Dynamic Programming

David Jacobo
jguillen@cimat.mx

July 24, 2015

Dynamic Programming (DP):

Also called **'memoization'** is a paradigm focused on solving **optimization problems**. For a problem to be solved with DP, it must exhibit an **optimal substructure and overlapping sub-problems**.

A critical part on designing DP solutions boils down to recognizing/defining the required **state and transitions**. There are two typical ways to express a DP solution: **bottom up and top down**.

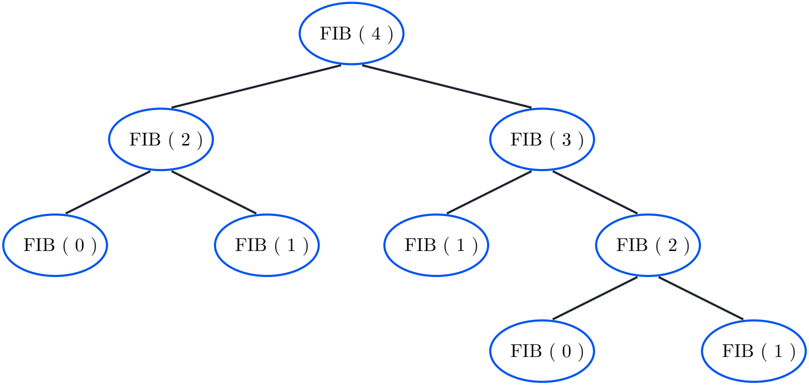
Outline

Fibonacci series: Recurrence formula

$$FIB(N) = \begin{cases} 1, & \text{when } N \leq 1 \\ FIB(N-2) + FIB(N-1), & \text{otherwise.} \end{cases} \tag{1}$$

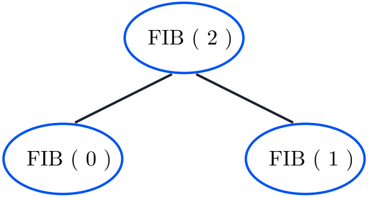
For some instances we actually know the solution, those are the **base cases**.

Fibonacci series: Recurrence tree



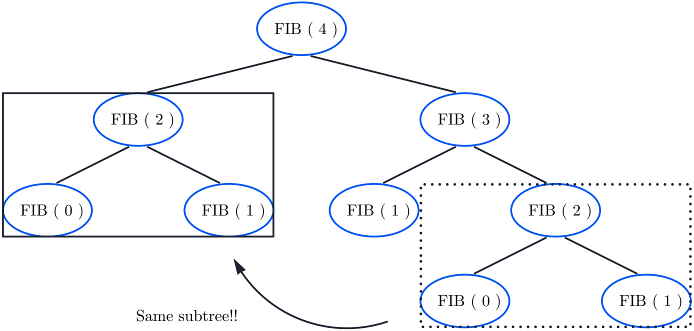
Optimal substructure

Refers to the fact that you need to know the **optimal solution for smaller instances** in order to expand your solution to the required size.



Overlapping sub-problems

Whenever you are calling the exact instance of the problem 2 or more times it's clear than an overlap exists.



State

A state is the list of parameters which represent each sub-problem in a unique way.

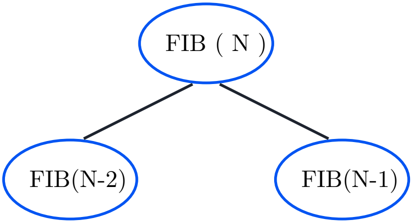
For our example, an integer suffices to represent each state:

indexes =	0	1	2	3	4
values =	1	1	2	3	5

Transition

A transition defines if you can directly pass from an state A, to an state B.

In this example a line represent the different transitions available, all of them relates N to N-1 and N-2, just like in the formula.



Outline

Bottom up

This variant solves and store the solutions for **all the smaller instances** before solving the bigger instance (which is the one that matters for us).

```
int state[A_LOT];
state[0] = state[1] = 1;

int fib(int N){
    for(int i=2;i<= N;++i)
        state[i] = state[i-1] + state[i-2];

    return state[N];
}
```

Top down

The top down approach relies in calling only the instances which are really needed for the given problem.

```
int state[A_LOT];
memset(state, NOT_CALC, sizeof(state));
state[0] = state[1] = 1;

int fib(int N){
    if(state[N]!=NOT_CALC)     return state[N];

    return state[N] = (fib(N-1) + fib(N-2));
}
```

So, which one is better?

Remember that both are based on the same recurrence formula, thus they are equaly correct.

Outline

Time complexity

Time complexity refers to the amount of time it will take to your algorithm to finish in terms of a given input.

O(M * S)

where: M stands for the total different states and S stands for the complexity of calculating each state.

Space complexity

The space complexity refers to the amount of memory necessary to store all the answers, so it basically express how many states your problem has.

$$\mathbf{O(M)}$$

where M stands for the total different states.

Outline

Again, the core of DP is finding the **optimal solution** for a problem, ranging from strings processing to robotics!

- Longest Common Subsequence
- Edit Distance
- Single Source Shortest Path
- Coin Change
- Knapsack
- Floyd-Warshall

Outline

Pro-tips

- Study the theory.
- Write solutions for the classic problems.
- Compare bottom-up vs top-down.

It is also worthy to explore the **sliding window** trick in order to save memory, or selecting a **different data structure** for saving the intermediate states specially when using the top-down approach. **DP is a trade-off between space and time!**

Q & A

References

- Competitive Programming site
- Algorists' repository