Standard Code Library

Your TeamName

Your School

August 6, 2025

Contents

一切的开始

宏定义

- 需要 C++11

```cpp
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef DEBUG
#ifndef ONLINE_JUDGE
#define zerol
#endif
#endif
#ifdef zerol
#define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#define err(...)
#endif
// -----------------------------------------------------------------------
```

- 调试时添加编译选项 `-DDEBUG`，提交时注释
- 注意检查判题系统编译选项，修改 `#ifndef ONLINE_JUDGE`
- `FOR` ++ 循环 `FOR(循环变量名称，循环变量起始值，循环变量结束值（不含）)`
- `FORD` --循环
- `err()` 调试时输出（支持单层迭代）
- `dbg()` 变色输出变量名和变量值（支持单层迭代）
- 黄色 `33`，蓝色 `34`，橙色 `31`

对拍

- Linux

```bash
#!/usr/bin/env bash
g++ -o r main.cpp -O2 -std=c++11
g++ -o std std.cpp -O2 -std=c++11
while true; do
    python gen.py > in
    ./std < in > stdout
    ./r < in > out
    if test $? -ne 0; then
        exit 0
    fi
    if diff stdout out; then
        printf "AC\n"
    else
        printf "GG\n"
        exit 0
    fi
done
```

- Windows

```bat
@echo off
setlocal enabledelayedexpansion

g++ -o r main.cpp -O2 -std=c++11
g++ -o std std.cpp -O2 -std=c++11

:loop
python gen.py > in
if !errorlevel! neq 0 exit /b
```

```

std.exe < in > stdout
if !errorlevel! neq 0 exit /b

r.exe < in > out
if !errorlevel! neq 0 exit /b

fc /b stdout out > nul
if !errorlevel! equ 0 (
    echo AC
) else (
    echo GG
    exit /b
)

goto loop
```

快速编译运行（配合无插件 VSC）

- Linux

```
#!/bin/bash
g++ $1.cpp -o $1 -O2 -std=c++14 -Wall -Dzerol -g
if $? -eq 0; then
    ./$1
fi
```

- Windows

```
@echo off
:: 参数为文件名（不含.cpp后缀）
g++ %1.cpp -o %1 -O2 -std=c++14 -Wall -Dzerol -g
if %errorlevel% equ 0 (
    %1.exe
)
```

数据结构

ST 表

- 一维

```
#define M 10

struct RMQ {
    int f[22][M];
    inline int highbit(int x) { return 31 - __builtin_clz(x); }
    void init(int* v, int n) {
        FOR (i, 0, n) f[0][i] = v[i];
        FOR (x, 1, highbit(n) + 1)
        FOR (i, 0, n - (1 << x) + 1)
        f[x][i] = min(f[x - 1][i], f[x - 1][i + (1 << (x - 1))]);
    }
    int get_min(int l, int r) {
        assert(l <= r);
        int t = highbit(r - l + 1);
        return min(f[t][l], f[t][r - (1 << t) + 1]);
    }
};
```

- 二维

```
#define maxn 10
LL n, m, a[maxn][maxn];

struct RMQ2D{
    int f[maxn][maxn][10][10];
    inline int highbit(int x) { return 31 - __builtin_clz(x); }
```

```
    inline int calc(int x, int y, int xx, int yy, int p, int q) {
        return max(
            max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
            max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
            );
    }
    void init() {
        FOR (x, 0, highbit(n) + 1)
        FOR (y, 0, highbit(m) + 1)
        FOR (i, 0, n - (1 << x) + 1)
        FOR (j, 0, m - (1 << y) + 1) {
            if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
            f[i][j][x][y] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
                );
        }
    }
    inline int get_max(int x, int y, int xx, int yy) {
        return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
    }
};
```

