CS 170

Collaborators: NONE

4 Recurrence Relations

(a) $T(n) = 4T(n/4) + 32n$

$$\begin{aligned}
T(n) &= 4T\left(\frac{n}{4}\right) + 32n \\
&= 4\left[4 \cdot T\left(\frac{n}{4}\right) + 32 \cdot \frac{n}{4}\right] + 32n \\
&= 4^2 \cdot T\left(\frac{n}{4^2}\right) + 2 \times 32n \\
&= 4^2\left[4 \cdot T\left(\frac{n}{4^3}\right) + 32 \cdot \frac{n}{4^2}\right] + 2 \times 32n \\
&= 4^3 \cdot T\left(\frac{n}{4^3}\right) + 3 \times 32n \\
&= \cdots \\
&= 4^{\log_4 n} \cdot T(1) + \log_4 n \cdot 32n \\
&= n \cdot T(1) + \log_4 n \cdot 32n
\end{aligned}$$

So, $T(n) = \Theta(n \cdot \log n)$.

(b) $T(n) = 4T(n/3) + n^2$

This recurrence relationship is equivalent to

$$T(n) - \frac{9}{5}n^2 = 4\left[T\left(\frac{n}{3}\right) - \frac{9}{5} \cdot \left(\frac{n}{3}\right)^2\right].$$

So,

$$T(n) - \frac{9}{5}n^2 = 4^{\log_3 n}\left[T(1) - \frac{9}{5} \cdot (1)^2\right] = n^{\log_3 4}\left[T(1) - \frac{9}{5} \cdot (1)^2\right].$$

Since $\log_3 4 < 2$, $T(n) = \Theta(n^2)$.

(c) $T(n) = T(3n/5) + T(4n/5)$ (We have $T(1) = 1$)

Let a and b are the lower bound and the upper bound of the range of $T(n)/n^2$ when $1 \leqslant n \leqslant 4$. That is,

$$an^2 \leqslant T(n) \leqslant bn^2, \forall n \in [1, 4].$$

We can prove by induction that for all integer $k \geqslant 2$:

$$an^2 \leqslant T(n) \leqslant bn^2, \forall n \in [1, k], \tag{1}$$

Here we can let n be a general real number.

The base case is just the definition of a and b. Assuming that (1) holds for $(k-1)$, where $k \geqslant 5$, let's consider the case for k. We just need to consider $n \in (k-1, k]$. Since $k \geqslant 5$, $3n/5 < 4n/5 \leqslant k-1$, so

$$a \cdot \left(\frac{3n}{5}\right)^2 \leqslant T\left(\frac{3n}{5}\right) \leqslant b \cdot \left(\frac{3n}{5}\right)^2, \text{ and } a \cdot \left(\frac{4n}{5}\right)^2 \leqslant T\left(\frac{4n}{5}\right) \leqslant b \cdot \left(\frac{4n}{5}\right)^2$$

So,

$$an^2 = a\cdot\left[\left(\frac{3n}{5}\right)^2 + \left(\frac{4n}{5}\right)^2\right] \leqslant T(n) = T\left(\frac{3n}{5}\right)+T\left(\frac{4n}{5}\right) \leqslant b\cdot\left[\left(\frac{3n}{5}\right)^2 + \left(\frac{4n}{5}\right)^2\right] = bn^2$$

As a result, $T(n) = \Theta(n^2)$.

5 In Between Functions

$f(n) = n^{\ln n}$ is a function that satisfies both these properties.

For a given $c > 0$, when $n > e^c$, $\ln n > c$, so $f(n) = n^{\ln n} > n^c$. That is, $f(n) = \Omega(n^c)$.

In addition, for a given $\alpha > 1$, there exists $M > 0$, s.t. when $n > M$, $\log_\alpha n^{\ln n} = \ln n \cdot \log_\alpha n < n$. So,

$$n^{\ln n} < \alpha^n, \forall n > M.$$

That is, $f(n) = \mathcal{O}(\alpha^n)$.

6 Sequences

(a) For $i \geqslant k$,

$$\begin{pmatrix} A_i \\ A_{i-1} \\ \vdots \\ A_{i-k+1} \end{pmatrix} = \begin{pmatrix} b_1 & b_2 & \cdots & b_k \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} A_{i-1} \\ A_{i-2} \\ \vdots \\ A_{i-k} \end{pmatrix}.$$

Let

$$\mathbf{B} = \begin{pmatrix} b_1 & b_2 & \cdots & b_k \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}.$$

To compute A_n, we just need to raise $\mathbf{B}$ to the power $n-k+1$. Our algorithm breaks into 2 steps: computing $\mathbf{B}^{n-k+1}$ and computing $\mathbf{B}^{n-k+1} \cdot (A_{k-1}, A_{k-2}, \cdots, A_0)^{\mathrm{T}}$.

