More Greedy Algorithms

CSE 421 23WI
Lecture 6

Announcements

No handouts today.

Homework 2 deadline is delayed until tomorrow 11:59 PM (late problems still last 48 hours so go until Thursday).

We're working to get you at least one of the bigger problems back; in the meantime solutions will go on Ed tonight.

In general: Our solutions will fit in one (typed) page; pretty soon we'll start requiring yours fit in two typed pages.

Outline

Last Time

What is a greedy algorithm? A “structural” argument.

Today

Other styles of greedy proof (exchange, greedy stays ahead)

Practice generating a greedy algorithm from start to (as far as we get)

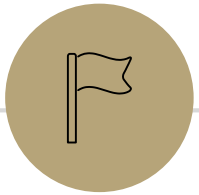
Section Tomorrow

Practice yourself; general problem solving process you’ll continue to use all quarter.

Friday

Finish the problem we started today

Greedy algorithms as approximation algorithms



An Exchange Argument

What about Kruskal's?

Exchange argument:

General outline:

Suppose, you didn't find the best one.

Suppose there's a better MST

Then there's something in the algorithm's solution that doesn't match OPT. (an edge that isn't a safe edge/that's heavier than it needs to be)

Swap (**exchange**) them, and finish the proof (arrive at a contradiction or show that your solution is equal in quality)!

Exchange Arguments

You almost always want the “extremality” trick.

Don’t just suppose “something is different”

Suppose something is different and focus on the *first* spot where the greedy algorithm made a different decision.

The next two slides are what happen when you don’t use the extremality trick. You end up finding “earlier and earlier counter-examples”

Kruskal's Proof (v1)

Suppose, for the sake of contradiction, T_K, the tree found by Kruskal's algorithm isn't a minimum spanning tree. Let T' be the true minimum spanning tree.

Let $e = (u, v)$ be an edge in T_K but not T'. Add e to T'. In doing so we created a cycle, C, (e along with the path from u to v in T', which exists because T' spanned.).

Our goal is to do an exchange argument – we need a new lighter tree!

We divide into cases,

Case 1: e is not the heaviest edge in C. Then delete the heaviest edge to create T''. Since e replaced the heavier edge, T'' is lighter than T'. And T'' is a spanning tree (T'' has n-1 edges and spans because T' did and we just deleted an edge on a cycle). But that contradicts T' being the MST!

Kruskal's Proof (v1, cont.)

We won't be able to reach a contradiction from the cycle, but we will find another edge to examine

Case 2: e is the heaviest edge in C.

Since Kruskal's added e to our graph, there must be some edge, f, on the cycle which was not in T_K. But f was processed before e by Kruskal's (since e is heavier). Which means f would have formed a cycle, C' in T_K had it been added when it was processed.

