Standard Code Library

Final Fantasy

Zhejiang University City College

November 6, 2021

Contents

一切的开始

CF 模板

- 需要 C++11

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

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;

#define dbg(x...) \
    do {          \
        cout << #x << " -> "; \
        err(x);   \
    } while (0)

void err() {
    cout << endl;
}

template<class T, class... Ts>
void err(T arg, Ts... args) {
    cout << arg << ' ';
    err(args...);
}

void read() {}

template<class T, class... Ts>
void read(T &x, Ts &... xs) {
    T f = 1;
    char ch;
    x = 0;
    for (ch = getchar(); ch < '0' || ch > '9'; ch = getchar()) {
        if (ch == '-') f = -1;
    }
    for (; ch >= '0' && ch <= '9'; ch = getchar()) x = x * 10 + ch - '0';
    x *= f;
    read(xs...);
}

mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
ll rng(ll l, ll r) {
    uniform_int_distribution<ll> uni(l, r);
    return uni(mt);
}

template <class T>
void myHash(T a[], int n, T w[] = nullptr) {
    set<T> st;
    for (int i = 0; i < n; i++) {
        st.insert(a[i]);
    }
    int tot = 0;
    map<T, int> mp;
    for (T x : st) {
        mp[x] = ++tot;
    }
    for (int i = 0; i < n; i++) {
        a[i] = mp[a[i]];
    }
    if (w != nullptr) {
        for (pair<T, int> p : mp) {
            w[p.second] = p.first;
        }
    }
}

```

```cpp
template<class T>
void unique(vector<T> &v) {
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
}

const int maxn = 1e5 + 7;
const int inf = 0x3f3f3f3f;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int mod = 1e9 + 7;

void run() {

}

/*

*/

int main() {

    int T = 1;
    read(T);
    while (T--) {
        run();
    }

    return 0;
}
```

STL

bitset

头文件

```
#include <bitset>
```

指定大小

```
bitset<1000> bs;  // a bitset with 1000 bits
```

构造函数

- `bitset()`：每一位都是 `false`。
- `bitset(unsigned long val)`：设为 `val` 的二进制形式。
- `bitset(const string& str)`：设为串 `str`。

运算符

- `operator []`：访问其特定的一位。
- `operator ==/!=`：比较两个 `bitset` 内容是否完全一样。
- `operator &/&=/|/| =/^/^=/~`：进行按位与/或/异或/取反操作。**`bitset` 只能与 `bitset` 进行位运算**，若要和整型进行位运算，要先将整型转换为 `bitset`。
- `operator <>/<<=/>>=`：进行二进制左移/右移。
- `operator <>`：流运算符，这意味着你可以通过 `cin/cout` 进行输入输出。

成员函数

- `count()`：返回 `true` 的数量。
- `size()`：返回 `bitset` 的大小。
- `test(pos)`：它和 `vector` 中的 `at()` 的作用是一样的，和 `[]` 运算符的区别就是越界检查。

- `any()`：若存在某一位是 `true` 则返回 `true`，否则返回 `false`。
- `none()`：若所有位都是 `false` 则返回 `true`，否则返回 `false`。
- `all()`：**C++11**，若所有位都是 `true` 则返回 `true`，否则返回 `false`。
- 1. `set()`：将整个 `bitset` 设置成 `true`。
  2. `set(pos, val = true)`：将某一位设置成 `true` / `false`。
- 1. `reset()`：将整个 `bitset` 设置成 `false`。
  2. `reset(pos)`：将某一位设置成 `false`。相当于 `set(pos, false)`。
- 1. `flip()`：翻转每一位。（相当于异或一个全是的 `bitset`）
  2. `flip(pos)`：翻转某一位。
- `to_string()`：返回转换成的字符串表达。
- `to_ulong()`：返回转换成的 `unsigned long` 表达（`long` 在 NT 及 32 位 POSIX 系统下与 `int` 一样，在 64 位 POSIX 下与 `long long` 一样）。
- `to_ullong()`：**C++11**，返回转换成的 `unsigned long long` 表达。

一些文档中没有的成员函数：

- `_Find_first()`：返回 `bitset` 第一个 `true` 的下标，若没有 `true` 则返回 `bitset` 的大小。
- `_Find_next(pos)`：返回 `pos` 后面（下标严格大于 `pos` 的位置）第一个 `true` 的下标，若 `pos` 后面没有 `true` 则返回 `bitset` 的大小。

vector

- 几乎在所有容器中，pop 和 $clear$ 只清除元素，不清除内存
- $vector$ 中释放内存的方法

```
vector<int>().swap(vec);
```

deque

- $deque$ 的内部实现是预先占用多个连续的存储块，添加元素并不需要重新分配空间
- 因此，向 $deque$ 两端添加/删除元素的时间复杂度很小，但是同时 $deque$ 初始化时便会占用较大的额外空间
- 非常不推荐同时使用多个 $deque$，如果一定要用（如多个单调队列），建议使用 $vector$ 加头尾指针模拟，使用 $vector.resize$ 初始化空间
- 以下写法在 hdu 一道 $528mb$ 的题中 MLE 了

```
deque<dqNode> dq[maxn]; //maxn = 1e6 + 5
```

一切的开始

重载哈希

```
// 重载哈希以支持 unordered_set & unordered_map
struct HashFunc {
    template<class T, class U>
    size_t operator() (const pair<T, U> &i) const {
        return hash<T>()(i.first) ^ hash<U>()(i.second);
    }
};
unordered_set<pll, HashFunc> st;
unordered_map<pll, int, Hash> mp;
```

随机

- 标准库的 `rand()` 函数范围较小，仅 $[0, 32768)$，容易被卡
- 因此取随机数建议用下面这个

1-gon 大爹的 gen

```cpp
mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
ll rng(ll l, ll r) {
    uniform_int_distribution<ll> uni(l, r);
    return uni(mt);
}
```

对拍相关

windows 下对拍.bat 文件

```
@echo off
:loop
    rand.exe > rand.in
    std.exe < rand.in > std.out
    my.exe < rand.in > my.out
    fc my.out std.out
if not errorlevel 1 goto loop
pause
goto loop
```

linux 下对拍.cpp 文件

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

using namespace std;

int main() {
    int i;
    for (i = 1; ; i++) {
        printf("The result of No. %d Case is: ", i);
        system("./rand > rand.in");
        system("./std < rand.in > std.out");
        system("./my < rand.in > my.out");
        if (system("diff std.out my.out")) {
            printf("Wrong Answer\n");
            printf("rand:\n");
            system("cat rand.in");
            return 0;
        } else printf("Accepted\n");
    }
}

/*
linux 下对拍
此文件为 duipai.cpp

其余组件:
对拍程序 my.cpp
标程 std.cpp
数据 rand.cpp
*/
```

rand.cpp

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

typedef long long ll;
using namespace std;

mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
ll rng(ll l, ll r) {
    uniform_int_distribution<ll> uni(l, r);
    return uni(mt);
}
```

```
//rng(l, r) 返回 [min, max] 区间值
```

int128 输入输出

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

using namespace std;

inline __int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}

inline void write(__int128 x) {
    if (x < 0) {
        x = -x;
        putchar('-');
    }
    if (x > 9) write(x / 10);
    putchar(x % 10 + '0');
}

/*
读入 x = read();
输出 write(x);
使用实例 a+b
*/

int main() {

    __int128 a = read();
    __int128 b = read();
    write(a + b);
    puts("");

    return 0;
}
```

分数模板

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

#define _for(i,a,b) for(int i = (a);i < (b);i ++)
const int maxn = 50003;

int gcd(int a,int b){if(b==0) return a;return gcd(b,a%b);}
int lcm(int a,int b){return a/gcd(a,b)*b;}

class Fraction
{
    public:
        int a,b;
        int sign(int x) {return (x>0?1:-1);}
        Fraction():a(0),b(1){}
        Fraction(int x):a(x),b(1){}
        Fraction(int x,int y)
        {
            int m = gcd(abs(x),abs(y));
```

```
            a = x/m*sign(y);
            if(a==0)b=1;else b = abs(y/m);
        }
        int get_denominator() {return b;}
        int get_numerator() {return a;}
        Fraction operator+(const Fraction &f)
        {
            int m = gcd(b,f.b);
            return Fraction(f.b/m*a+b/m*f.a,b/m*f.b);
        }
        Fraction operator-(const Fraction &f)
        {
            int m = gcd(b,f.b);
            return Fraction(f.b/m*a-b/m*f.a,b/m*f.b);
        }
        Fraction operator*(const Fraction &f)
        {
            int m1 = gcd(abs(a),f.b);
            int m2 = gcd(b,abs(f.a));
            return Fraction((a/m1)*(f.a/m2),(b/m2)*(f.b/m1));
        }
        Fraction operator/(const Fraction &f)
            {return (*this)*Fraction(f.b,f.a);}
        friend ostream &operator << (ostream &out,const Fraction &f)
        {
            if(f.a==0) cout << 0;
            else if(f.b==1) cout << f.a;
            else cout << f.a << '/' << f.b;
            return out;
        }
};

int main()
{
    Fraction p(18,10);
    Fraction o(11,17);
    cout << p*o << endl;
    return 0;
}
```

快读

简易版

```
namespace FastIO {
#define BUF_SIZE 1000000
    inline char nc() {
        static char buf[BUF_SIZE], *p1 = buf, *p2 = buf;
        return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, BUF_SIZE, stdin), p1 == p2) ? EOF : *p1++;
    }
    inline int read() {
        char ch = nc();
        int sum = 0;

        while (!(ch >= '0' && ch <= '9')) ch = nc();
        while (ch >= '0' && ch <= '9') {
            sum = sum * 10 + ch - '0';
            ch = nc();
        }
        return sum;
    }

#undef BUF_SIZE
}
```

全家桶

```
namespace FastIO{
    #define BUF_SIZE 100000
    #define OUT_SIZE 100000
```

```
#define ll long long
//fread->read

bool IOerror=0;
inline char nc(){
    static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
    if (p1==pend){
        p1=buf; pend=buf+fread(buf,1,BUF_SIZE,stdin);
        if (pend==p1){IOerror=1;return -1;}
        //{printf("IO error!\n");system("pause");for (;;);exit(0);}
    }
    return *p1++;
}
inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}
inline void read(int &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (sign)x=-x;
}
inline void read(ll &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (sign)x=-x;
}
inline void read(double &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (ch=='.'){
        double tmp=1; ch=nc();
        for (;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp*(ch-'0');
    }
    if (sign)x=-x;
}
inline void read(char *s){
    char ch=nc();
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    for (;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
    *s=0;
}
inline void read(char &c){
    for (c=nc();blank(c);c=nc());
    if (IOerror){c=-1;return;}
}
//fwrite->write
struct Ostream_fwrite{
    char *buf,*p1,*pend;
    Ostream_fwrite(){buf=new char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
    void out(char ch){
        if (p1==pend){
            fwrite(buf,1,BUF_SIZE,stdout);p1=buf;
        }
        *p1++=ch;
    }
    void print(int x){
        static char s[15],*s1;s1=s;
        if (!x)*s1++='0';if (x<0)out('-'),x=-x;
        while(x)*s1++=x%10+'0',x/=10;
        while(s1--!=s)out(*s1);
    }
    void println(int x){
        static char s[15],*s1;s1=s;
```

```
            if (!x)*s1++='0';if (x<0)out('-'),x=-x;
            while(x)*s1++=x%10+'0',x/=10;
            while(s1--!=s)out(*s1); out('\n');
        }
        void print(ll x){
            static char s[25],*s1;s1=s;
            if (!x)*s1++='0';if (x<0)out('-'),x=-x;
            while(x)*s1++=x%10+'0',x/=10;
            while(s1--!=s)out(*s1);
        }
        void println(ll x){
            static char s[25],*s1;s1=s;
            if (!x)*s1++='0';if (x<0)out('-'),x=-x;
            while(x)*s1++=x%10+'0',x/=10;
            while(s1--!=s)out(*s1); out('\n');
        }
        void print(double x,int y){
            static ll mul[]={1,10,100,1000,10000,100000,1000000,10000000,100000000,
                1000000000,10000000000LL,100000000000LL,1000000000000LL,10000000000000LL,
                100000000000000LL,1000000000000000LL,10000000000000000LL,100000000000000000LL};
            if (x<-1e-12)out('-'),x=-x;x*=mul[y];
            ll x1=(ll)floor(x); if (x-floor(x)>=0.5)++x1;
            ll x2=x1/mul[y],x3=x1-x2*mul[y]; print(x2);
            if (y>0){out('.'); for (size_t i=1;i<y&&x3*mul[i]<mul[y];out('0'),++i); print(x3);}
        }
        void println(double x,int y){print(x,y);out('\n');}
        void print(char *s){while (*s)out(*s++);}
        void println(char *s){while (*s)out(*s++);out('\n');}
        void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
        ~Ostream_fwrite(){flush();}
    }Ostream;
    inline void print(int x){Ostream.print(x);}
    inline void println(int x){Ostream.println(x);}
    inline void print_(int x){Ostream.print(x);Ostream.out(' ');}
    inline void print(char x){Ostream.out(x);}
    inline void println(char x){Ostream.out(x);Ostream.out('\n');}
    inline void print_(char x){Ostream.print(x);Ostream.out(' ');}
    inline void print(ll x){Ostream.print(x);}
    inline void println(ll x){Ostream.println(x);}
    inline void print_(ll x){Ostream.print(x);Ostream.out(' ');}
    inline void print(double x,int y){Ostream.print(x,y);}
    inline void println(double x,int y){Ostream.println(x,y);}
    inline void print_(double x,int y){Ostream.print(x,y);Ostream.out(' ');}
    inline void print(char *s){Ostream.print(s);}
    inline void println(char *s){Ostream.println(s);}
    inline void println(){Ostream.out('\n');}
    inline void flush(){Ostream.flush();}
    #undef ll
    #undef OUT_SIZE
    #undef BUF_SIZE
};
```

博弈

SG 函数

- 所有后继状态的 MEX

多个游戏组合

- 前提：多个公平游戏且游戏之间相互独立
- 结论：每个游戏的 SG 值异或和为 0 则先手必败，反之先手必胜

阶梯 NIM

题面

N 堆石子，两人轮流操作，一次操作为挑选一堆石子 i，将至少 1 个石子移动至 $i-1$ 位置（$i=1$ 则被移出游戏），不能操作者输。

结论

相当于对所有奇数位置上的石子堆做 NIM 游戏。即 $a_1⊠a_3⊠...=0 $，则先手必败

斐波那契博弈

题面

有一堆个数为 $n(n \geq 2)$ 的石子，游戏双方轮流取石子，规则如下

- 先手不能在第一次把所有石子取完，至少取 1 颗
- 之后每次取石子数范围 $[1, 2 * a_{i-1}]$，a_{i-1} 表示对手上一轮取石子的数量
- 取走最后一个石子的为赢家

结论

当 n 为 $Fibonacci$ 数时，先手必败。

威佐夫博弈

题面

有两堆石子，两人分先后手按最优策略取石子，每次可以从任意一堆石子中取任意多的石子或者从两堆石子中取同样多的石子，不能取的人输，给出两堆石子的数量 a，b，分析胜负情况。

$Beatty$ 定理

设 x, y 是正无理数且 $1/x + 1/y = 1$。记 P={ $\lfloor ix \rfloor$ | i 为正整数}, Q={ $\lfloor iy \rfloor$ | i 为正整数}, 则 P 与 Q 是 N+ 的一个划分, 即 $P \cap Q = \emptyset$ 且 $P \cup Q = N+$（正整数集）。

$Beatty$ 定理与威佐夫博弈的关联

假设 $x < y$ 且 P, Q 内数均按大小排序，(P_i, Q_i) 构成第 i 种先手必败态（此处证略）。在朴素威佐夫博弈中，对于所有必败态势，P_i 是当前最小的没有使用过的数，$Q_i = P_i + i$，即 $\lfloor iy \rfloor = \lfloor ix \rfloor + i$，因为对任意正整数 i 均成立，可得 $y = x + 1$。因此解方程 $1/x + 1/(x+1) = 1$ 可得 $x = ((1 + \sqrt{5})/2)$，因此有必败态通项 $(\lfloor m(1 + \sqrt{5})/2 \rfloor, \lfloor m(3 + \sqrt{5})/2 \rfloor)$。

结论

假设 $a < b$，当且仅当 $(b - a) * (\sqrt{5} + 1)/2 = a$ 时先手必败

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    ll a, b;
    scanf("%lld%lld", &a, &b);
    if(a > b) swap(a, b);
    ll tmp = abs(a - b);
    ll ans = tmp * (1.0 + sqrt(5.0)) / 2.0;
    printf("%d\n", ans == a ? 0 : 1);
    return 0;
}
```

