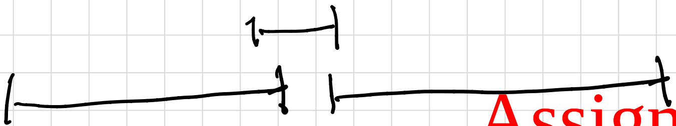
There are several possible greedy approaches.

1. select activity that starts earliest



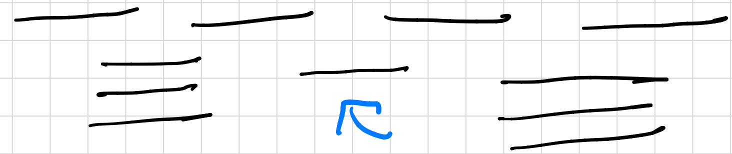
No.

2. select the shortest interval



No.

3. select the interval with fewest conflicts



No.

4. select the interval that ends earliest

For above we get cs, seminar, lunch, science, soccer

Slick implementation of approach 4:

Sort activities $1..n$ by end time.
$A := \emptyset$
for i **from** 1 **to** n **do**
    if activity i does not overlap $\underbrace{\text{with any activies in}}_{\text{just need to check last one}}$ A **then**
        $A := A \cup \{i\}$
    fi
od

Analysis:

$O(n \log n)$ to sort.
$O(n)$ for the loop.
Thus $O(n \log n)$ overall.

Correctness: We will see two basic ways to show greedy algorithms are correct:

1. greedy stays ahead all the time
2. "exchange" proof

Sketch of proof of correctness using method 1. (Formal proof by induction on next page.) Suppose greedy algorithm returns

$$a_1, a_2, \ldots, a_k,$$

sorted by endtime. Suppose an optimal solution is

$$b_1, b_2, \ldots, b_k, b_{k+1}, b_{k+2}, \ldots, b_\ell,$$

sorted by endtime.

Claim: $a_1, b_2, \ldots, b_k, b_{k+1}, b_{k+2}, \ldots, b_\ell$ is an optimal solution.

Why? $\text{end}(a_1) \leq \text{end}(b_1)$ so a_1 doesn't intersect with b_2.

Claim: $a_1, a_2, \ldots, b_k, b_{k+1}, b_{k+2}, \ldots, b_\ell$, is an optimal solution.

Why? b_2 does not intersect a_1 so greedy algorithm could have chosen it. Instead, it chose a_2: so $\text{end}(a_2) \leq \text{end}(b_2)$, leaving intervals distinct.

$\vdots$

Claim: $a_1, a_2 \ldots, a_k, b_{k+1}, \ldots, b_\ell$ is an optimal solution.

Claim: $k = \ell$ otherwise greedy algorihm would have continued to choose more intervals.

Here we use method 1.

Lemma: This algorithm returns a maximum size set A of disjoint intervals.

Proof: Let $A = \{a_1, \ldots, a_k\}$, sorted by end time.

Compare to an optimum solution $B = \{b_1, \ldots, b_\ell\}$, sorted by end time.

Thus $\ell \geq k$ and we want to prove $\ell = k$.

Idea: At every step we can do at least as good with the a_i's.

Claim: $a_1 \ldots a_i b_{i+1} \ldots b_\ell$ is an optimal solutions for all i

Proof: by induction on i

basis $i = 1$. a_1 had earliest end time of all intervals so end$(a_1) \leq$ end(b_1).

So replacing b_1 by a_1 gives disjoint intervals.

induction step Suppose $a_1 \ldots a_{i-1} b_i \ldots b_\ell$ is an optimal solution.

b_i does not intersect a_{i-1} so the greedy algorithm could have chosen it.

Instead, it chose a_i, so

$$\text{end}(a_i) \leq \text{end}(b_i)$$

and replacing b_i by a_i leave disjoint intervals.

This proves the claim. To finish proving the lemma:

If $k < \ell$ then $a_1 \ldots a_k b_{k+1} \ldots b_\ell$ is an optimal solution.

But then the greedy algorithms had more choices after a_k.

