Standard Code Library

Your TeamName

Your School

August 11, 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

```
@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
```

```
10
11    std.exe < in > stdout
12    if !errorlevel! neq 0 exit /b
13
14    r.exe < in > out
15    if !errorlevel! neq 0 exit /b
16
17    fc /b stdout out > nul
18    if !errorlevel! equ 0 (
19        echo AC
20    ) else (
21        echo GG
22        exit /b
23    )
24
25    goto loop
```

快速编译运行（配合无插件 VSC）

- Linux

```
1    #!/bin/bash
2    g++ $1.cpp -o $1 -O2 -std=c++14 -Wall -Dzerol -g
3    if $? -eq 0; then
4        ./$1
5    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

```
1    #define M 10
2
3    struct RMQ {
4        int f[22][M];
5        inline int highbit(int x) { return 31 - __builtin_clz(x); }
6        void init(int* v, int n) {
7            FOR (i, 0, n) f[0][i] = v[i];
8            FOR (x, 1, highbit(n) + 1)
9                FOR (i, 0, n - (1 << x) + 1)
10                   f[x][i] = min(f[x - 1][i], f[x - 1][i + (1 << (x - 1))]);
11       }
12       int get_min(int l, int r) {
13           assert(l <= r);
14           int t = highbit(r - l + 1);
15           return min(f[t][l], f[t][r - (1 << t) + 1]);
16       }
17   };
```

- 二维

```
1    #define maxn 10
2    LL n, m, a[maxn][maxn];
3
4    struct RMQ2D{
5        int f[maxn][maxn][10][10];
6        inline int highbit(int x) { return 31 - __builtin_clz(x); }
```

```
7     inline int calc(int x, int y, int xx, int yy, int p, int q) {
8         return max(
9             max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
10            max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
11            );
12    }
13    void init() {
14        FOR (x, 0, highbit(n) + 1)
15        FOR (y, 0, highbit(m) + 1)
16        FOR (i, 0, n - (1 << x) + 1)
17        FOR (j, 0, m - (1 << y) + 1) {
18            if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
19            f[i][j][x][y] = calc(
20                i, j,
21                i + (1 << x) - 1, j + (1 << y) - 1,
22                max(x - 1, 0), max(y - 1, 0)
23                );
24        }
25    }
26    inline int get_max(int x, int y, int xx, int yy) {
27        return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
28    }
29 };
```

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)

```
1  const LL INF = LONG_LONG_MAX;
2  #define maxn 10
3  LL n;
4
5  namespace SGT {
6      struct Q {
7          LL setv;
8          explicit Q(LL setv = -1): setv(setv) {}
9          void operator += (const Q& q) { if (q.setv != -1) setv = q.setv; }
10     };
11     struct P {
12         LL max;
13         explicit P(LL max = -INF): max(max) {}
14         void up(Q& q) { if (q.setv != -1) max = q.setv; }
15     };
16     template<typename T>
17     P operator & (T&& a, T&& b) {
18         return P(max(a.max, b.max));
19     }
20     P p[maxn << 2];
21     Q q[maxn << 2];
22 #define lson o * 2, l, (l + r) / 2
23 #define rson o * 2 + 1, (l + r) / 2 + 1, r
24     void up(int o, int l, int r) {
25         if (l == r) p[o] = P();
26         else p[o] = p[o * 2] & p[o * 2 + 1];
27         p[o].up(q[o]);
28     }
29     void down(int o, int l, int r) {
30         q[o * 2] += q[o]; q[o * 2 + 1] += q[o];
31         q[o] = Q();
32         up(lson); up(rson);
33     }
34     template<typename T>
35     void build(T&& f, int o = 1, int l = 1, int r = n) {
36         if (l == r) q[o] = f(l);
37         else { build(f, lson); build(f, rson); q[o] = Q(); }
38         up(o, l, r);
39     }
40     P query(int ql, int qr, int o = 1, int l = 1, int r = n) {
```