威佐夫博弈变种 (HDU 6869)

题面变化

从两堆石子取同样多的石子改为从两堆石子中分别取 x，y 个石子，$|x - y| <= k$，k 由题目给出。

Sol

设 (P_i, Q_i) 构成先手必败态，打表可得 P_i 是当前最小的没有使用过的数，$Q_i = P_i + (k+1)i$，套入 $Beatty$ 定理，有 $\lfloor iy \rfloor = \lfloor ix \rfloor + (k+1)i$,因为对任意正整数 i 均成立,可得 $y = x+k+1$,解方程 $1/x+1/(x+k+1) = 1$ 可得 $x = (1-k+\sqrt{k^2+2k+5})/2$,可求得结论中通项。

结论

所有的先手必败态为 $(\lfloor m(1-k+\sqrt{k^2+2k+5})/2 \rfloor, \lfloor m(3+k+\sqrt{k^2+2k+5})/2 \rfloor)$，$k=0$ 即朴素威佐夫博弈。

Code

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

typedef long long ll;

int main() {

    int T;
    scanf("%d", &T);
    while (T--) {
        int a, b; double k;
        scanf("%d%d%lf", &a, &b, &k);
        double r1 = 1 - k + sqrt((k + 1) * (k + 1) + 4);
        double r2 = 3 + k + sqrt((k + 1) * (k + 1) + 4);
        if (a > b) swap(a, b);
        int m = (int) ceil(a * 2 / r1);
        int m2 = (int) ceil((a + 1) * 2 / r1);
        if (m == m2) {
            printf("1\n");
            continue;
        }
        assert(m + 1 == m2);
        if (b == (ll) floor(m * r2 / 2)) {
            printf("0\n");
        } else {
            printf("1\n");
        }
    }

    return 0;
}
```

动态规划

数位 dp

```cpp
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同

ll dfs(int pos, /*state*/, bool lead, bool limit) {
    if(pos == 0) return 1;/* 是否返回 1 根据题意 */
    if(!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
    int up = limit ? a[pos] : 9;
    ll ans = 0;
    for(int i = 0; i <= up; i++) {
        if() ...
        else if()...
        ans += dfs(pos-1, /* 状态转移 */, lead && i==0, limit && i==a[pos]) //最后两个变量传参都是这样写的
    }
    if(!limit && !lead) dp[pos][state] = ans;
    return ans;
}

ll solve(ll x) {
    int pos = 0;
```

```
    while(x) {
        a[++pos] = x % 10;
        x /= 10;
    }
    return dfs(pos, /*state*/, true, true);//开始枚举时最高位有限制且有前导零
}

int main() {
    ll le, ri;
    while(~scanf("%lld%lld", &le, &ri)) {
        //初始化 dp 数组为-1
        printf("%lld\n", solve(ri) - solve(le - 1));
    }
}
```

数论

线性筛

```
//线性筛求素数与欧拉函数
const int MAXN = 5000005;

int phi[MAXN];
bool isPrime[MAXN];
int pri[MAXN]; //最小质因子

void getphi(int n, vector<int> &prime) {
    prime.clear();
    phi[1] = 1;
    for (int i = 0; i <= n; i++) {
        isPrime[i] = true;
    }

    for (int i = 2; i <= n; i++) {
        if (isPrime[i]) {
            prime.push_back(i);
            phi[i] = i - 1;
        }

        for (int v: prime) {
            if (i * v > n) break;
            isPrime[i * v] = false;
            pri[i * v] = v;
            if (i % v == 0) {
                phi[i * v] = phi[i] * v;
                break;
            } else {
                phi[i * v] = phi[i] * (v - 1);  //v - 1 == phi[v]
            }
        }
    }
}

/*
上限 n, prime 中存储素数
*/
```

欧拉降幂

$$x^k \% mod = x^{k \% \phi(mod) + \phi(mod)} \% mod \quad k \geq \phi(mod)$$

```
//欧拉降幂
#include <bits/stdc++.h>

typedef long long ll;
using namespace std;
const int maxn = 1000005;
const int inf = 0x3f3f3f3f;

```

```cpp
unordered_map<ll, ll> mp;

ll MOD(ll x, ll mod) {return x < mod ? x : x % mod + mod;}

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = MOD(res * a, mod);

        b /= 2;
        a = MOD(a * a, mod);
    }
    return res;
}

ll phi(ll x) {
    if (mp[x]) return mp[x];

    ll res = x;
    for (ll i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            res -= res / i;
            while (x % i == 0) x /= i;
        }
    }
    if (x > 1) {
        res -= res / x;
    }
    return mp[x] = res;
}

ll a[maxn];

ll solve(int l, int r, ll p) {
    if (p == 1) return MOD(a[l], p);
    if (l == r) return MOD(a[l], p);

    return qpow(a[l], solve(l + 1, r, phi(p)), p);
}

int main() {

    int n; ll p; scanf("%d%lld", &n, &p);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", a + i);
    }

    int q; scanf("%d", &q);
    while (q--) {
        int l, r; scanf("%d%d", &l, &r);
        printf("%lld\n", solve(l, r, p) % p);
    }

    return 0;
}
```

矩阵快速幂

```cpp
/*
需要取模时记得 mod
* 乘法
^ 快速幂
*/
typedef long long ll;
const int mod = 1e9 + 7;

struct Matrix {
#define N 3
    ll w[N][N];

    Matrix() {
```

```cpp
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                w[i][j] = 0;
            }
        }
    }
    Matrix(int _w[][N]) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                w[i][j] = _w[i][j];
            }
        }
    }

    Matrix operator * (Matrix m) {
        Matrix ret;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++) {
                    ret.w[i][j] += w[i][k] * m.w[k][j] % mod;
                    ret.w[i][j] %= mod;
                }
            }
        }
        return ret;
    }

    Matrix operator ^ (int y) {
        Matrix ret, x(w);
        for (int i = 0; i < N; i++) {
            ret.w[i][i] = 1;
        }
        while (y) {
            if (y & 1) {
                ret = ret * x;
            }
            y /= 2;
            x = x * x;
        }
        return ret;
    }
#undef N
};
```

Lucas 定理

对于质数 p，有

$$\binom{n}{m}\%p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \times \binom{n\%p}{m\%p}\%p$$

$\binom{n\%p}{m\%p}$ 直接求解，$\binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor}$ 递归 $Lucas$，当 $m=0$ 时，返回 1

时间复杂度 $O(f+g\times log_p n)$，f 为预处理组合数，g 为单次求组合数

```cpp
// 若 n<m, C(n,m,p)=0
typedef long long ll;

ll lucas(ll n, ll m, int p) {
    if (m == 0) return 1;
    return lucas(n / p, m / p, p) * C(n % p, m % p, p) % p;
}
```

差分推 x+y 组合数方案

```cpp
//差分推 x+y 组合数方案

#include<bits/stdc++.h>

typedef long long ll;
using namespace std;
```

```cpp
const int maxn = 2000005;
const int inf = 0x3f3f3f3f;

/*
A <= x <= B
C <= y <= D
s[i] 表示 x+y=i 的方案数
*/

int s[maxn];

int main() {

    int A, B, C, D;
    A = ; B = ;
    C = ; D = ;

    s[A + C]++;
    s[A + D + 1]--;
    s[B + C + 1]--;
    s[B + D + 2]++;

    for (int i = 1; i < maxn; i++) s[i] += s[i - 1];
    for (int i = 1; i < maxn; i++) s[i] += s[i - 1];

    for (int i = A + C; i <= B + D + 2; i++) {
        printf("s[%d] = %d\n", i, s[i]);
    }

    return 0;
}
```

Miller-Rabin & Pollard_rho（大数质因子分解）

```cpp
//Miller_Rabin & Pollard_rho 大数质因子分解 by @kuangbin
/*
适用范围 n < 2^63
初始化 factor.clear()
S  随机算法判定次数, S 越大, 判错概率越小
Miller_Rabin(n)  n 为素数-true(极小概率伪素数), 合数-false
check(a, n, x, t)  n 一定为合数-true, 不一定为合数-false
factor 存分解出的质因子

注意点:
-factor 中的质因子为乱序
-当 n+n 会爆 ll 时, 请使用 ull 或者 _int128
-特判 n == 1
*/
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int S = 20;

ll mult_mod(ll a, ll b, ll m) {
    a %= m; b %= m;
    ll ans = 0;
    while (b) {
        if (b & 1) {
            ans += a;
            ans %= m;
        }
        a <<= 1; b >>= 1;
        if (a >= m) a %= m;
    }
    return ans;
}

ll qpow(ll a, ll b, ll m) {
    a %= m;
```

```cpp
    ll ans = 1;
    while (b) {
        if (b & 1) ans = mult_mod(ans, a, m);
        a = mult_mod(a, a, m);
        b >>= 1;
    }
    return ans;
}

bool check(ll a, ll n, ll x, ll t) {
    ll ans = qpow(a, x, n);
    ll last = ans;
    for (int i = 1; i <= t; i++) {
        ans = mult_mod(ans, ans, n);
        if (ans == 1 && last != 1 && last != n - 1) return true;
        last = ans;
    }
    if (ans != 1) return true;
    return false;
}

bool Miller_Rabin(ll n) {
    if (n < 2)return false;
    if (n == 2)return true;
    if ((n & 1) == 0) return false;
    ll x = n - 1;
    ll t = 0;
    while ((x & 1) == 0) {
        x >>= 1;
        t++;
    }
    for (int i = 0; i < S; i++) {
        ll a = rand() % (n - 1) + 1;//rand() 需要 stdlib.h 头文件
        if (check(a, n, x, t)) {
            return false;
        }
    }
    return true;
}

vector<ll> factor;

ll gcd(ll a, ll b) {
    if (a == 0)return 1;
    if (a < 0) return gcd(-a, b);
    while (b) {
        ll t = a % b;
        a = b;
        b = t;
    }
    return a;
}

ll Pollard_rho(ll x, ll c) {
    ll i = 1, k = 2;
    ll x0 = rand() % x;
    ll y = x0;
    while (true) {
        i++;
        x0 = (mult_mod(x0, x0, x) + c) % x;
        ll d = gcd(y - x0, x);
        if (d != 1 && d != x) return d;
        if (y == x0) return x;
        if (i == k) {
            y = x0;
            k += k;
        }
    }
}

void findFac(ll n) {
```

```
    if (Miller_Rabin(n)) {
        factor.push_back(n);
        return;
    }
    ll p = n;
    while (p >= n) {
        p = Pollard_rho(p, rand() % (n - 1) + 1);
    }
    findFac(p);
    findFac(n / p);
}

