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 cutRoad(n) be the required (best possible price) value for a rod of lenght n. cutRod(n) can be written as following.

cutRod(n) = max(price[i] + cutRod(n-i-1)) for all i in {0, 1 .. 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;
```

```
}
```

Output:

```
Maximum Obtainable Value is 22
```

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

```
cR() ---> cutRod()

                            cR(4)
                  /        /      \     \
                 /        /        \     \
             cR(3)       cR(2)     cR(1)   cR(0)
            /  |  \       /  \       |
           /   |   \     /    \      |
      cR(2) cR(1) cR(0) cR(1) cR(0) cR(0)
     / \       |          |
    /   \      |          |
 cR(1) cR(0) cR(0)      cR(0)
```

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 suproblems are called again, this problem has Overlapping Subprolems 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;
}
```

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.

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming

Writing code in comment? Please use **ideone.com** and share the link here.

Ashish Maheshwari • 5 days ago

c++ implementation for both naive recursion and DP solution. it also includes function to print cuts made to reach optimal revenue:
https://github.com/maheshwari-...

• Reply • Share ›

beginner • a month ago

```
#include<stdio.h>

#include<limits.h>

int main()

{

int a[] ={3 ,5 ,8 ,9 ,10 ,17 , 17 ,20};

printf("%d",cut(a,8,7,0));

return 0;

}

int cut(int a[],int len,int index,int cost)

{

if(len<0)
```

see more

• Reply • Share ›

Mission Peace • 2 months ago

https://www.youtube.com/watch?... See this video

• Reply • Share ›

Saurabh Singh • 2 months ago

I am unable to understand why we have fixed the first part of the cut? We go on decomposing the second cut again but why not the first one..? Can any one answer?

• Reply • Share ›

Caffrey ➚ Saurabh Singh • 2 months ago

Yes you are right in thinking that the first cut can also be decomposed, just as the second one. It will give the same result. But, and here's the catch, it is not necessary.

Consider a rod of length 4. What does the current solution do?
4, 0 // No Cut
1 + (3)
2 + (2)
3 + (1)

where (i) implies that we take the optimal solution for i.

where (1) implies that we take the optimal solution for i.

If we do what you suggest we get,

(1) + (3)
(2) + (2)
(3) + (1)

which is basically doing repetitions, since (1) + (3) and (3) + (1) here would be doing the same thing.

see more

1 • Reply • Share ›

Eknoor • 3 months ago

/***Another Implementation using O(n^2) space complexity **/
#include<iostream>

#include<cstdio>
using namespace std;

#define MAX 1000

int dp[MAX][MAX];

int price[]={1, 5, 8, 9, 10, 17, 17, 20};

int max_value(int n,int prevlength)

{

if(n==0)

{dp[n][prevlength]=0;

see more

• Reply • Share ›

Arslan Ali Khan • 3 months ago

does anyone have a clue how this problem may be solved using iterative method besides recursion and dynamic programming

• Reply • Share ›

Waqas Yousaf Arslan Ali Khan • 3 months ago

ni honay wala gareeb ....=D

• Reply • Share ›

Dipankar Jana • 4 months ago

We don't need to assign int max_val = INT_MIN
for every iteration in the second code, only once before the loop will do fine.

• Reply • Share ›

Dipankar Jana • 4 months ago

A nice explanation of how to form the recursive substructure of this problem.
https://www.youtube.com/watch?...

• Reply • Share ›

DS+Algo=Placement • 4 months ago

Guys, please tell me if I am wrong,
Why are we taking price[j]+val[i-j-1], To choose optimal solution, we should take both the prices optimal i.e. val[j+1]+val[i-j-1].
I mean,
the inner loop should be like this:

```
for (j = 0; j <= (i/2); j++)
max_val = max(max_val, val[j+1] + val[i-j-1]);
```

• Reply • Share ›

sandeep ➚ DS+Algo=Placement • 4 months ago

u should set max_val to price[i-1] before the inner for loop then it's correct

• Reply • Share ›

sandeep ➚ DS+Algo=Placement • 4 months ago

both give the same answer

• Reply • Share ›

sandeep ➚ DS+Algo=Placement • 4 months ago

i

• Reply • Share ›

guest • 8 months ago

then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

shouldnt it be 25?

1 • Reply • Share ›

kaushik Lele ➚ guest • 7 months ago

Sentence is bit confusing; author meant ->
"..by cutting it into two pieces. One of length 2 and other of length 6. So total length 2+6=8 which is same as original size. And price is 5+17=22

• Reply • Share ›

sneha • 9 months ago

why is it price[i]+val[i-j-1]

• Reply • Share ›

kaushik Lele ➚ sneha • 7 months ago

array index starts with 0. But price[0] indicates piece of length 1.
price[1] indicates piece of length 2. and so on.

So price[j] indicates piece of length (j+1).

