Dynamic Programming

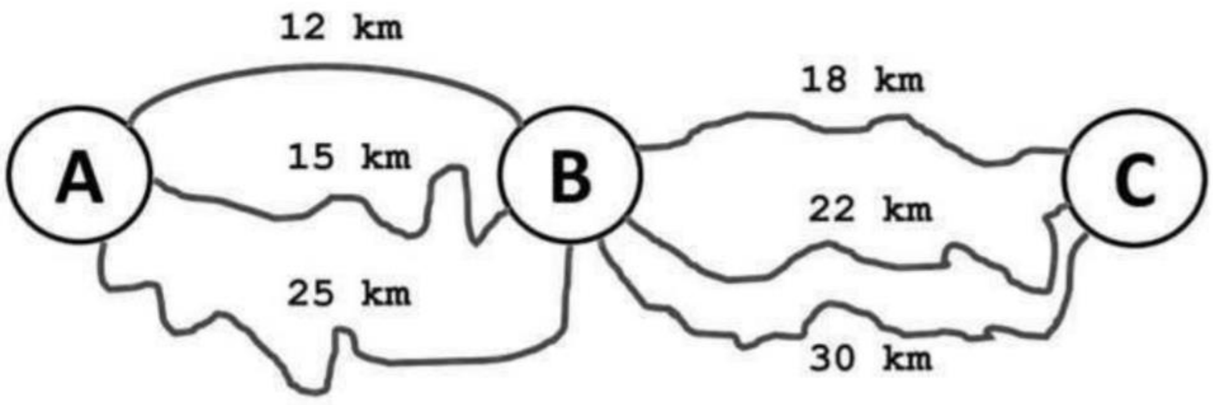
Dynamic programming refers to a problem-solving approach, in which we compute and store the result of simpler subproblems, in order to build up the solution to a complex problem.

It can be implemented by **memoization** or by **tabulation**.

There is nothing dynamic about Dynamic Programming 😲

https://cstheory.stackexchange.com/questions/5635/if-you-could-rename-dynamic-programming

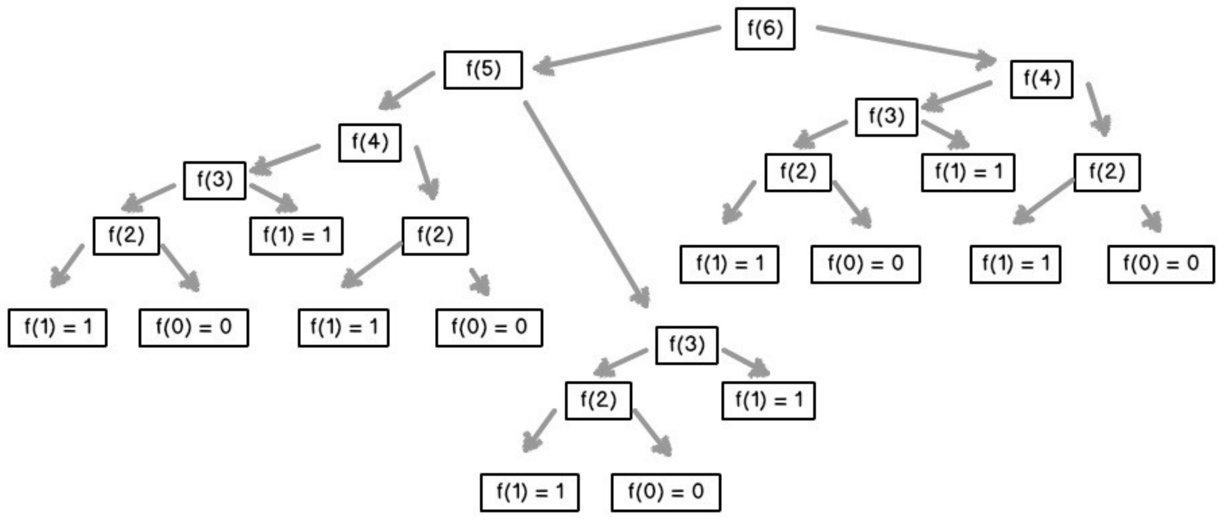
Optimal Substructure



Overlapping Subproblems

Fibonacci Series : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

```
public static int nthFibonacci(int n) {
            if(n == 0 || n == 1) {
                  return n;
            } else {
                  return nthFibonacci(n-1)+ nthFibonacci(n-2);
            }
}
```