int main() {
    srand(time(NULL));//需要 time.h 头文件//POJ 上 G++ 不能加这句话
    ll n;
    while (scanf("%lld", &n) != EOF) {
        if (n == 1) continue;
        factor.clear();
        findFac(n);
        sort(factor.begin(), factor.end());
    }
    return 0;
}
```

圆排列

- 从 n 个不同元素中，选出 m 个不重复元素，组成的不同圆排列有 $n!/(n-m)!/m$
- 每个位置有 n 种不同元素可选，且各个位置的元素可重复，组成的不同 m 圆排列有 $(\sum_{i=1}^{m} n*gcd(i,m))/m$

逆元

- 求单个：$inv[i]=i^{mod-2}\%mod$
- $O(n)$ 逆元表：$inv[i]=(mod-mod/i)*inv[mod\%i]\%mod$

组合数递推式

$C_{n+1}^{m+1}=C_n^m+C_n^{m+1}$

卡特兰数

C[0]=1, C[1]=1, C[2]=2

序列：1,1,2,5,14,42,132,429,1430,4832... 公式 $C[n]=(2n)!/(n!*(n+1)!)$

斐波那契

$F_{i-1}F_k+F_iF_{k+1}=F_{i+k}$

转移矩阵

$$\begin{bmatrix} a & b \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} b & a+b \end{bmatrix}$$

扩展欧几里得 (Ex GCD)

对于正整数 a, b, 若 $gcd(a,b)=1$, 则对于任意整数 s, 方程 $s=a*x+b*y$ 必定存在整数解, 且若 $s>a*b$, 必定存在非负整数解。

调和级数与欧拉常数

$\sum_{i=1}^{n} \frac{1}{i} = euler + ln(n)$

$euler = 0.57721566490153286060651209$

杂项结论

子序列循环

假设有一序列 $a_1, a_2, a_3 ... a_n$, 每次操作为选择任意长度的子序列 $a_1, a_2 .. a_m$(不连续), 变为 $a_m, a_1, a_2 ... a_{m-1}$, 代价为 m ， 另序列变为目标序列的最小代价为 $n - k$, k 为初始序列与目标序列下标相同的相同元素数量。

因子个数

有 N 个数，其中最大的数为 A，所有数的因子种数总和约为 $NlogA$

勾股数构造

若需要一组最小数为奇数的勾股数，可任意选取一个 3 或以上的奇数，将该数自乘为平方数，除以 2，答案加减 0.5 可得到两个新的数字，这两个数字连同一开始选取的奇数例如（5，12，13）# 莫队

普通莫队

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

using namespace std;
typedef long long ll;
const int inf = 0x3f3f3f3f;
const int maxn = 50005;

struct node {
    int l, r, id, part;

    bool operator < (const node &tmp) const {
        if (part == tmp.part) {
            return part & 1 ? r < tmp.r : r > tmp.r; //奇偶优化
        }
        return part < tmp.part;
    }
} p[maxn];

int a[maxn];
ll ans[maxn], res;

void update(int pos, int add) {
    //随题意
}

int main() {

    int n, m, k; scanf("%d%d%d", &n, &m, &k);
    for (int i = 1; i <= n; i++) scanf("%d", a + i);

    int block = sqrt(n);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &p[i].l, &p[i].r);
        p[i].id = i;
        p[i].part = (p[i].l - 1) / block + 1;
    }
    sort(p + 1, p + m + 1);

    int l = 1, r = 1;
    res = 0;    //也可能不是 0
    for (int i = 1; i <= m; i++) {
        while (l > p[i].l) update(--l, 1);
        while (r < p[i].r) update(++r, 1);
```

```cpp
        while (l < p[i].l) update(l++, -1);
        while (r > p[i].r) update(r--, -1);
        ans[p[i].id] = res;
    }

    for (int i = 1; i <= m; i++) {
        printf("%lld\n", ans[i]);
    }

    return 0;
}
```

带修改莫队

- 询问为 (l, r, t)，其中 t 为修改次数
- 先按 l 分块，再按 r 分块，最后按 t 排序，N 为序列长度，M 为总修改次数，推荐块长有 $\sqrt{N}$，$N^{\frac{2}{3}}$，$\sqrt[3]{N \times M}$。
- 块长为 $\sqrt[3]{N \times M}$ 时，理论复杂度最优，为 $N^{\frac{4}{3}} \times M^{\frac{1}{3}}$，另外 $(10^5)^{\frac{1}{3}} \approx 46$

洛谷 P1903

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

typedef pair<int, int> pi;
typedef long long ll;

const int maxn = 140005;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-7;

struct Query {
    int l, r, lp, rp, id, t;
    bool operator < (const Query &tmp) const {
        if (lp != tmp.lp) return lp < tmp.lp;
        if (rp != tmp.rp) return rp < tmp.rp;
        return t < tmp.t;
    }
};
struct Update {
    int i, old, now;
};
int a[maxn];
int ans[maxn];
int cnt[1000006];
int res;

void add_col(int c) {
    if (cnt[c] == 0) res++;
    cnt[c]++;
}
void del_col(int c) {
    if (cnt[c] == 1) res--;
    cnt[c]--;
}
void add(int i) {
    add_col(a[i]);
}
void del(int i) {
    del_col(a[i]);
}
void add_upd(Update upd, int l, int r) {
    if (upd.i >= l && upd.i <= r) {
        del_col(upd.old);
        add_col(upd.now);
    }
    assert(a[upd.i] == upd.old);
    a[upd.i] = upd.now;
```

```cpp
}
void del_upd(Update upd, int l, int r) {
    if (upd.i >= l && upd.i <= r) {
        del_col(upd.now);
        add_col(upd.old);
    }
    assert(a[upd.i] == upd.now);
    a[upd.i] = upd.old;
}

void run() {
    int n, q;
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++) {
        scanf("%d", a + i);
    }
    vector<Query> query;
    vector<Update> update;
    while (q--) {
        char op[5];
        int x, y;
        scanf("%s%d%d", op, &x, &y);
        if (op[0] == 'Q') {
            Query qi{x, y};
            qi.id = query.size();
            qi.t = update.size();
            query.push_back(qi);
        } else {
            update.push_back({x, a[x], y});
            a[x] = y;
        }
    }
    int block = pow(n, 0.333) * max(1.0, pow(update.size(), 0.333));
//    int block = ceil(exp((log(n) + max(1.0, log(update.size()))) / 3));
//    int block = pow(n, 0.666);
//    int block = sqrt(n);
    for (Query &qi : query) {
        qi.lp = qi.l / block;
        qi.rp = qi.r / block;
    }
    sort(query.begin(), query.end());
    int cur_t = update.size();
    int l = 1, r = 1;
    cnt[a[1]]++; res = 1;
    for (Query qi : query) {
        while (l > qi.l) add(--l);
        while (r < qi.r) add(++r);
        while (l < qi.l) del(l++);
        while (r > qi.r) del(r--);
        while (cur_t > qi.t) del_upd(update[--cur_t], l, r);
        while (cur_t < qi.t) add_upd(update[cur_t++], l, r);
        ans[qi.id] = res;
    }

    for (int i = 0; i < query.size(); i++) {
        printf("%d\n", ans[i]);
    }
}

int main() {

    int T = 1;
//    scanf("%d", &T);
    while (T--) {
        run();
    }

    return 0;
}
```

树上莫队

用欧拉序将树转化成线性结构

每次询问路径 (x, y)，假设 $L[x] < L[y]$

- 若 $lca(x, y) == x$，统计区间 $L[x]$ 到 $L[y]$ 中只出现过**一次**的点
- 若 $lca(x, y) \neq x$，统计区间 $R[x]$ 到 $L[y]$ 中只出现过**一次**的点，外加 $lca(x, y)$ 本身

SPOJ COT2

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

typedef pair<int, int> pi;
typedef long long ll;

const int maxn = 40004;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-7;

int col[maxn];
vector<int> G[maxn];
int d[maxn], fa[maxn];

int ola[maxn * 2], L[maxn], R[maxn], tot;

void dfs(int u, int depth) {
    ola[++tot] = u;
    L[u] = tot;
    d[u] = depth;
    for (int v : G[u]) {
        if (v == fa[u]) continue;
        fa[v] = u;
        dfs(v, depth + 1);
    }
    ola[++tot] = u;
    R[u] = tot;
}

int st[maxn][20];

void initST(int n) {
    st[0][0] = -1;
    for (int i = 1; i <= n; i++) st[i][0] = fa[i];
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; i <= n; i++) {
            if (st[i][j - 1] == -1) st[i][j] = -1;
            else st[i][j] = st[st[i][j - 1]][j - 1];
        }
    }
}

int LCA(int u, int v) {
    if (d[u] < d[v]) swap(u, v);
    for (int i = 0; d[u] != d[v]; i++) {
        if ((d[u] - d[v]) >> i & 1) {
            u = st[u][i];
        }
    }
    if (u == v) return u;

    for (int i = 19; i >= 0; i--) {
        if (st[u][i] != -1 && st[v][i] != -1 && st[u][i] != st[v][i]) {
            u = st[u][i];
            v = st[v][i];
        }
    }
    return st[u][0];
}
```

```
int block;

struct Query {
    int l, r, part, id, lca;

    Query() {}
    Query(int u, int v, int _id) {
        id = _id;
        if (L[u] > L[v]) swap(u, v);
        int _lca = LCA(u, v);
        if (_lca == u) {
            l = L[u];
            r = L[v];
            part = (l - 1) / block + 1;
            lca = 0;
        } else {
            l = R[u];
            r = L[v];
            part = (l - 1) / block + 1;
            lca = _lca;
        }
    }

    bool operator < (const Query &tmp) const {
        if (part == tmp.part) {
            return part & 1 ? r < tmp.r : r > tmp.r;
        } else {
            return part < tmp.part;
        }
    }
} query[100005];
int cnt[maxn], cnt_col[maxn];
int cur;

void add_col(int c) {
    if (cnt_col[c] == 0) {
        cur++;
    }
    cnt_col[c]++;
}

void del_col(int c) {
    if (cnt_col[c] == 1) {
        cur--;
    }
    cnt_col[c]--;
}

void add(int i) {
    int u = ola[i];
    cnt[u]++;
    if (cnt[u] == 1) {
        add_col(col[u]);
    } else if (cnt[u] == 2) {
        del_col(col[u]);
    }
}

void del(int i) {
    int u = ola[i];
    cnt[u]--;
    if (cnt[u] == 1) {
        add_col(col[u]);
    } else if (cnt[u] == 0) {
        del_col(col[u]);
    }
}

int ans[100005];
```

```cpp
void run() {
    int n, q;
    scanf("%d%d", &n, &q);
    set<int> s;
    map<int, int> mp;
    for (int i = 1; i <= n; i++) {
        scanf("%d", col + i);
        s.insert(col[i]);
    }
    int rak = 0;
    for (int i : s) {
        mp[i] = ++rak;
    }
    for (int i = 1; i <= n; i++) {
        col[i] = mp[col[i]];
    }
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    fa[1] = -1;
    dfs(1, 0);
    initST(n);

    block = sqrt(n * 2);
    for (int i = 1; i <= q; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        query[i] = Query(u, v, i);
    }
    sort(query + 1, query + 1 + q);

    int l = 1, r = 1;
    cnt[ola[1]]++;
    cnt_col[col[ola[1]]]++;
    cur = 1;
    for (int i = 1; i <= q; i++) {
        while (l > query[i].l) add(--l);
        while (r < query[i].r) add(++r);
        while (l < query[i].l) del(l++);
        while (r > query[i].r) del(r--);
        int tmp = 0;
        if (query[i].lca && cnt_col[col[query[i].lca]] == 0) {
            tmp = 1;
        }
        ans[query[i].id] = cur + tmp;
    }
    for (int i = 1; i <= q; i++) {
        printf("%d\n", ans[i]);
    }
}

int main() {

    int T = 1;
//    scanf("%d", &T);
    while (T--) {
        run();
    }

    return 0;
}
```

数据结构

树状数组

```
struct BIT {
    ll c[maxn];
    int n;

    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; i++) {
            c[i] = 0;
        }
    }
    int lowbit(int x) {
        return x & (-x);
    }
    void update(int i, int v) {
        while (i <= n) {
            c[i] += v;
            i += lowbit(i);
        }
    }
    ll getsum(int i) {
        ll ans = 0;
        while (i) {
            ans += c[i];
            i -= lowbit(i);
        }
        return ans;
    }
};
```

线段树

```
/*
区间乘、区间加，维护区间和

线段树题 考虑 3 个点
- mergeInfo info 合并，pushup 和 query 时使用
- Tag.mergeTag tag 向下合并，pushdown 时使用
- Info.applyTag tag 更新 info，pushdown 时使用
3 个点解决则题目解决
*/
const int maxn = 100005;
const int mod = 1e9 + 7;

int a[maxn];

struct Tag {
    ll mul, add;
    void clear() {
        mul = 1;
        add = 0;
    }
    void mergeTag(Tag k) {
        mul = mul * k.mul % mod;
        add = (add * k.mul % mod + k.add) % mod;
    }
};

struct Info {
    ll sum;
    int l, r;
    void applyTag(Tag k) {
        sum = sum * k.mul % mod;
        sum += k.add * (r - l + 1) % mod;
        sum %= mod;
    }
};
```

```cpp
struct SegTree {
    Info info[maxn << 2];
    Tag tag[maxn << 2];
    Info mergeInfo(Info x, Info y) {
        if (x.l == 0) return y;
        if (y.l == 0) return x;
        return Info{(x.sum + y.sum) % mod, x.l, y.r};
    }

    void applyTag(int x, Tag k) {
        tag[x].mergeTag(k);
        info[x].applyTag(k);
    }

    void pushup(int x) {
        info[x] = mergeInfo(info[x * 2], info[x * 2 + 1]);
    }

    void pushdown(int x) {
        applyTag(x * 2, tag[x]);
        applyTag(x * 2 + 1, tag[x]);
        tag[x].clear();
    }

    void build(int x, int l, int r) {
        tag[x].clear();
        if (l == r) {
            info[x].sum = a[l];
            info[x].l = info[x].r = l;
        } else {
            int mid = l + r >> 1;
            build(x * 2, l, mid);
            build(x * 2 + 1, mid + 1, r);
            pushup(x);
        }
    }

    void update(int x, int l, int r, int ql, int qr, Tag k) {
        if (ql > r || qr < l) return;
        if (ql <= l && r <= qr) {
            applyTag(x, k);
            return;
        }
        pushdown(x);
        int mid = l + r >> 1;
        update(x * 2, l, mid, ql, qr, k);
        update(x * 2 + 1, mid + 1, r, ql, qr, k);
        pushup(x);
    }

    Info query(int x, int l, int r, int ql, int qr) {
        if (ql > r || qr < l) return Info{0, 0, 0};
        if (ql <= l && r <= qr) {
            return info[x];
        }
        pushdown(x);
        int mid = l + r >> 1;
        return mergeInfo(query(x * 2, l, mid, ql, qr), query(x * 2 + 1, mid + 1, r, ql, qr));
    }
}
```

主席树

```cpp
//主席树求静态区间第 k 小
struct PersistentSegTreeNode {
    int w;
    int ls, rs;
};

struct PersistentSegTree {
```

```
    PersistentSegTreeNode info[maxn * 25];
    int tot;

    void init() {
        tot = 0;
    }

    void pushup(int x) {
        int ls = info[x].ls;
        int rs = info[x].rs;
        info[x].w = info[ls].w + info[rs].w;
    }

    int build(int l, int r) {
        int x = ++tot;
        if (l == r) {
            info[x].w = 0;
        } else {
            int mid = l + r >> 1;
            info[x].ls = build(l, mid);
            info[x].rs = build(mid + 1, r);
            pushup(x);
        }
        return x;
    }

    int newNode(int x) {
        ++tot;
        info[tot] = info[x];
        return tot;
    }

    int update(int x, int l, int r, int id, int w) {
        x = newNode(x);
        if (l == r) {
            info[x].w += w;
        } else {
            int mid = l + r >> 1;
            if (id <= mid) {
                info[x].ls = update(info[x].ls, l, mid, id, w);
            } else {
                info[x].rs = update(info[x].rs, mid + 1, r, id, w);
            }
            pushup(x);
        }
        return x;
    }

    int kth(int x, int y, int l, int r, int k) {
        if (l == r) return l;
        int mid = l + r >> 1;
        int lx = info[x].ls, rx = info[x].rs;
        int ly = info[y].ls, ry = info[y].rs;
        int ls = info[ly].w - info[lx].w;
        if (k <= ls) {
            return kth(lx, ly, l, mid, k);
        } else {
            return kth(rx, ry, mid + 1, r, k - ls);
        }
    }
};
```