```
41          if (ql > r || l > qr) return P();
42          if (ql <= l && r <= qr) return p[o];
43          down(o, l, r);
44          return query(ql, qr, lson) & query(ql, qr, rson);
45      }
46      void update(int ql, int qr, const Q& v, int o = 1, int l = 1, int r = n) {
47          if (ql > r || l > qr) return;
48          if (ql <= l && r <= qr) q[o] += v;
49          else {
50              down(o, l, r);
51              update(ql, qr, v, lson); update(ql, qr, v, rson);
52          }
53          up(o, l, r);
54      }
55  }
56
57  // -------------------------------------------------------------------
58  void solve(){
59      vector<LL> arr = {1, 5, 7, 4, 2, 8 ,3, 6, 10, 9};
60      n = arr.size();
61      SGT::build([&](int idx){
62          return SGT::Q(arr[idx-1]);
63      });
64      for(LL i=1; i<=n; i++){
65          dbg(SGT::query(1, i).max);
66      }
67      SGT::update(2, 4, SGT::Q(-3));
68      cout << "MODIFIED\n";
69      for(LL i=1; i<=n; i++){
70          dbg(SGT::query(1, i).max);
71      }
72  }
```

● 区间修改，区间累加，查询区间和、最大值、最小值。

```
1   #define maxn 100005
2   #define INF LONG_LONG_MAX
3   LL a[maxn], n;
4
5   struct IntervalTree {
6   #define ls o * 2, l, m
7   #define rs o * 2 + 1, m + 1, r
8       static const LL M = maxn * 4, RS = 1E18 - 1;
9       LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
10      void init() {
11          memset(addv, 0, sizeof addv);
12          fill(setv, setv + M, RS);
13          memset(minv, 0, sizeof minv);
14          memset(maxv, 0, sizeof maxv);
15          memset(sumv, 0, sizeof sumv);
16      }
17      void maintain(LL o, LL l, LL r) {
18          if (l < r) {
19              LL lc = o * 2, rc = o * 2 + 1;
20              sumv[o] = sumv[lc] + sumv[rc];
21              minv[o] = min(minv[lc], minv[rc]);
22              maxv[o] = max(maxv[lc], maxv[rc]);
23          } else sumv[o] = minv[o] = maxv[o] = 0;
24          if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
25          if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
26      }
27      void build(LL o, LL l, LL r) {
28          if (l == r) addv[o] = a[l];
29          else {
30              LL m = (l + r) / 2;
31              build(ls); build(rs);
32          }
33          maintain(o, l, r);
34      }
35      void pushdown(LL o) {
36          LL lc = o * 2, rc = o * 2 + 1;
37          if (setv[o] != RS) {
```

```cpp
                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);

```

```
109    LL sum, min_val, max_val;
110    IT.range_query(1, 5, sum, min_val, max_val);
111    cout << " " << sum << " " << min_val << " " << max_val << endl;
112
113    IT.range_add(2, 4, 2);
114    IT.range_query(1, 5, sum, min_val, max_val);
115    cout << " " << sum << " " << min_val << " " << max_val << endl;
116
117    IT.range_set(3, 5, 10);
118    IT.range_query(1, 5, sum, min_val, max_val);
119    cout << " " << sum << " " << min_val << " " << max_val << endl;
120
121    IT.range_query(2, 4, sum, min_val, max_val);
122    cout << " " << sum << " " << min_val << " " << max_val << endl;
123 }
```

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

```
1  #define M 100005
2
3  namespace BIT {
4      LL c[M]; // 注意初始化开销
5      inline int lowbit(int x) { return x & -x; }
6      void add(int x, LL v) { // 单点加
7          for (int i = x; i < M; i += lowbit(i))
8              c[i] += v;
9      }
10     LL sum(int x) { // 前缀和
11         LL ret = 0;
12         for (int i = x; i > 0; i -= lowbit(i))
13             ret += c[i];
14         return ret;
15     }
16     int kth(LL k) { // 频次统计下从小到大第 k 个，详见应用
17         int p = 0;
18         for (int lim = 1 << 20; lim; lim /= 2)
19             if (p + lim < M && c[p + lim] < k) {
20                 p += lim;
21                 k -= c[p];
22             }
23         return p + 1;
24     }
25     LL sum(int l, int r) { return sum(r) - sum(l - 1); } // 区间和
26     // 区间加（此时树状数组为差分数组，sum(x) 为第 x 个数的值）
27     void add(int l, int r, LL v) { add(l, v); add(r + 1, -v); }
28 }
29 // -------------------------------------------------------------------
30 void solve(){
31     vector<LL> a={9, 9, 9, 9, 5, 3, 3, 3, 1, 1};
32     LL n = a.size(), i;
33     for(i=1; i<=n; i++) BIT::add(a[i-1], 1);
34     // 1 1 3 3 3 5 9 9 9 9
35     for(i=1; i<=n; i++) cout << BIT::kth(i) << ' ';
36 }
```

- 区间修改、区间查询

