CS 603: Algorithms and Greedy Algorithms

Ellen Veomett

University of San Francisco

Outline

1. Guidelines for this Course
2. Algorithms
3. Greedy Algorithms

Everything is on Canvas!

- Syllabus
- Schedule (in Syllabus)
- Assignments
- Videos
- Lecture Slides
- Teaching staff, office hours
- Piazza
- Grades

Outline

Why study algorithms?

- Algorithms are how we solve problems!
- Ultimately, all written code is implementing an algorithm.
- We will learn general categories of algorithm types, and examples of their usage.
- We will learn how to show that an algorithm *works*. That it achieves what it was meant to achieve.
- We will practice showing the time complexity of our algorithms.

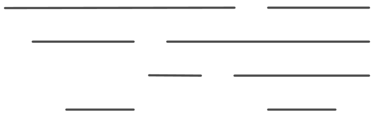
Outline

What makes a greedy algorithm

- In order to find a globally optimal solution, we take some small step that is locally optimal in some way.
- Greedy algorithms are frequently fairly easy to describe.
- In constructing a greedy algorithm, choosing a small locally optimal step *that results in a globally optimal solution* may not be obvious. (It sometimes may not even exist!)
- It is *imperative* that we prove that a greedy algorithm results in a globally optimal solution. Two typical methods (which frequently use induction):
    - Greedy algorithm stays ahead of any other optimal solution.
    - Any optimal solution can be *exchanged* step-by-step into a greedy solution.

Example: Interval Scheduling

- List of n tasks to be completed
- Each task has start time $s(i)$ and finish time $f(i)$, $i = 1, 2, \ldots, n$.
- Goal: pick k tasks with *nonoverlapping* times such that k (number of completed tasks) is as large as possible.



Example: Interval Scheduling

- List of n tasks to be completed
- Each task has start time $s(i)$ and finish time $f(i)$, $i = 1, 2, \dots, n$.
- Goal: pick k tasks with *nonoverlapping* times such that k (number of completed tasks) is as large as possible.

How *not* to solve Interval Scheduling

What should we use as our small *locally optimal step?*
The following reasonable guesses do *NOT* work:

How *not* to solve Interval Scheduling

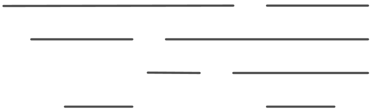
What should we use as our small *locally optimal step?*
The following reasonable guesses do *NOT* work:

- Pick the task that starts the earliest.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.

How *not* to solve Interval Scheduling

What should we use as our small *locally optimal step?*
The following reasonable guesses do *NOT* work:

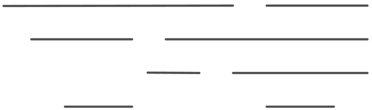
- Pick the task that starts the earliest.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.



How *not* to solve Interval Scheduling

What should we use as our small *locally optimal step?*
The following reasonable guesses do *NOT* work:

- Pick the task that starts the earliest.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.

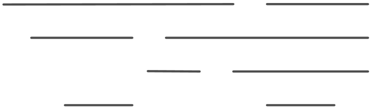


- Pick the task with the fewest conflicts.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.

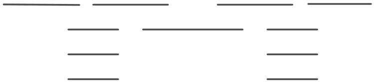
How *not* to solve Interval Scheduling

What should we use as our small *locally optimal step?*
The following reasonable guesses do *NOT* work:

- Pick the task that starts the earliest.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.



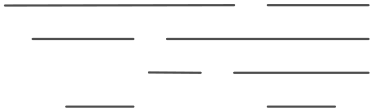
- Pick the task with the fewest conflicts.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.



