1

Lecture Greedy Algorithms Davos Camp 2016

Daniel Graf
daniel@soi.ch

What you all know (very) well:

- Simple iterative algorithms
  e.g. insertion sort
  1 3 7 9 5 8
- Recursion-based algorithms
  - divide-and-conquer
  - backtracking
  - dynamic programming

  } solve some subproblems and then combine the solutions

Outline 90mins

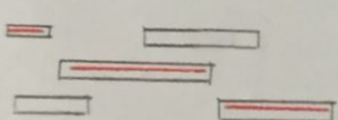
- 5 Intro
  - 3 techniques for analysis
    - 15 stay ahead (Interval scheduling)
    - 15 tight bounds (Interval coloring)
    - 15 exchange argument (Min Lateness)
- Applications / Tasks
  - 10 stabbing (tight bound, stay ahead)
  - 10 files on tape (exchange)
  - 20 Huffman tree (exchange)

- Today: Greedy algorithms
  - build the optimal solution step by step, making the best choice available in each step and never look → it „magically" results in the optimal solution
    tricky to analyze (show correctness), easy to get wrong („It works on all my examples, so it has to be correct" ☹)
  - If it works: very nice, simple & iterative
  - Greedy algorithms never work!!!*
    * Unless you prove otherwise. Instead try dynamic programming.

This Lecture:
- 3 techniques to argue about correctness of greedy algorithms, each with example.
- more applications/tasks in the end
- Sources: lecture notes by Jeff Erickson, slides by Adam Smith (see website)

1st technique: Greedy stays ahead

Example: Chaos Camp Lecture Schedule:
- Given a list of intervals (s_i, f_i), start and finish time of each lecture, no two start or end at the same time
- Want: Visit as many lectures as possible



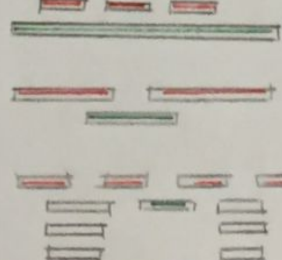
Greedy Ideas

- Earliest start time first
- Shortest lecture first
- Fewest conflict first

(they all work on the example)

Counter examples

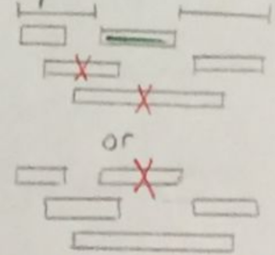
OPT
GREEDY



Golden Rule: try DP instead

- Subproblem: take lecture or not

take



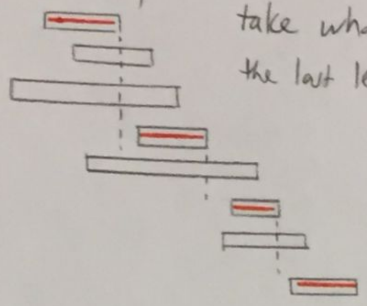
or

not

→ recurse on these two intervals
→ runtime $O(n^2)$ achievable

2 Working Greedy

- sort by deadline then greedily take whatever does not overlap the last lecture already chosen.



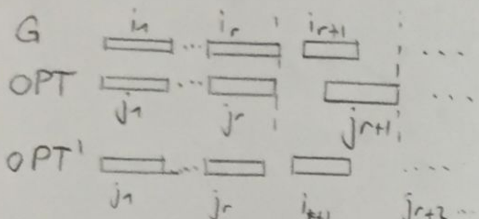
→ runtime $O(\underbrace{n \log n}_{\text{sortning}} + \underbrace{n}_{\text{sweep}})$

→ but can we prove it?

```
GREEDY SCHEDULE (S[], F[])
  sort F increasing and S accordingly
  count = 1
  X[count] = 1   // X stores the lectures we take
  for i = 2 to n
     if S(i) ≥ F[X[count]]
        count++
        X[count] = i
  return X
```

Proof: by contradiction, i.e. assume greedy solution G is not optimal

- Let OPT be an optimal solution. Which one?
- OPT agrees with G for as many initial lectures as possible
- OPT ≠ G (as we assume that G is not optimal)
- Look at first place where OPT and G differ:



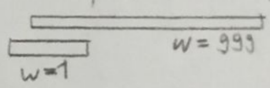
lecture i_{r+1} finishes before j_{r+1}

replacing j_{r+1} by i_{r+1} in OPT gives a new optimal solution OPT' that agrees more with G, a contradiction ↯ Hence G must also be optimal. □

So this greedy approach is correct because a greedy solution never lags behind any optimal solution.

Variant: Some lectures are more important than others → weighted version

Greedy fails horribly bad

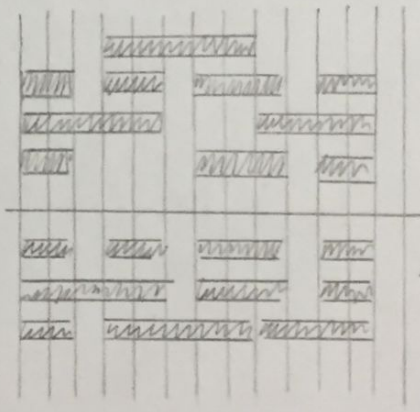


→ dynamic programming works

2^{nd} technique: tight bounds

Example: Lecture Room Assignments @ Chaos Camp

- Given: lecture intervals (s_i, f_i)
- Wanted: minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room



Question: Can we also do it with only 2 rooms?

