GZHU I_WANT_TO_EAT_MCDONALD'S

Snippet

C++

```cpp
#define pb push_back
#define sz(s) ((int)s.size())
#define all(vec) vec.begin(), vec.end()

typedef long long ll;
typedef vector<ll> VL;
typedef vector<int> VI;
typedef pair<int, int> pii;

#ifdef local
#define debug(x...) do { cout << "[ "#x" ] -> "; err(x); } while (0)
template <class T>
inline void _E(T x) { cout << x; }
template <class L, class R>
inline void _E(pair<L, R> arg) {
  cout << "("; _E(arg.first), _E(','), _E(' '), _E(arg.second); cout << ")";
}
template <template <class...> class T, class t>
inline void _E(T<t> arr) {
  cout << "[ ";
  for (auto it = begin(arr), en = end(arr); it != en; it++) {
    if (it != begin(arr)) cout << ", "; _E(*it);
  }
  cout << " ]";
}
inline void _E(string s) { cout << "\"" + s + "\""; }

inline void err() { cout << std::endl; }
template <class T, class... U>
inline void err(T arg, U... args) {
  _E(arg); if (sizeof...(args)) cout << ", "; err(args...);
}
#else
#define debug(...) do {} while (0)
#endif
```

Java

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

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();
    int taskNum = 1;
    // int taskNum = in.nextInt();
    solver.solve(taskNum, in, out);
    out.close();
  }

  public static class Task {
    void solve(int t, InputReader in, PrintWriter out) {

    }
  }

  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 BigInteger nextBigInteger() {
      return new BigInteger(next());
    }
  }
}
```

unordered_map

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().\
                                         time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

unordered_map<long long, int, custom_hash> safe_map;
```

io buffer

```
namespace io {
  const int SZ = (1 << 22) + 1;
  char buf[SZ], *ptr = NULL, *bnd = NULL;
  #define GC() ((ptr == bnd) ? (ptr = buf, bnd = buf + fread(buf, 1, SZ, stdin),
(ptr == bnd) ? EOF : (*(ptr++))) : (*(ptr++)))
  #define STATE(c) { if (c == '-') sgn = -1; else if (c == EOF) return false; }
  inline bool skip(const char& c) { return c < '0' || c > '9'; }
  template <class V>
  inline bool Read(V &v) {
    register char c, sgn = 1;
    while (skip(c = GC())) STATE(c);
    for (v = c - '0'; !skip(c = GC()); v = v * 10 + c - '0');
    return (v *= sgn), true;
  }
  char oBuf[SZ], *oCur = oBuf, *oBnd = oBuf + SZ, oStk[21], top = 0;
  inline void flush() { if (oCur - oBuf) fwrite(oBuf, 1, oCur - oBuf, stdout),
oCur = oBuf; }
  inline void pc(char c) { *(oCur++) = c; if (oCur == oBnd) flush(); }
  template <class V>
  inline void Print(V v) {
    if (!v) return pc('0');
    if (v < 0) v = -v, pc('-');
    while (v) oStk[top++] = v % 10, v /= 10;
    while (top) pc(oStk[--top] + '0');
  }
  template <class V>
  inline void Println(const V& v) { Print(v), pc('\n'); }
  struct flusher { ~flusher() { flush(); } } __flusher__;
}
using io::Read;
using io::Println;
```

DataStructure

区间增减树状数组

```
struct Interval {
  int N, base[2][maxn];
  void setN(int n) { N = n; }
  void init() { memset(base, 0, sizeof base); }
  void add(int at, int v) {
    if (!at) return;
    for (int i = at; i <= N; i += i & -i) {
      base[0][i] += v, base[1][i] -= v * at;
    }
  }
  void add(int l, int r, int v) {
    add(l, v), add(r + 1, -v);
  }
  int getSum(int at) {
    int sum = 0, mul = at + 1;
    for (int i = at; i; i -= i & -i) {
      sum += mul * base[0][i] + base[1][i];
    }
    return sum;
  }
  int query(int l, int r) {
    return getSum(r) - getSum(l - 1);
  }
};
```

