Design and Analysis of Algorithms

Lecture – 20

Dynamic Programming

Success is always inevitable with Hard Work and Perseverance

N. Ravitha Rajalakshmi

Learning Objective

- Learn the basics of dynamic programming

Change Problem

Find the minimum number of coins needed to make a change

Input: An integer money and positive integers $coin_1, coin_2, \ldots\ldots\ldots coin_d$

Output: The minimum number of coins with the available denominations that changes money

Greedy Strategy

- Example : Available denominations 1, 5, 10, 25 what is the minimum number of coins needed to make a change for 40

Greedy Choice : use the coin with maximum denomination

40 -> Select 25 -> 15 -> Select 10 -> 5 -> Select 5 -> 0

Function GreedyChange(money, coins)

Change – empty

```
while (money>0)
        coin = Find coin with largest denomination
                whose value is less than the money
        add coin to set Change
        money = money – coin

return Change
```

Greedy Strategy

- Example : Available denominations 1, 5, 10, 20, 25, 50 what is the minimum number of coins needed to make a change for 40

Greedy Choice : use the coin with maximum denomination

40 -> Select 25 -> 15 -> Select 10 -> 5 -> Select 5 -> 0

Optimal Solution is {20, 20}

Recursive Strategy

- Optimal solution for problem will be obtained from an optimal solution of subproblem

- How do we define problem instance

  Find Min number of coins for money x

Recursive Strategy

Available denominations 1, 5, 10, 20, 25, 50 what is the minimum number of coins needed to make a change for 40

Min number of coins for 40 , What are the options available?

1, 5, 10, 20, 25

Min number of coins for 40 =

Problem Instance

Min

Subproblem Instance

MinCoins (40-1) + 1

MinCoins (40-5) + 1

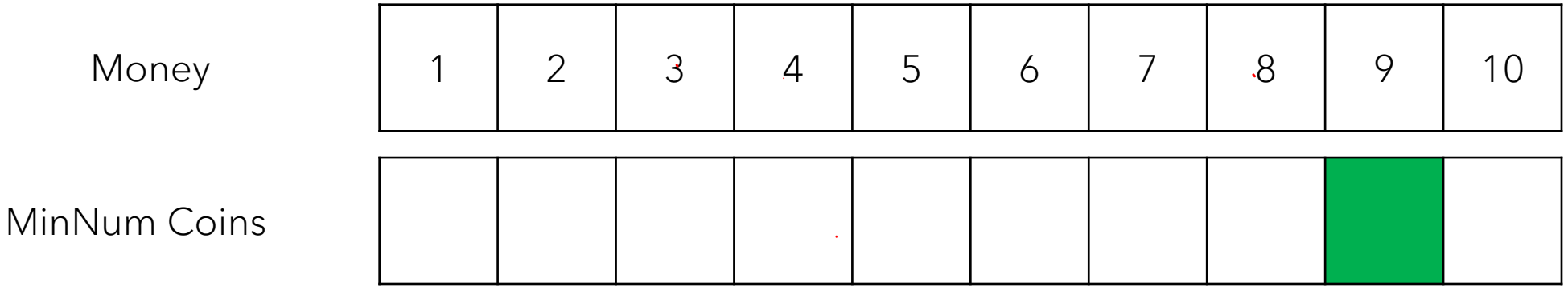
MinCoins (40-10) + 1

MinCoins (40-20) + 1

MinCoins (40-25) + 1

Recursive Strategy

- Given the denominations 6, 5, 1 What is the minimum number of coins needed to change 9 ?

