Binary Exponentiation

Dragan Marković

As computer scientists, brilliant and proud people, we are often asked to compute $\boldsymbol{A^B \% P}$, where $\boldsymbol{\%}$ denotes remainder by division (modulus), $\boldsymbol{B}$ is a positive integer and $\boldsymbol{P}$ is usually prime, but not necessarily.

First thing that comes to mind is to loop $\boldsymbol{B}$ times, and each time multiply our cumulative result with $\boldsymbol{A}$. This would achieve the desired result, however in $\boldsymbol{O(B)}$ time. It turns out there is a much better solution which runs in $\boldsymbol{O(logB)}$ time.

Now, thing we have to note is that every natural number can be **uniquely** represented as sum of powers of two. That is to say, for every natural number $\boldsymbol{N = \sum 2^i}$, this sum can be obtained by writing a number in binary base. Simple example: $\boldsymbol{53_{10} = 110101_2 = (2^0 + 2^2 + 2^4 + 2^5)_{10}}$. We do this by enumerating all bits from right to left, with the rightmost bit having index zero, and for every bit that is equal to $\boldsymbol{1}$ (set or turned on) we add $\boldsymbol{2^{bitIdx}}$ to the sum.

It's time to use this fact to our advantage, let's write $\boldsymbol{B}$ as this sum, $\boldsymbol{B = \sum 2^i}$. Now, our problem turns into calculating $\boldsymbol{A^{\sum 2^i} = \prod A^{(2^i)}}$. For example: $\boldsymbol{A^{53} = A^{2^0+2^2+2^4+2^5} = A^{2^0} \cdot A^{2^2} \cdot A^{2^4} \cdot A^{2^5}}$, since there can be at most $\boldsymbol{O(number\_of\_bits\_in\_B)}$ summands this part is $\boldsymbol{O(logB)}$.

We have our $\boldsymbol{O(logB)}$, only question that's remaining is, how do we compute $\boldsymbol{A^{2^i}}$ efficiently. We could again loop $\boldsymbol{2^i}$ times, but that defeats the entire purpose of the algorithm, since $\boldsymbol{2^i}$ can be as large as $\boldsymbol{B}$, when $\boldsymbol{B}$ itself is a power of two. Here we can use the fact that $\boldsymbol{A^{2^i} = (A^{2^{i-1}})^2}$, when we expand this we get $\boldsymbol{A^{2^i} = (...(((A^2)^2)^2)...)^2}$ ($\boldsymbol{A}$ squared, $\boldsymbol{i}$ times). So to calculate $\boldsymbol{A^{2^i}}$ we need to square the number $\boldsymbol{A}$, $\boldsymbol{i}$ times, since $\boldsymbol{i}$ is at most $\boldsymbol{log_2 B}$ this calculation is also $\boldsymbol{O(logB)}$.

Finally, with all this information in mind, we can devise the following algorithm:

1. *Iterate through all the bits of* $\boldsymbol{B}$ *from right to left.*
2. *At all times maintain* $\boldsymbol{A^{2^i}}$.
3. *When you encounter bit that is equal to* $\boldsymbol{1}$, *multiply the cumulative result with* $\boldsymbol{A^{2^i}}$.
4. *Do all of these operations modulo* $\boldsymbol{P}$ *to avoid overflows.*