COMPSCI 311: Introduction to Algorithms

Lecture 6: Greedy Algorithms

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon, Akshay Krishnamurthy, Andrew McGregor

11 February 2019

Greedy Algorithms

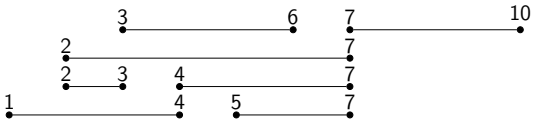
We are moving on to our study of algorithm design techniques:

- Greedy
- Divide-and-conquer
- Dynamic programming
- Network flow

Get a sense of "greedy" algorithms, then characterize them

Interval Scheduling

- In the 80s, you could only watch a given TV show at the time it was broadcast. What if you wanted to watch multiple shows and some of the broadcast times overlap?
- You want to watch the highest number of shows. Which subset of shows do you pick?



- Fine print: assume you like all shows equally, you only have one TV, and you need to watch shows in their entirety.

Formalizing Interval Scheduling

Let's formalize the problem

- Shows $1, 2, \ldots, n$
  (more generally: *requests* to be fulfilled with a given resource)
- s_j: start time of show j
- f_j, also written $f(j)$: finish time of show j
- Shows i and j are compatible if they don't overlap.
- Set A of shows is compatible if all pairs in A are compatible.
- Set A of shows is optimal
  if it is compatible and no other compatible set is larger.

Greedy Algorithms

- Main idea in greedy algorithms is to make one choice at a time in a "greedy" fashion.
  (Choose the thing that looks best, never look back. . .)
- We will sort shows in some "natural order" and choose shows one by one if they're compatible with the shows already chosen. Concretely:

$R \leftarrow$ set of all shows **sorted by some property**
$A \leftarrow \{\}$ ▷ selected shows
while R is not empty **do**
  take first show i from R
  add i to A
  delete i and all overlapping shows from R
end while

Clicker Question 1

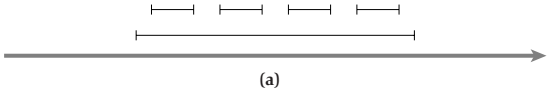
$R \leftarrow$ set of all shows **sorted by some property**
$A \leftarrow \{\}$ ▷ selected shows
while R is not empty **do**
  take first show i from R
  add i to A
  delete i and all overlapping shows from R
end while

Because the given algorithm includes sorting, we can deduce it is

A. $\Theta(n \log n)$
B. $O(n \log n)$
C. $\Omega(n \log n)$
D. None of the above

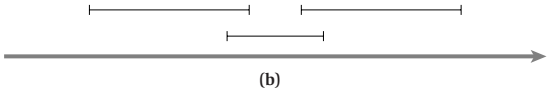
What's a "natural order" ?

- *Start Time*: Consider shows in ascending order of s_j.



(a)

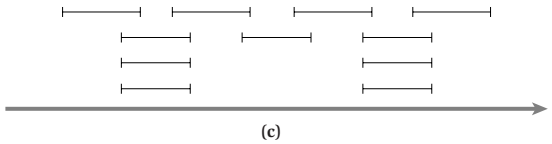
- *Shortest Time*: Consider shows in ascending order of $f_j - s_j$.



(b)

What's a "natural order" ?

- *Fewest Conflicts*: Let c_j be number of shows which overlap with show j. Consider shows in ascending order of c_j.



(c)

- *Finish Time*: Consider shows in ascending order of f_j.

We'll show that this works!

Analysis

Sorting shows by finish time gives an optimal solution in examples. Let's try to prove that it will always be optimal.

Let A be the set of shows returned by the algorithm when shows are sorted by finish time. What do we need to prove?

- A is compatible (obvious property of algorithm)
- A is optimal

We will prove A is optimal by a "greedy stays ahead" argument

Ordering by Finish Time is Optimal: "Greedy Stays Ahead"

- Let $A = i_1, \ldots, i_k$ be the intervals selected by the greedy algorithm
- Let $O = j_1, \ldots, j_m$ be the intervals of some optimal solution O
- Assume both are sorted by finish time

```
A: |--i1--||---i2---| ... |---ik---|
O: |---j1---||---j2---|     ...      |----jm----|
```

- Observation: $f(i_1) \leq f(j_1)$. The first show in A finishes no later than the first show in O.
- **Claim** ("greedy stays ahead"): $f(i_r) \leq f(j_r)$ for all $r = 1, 2, \ldots$. The rth show in A finishes no later than the rth show in O.

"Greedy Stays Ahead"

- **Claim**: $f(i_r) \leq f(j_r)$ for all $r = 1, 2, \ldots$
- **Proof** by induction on r
- **Base case** ($r = 1$): i_r is the first choice of the greedy algorithm, which has the earliest overall finish time, so $f(i_r) \leq f(j_r)$

Induction Step

- Assume inductively that $f(i_{r-1}) \leq f(j_{r-1})$ ($r \geq 2$)

- j_r is compatible with j_{r-1}, so $s(j_r) \geq f(j_{r-1})$
- $f(j_{r-1}) \geq f(i_{r-1})$ by inductive hypothesis
- Thus, $s(j_r) \geq f(i_{r-1})$ and interval j_r is in the set of available intervals when trying to select i_r
- Since we greedily select the earliest finish time, $f(i_r) \leq f(j_r)$, completing the inductive step

Clicker Question 2

```
A: |--i1--||---i2---| ... |---ik---|
O: |---j1---||---j2---|     ...      |----jm----|
```

Recall that k is the number of intervals in the greedy solution and m is the number of intervals in an optimal solution. What have we just proven?

