Standard Code Library

F0RE1GNERS

East China Normal University

September 9102

Contents

一切的开始 **4**
宏定义 . . . 4
快速读 . . . 4
快速输出 . . . 7
对拍 . . . 7
为什么 C++ 不自带这个? . . . 8

数据结构 **8**
树状数组 . . . 8
normal . . . 8
线段树 . . . 8
单点 . . . 8
lazy . . . 9
pbds . . . 10
hash . . . 10
tree . . . 11
ST 表 . . . 11
standard . . . 11

数学 **12**
第二类斯特林数 . . . 12
基础 . . . 12
亚线性筛 . . . 14
min_25 . . . 14
min25 cheat sheets . . . 14
杜教筛 . . . 14
素数测试 . . . 15
Pollard-Rho . . . 15
BM 线性递推 . . . 15
扩展欧几里得 . . . 16
类欧几里得 . . . 17
逆元 . . . 17
组合数 . . . 17
斯特灵数 . . . 19
第一类斯特灵数 . . . 19
第二类斯特灵数 . . . 19
FFT & NTT & FWT . . . 19
NTT . . . 19
FWT . . . 19
simpson 自适应积分 . . . 21
快速乘 . . . 21
原根 . . . 21
公式 . . . 21
一些数论公式 . . . 21
一些狄利克雷卷积 . . . 22
一些数论函数求和的例子 . . . 22
斐波那契数列性质 . . . 22
常见生成函数 . . . 23
佩尔方程 . . . 23
Burnside & Polya . . . 23
皮克定理 . . . 23
莫比乌斯反演 . . . 24
二项式反演 . . . 24
低阶等幂求和 . . . 24
一些组合公式 . . . 24

二次剩余 24
中国剩余定理 25
伯努利数和等幂求和 25
拉格朗日插值法 26
单纯形 26
BSGS 27
exBSGS 28
数论分块 29
约数个数和 29
博弈 29
多项式大餐 X 线性递推式 29
多项式辗转相除法 33
法雷序列 33
线性代数 34
线性基 34
高斯消元 37

图论 **38**
树 Hash 38
矩阵树定理 39
最短路 39
最小割树 41
全局最小割 StoerWagner 42
LCA 45
树上差分 45
网络流 46
带下界网络流: 49
树链剖分 51
二分图匹配 52
欧拉路径 53
强连通分量与 2-SAT 53
一般图匹配 54
Tarjan 56
简单缩环 56
割点 57
桥 57
强连通分量缩点 57
点双连通分量 / 广义圆方树 58
圆方树 58
最小树形图 59
差分约束 60
三元环、四元环 61
四元环 61
三元环 62
支配树 62
边双连通分量 63

计算几何 **64**
二维几何:点与向量 64
象限 65
线 65
点与线 65
线与线 66
多边形 66
面积、假凸包(字典序排序) 66
真凸包(极角排序) 67
旋转卡壳 68

半平面交 68
圆 69
三点求圆心 69
圆线交点、圆圆交点 69
圆圆位置关系 70
圆与多边形交 70
圆的离散化、面积并 70
最小圆覆盖 73
圆的反演 73
三维计算几何 73
旋转 74
线、面 74
三维凸包 75

字符串 **76**
SA 76
KMP 77
Manacher 77
Trie 77

杂项 **78**
日期 78
子集枚举 78
Java 79
Regex 79
Decimal Format 79
Sort 81
扩栈（本地使用） 81
心态崩了 81
心态崩了第二季 82

一切的开始

宏定义

- 需要 C++11

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

- 更多配色:
  - 33 黄色
  - 34 蓝色
  - 31 橙色
- POJ/BZOJ version

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <string>
#include <vector>
#include <set>
#include <queue>
#include <cstring>
#include <cassert>
using namespace std;
typedef long long LL;
#define FOR(i, x, y) for (LL i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (LL i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(args...) do { cout << "\033[32;1m" << #args<< " -> "; err(args); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<typename T, typename... Args>
void err(T a, Args... args) { cout << a << ' '; err(args...); }
#else
#define dbg(...)
#endif
// -----------------------------------------------------------------------------
```

- HDU Assert Patch

```cpp
#ifdef ONLINE_JUDGE
#define assert(condition) if (!(condition)) { int x = 1, y = 0; cout << x / y << endl; }
#endif
```

快速读

```cpp
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
}
template <typename T>
bool rn(T& v) {
    static char ch;
    while (ch != EOF && !isdigit(ch)) ch = nc();
    if (ch == EOF) return false;
    for (v = 0; isdigit(ch); ch = nc())
        v = v * 10 + ch - '0';
    return true;
}

template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) { putchar('0'); return; }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

- 支持负数
- 支持运算符

```
struct ios {
    inline char read(){
        static const int IN_LEN=1<<18|1;
        static char buf[IN_LEN],*s,*t;
        return (s==t)&&(t=(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t?-1:*s++;
    }

    template <typename _Tp> inline ios & operator >> (_Tp&x){
        static char c11,boo;
        for(c11=read(),boo=0;!isdigit(c11);c11=read()){
            if(c11==-1)return *this;
            boo|=c11=='-';
        }
        for(x=0;isdigit(c11);c11=read())x=x*10+(c11^'0');
        boo&&(x=-x);
        return *this;
    }

    int read(char *s) {
        int len = 0;
        char ch;
        for (ch=read(); ch=='\n' || ch == ' '; ch=read());
        if (ch == -1) {
            s[len] = 0;
            return -1;
        }
        for (; ch!='\n' && ch != ' ' && ch != -1;ch=read())
            s[len++] = ch;
        s[len] = 0;
        return len;
    }
} io;
```

万能流读入

```
struct ios {
    static const int IN_LEN=1<<18|1;
    char buf[IN_LEN],*s,*t;
    inline char read(){
        return (s==t)&&(t=(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t?-1:*s++;
    }
    inline bool isEOF() {
        return (s==t)&&(t=(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t;
    }
    inline ios & operator >> (int &x){
        static char c11,boo;
        for(c11=read(),boo=0;!isdigit(c11);c11=read()){
            if(c11==-1)return *this;
            boo|=c11=='-';
        }
        for(x=0;isdigit(c11);c11=read())x=x*10+(c11^'0');
        boo&&(x=-x);
        return *this;
    }

```

```cpp
    inline ios & operator >> (LL &x){
        static char c11,boo;
        for(c11=read(),boo=0;!isdigit(c11);c11=read()){
            if(c11==-1)return *this;
            boo|=c11=='-';
        }
        for(x=0;isdigit(c11);c11=read())x=x*10+(c11^'0');
        boo&&(x=-x);
        return *this;
    }

    inline ios &operator >> (char *s) {
        int len = 0;
        char ch;
        for (ch=read(); ch=='\n' || ch == ' '; ch=read());
        if (ch == -1) {
            s[len] = 0;
            return *this;
        }
        for (; ch!='\n' && ch != ' ' && ch != -1;ch=read())
            s[len++] = ch;
        s[len] = 0;
        return *this;
    }

   inline ios &operator>>(double &x)
    {

        char ch;
        bool neg = false, dec = false;
        double now = 0.1;
        for (ch=read(); !isdigit(ch) && (ch!='.' && ch!='-') && ch!=-1; ch=read());

        if (ch == '-') neg = true;
        else if (ch == '.') { x = 0; dec = true; }
        else if (ch != -1) x = ch-'0';
        else return *this;
        if (!dec) {
            for (ch=read(); isdigit(ch) && ch!=-1; ch=read()) {
                x = x * 10 + ch-'0';
            }
        }

        if (ch == '.')
            for (ch=read(); isdigit(ch) && ch!=-1; ch=read()) {
                x += now * (ch - '0'); now *= 0.1;
            }
        if (neg) x = -x;

        return *this;
    }

    inline ios &operator>>(long double &x)
    {

        char ch;
        bool neg = false, dec = false;
        double now = 0.1;
        for (ch=read(); !isdigit(ch) && (ch!='.' && ch!='-') && ch!=-1; ch=read());

        if (ch == '-') neg = true;
        else if (ch == '.') { x = 0; dec = true; }
        else if (ch != -1) x = ch-'0';
        else return *this;
        if (!dec) {
            for (ch=read(); isdigit(ch) && ch!=-1; ch=read()) {
                x = x * 10 + ch-'0';
            }
        }

        if (ch == '.')
```

```
            for (ch=read(); isdigit(ch) && ch!=-1; ch=read()) {
                x += now * (ch - '0'); now *= 0.1;
            }
        if (neg) x = -x;

        return *this;
    }
} io;
```

快速输出

- 别忘了 flush

```
namespace output {
    const int OutputBufferSize = 1e6+5;

    char buffer[OutputBufferSize];
    char *s = buffer;
    inline void flush() {
        fwrite(buffer, 1, s-buffer, stdout);
        s = buffer;
        fflush(stdout);
    }
    inline void print(const char ch) {
        if (s-buffer>OutputBufferSize-2) flush();
        *s++ = ch;
    }
    inline void print(char *str) {
        while (*str!=0) print(char(*str++));
    }
    inline void print(int x) {
        char buf[25] = {0}, *p = buf;
        if (x<0) print('-'), x=-x;
        if (x == 0) print('0');
        while (x) *(++p) = x%10, x/=10;
        while (p != buf) print(char(*(p--)+'0'));
    }
}
```

对拍

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

- 快速编译运行

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

为什么 C++ 不自带这个?

```
LL bin(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = x * x % MOD)
        if (n & 1) ret = ret * x % MOD;
    return ret;
}
inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
```

数据结构

树状数组

normal

```
template<typename T>
struct Bit {
    inline int lowbit(int i) { return i & (-i); }

    T dat[N];

    void clear(int n) {
        memset(dat, 0, sizeof(T) * (n + 1));
    }

    inline void add(int i, T x) {
        for (; i<N; i+=lowbit(i))
            dat[i] += x;
    }

    inline T sum(int i) {
        T res = 0;
        for (; i; i-=lowbit(i))
            res += dat[i];
        return res;
    }
};
```

线段树

单点

- 注意把骚操作去掉

```
template<typename T>
struct node {
    int id;
    T v;
    node(int id=-1, T v=-0x3f3f3f3f) :id(id), v(v) {}

    bool operator < (const node &other) const {
        return v < other.v;
    }
};

template<typename T>
struct segT {
    node<T> dat[M << 2];
    int nn;

    inline void pu(int rt) {
        dat[rt] = max(dat[rt << 1], dat[rt << 1 | 1]);
    }

    void init(int n) {
        nn = 1;
        while (nn < n) nn <<= 1;
        for (int i=1; i<=n; ++i)
```

```cpp
            dat[i + nn - 1] = node<T>(i);
        for (int i=nn+n; i<2*nn; ++i)
            dat[i] = node<T>();
        for (int i=nn-1; i>=1; --i)
            pu(i);
    }

    inline void u(int i, T x) {
        i += nn;
        if (dat[i].v >= x)
            return;
        dat[i].v = x;
        for (i>>=1; i; i>>=1)
            pu(i);
    }

    int L, R;
    node<T> q(int l, int r, int rt) {
        // dbg(l, r, rt, L, R, dat[rt].v);
        if (L <= l && r <= R)
            return dat[rt];
        int m = (l+r) >> 1;
        node<T> v1, v2;
        if (L <= m) v1 = q(l, m, rt<<1);
        if (m+1<=R) v2 = q(m+1, r, rt<<1|1);
        return max(v1, v2);
    }

    inline node<T> q(int l, int r) {
        ++l; ++r;
        if (l > r)
            return node<T>();
        L = l; R = r;
        return q(1, nn, 1);
    }
};
```

lazy

```cpp
template<typename T>
struct snode {
    int id;
    T v;
    snode (int id=0, T v=0):id(id), v(v) {}

    bool operator < (const snode &other) const {
        if (v != other.v) return v < other.v;
        else return id < other.id;
    }
};

// template<typename T>
// typedef snode T;
template<typename T>
struct segT {
    T dat[N << 2];
    LL lazy[N << 2];
    int nn;

    void init(int n) {
        nn = 1;
        while (nn < n)
            nn <<= 1;

        for (int i=1; i<=n; ++i)
            dat[i+nn-1] = snode<int>(i, 0);
        for (int i=nn+n; i<2*nn; ++i)
            dat[i] = snode<int>(-1, -INF);
        for (int i=nn-1; i>=0; --i)
            pu(i);
    }

    inline void pd(int rt) {
        if (lazy[rt]) {
            int ls = rt << 1, rs = rt << 1 | 1;
            dat[ls].v = dat[rs].v = lazy[ls] = lazy[rs] = lazy[rt];
            lazy[rt] = 0;
        }
    }

    inline void pu(int rt) {
        dat[rt] = max(dat[rt<<1], dat[rt<<1|1]);
    }


    int L, R;

    void u(int l, int r, int rt, int v) {
        if (L <= l && r <= R) {
            dat[rt].v = lazy[rt] = v;
            return;
        }
        int m = (l+r) >> 1;
        pd(rt);
        if (L <= m) u(l, m, rt<<1, v);
        if (m+1<=R) u(m+1, r, rt<<1|1, v);
        pu(rt);
    }
    T q(int l, int r, int rt) {
        // dbg(l, r, rt, dat[rt].v);
        if (L <= l && r <= R) {
            return dat[rt];
        }
        int m = (l + r) >> 1; pd(rt);
        T v1 = snode<int>(-1, -INF), v2 = snode<int>(-1, -INF);
        if (L <= m) v1 = q(l, m, rt<<1);
        if (m+1<=R) v2 = q(m+1, r, rt<<1|1);
        pu(rt);
        return max(v1, v2);
    }
    void u(int l, int r, int x) {
        // dbg(l, r, x);
        L = l;
        R = r;
        u(1, nn, 1, x);
    }
    T q(int l, int r) {
        L = l;
        R = r;
        return q(1, nn, 1);
    }
};

segT<snode<int>> seg;
```

pbds

头文件:

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
```

hash

用于替代 `unordered_map` 整数插入操作 stl 3.6s, pbds 探测法 2s

```
gp_hash_table<int, int> mp;
cc_hash_table<int, int> mp2;
```

tree

1. 用来替代 `map`，第二个类型填 `null_type` 就是 `set`
2. key 会自动去重

example:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
tree<int, int, greater<int>, rb_tree_tag, tree_order_statistics_node_update> tr;

/*
int key 类型 int value 类型，可用 null_type 替代
less<int> 表示 key 从小到大，对应 greater<int>
rb_tree_tag 红黑树，可用 splay_tree_tag 替代
insert erase order_by_key find_by_order
lower_bound upper_bound
a.join(b) 将 b 并入 a
a.split(v, b) 将 key <= v 保留给 a，其他给 b
*/

int main(int argc, char const *argv[])
{
    tr.insert(make_pair(2, 2));
    tr.insert(make_pair(1, 1));
    tr.find(1)->second ++;
    for (auto x : tr)
        cout << x.first << ' ' << x.second << endl;
    puts("===");
    tr.insert(make_pair(1, 3));
    for (auto x : tr)
        cout << x.first << ' ' << x.second << endl;
    puts("===");
    tr.erase(1);
    for (int i=0; i<tr.size(); ++i)
    {
        auto p = tr.find_by_order(i);
        cout << p->first << ' ' << p->second << ' ' << tr.order_of_key(p->first) << endl;
    }
    puts("===");
    auto x = tr.lower_bound(1);
    cout << (x == tr.end()) << endl;
    return 0;
}
/* outcome
2 2
1 2
===
2 2
1 2
===
2 2 0
===
1
*/
```

ST 表

standard

```
int p2[10], Log[N];
struct ST {
    static const int SP = 10;
    int dat[SP][N];
    void init() {
        p2[0] = 1;
        for (int i=1; i<SP; ++i)
            p2[i] = p2[i-1] << 1;
        Log[1] = 0;
        for (int i=2; i<N; ++i)
            Log[i] = Log[i >> 1] + 1;
    }
    void init(int a[], int n)
    {
        for (int i=1; i<=n; ++i)
            dat[0][i] = a[i];
        for (int i=1; i<SP; ++i)
            for (int j=1; j<=n; ++j)
                dat[i][j] = min(dat[i-1][j], dat[i-1][j + p2[i-1]]);
    }
    int q(int l, int r) {
        int Lg = Log[r-l+1];
        return min(dat[Lg][l], dat[Lg][r-p2[Lg]+1]);
    }
} ds[N];
```

数学

第二类斯特林数

第二类 Stirling 数实际上是集合的一个拆分，表示将 n 个不同的元素拆分成 m 个集合的方案数

$$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(m-k)^n$$

```
LL Stirling2(int n, int m) {
    LL ans = 0;
    for (int k=0; k<=m; ++k) {
        LL del = C(m, k) * bin(m-k, n, MOD);
        if (k % 2) {
            ans = (ans - del) % MOD + MOD;
        } else {
            ans = (ans + del) % MOD;
        }
    }
    return ans;
}
```

基础

- 整除
- 素因子分解
- 因数分解

```
namespace math {

int mu[N], prime[N], pcnt, phi[N];
int minp[N];
int smu[N];
LL sphi[N];
void init() {
    mu[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (!minp[i]) {
            minp[i] = i;
            prime[pcnt++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < pcnt; ++j) {
            LL nextp = 1LL * i * prime[j];
            if (nextp >= N) break;
            if (!minp[nextp]) minp[nextp] = prime[j];
            if (i % prime[j] == 0) {
                mu[nextp] = 0;
                break;
            } else {
                mu[nextp] = -mu[i];
            }
        }
    }
    for (int i = 1; i < N; ++i) {
        smu[i] = smu[i - 1] + mu[i];
        sphi[i] = sphi[i - 1] + phi[i];
    }
}

template <typename T>
vector<pair<T, int>> primeFac(T x) {
    vector<pair<T, int>> res;
    if (x < N)
        while (x > 1) {
            int cnt = 0, p = minp[x];
            while (x % p == 0) {
                ++cnt;
                x /= p;
            }
            res.emplace_back(p, cnt);
        }
    else {
        for (int i = 0; i < pcnt; ++i) {
            int p = prime[i];
            if (1LL * p * p > x) break;
            if (x % prime[i] == 0) {
                int cnt = 0;
                while (x % prime[i] == 0) {
                    ++cnt;
                    x /= prime[i];
                }
                res.emplace_back(p, cnt);
            }
        }
        if (x > 1) res.emplace_back(x, 1);
    }
    return res;
}

template <typename T>
vector<T> fac(T x) {
    vector<T> res;
    for (int i = 1; 1LL * i * i <= x; ++i) {
        if (x % i == 0) {
            res.push_back(i);
            if (x / i != i) res.push_back(x / i);
        }
    }
    return res;
}

int getmu(int x) {
    if (x < N)
        return mu[x];
    int res = 1;
    auto vp = primeFac(x);
    for (const auto &p : vp) {
        if (p.second > 1)
            return 0;
        else
            res = -res;
    }
    return res;
}

