Date : / /

Unbound Knapsack

You are given weight[] ⇒ weight of item
values[] ⇒ value of item
W = Maximum weight
Each item has infinite supplies (This is the diff btw 0/1 Knapsack & Unbound Knapsack).
Maximize the total value in Knapsack without exceeding capacity W

Sol wt → {2, 4, 6}
val → {5, 11, 13}
W = 10

Sol ① wt-6 (13) + wt-4 (11) = 24

② wt-6 (13) · (wt-2)×2 (5×2=10) = 23

③ (wt→4)×2 (11×2=22) + wt-2 (5) = 27

④ (wt-2) 5 = 25

Maximum possible solution.

Date : / /

(1) f(ind, W) ⇒ Till ind-1 with baggage weight W what will be the maximum value.

(2) Do possible stuff

```
nottake = 0 + f(ind-1, W);
take = INT_MIN;
if(wt[ind] <= W){
  take = val[ind] + f(ind, W - wt[ind])
return Max(take, nottake)
```

When there is infinite supply or can come to same ind hence don't increment ind lean it the same.

Base Cases

If there is only 1 element he should definitely steal it but the no: of times he can steal it matters.

No: of times he can steal = $\frac{W}{wt[0]} \times val[0]$

```
if(ind == 0){
  return (W/wt[0]) * Val[0];
```

Date : / /

Recursion

```
public int f(int ind, int W, int[] val, int[] wt){
  if(ind == 0){
    return (W/wt[0]) * val[0];
  }
  int notTake = f(ind-1, W, val, wt);
  int take=0;
  if(wt[ind] <= W){
    take = val[ind] + f(ind, W-wt[ind], val, wt);
  }
  return Math.max(take, notTake);
}
public .
```

Date : / /

Memoization

```
public int f(int ind, int W, int[] val, int[] wt, int[][] dp){
  if (ind == 0){
      return (W/wt[0]) * val[0];
  }
  if (dp[ind][W] != -1){
      return dp[ind][W];
  }
int notTake = f(ind-1, W, val, wt, dp);
int take = 0;
if (wt[ind] <= W){
  take = val[ind] + f(ind, W-wt[ind], val, wt, dp);
}
return dp[ind][W] = Math.max(take, notTake);
}
```

Date : / /

Tabulation

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

Date : / /

Space Optimization

```
int [] prev = new int[W+1]
int [] curr = new int[W+1]
for(int W=0 ; W<=W; W++) {
  prev[W] = (int)(W/wt[0]) * val[0];
}
for(int ind=1; ind<n ;ind++) {
  for(int W=0; W<=W; W++){
    int nottake = 0 + prev[W];
    int take = 0
    if(wt[ind] <= W) {
      take = val[ind] + curr[W-wt[ind]];
    }
    curr[W] = max(take, nottake);
}
prev = curr;
}
return prev[W];
}
```