

# 代码库

Blazar

2017 年 11 月 29 日

# 目录

<b>1 数学</b>	<b>2</b>		
1.1 快速求逆元 (内含 exgcd)	2		
1.2 中国剩余定理	2		
1.3 小步大步	2		
1.4 Miller Rabin 素数测试	3		
1.5 Pollard Rho 大数分解	3		
1.6 快速数论变换 (zky)	3		
1.7 原根	4		
1.8 线性递推	4		
1.9 直线下整点个数	4		
1.10 $1e9+7$ FFT	4		
1.11 自适应辛普森	5		
1.12 多项式求根	5		
<b>2 数据结构</b>	<b>6</b>		
2.1 lct	6		
2.2 树上莫队	6		
2.3 树状数组 kth	7		
2.4 虚树	7		
<b>3 图论</b>	<b>8</b>		
3.1 点双连通分量 (lyx)	8		
3.2 Hopcroft-Karp 求最大匹配	8		
3.3 KM 带权匹配	9		
3.4 2-SAT 问题	9		
3.5 有根树的同构	10		
3.6 Dominator Tree	10		
3.7 无向图最小割	11		
3.8 带花树	11		
<b>4 字符串</b>	<b>12</b>		
4.1 KMP 算法	12		
		4.2 扩展 KMP 算法	12
		4.3 AC 自动机	13
		4.4 后缀自动机	13
		4.4.1 广义后缀自动机 (多串)	13
		4.4.2 sam-ypm	14
		4.5 后缀数组	14
		4.6 Manacher	15
		4.7 循环串的最小表示	15
		<b>5 计算几何</b>	<b>15</b>
		5.1 二维几何	15
		5.2 凸包	16
		5.3 阿波罗尼茨圆	16
		5.4 三角形与圆交	16
		5.5 圆并	17
		5.6 整数半平面交	17
		5.7 三角形	18
		5.8 经纬度求球面最短距离	18
		5.9 长方体表面两点最短距离	18
		5.10 点到凸包切线	18
		5.11 直线与凸包的交点	19
		5.12 平面最近点对	19
		5.13 三维几何	19
		<b>6 其他</b>	<b>20</b>
		6.1 最小树形图	20
		6.2 DLX	20
		6.3 某年某月某日是星期几	21
		6.4 枚举大小为 $k$ 的子集	21
		6.5 环状最长公共子串	21
		6.6 LLMOD STL 内存清空开栈	22
		6.7 vimrc	22
		6.8 上下界网络流	22
		6.9 上下界费用流	23

6.10 Bernoulli 数 . . . . .	23
6.11 Java Hints . . . . .	23
<b>7 数学</b>	<b>24</b>
7.1 常用数学公式 . . . . .	24
7.1.1 求和公式 . . . . .	24
7.1.2 斐波那契数列 . . . . .	24
7.1.3 错排公式 . . . . .	24
7.1.4 莫比乌斯函数 . . . . .	24
7.1.5 伯恩赛德引理 . . . . .	25
7.1.6 五边形数定理 . . . . .	25
7.1.7 树的计数 . . . . .	25
7.1.8 欧拉公式 . . . . .	25
7.1.9 皮克定理 . . . . .	25
7.1.10 牛顿恒等式 . . . . .	25
7.2 平面几何公式 . . . . .	25
7.2.1 三角形 . . . . .	25
7.2.2 四边形 . . . . .	25
7.2.3 正 $n$ 边形 . . . . .	25
7.2.4 圆 . . . . .	25
7.2.5 棱柱 . . . . .	26
7.2.6 棱锥 . . . . .	26
7.2.7 棱台 . . . . .	26
7.2.8 圆柱 . . . . .	26
7.2.9 圆锥 . . . . .	26
7.2.10 圆台 . . . . .	26
7.2.11 球台 . . . . .	26
7.2.12 球扇形 . . . . .	26
7.3 积分表 . . . . .	26

## 数学

### 快速求逆元 (内含 exgcd)

使用条件:  $x \in [0, mod)$  并且  $x$  与  $mod$  互质

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(!b) return x = 1, y = 0, a;
    else {
        LL d = exgcd(b, a % b, x, y);
        LL t = x; x = y;
        y = t - a / b * y;
        return d;
    }
}
```

```
LL inv(LL a, LL p) {
    LL d, x, y;
    exgcd(a, p, d, x, y);
    return d == 1 ? (x + p) % p : -1;
}
```

### 中国剩余定理

返回结果:  $x \equiv r_i(mod\ p_i)$  ( $0 \leq i < n$ )

```
LL china(int n, int *a, int *m) {
    LL M = 1, d, x = 0, y;
    for(int i = 0; i < n; i++)
        M *= m[i];
    for(int i = 0; i < n; i++) {
        LL w = M / m[i];
        d = exgcd(m[i], w, d, y);
        y = (y % M + M) % M;
        x = (x + y * w % M * a[i]) % M;
    }
    while(x < 0) x += M;
    return x;
}
```

```
//merge Ax=B and ax=b to A'x=B'
void merge(LL &A, LL &B, LL a, LL b){
    LL x, y;
    sol(A, -a, b-B, x, y);
    A=lcm(A, a);
    B=(a*y+b)%A;
    B=(B+A)%A;
}
```

### 小步大步

返回结果:  $a^x = b (mod\ p)$  使用条件:  $p$  为质数

```

LL BSGS(LL a, LL b, LL p){
    LL m=0; for(;;m*m<=p;m++);
    map<LL, int> hash; hash[1]=0;
    LL e=1, amv=inv(pw(a,m,p),p);
    for(int i=1; i<m; i++){
        e=e*a%p;
        if(!hash.count(e))
            hash[e]=i;
        else break;
    }
    for(int i=0; i<m; i++){
        if(hash.count(b))
            return hash[b]+i*m;
        b=b*amv%p;
    }
    return -1;
}

LL solve2(LL a, LL b, LL p){
    //a^x=b (mod p)
    b%=p;
    LL e=1%p;
    for(int i=0; i<100; i++){
        if(e==b) return i;
        e=e*a%p;
    }
    int r=0;
    while(gcd(a,p)!=1){
        LL d=gcd(a,p);
        if(b%d) return -1;
        p/=d; b/=d; b=b*inv(a/d,p);
        r++;
    }
    LL res=BSGS(a,b,p);
    if(res==-1) return -1;
    return res+r;
}

```

## Miller Rabin 素数测试

```

const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool check(long long n, int base) {
    long long n2 = n - 1, res; int s = 0;
    while(n2 % 2 == 0) n2 >>= 1, s++;
    res = pw(base, n2, n);
    if((res == 1) || (res == n - 1)) return 1;
    while(s--) {
        res = mul(res, res, n);
        if(res == n - 1) return 1;
    }
    return 0; // n is not a strong pseudo prime
}

bool isprime(const long long &n) {
    if(n == 2) return true;
    if(n < 2 || n % 2 == 0) return false;
    for(int i = 0; i < 12 && BASE[i] < n; i++)
        if(!check(n, BASE[i])) return false;
    return true;
}

```

```

}

```

## Pollard Rho 大数分解

```

LL prho(LL n, LL c) {
    LL i = 1, k = 2, x = rand() % (n - 1) + 1, y = x;
    while(1) {
        i++; x = (x * x % n + c) % n;
        LL d = __gcd((y - x + n) % n, n);
        if(d > 1 && d < n) return d;
        if(y == x) return n;
        if(i == k) y = x, k <= 1;
    }
}

void factor(LL n, vector<LL>&fat) {
    if(n == 1) return;
    if(isprime(n)) {fat.push_back(n); return;}
    LL p = n;
    while(p >= n) p = prho(p, rand() % (n - 1) + 1);
    factor(p, fat); factor(n / p, fat);
}

```

## 快速数论变换 (zky)

返回结果:  $c_i = \sum_{0 \leq j \leq i} a_j \cdot b_{i-j} \pmod{n}$  ( $0 \leq i < n$ )

```

/*(mod,G)={{(81788929,7),(101711873,3),(167772161,3),
(377487361,7),(998244353,3),(1224736769,3),
(1300234241,3),(1484783617,5)}}*/
int mo = 998244353, G = 3;
void NTT(int a[], int n, int f) {
    for(register int i = 0; i < n; i++)
        if(i < rev[i])
            swap(a[i], a[rev[i]]);
    for (register int i = 2; i <= n; i <= 1) {
        static int exp[maxn];
        exp[0] = 1;
        exp[1] = pw(G, (mo - 1) / i);
        if(f == -1) exp[1] = pw(exp[1], mo - 2);
        for(register int k = 2; k < (i >> 1); k++)
            exp[k] = 1LL * exp[k - 1] * exp[1] % mo;
        for(register int j = 0; j < n; j += i) {
            for(register int k = 0; k < (i >> 1); k++) {
                register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
                register int A = pA, B = 1LL * pB * exp[k] % mo;
                pA = (A + B) % mo;
                pB = (A - B + mo) % mo;
            }
        }
    }
    if(f == -1) {
        int rv = pw(n, mo - 2) % mo;
        for(int i = 0; i < n; i++)
            a[i] = 1LL * a[i] * rv % mo;
    }
}

```

```

}
void mul(int m, int a[], int b[], int c[]) {
    int n = 1, len = 0;
    while(n < m)n <= 1, len++;
    for (int i = 1; i < n; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (len - 1));
    NTT(a, n, 1);
    NTT(b, n, 1);
    for(int i = 0; i < n; i++)
        c[i] = 1LL * a[i] * b[i] % mo;
    NTT(c, n, -1);
}

```

## 原根

```

vector<LL>fct;
bool check(LL x, LL g) {
    for(int i = 0; i < fct.size(); i++)
        if(pw(g, (x - 1) / fct[i], x) == 1)
            return 0;
    return 1;
}
LL findrt(LL x) {
    LL tmp = x - 1;
    for(int i = 2; i * i <= tmp; i++) {
        if(tmp % i == 0) {
            fct.push_back(i);
            while(tmp % i == 0)tmp /= i;
        }
    }
    if(tmp > 1) fct.push_back(tmp);
    // x is 1,2,4,p^n,2p^n
    // x has phi(phi(x)) primitive roots
    for(int i = 2; i < int(1e9); i++)
        if(check(x, i)) return i;
    return -1;
}

```

## 线性递推

//已知  $a_0, a_1, \dots, a_{m-1} \setminus \setminus$   
 $a_n = c_0 * a_{n-m} + \dots + c_{m-1} * a_{n-1} \setminus \setminus$   
 求  $a_n = v_0 * a_0 + v_1 * a_1 + \dots + v_{m-1} * a_{m-1} \setminus \setminus$

```

void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <= 1) {
        fill_n(u, m << 1, 0);
        int b(!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;

```

```

    } else {
        for(int i(0); i < m; i++) {
            for(int j(0), t(i + b); j < m; j++, t++) {
                u[t] = (u[t] + v[i] * v[j]) % p;
            }
        }
        for(int i((m << 1) - 1); i >= m; i--) {
            for(int j(0), t(i - m); j < m; j++, t++) {
                u[t] = (u[t] + c[j] * u[i]) % p;
            }
        }
    }
    copy(u, u + m, v);
}
//a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
for(int i(m); i < 2 * m; i++) {
    a[i] = 0;
    for(int j(0); j < m; j++) {
        a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
    }
}
for(int j(0); j < m; j++) {
    b[j] = 0;
    for(int i(0); i < m; i++) {
        b[j] = (b[j] + v[i] * a[i + j]) % p;
    }
}
for(int j(0); j < m; j++) {
    a[j] = b[j];
}
}
}

```

## 直线下整点个数

返回结果:  $\sum_{0 \leq i < n} \lfloor \frac{a+b \cdot i}{m} \rfloor$       使用条件:  $n, m > 0, a, b \geq 0$       时间复杂度:

$\mathcal{O}(n \log n)$

```

LL solve(LL n, LL a, LL b, LL m) {
    if(b == 0)
        return n * (a / m);
    if(a >= m || b >= m)
        return n * (a / m) + (n - 1) * n / 2 * (b / m) + solve(n, a % m, b % m, m);
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}

```

## 1e9+7 FFT

// double 精度对  $10^9 + 7$  取模最多可以做到  $2^{20}$   
const int MOD = 1000003;  
const double PI = acos(-1);  
typedef complex<double> Complex;  
const int N = 65536, L = 15, MASK = (1 << L) - 1;  
Complex w[N];  
void FFTInit() {

