CS 3100
Data Structures and Algorithms 2
Lecture 12: Intro. to Greedy Algorithms

Co-instructors: Robbie Hott and Ray Pettit
Spring 2024

Readings in CLRS 4th edition:

- Chapter 15. (Today, 15.1 and 15.2)

Announcements

- Quizzes 1-2 Thursday
  - Both quizzes taken the same day
  - Information on our class website
  - Review Session tonight at 6:30pm in Chem 402
  - If you have SDAC, please schedule for 1 exam (*not a quiz*)
- PS5 Coming Soon!
- Course email: cs3100@cshelpdesk.atlassian.net
- Office hours
  - No office hours this Sunday or over Spring Break. We'll start back next Sunday (but please check the calendar)
  - Office hours are not for "checking solutions"

Coin Changing: A “Simple” Algorithm

Finding the correct change with minimum number of coins

Problem: After someone has paid you cash for something, you must:

- Give back the right amount of change, and…
- Return the fewest number of coins!

Inputs: the dollar-amount to return

- Also, the set of possible coins

Output: a set of coins

Let’s talk about this in more detail

Coin Changing: A "Simple" Algorithm

Imagine a world without computerized cash registers!

The problem: Given an unlimited quantities of pennies, nickels, dimes, and quarters (worth value 1, 5, 10, 25 respectively), determine a set of coins (the *change*) for a given value x using the fewest number of coins.

How Would You Solve This?

Would this be your algorithm?

- Generate each possible set of coins that sum to *x*.
- Determine which of these sets has the fewest coins.

No, this is probably *not at all* what you thought of doing!

- It’s correct. But it’s a *brute force* approach.

What would you do?

- Take a moment and try to describe your approach as an algorithm.

Change Making Algorithm

Given: target value x, list of coins $C = [c_1, \dots, c_k]$
(in this case $C = [1, 5, 10, 25]$)

Repeatedly select the largest coin less than the remaining target value:

```
while ($x > 0$)
    let $c = \max(c_i \in \{c_1, \dots, c_k\} \mid c_i \leq x)$
    add $c$ to solution
    $x = x - c$
```

Observation: We can rewrite this to take $\lfloor n/c \rfloor$ copies of the next largest coin at each step, and reduce x by $(c \cdot \lfloor n/c \rfloor)$

Avoid call to max() by choosing next c_i from largest to smallest.

C must be sorted.

Let’s reflect on this

What’s its time-complexity? pseudo polynomial

- Looks like it’s $O(x)$ in the worst-case. (Why do I say that?)
  - Maybe it’s $O(kx)$ if I really have to do a max() operation at each step
  - Maybe it’s $O(k)$ if C is sorted. Or would it be $O(k \log k)$?

Does this algorithm always work? I.e. how can we prove it to be correct?

- Intuitively you know it’s true for US coins, right?

Some Terminology Before We Continue…

Optimization problems: terminology

- A solution must meet certain constraints:
  A solution is *feasible*

Example: All edges in solution are in graph, form a simple path.

- Solutions judged on some criteria:

  Objective function

Example: Sum of edge weights in path is smallest

- One (or more) feasible solutions that scores highest (by the objective function) is called the *optimal solution(s)*

Both **dynamic programming** and the **greedy approach** are often good choices for optimization problems.

Greedy Strategy: An Overview

Greedy strategy:

- Build solution by stages, adding one item to the partial solution we've found before this stage
- At each stage, make *locally optimal choice* based on the ***greedy choice*** (sometimes called the *greedy rule* or the *selection function)*
  - Locally optimal, i.e. best given what info we have now
- Irrevocable: a choice can't be un-done
- Sequence of locally optimal choices leads to globally optimal solution (hopefully)
  - Must prove this for a given problem!
  - Sometimes basis for *approximation algorithms* or *heuristic algorithms* used to get something close to optimal solution.

We’ve Seen Greedy Graph Algorithms

Dijkstra’s Shortest Path is greedy!

