Coin-change.

You have some coins of n different denominations.
No. of ways to pay an amount $= K$.
[get a change of K].

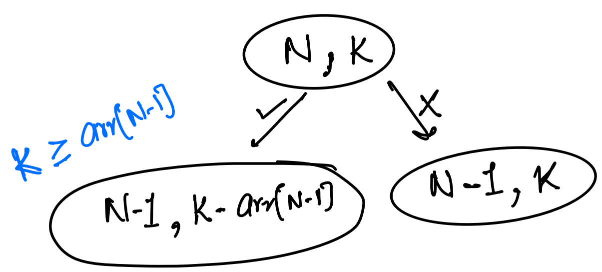
Note. → Can't take one denomination coin more than once.

Eg → 7 4 9 6 10 13 14 11

$K = 22$

7 4 11
9 13
7 9 6

idea. → Consider all the subsets. [Backtracking] T.C → $O(2^N)$



optimal substructure ✓
overlapping sub-problems ✓

$$\left[ways(N, K) = ways(N-1, K) + ways(N-1, K - arr[N-1]) \right]$$

int dp[N+1][K+1]

{ dp[i][j] ⇒ no. of ways in which amount j can be paid by using first i coins. }

top down.

```
int dp[N+1][K+1]; //initialise -1

int ways(int[] arr, int i, int j, int[][] dp) {     // i → N, j → K
    if (j == 0) { return 1 }   // 0 amount can be paid in 1 way
                               // i.e do nothing.
    if (i == 0) { return 0 }
    if (dp[i][j] != -1) { return dp[i][j] };

    dp[i][j] = ways(i-1, j);
    if (j >= arr[i-1]) {
        dp[i][j] += ways(i-1, j - arr[i-1])
    }
    return dp[i][j];
}
```

{ T.C → O(N*K)
S.C → O(N*K) }

Bottom-up.

{ dp[i][j] ⇒ no. of ways in which amount j can be paid by using first i coins. }

```
int dp[N+1][K+1];
// initialise row-0 with 0
// initialise col→0 with 1.

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

[T.C → O(N*K)
S.C → O(N*K)]

arr = [2, 3, 5, 7] , K = 10.

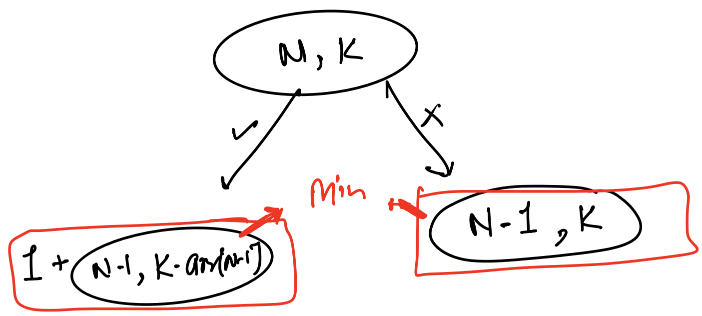
j

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

Q) Find minimum no. of coins to pay amount = K.

Eg → 7 4 9 6 10 13 14 11 , $\underline{K = 22}$

ans - [9, 13] ⇒ $\underline{2}$.



$$\left[\ \text{minCoins}(N, K) = \text{Min}\Big(\text{minCoins}(N-1, K),\ 1 + \text{minCoins}(N-1, K - arr[N-1])\Big)\ \right]$$

dp[N+1][K+1]

// initialise row = 0 with ∞

// initialise col = 0 with 0 (minimum coins required to pay amount = 0 is 0)

$$\Big[\Big[\ dp[i][j] = \min\Big(\mathrm{dip}[i-1][j],\ \underbrace{1 + dp[i-1][j - arr[i-1]]}_{j \geq arr[i-1]}\Big)$$

{ $\underline{dp[i][j]}$ → Minimum coins required to pay amount j from first i coins. }

any denomination any no. of times.

N, K

$\times$ → N-1, K

✓ → N, K- arr[N-1]

[similar to unbounded knapsack]

Q: N elements. (No. of ways to -

- Divide elements into two parts such that both parts have equal sum.)

arr →

1	5	3	6	9	2

1, 3, 9 5, 6, 2

[mandatory to distribute all elements to one of the parts.]

N → A, B

$$sum(A) = sum(B)$$

$$sum(A) + sum(B) = total\ sum.$$

$$sum(A) + sum(A) = total\ sum$$

$$2 \cdot sum(A) = total\ sum$$

$$\boxed{sum(A) = \frac{total\ sum}{2}}$$

$$K = \frac{total\ sum}{2}$$

(2)

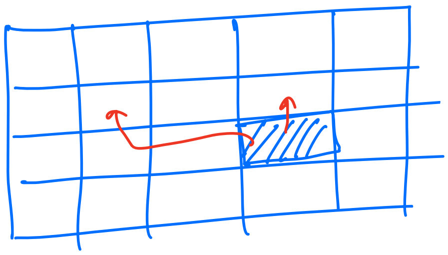
$$sum(A) - sum(B) = K$$

$$sum(A) + \cancel{sum(B)} = total\ sum$$

$$2 \cdot sum(A) = K + total\ Sum$$

$$\left\{ sum(A) = \frac{K + total\ Sum}{2} \right\}$$

- **KnapSack.** → int dp[N+1][W+1]



$O(N*W)$

↓ Bottom-up.

reduce S.C ?

↓

Yes.

val	weight
1	2
2	3
3	4

W = 7.

dp.

0	1	2	3	4	5	6	7
0	0	1	~~1~~ 2	~~1~~ 2	~~1~~ 3	~~1~~ 4	1

dp[j], val[i-1] + dp[j - wt[i-1]]

idea → start from R.H.S.

