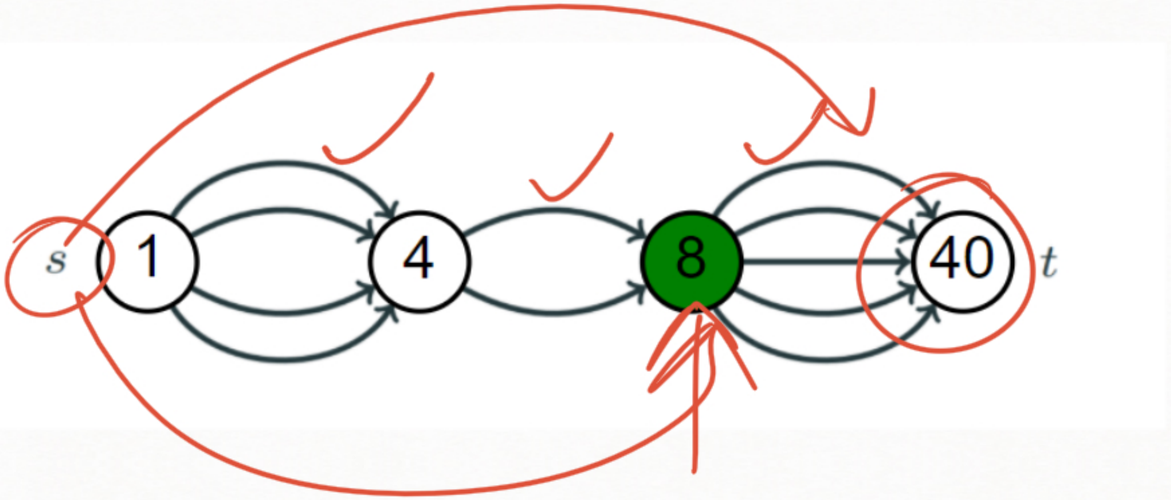
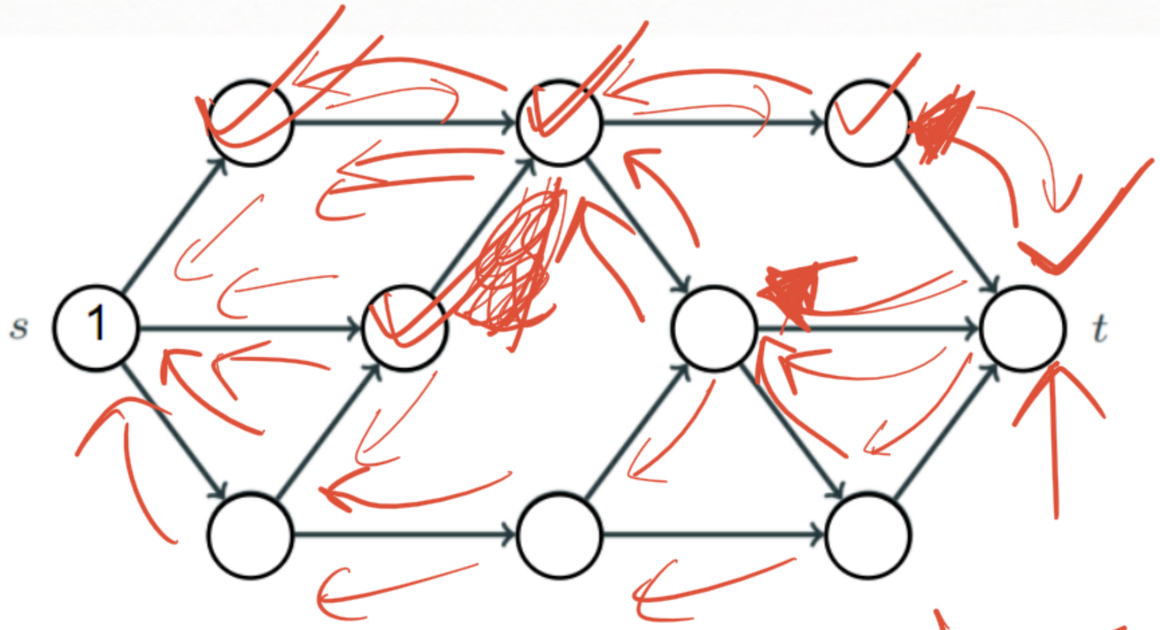

s
1
4
8
40
t

Recursive Soln

✓ Possible



→ But it will consume a lot of time.

