CS 341: Algorithms

Lecture 5: Greedy algorithms

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Goals

This chapter: the greedy paradigm through examples

- job scheduling
- interval scheduling
- more scheduling
- fractional knapsack
- and so on

Goals

This chapter: the greedy paradigm through examples

- job scheduling
- interval scheduling
- more scheduling
- fractional knapsack
- and so on

Computational model:

- word RAM
- assume all quantities we work with (weights, capacities, deadlines, ...) fit in a word

Overview

Greedy algorithms

Context: we are trying to solve a **combinatorial optimization** problem:

- have a **large, but finite,** set $\mathcal{S}$
- want to find an element E in $\mathcal{S}$ that **minimizes / maximizes** a cost function

Greedy algorithms

Context: we are trying to solve a **combinatorial optimization** problem:

- have a **large, but finite,** set $\mathcal{S}$
- want to find an element E in $\mathcal{S}$ that **minimizes / maximizes** a cost function

Greedy strategy:

- build E step-by-step
- don't think ahead, just try to improve as much as you can at every step
- simple algorithms
- but usually, no guarantee to get the optimal
- it is often **hard** to prove correctness, and **easy** to prove incorrectness.

Example: Huffman

Review from CS240: the **Huffman tree**

- we are given **"frequencies" $f_1, \dots, f_n$ for characters $c_1, \dots, c_n$**
- we build a **binary tree** for the whole code

Example: Huffman

Review from CS240: the **Huffman tree**

- we are given **"frequencies" $f_1, \dots, f_n$ for characters $c_1, \dots, c_n$**
- we build a **binary tree** for the whole code

Greedy strategy: we build the tree **bottom up**.

- create n single-letter trees
- define the **frequency** of a tree as the sum of the frequencies of the letters in it
- build the final tree by putting together smaller trees: **join the two trees with the least frequencies**

Claim

this minimizes $\sum_i f_i \times \{\text{length of encoding of } c_i\}$

Proof: takes some work. Progressively transform any other solution into the greedy one.

Minimizing completion time

The problem

Input:

- n jobs, with processing times $[t(1), \ldots, t(n)]$

The problem

Input:

- n jobs, with processing times $[t(1), \ldots, t(n)]$

Output:

- an ordering of the jobs that minimizes the **sum T of the completions times**
- **completion time:** how long it took **(since the beginning)** to complete a job

The problem

Input:

- n jobs, with processing times $[t(1), \ldots, t(n)]$

Output:

- an ordering of the jobs that minimizes the **sum T of the completions times**
- **completion time:** how long it took **(since the beginning)** to complete a job

Example:

- $n = 5$, processing times $[2, 8, 1, 10, 5]$
- in this order,
  $T = 2 \ + \ (8+2) \ + \ (1+8+2) \ + \ (10+1+8+2) \ + \ (5+10+1+8+2) = 70$
- in the order $[1, 2, 5, 8, 10]$,
  $T = 1 \ + \ (2+1) \ + \ (5+2+1) \ + \ (8+5+2+1) \ + \ (10+8+5+2+1) = 54$

Greedy algorithm

Algorithm:

- order the jobs in **non-decreasing** processing times

Greedy algorithm

Algorithm:

- order the jobs in **non-decreasing** processing times

Correctness (exchange argument)

- let $L = [e_1, \dots, e_n]$ be a permutation of $[1, \dots, n]$
- suppose that L is **not** in non-decreasing order of processing times.
  Can it be optimal?
- by assumption there exists i such that $\boldsymbol{t(e_i) > t(e_{i+1})}$
- sum of the completion times of L is $\boldsymbol{nt(e_1) + (n-1)t(e_2) + \cdots + t(e_n)}$
- the contribution of e_i and e_{i+1} is $\boldsymbol{(n-i+1)t(e_i) + (n-i)t(e_{i+1})}$
- now, **swap** e_i **and** e_{i+1} **to get a permutation** L'
- their contribution becomes $\boldsymbol{(n-i+1)t(e_{i+1}) + (n-i)t(e_i)}$
- nothing else changes so $\boldsymbol{T(L') - T(L) = t(e_{i+1}) - t(e_i)} < 0$
- so L **not optimal**

