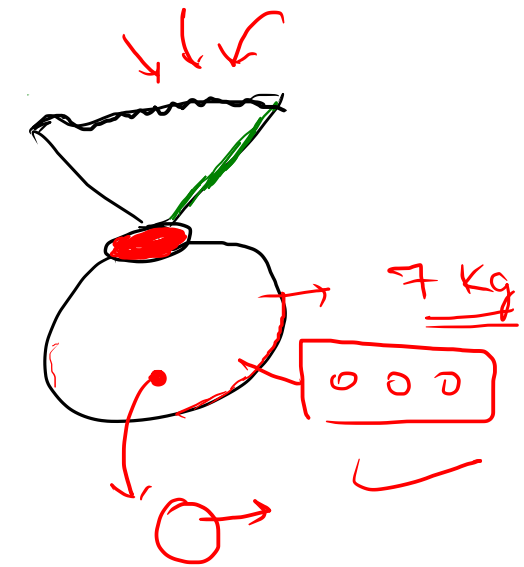
0 - 1 Knapsack problem

- Given Items, I_1 I_2 I_3 ...
- for every Items, we have input given, weight and price
- W = capacity of Bag



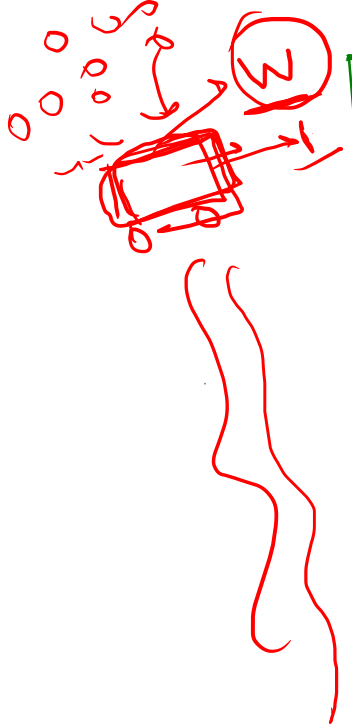
I_1 I_2 I_3 I_4

I/p : wt[] : {1, 3, 4, 5} → 4 items

pr[] : {1, 4, 5, 7}

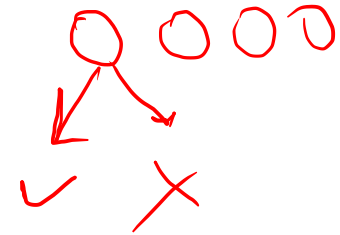
W : 7

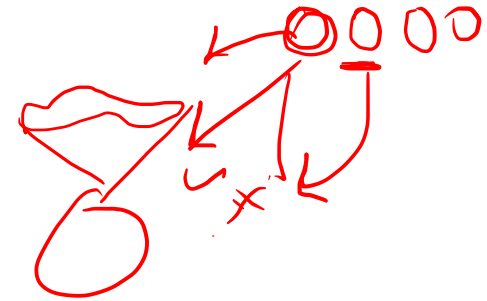
o/p : want to choose an item for bag such that, profit is maximum.



	I_1	I_2	I_3	I_4
Ip: wt[] :	1	3	4	5
pr[] :	1	4	5	7

W : 7

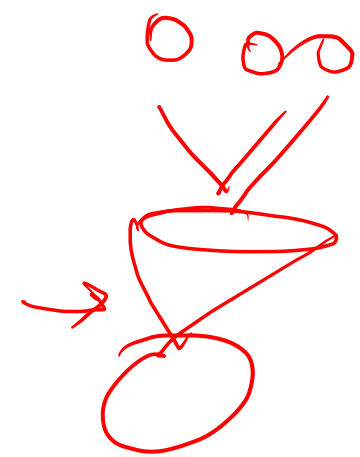




⇒ First identify whether it is DP or not ?

⇒ choice → For every element, we have option either to add it in the bag or not

⇒ optimal → we want to maximise profit.