Interval Scheduling: Greedy solution that works

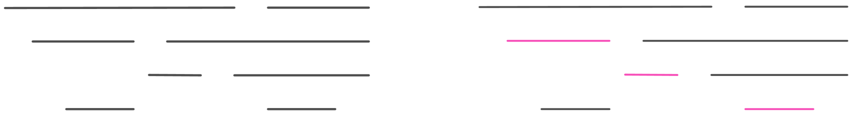
Interval Scheduling: Greedy solution that works

- Pick the task with the earliest finishing time.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.



Interval Scheduling: Greedy solution that works

- Pick the task with the earliest finishing time.
- Remove all tasks that conflict with this task.
- Repeat until no additional tasks can be added.



Implementation: can complete in $O(n\log(n))$ time

1. Sort the n requests by finishing time.
2. Select the first element in sorted array.
3. Step through the array (still sorted by finishing time) until you find the first element whose starting time is at or later than the most recently selected finishing time. Select that element
4. Repeat step 3 until you reach the end of the array.

Implementation: can complete in $O(n\log(n))$ time

1. Sort the n requests by finishing time.
2. Select the first element in sorted array.
3. Step through the array (still sorted by finishing time) until you find the first element whose starting time is at or later than the most recently selected finishing time. Select that element
4. Repeat step 3 until you reach the end of the array.

Why is this $n\log(n)$ time?

- Step 1 is $O(n\log(n))$ time.
- Step 2 is $O(1)$ time
- Step 3 (and 4) is $O(n)$ time total

In total:

$$O(n\log(n)) + O(1) + O(n) = O(n\log(n))$$

Work on implementing this now, LeetCode 435:
`https://leetcode.com/problems/non-overlapping-intervals/description/`

Proof that this greedy algorithm works

Let $g_1, g_2, \ldots, g_m$ be the indices of the intervals chosen by our greedy algorithm, ordered by finishing time:

$$f(g_1) < f(g_2) < \cdots < f(g_m)$$

Proof that this greedy algorithm works

Let $g_1, g_2, \ldots, g_m$ be the indices of the intervals chosen by our greedy algorithm, ordered by finishing time:

$$f(g_1) < f(g_2) < \cdots < f(g_m)$$

Suppose $o_1, o_2, \ldots, o_k$ are the indices of the intervals chosen by an *optimal* algorithm, ordered by finishing time:

$$f(o_1) < f(o_2) < \cdots < f(o_k)$$

Proof that this greedy algorithm works

Let $g_1, g_2, \ldots, g_m$ be the indices of the intervals chosen by our greedy algorithm, ordered by finishing time:

$$f(g_1) < f(g_2) < \cdots < f(g_m)$$

Suppose $o_1, o_2, \ldots, o_k$ are the indices of the intervals chosen by an *optimal* algorithm, ordered by finishing time:

$$f(o_1) < f(o_2) < \cdots < f(o_k)$$

We will show that $m = k$, which will prove that the greedy algorithm also proves an optimal solution.

Proof that this greedy algorithm works

Let $g_1, g_2, \ldots, g_m$ be the indices of the intervals chosen by our greedy algorithm, ordered by finishing time:

$$f(g_1) < f(g_2) < \cdots < f(g_m)$$

Suppose $o_1, o_2, \ldots, o_k$ are the indices of the intervals chosen by an *optimal* algorithm, ordered by finishing time:

$$f(o_1) < f(o_2) < \cdots < f(o_k)$$

We will show that $m = k$, which will prove that the greedy algorithm also proves an optimal solution.

First: note that, by the description of our greedy algorithm, we know that $f(g_1) \leq f(o_1)$.

Claim: $f(g_i) \leq f(o_i)$ for $i = 1, 2, \ldots, m$

Claim: $f(g_i) \leq f(o_i)$ for $i = 1, 2, \ldots, m$

We have proven that this is true for $i = 1$. Suppose, by induction, that it is true for i with $1 \leq i < m$.

Claim: $f(g_i) \leq f(o_i)$ for $i = 1, 2, \ldots, m$

We have proven that this is true for $i = 1$. Suppose, by induction, that it is true for i with $1 \leq i < m$.