无旋Treap

```
struct Treap {
#define ls(x) T[x].son[0]
#define rs(x) T[x].son[1]

  struct Node {
    int son[2], size, v, key, rev;
  } T[maxn];
  int tot, root;

  Treap() { tot = root = 0; }

  inline void init() { tot = root = 0; }

  inline void pushup(int i) {
    T[i].size = T[ls(i)].size + T[rs(i)].size + 1;
  }
  inline void pushdown(int i) {
    if (T[i].rev) {
      swap(ls(i), rs(i));
      T[ls(i)].rev ^= 1, T[rs(i)].rev ^= 1;
      T[i].rev = 0;
    }
  }
  void split(int rt, int &x, int &y, int v) {
    if (!rt) return (void)(x = y = 0);
    pushdown(rt);
    if (T[rt].v <= v) {
      x = rt, split(rs(rt), rs(x), y, v);
    } else {
      y = rt, split(ls(rt), x, ls(y), v);
    }
    pushup(rt);
  }
  void merge(int &rt, int x, int y) {
    if (!x || !y) {
      rt = x + y;
      return;
    }
    if (T[x].key < T[y].key) {
      pushdown(x), rt = x, merge(rs(rt), rs(x), y);
    } else {
      pushdown(y), rt = y, merge(ls(rt), x, ls(y));
    }
    pushup(rt);
  }
  inline void insert(int &rt, int v) {
    int x = 0, y = 0, z = ++tot;
    T[z].v = v, T[z].key = rand(), T[z].size = 1, T[z].rev = 0;
    split(rt, x, y, v), merge(x, x, z), merge(rt, x, y);
  }
  inline void erase(int &rt, int v) {
    int x = 0, y = 0, z = 0;
    split(rt, x, y, v), split(x, x, z, v - 1);
    merge(z, ls(z), rs(z)), merge(x, x, z), merge(rt, x, y);
  }
  inline int findkth(int rt, int k) {
    if (k == 0) return -inf;
    pushdown(rt);
    while (T[ls(rt)].size + 1 != k) {
      if (T[ls(rt)].size >= k) rt = ls(rt);
      else k -= (T[ls(rt)].size + 1), rt = rs(rt);
      pushdown(rt);
    }
    return T[rt].v;
  }
  inline int getrank(int &rt, int v) {
    int x = 0, y = 0, res;
    split(rt, x, y, v - 1), res = T[x].size + 1;
    return merge(rt, x, y), res;
  }
  inline int getpre(int &rt, int v) {
    int x = 0, y = 0, res;
    split(rt, x, y, v - 1), res = findkth(x, T[x].size);
    return merge(rt, x, y), res;
  }
  inline int getsuf(int &rt, int v) {
    int x = 0, y = 0, res;
    split(rt, x, y, v), res = findkth(y, 1);
    return merge(rt, x, y), res;
  }

  inline void insert(int v) { insert(root, v); }
  inline void erase(int v) { erase(root, v); }
  inline int findkth(int k) { return findkth(root, k); }
  inline int getrank(int v) { return getrank(root, v); }
  inline int getpre(int v) { return getpre(root, v); }
  inline int getsuf(int v) { return getsuf(root, v); }
} treap;
```

ST表

```
struct ST {
  vector<vector<int>> table;
  ST(vector<int> a = {}) {
    int n = a.size();
    table.resize(n, vector<int>(32 - __builtin_clz(n)));
    for (int i = 0; i < n; i++) {
      table[i][0] = a[i];
    }
    for (int j = 1; (1 << j) - 1 < n; j++) {
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        int x = table[i][j - 1], y = table[i + (1 << (j - 1))][j - 1];
        table[i][j] = min(x, y);
      }
    }
  }
  inline int getMin(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return min(table[l][k], table[r - (1 << k) + 1][k]);
  }
};
```

并查集(带权)

```
template <int NV> class Dsu {
  int anc[NV], weight[NV];
  void init(int n = NV) {
    iota(anc.begin(), next(anc.begin(), n), 0);
    fill(anc.begin(), next(anc.begin(), n), 0);
  }
  int find(int x) {
    if (x == anc[x]) return x;
    int fa = anc[x];
    anc[x] = find(anc[x]);
    weight[x] += weight[fa];
    return anc[x];
  }
  bool unite(int u, int v, int w = 0) {
    int a = find(u), b = find(v);
    if (a == b) return false;
    anc[b] = a;
    weight[b] = weight[u] + w - weight[v];
    return true;
  }
};
```

