**Q) Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don’t pick it (0-1 property).**

So basically, this is choice question as to whether to include item in our knapsack or to exclude it.

Now think when do we actually have the chance of including it? The answer is we can only include the item in our knapsack IFF the weight of the item is <= the weight of the knapsack. When the weight will be <= then we can think of either “including it” or “not including it”. Otherwise we HAVE to Exclude it because the capacity will be full.

Now there are 2 arrays of weights and costs and there is one integer input which represents the max capacity of the knapsack.

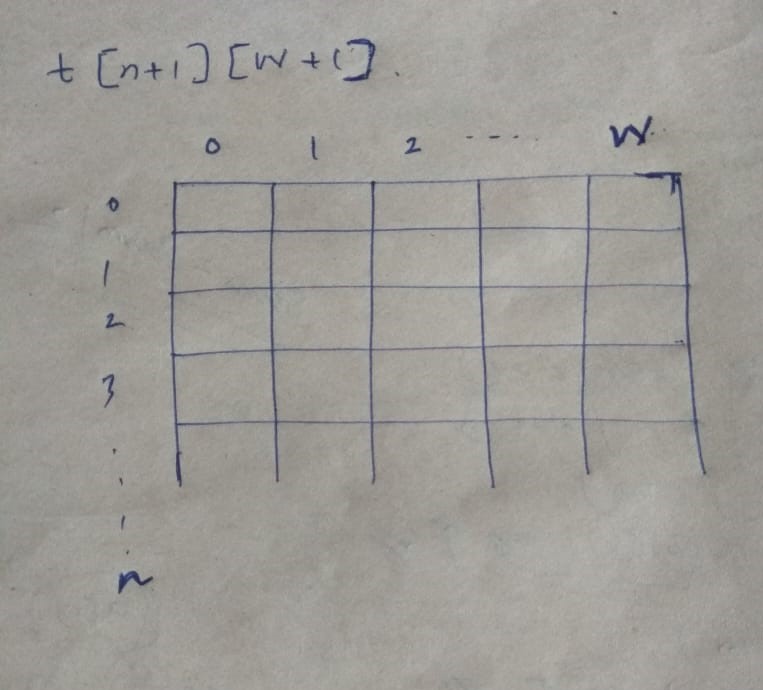
Ok so let’s think if we start from the end of the array and if the weight **w** in **w array** is less that the max weight W then we can include the item, get its price and add it to the dp matrix and reduce the max weight W by that amount else we will just discard the item i.e. we will ofc add it to our matrix but we will no subtract it from our max weight W.

Now here comes the question…… “What is this matrix that I’m talking about?”

Ok so by DP we mean we need a storage place for our results where we can store the values and extract them as and when necessary.

So, let’s make a matrix named t containing n+1 rows and W+1 column where n is the number of elements in the weight/value arrays and W is the max capacity of the knapsack.

We didn’t make a t[n][W] array coz we will then only be able to store from 0 to n-1 and 0 to W-1 values but we need to include n and W that’s why we are doing +1.

Ok so

Why we chose only n and W and not n and V ? the answer is only W and n is changing as we add elements in our knapsack.

Here i is analogous to n[i.e size of array] and j is analogous to W [max capacity]

So here is the code

**public** **class** Knapsack01 {

**public** **static** **int** knapsack(**int**[]w,**int**[]v,**int** n,**int** W)

{

// creation of matrix

**int**[][]t=**new** **int**[n+1][W+1];

//initialization of first row

/\*

this step is called the initialization step.If you think carefully the first row will have a size of matrix =0 which means it can have a maximum profit of 0. Its obvious that if there are no items/cost in the array then the maximum profit will be 0

\*/

**for**(**int** j=0;j<W+1;j++)

{

t[0][j]=0;

}

//initialization of first column

/\*

If you think carefully the first column will have a Max Capacity W of Knapsack =0 which means it can have a maximum profit of 0. Its obvious that if there is max capacity of the bag is 0 then ofc the maximum profit will be 0

\*/

**for**(**int** i=0;i<n+1;i++)

{

t[i][0]=0;

}

/\*

\* Main Logic Part

\*/

/\*So I is analogous to n and j is analogous to W. Ok so the as soon as you encounter a value weight that is <= W is w[i-1]<=j i.e. **w[n-1]<=W** we will now make a choice to either include the element in our bag or discard it. So if we include then we will take it out v[i-1], add it with the previously stored value, decrease the element count t[i-1] and subtract the max capacity by the weight we just added i.e. t[i-1][j-w[i-1]]. Now if we don’t wish to include that then we will decrease the element count, t[i-1] but not subtract the Max capacity since we are not including the item in our sack i.e. t[i-1][j-0]

\*

\*/

**for**(**int** i=1;i<n+1;i++)

{

**for**(**int** j=1;j<W+1;j++)

{

**if**(w[i-1]<=j)

{

t[i][j]=Math.*max*(v[i-1]+t[i-1][j-w[i-1]], 0+t[i-1][j-0]);

}

/\* ofc if the weight is > W then we cant add it hence the else part

\*

\*/

**else** {

t[i][j]=t[i-1][j-0];

}

}

}

/\*

\* the final answer will be stored in the last cell in the matrix coz in the last cell i=n,j= W i.e. all the elements have been processed

\*/

**return** t[n][W];

}

**public** **static** **void** main(String[] args) {

**int**[]w= {1,2,3,30,0,12};

**int**[]v= {10,20,30,30,12,54};

**int** n=w.length;

**int** W=10;

System.***out***.println(*knapsack*(w,v,n,W));

/\*

\* Output 72

\*/

}

}