← Notes

Lucas' Theorem. Wilson's Theorem

7 CodeMonk Lucas' Theorem Wilson's Theorem Binomial-coefficients Modular-arithmetic

2 LIVE EVENTS

Lucas' Theorem

Statement:

$C(N, K) \% MOD = (C(n_0, k_0) * C(n_1, k_1) * \ldots * C(n_{m-1}, k_{m-1})) \% MOD$

$n_0, n_1, \ldots n_{m-1}$ and $k_0, k_1, \ldots k_{m-1}$ are representations of the numbers **N** and **K** in the scale of notation with base **MOD**. In other words:

$N = n_0 * MOD^0 + n_1 * MOD^1 + \ldots + n_{m-1} * MOD^{m-1}$

$K = k_0 * MOD^0 + k_1 * MOD^1 + \ldots + k_{m-1} * MOD^{m-1}$

C(N, K) is Binomial coefficient (number of ways to choose **K** elements from a set of **N** elements).

Conditions: **MOD** is a prime number (look at the end of the article to know what can we do with not prime **MOD**), and you should be able to calculate **$C(n_i, k_i) \% MOD$**, where **$(0 \le n_i, k_i < MOD)$**.

Advices: this theorem is very useful in case **N ≥ MOD**, otherwise it's better to use formula **C(N, K) = N! / ((N - K)! * K!)** and tricks #2 or #3 from there. If **N ≥ MOD** then **N! % MOD = 0**, when **C(N, K) % MOD** is not necessary equals to **0**.

Realization: let's see how can we get representation of some number **N** in the scale of notation with base **MOD**:

```
vector<int> getRepresentation(int N) {
    vector<int> res;
    while (N > 0) {
        res.push_back(N % MOD);
        N /= MOD;
    }
    return res;
}
```

Let **n** will be representation of **N** and **k** will be representation of **K**. They are not necessary have the same length. If **K > N** we can easily say that **C(N, K) = 0**. Otherwise **k** has less or equal length than **n**. To make them the same length we can add some extra zeroes to **k** and make them both of length of **n**, or we can take only some first elements of **n** and make them both of length of **k**. The second way has more sense because **$C(n_i, 0) = 1$**.

So the main part of code looks like:

```
vector<int> n = getRepresentation(N);
vector<int> k = getRepresentation(K);
long long res = 1;
for (int i = 0; i < k.size(); ++i) {
    res = (res * C(n[i], k[i])) % MOD;
}
```

Let's talk about function **C(n[i], k[i])** in more detail. It's easy to see that **(0 ≤ n[i], k[i] < MOD)**, so we can use formula **C(N, K) = N! / ((N - K)! * K!)** and trick #3 from there:

```
int C(int N, int K) {
    if (K > N) {
        return 0;
    }
    return (((fact[N] * binpow(fact[N - K], MOD - 2)) % MOD) * bi
}
```

Let's precalc all possible factorials modulo **MOD** and store them in the array **fact**:

```
long long fact[MOD];
fact[0] = 1;
for (int i = 1; i < MOD; ++i) {
    fact[i] = (fact[i - 1] * i) % MOD;
}
```

Function **binpow** is just Fast exponentation, it can calculate **A^N % MOD** in **O(log(N))** time:

```
int binpow(int a, int n) {
    long long res = 1;
    while (n > 0) {
        if (n % 2 != 0) {
```

```
                res = (res * a) % MOD;
            }
            a = ((long long)a * a) % MOD;
            n /= 2;
        }
        return (int)res;
    }
```

If **n[i]** and **k[i]** are small enough instead of using formulas and tricks we can just precalc Pascal's triangle and then get **C(n[i], k[i])** in **O(1)**:

```
int C[MOD][MOD];
for (int i = 0; i < MOD; ++i) {
    for (int j = 0; j <= i; ++j) {
        if (i == 0 || j == 0) {
            C[i][j] = 1;
        } else {
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % MOD;
        }
    }
}
```

Trick with not prime MOD: let's factorize **MOD = mod_1^{q1} * mod_2^{q2} * ... * mod_m^{qm}** and calculate **C(N, K) % mod_1, C(N, K) % mod_2, ... C(N, K) % mod_m** using Lucas' Theorem. Now we can use Chinese remainder theorem to restore **C(N, K) % MOD**.

Wilson's Theorem

Statement:

Natural number N is a prime number if and only if (N - 1)! + 1 is divisible by N.

AUTHOR

Boris Sokolov
C++ Developer at Module...
Simferopol, Crimea, Russian Federation
2 notes

Write Note

My Notes

Drafts

TRENDING NOTES

Strings And String Functions
written by Vinay Singh

Segment Tree and Lazy Propagation
written by Akash Sharma

Number Theory - II
written by Tanmay Chaudhari

Matrix exponentiation
written by Mike Koltsov

Graph Theory - Part II
written by Pawel Kacprzak

more ...

RECRUIT

Developer Sourcing

Lateral Hiring

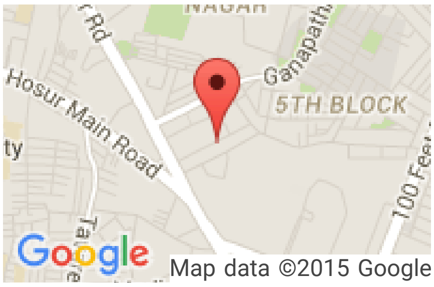
Campus Hiring

FAQs

Customers

Annual Report

REACH US



IIIrd Floor, Salarpuria Business Center,
4th B Cross Road, 5th A Block,
Koramangala Industrial Layout,
Bangalore, Karnataka 560095, India.

contact@hackerearth.com

+91-80-4155-4695

+1-650-461-4192