Greedy 1: Activity Selection
(Picking a Maximum Number of Disjoint Intervals)

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
In this lecture, we will commence our discussion of the **greedy** technique. In fact, this technique has a very simple rationale: simply make the **locally optimal** decision at each step. It is important to note that this technique does **not** always give a **globally optimal** solution. There are, however, problems where it does. The nontrivial part of applying the technique is to prove (or disprove) the global optimality.
Activity Selection

Problem definition

Input: A set $S$ of $n$ intervals of the form $[s, f)$ where $s$ and $f$ are integer values.
Output: A subset $T$ of disjoint intervals in $S$ with the largest size $|T|$.

Remark: You can think of $[s, f)$ as the duration of an activity, and consider the problem as picking the largest number of activities that do not have time conflicts.
**Example:** Suppose

\[ S = \{[1, 10), [3, 8), [6, 21), [12, 20), [15, 18), [18, 23), [21, 25)\} \].

An optimal solution is \( T = \{[3, 8), [15, 18), [18, 23)\} \). Optimal solutions may not be unique; here is another one: \( T' = \{[1, 10), [12, 20), [21, 25)\} \).
Complication: Once an interval is taken, those overlapping with it will have to be discarded. So one mistake may lead to a suboptimal solution.

It turns out that the following greedy strategy works: simply take the interval with the earliest finish time at each step.

Algorithm
Repeat the following steps until $S$ becomes empty:

- Add to $T$ the interval $I \in S$ with the smallest finish time.
- Remove from $S$ all the intervals intersecting $I$ (including $I$ itself)
**Example:** Suppose $S = \{(1, 10), (3, 8), (6, 21), (12, 20), (15, 18), (18, 23), (21, 25)\}$.

Sort the intervals in $S$ by finish time: $S = \{(3, 8), (1, 10), (15, 18), (12, 20), (6, 21), (18, 23), (21, 25)\}$.

We first add $[3, 8)$ to $T$, after which intervals $[3, 8), [1, 10)$ and $[6, 21)$ are removed. Now $S$ becomes $S = \{(15, 18), (12, 20), (18, 23), (21, 25)\}$. The next interval added to $T$ is $[15, 18)$, which shrinks $S$ further to $S = \{(18, 23), (21, 25)\}$. After $[18, 23)$ is added to $T$, $S$ becomes empty and the algorithm terminates.
Now comes the nontrivial part: prove the algorithm is correct, namely, it indeed returns an optimal solution. We will do so by mathematical induction.

**Base Step:** \( n = 1 \).
That is, \( S \) has only one interval, in which case the output of the algorithm is obviously optimal.

**Inductive Step:** Assuming that the algorithm is correct for all \( n \leq k \).
We will prove that it is also correct for \( n = k + 1 \).
Claim: Let $I = [s, f)$ be the interval in $S$ with the smallest finish time. There must be an optimal solution that contains $I$.

Proof: Let $T^*$ be an arbitrary optimal solution that does not contain $I$. We will turn $T^*$ into another optimal solution $T$ that contains $I$, and thereby finish the proof.

Let $I' = [s', f')$ be the interval in $T^*$ with the smallest finish time. We construct $T$ as follows: add all the intervals in $T^*$ to $T$ except $I'$, and finally add $I$ to $T$.

We will prove that all the intervals in $T$ are disjoint. This indicates that $T$ is also an optimal solution, and hence, will complete the proof.
Activity Selection

It suffices to prove that $\mathcal{I}$ cannot intersect with any other interval $\mathcal{J} \in T$.

Suppose on the contrary that there is such a $\mathcal{J} = [a, b)$. By definition of $\mathcal{I}'$, we must have $f' \leq b$. Combining this and the fact that $\mathcal{J}$ is disjoint with $\mathcal{I}'$, we assert that $f' \leq a$. On the other hand, by definition of $\mathcal{I}$, it must hold that $f \leq f'$. It thus follows that $f \leq a$. But this indicates that $\mathcal{I}$ and $\mathcal{J}$ are disjoint, giving a contradiction.
Think 1: Now that we know \( I \) must be in an optimal solution, how do we proceed with the induction proof that the algorithm is correct for \( n = k + 1 \)? This will be left as a regular exercise (solution provided in full).

Think 2: How to implement the algorithm in \( O(n \log n) \) time? This will be left as another regular exercise (again, solution provided in full).