线段树

朴素线段树

- 默认为最大值，可自行修改 `struct Q` `struct P` `P operator &`
- 注意建树时的下标问题 (1-based)

```
const LL INF = LONG_LONG_MAX;
#define maxn 10
LL n;

namespace SGT {
    struct Q {
        LL setv;
        explicit Q(LL setv = -1): setv(setv) {}
        void operator += (const Q& q) { if (q.setv != -1) setv = q.setv; }
    };
    struct P {
        LL max;
        explicit P(LL max = -INF): max(max) {}
        void up(Q& q) { if (q.setv != -1) max = q.setv; }
    };
    template<typename T>
    P operator & (T&& a, T&& b) {
        return P(max(a.max, b.max));
    }
    P p[maxn << 2];
    Q q[maxn << 2];
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
    void up(int o, int l, int r) {
        if (l == r) p[o] = P();
        else p[o] = p[o * 2] & p[o * 2 + 1];
        p[o].up(q[o]);
    }
    void down(int o, int l, int r) {
        q[o * 2] += q[o]; q[o * 2 + 1] += q[o];
        q[o] = Q();
        up(lson); up(rson);
    }
    template<typename T>
    void build(T&& f, int o = 1, int l = 1, int r = n) {
        if (l == r) q[o] = f(l);
        else { build(f, lson); build(f, rson); q[o] = Q(); }
        up(o, l, r);
    }
    P query(int ql, int qr, int o = 1, int l = 1, int r = n) {
```

```
        if (ql > r || l > qr) return P();
        if (ql <= l && r <= qr) return p[o];
        down(o, l, r);
        return query(ql, qr, lson) & query(ql, qr, rson);
    }
    void update(int ql, int qr, const Q& v, int o = 1, int l = 1, int r = n) {
        if (ql > r || l > qr) return;
        if (ql <= l && r <= qr) q[o] += v;
        else {
            down(o, l, r);
            update(ql, qr, v, lson); update(ql, qr, v, rson);
        }
        up(o, l, r);
    }
}

// ----------------------------------------------------------------------
void solve(){
    vector<LL> arr = {1, 5, 7, 4, 2, 8 ,3, 6, 10, 9};
    n = arr.size();
    SGT::build([&](int idx){
        return SGT::Q(arr[idx-1]);
    });
    for(LL i=1; i<=n; i++){
        dbg(SGT::query(1, i).max);
    }
    SGT::update(2, 4, SGT::Q(-3));
    cout << "MODIFIED\n";
    for(LL i=1; i<=n; i++){
        dbg(SGT::query(1, i).max);
    }
}
```

- 区间修改，区间累加，查询区间和、最大值、最小值。

```
#define maxn 100005
#define INF LONG_LONG_MAX
LL a[maxn], n;

struct IntervalTree {
#define ls o * 2, l, m
#define rs o * 2 + 1, m + 1, r
    static const LL M = maxn * 4, RS = 1E18 - 1;
    LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
    void init() {
        memset(addv, 0, sizeof addv);
        fill(setv, setv + M, RS);
        memset(minv, 0, sizeof minv);
        memset(maxv, 0, sizeof maxv);
        memset(sumv, 0, sizeof sumv);
    }
    void maintain(LL o, LL l, LL r) {
        if (l < r) {
            LL lc = o * 2, rc = o * 2 + 1;
            sumv[o] = sumv[lc] + sumv[rc];
            minv[o] = min(minv[lc], minv[rc]);
            maxv[o] = max(maxv[lc], maxv[rc]);
        } else sumv[o] = minv[o] = maxv[o] = 0;
        if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
        if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
    }
    void build(LL o, LL l, LL r) {
        if (l == r) addv[o] = a[l];
        else {
            LL m = (l + r) / 2;
            build(ls); build(rs);
        }
        maintain(o, l, r);
    }
    void pushdown(LL o) {
        LL lc = o * 2, rc = o * 2 + 1;
        if (setv[o] != RS) {
```

```
            setv[lc] = setv[rc] = setv[o];
            addv[lc] = addv[rc] = 0;
            setv[o] = RS;
        }
        if (addv[o]) {
            addv[lc] += addv[o]; addv[rc] += addv[o];
            addv[o] = 0;
        }
    }
    void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
        if (p <= r && l <= q){
            if (p <= l && r <= q) {
                if (op == 2) { setv[o] = v; addv[o] = 0; }
                else addv[o] += v;
            } else {
                pushdown(o);
                LL m = (l + r) / 2;
                update(p, q, ls, v, op); update(p, q, rs, v, op);
            }
        }
        maintain(o, l, r);
    }
    void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
        if (p > r || l > q) return;
        if (setv[o] != RS) {
            LL v = setv[o] + add + addv[o];
            ssum += v * (min(r, q) - max(l, p) + 1);
            smin = min(smin, v);
            smax = max(smax, v);
        } else if (p <= l && r <= q) {
            ssum += sumv[o] + add * (r - l + 1);
            smin = min(smin, minv[o] + add);
            smax = max(smax, maxv[o] + add);
        } else {
            LL m = (l + r) / 2;
            query(p, q, ls, add + addv[o], ssum, smin, smax);
            query(p, q, rs, add + addv[o], ssum, smin, smax);
        }
    }
    // 简化接口
    void build(int n) {
        build(1, 1, n);
    }

    void range_add(int l, int r, int val) {
        update(l, r, 1, 1, n, val, 1);
    }

    void range_set(int l, int r, int val) {
        update(l, r, 1, 1, n, val, 2);
    }

    void range_query(int l, int r, LL& sum, LL& min_val, LL& max_val) {
        sum = 0;
        min_val = INF;
        max_val = -INF;
        query(l, r, 1, 1, n, 0, sum, min_val, max_val);
    }
} IT;
// --------------------------------------------------------------------
void solve(){
    IT.init();

    n = 5;
    vector<int> data = {1, 3, 5, 7, 9};
    for (int i = 0; i < n; i++) {
        a[i + 1] = data[i];  // 注意：线段树从 1 开始索引
    }

    IT.build(n);