No! There are 3 lectures at the same time.

Let d be the max. number of concurrent lectures at any time („depth" of the lecture schedule)

Claim: d rooms are always enough!

(It is clear that we need at least d rooms.)

Proof: by algorithm
- consider lectures in increasing order of start time
- assign lectures to any available classroom → code
- implementation takes $O(n \log n)$ time, $O(n)$ space
- number of classrooms needed by GREEDYCLASSROOM is d: when the d^{th} classroom is allocated we were considering some lecture (s_j, f_j). As we sorted by start time, all $d-1$ lectures that block (s_j, f_j) have started before s_j and end after s_j □

```
GREEDYCLASSROOM(S[], F[])
  sort S increasingly and F accordingly
  d=0 // number of classrooms
  T[] // end of last lecture in each room
  for i = 1 to n
    if min_d T[d] > S[i]: { d++
                            T[d] = F[i]
    else: T[argmin_d T(d)] = F[i]
  return d
```

3rd technique: Exchange Argument

Example: Homework @ Chaos Camp

Given: tasks each with time t_i that is needed and a deadline d_i when the task should have been completed

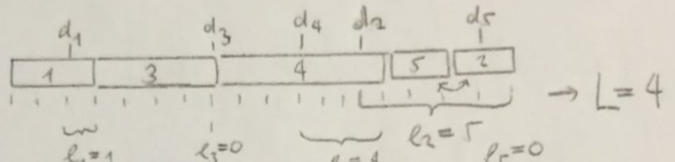
Wanted: order of the tasks that minimizes the maximum lateness L.

Find s_i, f_i for every task s.t. no two intervals overlap

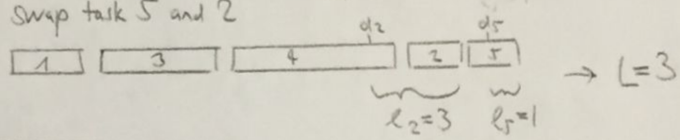
$$l_i = \max(0, f_i - d_i)$$
$$L = \max(l_i)$$

Example 5 tasks:

t_i	3	2	4	7	2
d_i	2	13	7	10	17



Swap task 5 and 2

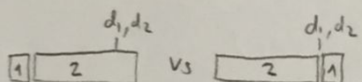


Greedy template: consider the jobs in some order

But what is a good order?

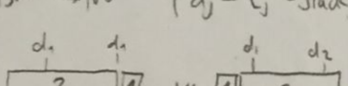
- Shortest Processing Time First → No

t_i	1	10
d_i	10	10



- Smallest Slack First → No $(d_j - t_j = slack_j)$

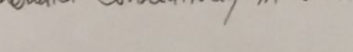
t_i	1	10
d_i	2	10



- Earliest Deadline First → Yes!

```
  sort tasks by increasing deadline
  t=0; L=0
  for i = 1 to n
    //assign task i to interval [t, t+t_i]
    s_i = t; f_i = t + t_i; t += t_i
    l_i = f_i - d_i; L = max(L, l_i)
  return L
```

Proof by contradiction, i.e. assume that greedy solution G is not optimal, so has a higher lateness than necessary.
- Let OPT be the optimum solution with the fewest inversions in the order of deadlines of the task schedule. (inversion = two tasks i, j where $d_j > d_i$ but j before i in OPT)
- As OPT ≠ G, OPT has at least one inversion. In that case, there is also an inverted pair of tasks that are scheduled consecutively in OPT:

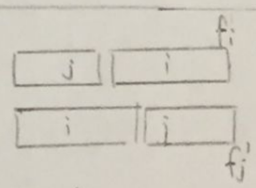


$d_i < d_j$

→ Idea: Let's see what happens if we swap i and j.

Before the swap OPT

After the swap OPT'



Claim: Lateness does not increase:

l = max lateness in OPT, l' = max lateness in OPT'

Proof
$$l_j' = f_j' - d_j \quad \text{// by definition}$$
$$= f_i - d_j \quad \text{// same finish time}$$
$$\leq f_i - d_i \quad \text{// sorted by } d: d_i \leq d_j$$
$$= l_i \leq l$$

also: $l_i' \leq l_i \leq l$ and $l_k' = l_k$ for all $k \notin \{i, j\}$

hence: $l' \leq l$ and OPT has not minimal # inversions ↯ □

Exchange Argument

- Assume that there is an optimal solution that is different from the greedy solution
- Find the „first" difference between the two solutions
- Argue that we can exchange the optimal choice for the greedy choice without degrading the solution

→ Inductive argument, similar to a DP proof.

