ADA Mini HW #4

Student Name: 林楷恩
Student ID: b07902075

(1) **Construct a DP table to fill knapsack with capacity W = 15**

i \ W	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
2	0	3	3	7	10	10	10	10	10	10	10	10	10	10	10	10
3	0	3	3	7	10	13	13	17	20	20	20	20	20	20	20	20
4	0	3	3	7	10	13	15	17	20	22	25	25	29	32	32	32
5	0	3	3	7	10	13	15	17	20	22	25	25	29	32	32	34
6	0	3	3	7	10	13	15	17	20	22	25	25	29	32	32	34
7	0	3	3	7	10	13	15	17	20	22	25	25	29	32	32	34

(2) **Modify the DP algorithm to 0/1 Knapsack problem mentioned in class so that we can find maximum total value with O(W) space.**

Observing the recursive definition of the optimal solution, I find that we only need the previous row of table, and for every w, the index of the value we need must ≤ w. Thus, for every i, if we compute the solution of the current row from larger w to smaller w, we can overwrite the value of dp[w] without concerns about loss of information. All the values we may use for transition are still the same as the previous row in 2D table. Since we no longer need to keep the previous rows at the hand, only a 1D array of size W + 1 is needed, which has space complexity of $O(W)$.

0/1 Knapsack Problem

```
input: N -> the number of objects , W -> the capacity of knapsack
       w[1..N] -> the weight of objects
       v[1..N] -> the value of objects

output: int -> the maximum value where the total weight does not exceed W

function ZOKP(N, W, w, v):
    dp ← an integer array of size W + 1

    // initializing
    for i from 0 to W:
        dp[i] = 0

    // for j < w[i], dp[i][j] = dp[i-1][j], I can just leave them unchanged.
    for i from 1 to N:
        for j from W to w[i]:
            dp[j] = max( dp[j], dp[ j - w[i] ] + v[i] )

    return dp[W]
```