

GZHU I WANT TO EAT KFC

Compile Commands

```
#pragma GCC optimize("-O3")
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

C++IO

```
// 简版
template <class I> void read(I& x) {
    char c;
    while ((c = getchar()) < '0' && c > '9');
    for (x = c - '0'; (c = getchar()) >= '0' && c <= '9'; x = x * 10 + c - '0');
}

// io完全体
namespace io {
    const int BUFLen = (1 << 21) + 1;
    bool EOFError;
    inline char gc() {
        static char buf[BUFLen], *st = nullptr, *ed = nullptr;
        if (st == ed) (ed = (st = buf) + fread(buf, 1, BUFLen, stdin));
        return (st == ed) ? -1 : (*st++);
    }
    inline bool check(char x) {
        return x == '-' || x == '\n' || x == ' ' || x == '\r' || x == '\t';
    }
    template <class I> inline void read(I& x) {
        char c; int f = 1;
        while (check(c = gc())) if (c == '-') f = -1;
        if (c == -1) { EOFError = 1; return; }
        for (x = c - '0'; (c = gc()) >= '0' && c <= '9'; x = x * 10 + (c & 15)); x *= f;
    }
    inline void gstr(char *s, int len) {
        char c; for (c = gc(); c < 'a' || c > 'z'; c = gc());
        if (c == -1) return;
        for (len = 0; c >= 'a' && c <= 'z'; c = gc()) s[len++] = c; s[len] = 0;
    }
    char obuf[BUFLen], *ost = obuf, *oed = obuf + BUFLen - 1, Stack[55], Top;
    inline void flush() { fwrite(obuf, 1, ost - obuf, stdout); ost = obuf; }
    inline void pc(char x) { *ost++ = x; if (ost == oed) flush(); }
    template <class I> inline void print(I x) {
        if (!x) pc('0');
        if (x < 0) pc('-'), x = -x;
        while (x) Stack[++Top] = x % 10 + '0', x /= 10;
    }
}
```

```

        while (Top) pc(Stack[Top--]);
    }
    template <class I> inline void println(I x) { print(x), pc('\n'); }
    inline void pstr(char *s) { for (int i = 0, n = strlen(s); i < n; i++) pc(s[i]); }
    struct IOFLUSHER { ~IOFLUSHER() { flush(); } } _ioflusher_;
}
using namespace io;

```

Java template

■ BigInteger

Modifier and Type	Method and Description
BigInteger	abs() Returns a BigInteger whose value is the absolute value of this BigInteger.
BigInteger	add(BigInteger val) Returns a BigInteger whose value is (this + val).
BigInteger	and(BigInteger val) Returns a BigInteger whose value is (this & val).
BigInteger	andNot(BigInteger val) Returns a BigInteger whose value is (this & ~val).
int	bitCount() Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	bitLength() Returns the number of bits in the minimal two's-complement representation of this BigInteger, <i>excluding</i> a sign bit.
byte	byteValueExact() Converts this BigInteger to a byte, checking for lost information.
BigInteger	clearBit(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	compareTo(BigInteger val) Compares this BigInteger with the specified BigInteger.
BigInteger	divide(BigInteger val) Returns a BigInteger whose value is (this / val).
BigInteger[]	divideAndRemainder(BigInteger val) Returns an array of two BigIntegers containing (this / val) followed by (this % val).
double	doubleValue() Converts this BigInteger to a double.
boolean	equals(Object x) Compares this BigInteger with the specified Object for equality.
BigInteger	shiftRight(int n) Returns a BigInteger whose value is (this >> n).
short	shortValueExact() Converts this BigInteger to a short, checking for lost information.
int	signum() Returns the signum function of this BigInteger.
BigInteger	subtract(BigInteger val) Returns a BigInteger whose value is (this - val).
boolean	testBit(int n) Returns true if and only if the designated bit is set.
byte[]	toByteArray() Returns a byte array containing the two's-complement representation of this BigInteger.
String	toString() Returns the decimal String representation of this BigInteger.
String	toString(int radix) Returns the String representation of this BigInteger in the given radix.
BigInteger	xor(BigInteger val) Returns a BigInteger whose value is (this ^ val).

BigInteger	mod(BigInteger m) Returns a BigInteger whose value is (this mod m).
BigInteger	modInverse(BigInteger m) Returns a BigInteger whose value is (this ⁻¹ mod m).
BigInteger	modPow(BigInteger exponent, BigInteger m) Returns a BigInteger whose value is (this ^{exponent} mod m).
BigInteger	multiply(BigInteger val) Returns a BigInteger whose value is (this * val).
BigInteger	negate() Returns a BigInteger whose value is (-this).
BigInteger	nextProbablePrime() Returns the first integer greater than this BigInteger that is probably prime.
BigInteger	not() Returns a BigInteger whose value is (~this).
BigInteger	or(BigInteger val) Returns a BigInteger whose value is (this val).
BigInteger	pow(int exponent) Returns a BigInteger whose value is (this ^{exponent}).
BigInteger	remainder(BigInteger val) Returns a BigInteger whose value is (this % val).
BigInteger	setBit(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
BigInteger	shiftLeft(int n) Returns a BigInteger whose value is (this << n).
BigInteger	flipBit(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
float	floatValue() Converts this BigInteger to a float.
BigInteger	gcd(BigInteger val) Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val).
int	getLowestSetBit() Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit).
int	hashCode() Returns the hash code for this BigInteger.
int	intValue() Converts this BigInteger to an int.
int	intValueExact() Converts this BigInteger to an int, checking for lost information.
boolean	isProbablePrime(int certainty) Returns true if this BigInteger is probably prime, false if it's definitely composite.
long	longValue() Converts this BigInteger to a long.
long	longValueExact() Converts this BigInteger to a long, checking for lost information.
BigInteger	max(BigInteger val) Returns the maximum of this BigInteger and val.
BigInteger	min(BigInteger val) Returns the minimum of this BigInteger and val.

Java

```
// min
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(new BufferedReader(System.in));
        PrintWriter out = new PrintWriter(System.out);
        Task solver = new Task();
        try {
```

```

        solver.solve(in, out);
    } catch (NoSuchElementException e) {
        out.close();
    }
}
private static BigInteger[] a = new BigInteger[1005];
public static class Task {
    public static void solve(Scanner in, PrintWriter out) {
        // Your code start here.

    }
}
}

```

```

// max
import java.io.*;
import java.util.StringTokenizer;

public class Main {
    public static void main(String[] args) {
        InputReader in = new InputReader(System.in);
        PrintWriter out = new PrintWriter(System.out);
        Task solver = new Task();
        try {
            while (true) {
                solver.solve(in, out);
            }
        } catch (RuntimeException e) {
            out.close();
        }
    }

    public static class Task {
        void solve(InputReader in, PrintWriter out) {
            // Code start here

        }
    }

    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());
    }
    public long nextLong() {
        return Long.parseLong(next());
    }
    public double nextDouble() {
        return Double.parseDouble(next());
    }
    public BigInteger nextBigInteger() {
        return new BigInteger(next());
    }
    public BigDecimal nextBigDecimal() {
        return new BigDecimal(next());
    }
}
}

```

数据结构

Fenwick_Tree(树状数组)

- 维护&求值均为 $O(\log n)$

```

struct Fenwick {
    static const int maxn = 1e5 + 5;
    int a[maxn];
    void init() { memset(a, 0, sizeof a); }
    inline int lowbit(int i) { return (i & (-i)); }
    void upd(int i, int x) {
        for (; i < maxn; i += lowbit(i)) {
            a[i] += x;
        }
    }
    int sum(int i) {
        int res = 0;
        for (; i > 0; i -= lowbit(i)) {
            res += a[i];
        }
        return res;
    }
    inline int query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    int upper_bound(int x) {
        int res = 0, ptr = 0;
        while ((1 << (ptr + 1)) <= n) ++ptr;
        for (; ptr >= 0; ptr--) {

```

```

        int p = res + (1 << ptr);
        if (p <= n && a[p] <= x) {
            x -= a[p];
            res += 1 << ptr;
        }
    }
    return res;
}
};

```

spare_table_rmq

- build $O(n \log n)$, query $O(1)$

```

const int MAX = 1e6 + 5;

int dp[MAX][25];

void rmq(int n) {
    int len = (int)(log(n) / log(2.0));
    for (int j = 1; j <= len; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int query(int l, int r) {
    int p = (int)(log(r - l + 1) / log(2.0));
    return min(dp[l][p], dp[r - (1 << p) + 1][p]);
}

```

Disjoin_set_union(并查集)

```

struct Dsu {
    static const int maxn = 1e5 + 5;
    int fa[maxn], sz[maxn];
    void init(int n) {
        for (int i = 0; i <= n; i++) {
            fa[i] = i, sz[i] = 0;
        }
    }
    int find(int x) {
        return x == fa[x] ? x : fa[x] = find(fa[x]);
    }
}

```

```

bool unite(int u, int v) {
    int a = find(u), b = find(v);
    if (a == b) return false;
    if (sz[a] < sz[b]) fa[a] = b;
    else fa[b] = a, sz[a] += (sz[a] == sz[b]);
    return true;
}
};

```

Segment_Tree(线段树)

- build $O(n \log n)$, update&modify $O(\log n)$, query $O(\log n)$

```

struct Segment_Tree {
    static const int maxn = 1e3 + 5;
    int a[maxn << 2], lazy[maxn << 2], cover[maxn << 2];
    bool covered[maxn << 2];

#define lson i << 1, l, mid
#define rson i << 1 | 1, mid + 1, r
#define Lson i << 1
#define Rson i << 1 | 1

    void build(int i, int l, int r, int *arr) {
        a[i] = lazy[i] = 0;
        if (l == r) {
            a[i] = arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(lson, arr), build(rson, arr);
    }

    inline void pushdown(int i) {
        if (covered[i]) {
            a[Lson] = a[Rson] = cover[Lson] = cover[Rson] = cover[i];
            covered[Lson] = covered[Rson] = true;
            lazy[Lson] = lazy[Rson] = 0;
            covered[i] = false;
        }
        if (lazy[i]) {
            a[Lson] += lazy[i];
            a[Rson] += lazy[i];
            lazy[Lson] += lazy[i];
            lazy[Rson] += lazy[i];
            lazy[i] = 0;
        }
    }

    inline void pushup(int i) {
        a[i] = min(a[Lson], a[Rson]);
    }
}

```

```

}

void update(int L, int R, int i, int l, int r, int x) {
    if (L <= l && r <= R) {
        a[i] += x, lazy[i] += x;
        return;
    }
    pushdown(i);
    int mid = (l + r) >> 1;
    if (R <= mid) update(L, R, lson, x);
    else if (mid < L) update(L, R, rson, x);
    else update(L, R, lson, x), update(L, R, rson, x);
    pushup(i);
}

void modify(int L, int R, int i, int l, int r, int x) {
    if (L <= l && r <= R) {
        a[i] = x; lazy[i] = 0; cover[i] = x; covered[i] = true;
        return;
    }
    pushdown(i);
    int mid = (l + r) >> 1;
    if (R <= mid) modify(L, R, lson, x);
    else if (mid < L) modify(L, R, rson, x);
    else modify(L, R, lson, x), modify(L, R, rson, x);
    pushup(i);
}

int query(int L, int R, int i, int l, int r) {
    if (L <= l && r <= R) {
        return a[i];
    }
    pushdown(i);
    int mid = (l + r) >> 1;
    if (R <= mid) return query(L, R, lson);
    else if (mid < L) return query(L, R, rson);
    else return min(query(L, R, lson), query(L, R, rson));
}
};

```

leftist_heap(左偏堆)

- 均摊 $O(\log n)$

