More Greedy; Course wrap-up

CSE 417 Winter 24
Lecture 27

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)

Change-Making

The greedy algorithm doesn't always work!

We forced you to observe that in a homework problem for those that did it.

But there are times where it does

For "standard" US coinage, the greedy algorithm works

And it also always works if your coins always exactly double in value.

Another reason to be very careful with greedy algorithms!

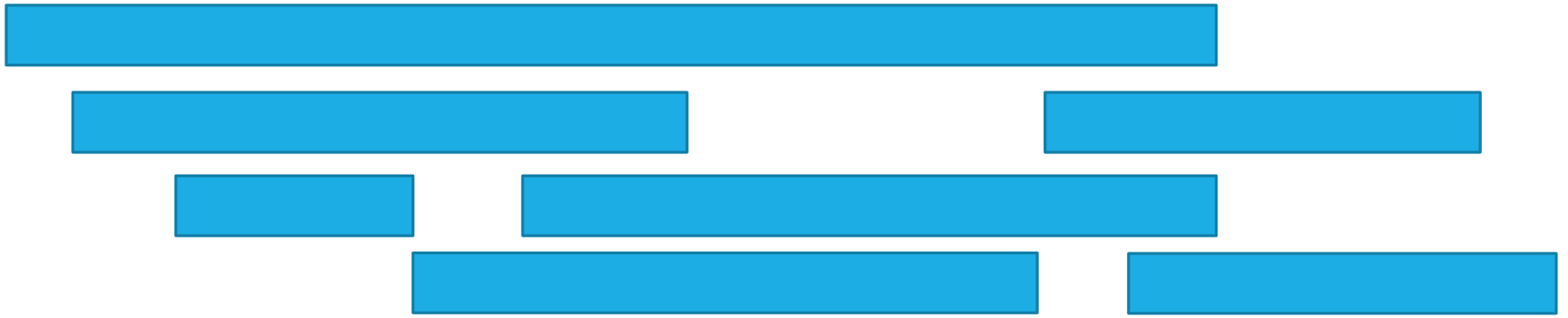
Also a good example of how you can sometimes avoid greedy if you can't figure out a proof – maybe there's a way to write a DP instead!

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.

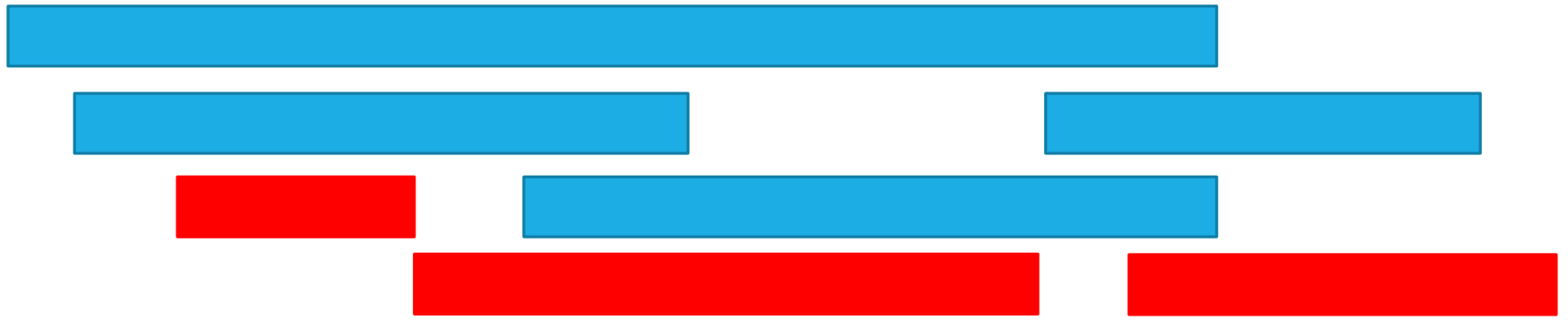


Interval Scheduling

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.