Build solution by adding item to partial solution

- Dijkstra’s: add edge to connect *k*th vertex, where the edges for the *k-1* already selected show the shortest paths to those *k-1* vertices

Greedy choice

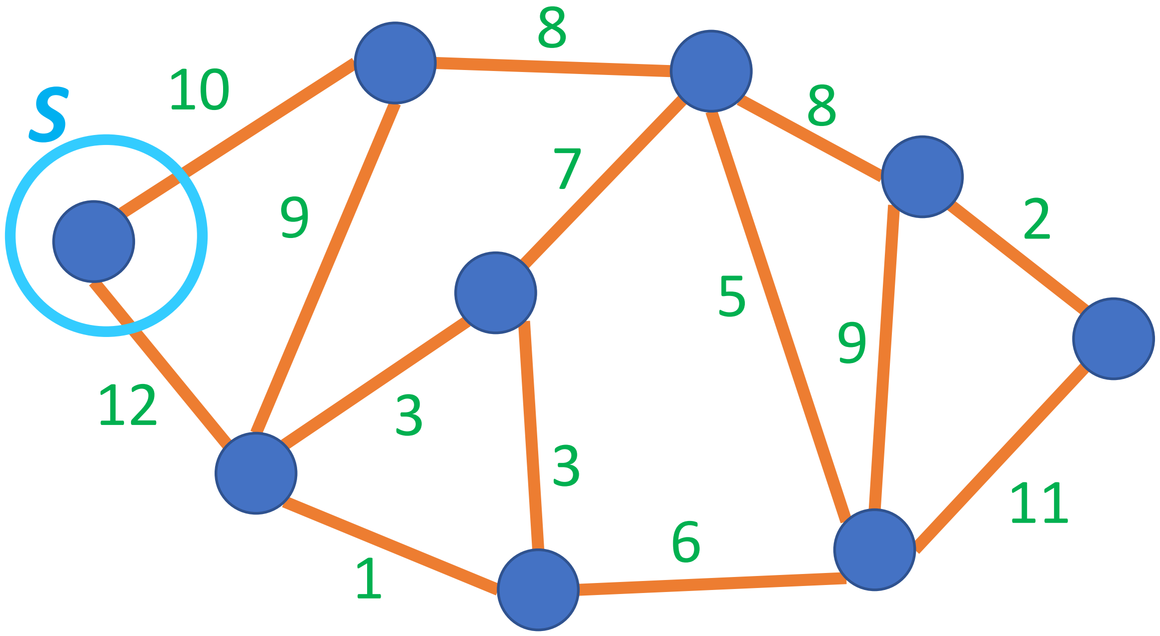
- Dijkstra’s: for all vertices connected to one of the *k-1* vertices processed, choose *w* where *dist(s,w)* is the minimum

We did have to prove that this sequence of locally optimal choices leads to globally optimal solution

