Today's Content:

- Coin Change problems
  - Ways to get required sum
  - min coins to get required sum
- Subset Sums Variations
  - Ways to divide into equal sums
  - Ways to divide sum that their diff = k
  - Minimize abs diff of 2 sets
- Space Optimization { If Time permits }
- A small paint problem, if time permits

Coin Change Problem:

Given N Coins & k Sum, no of ways we can get sum = k

Note: Every coin can be used at max 1 {0/1}

	1	2	3	4	5	6	7	8
Coin[ ] = {	7	4	9	6	11	5	13	8 }

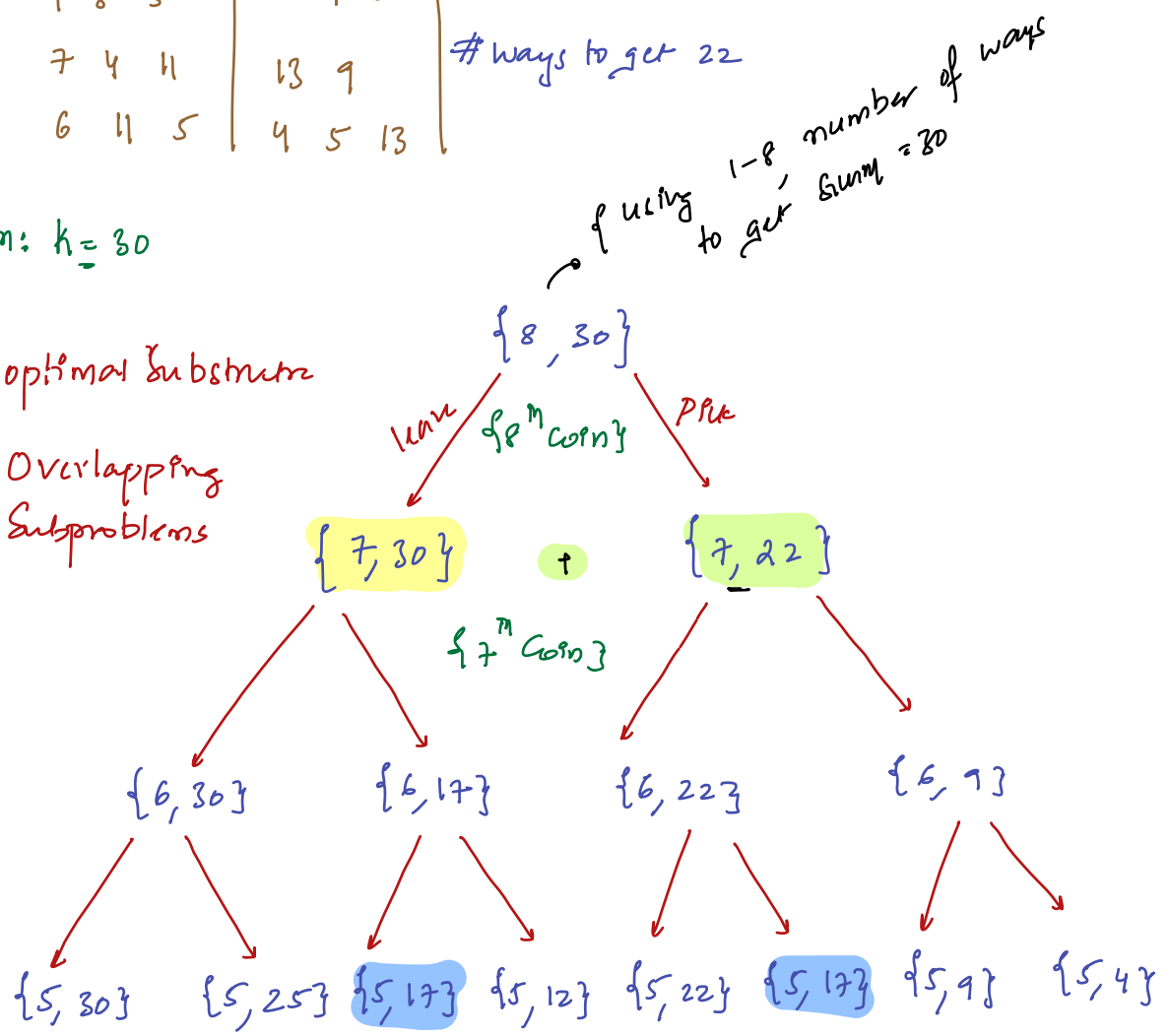
k = 22:

→	9 8 5	7 9 6	4 6 7 5
→	7 4 11	13 9	# ways to get 22
→	6 11 5	4 5 13	

Ex: k = 30

1) optimal Substructure

2) Overlapping Subproblems



// dp State:

dp[i][j] = { // Number of ways to get sum j Using [1 - i] coins }

// dp Expression:

{ j >= c[i] }

dp[i][j] = { dp[i-1][j] + dp[i-1][j - c[i]] }

// dp Table:

dp[N+1][K+1]

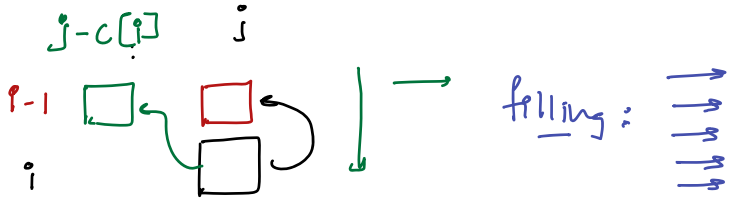
i

j : [0 K]

// Basc Conditions :

i = 0 : {no coins available} $\forall_{j=1}^{K}$ dp[0][j] = 0

j = 0 : { Target Sum = 0 } $\forall_{i=0}^{N}$ dp[i][0] = 1



1) dp[N+1][K+1]

Basc Conditions : {TODO}

TC : # States * TC for cons state

(N*K) * 1

TC : O(N*K)

SC : O(N*K) dpac Optimization

```
i = 1 ; i <= N ; i++) {
    j = 1 ; j <= K ; j++) {
        if( j >= c[i] ) { dp[i][j] = dp[i-1][j - c[i]] + dp[i-1][j] }
        else { dp[i][j] = dp[i-1][j] }
}
return dp[N][K]
```

Ex: 1 2 3 4 5

c[] : 1 2 3 6 7

{ j >= c[i] }

$dp[i][j] = \{ dp[i-1][j] + dp[i-1][j - c[i]] \}$

k = 10

	0	1	2	3	4	5	6	7	8	9	10
v 0	1	0	0	0	0	0	0	0	0	0	0
1 1	1	1	0	0	0	0	0	0	0	0	0
2 2	1	1	1	1	0	0	0	0	0	0	0
3 3	1	1	1	2	1	1	1	0	0	0	0
6 4	1	1	1	2	1	1	2	1	1	2	1
7 5	1	1	1	2	1	1	2	2	2	3	3 → // final ans = 3

// Optimization using 2 Rows ?

N = 5 k = 10

c[] : 1 2 3 6 7

	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0

$0^m \to 0$ $4^m \to 0$

$1^m \to 1$ $5^m \to 1$

$2^{nd} \to 0$

$3^{rd} \to 1$

$i^m \to \{ i \% 2 \}$

// Pseudo code :

```
dp[2][k+1];

// fill N=0
for(j=1; j<=k; j++){ dp[0][j] = 0 }

// fill j=0 i=0; i<=N; i++){ dp[i][0] = 1 }  → {Error}

for(i=1; i<=N; i++){
    dp[i%2][0] = 1;   } Base condition we fill
    for(j=1; j<=k; j++) {
        dp[i%2][j]  =  dp[(i-1)%2][j]
        if( j >= c[i]) {
            dp[i%2][j]  += dp[(i-1)%2][j-c[i]]
        }
    }
}
                                  → {10:35 time}
dp[N%2][k]
```

//SC: $2*(k+1) \rightarrow O(k)$

// If Space optimization → {Tracking back is not possible}

{Above thing Can also be done using 1 dp[k+1]} → few chang in code :{optional}

Cannot

SC: O(k)

{Doubts Qes} ← {Not possible} ← {Doubts Session}

// 28) Given N coins find min coins required to get Target Sum k?

Note: Every coin can be picked at max Once?