The other method of keeping track and solving the smallest subprob is way faster.

Coin Change problem

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

Output: The minimum number of coins with denominations $coin_1, \ldots, coin_d$ that changes *money*.

money = 40

denominations = 25 | 10 | 5

Output = 3

5 → 8 Coins = 8

10 → 4 Coins = 4

25 → 1 Coin + 5 (1 Coin) + 10 (1 Coin) = 3

10 (2 Coin) + 5 (4 Coins) = 6

Greedy Soln

greedy choice = choosing the largest den. first.

Wrong ✗

den = | 25 | 20 | 10 | 5 |

money = 40

Min = 2

2 Coins (20 Rupees)

20 x 2 = 40

what if I check all the combinations possible?

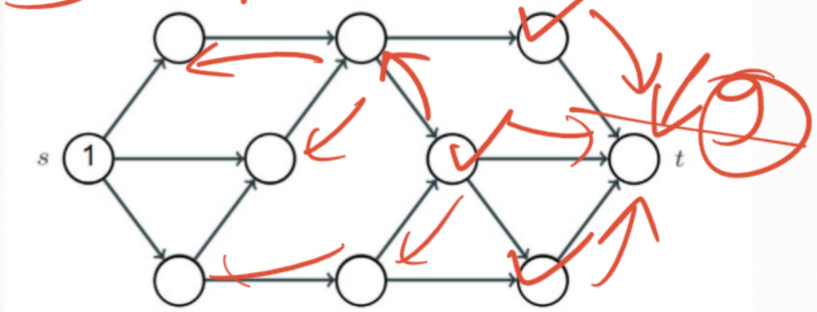
=) I am sure I will get the Answer (min. no. of coins).

Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(3) + 1 \\ MinNumCoins(4) + 1 \\ MinNumCoins(8) + 1 \end{cases}$$

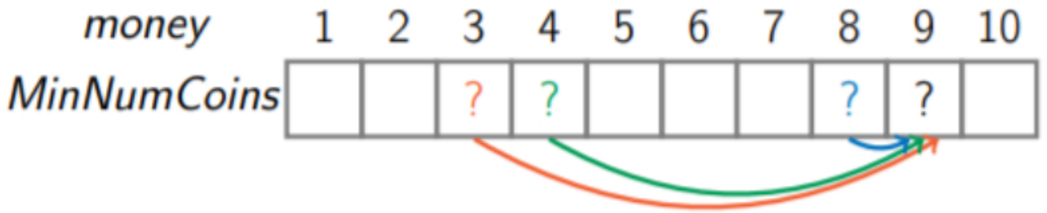
money	1	2	3	4	5	6	7	8	9	10
MinNumCoins			?	?				?	?	



Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

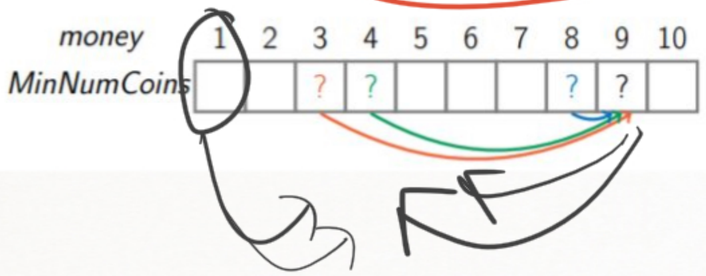
$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(3) + 1 \\ MinNumCoins(4) + 1 \\ MinNumCoins(8) + 1 \end{cases}$$