So it is actually
price[j] + val[i-(j+1)]

Simplifying it will become
price[j] + val[i- j -1)]

2 • Reply • Share ›

Priyal Rathi • 9 months ago

We should initialise max_val to price[i]. For the case where price[i] is maximum and there is no need to cut the rod into pieces.

Link: http://ideone.com/1ePb8l

1 • Reply • Share ›

piya ➚ Priyal Rathi • 6 months ago

you should initialize DP[0]=0;
to avoid garbage in index 0....

• Reply • Share ›

its_dark ➚ Priyal Rathi • 6 months ago

if the code handles the case for not cutting at j=i-1, then why do we have to initialize max_val to price[i] ?

and price[i] refers to length i+1, not i, right?

• Reply • Share ›

Rahul Gandhi ➚ Priyal Rathi • 9 months ago

The loop for j starts from 0. Which means that the first cut that we consider is of size 0 which essentially is the same as not cutting the rod.

• Reply • Share ›

Priyal Rathi ➚ Rahul Gandhi • 9 months ago

In the above solution Not cutting the rod case is handled for j=i-1, where

max_val = max(max_val, price[i-1] + val[0]); (val[0]=0 and price[i-1] is the price of rod of len i without cutting)

for eg: for i=3 price[0]=1; price[1]=5 and price[2]=8 => for rod of len 3 price[2] is the price without cutting the rod

For j=0 price[0] is the price of rod of length 1

1 • Reply • Share ›

Rahul Gandhi ➚ Priyal Rathi • 9 months ago

Even then, why would you initialize max_val to price[i], it would work with max_val initialized to int_min. Wouldn't it?

• Reply • Share ›

Rahul Gandhi ➚ Priyal Rathi • 9 months ago

Ah! Got it. Sorry, my bad. Thanks for correcting.

• Reply • Share ›

Vãîbhåv Joshî • 10 months ago

u can use max =price[i] instead of Integer.MAX_VALUE
http://ideone.com/s6Mc1c

• Reply • Share ›

Deen Bandhu Agarwal • 10 months ago

A simple solution !!!
just find that which part has maximum value per inch then calculate how many pieces of that length can be made with the size given to us and iterate the same logic until total length get over
this is my c++ implementation
http://ideone.com/82xes4

• Reply • Share ›

Hodor ➚ Deen Bandhu Agarwal • 10 months ago

Had this been greedy scenario in which fractional lengths are allowed then your solution would be correct.

1 • Reply • Share ›

Ayush Jain ➚ Deen Bandhu Agarwal • 10 months ago

Hello DB
You have applied Fractional Knapsack Algorithm while this problem is similar to 0-1 Knapsack.
So your program will give incorrect results in many cases. One such case is

value[]={1,48,72,100,1} and rod of length=5 is to be cut.

So your program will give output of 101 as maximum value while the maximum value can be obtained is 120 (by cutting in pieces of 2 and 3) but because of fractional knapsack algo as in your case the program will choose rod of length=4 as best because it's value/weight is larger than any other element which is incorrect in case of 0-1 knapsack.

1 ▲ | ▼ • Reply • Share ›

Deen Bandhu Agarwal ➚ Ayush Jain • 10 months ago

Thanks Ayush i got it :)

▲ | ▼ • Reply • Share ›

Shirsh Zibbu • a year ago

isn't this similar to the 0-1-knapsack problem?
not *same*, just similar.

▲ | ▼ • Reply • Share ›

Ayush Jain ➚ Shirsh Zibbu • 10 months ago

Yes it is.

▲ | ▼ • Reply • Share ›

Amit Kumar • a year ago

Good example. But I think something wrong with the output

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

For length 4, the price given by program is 10, which should be 9.

1 ▲ | ▼ • Reply • Share ›

alext ➚ Amit Kumar • a year ago

why is that? cut length-4 rod into 2 pieces of length-2 rod, which is 10

4 ▲ | ▼ • Reply • Share ›

prashant • a year ago

/*

int fun(int p[],int pl[],int n)

{

int k,max=0;

if(n==1)

return pl[0];

```
if(n==0)

return 0;

if(p[n]!=-1)

return p[n];

for(int i=1;i<=n;i++)
```

see more

• Reply • Share ›

arhtu • a year ago

```
public static int cutRod(int[] Price, int n, int[] OP)
{

int max_value = Price[n - 1];
//base
if (n == 1)
{
OP[0] = Price[0];
return Price[0];
}
//Just go thro half the length, eg for length=7 as(3+4) is the same as (4+3)
int toLength = n / 2;
for (int i = 0; i < toLength; i++){
if (OP[i] == 0) cutRod(Price, ((n - 1) - i), OP);
max_value = Math.Max(max_value, OP[i] + OP[(n - 2) - i]);//more efficient as you dont have to
calculate the sbuoptimal soulutions as we store it
}
OP[n - 1] = max_value;//store the suboptimal solutions
return max_value;
}
```