String

kmp

```
template <template<class...> class T, class t>
VI getfail(const T<t>& s) {
  int n = sz(s);
  VI fail(n + 1);
  for (int i = 0, j = fail[0] = -1; i < n; i++, j++) {
    while (~j && s[j] != s[i]) j = fail[j];
    fail[i + 1] = j + 1;
  }
  return fail;
}

// candidate
VI getfail(const string& s) {
  int n = sz(s);
  VI fail(n + 1);
  for (int i = 0, j = fail[0] = -1; i < n; i++, j++) {
    while (~j && s[i] != s[j]) j = fail[j];
    fail[i + 1] = (s[i + 1] == s[j + 1]) ? fail[j + 1] : (j + 1);
  }
  return fail;
}

template <template<class...> class T, class t>
int match(const T<t> &s, const T<t> &par, const VI &fail) {
  int n = sz(s), m = sz(par);
  for (int i = 0, j = 0; i < n; ) {
    while (~j && par[j] != s[i]) j = fail[j];
    ++i, ++j;
    if (j >= m) return i - m + 1;
  }
  return -1;
}
```

Z-function

```
VI Zfunc(string s) {
  int n = sz(s);
  VI z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

Suffix Array

```
template <typename T>
VI build_sa(int n, const T &s, int charset) {
  VI a(n);
  if (n == 0) {
    return a;
  }
  if (charset != -1) {
    VI aux(charset, 0);
    for (int i = 0; i < n; i++) {
      aux[ s[i] ]++;
    }
    int sum = 0;
    for (int i = 0; i < charset; i++) {
      int add = aux[i];
      aux[i] = sum;
      sum += add;
    }
    for (int i = 0; i < n; i++) {
      a[ aux[ s[i] ]++ ] = i;
    }
  } else {
    iota(a.begin(), a.end(), 0);
    sort(a.begin(), a.end(), [&s](int i, int j) { return s[i] < s[j]; });
  }
  VI sorted_by_second(n), ptr_group(n);
  VI new_group(n), group(n);
  group[ a[0] ] = 0;
  for (int i = 1; i < n; i++) {
    group[ a[i] ] = group[ a[i - 1] ] + ( !(s[ a[i] ] == s[ a[i - 1] ]) );
  }
  int cnt = group[a[n - 1]] + 1;
  int step = 1;
  while (cnt < n) {
    int at = 0;
    for (int i = n - step; i < n; i++) {
      sorted_by_second[at++] = i;
    }
    for (int i = 0; i < n; i++) {
      if (a[i] - step >= 0) {
        sorted_by_second[at++] = a[i] - step;
      }
    }
    for (int i = n - 1; i >= 0; i--) {
      ptr_group[ group[ a[i] ] ] = i;
    }
    for (int i = 0; i < n; i++) {
      int x = sorted_by_second[i];
      a[ ptr_group[ group[x] ]++ ] = x;
    }
    new_group[a[0]] = 0;
    for (int i = 1; i < n; i++) {
      if (group[ a[i] ] != group[ a[i - 1] ]) {
        new_group[ a[i] ] = new_group[ a[i - 1] ] + 1;
      } else {
        int pre = ( (a[i - 1] + step >= n) ? -1 : group[ a[i - 1] + step ] );
        int cur = ( (a[i] + step >= n) ? -1 : group[ a[i] + step ] );
        new_group[ a[i] ] = new_group[ a[i - 1] ] + (pre != cur);
      }
    }
    swap(group, new_group);
    cnt = group[ a[n - 1] ] + 1;
    step <<= 1;
  }
  return a;
}
```

one more