Recursive Change

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(3) + 1 \\ MinNumCoins(4) + 1 \\ MinNumCoins(8) + 1 \end{cases}$$



$$minnum(10-6) = \min \begin{cases} \times \\ \times \\ \rightarrow minnum(4-1) =) \end{cases}$$

$$minnum(5) = \min \begin{cases} \times \\ minnum(5-5) \\ minnum(5-1) \end{cases}$$

$$minnum(9) = \min \begin{cases} min(3) \\ min(4) \\ min(8) \end{cases}$$

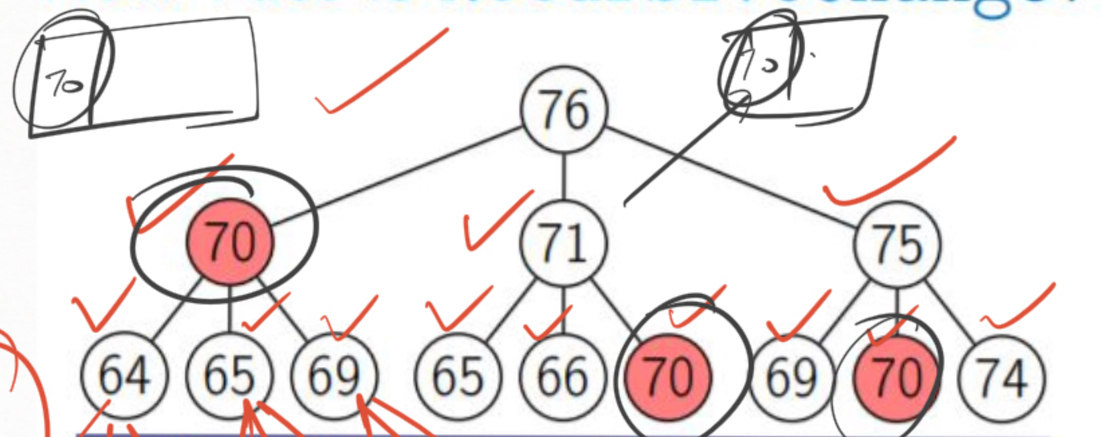
$$minnum(10) = \min \begin{cases} \checkmark minnum(10-6)+1 \ (6\ Deno) \\ \checkmark minnum(10-5)+1 \ (5\ Deno) \\ \checkmark minnum(10-1)+1 \ (1\ Deno) \end{cases}$$

Recurrence for Change Problem

$$MinNumCoins(money) =$$

$$\min \begin{cases} MinNumCoins(money - coin_1) + 1 \\ MinNumCoins(money - coin_2) + 1 \\ \cdots \\ MinNumCoins(money - coin_d) + 1 \end{cases}$$

How Fast is RecursiveChange?



the optimal coin combination for 70 cents is computed at least **three** times!

the optimal coin combination for 30 cents is computed **trillions** of times!