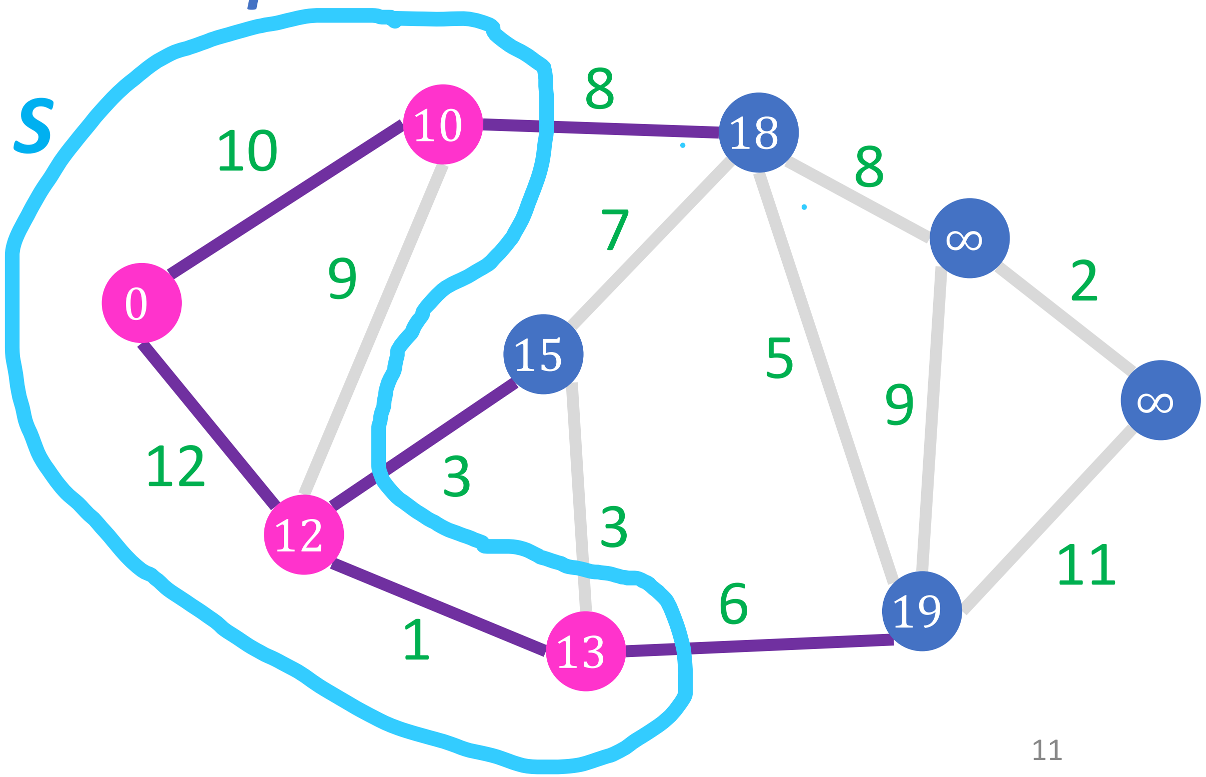
Dijkstra's Algorithm

1. Start with an empty tree S and add the source to S
2. Repeat $|V| - 1$ times:
   - At each step, add the node "nearest" to the source not yet in S to S

Initially:



At some point later:



Back to Coin Changing: Correctness?

Can you think of how you might argue this strategy (algorithm) always choose the optimal solution for coin-changing?