```
1  #define maxn 100005
2
3  namespace BIT {
4      int n;
5      int c[maxn], cc[maxn];
6      inline int lowbit(int x) { return x & -x; }
7      void init(int siz){ // 初始化
8          n = siz;
9          for(LL i=0; i<=n; i++){
10             c[i] = cc[i] = 0;
```

```cpp
        }
    }
    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

```cpp
#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

```
1   LL mul(LL a, LL b, LL m) {
2       LL ret = 0;
3       while (b) {
4           if (b & 1) {
5               ret += a;
6               if (ret >= m) ret -= m;
7           }
8           a += a;
9           if (a >= m) a -= m;
10          b >>= 1;
11      }
12      return ret;
13  }
```

- O(1)

```
1   LL mul(LL u, LL v, LL p) {
2       return (u * v - LL((long double) u * v / p) * p + p) % p;
3   }
4   LL mul(LL u, LL v, LL p) { // 卡常
5       LL t = u * v - LL((long double) u * v / p) * p;
6       return t < 0 ? t + p : t;
7   }
```

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）。

```
1   LL bin(LL x, LL n, LL MOD) {
2       LL ret = MOD != 1;
3       for (x %= MOD; n; n >>= 1, x = x * x % MOD)
4           if (n & 1) ret = ret * x % MOD;
5       return ret;
6   }
```

- 防爆 LL
- 前置模板：快速乘

```
1   LL bin(LL x, LL n, LL MOD) {
2       LL ret = MOD != 1;
3       for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
4           if (n & 1) ret = mul(ret, x, MOD);
5       return ret;
6   }
```

高精度

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

```
1   #include <algorithm>
2   #include <cstdio>
3   #include <string>
4   #include <vector>
5
6   struct BigIntTiny {
7       int sign;
8       std::vector<int> v;
9
10      BigIntTiny() : sign(1) {}
11      BigIntTiny(const std::string &s) { *this = s; }
12      BigIntTiny(int v) {
13          char buf[21];
14          sprintf(buf, "%d", v);
15          *this = buf;
16      }
```

```cpp
        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;
```

```
 88          if (absless(b)) return -(b - *this);
 89          BigIntTiny c;
 90          for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
 91              borrow += v[i] - b.get_pos(i);
 92              c.v.push_back(borrow);
 93              c.v.back() -= 10000 * (borrow >>= 31);
 94          }
 95          return c.setsign(sign, 0);
 96      }
 97      BigIntTiny operator*(const BigIntTiny &b) const {
 98          if (b < *this) return b * *this;
 99          BigIntTiny c, d = b;
100          for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
101              c.add_mul(d, v[i]);
102          return c.setsign(sign * b.sign, 0);
103      }
104      BigIntTiny operator/(const BigIntTiny &b) const {
105          BigIntTiny c, d;
106          BigIntTiny e=b;
107          e.sign=1;
108
109          d.v.resize(v.size());
110          double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
111                              (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
112          for (int i = (int)v.size() - 1; i >= 0; i--) {
113              c.v.insert(c.v.begin(), v[i]);
114              int m = (int)((c.get_pos((int)e.v.size()) * 10000 + c.get_pos((int)e.v.size() - 1)) * db);
115              c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
116              while (!(c < e))
117                  c = c - e, d.v[i] += 1;
118          }
119          return d.setsign(sign * b.sign, 0);
120      }
121      BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b * b; }
122      bool operator>(const BigIntTiny &b) const { return b < *this; }
123      bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
124      bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
125      bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
126  };
```

矩阵运算