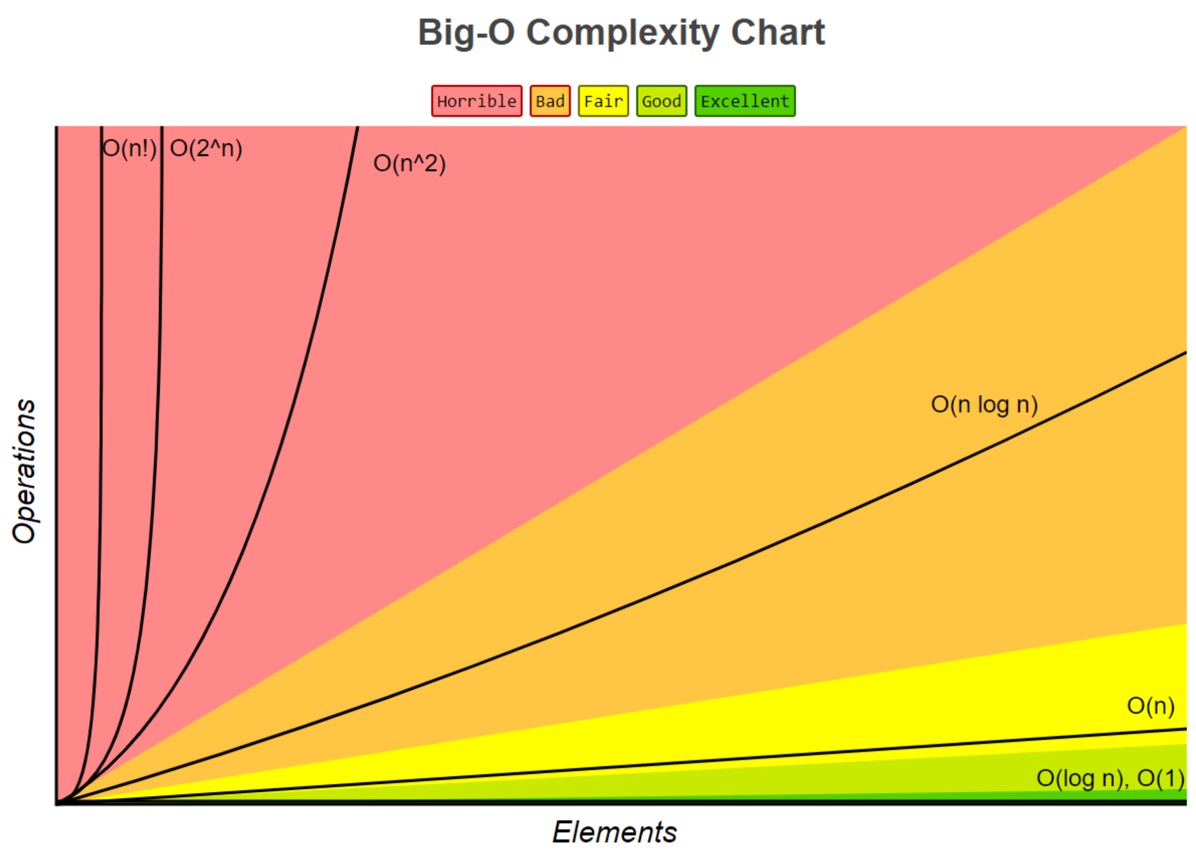


T(n) = T(n-1) + T(n-2) +1

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(n-1) + 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned}
T(n) &= 2\underbrace{\times T(n-1)}_{} + 1 \\
&= 2[2\times T(n-2) + 1] + 1 \\
&= 2^2 \underbrace{T(n-2) +}_{} 2 + 1 \\
&= 2^2 [2\times T(n-3) + 1] + 2^1 + 2^0 \\
&= 2^3\, T(n-3) + 2^2 + 2^1 + 2^0 \\
&\ldots \\
&= 2^k\, T(n-k) + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2^1 + 2^0 \\
&\ldots \\
&= 2^{n-1}\, T(1) + 2^{n-2} + 2^{n-3} + \ldots + 2^2 + 2^1 + 2^0 \\
&= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^2 + 2^1 + 2^0 \\
&= 2^n - 1 \qquad \boxed{\text{i.e., } T(n) \text{ is } O(2^n)}
\end{aligned}$$