```
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 Automa

```
struct SAM {
  int last, tot, sz[maxn << 1], len[maxn << 1], fa[maxn << 1];
  int ch[maxn << 1][30];

  SAM() {
    tot = 0, last = newNode(0), len[0] = -1;
    memset(sz, 0, sizeof sz);
  }
  inline int newNode(int v) {
    len[++tot] = v, fa[tot] = 0;
    memset(ch[tot], 0, sizeof ch[tot]);
    return tot;
  }
  void append(int c) {
    int p = last, u = newNode(len[last] + 1);
    for (; p && !ch[p][c]; p = fa[p]) {
      ch[p][c] = u;
    }
    if (p == 0) {
      fa[u] = 1;
    } else {
      int q = ch[p][c];
      if (len[q] == len[p] + 1) {
        fa[u] = q;
      } else {
        int nq = newNode(len[p] + 1);
        memcpy(ch[nq], ch[q], sizeof ch[q]);
        fa[nq] = fa[q], fa[u] = fa[q] = nq;
        for (; p && (ch[p][c] == q); p = fa[p]) {
          ch[p][c] = nq;
        }
      }
    }
    last = u;
  }
  void match(char *s) {
    int pos = 1, length = 0;
    for (int i = 0, n = strlen(s); i < n; i++) {
      while (pos && !ch[pos][s[i] - 'a']) {
        pos = fa[pos], length = len[pos];
      }
      if (pos) {
        ++length, pos = ch[pos][s[i] - 'a'];
        // update ans
      } else {
        pos = 1, length = 0;
      }
    }
  }
} sam;
```

最小表示法

对于一个字符串S，求S的循环的同构字符串S’中字典序最小的一个。

字符串"abcd"的循环同构字符串有：["abcd", "bcda", "cdab", "dabc"]。

```
int minPresentation(string &s) {
  int n = s.length();
  int i = 0, j = 1, k = 0;
  while (k < n && i < n && j < n) {
    if (s[(i + k) % n] == s[(j + k) % n]) {
      ++k;
    } else {
      s[(i + k) % n] > s[(j + k) % n] ? (i += k + 1) : (j += k + 1);
      i += (i == j);
      k = 0;
    }
  }
  return min(i, j);
}
```

Manacher

```
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;
}
```

AC Automa

```
namespace acam {
  struct Node {
    int son[26], fail;
    void init() {
      fail = 0;
      memset(son, 0, sizeof son);
    }
  } T[N];
  int tot;

#define Son(i, x) T[i].son[x]
#define trans(c) (c - 'A')

  void init() {
    tot = 0, T[0].init();
  }
  void insert(char *s, int index) {
    int cur = 0;
    for (int i = 0; s[i]; i++) {
      int c = trans(s[i]);
      if (!Son(cur, c)) {
        Son(cur, c) = ++tot;
        T[tot].init();
      }
      cur = Son(cur, c);
    }
  }
  void build() {
    queue<int> Q;
    for (int i = 0; i < 26; i++) {
      if (Son(0, i)) Q.push(Son(0, i));
    }
    while (!Q.empty()) {
      int u = Q.front(); Q.pop();
      for (int i = 0; i < 26; i++) {
        if (Son(u, i)) {
          T[Son(u, i)].fail = T[T[u].fail].son[i];
          Q.push(Son(u, i));
        } else {
          T[u].son[i] = T[T[u].fail].son[i];
        }
      }
    }
  }
  int query(char *t) {
    int ans = 0, cur = 0;
    for (int i = 0; t[i]; i++) {
      int c = trans(t[i]);
      cur = Son(cur, c);
      for (int j = cur; j; j = T[j].fail) {
        // upd ans;
      }
    }
    return ans;
  }
}
```

PAM

```
struct PAM {
  struct Node {
    int son[27], fail, len, dep;
    void init(int l, int f, int d = 0) {
      fail = f, len = l, dep = d;
      memset(son, 0, sizeof son);
    }
  } T[N + 5];
  int tot, prefix, suffix, l, r;
  int s[N * 2 + 5];

  void init() {
    ans = 0;
    tot = 1, l = N + 1, r = N, prefix = suffix = 0;
    T[0].init(0, 1), T[1].init(-1, 0);
    memset(s, 0, sizeof s);
  }
  void encode(int &c) {
    // keep c > 0
    c = c - 'a' + 1;
  }
  int pre_fail(int cur) {
    while (s[l + T[cur].len + 1] != s[l]) {
      cur = T[cur].fail;
    }
    return cur;
  }
  int suf_fail(int cur) {
    while (s[r - T[cur].len - 1] != s[r]) {
      cur = T[cur].fail;
    }
    return cur;
  }
  void push_front(int c) {
    encode(c), s[--l] = c;
    prefix = pre_fail(prefix);
    if (!T[prefix].son[c]) {
      int f = pre_fail(T[prefix].fail);
      T[++tot].init(T[prefix].len + 2, T[f].son[c], T[T[f].son[c]].dep + 1);
      T[prefix].son[c] = tot;
    }
    prefix = T[prefix].son[c];
    if (T[prefix].len == r - l + 1) {
      suffix = prefix;
    }
  }
  void push_back(int c) {
    encode(c), s[++r] = c;
    suffix = suf_fail(suffix);
    if (!T[suffix].son[c]) {
      int f = suf_fail(T[suffix].fail);
      T[++tot].init(T[suffix].len + 2, T[f].son[c], T[T[f].son[c]].dep + 1);
      T[suffix].son[c] = tot;
    }
    suffix = T[suffix].son[c];
    if (T[suffix].len == r - l + 1) {
      prefix = suffix;
    }
  }
} pam;
```

