CS 4102: Algorithms

Lecture 15: Greedy Algorithms

David Wu

Fall 2019

Today's Keywords

Greedy Algorithms

Choice Function

Change Making

Interval Scheduling

Exchange Argument

CLRS Readings: Chapter 16

Homework

HW5 due **Thursday, October 24**, 11pm

- Seam Carving
- Dynamic Programming (implementation)
- Java or Python

HW6 released Thursday

- Dynamic programming and greedy algorithms
- Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

Previously…

Given access to an unlimited number of pennies, nickels dimes, and quarters, give an algorithm which gives change for a target value x using the fewest number of coins

Previously…

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

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

while $x > 0$:
    let $c = \max(c_i \in \{c_1, \ldots, c_n\} \mid c_i \leq x)$
    add c to list L
    $x = x - c$
output L

Warm-Up

Suppose we added a new coin worth 11 cents

Give an efficient algorithm to find the minimum number of coins needed to give change for n cents

Greedy Algorithm Does Not Work

90 cents

Greedy Algorithm Does Not Work

90 cents

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the "last thing" done?
2. Select a good order for solving subproblems
   - "Top Down:" Solve each problem recursively
   - "Bottom Up:" Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

Identify Recursive Structure

$\mathrm{Min}(n)$: minimum number of coins needed to give change for n cents

Possibilities for last coin	Coins needed	
	$\mathrm{Min}(n-25)+1$	if $n \geq 25$
	$\mathrm{Min}(n-11)+1$	if $n \geq 11$
	$\mathrm{Min}(n-10)+1$	if $n \geq 10$
	$\mathrm{Min}(n-5)+1$	if $n \geq 5$
	$\mathrm{Min}(n-1)+1$	if $n \geq 1$

Identify Recursive Structure

$\text{Min}(n)$: minimum number of coins needed to give change for n cents

$$\text{Min}(n) = \begin{cases} \text{Min}(n-25)+1 & \text{if } n \geq 25 \\ \text{Min}(n-11)+1 & \text{if } n \geq 11 \\ \text{Min}(n-10)+1 & \text{if } n \geq 10 \\ \text{Min}(n-5)+1 & \text{if } n \geq 5 \\ \text{Min}(n-1)+1 & \text{if } n \geq 1 \end{cases}$$

Correctness: The optimal solution must be contained in one of these configurations

Base Case: $\text{Min}(0) = 0$

Running time: $O(kn)$

k is number of possible coins

Is this efficient?

Input size is $O(k \log n)$

No, this is pseudo-polynomial time

The Greedy Approach

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

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

while $x > 0$:
    let $c = \max(c_i \in \{c_1, \ldots, c_n\} \mid c_i \leq x)$
    add c to list L
    $x = x - c$
output L

Observation: We can rewrite this to take $\lfloor n/c \rfloor$ copies of the largest coin at each step

Running time: $O(k \log n)$ **Polynomial-time!**

The Greedy Approach

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

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

while $x > 0$:
    let $c = \max(c_i \in \{c_1, \dots, c_n\} \mid c_i \leq x)$
    add c to list L
    $x = x - c$
output L

Observation: We can rewrite

Running time: $O(k \log n)$

Greedy approach: Only consider a single case/subproblem, which gives an asymptotically-better algorithm. When can we use the greedy approach?

Polynomial-time!

Greedy Algorithms

Requires optimal substructure

- Solution to larger problem contains the solution to a smaller one
- Only a single subproblem to consider

General Blueprint:

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

Greedy vs. Dynamic Programming

Dynamic Programming:

- Require optimal substructure
- Optimal choice can be one of multiple smaller subproblems

Greedy:

- Require optimal substructure
- Only a single choice and a single subproblem

Greedy Algorithms

Require optimal substructure

- Solution to larger problem contains the solution to a smaller one
- Only one subproblem to consider!

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

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 $n < 5$

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

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

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

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

- 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 5 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 $25 \leq n$

- 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

Where Does the Proof Break?

