Optimization Problems

In the algorithms we have studied so far, correctness tended to be easier than efficiency. In optimization problems, we are interested in finding a *thing* which maximizes or minimizes some function.

In designing algorithms for optimization problem - we must prove that the algorithm in fact gives the best possible solution.

Greedy algorithms, which makes the best local decision at each step, occasionally produce a global optimum - but you need a proof!

Dynamic Programming

Dynamic Programming is a technique for computing recurrence relations efficiently by sorting partial results.

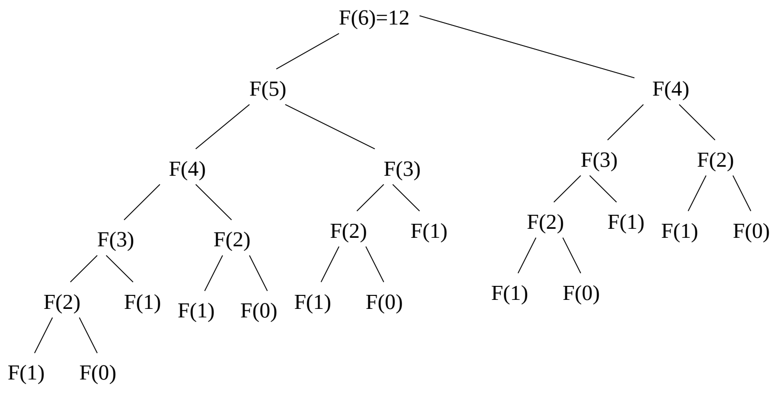
Computing Fibonacci Numbers

$$F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0, F_1 = 1$$

Implementing it as a recursive procedure is easy but slow!

We keep calculating the same value over and over!



How slow is slow?

$F_{n+1}/F_n \approx \phi = (1+\sqrt{5})/2 \approx 1.61803$

Thus $F_n \approx 1.6^n$, and since our recursion tree has 0 and 1 as leaves, means we have $\approx 1.6^n$ calls!

What about Dynamic Programming?

We can calculate F_n in linear time by storing small values:

```
$F_0 = 0$
$F_1 = 1$
For $i = 1$ to $n$
        $F_i = F_{i-1} + F_{i-2}$
```

Moral: we traded space for time.

Dynamic programming is a technique for efficiently computing recurrences by storing partial results.

Once you understand dynamic programming, it is usually easier to reinvent certain algorithms than try to look them up!

Dynamic programming is best understood by looking at a bunch of different examples.

I have found dynamic programming to be one of the most useful algorithmic techniques in practice:

- Morphing in Computer Graphics
- Data Compression for High Density Bar Codes
- Utilizing Gramatical Constraints for Telephone Keypads