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, your only opportunity to watch a specific TV show was the time it was broadcast. Unfortunately, on a given night there might be multiple shows that you want to watch and some of the broadcast times overlap.

Example: \([1, 4], [2, 3], [2, 7], [4, 7], [3, 6], [6, 10], [5, 7]\)

▶ 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, …, \(n\)
  (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 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:

\[
R \leftarrow \text{set of all shows sorted by some property} \\
A \leftarrow \{\} \quad \triangleright \text{selected shows} \\
\text{while } R \text{ is not empty do} \\
\quad \text{Take first show } i \text{ from } R \\
\quad \text{Add } i \text{ to } A \\
\quad \text{Delete } i \text{ and all overlapping shows from } R \\
\text{end while}
\]

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

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

Clicker Question 1
What’s a “natural order”?

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

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

(a)

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

(b)

(c)

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--- |

- Could it be the case that $m > k$?
- 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 $r$th show in $A$ finishes no later than the $r$th 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 $f(i_{r-1}) \leq f(j_{r-1})$
- Assume for sake of contradiction that $f(i_r) > f(j_r)$

| A: | ---i1--- | ... | ---i(r-1)--- | ------ir------ |
| O: | ---j1--- | ... | ---j(r-1)--- | ---jr------ |

- But it must be the case that $j_r$ is compatible with the first $r - 1$ shows in $A$, because (using induction hypothesis)

$s(j_r) \geq f(j_{r-1}) \geq f(i_{r-1})$

- Therefore, the greedy algorithm could have selected $j_r$ instead of $i_r$.

But $j_r$ finishes sooner than $i_r$, which contradicts the algorithm.
- Therefore, it must be the case that $f(i_r) \leq f(j_r)$
Running Time?

\( R \leftarrow \) set of all shows sorted by finishing time
\( A \leftarrow \{ \} \) \> 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

Can we make loop better than \( n^2 \)? Check conflict when selecting \( i \)

\( R \leftarrow \) set of all shows sorted by finishing time
\( A \leftarrow \{ \}, \text{end} = 0 \) \> last scheduled time

for show \( i \) from 1 to \( n \) do
  if \( s_i \geq \text{end} \) then
    Add \( i \) to \( A \); \( \text{end} = f_i \) \> \( O(1) \) iteration
  end if
end for

\( \Theta(n \log n) \) — dominated by sort
Change in abstract version of algorithm makes it more efficient!

Algorithm Design—Greedy

Greedy: make a single “greedy” choice at a time, don’t look back.

<table>
<thead>
<tr>
<th>Formulate problem</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design algorithm</td>
<td>easy</td>
</tr>
<tr>
<td>Prove correctness</td>
<td>hard</td>
</tr>
<tr>
<td>Analyze running time</td>
<td>easy</td>
</tr>
</tbody>
</table>

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?

Possible Greedy Approaches

- Suppose the available classrooms are numbered 1, 2, 3, . . .
- We could run a greedy algorithm . . . consider the lectures in
  some natural order, and assign the lecture to the classroom
  with the smallest numbered that is available.
- What’s a “natural order”?
  - Start Time: Consider classes in ascending order of \( s_j \).
  - Finish Time: Consider classes in ascending order of \( f_j \).
  - Shortest Time: Consider classes in ascending order of \( f_j - s_j \).
  - Fewest Conflicts: Let \( c_j \) be number of classes which overlap
    with show \( j \). Consider shows in ascending order of \( c_j \).

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 2

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

- A) In the room with the earliest finishing time
- B) In the room with the latest finishing time
- C) In a room where nothing was scheduled so far
- D) Does not matter

Correctness of Interval Partitioning

Claim: Every resource will be assigned a label.
Proof? By contradiction, otherwise would have higher depth.

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 $I_2$

Complexity of Interval Partitioning

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 some label in 1..$d$ is not excluded then
label $I_j$ with that label
end if
end for

Naive: $O(n \log n + n^2)$
Better: $O(n \log n + nd)$ (keep finishing time for each label)
Improved:
- keep a priority queue of last finishing times for each label
- find min in $O(1)$, update in $O(\log d)$
- outer loop becomes $O(n \log d)$

Clicker Question 3

An $O(n \log n + n \log d)$ implementation for interval partitioning is also

- A) $O(n \log n)$
- B) $O(n \log(nd))$
- C) Both A and B
- D) None of A or B