7.2 Event Scheduling

> Sections (like this one) which marked with a vertical line on the left side are commentary — discussions about the algorithm development process that wouldn't be part of a writeup.

Establish the problem.

- *Specifications.* Given a collection of events with start time $s(i)$ and finish time $f(i)$ $(0 \leq s(i) \leq f(i))$, find the largest set of non-overlapping events.

  Input: n events with start and finish times $s(i)$ and $f(i)$

  Legal input: $0 \leq s(i) \leq f(i)$

  Output: a set of events

  Legal solution: a non-overlapping set of events

  Optimization goal: largest set

- *Examples.*

Identify avenues of attack.

- *Known targets.*
- *Approach.* Series of choices.

  > (This was dictated.)

- *Paradigms and patterns.*

  Paradigm: greedy.

  > (Again, this was dictated.)

  > Identify the flavor, if applicable. Consider how this problem fits into each applicable pattern.

  This is a find a subset problem.

  Produce output: the next event in the solution

  Process input: for each event, do we pick it?

- *The series of choices.*

  Produce output: the next event in the solution

  Process input: whether to pick the current event

  > This is a subset problem and it doesn't matter what order the selected events are reported in. There's also not a natural order — there are several ways one could order a list of events, including randomly. This means that both ordering and legality are likely to factor into the greedy choice. Produce output bundles this all into the "next event" choice, while process input splits things up: the ordering choice is reflected in how the events are sorted before we started going through them, and then — since this is a maximize-the-subset problem — we will pick the event if it is legal to do so (no overlaps).
  >
  > So, thinking in terms of process input lets us focus on the ordering of events as the greedy choice.

  The series of choices: which event to consider next

- *Greedy choices and counterexamples.*

> To identify the possible greedy choices, identify what information there is to work with.
>
> We have start and finish times for each event, and from that can compute how long the event is.

The next event picked will always be one that doesn't overlap with already-chosen ones (needed for a legal solution). Plausible options for picking that event: any remaining event (simple!), the shortest remaining event (leaves more time for other events), the earliest-finishing event (leaves more time after for other events), the latest-starting event (leaves more time before for other events).

> Picking the longest event makes no sense with a goal of picking as many events as possible. Maybe the earliest starting event would finish earlier so there would be more time for other events, but that's more directly captured by the earliest-finishing event. A similar argument can be made for the latest-finishing event.
>
> Now look for counterexamples to eliminate the greedy choices that definitely don't work. A minimal counterexample for this problem involves three events with one overlapping the other two. Choose start and finish times so that the greedy choice picks the one when the correct answer would have been to pick the two.

Pick any non-overlapping event next. Event A runs 2-4pm, event B runs 5-7pm, event C runs 3-6pm. Any of the three could be picked; if C is picked, everything else overlaps but the optimal solution is to have picked A and B.

Pick the shortest non-overlapping event next. Event A runs 12-4pm, event B runs 5-9pm, event C runs 3-6pm. The algorithms picks C first but then everything else overlaps. The optimal solution is to have picked A and B.

Pick the earliest-finishing non-overlapping event next. Let C be the earliest-finishing event, which should be the wrong choice for a counterexample — A and B must then overlap with C so they can't also be picked but can't overlap with each other because in order for C (one event) to be the wrong choice, the correct answer must have more than one event. Furthermore, A and B must end after C does (C is the earliest-finishing) so they have to start before C finishes in order to overlap. Since they both span C's finish time, they must overlap each other. No counterexample!

Pick the latest-starting non-overlapping event next. This is effectively the same idea as earliest-finishing, just selecting events starting from the opposite end of the schedule.

Define the algorithm.

- *Main steps.*

  > For subset problems where there is not a natural order (and thus ordering is part — or most — of the greedy choice), a process input approach is convenient.

  ```
  for each event (in order of increasing finishing time)
    choose the event if it doesn't overlap any already-selected events
  ```

- *Exit condition.*

  When all of the events have been considered.

- *Setup.*

  Nothing to do.

  > (Sorting the events by increasing finish time so it is efficient to go through them in that order will be an implementation detail.)

- *Wrapup.*

  The set of events selected is the answer; nothing else to do here.