Graph

2-sat

```
struct twoSat {
  struct edge {
    int v, next;
    edge(int a = 0, int b = 0) : v(a), next(b) {}
  }G[maxm];
  int tot, head[maxn], mark[maxn], sz, stk[maxn];
  void init() {
    tot = 0;
    memset(mark, 0, sizeof mark);
    memset(head, -1, sizeof head);
  }
  // for every case u, (status[u] xor status[u ^ 1]) == true.
  //
  // addcase: if status[u] == true then status[v] == true,
  // but if status[u] == false then status[v] can be true or false.
  //
  void addcase(int u, int v) {
    G[tot] = edge(v, head[u]); head[u] = tot++;
  }
  int dfs(int u) {
    if (mark[u ^ 1]) return 0;
    if (mark[u]) return 1;
    stk[sz++] = u, mark[u] = 1;
    for (int i = head[u]; ~i; i = G[i].next) {
      if (!dfs(G[i].v)) return 0;
    }
    return 1;
  }
  int solve(int n) {
    for (int i = 0; i < n; i += 2) {
      if (!mark[i] && !mark[i ^ 1]) {
        sz = 0;
        if (!dfs(i)) {
          while (sz > 0) mark[stk[--sz]] = 0;
          if (!dfs(i ^ 1)) return 0;
        }
      }
    }
    return 1;
  }
}sat;
```

强连通

Tarjan

```
struct Scc {
  vector<int> G[maxn];
  int N, tag, tot, dfn[maxn], low[maxn], sccno[maxn];
  stack<int> S;

  void init(int n) {
    N = n, tag = tot = 0;
    for (int i = 1; i <= n; i++) {
      dfn[i] = low[i] = sccno[i] = 0;
      G[i].clear();
    }
  }
  void addedge(int u, int v) {
    G[u].push_back(v);
  }
  void dfs(int u) {
    dfn[u] = low[u] = ++tag;
    S.push(u);
    for (auto& v : G[u]) {
      if (!dfn[v]) {
        dfs(v);
        low[u] = min(low[u], low[v]);
      } else if (!sccno[v]) {
        low[u] = min(low[u], dfn[v]);
      }
    }
    if (low[u] == dfn[u]) {
      ++tot;
      while (true) {
        int x = S.top(); S.pop();
        sccno[x] = tot;
        if (x == u) break;
      }
    }
  }
  void solve() {
```

```cpp
    for (int i = 1; i <= N; i++) {
      if (!dfn[i]) dfs(i);
    }
  }
} scc;
```

kosaraju

```cpp
struct kosaraju {
  int N, tot, scc[maxn], vis[maxn];
  vector<int> G[maxn], R[maxn], acc;
  void init(int n) {
    N = n;
    tot = 0, acc.clear();
    for (int i = 1; i <= N; i++) {
      G[i].clear(), R[i].clear();
      vis[i] = 0, scc[i] = 0;
    }
  }
  void DFS1(int u) {
    vis[u] = 1;
    for (auto& v : G[u]) {
      if (!vis[v]) DFS1(v);
    }
    acc.push_back(u);
  }
  void DFS2(int u, int p) {
    scc[u] = p;
    for (auto& v : R[u]) {
      if (!scc[v]) DFS2(v, p);
    }
  }
  void solve() {
    for (int i = 1; i <= N; i++) {
      if (!vis[i]) DFS1(i);
    }
    reverse(acc.begin(), acc.end());
    for (auto& u : acc) {
      if (!scc[u]) DFS2(u, ++tot);
    }
  }
};
```

