CS4102 Algorithms

Spring 2021 – Floryan and Horton

Module 3, Day 4

- Proof of correctness for greedy coin change
  - (Solution not in textbook, is exercise in textbook)
- DP Solution to the Coin Change Problem
  - (Also given as an exercise in the textbook)

MAKING CHANGE

Everyone Already Knows Many Algorithms!

- Worked retail? You know how to make change!
- Example:
  - My item costs $4.37. I give you a five dollar bill. What do you give me in change?
  - Answer: two quarters, a dime, three pennies
  - Why? How do we figure that out?

Making Change

- The problem:
  - 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. (Do we have half-dollars? That affects the answer we give.)
- Output: a set of coins

- Note this problem statement is simply a transformation
  - Given input, generate output with certain properties
  - No statement about how to do it.
- Can you describe the algorithm you use?

Optimal Substructure

- This problem has ***optimal substructure***

- Claim (we will prove this):

- If $\mathrm{C} = \{c_1, c_2, \ldots, c_n\}$ is the optimal set of coins to make A cents of change:
- Then $C' = \{c_2, c_3, \ldots, c_n\}$ is the optimal set of coins to make $A - c_1$ cents of change.

A Change Algorithm

1. Consider the largest coin
2. How many go into the amount left?
3. Add that many of that coin to the output
4. Subtract the amount for those coins from the amount left to return
5. If the amount left is zero, done!
6. If not, consider next largest coin, and go back to Step 2

Evaluating Our Greedy Algorithm

- How much work does it do?
  - Say C is the amount of change, and N is the number of coins in our coin-set
  - Loop at most N times, and inside the loop we do:
    - A division
    - Add something to the output list
    - A subtraction, and a test
  - We say this is O(N), or linear in terms of the size of the coin-set
- Could we do better?
  - Is this an *optimal algorithm*?
  - We need to do a proof somehow to show this

Another Change Algorithm

- Give me another way to do this?

- Brute force:
  - Generate all possible combinations of coins that add up to the required amount
  - From these, choose the one with smallest number
- What would you say about this approach?

- There are other ways to solve this problem
  - *Dynamic programming*: build a table of solutions to small subproblems, work your way up

Algorithm for making change

- This algorithm makes change for an amount A using coins of denominations $denom[1] > denom[2] > \cdots > denom[n] = 1$.

