Algorithms:
Dynamic Programming

Coin Change Problem

Coin Change Problem

Given unlimited amounts of coins of denominations $c_1 > \ldots > c_d$, give change for amount M with the least number of coins.

Example: $c_1 = 25$, $c_2 = 10$, $c_3 = 5$, $c_4 = 1$ and $M = 48$

Greedy solution: $25*1 + 10*2 + 1*3 = c_1 + 2c_2 + 3c_4$

Greedy solution is

- optimal for any amount and "normal'' set of denominations
- may not be optimal for arbitrary coin denominations

Greedy Choice Principles

- Suppose you want to count out a certain amount of money, using the fewest possible coins.
- At each step, take the largest possible coin that does not overshoot.

- Example: To make Tk. 157/-, you,
  - Choose a Tk. 100/- note,
  - Choose a Tk. 50/- note,
  - Choose a Tk. 5/- coin,
  - Choose a Tk. 2/- coin.

Greedy Choice Principles: Failure

- To find the minimum number of US coins to make any amount, the greedy method always works
  - At each step, just choose the largest coin that does not overshoot the desired amount: 31¢ = (25+5+1)
- The greedy method would not work if we did not have 5¢ coins
  - For 31 cents, the greedy method gives seven coins (25+1+1+1+1+1+1), but we can do it with four (10+10+10+1)
- The greedy method also would not work if we had a 21¢ coin
  - For 63 cents, the greedy method gives six coins (25+25+10+1+1+1), but we can do it with three (21+21+21)
- The greedy algorithm results in a solution, but always not in an optimal solution
- How can we find the minimum number of coins for any given coin set?

Coin Change Problem: Example

Given the denominations 1, 3, and 5, what is the minimum number of coins needed to make change for a given value?

Value	1	2	3	4	5	6	7	8	9	10
Min # of coins	**1**		**1**		**1**					

Only one coin is needed to make change for the values 1, 3, and 5

Coin Change Problem: Example

Given the denominations 1, 3, and 5, what is the minimum number of coins needed to make change for a given value?

Value	1	2	3	4	5	6	7	8	9	10
Min # of coins	**1**	**2**	**1**	**2**	**1**	**2**		**2**		**2**

However, two coins are needed to make change for the values 2, 4, 6, 8, and 10.

Coin Change Problem: Example

Given the denominations 1, 3, and 5, what is the minimum number of coins needed to make change for a given value?

Value	1	2	3	4	5	6	7	8	9	10
Min # of coins	**1**	**2**	**1**	**2**	**1**	**2**	**3**	**2**	**3**	**2**

Lastly, three coins are needed to make change for the values 7 and 9

Coin Change Problem: Recurrence

This example is expressed by the following recurrence relation:

$$\text{minNumCoins}(M) = \min_{\text{of}} \begin{cases} \text{minNumCoins}(M-1) + 1 \\ \text{minNumCoins}(M-3) + 1 \\ \text{minNumCoins}(M-5) + 1 \end{cases}$$

Coin Change Problem: Recurrence

Given the denominations **c**: c_1, c_2, …, $c_{\mathbf{d}}$, the recurrence relation is:

$$\text{minNumCoins}(M) = \textbf{min of} \begin{cases} \text{minNumCoins}(M-c_1) + 1 \\ \text{minNumCoins}(M-c_2) + 1 \\ \quad\quad\ldots \\ \text{minNumCoins}(M-c_d) + 1 \end{cases}$$

