# Dynamic Programming - Truth I

*I read it in undergrad, thought about it and did not understand why it works !*

*I read it during my grad, thought about it and got some understanding!*

*I think about it often and to this day, I am gaining an insight into it!*

**My insight**

What is dynamic programming?

Its a bottom up approach to solve problems. And now, whats the notion of 'bottom up approach'? In simple terms, you break the problem into small parts. Then you start with those smaller constituents of the problem, and combine them one after the other and finally, after combining all the constituents, your solution is ready.

*Wait! I thought that was divide and conquer.*
*Or err ... that is recursion dude, you ain't know nothing.*

True indeed. This matches with the notion of divide and conquer technique. And divide and conquer can be formulated by recursion. If you have understood this, its time for you to know about another great truth.

***In dynamic programming, the smaller constituents or sub-problems OVERLAP!***

[I am going off your blog. Thinking intuitively , huh and all you could give me was bookish gibberish which I cannot comprehend.]

Wait! This is not gibberish and it is *one of the two* most important truths of DP. Let me explain it through an example. Do you know binary search and how we split the array into half each time during binary search? Remember that innocuous looking algorithm of binary search that is considered to be one of the most beautiful things on this planet existing within a dimension of **log n** time.

I hope you do remember that algorithm. If you don't, please go, read it and then come back. Now for those of you who are waiting for me to go further, here it is:

In binary search, you split the array into two equal halves, say L and R during one search. The number that you are searching for is either in L or in R. It cannot be in both (lets consider array of distinct elements only for now). If you take a pause and think now, you broke down your search problem into smaller parts: search in part L or search in part R. But the search will be successful in one of the parts. What does that imply? Your smaller parts do not overlap. Basically, either you go to part L and search further (never coming back to R again) or you go to R , search and discard part L. There is no notion of an inter-dependence between the two smaller parts for finding the solution(in this case, searching the element).

*Now do you see the notion of sub-problems overlapping! If the sub-problems overlap, we cannot get the final solution unless we solve and combine them in a certain way (bottom-up way).*

Not to confuse you further, I will put out my remaining truth of DP in the next post. Do give it a look.

# Dynamic Programming - Truth II

***A Dynamic programming problem has a optimal substructure.***

*The second most important truth of DP!*

What does this mean? It simply means that you cannot apply DP unless you are sure that the optimal solution of sub-problems will help you in getting the optimal solution of the bigger problem. And how would any sane person know if a problem has optimal sub-structure? Simple, by putting your grey cells to work.

Lets think about it taking a example.

*- Shortest Path*
The problem is to find a shortest path from a vertex u to vertex v in a graph G (lets forget about the nature of the graph for a while ... I am trying to lay out a concept).
Lets say we have i vertices (labelled from 1,2,3 .. i) from u to v in the actual shortest path. So the shortest path looks like this:

u ->1 -> 2 -> 3 -> .... ->i -> v

Now to reach vertex v through a shortest path, you must reach vertex i through the shortest path. Why? Because if you do not do so, and give me a solution that includes a path from u to v through i such that the path from u to i is not the shortest one, then I will simply replace the path from u to i with the shortest path and get a better solution than the one you gave to me!

Whoa! Do you see now the meaning of optimal sub-structure. You cannot solve shortest path from u to v optimally unless you solve shortest path from u to i optimally. And you cannot solve the shortest path from u to i optimally unless you solve the shortest path from u to j (where j is an intermediate vertex in the shortest path from u to i). And you cannot solve shortest path from u to j unless ... the story continues.

At this juncture, I hope you might be a little sensitive to DP and its truths. But a few of you might be stricken by the feeling that this all seems so dreadfully familiar to Divide and Conquer technique. Yes, indeed. This is similar, but yet different. The next part is the grand finale for the DP treatise where I will outline a few more insights into it. Till then, an interesting fact to ponder about:

*Solving single source shortest path to a destination vertex in a graph is as hard as solving single source shortest path to all the vertices. Think about it, have you seen a polynomial time algorithm that calculates the single source single destination shortest path only without calculating all other shortest paths from the source?*
*Voila! This is your chance to fame. :)*

# Dynamic Programming - This, That and the Universe

***Divide and Conquer versus Dynamic Programming***

Let me outline two very important truths here:

- *D&C is a top-down approach while DP is a bottom-up approach to solve problems.*
- *D&C can be used to solve a problem instance of DP.*

What then, is the difference, you may ask? Why DP? Why not D&C? The answer, my friend, is blowing in the wind. :)

I will give it a shot to explain this as simply as I can:

In D&C, you divide the main problem, say P into smaller parts. Lets consider, for the sake of simplicity, that P is broken down into two smaller sub-problems S1 and S2. Simply think of P as being: P = S1+ S2. Now, in a ***pure*** D&C setting, you would solve S1 and S2 separately, combine their solutions in a certain way and get the answer to P. Perfect!

In DP, the first realization you should have is that S1 and S2 overlap. Lets consider that S2 depends on S1 (overlap!). This means that S1 has to be solved before S2 can be solved. Now you may ask, why? Why isn't there any other way to solve S2? Because if P is a DP problem and P has a optimal sub-structure, then its sub-problems also have a optimal sub-structure. And if S1 and S2 have a optimal sub-structure and S2 is dependent on S1, you should optimally solve S1 first which would lead you to getting a optimal solution to S2.

And how does this all say that you cannot use D&C? As a matter of fact, you can. You could solve S1 in one part, S2 in another ... in a D&C setting. But that would involve more work because you are going to solve S1 twice, once by itself and once when you are trying to solve S2. That loosely-said***more-work*** grows exponentially as the overlapping sub-problems increase.

All that said and done, whats a simple, intuitive way to solve such a problem P which includes the notion of overlapping sub-problems and optimal sub-structure? Its simple, isn't it? Solve S1 first and remember the result. Then solve S2 utilizing the remembered result for S1 (S2 depends on S1). Then solve P using the solutions of S2 and S1. All you had to do was start from bottom (S1) and move gradually (through S2) towards up (P). That makes your computation easier, gives you a solution and a chance to boast about your DP prowess.

So long, DP!