<https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html>

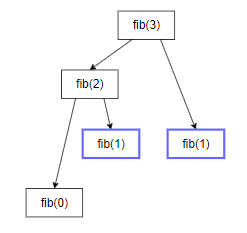
Dynamic programming vs memoization vs tabulation

Dynamic programming is a technique for solving problems recursively. It can be implemented by **memoization** or **tabulation**.

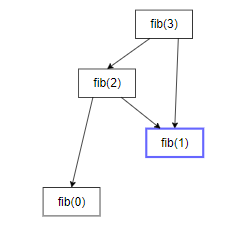
Dynamic programming

Dynamic programming, DP for short, can be used when the computations of subproblems overlap.

If you're computing for instance fib(3) (the third Fibonacci number), a naive implementation would compute fib(1) twice:



With a more clever DP implementation, the tree could be collapsed into a graph:



It doesn't look very impressive in this example, but imagine the difference for fib(1000). In fact the naive implementation makes O(2*n*) calls, while the DP version makes O(*n*) calls.

Memoization

Memoization refers to the technique of caching and reusing previously computed results. If you for instance have

f(x) {

return x + x

}

the memoized version would look like

f(x) {

if (mem[x] is undefined)

mem[x] = x + x

return mem[x]

}

The memoized fib function would thus look like this:

memFib(n) {

if (mem[n] is undefined)

if (n < 2) result = n

else result = memFib(n-2) + memFib(n-1)

mem[n] = result

return mem[n]

}

is enough to cause the tree to collapse into a graph as shown in the figures above. The calls would look something like

memFib(3)

memFib(2)

memFib(1) // cache the result

memFib(0)

memFib(1) // cached, will return mem[1]

This approach is **top-down** since the original problem, fib(3), is at the top in the above computation.

Tabulation

Tabulation is similar but focuses on filling the entries of the cache. Computing the values in the cache is easiest done iteratively. The tabulation version of fib would look like this:

tabFib(n) {

mem[0] = 0

mem[1] = 1

for i = 2...n

mem[i] = mem[i-2] + mem[i-1]

return mem[n]

}

Here the computation can be described as follows:

mem[0] = 0

mem[1] = 1

mem[2] = mem[0] + mem[1]

mem[3] = mem[1] + mem[2]

As opposed to the memoization technique, this computation is **bottom-up** since original problem, fib(3), is at the bottom of the computation.

Should I use tabulation or memoization?

If the original problem requires all subproblems to be solved, tabulation usually outperformes memoization by a constant factor. This is because tabulation has no overhead for recursion and can use a preallocated array rather than a hash map.

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