```

struct leftist_heap {
    static const int maxn = 2e5 + 5;
    static const int maxm = 1e3 + 5;

    int root[maxn], tot;
    int v[maxn], ch[maxn][2], size[maxn], stk[maxn], tp;

```



```

inline void init(int &n) {
    tot = tp = 0;
    memset(root, 0, (n + 1) * sizeof(int));
}

inline int newnode(int x) {
    int rt = (tp > 0) ? stk[--tp] : (tot++);
    v[rt] = x, size[rt] = 1, ch[rt][0] = ch[rt][1] = 0;
    return rt;
}

int merge(int a, int b) {
    if (!a || !b) {
        return a ? a : b;
    }
    if (v[a] > v[b]) {
        swap(a, b);
    }
    ch[a][1] = merge(ch[a][1], b);
    if (size[ch[a][0]] < size[ch[a][1]]) {
        swap(ch[a][0], ch[a][1]);
    }
    size[a] = size[ch[a][1]] + 1;
    return a;
}

void Merge(int a, int b) {
    root[a] = merge(root[a], root[b]);
    root[b] = 0;
}

void insert(int i, int val) {
    int tmp = newnode(val);
    root[i] = merge(root[i], tmp);
}

inline int top(int i) {
    return (root[i] > 0) ? v[root[i]] : -1;
}

inline void pop(int i) {
    (root[i] > 0) ? (stk[tp++] = root[i], root[i] = merge(ch[root[i]][0], ch[root[i]]
[1])) : 0;
}

};

```

Trie(字典树) $O(len)$

```

struct Trie {
    static const int maxn = 1e5 + 5;
    int tot, ch[maxn][30], ed[maxn], cnt[maxn];
};

```

```

void init() { tot = 0; cnt[0] = 0; ed[0] = 0; memset(ch[0], 0, sizeof ch[0]); }

int newnode() {
    cnt[++tot] = 0; ed[tot] = 0;
    memset(ch[tot], 0, sizeof ch[tot]);
    return tot;
}

void insert(char *s, int len) {
    int now = 0;
    for (int i = 0; i < len; i++) {
        if (ch[now][s[i] - 'a'] == 0) {
            ch[now][s[i] - 'a'] = newnode();
        }
        cnt[now]++;
        now = ch[now][s[i] - 'a'];
    }
    cnt[now]++;
    ed[now] = 1;
}

int count(char *s, int len) {
    int now = 0;
    for (int i = 0; i < len; i++) {
        if (ch[now][s[i] - 'a'] == 0) {
            return 0;
        }
        now = ch[now][s[i] - 'a'];
    }
    return cnt[now];
}

bool check(char *s, int len) {
    int now = 0;
    for (int i = 0; i < len; i++) {
        if (ch[now][s[i] - 'a'] == 0) {
            return false;
        }
        now = ch[now][s[i] - 'a'];
    }
    return ed[now];
}
};

```

BM推公式大法

```

#include <bits/stdc++.h>
using namespace std;
struct BM {
    static const int MAXN = 10005;

```

```

int n;
vector<double> ps[MAXN];
int pn, fail[MAXN];
double delta[MAXN];

void Solve(double x[], const int& n)
{
    pn = 0;
    memset(fail, 0, sizeof fail);
    memset(delta, 0, sizeof delta);

    ps[0].clear();
    for (int i = 1; i <= n; i++) {
        double dt = -x[i];
        for (int j = 0; j < ps[pn].size(); j++) {
            dt += x[i - j - 1] * ps[pn][j];
        }
        delta[i] = dt;

        if (fabs(dt) <= 1e-8) continue;
        fail[pn] = i;
        if (!pn) {
            ps[++pn].resize(1);
            continue;
        }

        vector<double>& ls = ps[pn - 1];
        double k = -dt / delta[fail[pn - 1]];
        vector<double> cur(i - fail[pn - 1] - 1);
        cur.push_back(-k);

        for (int j = 0; j < ls.size(); j++) {
            cur.push_back(ls[j] * k);
        }
        if (cur.size() < ps[pn].size()) {
            cur.resize(ps[pn].size());
        }
        for (int j = 0; j < ps[pn].size(); j++) {
            cur[j] += ps[pn][j];
        }

        ps[++pn] = cur;
    }
}

void print()
{
    for (int i = 0; i < ps[pn].size(); i++) {
        printf("%1f ", ps[pn][i]);
    }
    printf("\n");
}

}B;

double x[BM::MAXN];

```

```

int main()
{
    int n;
    while (scanf("%d", &n) == 1) {
        for (int i = 1; i <= n; i++) {
            scanf("%lf", &x[i]);
        }
        B.Solve(x, n);
        B.print();
    }
}

```

最长递增子序列 $O(n \log n)$

```

const int MAX = 500005;
typedef long long ll;

ll v[MAX], stack[MAX];

int upb(int l, int r, int k) {
    int mid;
    while (l < r) {
        mid = (l + r) >> 1;
        if (stack[mid] > k) r = mid;
        else l = mid + 1;
    }
    return r;
}

void solve() {
    int n; cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i];

    int top = 0;
    for (int i = 1; i <= n; i++) {
        if (top == 0) stack[++top] = v[i];
        else if (stack[top] <= v[i]) stack[++top] = v[i];
        else {
            int pos = upb(1, top, v[i]);
            stack[pos] = v[i];
        }
    }
    cout << top << endl;
}

```

最长回文子序列(记忆化搜索) $O(n^2)$

```

int len;
char v[1300];
int dp[1300][1300];

int dfs(int l, int r) {
    if (l == r) return 1;
    if (l > r) return 0;
    if (dp[l][r] != -1) return dp[l][r];

    int res = max(dfs(l, r - 1), dfs(l + 1, r));
    if (v[l] == v[r]) {
        res = max(res, dfs(l + 1, r - 1) + 2);
    }
    else {
        return dp[l][r] = res;
    }
}

int main() {
    while (scanf("%s", v) != EOF) {
        len = strlen(v);
        for (int i = 0; i < len; i++) {
            v[i] = tolower(v[i]);
        }
        memset(dp, -1, sizeof dp);
        printf("%d\n", len - dfs(0, len - 1));
    }
}

```

Fast Transform

$FFT O(n \log n)$

```

/*
Example:
    Complex[] a; int len1 = length(a);
    Complex[] b; int len2 = length(b);
    int Len = FFT::trans(len1 + len2 - 1);
    FFT::DFT(a, Len, 1);
    FFT::DFT(b, Len, 1);
    for i in range(0, 1):
        a[i] *= b[i]
    FFT::DFT(a, Len, -1);
*/
namespace FFT {
    const double PI = acos(-1.0);
    // Complex
    struct Complex {
        double r, i;
        Complex(double x = 0, double y = 0) : r(x), i(y) {}
    }
}

```

```

Complex(int n) : r(cos(2 * PI / n)), i(sin(2 * PI / n)) {}
Complex operator + (const Complex& b) const {
    return Complex(r + b.r, i + b.i);
}
Complex operator - (const Complex& b) const {
    return Complex(r - b.r, i - b.i);
}
Complex operator * (const Complex& b) const {
    return Complex(r * b.r - i * b.i, r * b.i + i * b.r);
}
friend Complex& operator *= (Complex& a, const Complex& b) {
    a = a * b;
    return a;
}
};

// bit reverse
void rev(Complex *a, int n) {
    for (int i = 1, j = n >> 1, k; i < n - 1; i++) {
        if (i < j) swap(a[i], a[j]);
        for (k = n >> 1; j >= k; j -= k, k >>= 1);
        j += k;
    }
}

// Discrete Fourier transform
// t -> 1, DFT
// t -> -1, IDFT
void DFT(Complex *a, int n, int t) {
    rev(a, n);
    for (int i = 2; i <= n; i <= 1) {
        Complex wi(i * t);
        for (int j = 0; j < n; j += i) {
            Complex w(1, 0);
            for (int k = j, h = i >> 1; k < j + h; k++) {
                Complex t = w * a[k + h], u = a[k];
                a[k] = u + t;
                a[k + h] = u - t;
                w *= wi;
            }
        }
    }
    if (t == -1) for (int i = 0; i < n; i++) a[i].r /= n;
}

// Get FFT_Len
// min(2 ^ p) which (2 ^ p) > x
int trans(int x) {
    int i = 0;
    for (; x > (1 << i); i++);
    return 1 << i;
}
}

```

$FWT O(n \log n)$

```
const int MOD = 1e9 + 7;

int qpow(int a, int t) {
    int b = 1;
    while (t > 1) {
        if (t & 1) b = b * a % MOD;
        a = a * a % MOD;
        t >>= 1;
    }
    return b;
}

const int inv2 = qpow(2, MOD - 2); // (x/2) % MOD == x*inv2 % MOD

inline int trans(int n) {
    int k = 1;
    for (; k < n; k <<= 1);
    return k;
}

void fwt(int a[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int i = 0, k = d << 1; i < n; i += k) {
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % MOD;
                a[i + j + d] = (x - y + MOD) % MOD;
                // xor : a[i + j] = x + y, a[i + j + d] = x - y
                // and : a[i + j] = x + y
                // or : a[i + j + d] = x + y
            }
        }
    }
}

void ifwt(int a[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int i = 0, k = d << 1; i < n; i += k) {
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = 111 * (x + y) * inv2 % MOD;
                a[i + j + d] = (111 * (x - y) * inv2 % MOD + MOD) % MOD;
                // xor : a[i + j] = (x + y) / 2, a[i + j + d] = (x - y) / 2
                // and : a[i + j] = x - y
                // or : a[i + j + d] = y - x;
            }
        }
    }
}
```

Graph(图相关)

匈牙利二分图匹配 $O(n(n + m))$ ~~(real?)~~