Another example of a greedy algorithm: Scheduling to minimize lateness.

assignments	time required	deadline
CS341	4 hrs	in 9 hrs
Math	2 hrs	in 6 hrs
Philosophy	3 hrs	in 14 hrs
CS350	10 hrs	in 25 hrs

Can you do everything by its deadline (ignoring sleep!)

How? (no parallel processing!)

Optimization Version (more general)

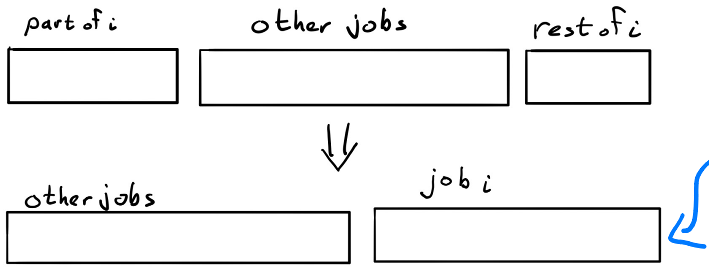
find a schedule, allowing some jobs to be late, but minimizing the maximum lateness

Note: this is different from minimizing sum of lateness
(= minimum average lateness)

Q: Why is the optimization problem more general?
A: A schedule completes all jobs on time if and only if its maximum lateness is 0.

Notation: Job i takes time t_i and has deadline d_i

Observation 1. You might as well finish a job once you start.



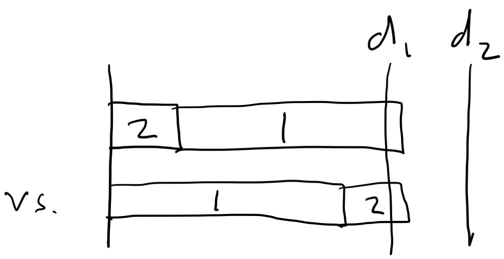
This is at least as good: the other jobs finish earlier and job i finished at same time.

Thus, each job should be done contiguously.

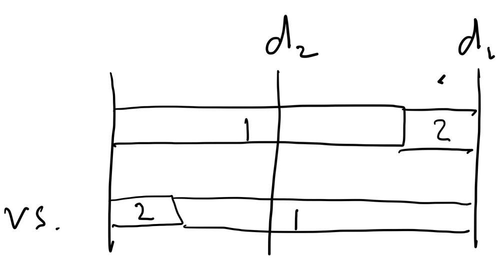
Observation 2. There's never any value in taking a break.

What are some greedy approaches?

- do short jobs first — not correct



- do jobs with less slack first: slack $= d_i - t_i$ — not correct



- jobs in order of deadline

  i.e., order jobs such that $d_1 \leq d_2 \leq \cdots \leq d_n$ and do them in that order

  check that this works on above examples

Greedy algorithm: order job by deadline, so $d_1 \leq d_2 \leq \cdots \leq d_n$.

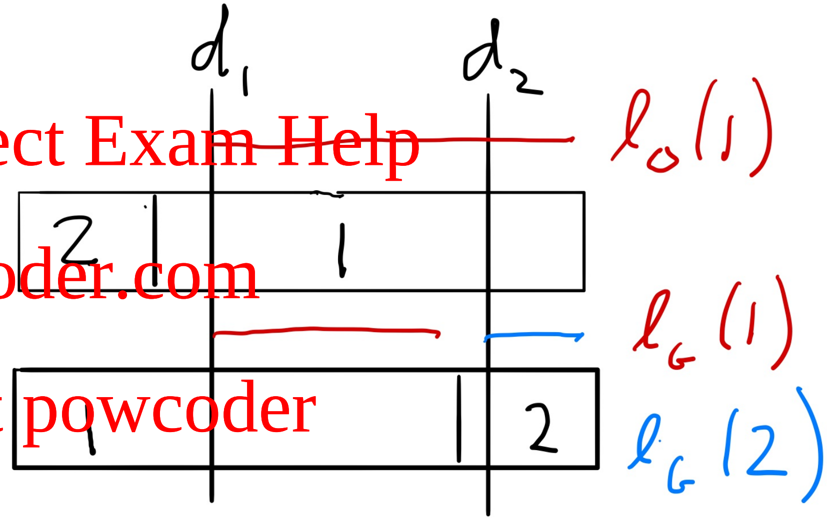
We will show that the greedy algorithm minimizes lateness.