```

```
} // namespace math
```

亚线性筛

min_25

- Weaver_zhu 版本，质因数个数

```
namespace min25 {
    const LL N = 1e6+5;
    LL id1[N], id2[N], w[N];
    LL n, B, m, pc;
    inline LL id(LL x) { return x > B? id2[n/x]:id1[x];}

    inline LL fp(LL x) { return 1;}
    inline LL sp(LL x) { return x%MOD-1; }
    inline LL fpk(LL p, LL k, LL pp) { return 0;}

    LL g[N], sg[N];

    LL S(LL n, int j) {
        LL ans = (g[id(n)] - sg[j] + MOD) % MOD;
        for (int k=j; k<pc; ++k) {
            LL p = prime[k]; if (p * p > n) break;
            for (LL pp=p, e=1; pp*p<=n; ++e, pp*=p) {
                LL del = S(n / pp, k+1);
                ans = (ans + fpk(p, e, pp) * del + fpk(p, e+1, pp*p)) % MOD;
            }
        }
        return ans;
    }

    LL solve(LL _n) {
        n = _n;
        B = sqrt(n + 0.5) + 1;
        m = 0;
        for (LL l=1, r, v; l<=n; l=r+1) {
            v = n / l; r = n / v;
            if (v <= B) id1[v] = m;
            else id2[r] = m;
            g[m] = sp(v);
            w[m++] = v;
        }
        pc = upper_bound(prime, prime+pcnt, B) - prime;
        for (int i=1; i<=pc; ++i) {
            sg[i] = (sg[i-1] + fp(prime[i-1])) % MOD;
        }

        for (int j=0; j<pc; ++j) {
            LL p = prime[j];
            for (int i=0; i<m; ++i) {
                if (p * p > w[i]) break;
                g[i] = (g[i] - fp(p) * (g[id(w[i]/p)] % MOD - sg[j])) % MOD + MOD;
            }
        }
        return S(n, 0);
    }
}
```

min25 cheat sheets

杜教筛

求 $S(n)=\sum_{i=1}^{n} f(i)$，其中 f 是一个积性函数。

构造一个积性函数 g，那么由 $(f*g)(n)=\sum_{d|n} f(d)g(\frac{n}{d})$，得到 $f(n)=(f*g)(n)-\sum_{d|n,d<n} f(d)g(\frac{n}{d})$。

当然，要能够由此计算 $S(n)$，会对 f,g 提出一些要求：

- $f*g$ 要能够快速求前缀和。

- g 要能够快速求分段和（前缀和）。
- 对于正常的积性函数 $g(1)=1$，所以不会有什么问题。

在预处理 $S(n)$ 前 $n^{\frac{2}{3}}$ 项的情况下复杂度是 $O(n^{\frac{2}{3}})$。

素数测试

- 前置：快速乘、快速幂
- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356
- http://miller-rabin.appspot.com/

```cpp
bool checkQ(LL a, LL n) {
    if (n == 2 || a >= n) return 1;
    if (n == 1 || !(n & 1)) return 0;
    LL d = n - 1;
    while (!(d & 1)) d >>= 1;
    LL t = bin(a, d, n);  // 不一定需要快速乘
    while (d != n - 1 && t != 1 && t != n - 1) {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
}

bool primeQ(LL n) {
    static vector<LL> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    if (n <= 1) return false;
    for (LL k: t) if (!checkQ(k, n)) return false;
    return true;
}
```

Pollard-Rho

- 大整数分解

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

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

BM 线性递推

- 杜教版

```cpp
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};
```

扩展欧几里得

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

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

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n)=\sum_{i=0}^n\lfloor\frac{ai+b}{c}\rfloor$: 当 $a \ge c$ or $b \ge c$ 时，$f(a,b,c,n)=(\frac{a}{c})n(n+1)/2+(\frac{b}{c})(n+1)+f(a \bmod c, b \bmod c, c, n)$；否则 $f(a,b,c,n)=nm-f(c,c-b-1,a,m-1)$。
- $g(a,b,c,n)=\sum_{i=0}^n i\lfloor\frac{ai+b}{c}\rfloor$: 当 $a \ge c$ or $b \ge c$ 时，$g(a,b,c,n)=(\frac{a}{c})n(n+1)(2n+1)/6+(\frac{b}{c})n(n+1)/2+g(a \bmod c, b \bmod c, c, n)$；否则 $g(a,b,c,n)=\frac{1}{2}(n(n+1)m-f(c,c-b-1,a,m-1)-h(c,c-b-1,a,m-1))$。
- $h(a,b,c,n)=\sum_{i=0}^n\lfloor\frac{ai+b}{c}\rfloor^2$: 当 $a \ge c$ or $b \ge c$ 时，$h(a,b,c,n)=(\frac{a}{c})^2n(n+1)(2n+1)/6+(\frac{b}{c})^2(n+1)+(\frac{a}{c})(\frac{b}{c})n(n+1)+h(a \bmod c, b \bmod c, c, n)+2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n)+2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；否则 $h(a,b,c,n)=nm(m+1)-2g(c,c-b-1,a,m-1)-2f(c,c-b-1,a,m-1)-f(a,b,c,n)$。

求两直线之间最小的 x, y 使得 $\frac{p1}{q1} < \frac{x}{y} < \frac{p2}{q2}$

```
void solve(LL p1,LL q1,LL p2,LL q2,LL &x,LL &y)
{
    LL l=p1/q1+1;
    if (l*q2<p2){x=l; y=1;return;}
    if (p1==0){x=1; y=q2/p2+1;return;}
    if (p1<=q1 && p2<=q2){solve(q2,p2,q1,p1,y,x);return;}
    LL t=p1/q1;
    solve(p1-q1*t,q1,p2-q2*t,q2,x,y);
    x+=y*t;
}
```

逆元

- 如果 p 不是素数，使用拓展欧几里得
- 前置模板：快速幂 / 扩展欧几里得

```
inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
LL get_inv(LL a, LL M) {
    static LL x, y;
    assert(ex_gcd(a, M, x, y) == 1);
    return (x % M + M) % M;
}
```

- 预处理 1~n 的逆元

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

组合数

- 如果数较小，模较大时使用逆元
- 前置模板：逆元-预处理阶乘及其逆元
- 卢卡斯定理
- 任意模数组合数（很慢慎用，基于 CRT）

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

- 如果模数较小，数字较大，使用 Lucas 定理
- 前置模板可选 1：求组合数（如果使用阶乘逆元，需 `fac_inv_init(MOD, MOD);`）
- 前置模板可选 2：模数不固定下使用，无法单独使用。

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

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

- 前置模板：素数，CRT

```
bool nop[MAXN];
vector<int> prime;

void get_prime() {
    // ...
}

vector<pair<int, int> > fac;

int cntfac(LL n, LL p) {
    int ans = 0;
    LL op = p;
    for (; n>=p; p*=op) {
        ans += n/p;
    }
    dbg("cntfac", n, op, ans);
    return ans;
}

LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    // ...
}

LL bin(LL a, LL b, LL p) {
    // ...
}

LL get_inv(LL a, LL M) {
    // ...
}

LL mo[MAXN], r[MAXN];

LL calcT(LL n, pair<int, int> p, LL M) {
    LL resT = 1;
    for (int i=1; i<=M; ++i) {
        if (i % p.first)
            resT = resT * i % M;
    }
    LL cntT = n / M, res = bin(resT, cntT, M), remain = n % M;
    dbg(cntT, resT, res, remain);
    for (int i=1; i<=remain; ++i) {
        if (i % p.first)
            res = res * i % M;
    }
    if (n>=p.first)
        res = res * calcT(n/p.first, p, M) % M;

    dbg("calcT", n, p.first, p.second, M, res);
    return res;
}

LL CRT(LL *m, LL *r, LL n) {
    // ...
}

LL lucas(int n, int m, LL p) {
    fac.clear();
    LL tp = p;
    int psiz = prime.size();
    FOR (j, 0, psiz) {
        if (1LL*prime[j]*prime[j] > tp) break;
        int cnt = 0;
        while (tp % prime[j] == 0) {
            tp /= prime[j];
            ++cnt;
        }
        fac.push_back(make_pair(prime[j], cnt));
    }
    if (tp != 1) fac.push_back(make_pair(tp, 1));

    int msiz = fac.size(); dbg(msiz);
    FOR (i, 0, msiz) {
        dbg(fac[i].first, fac[i].second);
        LL M = bin(fac[i].first, fac[i].second, MOD); // dont let it module

        mo[i] = M;
        int cntp = cntfac(n, fac[i].first) - cntfac(m, fac[i].first) - cntfac(n-m, fac[i].first);

        if (cntp >= fac[i].second) r[i] = 0;
        else {
            LL nM = bin(fac[i].first, fac[i].second - cntp, MOD); // dont let it module

           r[i] = calcT(n, fac[i], M) * get_inv(calcT(m, fac[i], M), M) % M * get_inv(calcT(n-m, fac[i], M), M) % M;
            dbg(r[i]);
            r[i] = r[i] * bin(fac[i].first, cntp, MOD) % M; // dont let it module
            dbg(nM, r[i]);
        }
        dbg(i, M, cntp, r[i]);
    }
    return CRT(mo, r, msiz);
}
```

斯特灵数

第一类斯特灵数

- 绝对值是 n 个元素划分为 k 个环排列的方案数。
- $s(n,k)=s(n-1,k-1)+(n-1)s(n-1,k)$

第二类斯特灵数

- n 个元素划分为 k 个等价类的方案数
- $S(n,k)=S(n-1,k-1)+kS(n-1,k)$

```
S[0][0] = 1;
FOR (i, 1, N)
    FOR (j, 1, i + 1) S[i][j] = (S[i - 1][j - 1] + j * S[i - 1][j]) % MOD;
```

FFT & NTT & FWT

NTT

已经退出历史舞台，详见多项式大餐

FWT

- $C_k=\sum_{i\oplus j=k} A_iB_j$
- FWT 完后需要先模一遍
- 对于 xor 运算，$FWT[i]=\sum\limits_{|j\&i|=0} a_j-\sum\limits_{|j\&i|=1} a_j=\sum\limits_{j}(-1)^{|j\&i|}$，$|j\&i|$ 表示 $i\&j$ 二进制 1 的个数的奇偶性
- 对于 or 运算，$FWT[i]=\sum\limits_{j|i=i} a[j]$，j 可以认为是 i 的二进制子集。
- 对于 and 运算，$FWT[i]=\sum\limits_{j\&i=i} a[i]$, i 可以认为是 j 的二进制子集

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

```
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

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

- 三进制异或 fwt

```
struct RC {
    int x, y;
    RC(int x=0, int y=0):x(x), y(y) {}
    inline RC operator + (const RC &other) {
        return RC(add(x+other.x), add(y+other.y));
    }
    inline RC operator - (const RC &other) {
        return RC(sub(x-other.x), sub(y-other.y));
    }

    /*
        attention: unusual definition!
        (a,b) = a+bw
        w^2+w+1=0
        therefore mul operator is different
        */
    inline RC operator * (const RC &other) {
        return RC(sub((1LL*x*other.x-1LL*y*other.y)%p), sub((1LL*x*other.y+1LL*y*other.x-1LL*y*other.y)%p));
    }
    static inline RC w31(const RC &a) {
        return RC(sub(-a.y), sub(a.x-a.y));
    }
    static inline RC w32(const RC &a) {
        return RC(sub(a.y-a.x), sub(-a.x));
    }
    bool operator < (const RC &other) const {
        if (x != other.x) return x < other.x;
        else return y < other.y;
    }
} f[MAXN<<2], g[MAXN<<2];
 void fwt(RC a[], bool rfwt = 0) { // n = 3 ^ m, wi 是单位负数根
    for (int d=1; d<n; d*=3) {
        for (int x=0; x<n; x+=3*d) {
            for (int i=x; i<x+d; ++i) {
                RC x = a[i], y = a[i+d], z = a[i+(d<<1)];
                a[i] = (1LL*x+y+z)%p;
                if (rfwt) {
                    a[i+d] = (1LL*x+1LL*W2*y+1LL*W1*z)%p;
                    a[i+(d<<1)] = (1LL*x+1LL*W1*y+1LL*W2*z)%p;
                } else {
                    a[i+d] = (x+1LL*W1*y+1LL*W2*z);
                    a[i+(d<<1)] = (1LL*x+1LL*W2*y+1LL*W1*z)%p;
                }
            }
        }
    }
    if (rfwt) {
        for (int i=0; i<n; ++i)
            a[i] =1LL* a[i] * invn % p;
    }
}
```

- k 进制异或 fwt
- 正变换矩阵 $A_{ij} = w_n^{ij}$ (0 based) $FWT[f] = Af$
- 逆变换矩阵 $A_{ij} = w_n^{-ij}$ (0 based) $rFWT[f] = Af$
- FWT 子集卷积

```
a[popcount(x)][x] = A[x]
b[popcount(x)][x] = B[x]
fwt(a[i]) fwt(b[i])
c[i + j][x] += a[i][x] * b[j][x]
rfwt(c[i])
ans[x] = c[popcount(x)][x]
```

simpson 自适应积分

```
LD simpson(LD l, LD r) {
    LD c = (l + r) / 2;
    return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
}

LD asr(LD l, LD r, LD eps, LD S) {
    LD m = (l + r) / 2;
    LD L = simpson(l, m), R = simpson(m, r);
    if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
    return asr(l, m, eps / 2, L) + asr(m, r, eps / 2, R);
}

LD asr(LD l, LD r, LD eps) { return asr(l, r, eps, simpson(l, r)); }
```

快速乘

- O(1)

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

原根

- 前置模板：素数筛，快速幂，分解质因数
- 要求 p 为质数

```
LL find_smallest_primitive_root(LL p) {
    get_factor(p - 1);
    FOR (i, 2, p) {
        bool flag = true;
        FOR (j, 0, f_sz)
            if (bin(i, (p - 1) / factor[j], p) == 1) {
                flag = false;
                break;
            }
        if (flag) return i;
    }
    assert(0); return -1;
}
```

公式

一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \bmod \phi(p) + \phi(p)} \pmod p$

- $\mu^2(n)=\sum_{d^2|n}\mu(d)$
- $\sum_{d|n}\varphi(d)=n$
- $\sum_{d|n}2^{\omega(d)}=\sigma_0(n^2)$，其中 ω 是不同素因子个数
- $\sum_{d|n}\mu^2(d)=2^{\omega(d)}$

一些狄利克雷卷积

- 满足交换律和结合律，$f*g$ 也是积性函数
- $id(n)=n\ I(n)=1$
- $\mu*1=[n==1]$ 枚举素因子种类
- 除数函数 $d(n)=\sum_{d|n}1=1*1$
- $\sigma(n)=\sum_{d|n}d=id*1$
- $\phi*1=id,\phi=\mu*id$
- $n=\prod_{i=1}^{t}p_i^{k_i},g(n)=f*1\Rightarrow g(n)=\prod_{i=1}^{t}\sum_{j=0}^{k_i}f(p_i^j)$ 相当于枚举素因数指数

一些数论函数求和的例子

- $\sum_{i=1}^{n}\sum_{d|i}f(d)=\sum_{i=1}^{n}\sum_{i=1}^{\lfloor\frac{n}{i}\rfloor}f(d)$ 计算 $f(d)$ 出现了多少次，杜教筛的核心思想
- $\sum\limits_{i=1}^{n}gcd(i,a)=\sum\limits_{d|a}d\sum\limits_{i=1}^{n}[gcd(i,a)==d]=\sum\limits_{d|a}d\sum\limits_{i=1}^{n}[gcd(i/d,a/d)==1]=\sum\limits_{d|a}d\sum\limits_{i=1}^{n}u*1(gcd(i/d,a/d))=$ $\sum\limits_{d|a}\sum\limits_{i}\sum_{t|gcd(i/d,a/d)}\mu(t)\ T=td\ ,\ d|T,=\sum\limits_{d}\sum\limits_{i}\sum_{T|gcd(a,d)}d\cdot\mu(T/d)=\sum\limits_{T|a}\frac{n}{T}\sum\limits_{d|T}d\mu(\frac{T}{d})=\sum\limits_{T|a}\frac{n}{T}\phi(T)$
- $\sum_{i=1}^{n}i[gcd(i,n)=1]=\frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=x]=\sum_{d}\mu(d)\lfloor\frac{n}{dx}\rfloor\lfloor\frac{m}{dx}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{m}gcd(i,j)=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|gcd(i,j)}\varphi(d)=\sum_{d}\varphi(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$
- $S(n)=\sum_{i=1}^{n}\mu(i)=1-\sum_{i=1}^{n}\sum_{d|i,d<i}\mu(d)\overset{t=\frac{i}{d}}{=}1-\sum_{t=2}^{n}S(\lfloor\frac{n}{t}\rfloor)$
  - 利用 $[n=1]=\sum_{d|n}\mu(d)$ 杜教筛
- $S(n)=\sum_{i=1}^{n}\varphi(i)=\sum_{i=1}^{n}i-\sum_{i=1}^{n}\sum_{d|i,d<i}\varphi(i)\overset{t=\frac{i}{d}}{=}\frac{i(i+1)}{2}-\sum_{t=2}^{n}S(\frac{n}{t})$
  - 利用 $n=\sum_{d|n}\varphi(d)$ 杜教筛
- $S(n)=\sum\limits_{i=1}^{n}i^2\phi(i)=\sum\limits_{i=1}^{n}i^3-\sum\limits_{i=2}^{n}i^2S(n/i)$ 杜教筛
- $\sum_{i=1}^{n}\mu^2(i)=\sum_{i=1}^{n}\sum_{d^2|n}\mu(d)=\sum_{d=1}^{\lfloor\sqrt{n}\rfloor}\mu(d)\lfloor\frac{n}{d^2}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{n}gcd^2(i,j)=\sum_{d}d^2\sum_{t}\mu(t)\lfloor\frac{n}{dt}\rfloor^2$
  $\overset{x=dt}{=}\sum_{x}\lfloor\frac{n}{x}\rfloor^2\sum_{d|x}d^2\mu(\frac{t}{x})$
- $\sum_{i=1}^{n}\varphi(i)=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}[i\perp j]-1=\frac{1}{2}\sum_{i=1}^{n}\mu(i)\cdot\lfloor\frac{n}{i}\rfloor^2-1$
- $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}ij[gcd(i,j)==d]=\sum\limits_{d=1}^{n}d\sum\limits_{i=1}^{n/d}\sum\limits_{j=1}^{n/d}ij[gcd(i,j)==1]=\sum\limits_{d=1}^{n}d\sum\limits_{i=1}^{n/d}i^2\phi(i)$ 然后后面的可以用杜教筛

斐波那契数列性质

- $F_{a+b}=F_{a-1}\cdot F_b+F_a\cdot F_{b+1}$
- $F_1+F_3+\dots+F_{2n-1}=F_{2n},F_2+F_4+\dots+F_{2n}=F_{2n+1}-1$
- $\sum_{i=1}^{n}F_i=F_{n+2}-1$
- $\sum_{i=1}^{n}F_i^2=F_n\cdot F_{n+1}$
- $F_n^2=(-1)^{n-1}+F_{n-1}\cdot F_{n+1}$
- $gcd(F_a,F_b)=F_{gcd(a,b)}$
- 模 n 周期（皮萨诺周期）
  - $\pi(p^k)=p^{k-1}\pi(p)$
  - $\pi(nm)=lcm(\pi(n),\pi(m)),\forall n\perp m$
  - $\pi(2)=3,\pi(5)=20$

  - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p-1$
  - $\forall p \equiv \pm 2 \pmod{5}, \pi(p) | 2p+2$

常见生成函数

- $(1+ax)^n = \sum_{k=0}^{n} \binom{n}{k} a^k x^k$
- $\dfrac{1-x^{r+1}}{1-x} = \sum_{k=0}^{n} x^k$
- $\dfrac{1}{1-ax} = \sum_{k=0}^{\infty} a^k x^k$
- $\dfrac{1}{(1-x)^2} = \sum_{k=0}^{\infty} (k+1) x^k$
- $\dfrac{1}{(1-x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $e^x = \sum_{k=0}^{\infty} \dfrac{x^k}{k!}$
- $\ln(1+x) = \sum_{k=0}^{\infty} \dfrac{(-1)^{k+1}}{k} x^k$

佩尔方程

若一个丢番图方程具有以下的形式：$x^2 - ny^2 = 1$。且 n 为正整数，则称此二元二次不定方程为**佩尔方程**。

若 n 是完全平方数，则这个方程式只有平凡解 $(\pm 1, 0)$（实际上对任意的 n，$(\pm 1, 0)$ 都是解）。对于其余情况，拉格朗日证明了佩尔方程总有非平凡解。而这些解可由 $\sqrt{n}$ 的连分数求出。

$$x = [a_0; a_1, a_2, a_3] = x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

设 $\frac{p_i}{q_i}$ 是 $\sqrt{n}$ 的连分数表示: $[a_0; a_1, a_2, a_3, ...]$ 的渐近分数列, 由连分数理论知存在 i 使得 (p_i, q_i) 为佩尔方程的解。取其中最小的 i, 将对应的 (p_i, q_i) 称为佩尔方程的基本解, 或最小解, 记作 (x_1, y_1), 则所有的解 (x_i, y_i) 可表示成如下形式: $x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$。或者由以下的递回关系式得到:

$x_{i+1} = x_1 x_i + n y_1 y_i, y_{i+1} = x_1 y_i + y_1 x_i$。

但是：佩尔方程千万不要去推（虽然推起来很有趣，但结果不一定好看，会是两个式子）。记住佩尔方程结果的形式通常是 $a_n = ka_{n-1} - a_{n-2}$（a_{n-2} 前的系数通常是 -1）。暴力 / 凑出两个基础解之后加上一个 0，容易解出 k 并验证。

Burnside & Polya

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$

注：X^g 是 g 下的不动点数量，也就是说有多少种东西用 g 作用之后可以保持不变。

- $|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$

注：用 m 种颜色染色，然后对于某一种置换 g，有 $c(g)$ 个置换环，为了保证置换后颜色仍然相同，每个置换环必须染成同色。

皮克定理

$2S = 2a + b - 2$

- S 多边形面积
- a 多边形内部点数
- b 多边形边上点数

莫比乌斯反演

- $g(n)=\sum_{d|n}f(d)\Leftrightarrow f(n)=\sum_{d|n}\mu(d)g(\frac{n}{d})$
- $f(n)=\sum_{n|d}g(d)\Leftrightarrow g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)$

二项式反演

- s 是非负整数 $a_n=\sum_{k=s}^{n}\binom{n}{k}b_k\Rightarrow b_n=\sum_{k=s}^{n}(-1)^{n-k}\binom{n}{k}a_k$

低阶等幂求和

- $\sum_{i=1}^{n}i^1=\frac{n(n+1)}{2}=\frac{1}{2}n^2+\frac{1}{2}n$
- $\sum_{i=1}^{n}i^2=\frac{n(n+1)(2n+1)}{6}=\frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n$
- $\sum_{i=1}^{n}i^3=\left[\frac{n(n+1)}{2}\right]^2=\frac{1}{4}n^4+\frac{1}{2}n^3+\frac{1}{4}n^2$
- $\sum_{i=1}^{n}i^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}=\frac{1}{5}n^5+\frac{1}{2}n^4+\frac{1}{3}n^3-\frac{1}{30}n$
- $\sum_{i=1}^{n}i^5=\frac{n^2(n+1)^2(2n^2+2n-1)}{12}=\frac{1}{6}n^6+\frac{1}{2}n^5+\frac{5}{12}n^4-\frac{1}{12}n^2$

一些组合公式

- 错排公式：$D_1=0,D_2=1,D_n=(n-1)(D_{n-1}+D_{n-2})=n!(\frac{1}{2!}-\frac{1}{3!}+\cdots+(-1)^n\frac{1}{n!})=\lfloor\frac{n!}{e}+0.5\rfloor$
- 卡塔兰数（n 对括号合法方案数，n 个结点二叉树个数，$n\times n$ 方格中对角线下方的单调路径数，凸 $n+2$ 边形的三角形划分数，n 个元素的合法出栈序列数）：$C_n=\frac{1}{n+1}\binom{2n}{n}=\frac{(2n)!}{(n+1)!n!}$

二次剩余

URAL 1132