```
const int maxn = 1e5 + 5;

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

void init() {
    for (int i = 0; i < maxn; i++) {
        e[i].clear();
    }
}

void addedge(int u, int v) {
    e[u].push_back(v);
    // e[v].push_back(u);
}

bool find(int u) {
    for (int i = 0; i < (int)e[u].size(); i++) {
        int v = e[u][i];
        if (!vis[v]) {
            vis[v] = 1;
            if (link[v] == -1 || find(link[v])) {
                link[v] = u;
                // link[u] = v;
                return true;
            }
        }
    }
    return false;
}

int solve(int n) {
    int res = 0;
    memset(link, -1, sizeof link);
    for (int i = 1; i <= n; i++) {
        if (link[i] == -1) {
            memset(vis, 0, sizeof vis);
            res += find(i);
        }
    }
    return res;
}
```

- 最小点覆盖:

点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点”覆盖了所有“边”。最小点覆盖 (minimum vertex covering) 就是点最少的点覆盖。

- 最小边覆盖:

边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边”覆盖了所有“点”。最小边覆盖(minimum edge covering)就是边最少的边覆盖。

- 最大点独立集：

独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。或者说是导出的子图是零图（没有边）的点集。最大独立集(maximum independent set)就是点最多的独立集。

- 最大边独立集：

边独立集又称匹配

边独立集即一个边集，满足边集中的任两边不邻接。最大边独立集(maximum edge independent set)就是边最多的边独立集

- 最大团：

团即一个点集，集合中任两个结点相邻。最大团(maximum clique)就是点最多的团。最小点支配集：

支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。最小支配集(minimum dominating set)就是点最少的支配集。最小边支配集：

边支配集即一个边集，使得所有边至少有一条邻接边在集合里。最小边支配集(minimum edge dominating set)就是边最少的边支配集。最小路径覆盖：

- 最小路径覆盖(path covering)：

是“路径”覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图G的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为0(单个点)。

- 二分图性质

二分图的最小点覆盖数 = 最大边匹配数

【即求最少的点使得每条边都至少和其中的一个点相关联，很显然直接取最大匹配的一段节点即可】二分图的最小边覆盖数 = 总顶点数 - 最大边匹配数 = 最大点独立集

二分图的最大点独立集 = 总顶点数 - 最大边匹配数

【很显然的把最大匹配两端的点都从顶点集中去掉这个时候剩余的点是独立集，这是 $|V|-2*|M|$ ，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持其独立集性质】二分图的最大边独立集 = 最大匹配数

DAG的最小路径覆盖 = 将每个点拆点后作最大匹配，结果为 $n-m$

结论

6个及以上的点集合，一定含有团或者独立集 (≥ 3 个点的大小)

$$KM O(n^3)$$

```
const int INF = 0x3f3f3f3f;
const int maxn = 205;
const int N = 205;
int nx, ny; // point num
int g[maxn][maxn]; // graph
int linker[maxn], lx[maxn], ly[maxn];
int slack[N];
bool visx[N], visy[N];
bool dfs(int x) {
    visx[x] = 1;
```

```

for (int y = 0; y < ny; y++) {
    if (visy[y]) continue;
    int tmp = lx[x] + ly[y] - g[x][y];
    if (tmp == 0) {
        visy[y] = 1;
        if (linker[y] == -1 || dfs(linker[y])) {
            linker[y] = x;
            return 1;
        }
    }
    else if (slack[y] > tmp) slack[y] = tmp;
}
return false;
}

int KM() {
    memset(linker, -1, sizeof linker);
    memset(ly, 0, sizeof ly);
    for (int i = 0; i < nx; i++) {
        lx[i] = -INF;
        for (int j = 0; j < ny; j++) {
            if (g[i][j] > lx[i]) {
                lx[i] = g[i][j];
            }
        }
    }
    for (int x = 0; x < nx; x++) {
        memset(slack, 0x3f, sizeof slack);
        while (1) {
            memset(visx, 0, sizeof visx);
            memset(visy, 0, sizeof visy);
            if (dfs(x)) break;
            int d = INF;
            for (int i = 0; i < ny; i++) {
                if (!visy[i] && d > slack[i]) {
                    d = slack[i];
                }
            }
            for (int i = 0; i < nx; i++) {
                if (visx[i]) {
                    lx[i] -= d;
                }
            }
            for (int i = 0; i < ny; i++) {
                if (visy[i]) {
                    ly[i] += d;
                }
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for (int i = 0; i < ny; i++) {
        if (~linker[i]) {
            res += g[linker[i]][i];
        }
    }
}

```

```

    return res;
}

```

Kruskal $O(n \log n)$

```

const int MAX = 1e5 + 5;

struct edge {
    int u, v, w;
    edge(int uu = 0, int vv = 0, int ww = 0) : u(uu), v(vv), w(ww) {}
    bool operator< (const edge& b) const { return w < b.w; }
}e[MAX * 2];

int tot, cnt, ans, fa[MAX];

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

inline void addedge(int u, int v, int w) { e[tot++] = edge(u, v, w); }

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

void kruskal(int n) {
    sort(e, e + tot);
    for (int i = 0; i < tot && cnt < n - 1; i++) {
        int u = find(e[i].u), v = find(e[i].v);
        if (u != v) {
            cnt++, ans += e[i].w;
            fa[fa[v]] = fa[u];
        }
    }
}

```

Dijkstra $O(n \log n)$

```

const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct edge { int v, w; edge(int vv = 0, int ww = 0) : v(vv), w(ww) {} };

vector<edge> G[MAX];
int dis[MAX], vis[MAX];

```

```

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

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

inline bool chkmin(int& x, int y) { return (x > y) ? (x = y, 1) : 0; }

void Dijkstra(int st, int ed) {
    priority_queue<pair<int, int>> Q;
    Q.emplace(0, st); dis[st] = 0;
    while (!Q.empty()) {
        int u = Q.top().second; Q.pop();
        if (u == ed) return;
        if (vis[u]) continue; else vis[u] = 1;
        for (edge& e : G[u]) {
            if (!vis[e.v] && chkmin(dis[e.v], dis[u] + e.w)) {
                Q.emplace(-dis[e.v], e.v);
            }
        }
    }
}

```

SPFA

```

const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct edge { int v, w; edge(int vv = 0, int ww = 0) : v(vv), w(ww) {} };

vector<edge> e[MAX];
int dis[MAX], inq[MAX];
queue<int> Q;

void init(int n) {
    for (; !Q.empty(); Q.pop());
    for (int i = 0; i <= n; i++) {
        e[i].clear();
        dis[i] = INF, inq[i] = 0;
    }
}

void addedge(int u, int v, int w) {
    e[u].emplace_back(v, w);
    e[v].emplace_back(u, w);
}

```

```

}

void spfa(int st, int ed) {
    Q.push(st); dis[st] = 0; inq[st] = 1;
    while (!Q.empty()) {
        int u = Q.front(); Q.pop(); inq[u] = 0;
        for (edge v : e[u]) {
            if (dis[v.v] > dis[u] + v.w) {
                dis[v.v] = dis[u] + v.w;
                if (!inq[v.v]) Q.push(v.v), inq[v.v] = 1;
            }
        }
    }
}

```

Dinic $O(V^2 E)$

```

const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;
struct edge {
    int v, w, next;
    edge(int vv = 0, int ww = 0, int nn = 0) : v(vv), w(ww), next(nn) {}
}e[MAX << 2];
int n, tot, ans, head[MAX], level[MAX];

void init() {
    tot = ans = 0;
    for (int i = 0; i <= n; i++) {
        head[i] = -1;
    }
}

void addedge(int u, int v, int w) {
    e[tot] = edge(v, w, head[u]); head[u] = tot++;
    e[tot] = edge(u, 0, head[v]); head[v] = tot++;
}

queue<int> Q;
bool bfs(int st, int ed) {
    for (int i = 0; i <= n; i++) level[i] = 0;
    while (!Q.empty()) Q.pop();
    Q.push(st); level[st] = 1;
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        for (int i = head[u]; ~i; i = e[i].next) {
            int v = e[i].v, w = e[i].w;
            if (level[v] == 0 && w > 0) {
                level[v] = level[u] + 1;
                Q.push(v);
            }
        }
    }
}

```

```

    }
    return level[ed] != 0;
}

int dfs(int u, int ed, int flow) {
    if (u == ed) return flow;
    int ret = 0;
    for (int i = head[u]; flow > 0 && (~i); i = e[i].next) {
        int v = e[i].v;
        if (level[v] == level[u] + 1 && e[i].w > 0) {
            int tmp = dfs(v, ed, min(flow, e[i].w));
            if (tmp == 0) continue;
            e[i].w -= tmp; e[i ^ 1].w += tmp;
            flow -= tmp; ret += tmp;
        }
    }
    return ret;
}

void Dinic(int st, int ed) {
    while (bfs(st, ed)) {
        ans += dfs(st, ed, INF);
    }
}

```

Tarjan Sceno(强连通) $O(n + m)$

```

const int MAX = 1e5 + 5;
vector<int> e[MAX];
stack<int> S;
int index, tot, def[MAX], low[MAX], ins[MAX], scc[MAX];

void init(int n) {
    for (int i = 0; i <= n; i++) {
        e[i].clear();
        scc[i] = -1;
    }
}

inline void addedge(int u, int v) {
    e[u].push_back(v);
}

void dfs(int u) {
    def[u] = low[u] = ++index;
    S.push(u); ins[u] = 1;
    int v;
    for (int i = 0; i < (int)e[u].size(); i++) {
        v = e[u][i];
        if (!def[v]) {
            dfs(v);
        }
    }
    low[u] = min(low[u], def[v]);
    if (def[u] == low[u]) {
        int v;
        do {
            v = S.top(); S.pop();
            scc[v] = tot++;
        } while (v != u);
    }
    ins[u] = 0;
}

```

```

        low[u] = min(low[u], low[v]);
    }
    else if (ins[v]) {
        low[u] = min(low[u], def[v]);
    }
}
if (def[u] == low[u]) {
    ++tot;
    do {
        v = S.top();
        S.pop(); ins[v] = 0;
        scc[v] = tot;
    } while (u != v);
}
}

void solve(int n) {
    index = tot = 0;
    for (int i = 0; i <= n; i++) {
        def[i] = low[i] = 0;
    }
    while (!S.empty()) S.pop();
    for (int i = 1; i <= n; i++) {
        (!def[i]) && (dfs(i), 1);
    }
}

```

倍增法LCA $O(\log n)$

