Todays Content:

1. Knap Sack Problems
    a. Rod Cutting
    b. Coin Picking - 1
    c. Coin Picking - 2
    d. Knap Sack - 3

Q: Given a rod of len: N & array A of len N.

cost[i] = price of rod length i (1 based index)

Find the max price that can be obtained by cutting the rod into some pieces & selling them

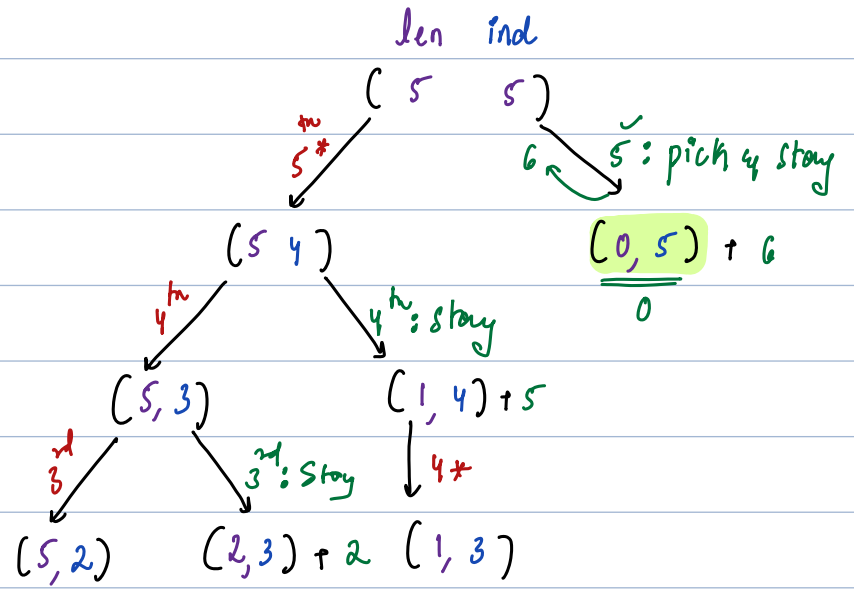
N = 5

	1	2	3	4	5
cost[ ] =	1	4	2	5	6

Cases	Cost
5	6
4 + 1	6
2 2 1	9
3 2	6

Trace:

	1	2	3	4	5
N = 5 cost[ ] =	1	4	2	5	6



(0-∞) : Knapsack.

Total weight	Total length : N
item weight	rod length ; index
item value	cost :
Same item pick multiple times	Same rod length multiple times

dp Code:

rod len

```
int dp[N+1][N+1]          TC: O(N^2)  SC: O(N^2) ⟶ O(N) : Todo
// dp[i][j] = Max cost with i items & len = j

N                          N
∀ dp[0][j] = 0             ∀ dp[i][0] = 0
j=0                        i=0

for(int i=1; i<=N; i++){
    for(int j=1; j<=N; j++){
        dp[i][j] = dp[i-1][j]; // ith not pick
        if(j>=i){                                  // ith pick & stay
            dp[i][j] = Math.max(dp[i-1][j], dp[i][j-i] + c[i])
        }
    }
}
// Final ans: Using N items, & length N: Max value
return dp[N][N]
```

Q8: Given an integer A[] representing coins.

A[i] = value of i^{th} coin

One coin can be used multiple times

Find no: of ways to select coins with sum = N.

Ordered selection = (x, y) ≠ (y, x)

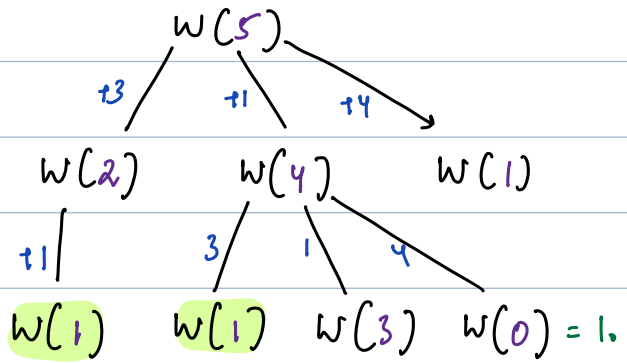
(2 3) ≠ (3 2)

Ex: N = 5

A[3] = { 3 1 4 } { 4 1 } { 1 4 }

{ 3 1 1 } { 1 3 1 } { 1 1 3 }

{ 1 1 1 1 1 }



1. Subproblems
2. Overlapping.

Code:

TC: O(N * M) SC: O(N)

N: no: of steps

M: coins

```
int dp[N+1];
// dp[i] = no: of ways to get sum = i
dp[0] = 1; // ways to get sum = 0 : { }
for(int i=1; i<=N; i++) {
    dp[i] = 0;
    for(int j=0; j< A.length; j++) {
        if (i >= A[j]) {
            dp[i] = dp[i] + dp[i - A[j]]
        }
    }
}
return dp[N];
```

38 Unordered Selection (x y) = (y x) 11:10

Given an integer A[k] representing coins.

A[i] = value of i^{th} coin // No: of coins

One coin can be used multiple times

Find no: of ways to select coins with sum = N.