```cpp
LL q1, q2, w;
struct P { // x + y * sqrt(w)
    LL x, y;
};

P pmul(const P& a, const P& b, LL p) {
    P res;
    res.x = (a.x * b.x + a.y * b.y % p * w) % p;
    res.y = (a.x * b.y + a.y * b.x) % p;
    return res;
}

P bin(P x, LL n, LL MOD) {
    P ret = {1, 0};
    for (; n; n >>= 1, x = pmul(x, x, MOD))
        if (n & 1) ret = pmul(ret, x, MOD);
    return ret;
}
LL Legendre(LL a, LL p) { return bin(a, (p - 1) >> 1, p); }

LL equation_solve(LL b, LL p) {
    if (p == 2) return 1;
    if ((Legendre(b, p) + 1) % p == 0)
        return -1;
    LL a;
    while (true) {
        a = rand() % p;
        w = ((a * a - b) % p + p) % p;
        if ((Legendre(w, p) + 1) % p == 0)
            break;
    }
    return bin({a, 1}, (p + 1) >> 1, p).x;
}

int main() {
    int T; cin >> T;
    while (T--) {
        LL a, p; cin >> a >> p;
        a = a % p;
        LL x = equation_solve(a, p);
        if (x == -1) {
            puts("No root");
        } else {
            LL y = p - x;
            if (x == y) cout << x << endl;
            else cout << min(x, y) << " " << max(x, y) << endl;
        }
    }
}
```

中国剩余定理

- 无解返回 -1
- 前置模板：扩展欧几里得

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

伯努利数和等幂求和

- 预处理逆元
- 预处理组合数
- $\sum_{i=0}^{n} i^k = \frac{1}{k+1}\sum_{i=0}^{k}\binom{k+1}{i}B_{k+1-i}(n+1)^i$.
- 也可以 $\sum_{i=0}^{n} i^k = \frac{1}{k+1}\sum_{i=0}^{k}\binom{k+1}{i}B^{+}_{k+1-i}n^i$。区别在于 $B^{+}_{1} = 1/2$。(心态崩了)

```
namespace Bernoulli {
    const int M = 100;
    LL inv[M] = {-1, 1};
    void inv_init(LL n, LL p) {
        FOR (i, 2, n)
            inv[i] = (p - p / i) * inv[p % i] % p;
    }

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

    LL B[M] = {1};
    void init() {
        inv_init(M, MOD);
        init_C(M);
        FOR (i, 1, M - 1) {
            LL& s = B[i] = 0;
            FOR (j, 0, i)
                s += C[i + 1][j] * B[j] % MOD;
            s = (s % MOD * -inv[i + 1] % MOD + MOD) % MOD;
        }
    }

    LL p[M] = {1};
    LL go(LL n, LL k) {
        n %= MOD;
        if (k == 0) return n;
        FOR (i, 1, k + 2)
            p[i] = p[i - 1] * (n + 1) % MOD;
        LL ret = 0;
        FOR (i, 1, k + 2)
            ret += C[k + 1][i] * B[k + 1 - i] % MOD * p[i] % MOD;
        ret = ret % MOD * inv[k + 1] % MOD;
        return ret;
    }
}
```

拉格朗日插值法

- 输入点值表示和幂次
- 输出多项式系数
- 运算为 MOD 的剩余系
- $O(k^2)$，如果插值在 $[0, k]$ 复杂度为 $O(k)$
- k 次多项式需要 $k+1$ 个插值
- 注意 **除法逆元去模**后为 0 的情况!!!（心态崩了

$$l_j = \prod_{i=0}^{k} \frac{x - x_j}{x_j - x_i} f(x) = \sum_{i=0}^{k} l_j y_j$$

```
namespace interlopation {
    LL go(LL y[], int n, LL x) {
        if (x < n) return y[x];
        x %= MOD;
        LL tot = 1;
        int hit = 0;
        for (int i=0; i<n; ++i) {
            if (x == i) {
                hit = 1;
            } else {
                tot = tot * (x + MOD -i) % MOD;
            }
        }
        LL ans = 0;
        // printf("%lld %d\n", tot, hit);

        for (int i=0; i<n; ++i) {
            LL del = 1;
            if (hit) {
                if (i == x) del = tot;
                else del = 0;
            } else {
                del = tot * get_inv(x+MOD-i, MOD) % MOD;
            }
            int sgn = (n-i-1) % 2 ? -1 : 1;
            del = (del * sgn + MOD) * invf[n-i-1] % MOD * invf[i] % MOD;
            ans = (ans + del * y[i]) % MOD;
        }
        return ans;
    }
}
```

单纯形

- uoj 97 分
- `simplex::init()` 和 `simplex::solve()`

```
namespace simplex {
const int N = MAXN;
const int M = N << 1;
const double eps = 1e-10,
INF = 1e100;
double a[M][M], v, ans[N];
int b[N], id[N<<1];

int n, m;

void pivot(int x, int y) {
    swap(id[n+x], id[y]);
    double t = a[x][y]; a[x][y] = 1;
    for (int i=0; i<=n; ++i) a[x][i] /= t;
    for (int i=0; i<=m; ++i) {
        if (i == x || sgn(a[i][y]) == 0) continue;
        t = a[i][y]; a[i][y] = 0;
        for (int j=0; j<=n; ++j) a[i][j] -= a[x][j] * t;
    }
}

int simplex() {
    while (1) {
        int x = 0, y = 0;
        for (int j=1; !y && j<=n; ++j) if (sgn(a[0][j]) == 1) y = j;
        if (!y) return 1;
        double minX = INF;
        for (int i=1; i<=m; ++i)
            if (sgn(a[i][y])==1 && minX > a[i][0] / a[i][y]) minX = a[i][0] / a[i][y], x = i;
        if (!x) return 0; // unbound
        pivot(x, y);
    }
}

int solve() {
// 1 means sol, 0 for unbound(INF), -1 for infeasible
    for (int i=1; i<=n; ++i) id[i] = i;
    while (1) {
        int x = 0, y = 0;
        for (int i=1; i<=m; ++i) if (sgn(a[i][0]) == -1 && (!x || rand()&1)) x = i;
        if (!x) {
            int res = simplex();
            if (res == 1) {
                v = -a[0][0];
                for (int i=1; i<=m; ++i) ans[id[n+i]] = a[i][0];
                for (int i=0; i<n; ++i) ans[i] = ans[i+1];
            }
            return res;
        }
        for (int j=1; j<=n; ++j) if (sgn(a[x][j]) == -1 && (!y || rand()&1)) y = j;
        if (!y) return -1; // infeasible
        pivot(x, y);
    }
}

void init(double _a[N][N], double b[N], double c[N], int _n, int _m) {
// _a[][], b[], c[] are 0-based
// m is the num of equations
// n is the num of variables, so m rows n cols
// _ax <= b / max(cx)
    n = _n; m = _m;
    for (int i=1; i<=n; ++i) a[0][i] = c[i-1];
    for (int i=1; i<=m; ++i)
    for (int j=1; j<=n; ++j)
        a[i][j] = _a[i-1][j-1];
    for (int i=1; i<=m; ++i)
        a[i][0] = b[i-1];
}

}
```

BSGS

- 针对多次询问，复杂度 $O(\sqrt{qn})$

```
namespace BSGS {
    LL a, p, m, n, q;
    unordered_map<LL, LL> mp;
    void init(LL _a, LL _p, LL _q = 1) {
        a = _a; p = _p;
        m = ceil(sqrt((double)p*_q+1.5));
        // dbg(m);
        n = ceil(1.0*p/m);
        // swap(n, m);
        // dbg(n, m);
        mp.clear();
        LL v = 1;
        for (int i=1; i<=m; ++i) {
            v = v * a % p;
            mp[v] = i;
        }
        q = v;
        dbg("init");
    }

    LL query(LL b) {
        dbg(q, bin(a, m, p));
        LL v = 1;
        LL invb = inv(b, p);
        dbg(invb);
        for (int i=1; i<=n; ++i) {
            v = v * q % p;
            LL tar = v * invb % p;
            if (mp.count(tar)) return i * m - mp[tar];
        }
        return -1;
    }
}
```

exBSGS

- 模数可以非素数

```
LL exBSGS(LL a, LL b, LL p) { // a^x = b (mod p)
    a %= p; b %= p;
    if (a == 0) return b > 1 ? -1 : b == 0 && p != 1;
    LL c = 0, q = 1;
    while (1) {
        LL g = __gcd(a, p);
        if (g == 1) break;
        if (b == 1) return c;
        if (b % g) return -1;
        ++c; b /= g; p /= g; q = a / g * q % p;
    }
    static map<LL, LL> mp; mp.clear();
    LL m = ceil(sqrt(p + 1.5))+10;
    LL v = 1;
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    /* 要求正数解
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    */
    FOR (i, 1, m + 1) {
        q = q * v % p;
        auto it = mp.find(q);
        if (it != mp.end()) return i * m - it->second + c;
    }
    return -1;
}
```

数论分块

$f(i)=\lfloor \frac{n}{i} \rfloor = v$ 时 i 的取值范围是 $[l,r]$。

```
for (LL l = 1, v, r; l <= N; l = r + 1) {
    v = N / l; r = N / v;
}
```

约数个数和

- $O(sqrt(n))$

```
namespace dsum {
LL sum(LL i) {
    LL x = i%MOD;
    x = (1+x)*x/2%MOD;
    return x;
}

LL solve(LL n) {
    LL tar = sqrt(n);
    LL ans = 0;
    FOR (i, 1, tar+1) {
        ans = (ans + i*((n/i)%MOD) %MOD) % MOD;
    }

    for (LL l=tar+1, r, v; l<=n; l=r+1) {
        v = n/l;
        r = n/v;
        ans = (ans + v * (sum(r) - sum(l-1) + MOD)%MOD) %MOD;
    }
    return ans;
}
}
```

博弈

- Nim 游戏：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或和非零。
- 阶梯 Nim 游戏：可以选择阶梯上某一堆中的若干颗向下推动一级，直到全部推下去。先手必胜条件是奇数阶梯的异或和非零（对于偶数阶梯的操作可以模仿）。
- Anti-SG：无法操作者胜。先手必胜的条件是：
  - SG 不为 0 且某个单一游戏的 SG 大于 1 。
  - SG 为 0 且没有单一游戏的 SG 大于 1。
- Every-SG：对所有单一游戏都要操作。先手必胜的条件是单一游戏中的最大 step 为奇数。
  - 对于终止状态 step 为 0
  - 对于 SG 为 0 的状态，step 是最大后继 step +1
  - 对于 SG 非 0 的状态，step 是最小后继 step +1
- 树上删边：叶子 SG 为 0，非叶子结点为所有子结点的 SG 值加 1 后的异或和。

尝试：

- 打表找规律
- 寻找一类必胜态（如对称局面）
- 直接博弈 dp
- 对局面进行离散化，然后严格按照 sg 定理（mex 函数）暴力求出 sg 函数找规律。然后尝试周期，注意周期应该忽略开头一小段（开头的用 dp 数组，太大的掐掉开头的按周期找）

多项式大餐 X 线性递推式

- MTT 基于毛子黑科技的任意模数 fft，链接 https://blog.csdn.net/lvzelong2014/article/details/80156989
- 加减乘除，模数乘
- 求逆
- 求线性递推式 $O(klogklogn)$
- bzoj4161

```cpp
#include <cstdio>
#include <cmath>
#include <cctype>
#include <algorithm>
#include <vector>

inline int read() {
    register int ret, cc, sign = 1;
    while (!isdigit(cc = getchar())){ sign = cc == '-' ? -1 : sign; }ret = cc-48;
    while ( isdigit(cc = getchar())) ret = cc-48+ret*10;
    return ret * sign;
}
inline void write(int x, char ch = '\n') {
    register int stk[20], tp;
    stk[tp = !x] = 0;
    while (x) stk[++tp] = x % 10, x /= 10;
    while (tp) putchar(stk[tp--] + '0');
    putchar(ch);
}


typedef std::vector<int> Poly;
const double PI = acos(-1.0);
inline int getrev(int);
inline int Qpow(int, int, int);
inline Poly Inverse(const Poly&);
inline void Read(Poly&);
inline void Print(const Poly&);
inline Poly operator * (const Poly&, const Poly&);
inline Poly operator - (const Poly&, const Poly&);
inline Poly operator + (const Poly&, const Poly&);
inline Poly operator / (const Poly&, const Poly&);
inline Poly operator % (const Poly&, const Poly&);

const int MAXN = 300010;
const int MOD = 1e9 + 7;
int g[MAXN];
int f[MAXN];
int N, K;
inline void mul_mod(int* a, int* b) {
    static int tmp[MAXN];
    for (int i = 0; i < K << 1; ++i) tmp[i] = 0;
    for (int i = 0; i < K; ++i)
        for (int j = 0; j < K; ++j)
            tmp[i+j] = (tmp[i+j] + 1ll*a[i]*b[j]%MOD) % MOD;
    for (int i = (K<<1)-1; i >= K; --i)
        for (int j = 1; j <= K; ++j)
            tmp[i-j] = (tmp[i-j] + 1ll*tmp[i]*g[j]%MOD) % MOD;
    for (int i = 0; i < K; ++i) a[i] = tmp[i];
}
Poly Tg, Rg;
inline Poly mul_mod(const Poly& lhs, const Poly& rhs) {
    Poly A = lhs * rhs, B = A;
    int len = K-1;
    std::reverse(A.begin(), A.end());
    A.resize(len);
    Poly C = A * Rg;
    C.resize(len);
    std::reverse(C.begin(), C.end());
    Poly D = B - Tg * C;
    D.resize(K);
    return D;
}
int a[MAXN];
int b[MAXN];
int main() {
#ifdef ARK
    freopen("3.in", "r", stdin);
#endif
    N = read(), K = read();
    Tg.resize(K+1), Rg.resize(K+1);
    for (int i = 1; i <= K; ++i) Tg[K-i] = Rg[K-i] = (MOD-read())%MOD;
    for (int i = 0; i <  K; ++i) f[i] = (read()+MOD)%MOD;
    Tg[K] = Rg[K] = 1;
    std::reverse(Rg.begin(), Rg.end());
    Rg.resize(K-1);
    Rg = Inverse(Rg);
    Poly A(K), B(K);
    A[0] = B[1] = 1;
    while (N) {
        if (N & 1) A = mul_mod(A, B);
        B = mul_mod(B, B); N >>= 1;
        //Print(B);
    }
    int ans = 0;
    for (int i = 0; i < K; ++i) ans = (ans + 1ll*A[i]*f[i]%MOD)%MOD;
    printf("%d\n", ans);
}

struct Complex {
    double x, y;
    Complex(double x = 0, double y = 0):x(x), y(y) { }
    inline Complex operator + (const Complex& rhs) const { return Complex(x + rhs.x, y + rhs.y); }
    inline Complex operator - (const Complex& rhs) const { return Complex(x - rhs.x, y - rhs.y); }
    inline Complex operator * (const Complex& rhs) const { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y *
 ↪  rhs.x); }
    inline Complex operator - () const { return Complex(-x, -y); }
    inline Complex operator ! () const { return Complex( x, -y); }
    void print() { printf("(%f, %f)\n", x, y); }
};