```

const int maxn = 1e4 + 5;

vector<int> e[maxn];
int dep[maxn], dp[maxn][15], maxb;

void init(int n) {
    for (int i = 0; i < maxn; i++) {
        e[i].clear(), dep[i] = 0;
        memset(dp[i], -1, sizeof dp[i]);
    }
    for (maxb = 0; (1 << maxb) <= n; ++maxb);
}

void DFS(int u, int d, int pre) {
    dp[u][0] = pre;
    dep[u] = d;
    for (int j = 1; j < maxb; j++) {
        if (~dp[u][j - 1]) {
            dp[u][j] = dp[dp[u][j - 1]][j - 1];
        }
    }
    for (int i = 0; i < (int)e[u].size(); i++) {

```

```

        if (e[u][i] != pre) {
            DFS(e[u][i], d + 1, u);
        }
    }
}

int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    for (int j = maxb - 1; ~j; j--) {
        if (dep[dp[u][j]] >= dep[v]) {
            u = dp[u][j];
        }
    }
    if (u == v) return u;
    for (int j = maxb - 1; ~j; j--) {
        if (dp[u][j] != dp[v][j]) {
            u = dp[u][j], v = dp[v][j];
        }
    }
    return dp[u][0];
}

```

Tarjan LCA $O(n + q)$ (离线)

```

const int maxn = 1e5 + 5;

struct query {
    int v, i;
    query(int a = 0, int b = 0) : v(a), i(b) {}
};

int fa[maxn];
int ans[maxn], color[maxn];
vector<int> e[maxn];
vector<query> Q[maxn];

void init() {
    for (int i = 0; i < maxn; i++) {
        fa[i] = i, ans[i] = -1, color[i] = 0;
        e[i].clear(), Q[i].clear();
    }
}

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

bool unite(int u, int v) {
    u = find(u), v = find(v);
    if (u != v) {
        fa[v] = u;
        return true;
    }
}

```



```

        return false;
    }

    void addedge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }

    void addquery(int u, int v, int i) {
        Q[u].push_back(query(v, i));
        Q[v].push_back(query(u, i));
    }

    void DFS(int u, int pre) {
        for (int i = 0; i < (int)e[u].size(); i++) {
            if (!color[e[u][i]] && e[u][i] != pre) {
                DFS(e[u][i], u);
            }
        }
        for (int i = 0; i < (int)Q[u].size(); i++) {
            if (color[Q[u][i].v]) {
                ans[Q[u][i].i] = find(Q[u][i].v);
            }
        }
        color[u] = 1;
        if (pre != -1) unite(pre, u);
    }

    void solve() { DFS(1, -1); }

```

拓扑排序 $O(n + m)$

```

const int maxn = 1e5 + 5;
vector<int> e[maxn];
int seq[maxn], vis[maxn], tot, circle;

void init(int n) {
    tot = 0; circle = 0;
    for (int i = 0; i <= n; i++) {
        e[i].clear(), vis[i] = 0;
    }
}

void addedge(int u, int v) {
    e[u].push_back(v);
}

void DFS(int u) {
    vis[u] = 1;
    for (int i = 0; i < (int)e[u].size(); i++) {
        int& v = e[u][i];
        if (vis[v] == 1) circle = true;
    }
}

```

```

        else if (vis[v] == 0) DFS(v);
    }
    vis[u] = 2;
    seq[tot++] = u;
}

void solve(int n) {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            DFS(i);
        }
    }
    reverse(seq, seq + tot);
}

```

String

AC_automaton

```

struct Aho_Corasick {
    static const int maxn = 5e5 + 5000;
    static const int sigma = 26;

    int tot, son[maxn][sigma], cnt[maxn], fail[maxn];

    inline void init() {
        tot = cnt[0] = fail[0] = 0;
        memset(son[0], 0, sizeof son[0]);
    }

    inline int trans(int x) {
        return x - 'a';
    }

    void insert(char *s) {
        int now = 0, n = strlen(s);
        for (int i = 0; i < n; i++) {
            int c = trans(s[i]);
            if (!son[now][c]) {
                cnt[++tot] = 0; fail[tot] = 0;
                memset(son[tot], 0, sizeof son[tot]);
                son[now][c] = tot;
            }
            now = son[now][c];
        }
        cnt[now]++;
    }

    std::queue<int> Q;

    void build() {

```

```

while (!Q.empty()) Q.pop();
for (int i = 0; i < sigma; i++) {
    if (son[0][i]) {
        Q.push(son[0][i]);
    }
}
while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int i = 0; i < sigma; i++) {
        if (son[u][i]) {
            fail[son[u][i]] = son[fail[u]][i];
            Q.push(son[u][i]);
        }
        else son[u][i] = son[fail[u]][i];
    }
}
}

int solve(char *s) {
    int ret = 0, n = strlen(s);
    for (int i = 0, now = 0, c; i < n; i++, now = son[now][c]) {
        c = trans(s[i]);
        for (int u = son[now][c]; u; u = fail[u]) {
            ret += cnt[u]; cnt[u] = 0;
        }
    }
    return ret;
}
}Accepted;

```

回文树

```

struct Palindromic_Tree {
    static const int maxn = 2e6 + 5;
    static const int char_db = 10;

    int tot, len[maxn], fail[maxn], ch[maxn][char_db];
    long long sum[maxn];

    // even root -> 0, odd root -> 1

    inline int newnode(int val) {
        sum[tot] = 0, len[tot] = val;
        memset(ch[tot], 0, sizeof ch[tot]);
        return tot++;
    }

    void init() {
        tot = 0;
        newnode(0); newnode(-1);
        fail[0] = 1, fail[1] = 0;
    }
}

```

```

    }

    int getfail(char *s, int cur, int i) {
        while (i - len[cur] - 1 < 0 || s[i] != s[i - len[cur] - 1]) {
            cur = fail[cur];
        }
        return cur;
    }

    void build(char *s, int n) {
        int cur = 1;
        for (int i = 0; i < n; i++) {
            cur = getfail(s, cur, i);
            if (!ch[cur][s[i] - '0']) {
                int nxt = newnode(len[cur] + 2);
                fail[nxt] = ch[getfail(s, fail[cur], i)][s[i] - '0'];
                ch[cur][s[i] - '0'] = nxt;
            }
            cur = ch[cur][s[i] - '0'];
        }
    }
}pt;

```

strHash

```

struct strhash {
    vector<ull> h, p;
    strhash(int n = 0) : h(n + 5, 0), p(n + 5, 0) {}
    void init(char *s) {
        for (int i = 0; s[i]; i++) {
            if (i) h[i] = h[i - 1] * 131, p[i] = p[i - 1] * 131;
            else p[i] = 1;
            h[i] += s[i] - 'a' + 1;
        }
    }
    inline ull gethash(int l, int r) {
        ull ret = h[r];
        if (l) ret -= h[l - 1] * p[r - l + 1];
        return ret;
    }
};

```

后缀数组

```

namespace SuffixArray {

```

```

const int maxn = "edit";

int wa[maxn], wb[maxn], c[maxn], d[maxn];

inline bool cmp(int *r, int a, int b, int k) {
    return (r[a] == r[b]) && (r[a + k] == r[b + k]);
}

void da(int *r, int *sa, int n, int m) {
    int i, j, p, *x = wa, *y = wb, *t;

    for (i = 0; i < m; i++) d[i] = 0;
    for (i = 0; i < n; i++) d[x[i]] = r[i]++;

    for (i = 1; i < m; i++) d[i] += d[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--d[x[i]]] = i;

    for (j = 1, p = 1; j <= n; j <= 1, m = p) {
        for (p = 0, i = n - j; i < n; i++) y[p++] = i;
        for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;

        for (i = 0; i < n; i++) c[i] = x[y[i]];
        for (i = 0; i < m; i++) d[i] = 0;

        for (i = 0; i < n; i++) d[c[i]]++;
        for (i = 1; i < m; i++) d[i] += d[i - 1];

        for (i = n - 1; i >= 0; i--) sa[--d[c[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? (p - 1) : (p++);
        }
    }
}

int rank[maxn], height[maxn];
void calheight(int *r, int *sa, int n) {
    int i, j, k = 0;
    for (i = 1; i <= n; i++) rank[sa[i]] = i;
    for (i = 0; i < n; i++) {
        if (k) --k;
        for (j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
        // blank
        height[rank[i]] = k;
    }
}
}

```

Suffix Array

- usage:

```
// s原数组, sa, n数组长度, m值域[0, m)
SuffixArray::da(s, sa, n, m);
// SuffixArray::rank, SuffixArray::height
calheight(r, sa, n);
```

- sa数组: sa_i 为 $Suffix_i$ 在字符串中第一次出现的位置
- height数组: $height_i = LongestPrefix(suffix(sa_{i-1}), suffix(sa_i))$
- $suffix_j$ 和 $suffix_k$ 的最长公共前缀为 $\min(height_{rank_j+1}, height_{rank_j+2}, \dots, height_{rank_k})$
- 求字符串中某两个后缀的最长公共前缀: 可转化为求height中某区间的最小值, 即RMQ问题
- 可重叠最长重复子串: $ans = \max height_i$
- 不可重叠最长重复子串: 二分答案, 扫height数组, 将连续的满足 $height_i \geq k$ 的后缀分为一组, 若同一组中存在 $sa_{max} - sa_{min} \geq k$, 说明存在一组不重叠的重复子串
- 可重叠的k次最长重复子串: 二分答案, 扫height数组, 分组, 判断是否存在个数不小于k的组

KMP

```
int n, m;
char s[1005], t[1005];
int fail[1005];

void getfail() {
    int i, j;
    n = strlen(s), m = strlen(t);
    for (i = 0, j = fail[0] = -1; i < m; i++) {
        while (j >= 0 && t[j] != t[i]) j = fail[j];
        fail[i + 1] = j + 1;
    }
}

int solve() {
    getfail();
    int i, j, res;
    for (i = j = res = 0; i < n; ) {
        while (j >= 0 && s[i] != t[j]) {
            j = fail[j];
        }
        ++i, ++j;
        if (j >= m) {
            ++res, j = 0;
        }
    }
    return res;
}
```

exKMP

