AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

22BIO211: Intelligence of Biological Systems - 2

INTRODUCTION TO DYNAMIC PROGRAMMING

Dr. Manjusha Nair M
Amrita School of Computing, Amritapuri

Email : manjushanair@am.amrita.edu
Contact No: 9447745519

The Change Problem

- How can a cashier make change using the fewest number of coins?
- Different currencies have different possible coin values, or denominations.
- The heuristic used by cashiers all over the world to make change, which we call GreedyChange, iteratively selects the largest coin denomination possible.

```
GreedyChange(money)
  change ← empty collection of coins
  while money > 0
    coin ← largest denomination that is less than or equal to money
    add a coin with denomination coin to the collection of coins change
    money ← money - coin
  return change
```

The Change Problem

- Does GreedyChange always return the minimum possible number of coins?
- GreedyChange is suboptimal for some denominations!
- Example
  - *Denomination - (120, 40, 30, 24, 20, 10, 5, 4, 1)*
  - *we want to change 48 units of currency*
  - *Greedychange selection-(40,5,1,1,1)*
  - *Minimum Selection - (24,24)*
- Since GreedyChange is incorrect, we need to come up with a different approach.

The Change Problem

- We can represent coins from *d* arbitrary denominations by an array of integers
  - *Coins* = $(coin_1, \ldots, coin_d)$
  - *where the values* $coin_i$ *are given in decreasing order.*
- Example : Consider the following denominations of coins
  - *Coins = (120, 40, 30, 24, 20, 10, 5, 4, 1)*
- We can represent another array of *d* positive integers ($change_1$, ..., $change_d$) such that
  - $coin_1 \cdot change_1 + \cdots + coin_d \cdot change_d = money$
- Example:
  - *Money = 48*
  - *Coins = (120, 40, 30, 24, 20, 10, 5, 4, 1)*
  - *Change = (0, 1, 0, 0, 0, 0, 0, 1, 0, 3) corresponding to (40,5,1,1,1)*
  - *or Change = (0, 0, 0, 2, 0, 0, 0, 0, 0) corresponding to (24,24)*

Recursive Solution to the Change Problem

- A recursive solution to this problem involves breaking down the problem into smaller subproblems and solving them recursively.
- For each coin, you have two choices: include the coin or exclude the coin.
- If you include the coin, reduce the amount by the coin's value and keep the number of coins the same.
  - *Because the money is decremented and the coin can be repeated as a choice*
- If you exclude the coin, keep the amount the same but reduce the number of coins considered by one.
  - *No effect on money and the coin can not be considered as a choice*

Recursive Solution to the Change Problem

- Let Coins = (5, 4, 1), then for a money m,

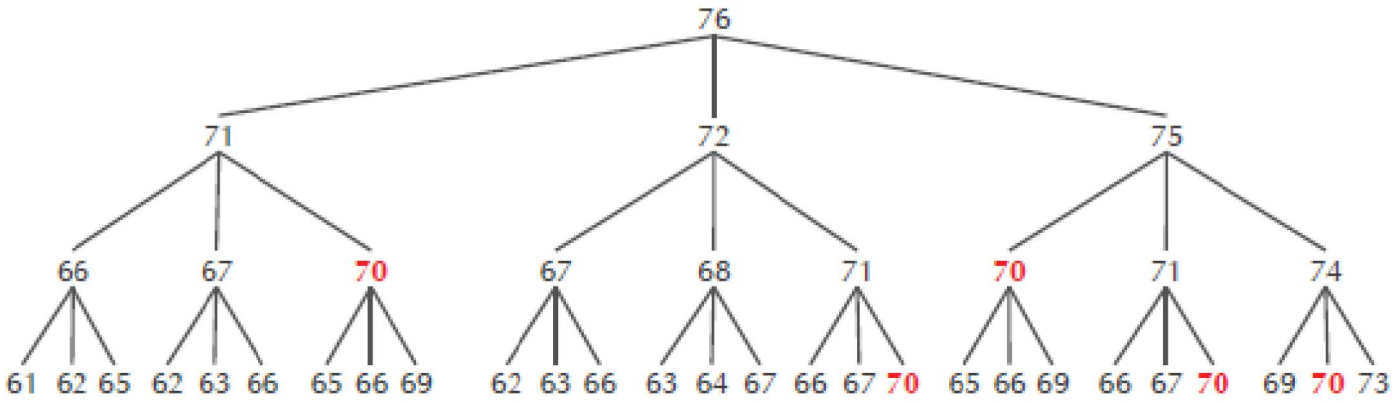
$$\text{MINNUMCOINS}(m) = \min \begin{cases} \text{MINNUMCOINS}(m-5)+1 \\ \text{MINNUMCOINS}(m-4)+1 \\ \text{MINNUMCOINS}(m-1)+1 \end{cases}$$

- We can form a recurrence relation from this
  - *For the general denominations Coins = ($coin_1$, … , $coin_d$), MinNumCoins(money) is equal to the minimum of d numbers:*

$$\text{MINNUMCOINS}(money) = \min \begin{cases} \text{MINNUMCOINS}(money - coin_1)+1 \\ \vdots \\ \text{MINNUMCOINS}(money - coin_d)+1 \end{cases}$$

Recursive Solution to the Change Problem

- Recursive algorithms are impractical because it recalculates the optimal coin combination for a given value of money over and over again.
  - *A tree illustrating Recursive change for money = 76 with Coins = (5, 4, 1).*



Recursive Solution to the Change Problem

- MinNumCoins(70) gets computed six times, five of which are shown in the figure .
- When MinNumCoins(70) is computed for the sixth time (corresponding to the path 76 → 75 → 74 → 73 → 72 → 71 → 70), RecursiveChange has already been called hundreds of times.
- MinNumCoins(30) will be computed billions of times!