Complex W[2][MAXN];
int rev[MAXN];
int Inv[MAXN];
inline int getrev(int n) {
    int bln = 1, bct = 0;
    while (bln <= n) bln <<= 1, bct++;
    for (int i = 0; i < bln; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bct - 1));
    for (int i = 0; i < bln; ++i) {
        W[0][i] = W[1][(bln-i)&(bln-1)] = Complex(cos(2*PI*i/bln), sin(2*PI*i/bln));
    }
    return bln;
}
inline void getinv(int n) {
    Inv[1] = 1;
    for (int i = 2; i < n; ++i)
        Inv[i] = 1ll * (MOD - MOD / i) * Inv[MOD % i] % MOD;
}
inline void FFT(Complex* a, int n, int opt) {
    for (int i = 0; i < n; ++i)
        if (i < rev[i]) std::swap(a[i], a[rev[i]]);
    for (int i = 1, t = n >> 1; i < n; i <<= 1, t >>= 1) {
        for (int j = 0, p = (i << 1); j < n; j += p) {
            Complex *w = W[opt];
            for (int k = 0; k < i; ++k, w += t) {
                Complex x = a[j + k];
                Complex y = *w * a[j + k + i];
                a[j + k] = x + y;
                a[j + k + i] = x - y;
            }
        }
    }
    if (opt) for (int i = 0; i < n; ++i) a[i].x /= n, a[i].y /= n;
}
inline Poly MTT(const Poly& A, const Poly& B, int n) {
    static Complex a[MAXN], b[MAXN], c[MAXN], d[MAXN];
    for (size_t i = 0; i < A.size(); ++i) a[i] = Complex(A[i] & 32767, A[i] >> 15);
    for (int i = A.size(); i < n; ++i) a[i] = Complex();
    for (size_t i = 0; i < B.size(); ++i) b[i] = Complex(B[i] & 32767, B[i] >> 15);
    for (int i = B.size(); i < n; ++i) b[i] = Complex();
    FFT(a, n, 0), FFT(b, n, 0);
    for (int i = 0; i < n; ++i) {
        int j = (n - i) & (n - 1);
        c[i] = Complex(a[i].x + a[j].x, a[i].y - a[j].y) * 0.5 * b[i];
        d[i] = Complex(a[i].y + a[j].y, a[j].x - a[i].x) * 0.5 * b[i];
    }
    FFT(c, n, 1), FFT(d, n, 1);
    Poly C(n);
    for (int i = 0; i < n; ++i) {
        long long u = llround(c[i].x) % MOD, v = llround(c[i].y) % MOD;
        long long x = llround(d[i].x) % MOD, y = llround(d[i].y) % MOD;
        C[i] = (u + ((v + x) << 15) % MOD + (y << 30)) % MOD;
    }
    return C;
}
inline int Qpow(int a, int p = MOD - 2, int mod = MOD) {
    int ret = 1;
    p += (p < 0) * (mod - 1);
    while (p) {
        if (p & 1) ret = 1ll * ret * a % mod;
        a = 1ll * a * a % mod; p >>= 1;
    }
    return ret;
}
inline Poly operator * (const Poly& A, const Poly& B) {
    int len = A.size() + B.size() - 1;
    int bln = getrev(len);
    Poly C = MTT(A, B, bln);
    C.resize(len);
    return C;
}
inline Poly operator + (const Poly& lhs, const Poly& rhs) {
    Poly A(std::max(lhs.size(), rhs.size()));
    for (size_t i = 0; i < A.size(); ++i)
        A[i] = ((i < lhs.size() ? lhs[i] : 0) + (i < rhs.size() ? rhs[i] : 0)) % MOD;
    return A;
}
inline Poly operator - (const Poly& lhs, const Poly& rhs) {
    Poly A(std::max(lhs.size(), rhs.size()));
    for (size_t i = 0; i < A.size(); ++i)
        A[i] = ((i < lhs.size() ? lhs[i] : 0) - (i < rhs.size() ? rhs[i] : 0) + MOD) % MOD;
    return A;
}

inline Poly Inverse(const Poly& A) {
    Poly B(1, Qpow(A[0], MOD - 2));
    int n = A.size() << 1;
    for (int i = 2; i < n; i <<= 1) {
        Poly C(A);
        C.resize(i);
        B = Poly(1, 2) * B - C * B * B;
        B.resize(i);
    }
    B.resize(A.size());
    return B;
}
inline Poly operator / (const Poly& lhs, const Poly& rhs) {
    return lhs * Inverse(rhs);
}
inline Poly operator % (const Poly& lhs, const Poly& rhs) {
    Poly A(lhs), B(rhs);
    int len = A.size() - B.size() + 1;
    std::reverse(A.begin(), A.end());
    std::reverse(B.begin(), B.end());
    A.resize(len), B.resize(len);
    Poly C = A * Inverse(B);
    C.resize(len);
    std::reverse(C.begin(), C.end());
    Poly D = lhs - rhs * C;
    D.resize(rhs.size() - 1);
    return D;
}
```

```
inline void Read(Poly& A) {
    std::generate(A.begin(), A.end(), read);
}
inline void Print(const Poly& A) {
    for (size_t i = 0; i < A.size(); ++i)
        printf("%d ", A[i]);
    puts("");
}
```

多项式辗转相除法

注意取模的时候多项式为 0 的情况。

这种实现的方法多项式为空的表示是 vector 里面只有一个 0

```
inline Poly operator%(const Poly &lhs, const Poly &rhs) {
    if (sz(lhs) < sz(rhs))
        return lhs;
    if (sz(rhs) == 1)
        return Poly(1, 0);
    Poly res = lhs;
    while (sz(res) >= sz(rhs)) {
        Poly r(sz(res) - sz(rhs), 0);
        r.insert(r.end(), rhs.begin(), rhs.end());
        int c = 1LL*res.back() * bin(r.back(), MOD-2, MOD) % MOD;
        for (int &x : r) {
            x = 1LL*x*c%MOD;
        }
        res = res - r;
        normalize(res);
        dbg(sz(r), sz(rhs), sz(res));
    }
    return res;
}

Poly pow(Poly a, int b, const Poly p) {
    Poly res = {1};
    for (a = a % p; b; b >>= 1, a = a * a % p) {
        dbg(sz(a), sz(p));
        myassert(sz(p) != sz(a));
        if (b & 1) res = res * a % p;
    }
    return res;
}

Poly gcd(Poly a, Poly b) {
    while (sz(a) > 1 && a.back() == 0) a.pop_back();
    while (sz(b) > 1 && b.back() == 0) b.pop_back();
    while (!(sz(b) == 1 && b.back() == 0)) {
        Poly c = a % b;
        while (sz(c) > 1 && c.back() == 0) c.pop_back();
        a = b;
        b = c;
    }
    return a;
}
```

法雷序列

- 输出两分数之间最小分母的最简分数

```
Frac cal(const Frac &fa, const Frac &fb) {
    LLL a = fa.a, b = fa.b, c = fb.a, d = fb.b;
    LLL x = a/b+1;
    if (x*d < c) return Frac(x, 1);
    if (!a) return Frac(1, d/c+1);
    if (a<=b && c<=d) {
        Frac t = cal(Frac(d, c), Frac(b, a));
        swap(t.a, t.b);
        return t;
    }
    x = a/b;
    Frac t = cal(Frac(a-b*x, b), Frac(c-d*x, d));
    t.a += t.b*x;
    return t;
}
```

- 输出大于某分数，且分子分母小于 n 的最小最简分数

```
Frac gen(LL a, LL b, int n) {
    pair<LL, LL> l(0, 1), mid(1, 1), r(1, 0), res(-1, -1);
    LL x=a, y=b;
    while (x != y && max(mid.first, mid.second) <= n) {
        if (a*mid.second < b*mid.first)
            res = mid;
        if (x < y) {
            tie(l, mid, r) = make_tuple(l, make_pair(l.first+mid.first, l.second+mid.second), mid);
            y -= x;
        } else {
            tie(l, mid, r) = make_tuple(mid, make_pair(mid.first+r.first, mid.second+r.second), r);
            x -= y;
        }
    }
    return Frac(res.first, res.second);
}
```

线性代数

线性基

- 求第 k 大的异或子集
- HDU 3949

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

const int MAXN = 1e4+5;
typedef long long LL;

int n, m;

LL a[MAXN], q[MAXN];

const int MAXM = 63;

struct linear_basis {
    int cnt;
    bool zero;
    LL p[MAXM], P[MAXN];
    int pcnt;

    static int highbit(LL x) {
        for (int j=MAXM-1; j>=0; --j) {
            if ((x>>j) & 1) return j;
        }
        return -1;
    }

    bool exist(LL x) {
        for (int j=MAXM-1; j>=0; --j)
            x = min(x, x^p[j]);
        return x == 0;
    }

    void init(LL a[], int n) {
        cnt = 0;
        memset(p, -1, sizeof p);
        zero = false;
        for (int i=0; i<n; ++i) {
            if (!a[i]) zero = true;
            LL x = a[i];
            for (int j=MAXM-1; j>=0; --j)
```

```
                    if ((x>>j) & 1) {
                        if (~p[j]) x ^= p[j];
                        else {
                            p[j] = x;
                            ++cnt;
                            break;
                        }
                    }
                if (x == 0) zero = true;
            }
            pcnt = 0;
            for (int i=0; i<MAXM; ++i) {
                if (~p[i]) for (int j=i-1; ~j; --j) {
                    if (~p[j] && ((p[i]>>j) & 1))
                        p[i] ^= p[j];
                }
                if (~p[i]) P[pcnt++] = p[i];
            }
            // printf("%d\n", zero);
        }

        LL findkth(LL k) {
            if (zero) --k;
            if (k > (1LL<<pcnt)-1) return -1;
            LL res = 0;
            for (int i=0; i<MAXM; ++i)
                if ((k >> i) & 1) res ^= P[i];
            return res;
        }

    } lb;



    int main() {
        int kk = 0;
        int t;
        scanf("%d", &t);
        while (~scanf("%d", &n)) {
            ++kk;
            printf("Case #%d:\n", kk);
            for (int i=0; i<n; ++i) {
                scanf("%lld", &a[i]);
            }
            scanf("%d", &m);
            for (int i=0; i<m; ++i) {
                scanf("%lld", &q[i]);
            }

            lb.init(a, n);

            for (int i=0; i<m; ++i) {
                LL x = lb.findkth(q[i]);
                printf("%lld\n", x);
            }
        }

        return 0;
    }
```

线性基求交

```
struct Linear_Basis{
    //basis vector
    ll basis[40];
    bool droped = false;
    void clear(){
        memset(basis,0,sizeof basis);
    }
    void ins(ll x){
        bool drop = true;
        for (int i=31;i>=0;i--){
            if (x & (1ll<< i)){
                if (!basis[i]){basis[i] = x;drop = false;break;}
                x ^= basis[i];
            }
        }
        droped |= drop;
    }
    bool can(ll val){
        for (int i=31;i>=0;i--){
            if (val & (1ll << i)){
                if (basis[i])val ^= basis[i];
                else return false;
            }
        }
        return val == 0;
    }
    void debug(){
        cerr<<"--------"<<endl;
        for (int i=31;i>=0;i--){
            if (basis[i]){
                cerr<<basis[i]<<endl;
            }
        }
    }
    Linear_Basis (const Linear_Basis & other){
        for (int i=0;i<=31;i++)basis[i] = other.basis[i];
    }
    Linear_Basis(bool full = false){
        if (full){
            for (int i=31;i>=0;i--){
                basis[i] = 1ll << i;
            }
        }else clear();
    }
    Linear_Basis operator & (const Linear_Basis &other)const{
        Linear_Basis ret = other;
        Linear_Basis c,d;
        for (int i=31;i>=0;i--){
            d.basis[i] = 1ll << i;
        }
        for (int i=31;i>=0;i--){
            if (basis[i]){
                ll v = basis[i],k=0;
                bool can = true;
                for (int j=31;j>=0;j--){
                    if (v & (1ll << j)){
                        if (ret.basis[j]){
                            v ^= ret.basis[j];
                            k ^= d.basis[j];
                        }else{
                            can = false;
                            ret.basis[j] = v;
                            d.basis[j] = k;
                            break;
                        }
                    }
                }
                if (can){
                    ll v = 0;
                    for (int j=31;j>=0;j--){
                        if (k & (1ll << j)){
                            v ^= other.basis[j];
                        }
                    }
                    c.ins(v);
                }
            }
        }
        return c;
    }
} dat[MAXN];
```

高斯消元

- n - 方程个数，m - 变量个数，a 是 n * (m + 1) 的增广矩阵，free 是否为自由变量
- 返回自由变量个数，-1 无解
- 浮点数版本

```
typedef double LD;
const LD eps = 1E-10;
const int maxn = 2000 + 10;

int n, m;
LD a[maxn][maxn], x[maxn];
bool free_x[maxn];

inline int sgn(LD x) { return (x > eps) - (x < -eps); }

int gauss(LD a[maxn][maxn], int n, int m) {
    memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
    int r = 0, c = 0;
    while (r < n && c < m) {
        int m_r = r;
        FOR (i, r + 1, n)
            if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
        if (m_r != r)
            FOR (j, c, m + 1)
                 swap(a[r][j], a[m_r][j]);
        if (!sgn(a[r][c])) {
            a[r][c] = 0;
            ++c;
            continue;
        }
        FOR (i, r + 1, n)
            if (a[i][c]) {
                LD t = a[i][c] / a[r][c];
                FOR (j, c, m + 1) a[i][j] -= a[r][j] * t;
            }
        ++r; ++c;
    }
    FOR (i, r, n)
        if (sgn(a[i][m])) return -1;
    if (r < m) {
        FORD (i, r - 1, -1) {
            int f_cnt = 0, k = -1;
            FOR (j, 0, m)
                if (sgn(a[i][j]) && free_x[j]) {
                    ++f_cnt;
                    k = j;
                }
            if(f_cnt > 0) continue;
            LD s = a[i][m];
            FOR (j, 0, m)
                if (j != k) s -= a[i][j] * x[j];
            x[k] = s / a[i][k];
            free_x[k] = 0;
        }
        return m - r;
    }
    FORD (i, m - 1, -1) {
        LD s = a[i][m];
        FOR (j, i + 1, m)
            s -= a[i][j] * x[j];
        x[i] = s / a[i][i];
    }
    return 0;
}
```

- 数据

```
3 4
1 1 -2 2
```

```
2 -3 5 1
4 -1 1 5
5 0 -1 7
// many

3 4
1 1 -2 2
2 -3 5 1
4 -1 -1 5
5 0 -1 0 2
// no

3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 1 0 7
// one
```

图论

树 Hash

素数 Hash + 双保险

```
typedef unsigned long long ULL;
namespace treehash {
    vector<int> *G;
    int n;
    using math::prime;
    const int mod = 998244353;
    inline ULL pack(ULL val, int sz) {
        return 2ull + 3ull * val + 7ull * prime[sz + 1];
    }

    int siz[N];

    ULL hashval[N], hashrt[N];
    ULL srchashval[N], srchashrt[N];
    inline int add(int x, int y) {
        x += y;
        return x >= mod ? x -= mod : x;
    }
    inline int sub(int x, int y) {
        x -= y;
        return x < 0 ? x += mod : x;
    }
    inline int mul(int x, int y) { return 1ll * x * y % mod; }

    void predfs(int u, int ff) {
        siz[u] = 1;
        hashval[u] = 1;
        int sz = 0;
        for (int v : G[u]) {
            if (v == ff) continue;
            predfs(v, u);
            sz++;
            siz[u] += siz[v];
            hashval[u] += hashval[v] * prime[siz[v]];
        }
        srchashval[u] = hashval[u];
        hashval[u] = pack(hashval[u], siz[u]);
    }
    void dfs(int u, int ff) {
        for (int v : G[u]) {
            if (v == ff) continue;
            ULL tmp = srchashrt[u] - hashval[v] * prime[siz[v]];
            tmp = pack(tmp, n - siz[v]);

            srchashrt[v] = srchashval[v] + tmp * prime[n - siz[v]];
            hashrt[v] = pack(srchashrt[v], n);
            dfs(v, u);
        }
    }

    vector<ULL> gethash(int _n, vector<int> _G[]) {
        n = _n;
        G = _G;
        vector<ULL> res(0);
        predfs(1, 0);
        hashrt[1] = hashval[1];
        srchashrt[1] = srchashval[1];
        dfs(1, 0);
        res.insert(res.begin(), hashrt, hashrt+n+1);
        return res;
    }
}
```

矩阵树定理

The following content is from Introduction to Linear Algebra-Wellesley-Cambridge Press (2016)

Let $G = (V, E)$ be an undirected finite graph with n vertices and m edges. We also consider weighted graphs with a weight function that assigns a non-negative real weight a_{vw} to each pair v, w of vertices. We require that the weights satisfy the following properties: * $a_{vw} > 0$ if $v, w \in E$, and $a_{vw} = 0$ if $v, w \notin E$ * $a_{vw} = a_{wv}$, for all $v, w \in V$

Unweighted graphs can be viewed as weighted graphs in which a_{vw} is the number of edges between v and w

In the unweighted case, the degree $\deg v$ of a vertex $v \in V$ is the number of edges of G incident with v. In the weighted case, it is defined by $\deg v = \sum\limits_{w \in V} a_{vw}$

Given a graph G, its (weighted) adjacency matrix $A(G) = (a_{vw})$ is the $n \times n$ matrix, with rows and columns indexed by V, whose entries are the edge-weights.

The degree matrix $D(G) = diag(\deg v : v \in V)$ is the diagonal matrix indexed by V with the vertex-degrees on the diagonal. The difference $L(G) = D(G) - A(G)$ is the Laplace matrix (or Laplacian) of G

Let G be a graph and let $L = L(G)$ be its Laplace matrix. If T is a spanning tree of G, let $a(T)$ be the product of the edgeweights of T . For any vertices v and w, the (weighted) number of spanning trees of G is

$$\pi(G) = \sum\{a(T) : \text{T is a spanning tree of G}\} = |\det L_v^w|$$

where L_v^w is the submatrix of L obtained by deleting the column corresponding to v and the row corresponding to w

最短路

```
namespace dijkstra {
typedef int Cost;
typedef pair<int, Cost> E;
typedef pair<int, int> P;

vector<E> G[N];
void addEdge(int u, int v, Cost w) {
    G[u].emplace_back(v, w);
    G[v].emplace_back(u, w);
}
template <typename T>
void dij(int s, int n, T d[], int t = 0) {
    // d[s] = 0;
    priority_queue< pair<T, int>, vector<pair<T, int>>, greater<pair<T, int>>> pq;
    pq.empace(0, s);
```

```cpp
    // sort();
    memset(d, -1, sizeof(d[0]) * (n + 1));
    d[s] = 0;
    while (!pq.empty()) {
        auto now = pq.top();
        pq.pop();
        if (now.second == t)
            return;
        int u = now.second;
        T dis = now.first;
        for (const auto &e : G[u]) {
            int v = e.first;
            Cost w = e.second;
            if (dis + w < d[v]) {
                d[v] = dis + w;
                pq.emplace(d[v], v);
            }
        }
    }
}
}  // namespace dijkstra
```

- spfa

```cpp
template<typename T>
struct E {
    int to;
    T cost;
    E(int to=0, T cost=0):to(to), cost(cost){}
};

template<typename T>
struct Spfa {
    int n;

    static const T INF = 0x3f3f3f3f;
    vector<vector<E<T>>> G;
    void init(int n) {
        this->n = n;
        G.resize(n+1);
        for (int i=0; i<=n; ++i)
            G[i].clear();
    }

    void addEdge(int u, int v, T cost) {
        // dbg(u, v, cost, n);
        assert(u <= n && v <= n);
        G[u].push_back(E<T>(v, cost));
    }

    void go(int s, T d[], int f[]) {
        fill(d, d+n+1, INF);
        d[s] = 0;
        queue<int> q;
        q.push(s);

        static bool in[N];
        memset(in, 0, sizeof(bool) * (n+1));
        in[s] = 1;
        f[s] = -1;

        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            in[u] = false;
            // dbg(u, d[u]);
            for (const auto &e : G[u]) {
                int v = e.to; T w = e.cost;
                if (d[u] + w < d[v]) {
                    d[v] = d[u] + w;
                    f[v] = u;
                    if (!in[v]) {
                            in[v] = 1;
                            q.push(v);
                        }
                    }
                }
            }
        }

    vector<int> findPath(int s, int t, T d[], int f[]) {
        vector<int> res;
        res.reserve(n);
        for (int v=t; v!=s; v=f[v]) {
            res.push_back(v);
        }
        res.push_back(s);
        reverse(res.begin(), res.end());
        return res;
    }
};
```

最小割树

从整个图开始每次随便选两个点进行网络流求最小割，割出来的的两个点集继续进行类似运算，随机选两个点再割直到只剩一个点位置。每次作最小割都是对整个图（原图）进行运算，相当与作 n 次 dinic。然后把两个点连带最小割的值当成树上要加的一条带权边。最小割树建好之后任意两点之间的最小割相当于在树上经过各个边的权值最小值