Advice about proofs:

Don't be general at first! Try special cases!
What is a good special case here?
$n = 2,\, d_1 < d_2$

the wrong/other solution, O:

the greedy solution, G:



O has job 2 before job 1 G has job 1 before job 2
$\ell_O(1)$ = lateness of job 1 in O, etc. for $\ell_O(2), \ell_G(1), \ell_G(2)$
ℓ_G - maximum lateness of greedy schedule $= \max\{\ell_G(1), \ell_G(2)\}$
ℓ_O - maximum lateness of other schedule $= \max\{\ell_O(1), \ell_O(2)\}$
$\ell_G(1) \leq \ell_O(1)$ because we moved 1 earlier
$\ell_G(2) \leq \ell_O(1)$ because $d_1 \leq d_2$
Therefore $\ell_G \leq \ell_O(1) \leq \ell_O$

Can we generalize?
The idea allows us to swap a pair of consecutive jobs if their deadlines are out of order, making the solution better (or at least not worse).
Next: a proof that greedy gives an optimal solution using an "exchange proof."

Theorem: The greedy algorithm gives an optimal solution, i.e., one that minimizes the maximum lateness.

Proof: – an "exchange proof" that converts any solution to the greedy one without increasing the maximum lateness.

Let $1, \ldots, n$ be ordering of jobs by greedy algorithm, i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$. Consider an optimal ordering of jobs. If it matches greedy, fine. Otherwise there must be two jobs that are consecutive in this ordering but in wrong order for greedy: i, j with $d_j \leq d_i$.

Claim: Swapping i and j gives a new optimal ordering. Furthermore, the new optimal ordering has fewer inversions. So repeated swaps will eventually give us the greedy ordering, which must then be optimal.

Aside: recall that an inversion is a pair out of order. Doing a swap of two consecutive elements that are out of order decreases the number of inversions.

e.g. 2 5 3 1 4 # inversions 5

swap

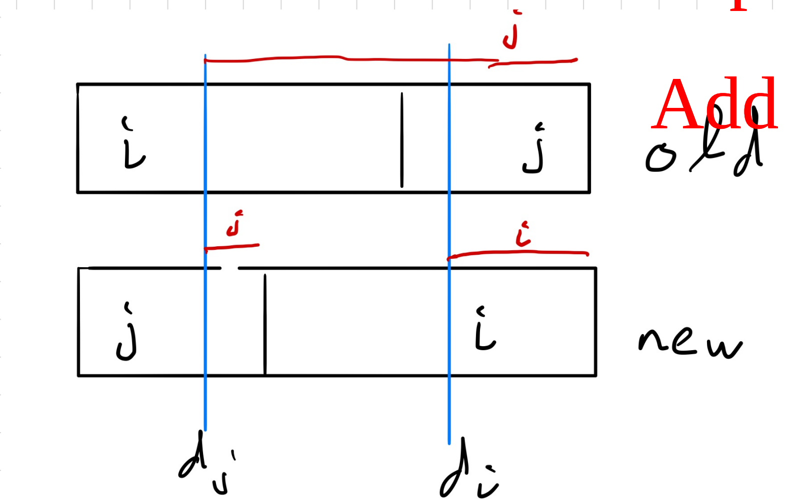
2 3 5 1 4 4

2 3 1 5 4 3

∴ eventually get sorted order

Proof of claim:

Consider swapping jobs i and j.



$\ell_N(j) \le \ell_O(j)$ because now we do j first

$\ell_N(i) \le \ell_O(j)$ because $d_j \le d_i$

And all other jobs have same lateness.

Thus $\ell_N \le \ell_O$. But ℓ_O was minimum. So $\ell_N = \ell_O$.

So we can swap until we get the greedy solution, ℓ unchanged.