```
 1  #define MOD 998244353
 2  #define M 10
 3
 4  struct Mat {
 5      LL m;
 6      LL v[M][M];
 7      Mat(int siz=2) {
 8          m = siz;
 9          for(int i=0; i<=m; i++){
10              for(int j=0; j<=m; j++){
11                  v[i][j] = 0;
12              }
13          }
14      }
15      void eye() { FOR (i, 0, m) v[i][i] = 1; }
16      LL* operator [] (LL x) { return v[x]; }
17      const LL* operator [] (LL x) const { return v[x]; }
18      Mat operator * (const Mat& B) {
19          const Mat& A = *this;
20          Mat ret;
21          FOR (k, 0, m)
22          FOR (i, 0, m) if (A[i][k])
23              FOR (j, 0, m)
24                  ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
25          return ret;
26      }
27      Mat pow(LL n) const {
28          Mat A = *this, ret; ret.eye();
29          for (; n; n >>= 1, A = A * A)
```

```
30              if (n & 1) ret = ret * A;
31          return ret;
32      }
33      Mat operator + (const Mat& B) {
34          const Mat& A = *this;
35          Mat ret;
36          FOR (i, 0, m)
37          FOR (j, 0, m)
38          ret[i][j] = (A[i][j] + B[i][j]) % MOD;
39          return ret;
40      }
41      void pprint() const {
42          FOR (i, 0, m)
43          FOR (j, 0, m)
44          printf("%lld%c", (*this)[i][j], j == m - 1 ? '\n' : ' ');
45      }
46  };
47  // -------------------------------------------------------------------
48  void solve(){
49      Mat mat1, mat2;
50      mat1.eye();
51      mat1[1][0] = 2; // 0-based
52      mat2.eye();
53      mat2[1][1] = 4;
54      Mat mat3 = mat1 * mat2;
55      mat3.pprint();
56  }
```

质数筛

- $\mathcal{O}(n)$

```
1   const LL p_max = 1E6 + 100;
2   LL pr[p_max], p_sz;
3   void get_prime() {
4       static bool vis[p_max];
5       FOR (i, 2, p_max) {
6           if (!vis[i]) pr[p_sz++] = i;
7           FOR (j, 0, p_sz) {
8               if (pr[j] * i >= p_max) break;
9               vis[pr[j] * i] = 1;
10              if (i % pr[j] == 0) break;
11          }
12      }
13  }
```

欧拉函数

朴素

```
1   int phi(int x)
2   {
3       int res = x;
4       for (int i = 2; i <= x / i; i ++ )
5           if (x % i == 0)
6           {
7               res = res / i * (i - 1);
8               while (x % i == 0) x /= i;
9           }
10      if (x > 1) res = res / x * (x - 1);
11
12      return res;
13  }
```

筛法求欧拉函数

- 前置模板: 质数筛

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

```
3    void get_phi() {
4        phi[1] = 1;
5        static bool vis[p_max];
6        static LL prime[p_max], p_sz, d;
7        FOR (i, 2, p_max) {
8            if (!vis[i]) {
9                prime[p_sz++] = i;
10               phi[i] = i - 1;
11           }
12           for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
13               vis[d] = 1;
14               if (i % prime[j] == 0) {
15                   phi[d] = phi[i] * prime[j];
16                   break;
17               }
18               else phi[d] = phi[i] * (prime[j] - 1);
19           }
20       }
21   }
```

素性测试

试除法

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

```
1    bool is_prime(int x)
2    {
3        if (x < 2) return false;
4        for (int i = 2; i <= x / i; i ++ )
5            if (x % i == 0)
6                return false;
7        return true;
8    }
```

Miller–Rabin

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

