CS660: Algorithms - Lecture 2

Instructor: Hoa Vu

San Diego State University

Divide and Conquer (Recursion) - Tower of Hanoi

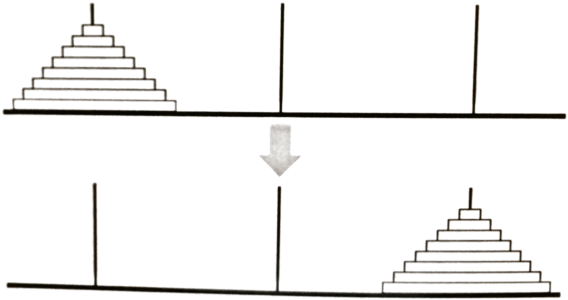

Figure 1.1. The (8-disk) Tower of Hanoi puzzle

- Move the discs from the original needle to the target needle.
- During the process, no bigger disc can be on top of a smaller disc.
- Can only move one disc at the top of a needle to another disc one at a time.
- Design an algorithm that output a solution to the problem.

Divide and Conquer (Recursion) - Tower of Hanoi

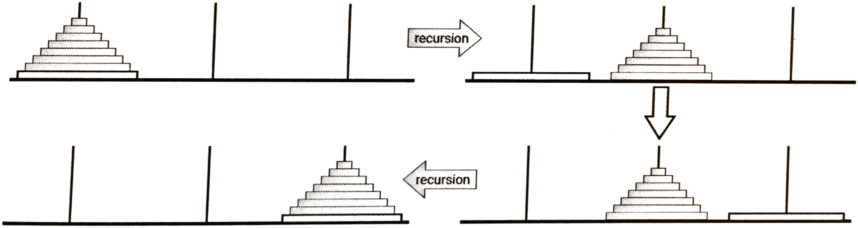

Figure 1.2. The Tower of Hanoi algorithm; ignore everything but the bottom disk.

- First, move the $n - 1$ smallest discs to the non-target needle.
- Move the largest disc to the target needle.
- Move the $n - 1$ smallest discs to the target needle.

Divide and Conquer (Recursion) - Tower of Hanoi

```
HANOI(n, src, dst, tmp):
  if n > 0
        HANOI(n − 1, src, tmp, dst)     ⟨⟨Recurse!⟩⟩
        move disk n from src to dst
        HANOI(n − 1, tmp, dst, src)     ⟨⟨Recurse!⟩⟩
```

Figure 1.4. A recursive algorithm to solve the Tower of Hanoi

Divide and Conquer (Recursion) - Tower of Hanoi

- The running time is given by the recurrence

$$T(n) = 2T(n-1) - 1$$

- Claim: $T(n) = 2^n - 1$. Since we do nothing when $n = 0$, $T(0) = 0$. Prove by induction.
- $T(0) = 0 = 2^0 - 1$.
- Suppose $T(i) = 2^i - 1$. Then $T(i+1) = 2T(i-1) + 1 = 2(2^i - 1) + 1 = 2^{i+1} - 1$.
- Therefore, the running time is $O(2^n + 1) = O(2^n)$.

Divide and Conquer

- Given a problem of size n.
- Strategy: reduce to solving similar problem of size $< n$ and combine.
- Example from last lecture: to sort a list $A[1 \dots n]$
  1. Sort($A[1 : \lceil n/2 \rceil]$).
  2. Sort($A[\lceil n/2 \rceil + 1 : n]$).
  3. Merge.
- Careful: you need to show how to solve the base case where n is small enough.
- The running time to solve sort a list of size n is given by the recurrence $T(n) = 2T(n/2) + O(n)$.
- Show that $T(n) = O(n \log n)$.

Master Theorem (one theorem to rule them all)

- Often, we want to solve recurrences in the form

$$T(n) \le aT(n/b) + cn^{\alpha} \text{ where } c \text{ is some constant.}$$

- Then,

$$T(n) = \begin{cases} O(n^{\alpha}) & \text{if } a < b^{\alpha} \\ O(n^{\log_b a}) & \text{if } a > b^{\alpha} \\ O(n^{\alpha} \log n) & \text{if } a = b^{\alpha} \end{cases}$$

- Check that merge sort takes $O(n \log n)$ time using Master theorem.

Fast Multiplication

- Suppose we want to multiply 2 numbers x and y of length n, say in decimal.
- We can check that addition can be done digit by digit in $O(n)$ time.
- Also, modulo 10^i can be done in constant time, i.e., $345 \bmod 10^2 = 45$ (just output the last i digits).
- Question: How fast can we multiply?