```
// nxt[i]: t[i...m-1]与t[0...m-1]的最长公共前缀
// extend[i]: s[i...n-1]与t[0...m-1]的最长公共前缀

const int maxn = 1e6 + 5;
int nxt[maxn], extend[maxn];
void exkmp(char *s, int n, char *t, int m) {
    int j = 0, k = 1;
    for (; j + 1 < m && t[j] == t[j + 1]; ++j);
    nxt[0] = m, nxt[1] = j;

    for (int i = 2; i < m; i++) {
        int p = nxt[k] + k - 1, L = nxt[i - k];
        if (p + 1 - i - L > 0) {
            nxt[i] = L;
        }
        else {
            for (j = (p - i + 1 > 0) ? (p - i + 1) : 0; i + j < m && t[i + j] == t[j]; ++j);
            // blank
            nxt[i] = j, k = i;
        }
    }

    j = k = 0;
    for (; j < n && j < m && t[j] == s[j]; ++j);
    extend[0] = j;

    for (int i = 1; i < n; i++) {
        int p = extend[k] + k - 1, L = nxt[i - k];
        if (p + 1 - i - L > 0) {
            extend[i] = L;
        }
        else {
            for (j = (p - i + 1 > 0) ? (p - i + 1) : 0; i + j < n && j < m && s[i + j] ==
t[j]; ++j);
            // blank
            extend[i] = j, k = i;
        }
    }
}
```

manacher

```
const int maxn = 110005;

int p[maxn << 1];
char str[maxn << 1];

int manacher(char *s, int n) {
```

```

str[0] = '$'; str[1] = '#';

for (int i = 0; i < n; i++) {
    str[(i << 1) + 2] = s[i];
    str[(i << 1) + 3] = '#';
}
n = (n + 1) << 1;
str[n] = 0;

int ret = 0, mx = 0, pos;
for (int i = 1; i < n; i++) {
    p[i] = mx > i ? min(p[(pos << 1) - i], mx - i) : 1;

    while (str[i - p[i]] == str[i + p[i]]) p[i]++;

    if (p[i] + i > mx) mx = p[i] + i, pos = i;

    ret = max(ret, p[i]);
}
return ret - 1;
}

// *****

// index start from one
int solve(char *s, int n) {
    int mx = 1; s[0] = '$'; s[++n] = 0;
    for (int i = 0, p, q; i < n; i++) {
        for (q = i; s[i + 1] == s[i]; ++i);
        for (p = i; s[q - 1] == s[p + 1]; --q, ++p);
        mx = max(mx, p - q + 1);
    }
    return mx;
}

```

数论

九余数定理

- 一个数各位数字之和等于这个数对9取模所得的数
- 每次将指数进行一次 $\log(N)$ 级别的变换
- 矩阵快速幂: 在 $O(\log(N))$ 级别的时间求第 n 个斐波那契数列 $f(n)=af(n-1)+bf(n-2)$
- 快速乘: 利用二进制实现 $ab \pmod p$, 防止溢出

母函数(组合数学)

- 详见模板和实例

五边形数定理

- 五边形数: 1, 5, 12, 22,
- 第 $(n-1)$ 个三角数 $+n^2$ 为第 n 个五边形数
- $S_n = S_{n-1} + 3n - 2$

zeckendorf定理

- 任何正整数可以表示为若干个不连续的Fibonacci数之和(斐波纳契博弈)

错排公式

- $d(n) = (n-1)(d[n-2] + d[n-1])$

不定方程

- 二元一次不定方程 $ax+by=c$ 有解的充要条件是 $(a,b)|c$

欧拉定理

- 欧拉函数: $\varphi(n)$ 是小于等于 n 的正整数中与 n 互质的数的数目
- 若 n, a 为正整数且 n 与 a 互质, 则 $a^{\varphi(n)} \equiv 1 \pmod{n}$
- 费马小定理(Fermat小定理): 对任意 a 和任意质数 p : $a^p \equiv a \pmod{p}$, 若 a 不能被 p 整除, $a^{(p-1)} \equiv 1 \pmod{p}$
- 欧拉降幂: $x^y \bmod p = x^{y \bmod \phi(p) + \phi(p)} \bmod p, y \geq \phi(p)$

```
# (x ^ y) % p
def calc(x, y, p):
    if y < p:
        return qpow(x, y, p)
    else:
        # (x ^ y) % p = (x ^ (y % phi(p) + phi(p))) % p
        return qpow(x, y % phi[p] + phi[p], p)
```

费马大定理 && 费马方程

- $x^n + y^n = z^n$, 由费马大定理可知: 若 $n \geq 2$ 且 n 为整数, 则不存在整数解 $(x, y, z)(xyz \neq 0)$

SG(Sprague-Grundy)函数

- 对于任意状态 x , 它的SG函数值 $g(x)=\text{mex}\{g(y)|y\text{是}x\text{的后继状态}\}$, mex 是一个对于非负整数集合 S 的运算, $\text{mex}(S)$ 为 S 中没有出现的最小非负整数
- 终止状态的SG函数值为0
- 博弈打表(待更新)

pick定理(实际上属于计算几何)

- 给定正方形格子点的简单多边形, i 为其内部格点数目, b 为其边上格点数, 则其面积 $A = i + b/2 - 1$

逆元(费马小定理)

- 当 p 为素数时, $a/b \pmod p = ab_1 \pmod p, b_1 = b^{p-2} \pmod p$

威尔逊定理

- 当且仅当 p 为素数时: $(p-1)! \equiv -1 \pmod p$

杨辉三角(应用于排列组合)

- 第 n 行的元素个数有 n 个
- 第 n 行的元素之和为 2^{n-1}
- 第 n 行第 m 个数的值为 $C(n-1, m-1)$, C 为组合数
- $(a+b)^n$ 展开后的各项系数等于第 $n+1$ 行的值
- 第 n 行第 m 个数的奇偶判断 $(m-1) \& (n-1) == (m-1)? \text{odd} : \text{even}$

第一类斯特林数

- $S(p, k)$ 表示把 p 个人分成 k 组作环排列的方案数
- $S(p, k) = (p-1) * S(p-1, k) + S(p-1, k-1), 1 \leq k \leq p-1$
- $S(p, 0) = 0, p \geq 1$
- $S(p, p) = 1, p \geq 0$
- 使第 p 个人单独构成一个环排列, 前 $p-1$ 人构成 $k-1$ 个环排列, 方案数 $S(p-1, k-1)$ 。
- 使前 $p-1$ 个人构成 k 个环排列, 第 p 个人插入到第 i 人左边, 方案数 $S(p-1, k)$

第二类斯特林数

- 将 p 个物体分成 k 个非空的不可辨别的集合的方案数
- $S(p, k) = k * S(p - 1, k) + S(p - 1, k - 1), 1 \leq k \leq p - 1$
- $S(p, 0) = 0, p \geq 1$
- $S(p, p) = 1, p \geq 0$
- 考虑第 p 个物品， p 可以单独构成一个非空集合，此时前 $p-1$ 个物品构成 $k-1$ 个非空的不可辨别的集合，方法数为 $S(p - 1, k - 1)$
- 也可以前 $p-1$ 种物品构成 k 个非空的不可辨别的集合，第 p 个物品放入任意一个中，这样有 $kS(p - 1, k)$ 种方法。

Lucas定理(大组合数取模)

- $C(n, m) \bmod p = C(n/p, m/p)C(n \bmod p, m \bmod p) \bmod p$, 其中 p 为质数

```
// C(n, m) % p = C(n / p, m / p) * C(n % p, m % p) % p
// p is a prime

typedef long long ll;
const int MOD = 1e9 + 7;

// calc C(n, m) % MOD
inline ll C(ll n, ll m);

ll lucas(ll n, ll m) {
    if (n < MOD && m < MOD) {
        return C(n, m);
    }
    return C(n % MOD, m % MOD) * lucas(n / MOD, m / MOD) % MOD;
}
```

欧拉&素数线性筛

```
struct Seive {
    int maxn;
    vector<int> phi;
    Seive(int n) : maxn(n + 5), phi(n + 5) {
        phi[0] = 0;
        phi[1] = 1;
        for(int i = 2; i < maxn; i++) {
            if(!phi[i]) {
                for(int j = i; j < maxn; j += i) {
                    if(!phi[j]) {
                        phi[j] = j;
                    }
                    phi[j] -= phi[j] / i;
                }
            }
        }
    }
}
```

```

    }
}
inline bool chkpri(int x) { return phi[x] == x - 1; }
inline int getphi(int x) { return phi[x]; }
};

```

组合数打表

```

const int MOD = 1e9 + 7;
long long C[1005][1005];
void init() {
    for (int i = C[0][0] = 1; i < 1005; i++) {
        for (int j = C[i][0] = 1; j <= i; j++) {
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % MOD;
        }
    }
}

```

Simpson求积分

```

double simpson(const double& a, const double& b)
{
    double c = (a + b) / 2;
    return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
}

double asr(double a, double b, double eps, double A)
{
    double c = (a + b) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps)
        return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}

```

Meissel-Lehmer(求 $[1, n]$ 之间的素数个数) $O(n^{2/3})$

```

namespace pcf{
    typedef long long ll;
    const int N = 5e6 + 2;
    bool np[N];
    int prime[N], pi[N];

    int getprime() {

```

```

    int cnt = 0;
    np[0] = np[1] = 1;
    pi[0] = pi[1] = 0;
    for (int i = 2; i < N; i++) {
        if (!np[i]) prime[++cnt] = i;
        pi[i] = cnt;
        for (int j = 1; j <= cnt && i * prime[j] < N; ++j) {
            np[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    return cnt;
}

const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;

int phi[PM + 1][M + 1], sz[M + 1];

void init() {
    getprime();
    sz[0] = 1;
    for (int i = 0; i <= PM; i++) {
        phi[i][0] = i;
    }
    for (int i = 1; i <= M; i++) {
        sz[i] = prime[i] * sz[i - 1];
        for (int j = 1; j <= PM; j++) {
            phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];
        }
    }
}

int sqrt2(ll x) {
    ll r = ll(sqrt(x - 0.1));
    while (r * r <= x) ++r;
    return int(r - 1);
}

int sqrt3(ll x) {
    ll r = ll(cbrt(x - 0.1));
    while (r * r * r <= x) ++r;
    return int(r - 1);
}

ll getphi(ll x, int s) {
    if (s == 0) {
        return x;
    }
    if (s <= M) {
        return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    }
    if (x <= prime[s] * prime[s]) {
        return pi[x] - s + 1;
    }
    if (x <= prime[s] * prime[s] * prime[s] && x < N) {
        int s2x = pi[sqrt2(x)];
    }
}

```