```
1    bool miller_rabin(LL n) {
2        static vector<LL> tester = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
3        if (n < 3 || n % 2 == 0) return n == 2;
4        if (n % 3 == 0) return n == 3;
5        LL u = n - 1, t = 0;
6        while (u % 2 == 0) u /= 2, ++t;
7        for (auto nt: tester) {
8            if(nt >= n) continue;
9            LL v = bin(nt, u, n);
10           if (v == 1) continue;
11           LL s;
12           for (s = 0; s < t; ++s) {
13               if (v == n - 1) break;
14               v = v * v % n;
15           }
16           if (s == t) return false;
17       }
18       return true;
19   }
```

质因数分解

朴素质因数分解

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

```
1   LL factor[30], f_sz, factor_exp[30];
2   void get_factor(LL x) {
3       f_sz = 0;
4       LL t = sqrt(x + 0.5);
5       for (LL i = 0; pr[i] <= t; ++i)
6           if (x % pr[i] == 0) {
7               factor_exp[f_sz] = 0;
8               while (x % pr[i] == 0) {
9                   x /= pr[i];
10                  ++factor_exp[f_sz];
11              }
12              factor[f_sz++] = pr[i];
13          }
14      if (x > 1) {
15          factor_exp[f_sz] = 1;
16          factor[f_sz++] = x;
17      }
18  }
```

- 不带指数

```
1   LL factor[30], f_sz;
2   void get_factor(LL x) {
3       f_sz = 0;
4       LL t = sqrt(x + 0.5);
5       for (LL i = 0; pr[i] <= t; ++i)
6           if (x % pr[i] == 0) {
7               factor[f_sz++] = pr[i];
8               while (x % pr[i] == 0) x /= pr[i];
9           }
10      if (x > 1) factor[f_sz++] = x;
11  }
```

Pollard-Rho

- 前置：素数测试

```
1   mt19937 mt(time(0));
2   LL pollard_rho(LL n, LL c) {
3       LL x = uniform_int_distribution<LL>(1, n - 1)(mt), y = x;
4       auto f = [&](LL v) { LL t = mul(v, v, n) + c; return t < n ? t : t - n; };
5       while (1) {
6           x = f(x); y = f(f(y));
7           if (x == y) return n;
8           LL d = gcd(abs(x - y), n);
9           if (d != 1) return d;
10      }
11  }
12
13  LL fac[100], fcnt;
14  void get_fac(LL n, LL cc = 19260817) {
15      if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
16      if (miller_rabin(n)) { fac[fcnt++] = n; return; }
17      LL p = n;
18      while (p == n) p = pollard_rho(n, --cc);
19      get_fac(p); get_fac(n / p);
20  }
21
22  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

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

```
5        FOR (j, 0, f_sz)
6            if (bin(i, (p - 1) / factor[j], p) == 1) {
7                flag = false;
8                break;
9            }
10           if (flag) return i;
11       }
12 //  assert(0);
13       return -1;
14 }
```

欧几里得

- 朴素

```
1  int gcd(int a, int b)
2  {
3      return b ? gcd(b, a % b) : a;
4  }
```

- 卡常

```
1  inline int ctz(LL x) { return __builtin_ctzll(x); }
2  LL gcd(LL a, LL b) {
3      if (!a) return b; if (!b) return a;
4      int t = ctz(a | b);
5      a >>= ctz(a);
6      do {
7          b >>= ctz(b);
8          if (a > b) swap(a, b);
9          b -= a;
10     } while (b);
11     return a << t;
12 }
```

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

```
1  LL ex_gcd(LL a, LL b, LL &x, LL &y) {
2      if (b == 0) { x = 1; y = 0; return a; }
3      LL ret = ex_gcd(b, a % b, y, x);
4      y -= a / b * x;
5      return ret;
6  }
```

中国剩余定理

- 求解线性同余方程

-

$$\begin{cases} x & \equiv r_1 \pmod{m_1} \\ x & \equiv r_2 \pmod{m_2} \\ & \vdots \\ x & \equiv r_k \pmod{m_k} \end{cases}$$

- 无解返回 -1

- 前置模板：扩展欧几里得

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

```
7        x = (r[i] - R) / d * x % (m[i] / d);
8        // 防爆 LL
9        // x = mul((r[i] - R) / d, x, m[i] / d);
10       R += x * M;
11       M = M / d * m[i];
12       R %= M;
13    }
14    return R >= 0 ? R : R + M;
15 }
```