	1	2	3	4	5	6	7	8
Coin[ ] = {	7	4	9	6	11	5	13	8 }

Eg: k = 30

1) optimal Substructure

2) Overlapping ✓

(8, 30)

min { (7, 30) (7, 22) + 1 }

dp[i][j] = { Min no: of coins needed to get sum = j, Using [1-i] Coins }

dp[i][j] = { // don't pick i^{th} : min (dp[i-1][j] ,
// pick i^{th} Coin : j >= c[i] : dp[i-1][j-c[i]] + 1) }

{INT_MAX} + 1

Table: dp[N+1][K+1]

j = 0 : // 0 sum

$\forall_{i=0}^{N}$ dp[i][0] = 0

// Target Sum = 5, We have 0 coins

dp[0][5] = 0, min coins needed to get sum = 5 ⇒ 0 ✗

i = 0 // 0 coins

$\forall_{j=1}^{k}$ dp[0][j] = {N+1} // if we get a better ans we will update in dp state

{INT_MAX}

Edge Case: {INT_MAX + 1} ⇒ {overflow}

// dp[i][j] > N :

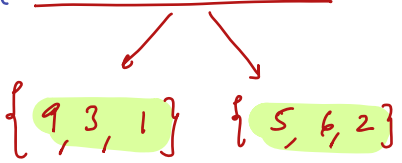
{ We cannot get sum = j

{ No Subset with Sum = j

3(8) Given N Elements, Divide all elements into 2 Subsets ?

Q) Find no: of ways we can divide all Elements in 2 Subsets, such that their Sum is same ?

Ex1: { 1 5 3 6 1 2 } → ans: 1



// Subsets

Idea: All Elements → A : Sum (A)

B : Sum (B)

Sum (A) + Sum (B) = Total Sum

Sum (A) = Sum (B)

Sum (A) + Sum (A) = TS

2 Sum (A) = TS

Sum (A) = TS/2 → If TS is odd not possible

Idea: Subset with Sum TS/2

Ex: 1 5 3 6 1 2 Sum = 13

Idea: Recursively get n = Number of Subsets with Sum = TS/2

Final ans = depends on Question

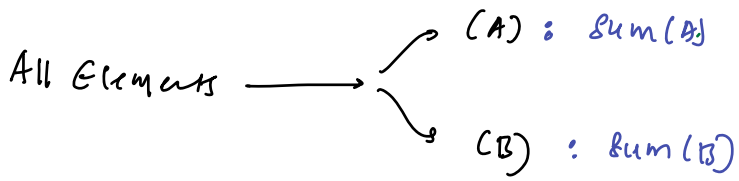
A: { 9 3 1 } → { Excluding 5 6 2 } also have to B

↓

A: { 5, 6, 2 } → { 9, 3, 1 }

48) Given N Elements, Divide all elements into 2 Subsets ?

Find no: of ways we can divide all elements in 2 Subsets, such that their sum diff = k ?



Sum(A) + Sum(B) = TS

Sum(A) - Sum(B) = k

---

2 Sum(A) = TS + k

Sum(A) = $\frac{TS + k}{2}$ → { If TS + k is even }

Calculate number of Subsets with Sum = $\left(\frac{TS + k}{2}\right)$

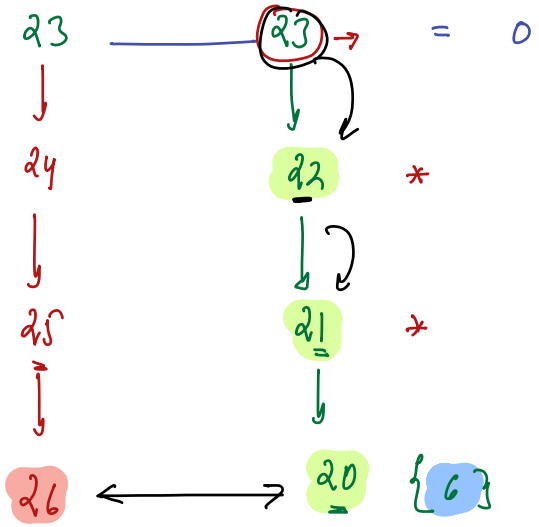
Sol] Given N Elements, Divide all elements into 2 Subsets ?

divide all elements into 2 subsets such that their sum diff should be min ?