SegmentTreeBeats

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

#define pr pair<int,int>
#define mk(a,b) make_pair(a,b)
#define LL long long
namespace SegmentTreeBeats{
```

```
typedef LL T;
#define lson l,mid,rt<<1
#define rson mid+1,r,rt<<1|1

const T inf=1e9+7;
const int maxn=5e5+5;
T sum[maxn<<2],col[maxn<<2],v[maxn<<2];

template<class cmp,T Inf>
struct INFO{
    T mv,smv;
    int cnt;
    const static T INF=Inf;
    INFO operator + (const INFO& p)const{
        INFO ans;
        if(cmp()(mv,p.mv)){
            ans.mv=p.mv;
            ans.cnt=p.cnt;
            ans.smv=cmp()(mv,p.smv)?p.smv:mv;
        }else if(cmp()(p.mv,mv)){
            ans.mv=mv;
            ans.cnt=cnt;
            ans.smv=cmp()(smv,p.mv)?p.mv:smv;
        }else{
            ans.mv=mv;
            ans.cnt=cnt+p.cnt;
            ans.smv=cmp()(smv,p.smv)?p.smv:smv;
        }
        return ans;
    }
};
INFO<less<T>,-inf>Mx[maxn<<2];
INFO<greater<T>,inf>Mn[maxn<<2];
void updata(int rt){
    sum[rt]=sum[rt<<1]+sum[rt<<1|1];
    Mx[rt]=Mx[rt<<1]+Mx[rt<<1|1];
    Mn[rt]=Mn[rt<<1]+Mn[rt<<1|1];
}
void build(int l,int r,int rt){
    col[rt]=0;
    if(l==r){
        Mx[rt].mv=Mn[rt].mv=sum[rt]=v[l];
        Mx[rt].cnt=Mn[rt].cnt=1;
        Mx[rt].smv=Mx[rt].INF;
        Mn[rt].smv=Mn[rt].INF;
        return;
    }
    int mid=(l+r)>>1;
    build(lson);
    build(rson);
    updata(rt);
}
inline void color(int l,int r,int rt,T val){
    sum[rt]+=val*(r-l+1);
    col[rt]+=val;
    if(Mx[rt].smv!=-inf)Mx[rt].smv+=val;
    if(Mn[rt].smv!=inf)Mn[rt].smv+=val;
    Mx[rt].mv+=val,Mn[rt].mv+=val;
}
inline void colorToLess(int rt,T val){
    if(Mx[rt].mv<=val)return;
    sum[rt]+=(val-Mx[rt].mv)*Mx[rt].cnt;
    if(Mn[rt].smv==Mx[rt].mv)Mn[rt].smv=val;
    if(Mn[rt].mv==Mx[rt].mv)Mn[rt].mv=val;
    Mx[rt].mv=val;
}
inline void colorToMore(int rt,T val){
    if(Mn[rt].mv>=val)return;
    sum[rt]+=(val-Mn[rt].mv)*Mn[rt].cnt;
    if(Mx[rt].smv==Mn[rt].mv)Mx[rt].smv=val;
    if(Mx[rt].mv==Mn[rt].mv)Mx[rt].mv=val;
```

```
        Mn[rt].mv=val;
    }
    inline void pushcol(int l,int r,int rt){
        if(col[rt]){
            int mid=(l+r)>>1;
            color(lson,col[rt]);
            color(rson,col[rt]);
            col[rt]=0;
        }
        colorToLess(rt<<1,Mx[rt].mv);
        colorToLess(rt<<1|1,Mx[rt].mv);
        colorToMore(rt<<1,Mn[rt].mv);
        colorToMore(rt<<1|1,Mn[rt].mv);
    }
    void toLess(int l,int r,int rt,int nl,int nr,T val){
        if(Mx[rt].mv<=val)return;
        if(nl<=l&&nr>=r&&Mx[rt].smv<val){
            colorToLess(rt,val);
            return;
        }
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        if(nl<=mid)toLess(lson,nl,nr,val);
        if(nr>mid)toLess(rson,nl,nr,val);
        updata(rt);
    }

    void toMore(int l,int r,int rt,int nl,int nr,T val){
        if(Mn[rt].mv>=val)return;
        if(nl<=l&&nr>=r&&Mn[rt].smv>val){
            colorToMore(rt,val);
            return;
        }
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        if(nl<=mid)toMore(lson,nl,nr,val);
        if(nr>mid)toMore(rson,nl,nr,val);
        updata(rt);
    }
    void modify(int l,int r,int rt,int nl,int nr,T val){
        if(nl<=l&&nr>=r){
            color(l,r,rt,val);
            return ;
        }
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        if(nl<=mid)modify(lson,nl,nr,val);
        if(nr>mid)modify(rson,nl,nr,val);
        updata(rt);
    }

    T querySum(int l,int r,int rt,int nl,int nr){
        if(nl<=l&&nr>=r)return sum[rt];
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        T ans=0;
        if(nl<=mid)ans+=querySum(lson,nl,nr);
        if(nr>mid)ans+=querySum(rson,nl,nr);
        return ans;
    }
    T queryMx(int l,int r,int rt,int nl,int nr){
        if(nl<=l&&nr>=r)return Mx[rt].mv;
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        T ans=Mx->INF;
        if(nl<=mid)ans=max(ans,queryMx(lson,nl,nr));
        if(nr>mid)ans=max(ans,queryMx(rson,nl,nr));
        return ans;
    }
    T queryMn(int l,int r,int rt,int nl,int nr){
        if(nl<=l&&nr>=r)return Mn[rt].mv;
```

```cpp
        pushcol(l,r,rt);
        int mid=(l+r)>>1;
        T ans=Mn->INF;
        if(nl<=mid)ans=min(ans,queryMn(lson,nl,nr));
        if(nr>mid)ans=min(ans,queryMn(rson,nl,nr));
        return ans;
    }

    #undef lson
    #undef rson
}
using namespace SegmentTreeBeats;

int main(){
  //  freopen("4695.in","r",stdin);
 //   freopen("4695.out","w",stdout);
    int n,m;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%lld",&v[i]);
    build(1,n,1);
    scanf("%d",&m);
    for(int i=1;i<=m;i++){
        int ck,l,r,val;
        scanf("%d%d%d",&ck,&l,&r);
        if(ck<=3)scanf("%d",&val);
        if(ck==1)modify(1,n,1,l,r,val); //区间加
        else if(ck==2)toMore(1,n,1,l,r,val);    //区间取 max
        else if(ck==3)toLess(1,n,1,l,r,val);    //区间取 min
        else if(ck==4)printf("%lld\n",querySum(1,n,1,l,r)); //查询区间和
        else if(ck==5)printf("%lld\n",queryMx(1,n,1,l,r));  //查询区间 max
        else printf("%lld\n",queryMn(1,n,1,l,r));   //查询区间 min
    }
}
```

fhq_Treap

洛谷 P3369/P6136 普通平衡树

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

typedef long long ll;
const int maxn = 100005;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;

struct TreapNode {
    int ls, rs;
    int w, sz, rnd;
} tr[maxn];
int tot;

int newNode(int w) {
    int p = ++tot;
    tr[p].ls = tr[p].rs = 0;
    tr[p].w = w;
    tr[p].sz = 1;
    tr[p].rnd = rand();
    return p;
}

void pushup(int p) {
    int ls = tr[p].ls, rs = tr[p].rs;
    tr[p].sz = tr[ls].sz + tr[rs].sz + 1;
}

void split(int p, int w, int &x, int &y) {
    if (!p) {
        x = y = 0;
        return;
    }
```

```cpp
    if (tr[p].w <= w) {
        x = p;
        split(tr[p].rs, w, tr[p].rs, y);
    } else {
        y = p;
        split(tr[p].ls, w, x, tr[p].ls);
    }
    pushup(p);
}

int merge(int x, int y) {
    if (!x || !y) return x + y;
    int ans = 0;
    if (tr[x].rnd > tr[y].rnd) {
        tr[x].rs = merge(tr[x].rs, y);
        ans = x;
    } else {
        tr[y].ls = merge(x, tr[y].ls);
        ans = y;
    }
    pushup(ans);
    return ans;
}

int kth(int p, int k) {
    int ls = tr[p].ls, rs = tr[p].rs;
    if (tr[ls].sz + 1 == k) return tr[p].w;
    else if (tr[ls].sz + 1 < k) return kth(tr[p].rs, k - tr[ls].sz - 1);
    else return kth(tr[p].ls, k);
}

int main() {
    srand(time(0));
    int q;
    scanf("%d", &q);
    int rt = 0;
    int x, y, z;
    while (q--) {
        int op, w;
        scanf("%d%d", &op, &w);
        if (op == 1) {
            split(rt, w, x, y);
            rt = merge(merge(x, newNode(w)), y);
        } else if (op == 2) {
            split(rt, w, x, y);
            split(x, w - 1, x, z);
            z = merge(tr[z].ls, tr[z].rs);
            rt = merge(merge(x, z), y);
        } else if (op == 3) {
            split(rt, w - 1, x, y);
            printf("%d\n", tr[x].sz + 1);
            rt = merge(x, y);
        } else if (op == 4) {
            printf("%d\n", kth(rt, w));
        } else if (op == 5) {
            split(rt, w - 1, x, y);
            printf("%d\n", kth(x, tr[x].sz));
            rt = merge(x, y);
        } else if (op == 6) {
            split(rt, w, x, y);
            printf("%d\n", kth(y, 1));
            rt = merge(x, y);
        }
    }

    return 0;
}

/*
若多组数据 初始化时 tot=0; rt=0;
6 种操作:
```

```
1 插入 w
2 删除 w 若重复则只删除 1 个
3 查询 w 的排名（比 w 小的数的个数 +1）
4 查询排名为 w 的数
5 求 w 的前驱（比 w 小的最大数）
6 求 w 的后继（比 w 大的最小数）
*/
```

洛谷 P2042 维护数列

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

typedef long long ll;
const int maxn = 500005;
const int inf = 0x3f3f3f3f;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int mod = 1e9 + 7;

struct TreapNode {
    int ls, rs;
    int w, sum, sz, rnd;
    int pre, suf, seq;
    int rev, cov;
} tr[maxn];
int rt;
queue<int> q;

int a[maxn];

void rev(int p) {
    tr[p].rev ^= 1;
    swap(tr[p].ls, tr[p].rs);
    swap(tr[p].pre, tr[p].suf);
}

void cov(int p, int c) {
    tr[p].cov = c;
    tr[p].w = c;
    tr[p].sum = c * tr[p].sz;
    tr[p].pre = tr[p].suf = max(0, tr[p].sum);
    tr[p].seq = max(c, tr[p].sum);
}

void del(int p) {
    if (!p) return;
    q.push(p);
    del(tr[p].ls);
    del(tr[p].rs);
}

void pushdown(int p) {
    int ls = tr[p].ls, rs = tr[p].rs;
    if (tr[p].rev) {
        if (ls) rev(ls);
        if (rs) rev(rs);
        tr[p].rev = 0;
    }
    if (tr[p].cov != inf) {
        if (ls) cov(ls, tr[p].cov);
        if (rs) cov(rs, tr[p].cov);
        tr[p].cov = inf;
    }
}

void pushup(int p) {
    int ls = tr[p].ls, rs = tr[p].rs;
    tr[p].sz = tr[ls].sz + tr[rs].sz + 1;
    tr[p].sum = tr[ls].sum + tr[rs].sum + tr[p].w;

    tr[p].pre = max(tr[ls].pre, tr[ls].sum + tr[p].w + tr[rs].pre);
```

```
    tr[p].suf = max(tr[rs].suf, tr[rs].sum + tr[p].w + tr[ls].suf);
    tr[p].seq = tr[ls].suf + tr[p].w + tr[rs].pre;
    if (ls) tr[p].seq = max(tr[p].seq, tr[ls].seq);
    if (rs) tr[p].seq = max(tr[p].seq, tr[rs].seq);
}

int newNode(int w) {
    int p = q.front();
    q.pop();
    tr[p].ls = tr[p].rs = 0;
    tr[p].w = tr[p].sum = w;
    tr[p].sz = 1;
    tr[p].rnd = rand();
    tr[p].pre = tr[p].suf = max(0, w); tr[p].seq = w;
    tr[p].rev = 0; tr[p].cov = inf;
    return p;
}

int build(int l, int r) {
    int mid = (l + r) >> 1;
    int p = newNode(a[mid]);
    if (l <= mid - 1) tr[p].ls = build(l, mid - 1);
    if (mid + 1 <= r) tr[p].rs = build(mid + 1, r);
    pushup(p);
    return p;
}

void split(int p, int sz, int &x, int &y) {
    if (!p) {
        x = y = 0;
        return;
    }
    pushdown(p);
    int ls = tr[p].ls, rs = tr[p].rs;
    if (tr[ls].sz >= sz) {
        y = p;
        split(ls, sz, x, tr[p].ls);
    } else {
        x = p;
        split(rs, sz - 1 - tr[ls].sz, tr[p].rs, y);
    }
    pushup(p);
}

int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (tr[x].rnd > tr[y].rnd) {
        pushdown(x);
        tr[x].rs = merge(tr[x].rs, y);
        pushup(x);
        return x;
    } else {
        pushdown(y);
        tr[y].ls = merge(x, tr[y].ls);
        pushup(y);
        return y;
    }
}

void INSERT() {
    int pos, tot;
    scanf("%d%d", &pos, &tot);
    if (tot == 0) return;
    for (int i = 1; i <= tot; i++) {
        scanf("%d", a + i);
    }
    int y = build(1, tot);
    int x, z;
    split(rt, pos, x, z);
    rt = merge(merge(x, y), z);
}
```