```

for (int i = 0; i < N; ++i)
    w[i] = Complex(cos(2 * i * PI / N), sin(2 * i * PI / N));
}
void FFT(Complex p[], int n) {
    for (int i = 1, j = 0; i < n - 1; ++i) {
        for (int s = n; j ^ s >= 1, ~j & s;);
        if (i < j) swap(p[i], p[j]);
    }
    for (int d = 0; (1 << d) < n; ++d) {
        int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
        for (int i = 0; i < n; i += m2) {
            for (int j = 0; j < m; ++j) {
                Complex &p1 = p[i + j + m], &p2 = p[i + j];
                Complex t = w[rm * j] * p1;
                p1 = p2 - t, p2 = p2 + t;
            } }
    }
    Complex A[N], B[N], C[N], D[N];
    void mul(int a[N], int b[N]) {
        for (int i = 0; i < N; ++i) {
            A[i] = Complex(a[i] >> L, a[i] & MASK);
            B[i] = Complex(b[i] >> L, b[i] & MASK);
        }
        FFT(A, N), FFT(B, N);
        for (int i = 0; i < N; ++i) {
            int j = (N - i) % N;
            Complex da = (A[i] - conj(A[j])) * Complex(0, -0.5),
                db = (A[i] + conj(A[j])) * Complex(0.5, 0),
                dc = (B[i] - conj(B[j])) * Complex(0, -0.5),
                dd = (B[i] + conj(B[j])) * Complex(0.5, 0);
            C[j] = da * dd + da * dc * Complex(0, 1);
            D[j] = db * dd + db * dc * Complex(0, 1);
        }
        FFT(C, N), FFT(D, N);
        for (int i = 0; i < N; ++i) {
            long long da = (long long)(C[i].imag() / N + 0.5) % MOD,
                db = (long long)(C[i].real() / N + 0.5) % MOD,
                dc = (long long)(D[i].imag() / N + 0.5) % MOD,
                dd = (long long)(D[i].real() / N + 0.5) % MOD;
            a[i] = ((dd << (L * 2)) + ((db + dc) << L) + da) % MOD;
        }
    }
}

```

## 自适应辛普森

```

double area(const double &left, const double &right) {
    double mid = (left + right) / 2;
    return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
}

double simpson(const double &left, const double &right,
    const double &eps, const double &area_sum) {
    double mid = (left + right) / 2;
    double area_left = area(left, mid);
    double area_right = area(mid, right);
    double area_total = area_left + area_right;

```

```

    if (std::abs(area_total - area_sum) < 15 * eps) {
        return area_total + (area_total - area_sum) / 15;
    }
    return simpson(left, mid, eps / 2, area_left)
        + simpson(mid, right, eps / 2, area_right);
}

double simpson(const double &left, const double &right, const double &eps) {
    return simpson(left, right, eps, area(left, right));
}

```

## 多项式求根

```

const double eps=1e-12;
double a[10][10];
typedef vector<double> vd;
int sgn(double x) { return x < -eps ? -1 : x > eps; }
double mypow(double x, int num){
    double ans=1.0;
    for(int i=1;i<=num;++i)ans*=x;
    return ans;
}
double f(int n, double x){
    double ans=0;
    for(int i=n;i>=0;--i)ans+=a[n][i]*mypow(x,i);
    return ans;
}
double getRoot(int n, double l, double r){
    if(sgn(f(n,l))==0)return l;
    if(sgn(f(n,r))==0)return r;
    double temp;
    if(sgn(f(n,l))>0)temp=-1;else temp=1;
    double m;
    for(int i=1;i<=10000;++i){
        m=(l+r)/2;
        double mid=f(n,m);
        if(sgn(mid)==0){
            return m;
        }
        if(mid*temp<0)l=m;else r=m;
    }
    return (l+r)/2;
}
vd did(int n){
    vd ret;
    if(n==1){
        ret.push_back(-1e10);
        ret.push_back(-a[n][0]/a[n][1]);
        ret.push_back(1e10);
        return ret;
    }
    vd mid=did(n-1);
    ret.push_back(-1e10);
    for(int i=0;i+1<mid.size();++i){
        int t1=sgn(f(n,mid[i])),t2=sgn(f(n,mid[i+1]));
        if(t1*t2>0)continue;

```

```

    ret.push_back(getRoot(n,mid[i],mid[i+1]));
}
ret.push_back(1e10);
return ret;
}
int main(){
    int n; scanf("%d",&n);
    for(int i=n;i>=0;--i){
        scanf("%lf",&a[n][i]);
    }
    for(int i=n-1;i>=0;--i)
        for(int j=0;j<=i;++j)a[i][j]=a[i+1][j+1]*(j+1);
    vd ans=did(n);
    sort(ans.begin(),ans.end());
    for(int i=1;i+1<ans.size();++i)printf("%.10f\n",ans[i]);
    return 0;
}

```

## 数据结构

### lct

```

struct LCT{
    int fa[N], c[N][2], rev[N], sz[N];
    void update(int o)
    {sz[o] = sz[c[o][0]] + sz[c[o][1]] + 1;}
    void pushdown(int o) {
        if(!rev[o]) return;
        rev[o] = 0;
        rev[c[o][0]] ^= 1;
        rev[c[o][1]] ^= 1;
        swap(c[o][0], c[o][1]);
    }
    bool ch(int o)
    {return o == c[fa[o]][1];}
    bool isroot(int o)
    {return c[fa[o]][0] != o && c[fa[o]][1] != o;}
    void setc(int x, int y, bool d) {
        if(x) fa[x] = y;
        if(y) c[y][d] = x;
    }
    void rotate(int x) {
        if(isroot(x)) return;
        int p = fa[x], d = ch(x);
        if(isroot(p)) fa[x] = fa[p];
        else setc(x, fa[p], ch(p));
        setc(c[x][d^1], p, d);
        setc(p, x, d^1);
        update(p); update(x);
    }
    void splay(int x) {
        static int q[N], top;
        int y = q[top = 1] = x;
        while(!isroot(y)) q[++top] = y = fa[y];
        while(top) pushdown(q[top--]);
    }
}

```

```

while(!isroot(x)) {
    if(!isroot(fa[x]))
        rotate(ch(fa[x]) == ch(x) ? fa[x] : x);
    rotate(x);
}
}
void access(int x) {
    for(int y = 0; x; y = x, x = fa[x])
        splay(x), c[x][1] = y, update(x);
}
void makeroot(int x)
{access(x), splay(x), rev[x] ^= 1;}
void link(int x, int y)
{makeroot(x), fa[x] = y, splay(x);}
void cut(int x, int y) {
    makeroot(x); access(y);
    splay(y); c[y][0] = fa[x] = 0;
}
}
};

```

### 树上莫队

```

int n, m;
int w[N];
vector<int> g[N];
int bid[N << 1];
struct Query{
    int l, r, extra, i;
    friend bool operator < (const Query &a, const Query &b) {
        if(bid[a.l] != bid[b.l])
            return bid[a.l] < bid[b.l];
        return a.r < b.r;
    }
} q[M];

void input(){
    vector<int> vs;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++){
        scanf("%d", &w[i]);
        vs.push_back(w[i]);
    }
    sort(vs.begin(), vs.end());
    vs.resize(unique(vs.begin(), vs.end()) - vs.begin());
    for(int i = 1; i <= n; i++)
        w[i] = lower_bound(vs.begin(), vs.end(), w[i]) - vs.begin() + 1;
    for(int i = 2; i <= m; i++){
        int a, b;
        scanf("%d%d", &a, &b);
        g[a].push_back(b);
        g[b].push_back(a);
    }
    for(int i = 1; i <= m; i++){
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].i = i;
    }
}

```

```

}

int dfs_clock, st[N], ed[N];
int fa[N][LOGN], dep[N];
int col[N << 1], id[N << 1];
void dfs(int x, int p){
    col[st[x] = ++dfs_clock] = w[x];
    id[st[x]] = x;
    fa[x][0] = p; dep[x] = dep[p] + 1;
    for(int i = 0; fa[x][i]; i++)
        fa[x][i + 1] = fa[fa[x][i]][i];
    for(auto y: g[x])
        if(y != p)
            dfs(y, x);
    col[ed[x] = ++dfs_clock] = w[x];
    id[ed[x]] = x;
}

int lca(int x, int y){
    if(dep[x] < dep[y]) swap(x, y);
    for(int i = LOGN - 1; i >= 0; i--)
        if(dep[fa[x][i]] >= dep[y])
            x = fa[x][i];
    if(x == y) return x;
    for(int i = LOGN - 1; i >= 0; i--)
        if(fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}

void prepare(){
    dfs_clock = 0;
    dfs(1, 0);
    int BS = (int)sqrt(dfs_clock + 0.5);
    for(int i = 1; i <= dfs_clock; i++)
        bid[i] = (i + BS - 1) / BS;
    for(int i = 1; i <= m; i++){
        int a = q[i].l;
        int b = q[i].r;
        int c = lca(a, b);
        if(st[a] > st[b]) swap(a, b);
        if(c == a){
            q[i].l = st[a];
            q[i].r = st[b];
            q[i].extra = 0;
        }
        else{
            q[i].l = ed[a];
            q[i].r = st[b];
            q[i].extra = c;
        }
    }
    sort(q + 1, q + m + 1);
}

```

```

int curans, ans[M], cnt[N];
bool state[N];
void rev(int x){
    int &c = cnt[col[x]];
    curans -= !!c;
    c += (state[id[x]] ^= 1) ? 1 : -1;
    curans += !!c;
}

void solve(){
    prepare();
    curans = 0;
    memset(cnt, 0, sizeof(cnt));
    memset(state, 0, sizeof(state));
    int l = 1, r = 0;
    for(int i = 1; i <= m; i++){
        while(l < q[i].l) rev(l++);
        while(l > q[i].l) rev(--l);
        while(r < q[i].r) rev(++r);
        while(r > q[i].r) rev(r--);
        if(q[i].extra) rev(st[q[i].extra]);
        ans[q[i].i] = curans;
        if(q[i].extra) rev(st[q[i].extra]);
    }
    for(int i = 1; i <= m; i++)
        printf("%d\n", ans[i]);
}

```

## 树状数组 kth

```

int find(int k){
    int cnt=0,ans=0;
    for(int i=22;i>=0;i--){
        ans+=(1<<i);
        if(ans>n || cnt+d[ans]>=k)ans-=(1<<i);
        else cnt+=d[ans];
    }
    return ans+1;
}

```

## 虚树

```

int a[maxn*2], sta[maxn*2];
int top=0, k;
void build(){
    top=0;
    sort(a, a+k, bydfn);
    k=unique(a, a+k)-a;
    sta[top++]=1; _n=k;
    for(int i=0; i<k; i++){
        int LCA=lca(a[i], sta[top-1]);
        while(dep[LCA]<dep[sta[top-1]]){
            if(dep[LCA]>=dep[sta[top-2]]){
                add_edge(LCA, sta[--top]);
                if(sta[top-1]!=LCA) sta[top++]=LCA;
            }
        }
    }
}

```



```

        break;
    }add_edge(sta[top-2],sta[top-1]);top--;
    }if(sta[top-1]!=a[i])sta[top++]=a[i];
}
while(top>1)
    add_edge(sta[top-2],sta[top-1]);top--;
for(int i=0;i<k;i++)inr[a[i]]=1;
}

```

## 图论

### 点双连通分量 (lyx)

```

#define SZ(x) ((int)x.size())
const int N = 400005; // N 开 2 倍点数, 因为新树会加入最多 n 个新点
const int M = 200005;
vector<int> g[N];
int bccno[N], bcc_cnt;
vector<int> bcc[N];
bool iscut[N];
struct Edge {
    int u, v;
} stk[M << 2];
int top, dfn[N], low[N], dfs_clock; // 注意栈大小为边数 4 倍
void dfs(int x, int fa)
{
    low[x] = dfn[x] = ++dfs_clock;
    int child = 0;
    for(int i = 0; i < SZ(g[x]); i++) {
        int y = g[x][i];
        if(!dfn[y]) {
            child++;
            stk[++top] = (Edge){x, y};
            dfs(y, x);
            low[x] = min(low[x], low[y]);
            if(low[y] >= dfn[x]) {
                iscut[x] = true;
                bcc[++bcc_cnt].clear();
                for(;;) {
                    Edge e = stk[top--];
                    if(bccno[e.u] != bcc_cnt) { bcc[bcc_cnt].push_back(e.u); bccno[e.u] =
↪ bcc_cnt; }
                    if(bccno[e.v] != bcc_cnt) { bcc[bcc_cnt].push_back(e.v); bccno[e.v] =
↪ bcc_cnt; }
                    if(e.u == x && e.v == y) break;
                }
            }
        } else if(y != fa && dfn[y] < dfn[x]) {
            stk[++top] = (Edge){x, y};
            low[x] = min(low[x], dfn[y]);
        }
    }
    if(fa == 0 && child == 1) iscut[x] = false;
}
void find_bcc() // 求点双连通分量, 需要时手动 1 到 n 清空, 1-based

```