Maybe argue along these lines:

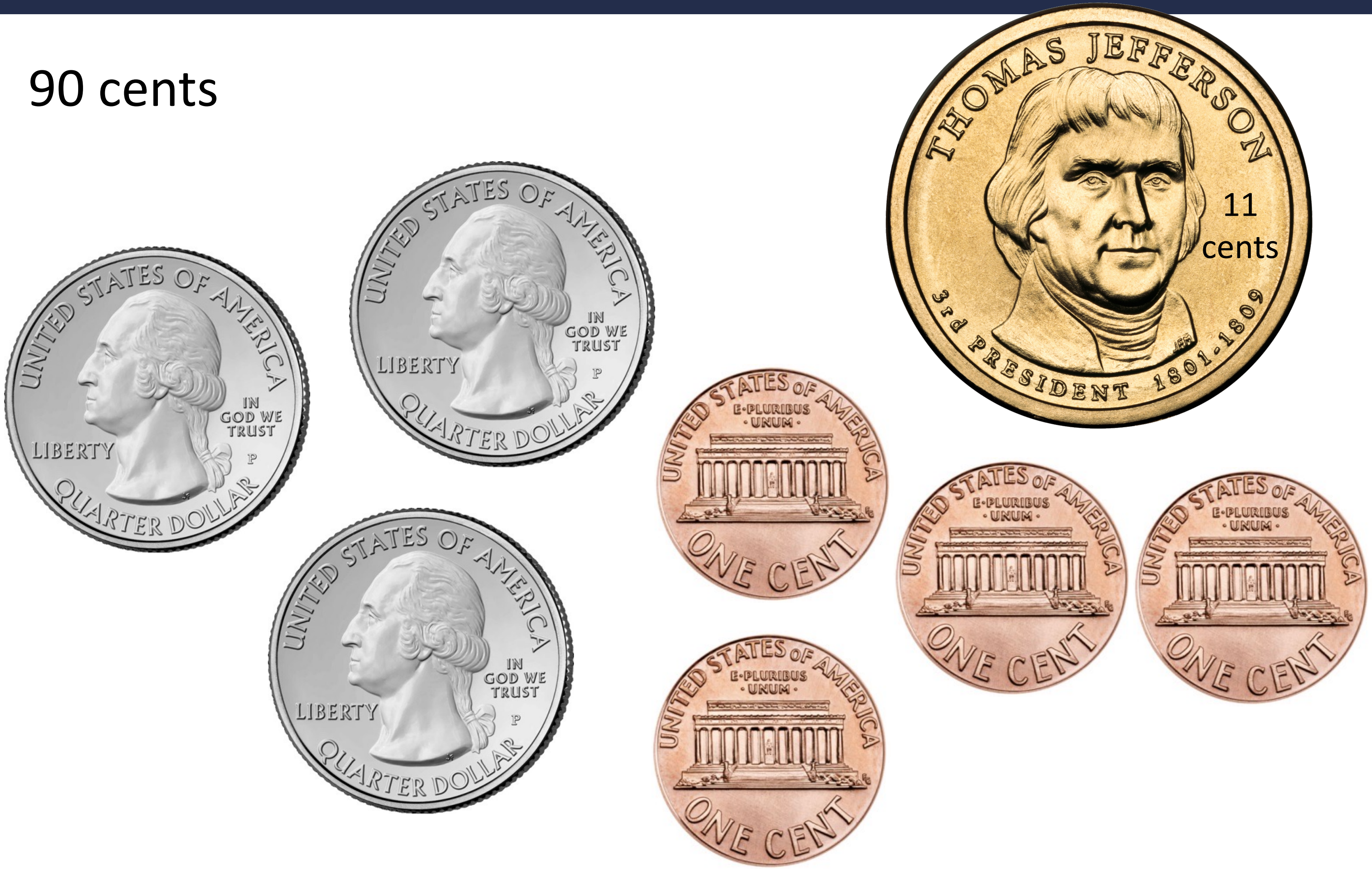
- If an algorithm did something different than what our algorithm does, then it won’t choose optimal solution.
- Or, if an algorithm did something different than what our algorithm does, we can swap what they did for what we do and we won't make their algorithm any worse. (Exchange argument)
- We’ll see proof later in slides.