Dynamic Programming

2 line code add

Recursive Soln → Memoization (DP) → Top-Down (DP)

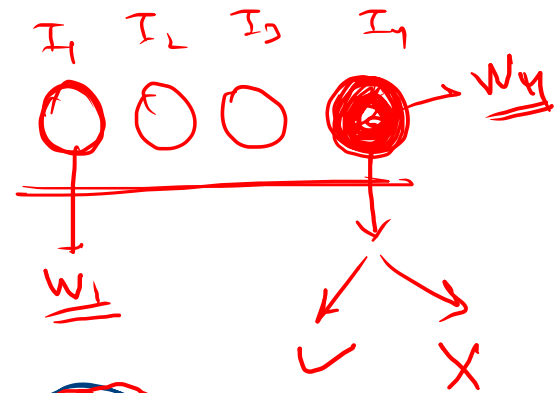
DP = Recursion + Storage

0-1 Knapsack Recursive

DP → Recursion

↓

** choice Diagram



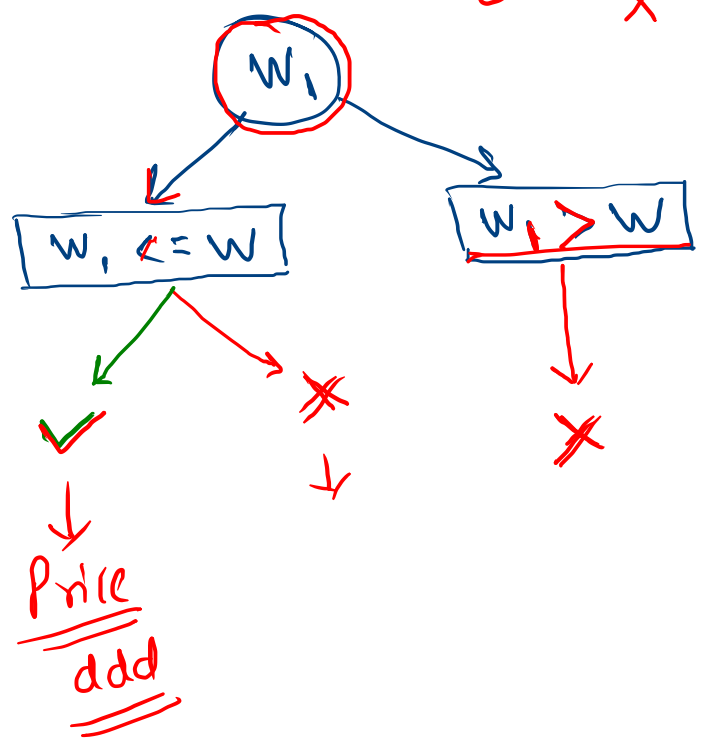
I1 I2 I3 I4

wt[] : { 1, 3, 4, 5 }

pr[] : { 1, 4, 5, 7 }

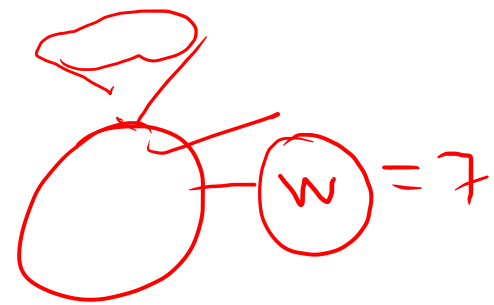
W : 7

capacity



Price

ddd

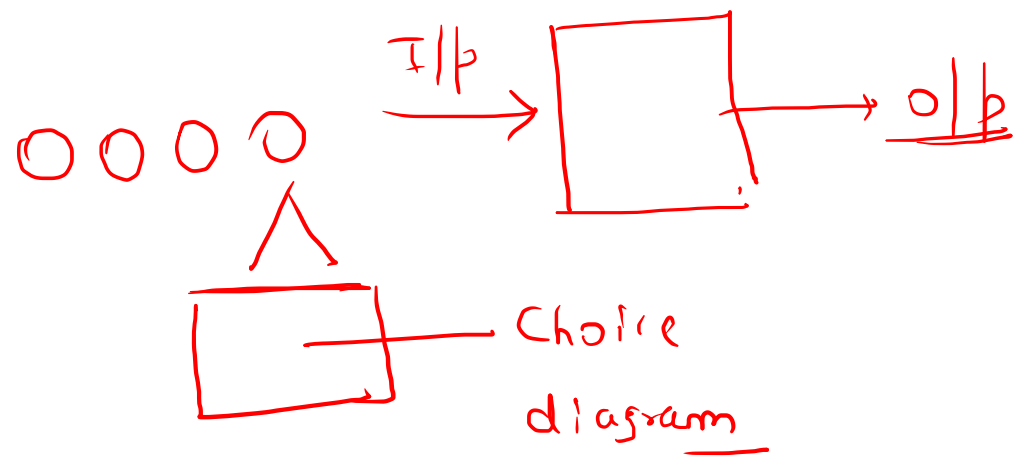


```
int Knapsack ( int wt[], int pr[], int W, int n)
{
    // Base Condition ✓
    // choice Diagram ✓
}
```

