DYNAMIC PROGRAMMING

(Part - 1)

This document mainly focuses on an introduction to dynamic programming. It is one of the most important programming concept that requires a lot of practise. There are a lot of variants of dynamic programming namely:

- Single dimensional dp
- 2-dimensional dp
- Dp with bitmasking
- Digit dp
- Dp on graphs (mainly trees)

At the end of the document there are links for various useful articles and problems that you can solve. Remember dp can be quite intimidating at the start, but once you get the grasp of it (with practise of course) you'll have a lot of fun solving dp problems.

In this first document we'll be focussing on one dimensional dp. The basic idea of dynamic programming is to remember (store) the answer for a previous sub-problem and use it further to solve the current problem. As an example let's consider the problem of finding n^{th} fibonacci number. Lets say $fib[i]$ stores the value of the i^{th} fibonacci number. As you already know that $fib[i] = fib[i-1] + fib[i-2]$. Here we can see that calculating $fib[i]$ requires previously solved sub-problems i.e. $fib[i-1]$ and $fib[i-2]$.

At first dynamic programming can seem to be non-intuitive and difficult. But that's okay. For solving any problem of dynamic programming you just have to think of how can I solve this problem if I already know the solution of a previous problem. Once you establish a relation between the current problem and previous sub-problems, the only task you are left with is to code the solution and get a green tick :).

Lets see an example on one dimensional dynamic programming.

Example: Let us try to solve a classical dp problem of maximum subarray sum . You have an array $arr[]$ of length N ($1 <= N <= 10^5$) where $-10^9 <= arr[i] <= 10^9$. You have to find the maximum among all the subarray sums of the given array. Remember that a subarray is a non-empty continuous segment of an array eg. indices $i,\ i+1, i+2, ...i+k$ make a subarray of length $k+1$.

Solution: Lets define an array $dp[]$ where $dp[i]$ denotes the value of maximum subarray sum among all the subarrays that end at index $i.$ Now, read the next paragraph very carefully.

Suppose you have already computed all the $dp[]$ values for all the indices less than $i.$ That is, you already have the values $dp[0],\ dp[1],\ dp[2],\ ...dp[i-1].$ Now, you want to find $dp[i]$. There are two cases possible . Either, the subarray ending at the index i is a one element subarray (the subarray only has the i^{th} element) or the subarray has more than one element (which means that the second last element will be $i-1$). For the second case you can observe that the maximum subarray sum ending at index i is nothing but the value of i^{th} element added to $dp[i-1]$. Hence , the relation established is

$$dp[i] = max(arr[i], arr[i] + dp[i-1])$$

So you see, this is what you have to do to solve dp problems. In the second document we'll be learning two dimensional dp and in the third we'll see dp with bitmasking. Till then try to solve the below given problems and happy coding :).

Useful Articles:

- Hackerearth tutorial

- Maximum subarray sum : geeksforgeeks tutorial
- Longest increasing subsequence: geeksforgeeks tutorial and youtube tutorial
- Rod cutting problem: geeksforgeeks tutorial and youtube tutorial

Problems:

- Problem-1
- Problem-2
- Problem-3
- Problem-4

You can try to solve other basic dp problems on hackerearth and hackerrank.