Brief introduction to Dynamic Programming

Recursion and memoization

Ayoub Ouarrak

14/02/2019

Recursion

Definition

Recursion occur when a thing is defined in terms of itself, in Mathematics and Computer Science a function exhibit recursive behaviour when it can be defined by the following properties:

- A **base case**.
- An **Induction step**, which execution will lead eventually to the base case

Naive Fibonacci

Example:

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & \forall n \geq 2 \end{cases} \tag{Fibonacci}$$

The same identity can be expressed algorithmically as follows:

```
F(n):
  if n = 0
    return 0
  else if n = 1
    return 1
  else
    return F(n-1) + F(n-2)
```

Recursion Tree

A recursion tree, as the name suggest, is tree where each node is a function call; is used to calculate recurrence relations and visualizing the execution of a recursive algorithm.

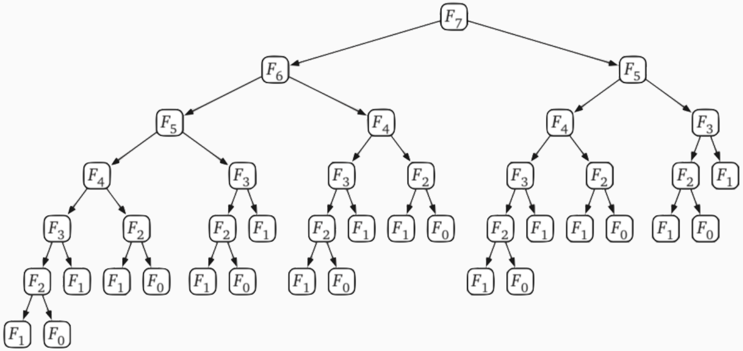

Figure 1: Recrusive tree for the execution of $F(7)$

Time complexity Fibonacci

The naive recursive algorithm for Fibonacci is awfully slow, with a run time complexity:

$$F_n = \theta(\phi^n)$$

The obvious reason for the recursive algorithm's lack of speed is that it computes the same Fibonacci numbers over and over and over. This is visible from the Recursive Tree.

Bonus: Tail Recursion

In tail recursion, the computation is done at the beginning and then the recursive call is executed (last statement).

```
F(n):
    // some code here
    return F(parameters...)
```

With tail recursion, the compiler is able to apply some optimization, only a stack frame will be needed with the advantage of eliminating a possible *StackOverflow*.

Note: Not all compilers are optimized for tail recursion, e.g. Java doesn't support tail recursion.

Dynamic Programming

Figure 2: Anakin happy to learn more about dynamic programming

History and Etymology

Dynamic Programming is a paradigm pioneered and formalized by **R. Bellman** during 1950s at the RAND corp. Regarding the etymology, Bellmann later claimed that he deliberately chose the name “dynamic programming” to hide the mathematical character of his work from his military bosses, who were actively hostile toward anything resembling mathematical research.
The word **programming** does not refer to writing code, but rather to the older sense of planning or scheduling, typically by filling in a table.
The word **dynamic** was not only a reference to the multistage, time-varying processes that Bellman and his colleagues were attempting to optimize, but also a marketing buzzword.

Applications

Dynamic Programming is not limited to Computer Science but have multiple applications

- Bioinformatics
- Economy
- Information theory
- Robotics
- ....

Memoization

As we saw for the naive algorithm of Fibonacci, the reason of the poor performance are duo to the fact that some intermediate values are calculated more than once.
We can speed up the algorithm considerably just by writing down the results of our recursive calls and looking them up again if we need them later.
This optimization technique, known as **memoization**, is a dynamic programming algorithm usually credited to Donald Michie in 1967.

Less naive Fibonacci

```
F(n):
  if n = 0
    return 0
  else if n = 1
    return 1
  else
    if Farr[n] is undefined then
      Farr[n] <- F(n-1) + F(n-2)

    return Farr[n]
```

By the introduction of memoization we reduced the time complexity from exponential to linear in n:

$$F_n = O(n)$$

Less naive Fibonacci recursive tree

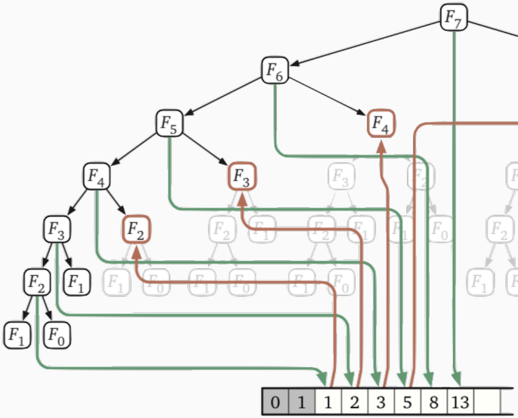

Figure 3: The recursion tree for F(7) using memoization. Downward green arrows indicate writing into the memoization array; upward red arrows indicate reading from the memoization array.

Smart Recursion

In a nutsheel, dynamic programming is *recursion without repetition*, by storing the solutions of intermediate *subproblems*, often but not always in some kind of array or table.
More important, dynamic programming is not about filling tables, it's about **Smart Recursion**.
This is translated by finding correct recurrence.

Composition of dynamic programming

When dynamic programming can be applied

To apply dynamic programming, we must make sure that the algorithm to optimize have the following properties:

- **Optimal substructure**.
- **Overlapping subproblems**.

Optimal substructure

A problem is said to have **optimal substructure** if an optimal solution can be constructed from optimal solutions of its subproblems.

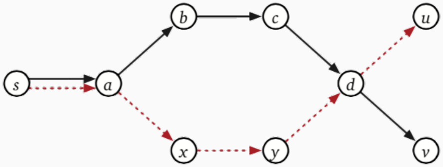

Figure 4: The problem of finding the shortest path from s to u has an Optimal substructure, the red arrows rappresent the optimal subproblems

Overlapping subproblems

A problem is said to have **overlapping subproblems** if the main problem can be broken down into subproblems which are reused several times or a recursive algorithm for the problem solves the same subproblem over and over.

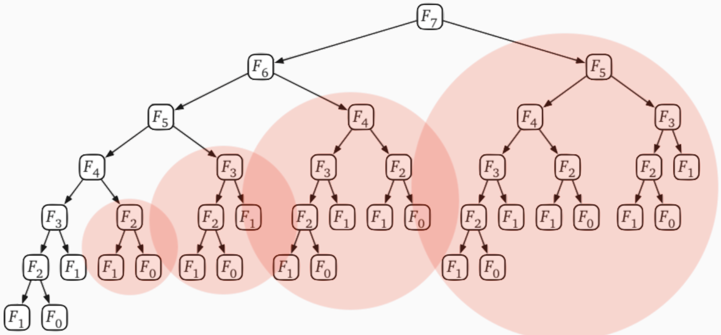

Figure 5: Overlapping subproblems for Fibonacci algorithm

Conclusion

Summary

To summarize,

- *Recursive trees* can help analyse recursive algorithms.
- *Dynamic programming* is a paradigm similar to *divide-et-conquer* that make use of *optimal substructure* and *overlapping subproblems* properties to reduce time complexity.
- *Memoization* is a dynamic programming tecnique that cache the intermediate result of subproblems to avoid the calculation of already known results.

More tecnical improvements that can be considered:

- Avoid db/external ws calls inside a recursive function/loop.
- Use *recursive tail calls* when possible, this will be taken in account by the compiler to generate an optimezed code.

Questions?

References

- **Algorithms**, by *Jeff Erickson*.
- **Introduction to Algorithms**, by *Charles E. Leiserson, Clifford Stein, Ronald Rivest, and Thomas H. Cormen*.