Ordered Selection = (x, y) = (y, x)

Ex: N = 6 0 1 2 {4 1 1} {3 3}

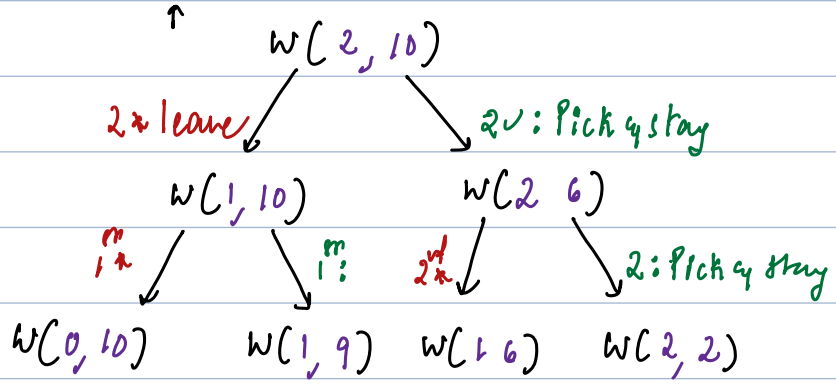
A[3] = {3 1 4} {3 1 1 1}

{1 1 1 1 1 1}

3..... 1.... 4....

Ex3: N = 10 0 1 2 1 4

A[3] {3 1 4}



Code: // Coins 0 based index.

int dp[k] [N+1] // Using k items get sum: N

//dp[i][j] = Using coins [0, i] ways to get sum j

0 1 2 .. i-1 i : j

x pick 4 stay

dp[i][j] = dp[i-1][j] + dp[i][j-A[i]]

if j >= A[i]

Final Ans: Using coins [0.. k-1] ways to get sum N.

return dp[k-1][N]

Q → Given N items with their value & weight

Find the maximum total value that can be bought with W <= K, Pick each item once.

Constraints:

$1 <= N <= 500$

$1 <= V[i] <= 50$

$1 <= W[i] <= 10^9$

$1 <= K <= 10^9$

Toys: N = 4

K = 50 : 1 2 3 4

W[ ] : 20 30 25 40 corr = 45 <= B ✓

V[ ] : 6 10 20 15 ans = 26.

Idea: knapsack 0/1

TC: $(N * K) = 500 * 10^9 > 10^9$ TLE.

Previous Idea:

dp[i][j] = max value we can get

i items | j weight

New Idea:

dp[i][j] = min weight

i items | value

Using i items get j value using min weight

$500 * 500 * 50 = 125 * 10^5 = 1.25 * 10^7$

New Idea:

dp[i][j] = min weight

i items | value

Using i items get j value using min weight

→ 500 * 500 * 50 = $125 * 10^5$ = $1.25 * 10^7$

dp[i][j] = Using i items get j value with min weight.

int V = N*50 // Max value can be obtained.

int dp[N+1][V+1]

TC: O(N*V) SC: O(N*V)

```
for(int j=0; j<=V; j++){
    dp[0][j] = INT_MAX  Using 0 items get j value Not possible
}
for(int i=0; i<=N; i++){
    dp[i][0] = 0
}

for(int i=1; i<=N; i++){
    for(int j=1; j<=V; j++){
        dp[i][j] = dp[i-1][j] // leave ith item
        if(j >= v[i]){                    // Pick ith item.
            dp[i][j] = Min(dp[i-1][j], dp[i-1][j - v[i]] + w[i])
        }
    }
}
```

1 2 .. i-1 (i: 30) (j: 25)

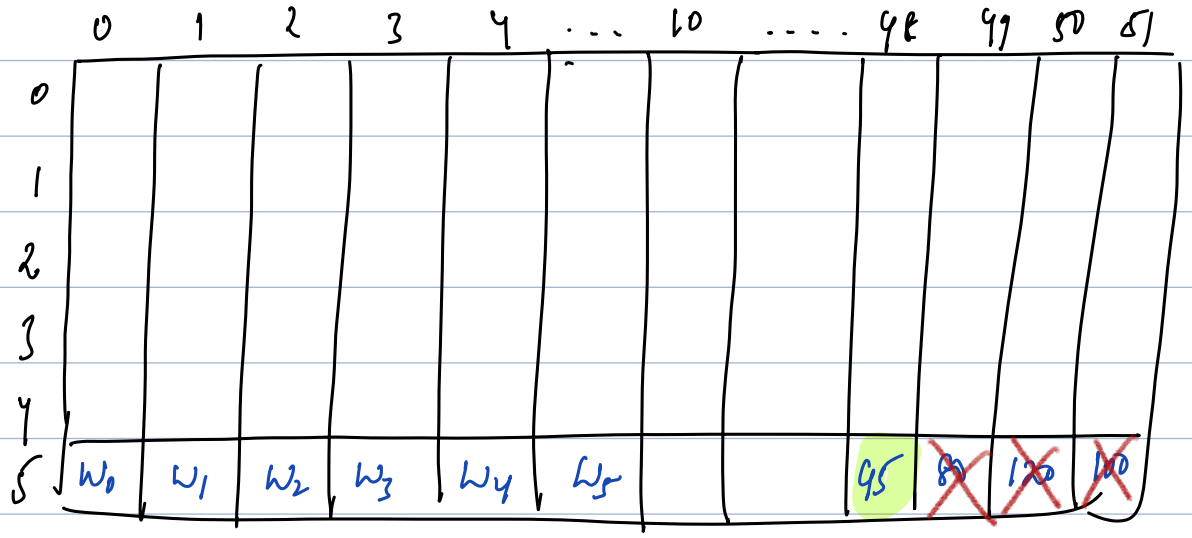
```
for(int j=V; j>=0; j--){
    if(dp[N][j] <= K){ return j }
        // min weight  given weight
}
```

N = 4

K = 50 : 1 2 3 4

W[ ] : 20 30 25 40 corr = 45 ✗ = B ✓

V[ ] : 6 10 20 15 ans = 26.



dp[i][j] = Using i items get j value with min weight.

dp[5][51] = using 5 items get 51 value with weight = 100