```

```cpp
    LL sum, min_val, max_val;
    IT.range_query(1, 5, sum, min_val, max_val);
    cout << " " << sum << " " << min_val << " " << max_val << endl;

    IT.range_add(2, 4, 2);
    IT.range_query(1, 5, sum, min_val, max_val);
    cout << " " << sum << " " << min_val << " " << max_val << endl;

    IT.range_set(3, 5, 10);
    IT.range_query(1, 5, sum, min_val, max_val);
    cout << " " << sum << " " << min_val << " " << max_val << endl;

    IT.range_query(2, 4, sum, min_val, max_val);
    cout << " " << sum << " " << min_val << " " << max_val << endl;
}
```

树状数组

- 单点修改，区间查询
- 频次统计下的 k 小值
- 维护差分数组时的区间修改，单点查询

```cpp
#define M 100005

namespace BIT {
    LL c[M]; // 注意初始化开销
    inline int lowbit(int x) { return x & -x; }
    void add(int x, LL v) { // 单点加
        for (int i = x; i < M; i += lowbit(i))
            c[i] += v;
    }
    LL sum(int x) { // 前缀和
        LL ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += c[i];
        return ret;
    }
    int kth(LL k) { // 频次统计下从小到大第 k 个，详见应用
        int p = 0;
        for (int lim = 1 << 20; lim; lim /= 2)
            if (p + lim < M && c[p + lim] < k) {
                p += lim;
                k -= c[p];
            }
        return p + 1;
    }
    LL sum(int l, int r) { return sum(r) - sum(l - 1); } // 区间和
    // 区间加（此时树状数组为差分数组，sum(x) 为第 x 个数的值）
    void add(int l, int r, LL v) { add(l, v); add(r + 1, -v); }
}
// ----------------------------------------------------------------------
void solve(){
    vector<LL> a={9, 9, 9, 9, 5, 3, 3, 3, 1, 1};
    LL n = a.size(), i;
    for(i=1; i<=n; i++) BIT::add(a[i-1], 1);
    // 1 1 3 3 3 5 9 9 9 9
    for(i=1; i<=n; i++) cout << BIT::kth(i) << ' ';
}
```

- 区间修改、区间查询

```cpp
#define maxn 100005

namespace BIT {
    int n;
    int c[maxn], cc[maxn];
    inline int lowbit(int x) { return x & -x; }
    void init(int siz){ // 初始化
        n = siz;
        for(LL i=0; i<=n; i++){
            c[i] = cc[i] = 0;
```

```
        }
    }
    void add(int x, int v) { // 不要用这个
        for (int i = x; i <= n; i += lowbit(i)) {
            c[i] += v; cc[i] += x * v;
        }
    }
    void add(int l, int r, int v) { add(l, v); add(r + 1, -v); } // 区间修改
    int sum(int x) { // 前缀和
        int ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * c[i] - cc[i];
        return ret;
    }
    int sum(int l, int r) { return sum(r) - sum(l - 1); } // 区间和
}
// ---------------------------------------------------------------------
void solve(){
    LL i, n=8;
    BIT::init(n);
    BIT::add(2, 4, 2);
    for(i=1; i<=n; i++) cout << BIT::sum(i, i) << ' ';
    cout << '\n';
    cout << BIT::sum(5) << '\n';
    cout << BIT::sum(2, 3) << '\n';
}
```