双连通

点双

```cpp
struct bcc {
  struct edge { int u, v; };
  vector<int> G[N], cont[N];
```

```
int Nx, tag, tot, dfn[N], bccno[N];
bool iscut[N];
stack<edge> S;

void init(int n) {
  Nx = n, tag = tot = 0;
  for (int i = 1; i <= Nx; i++) {
    G[i].clear();
    dfn[i] = bccno[i] = 0;
    iscut[i] = false;
  }
  while (!S.empty()) S.pop();
}
void addedge(int u, int v) {
  G[u].push_back(v), G[v].push_back(u);
}
int dfs(int u, int f) {
  int lowu = dfn[u] = ++tag;
  int child = 0;
  for (auto& v : G[u]) {
    if (!dfn[v]) {
      ++child, S.push({ u, v });
      int lowv = dfs(v, u);
      lowu = min(lowu, lowv);
      if (lowv >= dfn[u]) {
        iscut[u] = true;
        cont[++tot].clear();
        while (true) {
          edge e = S.top(); S.pop();
          if (bccno[e.u] != tot) {
            cont[tot].push_back(e.u);
            bccno[e.u] = tot;
          }
          if (bccno[e.v] != tot) {
            cont[tot].push_back(e.v);
            bccno[e.v] = tot;
          }
          if (e.u == u && e.v == v) {
            break;
          }
        }
      }
    } else if (dfn[v] < dfn[u] && v != f) {
      S.push({ u, v });
      lowu = min(lowu, dfn[v]);
    }
  }
  if (f < 0 && child == 1) {
    iscut[u] = false;
  }
  return lowu;
  }
  void solve() {
    for (int i = 1; i <= Nx; i++) {
      if (!dfn[i]) dfs(i, -1);
    }
  }
} gao;
```

割顶/桥

```
struct edge {
  int v, next;
} G[M];
int tot, h[N], ord, dfn[N], low[N];
bool iscut[N], isbridge[M];

void init() {
  tot = ord = 0;
  memset(h, -1, sizeof h);
  memset(dfn, 0, sizeof dfn);
  memset(low, 0, sizeof low);
  memset(iscut, false, sizeof iscut);
  memset(isbridge, false, sizeof false);
}

void addedge(int u, int v) {
  G[tot] = { v, h[u] }, h[u] = tot++;
  G[tot] = { u, h[v] }, h[v] = tot++;
}

void dfs(int u, int f) {
  low[u] = dfn[u] = ++ord;
  int child = 0;
  for (int i = h[u]; ~i; i = G[i].next) {
    edge &e = G[i];
    if (!dfn[e.v]) {
      ++child, dfs(e.v, u);
      low[u] = min(low[u], low[e.v]);
      if (low[e.v] >= dfn[u]) {
        iscut[u] = true;
      }
      if (low[e.v] > dfn[u]) {
        isbridge[i] = isbridge[i ^ 1] = true;
      }
    } else if (dfn[e.v] < dfn[u] && e.v != f) {
      low[u] = min(low[u], dfn[e.v]);
    }
  }
  if (f == -1 && child == 1) {
    iscut[u] = false;
  }
}

void solve(int n) {
  for (int i = 1; i <= n; i++) {
    if (!dfn[i]) dfs(i, -1);
  }
}
```

边双(kuangbin)

```
struct edge {
  int v, next;
  bool cut;
} G[M];
int tot, h[N];
int ord, top, bcc_cnt, bridge, dfn[N], low[N], in[N], stk[N];
bool instk[N];

void init() {
  tot = 0;
  memset(h, -1, sizeof h);
}

void addedge(int u, int v) {
  G[tot] = { v, h[u], false }, h[u] = tot++;
  G[tot] = { u, h[v], false }, h[v] = tot++;
}

void dfs(int u, int f) {
  low[u] = dfn[u] = ++ord;
  stk[top++] = u, instk[u] = true;
  int f_cnt = 0;
  for (int i = h[u]; ~i; i = G[i].next) {
    int v = G[i].v;
    if (v == f && f_cnt == 0) { ++f_cnt; continue; }
    if (!dfn[v]) {
      dfs(v, u);
      if (low[u] > low[v]) low[u] = low[v];
      if (low[v] > dfn[u]) {
        ++bridge;
        G[i].cut = G[i ^ 1].cut = true;
      }
    } else if (instk[v] && low[u] > dfn[v]) {
      low[u] = dfn[v];
    }
  }
  if (low[u] == dfn[u]) {
    int v;
    ++bcc_cnt;
    do {
      v = stk[--top];
      instk[v] = false;
      in[v] = bcc_cnt;
    } while (v != u);
  }
}

void solve(int n) {
  for (int i = 1; i <= n; i++) {
    if (!dfn[i]) dfs(i, -1);
  }
}
```