需要 dinic, 树上倍增（或 hld

- `vector<pair<int, int>> G[]` 为要建立的边
- PS：luogu 的数据太傻逼了，标号正常的连通图和 std 对拍了一万年（小数据和大数据）都是 ok 的

```
struct MinCutTree
{
    dinic::Dinic dc;

    bool deep[N];
    int *u, *v, *w, n, m;
    int node[N], tmp1[N], tmp2[N];

    vector<pair<int, int>> *G;

    void add(int u, int v, int w)
    {
        G[u].emplace_back(v, w);
        G[v].emplace_back(u, w);
    }

    void init(int n, int m, int u[], int v[], int w[], vector<pair<int, int>> G[])
    {
        this->n = n;
        this->m = m;
        this->u = u;
        this->v = v;
        this->w = w;
        this->G = G;
        dbg("mct.init", n, m);
        for (int i=0; i<m; ++i)
            dbg(i, u[i], v[i], w[i]);
    }

    void build(int l, int r)
    {
        if (l == r)
            return;
        int s = node[l], t = node[l + 1];
        dc.init(n, m, s, t, u, v, w);
        int res = dc.go();
        dc.cut(deep, 1, n);
        dbg(s, t, res);
        add(s, t, res);
        int tcnt1 = 0, tcnt2 = 0;
        for (int i = l; i <= r; ++i)
        {
            if (deep[node[i]])
                tmp1[tcnt1++] = node[i];
            else
                tmp2[tcnt2++] = node[i];
        }
        // int ncnt = 0;
        for (int i = 0; i < tcnt1; ++i)
            node[l + i] = tmp1[i];
        for (int i = 0; i < tcnt2; ++i)
            node[l + tcnt1 + i] = tmp2[i];
        build(l, l + tcnt1 - 1);
        build(l + tcnt1, r);
    }

    void go()
    {
        for (int i = 1; i <= n; ++i)
            node[i] = i;
        build(1, n);
    }
} mct;
```

全局最小割 StoerWagner

概念：

- 无向图的割：有无向图 $G = (V, E)$，设 C 为图 G 中一些弧的集合，若从 G 中删去 C 中的所有弧能使图 G 不是连通图，称 C 图 G 的一个割。
- $S - T$ 割：使得顶点 S 与顶点 T 不再连通的割，称为 $S - T$
- $S - T$ 最小割：包含的弧的权和最小的 $S - T$ 割，称为 $S - T$ 最小割。
- 全局最小割：包含的弧的权和最小的割，称为全局最小割。
- 诱导割 (induced cut)：令图 $G = (V, E)$ 的一个割为 C，则割 C 在图 G 的子图 $G' = (V', E')$ 中的部分称为割 C 的诱导割。（类似于概念诱导子图 (induced subgraph)）

流程

```
def MinimumCutPhase(G, w, a):
    A ← {a}
    while A ≠ V:
        把与 A 联系最紧密（most tightly）的顶点加入 A 中
    cut-of-the-phase ← w(A \ t, t)
    合并最后两个加入到 A 的顶点 s、t
    return cut-of-the-phase

def StoerWagner(G, w, a):
    while |V| > 1
        MinimumCutPhase(G, w, a)
        根据返回值更新最小割
```

代码

- $O(V^3)$ 常数不（太）大
- 邻接矩阵存储

```
const int N = 605;
int g[N][N];
namespace stoerwagner
{
    int id[N];
    int add(int g[N][N], int a, int m)
    {
        static int S[N], scnt;
        static int cmp[N];
        static bool in[N];

```

```cpp
        scnt = 0;
        memset(cmp, 0, sizeof(cmp[0]) * (m + 1));
        memset(in, 0, sizeof(in[0]) * (m + 1));

        S[scnt++] = id[a]; in[a] = 1;

        int mini = 0, minv = 0;

        while (scnt < m)
        {
            mini = -1; minv = -1;
            for (int i=1; i<=m; ++i)
            {
                if (in[i]) continue;
                cmp[i] += g[id[i]][S[scnt-1]];
                if (cmp[i] > minv)
                {
                    minv = cmp[i];
                    mini = i;
                }
            }
            S[scnt++] = id[mini];
            in[mini] = true;
        }

        int s = S[scnt-1], t = S[scnt-2];

        if (s > t) swap(s, t);
        for (int i=1; i<=m; ++i)
        {
            g[s][id[i]] += g[t][id[i]];
            g[id[i]][s] += g[t][id[i]];
            g[t][id[i]] = g[id[i]][t] = 0;
        }

        for (int i=1; i<=m; ++i)
        {
            if (id[i] == t)
            {
                for (int j=i+1; j<=m; ++j)
                    id[j-1] = id[j];
                break;
            }
        }
        return minv;
    }

    int go(int g[N][N], int n, int a = 1)
    {
        int m = n;
        for (int i=1; i<=m; ++i)
            id[i] = i;
        int ans = 0x7fffffff;
        for (; m > 1; --m)
        {
            ans = min(ans, add(g, a, m));
        }
        return ans;
    }
}
```

邻接表版本 HDU 6081 7000ms 左右，常数很大的样子，复杂度不是严格的 $O(VE + V^2 logV)$

```cpp
const int N = 3e3 + 5,
          M = 1e6 * 2 + 50,
          INF = 0x3f3f3f3f;

struct Edge
{
    int to, next, w;
} e[M];
int ecnt;
int head[N], tail[N], link[N];

struct dsu
{
    int f[N];
    void init(int n)
    {
        for (int i = 0; i <= n; ++i)
            f[i] = i;
    }
    int find(int x)
    {
        return x == f[x] ? x : (f[x] = find(f[x]));
    }
    bool connect(int x, int y)
    {
        x = find(x);
        y = find(y);
        f[x] = y;
        return x != y;
    }

    bool test(int x, int y)
    {
        x = find(x);
        y = find(y);
        return x != y;
    }
} ds;

int n, m;
void init(int n)
{
    memset(head, -1, sizeof(int) * (n + 1));
    ds.init(n);
    ecnt = 0;
}

void add_edge(int u, int v, int w)
{
    e[ecnt] = {v, head[u], w};
    head[u] = ecnt++;
    e[ecnt] = {u, head[v], w};
    head[v] = ecnt++;
}

namespace stoerwagner
{
int mincutphase(int cnt, int &s, int &t)
{
    static bool vis[N];
    static int val[N];

    memset(vis + 1, 0, sizeof(vis[0]) * n);
    memset(val + 1, 0, sizeof(int) * n);

    priority_queue<pair<int, int>> que;
    t = 1;
    while (--cnt)
    {
        vis[s = t] = true;
        for (int u = s; ~u; u = link[u])
        {
            for (int p = head[u]; ~p; p = e[p].next)
            {
                int v = ds.find(e[p].to);
                if (!vis[v])
                {
                    que.push(make_pair(val[v] += e[p].w, v));
                }
            }
        }
        t = 0;
        while (!t)
        {
            if (que.empty())
                return 0; // not connected graph
            auto pa = que.top();
            que.pop();
            if (val[pa.second] == pa.first)
                t = pa.second;
        }
    }
    return val[t];
}
int go()
{
    int res = 0x7fffffff;
    for (int i = 1; i <= n; ++i)
    {
        tail[i] = i;
        link[i] = -1;
    }
    for (int i = n, s, t; i > 1; --i)
    {
        res = min(res, mincutphase(i, s, t));
        if (res == 0)
            break;
        if (s > t)
            swap(s, t);
        ds.connect(s, t);
        link[tail[t]] = s;
        tail[t] = tail[s];
    }
    return res;
}
} // namespace stoerwagner
```

LCA

- 倍增

```
void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
    FORD (i, SP - 1, -1) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}
```

树上差分

对一堆路径上的边或点作区间加法后，最后询问每个边或点的权值

如果整体加上 x：

- 点 `(u,v), lca(u,v) = o, f[o] = p`：`diff[u]+=x,diff[v]+=x,diff[o]-=x,diff[p]-=x;`
- 边 `diff[u]+=x,diff[v]+=x,diff[o]-=2*x;`

网络流

- 最大流

```cpp
namespace dinic
{
struct E
{
    int to;
    LL cp;
    E(int to = 0, LL cp = 0) : to(to), cp(cp) {}
};
struct Dinic
{
    static const int INF = 0x7fffffff;
    int m, s, t, n;

    E edges[M];
    int ecnt;
    vector<int> G[N];

    int d[N], cur[N];

    void init(int n, int s, int t)
    {
        this->s = s;
        this->t = t;
        this->n = n;
        ecnt = m = 0;
        for (int i = 0; i <= n; ++i)
            G[i].clear();
    }

    void addedge(int u, int v, LL cap)
    {
        edges[ecnt++] = E(v, cap);
        edges[ecnt++] = E(u, 0);
        G[u].push_back(m++);
        G[v].push_back(m++);
    }

    bool bfs()
    {
        memset(d, 0, sizeof(int) * (n + 2));
        queue<int> Q;
        Q.push(s);
        d[s] = 1;
        while (!Q.empty())
        {
            int x = Q.front();
            Q.pop();
            for (int &i : G[x])
            {
                E &e = edges[i];
                if (!d[e.to] && e.cp > 0)
                {
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return d[t];
    }

    LL DFS(int u, LL cp)
    {
        if (u == t || !cp)
            return cp;
        LL tmp = cp;
        LL f;
        for (int &i = cur[u]; i < (int)G[u].size(); ++i)
        {
            E &e = edges[G[u][i]];
            if (d[u] + 1 == d[e.to])
            {
                f = DFS(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[G[u][i] ^ 1].cp += f;
                cp -= f;
                if (!cp)
                    break;
            }
        }
        return tmp - cp;
    }

    LL go(int S = -1, int T = -1)
    {
        if ((~S) && (~T))
        {
            s = S;
            t = T;
        }
        LL flow = 0;
        while (bfs())
        {
            memset(cur, 0, sizeof(int) * (n + 1));
            flow += DFS(s, INF);
        }
        return flow;
    }

};
} // namespace dinic
```

- 费用流
- zkw 费用流（代码长度没有优势）
- 不允许有负权边

```
struct E {
    int to, cp, v;
    E() {}
    E(int to, int cp, int v): to(to), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t, cost, D;
    vector<E> edges;
    vector<int> G[maxn];
    bool vis[maxn];

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        FOR (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(to, cap, cost);
        edges.emplace_back(from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    int aug(int u, int cp) {
        if (u == t) {
            cost += D * cp;
            return cp;
        }
        vis[u] = true;
        int tmp = cp;
        for (int idx: G[u]) {
            E& e = edges[idx];
            if (e.cp && !e.v && !vis[e.to]) {
                int f = aug(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[idx ^ 1].cp += f;
                cp -= f;
                if (!cp) break;
            }
        }
        return tmp - cp;
    }

    bool modlabel() {
        int d = INF;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    E& e = edges[idx];
                    if (e.cp && !vis[e.to]) d = min(d, e.v);
                }
        if (d == INF) return false;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    edges[idx].v -= d;
                    edges[idx ^ 1].v += d;
                }
        D += d;
        return true;
    }

    int go(int k) {
        cost = D = 0;
        int flow = 0;
        while (true) {
            memset(vis, 0, sizeof vis);
            int t = aug(s, INF);
            if (!t && !modlabel()) break;
            flow += t;
        }
        return cost;
    }
} MM;
```

- dij 费用流
- 对负权费用流的优化，改变负权后最短路变成 dij 了

```
namespace mcf {
    const int N = 1e5+50;
    template<typename T>
    struct E {
        int to, cp;
        T cost;
        int rev;
        E(int to=0, int cp=0, T cost=0, int rev=0):to(to), cp(cp), cost(cost), rev(rev){}
    };

    template<typename T>
    struct Mcmf {
        int v, prev[N], pree[N];
        T h[N], dis[N];
        vector<E<T>> G[N];
        void init(int n) {
            v = n;
            for (int i=0; i<=v; ++i)
                G[i].clear();
        }
        void addEdge(int from, int to, int cp, T cost) {
            G[from].emplace_back(to, cp, cost, G[to].size());
            G[to].emplace_back(from, 0, -cost, G[from].size()-1);
        }
        T go(int s, int t, int f, int &flow) {
            const T INF = 0x3f3f3f3f;
            T res = 0;
            memset(h, 0, sizeof (h[0]) * (v + 1));
            while (f) {
                priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int>>> q;
                fill(dis, dis+v+1, INF);
                dis[s] = 0;
                q.emplace(0, s);
                while (!q.empty()) {
                    auto now = q.top();
                    q.pop();
                    int v = now.second;
                    if (dis[v] < now.first)
                        continue;
                    for (int i=0; i<sz(G[v]); ++i) {
                        auto &e = G[v][i];
                        if (e.cp > 0 && dis[e.to] > dis[v] + e.cost + h[v] - h[e.to]) {
                            dis[e.to] = dis[v] + e.cost + h[v] - h[e.to];
                            prev[e.to] = v;
                            pree[e.to] = i;
                            q.emplace(dis[e.to], e.to);
                        }
                    }
                }
                if (dis[t] == INF)
                    break;
                for (int i=0; i<=v; ++i)
                    h[i] += dis[i];
                int d = f;
                for (int v=t; v!=s; v=prev[v])
                    d = min(d, G[prev[v]][pree[v]].cp);
                f -= d;
                flow += d;
                res += d * h[t];
                for (int v=t; v!=s; v=prev[v]) {
                    auto &e = G[prev[v]][pree[v]];
                    e.cp -= d;
                    G[v][e.rev].cp += d;
                }
            }
            return res;
        }
    };
}
```

带下界网络流:

- 无源汇：$u \to v$ 边容量为 $[l, r]$，连容量 $r - l$，虚拟源点到 v 连 l，u 到虚拟汇点连 l。检查时查看 S 的出边是否都是满流
- edges 中的边号前 0-m-1 * 2 为图中的边及其反向边
- 之后跟着是超级源点汇点的边
- 反向边的 cp 便是原边的流量

判断是否形成循环流，如果能 printTrace 输出每条边的流量

```
class extdinic1
{
public:
    dinic::Dinic dc;
    int S, T;
    LL tf[N];
    LL sum;
    void init(int n, int m, int u[], int v[], int w1[], int w2[])
    {
        S = ++n; T = ++n;
        dc.init(n, S, T);
        memset(tf, 0, sizeof(LL) * (n+1));
        for (int i=0; i<m; ++i)
        {
            tf[v[i]] += w1[i];
            tf[u[i]] -= w1[i];
            dc.addedge(u[i], v[i], w2[i] - w1[i]);
        }
        sum = 0;
        for (int i=1; i<=n-2; ++i)
        {
            if (tf[i] < 0)
            {
                dc.addedge(i, T, -tf[i]);
            }
            else if (tf[i] > 0)
            {
                dc.addedge(S, i, tf[i]);
                sum += tf[i];
            }
        }
    }

    LL go()
    {
        LL ans = dc.go();
        if (ans != sum) return -1;
        else return ans;
    }
};
```

- 有源汇：为了让流能循环使用，连 $T \to S$，容量 ∞。
- 最大流：跑完可行流后，加 $S' \to S$，$T \to T'$，最大流就是答案（$T \to S$ 的流量自动退回去了，这一部分就是下界部分的流量）。

```
class extdinic2 : public extdinic1
{
public:
    static const LL INF = 1e18;
    int s, t;

    void init(int n, int m, int s, int t, int u[], int v[], int w1[], int w2[])
    {
        this->s = s; this->t = t;
        extdinic1::init(n, m, u, v, w1, w2);
    }
    LL go()
    {
        dc.addedge(t, s, INF);
        LL ans = extdinic1::go(); if (ans == -1) return -1;
        ans = dc.go(s, t);
        return ans;
    }
} dc;
```

- 最小流：T 到 S 的那条边的实际流量，减去删掉那条边后 T 到 S 的最大流。
- loj117
- 网上说可能会减成负的，还要有限地供应 S 之后，再跑一遍 S 到 T 的。

```
class extdinic3 : public extdinic2
{
public:
    LL go()
    {
        dc.addedge(t, s, INF);
        LL ans = extdinic1::go();
        if (ans == -1) return -1;
        ans = dc.edges[dc.ecnt-1].cp;
        dc.edges[dc.ecnt-1].cp = dc.edges[dc.ecnt-2].cp = 0;
        ans -= dc.go(t, s);
        return ans;
    }
} dc;
```

- 最小费用可行（最小费用）流：模仿带下界可行流，构造循环流。下界部分的钱事先算好，连边就不要钱了，然后只跑两个界限之间的费用。流量是假的，好像只能用费用？好像这样是最小的费用？

loj2226

```cpp
class extmcf1
{
public:
    mcf::Mcmf mcf;
    int d[N];
    int sum, S, T, s, t;
    int baseCost;
    void init(int n, int m, int s, int t, int u[], int v[], int c[], int w1[], int w2[])
    {
        memset(d, 0, sizeof(int) * (n + 1));
        sum = baseCost = 0; S = ++n; T = ++n;
        this->s = s; this->t = t;
        mcf.init(n);


        for (int i=0; i<m; ++i)
        {
            d[v[i]] += w1[i];
            d[u[i]] -= w1[i];
            sum += w1[i];
            baseCost += w1[i] * c[i];
            mcf.addEdge(u[i], v[i], w2[i] - w1[i], c[i]);
        }

        for (int i=1; i<=n; ++i)
        {
            if (d[i] > 0)
            {
                mcf.addEdge(S, i, d[i], 0);
            }
            else if (d[i] < 0)
            {
                mcf.addEdge(i, T, -d[i], 0);
            }
        }

        mcf.addEdge(t, s, 0x3f3f3f3f, 0);
    }

    int go(int &flow)
    {
        flow = 0;
        int cost = baseCost;
        // printf("baseCost=%d\n", baseCost);
        cost += mcf.go(S, T, 0x3f3f3f3f, flow);
        flow += d[t];
        return cost;
    }
} mcmf;
```

树链剖分

- top, f, d, rid 给出 id[v], dep, id, fa, son, siz 给出原始 id
- 使用 init

```cpp
const int N = 500000+5;

int dep[N], id[N], fa[N];
vector<int> G[N];

namespace HLD {
    int top[N], d[N], f[N], son[N], rid[N];
    int dfn;

    int siz[N];

    void dfs1(int u, int la, int depth) {
        dep[u] = depth;
        fa[u] = la;
        son[u] = -1;
        siz[u] = 1;
        for (int v : G[u]) {
            if (v == la) continue;
            dfs1(v, u, depth+1);
            siz[u] += siz[v];
            if (!~son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
    }

    void dfs2(int u, int la, int t) {
        id[u] = ++dfn;
        rid[id[u]] = u;
        top[id[u]] = id[t];
        d[id[u]] = dep[u];
        f[id[u]] = id[la];

        for (int v : G[u]) {
            if (v == la) continue;
            if (v == son[u]) dfs2(v, u, t);
            else dfs2(v, u, v);
        }
    }

    int lca(int u, int v) {
        for (u=id[u], v=id[v];top[u]!=top[v]; u=f[top[u]]) {
            if (d[top[u]] < d[top[v]]) swap(u, v);
        }
        if (d[u] < d[v]) swap(u, v);
        return rid[v];
    }

    void init(int root) {
        dfn = 0;
        dfs1(root, root, 1);
        dfs2(root, root, root);
    }

    // non-checked
    int up(int u, int step) {
        for (u=id[u]; step; u=f[u], --step) {
            int t = min(d[u] - d[top[u]], step);
            u -= t;
            step -= t;
        }
        return rid[u];
    }

    // non-checked
    int finds(int u, int rt) { // find the lowest v that is the ancestor of u
        u = id[u]; rt = id[rt];
        while (top[u] != top[rt]) {
            u = top[u];
            if (fa[u] == rt) return u;
            u = fa[u];
        }
        return rid[rt + 1];
    }
}
```