Fast Multiplication

- Naive algorithm:
- $x = 10^{n-1}x_{n-1} + 10^{n-2}x_{n-1} + \ldots + 10^0 x_0$.
- $y = 10^{n-1}y_{n-1} + 10^{n-2}y_{n-1} + \ldots + 10^0 y_0$.
- Compute n^2 terms in $10^{i+j}x_i y_j$ and add them up. Note that x_i and y_j are single digit numbers and therefore can be multiplied in constant time. Hence, the total running time is $O(n^2)$.

Fast Multiplication

- First divide-and-conquer attempt: $x = 10^m a + b$ and $y = 10^m c + d$.
- $(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$.

$\underline{\text{SPLITMULTIPLY}(x, y, n)\text{:}}$

$\quad$ if $n = 1$

$\quad\quad$ return $x \cdot y$

$\quad$ else

$\quad\quad m \leftarrow \lceil n/2 \rceil$

$\quad\quad a \leftarrow \lfloor x/10^m \rfloor;\ \ b \leftarrow x \bmod 10^m \qquad \langle\langle x = 10^m a + b \rangle\rangle$

$\quad\quad c \leftarrow \lfloor y/10^m \rfloor;\ \ d \leftarrow y \bmod 10^m \qquad \langle\langle y = 10^m c + d \rangle\rangle$

$\quad\quad e \leftarrow \text{SPLITMULTIPLY}(a, c, m)$

$\quad\quad f \leftarrow \text{SPLITMULTIPLY}(b, d, m)$

$\quad\quad g \leftarrow \text{SPLITMULTIPLY}(b, c, m)$

$\quad\quad h \leftarrow \text{SPLITMULTIPLY}(a, d, m)$

$\quad\quad$ return $10^{2m} e + 10^m (g + h) + f$

Fast Multiplication

- Observation: $ac + bd - (a-b)(c-d) = bc + ad$.
- So we only need to use 3 recursive calls
  $FastMultiply(a, c, m)$,
  $FastMultiply(b, d, m)$, and
  $FastMultiply(a-b, c-d, m)$.

Fast Multiplication

$\underline{\text{FastMultiply}(x, y, n)\text{:}}$

if $n = 1$

    return $x \cdot y$

else

    $m \leftarrow \lceil n/2 \rceil$

    $a \leftarrow \lfloor x/10^m \rfloor$; $b \leftarrow x \bmod 10^m$ $\langle\langle x = 10^m a + b \rangle\rangle$

    $c \leftarrow \lfloor y/10^m \rfloor$; $d \leftarrow y \bmod 10^m$ $\langle\langle y = 10^m c + d \rangle\rangle$

    $e \leftarrow \text{FastMultiply}(a, c, m)$

    $f \leftarrow \text{FastMultiply}(b, d, m)$

    $g \leftarrow \text{FastMultiply}(a - b, c - d, m)$

    return $10^{2m} e + 10^m (e + f - g) + f$

Fast Multiplication

- Running time is given by the recurrence $T(n) \leq 3T(n/2) + O(n)$.
- By Master theorem, the running time is $O(n^{\log_2 3}) \approx O(n^{1.58496})$.

Fast Multiplication

- The fastest algorithm for multiplying two n-digit numbers runs in $O(n \log n)$ time! Only recently discovered in 2019.
- The best previous algorithm runs in $O(n \log n \log \log n)$ time, discovered in 1979.

Strassen's algorithm

- We want to multiply two n-by-n matrices X and Z.
- $Z = XY$ then $Z_{ij} = \sum_k A_{ik} B_{kj}$.
- Naive algorithm: $O(n^3)$ time.

Strassen's algorithm

- For simplicity, assume n is a power of 2.

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

- Then,

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

- $T(n) = 8T(n/2) + O(n^2)$ which is $O(n^3)$ using Master theorem. No improvement.

Strassen's algorithm

- For simplicity, assume n is a power of 2.

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

- Then,

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

- Where
  $P_1 = A(F - H), P_2 = (A + B)H, P_3 = (C + D)E, P_4 = D(G - E)$
  $P_5 = (A + D)(E + H), P_6 = (B - D)(G + H), G_7 = (A - C)(E + F)$.
- The new running time is $T(n) = 7T(n/2) + O(n^2)$. By master theorem, the running time is $O(n^{\log_2 7}) \approx O(n^{2.81})$.