```

{
    memset(dfn, 0, sizeof(dfn));
    memset(iscut, 0, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
    dfs_clock = bcc_cnt = 0;
    for(int i = 1; i <= n; i++)
        if(!dfn[i])
            dfs(i, 0);
}
vector<int> G[N];
void prepare() { // 建出缩点后的树
    for(int i = 1; i <= n + bcc_cnt; i++)
        G[i].clear();
    for(int i = 1; i <= bcc_cnt; i++) {
        int x = i + n;
        for(int j = 0; j < SZ(bcc[i]); j++) {
            int y = bcc[i][j];
            G[x].push_back(y);
            G[y].push_back(x);
        }
    }
}

```

### Hopcroft-Karp 求最大匹配

```

int matchx[N], matchy[N], level[N];

bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
            matchx[x] = y;
            matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}

int solve() {
    std::fill(matchx, matchx + n, -1);
    std::fill(matchy, matchy + m, -1);
    for (int answer = 0; ; ) {
        std::vector<int> queue;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                queue.push_back(i);
            } else {
                level[i] = -1;
            }
        }
        for (int head = 0; head < (int)queue.size(); ++head) {
            int x = queue[head];

```

```

        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            int w = matchy[y];
            if (w != -1 && level[w] < 0) {
                level[w] = level[x] + 1;
                queue.push_back(w);
            }
        }
    }
    int delta = 0;
    for (int i = 0; i < n; ++i) {
        if (matchx[i] == -1 && dfs(i)) {
            delta++;
        }
    }
    if (delta == 0) {
        return answer;
    } else {
        answer += delta;
    }
}
}

```

## KM 带权匹配

**注意事项：**最小权完美匹配，复杂度为  $O(|V|^3)$ 。

```

int DFS(int x){
    visx[x] = 1;
    for (int y = 1; y <= ny; y++){
        if (visy[y]) continue;
        int t = lx[x] + ly[y] - w[x][y];
        if (t == 0) {
            visy[y] = 1;
            if (link[y] == -1 || DFS(link[y])){
                link[y] = x;
                return 1;
            }
        }
        else slack[y] = min(slack[y], t);
    }
    return 0;
}

int KM(){
    int i, j;
    memset(link, -1, sizeof(link));
    memset(ly, 0, sizeof(ly));
    for (i = 1; i <= nx; i++){
        for (j = 1; lx[i] = -inf; j <= ny; j++){
            lx[i] = max(lx[i], w[i][j]);
        }
    }
    for (int x = 1; x <= nx; x++){
        for (i = 1; i <= ny; i++) slack[i] = inf;
        while (true) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));

```

```

            if (DFS(x)) break;
            int d = inf;
            for (i = 1; i <= ny; i++){
                if (!visy[i] && d > slack[i]) d = slack[i];
            }
            for (i = 1; i <= nx; i++){
                if (visx[i]) lx[i] -= d;
            }
            for (i = 1; i <= ny; i++){
                if (visy[i]) ly[i] += d;
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for (i = 1; i <= ny; i++)
        if (link[i] > -1) res += w[link[i]][i];
    return res;
}

```

## 2-SAT 问题

```

int stamp, comps, top;
int dfn[N], low[N], comp[N], stack[N];

void add(int x, int a, int y, int b) {
    edge[x << 1 | a].push_back(y << 1 | b);
}

void tarjan(int x) {
    dfn[x] = low[x] = ++stamp;
    stack[top++] = x;
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = std::min(low[x], low[y]);
        } else if (!comp[y]) {
            low[x] = std::min(low[x], dfn[y]);
        }
    }
    if (low[x] == dfn[x]) {
        comps++;
        do {
            int y = stack[--top];
            comp[y] = comps;
        } while (stack[top] != x);
    }
}

bool solve() {
    int counter = n + n + 1;
    stamp = top = comps = 0;
    std::fill(dfn, dfn + counter, 0);
    std::fill(comp, comp + counter, 0);
    for (int i = 0; i < counter; ++i) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
}

```

```

}
for (int i = 0; i < n; ++i) {
    if (comp[i << 1] == comp[i << 1 | 1]) {
        return false;
    }
    answer[i] = (comp[i << 1 | 1] < comp[i << 1]);
}
return true;
}

```

## 有根树的同构

```
const unsigned long long MAGIC = 4423;
```

```
unsigned long long magic[N];
std::pair<unsigned long long, int> hash[N];
```

```

void solve(int root) {
    magic[0] = 1;
    for (int i = 1; i <= n; ++i) {
        magic[i] = magic[i - 1] * MAGIC;
    }
    std::vector<int> queue;
    queue.push_back(root);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            queue.push_back(y);
        }
    }
    for (int index = n - 1; index >= 0; --index) {
        int x = queue[index];
        hash[x] = std::make_pair(0, 0);

        std::vector<std::pair<unsigned long long, int> > value;
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            value.push_back(hash[y]);
        }
        std::sort(value.begin(), value.end());

        hash[x].first = hash[x].first * magic[1] + 37;
        hash[x].second++;
        for (int i = 0; i < (int)value.size(); ++i) {
            hash[x].first = hash[x].first * magic[value[i].second] + value[i].first;
            hash[x].second += value[i].second;
        }
        hash[x].first = hash[x].first * magic[1] + 41;
        hash[x].second++;
    }
}

```

## Dominator Tree

```

class Edge{
public:
    int size;
    int begin[MAXN], dest[MAXM], next[MAXM];
    void clear(int n){
        size = 0;
        fill(begin, begin + n, -1);
    }
    Edge(int n = MAXN){
        clear(n);
    }
    void add_edge(int u, int v){
        dest[size] = v;
        next[size] = begin[u];
        begin[u] = size++;
    }
};

class dominator{
public:
    int dfn[MAXN], sdom[MAXN], idom[MAXN], id[MAXN], f[MAXN], fa[MAXN], smin[MAXN],
    ↪ stamp;

    void predfs(int x, const Edge &succ){
        id[dfn[x] = stamp++] = x;
        for(int i = succ.begin[x]; ~i; i = succ.next[i]){
            int y = succ.dest[i];
            if(dfn[y] < 0){
                f[y] = x;
                predfs(y, succ);
            }
        }
    }
    int getfa(int x){
        if(fa[x] == x)
            return x;
        int ret = getfa(fa[x]);
        if(dfn[sdom[smin[fa[x]]]] < dfn[sdom[smin[x]]])
            smin[x] = smin[fa[x]];
        return fa[x] = ret;
    }
    void solve(int s, int n, const Edge &succ){
        fill(dfn, dfn + n, -1);
        fill(idom, idom + n, -1);
        static Edge pred, tmp;
        pred.clear(n);
        for(int i = 0; i < n; ++i)
            for(int j = succ.begin[i]; ~j; j = succ.next[j])
                pred.add_edge(succ.dest[j], i);
        stamp = 0;
        tmp.clear(n);
        predfs(s, succ);
        for(int i = 0; i < stamp; ++i)
            fa[id[i]] = smin[id[i]] = id[i];
    }
}

```

```

for(int o = stamp - 1; o >= 0; --o){
    int x = id[o];
    if(o){
        sdom[x] = f[x];
        for(int i = pred.begin[x]; ~i; i = pred.next[i]){
            int p = pred.dest[i];
            if(dfn[p] < 0)
                continue;
            if(dfn[p] > dfn[x]){
                getfa(p);
                p = sdom[smin[p]];
            }
            if(dfn[sdom[x]] > dfn[p])
                sdom[x] = p;
        }
        tmp.add_edge(sdom[x], x);
    }
    while(~tmp.begin[x]){
        int y = tmp.dest[tmp.begin[x]];
        tmp.begin[x] = tmp.next[tmp.begin[x]];
        getfa(y);
        if(x != sdom[smin[y]])
            idom[y] = smin[y];
        else
            idom[y] = x;
    }
    for(int i = succ.begin[x]; ~i; i = succ.next[i])
        if(f[succ.dest[i]] == x)
            fa[succ.dest[i]] = x;
    }
    idom[s] = s;
    for(int i = 1; i < stamp; ++i){
        int x = id[i];
        if(idom[x] != sdom[x])
            idom[x] = idom[idom[x]];
    }
}
};

```

## 无向图最小割

```
int node[N], dist[N];
bool visit[N];

int solve(int n) {
    int answer = INT_MAX;
    for (int i = 0; i < n; ++i) {
        node[i] = i;
    }
    while (n > 1) {
        int max = 1;
        for (int i = 0; i < n; ++i) {
            dist[node[i]] = graph[node[0]][node[i]];
            if (dist[node[i]] > dist[node[max]]) {
                max = i;
            }
        }
    }
}
```

```

    }
    int prev = 0;
    memset(visit, 0, sizeof(visit));
    visit[node[0]] = true;
    for (int i = 1; i < n; ++i) {
        if (i == n - 1) {
            answer = std::min(answer, dist[node[max]]);
            for (int k = 0; k < n; ++k) {
                graph[node[k]][node[prev]] =
                    (graph[node[prev]][node[k]] + graph[node[k]][node[max]]);
            }
            node[max] = node[--n];
        }
        visit[node[max]] = true;
        prev = max;
        max = -1;
        for (int j = 1; j < n; ++j) {
            if (!visit[node[j]]) {
                dist[node[j]] += graph[node[prev]][node[j]];
                if (max == -1 || dist[node[max]] < dist[node[j]]) {
                    max = j;
                }
            }
        }
    }
}

return answer;
}

```

## 带花树

```
int match[N], belong[N], next[N], mark[N], visit[N];
std::vector<int> queue;
int find(int x) {
    if (belong[x] != x)
        belong[x] = find(belong[x]);
    return belong[x];
}
void merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x != y)
        belong[x] = y;
}
int lca(int x, int y) {
    static int stamp = 0;
    stamp++;
    while (true) {
        if (x != -1) {
            x = find(x);
            if (visit[x] == stamp)
                return x;
            visit[x] = stamp;
            if (match[x] != -1)
                x = next[match[x]];
        }
        else
```

```

        x = -1;
    }
    std::swap(x, y);
}
}
void group(int a, int p) {
    while (a != p) {
        int b = match[a], c = next[b];
        if (find(c) != p)
            next[c] = b;
        if (mark[b] == 2) {
            mark[b] = 1;
            queue.push_back(b);
        }
        if (mark[c] == 2) {
            mark[c] = 1;
            queue.push_back(c);
        }
        merge(a, b);
        merge(b, c);
        a = c;
    }
}
void augment(int source) {
    queue.clear();
    for (int i = 0; i < n; ++i) {
        next[i] = visit[i] = -1;
        belong[i] = i;
        mark[i] = 0;
    }
    mark[source] = 1;
    queue.push_back(source);
    for (int head = 0; head < (int)queue.size() && match[source] == -1; ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            if (match[x] == y || find(x) == find(y) || mark[y] == 2) {
                continue;
            }
            if (mark[y] == 1) {
                int r = lca(x, y);
                if (find(x) != r)
                    next[x] = y;
                if (find(y) != r)
                    next[y] = x;
                group(x, r);
                group(y, r);
            } else if (match[y] == -1) {
                next[y] = x;
                for (int u = y; u != -1; ) {
                    int v = next[u];
                    int mv = match[v];
                    match[v] = u;
                    match[u] = v;
                    u = mv;
                }
            }
        }
    }
}

```

```

    }
    break;
} else {
    next[y] = x;
    mark[y] = 2;
    mark[match[y]] = 1;
    queue.push_back(match[y]);
}
}
}
}
int solve() {
    std::fill(match, match + n, -1);
    for (int i = 0; i < n; ++i)
        if (match[i] == -1)
            augment(i);
    int answer = 0;
    for (int i = 0; i < n; ++i)
        answer += (match[i] != -1);
    return answer;
}

```

## 字符串

### KMP 算法