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

```
1 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

```
1 LL get_inv(LL a, LL M) {
2     static LL x, y;
3     assert(exgcd(a, M, x, y) == 1);
4     return (x % M + M) % M;
5 }
```

- 预处理 1~n 的逆元

```
1 LL inv[N];
2 void inv_init(LL n, LL p) {
3     inv[1] = 1;
4     FOR (i, 2, n)
5         inv[i] = (p - p / i) * inv[p % i] % p;
6 }
```

- 预处理阶乘及其逆元

```
1 LL invf[M], fac[M] = {1};
2 void fac_inv_init(LL n, LL p) {
3     FOR (i, 1, n)
4         fac[i] = i * fac[i - 1] % p;
5     invf[n - 1] = bin(fac[n - 1], p - 2, p);
6     FORD (i, n - 2, -1)
7         invf[i] = invf[i + 1] * (i + 1) % p;
8 }
```

组合数

组合数预处理（递推法）

```
1 LL C[M][M];
2 void init_C(int n) {
3     FOR (i, 0, n) {
4         C[i][0] = C[i][i] = 1;
5         FOR (j, 1, i)
6             C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
7     }
8 }
```

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

```
1 inline LL C(LL n, LL m) { // n >= m >= 0
2     return n < m || m < 0 ? 0 : fac[n] * invf[m] % MOD * invf[n - m] % MOD;
3 }
```

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);`）

```
1  LL C(LL n, LL m) { // m >= n >= 0
2      if (m - n < n) n = m - n;
3      if (n < 0) return 0;
4      LL ret = 1;
5      FOR (i, 1, n + 1)
6          ret = ret * (m - n + i) % MOD * bin(i, MOD - 2, MOD) % MOD;
7      return ret;
8  }
```

- 前置模板可选 2：模数不固定下使用，无法单独使用。

```
1  LL Lucas(LL n, LL m) { // m >= n >= 0
2      return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD : 1;
3  }
```

求具体值

- 分解质因数法

```
1  int primes[N], cnt;      // 存储所有质数
2  int sum[N];       // 存储每个质数的次数
3  bool st[N];        // 存储每个数是否已被筛掉
4
5  void get_primes(int n)      // 线性筛法求素数
6  {
7      for (int i = 2; i <= n; i ++ )
8      {
9          if (!st[i]) primes[cnt ++ ] = i;
10         for (int j = 0; primes[j] <= n / i; j ++ )
11         {
12             st[primes[j] * i] = true;
13             if (i % primes[j] == 0) break;
14         }
15     }
16 }
17
18
19 int get(int n, int p)        // 求 n! 中的次数
20 {
21     int res = 0;
22     while (n)
23     {
24         res += n / p;
25         n /= p;
26     }
27     return res;
28 }
29
30
31 vector<int> mul(vector<int> a, int b)        // 高精度乘低精度模板
32 {
33     vector<int> c;
34     int t = 0;
35     for (int i = 0; i < a.size(); i ++ )
36     {
37         t += a[i] * b;
38         c.push_back(t % 10);
39         t /= 10;
40     }
41
42     while (t)
43     {
44         c.push_back(t % 10);
45         t /= 10;
46     }
47
48     return c;
```

```
49    }
50
51    get_primes(a);    // 预处理范围内的所有质数
52
53    for (int i = 0; i < cnt; i ++ )      // 求每个质因数的次数
54    {
55        int p = primes[i];
56        sum[i] = get(a, p) - get(b, p) - get(a - b, p);
57    }
58
59    vector<int> res;
60    res.push_back(1);
61
62    for (int i = 0; i < cnt; i ++ )      // 用高精度乘法将所有质因子相乘
63        for (int j = 0; j < sum[i]; j ++ )
64            res = mul(res, primes[i]);
```

FFT & NTT & FWT

FFT

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