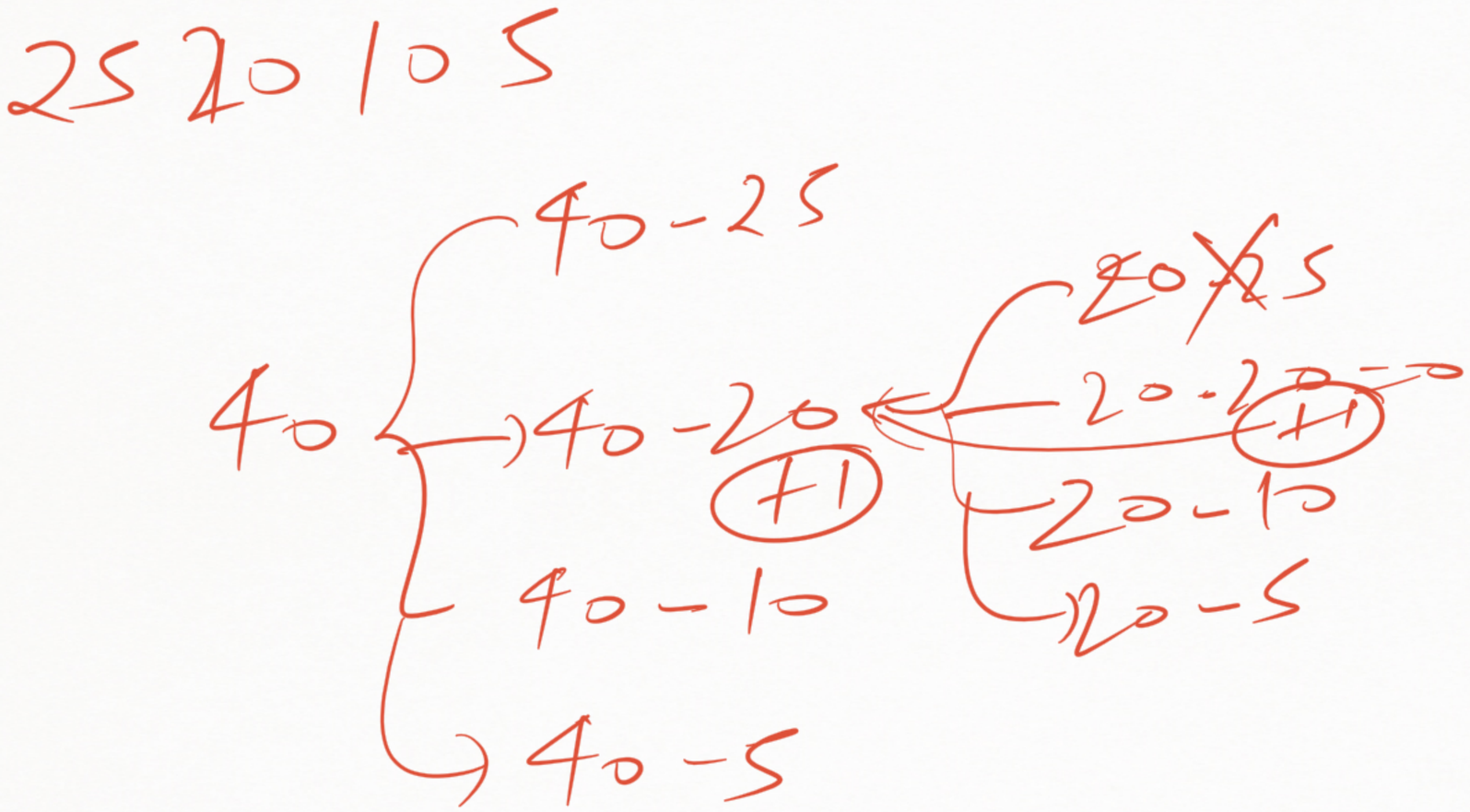
Money = 76

denos = [1, 5, 6]

76 { 76 - 6 = 70
76 - 5 = 71
76 - 1 = 75

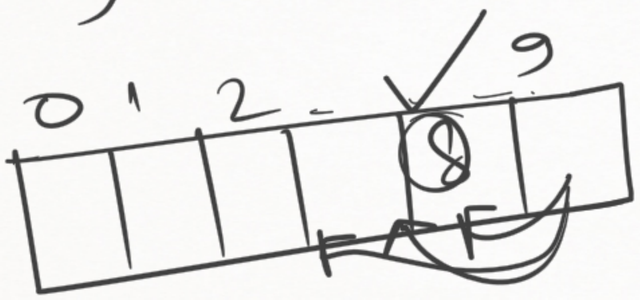
→ Very Very Huge Tree of Calculations

→ We are calculate the same thing again & again!