```

void getnex(char *s, int *nex){
    int n = strlen(s + 1);
    for(int j = 0, i = 2; i <= n; i++){
        while(j && s[j + 1] != s[i]) j = nex[j];
        if(s[i] == s[j + 1]) j++;
        nex[i] = j;
    }
}

```

### 扩展 KMP 算法

```

//nex[i] 表示 s 和其后缀 s[i, n] 的 lcp 的长度
void getnext(char s[], int n, int nex[])
{
    nex[1] = n;
    int &t = nex[2] = 0;
    for(; t + 2 <= n && s[1 + t] == s[2 + t]; t++);
    int pos = 2;
    for(int i = 3; i <= n; i++){
        if(i + nex[i - pos + 1] < pos + nex[pos])
            nex[i] = nex[i - pos + 1];
        else{
            int j = max(0, nex[pos] + pos - i);
            for(; i + j <= n && s[i + j] == s[j + 1]; j++);
            nex[i] = j; pos = i;
        }
    }
}

```

```
//extend[i] 表示 s2 和 s1 后缀 s1[i, n] 的 lcp 的长度
void getextend(char s1[], char s2[], int extend[])
{
    int n = strlen(s1 + 1), m = strlen(s2 + 1);
    getnext(s2, m, next);
    int &t = extend[1] = 0, pos = 1;
    for(; t < n && t < m && s1[1 + t] == s2[1 + t]; t++);
    for(int i = 2; i <= n; i++){
        if(i + nex[i - pos + 1] < pos + extend[pos])
            extend[i] = nex[i - pos + 1];
        else{
            int j = max(0, extend[pos] + pos - i);
            for(; i + j <= n && j < m && s1[i + j] == s2[j + 1]; j++);
            extend[i] = j; pos = i;
        }
    }
}
```

## AC 自动机

```
const int C = 26, L = 1e5 + 5, N = 5e5 + 10;
int n, root, cnt, fail[N], son[N][26], num[N];
char s[L];
inline int newNode(){
    cnt++; fail[cnt] = num[cnt] = 0;
    memset(son[cnt], 0, sizeof(son[cnt]));
    return cnt;
}
void insert(char *s){
    int n = strlen(s + 1), now = 1;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        if(!son[now][c]) son[now][c] = newNode();
        now = son[now][c];
    }
    num[now]++;
}
void getfail(){
    static queue<int> Q;
    fail[root] = 0;
    Q.push(root);
    while(!Q.empty()){
        int now = Q.front();
        Q.pop();
        for(int i = 0; i < C; i++)
            if(son[now][i]){
                Q.push(son[now][i]);
                int p = fail[now];
                while(!son[p][i]) p = fail[p];
                fail[son[now][i]] = son[p][i];
            }
        else son[now][i] = son[fail[now]][i];
    }
}
int main(){
    cnt = 0; root = newNode();
```

```
scanf("%d", &n);
for(int i = 0; i < C; i++) son[0][i] = 1;
for(int i = 1; i <= n; i++){
    scanf("%s", s + 1);
    insert(s);
}
getfail();
return 0;
}
```

## 后缀自动机

### 广义后缀自动机（多串）

注意事项：空间是插入字符串总长度的 2 倍并注意字符集大小。

```
const int N = 251010, C = 26;
int tot, las, root;
struct Node
{
    int son[C], len, par;
    void clear(){
        memset(son, 0, sizeof(son));
        par = len = 0;
    }
}node[N << 1];
inline int newNode(){return node[++tot].clear(), tot;}
void extend(int c)
{
    int p = las;
    if (node[p].son[c]) {
        int q = node[p].son[c];
        if (node[p].len + 1 == node[q].len) las = q;
        else{
            int nq = newNode();
            las = nq; node[nq] = node[q];
            node[nq].len = node[p].len + 1; node[q].par = nq;
            for (; p && node[p].son[c] == q; p = node[p].par)
                node[p].son[c] = nq;
        }
    }
    else{ // Naive Suffix Automaton
        int np = newNode();
        las = np; node[np].len = node[p].len + 1;
        for (; p && !node[p].son[c]; p = node[p].par)
            node[p].son[c] = np;
        if (!p) node[np].par = root;
        else{
            int q = node[p].son[c];
            if (node[p].len + 1 == node[q].len)
                node[np].par = q;
            else{
                int nq = newNode();
                node[nq] = node[q];
                node[nq].len = node[p].len + 1;
```

```

        node[q].par = node[np].par = nq;
        for (; p && node[p].son[c] == q; p = node[p].par)
            node[p].son[c] = nq;
    }
}
}
}
void add(char *s)
{
    int len = strlen(s + 1); las = root;
    for(int i = 1; i <= len; i++) extend(s[i] - 'a');
}

```

### sam-ypm

#### sam-nsustr

//SAM 利用后缀树进行计算, 由儿子向 parent 更新

void extend(int c); //传入转化为数字之后的字符, 从 0 开始

```

int main(){
    static char s[N];
    while(scanf("%s", s + 1) == 1){
        tot = 0;
        root = las = newNode();
        int n = strlen(s + 1);
        for(int i = 1; i <= n; i++) extend(s[i] - 'a');
        static int cnt[N], order[N << 1];
        memset(cnt, 0, sizeof(*cnt) * (n + 5));
        for(int i = 1; i <= tot; i++) cnt[node[i].len]++;
        for(int i = 1; i <= n; i++) cnt[i] += cnt[i - 1];
        for(int i = tot; i; i--) order[ cnt[node[i].len]-- ] = i;
        static int dp[N]; memset(dp, 0, sizeof(dp));
        //dp[i] 为长度为 i 的子串中出现次数最多的串的出现次数
        for(int now = root, i = 1; i <= n; i++){
            now = node[now].son[s[i] - 'a'];
            node[now].count++;
        }
        for(int i = tot; i; i--){
            Node &now = node[order[i]];
            dp[now.len] = max(dp[now.len], now.count);
            node[now.par].count += now.count;
        }
        for(int i = n - 1; i; i--) dp[i] = max(dp[i], dp[i + 1]);
        for(int i = 1; i <= n; i++) printf("%d\n", dp[i]);
    }
}

```

#### sam-lcs

```

int main(){
    static char s[N];
    scanf("%s", s + 1);
    tot = 0; root = las = newNode();
    int n = strlen(s + 1);
    for(int i = 1; i <= n; i++)
        extend(s[i] - 'a');
}

```

```

static int cnt[N], order[N << 1];
memset(cnt, 0, sizeof(*cnt) * (n + 5));
for(int i = 1; i <= tot; i++) cnt[node[i].len]++;
for(int i = 1; i <= n; i++) cnt[i] += cnt[i - 1];
for(int i = tot; i; i--) order[ cnt[node[i].len]-- ] = i;
static int ANS[N << 1], dp[N << 1];
memset(dp, 0, sizeof(*dp) * (tot + 5));
for(int i = 1; i <= tot; i++) ANS[i] = node[i].len;
while(scanf("%s", s + 1) == 1){
    n = strlen(s + 1);
    for(int now = root, len = 0, i = 1; i <= n; i++){
        int c = s[i] - 'a';
        while(now != root && !node[now].son[c])
            now = node[now].par;
        if(node[now].son[c]){
            len = min(len, node[now].len) + 1;
            now = node[now].son[c];
        }
        else len = 0;
        dp[now] = max(dp[now], len);
    }
    for(int i = tot; i; i--){
        int now = order[i];
        dp[node[now].par] = max(dp[node[now].par], dp[now]);
        ANS[now] = min(ANS[now], dp[now]);
        dp[now] = 0;
    }
}
int ans = 0;
for(int i = 1; i <= tot; i++) ans = max(ans, ANS[i]);
printf("%d\n", ans);
}

```

### 后缀数组

注意事项:  $\mathcal{O}(n \log n)$  倍增构造。

```

#define ws wws
const int MAXN = 201010;
int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];
int sa[MAXN], rk[MAXN], height[MAXN];
char s[MAXN];
inline bool cmp(int *r, int a, int b, int l)
{return r[a] == r[b] && r[a + l] == r[b + l];}
void SA(char *r, int *sa, int n, int m){
    int *x = wa, *y = wb;
    for(int i = 1; i <= m; i++) ws[i] = 0;
    for(int i = 1; i <= n; i++) ws[x[i]] = r[i]++;
    for(int i = 1; i <= m; i++) ws[i] += ws[i - 1];
    for(int i = n; i > 0; i--) sa[ ws[x[i]]-- ] = i;
    for(int j = 1, p = 0; p < n; j <= 1, m = p){
        p = 0;
        for(int i = n - j + 1; i <= n; i++) y[++p] = i;
        for(int i = 1; i <= n; i++) if(sa[i] > j) y[++p] = sa[i] - j;
        for(int i = 1; i <= n; i++) wv[i] = x[y[i]];
    }
}

```

```

    for(int i = 1; i <= m; i++)ws[i] = 0;
    for(int i = 1; i <= n; i++)ws[wv[i]]++;
    for(int i = 1; i <= m; i++)ws[i] += ws[i - 1];
    for(int i = n; i > 0; i--)sa[ws[wv[i]]--] = y[i];
    swap(x, y); x[sa[1]] = p = 1;
    for(int i = 2; i <= n; i++)
        x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p : ++p;
}
}
void getheight(char *r, int *sa, int *rk, int *h, int n){
    for(int i = 1; i <= n; i++) rk[sa[i]] = i;
    for(int i = 1, p = 0; i <= n; i++, p ? p-- : 0){
        int j = sa[rk[i] - 1];
        while(r[i + p] == r[j + p]) p++;
        h[rk[i]] = p;
    }
}

```

## Manacher

注意事项：1-based 算法，请注意下标。

```

void manacher(char *st){
    static char s[N << 1];
    static int p[N << 1];
    int n = strlen(st + 1);
    s[0] = '$'; s[1] = '#';
    for(int i = 1; i <= n; i++)
        s[i << 1] = st[i], s[(i << 1) + 1] = '#';
    s[(n = n * 2 + 1) + 1] = 0;
    int pos, mx = 0, res = 0;
    for(int i = 1; i <= n; i++){
        p[i] = (mx > i) ? min(p[pos * 2 - i], mx - i) : 1;
        while(s[i + p[i]] == s[i - p[i]]) p[i]++;
        if(p[i] + i - 1 > mx) mx = p[i] + i - 1, pos = i;
    }
}

```

## 循环串的最小表示

注意事项：0-Based 算法，请注意下标。

```

int getmin(char *s, int n){// 0-base
    int i = 0, j = 1, k = 0;
    while(i < n && j < n && k < n){
        int x = i + k; if(x >= n) x -= n;
        int y = j + k; if(y >= n) y -= n;
        if(s[x] == s[y]) k++;
        else{
            if(s[x] > s[y]) i += k + 1;
            else j += k + 1;
            if(i == j) j++;
            k = 0;
        }
    }
}

```

```

    return min(i, j);
}

```

## 计算几何

### 二维几何

```

// 求圆与直线的交点
bool isCL(Circle a, Line l, P &p1, P &p2) {
    D x = (l.s - a.o) % l.d,
      y = l.d.sqrLen(),
      d = x * x - y * ((l.s - a.o).sqrLen() - a.r * a.r);
    if (sign(d) < 0) return false;
    P p = l.s - x / y * l.d, delta = sqrt(max((D)0., d)) / y * l.d;
    p1 = p + delta, p2 = p - delta;
    return true;
}

// 求圆与圆的交面积
D areaCC(const Circle &c1, const Circle &c2) {
    D d = (c1.o - c2.o).len();
    if (sign(d - (c1.r + c2.r)) >= 0) {
        return 0;
    }
    if (sign(d - abs(c1.r - c2.r)) <= 0) {
        D r = min(c1.r, c2.r);
        return r * r * pi;
    }
    D x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
      t1 = acos(min(1., max(-1., x / c1.r))), t2 = acos(min(1., max(-1., (d - x) /
    ↪ c2.r)));
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}

// 求圆与圆的交点，注意调用前要先判定重圆
bool isCC(Circle a, Circle b, P &p1, P &p2) {
    D s1 = (a.o - b.o).len();
    if (sign(s1 - a.r - b.r) > 0 || sign(s1 - abs(a.r - b.r)) < 0) return false;
    D s2 = (a.r * a.r - b.r * b.r) / s1;
    D aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
    P o = aa / (aa + bb) * (b.o - a.o) + a.o;
    P delta = sqrt(max(0., a.r * a.r - aa * aa)) * (b.o - a.o).zoom(1).rev();
    p1 = o + delta, p2 = o - delta;
    return true;
}