Warm Up?, take 2

Given access to unlimited quantities of pennies, nickels, dimes, toms, and quarters (worth value 1, 5, 10, 11, 25 respectively), give 90 cents change using the **fewest** number of coins.

Greedy method’s solution

90 cents



Greedy solution not optimal!

90 cents

Warm Up?, take 2

Given access to unlimited quantities of pennies, nickels, dimes, toms, and quarters (worth value 1, 5, 10, 11, 25 respectively), give 90 cents change using the **fewest** number of coins.

We can solve coin changing with dynamic programming (to be discussed soon).

That strategy <u>will</u> work for this set of coins!

Summary of the Greedy Approach

Problem must have Optimal Substructure

- Optimal solution to a problem contains optimal solutions to subproblems
- Next slide has more details

Idea:

1. Identify a greedy choice property
    - How to make a choice guaranteed to be included in some optimal solution
2. Repeatedly apply the choice property until no subproblems remain

Greedy approach only considers one subproblem at each stage

Change Making Choice Property

Our algorithm's **Greedy choice:**
Choose largest coin less than or equal to target value

Leads to optimal solution?

- For standard U.S. coins: Yes, coin chosen must be part of some optimal solution. We can prove it!
- For "unusual" sets of coins? We saw a counter-example.
- For U.S. postage stamps? Hmm…

More on Optimal Substructure Property

Detailed discussion in CLRS 14.3 (chapter on Dynamic Programming)

- If A is an optimal solution to a problem, then the components of A are optimal solutions to subproblems

Another example: Shortest Path in graph problem

- Say P is min-length path from CHO to LA and includes DAL
- Let P_1 be component of P from CHO to DAL, and P_2 be component of P from DAL to LA
- P_1 must be shortest path from CHO to DAL, and P_2 must be shortest path from DAL to LA
- Why is this true? Can you prove it? Yes, by contradiction. (Try this at home!)

Correctness of Greedy Algorithm

Optimal solution must satisfy following properties:

- At most 4 pennies
- At most 1 nickel
- At most 2 dimes
- Cannot contain 2 dimes and 1 nickel

Correctness of Greedy Algorithm

Claim: argue that at every step, greedy choice is part of some optimal solution

Case 1: Suppose $x < 5$

- Optimal solution must contain a penny (no other option available)
- **Greedy choice:** penny

Case 2: Suppose $5 \leq x < 10$

- Optimal solution must contain a nickel
  - Suppose otherwise. Then optimal solution can only contain pennies (there are no other options), so it must contain $x > 4$ pennies (contradiction)
- **Greedy choice:** nickel

Case 3: Suppose $10 \leq x < 25$

- Optimal solution must contain a dime
  - Suppose otherwise. By construction, the optimal solution can contain at most 1 nickel, so there must be at least 6 pennies in the optimal solution (contradiction)
- **Greedy choice:** dime

Correctness of Greedy Algorithm

Claim: argue that at every step, greedy choice is part of some optimal solution

Case 4: Suppose $25 \leq x$

- Optimal solution must contain a quarter
  - Suppose otherwise. There are two possibilities for the optimal solution:
    - If it contains 2 dimes, then it can contain 0 nickels, in which case it contains at least 5 pennies (contradiction)
    - If it contains fewer than 2 dimes, then it can contain at most 1 nickel, so it must also contain at least 10 pennies (contradiction)
- **Greedy choice:** quarter

Conclusion: in every case, the greedy choice is consistent with some optimal solution

Correctness of Greedy Algorithm

What about that 11-cent coin, the “tom”? How’s that break this proof?

