Recursion and Dynamic Programming

People often joke that in order to understand recursion, you must first understand recursion

Giulia Carocari, Giacomo Fabris, Francesco Lotito

Università degli Studi di Trento

ICPC Training @ UniTN
Day 5 - April 10, 2019

Table of Contents

Table of Contents

A - Almost Union Find

Technique

Well...guess what.
Problem applying straightforward UFDS: the "move" operation.

Suggestions

- We can easily move elements from one DS to another if they are leaves
- Things get messy if we need to move an internal node
- Note: if an element of the set is a leaf, it will continue to be a leaf no matter how many merge operations we invoke.

A - Almost Union Find

Solution

Initialization: Make the set $S[i]$ be a child of $S[n+i]$

Move (p, q): Find roots: `x = find(p); y = find(q)`
Then, set `root[p] = y`.
Note that p, q will always be leaves.

Union: No change needed.

Note

Declare a FT of 64 bit integers (perhaps 32 bit unsigned could have been enough).

B - 10 kinds of people

Technique

Seems like flood fill / find connected components. A BFS/DFS will probably result in TLE.
Use Union Find.

Solution

- Each cell of the matrix belongs to one and only one set.
- Merge connected sets.
- Find roots of queries' elements. If the root is the same, solution may be `decimal` or `binary`. Otherwise is `neither`

C - Guess the Data Structure

Technique

Simulation

Solution

- Create a stack, a queue, a pqueue.
- Insertion: operate insertions on all containers.
- Extraction: extract from all containers, compare result with input.

D - Fenwick Tree

Technique

Fenwick Tree - Range Sum, Point Update

Solution

- See last week's slides
- Use 64 bit integers
- Heavy I/O, use `stdio.h`

Table of Contents

Introduction

Notes

- No Kattis contest today (yeah, I know you love me)
- Small amount of theory today (You're loving me even more)
- We will solve some random problems together
- I'm planning to publish some notes on Dynamic Programming on `https://www.fralotito.me`

Introduction

Notes

- No Kattis contest today (yeah, I know you love me)
- Small amount of theory today (You're loving me even more)
- We will solve some random problems together
- I'm planning to publish some notes on Dynamic Programming on `https://www.fralotito.me`

You need to have an account on

- AtCoder `https://atcoder.jp/`
- Kattis `https://open.kattis.com/`

Table of Contents

Divide et whaaat?

Let's Google it

"A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem."

Some examples

- Binary search
- Euclidean Algorithm (GCD)
- Merge sort / Quick sort
- Fast Fourier Transform

Some examples

- Binary search
- Euclidean Algorithm (GCD)
- Merge sort / Quick sort
- Fast Fourier Transform

Note: You can click on them and get a tutorial

Table of Contents

Fibonacci

Problem definition

The Fibonacci numbers are the numbers in the following integer sequence:
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ..$
In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation:

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

Print the n-th Fibonacci Number.

Fibonacci

Problem definition

The Fibonacci numbers are the numbers in the following integer sequence: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ..$
In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation:

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

Print the n-th Fibonacci Number.

Solution

Well.. there are lots of them :)

Fibonacci

Solution

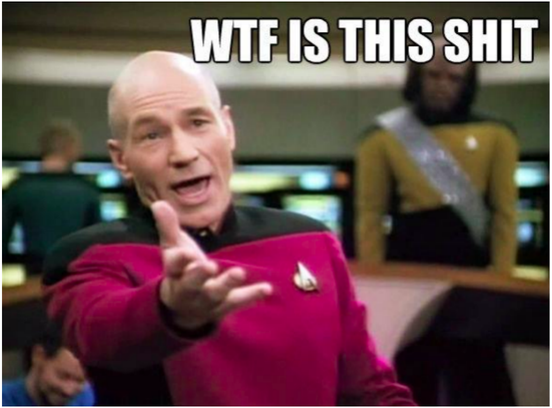
- Use recursion: exponential

Fibonacci

Solution

- Use recursion: exponential

After one day of computation..