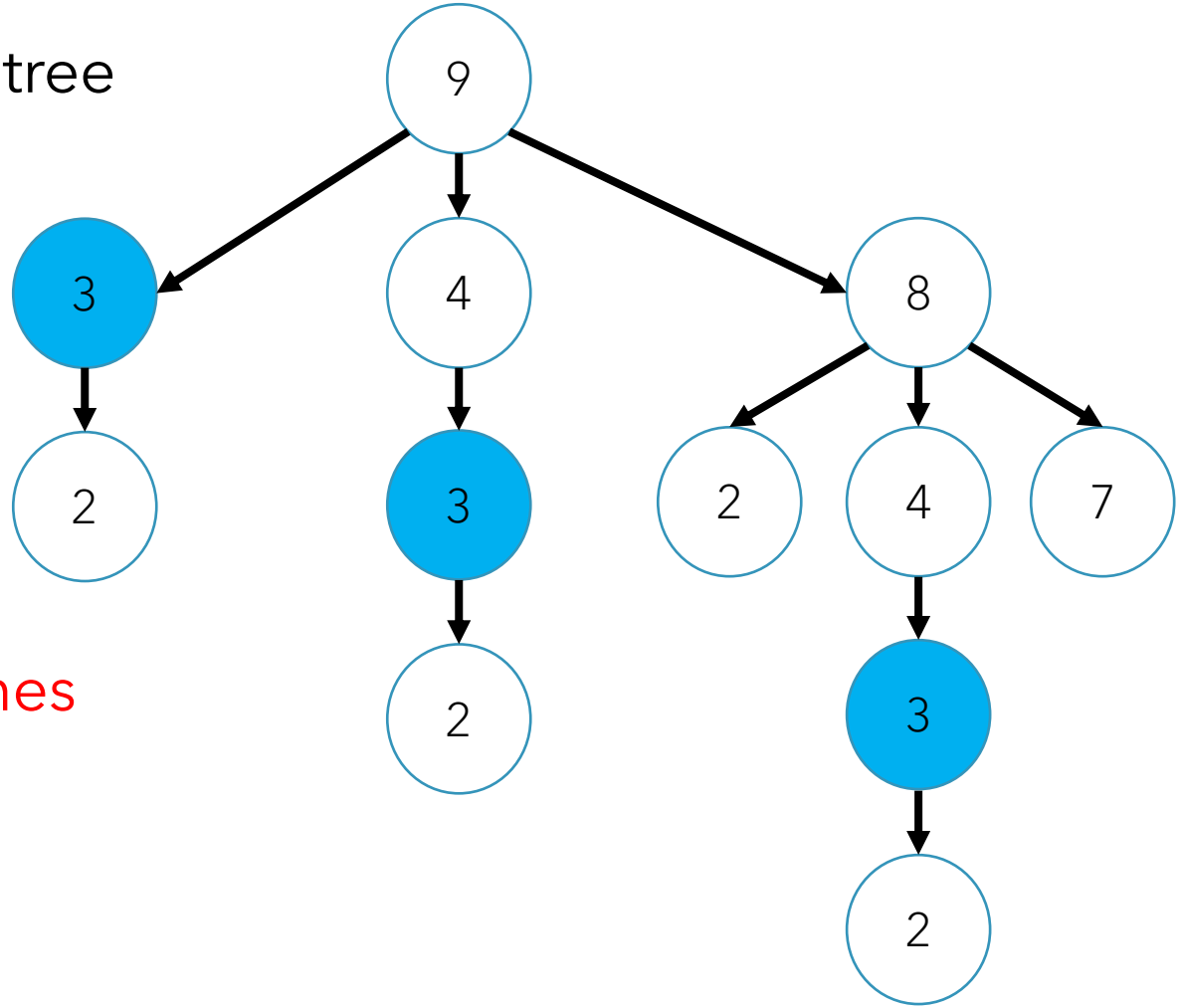


Function RecursiveChange(money, coins)

```
if (money == 0)
        return 0
else
        MinNumCoins = Inf
        for  i from 1 to |Coins|
                if(money≥ coins_i):
                  NumCoins = RecursiveChange(money-coins_i,coins)
                  if(NumCoins + 1 < MinNumCoins):
                        MinNumCoins = NumCoins + 1
        return MinNumCoins
```

Issues with Recursive Strategy

- Its too slow
- Trace the problem using recursion tree



A sub-problem is solved multiple times

Dynamic Programming

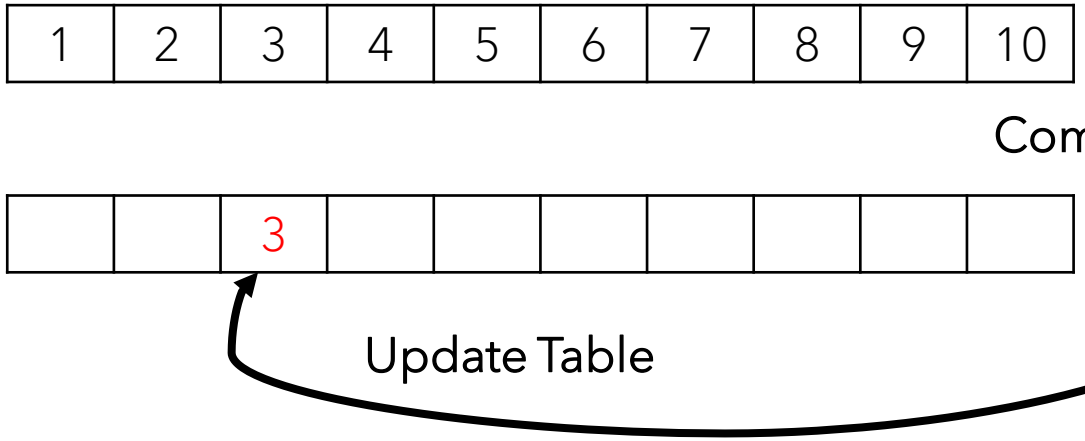
- Programming model that can be applied when solution exhibits the following properties

