CS4102 Algorithms

Fall 2019

Given access to unlimited quantities of pennies, nickels, dimes, and quarters (worth value 1, 5, 10, 25 respectively), provide an algorithm which gives change for a given value x using the fewest number of coins.

Change Making Algorithm

- 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 $c = \max(c_i \in \{c_1, \ldots, c_n\} \mid c_i \leq x)$
  print c
  $x = x - c$

Why does this always work?

- If $x < 5$, then pennies only
  - Else 5 pennies can be exchanged for a nickel

Only case Greedy uses pennies!

- If $5 \leq x < 10$ we must have a nickel
  - Else 2 nickels can be exchanged for a dime

Only case Greedy uses nickels!

- If $10 \leq x < 25$ we must have at least 1 dime
  - Else 3 dimes can be exchanged for a quarter and a nickel

Only case Greedy uses dimes!

- If $x \geq 25$ we must have at least 1 quarter

Only case Greedy uses quarters!

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 solution

90 cents

Greedy solution

90 cents

Today's Keywords

- Greedy Algorithms
- Choice Function
- Change Making
- Interval Scheduling
- Exchange Argument

CLRS Readings

- Chapter 16

Homeworks

- Homework 5 due Thursday at 11pm
  – Seam Carving!
  – Dynamic Programming (implementation)
  – Java or Python
- Homework 6 out Thursday
  – Dynamic Programming and Greedy Algorithms
  – Written (using Latex!)

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.

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

Identify Recursive Structure

Change(n): minimum number of coins needed to give change for n cents

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

Identify Recursive Structure

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

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

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

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

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

Greedy Change Making

- 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 $c = \max(c_i \in \{c_1, \ldots, c_n\} \mid c_i \leq x)$
  print c
  $x = x - c$

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!**

Greedy Change Making

- 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 $c = \max(c_i \in \{c_1, \dots, c_n\} \mid c_i \leq x)$
  print c
  $x = x$

Observation: We can rewrite

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

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

Greedy vs DP

- Dynamic Programming:
  - Require Optimal Substructure
  - Several choices for which small subproblem
- Greedy:
  - Require Optimal Substructure
  - Must only consider one choice for small subproblem

Log Cutting:
Maximum profit for each last cut

Longest Common Subsequence:
Max length with same last character or with one or the other

Seam Carving:
Min energy seam that could connect with this pixel

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

Change Making Choice Property

- Largest coin less than or equal to target value must be part of some optimal solution (for standard U.S. coins)

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 <u>some</u> optimal solution

- **Case 1:** Suppose $n < 5$
  - Optimal solution <u>must</u> contain a penny (no other option available)
  - **Greedy choice:** penny
- **Case 2:** Suppose $5 \leq n < 10$
  - Optimal solution <u>must</u> 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 <u>must</u> 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 $25 \leq n$
  - 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

