Date : / /

0/1 Knapsack Problem

You are given

weights[] ⇒ weight of item W: Maximum weight

values[] ⇒ value of item

n ⇒ No: of items

Each item can only be choosen once

Maximize the total value in the knapsack without exceeding capacity W

Ex :- n = 3

wt = 3 4 5

val = 30 50 60

Sol :- Best choice : item weighing 3 & 5

Total value = 30 + 60 = 90

Date : / /

Recursion

$TC = 2^n$

$SC = O(N)$

```
f(ind, w) {
if(ind == 0) {
  if(wt[0] <= w) return val[0];
  else return 0;
}
notTake = 0 + f(ind-1, w)
take = INT_MIN;
if(wt[ind] <= w)
    take = val[ind] + f(ind-1, w-wt[ind])
return max(notTake, take)
}
```

Memoization

Indexes ⇒ ind, w

Maximum [n] [w+1] ⇒ dp array

```
if(ind==0){
if(wt[0]<=w)return val[0]; else return 0; }
if(dp[ind][w] != -1) return dp[ind][w];
int notTake = 0 + f(ind-1, w)
   take = INT_MIN;
```

Date : / /

Memoization

indexes ⇒ ind, w

maxvalue[n][w+1]

Create a dp[n][w+1]

```
if (ind == 0) {
    if (wt[0] <= w) return val[0];
    else return 0;
}
if (dp[ind][w] != -1)
    return dp[ind][w]
int nottake = 0 + f(ind-1, w)
int take = Integer.MIN_VALUE;
if (wt[ind] <= w) {
    take = val[ind] + f(ind-1, w - wt[ind])
}
return dp[ind][w] = Math.max(nottake, take);
}
```

Date : / /

Tabulation

```
//Base Case
for(int w=wt[0]; w<=W; w++){
   dp[0][w]=val[0];
}
for(int ind=1; ind<n; ind++){
    for(int w=0; w<=W; w++){
   int notTake=dp[ind-1][w];
   int take=INT_MIN;
    if(wt[ind]<=w){
        take=val[ind]+dp[ind-1][w-wt[ind]];
    }
    dp[ind][w]=Math.max(notTake,take);
}
}
return dp[n-1][W];
```

Date : / /

Space Optimization

```
int [] prev = new int[W+1];
int [] curr = new int[W+1];

for(int w=wt[0]; w<=W; w++){
    prev[w] = val[0];
}
for(int ind=1; ind<n; ind++){
    for(int w=0; w<=W; w++){
    int notTake = 0 + prev[w];
    int take = INT_MIN
    if(wt[ind]<=w){
        take = val[ind] + prev[w - wt[ind]];
    }
    curr[w] = Math.max(notTake, take);
}
prev = curr
}
return prev[W];
```

Date : / /

prev[w - wt[ind]]

This line explain that if i using the elements in the left hand side of the array so we can start the weights from the right i.e., max rather from the left.

ie.. for(int ind=1; ind<n; ind++){

for(int w=W; w>=0, w--){

- - - -

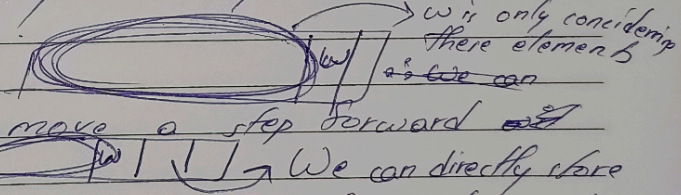
- - - -

}

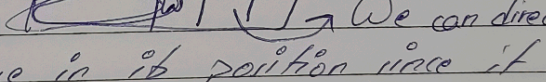
This also works, since it only taking the elements of prev row.

⇒ Since it is taking elements of prev row & if i only considering elements on its left side.

ie..



∴ As we move a step forward



the value in its position since it is not disturbing the elements on its right hand side. Which indicates we can use just 1 array instead of using 2

Date : / /

Optimizing with 1 array

```
int [] prev = new int [W+1]

for(int w = wt[0]; w <= W; w++) prev[w] = val[0];

for (int ind = 1; ind < n; ind++) {
   for (int w = W; w >= 0; w--) {
int notTake = 0 + prev[w];
  int take = INT_MIN;
if (wt[ind] <= w) {
    take = val[ind] + prev[w - wt[ind]];
}
  Prev[w] = max(take, notTake);
}}
return prev[W];
```