Fibonacci

Let's make Fibonacci great again

- Add memoization (DP): $O(n) time, space$

Fibonacci

Let's make Fibonacci great again

- Add memoization (DP): $O(n) time, space$
- Bottom-up (DP): $O(n) time, space$

Fibonacci

Let's make Fibonacci great again

- Add memoization (DP): $O(n) time, space$
- Bottom-up (DP): $O(n) time, space$
- Bottom-up + space optimization (DP): $O(n) time, O(1) space$

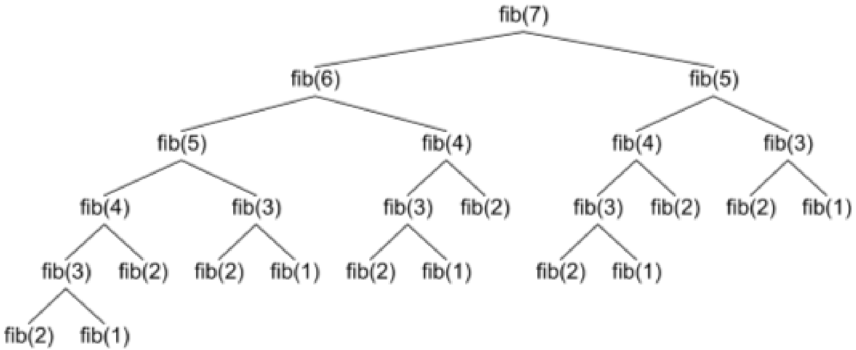
Fibonacci

Let's make Fibonacci great again

- Add memoization (DP): $O(n) time, space$
- Bottom-up (DP): $O(n) time, space$
- Bottom-up + space optimization (DP): $O(n) time, O(1) space$
- Closed formula: $O(1) time, space$

$$f(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right]$$

Why does DP work?



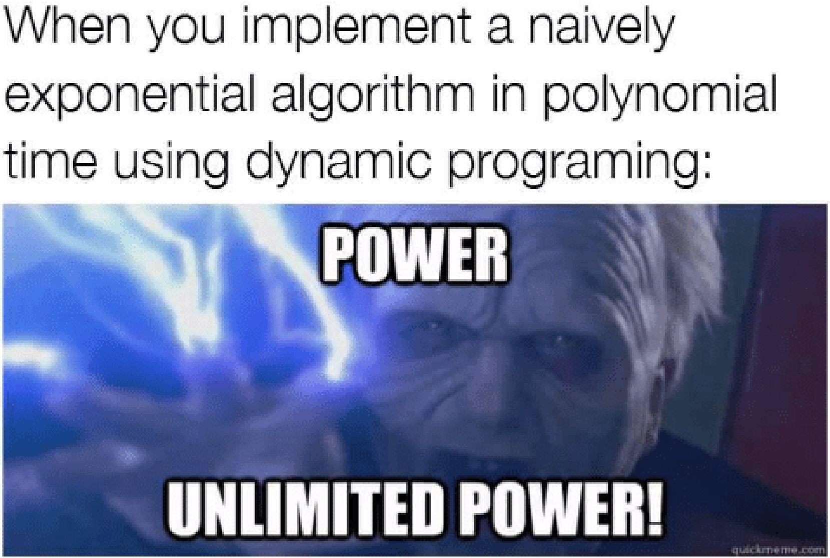

When you implement a naively exponential algorithm in polynomial time using dynamic programing:
POWER
UNLIMITED POWER!
quickmeme.com

Table of Contents

Dynamic Programming

Idea

The idea is very simple, if you have solved a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again.. shortly 'Remember your Past'

Dynamic Programming

Top-down vs Bottom up

- If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor.
    - No overhead for recursion and less overhead for maintaining table.
    - There are some problems for which the regular pattern of table accesses in the dynamic-programming algorithm can be exploited to reduce the time or space requirements even further.
- If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required.

Educational DP contest: problems A to G, **NO E**