All Elements → A, B } [Sum (A) - Sum (B) is minimum]

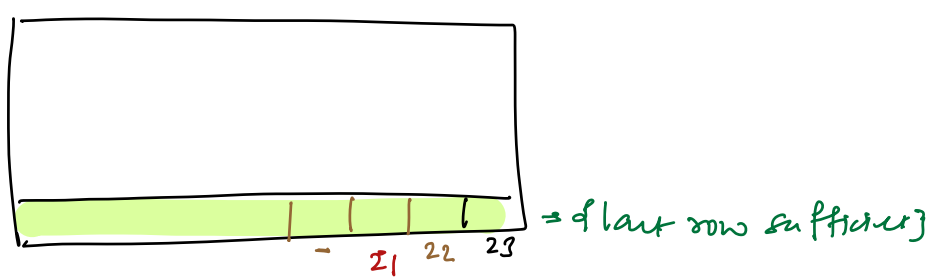
Eg: ar[5] = { 3 6 2 9 26 } → TS: 46

| Sum(A) - Sum(B) |



A	B
TS/2	TS/2
TS/2 + 1	TS/2 - 1 *
TS/2 + 2	TS/2 - 2

// When we check sub = 23



// Pseudocode:

// given N elements, min subset diff ?

// We need to construct dp[N+1][TS/2 + 1] ? ⇒

↳ {filling this table} TODO

```
i = TS/2
while ( i >= 0) {
    dp[N][i] = True
        We have a subset = possible
        A = TS - i        B = i        } diff between these 2 sums
        return TS - 2i
    }
    i = i - 1
}
```

Ex: {100}

	sum	sum diff
A: {100}	100	100
B: { }	0	

A — B

100 — 0 { 100 } ans

100
99
98
.

// → 1 Dp

→ 2d matrix

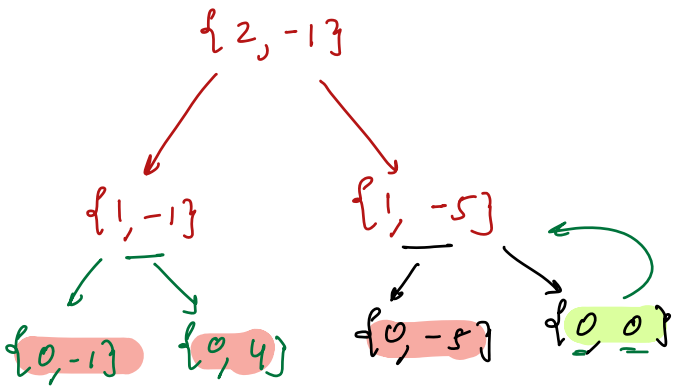
→ strings

→ knapsack → 0/1
→ 0/∞

→ Subset Sums

→ { MCM → Stock buy sell
Doubts → { }

// -5, 4 : {-1}



$dp[i][j] = dp[i-1][j] \;||\; dp[i-1][j-c[i]]$

or

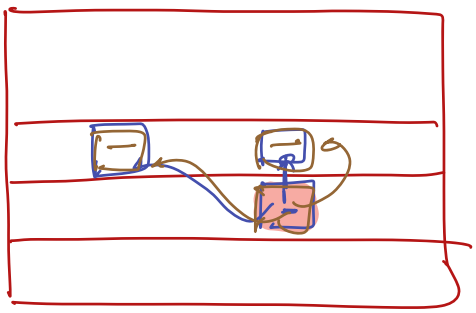
// We need to store states in hashmap

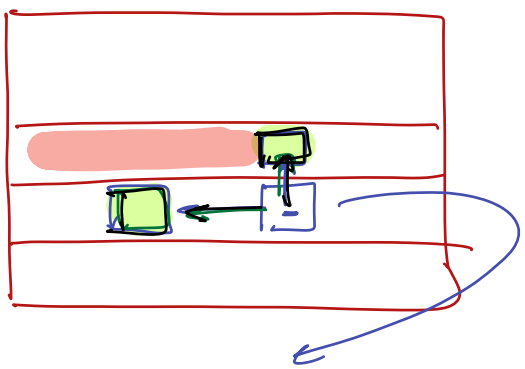
// Top-down Recursive approach using hashmap

// flashmap <string, val> hm;

```
bool TargetSum(int N, int k, int c[]){
    if(N==0){ // no more coins
        return k==0
    }
    string s = to_string(N) + to_string(k)
    if(!hm.contains(s)){
        bool ch = (TargetSum(N-1, k, c) ||
                   TargetSum(N-1, k-c[N], c))
        hm[s] = ch
    }
    return hm[s]
}
```

