GSourceCode

GSOURCECODE

INTRODUCTION TO DYNAMIC PROGRAMMING – CUTTING RODS

12 Apr 2012 · *by* MGhareeb · *in Algorithms, Dynamic Programming, Problems, Utilities · 2 Comments*

2 0 Rate This

This problems is presented in **Introduction to Algorithms** as an intro to *Dynamic Programming*.
Given a rod of length `n` inches and a table of prices p_i for rod lengths: `i = 1, 2, ... n`, determine the maximum revenue r_n obtainable by cutting up the rod to pieces and selling them.
The table looks like this

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

A naive solution could be

```
CUT-ROD(p, n)
q = 0
for i = 1 to n
        q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

This solution has exponential asymptotic time. So, we're introduced to dynamic programing.
The method works as follows:

- We rearrange for each subproblem to be solved only once.
- If we need to refer to this subproblem's solution again later, we can just look it up in a hash table or an array.

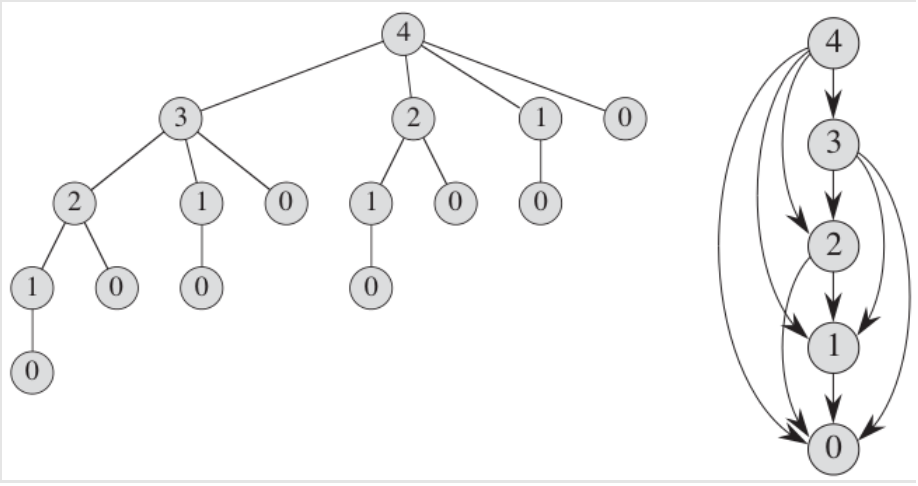
The modified version of the previous algorithm is:

```
CUT-ROD(p, n)
if CUT-ROD(p, n) is solved before
// use previous solutions
        return s[n]
q = 0
for i = 1 to n
        q = max(q, p[i] + CUT-ROD(p, n - i))
s[n] = q
// store the new solution
return q
```

There's a *top-down* approach and a *bottom-up* approach:

- The **top-down** approach recurses from larger problems to smaller subproblems until it reaches a pre-calculated subproblem. After that, it returns then combines the solutions of subproblems to solve the larger ones. The previous algorithm is top-down.
- The **bottom-up** method, as you can tell from its name, solves all the required subproblems first then the larger ones. Both methods are $O(n^2)$ but the bottom-up way has better constants.

To get a clear insight of what the difference is, see the following subproblem tree:



The subproblem graph for the rod-cutting problem with n = 4.
The vertex labels give the sizes of the corresponding subproblems.
A directed edge (x, y) indicates that we need a solution to subproblem y when solving subproblem x.
To the left is the naive solution.
To the right is the DP one.

Implemented below in `C++`.

```
#include <iostream>
#include <cstring>
using namespace std;

const int N = 1000;
int p[11];
int r[N], s[N];
```

- Android
- Data Structure
- Desktop Applications
- J2ME
- Operations Research

```
 8
 9  // initializer for prieces and optimal solutions
10  void init() {
11      memset(r, -1, N);
12      r[0] = 0;
13      p[0] = 0;
14      p[1] = 1;
15      p[2] = 5;
16      p[3] = 8;
17      p[4] = 9;
18      p[5] = 10;
19      p[6] = 17;
20      p[7] = 17;
21      p[8] = 20;
22      p[9] = 24;
23      p[10] = 30;
24  }
25
26  // naieve exponential solution
27  int cutRod(int n) {
28      int q = 0;
29      for (int i = 1; i <= n; ++i)
30          q = max(q, p[i] + cutRod(n - i));
31      return q;
32  }
33
34  // top-down solution
35  int topDownCutRod(int n) {
36      if (r[n] != -1)
37          return r[n];
38      int q = 0;
39      for (int i = 1; i <= n; ++i)
40          q = max(q, p[i] + topDownCutRod(n - i));
41      return r[n] = q;
42  }
43
44  // bottom-up solution
45  int buttomUpCutRod(int n) {
46      if (r[n] != -1)
47          return r[n];
48      for (int j = 1; j <= n; ++j) {
49          int q = 0;
50          for (int i = 1; i <= j; ++i)
51              q = max(q, p[i] + r[j - i]);
52          r[j] = q;
53      }
54      return r[n];
55  }
56
57  // bottom-up solution that maintains not only the
58  int extendedButtomUpCutRod(int n) {
59      if (r[n] != -1)
60          return r[n];
61      for (int j = 1; j <= n; ++j) {
62          int q = 0;
63          for (int i = 1; i <= j; ++i)
64              if (q < p[i] + r[j - i]) {
65                  q = p[i] + r[j - i];
66                  s[j] = i;
67              }
68          r[j] = q;
69      }
70      return r[n];
71  }
72
73  // prins the extended method's output
74  void printCutRodSoln(int n) {
75      do
76          cout << s[n] << " ";
77      while ((n -= s[n]) > 0);
78  }
```

```
79
80  int main() {
81      init();
82      int n;
83      cin >> n;
84      cout << extendedButtomUpCutRod(n) << endl;
85      printCutRodSoln(n);
86      return 0;
87  }
```


You May Like

- 1.

Share

Related

10943 - How do you add?
In "Algorithms"

Maximum Subarray - Kadane
In "Algorithms"

1213 - Sum of Different Primes
In "Algorithms"

Tags: algorithms, analysis, bottom, brute force, cut, cutting, down, dp, dynamic, introduction, programming, rod, rods, subproblem, top, up

2 comments

638 – Let Me Count The Ways | · 28 Apr 2012 - 8:12 pm · *Reply→*

[...] – Let Me Count The Ways Posted on 28 April 2012 by MGhareeb32 A coin change Dynamic Programming problem. A buttom-up approach C [...]

Kaidul · 15 May 2013 - 1:59 pm · *Reply→*

Nice explanation. I was looking for help for UVA – 10003 and found it. Carry on!

std::cout <<

Enter your comment here...

← Flood fill and area calculation

11703 Sqrt Log Sin →

TOP 5

- 11703 Sqrt Log Sin
- Introduction to Dynamic Programming - Cutting Rods
- 272 - TEX Quotes
- 10684 - The jackpot
- gsourcecode

BLOG STATS

- 31,079 hits

AUTHORS

MGhareeb

Ad-hoc Algorithms Android Binary Search CompleteSearch Data Structure Desktop Applications Dynamic Programming Geometry Graph Heuristic Search J2ME Math Operations Research Problems Utilities

META

- Register
- Log in
- Entries RSS
- Comments RSS
- Blog at WordPress.com.