欧拉路

无向

```
// undirected, 0-base
template <int NV> class Hierholzer {
public:
  vector<int> path;
  multiset<int> G[NV];

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

  void dfs(int cur) {
    while (!G[cur].empty()) {
      int tar = *G[cur].begin();
      G[cur].erase(G[cur].begin());
      G[tar].erase(G[tar].find(cur));
      dfs(tar);
    }
    path.push_back(cur);
  }

  bool get() {
    int src = -1, odd = 0, tot = 0;
    for (int i = 0; i < NV; i++) {
      tot += G[i].size();
      if (G[i].size() % 2 == 1) {
        odd++, src = (~src) ? src : i;
      }
    }
    if (odd != 0 && odd != 2) return false;
    dfs(odd ? src : 0);
    reverse(path.begin(), path.end());
    return (int)path.size() == tot / 2 + 1;
  }

  vector<int> get(int src) {
    dfs(src);
    reverse(path.begin(), path.end());
    return path;
  }
};
```

有向

```
// directed, 0-base.
template <int NV> class Hierholzer {
public:
  int deg[NV];
  vector<int> path;
  multiset<int> G[NV];

  void addedge(int u, int v) {
    G[u].insert(v), deg[u]++, deg[v]--;
  }

  void dfs(int cur) {
    while (!G[cur].empty()) {
      int tar = *G[cur].begin();
      G[cur].erase(G[cur].begin());
      dfs(tar);
    }
    path.push_back(cur);
  }

  bool get() {
    int src = -1, tot = 0, U = 0, D = 0, UZ = 0;
    for (int i = 0; i < NV; i++) {
      tot += G[i].size();
      if (deg[i] != 0) {
        U += (deg[i] == 1), D += (deg[i] == -1), UZ++;
        src = (~src) ? src : i;
      }
    }
    if (UZ != 0 && (UZ != 2 || U != 1 || D != 1)) return false;
    dfs(UZ ? src : 0);
    reverse(path.begin(), path.end());
    return (int)path.size() == tot + 1;
  }

  vector<int> get(int src) {
    dfs(src);
    reverse(path.begin(), path.end());
    return path;
  }
};
```

费用流

```
struct edge {
  int v, cost, flow, cap, next;
  edge() {}
  edge(int V, int Cost, int Flow, int Cap, int nxt) : \
      v(V), cost(Cost), flow(Flow), cap(Cap), next(nxt) {}
} G[maxm << 1];
int tot, head[maxn], cost[maxn], inq[maxn], pre[maxn];

void init() {
  tot = 0;
  memset(head, -1, sizeof head);
}

void addedge(int u, int v, int cap, int cost) {
  G[tot] = edge(v, cost, 0, cap, head[u]); head[u] = tot++;
  G[tot] = edge(u, -cost, cap, cap, head[v]); head[v] = tot++;
}

bool spfa(int src, int dst) {
  memset(inq, 0, sizeof inq);
  memset(pre, -1, sizeof pre);
  memset(cost, 0x3f, sizeof cost);
  queue<int> Q; Q.push(src), cost[src] = 0;
  while (!Q.empty()) {
    int u = Q.front(); Q.pop(), inq[u] = 0;
    for (int i = head[u]; ~i; i = G[i].next) {
      edge &e = G[i];
      if (e.flow < e.cap && chkmin(cost[e.v], cost[u] + e.cost)) {
        pre[e.v] = i;
        if (!inq[e.v]) Q.push(e.v), inq[e.v] = 1;
      }
    }
  }
  return cost[dst] < 0x3f3f3f3f;
}

pair<int, int> mcmf(int src, int dst) {
  int totCost = 0, totFlow = 0;
  while (spfa(src, dst)) {
    int maxFlow = 0x3f3f3f3f;
    for (int u = dst; u != src; u = G[pre[u] ^ 1].v) {
      edge &e = G[pre[u]]; // , &r = G[pre[u] ^ 1];
      maxFlow = min(maxFlow, e.cap - e.flow);
    }
    totCost += maxFlow * cost[dst], totFlow += maxFlow;
    for (int u = dst; u != src; u = G[pre[u] ^ 1].v) {
      edge &e = G[pre[u]], &r = G[pre[u] ^ 1];
      e.flow += maxFlow, r.flow -= maxFlow;
    }
  }
  return { totFlow, totCost };
}
```

