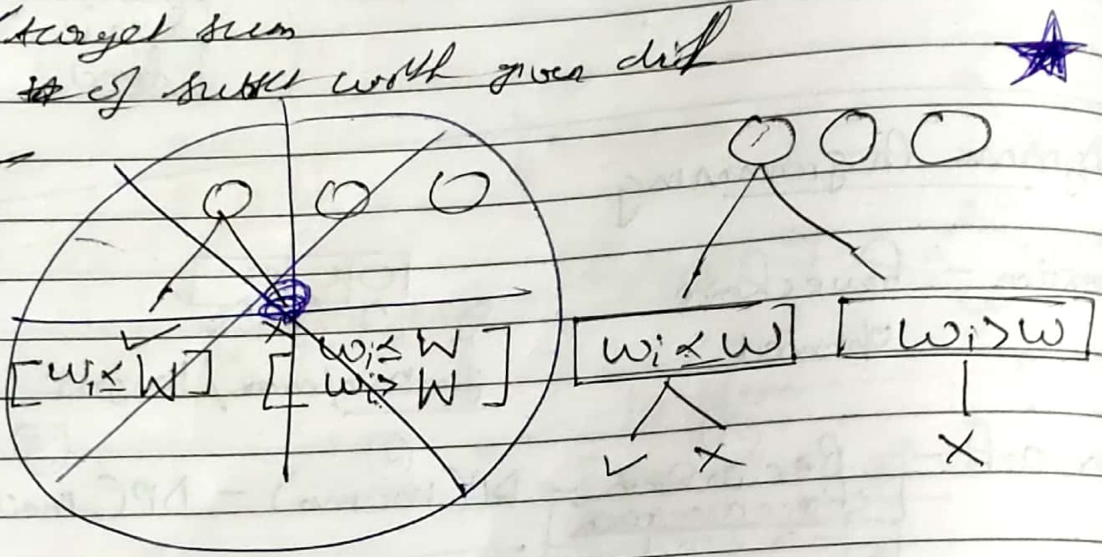
0-1 knapsack

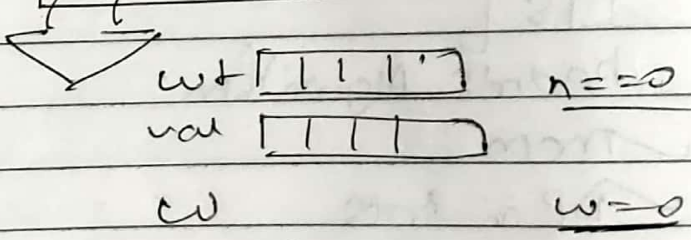
✓subset sum
✓equal sum partition
count of subset sum
✓min subset sum diff
✓target sum
# of subset with given diff

Recursive



Base Condn

```
if (n==0 or w==0)
    return 0
```



choice diagram

```
if wt[n-1] ≤ w:
    return max( val[n-1] +
                solve( ~~w-wt[n-1]~~,
                solve( — )
              )
else:
    return solve( — )
```

memoization

Base Condn

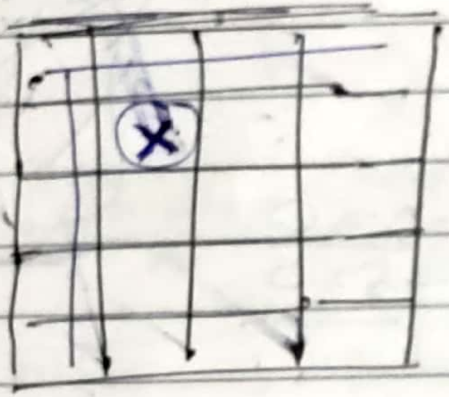
```
if memo[][] != None
    return memo[][]
```

Choice tree

```
memo[][] = ans
return memo[x]
```

[disadvantage: Stack overflow | little slower]

tabular



if (wt[j] -

① initilization → base condn

② solve → memo

i —

j —

01 knapsack bottom-up

Recursive → bottom up

~~take them~~
according to matrix

```
(1) solve (n, w)
{
(2) # base condition
    if (n==0 or w==0)
        return 0

(3) # choice diagram
        w - wt[n-1]
    if ~~wt[n-1]~~ ... >= 0:
        return (max(
    val[n-1] + solve(n, w-wt[n-1], ...)

    else:
        return solve(n-1, w)
```

```
memo = [[None for (n+1)]
         for (w+1)]]

# base cond^n
for (n = (0 -> n+1))
    for (j = (0 -> w+1))
    if n==0 or j==0
        memo[i][j] = 0

# choice diagram
for n in range(1, n+1)
    for w in range(1, w+1)
    ~~if wt(n-1)-w~~ >= 0
    if (w - wt[n-1]) >= 0
        memo[n][w] = max(
    val[n-1] + t[][w-wt[n-1], ]

    else:
        memo[n][w] =
        memo[n-1][w]
```