- Input Parameters: *denom, A*
- Output Parameters: None

- ```
  greedy_coin_change(denom, A) {
      i = 1
      while (A > 0) {
          c = A / denom[i]
          println("use " + c + " coins of denomination " + denom[i])
          A = A - c * denom[i]
          i = i + 1
      }
  }
  ```
  ```

Making change proof

- Another methodology for proving correctness of greedy algorithms:

- A greedy algorithm is correct if the following hold:
  - The problem has ***optimal substructure***
  - The algorithm has the ***greedy choice property*** (see next slide)

Making change proof

- What is the ***greedy choice property***?
- Your algorithm makes some greedy choice and then continues
  - e.g., choose largest coin, then continue
- Prove that the ***one thing*** the greedy algorithm selects MUST be in some optimal solution to the problem.

Making change proof

- Proving the ***greedy choice property***?

- Claim: For making A cents of change, some optimal solution MUST contain the largest coin such that $c_i \leq A$

Proof

- Overview of proof:
  - Assume largest coin NOT in some optimal solution
  - Ok, some other coins must be in there instead.
  - 4 Cases:
    - Largest coin that fits is penny (1 cent) //this one is trivial though!
    - Largest coin that fits is nickel (5 cent)
    - Largest coin that fits is dime (10 cent)
    - Largest coin that fits is quarter (25 cent)

Proof

- Largest coin that fits is penny (1 cent) //this one is trivial though!
  - means A < 5
  - Only penny fits, so penny must be in some optimal solution!

- Largest coin that fits is nickel (5 cent)
  - Assume nickel not in optimal solution. Note A >= 5
  - Pennies are only other option, so 5 or more pennies in optimal solution
  - But I can swap out 5 of those pennies with a nickel
    - Solution decreases by 4 coins!! Contradiction!!

Proof

- Largest coin that fits is Dime (10 cent)
  - Assume dime not in optimal solution. Note A >= 10 and A < 25
  - So the optimal solution contains:
    - >= 2 nickels, some number of pennies (might be 0)
    - 1 nickel, some pennies (at least 5)
    - all pennies (more than 10)
  - In each case above, I can swap a dime in for some combination of nickels or pennies
    - Solution decreases by 1, 5, or 9 coins respectively. Contradiction!

- Largest coin that fits is quarter (25 cent)
  - Assume quarter not in optimal solution. Note A >= 25
  - So the optimal solution contains:

Proof

- Largest coin that fits is quarter (25 cent)
  - Assume quarter not in optimal solution. Note A >= 25
  - So the optimal solution contains:
    - 2 dimes, 1 nickel, some pennies maybe
    - 2 dimes, 0 nickels, 5 or more pennies
    - 1 dime, 3 nickels, 0 or more pennies
    - 1 dime, 2 nickels, 5 or more pennies
    - 1 dime, 1 nickel, 10 or more pennies
    - 1 dime, 0 nickel, 15 or more pennies
    - 0 dime, 5 nickels, 0 or more pennies
    - …
- For each case above, a quarter can be swapped back in for more than 1 coin to make the solution better!! Contradiction!

How would a failed proof work?

- Prove greedy choice property for denominations 1, 6, and 10
- This is going to fail because the algorithm doesn't work. Let's see it!
  - For A = 12, greedy outputs 10,1,1
  - Best answer is 6,6

How would a failed proof work?

- Largest coin that fits is Dime (10 cent)
  - Assume dime not in optimal solution. Note A >= 10
  - So the optimal solution contains:
    - 2 or more six-cent coins, pennies maybe (could be 0)
    - 1 six-cent coin, at least 4 pennies
    - 0 six-cent coins, at least 10 pennies

- For the second two, we can do the exchange, but NOT for the first one. The proof doesn't work!!

DYNAMIC PROGRAMMING: MAKING CHANGE

Recall coin change

- Given an amount A, pick the minimum amount of coins that yield A

- Recall the greedy algorithm:
  - Use the biggest coin possible, and use as many of those as possible
  - Repeat with successively smaller denominations

- This algorithm won't work with non-standard denominations

Greedy algorithm

- Given coin cent amounts of 10, 6, 5, and 1
- Compute the coins needed for 12 cents
  - The greedy algorithm picks {10, 1, 1}
  - But {6, 6} is more optimal (fewer coins)

Definitions

- We define an array denom which holds the denominations of the coins such that:
  - denom[1] > denom[2] > … > denom[n] = 1
  - In other words, we sort the coin denominations in decreasing order, ending with a penny
- We are obtaining change for an amount A
- Consider the i,j problem:
  - The available denominations are denom[i] through denom[n], where i ≥ 1 (i.e. the smaller n-i+1 coins)
    - Note: when i is large, you're working with fewer types of coins, and when i=1 you're working with your complete set
  - The amount we are looking for is j, where j ≤ A (i.e. the remaining amount of money)

The i,j problem

- Consider the i,j problem: (Remember, i is which coins, and j is the amount)
  - The available denominations are denom[i] through denom[n], where i ≥ 1 (i.e. the smaller n-i+1 coins)
  - The amount we are looking for is j, where j ≤ A (i.e. the remaining amount of money)
- Given coins of denominations 10, 6, and 1, here's the table showing how to create change up to 12 cents:

Our answer!

j (the amount)

		0	1	2	3	4	5	6	7	8	9	10	11	12	
i	**1**	0	1	2	3	4	5	1	2	3	4	1	2	2	Can use 1, 6 & 10
	2	0	1	2	3	4	5	1	2	3	4	5	6	2	Can use 1 & 6
	3	0	1	2	3	4	5	6	7	8	9	10	11	12	Can use 1

Solving the problem

- How to solve the i,j problem (Remember, i is which coins, and j is the amount)
  - If denom[i] > j, then not possible to include this coin
    - Then the solution is the same as the (i+1),j problem (same amount, but with one fewer of the coin-options)
    - In the table, that's the cell right below the current cell.
    - Is this making the problem simpler?
  - Maybe the best answer does use a coin of denomination i
    - Then the solution is 1 more than the i,(j-denom[i]) problem
    - j changes to j-denom[i] because we subtract off the value of the coin used
    - i doesn't change because there could be multiple coins of denomination i used in the solution
  - Maybe the best answer does NOT use a coin of denomination i
    - Then the solution is the same as the (i+1),j problem
    - In the table, that's the cell right below the current cell

The formulaic solution

- The solution becomes:

$$C[i][j] = \begin{cases} C[i+1][j] & \text{if } denom[i] > j \\ \min\left(C[i+1][j], 1 + C[i][j - denom[i]]\right) & \text{if } denom[i] \leq j \end{cases}$$

- Where C[i][0] = 0 for all values of i

- If we have a penny, then C[n][j] = j
  - This is required to get all amounts, so we assume a penny is the smallest denomination

Recursive solution

- The solution:

$$C[i][j] = \begin{cases} C[i+1][j] & \text{if } denom[i] > j \\ \min\left(C[i+1][j], 1 + C[i][j - denom[i]]\right) & \text{if } denom[i] \leq j \end{cases}$$

- Note that a given problem (C[i][j]) is expressed in terms of sub-problems

- We can write a solution now using memorization with a top-down solution (recursive calls), or a bottom-up approach (build a table)

The bottom-up algorithm