```
1    typedef double LD;
2    const LD PI = acos(-1.0);
3
4    struct Complex {
5        LD r, i;
6        Complex(LD r = 0, LD i = 0) : r(r), i(i) {}
7        Complex operator + (const Complex& other) const {
8            return Complex(r + other.r, i + other.i);
9        }
10        Complex operator - (const Complex& other) const {
11            return Complex(r - other.r, i - other.i);
12        }
13        Complex operator * (const Complex& other) const {
14            return Complex(r * other.r - i * other.i, r * other.i + i * other.r);
15        }
16    };
17
18    // 快速傅里叶变换, p=1 为正向, p=-1 为反向
19    void FFT(vector<Complex>& x, int p) {
20        int n = x.size();
21        for (int i = 0, t = 0; i < n; ++i) {
22            if (i > t) swap(x[i], x[t]);
23            for (int j = n >> 1; (t ^= j) < j; j >>= 1);
24        }
25        for (int h = 2; h <= n; h <<= 1) {
26            Complex wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
27            for (int i = 0; i < n; i += h) {
28                Complex w(1, 0);
29                for (int j = 0; j < h / 2; ++j) {
30                    Complex u = x[i + j];
31                    Complex v = x[i + j + h/2] * w;
32                    x[i + j] = u + v;
33                    x[i + j + h/2] = u - v;
34                    w = w * wn;
35                }
36            }
37        }
38        if (p == -1) {
39            for (int i = 0; i < n; ++i) {
40                x[i].r /= n;
41            }
42        }
43    }
44
45    // 计算两个多项式的卷积，返回结果多项式的系数向量
46    vector<LD> convolution(const vector<LD>& a, const vector<LD>& b) {
47        int len = 1;
```

```
48      int n = a.size(), m = b.size();
49      while (len < n + m - 1) len <<= 1;
50      vector<Complex> fa(len), fb(len);
51      for (int i = 0; i < n; ++i) fa[i] = Complex(a[i], 0);
52      for (int i = 0; i < m; ++i) fb[i] = Complex(b[i], 0);
53      FFT(fa, 1);
54      FFT(fb, 1);
55      for (int i = 0; i < len; ++i) {
56          fa[i] = fa[i] * fb[i];
57      }
58      FFT(fa, -1);
59      vector<LD> res(n + m - 1);
60      for (int i = 0; i < n + m - 1; ++i) {
61          res[i] = fa[i].r;
62      }
63      return res;
64  }
```

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

```
1   const int N = 1e5+10;
2   const int MOD = 998244353; // 模数
3   const int G = 3; // 原根
4
5   LL wn[N << 2], rev[N << 2];
6   int NTT_init(int n_) {
7       int step = 0; int n = 1;
8       for ( ; n < n_; n <<= 1) ++step;
9       FOR (i, 1, n)
10      rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
11      int g = bin(G, (MOD - 1) / n, MOD);
12      wn[0] = 1;
13      for (int i = 1; i <= n; ++i)
14          wn[i] = wn[i - 1] * g % MOD;
15      return n;
16  }
17
18  void NTT(vector<LL> &a, int n, int f) {
19      FOR (i, 0, n) if (i < rev[i])
20          std::swap(a[i], a[rev[i]]);
21      for (int k = 1; k < n; k <<= 1) {
22          for (int i = 0; i < n; i += (k << 1)) {
23              int t = n / (k << 1);
24              FOR (j, 0, k) {
25                  LL w = f == 1 ? wn[t * j] : wn[n - t * j];
26                  LL x = a[i + j];
27                  LL y = a[i + j + k] * w % MOD;
28                  a[i + j] = (x + y) % MOD;
29                  a[i + j + k] = (x - y + MOD) % MOD;
30              }
31          }
32      }
33      if (f == -1) {
34          LL ninv = get_inv(n, MOD);
35          FOR (i, 0, n)
36          a[i] = a[i] * ninv % MOD;
37      }
38  }
39
40  vector<LL> conv(vector<LL> a, vector<LL> b){
41      int len_a = a.size(), len_b = b.size();
42      int len = len_a + len_b - 1;
43      int n = NTT_init(len);
44      a.resize(n);
45      b.resize(n);
46      NTT(a, n, 1);
47      NTT(b, n, 1);
48      vector<LL> c(n);
```