⌈Side note: Minimizing the sum of latenesses is hard!⌉

Summary of the techniques

- Greedy stays ahead
  In every step greedy is at least as good as OPT
- Tight bounds
  Show a simple lower bound on the solution. Then show that greedy achieves it.
- Exchange argument
  Transform any solution into the greedy one step by step.

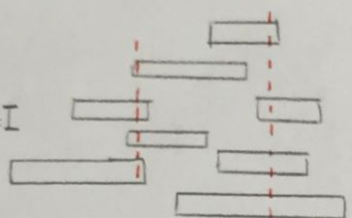
Applications

Task: Lecture Police

At some moments in time the police wants to check all currently running lectures to assure their quality. They want to be discreet, so check at as few times as possible but still visit all the lectures.

Given: n intervals $[s_i, f_i]$

Wanted: fewest times $t_1, t_2, \dots, t_T$
s.t. $\forall$ interval $I, \exists$ time t: $t \in I$



Solution
- Compute largest non-overlapping set (as in the first example: greedy by end time)
- Then take all end times of these lectures as check points

Proof:
- Stay ahead argument (or rather: "stay behind" here)
  We take the first check point as late as possible.
  Then we can use the same „modify the OPT" argument as before
- Tight bound argument
  The non-overlapping lecture set found by the greedy algorithm clearly is a lower bound on the solution. Why is it enough?

Task FileTape
- n files of lengths $L[i]$ are stored on a tape
- reading file i costs $\sum_{j=1}^{i} L[j]$ as we have to read everything before as well
- how should we arrange the files on the tape to minimize the total reading time $\sum_{i=1}^{n}\sum_{j=1}^{i} L[j]$

Solution
- Order the files in increasing size [1][2][5]
  $1+(1+2)+(1+2+5)$
- It seems obviously true and now we can also very easily show it

Proof. Assume OPT is not sorted so $L[i] > L[i+1]$ for some
Then we swap i and $i+1$ to get] [L[i-1]] [L[i+1]] [L[i]] [...]
with a cost that changed by
$L[i] - L[i+1] < 0$ so the total cost decreased ↯
now in one summand more / now in one summand less

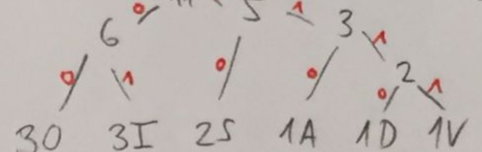
Variants:
- equal size, but with access frequencies $F[i]$
  $\min \sum_{i=1}^{n} i \cdot F[i]$ → sort decreasingly by $F[i]$
- both $L[i]$ and $F[i]$ ⇝ Think about it.

Huffman Codes (just the idea, you can now understand the proof)

Encode: "SOI IOI DAVOS" into morse code
but with an optimal morse code (binary sequence for letter)

Question: How do we find the optimal encoding scheme that we can revert (so called prefix-free) but takes as few bits for our message as possible

Steps:
- Analyze letter frequencies: 3xO, 3xI, 2xS, 1xA, 1xD, 1xV
- Build tree greedily by always combining least frequent pair



- Encode
  10|00|01|01|00|01
  1110|110|1111|00|
  10
  not needed

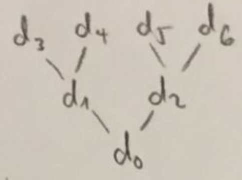
Proof
- Show that two least frequent letters can be siblings (neighboring leafs) in an optimal tree → Implies that a last step can be greedy
- Show that we can do this repeatedly (induction on the tree)

→ See linked lecture notes for details

5 Task Yoghurt

Key Idea:

Do it from the end!

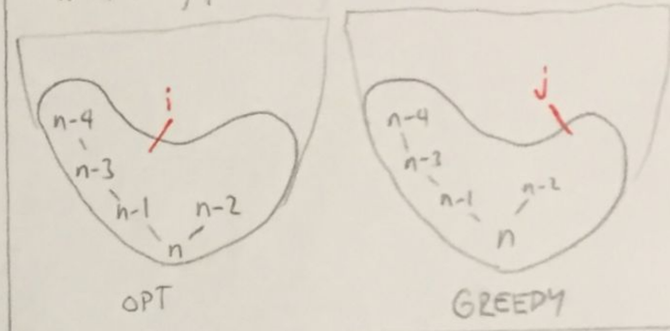


Which yoghurt does Daniel eat last?

Why can we do greedily: take the available yoghurt with the highest deadline the latest

Proof: Greedy stays ahead (from behind)

Let OPT be the solution that agrees with Greedy the most when looking from behind



We know $d_i < d_j$ because Greedy takes j not i.

OPT $[\cdots]\, j\, [\cdots]\, i\, \underbrace{\cdots\cdots}_{\text{same as greedy}}$

OPT' $\underbrace{[\cdots]}_{\text{unchanged}}\,\underbrace{[\cdots]\, i}_{\text{done earlier}}\, \underset{\substack{\uparrow \\ \text{fine as} \\ d_i < d_j}}{j}\, \underbrace{\qquad\qquad}_{\text{same as greedy}}$

$\lightning$ $\square$