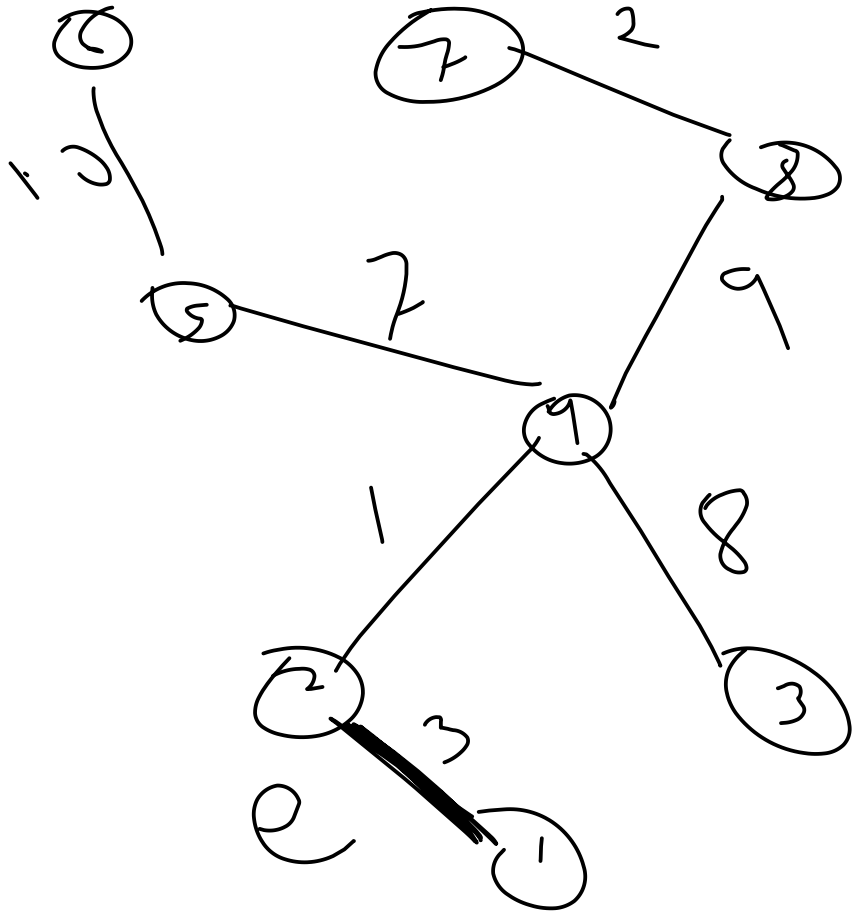
By the process ordering, f is the heaviest edge in C'. There are no cycles in T' (since it's a tree) so there is an edge (call it e') in C' that is not in T'. This new edge e' meets exactly the assumptions we had on e, but is lighter.

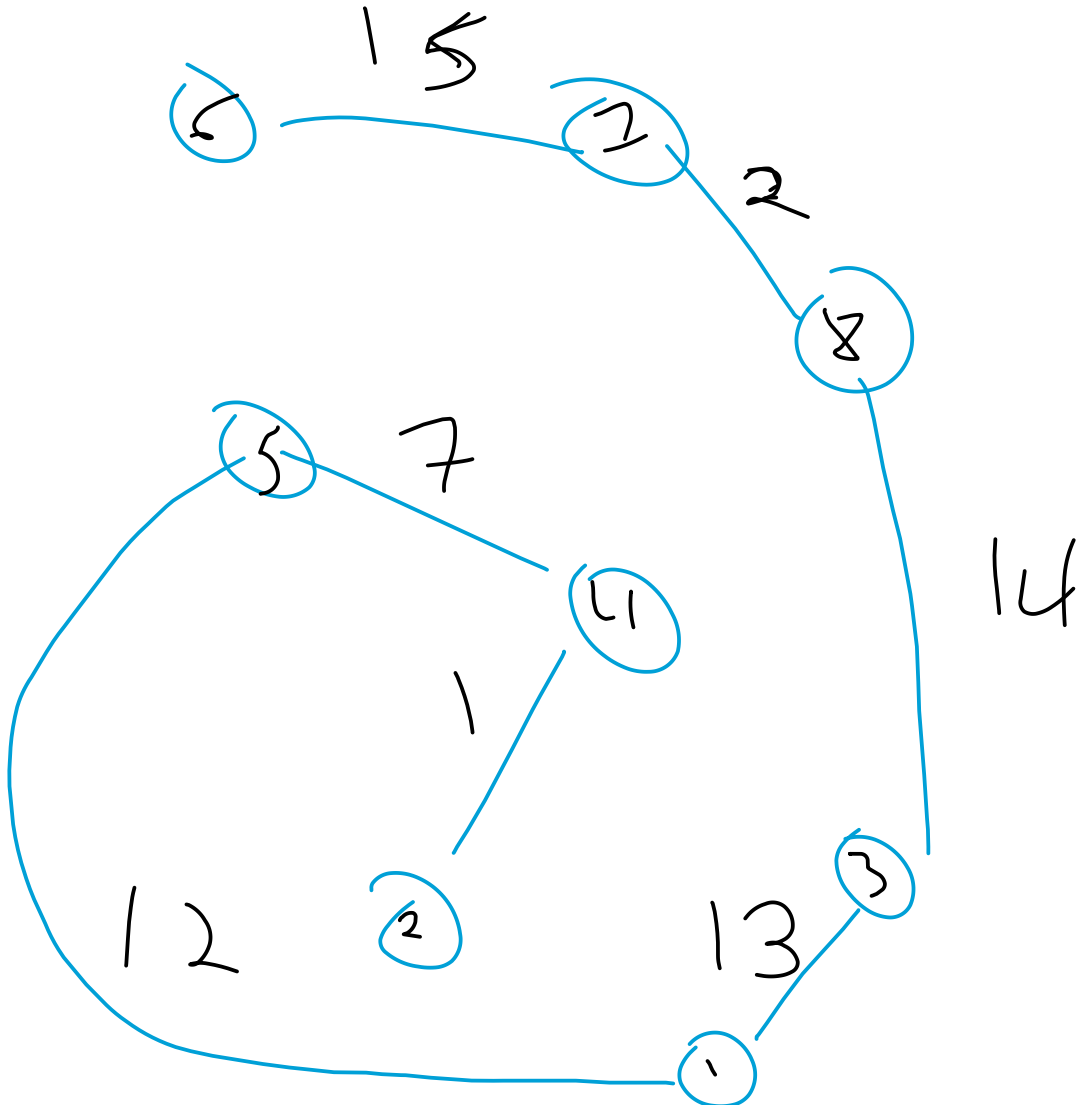
Repeat the original argument on e'. Since the graph is finite, we must eventually hit Case 1, which gives our needed contradiction.