Interval Scheduling

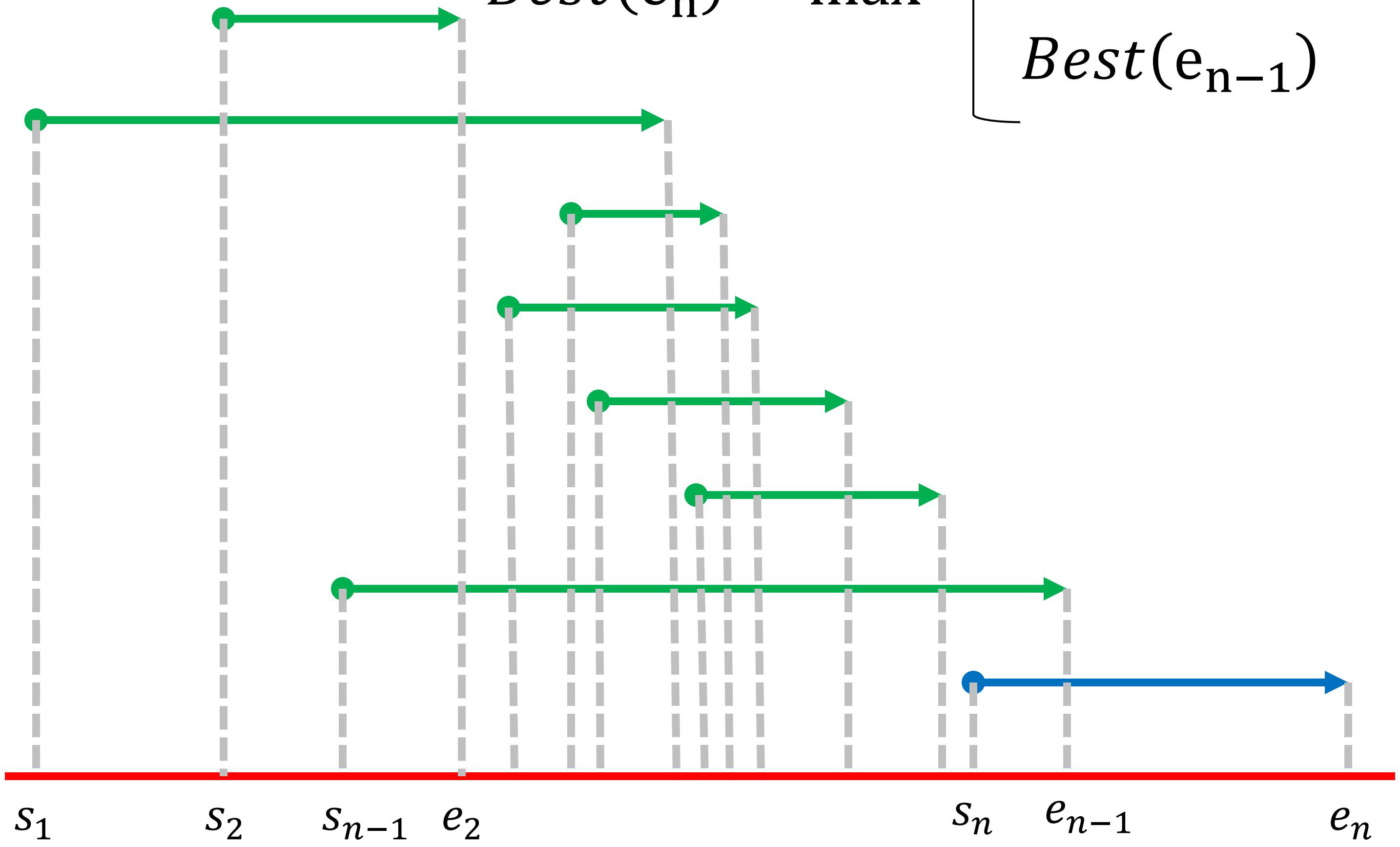
- Input: List of events with their start and end times (sorted by end time)
- Output: largest set of non-conflicting events (start time of each event is after the end time of all preceding events)

[1, 2.25]	Alumni Lunch
[2, 3.25]	CS4102
[3, 4]	CHS Prom
[4, 5.25]	Bingo
[4.5, 6]	SCUBA lessons
[5, 7.5]	Roller Derby Bout
[7.75, 11]	UVA Football watch party

Interval Scheduling DP

$Best(t)$ = max # events that can be scheduled before time t

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

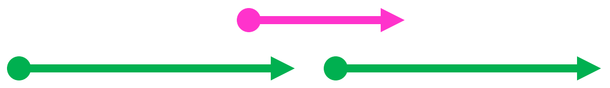


Greedy Interval Scheduling

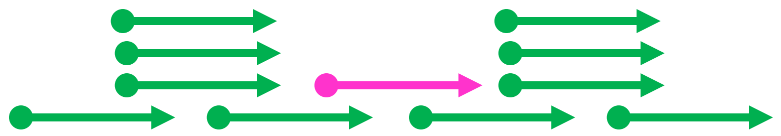
- Step 1: Identify a greedy choice property