Claim: argue that at every step, greedy choic

Case 1: SupposeSuppose otherwise. Then optimal solution can only contain pennies (ther

- **Greedy choice:** nicke

This argument no longer holds. Sometimes, it’s better to take the dime; other times, it’s better to take the 11-cent piece.
For 15: 1 tom + 4 pennies vs. 1 dime + 1 nickel.
For 12: 1 tom + 1 penny vs. 1 dime + 2 pennies

Revised Case 3: Suppose $11 \leq x < 25$

- Optimal solution must contain a ~~dime~~ tom
  - Suppose otherwise. By construction, the optimal solution can contain at most 1 nickel, so there must be at least 6 pennies in the optimal solution (contradiction).
- **Greedy choice:** ~~dime~~ tom

Wrap-up on Greedy basics

An approach to solving ***optimization problems***

- Finds ***optimal solution*** among set of ***feasible solutions***

Works in stages, applying ***greedy choice*** at each stage

- Makes locally optimal choice, with goal of reaching overall optimal solution for entire problem

Proof needed to show correctness

Remember: Problem must have ***optimal substructure property***

- This will also be true for problems solved by ***dynamic programming***

Interval Scheduling

CLRS Section 15.1

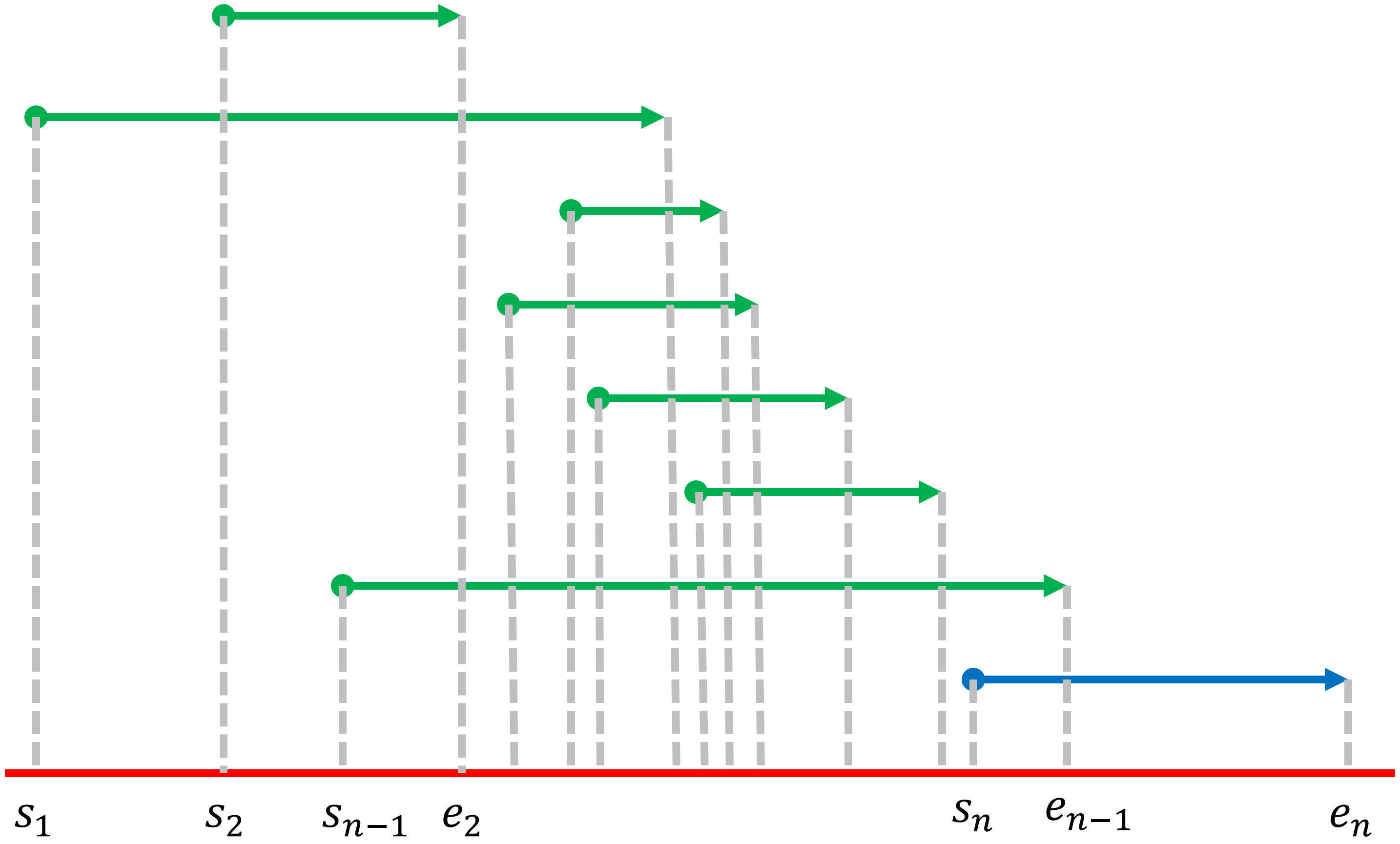
Interval Scheduling

Input: List of events with their start and end times (sorted by end time)

Output: largest set of non-conflicting events (start time of each event is after the end time of all preceding events)

[1, 2.25]	Lunch with friends at Roots
[2, 3:30]	CS3100 Office Hours
[3, 4]	Streaming CS department talk
[4, 5.25]	Afternoon Tea
[4.5, 6]	Discussion section
[5, 7.5]	Super Smash Brothers game night
[7.75, 11]	UVA Basketball watch party

Interval Scheduling Overview



Greedy Interval Scheduling

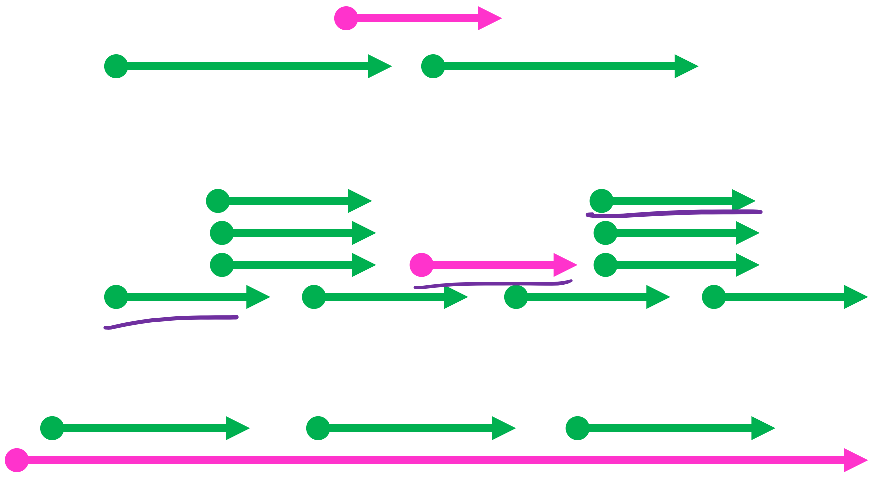
Step 1: Identify a greedy choice property

- one that ends first
- shortest duration
- least overlap
- one that starts first

Greedy Interval Scheduling

Step 1: Identify a greedy choice property

- Options:
  - Shortest interval
  - Fewest conflicts
  - Earliest start
  - Earliest end



