CS4102 Algorithms

Spring 2020 – slides for both Hott and Horton’s sections

It’s time to “change” things up and start our unit on Greedy Algorithms! ☺

Where We're Going

- Terminology about optimization problems and greedy algorithms *(this video)*
- Example 1: Coin Changing *(this video)*
  - Contrast with dynamic programming approach
  - Proofs of correctness
- Example 2: Interval Scheduling *(next video)*
- …
- **Textbook readings:** CLRS Chapter 16 (go for it!)

Coin Changing

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 $C = [c_1, \ldots, c_k]$
  (in this case $C = [1, 5, 10, 25]$)
- Repeatedly select the largest coin less than the remaining target value:

while($x > 0$)
  let $c = \max(c_i \in \{c_1, \ldots, c_k\} \mid c_i \leq x)$
  print c
  $x = x - c$

Let's reflect on this

- What's its time-complexity?
  - 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
  - We'll come back to this.
- Does this 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.

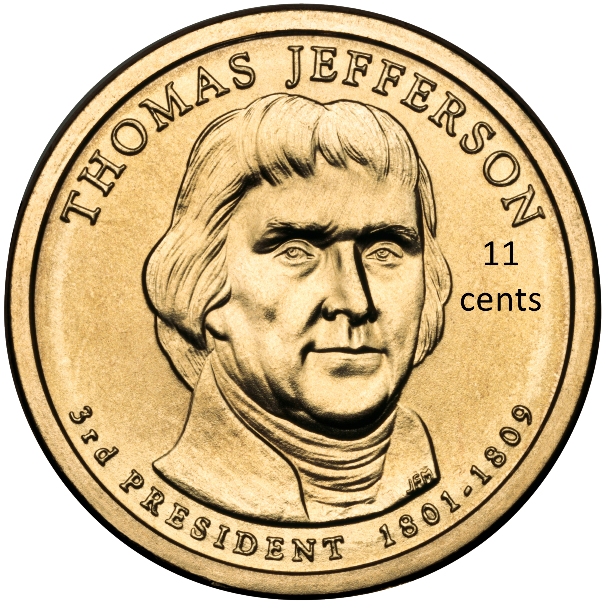
Back to Coin Changing: Correctness?

- 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.
  – 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, too.

Will that work for this set of coins?

Dynamic Programming

- Requires Optimal Substructure
  - – Optimal solution to a problem contains optimal solutions to subproblems
- 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

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

<u>Possibilities for last coin</u>	<u>Coins needed</u>	
	$\text{Change}(x-25)+1$	if $x \geq 25$
	$\text{Change}(x-11)+1$	if $x \geq 11$
	$\text{Change}(x-10)+1$	if $x \geq 10$
	$\text{Change}(x-5)+1$	if $x \geq 5$
	$\text{Change}(x-1)+1$	if $x \geq 1$

Of course we need to define a data structure to remember partial results and fill it in some order.
We won't address that here.

Identify Recursive Structure

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

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

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

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

Size of input x is how many bits to store x.
Size $n = \lg x$ so $x = 2^{\lg x}$, so $O(k\, 2^n)$

Running time: $O(kx)$

k is number of possible coins

Is this efficient? Isn't it polynomial?

No, this is pseudo-polynomial time

Greedy Change Making

- Given: target value x, list of coins $C = [c_1, \ldots, c_k]$
  (in this case $C = [1, 5, 10, 25]$)
- *Greedy choice:* Repeatedly select the largest coin less than the remaining target value:

while($x > 0$)
  let $c = \max(c_i \in \{c_1, \ldots, c_k\} \mid c_i \leq x)$
  print c
  $x = x - c$

Observation: We can rewrite this to take $\lfloor n/c \rfloor$ copies of the largest coin at each step. Then loop over values c_i from largest to smallest. Then if C is sorted….

Running time: $O(k)$

k is number of possible coins

Polynomial-time!

Greedy Change Making

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

Observation: We can rewrite t

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

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

k is number of possible coins

Polynomial-time! Size $n = \log x$

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
  - Optimal solution to a problem contains optimal solutions to subproblems
  - 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

- 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…

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 $x < 5$
  - Optimal solution <u>must</u> contain a penny (no other option available)
  - **Greedy choice:** penny
- **Case 2:** Suppose $5 \leq x < 10$
  - 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 $x > 4$ pennies (contradiction)
  - **Greedy choice:** nickel
- **Case 3:** Suppose $10 \leq x < 25$
  - 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 $25 \le x$
  - 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 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 co (contradiction)
  - **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.
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***
- Problem must have ***optimal substructure property***
- 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

Need more on Optimal Substructure Property?

- Detailed discussion on p. 379 of CLRS (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!)