25 20 10 5
40
40-25
40-20
+1
40-10
40-5
20-5
20-10
+1

den = 1 6 5

g = ?



g - 1 = 8

g - 1 = 3

g - 5 = 4

Hint

Wouldn't it be nice to know all the answers for changing $40 - coin_i$ by the time we need to compute an optimal way of changing 40

Instead of the time-consuming calls to

`RecursiveChange`$(money - coin_i, coins)$

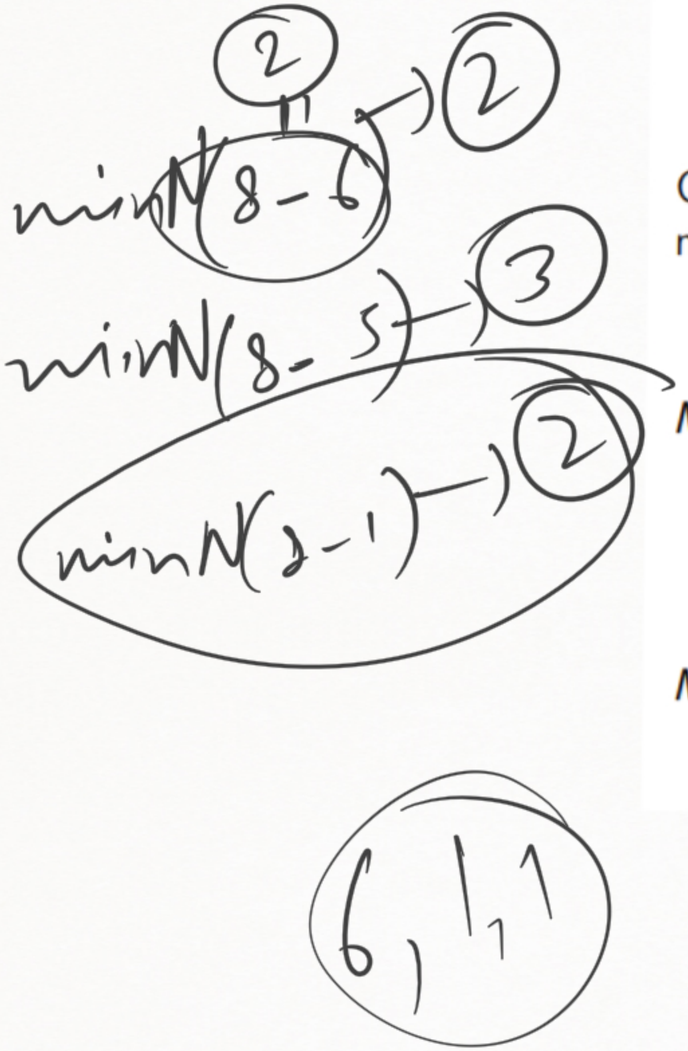
we would simply look up these values!

