4. Fast exponentiation

Thm 4.1 Let G be a semigroup and assume we can multiply two el. of G in $O(1)$. Then, we can compute x^k for $x \in G$ and $k \geq 1$ in time $O(\log k)$ (for large n) on an $O(\log k)$-bit RAM.

Pf If $2 \mid n$, then $x^k = (x^{k/2})^2$.

If $2 \nmid n$, then $x^k = (\underbrace{x^{(k-1)/2}}_{\substack{\text{computable} \\ \text{in} \leq C \log_2 \frac{k}{2} \\ = C(\log_2 k - 1)}})^2 \cdot x$

$\square$

(See also pf 2 on following page.)

Thm 4.2 We can compute x^k for any binary integers $x \in \mathbb{Z}$ with $\leq n$ bits and $k \geq 1$ in time $O(nk)$ on an $O(\log(nk))$-bit RAM.

Pf As for Thm 4.1, using that x^k has $O(nk)$ bits, so after computing $x^{\lfloor k/2 \rfloor}$ recursively in time $\leq C n \cdot \frac{k}{2}$, we can compute x^k in time $O(nk)$. $\leadsto$ total time: geom. series. $\square$

Rmk The obvious method ($x^k = x^{k-1} \cdot x$) would (usually) have running time $\Theta(nk^2)$ because the nr. of digits in step i is $\sim ni$ and there are k steps.

Pf 2 of Thm 4.1: Write $k = \sum_{i=0}^{m} a_i 2^i$, $a_i \in \{0,1\}$

$$\Rightarrow x^k = \prod_{\substack{i: \\ a_i = 1}} x^{2^i}.$$

Compute $b_i := x^{2^i}$ for $i = 0, \dots, m$ using the recurrence $b_{i+1} = b_i^2$.

Then, compute the product $\prod_{\substack{i: \\ a_i=1}} b_i$. $\square$

Rmk Similar for $(\mathbb{Q},\cdot)$, *if* you allow nonreduced fractions $\frac{p}{q}$ (reducing the final result would take time $O(nk\log(nk))$).

$(M_n(\mathbb{Q}),\cdot)$,

$(K[X],\cdot)$, $(K(X),\cdot)$, ...

Warning You can often do faster!

E.g. for the semigroup $(\mathbb{Q},+)$: you can compute $k\cdot x$ in $O(n+\log k)$.

Cor 4.3 You can compute the n-th Fibonacci number F_n in time $O(n)$ ...

F_n: $O(n)$ digits

You can compute $F_n \bmod p$ in time $O_p(\log n)$.

Pf
$$\begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n}_{\substack{O(n)\text{-bit matrice} \\ \text{computable} \\ \text{in } O(n).}} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

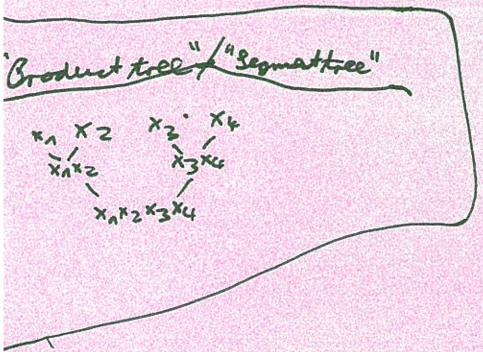
$\square$

5. Multiplying more than two things

Thm 5.1 We can compute the prod. $x_1 \cdots x_k$ for any bin. int. $x_1, \dots, x_k$ with $\le n$ bits in time $\mathcal{O}(nk \log k)$ on an $\mathcal{O}(\log(nk))$-bit RAM.

Pf w.l.o.g. $k = 2^a$.

$$x_1 \cdots x_k = \underbrace{\underbrace{(x_1 \cdots x_{2^{a-1}})}_{\substack{\mathcal{O}(n \cdot 2^{a-1}) \text{ bits} \\ \text{time } \le C n \cdot 2^{a-1}(a-1)}} \underbrace{(x_{2^{a-1}+1} \cdots x_{2^a})}_{\dots}}_{\substack{\mathcal{O}(n \cdot 2^a) \text{ bits}}}$$



time $\le C n \cdot 2^a (a-1) + \mathcal{O}(n \cdot 2^a)$

$\le C n 2^a a$ for large C. $\square$

Rmk Obvious alg.: time $\mathcal{O}(nk^2)$

Rmk Similar for $(\mathbb{Q}, \cdot)$, $(M_n(\mathbb{Q}), \cdot)$, $(\mathbb{Q}, +)$, ...