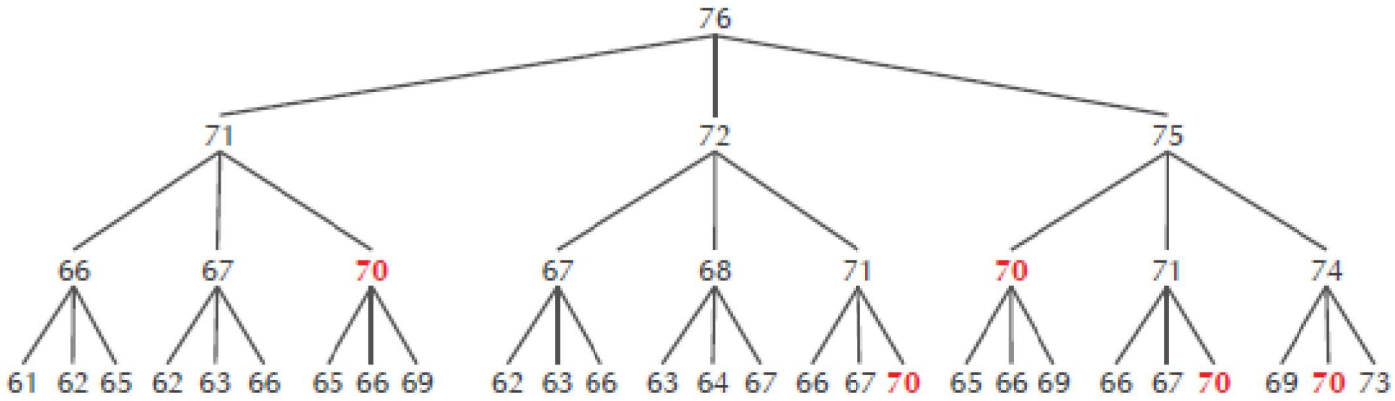
Recursive Solution to the Change Problem

```
RecursiveChange(money, Coins)
  if money = 0
    return 0
  MinNumCoins ← ∞
  for i ← 0 to |Coins| - 1
    if money ≥ coin_i
      NumCoins ← RecursiveChange(money - coin_i, Coins)
      if NumCoins + 1 < MinNumCoins
        MinNumCoins ← NumCoins + 1
  return MinNumCoins
```

Dynamic Programming solution to the Change Problem

- The change problem is the problem of finding the optimal distribution of change among a given amount of money and a set of coin denominations.
- Dynamic programming is an optimization technique used to solve complex problems by breaking them down into smaller, simpler subproblems.
- Dynamic programming can be applied to the change problem by using an array or table to store the solutions to subproblems.
- In the context of the change problem, dynamic programming is often used to determine the minimum number of coins required to make a given amount of change.

Dynamic Programming solution to the Change Problem



- Instead of computing *MinNumCoins(m)* for every value of *m* from 76 downward toward *m* = 0 via recursive calls,
  - *we will invert our thinking and compute MinNumCoins(m) from m = 0 upward toward 76,*
  - *storing all these values in an array so that we only need to compute MinNumCoins(m) once for each value of m.*

Dynamic Programming solution to the Change Problem

- MinNumCoins(m) is still computed via the same recurrence relation:

Coins = (5, 4, 1)

$$\text{MINNUMCOINS}(m) = \min \begin{cases} \text{MINNUMCOINS}(m-5)+1 \\ \text{MINNUMCOINS}(m-4)+1 \\ \text{MINNUMCOINS}(m-1)+1 \end{cases}$$

$$\text{MINNUMCOINS}(6) = \min \begin{cases} \text{MINNUMCOINS}(1)+1 = 2 \\ \text{MINNUMCOINS}(2)+1 = 3 \\ \text{MINNUMCOINS}(5)+1 = 2 \end{cases} = 2.$$

$$\text{MINNUMCOINS}(7) = \min \begin{cases} \text{MINNUMCOINS}(2)+1 = 3 \\ \text{MINNUMCOINS}(3)+1 = 4 \\ \text{MINNUMCOINS}(6)+1 = 3 \end{cases} = 3.$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12
MINNUMCOINS(m)	0	1	2	3	1	1	2	3	2	2	2	3	3

Dynamic Programming solution to the Change Problem

- Notice that MinNumCoins(2) is used in the computation of both MinNumCoins(6) and MinNumCoins(7)
  - *but instead of draining computational resources by having to compute this value from scratch both times, we can simply consult the pre-computed value in an array*

m	0	1	2	3	4	5	6	7	8	9	10	11	12
MINNUMCOINS(m)	0	1	2	3	1	1	2	3	2	2	2	3	3

- If money = 10^9, this algorithm requires a huge array of size 10^9.

Dynamic Programming solution to the Change Problem

```
DPChange(money, Coins)
  MinNumCoins(0) ← 0
  for m ← 1 to money
    MinNumCoins(m) ← ∞
      for i ← 0 to |Coins| - 1
        if m ≥ coin_i
          if MinNumCoins(m - coin_i) + 1 < MinNumCoins(m)
            MinNumCoins(m) ← MinNumCoins(m - coin_i) + 1
  output MinNumCoins(money)
```

Summary

- The Change Problem
- Recursive Solution to the Change Problem
- Dynamic Programming solution to the Change Problem