**ALGORITHM ASSIGNMENT**

**NAME: DEEPTI SHARMA**

**ROLLNO: 2014315**

**CH: 15 DYNAMIC PROGRAMMING**

**TOPIC: 15.1: ROD CUTTING**

***Question 1***: Show that equation (15.4) follows from equation (15.3) and the initial condition T (0) =1.

***Answer:*** The equation given 15.3 states that:

The initial 1 is for the call at the root, and the term T (j) counts the number of calls (including recursive calls) due to the call CUT-ROD (p, n-i) where j=n-i.

It is given that the initial condition is, T (0) =1.

T (1) =1+T (0) =2

T (2) =1+T (1) +T (0)

T (3) =1+ T (2) +T (1) +T (0)

Therefore it is observable that,

T(n)=1+(T(0)+ T(1)+ T(2)+ T(3)+ T(4)……………………+T(n-1))

T(n)-T(n-1)= 1+(T(0)+ T(1)+ T(2)+ T(3)+ T(4)……………………+T(n-2))

=T (n-1)

Implies,

T (n) =T (n-1) +T (n-1)

=2\*T (n-1)

Solving by substitution method,

T (n) =2\*(2\*T (n-2))

=2\*(2\*(2\*T (n-3))

=2(2(2(2……………………2(T (n-k))))

Assuming that, n-k=0

Then, n=k.

T (n) =2k (T (0))

=2n (1)

Hence, T (n) =2**n,** as stated in equation 15.4.

***Question 2:*** Show, by means of a counter example, that the following “greedy” strategy does not always determine an optimal way to cut rods. Define the ***density*** of a rod of length i to be p/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where 1 <= i <= n having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n - i.

***Answer:*** As per the **greedy approach**, one must simply select the length i where the density (p/i) is found to be maximum.

While, **the dynamic approach**, suggest that we should try out all possible examples and store the maximum possible revenue.

But the greedy approach might not be an optimal approach, in case some other combination other maximum density one yields better revenue.

Take an example,

Suppose we have n=6, that is the length of the rod to be cut

And p [] denotes a price table for various length

Let p[ ]={0,0,3,5,0,0}

Now as per **greedy approach**, the rod shall be cut into 2 parts, one of length 4(maximum density =5/4=1.25) and the other of length 2, because that will be the only option left.

**Hence the revenue generated =5+0=5**

But if we move according to the **dynamic approach**, then we get an optimal solution that the rod must be cut into 2 parts, each of length 3,

**Hence the revenue generated =3+3=6**

Hence, by this counter example, we got an idea that the dynamic approach for solving this problem is a much better option for an optimal solution.

***Question 3:*** Consider a modification of the rod-cutting problem in which, in addition to a price pi for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

***Answer:***

MODIFIED-EXTENDED-BOTTOM-UP-CUT-ROD(p, n, c)  
1 let r[0..n] and s[0..n] be new arrays  
2 r[ 0 ] = 0  
3 for j = 1 to n  
4     q = p[ j ]  
5     s[ j ] = j  
6   for i = 1 to j - 1  
7         if q < p[ i ] + r[ j - i ] - c  
8             q = p[ i ] + r[ j - i ] - c  
9             s[ j ] = i  
10   r[ j ] = q  
11 return r and s

***Question 4:*** Modify MEMOIZED-CUT-ROD to return not only the value but the actual solution, too.

***Answer:***

MODIFIED-MEMOIZED-CUT-ROD(p, n)  
1 let r[0..n] and s[0..n] be a new array  
2 for i = 0 to n  
3   r[i ] = -INF   
4 return MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n, r, s)  
  
MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n, r, s)  
1 if r[n] >= 0  
2     return r[n]  
3 if n == 0  
4     q = 0  
5 else q = -INF  
6     for i = 1 to n  
7        if q < p[i] + MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n-i, r, s)  
8            q = p[i] + MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n-i, r, s)  
9            s[n] = i  
10   r[n] = q  
11 return q and s  
  
PRINT-CUT-ROD-SOLUTION(p, n)  
1 (r, s) = MODIFIED-CUT-ROD(p, n)  
2 while n > 0  
3     print s[n]  
4     n = n - s[n]

***Question 5:*** The Fibonacci numbers are defined by recurrence. Give an O (n)-time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the sub problem graph. How many vertices and edges are in the graph?

MEMOIZED-FIBONACCI

1. If(n == 0)
2. Return M[0]
3. If(n == 1)
4. Return M[1]
5. If( Fib(n-2) is not yet called )
6. Call Fib(n-2)
7. If (Fib(n-1) is not yet called )
8. Call Fib(n-1)
9. M[n]=M[n-1]+M[n-2]
10. Return M[n]

This algorithm has a linear running time that is, O (n).

Sub problem graph when n=5.

There are n vertices and each vertex has two outgoing edges.

The number of computations to be executed is (n-2)\*2 +2. (The value of Fib (1) and Fib (0) is known, which gives O (1)-time complexity.)

Therefore, time complexity is O (n).