```
void DELETE() {
    int pos, tot;
    scanf("%d%d", &pos, &tot);
    if (tot == 0) return;
    int x, y, z;
    split(rt, pos - 1, x, y);
    split(y, tot, y, z);
    del(y);
    rt = merge(x, z);
}

void COVER() {
    int pos, tot, c;
    scanf("%d%d%d", &pos, &tot, &c);
    if (tot == 0) return;
    int x, y, z;
    split(rt, pos - 1, x, y);
    split(y, tot, y, z);
    cov(y, c);
    rt = merge(merge(x, y), z);
}

void REVERSE() {
    int pos, tot;
    scanf("%d%d", &pos, &tot);
    if (tot == 0) return;
    int x, y, z;
    split(rt, pos - 1, x, y);
    split(y, tot, y, z);
    rev(y);
    rt = merge(merge(x, y), z);
}

void GETSUM() {
    int pos, tot;
    scanf("%d%d", &pos, &tot);
    if (tot == 0) {
        printf("0\n");
        return;
    }
    int x, y, z;
    split(rt, pos - 1, x, y);
    split(y, tot, y, z);
    printf("%d\n", tr[y].sum);
    rt = merge(merge(x, y), z);
}

void MAXSUM() {
    printf("%d\n", tr[rt].seq);
}

int main() {
    srand(time(0));
    for (int i = 1; i < maxn; i++) {
        q.push(i);
    }

    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", a + i);
    }
    rt = build(1, n);

    while (m--) {
        char op[20];
        scanf("%s", op);
        if (op[0] == 'I') {
            INSERT();
        } else if (op[0] == 'D') {
```

```
            DELETE();
        } else if (op[0] == 'M' && op[2] == 'K') {
            COVER();
        } else if (op[0] == 'R') {
            REVERSE();
        } else if (op[0] == 'G') {
            GETSUM();
        } else if (op[0] == 'M' && op[2] == 'X') {
            MAXSUM();
        }
    }

    return 0;
}

/*
INSERT(); 往第 pos 个数后插入 tot 个数
DELETE(); 从 pos 起删除 tot 个数（含 pos）
COVER(); 从 pos 起 tot 个数均修改为 c（含 pos）
REVERSE(); 从 pos 起 tot 个数翻转（含 pos）
GETSUM(); 从 pos 起 tot 个数求和（含 pos）
MAXSUM(); 求整个数列的最大连续子段和（不可取空段）
*/
```

树

倍增 LCA

```
/*
首先 dfs 一遍找出所有节点的父亲 fa 和深度 d
fa[rt] = -1;
*/

int st[maxn][20];  //n < (1<<20)

void initLCA(int n) {
    st[0][0] = -1;
    for (int i = 1; i <= n; i++) st[i][0] = fa[i];

    for (int j = 1; j < 20; j++) {
        for (int i = 0; i <= n; i++) {
            if (st[i][j - 1] < 0) st[i][j] = -1;
            else st[i][j] = st[st[i][j - 1]][j - 1];
        }
    }
}

int LCA(int u, int v) {
    if (d[u] < d[v]) swap(u, v);
    for (int i = 0; d[u] != d[v]; i++) {
        if ((d[u] - d[v]) >> i & 1) {
            u = st[u][i];
        }
    }
    if (u == v) return u;

    for (int i = 19; i >= 0; i--) {
        if (st[u][i] != -1 && st[v][i] != -1 && st[u][i] != st[v][i]) {
            u = st[u][i];
            v = st[v][i];
        }
    }
    return st[u][0];
}
```

LCA 的个数

- 结论：树上选取 n 个结点，两两之间取 LCA，本质不同的 LCA 最多为 $n-1$ 个

- 证明：往已有的 n 个点中加入一个新点，最多增加一个新的本质不同的 LCA，该 LCA 为新点与旧点的所有 LCA 中距离新点最近的 LCA，然后数学归纳法。
- 补充：选取的 n 个结点没有任何两点互相为祖先的话，必然有 $n-1$ 个 LCA，若有点互为祖先，依然可能有 $n-1$ 个 LCA。

LCA 与 DFS 序

- 结论：树上选取一个点集 $\{S\}$，点集中一定存在至少一对点对 (u,v) 使得 $LCA(u,v) = LCA(S)$，且当 (u,v) 分别为点集 $\{S\}$ 中 DFS 序最小与最大时，该结论一定成立。
- 该结论等价于：对于点对 $(u,v)(DFN[u] < DFN[v])$，定点 u 和动点 v 的 LCA 随 v 的 DFS 序增大而不减，定点 v 和动点 u 的 LCA 随 u 的 DFS 序减小而不减。

点到链的最短距离（HDU 5296）

- 题意：边权树上给出一个点集 $\{S\}$ 和一个点 $u(u \notin S)$，从 S 中任选两个点 (x,y) 组成一条链，问组成的链离 u 的最短距离是多少。

- 做法：从 S 中选点，从 DFS 序小于 u 的点中选出 DFS 序最大的一个点 x，从 DFS 序大于 u 的点中选出 DFS 序最小的一个点 y，若不存在这样的 x 或者 y，则选取 DFS 序最小和最大的两个点作为 x 和 y。求 u 到链 (x,y) 的距离。

- 求值：$dis[i]$ 表示点 i 到根节点的距离，$ans = dis[u] - dis[lca(u,x)] - dis[lca(u,y)] + dis[lca(x,y)]$

点分治

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

using namespace std;
typedef long long ll;
const int maxn = 20005;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;

struct edge {
    int to, val;
};
vector<edge> mp[maxn];

int mini, rt, totSZ;
int sz[maxn], dis[maxn];
bool vis[maxn];

int l, r, q[maxn];  //q 为每次得到的距离合集

int n, ans; //ans 记录合法点对

void getRT(int u, int pre) {    //每次调用 getRT() 前使 mini=inf, totSZ = sz[v];
    sz[u] = 1;
    int mxSub = 0;
    for (auto it: mp[u]) {
        int v = it.to;
        if (v == pre || vis[v]) continue;

        getRT(v, u);
        sz[u] += sz[v];
        mxSub = max(mxSub, sz[v]);
    }

    int mx = max(mxSub, totSZ - sz[u]);
    if (mx < mini) {
        mini = mx;
        rt = u;
    }
}

void getDIS(int u, int pre) {
    q[++r] = dis[u];
    for (auto it: mp[u]) {
        int v = it.to, val = it.val;
```

```cpp
        if (v == pre || vis[v]) continue;
        dis[v] = dis[u] + val;
        getDIS(v, u);
    }
}

int calc(int u, int val) {
    l = 1, r = 0;
    dis[u] = val;
    getDIS(u, 0);

    //按照题意处理 q

    return sum;
}

void dfs(int u) {
    vis[u] = true;
    ans += calc(u, 0);
    for (auto it: mp[u]) {
        int v = it.to, val = it.val;
        if (vis[v]) continue;

        ans -= calc(v, val);

        mini = inf;
        totSZ = sz[v];
        getRT(v, 0);
        dfs(rt);
    }
}

int main() {

    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; i++) {
            mp[i].clear();
            vis[i] = false;
        }
        ans = 0;

        for (int i = 1; i < n; i++) {
            int u, v, val;
            scanf("%d%d%d", &u, &v, &val);
            mp[u].push_back(edge{v, val});
            mp[v].push_back(edge{u, val});
        }

        mini = inf;
        totSZ = n;
        getRT(1, 0);
        dfs(rt);

        printf("%d\n", ans);
    }

    return 0;
}
```

异或最小生成树（CF 888G）

题面

给出 n 点和他们的权值 a_i，连接点 i 与点 j 的边权值为 $a_i \bigoplus a_j$，求最小生成树。

$Boruvka$ 算法

虽然我是按照 $Kruskal$ 的思路解的这题，但看大部分题解都说是 $Boruvka$，且我看代码和我没啥区别，姑且把这个偏门算法贴在这里。

- 算法流程：对于每一个连通块，枚举其出边。取其最小出边，合并两个连通块。

- 复杂度：每次合并联通块个数减少一半，$O(nlogn)$。

Solution

- 将所有点权加入 $0-1Trie$ 中，n 个点看成 n 个叶子结点，则连接两点 i, j 的边权值显然从 $LCA(i, j)$ 开始计算贡献，考虑 $Kruskal$，显然应该从最深的 LCA 开始连边。
- 结论 1：若一个 $0-1Trie$ 结点有 2 个儿子，则该结点必为某两叶子结点的 LCA
- 结论 2：树上 n 个点两两取 LCA，最多 $n-1$ 个本质不同的 LCA。(详情参考数论相关 $-LCA$ 的个数)
- 推论：这颗 $Trie$ 上最多 $n-1$ 个点有 2 个儿子
- 2 个儿子的结点实际相当于 0 儿子与 1 儿子分别为联通块，因为单个儿子子树内的点与本子树内的点相连一定优于与子树外的点相连。因此找到最短的一条连接两个联通块的边即可。
- 找最短边的过程：枚举 0 儿子内的左端点，在 1 儿子内按 $Trie$ 树遍历的方式找使异或值最小的右端点。
- 时间复杂度：对于每个 2 儿子点，找最短边时枚举左端点的极限数量为其子树大小，在最大度为 2 的树上，n 个**不同**点的子树大小和为 $n + (n/2 + n/2) + (n/4 + n/4 + n/4 + n/4) + (n/8 * 8)...$ 即 $O(nlogn)$，对每个左端点找右端点 $O(logn)$，因此总复杂度 $O(nlog^2n)$。

Code

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

typedef long long ll;
const int maxn = 200005;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;

struct TrieNode {
    int son[2];
} trie[maxn * 30];
int tot = 0;
int L[maxn * 30], R[maxn * 30];

int a[maxn];

void newNode() {
    tot++;
    trie[tot].son[0] = 0;
    trie[tot].son[1] = 0;
}

void insert(int val, int id) {
    int now = 0;
    for (int i = 29; i >= 0; i--) {
        int ch = (val >> i) & 1;
        if (!trie[now].son[ch]) {
            newNode();
            trie[now].son[ch] = tot;
            L[tot] = id;
        }
        now = trie[now].son[ch];
        R[now] = id;
    }
}

int query(int now, int val, int d) {
    if (d < 0) return 0;
    int ans = 0;
    for (int i = d; i >= 0; i--) {
        int ch = (val >> i) & 1;
        if (trie[now].son[ch]) {
            now = trie[now].son[ch];
        } else {
            ans += (1 << i);
            now = trie[now].son[ch ^ 1];
        }
    }
    return ans;
}
```

```cpp

ll dfs(int u, int d) {
    int x = trie[u].son[0];
    int y = trie[u].son[1];
    if (x && y) {
        int mini = (1 << 30);
        for (int i = L[x]; i <= R[x]; i++) {
            mini = min(mini, query(y, a[i], d - 1) + (1 << d));
        }
        return dfs(x, d - 1) + dfs(y, d - 1) + mini;
    } else if (x) {
        return dfs(x, d - 1);
    } else if (y) {
        return dfs(y, d - 1);
    } else {
        return 0;
    }
}

int main() {

    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", a + i);
    }
    sort(a + 1, a + 1 + n);
    for (int i = 1; i <= n; i++) {
        insert(a[i], i);
    }
    L[0] = 1; R[0] = n;

    ll ans = dfs(0, 29);
    printf("%lld\n", ans);

    return 0;
}
```

图论

Tarjan

求边双，桥

- 判定桥 (u,v)：u 和 v 不在同一边双中，(u,v) 是桥

```cpp
pii edge[M];
vector<pii> G[N]; //存边标号
int dfn[N], low[N], tot;

stack<int> stk;
bool inST[N];

int grp[N];

void tarjanE(int u, int from) {
    dfn[u] = low[u] = ++tot;
    stk.push(u);
    inST[u] = true;
    for (pii e : G[u]) {
        if (e.second == from) continue;
        int v = e.first;
        if (!dfn[v]) {
            tarjanE(v, e.second);
            low[u] = min(low[u], low[v]);
        } else if (inST[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
```

```cpp
        while (inST[u]) {
            int v = stk.top();
            stk.pop();
            inST[v] = false;
            grp[v] = u;
        }
    }
}
```

求点双、割点

```cpp
pii edge[M];
vector<pii> G[N];
int dfn[N], low[N], tot;

vector<vector<int> > dcc; //存点双边集
bool cut[N];
stack<int> stkE; //存边标号
bool vis[M]; //每条边仅访问一次，因为每条边只属于一个点双

void tarjan_dcc(int u, int from) {
    dfn[u] = low[u] = ++tot;
    int child = 0;
    for (pii e : G[u]) {
        if (vis[e.second]) continue;
        vis[e.second] = true;
        stkE.push(e.second);
        int v = e.first;
        if (!dfn[v]) {
            child++;
            tarjan_dcc(v, e.second);
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v]) { //x 是割点，并且搓出个点双边集合
                cut[u] = true;
                vector<int> tmp;
                while (stkE.top() != e.second) {
                    tmp.push_back(stkE.top());
                    stkE.pop();
                }
                stkE.pop();
                tmp.push_back(e.second);
                dcc.push_back(tmp);
            }
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (from == 0 && child < 2) cut[u] = false; //根特殊处理
}
```

求 SCC（略）

DAG 与拓扑

- 对于一张 DAG 上的任意拓扑序，对任意点 u，一定不可能到达拓扑序小于 u 的点，否则要么成环，要么不符合拓扑
- 对于一张 DAG，某点 u 能被所有点到达的充要条件是：u 是当前唯一出度为 0 的点

匈牙利算法

```cpp
//匈牙利算法求二分图最大匹配 时间 O(Nx*E)
#include <bits/stdc++.h>

using namespace std;
const int maxn = 2005;

vector<int> mp[maxn];
bool used[maxn];
int cx[maxn], cy[maxn];
```

```
int Nx, Ny;

void init() {
    for (int i = 1; i <= Nx; i++) mp[i].clear();
}

void add_edge(int x, int y) {
    mp[x].push_back(y);
}

bool dfs(int x) {
    int sz = mp[x].size();
    for (int i = 0; i < sz; i++) {
        int y = mp[x][i];
        if (!used[y]) {
            used[y] = true;
            if (cy[y] == 0 || dfs(cy[y])) {
                cx[x] = y;
                cy[y] = x;
                return true;
            }
        }
    }
    return false;
}

int max_match() {
    for (int i = 1; i <= Nx; i++) cx[i] = 0;
    for (int i = 1; i <= Ny; i++) cy[i] = 0;

    int ans = 0;
    for (int i = 1; i <= Nx; i++) {
        for (int j = 1; j <= Ny; j++) {
            used[j] = false;
        }
        if (dfs(i)) ans++;
    }
    return ans;
}

int main() {

    int n, m, e;
    scanf("%d%d%d", &n, &m, &e);
    Nx = n, Ny = m;
    init();

    for (int i = 0; i < e; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        add_edge(x, y);
    }

    printf("%d", max_match());

    return 0;
}

/*
Nx 为左，Ny 为右
cx[],cy[] 为匹配成功后情况
cx[i]=j 表示左边 i 连右边 j

调用 init() 前先赋值 Nx, Ny
初始化 init();
加边 add_edge(x, y);
二分图最大匹配答案 max_match()
*/
```

HK 求二分图最大匹配

