More Greedy Algorithms

CSE 417 Winter 21
Lecture 8

Trip Planning

Your goal is to follow a pre-set route from New York to Los Angeles.

You can drive 500 miles in a day, but you need to make sure you can stop at a hotel every night (all possibilities premarked on your map)

You'd like to stop for the fewest number of nights possible – what should you plan?

Greedy: Go as far as you can every night.

Is greedy optimal?

Or is there some reason to "stop short" that might let you go further the next night?

Trip Planning

Greedy works!

Because "greedy stays ahead"

Let g_i be the hotel you stop at on night i in the greedy algorithm.

Let OPT_i be the hotel you stop at in the optimal plan (the fewest nights plan).

Claim: g_i is always at least as far along as OPT_i.

Base Case: $i = 1$, OPT and the algorithm choose between the same set of hotels (all at most 500 miles from the start), g_i is the farthest of those by the algorithm definition, so g_i is at least as far as OPT_i.

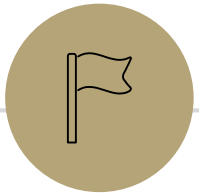
Trip Planning

Inductive Hypothesis: Suppose through the first k hotels, g_k is farther along than OPT_k.

Inductive Step:

When we select g_{k+1}, we can choose any hotel within 500 miles of g_k, since g_k is at least as far along as OPT_k everything less than 500 miles after OPT_k is also less than 500 miles after g_k. Since we take the farthest along hotel, g_{k+1} is at least as far along as OPT_{k+1}.

Wrapping up: Since g_n is greater than or equal to OPT_n for every n, The last g_i must be at least as far along as OPT_i, so we don't need an extra night compared to OPT – the greedy algorithm is optimal!



Wrapping MSTs

Other MST Algorithms

You know Prim's and Kruskal's already.

Option 3: Reverse-Delete algorithm

Start from the full graph

Sort edges in **decreasing** order, delete an edge if it won't disconnect the graph.

NOT practical (Prim's and Kruskal's are at least as fast, and conceptually easier), but fun fact!

Other MST Algorithms

How would you prove Reverse-Delete works?

Structural Proof?

Exchange Argument?

Greedy Stays Ahead?

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Other MST Algorithms

Option 4: Boruvka's Algorithm (also called Sollin's Algorithm)

Start with empty graph, use BFS to find lightest edge leaving each component.

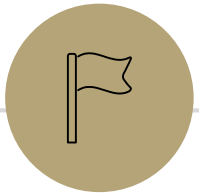
Add ALL such edges found (they're all safe edges)

Recurse until the graph is all one component (i.e. a tree)

Consider it for your practical applications!

It naturally parallelizes (unlike the other MST algorithms),

Has same worst case running time as Prim's/Kruskal's!



More Greedy

Change-Making

Suppose you need to “make change” with the fewest number of coins possible.

Greedy algorithm:

Take the biggest coin less than the change remaining.

Is the greedy algorithm optimal if you have

1 cent coins, 10 cent coins, and 15 cent coins?

What about for U.S. coinage (1, 5, 10, 25, 50, 100)

Change-Making

Suppose you need to "make change" with the fewest number of coins possible.

Take the biggest coin less than the change remaining.

Is the greedy algorithm optimal if you have

1 cent coins, 10 cent coins, and 15 cent coins?

What about for U.S. coinage (1, 5, 10, 25, 50, 100)

Introduce yourselves!

If you can turn your video on, please do.
If you can't, please unmute and say hi.
If you can't do either, say "hi" in chat.

Choose someone to share screen, showing this pdf.

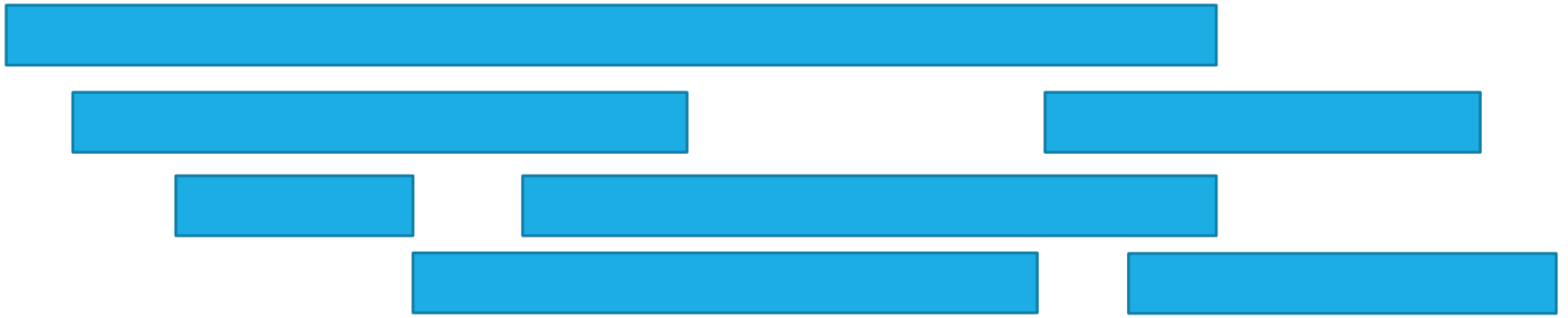
Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Interval Scheduling

You have a single processor, and a set of jobs with fixed start and end times.