• Reply • Share ›

arhtu • a year ago

```
public static int cutRod(int[] Price, int n, int[] OP)

        {
```

```
int max_value = Price[n - 1];



if (n == 1)

            {

                OP[0] = Price[0];
```

see more

• Reply • Share ›

Sourin Sutradhar • a year ago

#include<iostream>
using namespace std;
int cutRod(int *arr, int size)
{
int max = 0;
if(size == 0)
return 0;
for(int i = 0; i < size; i++)
max = (arr[i]+cutRod(arr, size-i-1)>max)? arr[i]+cutRod(arr, size-i-1):max;
return max;
}
int main()
{
int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};
int size = sizeof(arr)/sizeof(arr[0]);
cout<<"\nMaximum Obtainable Value is " << cutRod(arr, size)<<endl; return="" 0;="" }="">

1 • Reply • Share ›

Vijay Apurva • 2 years ago

Shiwakant Bharti i have tested it . if we repeat inner loop till i then we swap every value twice . we can reduce it by making inner loop till i/2+1

• Reply • Share ›

Shiwakant Bharti • 2 years ago

@Vijay: This looks correct to me. Not tested it though.

• Reply • Share ›

Anas Mourad • 2 years ago

Knapsack

Knapsack

• Reply • Share ›

Vijay Apurva • 2 years ago

we can do better.
we can reduce 2nd loop up to i/2 + 1.

#include<stdio.h>
int max(int a, int b).
{return (a > b)? a : b;}.

int cutRod(int arr[], int n){.
int temp[n], i,j;.
for(i=0;i<n;i++).
temp[i]=arr[i];.

for(i=1;i<n;i++)
for(j=0;j<i/2+1;j++).
if(temp[i]<temp[j]+temp[i-j-1]).
temp[i]=temp[j]+temp[i-j-1];.

return temp[n-1];.
}

see more

• Reply • Share ›

Chaitanya • 2 years ago

In the given recurrence relation you have excluded the role of 'l' (lengths array).

There are two cases possible
(1) ith length cut can be done
(2) ith lenght cut can not be done

curRod(N, n) = Max {
cutRod(N-L[i], n-i-1) + P[i], --- (1)
cutRod(N, n-i-1) ---------- (2)
}; 0 <= i < n

[sourcecode language="C"]
//p is price array, l is lengths array
int cutRod(int *p, int *l, int N, int n)
{
printf("R-%d--%d\n", N, n);
if(N <= 0)
return 0;

see more

• Reply • Share ›

Tushar Singhal • 2 years ago

in the above case why not max_val = max(max_val, price[j] + val[i-j-1]); be replaced by max_val = max(max_val, val[j]+val[j-i]);.

max_val = max(max_val,

5 • Reply • Share ›

Sadiqin Ibnu Adam • 2 years ago

In UVa/ICPC problem, what code/id for this problem?

• Reply • Share ›

ganglu • 2 years ago

u can see a video explanation of the algorithm on the link below

http://www.youtube.com/watch?v...

2 • Reply • Share ›

kartheek • 2 years ago

/* 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+i, n-i-1));

return max_val;
}

• Reply • Share ›

kartheek • 2 years ago

```
/* 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)
```

```
    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+i, n-i-1));

    return max_val;
}
```

THe recursive function should work on the new array (price + i) which is a part of the smaller one is it! Correct me if I'm wrong

• Reply • Share ›

gautamvs • 3 years ago

This is solution I have written:

```
 /* Paste your code here (You may delete these lines if not writing code) */

public class RodDivision {

        public static int divideRod(int[] values, int[] answers, int n){
                if(n==0) return 0;
                if(answers[n]!=-1) return answers[n];
                int max=-1, ans;
                for(int i=1; i<=n; i++){
                        ans=values[i]+divideRod(values, answers, n-i);
                        if(ans>max)
                                max=ans;
                }
                answers[n]=max;
                return max;
        }
```

see more

• Reply • Share ›

NIPUN • 3 years ago

```
 #include<stdio.h>
int cutRod(int price[],int n)
{
int max_val= max((n+1)*price[0],price[n]);
```

```
int i,max_sofar=0;

for(i=0;i<n;i++)
{
 max_sofar = price[i]+price[n-i-1];
    if(max_sofar >max_val)
        max_val = max_sofar ;
}
return max_val;
}
int max(int a, int b) { return (a > b)? a : b;}
int main()
{
    int arr[] = {2, 5, 8, 9, 10, 17, 17, 20};
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Obtainable Value is %d\n", cutRod(arr, size-1));
    getchar();
    return 0;
}
```

-
-
-
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
-

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- Follow @GeeksforGeeks

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

because we first fill zero in first col and...

Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago