Notes for competitive programming

James Sungarda

May 28, 2025

1 Useful algorithms

1.1 Binary exponentiation

The calculation a^b can be calculated as $a^{b/2^2}$ for *even* b or $a^{b/2^2+1}$ for *odd* b.
We do this recursively and thus the time complexity is $O(\log b)$.

```
long long binpow(long long a, long long b) {
    long long result = 1;

    while (b > 0) {
        if (b % 2 == 1) result *= a;

        a *= a;
        b /= 2;
    }

    return result;
}
```

Note: The recursive version is a more intuitive way to understand the algorithm, but for efficiency sake we will be using the iterative version.
The idea of the iterative version works by the realizing that every number b can be represented in binary.
e.g. $b = 13$ can be represented as 1101_2 which is equal to $2^3 + 2^2 + 2^0$.
Thus we "multiply" result for every 1 in the binary representation of b.

1.2 Binary search

Binary search works by having two pointers *left* and *right* that point to the first and last elements of the array respectively.
The time complexity is $O(\log n)$.

```
int binary_search(int arr[], int n, int x) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) left = mid + 1;
        else right = mid - 1;
    }
    return -1; // not found
}
```

2 Concepts

2.1 Bitwise operators

Used to manipulate bits of numbers.

- **AND** & : 1 if both bits are 1
- **OR** | : 1 if at least one bit is 1
- **XOR** $\wedge$: 1 if bits are different
- **NOT** $\sim$: flips the bits
- **Left shift** $\ll$: shifts bits to the left
- **Right shift** $\gg$: shifts bits to the right

2.1.1 Multiplication and division by 2

Multiplication and division by 2 can be done by using left and right shift respectively.
e.g. An integer $a = 11$ can be represented in binary as 01011_2, or $2^3 + 2^1 + 2^0 = 11$. Multiplying a by 2 can be done by shifting the bits to the left by 1:
$01011_2 \ll 1 = 10110_2$ which is equal to $2^{3+1} + 2^{1+1} + 2^{0+1} = 2^4 + 2^2 + 2^1 = 22$.

2.1.2 Even and odd numbers

An integer a is even if $a\&1 = 0$ and odd if $a\&1 = 1$.
This is because the last bit of an integer is 1 if it is odd and 0 if it is even.

2.1.3 Fast division

Fast division is done by using bitwise operators.
The idea is to use the fact that a/b can be represented as $a \times (b^{-1})$.
We can use the `binpow` function to calculate b^{-1} as b^{MOD-2} where MOD is a prime number.

This is done by using Fermat's little theorem which states that if p is a prime number and a is an integer not divisible by p, then $a^{p-1} \equiv 1 \mod p$.
Thus, we can calculate $a^{-1} \equiv a^{p-2} \mod p$.

```cpp
#include <bits/stdc++.h>

using namespace std;

#define int long long

const int MOD = 1e9 + 7;
int binpow(int a, int b) {
    int result = 1;
    while (b > 0) {
        if (b % 2 == 1) result = (result * a) % MOD;
        a = (a * a) % MOD;
        b /= 2;
    }
    return result;
}
```

```
int fast_div(int a, int b) {
    return (a * binpow(b, MOD - 2)) % MOD;
}
```

3 Data Structures

3.1 Stack

Stack follows the *LIFO* (Last In First Out) principle.
When a new element is inserted, it will be put first. Similarly, Last element added is the first element to be removed.

Functions: .pop(), .first(), empty(), size(), .push()

4 Useful snippets

4.1 Fast input/output

Contains other ease of use functions.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define fastio ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int32_t main() {
    fastio;

    // Your code here
}
```