二分图匹配

- 最小覆盖数 = 最大匹配数
- 最大独立集 = 顶点数 - 二分图匹配数
- DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数
- 二分图最大权完美匹配 KM

```
namespace R {
    const int M = 400 + 5;
    const int INF = 2E9;
    int n;
    int w[M][M], kx[M], ky[M], py[M], vy[M], slk[M], pre[M];

    LL KM() {
        FOR (i, 1, n + 1)
            FOR (j, 1, n + 1)
                kx[i] = max(kx[i], w[i][j]);
        FOR (i, 1, n + 1) {
            fill(vy, vy + n + 1, 0);
            fill(slk, slk + n + 1, INF);
            fill(pre, pre + n + 1, 0);
            int k = 0, p = -1;
            for (py[k = 0] = i; py[k]; k = p) {
                int d = INF;
                vy[k] = 1;
                int x = py[k];
                FOR (j, 1, n + 1)
                    if (!vy[j]) {
                        int t = kx[x] + ky[j] - w[x][j];
                        if (t < slk[j]) { slk[j] = t; pre[j] = k; }
                        if (slk[j] < d) { d = slk[j]; p = j; }
                    }
                FOR (j, 0, n + 1)
                    if (vy[j]) { kx[py[j]] -= d; ky[j] += d; }
                    else slk[j] -= d;
            }
            for (; k; k = pre[k]) py[k] = py[pre[k]];
        }
        LL ans = 0;
        FOR (i, 1, n + 1) ans += kx[i] + ky[i];
        return ans;
    }
}
```

欧拉路径

```
int S[N << 1], top;
Edge edges[N << 1];
set<int> G[N];

void DFS(int u) {
    S[top++] = u;
    for (int eid: G[u]) {
        int v = edges[eid].get_other(u);
        G[u].erase(eid);
        G[v].erase(eid);
        DFS(v);
        return;
    }
}

void fleury(int start) {
    int u = start;
    top = 0; path.clear();
    S[top++] = u;
    while (top) {
        u = S[--top];
        if (!G[u].empty())
            DFS(u);
        else path.push_back(u);
    }
}
```

强连通分量与 2-SAT

需要加边，定义 `sat::n` 表示总节点个数，记得开双倍点

- a || b: (a’ ,b), (b’ ,a)
- a: (a’ , a)
- !a: (a, a’)
- 其他的先作变换然后再套上面公式
- sat::cmp[a] > sat::cmp[a’] 表示选 a，如果相等则无解

```cpp
namespace sat
{
const int N = 2e6 + 5;
int n;
vector<int> G[N], rG[N], vs;
int used[N], cmp[N];

void add_edge(int from, int to)
{
    G[from].push_back(to);
    rG[to].push_back(from);
}

void dfs(int v)
{
    dbg(v);
    used[v] = true;
    for (int u : G[v])
    {
        if (!used[u])
            dfs(u);
    }
    vs.push_back(v);
}

void rdfs(int v, int k)
{
    dbg(v, k);
    used[v] = true;
    cmp[v] = k;
    for (int u : rG[v])
        if (!used[u])
            rdfs(u, k);
}

int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for (int v = 1; v <= n; ++v)
        if (!used[v]) dfs(v);
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = (int) vs.size() - 1; i >= 0; --i)
        if (!used[vs[i]]) rdfs(vs[i], k++);
    return k;
}


} // namespace sat
```

一般图匹配

带花树。复杂度 $O(n^3)$。

```cpp
int n;
vector<int> G[N];
int fa[N], mt[N], pre[N], mk[N];
int lca_clk, lca_mk[N];
pair<int, int> ce[N];

void connect(int u, int v) {
    mt[u] = v;
    mt[v] = u;
}
```

```
int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

void flip(int s, int u) {
    if (s == u) return;
    if (mk[u] == 2) {
        int v1 = ce[u].first, v2 = ce[u].second;
        flip(mt[u], v1);
        flip(s, v2);
        connect(v1, v2);
    } else {
        flip(s, pre[mt[u]]);
        connect(pre[mt[u]], mt[u]);
    }
}

int get_lca(int u, int v) {
    lca_clk++;
    for (u = find(u), v = find(v); ; u = find(pre[u]), v = find(pre[v])) {
        if (u && lca_mk[u] == lca_clk) return u;
        lca_mk[u] = lca_clk;
        if (v && lca_mk[v] == lca_clk) return v;
        lca_mk[v] = lca_clk;
    }
}

void access(int u, int p, const pair<int, int>& c, vector<int>& q) {
    for (u = find(u); u != p; u = find(pre[u])) {
        if (mk[u] == 2) {
            ce[u] = c;
            q.push_back(u);
        }
        fa[find(u)] = find(p);
    }
}

bool aug(int s) {
    fill(mk, mk + n + 1, 0);
    fill(pre, pre + n + 1, 0);
    iota(fa, fa + n + 1, 0);
    vector<int> q = {s};
    mk[s] = 1;
    int t = 0;
    for (int t = 0; t < (int) q.size(); ++t) {
        // q size can be changed
        int u = q[t];
        for (int &v: G[u]) {
            if (find(v) == find(u)) continue;
            if (!mk[v] && !mt[v]) {
                flip(s, u);
                connect(u, v);
                return true;
            } else if (!mk[v]) {
                int w = mt[v];
                mk[v] = 2; mk[w] = 1;
                pre[w] = v; pre[v] = u;
                q.push_back(w);
            } else if (mk[find(v)] == 1) {
                int p = get_lca(u, v);
                access(u, p, {u, v}, q);
                access(v, p, {v, u}, q);
            }
        }
    }
    return false;
}

int match() {
    fill(mt + 1, mt + n + 1, 0);
    lca_clk = 0;
    int ans = 0;
    FOR (i, 1, n + 1)
```

```
        if (!mt[i]) ans += aug(i);
    return ans;
}

int main() {
    int m; cin >> n >> m;
    while (m--) {
        int u, v; scanf("%d%d", &u, &v);
        G[u].push_back(v); G[v].push_back(u);
    }
    printf("%d\n", match());
    FOR (i, 1, n + 1) printf("%d%c", mt[i], i == _i - 1 ? '\n' : ' ');
    return 0;
}
```

Tarjan

简单缩环

- 每个点最多属于一个简单环
- 缩完点之后是一颗树
- 利用返祖边遍历每一个环
- G 是原图下标从 0 开始，G2 是缩点之后的图下标从 1 开始
- vis 表示 G2 中该点是不是原来的一个环

```
namespace simplecc {
    int fa[N], dep[N];

    int bl[N], B;

    bool vis[N];

    void init(int n) {
        B = 0;
        memset(dep, 0, sizeof(int) * (n + 1));
        memset(bl, 0, sizeof(int) * (n + 1));
    }

    void get(int x, int y) {
        if (dep[x] < dep[y]) return;
        bl[y] = ++B; vis[B] = 1;
        for (; x!=y; x=fa[x]) bl[x] = B;
    }

    void dfs(int u, int f) {
        fa[u] = f;
        dep[u] = dep[f] + 1;

        for (int v : G[u]) {
            if (v == f) continue;
            if (dep[v]) get(u, v);
            else dfs(v, u);
        }
    }

    void go(int n) {
        init(n);
        dfs(0, 0);
        for (int i=0; i<n; ++i) {
            if (!bl[i]) bl[i] = ++B;
        }

        for (int i=0; i<n; ++i) {
            for (int j : G[i]) {
                int u = bl[i], v = bl[j];
                if (u != v) G2[u].push_back(v);
            }
        }
    }
}
```

割点

- 判断割点
- 注意原图可能不连通

```
int dfn[N], low[N], clk;
void init() { clk = 0; memset(dfn, 0, sizeof dfn); }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int cc = fa != -1;
    for (int& v: G[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            cc += low[v] >= dfn[u];
        } else low[u] = min(low[u], dfn[v]);
    }
    if (cc > 1) // ...
}
```

桥

- 注意原图不连通和重边

```
int dfn[N], low[N], clk;
void init() { memset(dfn, 0, sizeof dfn); clk = 0; }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int _fst = 0;
    for (E& e: G[u]) {
        int v = e.to; if (v == fa && ++_fst == 1) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            if (low[v] > dfn[u]) // ...
            low[u] = min(low[u], low[v]);
        } else low[u] = min(low[u], dfn[v]);
    }
}
```

强连通分量缩点

- vector<int> G[N] 准备好
- 别忘了不连通的情况
- go 把整个图的缩圈都搞了

```
namespace tarjan {
    int low[N], dfn[N], clk, B, bl[N];
    vector<int> bcc[N];
    void init(int n) {
        B = clk = 0;
        memset(dfn, 0, sizeof(int) * (n+1));
        for (int i=0; i<n; ++i) bcc[i].clear();
    }

    void tarjan(int u) {
        static int st[N], p;
        static bool in[N];
        dfn[u] = low[u] = ++clk;
        st[p++] = u; in[u] = true;
        for (int &v : G[u]) {
            if (!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else if (in[v]) low[u] = min(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            while (1) {
                int x = st[--p]; in[x] = false;
                bl[x] = B; bcc[B].push_back(x);
                if (x == u) break;
            }
            ++ B;
        }
    }

    void go(int n) {
        init(n);
        for (int i=0; i<n; ++i) {
            if (!dfn[i]) tarjan(i);
        }
    }
}
```

点双连通分量 / 广义圆方树

- 数组开两倍
- 一条边也被计入点双了（适合拿来建圆方树），可以用点数 <= 边数过滤

```
struct E { int to, nxt; } e[N];
int hd[N], ecnt;
void addedge(int u, int v) {
    e[ecnt] = {v, hd[u]};
    hd[u] = ecnt++;
}
int low[N], dfn[N], clk, B, bno[N];
vector<int> bc[N], be[N];
bool vise[N];
void init() {
    memset(vise, 0, sizeof vise);
    memset(hd, -1, sizeof hd);
    memset(dfn, 0, sizeof dfn);
    memset(bno, -1, sizeof bno);
    B = clk = ecnt = 0;
}

void tarjan(int u, int feid) {
    static int st[N], p;
    static auto add = [&](int x) {
        if (bno[x] != B) { bno[x] = B; bc[B].push_back(x); }
    };
    low[u] = dfn[u] = ++clk;
    for (int i = hd[u]; ~i; i = e[i].nxt) {
        if ((feid ^ i) == 1) continue;
        if (!vise[i]) { st[p++] = i; vise[i] = vise[i ^ 1] = true; }
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v, i);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                bc[B].clear(); be[B].clear();
                while (1) {
                    int eid = st[--p];
                    add(e[eid].to); add(e[eid ^ 1].to);
                    be[B].push_back(eid);
                    if ((eid ^ i) <= 1) break;
                }
                ++B;
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}
```

圆方树

- 从仙人掌建圆方树
- N 至少边数 × 2

```
vector<int> G[N];
int nn;
```

```

struct E { int to, nxt; };
namespace C {
    E e[N * 2];
    int hd[N], ecnt;
    void addedge(int u, int v) {
        e[ecnt] = {v, hd[u]};
        hd[u] = ecnt++;
    }
    int idx[N], clk, fa[N];
    bool ring[N];
    void init() { ecnt = 0; memset(hd, -1, sizeof hd); clk = 0; }
    void dfs(int u, int feid) {
        idx[u] = ++clk;
        for (int i = hd[u]; ~i; i = e[i].nxt) {
            if ((i ^ feid) == 1) continue;
            int v = e[i].to;
            if (!idx[v]) {
                fa[v] = u; ring[u] = false;
                dfs(v, i);
                if (!ring[u]) { G[u].push_back(v); G[v].push_back(u); }
            } else if (idx[v] < idx[u]) {
                ++nn;
                G[nn].push_back(v); G[v].push_back(nn); // 强行把环的根放在最前面
                for (int x = u; x != v; x = fa[x]) {
                    ring[x] = true;
                    G[nn].push_back(x); G[x].push_back(nn);
                }
                ring[v] = true;
            }
        }
    }
}
```

最小树形图

- tarjan 做法

```
const int N=3e5+10;
const LL INF=1e15;
typedef pair<LL,int> P;

struct MTD
{
    set<P> G[N];
    LL shift[N];
    bool vis[N];
    int n,m,stk[N],top,fa[N];
    int Fa(int x){return fa[x]==x?x:fa[x]=Fa(fa[x]);}
    void init(int _n,int _m)
    {
        n=_n;m=_m;
        for (int i=0;i<=n;i++) fa[i]=i;
    }
    LL solve()
    {
        LL ans=0;
        for (int i=2;i<=n;i++)
        {
            int u=i;top=0;
            while(Fa(u)!=Fa(1))
            {
                vis[u]=true;stk[top++]=u;
                auto it=G[u].begin();
                while(it!=G[u].end())
                {
                    int v=it->second;
                    if (Fa(v)==Fa(u)) it=G[u].erase(it);
                    else break;
                }
                if(it==G[u].end())return -INF;
                LL lb=it->first;
                int v=it->second;
                G[u].erase(it);
                v=Fa(v);
                ans+=lb+shift[u];
                shift[u]-=lb;
                if(vis[v] && Fa(v)!=Fa(1))
                {
                    int x=v;
                    while(stk[top-1]!=v)
                    {
                        int y=stk[--top];
                        fa[Fa(y)]=Fa(x);
                        if(G[x].size()<G[y].size()) G[x].swap(G[y]),swap(shift[x],shift[y]);
                        for(auto pr: G[y])
                            G[x].insert(P(pr.first+shift[y]-shift[x],pr.second));
                        G[y].clear();
                    }
                }
                u=v;
            }
            while(top--) fa[Fa(stk[top])]=1;
        }
        for (int i=2;i<=n;i++)
            if(Fa(i)!=Fa(1)) return -INF;
        return ans;
    }
}tree;

int n,m;

int main()
{
    scanf("%d%d",&n,&m);
    tree.init(n,m);
    int u,v,w;
    for (int i=0;i<m;i++)
        scanf("%d%d%d",&u,&v,&w),tree.G[v].insert(P(w,u));
    LL ans=tree.solve();
    if (ans==-INF) puts("NO");else printf("%lld\n",ans);
}
```

差分约束

一个系统 n 个变量和 m 个约束条件组成，每个约束条件形如 $x_j - x_i \leq b_k$。可以发现每个约束条件都形如最短路中的三角不等式 $d_u - d_v \leq w_{u,v}$。因此连一条边 (i, j, b_k) 建图。

若要使得所有量两两的值最接近，源点到各点的距离初始成 0，跑最远路。

若要使得某一变量与其他变量的差尽可能大，则源点到各点距离初始化成 ∞，跑最短路。

最短路版本：luogu p5960

```
const int N = 5e3 + 5;
typedef pair<int, int> edge;
vector<edge> G[N];
int n, m;
namespace spfa
{
bool in[N];
int cnt[N];

bool go(int s, int dist[], int n)
{
    memset(in, 0, sizeof(in[0]) * (n + 1));
    memset(cnt, 0, sizeof(cnt[0]) * (n + 1));
    memset(dist, 0x3f3f3f3f, sizeof(dist[0]) * (n + 1));

    dist[s] = 0;
    cnt[s] = 1;
    in[s] = 1;
    queue<int> Q;
    Q.push(s);

    while (!Q.empty())
    {
        int now = Q.front();
        // printf("now=%d\n", now);
        Q.pop(); in[now] = false;
        for (const auto &p : G[now])
        {
            int v = p.first, cost = p.second;
            // printf("now=%d, v=%d, cost=%d\n", now, v, cost);
            if (dist[v] > dist[now] + cost)
            {
                dist[v] = dist[now] + cost;
                if (!in[v])
                {
                    ++cnt[v];
                    if (cnt[v] > n)
                        return true;
                    in[v] = true;
                    Q.push(v);
                }
            }
        }
    }
    return false;
}
} // namespace spfa

inline void add(int u, int v, int w)
{
    // printf("%d %d %d\n", u, v, w);
    G[u].emplace_back(v, w);
}

int a[N][3], x[N];
namespace diff_constraint
{
bool go(int a[][3], int x[], int n)
{
    for (int i=1; i<=n; ++i)
        G[i].clear();
    for (int i=0; i<n; ++i)
    {
        add(a[i][1], a[i][0], a[i][2]);
    }
    for (int i=1; i<=n; ++i)
        add(n+1, i, 0);
    return spfa::go(n+1, x, n+1);
}
}
```

三元环、四元环

四元环

考虑这样一个四元环，将答案统计在度数最大的点 b 上。考虑枚举点 u，然后枚举与其相邻的点 v，然后再枚举所有度数比 v 大的与 v 相邻的点，这些点显然都可能作为 b 点，我们维护一个计数器来计算之前 b 被枚举多少次，答案加上计数器的值，然后计数器加一。

枚举完 u 之后，我们用和枚举时一样的方法来清空计数器就好了。

任何一个点，与其直接相连的度数大于等于它的点最多只有 $\sqrt{2m}$ 个。所以复杂度 $O(m\sqrt{m})$。

```
LL cycle4() {
    LL ans = 0;
    iota(kth, kth + n + 1, 0);
    sort(kth, kth + n, [&](int x, int y) { return deg[x] < deg[y]; });
    FOR (i, 1, n + 1) rk[kth[i]] = i;
    FOR (u, 1, n + 1)
        for (int v: G[u])
```

```
            if (rk[v] > rk[u]) key[u].push_back(v);
    FOR (u, 1, n + 1) {
        for (int v: G[u])
            for (int w: key[v])
                if (rk[w] > rk[u]) ans += cnt[w]++;
        for (int v: G[u])
            for (int w: key[v])
                if (rk[w] > rk[u]) --cnt[w];
    }
    return ans;
}
```

三元环

将点分成度入小于 $\sqrt{m}$ 和超过 $\sqrt{m}$ 的两类。现求包含第一类点的三元环个数。由于边数较少，直接枚举两条边即可。由于一个点度数不超过 $\sqrt{m}$，所以一条边最多被枚举 $\sqrt{m}$ 次，复杂度 $O(m\sqrt{m})$。再求不包含第一类点的三元环个数，由于这样的点不超过 $\sqrt{m}$ 个，所以复杂度也是 $O(m\sqrt{m})$。

对于每条无向边 (u,v)，如果 $d_u < d_v$，那么连有向边 (u,v)，否则有向边 (v,u)。度数相等的按第二关键字判断。然后枚举每个点 x，假设 x 是三元组中度数最小的点，然后暴力往后面枚举两条边找到 y，判断 (x,y) 是否有边即可。复杂度也是 $O(m\sqrt{m})$。

```
int cycle3() {
    int ans = 0;
    for (E &e: edges) { deg[e.u]++; deg[e.v]++; }
    for (E &e: edges) {
        if (deg[e.u] < deg[e.v] || (deg[e.u] == deg[e.v] && e.u < e.v))
            G[e.u].push_back(e.v);
        else G[e.v].push_back(e.u);
    }
    FOR (x, 1, n + 1) {
        for (int y: G[x]) p[y] = x;
        for (int y: G[x]) for (int z: G[y]) if (p[z] == x) ans++;
    }
    return ans;
}
```

