***Dynamic Programming***

***Definition:***

Dynamic Programming is used to combine the solution of sub - questions just as Divide and Conquer Algorithm.

* Divide and Conquer Algorithm - Divide question into several independent sub-questions, solve these questions and get results of these questions recursively, combine all solutions and get the final original result.

*(Divide and Conquer Algorithm needs to solve the public sub-questions for multi-times.)*

* Dynamic Programming Algorithm - Used in the situation where there have overlapping sub-questions, which is to say the algorithm solves the problem once and save the result in the table, no need to recalculate the question when meets with a sub-question, but just check the table for the result.

*(Dynamic Programming Algorithm can be used to escape the situation that recalculate the results of sub-questions.)*

***Scenario - Optimization Problem***

These kind of problems have a lot of doable solutions, however, each solution only has one value. We hope to find *Optimization Solution* (Maximum or Minimum). This kind of solution is call *an Optimal Solution* but not the Optimal Solution, since there may have several solutions can get the Best Result.

***Four Steps for Dynamic Programming Algorithm***

1. Design Optimized Structure Feature.
2. Define Optimization Solution Recursively.
3. Calculate Optimized Solution, normally by using from Bottom to Up Method.
4. Construct Final Optimized Solution by Calculated Information.

***Steer Bar Cutting***

The first example of Dynamic Programming Algorithm is to solve ***How to Cut Steer Bar***. Serling Company buy long Steel Bar and cut it to short Steel Bar and sell out. However, there has no extra cost for Cutting Steer Bar while company wants to know the best Cutting Steer Bar.

***The Price Table:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Len-i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Pri-pi** | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 | 24 | 30 |
| **Re-i** | 1 | 5 | 8 | 10 | 13 | 17 | 18 | 22 | 25 | 30 |

***The problem can be described as below:***

Given one length n inches steer bar and a price table pi(i = 1, 2, 3, ..., n), ask to get the Steer Bar Cutting Schema, make the selling profits rn the most value. ***Attention:***

If the *length n steer bar* has *the biggest cost pn* as long as the Optimized Solution will no need to cut.

*With length n Steer Bar, we have 2 ^ (n - 1) choices.* For the most left side inch, we can choose to cut or not.

***Solution Description:***

Generally, we can use the shorter steel bar below to describe:

***rn = max(pn, r1 + rn-1, r2 + rn-2, r3 + rn-3, ... , rn-1 + r1)***

Here, in the formulation, pn means that the steel bar will not be cut and the profit will be got. Others, r1 means that the steel bar will be cut in the first inch and rn-1 will be the remaining profits. Because we can not make sure which i will helps get the maximum profit, then enumeration will helps.

In order to get the final Optimization Solution, we need to divide the whole Steel Bar into two independent one and get the Optimization Solution for two independent Steel Bar. Finally, we can get the simpler mathematics version:

***rn = max(pi + rn-i) (1 <= i <= n)***

***According to the Mathematics Expression, then we can get all Maximum Profits.***

|  |  |
| --- | --- |
| Number | Maximum Cost |
| 1 | 1 |
| 2 | Max(2, 5) = 5 |
| 3 | Max(3, 6, 8) = 8 |
| 4 | Max(9, 10, 9) = 10 |
| 5 | Max(11, 13, 10) = 13 |
| 6 | Max(14, 15, 16, 17) = 17 |
| 7 | Max(18, 18, 18, 17) = 18 |
| 8 | Max(19, 22, 21, 20) = 22 |
| 9 | Max(23, 23, 25, 23, 24) = 25 |
| 10 | Max(26, 27, 26, 27, 26, 30) = 30 |

***Code:***

***Let r [ n ] be a new array.***

***Initialize r[ n ] = {0, -8, -8, ..., -8};***

***FOR ( i = 1; i <= n; i++ )***

***{***

***q = -8;***

***FOR ( j = 1; j <= i; j++ )***

***{***

***IF ( q <= p[ j ] + r[ i - j ] )***

***{***

***q = p[ j ] + r[ i - j ];***

***}***

***}***

***r[ i ] = q;***

***}***

***return r[n];***