# CSE202
# 2 – Greedy Algorithms

TA: Joseph L.

## I. Introduction

- In problems that require minimization or maximization of some quantity, sometimes the most obvious approach produces optimal results.
    - E.g., in the interval scheduling problem, to be discussed in more detail in the next section, you want to schedule as many (out of $n$) time intervals as possible, with no overlaps. An optimal greedy algorithm simply selects, iteratively, whichever remaining meeting finishes earliest.
- These algorithms are often easy to come up with, and trickier to prove (compared to divide-and-conquer).
- To optimize runtime, carefully consider the following:
    - What precomputations (e.g., sorting) are needed?
    - What data structures can be used to efficiently do a particular task?
- In this chapter, we will cover greedy algorithms that *happen* to always give optimal solutions. Later on, we will use greedy algorithms to approximate solutions to problems that are otherwise computationally intractable.
- Proof by the **Exchange Argument**
    - This is the proof method that we will primarily use to prove greedy algorithms optimal.
    - First, picture a theoretical optimal solution *OPT* and a theoretical greedy solution *G*, produced by our algorithm.
    - In general, we will *iteratively* identify parts of the solution in which OPT and *G* disagree, and then argue that changing that part of *OPT* to agree with *G* will still produce another optimal solution *OPT'*. Further specifics will vary depending on the problem, and we encourage you to see some examples.
    - Important: in general, we do NOT attempt to prove that *OPT=G*, as both are theoretical solutions. The best that we can do is prove that *G* is necessarily *one of potentially many* optimal solutions, unless the nature of the problem dictates that there can be only one solution.

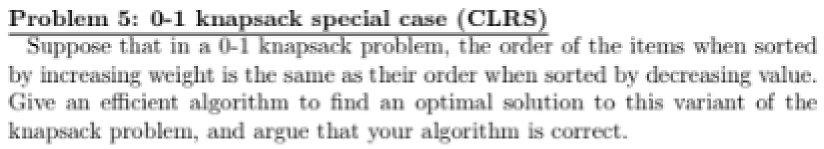
## II. Interval Scheduling

- *Description*: As alluded to earlier, we have $n$ potential intervals, represented by start and finish time - $m_i = (s_i, f_i)$ for $i \in \{1,2,\dots,n\}$. We would like to select as many of these intervals as possible, with no overlaps.
- *Algorithm*
    - Initialize an empty list *G* (to be our solution) and "latest finish time" $f = 0$.
    - Repeat for $j \in \{1,2,\dots,n\}$:
        - Consider the meeting $m_j$, which has the $j$th earliest finish time.
        - If $s_j \geq f$ (i.e., the meeting starts after our last selected meeting ends): append $m_j$ to *G* and set $f \leftarrow f_j$.
        - Else: ignore $m_j$ and move on, as it overlaps with at least one of our already selected intervals.
- *Complexity*:
    - Using a min-heap, we can retrieve the interval with the next earliest end time in $O(logn)$. If we do this $n$ times, our total is $\boldsymbol{O(nlogn)}$.
    - Alternatively, sort the list first in $O(nlogn)$. The rets of the algorithm now takes $O(n)$, as we can now retrieve the interval with the $j$th earliest finish time in $O(1)$. Our total is still $O(nlogn)$.
- *Correctness, by the Exchange Argument*

- We will use the Exchange Argument alluded to before, along with induction.
- Picture a theoretical optimal solution *OPT* of size $k$ and a theoretical greedy solution *G* of size $m$, both sorted by finish time $f_j$ for ease of comparison.
  - *OPT*: $[o_1, o_2, \dots, o_k]$, where $o_j = (s_{j,o}, f_{j,o})$.
  - *G*: $[g_1, g_2, \dots, g_m]$, where $g_j = (s_{j,g}, f_{j,g})$
- We will, for each $g_i \in G$, use induction to prove that either $o_1 = g_1$ or that $g_1$ can replace $o_1$ without causing overlaps. Lastly, we will prove that $k = m$ using a proof by contradiction.
- *Base Case*: We will start by considering the first two intervals $o_1$ and $g_1$ of each solution.
  - Consider the following two cases.
    - $o_1 = g_1$. In this case, *G* and *OPT* agree thus far, so we can proceed with the remaining $k-1$ elements in *G*.
    - $o_1 \neq g_1$. In this case, we claim that we can replace $o_1$ with $g_1$ in *OPT*, to get the solution *OPT'*, which does not contain overlaps and is still optimal (as it has the same number of intervals).
      - Proof: Suppose that, if we replace $o_1$ with $g_1$, the resulting solution *OPT'* contains overlaps. As we are currently considering the element that ends earliest in both solutions, we know that a potential overlap must happen with the second element $o_2$. However, due to our greedy algorithm's selection criteria, $f_{1,g} \leq f_{1,o}$ ($g_1$ ends earlier than $o_1$) - thus, if this replacement causes an overlap, then $o_1$ has to have caused an overlap as well. This contradicts our knowledge that *OPT* is a solution at all, let alone an optimal one. Therefore, we have proven by contradiction that *OPT'* is a valid, still-optimal solution.
  - We have now proven the existence of an optimal solution that includes $g_1$.
- *Induction Hypothesis*: Suppose there exists an optimal solution *OPT''* that agrees with *G* on the first $n \geq 1$ elements.
- *Induction Step*: We would like to prove that if the hypothesis is satisfied, then there exists an optimal solution *OPT** that agrees with *G* on the first $n+1$ intervals.
  - Consider the steps in our base case. If the first $n$ elements match, then we essentially have a subproblem in which we need consider only the intervals that begin after $f_{n,g}$. Thus, we can make very similar arguments for the following cases:
    - $o_{n+1} = g_{n+1}$: *OPT** clearly exists.
    - $o_{n+1} \neq g_{n+1}$: Because *OPT''* is a valid solution, we know that it has no overlaps. Within *OPT''*, we replace $o_{n+1}$ with $g_{n+1}$ to get *OPT**; suppose this causes an overlap. Then, the conflict must be with $o_{n+2}$. However, $g_{n+1}$ ends earlier than $o_{n+1}$ and therefore, if this is the case, then $o_{n+1}$ must have caused an overlap too. This contradicts our knowledge that *OPT''* is a valid solution, and therefore our supposition is false.
  - We have now proven the existence of an optimal solution that agrees with *G* on the first $n+1$ intervals.
- *Conclusion*: Our base case, hypothesis, and step are valid. Therefore, by induction, we have proven the existence of an optimal solution, of $k$ intervals, that contains all $m$ intervals of a greedy solution *G*.
- Now, we need only prove that $k = m$.
  - Proof: We have our optimal solution *OPT* that contains all $m$ intervals that are in *OPT*. Suppose that despite this, $m \neq k$. As *OPT* is optimal, it must then have more intervals: $k > m$. Because our greedy algorithm repeatedly adds the remaining interval with the earliest finish time, any interval $i$ that is contained only in *OPT* must begin after $f_{m,g}$. However, if so, then it can be