```

        ll ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
        for (int i = s + 1; i <= s2x; i++) {
            ans += pi[x / prime[i]];
        }
        return ans;
    }
    return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
}

ll getpi(ll x) {
    if (x < N) {
        return pi[x];
    }
    ll ans = getphi(x, pi[sqrt3(x)] + pi[sqrt3(x)] - 1;
    for (int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) {
        ans -= getpi(x / prime[i]) - i + 1;
    }
    return ans;
}

ll lehmer(ll x) {
    if (x < N) {
        return pi[x];
    }
    int a = int(lehmer(sqrt2(sqrt2(x))));
    int b = int(lehmer(sqrt2(x)));
    int c = int(lehmer(sqrt3(x)));
    ll sum = getphi(x, a) + ll(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++) {
        ll w = x / prime[i];
        sum -= lehmer(w);
        if (i > c) {
            continue;
        }
        ll lim = lehmer(sqrt2(w));
        for (int j = i; j <= lim; j++) {
            sum -= lehmer(w / prime[j]) - (j - 1);
        }
    }
    return sum;
}
}

```

扩展欧几里得与中国剩余定理

```

p11 exgcd(const ll x, const ll y)
{
    if (!y)
    {
        return make_pair(1, 0);
    }
    p11 cur = exgcd(y, x % y);

```

```

        return make_pair(cur.second, cur.first - (x / y) * cur.second);
    }
    pll crt(const vector<pll> & v)
    {
        //v里每个pll中first为被模数, second为模数
        ll a = 1, r = 0;
        const int len = v.size();
        for(int i = 0; i < len; i++) {
            pll cur = exgcd(a, v[i].first);
            ll gcd = a * cur.first + v[i].first * cur.second;
            if((v[i].second - r) % gcd != 0){
                return make_pair(-1, -1);
            }
            const ll p = v[i].first / gcd;
            r += mod(cur.first * ((v[i].second - r) / gcd), p) * a;
            a *= p;
        }
        return make_pair(a, r);
    }
}

```

大随机数生成与素性测试

```

ull randull()
{
    static random_device rd;
    static mt19937_64 eng(rd());
    static uniform_int_distribution<ull>distr;
    return distr(eng);
}
ull randint(ull const& min = 0, ull const& max = 0)
{
    return double(randull()) / ULLONG_MAX * (max - min + 1) + min;
}
bool is_prime(ll n)
{
    if (n == 2) return true;
    if (n < 2 || (~n & 1)) return false;
    ll m = n - 1, k = 0;
    while (!(m & 1))
    {
        k++;
        m >>= 1;
    }
    for (int i = 1; i <= 30; i++)
    {
        ll a = randint((ull)1, (ull)(n - 1));
        ll x = fpow(a, m, n);
        ll y;
        for (int j = 1; j <= k; j++)
        {
            y = fmul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1)

```

```

        {
            return false;
        }
        x = y;
    }
    if (y != 1)
    {
        return false;
    }
}
return true;
}

```

蔡勒公式

```

int zeller(int y, int m, int d)
{
    //星期日为0
    if (m == 1 || m == 2)
    {
        m += 12;
        y--;
    }
    int c = y / 100;
    y = y % 100;
    int w = y + y / 4 + c / 4 - 2 * c + (26 * (m + 1)) / 10 + d - 1;
    w = ((w % 7) + 7) % 7;
    return w;
}

```

后缀表达式

```

const int MXLEN = 1000 + 5;
int fst[MXLEN];
char str[MXLEN];

typedef double CSS;

CSS jud(int begin, int end)
{
    int i;
    CSS k;
    //越往后优先级越高
    for (i = begin; i <= end; i++)
    {
        if (str[i] == '+' && fst[i] == fst[begin])
        {
            k = jud(begin, i - 1) + jud(i + 1, end);

```



```

        return k;
    }
}
for (i = end; i >= begin; i--)
{
    if (str[i] == '-' && fst[i] == fst[begin])
    {
        k = jud(begin, i - 1) - jud(i + 1, end);
        return k;
    }
}
if (str[begin] == '(')
{
    for (i = begin + 1; fst[i] >= fst[begin + 1]; i++);

    k = jud(begin + 1, i - 1);
}
else
{
    char *p = str;
    sscanf(p + begin, "%lf", &k);
}
return k;
}

CSS solve()
{
    const int len = strlen(str);
    for (int i = 1; i <= len - 1; i++)
    {
        if (str[i - 1] == '(')
        {
            fst[i] = fst[i - 1] + 1;
        }
        else
        {
            if (str[i] == ')')
            {
                fst[i] = fst[i - 1] - 1;
            }
            else
            {
                fst[i] = fst[i - 1];
            }
        }
    }
    return jud(0, len);
}
}

```

java大数牛顿迭代法开方

```

public static BigInteger sqrt(BigInteger n) {

```

```

String a = n.toString();
final int len = a.length();
if((~len & 1) == 1) {
    a = a.substring(0, len / 2 + 1);
} else {
    a = a.substring(0, len / 2);
}
BigInteger x = new BigInteger(a);
final BigInteger two = BigInteger.valueOf(2);
if(a == "0" || a == "1") {
    return n;
} else {
    while(n.compareTo(x.multiply(x)) < 0) {
        x = (x.add(n.divide(x))).divide(two);
    }
    return x;
}
}

```

计算几何

多边形

```

#define MAXN 1000//点数量上限
#define offset 10000//点坐标上限
#define eps 1e-8
#define zero(x) (((x)>0?(x):- (x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?-1:0))
struct point{double x,y;};//点
struct line{point a,b;};//线
//叉积
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&!(s[1]|s[2]);i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[1]|s[2];
}
//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&(s[0]&&s[1]|s[2]);i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[0]&&s[1]|s[2];
}
//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
    int i,s[3]={1,1,1};

```

```

    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[1]|s[2];
}
//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0
int inside_convex_v2(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[0]&&s[1]|s[2];
}
//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge表示点在多边形边上时的返回值
int inside_polygon(point q,int n,point* p,int on_edge=1){
    point q2;
    int i=0,count;
    while (i<n)
        for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
            if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&
                (p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
                return on_edge;
            else if (zero(xmult(q,q2,p[i])))
                break;
            else if (xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-
                eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])<-eps)
                count++;
    return count&1;
}
inline int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

inline int dot_online_in(point p,point l1,point l2){
    return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1
int inside_polygon(point l1,point l2,int n,point* p){
    point t[MAXN],tt;
    int i,j,k=0;
    if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
        return 0;
    for (i=0;i<n;i++)
        if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
            return 0;
        else if (dot_online_in(l1,p[i],p[(i+1)%n]))
            t[k++]=l1;
        else if (dot_online_in(l2,p[i],p[(i+1)%n]))
            t[k++]=l2;
        else if (dot_online_in(p[i],l1,l2))
            t[k++]=p[i];
    for (i=0;i<k;i++)
        for (j=i+1;j<k;j++){
            tt.x=(t[i].x+t[j].x)/2;
            tt.y=(t[i].y+t[j].y)/2;
            if (!inside_polygon(tt,n,p))
                return 0;
        }
}

```

```

        return 1;
    }
    point intersection(line u,line v){
        point ret=u.a;
        double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
            /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
        ret.x+=(u.b.x-u.a.x)*t;
        ret.y+=(u.b.y-u.a.y)*t;
        return ret;
    }
    point barycenter(point a,point b,point c){
        line u,v;
        u.a.x=(a.x+b.x)/2;
        u.a.y=(a.y+b.y)/2;
        u.b=c;
        v.a.x=(a.x+c.x)/2;
        v.a.y=(a.y+c.y)/2;
        v.b=b;
        return intersection(u,v);
    }
    //多边形重心
    point barycenter(int n,point* p){
        point ret,t;
        double t1=0,t2;
        int i;
        ret.x=ret.y=0;
        for (i=1;i<n-1;i++){
            if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
                t=barycenter(p[0],p[i],p[i+1]);
                ret.x+=t.x*t2;
                ret.y+=t.y*t2;
                t1+=t2;
            }
        }
        if (fabs(t1)>eps)
            ret.x/=t1,ret.y/=t1;
        return ret;
    }
}

```

多边形切割

```

#define MAXN 1000//点数量上限
#define offset 10000//点坐标上限
#define eps 1e-8
#define zero(x) (((x)>0?(x):-<(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?-1:0))
struct point{double x,y;};//点
struct line{point a,b;};//线
//可用于半平面交
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
int same_side(point p1,point p2,point l1,point l2){

```

```

        return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
    }
    point intersection(point u1,point u2,point v1,point v2){
        point ret=u1;
        double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
        ret.x+=(u2.x-u1.x)*t;
        ret.y+=(u2.y-u1.y)*t;
        return ret;
    }
    //将多边形沿l1,l2确定的直线切割在side侧切割,保证l1,l2,side不共线
    void polygon_cut(int& n,point* p,point l1,point l2,point side){
        point pp[MAXN];
        int m=0,i;
        for (i=0;i<n;i++){
            if (same_side(p[i],side,l1,l2))
                pp[m++]=p[i];
            if (!same_side(p[i],p[(i+1)%n],l1,l2)&&!
                (zero(xmult(p[i],l1,l2))&&zero(xmult(p[(i+1)%n],l1,l2))))
                pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);
        }
        for (n=i=0;i<m;i++)
            if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))
                p[n++]=pp[i];
        if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))
            n--;
        if (n<3)
            n=0;
    }
}

```

浮点函数

```

//浮点几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-<(x))<eps)
struct point{double x,y;};
struct line{point a,b;};
//计算cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
//计算dot product (P1-P0).(P2-P0)
double dmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}

```

```

//两点距离
double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
//判三点共线
int dots_inline(point p1,point p2,point p3){
    return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
    return zero(xmult(x1,y1,x2,y2,x3,y3));
}
//判点是否在线段上,包括端点
int dot_online_in(point p,line l){
    return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2){
    return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
    return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
}
//判点是否在线段上,不包括端点
int dot_online_ex(point p,line l){
    return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-
1.b.x)||!zero(p.y-l.b.y));
}
int dot_online_ex(point p,point l1,point l2){
    return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-
12.x)||!zero(p.y-l2.y));
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
    return dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-
x2)||!zero(y-y2));
}
//判两点在线段同侧,点在线段上返回0
int same_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}
//判两点在线段异侧,点在线段上返回0
int opposite_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
//判两直线平行
int parallel(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
    return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
}

```

```

}
//判两直线垂直
int perpendicular(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
    return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}
//判两线段相交,包括端点和部分重合
int intersect_in(line u,line v){
    if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
    if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
        return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
    return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u
2);
}
//判两线段相交,不包括端点和部分重合
int intersect_ex(line u,line v){
    return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
    return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
//计算两直线交点,注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point intersection(line u,line v){
    point ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
        /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}
point intersection(point u1,point u2,point v1,point v2){
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
        /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}
//点到直线上的最近点
point ptoline(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    return intersection(p,t,l1,l2);
}

```