Greedy Interval Scheduling

- Step 1: Identify a greedy choice property
  - Options:
    - Shortest interval



    - Fewest conflicts



    - Earliest start

    - Earliest end

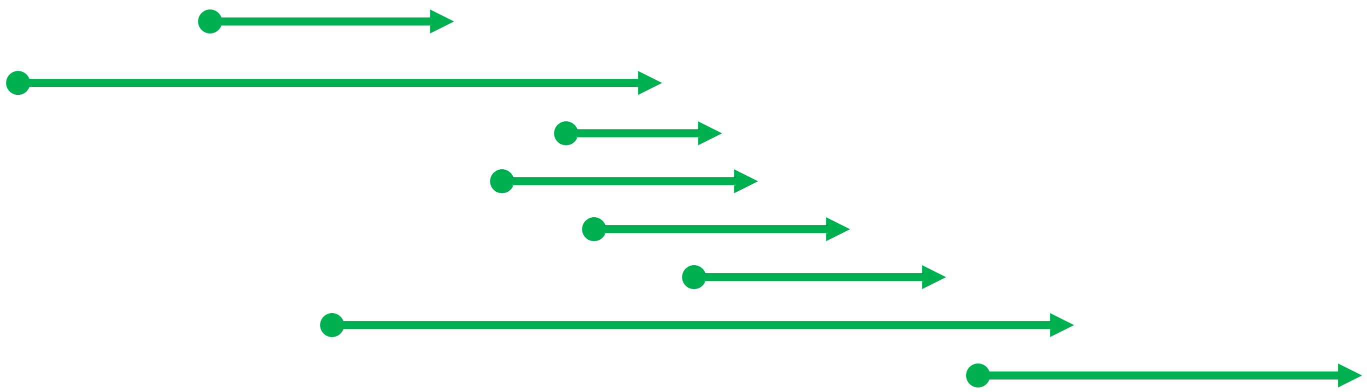
Prove using Exchange Argument

Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

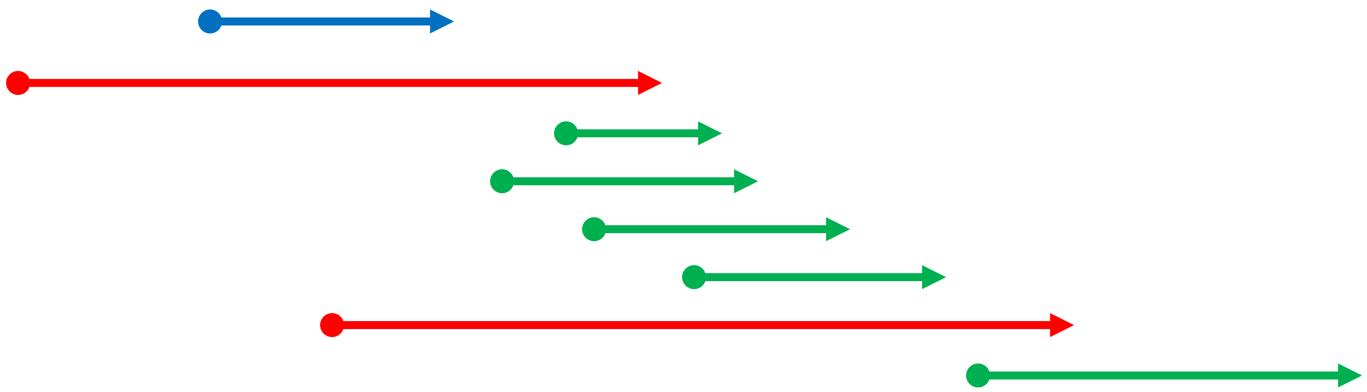


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

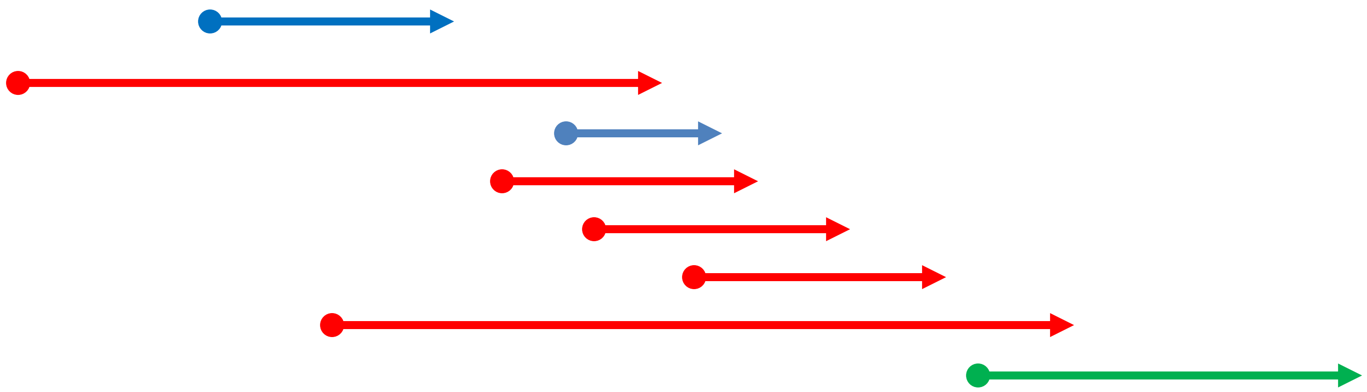


Interval Scheduling Algorithm

Find event ending earliest, add to solution,

Remove it and all conflicting events,