// 求点到圆的切点，按关于点的顺时针方向返回两个点，rev 必须是 (-y, x)
bool tanCP(const Circle &c, const P &p0, P &p1, P &p2) {
    D x = (p0 - c.o).sqrLen(), d = x - c.r * c.r;
    if (d < eps) return false; // 点在圆上认为没有切点
    P p = c.r * c.r / x * (p0 - c.o);
    P delta = (-c.r * sqrt(d) / x * (p0 - c.o)).rev();
    p1 = c.o + p + delta;
    p2 = c.o + p - delta;
    return true;
}

// 求圆到圆的外共切线，按关于 c1.o 的顺时针方向返回两条线，rev 必须是 (-y, x)
vector<Line> extanCC(const Circle &c1, const Circle &c2) {

```



```

vector<Line> ret;
if (sign(c1.r - c2.r) == 0) {
    P dir = c2.o - c1.o;
    dir = (c1.r / dir.len() * dir).rev();
    ret.push_back(Line(c1.o + dir, c2.o - c1.o));
    ret.push_back(Line(c1.o - dir, c2.o - c1.o));
} else {
    P p = 1. / (c1.r - c2.r) * (-c2.r * c1.o + c1.r * c2.o);
    P p1, p2, q1, q2;
    if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) {
        if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
        ret.push_back(Line(p1, q1 - p1));
        ret.push_back(Line(p2, q2 - p2));
    }
}
return ret;
}
// 求圆到圆的内共切线, 按关于 c1.o 的顺时针方向返回两条线, rev 必须是 (-y, x)
vector<Line> intanCC(const Circle &c1, const Circle &c2) {
    vector<Line> ret;
    P p = 1. / (c1.r + c2.r) * (c2.r * c1.o + c1.r * c2.o);
    P p1, p2, q1, q2;
    if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { // 两圆相切认为没有切线
        ret.push_back(Line(p1, q1 - p1));
        ret.push_back(Line(p2, q2 - p2));
    }
    return ret;
}
bool contain(vector<P> poly, P p) { // 判断点 p 是否被多边形包含, 包括落在边界上
    int ret = 0, n = poly.size();
    for(int i = 0; i < n; ++i) {
        P u = poly[i], v = poly[(i + 1) % n];
        if (onSeg(p, u, v)) return true; // 在边界上
        if (sign(u.y - v.y) <= 0) swap(u, v);
        if (sign(p.y - u.y) > 0 || sign(p.y - v.y) <= 0) continue;
        ret += sign((v - p) * (u - p)) > 0;
    }
    return ret & 1;
}
vector<P> convexCut(const vector<P>&ps, Line l) {
    // 用半平面 (s,d) 的逆时针方向去切凸多边形
    vector<P> qs;
    int n = ps.size();
    for (int i = 0; i < n; ++i) {
        Point p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = sign(l.d * (p1 - l.s)), d2 = sign(l.d * (p2 - l.s));
        if (d1 >= 0) qs.push_back(p1);
        if (d1 * d2 < 0) qs.push_back(isLL(Line(p1, p2 - p1), l));
    }
    return qs;
}

```

## 凸包

```

inline bool turn_left(const Point &a, const Point &b, const Point &c) {
    return sgn(det(b - a, c - a)) >= 0;
}

void convex_hull(vector<Data> p, vector<Data> &res) {
    int n = (int)p.size(), cnt = 0;
    sort(p.begin(), p.end(), [&](const Data &a, const Data &b) {
        if (fabs(a.p.x - b.p.x) < eps) return a.p.y > b.p.y;
        return a.p.x < b.p.x; });
    res.clear();
    for(int i = 0; i < n; i++) {
        while(cnt > 1 && turn_left(res[cnt - 2].p, p[i].p, res[cnt - 1].p)) {
            cnt--;
            res.pop_back();
        }
        res.push_back(p[i]);
        ++cnt;
    }
    int fixed = cnt;
    for(int i = n - 2; i >= 0; i--) {
        while(cnt > fixed && turn_left(res[cnt - 2].p, p[i].p, res[cnt - 1].p)) {
            --cnt;
            res.pop_back();
        }
        res.push_back(p[i]);
        ++cnt;
    }
}

```

## 阿波罗尼茨圆

硬币问题: 易知两两相切的圆半径为  $r_1, r_2, r_3$ , 求与他们都相切的圆的半径  $r_4$   
 分母取负号, 答案再取绝对值, 为外切圆半径  
 分母取正号为内切圆半径

$$// r_4^{\pm} = \frac{r_1 r_2 r_3}{r_1 r_2 + r_1 r_3 + r_2 r_3 \pm 2\sqrt{r_1 r_2 r_3 (r_1 + r_2 + r_3)}}$$

## 三角形与圆交

// 反三角函数要在  $[-1, 1]$  中, sqrt 要与 0 取 max 别忘了取正负  
 // 改成周长请用注释, res1 为直线长度, res2 为弧线长度  
 // 多边形与圆求交时, 相切精度比较差

```

D areaCT(P pa, P pb, D r) { //, D & res1, D & res2) {
    if (pa.len() < pb.len()) swap(pa, pb);
    if (sign(pb.len()) == 0) return 0;
    // if (sign(pb.len()) == 0) { res1 += min(r, pa.len()); return; }
    D a = pb.len(), b = pa.len(), c = (pb - pa).len();
    D sinB = fabs(pb * (pb - pa)), cosB = pb % (pb - pa), area = fabs(pa * pb);
    D S, B = atan2(sinB, cosB), C = atan2(area, pa % pb);
    sinB /= a * c; cosB /= a * c;
    if (a > r) {
        S = C / 2 * r * r; D h = area / c; // res2 += -1 * sgn * C * r; D h = area / c;
        if (h < r && B < pi / 2) {
            // res2 -= -1 * sgn * 2 * acos(max((D)-1., min((D)1., h / r))) * r;

```

```

        //res1 += 2 * sqrt(max((D)0., r * r - h * h));
        S -= (acos(max((D)-1., min((D)1., h / r))) * r * r - h * sqrt(max((D)0.,
        ↪ , r * r - h * h)));
    }
} else if (b > r) {
    D theta = pi - B - asin(max((D)-1., min((D)1., sinB / r * a)));
    S = a * r * sin(theta) / 2 + (C - theta) / 2 * r * r;
    //res2 += -1 * sgn * (C - theta) * r;
    //res1 += sqrt(max((D)0., r * r + a * a - 2 * r * a * cos(theta)));
} else S = area / 2; //res1 += (pb - pa).len();
return S;
}

```

## 圆并

```

struct Event {
    P p; D ang; int delta;
    Event (P p = Point(0, 0), D ang = 0, int delta = 0) : p(p), ang(ang), delta(delta)
    ↪ {}
};
bool operator < (const Event &a, const Event &b) { return a.ang < b.ang; }
void addEvent(const Circle &a, const Circle &b, vector<Event> &evt, int &cnt) {
    D d2 = (a.o - b.o).sqrLen(), dRatio = ((a.r - b.r) * (a.r + b.r) / d2 + 1) / 2,
    pRatio = sqrt(max((D)0., -(d2 - sqr(a.r - b.r)) * (d2 - sqr(a.r + b.r)) / (d2 *
    ↪ d2 * 4)));
    P d = b.o - a.o, p = d.rot(pi / 2),
    q0 = a.o + d * dRatio + p * pRatio,
    q1 = a.o + d * dRatio - p * pRatio;
    D ang0 = (q0 - a.o).ang(), ang1 = (q1 - a.o).ang();
    evt.emplace_back(q1, ang1, 1); evt.emplace_back(q0, ang0, -1);
    cnt += ang1 > ang0;
}
bool issame(const Circle &a, const Circle &b) { return sign((a.o - b.o).len()) == 0
    ↪ && sign(a.r - b.r) == 0; }
bool overlap(const Circle &a, const Circle &b) { return sign(a.r - b.r - (a.o -
    ↪ b.o).len()) >= 0; }
bool intersect(const Circle &a, const Circle &b) { return sign((a.o - b.o).len() -
    ↪ a.r - b.r) < 0; }
int C;
Circle c[N];
double area[N];
void solve() { // 返回覆盖至少 k 次的面积
    memset(area, 0, sizeof(D) * (C + 1));
    for (int i = 0; i < C; ++i) {
        int cnt = 1;
        vector<Event> evt;
        for (int j = 0; j < i; ++j) if (issame(c[i], c[j])) ++cnt;
        for (int j = 0; j < C; ++j)
            if (j != i && !issame(c[i], c[j]) && overlap(c[j], c[i]))
                ++cnt;
        for (int j = 0; j < C; ++j)
            if (j != i && !overlap(c[j], c[i]) && !overlap(c[i], c[j]) && intersect(c[i],
            ↪ c[j]))
                addEvent(c[i], c[j], evt, cnt);
        if (evt.empty()) area[cnt] += PI * c[i].r * c[i].r;
    }
}

```

```

else {
    sort(evt.begin(), evt.end());
    evt.push_back(evt.front());
    for (int j = 0; j + 1 < (int)evt.size(); ++j) {
        cnt += evt[j].delta;
        area[cnt] += det(evt[j].p, evt[j + 1].p) / 2;
        D ang = evt[j + 1].ang - evt[j].ang;
        if (ang < 0) ang += PI * 2;
        area[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang) * c[i].r * c[i].r / 2;
    } } }
}

```

## 整数半平面交

```

typedef __int128 J; // 坐标 |1e9| 就要用 int128 来判断
struct Line {
    bool include(P a) const { return (a - s) * d >= 0; } // 严格去掉 =
    bool include(Line a, Line b) const {
        J l1(a.d * b.d);
        if (!l1) return true;
        J x(l1 * (a.s.x - s.x)), y(l1 * (a.s.y - s.y));
        J l2((b.s - a.s) * b.d);
        x += l2 * a.d.x; y += l2 * a.d.y;
        J res(x * d.y - y * d.x);
        return l1 > 0 ? res >= 0 : res <= 0; // 严格去掉 =
    }
};
bool HPI(vector<Line> v) { // 返回 v 中每个射线的右侧的交是否非空
    sort(v.begin(), v.end()); // 按方向排极角序
    { // 同方向取最严格的一个
        vector<Line> t; int n(v.size());
        for (int i(0), j; i < n; i = j) {
            LL mx(-9e18); int mxi;
            for (j = i; j < n && v[i].d * v[j].d == 0; j++) {
                LL tmp(v[j].s * v[i].d);
                if (tmp > mx)
                    mx = tmp, mxi = j;
            }
            t.push_back(v[mxi]);
        }
        swap(v, t);
    }
    deque<Line> res;
    bool emp(false);
    for (auto i : v) {
        if (res.size() == 1) {
            if (res[0].d * i.d == 0 && !i.include(res[0].s)) {
                res.pop_back();
                emp = true;
            }
        } else if (res.size() >= 2) {
            while (res.size() >= 2u && !i.include(res.back(), res[res.size() - 2])) {
                if (i.d * res[res.size() - 2].d == 0 || !res.back().include(i, res[res.size()
                ↪ - 2])) {
                    emp = true;
                    break;
                }
            }
        }
    }
}

```

```

    }
    res.pop_back();
}
while(res.size() >= 2u && !i.include(res[0], res[1])) res.pop_front();
}
if(emp) break;
res.push_back(i);
}
while (res.size() > 2u && !res[0].include(res.back(), res[res.size() - 2]))
    res.pop_back();
return !emp; // emp: 是否为空, res 按顺序即为半平面交
}

```

## 三角形

