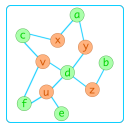
Dynamic Programming

Ulises Tirado Zatarain [1]
(ulises.tirado@cimat.mx)

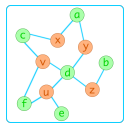
[1] Algorists Group

November, 2020

Outline



Outline

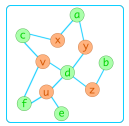


What is dynamic programming?

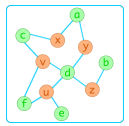
Definition (Dynamic Programming)

It's a technique for mathematical programming (optimization), or in other words, a paradigm to problem solving. So, the word "programming" is not directly meaning a computer program or an algorithm. The actual meaning is more similar to linear programming or planning or taking decisions.

- It consists in:
  - Split the problem in smaller sub-problems:
    - recurrence relationship
    - optimal sub-structure
  - We stop when have a trivial problems (base cases)
  - Store smaller solutions (memoization, states)
  - Merge the solutions when is need it
  - Difference with D&C: **overlapping and compute a value**

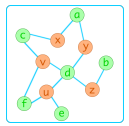


Outline



Recurrence relationship and tree of calls

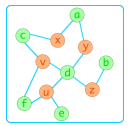
Fibonacci sequence $(0,1,1,2,3,5,\ldots)$ is the simplest examples



Recurrence relationship and tree of calls

Fibonacci sequence $(0,1,1,2,3,5,\ldots)$ is the simplest examples

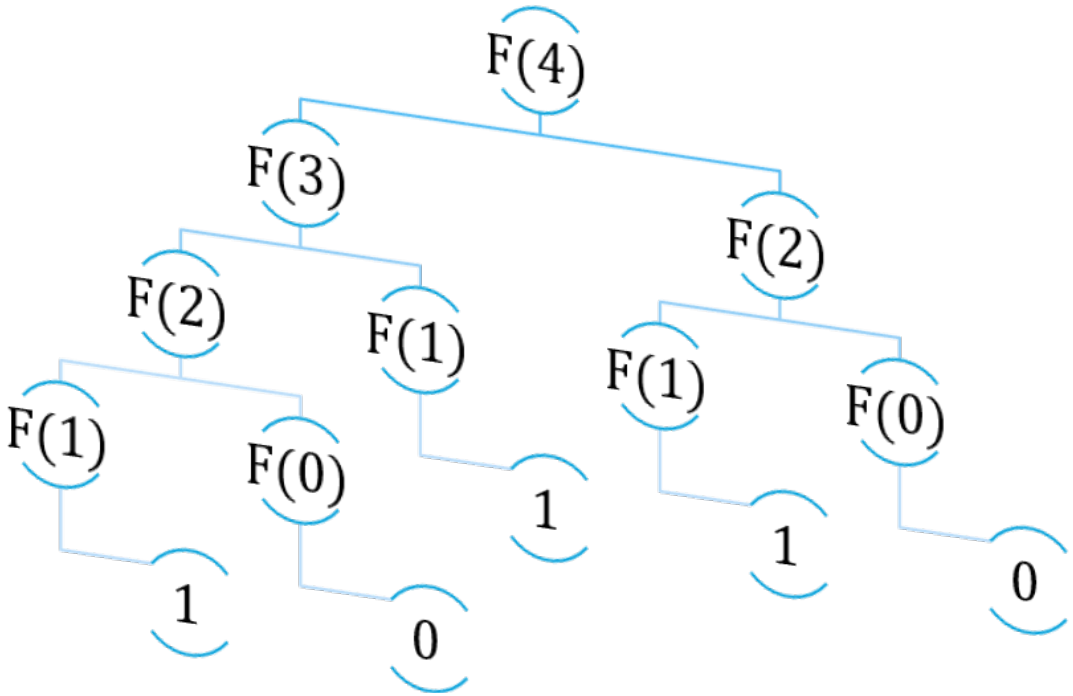
$$F(n) = \begin{cases} 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \\ 0 & \text{otherwise} \end{cases}$$

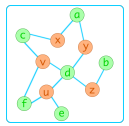


Recurrence relationship and tree of calls

Fibonacci sequence $(0,1,1,2,3,5,\ldots)$ is the simplest examples

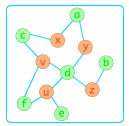
$$F(n) = \begin{cases} 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \\ 0 & \text{otherwise} \end{cases}$$





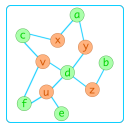
Fibonacci classic iterative implementation

```
const int MAX = 90;
function long int fibonacci(int n):
    static long int memo[MAX+1];
    if n < 0 or n > MAX then throw "Out of range.";
    memo[0] = 0; memo[1] = 1;
    if memo[n] == UNDEFINED then
        for k = 2 to n do:
            memo[k] = memo[k-1] + memo[k-2];
        end
    end
    return memo[n];
end
```



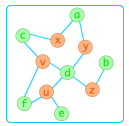
Fibonacci classic recursive implementation

```
const int MAX = 90;
function long int fibonacci(int n):
    if n < 0 or n > MAX then throw Out of range.;
    if memo[n] ≠ UNDEFINED then return memo[n];
    return memo[n] = fibonacci(n − 1) + fibonacci(n − 2);
end
```

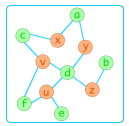


Problems with the classic implementations

We can see that $F(2)$ is computed twice (this is the **overlapping**).



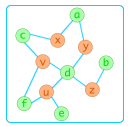
Outline



Design and implementations

There are two ways to implement dynamic programming:

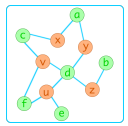
- Bottom-up (iterative implementation)
  - Solve **all the the possible smaller problems** before the bigger one
- Top-down (recursive implementation)
  - Solve **only the instances which are actually needed** for a given problem
- Both solve the problem in efficient way, the discussion could be end up in religious arguments.



Fibonacci bottom-up implementation

Solve all the the possible smaller problems before the bigger one:

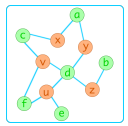
```
const int MAX = 90;
const long int UNDEFINED = -1;
function long int fibonacci(int n):
    static long int memo[MAX+1];
    if n < 0 or n > MAX then throw "Out of range.";
    memo[0] = 0; memo[1] = 1;
    if memo[n] == UNDEFINED then
        for k = 2 to n do:
            memo[k] = memo[k-1] + memo[k-2];
        end
    end
    return memo[n];
end
```



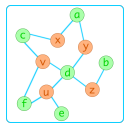
Fibonacci top-down implementation

Solve only the instances which are actually needed for a given problem:

```
const int MAX = 90;
const long int UNDEFINED = -1;
function long int fibonacci(int n):
    static long int memo[MAX+1];
    if n < 0 or n > MAX then throw "Out of range.";
    memo[0] = 0; memo[1] = 1;
    if memo[n] ≠ UNDEFINED then return memo[n];
    return memo[n] =fibonacci(n-1)+fibonacci(n-2);
end
```



Analysis



References I

- Introduction to Algorithms, Thomas H. Cormen
- Algorists: Github Repository
- Wikipedia: Dynamic Programming
- HackerRank
- CodeForces
- OmegaUp
- LeetCode