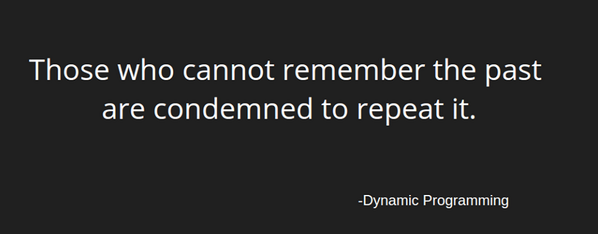
Dynamic Programming



Let us say that we have a machine, and to determine its state at time **t**, we have certain quantities called state **variables.** There will be certain times when we have to make a decision which affects the state of the system, which may or may not be known to us in advance. These decisions or changes are equivalent to transformations of state variables. The results of the previous decisions help us in choosing the future ones.

We need to break up a proble into a series of overlapping sub-problems, and build up solutions to

larger and larger sub-problems.

Some famous Dynamic Programming algorithms are:

* [Unix diff](https://en.wikipedia.org/wiki/Diff_utility) for comparing two files
* [Bellman-Ford](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm) for shortest path routing in networks
* [TeX](http://en.wikipedia.org/wiki/TeX) the ancestor of LaTeX
* [WASP](https://en.wikipedia.org/wiki/WASP_%28cricket_calculation_tool%29) - Winning and Score Predictor

Majority of the Dynamic Programming problems can be categorized into two types:

1. **Optimization problems:** The optimization problems expect you to select a feasible solution, so that the value of the required function is minimized or maximized
2. **Combinatorial problems:** Combinatorial problems expect you to figure out the number of ways to do something, or the probability of some event happening.

Every Dynamic Programming problem has a schema to be followed:

1. Show that the problem can be broken down into optimal sub-problems.
2. Apply Intuition at each step
3. Recursively define the value of the solution by expressing it in terms of optimal solutions for smaller sub-problems.
4. Compute the value of the optimal solution in bottom-up fashion.
5. Construct an optimal solution from the computed information

Let us say that you are given a number **N,** you've to find the number of different ways to write it as

the sum of 1, 3 and 4

[A] if we have only 1

[B] intuition: As we have only 1 DP[N-1] == DP[N]

[C] DP[N] = DP[N-1]

[A] if we have only 4

[B] intuition: As we have only 4 DP[N-4] == DP[N]

[C] DP[N] = DP[N-4]

[A] if we have only 3

[B] intuition: As we have only 3 DP[N-3] == DP[N]

[C] DP[N] = DP[N-3]

[D] DP[N] = DP[N-3] or DP[N-1] or DP[N-4]

[D] DP[N-3] = DP[N-6] + DP[N-4] + DP[N-7]

[D] Base case:

[B] Intuition:

[D] Get the base case where N = 4

DP[0] = 0

DP[1] = 1 , DP[2] = 1, DP[3] = 2

Sol:

[E]

DP[0] = DP[1] = DP[2] = 1; DP[3] = 2;

for (i = 4; i <= n; i++) {

DP[i] = DP[i-1] + DP[i-3] + DP[i-4];

}

Problem:

"Imagine you have a collection of **N** wines placed next to each other on a shelf. For simplicity, let's number the wines from left to right as they are standing on the shelf with integers from **1 to N**, respectively. The price of the ith wine is pi. (prices of different wines can be different).

Because the wines get better every year, supposing today is the year 1, on year y the price of the ith wine will be y\*pi, i.e. y-times the value that current year.

You want to sell all the wines you have, but you want to sell exactly one wine per year, starting on this year. One more constraint - on each year you are allowed to sell only either the leftmost or the rightmost wine on the shelf and you are not allowed to reorder the wines on the shelf (i.e. they must stay in the same order as they are in the beginning).

You want to find out, what is the maximum profit you can get, if you sell the wines in optimal order?"

Sol:

=================== Either left most or right most, Once a year

P1 p2 p3 p4 =

1 4 2 3 =

===================

B : Lets take wine from left most 1\*1 = 1\*2

B : Right Most = 3\*1 , 3\*2

Max(1\*1,1\*2 ,3\*1,3\*2)

Combination is max (1\*1 + 3\*2) or (1\*2 + 3\*1) => sell p1 year = year+1

3\*2,4\*3 or 4\*2,3\*3 1,3,4,2 1\*1 +