Capacity

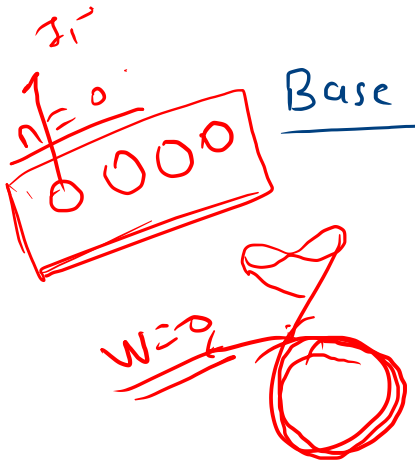
no. of Items ~~in the bag~~ considered.

Recursive function



```
int Knapsack ( int wt[], int pr[], int W, int n)
{
    // Base Condition

    // choice Diagram
}
```

Variables



Base condition : think of the Smallest Valid input

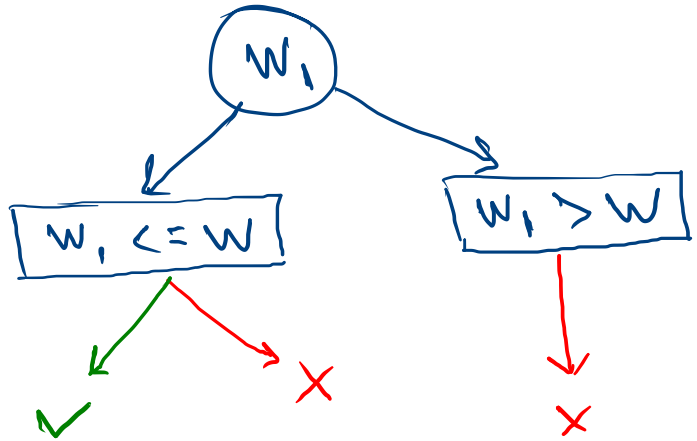
think of max profit at Base condition

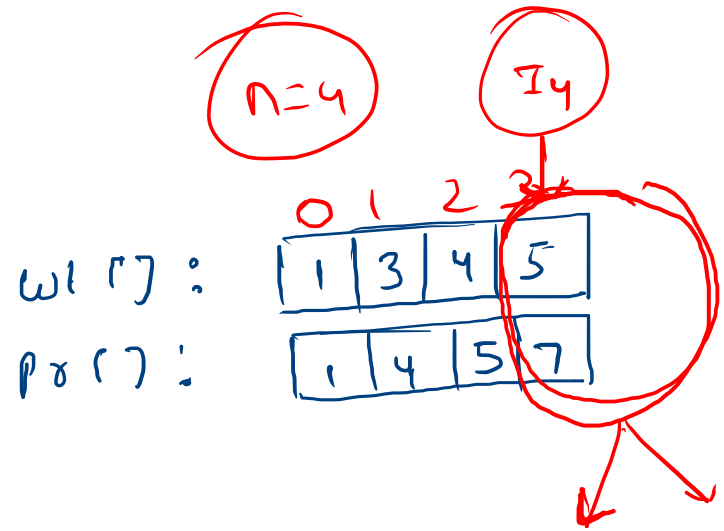

```
if( n==0)  return 0
if( W==0)  return 0
```

```
if ( n==0 || W==0)
  return 0;
```