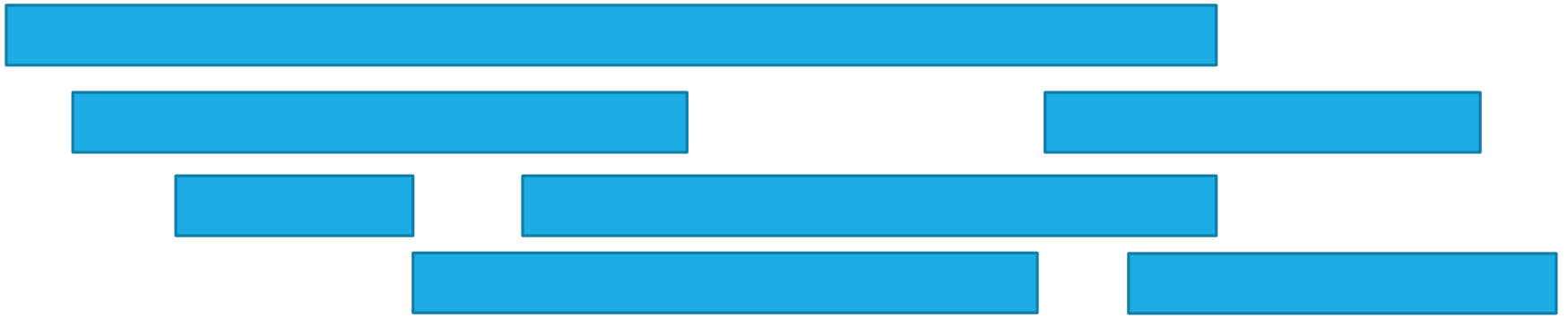
3 other non-overlapping intervals

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!

Fewest Overlaps counter-example

The top middle item will be selected first, eliminating the chance of getting the 4 intervals in OPT.

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.

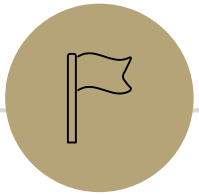
When they work AND you can prove they work, they're great!

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.



Course Wrap; Which Technique

Which Technique?

When I see a new problem, how do I know which option to use?

Try modeling first – use the problems we've already solved

Preferences? Try stable matching.

Assignments? Try network flow.

Etc.

Which Technique?

Modeling Didn’t work?

What does this remind you of?

Then try:
recursive thinking (will lead to DP or D&C)

Ask how you would represent the problem visually (might lead to graphs)

Finally: remember your problem might be hard!

We've Covered A LOT

Stable Matchings
Proof fundamentals
Modifications of BFS/DFS
Divide & Conquer
Dynamic Programming
LPs
Network Flow (and assignment problems)
Reductions, P vs. NP
Coping with NP-completeness
Greedy algorithms

What Comes next?

CSE 417 is intended as a "last" undergraduate course for those of you about to graduate. But if you are hoping for more, consider:

CSE 422 (toolkit for modern algorithms)

Mathematics and algorithms for larger data sets/ML, approximate solutions

CSE 521 (Graduate algorithms)

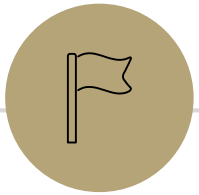
More math background expected, more approximation algorithms, and a few other topics.

CSE 431 (Complexity theory)

Proof-based (all your homework problems are proofs). More on NP, undecidability, other notions of "hard."

Courses in Math & AMATH (especially for LPs and network flow)

Math 461, 462 (combinatorics); Math 407, 408 (LPs, convex optimization); Math 409 (lots of overlap with this course)
Math 514 (deep theory behind LPs, MSTs, etc.)



Tech Interview Advice

Resources

Cracking the Coding Interview (McDowell) or another book like it.
Copies are at the library!

General Advice from Kasey Champion

Leetcode (we…borrowed…more than one homework problem from there).