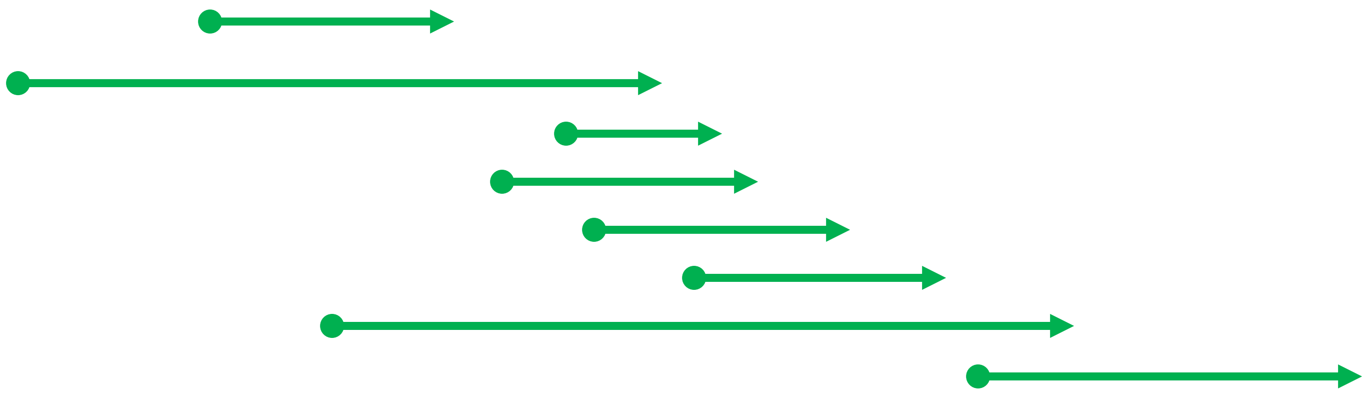
✓ Prove using Exchange Argument

Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

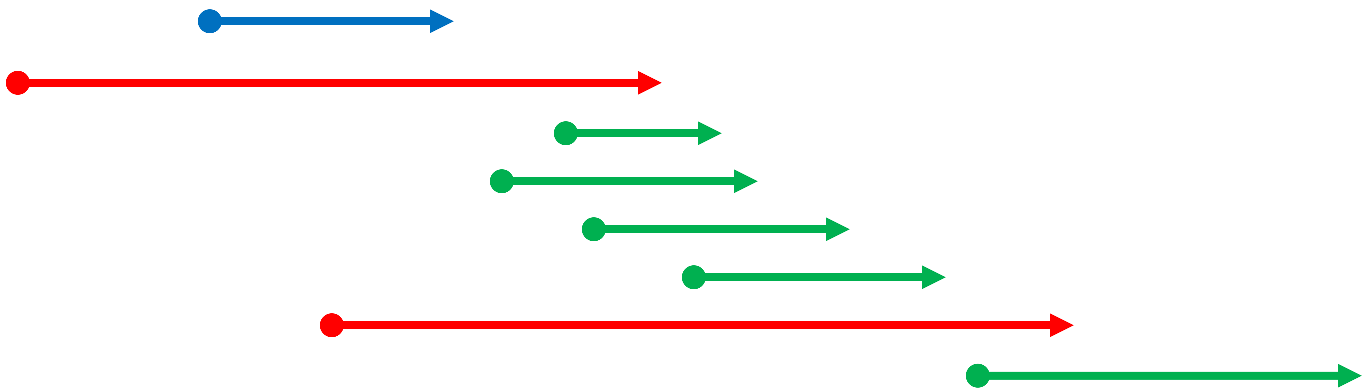


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

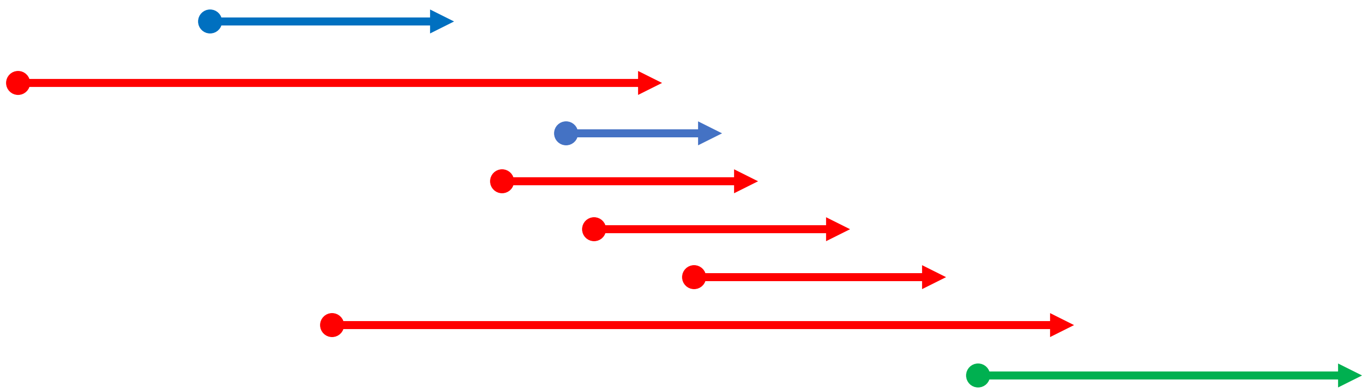


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

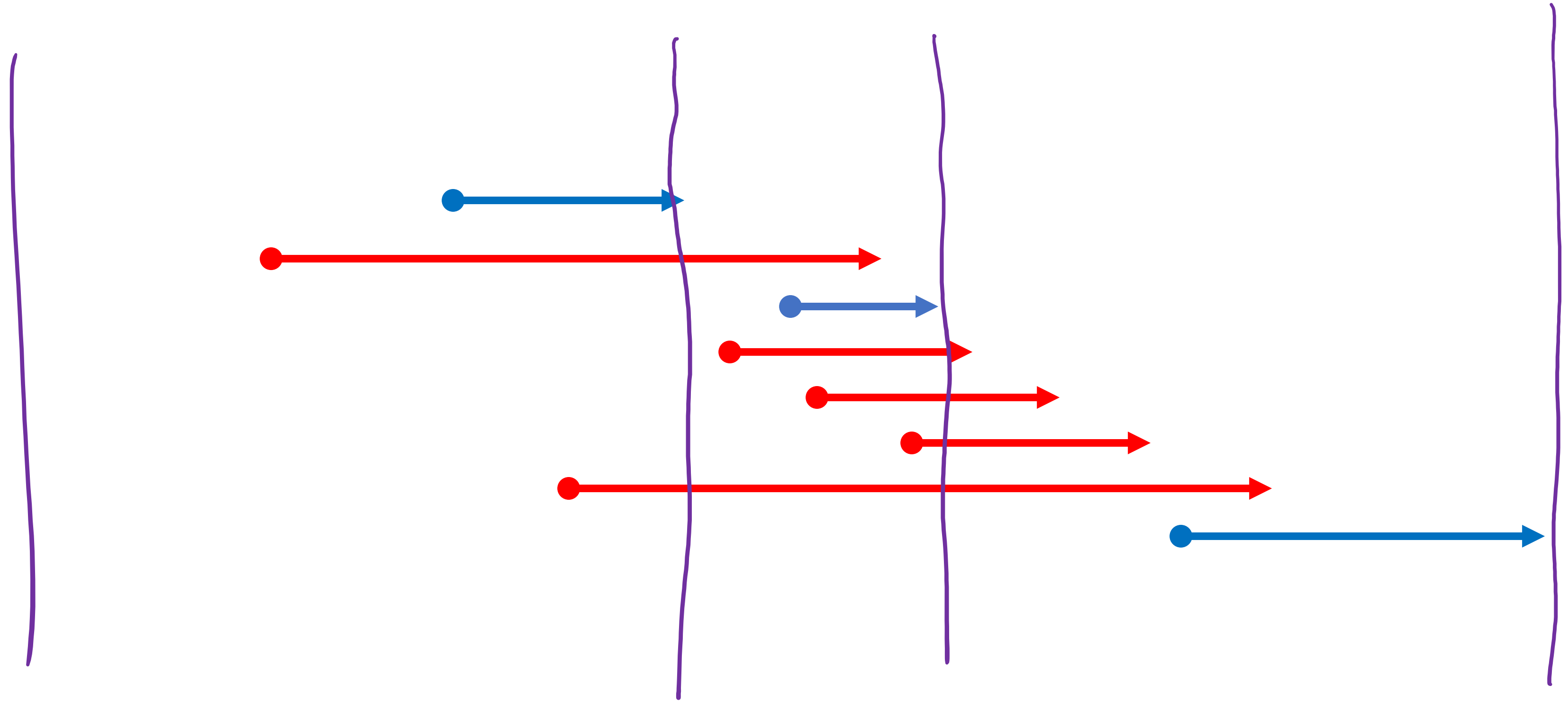


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution



Interval Scheduling Run Time

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

```
Sort intervals by finish time

StartTime = 0
for each interval (in order of finish time):
    if begin of interval > StartTime:
        add interval to solution
        StartTime = end of interval
```

Exchange argument

Shows correctness of a greedy algorithm

Idea:

- Show exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse
- How to show my sandwich is at least as good as yours:
  - Show: “I can remove any item from your sandwich, and it would be no worse by replacing it with the same item from my sandwich”

Exchange Argument for Earliest End Time

Exchange Argument for Earliest End Time

Claim: earliest ending interval is always part of some optimal solution

Let $OPT_{i,j}$ be an optimal solution for time range $[i,j]$

Let a^* be the first interval in $[i,j]$ to finish overall

case 1: If $a^* \in OPT_{i,j}$ then claim holds ✓

Else if $a^* \notin OPT_{i,j}$, let a be the first interval to end in $OPT_{i,j}$

- By definition a^* ends before a, and therefore does not conflict with any other events in $OPT_{i,j}$
- Therefore $OPT_{i,j} - \{a\} + \{a^*\}$ is also an optimal solution
- Thus claim holds ✓