→ max profit

Code of choice diagram





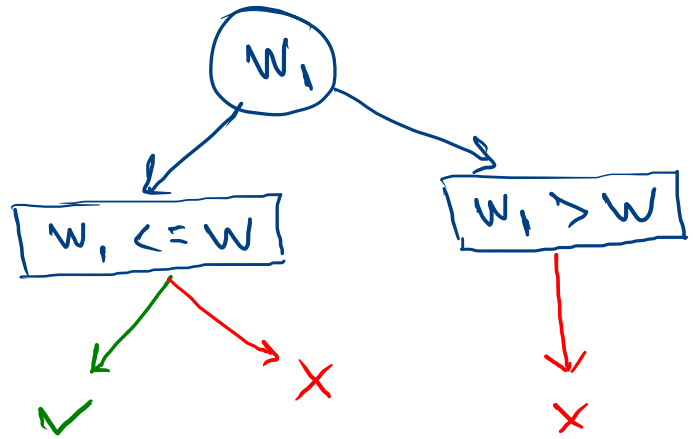
wt[n-1] = wt[4-1]
= wt[3]
W

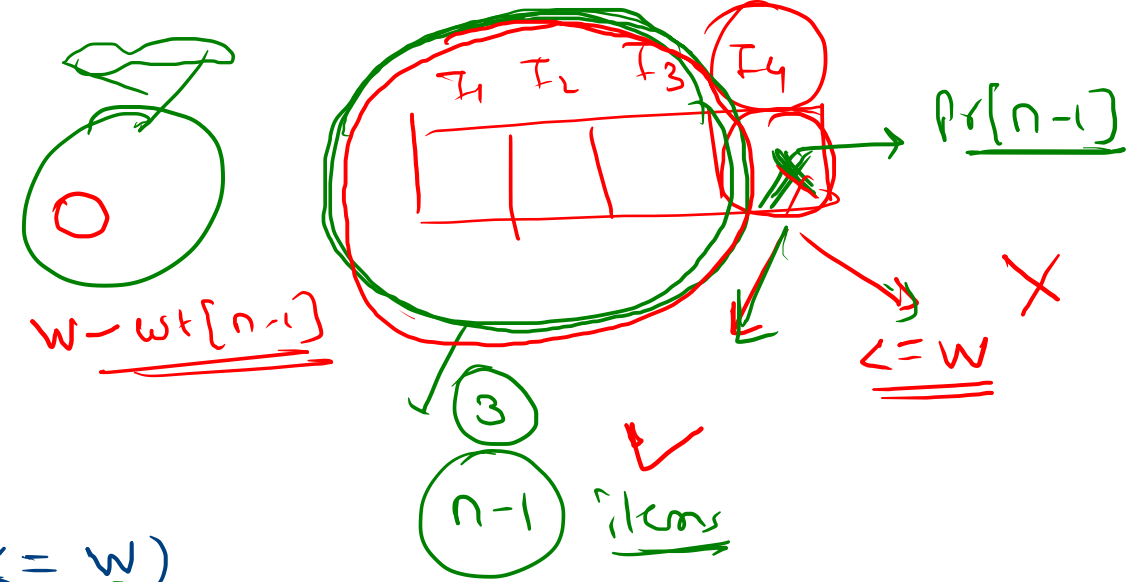
→ pr[n-1]

kp(wt[], pr[], W, n)