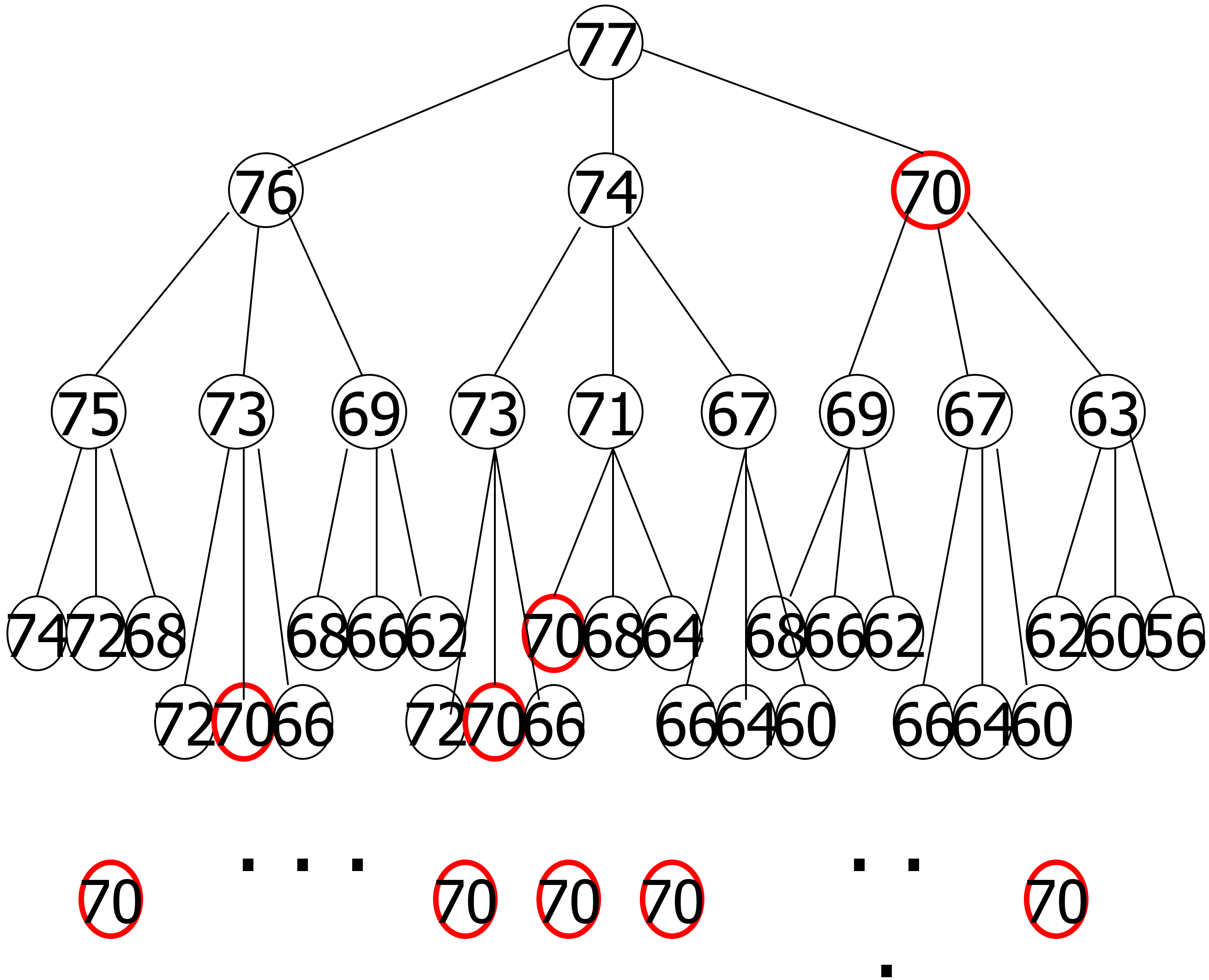
Coin Change Problem: A Recursive Algorithm

```
RecursiveChange(M, c, d)
  if M = 0
    return 0
  bestNumCoins ← infinity
  for i ← 1 to d
    if M ≥ c_i
      numCoins ← RecursiveChange(M – c_i , c, d)
        if numCoins + 1 < bestNumCoins
          bestNumCoins ← numCoins + 1
  return bestNumCoins
```

RecursiveChange is not Efficient

- It recalculates the optimal coin combination for a given amount of money repeatedly
- i.e., $\boldsymbol{M} = 77$, $\boldsymbol{c} = (1, 3, 7)$:
  - Optimal coin for 70 cents is computed **9** times!