- 三维

```
#define maxn 105

namespace BIT{
    int n;
    LL c[maxn][maxn][maxn];
    inline int lowbit(int x) { return x & -x; }
    void init(int siz){
        n = siz;
        for(int i=0; i<=n; i++){
            for(int j=0; j<=n; j++){
                for(int k=0; k<=n; k++){
                    c[i][j][k] = 0;
                }
            }
        }
    }
    void update(int x, int y, int z, int d) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= n; j += lowbit(j))
                for (int k = z; k <= n; k += lowbit(k))
                    c[i][j][k] += d;
    }
    LL query(int x, int y, int z) {
        LL ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            for (int j = y; j > 0; j -= lowbit(j))
                for (int k = z; k > 0; k -= lowbit(k))
                    ret += c[i][j][k];
        return ret;
    }
    LL solve(int x, int y, int z, int xx, int yy, int zz) {
        return    query(xx, yy, zz)
        - query(xx, yy, z - 1)
        - query(xx, y - 1, zz)
        - query(x - 1, yy, zz)
        + query(xx, y - 1, z - 1)
        + query(x - 1, yy, z - 1)
        + query(x - 1, y - 1, zz)
        - query(x - 1, y - 1, z - 1);
    }
}
```

数学

快速乘

```
LL mul(LL a, LL b, LL m) {
    LL ret = 0;
    while (b) {
        if (b & 1) {
            ret += a;
            if (ret >= m) ret -= m;
        }
        a += a;
        if (a >= m) a -= m;
        b >>= 1;
    }
    return ret;
}
```

- O(1)

```
LL mul(LL u, LL v, LL p) {
    return (u * v - LL((long double) u * v / p) * p + p) % p;
}
LL mul(LL u, LL v, LL p) { // 卡常
    LL t = u * v - LL((long double) u * v / p) * p;
    return t < 0 ? t + p : t;
}
```

快速幂

- 如果模数是素数，则可在函数体内加上 `n %= MOD - 1;`（费马小定理）。

```
LL bin(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = x * x % MOD)
        if (n & 1) ret = ret * x % MOD;
    return ret;
}
```

- 防爆 LL
- 前置模板：快速乘

```
LL bin(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
        if (n & 1) ret = mul(ret, x, MOD);
    return ret;
}
```

高精度

- https://github.com/Baobaobear/MiniBigInteger/blob/main/bigint_tiny.h，带有压位优化
- 按需实现

```
#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>

struct BigIntTiny {
    int sign;
    std::vector<int> v;

    BigIntTiny() : sign(1) {}
    BigIntTiny(const std::string &s) { *this = s; }
    BigIntTiny(int v) {
        char buf[21];
        sprintf(buf, "%d", v);
        *this = buf;
    }
```

```
    void zip(int unzip) {
        if (unzip == 0) {
            for (int i = 0; i < (int)v.size(); i++)
               v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i * 4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
        } else
            for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >= 0; i--)
                a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] = (i & 1) ? a / 10 : a % 10;
        setsign(1, 1);
    }
    int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos]; }
    BigIntTiny &setsign(int newsign, int rev) {
        for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
            v.erase(v.begin() + i);
        sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ? newsign * sign : newsign);
        return *this;
    }
    std::string to_str() const {
        BigIntTiny b = *this;
        std::string s;
        for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
            s += char(*(b.v.rbegin() + i) + '0');
        return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
    }
    bool absless(const BigIntTiny &b) const {
        if (v.size() != b.v.size()) return v.size() < b.v.size();
        for (int i = (int)v.size() - 1; i >= 0; i--)
            if (v[i] != b.v[i]) return v[i] < b.v[i];
        return false;
    }
    BigIntTiny operator-() const {
        BigIntTiny c = *this;
        c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
        return c;
    }
    BigIntTiny &operator=(const std::string &s) {
        if (s[0] == '-')
            *this = s.substr(1);
        else {
            for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
                v.push_back(*(s.rbegin() + i) - '0');
            zip(0);
        }
        return setsign(s[0] == '-' ? -1 : 1, sign = 1);
    }
    bool operator<(const BigIntTiny &b) const {
        return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) : b.absless(*this));
    }
    bool operator==(const BigIntTiny &b) const { return v == b.v && sign == b.sign; }
    BigIntTiny &operator+=(const BigIntTiny &b) {
        if (sign != b.sign) return *this = (*this) - -b;
        v.resize(std::max(v.size(), b.v.size()) + 1);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i);
            v[i] = carry % 10000, carry /= 10000;
        }
        return setsign(sign, 0);
    }
    BigIntTiny operator+(const BigIntTiny &b) const {
        BigIntTiny c = *this;
        return c += b;
    }
    void add_mul(const BigIntTiny &b, int mul) {
        v.resize(std::max(v.size(), b.v.size()) + 2);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i) * mul;
            v[i] = carry % 10000, carry /= 10000;
        }
    }
    BigIntTiny operator-(const BigIntTiny &b) const {
        if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
        if (sign != b.sign) return (*this) + -b;
```

```cpp
        if (absless(b)) return -(b - *this);
        BigIntTiny c;
        for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
            borrow += v[i] - b.get_pos(i);
            c.v.push_back(borrow);
            c.v.back() -= 10000 * (borrow >>= 31);
        }
        return c.setsign(sign, 0);
    }
    BigIntTiny operator*(const BigIntTiny &b) const {
        if (b < *this) return b * *this;
        BigIntTiny c, d = b;
        for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
            c.add_mul(d, v[i]);
        return c.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator/(const BigIntTiny &b) const {
        BigIntTiny c, d;
        BigIntTiny e=b;
        e.sign=1;

        d.v.resize(v.size());
        double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
                           (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
        for (int i = (int)v.size() - 1; i >= 0; i--) {
            c.v.insert(c.v.begin(), v[i]);
            int m = (int)((c.get_pos((int)e.v.size()) * 10000 + c.get_pos((int)e.v.size() - 1)) * db);
            c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
            while (!(c < e))
                c = c - e, d.v[i] += 1;
        }
        return d.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b * b; }
    bool operator>(const BigIntTiny &b) const { return b < *this; }
    bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
    bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
    bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
};
```

矩阵运算

```cpp
#define MOD 998244353
#define M 10

struct Mat {
    LL m;
    LL v[M][M];
    Mat(int siz=2) {
        m = siz;
        for(int i=0; i<=m; i++){
            for(int j=0; j<=m; j++){
                v[i][j] = 0;
            }
        }
    }
    void eye() { FOR (i, 0, m) v[i][i] = 1; }
    LL* operator [] (LL x) { return v[x]; }
    const LL* operator [] (LL x) const { return v[x]; }
    Mat operator * (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (k, 0, m)
        FOR (i, 0, m) if (A[i][k])
            FOR (j, 0, m)
            ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
        return ret;
    }
    Mat pow(LL n) const {
        Mat A = *this, ret; ret.eye();
        for (; n; n >>= 1, A = A * A)
```

```
            if (n & 1) ret = ret * A;
        return ret;
    }
    Mat operator + (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (i, 0, m)
        FOR (j, 0, m)
        ret[i][j] = (A[i][j] + B[i][j]) % MOD;
        return ret;
    }
    void pprint() const {
        FOR (i, 0, m)
        FOR (j, 0, m)
        printf("%lld%c", (*this)[i][j], j == m - 1 ? '\n' : ' ');
    }
};
// -------------------------------------------------------------------
void solve(){
    Mat mat1, mat2;
    mat1.eye();
    mat1[1][0] = 2; // 0-based
    mat2.eye();
    mat2[1][1] = 4;
    Mat mat3 = mat1 * mat2;
    mat3.pprint();
}
```

质数筛

- $\mathcal{O}(n)$

```
const LL p_max = 1E6 + 100;
LL pr[p_max], p_sz;
void get_prime() {
    static bool vis[p_max];
    FOR (i, 2, p_max) {
        if (!vis[i]) pr[p_sz++] = i;
        FOR (j, 0, p_sz) {
            if (pr[j] * i >= p_max) break;
            vis[pr[j] * i] = 1;
            if (i % pr[j] == 0) break;
        }
    }
}
```

欧拉函数

朴素

```
int phi(int x)
{
    int res = x;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);

    return res;
}
```

筛法求欧拉函数

- 前置模板：质数筛

```
const LL p_max = 1E5 + 100;
LL phi[p_max];
```

```cpp
void get_phi() {
    phi[1] = 1;
    static bool vis[p_max];
    static LL prime[p_max], p_sz, d;
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            phi[i] = i - 1;
        }
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}
```

素性测试

试除法

- $\mathcal{O}(\sqrt{n})$

```cpp
bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
            return false;
    return true;
}
```

Miller-Rabin 素性测试

- 前置：快速幂
- $\mathcal{O}(k \times \log^3 n)$

```cpp
bool miller_rabin(LL n) {
    static vector<LL> tester = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    if (n < 3 || n % 2 == 0) return n == 2;
    if (n % 3 == 0) return n == 3;
    LL u = n - 1, t = 0;
    while (u % 2 == 0) u /= 2, ++t;
    for (auto nt: tester) {
        if(nt >= n) continue;
        LL v = bin(nt, u, n);
        if (v == 1) continue;
        LL s;
        for (s = 0; s < t; ++s) {
            if (v == n - 1) break;
            v = v * v % n;
        }
        if (s == t) return false;
    }
    return true;
}
```