```

P fermat(const P& a, const P& b, const P& c) {
    D ab((b - a).len()), bc((b - c).len()), ca((c - a).len());
    D cosa((b - a) % (c - a) / ab / ca);
    D cosb((a - b) % (c - b) / ab / bc);
    D cosc((b - c) % (a - c) / ca / bc);
    P mid; D sq3(sqrt(3) / 2);
    if(sign((b - a) * (c - a)) < 0) swap(b, c);
    if(sign(cosa + 0.5) < 0) mid = a;
    else if(sign(cosb + 0.5) < 0) mid = b;
    else if(sign(cosc + 0.5) < 0) mid = c;
    else mid = intersection(Line(a, c + (b - c).rot(sq3) - a), Line(c, b + (a -
        ↪ b).rot(sq3) - c));
    return mid;
    // mid 为三角形 abc 费马点, 要求 abc 非退化
    length = (mid - a).len() + (mid - b).len() + (mid - c).len();
    // 以下求法仅在三角形三个角均小于 120 度时, 可以求出 ans 为费马点到 abc 三点距离和
    length = (a - c - (b - c).rot(sq3)).len();
}
P inCenter(const P & A, const P & B, const P & C) { // 内心
    D a = (B - C).len(), b = (C - A).len(), c = (A - B).len(),
    s = abs((B - A) * (C - A)),
    r = s / (a + b + c); // 内接圆半径
    return 1. / (a + b + c) * (A * a + B * b + C * c);
    ↪ // 偏心则将对应点前两个加号改为减号
}
P circumCenter(const P & a, const P & b, const P & c) { // 外心
    P bb = b - a, cc = c - a;
    // 半径为 a * b * c / 4 / S, a, b, c 为边长, S 为面积
    D db = bb.sqrLen(), dc = cc.sqrLen(), d = 2 * (bb * cc);
    return a - 1. / d * P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc);
}
P orthoCenter(const P & a, const P & b, const P & c) { // 垂心
    P ba = b - a, ca = c - a, bc = b - c;
    D Y = ba.y * ca.y * bc.y,
    A = ca.x * ba.y - ba.x * ca.y,
    x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
    y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
    return P(x0, y0);
}

```

## 经纬度求球面最短距离

```

double sphereDis(double lon1, double lat1, double lon2, double lat2, double R) {
    return R * acos(cos(lat1) * cos(lat2) * cos(lon1 - lon2) + sin(lat1) * sin(lat2));
}

```

## 长方体表面两点最短距离

```

int r;
void turn(int i, int j, int x, int y, int z, int x0, int y0, int L, int W, int H) {
    if (z==0) { int R = x*x+y*y; if (R<r) r=R;
    } else {
        if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);
        if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);
        if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0, x0-H, y0, H, W, L);
        if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0, x0, y0-H, L, H, W);
    }
}
int main(){
    int L, H, W, x1, y1, z1, x2, y2, z2;
    cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
    if (z1!=0 && z1!=H) if (y1==0 || y1==W)
        swap(y1,z1), std::swap(y2,z2), std::swap(W,H);
    else swap(x1,z1), std::swap(x2,z2), std::swap(L,H);
    if (z1==H) z1=0, z2=H-z2;
    r=0x3fffffff;
    turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
    cout<<r<<endl;
}

```

## 点到凸包切线

```

P lb(P x, vector<P> & v, int le, int ri, int sg) {
    if (le > ri) le = ri;
    int s(le), t(ri);
    while (le != ri) {
        int mid((le + ri) / 2);
        if (sign((v[mid] - x) * (v[mid + 1] - v[mid]))) == sg)
            le = mid + 1; else ri = mid;
    }
    return x - v[le]; // le 即为下标, 按需返回
}

```

// v[0] 为顺时针上凸壳, v[1] 为顺时针下凸壳, 均允许起始两个点横坐标相同  
 // 返回值为真代表严格在凸包外, 顺时针旋转在 d1 方向先碰到凸包

```

bool getTan(P x, vector<P> * v, P & d1, P & d2) {
    if (x.x < v[0][0].x) {
        d1 = lb(x, v[0], 0, sz(v[0]) - 1, 1);
        d2 = lb(x, v[1], 0, sz(v[1]) - 1, -1);
        return true;
    } else if(x.x > v[0].back().x) {
        d1 = lb(x, v[1], 0, sz(v[1]) - 1, 1);
        d2 = lb(x, v[0], 0, sz(v[0]) - 1, -1);
        return true;
    } else {
        for(int d(0); d < 2; d++) {

```

```

int id(lower_bound(v[d].begin(), v[d].end(), x,
[&](const P & a, const P & b) {
    return d == 0 ? a < b : b < a;
}) - v[d].begin());
if (id && (id == sz(v[d]) || (v[d][id - 1] - x) * (v[d][id] - x) > 0)) {
    d1 = lb(x, v[d], id, sz(v[d]) - 1, 1);
    d2 = lb(x, v[d], 0, id, -1);
    return true;
}
}
}
return false;
}
}

```

## 直线与凸包的交点

// a 是顺时针凸包, i1 为 x 最小的点, j1 为 x 最大的点 需保证 j1 > i1  
 // n 是凸包上的点数, a 需复制多份或写循环数组类

```

int lowerBound(int le, int ri, const P & dir) {
    while (le < ri) {
        int mid((le + ri) / 2);
        if (sign((a[mid + 1] - a[mid]) * dir) <= 0) {
            le = mid + 1;
        } else ri = mid;
    }
    return le;
}

int boundLower(int le, int ri, const P & s, const P & t) {
    while (le < ri) {
        int mid((le + ri + 1) / 2);
        if (sign((a[mid] - s) * (t - s)) <= 0) {
            le = mid;
        } else ri = mid - 1;
    }
    return le;
}

```

```

void calc(P s, P t) {
    if (t < s) swap(t, s);
    int i3(lowerBound(i1, j1, t - s)); // 和上凸包的切点
    int j3(lowerBound(j1, i1 + n, s - t)); // 和下凸包的切点
    int i4(boundLower(i3, j3, s, t));

```

→ // 如果有交则是右侧的交点, 与 a[i4]~a[i4+1] 相交 要判断是否有交的话 就手动 check 一下  
 int j4(boundLower(j3, i3 + n, t, s)); // 如果有交左侧的交点, 与 a[j4]~a[j4+1] 相交  
 // 返回的下标不一定在 [0 ~ n-1] 内

## 平面最近点对

// Create: 2017-10-22 20:15:34  
 #include <bits/stdc++.h>

using namespace std;

const int N = 100005;

```

struct Data {
    double x, y;
};

double sqr(double x) {
    return x * x;
}

double dis(Data a, Data b) {
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

```

```

int n;
Data p[N], q[N];

```

```

double solve(int l, int r) {
    if (l == r) return 1e18;
    if (l + 1 == r) return dis(p[l], p[r]);
    int m = (l + r) / 2;
    double d = min(solve(l, m), solve(m + 1, r));
    int qt = 0;
    for (int i = l; i <= r; i++) {
        if (fabs(p[m].x - p[i].x) <= d) {
            q[++qt] = p[i];
        }
    }
    sort(q + 1, q + qt + 1, [&](const Data &a, const Data &b) {
        return a.y < b.y; });
    for (int i = 1; i <= qt; i++) {
        for (int j = i + 1; j <= qt; j++) {
            if (q[j].y - q[i].y >= d) break;
            d = min(d, dis(q[i], q[j]));
        }
    }
    return d;
}

```

```

int main()
{
    while (scanf("%d", &n) == 1 && n) {
        for (int i = 1; i <= n; i++) {
            scanf("%lf%lf", &p[i].x, &p[i].y);
        }
        sort(p + 1, p + n + 1, [&](const Data &a, const Data &b) {
            return a.x < b.x || (a.x == b.x && a.y < b.y); });
        double ans = solve(1, n);
        printf("%.2f\n", ans / 2);
    }
    return 0;
}

```

## 三维几何

/\* 大拇指指向 x 轴正方向时, 4 指弯曲由 y 轴正方向指向 z 轴正方向  
 大拇指沿着原点到点 (x, y, z) 的向量, 4 指弯曲方向旋转 w 度 \*/

```

/* (x, y, z) * A = (x_new, y_new, z_new), 行向量右乘转移矩阵 */
void calc(D x, D y, D z, D w) {
    w = w * pi / 180;
    memset(a, 0, sizeof(a));
    s1 = x * x + y * y + z * z;
    a[0][0] = ((y*y+z*z)*cos(w)+x*x)/s1; a[0][1] = x*y*(1-cos(w))/s1+z*sin(w)/sqrt(s1);
    ↪ a[0][2] = x*z*(1-cos(w))/s1-y*sin(w)/sqrt(s1);
    a[1][0] = x*y*(1-cos(w))/s1-z*sin(w)/sqrt(s1); a[1][1] = ((x*x+z*z)*cos(w)+y*y)/s1;
    ↪ a[1][2] = y*z*(1-cos(w))/s1+x*sin(w)/sqrt(s1);
    a[2][0] = x*z*(1-cos(w))/s1+y*sin(w)/sqrt(s1); a[2][1] =
    ↪ y*z*(1-cos(w))/s1-x*sin(w)/sqrt(s1); a[2][2] = ((x*x+y*y)*cos(w)+z*z)/s1;
}
// 求平面和直线的交点
Point3D intersection(const Point3D &a, const Point3D &b, const Point3D &c, const
    ↪ Point3D &l0, const Point3D &l1) {
    Point3D p = pVec(a, b, c); // 平面法向量
    double t = (p.x * (a.x - l0.x) + p.y * (a.y - l0.y) + p.z * (a.z - l0.z)) / (p.x *
    ↪ (l1.x - l0.x) + p.y * (l1.y - l0.y) + p.z * (l1.z - l0.z));
    return l0 + (l1 - l0) * t;
}

```

## 其他

### 最小树形图

```

const int maxn=1100;

int n,m , g[maxn][maxn] , used[maxn] , pass[maxn] , eg[maxn] , more , queue[maxn];

void combine (int id , int &sum ) {
    int tot = 0 , from , i , j , k ;
    for ( ; id!=0 && !pass[ id ] ; id=eg[id] ) {
        queue[tot++]=id ; pass[id]=1;
    }
    for ( from=0; from<tot && queue[from]!=id ; from++);
    if ( from==tot ) return ;
    more = 1 ;
    for ( i=from ; i<tot ; i++) {
        sum+=g[eg[queue[i]]][queue[i]] ;
        if ( i!=from ) {
            used[queue[i]]=1;
            for ( j = 1 ; j <= n ; j++) if ( !used[j] )
                ↪ if ( g[queue[i]][j]<g[id][j] ) g[id][j]=g[queue[i]][j] ;
        }
    }
    for ( i=1; i<=n ; i++) if ( !used[i] && i!=id ) {
        for ( j=from ; j<tot ; j++){
            k=queue[j];
            if ( g[i][id]>g[i][k]-g[eg[k]][k] ) g[i][id]=g[i][k]-g[eg[k]][k];
        }
    }
}

int mdst( int root ) { // return the total length of MDST
    int i , j , k , sum = 0 ;

```

```

memset ( used , 0 , sizeof ( used ) ) ;
for ( more =1; more ; ) {
    more = 0 ;
    memset (eg,0,sizeof(eg)) ;
    for ( i=1 ; i <= n ; i ++ ) if ( !used[i] && i!=root ) {
        for ( j=1 , k=0 ; j <= n ; j ++ ) if ( !used[j] && i!=j )
            ↪ if ( k==0 || g[j][i] < g[k][i] ) k=j ;
        eg[i] = k ;
    }
    memset(pass,0,sizeof(pass));
    for ( i=1; i<=n ; i++ ) if ( !used[i] && !pass[i] && i!= root ) combine ( i , sum
    ↪ ) ;
}
for ( i =1; i<=n ; i ++ ) if ( !used[i] && i!= root ) sum+=g[eg[i]][i];
return sum ;
}

```

### DLX

```

int n,m,K;
struct DLX{
    int L[maxn],R[maxn],U[maxn],D[maxn];
    int sz,col[maxn],row[maxn],s[maxn],H[maxn];
    bool vis[233];
    int ans[maxn],cnt;
    void init(int m){
        for(int i=0;i<=m;i++){
            L[i]=i-1;R[i]=i+1;
            U[i]=D[i]=i;s[i]=0;
        }
        memset(H,-1,sizeof H);
        L[0]=m;R[m]=0;sz=m+1;
    }
    void Link(int r,int c){
        U[sz]=c;D[sz]=D[c];U[D[c]]=sz;D[c]=sz;
        if(H[r]<0)H[r]=L[sz]=R[sz]=sz;
        else{
            L[sz]=H[r];R[sz]=R[H[r]];
            L[R[H[r]]]=sz;R[H[r]]=sz;
        }
        s[c]++;col[sz]=c;row[sz]=r;sz++;
    }
    void remove(int c){
        for(int i=D[c];i!=c;i=D[i])
            L[R[i]]=L[i],R[L[i]]=R[i];
    }
    void resume(int c){
        for(int i=U[c];i!=c;i=U[i])
            L[R[i]]=R[L[i]]=i;
    }
    int A(){
        int res=0;
        memset(vis,0,sizeof vis);
        for(int i=R[0];i=R[i])if(!vis[i]){
            vis[i]=1;res++;
            for(int j=D[i];j!=i;j=D[j])

```