General resume advice (and other resources). Some specific to Allen School.

Technical Questions

Your interviewer wants to see

1. What you know
2. Can you communicate technical ideas
3. Can you problem-solve

Getting the “right” solution helps with point 1. Not points 2 and 3.

You also need to talk through what you see in the problem/how you solve it.

The best way is to practice. Find a friend, a set of interview questions, and a whiteboard and practice.

Technical Questions

Show you can problem-solve.

It's tempting to "skip" the problem-solving steps if you know the answer. It's a good idea to still do all of them.

You can go quicker, but give them something.

Don't feel obligated to use this exact method – others exist, but you should hit these points. **Even if you know the best algorithm from the time you read the problem statement.**

A sample Problem: https://leetcode.com/problems/house-robber/

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

TEBOW IT

Talk

Example

Baseline (aka Brute Force)

Optimize

Walk-Through

Implement

Test

https://commons.wikimedia.org/wiki/File:Tim_Tebow_in_the_dugout.jpg

Tim Tebow:
Football player turned football analyst turned simultaneous-baseball-player-and-football-analyst turned football player turned football analyst again.

Acronym from Kasey Champion; similar process in *Cracking the Coding Interview*

Talk

Give you time to take deep breaths
Confirm you haven't missed anything
Can sometimes get a simplifying assumption added.

Make sure you understand every technical term in the prompt (if any).

Ask questions to make sure you've got the idea.

Ask about simplifying assumptions.

Pick out any key pieces of the problem

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

Talk

If I enter houses 2 and 4 only, I don't set off an alarm, right?

The houses are just in a line, I'm only worrying about "one side" of the street?

I'm planning just one night?

The amount of money is an integer? Can it be negative?

I don't have to record which houses, just the final number?

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

Talk

Self-checks

Do I know the method signature (inputs and return type)?

Is there any information in the prompt that I feel is useless?

There usually isn't any, so maybe ask a question about that

Example

You'll need these later!

Make a sample input or two, find the answer and **confirm it's right with the interviewer**

Example 1:

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

Baseline

Finally, designing the algorithm. What we came here for.

First algorithm that comes to mind.

Doesn't matter how slow.[1]

This is a good problem solving technique, sometimes you want to optimize the first thing you think of.

Sometimes you don't, but it's still useful for you as a human.

You're going to be nervous. It's a lot easier to think creatively with the baseline in your back pocket. You won't end this interview having written no code.

[1] ok, it does matter how slow, but that's not the point ***yet***

Baseline

If you see what to do right away that's fine! You can short-circuit this step. But if it doesn't work come back here! I don't like to spend more than a few minutes with no back-up plan.

Remember the examples you did! Those can help now. Were you doing a brute force on those to solve them?

Baseline

Well I guess we could check all 2^n sets of houses.
Will be really slow, but will work.

Let's try to do better.

Optimize

Sometimes you're going to speed up the baseline (say with better data structures).
Other times you're starting from scracth

Now, let's get a better algorithm.

What did you see in your examples?

What does this remind you of?

We're finding the maximum something in an array, that might remind you of maximum subarray sum. Maybe DP?

Let's start there…

Optimize

Let's think recursively. Start with the last house.

We're currently thinking about house i.

If we want to include i, then we want…

If we want to exclude i then we want…

Optimize

Let's think recursively. Start with the last house.

We're currently thinking about house i.

If we want to include i, then we want…

If we want to exclude i then we want…

Optimize

Let's think recursively. Start with the last house.

We're currently thinking about house i.

If we want to include i, then we want…

The best robbing plan from 0 to $i - 2$

If we want to exclude i then we want…

The best robbing plan from 0 to $i - 1$. We don't have to include $i - 1$ through.

Let OPT be the value of the best robbing plan for the array from 0 to i.

Optimize

You could also get an

OPT/Include split like we've often seen if you jump right there.

Walk-Through

Now we've got our idea…time for a very quick check

What happens on one of your small examples?