```

}
//点到直线距离
double disptoline(point p,line l){
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
    return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
}
//点到线段上的最近点
point ptoseg(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
    return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
        return distance(p,l1)<distance(p,l2)?l1:l2;
    return intersection(p,t,l1,l2);
}
//点到线段距离
double disptoseg(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
        return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
//矢量V以P为顶点逆时针旋转angle并放大scale倍
point rotate(point v,point p,double angle,double scale){
    point ret=p;
    v.x-=p.x,v.y-=p.y;
    p.x=scale*cos(angle);
    p.y=scale*sin(angle);
    ret.x+=v.x*p.x-v.y*p.y;
    ret.y+=v.x*p.y+v.y*p.x;
    return ret;
}
//p点关于直线L的对称点
point symmetricalPointofLine(point p, line L)
{
    point p2;
    double d;
    d = L.a * L.a + L.b * L.b;

```



```

    p2.x = (L.b * L.b * p.x - L.a * L.a * p.x -
            2 * L.a * L.b * p.y - 2 * L.a * L.c) / d;
    p2.y = (L.a * L.a * p.y - L.b * L.b * p.y -
            2 * L.a * L.b * p.x - 2 * L.b * L.c) / d;
    return p2;
}
//求两点的平分线
line bisector(point& a, point& b) {
    line ab, ans; ab.set(a, b);
    double midx = (a.x + b.x)/2.0, midy = (a.y + b.y)/2.0;
    ans.a = -ab.b, ans.b = -ab.a, ans.c = -ab.b * midx + ab.a * midy;
    return ans;
}
// 已知入射线、镜面，求反射线。
// a1,b1,c1为镜面直线方程(a1 x + b1 y + c1 = 0 ,下同)系数;
a2,b2,c2为入射光直线方程系数;
a,b,c为反射光直线方程系数.
// 光是有方向的，使用时注意：入射光向量:<-b2,a2>; 反射光向量:<b,-a>.
// 不要忘记结果中可能会有"negative zeros"
void reflect(double a1,double b1,double c1,
double a2,double b2,double c2,
double &a,double &b,double &c)
{
    double n,m;
    double tpb,tpa;
    tpb=b1*b2+a1*a2;
    tpa=a2*b1-a1*b2;
    m=(tpb*b1+tpa*a1)/(b1*b1+a1*a1);
    n=(tpa*b1-tpb*a1)/(b1*b1+a1*a1);
    if(fabs(a1*b2-a2*b1)<1e-20)
    {
        a=a2;b=b2;c=c2;
        return;
    }
    double xx,yy; //(xx,yy)是入射线与镜面的交点。
    xx=(b1*c2-b2*c1)/(a1*b2-a2*b1);
    yy=(a2*c1-a1*c2)/(a1*b2-a2*b1);
    a=n;
    b=-m;
    c=m*yy-xx*n;
}

```

面积

```

#include math.h
struct point{double x,y};
//计算cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)(p2.y-p0.y)-(p2.x-p0.x)(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)(y2-y0)-(x2-x0)(y1-y0);
}

```

```

}
//计算三角形面积,输入三顶点
double area_triangle(point p1,point p2,point p3){
    return fabs(xmult(p1,p2,p3))/2;
}
double area_triangle(double x1,double y1,double x2,double y2,double x3,double y3){
    return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;
}
//计算三角形面积,输入三边长
double area_triangle(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
//计算多边形面积,顶点按顺时针或逆时针给出
double area_polygon(int n,point p){
    double s1=0,s2=0;
    int i;
    for (i=0;i<n;i++){
        s1+=p[(i+1)%n].yp[i].x,s2+=p[(i+1)%n].yp[(i+2)%n].x;
    }
    return fabs(s1-s2)/2;
}

```

球面

```

#include <math.h>
const double pi=acos(-1);
//计算圆心角lat表示纬度, -90<=w<=90,lng表示经度
//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi
double angle(double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return acos(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2));
}
//计算距离,r为球半径
double line_dist(double r,double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return r*sqrt(2-2*(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2)));
}
//计算球面距离,r为球半径
inline double sphere_dist(double r,double lng1,double lat1,double lng2,double lat2){
    return r*angle(lng1,lat1,lng2,lat2);
}
//球面反射

```

```

#include <stdio>
#include <math>
const int size = 555;
const double eps = 1e-9;
struct point {double x, y, z;} centre = {0, 0, 0};
struct circle {point o; double r;} cir[size];
struct ray {point s, dir;} l;
int n;
int dcmp (double x){return x < -eps ? -1 : x > eps;}
double sqr (double x){return x*x;}
double dot (point a, point b){return a.x * b.x + a.y * b.y + a.z * b.z;}
double dis2 (point a, point b){return sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);}
double disToLine2 (point a, ray l){**** 点到直线L的距离的平方 **/
    point tmp;
    tmp.x = l.dir.y * (a.z - l.s.z) - l.dir.z * (a.y - l.s.y);
    tmp.y = -l.dir.x * (a.z - l.s.z) + l.dir.z * (a.x - l.s.x);
    tmp.z = l.dir.x * (a.y - l.s.y) - l.dir.y * (a.x - l.s.x);
    return dis2 (tmp, centre) / dis2 (l.dir, centre);
}
**** 用向量法求交点 ****/
bool find (circle p, ray l, double &k, point &t)
{
    double h2 = disToLine2 (p.o, l);
    // printf ("h2 = %lf\n", h2);
    if (dcmp(p.r*p.r - h2) < 0) return false;
    point tmp;
    tmp.x = p.o.x - l.s.x;
    tmp.y = p.o.y - l.s.y;
    tmp.z = p.o.z - l.s.z;
    if (dcmp(dot(tmp, l.dir)) <= 0) return false;
    k = sqrt(dis2(p.o, l.s) - h2) - sqrt(p.r*p.r - h2);
    double k1 = k / sqrt(dis2(l.dir, centre));
    t.x = l.s.x + k1 * l.dir.x;
    t.y = l.s.y + k1 * l.dir.y;
    t.z = l.s.z + k1 * l.dir.z;
    return true;
}
/*计算新射线的起点和方向 */
void newRay (ray &l, ray l1, point inter)
{
    double k = - 2 * dot(l.dir, l1.dir);
    l.dir.x += l1.dir.x * k;
    l.dir.y += l1.dir.y * k;
    l.dir.z += l1.dir.z * k;
    l.s = inter;
}
/* 返回的是最先相交的球的编号,均不相交,返回-1 */
int update ()
{
    int sign = -1, i;
    double k = 1e100, tmp;
    point inter, t;
    for (i = 1; i <= n; i++){ //找到最先相交的球
        if (!find (cir[i], l, tmp, t)) continue;
        if (dcmp (tmp - k) < 0) k = tmp, inter = t, sign = i;
    }
    //ray 变向

```

```

    if (sign == -1) return sign;
    ray l1;
    l1.s = cir[sign].o;
    l1.dir.x = (inter.x - l1.s.x) / cir[sign].r;
    l1.dir.y = (inter.y - l1.s.y) / cir[sign].r;
    l1.dir.z = (inter.z - l1.s.z) / cir[sign].r;
    newRay (l, l1, inter);
    return sign;
}
int main ()
{
    // freopen ("in", "r", stdin);
    int i;
    scanf ("%d", &n);
    for (i = 1; i <= n; i++) //输入空间的球位置
        scanf ("%lf%lf%lf%lf", &cir[i].o.x, &cir[i].o.y, &cir[i].o.z, &cir[i].r);
    scanf ("%lf%lf%lf%lf%lf%lf", &l.s.x, &l.s.y, &l.s.z, &l.dir.x, &l.dir.y, &l.dir.z);
    for (i = 0; i <= 10; i++){ //最多输出十次相交的球的编号
        int sign = update ();
        if (sign == -1) break;
        if (i == 0) printf ("%d", sign);
        else if (i < 10) printf (" %d", sign);
        else printf (" etc.");
    }
    puts ("");
}

```

三角形

```

#include <math.h>
struct point{double x,y;};
struct line{point a,b;};
double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
point intersection(line u,line v){
    point ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
        /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}
//外心
point circumcenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b.x=u.a.x-a.y+b.y;
    u.b.y=u.a.y+a.x-b.x;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
}

```

```

        v.b.x=v.a.x-a.y+c.y;
        v.b.y=v.a.y+a.x-c.x;
        return intersection(u,v);
    }
    //内心
    point incenter(point a,point b,point c){
        line u,v;
        double m,n;
        u.a=a;
        m=atan2(b.y-a.y,b.x-a.x);
        n=atan2(c.y-a.y,c.x-a.x);
        u.b.x=u.a.x+cos((m+n)/2);
        u.b.y=u.a.y+sin((m+n)/2);
        v.a=b;
        m=atan2(a.y-b.y,a.x-b.x);
        n=atan2(c.y-b.y,c.x-b.x);
        v.b.x=v.a.x+cos((m+n)/2);
        v.b.y=v.a.y+sin((m+n)/2);
        return intersection(u,v);
    }
    //垂心
    point perpercenter(point a,point b,point c){
        line u,v;
        u.a=c;
        u.b.x=u.a.x-a.y+b.y;
        u.b.y=u.a.y+a.x-b.x;
        v.a=b;
        v.b.x=v.a.x-a.y+c.y;
        v.b.y=v.a.y+a.x-c.x;
        return intersection(u,v);
    }
    //重心
    //到三角形三顶点距离的平方和最小的点
    //三角形内到三边距离之积最大的点
    point barycenter(point a,point b,point c){
        line u,v;
        u.a.x=(a.x+b.x)/2;
        u.a.y=(a.y+b.y)/2;
        u.b=c;
        v.a.x=(a.x+c.x)/2;
        v.a.y=(a.y+c.y)/2;
        v.b=b;
        return intersection(u,v);
    }
    //费马点
    //到三角形三顶点距离之和最小的点
    point fermentpoint(point a,point b,point c){
        point u,v;
        double step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
        int i,j,k;
        u.x=(a.x+b.x+c.x)/3;
        u.y=(a.y+b.y+c.y)/3;
        while (step>1e-10)
            for (k=0;k<10;step/=2,k++)
                for (i=-1;i<=1;i++)
                    for (j=-1;j<=1;j++){
                        v.x=u.x+step*i;

```