```

        for(int k=R[j];k!=j;k=R[k])
            vis[col[k]]=1;
    }
    return res;
}
void dfs(int d,int &ans){
    if(R[0]==0){ans=min(ans,d);return;}
    if(d+A()>=ans)return;
    int tmp=23333,c;
    for(int i=R[0];i;i=R[i])
        if(tmp>s[i])tmp=s[i],c=i;
    for(int i=D[c];i!=c;i=D[i]){
        remove(i);
        for(int j=R[i];j!=i;j=R[j])remove(j);
        dfs(d+1,ans);
        for(int j=L[i];j!=i;j=L[j])resume(j);
        resume(i);
    }
}
void del(int c){//exactly cover
    L[R[c]]=L[c];R[L[c]]=R[c];
    for(int i=D[c];i!=c;i=D[i])
        for(int j=R[i];j!=i;j=R[j])
            U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
}
void add(int c){ //exactly cover
    R[L[c]]=L[R[c]]=c;
    for(int i=U[c];i!=c;i=U[i])
        for(int j=L[i];j!=i;j=L[j])
            ++s[col[U[D[j]]=D[U[j]]=j]];
}
bool dfs2(int k){//exactly cover
    if(!R[0]){
        cnt=k;return 1;
    }
    int c=R[0];
    for(int i=R[0];i;i=R[i])
        if(s[c]>s[i])c=i;
    del(c);
    for(int i=D[c];i!=c;i=D[i]){
        for(int j=R[i];j!=i;j=R[j])
            del(col[j]);
        ans[k]=row[i];if(dfs2(k+1))return true;
        for(int j=L[i];j!=i;j=L[j])
            add(col[j]);
        }
    add(c);
    return 0;
}
}dlx;
int main(){
    dlx.init(n);
    for(int i=1;i<=m;i++)
        for(int j=1;j<=n;j++)
            if(dis(station[i],city[j])<mid-eps)

```

```

        dlx.Link(i,j);
    }dlx.dfs(0,ans);
}

```

## 某年某月某日是星期几

```

int solve(int year, int month, int day) {
    int answer;
    if (month == 1 || month == 2) {
        month += 12;
        year--;
    }
    if ((year < 1752) || (year == 1752 && month < 9) ||
        (year == 1752 && month == 9 && day < 3)) {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4 + 5) % 7;
    } else {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4
            - year / 100 + year / 400) % 7;
    }
    return answer;
}

```

## 枚举大小为 $k$ 的子集

使用条件:  $k > 0$

```

void solve(int n, int k) {
    for (int comb = (1 << k) - 1; comb < (1 << n); ) {
        // ...
        int x = comb & -comb, y = comb + x;
        comb = (((comb & ~y) / x) >> 1) | y;
    }
}

```

## 环状最长公共子串

```

int n, a[N << 1], b[N << 1];

bool has(int i, int j) {
    return a[(i - 1) % n] == b[(j - 1) % n];
}

const int DELTA[3][2] = {{0, -1}, {-1, -1}, {-1, 0}};

int from[N][N];

int solve() {
    memset(from, 0, sizeof(from));
    int ret = 0;
    for (int i = 1; i <= 2 * n; ++i) {
        from[i][0] = 2;
        int left = 0, up = 0;
        for (int j = 1; j <= n; ++j) {
            int upleft = up + 1 + !!from[i - 1][j];

```

```

    if (!has(i, j)) {
        upleft = INT_MIN;
    }
    int max = std::max(left, std::max(upleft, up));
    if (left == max) {
        from[i][j] = 0;
    } else if (upleft == max) {
        from[i][j] = 1;
    } else {
        from[i][j] = 2;
    }
    left = max;
}
if (i >= n) {
    int count = 0;
    for (int x = i, y = n; y; ) {
        int t = from[x][y];
        count += t == 1;
        x += DELTA[t][0];
        y += DELTA[t][1];
    }
    ret = std::max(ret, count);
    int x = i - n + 1;
    from[x][0] = 0;
    int y = 0;
    while (y <= n && from[x][y] == 0) {
        y++;
    }
    for (; x <= i; ++x) {
        from[x][y] = 0;
        if (x == i) {
            break;
        }
        for (; y <= n; ++y) {
            if (from[x + 1][y] == 2) {
                break;
            }
            if (y + 1 <= n && from[x + 1][y + 1] == 1) {
                y++;
                break;
            }
        }
    }
}
}
return ret;
}

```

## LLMOD STL 内存清空开栈

```

LL multiplyMod(LL a, LL b, LL P) { // `需要保证 a 和 b 非负`
    LL t = (a * b - LL((long double)a / P * b + 1e-3) * P) % P;
    return t < 0 : t + P : t;
}

```

```

template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}

```

```

register char *_sp __asm__("rsp");
int main() {
    const int size = 400 << 20; // 400MB
    static char *sys, *mine(new char[size] + size - 4096);
    sys = _sp; _sp = mine; _main(); _sp = sys;
}

```

## vimrc

colo morning

```

set ru nu cin ts=4 sts=4 sw=4 hls is ar acd bs=2 mouse=a ls=2 fdm=syntax fdl=100
set makeprg=g++\ %:r.cpp\ -o\ %:r\ -g\ -std=c++11\ -Wall\ -Wextra\ -Wconversion

```

nmap <C-A> ggVG

vmap <C-C> "+y

noremap <C-V> "+P

map <F3> :vnew %:r.in<cr>

map <F4> :!gedit %<cr>

map <F5> :!time ./%:r<cr>

map <F8> :!time ./%:r < %:r.in<cr>

map <F9> :make<cr>

map <C-F9> :!g++ %:r.cpp -o %:r -g -O2 -std=c++11<cr>

map <F10> :!gdb ./%:r<cr>

## 上下界网络流

### 无源汇的上下界可行流

建立超级源点  $S^*$  和超级汇点  $T^*$ ，对于原图每条边  $(u, v)$  在新网络中连如下三条边： $S^* \rightarrow v$ ，容量为  $B(u, v)$ ； $u \rightarrow T^*$ ，容量为  $B(u, v)$ ； $u \rightarrow v$ ，容量为  $C(u, v) - B(u, v)$ 。最后求新网络的最大流，判断从超级源点  $S^*$  出发的边是否都满流即可，边  $(u, v)$  的最终解中的实际流量为  $G(u, v) + B(u, v)$ 。

### 有源汇的上下界可行流

从汇点  $T$  到源点  $S$  连一条上界为  $\infty$ ，下界为  $0$  的边。按照无源汇的上下界可行流一样做即可，流量即为  $T \rightarrow S$  边上的流量。



### 有源汇的上下界最大流

1. 在有源汇的上下界可行流中，从汇点  $T$  到源点  $S$  的边改为连一条上界为  $\infty$ ，下届为  $x$  的边。 $x$  满足二分性质，找到最大的  $x$  使得新网络存在无源汇的上下界可行流即为原图的最大流。
2. 从汇点  $T$  到源点  $S$  连一条上界为  $\infty$ ，下界为  $0$  的边，变成无源汇的网络。按照无源汇的上下界可行流的方法，建立超级源点  $S^*$  和超级汇点  $T^*$ ，求一遍  $S^* \rightarrow T^*$  的最大流，再将从汇点  $T$  到源点  $S$  的这条边拆掉，求一次  $S \rightarrow T$  的最大流即可。

### 有源汇的上下界最小流

1. 在有源汇的上下界可行流中，从汇点  $T$  到源点  $S$  的边改为连一条上界为  $x$ ，下界为  $0$  的边。 $x$  满足二分性质，找到最小的  $x$  使得新网络存在无源汇的上下界可行流即为原图的最小流。
2. 按照无源汇的上下界可行流的方法，建立超级源点  $S^*$  与超级汇点  $T^*$ ，求一遍  $S^* \rightarrow T^*$  的最大流，但是注意这一次不加上汇点  $T$  到源点  $S$  的这条边，即不使之改为无源汇的网络去求解。求完后，再加上那条汇点  $T$  到源点  $S$  上界  $\infty$  的边。因为这条边下界为  $0$ ，所以  $S^*$ ， $T^*$  无影响，再直接求一次  $S^* \rightarrow T^*$  的最大流。若超级源点  $S^*$  出发的边全部满流，则  $T \rightarrow S$  边上的流量即为原图的最小流，否则无解。

### 上下界费用流

**来源：BZOJ 3876** 设汇  $t$ ，源  $s$ ，超级源  $S$ ，超级汇  $T$ ，本质是每条边的下界为  $1$ ，上界为  $\text{MAX}$ ，跑一遍有源汇的上下界最小费用最小流。（因为上界无穷大，所以只要满足所有下界的最小费用最小流）

1. 对每个点  $x$ ：从  $x$  到  $t$  连一条费用为  $0$ ，流量为  $\text{MAX}$  的边，表示可以任意停止当前的剧情（接下来的剧情从更优的路径去走，画个样例就知道了）
2. 对于每一条边权为  $z$  的边  $x \rightarrow y$ ：
  - 从  $S$  到  $y$  连一条流量为  $1$ ，费用为  $z$  的边，代表这条边至少要被走一次。
  - 从  $x$  到  $y$  连一条流量为  $\text{MAX}$ ，费用为  $z$  的边，代表这条边除了至少走的一次之外还可以随便走。

- 从  $x$  到  $T$  连一条流量为  $1$ ，费用为  $0$  的边。（注意是每一条  $x \rightarrow y$  的边都连，或者你可以记下  $x$  的出边数  $K_x$ ，连一次流量为  $K_x$ ，费用为  $0$  的边）。

建完图后从  $S$  到  $T$  跑一遍费用流，即可。（当前跑出来的就是满足上下界的最小费用最小流了）

### Bernoulli 数

1. 初始化： $B_0(n) = 1$

2. 递推公式：

$$B_m(n) = n^m - \sum_{k=0}^{m-1} \binom{m}{k} \frac{B_k(n)}{m-k+1}$$

3. 应用：

$$\sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} n^{m+1-k}$$

### Java Hints

```
import java.util.*;
import java.math.*;
import java.io.*;

public class Main{
    static class Task{
        void solve(int testId, InputReader cin, PrintWriter cout) {
            // Write down the code you want
        }
    };

    public static void main(String args[]) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        // Scanner cin = new Scanner(System.in);
        // cin.nextLong();
        // System.out.println(AnsA+" "+AnsB);
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }
    }
}
```



```

public String next() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
        try {
            tokenizer = new StringTokenizer(reader.readLine());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    return tokenizer.nextToken();
}