added without causing an overlap to *G*, so our algorithm would have considered and added it. Therefore, our supposition is false, and $k = m$.
   - We have now proven the existence of an optimal solution that is equal to any solution produced by our greedy algorithm, so we are done.
- Notes
   - K&T proves this problem in a different way. There is some overlap (pun), but it should be enlightening to read about their different way of proving the same key properties. We instead used the exchange method here because it is generally what you will use in this class for greedy algorithms.
   - We did a full, formal induction here, but in general a proof by the exchange method, for a problem like this one, is sufficient if it successfully explores the cases $o_1 = g_1$ and $o_1 \neq g_1$, and also proves that *OPT* and *G* have the same size.

**III. Sample Problem**
We will solve the following problem from the course website.



**Problem 5: 0-1 knapsack special case (CLRS)**
Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

*Background*: In the 0-1 knapsack problem, there exist $n$ treasures of (potentially) disparate values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, as well as a bag of capacity $W$. The goal is to select some subset of them to fill the bag with as much value as possible without exceeding the capacity. For more information: https://en.wikipedia.org/wiki/Knapsack_problem

*Overview*: The 0-1 knapsack problem is ordinarily NP-Complete. However, like many NP-Complete problems, it can have *special cases* (i.e., some key aspect of the problem is restricted to greatly reduce entropy) that are efficiently solvable. They key phrase here is "the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value." Before moving on, please think carefully about what this entails.

*Input*: A list of $n$ items
*Output*: The subset of items with the most value, subject to the weight capacity $W$

*Observations*: Intuitively, if one were to write a greedy algorithm for a problem like this, one might first consider repeatedly choosing items based on the $\frac{value}{weight}$ ratios. The problem with this would be whether the chosen item fits in the remaining capacity of the knapsack; as it turns out, no greedy algorithm can address this issue for the generic 0-1 knapsack, and that's why the problem is not solvable with a greedy algorithm (or for that matter, with any algorithm with sub-exponential complexity). However, on analyzing our earlier "key phrase", we reach the following conclusions about this variant of the problem:

- The lighter items are also the more valuable ones.
- The lighter items therefore have the higher $\frac{value}{weight}$ ratios. By sorting by increasing weight, we're also sorting by decreasing $\frac{value}{weight}$ ratio.
- Therefore, either the item with the highest $\frac{value}{weight}$ ratio fits, or *no* remaining item fits.

We will now formally describe our algorithm.

*Algorithm*: Sort items by increasing weight. Iteratively choose the lightest item if it fits or, otherwise, terminate and return what we've chosen.

*Complexity*: The initial sort is $O(nlogn)$. Each iteration afterwards is $O(1)$, as we simply need to check whether the total sum exceeds $W$ - we do that $n$ times for $O(n)$. The total complexity is therefore bottlenecked by the sort at $\boldsymbol{O(nlogn)}$.

*Correctness*: Use the exchange argument. Imagine an optimal solution *OPT* and our greedy solution *G*; for ease of comparison, sort items in both by increasing weight. For the purpose of comparison, we will consider items to be *equal* if they have the same weight and value (so duplicate items are equal to each other). Now, for each item $o_i$ and $g_i$, we account for the following two cases:

- $o_i = g_i$. The solutions agree, so we can move on.
- $o_i \neq g_i$. We argue that this is not possible.
    - Suppose there exists an optimal solution *OPT* in which $o_i \neq g_i$. Then, because the greedy algorithm prioritizes items with higher value per weight, that means that over the first $i$ items, the optimal solution has a lower average value per weight. The only way the optimal solution could "catch up" would be through containing more items. However, the less valuable items are also the heavier ones, so it is impossible for *OPT* (or any other solution) to contain more items than *G*. Therefore, this case can only lead to *OPT* being worse than *G*; however, that would contradict our knowledge that *OPT* is optimal. Therefore, by proof by contradiction, there is no optimal solution in which $o_i \neq g_i$.

Therefore, we have proven that any optimal solution is necessarily *equal* to the greedy solution and, by extension, that this problem only has one valid solution for a given set of parameters.

*Note*: Once again, we did not set out specifically to prove that *OPT=G*. They just happened to be always equal in this particular problem.

**References**

1. Kleinberg, Jon, and Éva Tardos. *Algorithm Design*. Boston: Pearson/Addison-Wesley, 2006. Print.