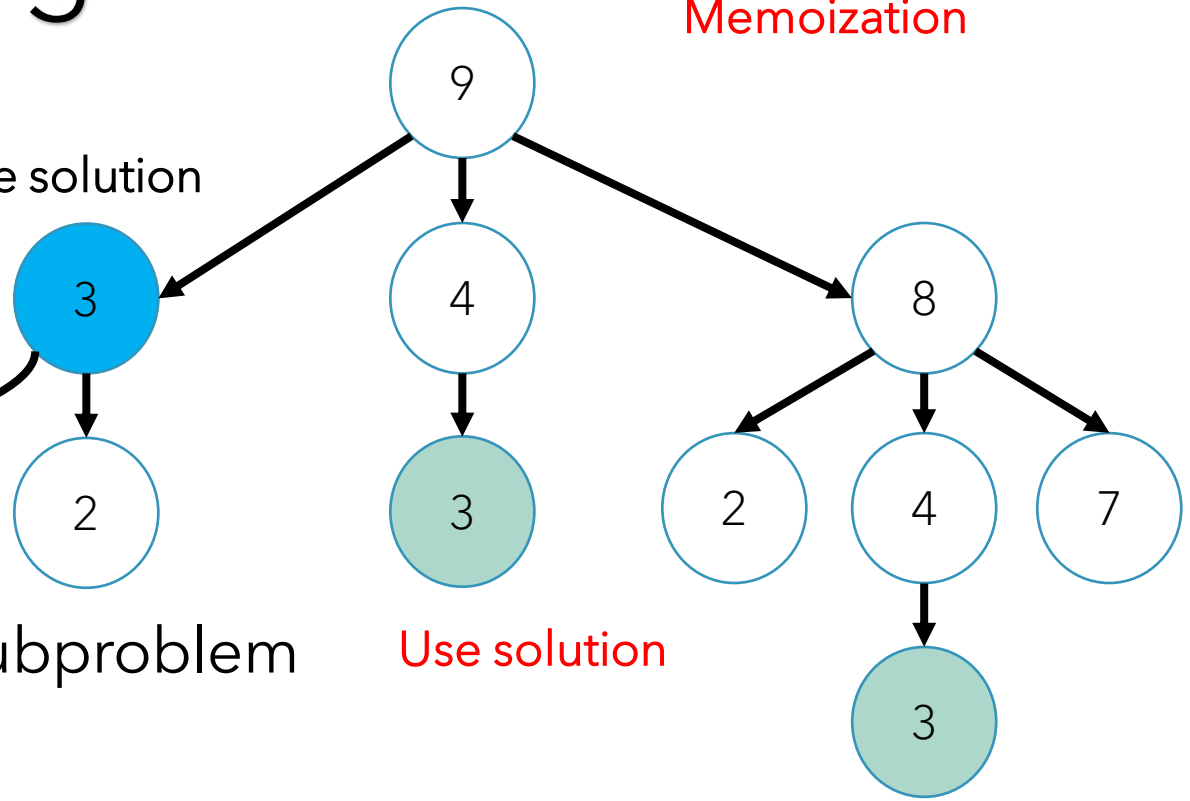
  Optimal substructure Mincoins (money) = $\underset{1 \le i \le |c|}{min}(\mathbf{Mincoins(money - c[i])})$

  Overlapping subproblem

    A sub problem is solved more than once

Top Down Programming





- Use recursion
- Use a table to maintains the results of subproblem
- Whenever a problem has to be solved

  if solution is available in the table, use it

  Else compute the solution and populate it in the table

Pause & Think

- What are the different subproblems that can arise?

  The money can be reduced to any value depending on the input

  Possible values will be between 0 and Money (Original Amount)

- What will be the size of the array ?

  Equal to the amount

Let soln be the array that contains solutions for the subproblem

Function RecursiveChange(money, coins)

```
if (money == 0)
        return 0
else {                                  Compute solution for
                                        new subproblems
        if(soln[money] = -1){
                MinNumCoins = Inf
                for  i from 1 to |Coins|
                        if(money≥ coin_s_i):
                            NumCoins = RecursiveChange(money-coin_s_i,coins)
                            if(NumCoins + 1 < MinNumCoins):
                                MinNumCoins = NumCoins + 1
                return MinNumCoins
        } else{ return soln[money]}
}                                       Use the existing solution
```

Bottom up Programming

- Tabulation
- Solve all the subproblems starting with the smallest subproblem

Function Tabulation(money, coins)

```
soln[0] = 0
for  i  from 1 to money:
        for j from 0 to |coins|:
                if(i-coins[j] >=0){
                    if(soln[i]>soln[i-coins[j]] + 1){
                        soln[i] = soln[i-coins[j]]+1
                    }
                }
return soln[money]
```

Time Complexity
Basic operation – Assignment

O(money * |coins|)

Summary

- Discussed about recursive strategy for the problem of coin denomination problem.

Thank You

Happy Learning

Success is always inevitable with Hard Work and Perseverance