public int nextInt() {
    return Integer.parseInt(next());
}
}

// Arrays
int a[];
.fill(a[, int fromIndex, int toIndex],val); | .sort(a[, int fromIndex, int toIndex])
// String
String s;
.charAt(int i); | compareTo(String) | compareToIgnoreCase () | contains(String) |
length () | substring(int l, int len)
// BigInteger
.abs() | .add() | bitLength () | subtract () | divide () | remainder () |
↪ divideAndRemainder () | modPow(b, c) |
pow(int) | multiply () | compareTo () |
gcd() | intValue () | longValue () | isProbablePrime(int c) (1 - 1/2^c) |
nextProbablePrime () | shiftLeft(int) | valueOf ()
// BigDecimal
.ROUND_CEILING | ROUND_DOWN_FLOOR | ROUND_HALF_DOWN | ROUND_HALF_EVEN | ROUND_HALF_UP
↪ | ROUND_UP
.divide(BigDecimal b, int scale , int round_mode) | doubleValue () |
↪ movePointLeft(int) | pow(int) |
setScale(int scale , int round_mode) | stripTrailingZeros ()
BigDecimal.setScale()方法用于格式化小数点
setScale(1)表示保留一位小数,默认用四舍五入方式
setScale(1,BigDecimal.ROUND_DOWN)直接删除多余的小数位,如 2.35会变成 2.3
setScale(1,BigDecimal.ROUND_UP)进位处理,2.35变成 2.4
setScale(1,BigDecimal.ROUND_HALF_UP)四舍五入,2.35变成 2.4
setScale(1,BigDecimal.ROUND_HALF_DOWN)四舍五入,2.35变成 2.3,如果是 5 则向下舍
setScale(1,BigDecimal.ROUND_CEILING)接近正无穷大的舍入
setScale(1,BigDecimal.ROUND_FLOOR)接近负无穷大的舍入,数字>0=ROUND_UP,数字<0=ROUND_DOWN
setScale(1,BigDecimal.ROUND_HALF_EVEN)向最接近的数字舍入,如果距离相等则向相邻的偶数舍入
// StringBuilder
StringBuilder sb = new StringBuilder ();
sb.append(elem) | out.println(sb)

```

## 数学

### 常用数学公式

#### 求和公式

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

#### 斐波那契数列

- $fib_0 = 0, fib_1 = 1, fib_n = fib_{n-1} + fib_{n-2}$
- $fib_{n+2} \cdot fib_n - fib_{n+1}^2 = (-1)^{n+1}$
- $fib_{-n} = (-1)^{n-1} fib_n$
- $fib_{n+k} = fib_k \cdot fib_{n+1} + fib_{k-1} \cdot fib_n$
- $gcd(fib_m, fib_n) = fib_{gcd(m,n)}$
- $fib_m | fib_n^2 \Leftrightarrow n fib_n | m$

#### 错排公式

- $D_n = (n-1)(D_{n-2} - D_{n-1})$
- $D_n = n! \cdot (1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + \frac{(-1)^n}{n!})$

#### 莫比乌斯函数

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(\frac{n}{d}) \quad g(x) = \sum_{n=1}^{[x]} f(\frac{x}{n}) \Leftrightarrow f(x) = \sum_{n=1}^{[x]} \mu(n)g(\frac{x}{n})$$

伯恩赛德引理

设  $G$  是一个有限群,作用在集合  $X$  上.对每个  $g$  属于  $G$ ,令  $X^g$  表示  $X$  中在  $g$  作用下的不动元素,轨道数 (记作  $|X/G|$ ) 由如下公式给出:  $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$ .

五边形数定理

设  $p(n)$  是  $n$  的拆分数,有  $p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$

树的计数

- 1. 有根树计数:  $n+1$  个结点的有根树的个数为  $a_{n+1} = \frac{\sum_{j=1}^n j \cdot a_j \cdot S_{n,j}}{n}$  其中,  $S_{n,j} = \sum_{i=1}^{n/j} a_{n+1-ij} = S_{n-j,j} + a_{n+1-j}$
- 2. 无根树计数: 当  $n$  为奇数时,  $n$  个结点的无根树的个数为  $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$  当  $n$  为偶数时,  $n$  个结点的无根树的个数为  $a_n - \sum_{i=1}^{n/2} a_i a_{n-i} + \frac{1}{2} a_{\frac{n}{2}} (a_{\frac{n}{2}} + 1)$
- 3.  $n$  个结点的完全图的生成树个数为  $n^{n-2}$
- 4. 矩阵 - 树定理: 图  $G$  由  $n$  个结点构成, 设  $\mathbf{A}[G]$  为图  $G$  的邻接矩阵、 $\mathbf{D}[G]$  为图  $G$  的度数矩阵, 则图  $G$  的不同生成树的个数为  $\mathbf{C}[G] = \mathbf{D}[G] - \mathbf{A}[G]$  的任意一个  $n-1$  阶主子式的行列式值。

欧拉公式

平面图的顶点个数、边数和面的个数有如下关系:  $V - E + F = C + 1$  其中,  $V$  是顶点的数目,  $E$  是边的数目,  $F$  是面的数目,  $C$  是组成图形的连通部分的数目. 当图是单连通图的时候, 公式简化为:  $V - E + F = 2$

皮克定理

给定顶点坐标均是整点 (或正方形格点) 的简单多边形, 其面积  $A$  和内部格点数目  $i$ 、边上格点数目  $b$  的关系:  $A = i + \frac{b}{2} - 1$

牛顿恒等式

设  $\prod_{i=1}^n (x - x_i) = a_n + a_{n-1}x + \cdots + a_1x^{n-1} + a_0x^n$   $p_k = \sum_{i=1}^n x_i^k$  则  $a_0p_k + a_1p_{k-1} + \cdots + a_{k-1}p_1 + ka_k = 0$

特别地,对于  $|\mathbf{A} - \lambda \mathbf{E}| = (-1)^n (a_n + a_{n-1}\lambda + \cdots + a_1\lambda^{n-1} + a_0\lambda^n)$  有  $p_k = Tr(\mathbf{A}^k)$

平面几何公式

三角形

- 1. 面积  $S = \frac{a \cdot H_a}{2} = \frac{ab \cdot \sin C}{2} = \sqrt{p(p-a)(p-b)(p-c)}$
- 2. 中线  $M_a = \frac{\sqrt{2(b^2+c^2)-a^2}}{2} = \frac{\sqrt{b^2+c^2+2bc \cdot \cos A}}{2}$

- 3. 角平分线  $T_a = \frac{\sqrt{bc \cdot [(b+c)^2 - a^2]}}{b+c} = \frac{2bc \cos \frac{A}{2}}{b+c}$
- 4. 高线  $H_a = b \sin C = c \sin B = \sqrt{b^2 - (\frac{a^2+b^2-c^2}{2a})^2}$
- 5. 内切圆半径

$$r = \frac{S}{p} = \frac{\arcsin \frac{B}{2} \cdot \sin \frac{C}{2}}{\sin \frac{B+C}{2}} = 4R \cdot \sin \frac{A}{2} \sin \frac{B}{2} \sin \frac{C}{2}$$
$$= \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} = p \cdot \tan \frac{A}{2} \tan \frac{B}{2} \tan \frac{C}{2}$$

- 6. 外接圆半径  $R = \frac{abc}{4S} = \frac{a}{2 \sin A} = \frac{b}{2 \sin B} = \frac{c}{2 \sin C}$

四边形

$D_1, D_2$  为对角线,  $M$  对角线中点连线,  $A$  为对角线夹角,  $p$  为半周长

- 1.  $a^2 + b^2 + c^2 + d^2 = D_1^2 + D_2^2 + 4M^2$
- 2.  $S = \frac{1}{2} D_1 D_2 \sin A$
- 3. 对于圆内接四边形  $ac + bd = D_1 D_2$
- 4. 对于圆内接四边形  $S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$

正  $n$  边形

$R$  为外接圆半径,  $r$  为内切圆半径

- 1. 中心角  $A = \frac{2\pi}{n}$
- 2. 内角  $C = \frac{n-2}{n} \pi$
- 3. 边长  $a = 2\sqrt{R^2 - r^2} = 2R \cdot \sin \frac{A}{2} = 2r \cdot \tan \frac{A}{2}$
- 4. 面积  $S = \frac{nar}{2} = nr^2 \cdot \tan \frac{A}{2} = \frac{nR^2}{2} \cdot \sin A = \frac{na^2}{4 \tan \frac{A}{2}}$

圆

- 1. 弧长  $l = rA$
- 2. 弦长  $a = 2\sqrt{2hr - h^2} = 2r \cdot \sin \frac{A}{2}$
- 3. 弓形高  $h = r - \sqrt{r^2 - \frac{a^2}{4}} = r(1 - \cos \frac{A}{2}) = \frac{1}{2} \cdot \arctan \frac{A}{4}$

- 4. 扇形面积  $S_1 = \frac{rl}{2} = \frac{r^2 A}{2}$
- 5. 弓形面积  $S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2}{2}(A - \sin A)$

棱柱

- 1. 体积  $V = Ah$   $A$  为底面积,  $h$  为高
- 2. 侧面积  $S = lp$   $l$  为棱长,  $p$  为直截面周长
- 3. 全面积  $T = S + 2A$

棱锥

- 1. 体积  $V = Ah$   $A$  为底面积,  $h$  为高
- 2. 正棱锥侧面积  $S = lp$   $l$  为棱长,  $p$  为直截面周长
- 3. 正棱锥全面积  $T = S + 2A$

棱台

- 1. 体积  $V = (A_1 + A_2 + \sqrt{A_1 A_2}) \cdot \frac{h}{3}$   $A_1, A_2$  为上下底面积,  $h$  为高
- 2. 正棱台侧面积  $S = \frac{p_1 + p_2}{2} l$   $p_1, p_2$  为上下底面周长,  $l$  为斜高
- 3. 正棱台全面积  $T = S + A_1 + A_2$

圆柱

- 1. 侧面积  $S = 2\pi rh$
- 2. 全面积  $T = 2\pi r(h + r)$
- 3. 体积  $V = \pi r^2 h$

圆锥

- 1. 母线  $l = \sqrt{h^2 + r^2}$
- 2. 侧面积  $S = \pi rl$
- 3. 全面积  $T = \pi r(l + r)$
- 4. 体积  $V = \frac{\pi}{3} r^2 h$

圆台

- 1. 母线  $l = \sqrt{h^2 + (r_1 - r_2)^2}$
- 2. 侧面积  $S = \pi(r_1 + r_2)l$
- 3. 全面积  $T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$
- 4. 体积  $V = \frac{\pi}{3}(r_1^2 + r_2^2 + r_1 r_2)h$

球台

- 1. 侧面积  $S = 2\pi rh$
- 2. 全面积  $T = \pi(2rh + r_1^2 + r_2^2)$
- 3. 体积  $V = \frac{\pi h[3(r_1^2 + r_2^2) + h^2]}{6}$

球扇形

- 1. 全面积  $T = \pi r(2h + r_0)$   $h$  为球冠高,  $r_0$  为球冠底面半径
- 2. 体积  $V = \frac{2}{3}\pi r^2 h$

积分表

$$\int \frac{1}{1+x^2} dx = \tan^{-1} x$$
$$\int \frac{1}{a^2+x^2} dx = \frac{1}{a} \tan^{-1} \frac{x}{a}$$
$$\int \frac{x}{a^2+x^2} dx = \frac{1}{2} \ln |a^2 + x^2|$$
$$\int \frac{x^2}{a^2+x^2} dx = x - a \tan^{-1} \frac{x}{a}$$
$$\int \sqrt{x^2 \pm a^2} dx = \frac{1}{2} x \sqrt{x^2 \pm a^2} \pm \frac{1}{2} a^2 \ln |x + \sqrt{x^2 \pm a^2}|$$
$$\int \sqrt{a^2 - x^2} dx = \frac{1}{2} x \sqrt{a^2 - x^2} + \frac{1}{2} a^2 \tan^{-1} \frac{x}{\sqrt{a^2 - x^2}}$$
$$\int \frac{x^2}{\sqrt{x^2 \pm a^2}} dx = \frac{1}{2} x \sqrt{x^2 \pm a^2} \mp \frac{1}{2} a^2 \ln |x + \sqrt{x^2 \pm a^2}|$$
$$\int \frac{1}{\sqrt{x^2 \pm a^2}} dx = \ln |x + \sqrt{x^2 \pm a^2}|$$
$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \sin^{-1} \frac{x}{a}$$
$$\int \frac{x}{\sqrt{x^2 \pm a^2}} dx = \sqrt{x^2 \pm a^2}$$
$$\int \frac{x}{\sqrt{a^2 - x^2}} dx = -\sqrt{a^2 - x^2}$$
$$\int \sqrt{ax^2 + bx + c} dx = \frac{b+2ax}{4a} \sqrt{ax^2 + bx + c} + \frac{4ac-b^2}{8a^{3/2}} \ln \left| 2ax + b + 2\sqrt{a(ax^2 + bx + c)} \right|$$

$$\int x^n e^{ax} dx = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx$$

$$\int \sin^2 ax dx = \frac{x}{2} - \frac{1}{4a} \sin 2ax$$

$$\int \sin^3 ax dx = -\frac{3 \cos ax}{4a} + \frac{\cos 3ax}{12a}$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a}$$

$$\int \cos^3 ax dx = \frac{3 \sin ax}{4a} + \frac{\sin 3ax}{12a}$$

$$\int \tan ax dx = -\frac{1}{a} \ln \cos ax$$

$$\int \tan^2 ax dx = -x + \frac{1}{a} \tan ax$$

$$\int x \cos ax dx = \frac{1}{a^2} \cos ax + \frac{x}{a} \sin ax$$

$$\int x^2 \cos ax dx = \frac{2x \cos ax}{a^2} + \frac{a^2 x^2 - 2}{a^3} \sin ax$$

$$\int x \sin ax dx = -\frac{x \cos ax}{a} + \frac{\sin ax}{a^2}$$

$$\int x^2 \sin ax dx = \frac{2 - a^2 x^2}{a^3} \cos ax + \frac{2x \sin ax}{a^2}$$