Ajob Subsequence

Editorial by Bidhan

Solution

The number of ways to choose a subsequence of length `$Y$` from a word of length `$X$` is `$\binom{X}{Y}$`.

So, the total number of ways to choose the desired subsequence abiding by the specifications explained in the problem statement is,

`
$$\binom{N}{N-K} + \binom{N-1}{N-K-1} + \binom{N-2}{N-K-2} + \cdots + \binom{K}{0}$$
`

It can be shown that this sum is equal to `$\binom{N+1}{N-K}$`, as follows:

For binomial coefficients we know that, `
$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$
`

This is called Pascal's identity.

So, `
$$\begin{aligned}\binom{N+1}{N-K} &= \binom{N}{N-K} + \binom{N}{N-K-1} \\ &= \binom{N}{N-K} + \binom{N-1}{N-K-1} + \binom{N-1}{N-K-2} \\ &= \binom{N}{N-K} + \binom{N-1}{N-K-1} + \binom{N-2}{N-K-2} + \binom{N-2}{N-K-3} \\ &= \binom{N}{N-K} + \binom{N-1}{N-K-1} + \binom{N-2}{N-K-2} + \cdots + \binom{K}{0}\end{aligned}$$
`

which is what we wanted to show.

This formula is actually called the *Hockey Stick Pattern*, because if you try to plot the given binomial coefficients in Pascal's triangle, we get some nice hockey stick patterns (see this link for more).

Binomial Coefficient modulo prime `$p$`

Now, to calculate `$\binom{N+1}{N-K}$` for huge numbers, we need to calculate factorial of `$N+1, N-K, K+1$` efficiently.

We can calculate the factorial of huge number `$n$` modulo `$p$` (where `$p$` is a prime) in `$O(p \log_p n)$`. (A very nice tutorial written by **e-maxx** exists here. I have used the exact function implementation in setter's code for better understanding of its usage).

Statistics
Difficulty: 0.772727272
Required Knowledge: C
Publish Date: Jul 01 201

Originally featured in A
Math Programming Con

Note that the powers of `$p$` are not eliminated in that method. We need to be careful while calculating end eliminating it.

Binomial Coefficient modulo prime `$p$` (alternative way)

An alternative way to calculate `$\binom{N+1}{N-K}$` modulo a prime `$p$` is to use Lucas' theorem, which says that:

`

$$‘\binom{n}{r} \equiv \binom{\lfloor n/p \rfloor}{\lfloor r/p \rfloor} \cdot \binom{n \bmod p}{r \bmod p} \pmod p‘$$

`

Using this, we can now calculate the binomial coefficient modulo `$p$` as follows (Python code):

```
def C(n,r):
    if r < 0 or r > n:
        return 0
    if r == 0 or r == n:
        return 1
    if n >= p:
        return C(n/p, r/p) * C(n%p, r%p) % p
    return fac[n] * inv_fac[r] % p * inv_fac[n-r] % p
```

This assumes that you have already calculated all the factorials and inverse factorials from `$0$` to `$p-1$` in the arrays `fac` and `inv_fac`. This precalculation can be done in `$O(p)$` time as follows:

```
inv     = [1] * p
fac     = [1] * p
inv_fac = [1] * p

# calculate inverses mod p
for i in xrange(2,p):
    inv[i] = (p - p/i) * inv[p%i] % p

# calculate factorials and inverse factorials mod p
for i in xrange(2,p):
    fac[i] = fac[i-1] * i % p
    inv_fac[i] = inv_fac[i-1] * inv[i] % p
```

Set by Bidhan

Problem Setter's code :

```
/*
 * Bidhan Roy
 * University of Dhaka
 */

using namespace std;
#include <bits/stdc++.h>

#define foreach(i,n) for(__typeof((n).begin())i =(n).begin();i!=(n).end();i++)
#define sgn(x,y) ((x)+eps<(y)?-1:((x)>eps+(y)?1:0))
#define rep(i,n) for(__typeof(n) i=0; i<(n); i++)
#define mem(x,val) memset((x),(val),sizeof(x));
#define rite(x) freopen(x,"w",stdout);
#define read(x) freopen(x,"r",stdin);
#define all(x) x.begin(),x.end()
#define sz(x) ((i64)x.size())
#define sqr(x) ((x)*(x))
#define pb push_back
#define mp make_pair
#define clr clear()
#define inf (1<<30)
#define ins insert
#define xx first
#define yy second
#define eps 1e-9

typedef long long i64;
typedef unsigned long long ui64;
typedef string st;
typedef vector<i64> vi;
typedef vector<st> vs;
typedef map<i64,i64> mii;
```

```
typedef map<st,i64> msi;
typedef set<i64> si;
typedef set<st> ss;
typedef pair<i64,i64> pii;
typedef vector<pii> vpii;

i64 mod;

i64 Pow(i64 b,i64 p,i64 m){
    i64 ret=1;
    for(i64 i=(1LL<<62); i; i>>=1){
        ret=(ret*ret)%m;
        if(p&i) ret=(ret*b)%m;
    }
    return ret;
}

i64 factmod (i64 n,i64 p) {
    i64 res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
        for (i64 i=2; i<=n%p; ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}

i64 calc(i64 a,i64 b){
    i64 now=1;
    i64 ret=0;
    while(now<=a/b){
        now*=b;
        ret+=a/now;
    }
    return ret;
}

i64 NCR(i64 n,i64 r){
    i64 powerOfMod=0;
    i64 ret=factmod(n,mod); powerOfMod+=calc(n,mod);
    i64 down=factmod(r,mod); powerOfMod-=calc(r,mod);
    down*=factmod(n-r,mod); powerOfMod-=calc((n-r),mod);
    down%=mod;
    down=Pow(down,mod-2,mod);
    ret*=down;
    ret%=mod;
    if(powerOfMod>0)return 0;
    return ret%mod;
}

int main(){
    ios_base::sync_with_stdio(0);
    int test;
    cin>>test;
    while( test-- ){
        i64 n,k,p;
        cin>>n>>k>>p;
        mod=p;
        cout<<NCR(n+1,n-k)<<endl;
    }
    return 0;
}
```