

## Dynamic Programming | Set 13 (Cutting a Rod)

Given a rod of length  $n$  inches and an array of prices that contains prices of all pieces of size smaller than  $n$ . Determine the maximum value obtainable by cutting up the rod and selling the pieces. For example, if length of the rod is 8 and the values of different pieces are given as following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

length		1	2	3	4	5	6	7	8
-----									
price		1	5	8	9	10	17	17	20

And if the prices are as following, then the maximum obtainable value is 24 (by cutting in eight pieces of length 1)

length		1	2	3	4	5	6	7	8
-----									
price		3	5	8	9	10	17	17	20

The naive solution for this problem is to generate all configurations of different pieces and find the highest priced configuration. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

### 1) Optimal Substructure:

We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut.

Let  $\text{cutRod}(n)$  be the required (best possible price) value for a rod of length  $n$ .  $\text{cutRod}(n)$  can be written as following.

$$\text{cutRod}(n) = \max(\text{price}[i] + \text{cutRod}(n-i-1)) \text{ for all } i \text{ in } \{0, 1 \dots n-1\}$$

### 2) Overlapping Subproblems

Following is simple recursive implementation of the Rod Cutting problem. The implementation simply follows the recursive structure mentioned above.

```
// A Naive recursive solution for Rod cutting problem
#include<stdio.h>
```

```

#include<limits.h>

// A utility function to get the maximum of two integers
int max(int a, int b) { return (a > b)? a : b;}

/* Returns the best obtainable price for a rod of length n and
   price[] as prices of different pieces */
int cutRod(int price[], int n)
{
    if (n <= 0)
        return 0;
    int max_val = INT_MIN;

    // Recursively cut the rod in different pieces and compare different
    // configurations
    for (int i = 0; i<n; i++)
        max_val = max(max_val, price[i] + cutRod(price, n-i-1));

    return max_val;
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Obtainable Value is %d\n", cutRod(arr, size));
    getchar();
    return 0;
}

```

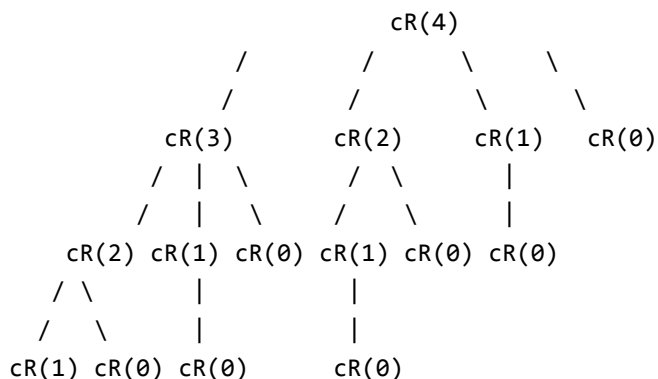
[Run on IDE](#)

Output:

Maximum Obtainable Value is 22

Considering the above implementation, following is recursion tree for a Rod of length 4.

cR() ---> cutRod()



In the above partial recursion tree, cR(2) is being solved twice. We can see that there are many subproblems which are solved again and again. Since same subproblems are called again, this problem has Overlapping Subproblems property. So the Rod Cutting problem has both properties (see [this](#) and [this](#)) of a dynamic

programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array val[] in bottom up manner.

```
// A Dynamic Programming solution for Rod cutting problem
#include<stdio.h>
#include<limits.h>

// A utility function to get the maximum of two integers
int max(int a, int b) { return (a > b)? a : b;}

/* Returns the best obtainable price for a rod of length n and
price[] as prices of different pieces */
int cutRod(int price[], int n)
{
    int val[n+1];
    val[0] = 0;
    int i, j;

    // Build the table val[] in bottom up manner and return the last entry
    // from the table
    for (i = 1; i<=n; i++)
    {
        int max_val = INT_MIN;
        for (j = 0; j < i; j++)
            max_val = max(max_val, price[j] + val[i-j-1]);
        val[i] = max_val;
    }

    return val[n];
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Obtainable Value is %d\n", cutRod(arr, size));
    getchar();
    return 0;
}
```

[Run on IDE](#)

Output:

```
Maximum Obtainable Value is 22
```

Time Complexity of the above implementation is  $O(n^2)$  which is much better than the worst case time complexity of Naive Recursive implementation.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

[84 Comments](#) Category: [Misc](#) Tags: [Dynamic Programming](#)