`nums`

0	1	2	3	4
2	7	9	3	1

`OPT`

0	1	2	3	4

Walk-Through

You might notice bugs or edge cases (like the base cases here!) if you go slowly

Now we've got our idea…time for a very quick check

What happens on one of your small examples?

`nums`

0	1	2	3	4
2	7	9	3	1

`OPT`

0	1	2	3	4
2	7	11	11	12

Implement

Tips: Use default data structures.

Your interviewer probably doesn't remember all the tiny details of java default libraries either, so feel free to say "I'm going to assume the library has an X method" or "it works in this way" if you can't remember whether it's the Java or C++ version that works a certain way.

You might never have written code on a whiteboard (or typed without an IDE auto-completing things). You should practice those skills!!

Use comments and break-off easy methods/repeated code.

If you're running out of time, your interviewer might let you skip it.

```
int robbingNumber(int[] input){
  int[] opt = new int[input.size]; //opt robbing number where we can access homes 0,…,i.
  opt[0] = input[0];
  opt[1] = Math.max(input[0], input[1]);
  for(int i=0; i < input.size; i++){
        opt[i] = Math.max( opt[i-1], opt[i-2]+input[i]);
}
  return opt[input.size-1];
```

Test

If you haven't already, do your big-O analysis

Then test your code!

- Use the examples you made early on
- Think about small cases (empty input, null input)
- Think about edge cases (repeated numbers)
- Double-check you've used all assumptions

Another Problem

Drive, a new rideshare company, focuses on getting their customers long-distances by having many individual drivers take them through separate parts of the journey. Drive has decided on a set of specific locations where pickups and drop-offs can happen, and they have the cost to ride between every pair of locations.

To incentivize you to try the system, Drive has given you a token for one free ride. Because the system is new, they haven't mapped out how you'll go long distances. It's up to you to decide which locations to visit on the way from your starting point to the end point and which one you will use your coupon on.

Describe an algorithm that will let you get from your given start and end points while spending the minimum amount of money (with your token).

A Very Fun Trick

Super optimized trick: Make two copies of the graph, for every edge (u, v) also add a weight $\mathbf{0}$ copy from u in copy 1 to v in copy 2 (along with ones within each copy at weight equal to the time to drive).

Jumping from copy 1 to copy 2 is using your token. Dijkstra's from start in copy 1 to destination in copy 2 is your final answer right away.

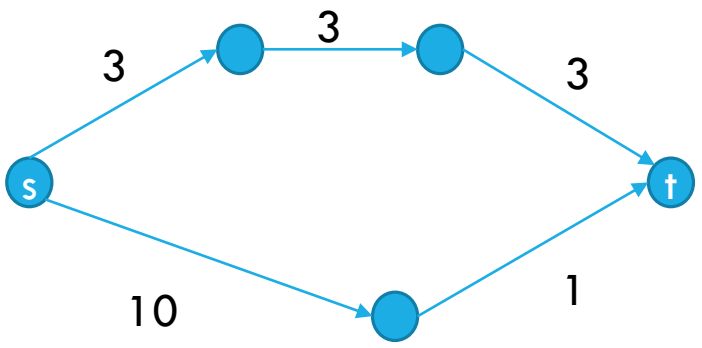
Multiple graph copies is surprisingly useful to represent these "you can do a weird thing a constant number of times" questions.

Other options

One at a time, make every edge weight 0 and run Dijkstra's.

A factor of m slower than optimal, but definitely works.

Cannot just find a shortest path and use a token on the most expensive ride.



One More For the Road

Given a list of integers, list all pairs $(A[i], A[j])$ such that $A[i] + A[j] = 10$.

Optimized options

There are multiple options here:

Data structures-based (use a hash table, lookup $A[i]$'s potential paired value)

Clever trick: sorting makes this easier!

Sort the list. You can use binary search to find where the "potential pair" should be.