EECS 336: Lecture 1: Introduction to Algorithms

Algorithms for Fibonacci Numbers: memoization, repeated-squaring

Reading: Chapter 2 & 3.

Announcements:

- notes on Canvas
- discussion of syllabus on Thursday.

Algorithms

- algorithms are everywhere, examples:
  - digital computers,
  - parliamentary procedure,
  - scientific method,
  - biological processes.
- algorithm design and analysis governs everything.
- good algorithms are closest things to magic.

  cf. Arthur Benjamin does mathemagic
- course philosophy: no particular algorithm is important.
- course goals: how to design, analyze, and think about algorithms.
- we will not cover anything you could figure out on your own.

Algorithms for Fibonacci Numbers

"0, 1, 1, 2, 3, 5, 8, 13, 21, ..."

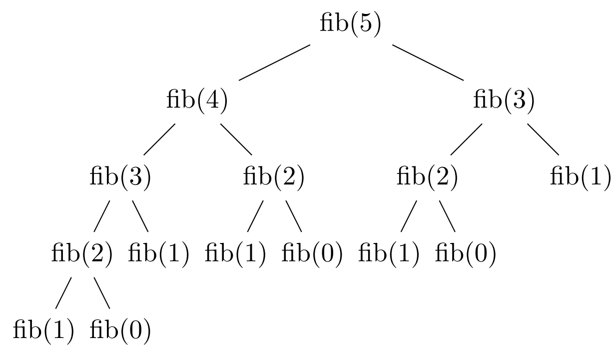
Question: recursive alg?

Algorithm: Recursive Fibonacci

fib(k):

1. if $k \leq 1$ return k
2. (else) return fib$(k-1)$ + fib$(k-2)$

Example:



Analysis

"what is runtime?"

$$
\begin{aligned}
\text{Let } T(k) &= \text{number of calls to fib} \\
T(0) &= T(1) = 1 \\
T(k) &= T(k-1) + T(k-2) \\
&\geq 2T(k-2) \\
&\geq 2 \times 2T(k-4) \\
&\geq \underbrace{2 \times 2 \times \ldots \times 2}_{\text{(k/2 times)}} \times 1 \\
&= 2^{\frac{k}{2}}
\end{aligned}
$$

Conclusion: at least "exponential time"!

Remembering Redundant Computation (memoization)

Idea: remember redundant computation (memoize)

Algorithm: Memoized Recursive Fibonacci

fib-helper(k):

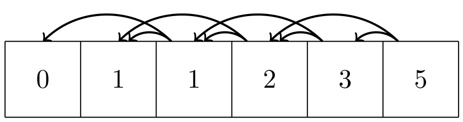
1. if memo[k] ≤ 0

   memo[k] = fib-helper(k - 1) + fib-helper(k - 2)
2. return memo[k]

fib(k):

1. memo = new int[k]
2. memo[0] = 0; memo[1] = 1; memo[2,...,k] = -1.
3. return fib-helper(k)

Example:



Analysis:

- cost to fill in each entry: 1 additions.
- number of entries: k
- total cost: T(k) = k additions.

Conclusion: "linear time."

Note: memoizing redundant computation is an essential part of "dynamic programming."

Iterative Algorithm

Algorithm: Iterative Memoized Fibonacci

fib(k):

1. memo = new int[k];
2. memo[0] = 0; memo[1] = 1;
3. for i = 2...k

   memo[i] = memo[i-1] + memo[i-2]
4. return memo[k]

Question: Can we compute fib with less memory (space)?

Algorithm: Iterative Fibonacci

fib(k):

1. last[0] = 0; last[1] = 1;
2. for i = 2...k
   - tmp = last[1]
   - last[1] = last[0] + last[1]
   - last[0] = tmp
3. return last[1]

Question: fast alg?

Fast Fibonacci

Note: algorithm operates on last like a matrix multiply

fib(k):

1. z = [0,1]; $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
2. multiply $z \times \underbrace{A \times A \times ... \times A}_{k-1 \text{ times}}$
3. return z[1]

Note: just need to compute $z \times A^{k-1}$

Exponentiation

"compute A^k"

Note: If $k = k_1 + k_2$ then $A^k = A^{k_1} A^{k_2}$

- compute A^{k_1} and A^{k_2} and multiply.
- if $k_1 = k_2$ then redundant computation

Idea: factor $A^k = (A^{k/2})^2 \times A^{k\%2}$

Algorithm: Repeated Squaring

1. if $k = 1$ return A
2. k' = $\lfloor k/2 \rfloor$
3. B = repeated-square(A, k')
4. if k odd

   return $B \times B \times A$
5. else

   return $B \times B$

Analysis

Let T(k) = number of multiplies.

$$\begin{aligned} T(1) &= 0 \\ T(k) &\geq T(k/2) + 2 \\ &= T(k/4) + 2 + 2 \\ &= \underbrace{2 + 2 + 2...2}_{\log k \text{ times}} \\ &= 2\log(k) \end{aligned}$$

Note: finding subproblems is important part of "divide and conquer"

Algorithm: Fibonacci numbers via repeated squaring

fib(k):

1. A = $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
2. z = [0,1] $\times$ repeated-square(A, $k-1$)
3. return z[1]

Analysis

$2 \log k$ 2×2 matrix multiplies.

Conclusions

- runtime analysis
- memoization
- divide and conquer