Greedy algorithm

Algorithm:

- order the jobs in **non-decreasing** processing times

Review from CS240

- optimal static order for linked list implementation of dictionaries
- same result (up to reverse), same proof

Interval scheduling

The problem

Input:

- n intervals $I_1 = [s_1, f_1], \ldots, I_n = [s_n, f_n]$ start time, finish time
- also write $s_j = \mathsf{start}(I_j)$, $f_j = \mathsf{finish}(I_j)$

The problem

Input:

- n intervals $I_1 = [s_1, f_1], \ldots, I_n = [s_n, f_n]$ start time, finish time
- also write $s_j = \textbf{start}(I_j)$, $f_j = \textbf{finish}(I_j)$

Output:

- a choice T of intervals that **do not overlap** and that has **maximal cardinality**

The problem

Input:

- n intervals $I_1 = [s_1, f_1], \ldots, I_n = [s_n, f_n]$ start time, finish time
- also write $\boldsymbol{s_j} = \textbf{start}(\boldsymbol{I_j})$, $\boldsymbol{f_j} = \textbf{finish}(\boldsymbol{I_j})$

Output:

- a choice T of intervals that **do not overlap** and that has **maximal cardinality**

Example: A car rental company has the following requests for a given day:

I_1: 2pm to 8pm

I_2: 3pm to 4pm

I_3: 5pm to 6pm

Answer is $T = [I_2, I_3]$.

A few attempts

Attempt 1:

- pick the interval with the **earliest starting time** that creates no conflict

A few attempts

Attempt 1:

- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

A few attempts

Attempt 1:

- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

Attempt 2:

- pick the **shortest interval** that creates no conflict

A few attempts

Attempt 1:

- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

Attempt 2:

- pick the **shortest interval** that creates no conflict
- **no**, for example

A few attempts

Attempt 1:

- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

Attempt 2:

- pick the **shortest interval** that creates no conflict
- **no**, for example 

Attempt 3:

- pick the interval with the **fewest overlaps** that creates no conflict

A few attempts

Attempt 1:

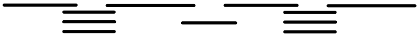
- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

Attempt 2:

- pick the **shortest interval** that creates no conflict
- **no**, for example

Attempt 3:

- pick the interval with the **fewest overlaps** that creates no conflict
- **no**, for example



A few attempts

Attempt 1:

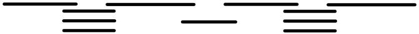
- pick the interval with the **earliest starting time** that creates no conflict
- **no**, previous example

Attempt 2:

- pick the **shortest interval** that creates no conflict
- **no**, for example

Attempt 3:

- pick the interval with the **fewest overlaps** that creates no conflict
- **no**, for example



Attempt 4:

- pick the interval with the earliest finish time, that creates no conflict

An $O(n \log(n))$ implementation

Greedy($\boldsymbol{I} = [I_1, \ldots, I_n]$)

$T \leftarrow [\,]$
sort $\boldsymbol{I}$ by non-decreasing finish time
for $k = 1, \ldots, n$ **do**
    if I_k does not overlap the last entry in T
        append I_k to T

Correctness: greedy is optimal

Let

- $T = [\boldsymbol{x_1, \ldots, x_p}]$ be the intervals chosen by algorithm,
- $S = [\boldsymbol{y_1, \ldots, y_q}]$ be any choice without overlaps,
- both sorted by increasing finish time
- want to prove $\boldsymbol{p \geq q}$

Proof (again, by an exchange argument)