Note that the interval indexed by o_{i+1} does not conflict with the intervals indexed by $g_1, g_2, \ldots, g_i$, because those intervals all end by $f(g_i) \leq f(o_i)$.

Claim: $f(g_i) \leq f(o_i)$ for $i = 1, 2, \ldots, m$

We have proven that this is true for $i = 1$. Suppose, by induction, that it is true for i with $1 \leq i < m$.

Note that the interval indexed by o_{i+1} does not conflict with the intervals indexed by $g_1, g_2, \ldots, g_i$, because those intervals all end by $f(g_i) \leq f(o_i)$.

Thus, the interval indexed by o_{i+1} *could* have been chosen at this step of the greedy algorithm. Since the interval indexed by g_{i+1} was chosen, this implies that $f(g_{i+1}) \leq f(o_{i+1})$. Thus, our claim is true.

Claim: $m = k$

Since the intervals indexed by $o_1, o_2, \ldots, o_k$ correspond to an optimal solution, we already know that $m \leq k$. So we only need to show that $m \geq k$.

Claim: $m = k$

Since the intervals indexed by $o_1, o_2, \ldots, o_k$ correspond to an optimal solution, we already know that $m \leq k$. So we only need to show that $m \geq k$.

We already know that $f(g_m) \leq f(o_m)$. If there is another interval, o_{m+1} in the optimal solution, this interval could have been added at the end of our greedy algorithm.

Claim: $m = k$

Since the intervals indexed by $o_1, o_2, \ldots, o_k$ correspond to an optimal solution, we already know that $m \leq k$. So we only need to show that $m \geq k$.

We already know that $f(g_m) \leq f(o_m)$. If there is another interval, o_{m+1} in the optimal solution, this interval could have been added at the end of our greedy algorithm.

But the greedy algorithm only ended once we couldn't add any more intervals. Thus, we must have $m = k$.

Claim: $m = k$

Since the intervals indexed by $o_1, o_2, \ldots, o_k$ correspond to an optimal solution, we already know that $m \leq k$. So we only need to show that $m \geq k$.

We already know that $f(g_m) \leq f(o_m)$. If there is another interval, o_{m+1} in the optimal solution, this interval could have been added at the end of our greedy algorithm.

But the greedy algorithm only ended once we couldn't add any more intervals. Thus, we must have $m = k$.

Thus, our greedy algorithm results in the same number of intervals as an optimal algorithm, and we have proven the greedy algorithm gives an optimal solution.

Summary

- Greedy algorithms use a rule to choose a small locally optimal step *that results in a globally optimal solution*.

Summary

- Greedy algorithms use a rule to choose a small locally optimal step *that results in a globally optimal solution*.
- Choosing that locally optimal step can be tricky; many ideas will not lead to a globally optimal solution.

Summary

- Greedy algorithms use a rule to choose a small locally optimal step *that results in a globally optimal solution*.
- Choosing that locally optimal step can be tricky; many ideas will not lead to a globally optimal solution.
- We must prove that an algorithm does result in the desired solution. Typical techniques:

Summary

- Greedy algorithms use a rule to choose a small locally optimal step *that results in a globally optimal solution*.
- Choosing that locally optimal step can be tricky; many ideas will not lead to a globally optimal solution.
- We must prove that an algorithm does result in the desired solution. Typical techniques:
  - Greedy algorithm stays ahead of any other optimal solution. (This is what we used today!)

Summary

- Greedy algorithms use a rule to choose a small locally optimal step *that results in a globally optimal solution*.
- Choosing that locally optimal step can be tricky; many ideas will not lead to a globally optimal solution.
- We must prove that an algorithm does result in the desired solution. Typical techniques:
    - Greedy algorithm stays ahead of any other optimal solution. (This is what we used today!)
    - Any optimal solution can be *exchanged* step-by-step into a greedy solution.