An Introduction To Dynamic Programming

S.M Ahsanul Kabir Kowshid
Student ID : 1505102

Waqar Hassan Khan
Student ID : 1505107

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
(BUET)
Dhaka 1000
July 19, 2018

Contents

1 Introduction

Dynamic programming is a method that in general solves optimization problems that involve making a sequence of decisions by determining, for each decision, subproblems that can be solved in like fashion, such that an optimal solution of the original problem can be found from optimal solutions of subproblems.
This method is based on Bellman's Principle of Optimality, which he phrased as follows

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

More succinctly, this principle asserts that "optimal policies have optimal sub-policies." That the principle is valid follows from the observation that, if a policy has a sub-policy that is not optimal, then replacement of the subpolicy by an optimal sub-policy would improve the original policy. The principle of optimality is also known as the "optimal substructure" property.

So the main properties of **DP** is as follows

- Overlapping Subproblems
- Optimal Substructure

1.1 Overlapping Subproblems

The subproblems of a problem may overlap while solving using **Divide and Conquer**, that is we may end solving the same thing over and over again. In DP we keep track of the solved sub-problems and avoid solving them more than once. Once solved it is stored to use for farther use.

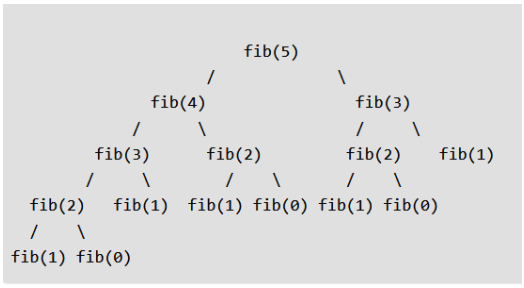

Figure 1: Overlapping Subproblems in determination of Fibonacci series

1.2 Optimal Sub-Structure Property

A given problem has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems. For example, the Shortest Path problem has following optimal substructure property: If a node x lies in the shortest path from a source node u to destination node v then the shortest path from u to v is combination of shortest path from u to x and shortest path from x to v. The standard All Pair Shortest Path algorithms like Floyd–Warshall is an example of Dynamic Programming.

2 Solving a DP

Steps to solve a DP:

1. Identify if it is a DP problem - typically requires maximization or minimization of a problem

2. Decide a state expression with least parameters - A state can be defined as the set of parameters that can uniquely identify a certain position or standing in the given problem. This set of parameters should be as small as possible to reduce state space.

   For example: In our famous Knapsack problem, we define our state by two parameters index and weight i.e DP[index][weight]. Here DP[index][weight] tells us the maximum profit it can make by taking items from range 0 to index having the capacity of sack to be weight. Therefore, here the parameters index and weight together can uniquely identify a subproblem for the knapsack problem.

3. Formulate state relationship - find a relation between previous states to reach the current state.

4. Do tabulation (or add memoization)

Tabulation Bottom Up Dynamic Programming (starting from dp[0] we move up)

Memoization Top Down Dynamic Programming (starting from dp[n] we use recursion here)

3 Some Solved Problems

3.1 Calculating $C(n, r)$

Calculating combination or $C(n, r)$ using dynamic programming is really easy. We can see the optimal substructure property from the figure below, $C(n, r) = C(n-1, r-1) + C(n-1, r)$. Here we can either take an object then we have $C(n-1, r-1)$ ways to choose or we left out that object then we have $C(n, r-1)$ ways to choose
We can also see overlapping sub-problems here in the figure marked red.
The

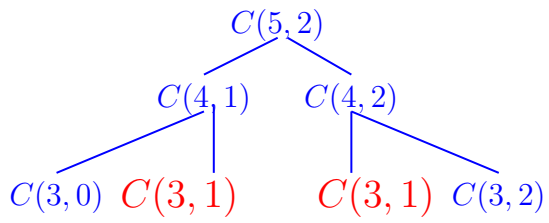

Figure 2: dynamic programming properties of C(n,r)

3.2 $C(n, r)$

```
/***from dust i have come, dust i will be***/

#include<bits/stdc++.h>

typedef long long int ll;
```

```
6
7  #define dbg printf("in\n")
8  #define nl printf("\n");
9  #define N 100
10
11 using namespace std;
12
13 ll dp[N][N];
14
15 ll nCr(int n, int r)
16 {
17     if(r==1) return n;
18     else if(r==n) return 1;
19
20     if(dp[n][r]!=-1) return dp[n][r];
21
22     return dp[n][r]=nCr(n-1,r)+nCr(n-1,r-1);
23 }
24
25 int main()
26 {
27     memset(dp,-1,sizeof(dp));
28     cout<<nCr(5,2); nl
29     cout<<nCr(9,6);
30
31     return 0;
32 }
```

4 Conclusion

Dynamic Programming has many classic problems and other problems need intuition and practice to solve.