Your goal is to maximize the number of jobs you can process.

I.e. choose the maximum number of non-overlapping intervals.

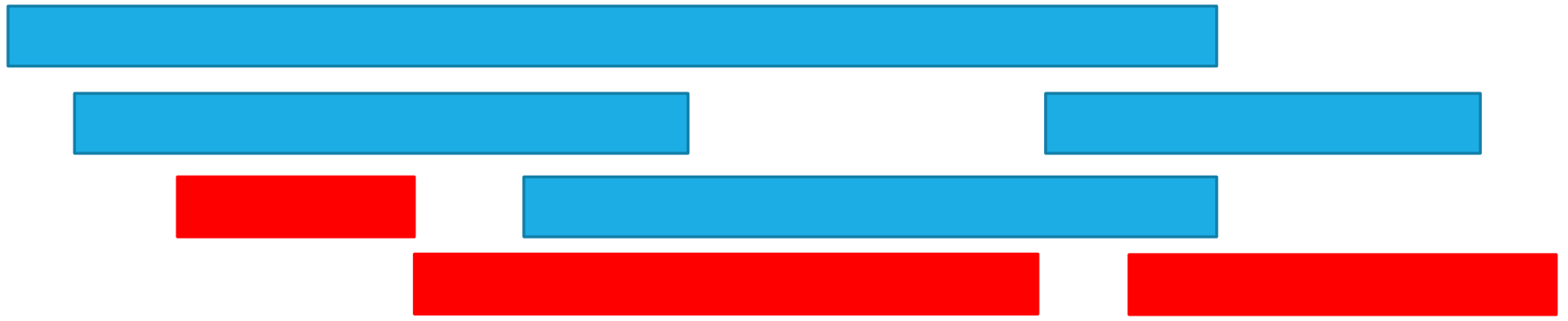


Interval Scheduling

You have a single processor, and a set of jobs with fixed start and end times.

Your goal is to maximize the number of jobs you can process.

I.e. choose the maximum number of non-overlapping intervals.



3 non-overlapping intervals

Interval Scheduling

You have a single processor, and a set of jobs with fixed start and end times.

Your goal is to maximize the number of jobs you can process.

I.e. choose the maximum number of non-overlapping intervals.

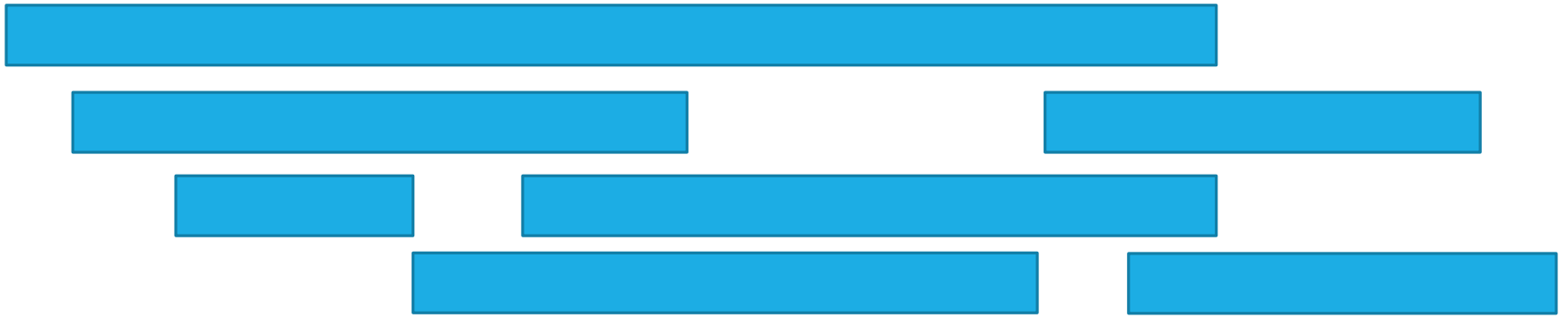
3 other non-overlapping intervals

Interval Scheduling

You have a single processor, and a set of jobs with fixed start and end times.

Your goal is to maximize the number of jobs you can process.

I.e. choose the maximum number of non-overlapping intervals.



OPT is 3 – there is no way to have 4 non-overlapping intervals;
both the red and purple solutions are equally good.

Greedy Ideas

To specify a greedy algorithm, we need to:

Order the elements (intervals)

Choose a rule for deciding whether to add.
Rule: Add interval as long as it doesn't overlap with those we've already selected.

What ordering should we use?

Think of **at least two** orderings you think might work.