```cpp
//Hopcroft-Karp, O(sqrt(n) * m)
#include <bits/stdc++.h>

typedef long long ll;
using namespace std;
const int maxn = 2005;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;

vector<int> mp[maxn];

int Nx, Ny, dis;
int dx[maxn], dy[maxn];
int cx[maxn], cy[maxn];
bool used[maxn];

void init() {
    for (int i = 1; i <= Nx; i++) mp[i].clear();
}

void add_edge(int x, int y) {
    mp[x].push_back(y);
}

bool bfs() {
    queue<int> que;
    dis = inf;

    for (int i = 1; i <= Nx; i++) dx[i] = -1;
    for (int i = 1; i <= Ny; i++) dy[i] = -1;

    for (int i = 1; i <= Nx; i++) {
        if (cx[i] == -1) {
            que.push(i);
            dx[i] = 0;
        }
    }
    while (!que.empty()) {
        int x = que.front(); que.pop();
        if (dx[x] > dis) break;
        for (int i = 0; i < mp[x].size(); i++) {
            int y =  mp[x][i];
            if (dy[y] == -1) {
                dy[y] = dx[x] + 1;
                if (cy[y] == -1) dis = dy[y];
                else {
                    dx[cy[y]] = dy[y] + 1;
                    que.push(cy[y]);
                }
            }
        }
    }
    return dis != inf;
}

int dfs(int x) {
    int sz = mp[x].size();
    for (int i = 0; i < sz; i++) {
        int y = mp[x][i];
        if (!used[y] && dy[y] == dx[x] + 1) {
            used[y] = true;
            if (cy[y] != -1 && dy[y] == dis) continue;

            if (cy[y] == -1 || dfs(cy[y])) {
                cx[x] = y;
                cy[y] = x;
                return 1;
            }
        }
```

```cpp
    }
    return 0;
}

int max_match() {
    for (int i = 1; i <= Nx; i++) cx[i] = -1;
    for (int i = 1; i <= Ny; i++) cy[i] = -1;

    int ans = 0;
    while (bfs()) {
        for (int i = 1; i <= Ny; i++) {
            used[i] = false;
        }
        for (int i = 1; i <= Nx; i++) {
            if (cx[i] == -1) {
                ans += dfs(i);
            }
        }
    }
    return ans;
}

int main() {

    int n, m, e;
    scanf("%d%d%d", &n, &m, &e);
    Nx = n, Ny = m;
    init();

    for (int i = 0; i < e; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        add_edge(x, y);
    }

    printf("%d", max_match());

    return 0;
}

/*
Nx 为左, Ny 为右
cx[],cy[] 为匹配成功后情况
cx[i]=j 表示左边 i 连右边 j

调用 init() 前先赋值 Nx, Ny
初始化 init();
加边 add_edge(x, y);
二分图最大匹配答案 max_match()
*/
```

KM 求二分图最佳完备匹配

```cpp
//KM 求二分图最佳完备匹配  O(n^3)
struct KM { //by @tokiisukaze
#define type int
#define inf 0x3f3f3f3f
    static const int N = 505;
    int n, mx[N], my[N], prv[N];
    type slk[N], lx[N], ly[N], w[N][N];
    bool vx[N], vy[N];

    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                w[i][j] = 0;
            }
        }
    }

```

```cpp
    void addEdge(int x, int y, type val) { w[x][y] = val; }

    void match(int y) { while (y) swap(y, mx[my[y] = prv[y]]); }

    void bfs(int x) {
        int i, y;
        type d;
        for (i = 1; i <= n; i++) {
            vx[i] = vy[i] = false;
            slk[i] = inf;
        }
        queue<int> q;
        q.push(x);
        vx[x] = true;
        while (true) {
            while (!q.empty()) {
                x = q.front();
                q.pop();
                for (y = 1; y <= n; y++) {
                    d = lx[x] + ly[y] - w[x][y];
                    if (!vy[y] && d <= slk[y]) {
                        prv[y] = x;
                        if (!d) {
                            if (!my[y]) return match(y);
                            q.push(my[y]);
                            vx[my[y]] = true;
                            vy[y] = true;
                        } else slk[y] = d;
                    }
                }
            }
            d = inf + 1;
            for (i = 1; i <= n; i++) {
                if (!vy[i] && slk[i] < d) {
                    d = slk[i];
                    y = i;
                }
            }
            for (i = 1; i <= n; i++) {
                if (vx[i]) lx[i] -= d;
                if (vy[i]) ly[i] += d;
                else slk[i] -= d;
            }
            if (!my[y]) return match(y);
            q.push(my[y]);
            vx[my[y]] = true;
            vy[y] = true;
        }
    }

    type max_match() {
        int i;
        type res;
        for (i = 1; i <= n; i++) {
            mx[i] = my[i] = ly[i] = 0;
            lx[i] = *max_element(w[i] + 1, w[i] + n + 1);
        }
        for (i = 1; i <= n; i++) bfs(i);
        res = 0;
        for (i = 1; i <= n; i++) res += lx[i] + ly[i];
        return res;
    }

#undef type
#undef inf
} km;
/*
O(n^3)
km.init(n);
km.addEdge(a,b,val); a,b: 1~n

```

```
非常重要:
inf >= abs(max_match())
inf >= abs(max_match())
inf >= abs(max_match())

判断是否完备匹配的方法:
不连接的边赋值-infx (infx 必须远小于 inf, 否则不满足上述 inf 条件)
保证 infx>|N|*|val|
使 max_match 匹配值离谱
*/
```

Dinic 最大流

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

typedef long long ll;
using namespace std;

#define INF 0x3f3f3f3f3f3f3f3f

struct Dinic {
    static const int N = 1010;

    struct Edge {
        int from, to;
        ll cap, flow;

        Edge(int u, int v, ll c, ll f) : from(u), to(v), cap(c), flow(f) {}
    };

    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[N];
    int d[N], cur[N];
    bool vis[N];

    void init(int n) {
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }

    void AddEdge(int from, int to, ll cap) {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!Q.empty()) {
            int x = Q.front();
            Q.pop();
            for (int i = 0; i < G[x].size(); i++) {
                Edge &e = edges[G[x][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return vis[t];
    }

```

```cpp
    ll DFS(int x, ll a) {
        if (x == t || a == 0) return a;
        ll flow = 0, f;
        for (int &i = cur[x]; i < G[x].size(); i++) {
            Edge &e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }

    ll Maxflow(int s, int t) {
        this->s = s;
        this->t = t;
        ll flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, INF);
        }
        return flow;
    }
};

/*
n 点 m 边 s 起 t 终 点编号 1~n
init(n) 清空邻接表
AddEdge(from, to, cap) from 到 to 的有向边，流量 cap
Maxflow(s, t) s 起 t 终的最大流量
*/
```

最小费用最大流

```cpp
//最小费用最大流
#include <bits/stdc++.h>

typedef long long ll;
using namespace std;

#define INF 0x3f3f3f3f

template<class T>
struct Minimum_Cost_Flow_Dijkstra {
#define N 5050
#define P pair<T, int>
    struct edge {
        int to;
        T cap, cost, rev;
    };
    T flow, res, dist[N], h[N];
    vector<edge> G[N];
    int preV[N], preE[N], n;
    inline void init(int x) {
        n = x;
        for (int i = 0; i <= n; i++) {
            G[i].clear();
        }
    }

    inline void addEdge(int from, int to, T cap, T cost) {
        G[from].push_back((edge) {to, cap, cost, (int) G[to].size()});
        G[to].push_back((edge) {from, 0, -cost, (int) G[from].size() - 1});
    }

    inline void min_cost_flow(int s, int t, T f) {
        fill(h + 1, h + 1 + n, 0);
        flow = res = 0;
```

```cpp
        while (f > 0) {
            priority_queue<P, vector<P>, greater<P> > D;
            memset(dist, INF, sizeof dist);
            dist[s] = 0;
            D.push(P(0, s));
            while (!D.empty()) {
                P now = D.top();
                D.pop();
                if (dist[now.second] < now.first) continue;

                int v = now.second;
                for (int i = 0; i < (int) G[v].size(); ++i) {
                    edge &e = G[v][i];
                    if (e.cap > 0 &&
                        dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
                        dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                        preV[e.to] = v;
                        preE[e.to] = i;
                        D.push(P(dist[e.to], e.to));
                    }
                }
            }
            if (dist[t] == INF) break;

            for (int i = 1; i <= n; ++i) {
                h[i] += dist[i];
            }
            T d = f;
            for (int v = t; v != s; v = preV[v]) {
                d = min(d, G[preV[v]][preE[v]].cap);
            }
            f -= d;
            flow += d;
            res += d * h[t];
            for (int v = t; v != s; v = preV[v]) {
                edge &e = G[preV[v]][preE[v]];
                e.cap -= d;
                G[v][e.rev].cap += d;
            }
        }
    }
#undef N
#undef P
};

/*
Minimum_Cost_Flow_Dijkstra<T> mcmf;
T 为 int 或 ll
记得修改对应 INF 值
记得修改对应 INF 值
记得修改对应 INF 值

mcmf.init(n) 初始化 n 和邻接表
mcmf.addEdge(from, to, cap, cost) from 源点, to 汇点, cap 边最大流量, cost 边单位流量费用
mcmf.min_cost_flow(s, t, INF) 起点, 终点, 总流量限制
mcmf.flow = 最大流
mcmf.res = 最大流情况下最小费用

使用样例
mcmf.min_cost_flow(s, t, INF);
printf("%d %d\n", mcmf.flow, mcmf.res);
*/
```

MST

基础结论

- 对于图中任意一个点 u，对于从 u 连出去的所有边，边权最小的一条至少存在于一颗 MST 上
- 该结论是 $Prim$ 与 $Kruskal$ 的正确性基础
- 在 $Prim$ 中，将当前的生成树视作一个点

- 在 $Kruskal$ 中，将若干连通分量视作若干个点

MST 的边权分布

- 对于任意一个图，若存在多种 MST，其边权分布不变
- 即若在 MST_1 中有 3 条长度 1 的边，则在 MST_2 中也有 3 条长度 1 的边
- 同时该结论也意味着不存在 $1+3=2+2$ 的不同 MST
- bzoj1016

路径上最长边最小

- 对于任意一个连通图，从 A 点走到 B 点的所有路径，要使路径上最长的边最小，最小的情况一定出现在 MST 中的 $Path(A,B)$。
- bzoj3732

基尔霍夫矩阵求生成树个数

```cpp
//基尔霍夫矩阵求无向图最小生成树个数
#include <bits/stdc++.h>

typedef long long ll;
using namespace std;
const int maxn = 305;
const int mod = 1e9 + 7;

int a[maxn][maxn];

int Gauss(int n) {
    int ans = 1;
    for (int i = 1; i <= n; ++i) {
        for (int k = i + 1; k <= n; ++k) {
            while (a[k][i]) {
                int d = a[i][i] / a[k][i];
                for (int j = i; j <= n; ++j) {
                    a[i][j] = (a[i][j] - 1LL * d * a[k][j] % mod + mod) % mod;
                }

                swap(a[i], a[k]);
                ans = -ans;
            }
        }
        ans = 1LL * ans * a[i][i] % mod, ans = (ans + mod) % mod;
    }
    return ans;
}

void addEdge(int u, int v) {
    --a[u][v], --a[v][u], ++a[u][u], ++a[v][v];
}

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    while (m--) {
        int u, v;
        scanf("%d%d", &u, &v);
        addEdge(u, v);
    }
    printf("%d\n", Gauss(n - 1));

    return 0;
}

/*
点 1~n
addEdge(u, v);
ans = Gauss(n - 1);
*/
```

欧拉图

定义

- 欧拉通路：通过图中所有边的简单路
- 欧拉回路：闭合的欧拉路

判定

欧拉回路存在的充要条件

- 无向图：当且仅当该图所有顶点度数都为偶数，且该图是连通图
- 有向图：所有顶点的入度等于出度且该图是连通图
- 混合图
  - 无向边随意定向，得到每个点的 $x = deg \ - deg$ ，若存在 x 为奇数，则不存在
  - 另所有 $x := x/2$，建图网络流
  - 对于 $x > 0$ （出 > 入）的点，加边 (s, i, x)
  - 对于 $x < 0$ （入 > 出）的点，加边 $(i, t, |x|)$
  - 对于随意定向的无向边 (i, j)，加边 $(i, j, 1)$
  - 存在满流则存在欧拉回路，将流量为 1 的无向边反转即可得到欧拉图

半欧拉图

- 定义：存在欧拉通路，不存在欧拉回路
- 充要条件：仅由两个点度数为奇数，这两个点分别为起点和终点

求解（Hierholzer）

- 从栈顶到栈底输出即为欧拉回路
- 求字典序最小解：优先搜索较小点 / 边

```
stack<pii> stk;
bool vis[maxn]; // vis 与 cur 大小应开 M
int cur[maxn];

void dfs(int u) {
    for (; cur[u] < G[u].size(); cur[u]++) {
        int v = G[u][cur[u]].first;
        int eid = G[u][cur[u]].second;
        if (!vis[eid]) {
            vis[eid] = true;
            dfs(v);
            stk.push({u, v});
        }
    }
}
```

奇环、偶环、负环

判断奇环

方法一：二分图染色法，二分图一定不存在奇环，因此判断给出图是否是二分图即可。

方法二：带权并查集。

```
//带权并查集
int find(int x)
{
    if(x==pre[x])return x;
    int fa=pre[x];
    pre[x]=find(pre[x]);
    val[x]^=val[fa];
    return pre[x];
}
```

```cpp
void solve()
{
    scanf("%d",&M);
    while(M--)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        add_edge(x,y);
        add_edge(y,x);
        int f1=find(x),f2=find(y);
        if(f1!=f2)
        {
            pre[f1]=f2;
            val[f1]=val[x]^val[y]^1;
        }
        else if(val[x]^val[y]==0)
            hasc=1;
    }
}
```