Rmk There are better alg. for $(\mathbb{Z}, +)$, $(\mathbb{Z}, \max)$, $(F_2, \cdot)$, ...

free group over two elements

Don't reduce intermediate results! (Computing the gcd would take nonlinear time.

Cor 5.2 Given integers $x_1, \dots, x_k$ with $\le n$ bits, you can compute the numerator and denom. p, q of a continued fraction

$$\frac{p}{q} = x_1 + \cfrac{1}{x_2 + \cfrac{1}{\ddots \, \frac{1}{x_k}}}$$

in $O(nk\log(k))$ on an $O(\log(nk))$-bit RAM.

Pf By induction,

$$\begin{pmatrix} p \\ q \end{pmatrix} = \underbrace{\begin{pmatrix} x_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} x_k & 1 \\ 1 & 0 \end{pmatrix}}_{\text{compute this prod. in } O(nk\log k)} \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

$\square$

Cor 5.3 Given integers $a_0, \dots, a_n$ and x with $\le m$ bits, you can compute $\sum_{i=0}^{n} a_i x^i$ (in binary) in time $O(mn \log n)$.

Pf By induction,

$$\begin{pmatrix} \sum_{i=0}^{n} a_i x^i \\ x^{n+1} \end{pmatrix} = \begin{pmatrix} 1 & a_n \\ 0 & x \end{pmatrix} \cdots \begin{pmatrix} 1 & a_0 \\ 0 & x \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

$\square$

What if $x_1, \dots, x_k$ have very different numbers of bits?

Thm 5.4 Let $x_1, \dots, x_k$ be integers with $n_1, \dots, n_k$ bits.

We can compute $x_1 \cdots x_k$ in time

$$O\left(\sum_{i=1}^{k} n_i \left(\log \frac{n_1 + \dots + n_k}{n_i} + 1 \right) \right)$$ on an $O(\log(\sum n_i))$-bit RAM.

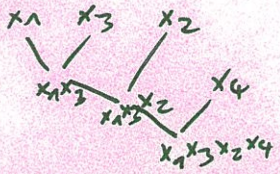
Pf ~~Start with the list of numbers.~~

W.l.o.g. $x_1, \dots, x_n \neq 0$, $\Rightarrow n_i = \log|x_i| + O(1)$.

Start with the list $x_1, \dots, x_n$.

In each step, replace the two integers t from the list with the smallest $\log|t|$ by their product, until there's just one number.

$\leadsto$ Product tree



Running time $\leq \sum_i \log|x_i| \cdot$ (distance of x_i from root)

$= \Big[(n_1 + \dots + n_k)$ times the average length of Huffman code over $x_1, \dots, x_n$ if the probability of x_i is $p_i = \frac{n_i}{n_1 + \dots + n_k} \Big]$

~~Shannon entropy~~

$= (n_1 + \dots + n_k) \cdot \Big[\underbrace{\text{Shannon entropy}}_{\sum p_i \log \frac{1}{p_i}} + O(1) \Big]$

$\square$

A Theorem about Shannon codes
(Shannon: A Mathematical Theory of Communication; or look up "Shannon-Fano coding" on Wikipedia)

Thm 5.5 Let x be an int. with n bits and let $y_1, \dots, y_k$ be integers with $m_1, \dots, m_k$ bits.
We can compute $x \bmod y_1, \dots, x \bmod y_k$ in time

$$O\left(n + \sum m_i \left(\log \frac{m_1 + \dots + m_k}{m_i} + 1\right)\right) \quad \dots$$

Pf Consider the product tree for $y_1, \dots, y_k$ constructed in the proof of Thm 5.4.

For each node (labeled t), compute $x \bmod t$, starting from the root. Note that if the parent node is labeled s, then $\underset{t \mid s, \text{ so}}{(x \bmod t)} = ((\underbrace{x \bmod s}_{< |s|}) \bmod t)$.

$\square$