Greedy Algorithm

Some possibilities

Earliest end time (add if no overlap with previous selected)

Latest end time

Earliest start time

Latest start time

Shortest interval

Fewest overlaps (with remaining intervals)

Greedy

That list slide is the real difficulty with greedy algorithms.

All of those look at least somewhat plausible at first glance.

With MSTs that was fine – those ideas all worked!

It's not fine here.

They don't all work.

As a first step – try to find counter-examples to narrow down

Greedy Algorithm

Earliest end time

Latest end time

Earliest start time

Latest start time

Shortest interval

Fewest overlaps (with remaining intervals)

Take Earliest Start Time – Counter Example

Take Earliest Start Time – Counter Example

Taking the one with the earliest start time doesn't give us the best answer.

Shortest Interval

Shortest Interval

Algorithm finds

Optimum

Taking the shortest interval first doesn't give us the best answer

Greedy Algorithm

Earliest end time

Latest end time ✖

Earliest start time ✖

Latest start time

Shortest interval ✖

Fewest overlaps (with remaining intervals)

Earliest End Time

Intuition: If u has the earliest end time, and u overlaps with v and w then v and w also overlap.

Why?

If u and v overlap, then both are "active" at the instant before u ends (otherwise v would have an earlier end time).

Otherwise v would have an earlier end time than u! By the same reasoning, w is also "active" the instant before u ends. So v and w also overlap with each other.

Earliest End Time

Can you prove it correct?

Do you want to use

Structural Result

Exchange Argument

Greedy Stays Ahead

Exchange Argument

Let $A = a_1, a_2, \ldots, a_k$ be the set of intervals selected by the greedy algorithm, ordered by endtime

OPT= $o_1, o_2, \ldots, o_\ell$ be the maximum set of intervals, ordered by endtime.

Our goal will be to "exchange" to show A has at least as many elements as OPT.

Let a_i, o_i be the first two elements where a_i and o_i aren't the same. Since a_{i-1} and o_{i-1} are the same, neither a_i nor o_i overlaps with any of $o_1, \ldots, o_{i-1}$. And by the greedy choice, a_i ends no later than o_i so a_i doesn't overlap with o_{i+1}. So we can exchange a_i into OPT, replacing o_i and still have OPT be valid.

Exchange Argument

Repeat this argument until we have changed OPT into A.

Can OPT have more elements than A?

No! After repeating the argument, we could change every element of OPT to A. If OPT had another element, it wouldn't overlap with anything in OPT, and therefore can't overlap with anything in A after all the swaps. But then the greedy algorithm would have added it to A.

So A has the same number of elements as OPT does, and we really found an optimal

Greedy Stays Ahead

Let $A = a_1, a_2, \ldots, a_k$ be the set of intervals selected by the greedy algorithm, ordered by endtime

OPT= $o_1, o_2, \ldots, o_\ell$ be the maximum set of intervals, ordered by endtime.

Our goal will be to show that for every i, a_i ends no later than o_i.

Proof by induction:

Base case: a_1 has the earliest end time of any interval (since there are no other intervals in the set when we consider a_1 we always include it), thus a_1 ends no later then o_1.

Greedy Stays Ahead

Inductive Hypothesis: Suppose for all $i \leq k$, a_i ends no later than o_i.

IS: Since (by IH) a_k ends no later than o_k, greedy has access to everything that doesn't overlap with a_k. Since a_k ends no later than o_k, that includes o_{k+1}. Since we take the first one that doesn't overlap, a_{k+1} will also end before o_{k+1}.

Therefore a_{k+1} ends no later than o_{k+1}

Wrapping Up: Since every a_i ends no later than o_i, the last interval greedy selects (a_n) is no later than o_n. There cannot be an o_{n+1}, as if it didn't overlap with o_n it also wouldn't overlap with a_n and would have been added by greedy.

Greedy Algorithm

Earliest end time ✓

Latest end time ✗

Earliest start time ✗

Latest start time

Shortest interval ✗

Fewest overlaps (with remaining intervals)

Other Greedy Algorithms

It turns out latest start time and fewest overlaps also work.

Latest start time is actually the same as earliest end time (imagine "reflecting" all the jobs along the time axis – the one with the earliest end time ends up having the last start time).

What about fewest overlaps?

Easiest proof Robbie knows observes that fewest overlaps means you have the earliest finish time (among a certain subset of the intervals)

Greedy Algorithm

Earliest end time ✓

Latest end time ✗

Earliest start time ✗

Latest start time ✓

Shortest interval ✗

Fewest overlaps (with remaining intervals) ✓

Summary

Greedy algorithms

You'll probably have 2 (or 3…or 6) ideas for greedy algorithms. Check some simple examples before you implement!
Greedy algorithms rarely work.

When they work AND you can prove they work, they're great!

Proofs are often tricky

Structural results are the hardest to come up with, but the most versatile.

Greedy stays ahead usually use induction

Exchange start with the **first** difference between greedy and optimal.