- *Special cases.*

> Duplicates or ties (where two or more things satisfy the greedy choice) can cause problems.

Multiple events with the same finishing time will get sorted into some order, so duplicates don't creating any problems in carrying out the algorithm.

- *Algorithm.*

  Given a collection of events with start time $s(i)$ and finish time $f(i)$ $(0 \leq s(i) \leq f(i))$, find the largest set of non-overlapping events.

  Input: n events with start and finish times $s(i)$ and $f(i)$

  Legal input: $0 \leq s(i) \leq f(i)$

  Output: a set of events

```
for each event (in order of increasing finishing time)
  choose the event if it doesn't overlap any already-selected events
```

Show termination and correctness.

- *Termination.*
  - *Measure of progress.*
    The number of events considered.
  - *Making progress.*
    Each iteration makes a decision (take or not take) about a previously not-yet-decided-about event.
  - *The end is reached.*
    Since events are not considered twice, all will eventually have been considered.
- *Correctness.*
  - *Loop invariant.*

    > The general pattern is
    >
    > `After` k `iterations, ...`
    >
    > The loop invariant needs to address both legality and optimality. For optimality, we try a staying ahead argument rather than a direct assertion of having the best solution so far.
    >
    > A staying ahead argument requires an apples-to-apples comparison between the algorithm's solution-so-far and a portion of the optimal solution. First is to observe that since there's not a required or natural ordering for the elements in the solution, we'll assume the optimal's set of events to be listed in the same order that the algorithm would have picked them (had the algorithm picked those events). Then, for process input, "after k iterations" means after k input elements have been processed — but with the optimal we have only the solution to work with. Instead, consider how many output elements have been produced by the algorithm after those k iterations and compare the algorithm's solution so far to the same number of elements in the optimal's solution.
    >
    > The optimization criterion is the number of events picked, but we're comparing the algorithm's solution-so-far to the first part of the optimal solution based on the number of events picked so far, so that doesn't work for staying ahead. What about the measure of progress? That's based on events considered so far, so that doesn't help. What else do we have to work with? Events are being considered in order of finishing time, so we could compare that — the earlier the k' events picked so far finish, the better, because an earlier finishing time means more time remaining to hopefully schedule some additional events.

    After the first k events have been considered and $k' \leq k$ have been picked,

    * none of the events that have been picked overlap, and

  - the finish time of the algorithm's k'th event is no later than the finish time of the optimal's k'th event.

- *Establish the loop invariant.*

  > Show for $k = 0$ to establish that the setup is fine, and for $k = 1$ (after the first iteration) to establish that the greedy choice starts off OK.

  $k = 0$: Before any events have been considered, the algorithm has picked none and we compare that to the first none events in the optimal solution. There are no finish times to compare so the algorithm certainly can't be behind yet.

  $k = 1$: The algorithm will pick an event on the first iteration because there's nothing already picked that it could overlap with, so $k' = 1$.

  After the first event has been considered and picked,

  - none of the events that have been picked overlap because only one event has been picked, and
  - the finish time of the algorithm's first event is no later than the finish time of the optimal's first event because the algorithm picked the earliest-finishing event. There's nothing earlier for the optimal to have picked.

- *Maintain the loop invariant.*

  Assume that after the first k events have been considered and $k' \leq k$ have been picked,

  - none of the events that have been picked overlap, and
  - the finish time of the algorithm's k'th event is no later than the finish time of the optimal's k'th event.

  Show that after the first $k + 1$ events have been considered and $k' \leq k + 1$ have been picked,

  - none of the events that have been picked overlap, and
  - the finish time of the algorithm's k'th event is no later than the finish time of the optimal's k'th event.

  Legality. The algorithm only picks events that don't overlap with already-picked events, so whether or not an event was picked in this iteration, there are no new overlaps (and there weren't any before this iteration).

  Optimality. Consider the two cases: either the algorithm picked an event in this iteration or it didn't.

  - *No event picked.* With no event picked, k' doesn't change and thus the finish time of the algorithm's k'th event is still no later than the finish time of the optimal's k'th event.
  - *An event picked.* Assume this is the step where things go wrong: $f(A_{k'}) \leq f(O_{k'})$ but $f(A_{k'+1}) > f(O_{k'+1})$ where A_i and O_i refer to the algorithm's and the optimal's ith events, respectively.
    First, note that events $A_{k'+1}$ and $O_{k'+1}$ must be different events since they have different finish times.
    Since $A_{k'+1}$ doesn't overlap with events $A_1, ..., A_k$ and the algorithm considers events in order of finish time, event $O_{k'+1}$ must overlap with something the algorithm already picked or else it would have been chosen instead of $A_{k'+1}$ since it ends earlier than $A_{k'+1}$. Overlapping with something $A_1, ..., A'_k$ means $s(O_{k'+1}) < f(A'_k)$ — event $O_{k'+1}$ must start before event A'_k (the latest-finishing event previously selected by the algorithm) finishes.
    But no events in an optimal solution can overlap, and since the optimal's events are being considered in order of finishing time, $s(O_{k'+1}) \geq f(O'_k)$. The loop invariant gives us that $f(A'_k) \leq f(O'_k)$ so $s(O_{k'+1}) \geq f(A'_k)$ — which contradicts the requirement that $s(O_{k'+1}) < f(A'_k)$ from the previous paragraph.
    Thus the assumption that the algorithm went wrong in this step is at fault, and $f(A_{k'+1}) \leq f(O_{k'+1})$.

- *Final answer.*

> Show that the setup + main steps + wrapup yields the final answer. Since there aren't any setup or wrapup steps, show that the loop invariant plus the exit condition results in the final answer.

Legality. The loop invariant gives us that after k events have been considered, none of the picked events overlap. The exit condition is that all n events have been considered. So, none of the picked events overlap.

Optimality. The loop invariant gives us that after k events have been considered and $k' \leq k$ have been picked, the finish time of the algorithm's k'th event is no later than the finish time of the optimal's k'th event. But what we need is that k' is the largest number of events. Let $|A|$ and $|O|$ denote the number of events in (size of) the algorithm's and optimal solutions, respectively. There are three possibilities:

* $|A| < |O|$
* $|A| = |O|$
* $|A| > |O|$

> $|A| = |O|$ is what we want — the algorithm found a solution with the same number of events as the optimal solution. Show this by showing that the other two cases are impossible.

$|A| > |O|$ is impossible because the optimal solution by definition has the most events. The algorithm can't have found a legal solution with more events.

If $|A| < |O|$, there is at least one more event in the optimal solution than in the algorithm's solution. Since the events in the optimal solution are being considered in order of finish time, $f(O_{|A|+1}) \geq f(O_{|A|})$. The optimal solution does not contain overlapping events, so this means $s(O_{|A|+1}) \geq f(O_{|A|})$ — and, because $f(A_{|A|}) \leq f(O_{|A|})$ by the loop invariant, $s(O_{|A|+1}) \geq f(A_{|A|})$. But if the extra event in the optimal solution starts after the algorithm's last event finishes, it can't overlap with any other events that the algorithm picked — so it can't exist or else the algorithm would have picked it. Thus this case is also impossible.

Since $|A| > |O|$ and $|A| < |O|$ are both impossible, it must be the case that $|A| = |O|$ and thus the algorithm produces an optimal solution.

Determine efficiency.

- *Implementation.*

  Sort the events in order of increasing finish time. Then it is $O(1)$ to find the next event to consider.

  The algorithm also needs to check whether event k overlaps with any already-picked events. Since the algorithm picks events in order of increasing finish times, the last-selected event will have the latest finish time of any already selected. Thus we only need to check that event k's starting time is no earlier than the last-selected event's finish time to know that there aren't any overlaps with any already-selected events. This is $O(1)$.

  Picked events must be added to an unordered collection. This is $O(1)$.

  > $O(1)$ to add to the end of an array, the beginning of a linked list, or to a hashtable set.

- *Time and space.*

  The total running time is $O(n \log n)$ — $O(n \log n)$ to sort n events by finish time, then n iterations with an $O(1)$ overlap check. Adding an event to the collection of selected events is also $O(1)$.

- *Room for improvement.*

  > Seems unlikely to beat $O(n)$ once the events are sorted.