The RecursiveChange Tree



We Can Do Better

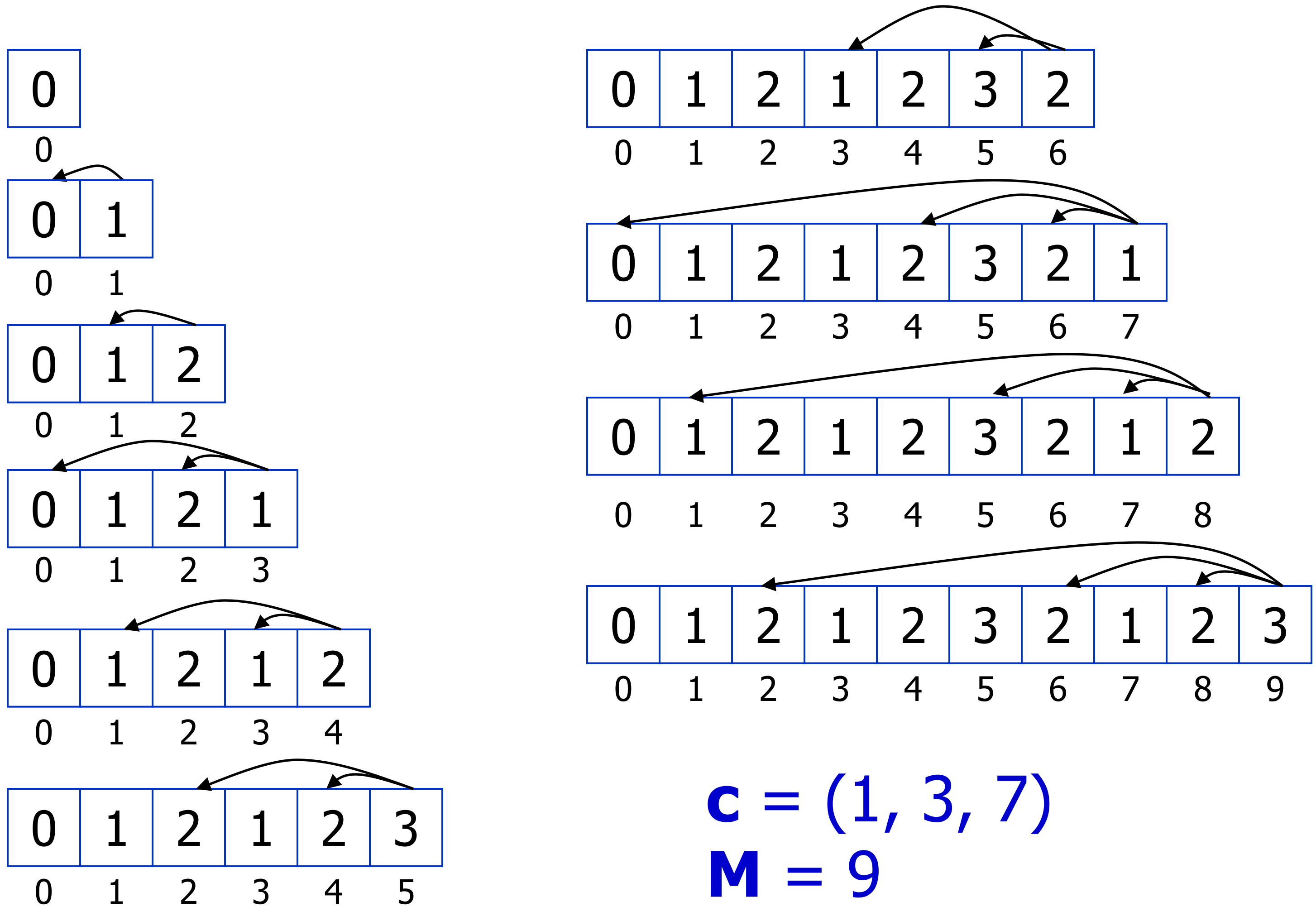
- We are re-computing values in our algorithm more than once
- Save results of each computation for 0 to M
- This way, we can do a reference call to find an already computed value, instead of re-computing each time
- Running time $M*d$, where M is the value of money and d is the number of denominations

Coin Change Problem: Dynamic Programming

```
DPChange(M, c, d)
  bestNumCoins_0 ← 0
  for m ← 1 to M
    bestNumCoins_m ← infinity
    for i ← 1 to d
      if m ≥ c_i
        if bestNumCoins_{m – c_i} + 1 < bestNumCoins_m
          bestNumCoins_m ← bestNumCoins_{m – c_i} + 1
  return bestNumCoins_M
```

Running time: $O(M*d)$

DPChange: Example



c = (1, 3, 7)

M = 9

Coin Change Problem

Goal: Convert some amount of money $\boldsymbol{M}$ into given denominations, using the fewest possible number of coins

Input: An amount of money $\boldsymbol{M}$, and an array of $\boldsymbol{d}$ denominations $\boldsymbol{c} = (c_1, c_2, \ldots, c_d)$, in a decreasing order of value ($c_1 > c_2 > \ldots > c_d$)

Output: A list of d integers $i_1, i_2, \ldots, i_d$ such that

$c_1 i_1 + c_2 i_2 + \ldots + c_d i_d = \boldsymbol{M}$ and $i_1 + i_2 + \ldots + i_d$ is minimal