质因数分解

朴素质因数分解

- 前置模板：素数筛
- 带指数
- $\mathcal{O}(\frac{\sqrt{N}}{\ln N})$

```
LL factor[30], f_sz, factor_exp[30];
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % pr[i] == 0) {
                x /= pr[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = pr[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}
```

- 不带指数

```
LL factor[30], f_sz;
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor[f_sz++] = pr[i];
            while (x % pr[i] == 0) x /= pr[i];
        }
    if (x > 1) factor[f_sz++] = x;
}
```

Pollard-Rho

- 前置：素数测试

```
mt19937 mt(time(0));
LL pollard_rho(LL n, LL c) {
    LL x = uniform_int_distribution<LL>(1, n - 1)(mt), y = x;
    auto f = [&](LL v) { LL t = mul(v, v, n) + c; return t < n ? t : t - n; };
    while (1) {
        x = f(x); y = f(f(y));
        if (x == y) return n;
        LL d = gcd(abs(x - y), n);
        if (d != 1) return d;
    }
}

LL fac[100], fcnt;
void get_fac(LL n, LL cc = 19260817) {
    if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
    if (miller_rabin(n)) { fac[fcnt++] = n; return; }
    LL p = n;
    while (p == n) p = pollard_rho(n, --cc);
    get_fac(p); get_fac(n / p);
}

void go_fac(LL n) { fcnt = 0; if (n > 1) get_fac(n); }
```

原根

- 前置模板：质因数分解、快速幂
- 要求 p 为质数
- 别忘了调用质因数分解的函数

```
LL find_smallest_primitive_root(LL p) {
    get_factor(p - 1);
    FOR (i, 2, p) {
        bool flag = true;
```

```
        FOR (j, 0, f_sz)
        if (bin(i, (p - 1) / factor[j], p) == 1) {
            flag = false;
            break;
        }
        if (flag) return i;
    }
//  assert(0);
    return -1;
}
```

欧几里得

- 朴素

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```

- 卡常

```
inline int ctz(LL x) { return __builtin_ctzll(x); }
LL gcd(LL a, LL b) {
    if (!a) return b; if (!b) return a;
    int t = ctz(a | b);
    a >>= ctz(a);
    do {
        b >>= ctz(b);
        if (a > b) swap(a, b);
        b -= a;
    } while (b);
    return a << t;
}
```

扩展欧几里得

- 求 $ax + by = gcd(a, b)$ 的一组解
- 如果 a 和 b 互素，那么 x 是 a 在模 b 下的逆元
- 注意 x 和 y 可能是负数

```
LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) { x = 1; y = 0; return a; }
    LL ret = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
```

中国剩余定理

- 求解线性同于方程
- $$\begin{cases} x & \equiv r_1 \pmod{m_1} \\ x & \equiv r_2 \pmod{m_2} \\ & \vdots \\ x & \equiv r_k \pmod{m_k} \end{cases}$$
- 无解返回 -1
- 前置模板：扩展欧几里得

```
LL CRT(LL *m, LL *r, LL n) {
    if (!n) return 0;
    LL M = m[0], R = r[0], x, y, d;
    FOR (i, 1, n) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
```

```
        x = (r[i] - R) / d * x % (m[i] / d);
        // 防爆 LL
        // x = mul((r[i] - R) / d, x, m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}
```

逆元

- 如果 p 是素数，使用快速幂（费马小定理）
- 前置模板：快速幂

```
inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
```

- 如果 p 不是素数，使用拓展欧几里得
- 前置模板：扩展欧几里得

```
LL get_inv(LL a, LL M) {
    static LL x, y;
    assert(exgcd(a, M, x, y) == 1);
    return (x % M + M) % M;
}
```

- 预处理 1~n 的逆元

```
LL inv[N];
void inv_init(LL n, LL p) {
    inv[1] = 1;
    FOR (i, 2, n)
        inv[i] = (p - p / i) * inv[p % i] % p;
}
```

- 预处理阶乘及其逆元

```
LL invf[M], fac[M] = {1};
void fac_inv_init(LL n, LL p) {
    FOR (i, 1, n)
        fac[i] = i * fac[i - 1] % p;
    invf[n - 1] = bin(fac[n - 1], p - 2, p);
    FORD (i, n - 2, -1)
        invf[i] = invf[i + 1] * (i + 1) % p;
}
```

组合数

组合数预处理（递推法）

```
LL C[M][M];
void init_C(int n) {
    FOR (i, 0, n) {
        C[i][0] = C[i][i] = 1;
        FOR (j, 1, i)
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
    }
}
```

预处理逆元法

- 如果数较小，模较大时使用逆元
- 前置模板：逆元-预处理阶乘及其逆元

```
inline LL C(LL n, LL m) { // n >= m >= 0
    return n < m || m < 0 ? 0 : fac[n] * invf[m] % MOD * invf[n - m] % MOD;
}
```

Lucas 定理

- 如果模数较小，数字较大，使用 Lucas 定理
- 前置模板可选 1：求组合数（如果使用阶乘逆元，需 `fac_inv_init(MOD, MOD);`）

```
LL C(LL n, LL m) { // m >= n >= 0
    if (m - n < n) n = m - n;
    if (n < 0) return 0;
    LL ret = 1;
    FOR (i, 1, n + 1)
        ret = ret * (m - n + i) % MOD * bin(i, MOD - 2, MOD) % MOD;
    return ret;
}
```

- 前置模板可选 2：模数不固定下使用，无法单独使用。

```
LL Lucas(LL n, LL m) { // m >= n >= 0
    return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD : 1;
}
```

求具体值

- 分解质因数法

```
int primes[N], cnt;     // 存储所有质数
int sum[N];     // 存储每个质数的次数
bool st[N];     // 存储每个数是否已被筛掉

void get_primes(int n)      // 线性筛法求素数
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int get(int n, int p)       // 求 n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}


vector<int> mul(vector<int> a, int b)       // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
        c.push_back(t % 10);
        t /= 10;
    }

    return c;
```

```
}

get_primes(a);  // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )     // 求每个质因数的次数
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )     // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);
```

FFT & NTT & FWT

FFT

- 计算多项式乘法，可用于高精度乘法
- $\mathcal{O}(n\log n)$

```
typedef double LD;
const LD PI = acos(-1.0);

struct Complex {
    LD r, i;
    Complex(LD r = 0, LD i = 0) : r(r), i(i) {}
    Complex operator + (const Complex& other) const {
        return Complex(r + other.r, i + other.i);
    }
    Complex operator - (const Complex& other) const {
        return Complex(r - other.r, i - other.i);
    }
    Complex operator * (const Complex& other) const {
        return Complex(r * other.r - i * other.i, r * other.i + i * other.r);
    }
};

// 快速傅里叶变换，p=1 为正向，p=-1 为反向
void FFT(vector<Complex>& x, int p) {
    int n = x.size();
    for (int i = 0, t = 0; i < n; ++i) {
        if (i > t) swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1);
    }
    for (int h = 2; h <= n; h <<= 1) {
        Complex wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
        for (int i = 0; i < n; i += h) {
            Complex w(1, 0);
            for (int j = 0; j < h / 2; ++j) {
                Complex u = x[i + j];
                Complex v = x[i + j + h/2] * w;
                x[i + j] = u + v;
                x[i + j + h/2] = u - v;
                w = w * wn;
            }
        }
    }
    if (p == -1) {
        for (int i = 0; i < n; ++i) {
            x[i].r /= n;
        }
    }
}

// 计算两个多项式的卷积，返回结果多项式的系数向量
vector<LD> convolution(const vector<LD>& a, const vector<LD>& b) {
    int len = 1;
```

```cpp
    int n = a.size(), m = b.size();
    while (len < n + m - 1) len <<= 1;
    vector<Complex> fa(len), fb(len);
    for (int i = 0; i < n; ++i) fa[i] = Complex(a[i], 0);
    for (int i = 0; i < m; ++i) fb[i] = Complex(b[i], 0);
    FFT(fa, 1);
    FFT(fb, 1);
    for (int i = 0; i < len; ++i) {
        fa[i] = fa[i] * fb[i];
    }
    FFT(fa, -1);
    vector<LD> res(n + m - 1);
    for (int i = 0; i < n + m - 1; ++i) {
        res[i] = fa[i].r;
    }
    return res;
}
```

NTT

- 用于大整数乘法时，位数不宜过高（在 MOD=998244353 的情况下，总位数不超过 $12324004(3510^2)$）
- 前置模板：快速幂、逆元

```cpp
const int N = 1e5+10;
const int MOD = 998244353; // 模数
const int G = 3; // 原根

LL wn[N << 2], rev[N << 2];
int NTT_init(int n_) {
    int step = 0; int n = 1;
    for ( ; n < n_; n <<= 1) ++step;
    FOR (i, 1, n)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
    int g = bin(G, (MOD - 1) / n, MOD);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % MOD;
    return n;
}

void NTT(vector<LL> &a, int n, int f) {
    FOR (i, 0, n) if (i < rev[i])
        std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);
            FOR (j, 0, k) {
                LL w = f == 1 ? wn[t * j] : wn[n - t * j];
                LL x = a[i + j];
                LL y = a[i + j + k] * w % MOD;
                a[i + j] = (x + y) % MOD;
                a[i + j + k] = (x - y + MOD) % MOD;
            }
        }
    }
    if (f == -1) {
        LL ninv = get_inv(n, MOD);
        FOR (i, 0, n)
        a[i] = a[i] * ninv % MOD;
    }
}

vector<LL> conv(vector<LL> a, vector<LL> b){
    int len_a = a.size(), len_b = b.size();
    int len = len_a + len_b - 1;
    int n = NTT_init(len);
    a.resize(n);
    b.resize(n);
    NTT(a, n, 1);
    NTT(b, n, 1);
    vector<LL> c(n);
```

```
    for (int i = 0; i < n; ++i) {
        c[i] = a[i] * b[i] % MOD;
    }
    NTT(c, n, -1);
    vector<LL> res(len);
    for (int i = 0; i < len; ++i) {
        res[i] = c[i];
    }
    return res;
}
```

FWT

```
const LL MOD = 998244353;

template<typename T>
void fwt(vector<LL> &a, int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

void AND(LL& a, LL& b) { a += b; }
void OR(LL& a, LL& b) { b += a; }
void XOR (LL& a, LL& b) {
    LL x = a, y = b;
    a = (x + y) % MOD;
    b = (x - y + MOD) % MOD;
}
void rAND(LL& a, LL& b) { a -= b; }
void rOR(LL& a, LL& b) { b -= a; }
void rXOR(LL& a, LL& b) {
    static LL INV2 = (MOD + 1) / 2;
    LL x = a, y = b;
    a = (x + y) * INV2 % MOD;
    b = (x - y + MOD) * INV2 % MOD;
}

int next_power_of_two(int n) {
    if (n <= 0) return 1;
    // __lg(n-1) 返回 n-1 的最高位所在位置（0-based）
    return 1 << (__lg(n - 1) + 1);
}

template<typename T, typename F>
vector<LL> conv(vector<LL> a, vector<LL> b, T f, F inv_f){
    LL len_a = a.size(), len_b = b.size(), len = max(len_a, len_b), n = next_power_of_two(len);
    a.resize(n), b.resize(n);
    fwt(a, n, f), fwt(b, n, f);
    vector<LL> c(n);
    for (int i = 0; i < n; i++) {
        c[i] = a[i] * b[i] % MOD;
    }
    fwt(c, n, inv_f);
    // 提取结果 (可选)
    c.resize(len);
    return c;
}
```

线性基

贪心法

可查询最大异或和

```
struct BasisGreedy{
    ULL p[64];
    BasisGreedy(){memset(p, 0, sizeof p);}
    void insert(ULL x) {
```

```cpp
        for (int i = 63; ~i; --i) {
            if (!(x >> i))  // x 的第 i 位是 0
                continue;
            if (!p[i]) {
                p[i] = x;
                break;
            }
            x ^= p[i];
        }
    }
    ULL query_max(){
        ULL ans = 0;
        for (int i = 63; ~i; --i) {
            ans = std::max(ans, ans ^ p[i]);
        }
        return ans;
    }
};
```

高斯消元法

可查询任意大异或和

```cpp
struct BasisGauss{
    vector<ULL> a;
    LL n, tmp, cnt;

    BasisGauss(){a = {0};}

    void insert(ULL x){
        a.push_back(x);
    }

    void init(){
        n = (LL)a.size() - 1;
        LL k=1;
        for(int i=63;i>=0;i--){
            int t=0;
            for(LL j=k;j<=n;j++){
                if((a[j]>>i)&1){
                    t=j;
                    break;
                }
            }
            if(t){
                swap(a[k],a[t]);
                for(LL j=1;j<=n;j++){
                    if(j!=k&&(a[j]>>i)&1) a[j]^=a[k];
                }
                k++;
            }
        }
        cnt = k-1;
        tmp = 1LL << cnt;
        if(cnt==n) tmp--;
    }

    LL query_xth(LL x){ // 从小到大，若 x 为负数，则查询倒数第几个
        if(x<0) x = tmp + x + 1;
        if(x>tmp) return -1;
        else{
            if(n>cnt) x--;
            LL ans=0;
            for(LL i=0; i<cnt; i++){
                if((x>>i)&1) ans^=a[cnt-i];
            }
            return ans;
        }
    }
};
```

图论

计算几何

字符串

杂项