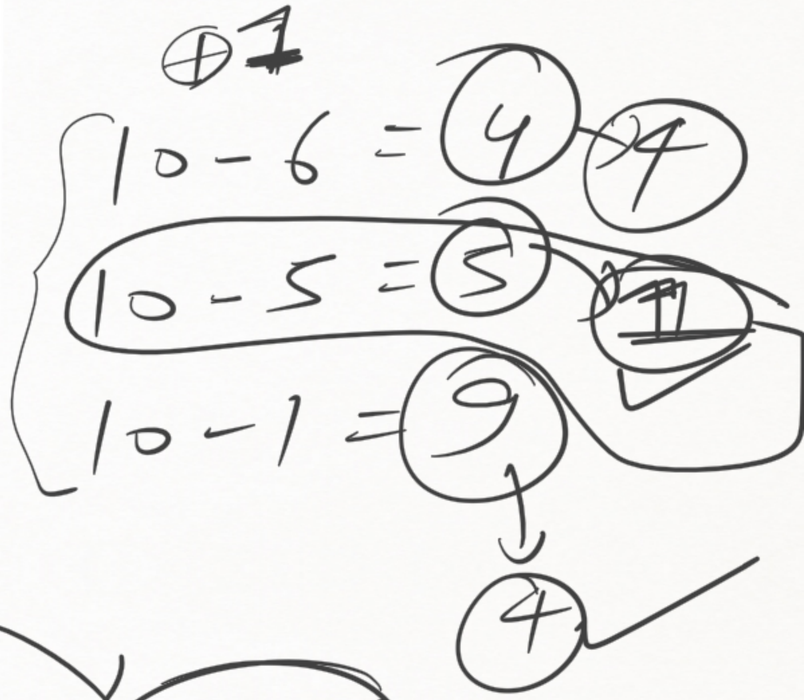
Recursive Change

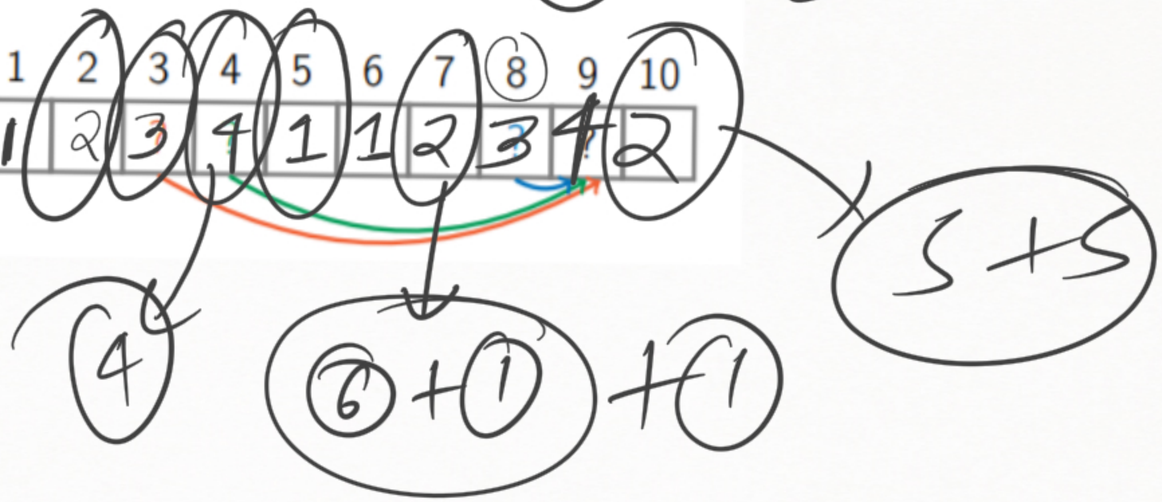
Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(3) + 1 \\ MinNumCoins(4) + 1 \\ MinNumCoins(8) + 1 \end{cases}$$