Repeat until all events removed, return solution

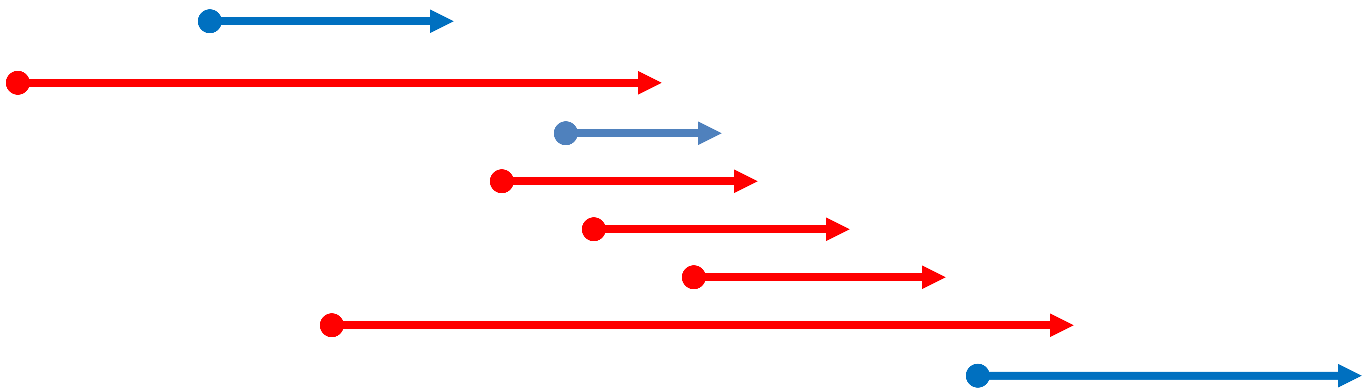


Interval Scheduling Algorithm

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

Equivalent way

StartTime = 0
For each interval (in order of finish time): $O(n)$
    if begin of interval < StartTime or end of interval < StartTime: $O(1)$
        do nothing
    else:
        add interval to solution $O(1)$
        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

- 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
- 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