Suppose we added a new coin worth 11 cents

Give an efficient algorithm to find the minimum number of coins needed to give change for n cents

Correctness of Greedy Algorithm

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

Case 1: Suppose $n < 5$

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

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

- Optimal solution must contain a nickel
  - Suppose otherwise. Then optimal s… other options), so it must contain n…
- **Greedy choice:** nickel

This argument no longer holds. Sometimes, it's better to take the dime; other times, it's better to take the 11-cent piece.

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

- 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

Interval Scheduling

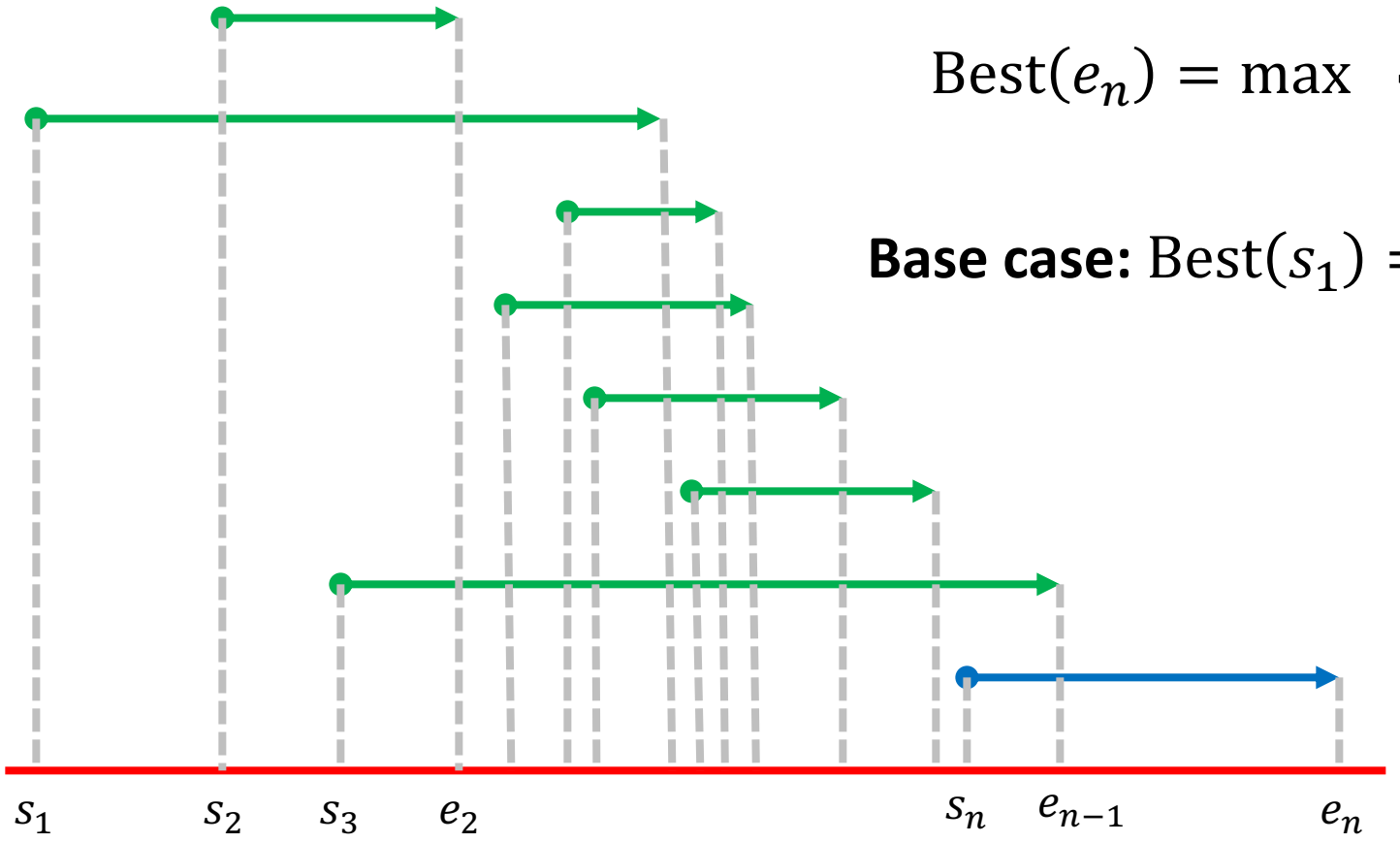
Input: List of events with their start and end times (sorted by end time)