判断偶环

tarjan 分离出所有边双联通分量，分别检查是否存在偶环。

除非一个边双联通分量仅仅只是一个奇环，否则其中必定存在偶环。

例题

hdu5215，给出 n 点 m 边的无重边无自环无向图，求是否存在奇环偶环。

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

typedef long long ll;
using namespace std;
const int maxn = 100005;
const int maxm = 200005;
const int inf = 0x3f3f3f3f;
const int mod = 998244353;

vector<int> e[maxn];
int vis[maxn];

void init(int n) {
    for (int i = 1; i <= n; i++) {
        e[i].clear();
    }
}

bool dfs(int u, int w) {
    vis[u] = w;
    bool ans = false;
    for (int v: e[u]) {
        if (vis[v] == -1) {
            ans |= dfs(v, w ^ 1);
        } else {
            if (vis[v] == w) {
                ans = true;
            }
        }
    }
    return ans;
}

bool oddCircle(int n) {
    bool ans = false;
    for (int i = 1; i <= n; i++) {
        vis[i] = -1;
    }
    for (int i = 1; i <= n; i++) {
        if (vis[i] == -1) {
```

```cpp
            ans |= dfs(i, 0);
        }
    }
    return ans;
}

int dfn[maxn], low[maxn], idx;
stack<int> stk;
bool inST[maxn];
int pre[maxn];
vector<int> vec[maxn];

void Union(int u, int v) {
    pre[v] = u;
}

void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++idx;

    stk.push(u);
    inST[u] = true;

    for (int v: e[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
        } else if (inST[v]) {
            low[u] = min(low[u], low[v]);
        }
    }

    if (dfn[u] == low[u]) {
        while (stk.top() != u) {
            int v = stk.top();
            stk.pop();
            inST[v] = false;
            Union(u, v);
        }
        stk.pop();
        inST[u] = false;
        Union(u, u);
    }
}

bool check(vector<int> &p) {
    map<int, bool> mp;
    for (int u: p) {
        mp[u] = true;
    }

    int cntE = 0;
    for (int u: p) {
        for (int v: e[u]) {
            if (mp[v]) cntE++;
        }
    }
    cntE /= 2;

    return cntE != p.size() || cntE % 2 == 0;
}

bool evenCircle(int n) {
    idx = 0;
    for (int i = 1; i <= n; i++) {
        dfn[i] = 0;
        inST[i] = false;
        pre[i] = i;
        vec[i].clear();
    }
    while (!stk.empty()) stk.pop();
```

```cpp

    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) {
            tarjan(i, 0);
        }
    }

    for (int i = 1; i <= n; i++) {
        vec[pre[i]].push_back(i);
    }

    bool ans = false;
    for (int i = 1; i <= n; i++) {
        if (pre[i] == i && vec[i].size() > 2) {
            ans |= check(vec[i]);
        }
    }
    return ans;
}

int main() {

    int T;
    scanf("%d", &T);
    while (T--) {
        int n, m;
        scanf("%d%d", &n, &m);
        init(n);
        for (int i = 0; i < m; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            e[u].push_back(v);
            e[v].push_back(u);
        }

        printf("%s\n", oddCircle(n) ? "YES" : "NO");
        printf("%s\n", evenCircle(n) ? "YES" : "NO");
    }

    return 0;
}
```

判断负环

有的题能卡 SPFA 但是卡不了 BellmanFord，因为 BF 常数小

SPFA

SPFA 过程中，某个点入队 N 次，即说明存在负环

```cpp
/*
c[i] 表示点 i 入队次数
有负环 return true
*/

int dis[maxn];
bool inq[maxn];
int c[maxn];

bool checkNegativeRing(int n) {
    deque<int> q;
    for (int i = 1; i <= n; i++) {
        dis[i] = inf;
        inq[i] = false;
        c[i] = 0;
    }
    dis[1] = 0;
    inq[1] = true;
    c[1] = 1;
    q.push_back(1);
```

```cpp
    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        inq[u] = false;
        for (const Edge &e: G[u]) {
            int v = e.to, w = e.w;
            if (dis[u] + w < dis[v]) {
                dis[v] = dis[u] + w;
                if (inq[v]) continue;
                if (q.empty() || dis[v] < dis[q.front()]) {  //SLF 优化
                    q.push_front(v);
                } else {
                    q.push_back(v);
                }
                inq[v] = true;
                if (++c[v] > n) return true;
            }
        }
    }
    return false;
}
```

BellmanFord

BellmanFord 过程中，第 N 轮松弛仍成功，说明存在负环

```cpp
/*
有负环 return true
*/
bool bellmanFord(int n) {
    for (int i = 0; i <= n; i++) dis[i] = inf;
    dis[0] = 0;
    bool ok;
    for (int i = 1; i <= n + 1; i++) {
        ok = false;
        for (int u = 0; u <= n; u++) {
            for (const Edge &e : G[u]) {
                int v = e.to, w = e.w;
                if (dis[u] + w < dis[v]) {
                    dis[v] = dis[u] + w;
                    ok = true;
                }
            }
        }
        if (!ok) break;
    }
    return ok;
}
```

2-SAT

方法

- 若选择 x 后必须选择 y，则从 x 向 y 建一条边，举例：
  - 若 a 与 b 不能同时选，则建边 $a \to \neg b$，$b \to \neg a$
  - 若 a 与 b 要么都选要么都不选，则建边 $a \to b$，$\neg a \to \neg b$，$b \to a$，$\neg b \to \neg a$
  - 若 a 必须选，则建边 $\neg a \to a$
  - 若 a 不能选，则建边 $a \to \neg a$
  - 按此规则建图 G
- 对 G 进行强连通缩点，得到图 G'
  - 若有点 a 与 $\neg a$ 属于同一强连通分量，则无解
  - 否则必定有解
- 将 G' 上的所有边反向，然后跑拓扑求一组合法解
  - 若当前点可选，选择该点，将与该点矛盾的点标记为不可选
  - 若当前点不可选，跳过
  - 最后选出的所有点为一组合法解

注意点

- 初始化时，注意区分 n 与 $n \times 2$，$maxn$ 开双倍
- 大前提：每个集合仅含 2 个元素

Code

```
/*
下标从 1 开始，点映射规则 i->(2i-1, 2i)
init(n) 初始化
addEdge(x, cx, y, cy) 加边
bool solve(int a[]) 求解，返回 false 表示无解，a[] 存解
bool bruteForce(int a[]) 爆搜求字典序最小解，复杂度 O(n*m)
*/

template<class T>
void unique(vector<T> &v) {
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
}

const int maxn = 1e6 + 7;

struct TwoSat {
    // 映射
    int pid(int x, int c) {
        return x * 2 - 1 + c;
    }
    // 取非
    int rid(int x) {
        if (x & 1) return x + 1;
        return x - 1;
    }

    vector<pii> G[maxn * 2];
    int n;

    int dfn[maxn * 2], low[maxn * 2], tot;
    bool inST[maxn * 2];
    stack<int> stk;

    int grp[maxn * 2];

    void tarjan(int u, int from) {
        dfn[u] = low[u] = ++tot;
        stk.push(u);
        inST[u] = true;

        for (pii e : G[u]) {
            if (e.second == from) continue;
            int v = e.first;
            if (!dfn[v]) {
                tarjan(v, e.second);
                low[u] = min(low[u], low[v]);
            } else if (inST[v]) {
                low[u] = min(low[u], low[v]);
            }
        }

        if (low[u] == dfn[u]) {
            while (stk.top() != u) {
                grp[stk.top()] = u;
                inST[stk.top()] = false;
                stk.pop();
            }
            stk.pop();
            grp[u] = u;
            inST[u] = false;
        }
    }

```

```
    vector<int> rG[maxn * 2], sG[maxn * 2];

    void build() {
        for (int i = 1; i <= n * 2; i++) {
            for (pii e : G[i]) {
                int v = e.first;
                if (grp[i] == grp[v]) continue;
                rG[grp[v]].push_back(grp[i]);
            }
        }
        for (int i = 1; i <= n; i++) {
            int gx = grp[pid(i, 0)];
            int gy = grp[pid(i, 1)];
            sG[gx].push_back(gy);
            sG[gy].push_back(gx);
        }
        for (int i = 1; i <= n * 2; i++) {
            if (i != grp[i]) continue;
            unique(rG[i]);
            unique(sG[i]);
            assert(sG[i].size() == 1);
        }
    }

    int in[maxn * 2];
    bool vis[maxn * 2];

    void top() {
        for (int i = 1; i <= n * 2; i++) {
            vis[i] = true;
            in[i] = 0;
        }
        for (int i = 1; i <= n * 2; i++) {
            for (int v : rG[i]) in[v]++;
        }
        queue<int> q;
        for (int i = 1; i <= n * 2; i++) {
            if (!in[i]) q.push(i);
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (vis[u]) {
                for (int v : sG[u]) vis[v] = false;
            }
            for (int v : rG[u]) {
                in[v]--;
                if (!in[v]) q.push(v);
            }
        }
    }

    int eid;

    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n * 2; i++) {
            G[i].clear();
            rG[i].clear();
            sG[i].clear();
            dfn[i] = 0;
        }
        eid = 0;
        tot = 0;
    }

    void addEdge(int x, int cx, int y, int cy) {
        G[pid(x, cx)].emplace_back(pid(y, cy), ++eid);
    }

    bool solve(int a[] = nullptr) {
```

```cpp
        for (int i = 1; i <= n * 2; i++) {
            if (!dfn[i]) tarjan(i, 0);
        }
        for (int i = 1; i <= n; i++) {
            if (grp[pid(i, 0)] == grp[pid(i, 1)]) {
                return false;
            }
        }
        build();
        top();
        if (a != nullptr) {
            for (int i = 1; i <= n; i++) {
                a[i] = vis[grp[pid(i, 0)]] ? 0 : 1;
            }
        }
        return true;
    }

    //爆搜求字典序最小方案
    vector<int> tmp;

    bool dfs(int u) {
        if (vis[rid(u)]) return false;
        if (vis[u]) return true;
        vis[u] = true;
        tmp.push_back(u);
        for (pii e : G[u]) {
            if (!dfs(e.first)) return false;
        }
        return true;
    }

    bool bruteForce(int a[]) {
        for (int i = 1; i <= n * 2; i++) {
            vis[i] = false;
            sort(G[i].begin(), G[i].end());
        }
        for (int i = 1; i <= n * 2; i += 2) {
            if (!vis[i] && !vis[i + 1]) {
                tmp.clear();
                if (!dfs(i)) {
                    for (int j : tmp) vis[j] = false;
                    if (!dfs(i + 1)) return false;
                }
            }
        }
        for (int i = 1; i <= n; i++) {
            a[i] = vis[pid(i, 0)] ? 0 : 1;
        }
        return true;
    }
};
```

团:

团 (Clique): n 个顶点的完全图

极大团 (Maximal Clique): 表示无法是其他团的子团

最大团 (Maximum Clique): 点最多的极大团

无向图的最大团 = 该无向图补图的最大独立集

匹配、边覆盖、独立集、顶点覆盖

概念

- 匹配：在 G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：在 G 中的任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$（边覆盖所有点）

- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$
- 顶点覆盖：在 G 中的任意边都有至少一个端点属于 S 的顶点集合 $S \subseteq V$（顶点覆盖所有边）
- 最大匹配 M_{max} 最小边覆盖 F_{min} 最大独立集 S_{max} 最小顶点覆盖 S_{min}

关系

- 对于任意无孤立点的图：$|M_{max}| + |F_{max}| = |V|$，即【最大匹配数 + 最小边覆盖数 = 顶点数】
- 对于任意图：$|S_{max}| + |S_{min}| = |V|$，即【最大独立集数 + 最小顶点覆盖数 = 顶点数】

求解

- 在二分图中：$|M_{max}| = |S_{min}|$，即【最大匹配数 = 最小顶点覆盖数】，因此求出 M_{max} 即可

最小路径覆盖

定义

给定有向图 $G = (V, E)$，设 P 是 G 的一个简单路径（顶点不相交）的集合。如果 V 中每个顶点恰好在 P 的一条路上，称 P 为 G 的一个路径覆盖。P 中路径可以从 V 的任何一个顶点开始，长度任意（可以为 0），G 的最小路径覆盖是 G 的所含路径条数最少的路径覆盖。

简述：选若干条起点任意、长度任意、不经过相同顶点的路径，使得路径覆盖所有顶点，且路径条数最少。

解法

拆分 G 中的所有点 u 为 $u1$ 和 $u2$，若存在边 $u- > v$，则在新图上建边 $u1- > v2$，新图显然是张二分图，求出其最大匹配 M_{max}，$ans = |G| - M_{max}$（$|G|$ 为原图顶点数，不是新建的二分图顶点数）

洛谷 P2764 最小路径覆盖问题

代 码 待 补 全

字符串

字符串 Hash

双哈希，重载 pair 运算符，重载哈希以支持 unordered_set

```cpp
/*
scanf("%s", s);
Hash h;
h.init(s);
h.ask(1, n);
*/
pll operator % (const pll &p1, const pll &p2) {
    return {p1.first % p2.first, p1.second % p2.second};
}
pll operator * (const pll &p1, const pll &p2) {
    return {p1.first * p2.first, p1.second * p2.second};
}
pll operator + (const pll &p1, const pll &p2) {
    return {p1.first + p2.first, p1.second + p2.second};
}
pll operator - (const pll &p1, const pll &p2) {
    return {p1.first - p2.first, p1.second - p2.second};
}

struct Hash {
    string s;
    vector<pll> f;
    int n;
    // index from 1
    void init(char ss[]) {
        s = " ";
        s += string(ss);
        n = (int) s.length() - 1;
```

```cpp
        f.resize(n + 1, {0, 0});
        for (int i = 1; i <= n; i++) {
            int ch = s[i] - 'a';
            f[i] = (f[i - 1] * base % mod + pll{ch, ch}) % mod;
        }
    }
    // [l, r]
    pll ask(int l, int r) {
        return (f[r] - f[l - 1] * pw[r - l + 1] % mod + mod) % mod;
    }
};
```

KMP 单模板串匹配

