2019 ADA miniHW 4

b07902064 資工二 蔡銘軒

October 21, 2019

(1) We can use the following algorithm to construct the DP table

```
Knapsack:
   Create array DP[N][W] = {0}
   value[N] to store value of objects
   weiget[N] to store weight of objects
   // for convenience, we use 1-based index
   for i in N: //consider the previous i objects
      for j in W:
         if weight[i] > j:
            DP[i][j] = DP[i - 1][j]
         else:
            DP[i][j] = max(DP[i - 1][j], DP[i - 1][j - weight[i]] +
                 value[i])
   return DP[N][W]
```

And the table will be:

i/w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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) We can see that when deciding the value for $DP[i][j]$ for $j \in [0, W]$, we only need information stored in $DP[i-1][j]$ for $j \in [0, W]$. So we can modify the the above algorithm to improve space complexity to $O(W)$

```
Knapsack:
   Create array DP[2][W] = {0}
   value[N] to store value of objects
   weiget[N] to store weight of objects
   // for convenience, we use 1-based index
   for i in N: //consider the previous i objects
      for j in W:
         if weight[i] > j:
            DP[i % 2][j] = DP[(i - 1) % 2][j]
         else:
            DP[i % 2][j] = max(DP[(i - 1) % 2][j], DP[(i - 1) %
                2][j - weight[i]] + value[i])
   return DP[N % 2][W]
```

Reference

b07902028 林鶴哲