- by induction: for $k = 0, \ldots, q$, $\boldsymbol{p \geq k}$ and $[\boldsymbol{x_1, \cdots, x_k, y_{k+1}, \cdots, y_q}]$ **has no overlap and is sorted by increasing finish time**
- OK for $k = 0$, so we suppose true for some $k < q$, and prove for $k + 1$
- since $[x_1, \ldots, x_k, y_{k+1}]$ is satisfiable, the algorithm didn't stop at x_k. So $\boldsymbol{p \geq k + 1}$.
- by definition of x_{k+1}, **finish**$(x_{k+1}) \leq$ **finish**(y_{k+1}). So we can replace y_{k+1} by x_{k+1} and we get $[x_1, \cdots, x_{k+1}, y_{k+2}, \cdots, y_q]$, which is still satisfiable and sorted by increasing finish time

Interval coloring

The problem

Input:

- n intervals $I_1 = [s_1, f_1], \ldots, I_n = [s_n, f_n]$ start time, finish time
- also write $\boldsymbol{s_j} = \textbf{start}(\boldsymbol{I_j})$, $\boldsymbol{f_j} = \textbf{finish}(\boldsymbol{I_j})$

The problem

Input:

- n intervals $I_1 = [s_1, f_1], \ldots, I_n = [s_n, f_n]$ start time, finish time
- also write $\boldsymbol{s_j} = \textbf{start}(\boldsymbol{I_j})$, $\boldsymbol{f_j} = \textbf{finish}(\boldsymbol{I_j})$

Output:

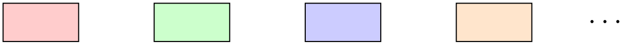
- assignment of **colors** to each interval
- overlapping intervals get **different colors**
- **minimize** the number of colors used overall

Remarks:

- another version: finding classrooms for lectures
- colors $\leftrightarrow$ numbers $1, 2, \ldots$
- $\textbf{finish}(\boldsymbol{I_j}) = \textbf{start}(\boldsymbol{I_k})$ not an overlap

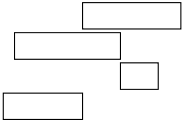
A few attempts

Available colors:



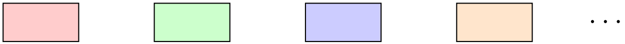
Attempt 1:

- sort intervals by **non-decreasing finish times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



A few attempts

Available colors:

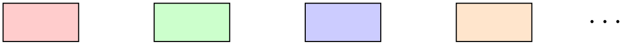


Attempt 1:

- sort intervals by **non-decreasing finish times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)

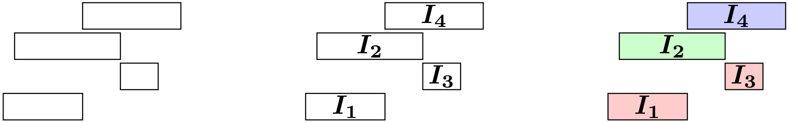
A few attempts

Available colors:



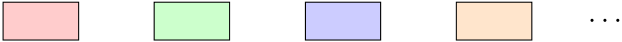
Attempt 1:

- sort intervals by **non-decreasing finish times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



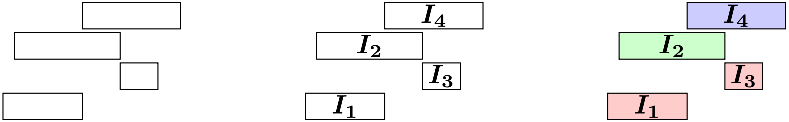
A few attempts

Available colors:



Attempt 1:

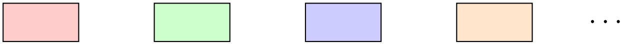
- sort intervals by **non-decreasing finish times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



- does not work

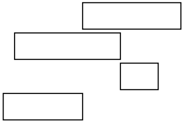
A few attempts

Available colors:



Attempt 2:

- sort intervals **from shortest to longest**
- for $j = 1, \dots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \dots, I_{j-1}$)



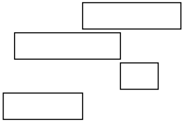
A few attempts

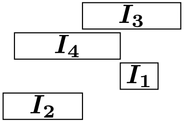
Available colors:

…

Attempt 2:

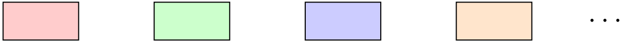
- sort intervals **from shortest to longest**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)





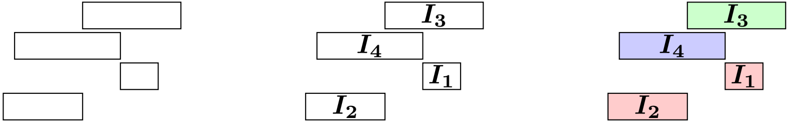
A few attempts

Available colors:



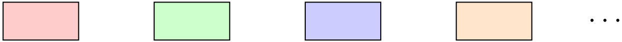
Attempt 2:

- sort intervals **from shortest to longest**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



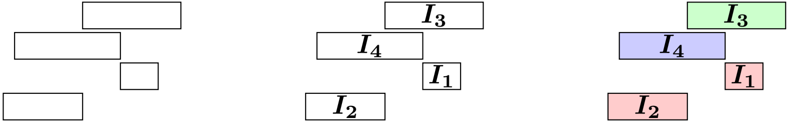
A few attempts

Available colors:



Attempt 2:

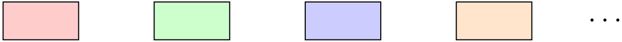
- sort intervals **from shortest to longest**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



- does not work

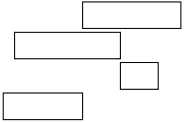
A few attempts

Available colors:



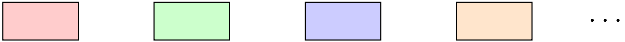
Attempt 3:

- sort intervals **by non-decreasing start times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



A few attempts

Available colors:

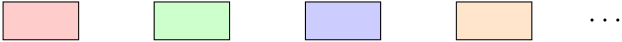


Attempt 3:

- sort intervals **by non-decreasing start times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)

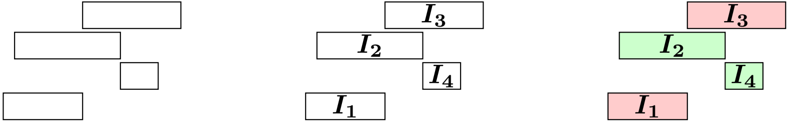
A few attempts

Available colors:



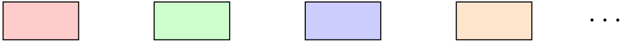
Attempt 3:

- sort intervals **by non-decreasing start times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



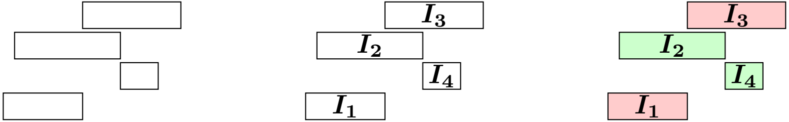
A few attempts

Available colors:



Attempt 3:

- sort intervals **by non-decreasing start times**
- for $j = 1, \ldots, n$, use the **first possible color** for I_j (no same-color overlap with $I_1, \ldots, I_{j-1}$)



- maybe, needs proof

Correctness of the third attempt

Claim

Suppose the output uses $\boldsymbol{k}$ colors. Then, **we cannot use fewer**.

Correctness of the third attempt

Claim

Suppose the output uses $\boldsymbol{k}$ colors. Then, **we cannot use fewer**.

Proof

- suppose we color I_t with color k
- so I_t overlaps with $k-1$ intervals, say $I_{\alpha_1}, \ldots, I_{\alpha_{k-1}}$ seen previously
- so for all $j = 1, \ldots, k-1$, $\boldsymbol{s_{\alpha_j} \leq s_t < f_{\alpha_j}}$
- so at time s_t, we can't do with less than k colors

Exercises

- $\boldsymbol{\Theta(n \log(n) + nk)}$ easy. Give a $\boldsymbol{\Theta(n \log(n))}$ algorithm
- write an exchange-based proof