```
49        for (int i = 0; i < n; ++i) {
50            c[i] = a[i] * b[i] % MOD;
51        }
52        NTT(c, n, -1);
53        vector<LL> res(len);
54        for (int i = 0; i < len; ++i) {
55            res[i] = c[i];
56        }
57        return res;
58    }
```

FWT

```
1    const LL MOD = 998244353;
2
3    template<typename T>
4    void fwt(vector<LL> &a, int n, T f) {
5        for (int d = 1; d < n; d *= 2)
6            for (int i = 0, t = d * 2; i < n; i += t)
7                FOR (j, 0, d)
8                    f(a[i + j], a[i + j + d]);
9    }
10
11   void AND(LL& a, LL& b) { a += b; }
12   void OR(LL& a, LL& b) { b += a; }
13   void XOR (LL& a, LL& b) {
14       LL x = a, y = b;
15       a = (x + y) % MOD;
16       b = (x - y + MOD) % MOD;
17   }
18   void rAND(LL& a, LL& b) { a -= b; }
19   void rOR(LL& a, LL& b) { b -= a; }
20   void rXOR(LL& a, LL& b) {
21       static LL INV2 = (MOD + 1) / 2;
22       LL x = a, y = b;
23       a = (x + y) * INV2 % MOD;
24       b = (x - y + MOD) * INV2 % MOD;
25   }
26
27   int next_power_of_two(int n) {
28       if (n <= 0) return 1;
29       // __lg(n-1) 返回 n-1 的最高位所在位置 (0-based)
30       return 1 << (__lg(n - 1) + 1);
31   }
32
33   template<typename T, typename F>
34   vector<LL> conv(vector<LL> a, vector<LL> b, T f, F inv_f){
35       LL len_a = a.size(), len_b = b.size(), len = max(len_a, len_b), n = next_power_of_two(len);
36       a.resize(n), b.resize(n);
37       fwt(a, n, f), fwt(b, n, f);
38       vector<LL> c(n);
39       for (int i = 0; i < n; i++) {
40           c[i] = a[i] * b[i] % MOD;
41       }
42       fwt(c, n, inv_f);
43       // 提取结果 (可选)
44       c.resize(len);
45       return c;
46   }
```

线性基

贪心法

可查询最大异或和

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

```
5        for (int i = 63; ~i; --i) {
6            if (!(x >> i))   // x 的第 i 位是 0
7                continue;
8            if (!p[i]) {
9                p[i] = x;
10                break;
11            }
12            x ^= p[i];
13        }
14    }
15    ULL query_max(){
16        ULL ans = 0;
17        for (int i = 63; ~i; --i) {
18            ans = std::max(ans, ans ^ p[i]);
19        }
20        return ans;
21    }
22 };
```

高斯消元法

可查询任意大异或和

```
1  struct BasisGauss{
2      vector<ULL> a;
3      LL n, tmp, cnt;
4
5      BasisGauss(){a = {0};}
6
7      void insert(ULL x){
8          a.push_back(x);
9      }
10
11      void init(){
12          n = (LL)a.size() - 1;
13          LL k=1;
14          for(int i=63;i>=0;i--){
15              int t=0;
16              for(LL j=k;j<=n;j++){
17                  if((a[j]>>i)&1){
18                      t=j;
19                      break;
20                  }
21              }
22              if(t){
23                  swap(a[k],a[t]);
24                  for(LL j=1;j<=n;j++){
25                      if(j!=k&&(a[j]>>i)&1) a[j]^=a[k];
26                  }
27                  k++;
28              }
29          }
30          cnt = k-1;
31          tmp = 1LL << cnt;
32          if(cnt==n) tmp--;
33      }
34
35      LL query_xth(LL x){ // 从小到大, 若 x 为负数, 则查询倒数第几个
36          if(x<0) x = tmp + x + 1;
37          if(x>tmp) return -1;
38          else{
39              if(n>cnt) x--;
40              LL ans=0;
41              for(LL i=0; i<cnt; i++){
42                  if((x>>i)&1) ans^=a[cnt-i];
43              }
44              return ans;
45          }
46      }
47 };
```

图论

计算几何

字符串

杂项