```
int dp[W+1];
for( i=1; i <= N; i++){
    for( j=W; j >= 0; j--){
        dp[j] = max( dp[j], dp[j - wt[i-1]] + val[i-1] );
                            • j >= wt[i-1]
    }
}
return dp[W];
```

constraints -

$$\left[\begin{array}{l} 1 \le N \le 500 \\ 1 \le W \le 10^9 \\ 1 \le weight(i) \le 10^9 \\ 1 \le value(i) \le 50 \end{array}\right]$$

[500 * 50 = 25,000]

luxury → 1 2 3 4 5 6 7 8

$$\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 4 & 6 & 7 & 10 & 12 & 14 & 15 \end{array}$$

Budget - 10.

maximum luxury that you can buy?

values - [2 1 3] , W = 7.

wt - [3 2 4]

values -

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 3 & 4 & 6 & 7 & 9 \end{array}$$

N , value.

✓   min   ✗

wt[N-1] + N-1 , value - val[i-1]   N-1 , value

[dp[i][j] = min weight required to get value - j from first i - items]

value → 2 1 3   W = 7.

weight → 3 2 4

dp[N+1][maxValue + 1]

maxValue = 2+1+3 = 6

val.	wt
2	3
1	2
3	4

	0	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞	∞
1	0	∞	3	∞	∞	∞	∞
2	0	2	3	5	∞	∞	∞
3	0	2	3	4	6	7	9

dp[i-1],
cot[i-1] +
dp[i-1][j - val[i-1]]

with 0 elements, it is not possible to get val > 0.
∵ min bagpack capacity is asked ∴ ans → ∞.

ans = 5.

pseudo-code

maxValue = $\sum val$

```
int dp[N+1][maxValue+1]

// initialise row→0 with ∞
// initialise col→0 with 0

for( i=1 ; i ≤ N ; i++){
    for( j=1 ; j ≤ maxValue ; j++){
        dp[i][j] = dp[i-1][j];
        if ( j ≥ val[i-1] ){
            dp[i][j] = Math.min( dp[i-1][j],
                                 wt[i-1] + dp[i-1][j-val[i-1]] )
        }
    }
}

ans = 0

for( j= maxValue ; j ≥ 0 ; j--){
    if (dp[N-1][j] ≤ W){
        ans = j ; break ;
    }
}
return ans ;
```

T.C → $O(N * \sum val)$

S.C → $O(N * \sum val)$