Official UWCS ProgComp Editorial

In Progress...

Ed & Discrim

Contents

Hints

5. Wall-o-buttons

5.1 Hint 1

Think recursively. Say we order the buttons as follows:

$$1 \quad 2 \quad 3 \quad \cdots \quad n$$

Say that we press button 1.

$$1 \quad 2 \quad 3 \quad \cdots \quad n$$

Now, we can't press button 2, but we do have a choice when it comes to button 3. In fact, since buttons 1 and 2 are out of commission, it's as if we just have

$$3 \quad 4 \quad 5 \quad \cdots \quad n$$

Reindexing... we have

$$1 \quad 2 \quad 3 \quad \cdots \quad n-2$$

What happens if we don't press button 1?

5.2 Hint 2

Is there a quick way to compute $a^b \pmod{M}$ in $\log(b)$ time?

Say I wanted to evaluate 3^{21}, and I don't want to just do $\underbrace{3 \cdot 3 \cdot 3 \cdot \ldots \cdot 3}_{21}$ (**21** multiplications).

What if we tried first evaluating $\{3, 3^2\}$ and using these numbers? We can then evaluate $3^{21} = \underbrace{3^2 \cdot 3^2 \cdot 3^2 \cdot \ldots \cdot 3^2}_{10} \cdot 3$ for a total of **1+11** multiplications, 1 to get 3^2 and 11 to find 3^{21}.

This seems promising, let's go one step further.

$\{3, 3^2, 3^4\}$ and we have $3^{21} = \left(3^4\right)^5 3$ (**2** + **6** multiplications).
$\{3, 3^2, 3^4, 3^8\}$ and we have $3^{21} = \left(3^8\right)^2 \cdot 3^4 \cdot 3$ (**3** + **4** multiplications).

This motivates a fast binary exponentiation algorithm. Perform $\log_2(b)$ multiplications to create the "basis" $\{a, a^2, \ldots, a^{2^k}\}$, then we can evaluate $a^b = a^{2^b_1} \cdot a^{2^b_2} \ldots a^{2^b_k}$. The most multiplications we'd ever have in this stage is if b is one less than a power of 2, and in that case we'd have the number of binary digits of b as the number of multiplications, which is $\sim O(\log_2(b))$.

So the entire thing runs in logarithmic time. This is relevant to the original problem at hand.

5.3 Solution

Look at Hint 1 and Hint 2 before proceeding.

So, in Hint 1, we saw that if we hit Button 1, it's equivalent to counting the number of ways of pressing $n-2$ buttons.
If we don't hit Button 1, then we have the freedom to hit Button 2 (or not). It's equivalent to counting the number of ways to press $n-1$ buttons.

If we define $f(i)$ to be the number of ways to hit i buttons, then we have

$$f(i) = \underbrace{f(i-1)}_{\text{No hitting 1}} + \underbrace{f(i-2)}_{\text{Hitting 1}}$$

This is the recurrence! But what are our base cases? $f(1) = 2$ and $f(2) = 3$.

It turns out that $f(n) = F_{n+2}$, where F_n is the **Fibonacci** sequence. So it's just shifted over by 2.

So how can we use Hint 2 to find F_n fast, even if $n \sim 10^{18}$?

$$\begin{aligned} F_n &= F_{n-1} + F_{n-2} \\ F_{n-1} &= F_{n-1} \end{aligned}$$

Turns out, we can express this as a matrix.

$$\underbrace{\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}}_{\mathbf{F_n}} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}}_{\mathbf{F_{n-1}}}$$

$$\mathbf{F_n} = A\mathbf{F_{n-1}}$$

Let's repeat this.

$$\begin{aligned} \mathbf{F_n} &= A\mathbf{F_{n-1}} \\ \mathbf{F_n} &= A^2\mathbf{F_{n-2}} \\ &\vdots \\ \mathbf{F_n} &= A^n\mathbf{F_0} \end{aligned}$$

We define $F_{-1} = 0$ for convenience. Anyway, that A^n thing should look familiar. In fact, it's pretty much the exact same thing as Hint 2. Whether it be a matrix or an integer, as long as it supports associativity for $\times$, the presence of an identity element (1), and closure $(\forall\, a, b \in S, a \times b \in S)$, we can use **binary exponentiation**.

> **Ⓡ Remark**
>
> More formally, the set of elements must form a monoid for binary exponentiation to hold. (There are weaker characterisations, like power-associativity and magmas, but the monoid one is the most common).