↓ no. of Items

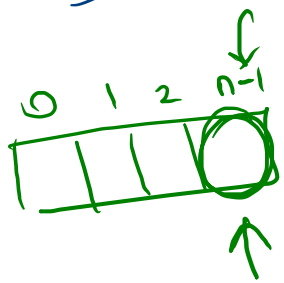
Code of choice diagram



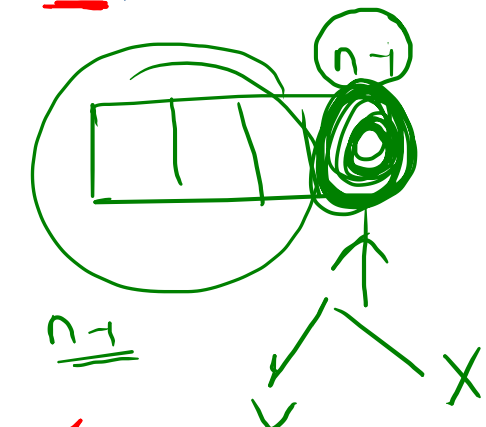



```
if ( wt[n-1] <= W)
return max
( Pr[n-1] + Knapsack ( wt[], pr[], W - wt[n-1], n-1),
  Knapsack ( wt[], pr[], W, n-1) )
else
{
  Knapsack ( wt[], pr[], W, n-1) ;
}
```

Final code



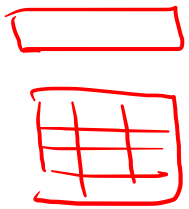

```
int Knapsack ( int wt[], int pr[], int W, int n)
{
  // Base Condition
  if ( n==0 || W==0) return 0;

  // choice Diagram  -> index
  if ( wt[n-1] <= W)
  return max
  ( pr[n-1] + Knapsack ( wt[], pr[], W - wt[n-1], n-1),
    Knapsack ( wt[], pr[], W, n-1) )
  else
  {  return Knapsack ( wt[], pr[], W, n-1) ;
  }
}
```



Convert Recursive function ⟶ Memoization

1D

2D



⇒ we will create matrix of n × m

⇒ Check variable in the iput Paramder of Recursive function

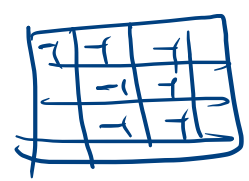
How to identify value of n and m ??

⇒ Knapsack (wt[], pr[], W, n)

↳ These two are changing variables

** int dp[n+1][W+1]

memset (dp, -1, sizeof (dp));

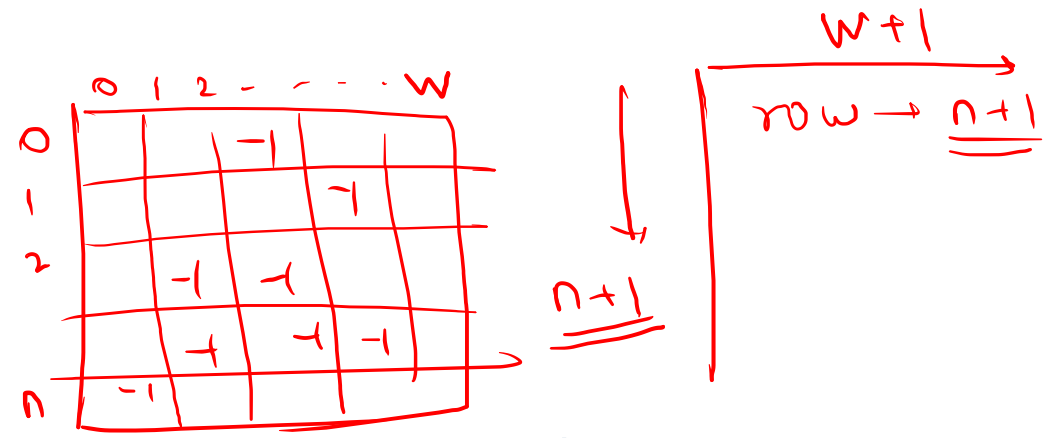


dp[n+1][W+1]

dp[W+1][n+1]

C++

```
vector<vector<int>> dp(n+1, vector<int>(W+1, -1));
```



Vector < vector <int> > dp (n+1, vector <int> (W+1, -1));

Final code

```
int Knapsack ( int wt[], int pr[], int W, int n)
{
    if ( n==0 || W==0) return 0;
    if ( dp[n][W] != -1) return dp[n][W];
    if ( wt[n-1] <= W)
    return dp[n][W] =
    max( pr[n-1] + Knapsack ( wt[], pr[], W - wt[n-1], n-1),
         Knapsack ( wt[], pr[], W, n-1) )
    else
    { return dp[n][W] = Knapsack ( wt[], pr[], W, n-1) ;
    }
}
```

index

dp

no. of items Consids