// space Optimize → 1D

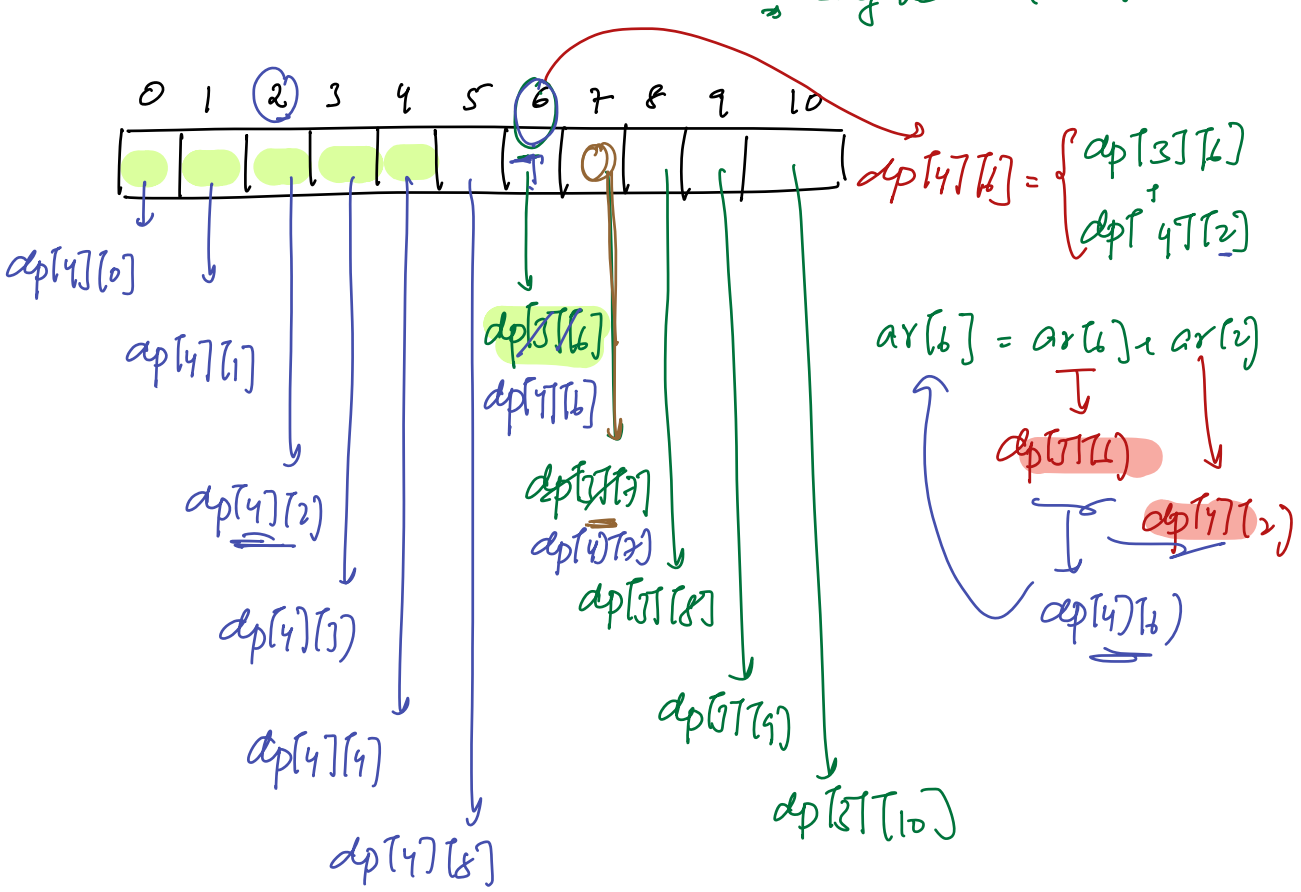




{ dp Expression : dp[i][j] = dp[i-1][j], dp[i][j-c[i]] }

// Example:

N=5, k=10 minimum = 2



ar[7] = dp[3][7]

dp[4][7] = dp[3][7] + dp[4][5]

ar[7] = ar[7] + ar[5]