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

[1:00, 2:15]	Alumni Lunch
[3:00, 4:00]	CHS Prom
[3:30, 4:45]	CS 4102
[4:00, 5:15]	Bingo
[4:30, 6:00]	SCUBA lessons
[5:00, 7:30]	Roller Derby Bout
[7:45, 11:00]	Football game

Dynamic Programming Interval Scheduling

$\text{Best}(t) =$ max number of events that can be scheduled before time t



Last action: include last event or exclude last event

$$\text{Best}(e_n) = \max \begin{cases} \text{Best}(s_n) + 1 & \text{Include event } n \\ \text{Best}(e_{n-1}) & \text{Exclude event } n \end{cases}$$

Base case: $\text{Best}(s_1) = 0$

Greedy Algorithms

Require optimal substructure

- Solution to larger problem contains the solution to a smaller one
- Only one subproblem to consider!

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 Algorithms

Require optimal substructure

- Solution to larger problem contains the solution to a smaller one
- Only one subproblem to consider!

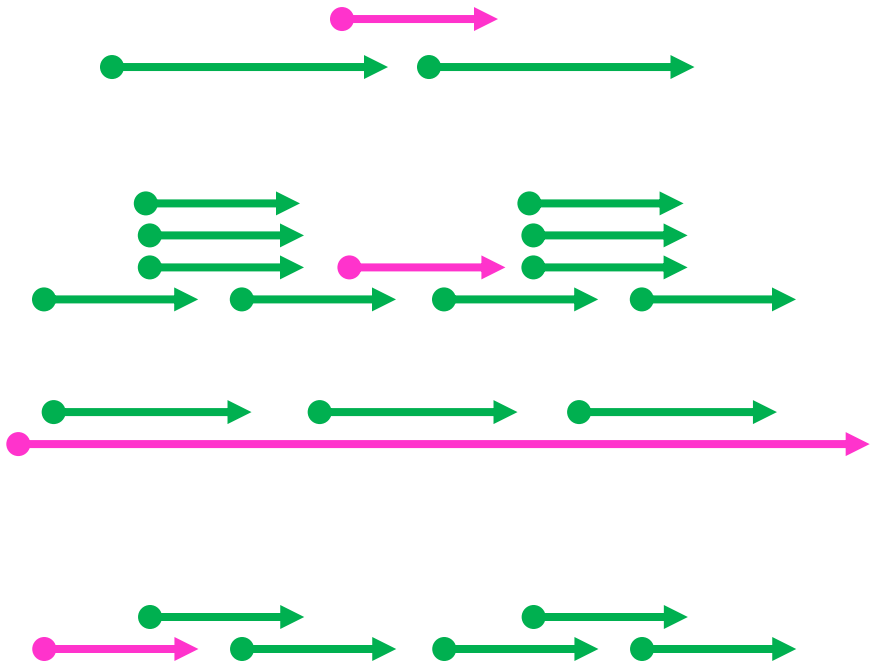
Idea:

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 Interval Scheduling

Step 1: Identify a greedy choice property

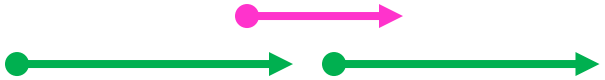
- Shortest interval
- Fewest conflicts
- Earliest start
- Earliest end