```cpp
//s 为匹配串, t 为模板串

int nxt[maxn];

void GetNext() {
    int j = 0, k = -1;
    nxt[0] = -1;

    while (j < m) {
        if (k == -1 || t[j] == t[k]) {
            j++, k++;
            nxt[j] = k;
        } else {
            k = nxt[k];
        }
    }
}

int kmp() {
    GetNext();

    int i = 0, j = 0;
    int ans = 0;
    while (i < n) {
        if (j == -1 || s[i] == t[j]) {
            i++, j++;
        } else {
            j = nxt[j];
        }

        if (j == m) {
            ans++;
            j = nxt[j];
        }
    }

    return ans;
}
```

拓展 kmp

```cpp
//exkmp 求 t 与 s 的每一个后缀的 LCP
#include <bits/stdc++.h>

typedef long long ll;
using namespace std;
const int maxn = 100005;

char s[maxn], t[maxn];  //t 为模板串
int lenS, lenT;
int ex[maxn], nxt[maxn];

void getNext() {
    int i = 0;
    while (i + 1 < lenT && t[i] == t[i + 1]) i++;
    nxt[0] = lenT, nxt[1] = i;
```

```cpp

    int p = 1, mx = 1 + i;
    for (i = 2; i < lenT; i++) {
        if (nxt[i - p] + i < mx) {
            nxt[i] = nxt[i - p];
        } else {
            int j = max(mx - i, 0);
            while (i + j < lenT && t[i + j] == t[j]) j++;
            nxt[i] = j;
            p = i, mx = i + j;
        }
    }
}

void exKMP() {
    getNext();

    int i = 0, p = 0, mx;
    while (s[i] == t[i] && i < lenS && i < lenT) i++;
    ex[0] = i, mx = i;

    for (i = 1; i < lenS; i++) {
        if (nxt[i - p] + i < mx) {
            ex[i] = nxt[i - p];
        } else {
            int j = max(mx - i, 0);
            while (i + j < lenS && j < lenT && s[i + j] == t[j]) j++;
            ex[i] = j;
            p = i, mx = i + j;
        }
    }
}

int main() {

    while (~scanf("%s%s", s, t)) {
        lenS = strlen(s), lenT = strlen(t);
        exKMP();

    }

    return 0;
}
```

AC 自动机

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

using namespace std;
typedef long long ll;
const int maxn = 200005;
const int maxm = 2000005;
const int inf = 0x3f3f3f3f;

int n, tot;
char s[maxm];

struct AC_auto {
    int fail;
    int nxt[26];
    int end;
    int last;   //上一个结束节点
} ac[maxn];

inline void newNode() {
    tot++;
    memset(ac[tot].nxt, 0, sizeof(ac[tot].nxt));
    ac[tot].end = ac[tot].fail = 0;
    ac[tot].last = 0;
}
```

```cpp

void init() {
    tot = 0; newNode();
}

void Build() {
    int len = strlen(s);
    int now = 1;
    for (int i = 0; i < len; i++) {
        int ch = s[i] - 'a';
        if (ac[now].nxt[ch] == 0) {
            newNode();
            ac[now].nxt[ch] = tot;
        }
        now = ac[now].nxt[ch];
    }
    ac[now].end++;
}

void GetFail() {
    for (int i = 0; i < 26; i++) ac[0].nxt[i] = 1;
    ac[1].fail = 0;

    queue<int> q;
    q.push(1);
    while (!q.empty()) {
        int u = q.front(); q.pop();

        for (int i = 0; i < 26; i++) {
            int v = ac[u].nxt[i];
            int failTo = ac[u].fail;
            if (!v) {
                ac[u].nxt[i] = ac[failTo].nxt[i];
                continue;
            }

            while (failTo && !ac[failTo].nxt[i]) failTo = ac[failTo].fail;

            failTo = ac[failTo].nxt[i];
            ac[v].fail = failTo;
            ac[v].last = ac[failTo].end ? failTo : ac[failTo].last;

            q.push(v);
        }
    }
}

//匹配成功时操作函数 Count()
inline void Count(int now) {
    while (now) {
        //计数、打印，视题目要求改动此处

        now = ac[now].last;
    }
}

void Query() {
    int len = strlen(s), now = 1;
    for (int i = 0; i < len; i++) {
        int ch = s[i] - 'a';
        now = ac[now].nxt[ch];

        if (ac[now].end) Count(now);
        else if (ac[now].last) Count(ac[now].last);
    }
}

void solve() {
    //计数、打印，视题目要求改动此处

}
```

```cpp

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
#endif

    while (~scanf("%d", &n)) {
        if (n == 0) break;
        init();

        for (int i = 0; i < n; i++) {
            scanf("%s", s);
            Build();
        }
        GetFail();

        scanf("%s", s);
        Query();

        solve();
    }

    return 0;
}
```

Trie 图

```cpp
//trie 图记每个模板串出现次数
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int maxn = 200005;
const int maxm = 2000005;
const int inf = 0x3f3f3f3f;

int n, tot;
char s[maxm];

struct AC_auto {
    int fail;
    int nxt[26];
    int end;
    int last;   //上一个结束节点

    vector<int> mp;
    vector<int> id;
    int cnt;
} ac[maxn];

int ans[maxn];

inline void newNode() {
    tot++;
    memset(ac[tot].nxt, 0, sizeof(ac[tot].nxt));
    ac[tot].end = ac[tot].fail = 0;
    ac[tot].last = 0;
    ac[tot].id.clear();
    ac[tot].mp.clear();
    ac[tot].cnt = 0;
}

void init() {
    tot = 0; newNode();
    for (int i = 0; i < n; i++) ans[i] = 0;
}

void Build(int id) {
    int len = strlen(s);
    int now = 1;
```

```
    for (int i = 0; i < len; i++) {
        int ch = s[i] - 'a';
        if (ac[now].nxt[ch] == 0) {
            newNode();
            ac[now].nxt[ch] = tot;
        }
        now = ac[now].nxt[ch];
    }
    ac[now].end = 1;
    ac[now].id.push_back(id);
}

void GetFail() {
    for (int i = 0; i < 26; i++) ac[0].nxt[i] = 1;
    ac[1].fail = 0;

    queue<int> q;
    q.push(1);
    while (!q.empty()) {
        int u = q.front(); q.pop();

        for (int i = 0; i < 26; i++) {
            int v = ac[u].nxt[i];
            int failTo = ac[u].fail;
            if (!v) {
                ac[u].nxt[i] = ac[failTo].nxt[i];
                continue;
            }

            while (failTo && !ac[failTo].nxt[i]) failTo = ac[failTo].fail;

            failTo = ac[failTo].nxt[i];
            ac[v].fail = failTo;
            ac[v].last = ac[failTo].end ? failTo : ac[failTo].last;

            if (ac[v].end) ac[ac[v].last].mp.push_back(v);

            q.push(v);
        }
    }
}

inline void Count(int now) {
    ac[now].cnt++;
}

void Query() {
    int len = strlen(s), now = 1;
    for (int i = 0; i < len; i++) {
        int ch = s[i] - 'a';
        now = ac[now].nxt[ch];

        if (ac[now].end) Count(now);
        else if (ac[now].last) Count(ac[now].last);
    }
}

int dfs(int u) {
    int size = ac[u].mp.size();
    int sum = 0;
    for (int i = 0; i < size; i++) {
        int v = ac[u].mp[i];
        sum += dfs(v);
    }

    size = ac[u].id.size();
    for (int i = 0; i < size; i++) {
        int id = ac[u].id[i];
        ans[id] = sum + ac[u].cnt;
    }

```

```
    return sum + ac[u].cnt;
}

void solve() {
    //计数、打印，视题目要求改动此处
    for (int i = 0; i < n; i++) printf("%d\n", ans[i]);
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
#endif

    while (scanf("%d", &n) != EOF) {
        if (n == 0) break;
        init();

        for (int i = 0; i < n; i++) {
            scanf("%s", s);
            Build(i);
        }
        GetFail();

        scanf("%s", s);
        Query();
        dfs(0);

        solve();
    }

    return 0;
}
```

Manacher

二：算法过程分析

由于回文分为偶回文（比如 bccb）和奇回文（比如 bcacb），而在处理奇偶问题上会比较繁琐，所以这里我们使用一个技巧，在字符插入一个字符（前提这个字符未出现在串里）。举个例子：s="abbahopxpo"，转换为s_new="$#a#b#b#a#h#o#p#x#p#o#"（这里的字符 $ 只是为了防止越界，下面代码会有说明），如此，s 里起初有一个偶回文abba和一个奇回文opxpo，被转换为#a#b#b#a#和 #o#p#x#p#o#，长度都转换成了奇数。

定义一个辅助数组int p[]，p[i]表示以s_new[i]为中心的最长回文的半径，例如：

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
s_new[i]	$	#	a	#	b	#	b	#	a	#	h	#	o	#	p	#	x	#	p	#
p[i]		1	2	1	4	5	2	1	2	1	2	1	2	1	2	1	6	1	2	1

可以看出，p[i]-1正好是原字符串中最长回文串的长度。

Manacher 算法之所以快，就快在对 p 数组的求法上有个捷径，看下图：

设置两个变量，mx 和 id 。

mx 代表以s_new[id]为中心的最长回文最右边界，也就是mx=id+p[id]。

假设我们现在求p[i]，也就是以s_new[i]为中心的最长回文半径，如果i<mx，如上图，那么：

```
if (i < mx)
        p[i] = min(p[2 * id - i], mx - i);
```

2 * id -i其实就是等于 j，p[j]表示以s_new[j]为中心的最长回文半径，见上图，因为 i 和 j 关于 id 对称，我们利用p[j]来加快查找。

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

typedef long long ll;
using namespace std;
const int maxn = 100005;
const int inf = 0x3f3f3f3f;

char s[maxn];
char ss[maxn * 2];
int p[maxn * 2];

int init() {
    int len = strlen(s);
    ss[0] = '$', ss[1] = '#';
    int j = 2;
    for (int i = 0; i < len; i++) {
        ss[j++] = s[i];
        ss[j++] = '#';
    }
    ss[j] = '\0';

    return j;
}

void manacher() {
    int len = init();

    int id;
    int mx = 0;
    for (int i = 1; i < len; i++) {
```

```cpp
        if (i < mx) {
            p[i] = min(p[2 * id - i], mx - i);
        } else {
            p[i] = 1;
        }

        while (ss[i - p[i]] == ss[i + p[i]]) {
            p[i]++;
        }

        if (i + p[i] > mx) {
            id = i;
            mx = i + p[i];
        }
    }

    for (int i = 1; i < len; i++) {
        p[i]--;
    }
}

int main() {

    scanf("%s", s);
    manacher();

    return 0;
}
```

回文自动机

```cpp
//回文自动机求所有本质不同的回文串
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int maxn = 300005;

struct PamNode {
    int len, cnt, fail;
    int son[26];
}pam[maxn];

char s[maxn];
int last, tot;

inline void newNode(int len) {
    pam[tot].len = len, pam[tot].cnt = 0;
    for (int i = 0; i < 26; i++) pam[tot].son[i] = 0;

    tot++;
}

void init() {
    last = 0, tot = 0;
    newNode(0), newNode(-1);
    pam[0].fail = 1;
}

inline int GetFail(int now, int i) {
    while (s[i - pam[now].len - 1] != s[i]) now = pam[now].fail;
    return now;
}

void PAM() {
    init();

    int len = strlen(s + 1);
    for (int i = 1; i <= len; i++) {
        int ch = s[i] - 'a';
        last = GetFail(last, i);
```

```cpp
            if (!pam[last].son[ch]) {
                newNode(pam[last].len + 2);
                int x = GetFail(pam[last].fail, i);
                pam[tot - 1].fail = pam[x].son[ch];

                pam[last].son[ch] = tot - 1;    //必须先求 fail 再连接节点
            }
            last = pam[last].son[ch];

            pam[last].cnt++;
        }
        for (int i = tot - 1; i >= 0; i--) pam[pam[i].fail].cnt += pam[i].cnt;
}

int main() {

    while (scanf("%s", s + 1) != EOF) {
        PAM();
    }
    return 0;
}
```

后缀数组

求本质不同的子串的数量

第 i 个后缀对答案的贡献为 $len - sa_i + 1 - height_i$

$ans = len * (len + 1)/2 - \sum_{i=2}^{n} height_i$

求长度在 [0, p] 区间的本质不同的子串的数量

第 i 个后缀对答案的贡献为 $max(0, min(p, len - sa_i + 1) - height_i)$

求一个串中最长的不可重叠的重复子串

求一个串中最长的不可重叠的重复子串，二分答案长度 k，对所有后缀分组，每组的 LCP >= k，检查每组是否有不重叠串

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

typedef long long ll;
using namespace std;
const int maxn = 1000005;
const int mod = 1e9 + 7;

char s[maxn];
int len, sz, rak[maxn], last[maxn], sa[maxn], tax[maxn], tp[maxn];

int height[maxn], H[maxn]; //height[i] = lcp(sa[i], sa[i - 1]), H[i] = height[rak[i]].

void Qsort() {
    for (int i = 0; i <= sz; i++) tax[i] = 0;
    for (int i = 1; i <= len; i++) tax[rak[i]]++;
    for (int i = 1; i <= sz; i++) tax[i] += tax[i - 1];
    for (int i = len; i >= 1; i--) sa[tax[rak[tp[i]]]--] = tp[i];
}

void SuffixSort() {
    sz = 1005;    //根据桶的大小修改初值
    for (int i = 1; i <= len; i++) rak[i] = s[i], tp[i] = i;
    Qsort();

    for (int step = 1, p = 0; p < len; sz = p, step <<= 1) {
        p = 0;
        for (int i = 1; i <= step; i++) tp[++p] = len - step + i;
        for (int i = 1; i <= len; i++) if (sa[i] > step) tp[++p] = sa[i] - step;
        Qsort();

        swap(rak, last);
        rak[sa[1]] = p = 1;
```

```
        for (int i = 2; i <= len; i++) {
            if (last[sa[i - 1]] == last[sa[i]] && last[sa[i - 1] + step] == last[sa[i] + step]) {
                rak[sa[i]] = p;
            } else {
                rak[sa[i]] = ++p;
            }
        }
    }
}

void GetHeight() {
    int k = 0;
    for (int i = 1; i <= len; i++) {
        if (k) k--;
        int j = sa[rak[i] - 1];
        while (s[i + k] == s[j + k]) k++;
        H[i] = height[rak[i]] = k;
    }
}

int main() {

    while (~scanf("%s", s + 1)) {
        len = strlen(s + 1);
        SuffixSort();
        GetHeight();

    }

    return 0;
}
```