Algorithm 1. Matrix Exponentiation

```
function matexp(B, n)
Input:  a k × k matrix B, an integer number n
Output:  B^n

if n = 1:  return B
C = matexp(B, ⌊n/2⌋)
if n is even:
    return C^2
else:
    return BC^2
```

The correctness of this recursive algorithm is self-evident.

Since we compute all numbers mod 50, and k is a constant, the computation of $\mathbf{C}^2$ or $\mathbf{BC}^2$ takes some constant time C. So, the time for evaluate `matexp`$(\mathbf{B}, n)$ is $T(n) \leqslant T(n/2) + C$. Therefore, $T(n) = \mathcal{O}(\log n)$. In addtion, computing $\mathbf{B}^{n-k+1} \cdot (A_{k-1}$ costs some constant time. As a result, A_n can be computed by this algorithm in $\mathcal{O}(\log n)$ time.

(b) Note that we only want the answer mod 50, we can design an look-up algorithm to get A_n in some constant time. Consider vector $(a_0, a_1, \cdots, a_{k-1})^{\mathrm{T}}$, where $a_i \in \{1, 2, \cdots, 50\}$ $(i = 0, 1, \cdots, k-1)$. Such vectors can only have $50k$ different possible values. For $(50k+1)$ vectors $(A_0, A_1, \cdots, A_{k-1})^{\mathrm{T}}$, $(A_1, A_2, \cdots, A_k)^{\mathrm{T}}$, $\cdots$, $(A_{50k}, A_{50k+1}, \cdots, A_{51k-1})^{\mathrm{T}}$, according to Pigeonhole principle, there exist two equal vectors $(A_r, A_{r+1}, \cdots, A_{r+k-1})^{\mathrm{T}} = (A_s, A_{s+1}, \cdots, A_{s+k-1})^{\mathrm{T}}$, where $0 \leqslant r < s \leqslant 50k$. Such vectors will repeat every $(s-r)$ iterations, and A_n repeats as well. $(s-r)$ depends on $k, b_1, b_2, \cdots, b_k$, but not n.

Therefore, we can look A_n up by computing $(n-r) \mod (s-r)$.

7 Decimal to Binary

Let $a = \overline{a_{n-1}\cdots a_1 a_0} = a_{n-1}\cdot 10^{n-1} + \cdots + a_1 \cdot 10 + a_0 = a_h \cdot 10^{n/2-1} + a_l$, where $a_h = a_{n-1}\cdot 10^{n/2-1} + \cdots + a_{1+n/2}\cdot 10 + a_{n/2}$, $a_l = a_{n/2-1}\cdot 10^{n/2-1} + \cdots + a_1\cdot 10 + a_0$. The divide-and-conquer algorithm is:

Algorithm 2. Decimal to Binary

function to_binary(a)

Input: the decimal representation of an integer $a = \overline{a_{n-1}\cdots a_1 a_0} = a_{n-1}\cdot 10^{n-1} + \cdots + a_1\cdot 10 + a_0 = a_h\cdot 10^{n/2-1} + a_l$

Output: the binary representation of a

if $n = 0$: return a_0's binary representation.
$b_h = \texttt{to\_binary}(a_h)$
$b_l = \texttt{to\_binary}(a_l)$
$p = \texttt{pow}(1010_2, n/2)$
return $b_h \cdot p + b_l$ (using Karatsuba's algorithm)

To analyze its running time, we first claim that computing 10^n in binary takes $\mathcal{O}(n^{\log_2 3})$ time using Karatsuba's algorithm:

Algorithm 3. Binary Exponentiation

function pow(1010_2, n)

Input: a binary number 1010_2 (10 in decimal), an integer n

Output: $(1010_2)^n$ in binary

```
if k = 0:  return 1010_2
y = pow(1010_2, ⌊n/2⌋)
if n is even:
    return y · y (using Karatsuba's algorithm)
else:
    return 1010_2 · y · y (using Karatsuba's algorithm)
```

The running time for Algorithm 3 is $T_3(n)$.

$$T_3(n) = T_3(n/2) + \mathcal{O}(n^{\log_2 3}) \tag{2}$$

From master theorem, we have $T_3(n) = \mathcal{O}(n^{\log_2 3})$.

Now, we come back to Algorithm 2. When calling `to_binary`, besides 2 recursive calling, we need to compute p and a multiplication and a addition, which takes $\mathcal{O}(n^{\log_2 3})$ time. So, the running time of Algorithm 2:

$$T_2(n) = 2T_2(n/2) + \mathcal{O}(n^{\log_2 3}). \tag{3}$$

According to master theorem, $T_2 = \mathcal{O}(n^{\log_2 3})$.