Kruskal's Proof (pretty version)

Suppose, for the sake of contradiction, T_K, the tree found by Kruskal's algorithm isn't a minimum spanning tree. Let T' be the true minimum spanning tree.

Let $e = (u, v)$ be the lightest edge in T_K but not in T'.


0
7
2
8
12
5
7
9
9
1
8
2
3
3
1

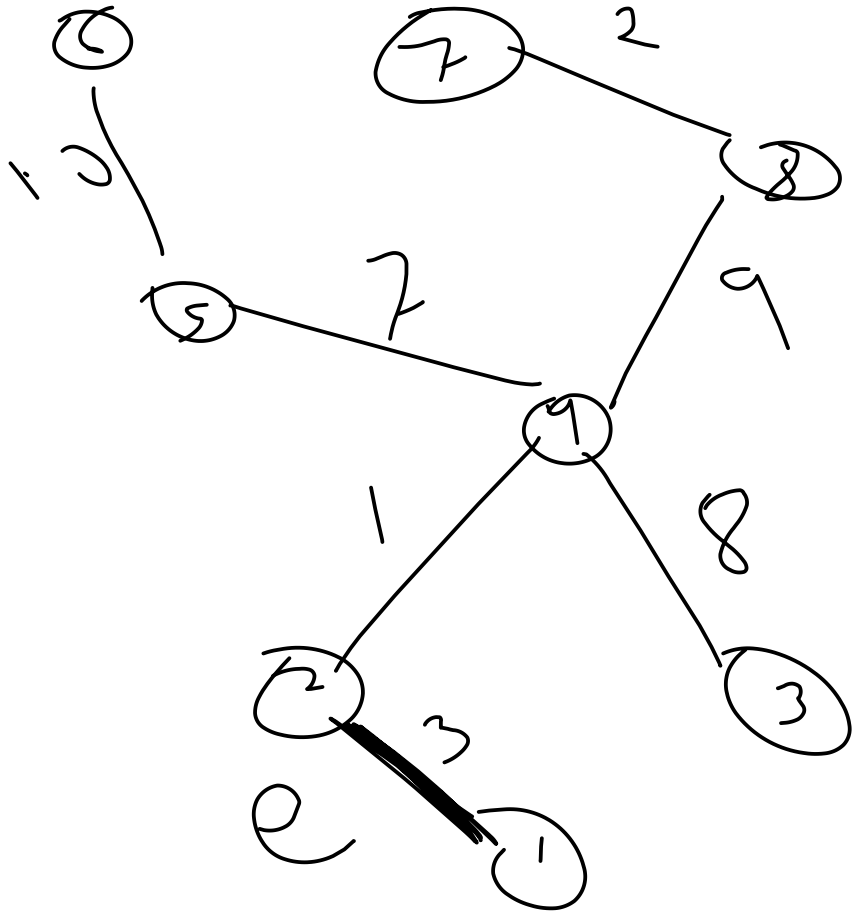

15
6
7
2
8
5
7
14
11
1
12
2
13
3
1

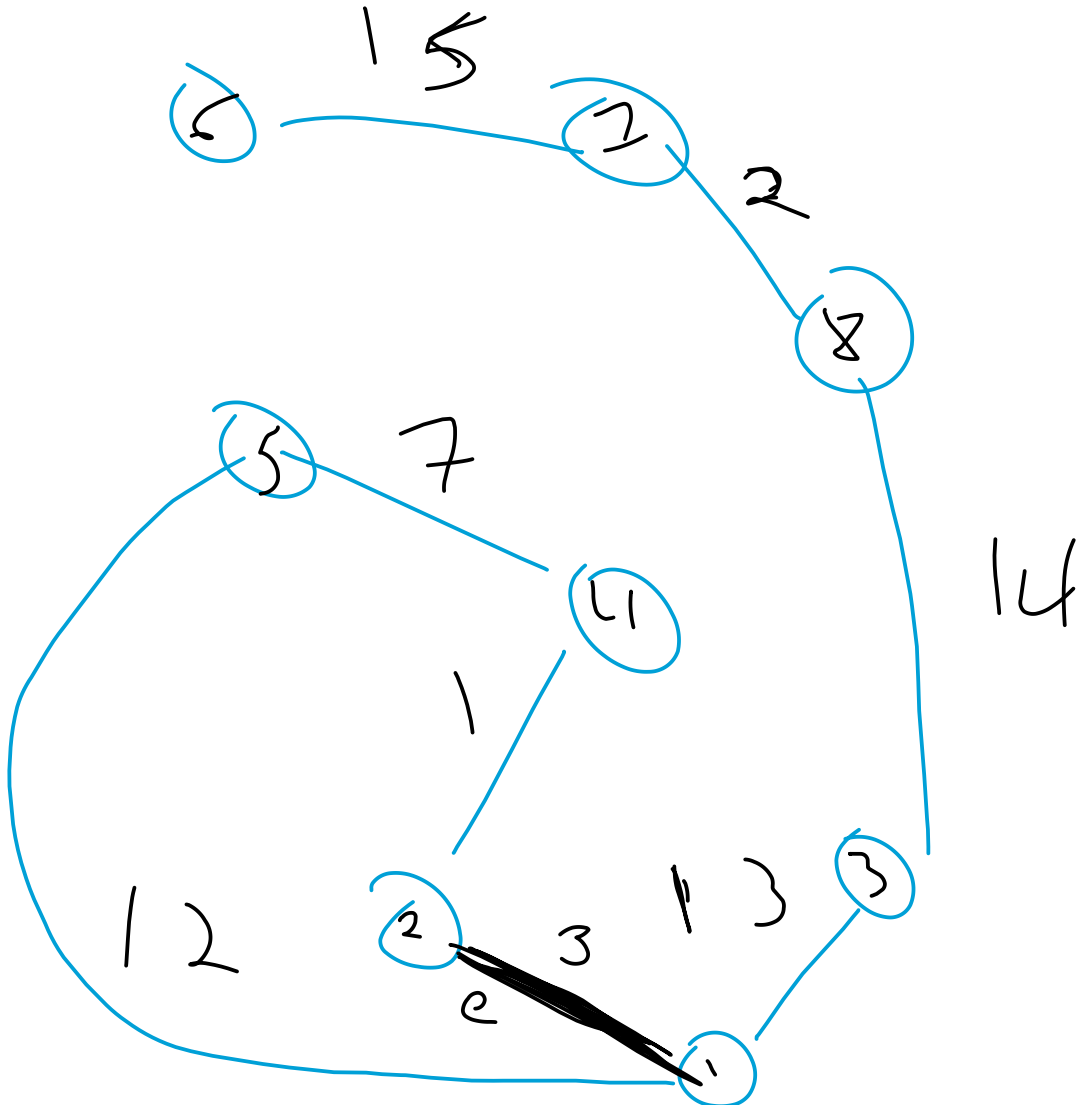
Kruskal's Proof (pretty version)

Suppose, for the sake of contradiction, T_K, the tree found by Kruskal's algorithm isn't a minimum spanning tree. Let T' be the true minimum spanning tree.