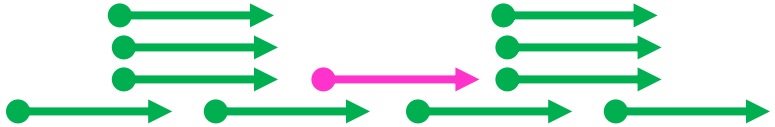
Greedy Interval Scheduling

Step 1: Identify a greedy choice property

- Shortest interval



- Fewest conflicts



- Earliest start

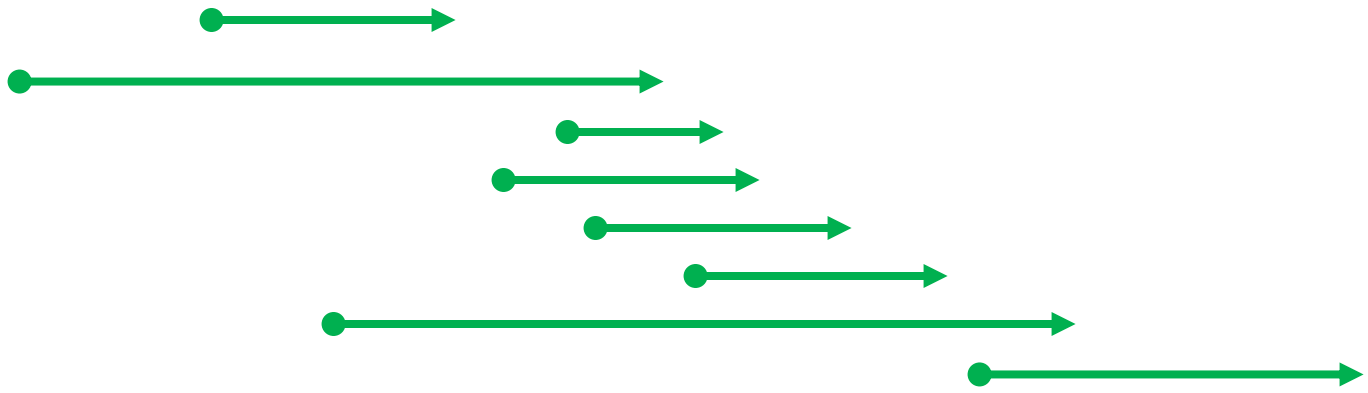
- Earliest end

Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

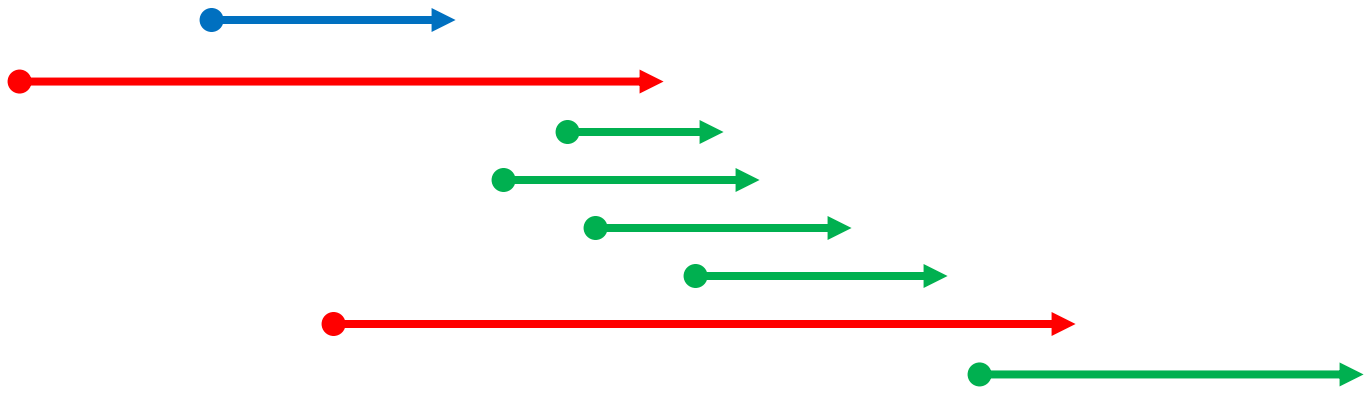


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

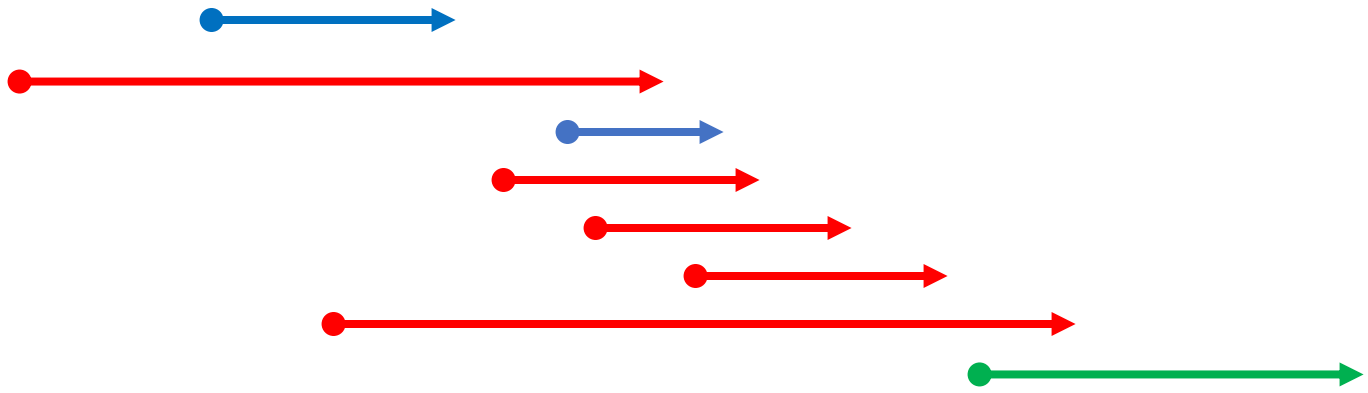


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

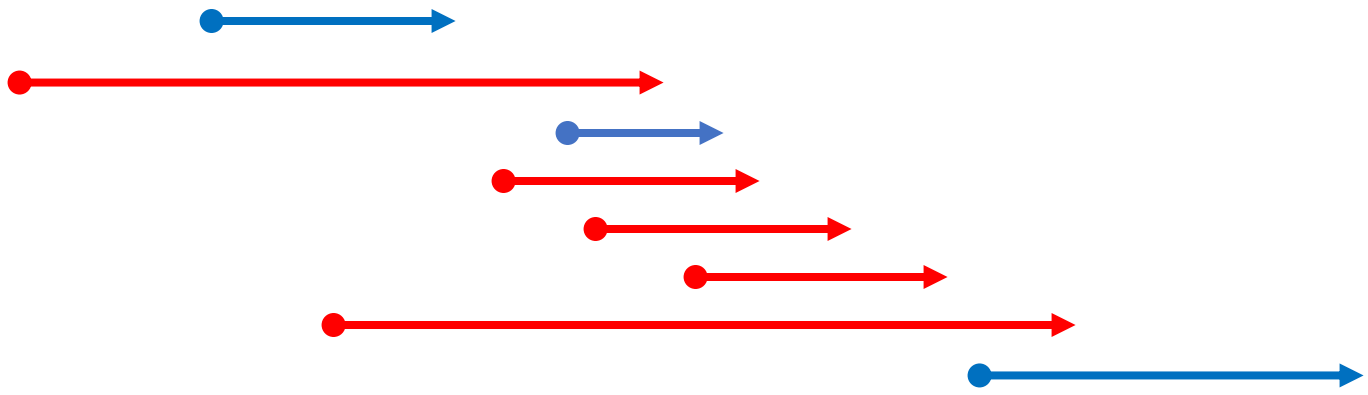


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution



Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

```
time = 0
for each i = 1,...,n:
      if start[i] < time:
            continue
      else:
            solution.add(i)
            time = end[i]
return solution
```

assume list sorted by interval end time

Running time: $O(n)$

Proof of Correctness: Exchange Argument

Common technique to show correctness of a greedy algorithm

General idea: argue that at every step, the greedy choice is part of some optimal solution

Approach: Start with an arbitrary optimal solution and show that exchanging an item from the optimal solution with your greedy choice makes the new solution no worse (i.e., the greedy choice is as good as the optimal choice)

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: $a^* \in OPT_{i,j}$

- Then Claim holds by definition

Case 2: $a^* \notin OPT_{i,j}$

- 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 optimal solution
- Then claim holds