20221012 1602

CP - Knapsack

- Given weight 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, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W.

Method 1

- Recursion by Brute-Force algorithm OR Exhaustive Search
- Approach: A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subset whose total weight is smaller than W, From all such subset, pick the maximum value subset

Question 1: Recursive Definition of function

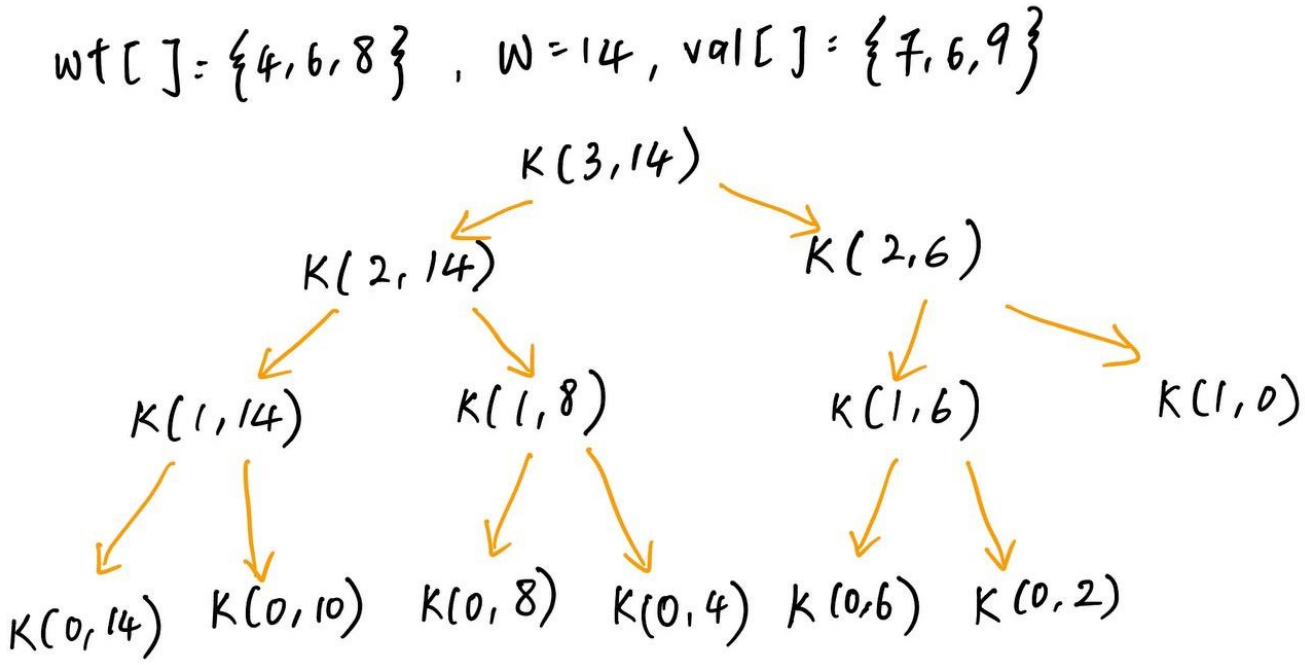
- Therefore, the maximum value that can be obtained from 'n' items is the max of the following two values.
  - Maximum value obtained by n-1 items and W weight (excluding nth item)
  - Value of nth item plus maximum value obtained by n-1 items and W minus the weight of the nth item (including nth item)

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int W, int wt[], int val[], int n){
        if (n==0 || W==0)
                return 0;

        if (wt[n-1]>W)
                return knapsack(W,wt,val,n-1);
        else
                return max(knapsack(W,wt,val,n-1),val[n-1]+knapsack(W-
wt[n-1],wt,val,n-1));
}

int main(){
        int val[] = {7, 6, 9};
        int wt[] = { 4, 6, 8};
        int W = 14;
        int n = 3;
```

```
        cout << knapsack(W,wt,val,n) << endl;
}
```

Time complexity of this naive recursive is O(2^n).
Space Complexity: O(N)

Question 2: Draw the subproblem graph for function(14)



Method 2

- Using dynamic programming with 1-D array

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int W, int wt[], int val[], int n){
        vector <int> ans(W+1,0);

        for (int i=1; i<=n; i++){
                for (int w=W; w>=0; w--){
                        if (wt[i-1]<=w){
                                ans.at(w) = max(ans.at(w), ans.at(w-
wt[i-1])+val[i-1]);
                        }
                }
```

```
        }
        return ans[W];
}

int main(){
        int val[] = {7, 6, 9};
        int wt[] = { 4, 6, 8};
        int W = 14;
        int n = 3;
        int val1[] = {7, 6, 9};
        int wt1[] = { 5, 6, 8};
        cout << knapsack(W,wt,val,n) << endl;
        cout << knapsack(W,wt1,val1,n) << endl;
}
```

Related:

1. Recursion Tree and DAG (Dynamic Programming/DP) - VisuAlgo

---

References:

1. 0-1 Knapsack Problem | DP-10 - GeeksforGeeks