Let $e = (u, v)$ be the lightest edge in T_K but not in T'. Add e to T', and we will create a cycle (because there is a way to get from u to v in T' by it being a spanning tree).

We claim that e is not the heaviest edge on the cycle. Since e is the lightest in T_K but not T', everything lighter than e in T_K is also in T'. We put e in T_K so it didn't create a cycle there. That means there is an edge on the cycle heavier than e.


0
7
2
8
12
5
7
9
9
1
8
2
3
3
e
1


15
6
7
2
8
14
5
7
11
1
12
2
3
13
3
e
1

Kruskal's Proof (pretty version)

Suppose, for the sake of contradiction, T_K, the tree found by Kruskal's algorithm isn't a minimum spanning tree. Let T' be the true minimum spanning tree.

Let $e = (u, v)$ be the lightest edge in T_K but not in T'. Add e to T', and we will create a cycle (because there is a way to get from u to v in T' by it being a spanning tree).

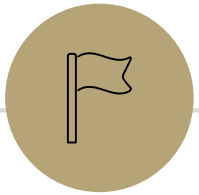
We claim that e is not the heaviest edge on the cycle. Since e is the lightest in T_K but not T', everything lighter than e in T_K is also in T'. We put e in T_K so it didn't create a cycle there. That means there is an edge on the cycle heavier than e. Delete that edge, and call the resulting graph T''. Observe that T'' is a spanning tree (it has $n - 1$ edges, and spans all the same vertices T' did since we deleted an edge from a cycle). But it has less weight than T' which was supposed to be the MST. That's a contradiction!

Hey…Wait a minute

That was pretty similar to last time. They both used an “exchange” idea.

The boundaries between the proof principles are a little blurry…

They’re meant to be useful for you for thinking about “where to start” with a proof, not be a beautiful taxonomy of exactly what you do in every possible proof.



Greedy Stays Ahead

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.

Intuition: they start at the same point before day 1, and greedy goes as far as possible, so is "ahead" after day 1.

And if greedy is "ahead" at the start of the day, it will continue to be ahead at the end of the day (since it goes as far as possible, and the distance you can go doesn't depend on where you start).

Therefore it's always ahead. And so it uses at most the same number of days as all other solutions.

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.

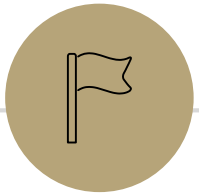
Base Case: $i = 1$, OPT and the algorithm choose between the same set of hotels (all at most 500 miles from the start), g_i is the farthest of those by the algorithm definition, so g_i is at least as far as OPT_i.

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}.



Interval Scheduling

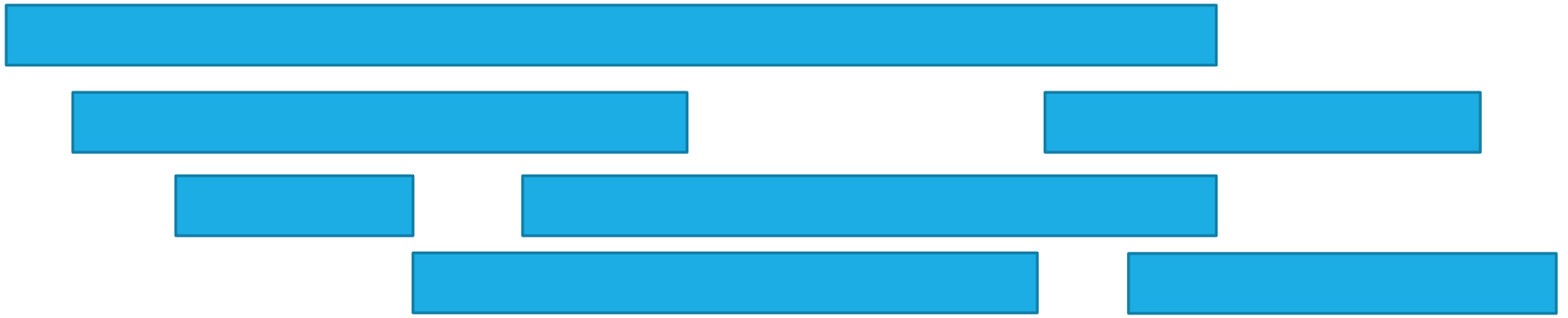
From problem statement to algorithm ideas to algorithm to proof

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.