The naive recursive implementation have runtime complexity of O(2^n)



Techniques of Dynamic Programming :

Memoization

Memoization refers to the technique of caching and reusing previously computed results.

Tabulation

Tabulation is similar to Memoization but in tabulation we fill the cache iteratively.

Tabulation :

No recursion, we fill the cache iteratively using loops.

- **Memoization : Top Down approach**
- **Tabulation : Bottom Up approach**

Note : DP version have runtime complexity of O(n).

Should I use tabulation or memoization ?

If the original problem requires all subproblems to be solved, tabulation usually outperforms memoization by a constant factor. This is because tabulation has no overhead for recursion.

If only some of the subproblems needs to be solved for the original problem to be solved, then memoization is preferable since the subproblems are solved lazily, i.e. precisely only the computations which are needed are carried out.

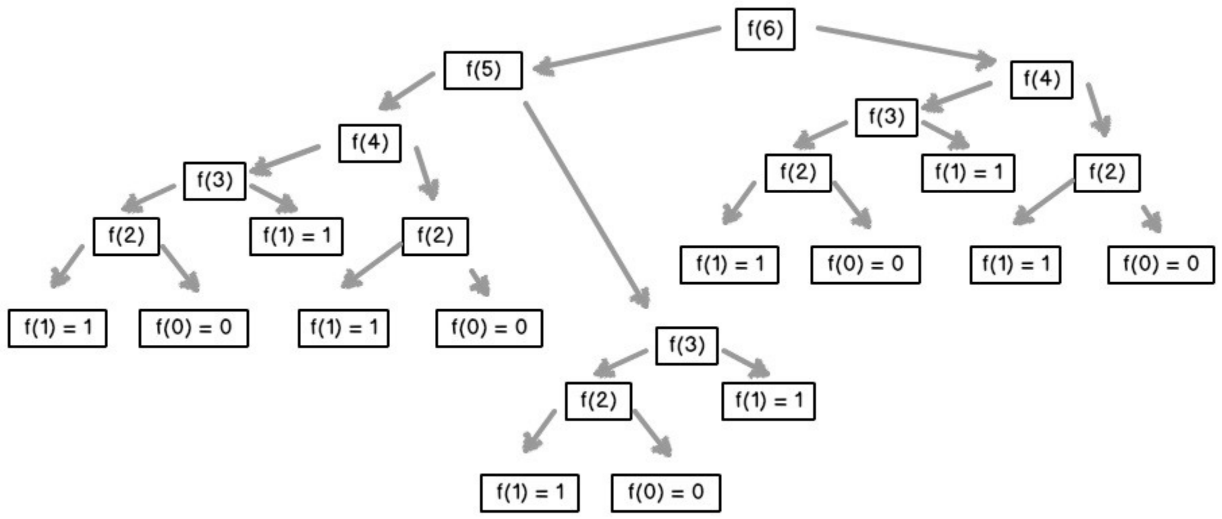
Note : In Dynamic Programming we mostly use Tabulation (iterative method)

Dynamic Programming and Divide and Conquer

Dynamic programming is similar to the divide-and-conquer approach, in that the solution of a large problem depends on previously obtained solutions to easier subproblems.

But the significant difference, is that in dynamic programming subproblems overlap.

By overlap, we mean that the result of a subproblem can be used in the solution of two other different subproblems. e.g. result of f(4), can be used in both while calculating f(5) and f(6)



But in, divide-and-conquer approach each subproblem is different (subproblems do not overlap). Since every time we will be solving a different subproblem, caching the value of a subproblem doesn't make sense.