```

        v.y=u.y+step*j;
        if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))
            u=v;
    }

    return u;
}

//求曲率半径 三角形内最大可围成面积
#include<iostream>
#include<cmath>
using namespace std;
const double pi=3.14159265358979;
int main()
{
    double a,b,c,d,p,s,r,ans,R,x,l; int T=0;
    while(cin>>a>>b>>c>>d&&a+b+c+d)
    {
        T++;
        l=a+b+c;
        p=l/2;
        s=sqrt(p*(p-a)*(p-b)*(p-c));
        R= s /p;
        if (d >= l)  ans = s;
        else if(2*pi*R>=d) ans=d*d/(4*pi);
        else
        {
            r = (l-d)/((1/R)-(2*pi));
            x = r*r*s/(R*R);
            ans = s - x + pi * r * r;
        }
        printf("Case %d: %.21f\n",T,ans);
    }
    return 0;
}

```

凸包

```

#include<stdio.h>
#include<math.h>
#include<string.h>
#include<algorithm>
using namespace std;
struct node
{
    int x,y;
} a[105],p[105];
int top,n;
double cross(node p0,node p1,node p2)//计算叉乘, 注意p0,p1,p2的位置, 这个决定了方向
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}
double dis(node a,node b)//计算距离, 这个用在了当两个点在一条直线上

```

```

{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
bool cmp(node p1,node p2)//极角排序
{
    double z=cross(a[0],p1,p2);
    if(z>0||(z==0&&dis(a[0],p1)<dis(a[0],p2)))
        return 1;
    return 0;
}
void Graham()
{
    int k=0;
    for(int i=0; i<n; i++)
        if(a[i].y<a[k].y||(a[i].y==a[k].y&&a[i].x<a[k].x))
            k=i;
    swap(a[0],a[k]);//找p[0]
    sort(a+1,a+n,cmp);
    top=1;
    p[0]=a[0];
    p[1]=a[1];
    for(int i=2; i<n; i++)//控制进栈出栈
    {
        while(cross(p[top-1],p[top],a[i])<0&&top)
            top--;
        top++;
        p[top]=a[i];
    }
}
int main()
{
    int m;
    scanf("%d",&m);
    while(m--)
    {
        scanf("%d",&n);
        for(int i=0; i<n; i++)
        {
            scanf("%d%d",&a[i].x,&a[i].y);//输入所有点
        }
        Graham();
        for(int i=0; i<=top; i++)
        {
            printf("%d %d\n",p[i].x,p[i].y);//输出凸包点
        }
    }
    return 0;
}

```

网格

```
#define abs(x) ((x)>0?(x):- (x))
```

```

struct point{int x,y;};
int gcd(int a,int b){return b?gcd(b,a%b):a;}
//多边形上的网格点个数
int grid_onedge(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++){
        ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
    }
    return ret;
}
//多边形内的网格点个数
int grid_inside(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++){
        ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
    }
    return (abs(ret)-grid_onedge(n,p))/2+1;
}

```

半平面交

```

//对于给出点的顺时针和逆时针顺序不同,只需要加个 reverse 函数将点的顺序颠倒
int sgn(double x)
{ //符号函数
    if(fabs(x) < eps) return 0;
    if(x < 0) return -1;
    else return 1;
}

struct Point
{ //点
    double x,y;
    Point(){ }
    Point(double _x,double _y)
    {
        x = _x; y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    double operator ^(const Point &b)const
    { //叉积
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    { //点积
        return x*b.x + y*b.y;
    }
};

struct Line
{ //向量
    Point s,e; //两点
    double k; //斜率
    Line(){ }
}

```



```

Line(Point _s,Point _e)
{ //构造
    s = _s; e = _e;
    k = atan2(e.y - s.y,e.x - s.x);
}
Point operator &(const Line &b)const
{ //求两直线交点
    Point res = s;
    double t = ((s - b.s)^(b.s - b.e))/((s - e)^(b.s - b.e));
    res.x += (e.x - s.x)*t;
    res.y += (e.y - s.y)*t;
    return res;
}
};
Line Q[MAXN];
Point p[MAXN]; //记录最初给的点集
Line line[MAXN]; //由最初的点集生成直线的集合
Point pp[MAXN]; //记录半平面交的结果的点集
//半平面交, 直线的左边代表有效区域
bool HPIcmp(Line a,Line b)
{ //直线排序函数
    if(fabs(a.k - b.k) > eps)return a.k < b.k; //斜率排序
    //斜率相同我也不知道怎么办
    return ((a.s - b.s)^(b.e - b.s)) < 0;
}
void HPI(Line line[], int n, Point res[], int &resn)
{ //line是半平面交的直线的集合 n是直线的条数 res是结果
  //的点集 resn是点集里面点的个数
    int tot = n;
    sort(line,line+n,HPIcmp);
    tot = 1;
    for(int i = 1;i < n;i++)
        if(fabs(line[i].k - line[i-1].k) > eps) //去掉斜率重复的
            line[tot++] = line[i];
    int head = 0, tail = 1;
    Q[0] = line[0];
    Q[1] = line[1];
    resn = 0;
    for(int i = 2; i < tot; i++)
    {
        if(fabs((Q[tail].e-Q[tail].s)^(Q[tail-1].e-Q[tail-1].s)) < eps ||
            fabs((Q[head].e-Q[head].s)^(Q[head+1].e-Q[head+1].s)) < eps)
            return;
        while(head < tail && (((Q[tail]&Q[tail-1]) -
            line[i].s)^(line[i].e-line[i].s)) > eps)
            tail--;
        while(head < tail && (((Q[head]&Q[head+1]) -
            line[i].s)^(line[i].e-line[i].s)) > eps)
            head++;
        Q[++tail] = line[i];
    }
    while(head < tail && (((Q[tail]&Q[tail-1]) -
    Q[head].s)^(Q[head].e-Q[head].s)) > eps)
        tail--;
    while(head < tail && (((Q[head]&Q[head+1]) -
    Q[tail].s)^(Q[tail].e-Q[tail].e)) > eps)
        head++;
}

```

```

        if(tail <= head + 1) return;
        for(int i = head; i < tail; i++)
            res[resn++] = Q[i]&Q[i+1];
        if(head < tail - 1)
            res[resn++] = Q[head]&Q[tail];
    }
    double dist(Point a,Point b)
    { //两点间距离
        return sqrt((a-b)*(a-b));
    }
    void change(Point a,Point b,Point &c,Point &d,double p)
    { //将线段ab往左移动距离p,修改得到线段cd
        double len=dist(a,b);
        /*三角形相似推出下面公式*/
        double dx=(a.y-b.y)*p/len;
        double dy=(b.x-a.x)*p/len;
        c.x=a.x+dx; c.y=a.y+dy;
        d.x=b.x+dx; d.y=b.y+dy;
    }
    double BSearch()
    { //二分搜索
        double l=0,r=100000;
        double ans=0;
        while(r-l>=eps)
        {
            double mid=(l+r)/2;
            for(int i=0;i < n;i++)
            {
                Point t1,t2;
                change(p[i],p[(i+1)%n],t1,t2,mid);
                line[i]=Line(t1,t2);
            }
            int resn;
            HPI(line,n,pp,resn);
            //等于0说明移多了
            if(resn==0) r=mid-eps;
            else l=mid+eps;
        }
        return l;
    }
}

```

圆与多边形交

```

const double eps = 1e-8;           //浮点数精度控制
struct point                       //点或者向量结构
{
    double x,y;
    point(double _x=0.0,double _y=0.0)
        : x(_x),y(_y) {}
    point operator - (const point & v)
    {
        return point(x-v.x,y-v.y);
    }
}

```

```

    }
    double sgrx()                //向量的模
    {
        return sqrt(x*x+y*y);
    }
};
double xmult(point & p1,point & p2,point & p0)    //叉乘
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}
double distancex(point & p1,point & p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
point intersection(point u1,point u2,point v1,point v2)    //两直线交点
{
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}
void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2){
    point p=c;
    double t;
    p.x+=l1.y-l2.y;
    p.y+=l2.x-l1.x;
    p=intersection(p,c,l1,l2);
    t=sqrt(r*r-distancex(p,c)*distancex(p,c))/distancex(l1,l2);
    p1.x=p.x+(l2.x-l1.x)*t;
    p1.y=p.y+(l2.y-l1.y)*t;
    p2.x=p.x-(l2.x-l1.x)*t;
    p2.y=p.y-(l2.y-l1.y)*t;
}
point ptoseg(point p,point l1,point l2)    //点到线段的最近距离
{
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
        return distancex(p,l1)<distancex(p,l2)?l1:l2;
    return intersection(p,t,l1,l2);
}
double distp(point & a,point & b)
{
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}
double Direct_Triangle_Circle_Area(point a,point b,point o,double r)
{
    double sign=1.0;
    a=a-o;
    b=b-o;
    o=point(0.0,0.0);
    if(fabs(xmult(a,b,o))<eps) return 0.0;
    if(distp(a,o)>distp(b,o))
    {
        swap(a,b);
    }
}

```

```

        sign=-1.0;
    }
    if(distp(a,o)<r*r+eps)
    {
        if(distp(b,o)<r*r+eps) return xmult(a,b,o)/2.0*sign;
        point p1,p2;
        intersection_line_circle(o,r,a,b,p1,p2);
        if(distancex(p1,b)>distancex(p2,b)) swap(p1,p2);
        double ret1=fabs(xmult(a,p1,o));
        double ret2=acos( p1*b/p1.sqr()/b.sqr() )*r*r;
        double ret=(ret1+ret2)/2.0;
        if(xmult(a,b,o)<eps && sign>0.0 || xmult(a,b,o)>eps && sign<0.0) ret=-ret;
        return ret;
    }
    point ins=ptoseg(o,a,b);
    if(distp(o,ins)>r*r-eps)
    {
        double ret=acos( a*b/a.sqr()/b.sqr() )*r*r/2.0;
        if(xmult(a,b,o)<eps && sign>0.0 || xmult(a,b,o)>eps && sign<0.0) ret=-ret;
        return ret;
    }
    point p1,p2;
    intersection_line_circle(o,r,a,b,p1,p2);
    double cm=r/(distancex(o,a)-r);
    point m=point( (o.x+cm*a.x)/(1+cm) , (o.y+cm*a.y)/(1+cm) );
    double cn=r/(distancex(o,b)-r);
    point n=point( (o.x+cn*b.x)/(1+cn) , (o.y+cn*b.y)/(1+cn) );
    double ret1 = acos( m*n/m.sqr()/n.sqr() )*r*r;
    double ret2 = acos( p1*p2/p1.sqr()/p2.sqr() )*r*r-fabs(xmult(p1,p2,o));
    double ret=(ret1-ret2)/2.0;
    if(xmult(a,b,o)<eps && sign>0.0 || xmult(a,b,o)>eps && sign<0.0) ret=-ret;
    return ret;
}

```