```
dynamic_coin_change1 (denom, A, C) {
    n = denom.last
    for j = 0 to A
            C[n][j] = j
    for i = n-1 down to 1
            for j = 0 to A
                    if ( denom[j] > j | |
                        C[i+1][j] < 1 + C[i][j-denom[i]] )
                            C[i][j] = C[i+1][j]
                    else
                            C[i][j] = 1 + C[i][j-denom[i]]
}
```

Time complexity?

Constant time to file each cell in the table. So $\Theta(n \cdot A)$ where n is the number of coins and A is the amount

But how to get the coins chosen?

- It's easy to trace back through the values
- Or, we could keep a *used* Boolean array
  - If used[i][j] is true, then the solution for i,j does use a coin of denom[i] for amount j
  - If false, it does not

j

i	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	F	F	F	F	F	F	F	F	F	F	T	T	F	Can use 1, 6 & 10
2	F	F	F	F	F	F	T	T	T	T	T	T	T	Can use 1 & 6
3	F	T	T	T	T	T	T	T	T	T	T	T	T	Can use 1

Recording the answers

```
dynamic_coin_change2 (denom, A, C, used) {
    n = denom.last
    for j = 0 to A
            C[n][j] = j
            used[n][j] = true
    for i = n-1 downto 1
            for j = 0 to A
                    if ( denom[j] > j ||
                        C[i+1][j] < 1+C[i][j-denom[i]] )
                            C[i][j] = C[i+1][j]
                            used[i][j] = false
                    else
                            C[i][j] = 1 + C[i][j-denom[i]]
                            used[i][j] = true
}
```

Obtaining the coin set

```
optimal_coins_set (i, j, denom, used) {
  if ( j == 0 )
      return
  if ( used[i][j] )
      println ("Use coin of denomination " + denom[i])
      optimal_coins_set (i, j-denom[i], denom, used)
  else
       optimal_coins_set (i+1, j, denom, used)
}
```