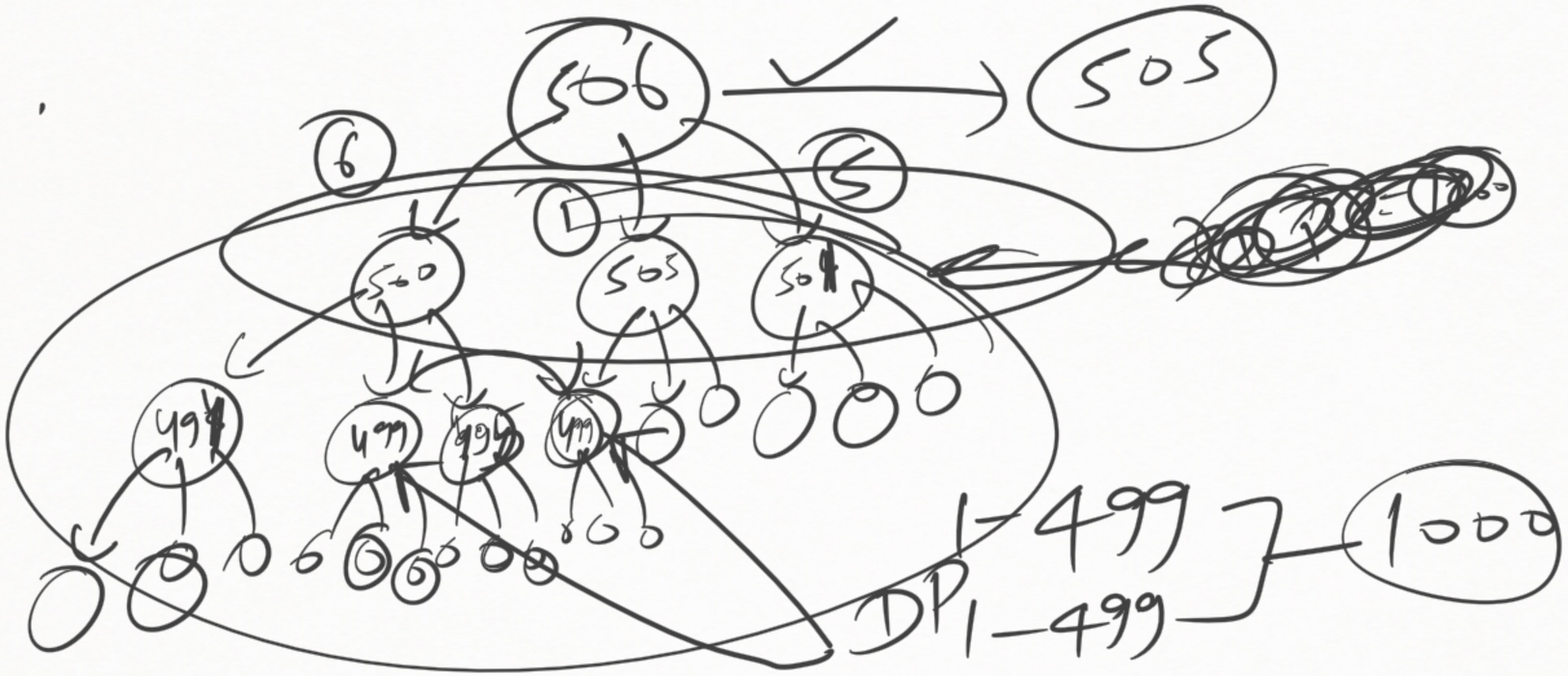
money	1	2	3	4	5	6	7	8	9	10
MinNumCoins	1	2	3	4	1	1	2	3	4	2







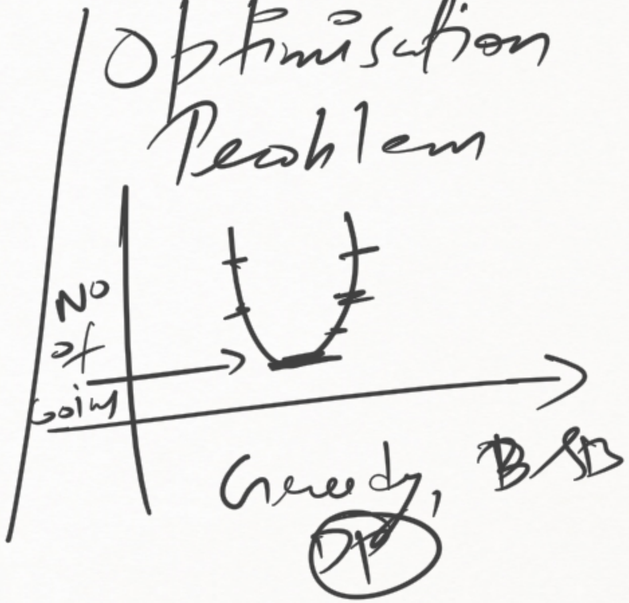
Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using **Dynamic Programming**. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.



Conditions for DP

Optimal Substructure

Overlapping Subproblem.



```
DPChange(money, coins)
MinNumCoins(0) ← 0
for m from 1 to money:
    MinNumCoins(m) ← ∞
    for i from 1 to |coins|:
        if m ≥ coin_i:
            NumCoins ← MinNumCoins(m − coin_i) + 1
            if NumCoins < MinNumCoins(m):
                MinNumCoins(m) ← NumCoins
return MinNumCoins(money)
```