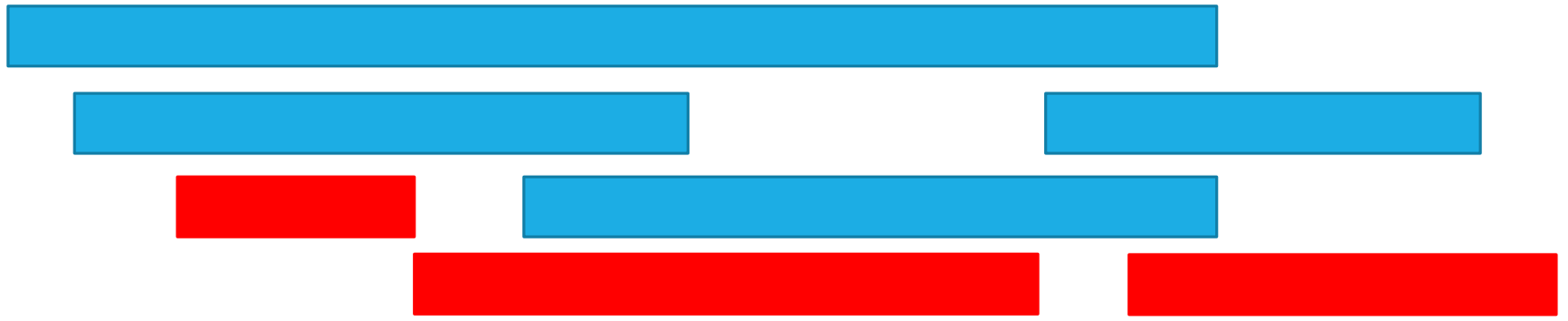


Interval Scheduling

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.

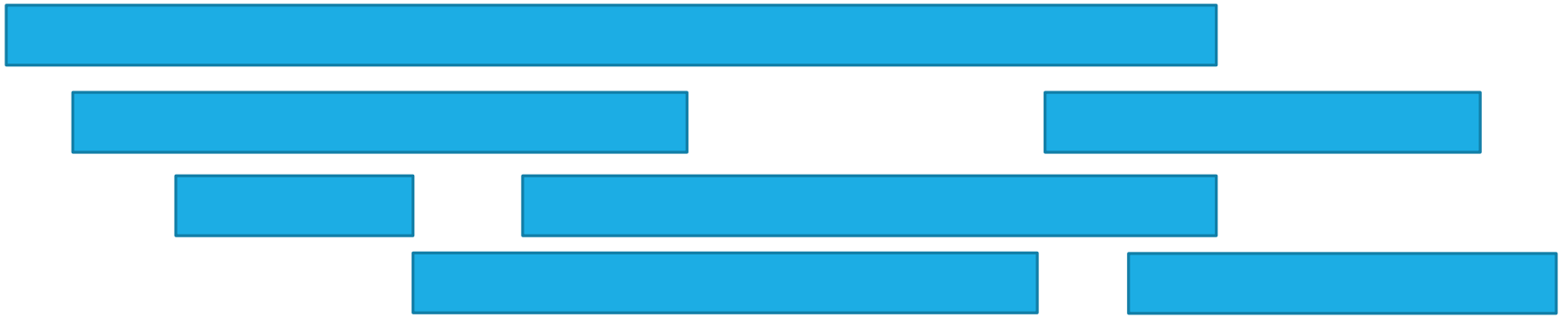
3 other non-overlapping intervals

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 also works.

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?

Doesn't work. ☹ Counter-examples are a little more complicated than the others.

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.