A. $f(i_r) \leq f(j_r)$ for $r = 1, 2, \ldots, m$

B. $f(i_r) \leq f(j_r)$ for $r = 1, 2, \ldots, k$

C. The greedy algorithm is optimal.

D. None of the above.

Optimality

```
A: |--i1--||---i2---| ... |---ik---|
O: |---j1---||---j2---|     |---jk---| ... |----jm----|
```

Can it be the case that $k < m$?

No. Because "greedy stays ahead", intervals j_{k+1} through j_m would be compatible with the greedy solution, and the greedy algorithm would not terminate until adding them.

Running Time?

$R \leftarrow$ set of all shows **sorted by finishing time**
$A \leftarrow \{\}$
while R is not empty **do**
  take first show i from R
  add i to A
  delete i and all overlapping shows from R ▷ $O(n)$?
end while

Can we make loop better than n^2?

Running Time?

$R \leftarrow$ set of all shows **sorted by finishing time**
$A \leftarrow \{\}$, $end = 0$ ▷ last scheduled time
for show i from 1 to n **do**
  if $s_i \geq end$ **then**
    add i to A; $end = f_i$ ▷ $O(1)$
  end if
end for

$\Theta(n \log n)$ — dominated by sort

Algorithm Design—Greedy

Greedy: make a single "greedy" choice at a time, don't look back.

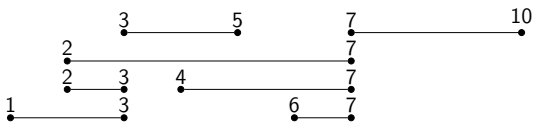
Learning goals:

	Greedy
Formulate problem	
Design algorithm	
Prove correctness	✓
Analyze running time	
Specific algorithms	Dijkstra, MST

Focus is on proof techniques. Next time: another proof technique.

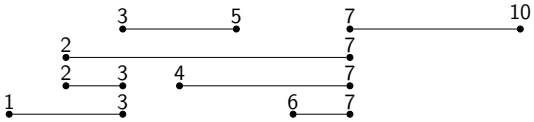
Problem 2: Interval Partitioning

- Suppose you are in charge of UMass classrooms.
- There are n classes to be scheduled on a Monday where class j starts at time s_j and finishes at time f_j
- Your goal is to schedule *all* the classes such that the minimum number of classrooms get used throughout the day.
  Obviously two classes that overlap can't use the same room.



How Many Classrooms Are Needed?

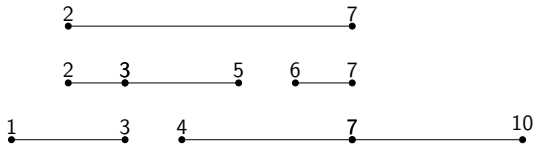
- Consider all points on the timeline
- Count how many classes run at that time
- Maximum number is called the *depth* of the set of intervals



- It's a lower bound on number of rooms needed (why?)
- Is this number sufficient ?

A Greedy Approach

- Process classes in order of start time
- For each class, either allocate a new room, or reuse an already allocated room if the last class in that room has completed



Interval Partitioning Algorithm

sort the intervals by starting time
for $j = 1$ to n **do**
  for each $i < j$ overlapping interval j **do**
    exclude label of I_i for scheduling I_j
  end for
  if there is some nonexcluded label in 1 .. d **then**
    label I_j with that label
  end if
end for

Clicker Question 3

If the class with the next starting time is compatible with several rooms, it should be scheduled

A. In a room with the fewest classes scheduled so far

B. In the room with the latest finishing time

C. In the room with the earliest finishing time

D. Does not matter

Correctness of Interval Partitioning

Claim: Every resource will be assigned a label.

Proof? By contradiction.

- Suppose it uses more than d rooms
- There are d classes running when the $d + 1^{\text{st}}$ room is allocated.
- This set of $d + 1$ classes overlap, therefore the depth is greater than d.

Claim: No two resources are assigned the same label.

Proof? Assume two intervals overlap, I_1 starting before I_2
Label of I_1 is excluded when scheduling I_2

Corollary: the greedy algorithm uses exactly d rooms, and is therefore optimal.

Complexity of Interval Partitioning

Keep finishing time for each label

sort the intervals by starting time
let $end[\ell] = 0$ for all ℓ in 1 .. d
for $j = 1$ to n **do**
  for $\ell = 1$ to d **do**
    if $s_i \geq end[\ell]$ **then**
      assign I_j to label ℓ; break;
    end if
  end for
  if $\ell > d$ **then return** "unschedulable"
  end if
end for

Complexity? $O(n \log n + nd)$. Can we do better ?

- keep a priority queue of last finishing times for each label
- find $\min$ in $O(1)$, update in $O(\log d)$, loop becomes $O(n \log d)$

Since $d \leq n$, we have $O(n \log n + n \log d) = O(n \log n)$