支配树

- semi[x] 半必经点（就是 x 的祖先 z 中，能不经过 z 和 x 之间的树上的点而到达 x 的点中深度最小的）
- idom[x] 最近必经点（就是深度最大的根到 x 的必经点）

```
vector<int> G[N], rG[N];
vector<int> dt[N];

namespace tl{
    int fa[N], idx[N], clk, ridx[N];
    int c[N], best[N], semi[N], idom[N];
    void init(int n) {
        clk = 0;
        fill(c, c + n + 1, -1);
        FOR (i, 1, n + 1) dt[i].clear();
        FOR (i, 1, n + 1) semi[i] = best[i] = i;
        fill(idx, idx + n + 1, 0);
    }
    void dfs(int u) {
        idx[u] = ++clk; ridx[clk] = u;
        for (int& v: G[u]) if (!idx[v]) { fa[v] = u; dfs(v); }
    }
    int fix(int x) {
        if (c[x] == -1) return x;
        int &f = c[x], rt = fix(f);
        if (idx[semi[best[x]]] > idx[semi[best[f]]]) best[x] = best[f];
        return f = rt;
    }
    void go(int rt) {
        dfs(rt);
        FORD (i, clk, 1) {
            int x = ridx[i], mn = clk + 1;
            for (int& u: rG[x]) {
                if (!idx[u]) continue;  // 可能不能到达所有点
                fix(u); mn = min(mn, idx[semi[best[u]]]);
            }
            c[x] = fa[x];
            dt[semi[x] = ridx[mn]].push_back(x);
            x = ridx[i - 1];
            for (int& u: dt[x]) {
                fix(u);
                if (semi[best[u]] != x) idom[u] = best[u];
                else idom[u] = x;
            }
            dt[x].clear();
        }

        FOR (i, 2, clk + 1) {
            int u = ridx[i];
            if (idom[u] != semi[u]) idom[u] = idom[idom[u]];
            dt[idom[u]].push_back(u);
        }
    }
}

int main()
{
    int x,p,q;
    scanf("%d%d%d",&n,&m,&x);
    for (int i=1;i<=m;i++)
    {
        scanf("%d%d",&p,&q);
        G[p].push_back(q);
    }
    tree.init(n);
    tree.go(x);
}
```

边双连通分量

- 邻接表编写
- 边从 0 开始标号，点从 1 开始
- init() 清空，go() 搞好 id[]，也就是缩点编号
- POJ3177, ZOJ4097

```
vector<int> u, v;
namespace EDCC {
vector<int> dfn, low, id;
vector<bool> insta;
stack<int> sta;

int edcc;
vector< vector< pair<int, int> > > G;
int clk;

void init(int n) {
    clk = 0;
    dfn.resize(n+1);
    fill(dfn.begin(), dfn.end(), 0);
    low.resize(n+1);
    fill(low.begin(), low.end(), 0);
    G.resize(n+1);
    FOR (i, 1, n+1)
        G[i].clear();
    id.resize(n+1);
    fill(id.begin(), id.end(), 0);
    while (!sta.empty()) sta.pop();
    insta.resize(n+1);
    fill(insta.begin(), insta.end(), 0);
    edcc = 0;
}

void add(int a, int b, int id) {
    if (a != b) {
        G[a].push_back(make_pair(b, id));
        G[b].push_back(make_pair(a, id));
    }
}

void tarjan(int now, int fa, int reid) {
    // dbg("tarjan", now);
    dfn[now] = ++clk;
    low[now] = dfn[now];
    insta[now] = 1;
    sta.push(now);
    for (const pii &e:G[now]) {
        int nt = e.first, eid = e.second;
        if (eid == reid) continue;
        if (!dfn[nt]) {
            tarjan(nt, now, eid);
            low[now] = min(low[nt], low[now]);
        } else if (insta[nt])
            low[now] = min(dfn[nt], low[now]);
    }

    if (dfn[now] == low[now]) {
        ++edcc;
        while (!sta.empty()) {
            int v = sta.top();
            id[v] = edcc;
            insta[v] = 0;
            sta.pop();
            if (v == now) break;
        }
    }
}
void go(int n) {
    FOR (i, 0, sz(u)) {
        add(u[i], v[i], i);
    }
    FOR (i, 1, n+1) {
        if (!dfn[i]) {
            if (!sta.empty()) {
                while(1){}
            }
            tarjan(i, i, -1);
        }
    }
}
}
```

计算几何

二维几何：点与向量

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
```

```cpp
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

象限

```cpp
// 象限
int quad(P p) {
    int x = sgn(p.x), y = sgn(p.y);
    if (x > 0 && y >= 0) return 1;
    if (x <= 0 && y > 0) return 2;
    if (x < 0 && y <= 0) return 3;
    if (x >= 0 && y < 0) return 4;
    assert(0);
}

// 仅适用于参照点在所有点一侧的情况
struct cmp_angle {
    P p;
    bool operator () (const P& a, const P& b) {
//        int qa = quad(a - p), qb = quad(b - p);
//        if (qa != qb) return qa < qb;
        int d = sgn(cross(a, b, p));
        if (d) return d > 0;
        return dist(a - p) < dist(b - p);
    }
};
```

线

```cpp
// 是否平行
bool parallel(const L& a, const L& b) {
    return !sgn(det(P(a), P(b)));
}
// 直线是否相等
bool l_eq(const L& a, const L& b) {
    return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t));
}
// 逆时针旋转 r 弧度
P rotation(const P& p, const LD& r) { return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r)); }
P RotateCCW90(const P& p) { return P(-p.y, p.x); }
P RotateCW90(const P& p) { return P(p.y, -p.x); }
// 单位法向量
V normal(const V& v) { return V(-v.y, v.x) / dist(v); }
```

点与线

```cpp
// 点在线段上  <= 0 包含端点 < 0 则不包含
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}
// 点到直线距离
LD dist_to_line(const P& p, const L& l) {
    return fabs(cross(l.s, l.t, p)) / dist(l);
}
// 点到线段距离
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}
```

线与线

```
// 求直线交 需要事先保证有界
P l_intersection(const L& a, const L& b) {
    LD s1 = det(P(a), b.s - a.s), s2 = det(P(a), b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
// 向量夹角的弧度
LD angle(const V& a, const V& b) {
    LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
    if (sgn(dot(a, b)) < 0) r = PI - r;
    return r;
}
// 线段和直线是否有交   1 = 规范, 2 = 不规范
int s_l_cross(const L& seg, const L& line) {
    int d1 = sgn(cross(line.s, line.t, seg.s));
    int d2 = sgn(cross(line.s, line.t, seg.t));
    if ((d1 ^ d2) == -2) return 1; // proper
    if (d1 == 0 || d2 == 0) return 2;
    return 0;
}
// 线段的交   1 = 规范, 2 = 不规范
int s_cross(const L& a, const L& b, P& p) {
    int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
    int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
    if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
    if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
    if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
    if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
    return 0;
}
```

多边形

面积、假凸包（字典序排序）

```
typedef vector<P> S;

// 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
int inside(const S& s, const P& p) {
    int cnt = 0;
    FOR (i, 0, s.size()) {
        P a = s[i], b = s[nxt(i)];
        if (p_on_seg(p, L(a, b))) return -1;
        if (sgn(a.y - b.y) <= 0) swap(a, b);
        if (sgn(p.y - a.y) > 0) continue;
        if (sgn(p.y - b.y) <= 0) continue;
        cnt += sgn(cross(b, a, p)) > 0;
    }
    return bool(cnt & 1);
}
// 多边形面积，有向面积可能为负
LD polygon_area(const S& s) {
    LD ret = 0;
    FOR (i, 1, (LL)s.size() - 1)
        ret += cross(s[i], s[i + 1], s[0]);
    return ret / 2;
}
// 构建凸包 点不可以重复 < 0 边上可以有点， <= 0 则不能
// 会改变输入点的顺序
const int MAX_N = 1000;
S convex_hull(S& s) {
//    assert(s.size() >= 3);
    sort(s.begin(), s.end());
    S ret(MAX_N * 2);
    int sz = 0;
    FOR (i, 0, s.size()) {
        while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    int k = sz;
    FORD (i, (LL)s.size() - 2, -1) {
        while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}

P ComputeCentroid(const vector<P> &p) {
    P c(0, 0);
    LD scale = 6.0 * polygon_area(p);
    for (unsigned i = 0; i < p.size(); i++) {
        unsigned j = (i + 1) % p.size();
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
    }
    return c / scale;
}
```

真凸包（极角排序）

```
typedef double LD;

const LD eps = 1e-8;
const int N = 1e5+5;
int sgn(LD x) { return (x > eps) - (x < -eps);}

struct P {
    LD x, y;
    P(LD x=0, LD y=0):x(x), y(y){}
    LD operator * (const P &other) const {
        return x * other.y - y * other.x;
    }
    bool operator < (const P &other) const {
        return x == other.x ? y < other.y : x < other.x;
    }

    P operator - (const P &other) const {
        return P(x - other.x, y - other.y);
    }

    bool operator == (const P &other) const {

        return sgn(x - other.x) == 0 && sgn(y - other.y) == 0;
    }

    static P src;

    static void setSrc(const P &s) {
        src = s;
    }

    static bool cmp(const P &a, const P &b) {
        if (a == src) return 1;
        else if (b == src) return 0;
```

```cpp
        return (a-src) * (b-src) >= 0;
    }

    LD dist(const P &other) const {
        LD dx = x-other.x, dy = y-other.y;
        return sqrt(dx*dx+dy*dy);
    }
};
P P::src;
int Convex(P a[], int n, P s[]) {
    int top = 0;
    P::src = a[0];
    for (int i=1; i<n; ++i) {
        if (a[i].y < P::src.y) P::src = a[i];
    }
    sort(a, a+n, P::cmp);
    s[top++] = a[0];
    s[top++] = a[1];
    s[top++] = a[2];
    for (int i=3; i<n; ++i) {
        while (top >= 3 && sgn((a[i] - s[top-2]) * (s[top-1] - s[top-2])) >= 0) --top;
        s[top++] = a[i];
    }
    return top;
}
```

旋转卡壳

```cpp
LD rotatingCalipers(vector<P>& qs) {
    int n = qs.size();
    if (n == 2)
        return dist(qs[0] - qs[1]);
    int i = 0, j = 0;
    FOR (k, 0, n) {
        if (!(qs[i] < qs[k])) i = k;
        if (qs[j] < qs[k]) j = k;
    }
    LD res = 0;
    int si = i, sj = j;
    while (i != sj || j != si) {
        res = max(res, dist(qs[i] - qs[j]));
        if (sgn(cross(qs[(i+1)%n] - qs[i], qs[(j+1)%n] - qs[j])) < 0)
            i = (i + 1) % n;
        else j = (j + 1) % n;
    }
    return res;
}

int main() {
    int n;
    while (cin >> n) {
        S v(n);
        FOR (i, 0, n) cin >> v[i].x >> v[i].y;
        convex_hull(v);
        printf("%.0f\n", rotatingCalipers(v));
    }
}
```

半平面交

```cpp
struct LV {
    P p, v; LD ang;
    LV() {}
    LV(P s, P t): p(s), v(t - s) { ang = atan2(v.y, v.x); }
};  // 另一种向量表示

bool operator < (const LV &a, const LV& b) { return a.ang < b.ang; }
bool on_left(const LV& l, const P& p) { return sgn(cross(l.v, p - l.p)) >= 0; }
P l_intersection(const LV& a, const LV& b) {
    P u = a.p - b.p; LD t = cross(b.v, u) / cross(a.v, b.v);
    return a.p + a.v * t;
}

S half_plane_intersection(vector<LV>& L) {
    int n = L.size(), fi, la;
    sort(L.begin(), L.end());
    vector<P> p(n); vector<LV> q(n);
    q[fi = la = 0] = L[0];
    FOR (i, 1, n) {
        while (fi < la && !on_left(L[i], p[la - 1])) la--;
        while (fi < la && !on_left(L[i], p[fi])) fi++;
        q[++la] = L[i];
        if (sgn(cross(q[la].v, q[la - 1].v)) == 0) {
            la--;
            if (on_left(q[la], L[i].p)) q[la] = L[i];
        }
        if (fi < la) p[la - 1] = l_intersection(q[la - 1], q[la]);
    }
    while (fi < la && !on_left(q[fi], p[la - 1])) la--;
    if (la - fi <= 1) return vector<P>();
    p[la] = l_intersection(q[la], q[fi]);
    return vector<P>(p.begin() + fi, p.begin() + la + 1);
}

S convex_intersection(const vector<P> &v1, const vector<P> &v2) {
    vector<LV> h; int n = v1.size(), m = v2.size();
    FOR (i, 0, n) h.push_back(LV(v1[i], v1[(i + 1) % n]));
    FOR (i, 0, m) h.push_back(LV(v2[i], v2[(i + 1) % m]));
    return half_plane_intersection(h);
}
```

圆

```
struct C {
    P p; LD r;
    C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
    C(P p, LD r): p(p), r(r) {}
};
```

三点求圆心

```
P compute_circle_center(P a, P b, P c) {
    b = (a + b) / 2;
    c = (a + c) / 2;
    return l_intersection({b, b + RotateCW90(a - b)}, {c , c + RotateCW90(a - c)});
}
```

圆线交点、圆圆交点

- 圆和线的交点关于圆心是顺时针的

```
vector<P> c_l_intersection(const L& l, const C& c) {
    vector<P> ret;
    P b(l), a = l.s - c.p;
    LD x = dot(b, b), y = dot(a, b), z = dot(a, a) - c.r * c.r;
    LD D = y * y - x * z;
    if (sgn(D) < 0) return ret;
    ret.push_back(c.p + a + b * (-y + sqrt(D + eps)) / x);
    if (sgn(D) > 0) ret.push_back(c.p + a + b * (-y - sqrt(D)) / x);
    return ret;
}

vector<P> c_c_intersection(C a, C b) {
    vector<P> ret;
    LD d = dist(a.p - b.p);
    if (sgn(d) == 0 || sgn(d - (a.r + b.r)) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
        return ret;
    LD x = (d * d - b.r * b.r + a.r * a.r) / (2 * d);
    LD y = sqrt(a.r * a.r - x * x);
    P v = (b.p - a.p) / d;
    ret.push_back(a.p + v * x + RotateCCW90(v) * y);
    if (sgn(y) > 0) ret.push_back(a.p + v * x - RotateCCW90(v) * y);
    return ret;
}
```

圆圆位置关系

```
// 1: 内含 2: 内切 3: 相交 4: 外切 5: 相离
int c_c_relation(const C& a, const C& v) {
    LD d = dist(a.p - v.p);
    if (sgn(d - a.r - v.r) > 0) return 5;
    if (sgn(d - a.r - v.r) == 0) return 4;
    LD l = fabs(a.r - v.r);
    if (sgn(d - l) > 0) return 3;
    if (sgn(d - l) == 0) return 2;
    if (sgn(d - l) < 0) return 1;
}
```

圆与多边形交

- HDU 5130
- 注意顺时针逆时针（可能要取绝对值）

```
LD sector_area(const P& a, const P& b, LD r) {
    LD th = atan2(a.y, a.x) - atan2(b.y, b.x);
    while (th <= 0) th += 2 * PI;
    while (th > 2 * PI) th -= 2 * PI;
    th = min(th, 2 * PI - th);
    return r * r * th / 2;
}

LD c_tri_area(P a, P b, P center, LD r) {
    a = a - center; b = b - center;
    int ina = sgn(dist(a) - r) < 0, inb = sgn(dist(b) - r) < 0;
    // dbg(a, b, ina, inb);
    if (ina && inb) {
        return fabs(cross(a, b)) / 2;
    } else {
        auto p = c_l_intersection(L(a, b), C(0, 0, r));
        if (ina ^ inb) {
            auto cr = p_on_seg(p[0], L(a, b)) ? p[0] : p[1];
            if (ina) return sector_area(b, cr, r) + fabs(cross(a, cr)) / 2;
            else return sector_area(a, cr, r) + fabs(cross(b, cr)) / 2;
        } else {
            if ((int) p.size() == 2 && p_on_seg(p[0], L(a, b))) {
                if (dist(p[0] - a) > dist(p[1] - a)) swap(p[0], p[1]);
                return sector_area(a, p[0], r) + sector_area(p[1], b, r)
                    + fabs(cross(p[0], p[1])) / 2;
            } else return sector_area(a, b, r);
        }
    }
}

typedef vector<P> S;
LD c_poly_area(S poly, const C& c) {
    LD ret = 0; int n = poly.size();
    FOR (i, 0, n) {
        int t = sgn(cross(poly[i] - c.p, poly[(i + 1) % n] - c.p));
        if (t) ret += t * c_tri_area(poly[i], poly[(i + 1) % n], c.p, c.r);
    }
    return ret;
}
```

圆的离散化、面积并

SPOJ: CIRU, EOJ: 284

- 版本 1：复杂度 $O(n^3 \log n)$。虽然常数小，但还是难以接受。
- 优点？想不出来。

- 原理上是用竖线进行切分，然后对每一个切片分别计算。
- 扫描线部分可以魔改，求各种东西。

```cpp
inline LD rt(LD x) { return sgn(x) == 0 ? 0 : sqrt(x); }
inline LD sq(LD x) { return x * x; }

// 圆弧
// 如果按照 x 离散化，圆弧是 " 横着的"
// 记录圆弧的左端点、右端点、中点的坐标，和圆弧所在的圆
// 调用构造要保证 c.x - x.r <= xl < xr <= c.y + x.r
// t = 1 下圆弧 t = -1 上圆弧
struct CV {
    LD yl, yr, ym; C o; int type;
    CV() {}
    CV(LD yl, LD yr, LD ym, C c, int t)
        : yl(yl), yr(yr), ym(ym), type(t), o(c) {}
};

// 辅助函数 求圆上纵坐标
pair<LD, LD> c_point_eval(const C& c, LD x) {
    LD d = fabs(c.p.x - x), h = rt(sq(c.r) - sq(d));
    return {c.p.y - h, c.p.y + h};
}
// 构造上下圆弧
pair<CV, CV> pairwise_curves(const C& c, LD xl, LD xr) {
    LD yl1, yl2, yr1, yr2, ym1, ym2;
    tie(yl1, yl2) = c_point_eval(c, xl);
    tie(ym1, ym2) = c_point_eval(c, (xl + xr) / 2);
    tie(yr1, yr2) = c_point_eval(c, xr);
    return {CV(yl1, yr1, ym1, c, 1), CV(yl2, yr2, ym2, c, -1)};
}

// 离散化之后同一切片内的圆弧应该是不相交的
bool operator < (const CV& a, const CV& b) { return a.ym < b.ym; }
// 计算圆弧和连接圆弧端点的线段构成的封闭图形的面积
LD cv_area(const CV& v, LD xl, LD xr) {
    LD l = rt(sq(xr - xl) + sq(v.yr - v.yl));
    LD d = rt(sq(v.o.r) - sq(l / 2));
    LD ang = atan(l / d / 2);
    return ang * sq(v.o.r) - d * l / 2;
}

LD circle_union(const vector<C>& cs) {
    int n = cs.size();
    vector<LD> xs;
    FOR (i, 0, n) {
        xs.push_back(cs[i].p.x - cs[i].r);
        xs.push_back(cs[i].p.x);
        xs.push_back(cs[i].p.x + cs[i].r);
        FOR (j, i + 1, n) {
            auto pts = c_c_intersection(cs[i], cs[j]);
            for (auto& p: pts) xs.push_back(p.x);
        }
    }
    sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end(), [](LD x, LD y) { return sgn(x - y) == 0; }), xs.end());
    LD ans = 0;
    FOR (i, 0, (int) xs.size() - 1) {
        LD xl = xs[i], xr = xs[i + 1];
        vector<CV> intv;
        FOR (k, 0, n) {
            auto& c = cs[k];
            if (sgn(c.p.x - c.r - xl) <= 0 && sgn(c.p.x + c.r - xr) >= 0) {
                auto t = pairwise_curves(c, xl, xr);
                intv.push_back(t.first); intv.push_back(t.second);
            }
        }
        sort(intv.begin(), intv.end());

        vector<LD> areas(intv.size());
        FOR (i, 0, intv.size()) areas[i] = cv_area(intv[i], xl, xr);

        int cc = 0;
        FOR (i, 0, intv.size()) {
            if (cc > 0) {
                ans += (intv[i].yl - intv[i - 1].yl + intv[i].yr - intv[i - 1].yr) * (xr - xl) / 2;
                ans += intv[i - 1].type * areas[i - 1];
                ans -= intv[i].type * areas[i];
            }
            cc += intv[i].type;
        }
    }
    return ans;
}
```

- 版本 2：复杂度 $O(n^2 \log n)$。
- 原理是：认为所求部分是一个奇怪的多边形 + 若干弓形。然后对于每个圆分别求贡献的弓形，并累加多边形有向面积。
- 同样可以魔改扫描线的部分，用于求周长、至少覆盖 k 次等等。
- 内含、内切、同一个圆的情况，通常需要特殊处理。
- 下面的代码是 k 圆覆盖。

```cpp
inline LD angle(const P& p) { return atan2(p.y, p.x); }

// 圆弧上的点
// p 是相对于圆心的坐标
// a 是在圆上的 atan2 [-PI, PI]
struct CP {
    P p; LD a; int t;
    CP() {}
    CP(P p, LD a, int t): p(p), a(a), t(t) {}
};
bool operator < (const CP& u, const CP& v) { return u.a < v.a; }
LD cv_area(LD r, const CP& q1, const CP& q2) {
    return (r * r * (q2.a - q1.a) - cross(q1.p, q2.p)) / 2;
}

LD ans[N];
void circle_union(const vector<C>& cs) {
    int n = cs.size();
    FOR (i, 0, n) {
        // 有相同的圆的话只考虑第一次出现
        bool ok = true;
        FOR (j, 0, i)
            if (sgn(cs[i].r - cs[j].r) == 0 && cs[i].p == cs[j].p) {
                ok = false;
                break;
            }
        if (!ok) continue;
        auto& c = cs[i];
        vector<CP> ev;
        int belong_to = 0;
        P bound = c.p + P(-c.r, 0);
        ev.emplace_back(bound, -PI, 0);
        ev.emplace_back(bound, PI, 0);
        FOR (j, 0, n) {
            if (i == j) continue;
            if (c_c_relation(c, cs[j]) <= 2) {
                if (sgn(cs[j].r - c.r) >= 0) // 完全被另一个圆包含，等于说叠了一层
                    belong_to++;
                continue;
            }
            auto its = c_c_intersection(c, cs[j]);
            if (its.size() == 2) {
                P p = its[1] - c.p, q = its[0] - c.p;
                LD a = angle(p), b = angle(q);
                if (sgn(a - b) > 0) {
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(bound, PI, -1);
                    ev.emplace_back(bound, -PI, 1);
                    ev.emplace_back(q, b, -1);
                } else {
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(q, b, -1);
                }
            }
        }
        sort(ev.begin(), ev.end());
        int cc = ev[0].t;
        FOR (j, 1, ev.size()) {
            int t = cc + belong_to;
            ans[t] += cross(ev[j - 1].p + c.p, ev[j].p + c.p) / 2;
            ans[t] += cv_area(c.r, ev[j - 1], ev[j]);
            cc += ev[j].t;
        }
    }
}
```

最小圆覆盖

- 随机增量。期望复杂度 $O(n)$。

```
P compute_circle_center(P a, P b) { return (a + b) / 2; }
bool p_in_circle(const P& p, const C& c) {
    return sgn(dist(p - c.p) - c.r) <= 0;
}
C min_circle_cover(const vector<P> &in) {
    vector<P> a(in.begin(), in.end());
    dbg(a.size());
    random_shuffle(a.begin(), a.end());
    P c = a[0]; LD r = 0; int n = a.size();
    FOR (i, 1, n) if (!p_in_circle(a[i], {c, r})) {
        c = a[i]; r = 0;
        FOR (j, 0, i) if (!p_in_circle(a[j], {c, r})) {
            c = compute_circle_center(a[i], a[j]);
            r = dist(a[j] - c);
            FOR (k, 0, j) if (!p_in_circle(a[k], {c, r})) {
                c = compute_circle_center(a[i], a[j], a[k]);
                r = dist(a[k] - c);
            }
        }
    }
    return {c, r};
}
```

圆的反演

```
C inv(C c, const P& o) {
    LD d = dist(c.p - o);
    assert(sgn(d) != 0);
    LD a = 1 / (d - c.r);
    LD b = 1 / (d + c.r);
    c.r = (a - b) / 2 * R2;
    c.p = o + (c.p - o) * ((a + b) * R2 / 2 / d);
    return c;
}
```

三维计算几何

```
struct P;
struct L;
typedef P V;

struct P {
    LD x, y, z;
    explicit P(LD x = 0, LD y = 0, LD z = 0): x(x), y(y), z(z) {}
    explicit P(const L& l);
};

struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
```

```cpp
};

struct F {
    P a, b, c;
    F() {}
    F(P a, P b, P c): a(a), b(b), c(c) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y, a.z + b.z); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y, a.z - b.z); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k, a.z * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k, a.z / k); }
inline int operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && (sgn(a.y - b.y) < 0 ||
                                  (sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0)));
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y) && !sgn(a.z - b.z); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << "," << p.z << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y >> p.z);
}

// --------------------------------------------
LD dist2(const P& p) { return p.x * p.x + p.y * p.y + p.z * p.z; }
LD dist(const P& p) { return sqrt(dist2(p)); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
P cross(const P& v, const P& w) {
    return P(v.y * w.z - v.z * w.y, v.z * w.x - v.x * w.z, v.x * w.y - v.y * w.x);
}
LD mix(const V& a, const V& b, const V& c) { return dot(a, cross(b, c)); }
```

旋转

```cpp
// 逆时针旋转 r 弧度
// axis = 0 绕 x 轴
// axis = 1 绕 y 轴
// axis = 2 绕 z 轴
P rotation(const P& p, const LD& r, int axis = 0) {
    if (axis == 0)
        return P(p.x, p.y * cos(r) - p.z * sin(r), p.y * sin(r) + p.z * cos(r));
    else if (axis == 1)
        return P(p.z * cos(r) - p.x * sin(r), p.y, p.z * sin(r) + p.x * cos(r));
    else if (axis == 2)
        return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r), p.z);
}
// n 是单位向量 表示旋转轴
// 模板是顺时针的
P rotation(const P& p, const LD& r, const P& n) {
    LD c = cos(r), s = sin(r), x = n.x, y = n.y, z = n.z;
    // dbg(c, s);
    return P((x * x * (1 - c) + c) * p.x + (x * y * (1 - c) + z * s) * p.y + (x * z * (1 - c) - y * s) * p.z,
             (x * y * (1 - c) - z * s) * p.x + (y * y * (1 - c) + c) * p.y + (y * z * (1 - c) + x * s) * p.z,
             (x * z * (1 - c) + y * s) * p.x + (y * z * (1 - c) - x * s) * p.y + (z * z * (1 - c) + c) * p.z);
}
```

线、面

函数相互依赖，所以交织在一起了。

```cpp
// 点在线段上  <= 0 包含端点 < 0 则不包含
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(dist2(cross(p - a, b - a))) && sgn(dot(p - a, p - b)) <= 0;
}
// 点到直线距离
LD dist_to_line(const P& p, const L& l) {
    return dist(cross(l.s - p, l.t - p)) / dist(l);
}
```

```
// 点到线段距离
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l.s);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}

P norm(const F& f) { return cross(f.a - f.b, f.b - f.c); }
int p_on_plane(const F& f, const P& p) { return sgn(dot(norm(f), p - f.a)) == 0; }

// 判两点在线段异侧 点在线段上返回 0 不共面无意义
int opposite_side(const P& u, const P& v, const L& l) {
    return sgn(dot(cross(P(l), u - l.s), cross(P(l), v - l.s))) < 0;
}

bool parallel(const L& a, const L& b) { return !sgn(dist2(cross(P(a), P(b)))); }
// 线段相交
int s_intersect(const L& u, const L& v) {
    return p_on_plane(F(u.s, u.t, v.s), v.t) &&
           opposite_side(u.s, u.t, v) &&
           opposite_side(v.s, v.t, u);
}
```

三维凸包

增量法。先将所有的点打乱顺序，然后选择四个不共面的点组成一个四面体，如果找不到说明凸包不存在。然后遍历剩余的点，不断更新凸包。对遍历到的点做如下处理。

1. 如果点在凸包内，则不更新。
2. 如果点在凸包外，那么找到所有原凸包上所有分隔了对于这个点可见面和不可见面的边，以这样的边的两个点和新的点创建新的面加入凸包中。

```
struct FT {
    int a, b, c;
    FT() { }
    FT(int a, int b, int c) : a(a), b(b), c(c) { }
};

bool p_on_line(const P& p, const L& l) {
    return !sgn(dist2(cross(p - l.s, P(l))));
}

vector<F> convex_hull(vector<P> &p) {
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    random_shuffle(p.begin(), p.end());
    vector<FT> face;
    FOR (i, 2, p.size()) {
        if (p_on_line(p[i], L(p[0], p[1]))) continue;
        swap(p[i], p[2]);
        FOR (j, i + 1, p.size())
            if (sgn(mix(p[1] - p[0], p[2] - p[1], p[j] - p[0]))) {
                swap(p[j], p[3]);
                face.emplace_back(0, 1, 2);
                face.emplace_back(0, 2, 1);
                goto found;
            }
    }
found:
    vector<vector<int>> mk(p.size(), vector<int>(p.size()));
    FOR (v, 3, p.size()) {
        vector<FT> tmp;
        FOR (i, 0, face.size()) {
            int a = face[i].a, b = face[i].b, c = face[i].c;
            if (sgn(mix(p[a] - p[v], p[b] - p[v], p[c] - p[v])) < 0) {
                mk[a][b] = mk[b][a] = v;
                mk[b][c] = mk[c][b] = v;
                mk[c][a] = mk[a][c] = v;
```

```cpp
            } else tmp.push_back(face[i]);
        }
        face = tmp;
        FOR (i, 0, tmp.size()) {
            int a = face[i].a, b = face[i].b, c = face[i].c;
            if (mk[a][b] == v) face.emplace_back(b, a, v);
            if (mk[b][c] == v) face.emplace_back(c, b, v);
            if (mk[c][a] == v) face.emplace_back(a, c, v);
        }
    }
    vector<F> out;
    FOR (i, 0, face.size())
        out.emplace_back(p[face[i].a], p[face[i].b], p[face[i].c]);
    return out;
}
```

字符串

SA

- n：串长
- m：字符集大小
- s[0..n−1]：字符串
- sa[1..n]：字典序第 i 小的是哪个后缀
- rank[0..n−1]：后缀 i 的排名
- height[i]：lcp(sa[i], sa[i−1])

```cpp
#include <bits/stdc++.h>
using namespace std;
#define rank _rank
const int N = 1e5+50;
int n, rank[N], sa[N], height[N], tmp[N], cnt[N];
char s[N];
void suffixarray(int n, int m)
{
    int i, j, k;
    n++;
    for (i = 0; i < n * 2 + 5; i++)
        rank[i] = sa[i] = height[i] = tmp[i] = 0;
    for (i = 0; i < m; i++)
        cnt[i] = 0;
    for (i = 0; i < n; i++)
        cnt[rank[i] = s[i]]++;
    for (i = 1; i < m; i++)
        cnt[i] += cnt[i-1];
    for (i = 0; i < n; i++)
        sa[--cnt[rank[i]]] = i;
    for (k = 1; k <= n; k <<= 1)
    {
        for (i = 0; i < n; i++)
        {
            j = sa[i]-k;
            if (j < 0)
                j += n;
            tmp[cnt[rank[j]]++] = j;
        }
        sa[tmp[cnt[0] = 0]] = j = 0;
        for (i = 1; i < n; i++)
        {
            if (rank[tmp[i]] != rank[tmp[i-1]] || rank[tmp[i] + k] != rank[tmp[i-1] + k])
                cnt[++j] = i;
            sa[tmp[i]] = j;
        }
        memcpy(rank, sa, n * sizeof(int));
        memcpy(sa, tmp, n * sizeof(int));
        if (j >= n-1)
            break;
    }
    for (j = rank[height[i = k = 0] = 0]; i < n-1; i++, k++)
        while (~k && s[i] != s[sa[j-1] + k])
            height[j] = k--, j = rank[sa[j] + 1];
}
```

KMP

```
struct kmp
{
    int s[1000010],t[1000010],Next[1000010];
    void Pre_KMP()
    {
        for (int i=0;i<=m;i++)
            Next[i]=0;
        int j=0,k=-1;
        Next[0]=-1;
        while(j<m)
        {
            if (k==-1||t[j]==t[k]) Next[++j]=++k;
            else k=Next[k];
        }
    }
    int KMP()
    {
        int i=0,j=0;
        while(i<n&&j<m)
        {
            if (j==-1||s[i]==t[j]) i++,j++;
            else j=Next[j];
        }
        if (j==m) return i-m;
        else return -1;
    }
}K;
```

Manacher

```
int RL[N << 1], b[N << 1], bcnt;
void manacher(int* a, int n) { // "abc" => "#a#b#a#"
    int r = 0, p = 0;
    for (int i=0; i<n; ++i) {
        if (i < r) RL[i] = min(RL[2 * p - i], r - i);
        else RL[i] = 1;
        while (i - RL[i] >= 0 && i + RL[i] < n && a[i - RL[i]] == a[i + RL[i]])
            RL[i]++;
        if (RL[i] + i - 1 > r) { r = RL[i] + i - 1; p = i; }
    }
    for (int i=0; i<n; ++i) --RL[i];
}
```

Trie

```
struct node {
    int nxt[M], dep, siz, fa;
} nodes[N];

int tot;
int newnode(int fa = 0) {
    memset(&nodes[++tot], 0, sizeof(node));
    if (fa) {
        nodes[tot].fa = fa;
        nodes[tot].dep = nodes[fa].dep + 1;
    }
    return tot;
}

int root;
void insert(char a[], int n) {
    int cur = root;
    for (int i=0; i<n; ++i) {
        int now = a[i] - 'A';
        if (!nodes[cur].nxt[now]) {
            nodes[cur].nxt[now] = newnode(cur);
        }
        cur = nodes[cur].nxt[now];
    }

    ++nodes[cur].siz;
}
```

杂项

日期

```
// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

子集枚举

- 枚举真子集

```
for (int s = (S - 1) & S; s; s = (s - 1) & S)
```

- 枚举大小为 k 的子集

```
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n) {
        f(t);
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

Java

Regex

```java
// Code which demonstrates the use of Java's regular expression libraries.
// This is a solution for
//
//   Loglan: a logical language
//   http://acm.uva.es/p/v1/134.html

import java.util.*;
import java.util.regex.*;

public class LogLan {

    public static void main(String args[]) {

        String regex = BuildRegex();
        Pattern pattern = Pattern.compile(regex);

        Scanner s = new Scanner(System.in);
        while (true) {

            // In this problem, each sentence consists of multiple lines, where the last
            // line is terminated by a period.  The code below reads lines until
            // encountering a line whose final character is a '.'.  Note the use of
            //
            //    s.length() to get length of string
            //    s.charAt() to extract characters from a Java string
            //    s.trim() to remove whitespace from the beginning and end of Java string
            //
            // Other useful String manipulation methods include
            //
            //    s.compareTo(t) < 0 if s < t, lexicographically
            //    s.indexOf("apple") returns index of first occurrence of "apple" in s
            //    s.lastIndexOf("apple") returns index of last occurrence of "apple" in s
            //    s.replace(c,d) replaces occurrences of character c with d
            //    s.startsWith("apple) returns (s.indexOf("apple") == 0)
            //    s.toLowerCase() / s.toUpperCase() returns a new lower/uppercased string
            //
            //    Integer.parseInt(s) converts s to an integer (32-bit)
            //    Long.parseLong(s) converts s to a long (64-bit)
            //    Double.parseDouble(s) converts s to a double

            String sentence = "";
            while (true) {
                sentence = (sentence + " " + s.nextLine()).trim();
                if (sentence.equals("#")) return;
                if (sentence.charAt(sentence.length() - 1) == '.') break;
            }

            // now, we remove the period, and match the regular expression

            String removed_period = sentence.substring(0, sentence.length() - 1).trim();
            if (pattern.matcher(removed_period).find()) {
                System.out.println("Good");
            } else {
                System.out.println("Bad!");
            }
        }
    }
}
```

Decimal Format

```java
// examples for printing floating point numbers

```

```java
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

public class DecFormat {
    public static void main(String[] args) {
        DecimalFormat fmt;

        // round to at most 2 digits, leave of digits if not needed
        fmt = new DecimalFormat("#.##");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345
        System.out.println(fmt.format(0.0)); // produces 0
        System.out.println(fmt.format(0.01)); // produces .1

        // round to precisely 2 digits
        fmt = new DecimalFormat("#.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces .00

        // round to precisely 2 digits, force leading zero
        fmt = new DecimalFormat("0.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces 0.00

        // round to precisely 2 digits, force leading zeros
        fmt = new DecimalFormat("000000000.00");
        System.out.println(fmt.format(12345.6789)); // produces 000012345.68
        System.out.println(fmt.format(12345.0)); // produces 000012345.00
        System.out.println(fmt.format(0.0)); // produces 000000000.00

        // force leading '+'
        fmt = new DecimalFormat("+0;-0");
        System.out.println(fmt.format(12345.6789)); // produces +12346
        System.out.println(fmt.format(-12345.6789)); // produces -12346
        System.out.println(fmt.format(0)); // produces +0

        // force leading positive/negative, pad to 2
        fmt = new DecimalFormat("positive 00;negative 0");
        System.out.println(fmt.format(1)); // produces "positive 01"
        System.out.println(fmt.format(-1)); // produces "negative 01"

        // qoute special chars (#)
        fmt = new DecimalFormat("text with '#' followed by #");
        System.out.println(fmt.format(12.34)); // produces "text with # followed by 12"

        // always show "."
        fmt = new DecimalFormat("#.#");
        fmt.setDecimalSeparatorAlwaysShown(true);
        System.out.println(fmt.format(12.34)); // produces "12.3"
        System.out.println(fmt.format(12)); // produces "12."
        System.out.println(fmt.format(0.34)); // produces "0.3"

        // different grouping distances:
        fmt = new DecimalFormat("#,####.###");
        System.out.println(fmt.format(123456789.123)); // produces "1,2345,6789.123"

        // scientific:
        fmt = new DecimalFormat("0.000E00");
        System.out.println(fmt.format(123456789.123)); // produces "1.235E08"
        System.out.println(fmt.format(-0.000234)); // produces "-2.34E-04"

        // using variable number of digits:
        fmt = new DecimalFormat("0");
        System.out.println(fmt.format(123.123)); // produces "123"
        fmt.setMinimumFractionDigits(8);
        System.out.println(fmt.format(123.123)); // produces "123.12300000"
        fmt.setMaximumFractionDigits(0);
        System.out.println(fmt.format(123.123)); // produces "123"

        // note: to pad with spaces, you need to do it yourself:
        // String out = fmt.format(...)
        // while (out.length() < targlength) out = " "+out;
    }
}
```

Sort

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Employee implements Comparable<Employee> {
    private int id;
    private String name;
    private int age;

    public Employee(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Employee o) {
        if (id > o.id) {
            return 1;
        } else if (id < o.id) {
            return -1;
        }
        return 0;
    }

    public static void main(String[] args) {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee(2, "Java", 20));
        list.add(new Employee(1, "C", 30));
        list.add(new Employee(3, "C#", 10));
        Collections.sort(list);
    }
}
```

扩栈（本地使用）

```cpp
#include <sys/resource.h>
void init_stack(){
    const rlim_t kStackSize = 512 * 1024 * 1024;
    struct rlimit rl;
    int result;
    result = getrlimit(RLIMIT_STACK, &rl);
    if (result == 0) {
        if (rl.rlim_cur < kStackSize) {
            rl.rlim_cur = kStackSize;
            result = setrlimit(RLIMIT_STACK, &rl);
            if (result != 0) {
                fprintf(stderr, "setrlimit returned result = %d\n", result);
            }
        }
    }
}
```

心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限

- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小（2 倍，4 倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序，初始化或者询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 不要把 int 开成 char
- 清零的时候全部用 0~n+1。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 `l + (r - l) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化

心态崩了第二季

- 点双时候考虑边粘在一起的情况，边双考虑点粘在一起的情况
- 边数组和点数组的大小经常不一样