二分图匹配

```
struct maxMatch {
  int link[maxn], vis[maxn];
  bool find(int u) {
    for (int i = head[u]; i != -1; i = G[i].next) {
      int v = G[i].v;
      if (!vis[v]) {
        vis[v] = 1;
        if (link[v] == -1 || find(link[v])) {
          link[v] = u;
          // link[u] = v;
          return true;
        }
      }
    }
    return false;
  }
  int getans(int n) {
    int ans = 0;
    memset(link, -1, sizeof link);
    for (int i = 1; i <= n; i++) {
      if (link[i] == -1) {
        memset(vis, 0, sizeof vis);
        if (find(i)) ++ans;
      }
    }
    return ans;
  }
};
```

树剖(lca为例)

```
int SZ[N], fa[N], son[N], top[N], dep[N];
int dfn, in[N], out[N];

void getsz(int u, int d, int f) {
  SZ[u] = 1, dep[u] = d, fa[u] = f;
  son[u] = 0;
  for (auto& v : G[u]) {
    if (v != f) {
      getsz(v, d + 1, u);
      SZ[u] += SZ[v];
      if (SZ[son[u]] < SZ[v]) son[u] = v;
    }
  }
}

void dfs(int u, int t) {
  in[u] = ++dfn, top[u] = t;
  if (son[u]) dfs(son[u], t);
  for (auto& v : G[u]) {
    if (v != fa[u] && v != son[u]) {
      dfs(v, v);
    }
  }
  out[u] = dfn;
}

int getlca(int u, int v) {
  for (; top[u] != top[v]; u = fa[top[u]]) {
    if (dep[top[u]] < dep[top[v]]) swap(u, v);
  }
  return dep[u] < dep[v] ? u : v;
}
```

KM

```
const int N = 505;
const int maxn = 505;
const int INF = 0x3f3f3f3f;

int nx, ny; // point num
int G[maxn][maxn]; // graph
int link[maxn], lx[maxn], ly[maxn], 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 (link[y] == -1 || dfs(link[y])) {
        link[y] = x;
        return true;
      }
    } else if (slack[y] > tmp) {
      slack[y] = tmp;
    }
  }
  return false;
}

int KM() {
  memset(link, -1, sizeof link);
  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];
      }
      if (d == INF) return -1;
      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 (~link[i]) res += G[link[i]][i];
  }
  return res;
}
```

isap

```
const int N = 1e2 + 5;
const int M = 2e4 + 5;
const int inf = 0x3f3f3f3f;

struct edge {
  int v, flow, cap, next;
} G[M];
int tot, n, src, dst, h[N], cur[N], gap[N], dep[N];
```

```
void init() {
  tot = 0;
  memset(h, -1, sizeof h);
}

void addedge(int u, int v, int w) {
  G[tot] = { v, 0, w, h[u] }, h[u] = tot++;
  G[tot] = { u, w, w, h[v] }, h[v] = tot++;
}

void bfs() {
  memset(gap, 0, sizeof gap);
  memset(dep, -1, sizeof dep);

  queue<int> Q; Q.push(dst);
  dep[dst] = 0, gap[0] = 1;
  while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int i = h[u]; ~i; i = G[i].next) {
      int v = G[i].v;
      if (~dep[v]) continue;
      Q.push(v), dep[v] = dep[u] + 1, gap[dep[v]]++;
    }
  }
}

int dfs(int u, int flow) {
  if (u == dst) return flow;
  int used = 0;
  for (int &i = cur[u]; ~i; i = G[i].next) {
    edge &e = G[i];
    if (e.flow < e.cap && dep[e.v] + 1 == dep[u]) {
      int tmp = dfs(e.v, min(e.cap - e.flow, flow - used));
      if (tmp == 0) continue;
      e.flow += tmp, G[i ^ 1].flow -= tmp, used += tmp;
      if (used == flow) return used;
    }
  }
  --gap[dep[u]];
  if (!gap[dep[u]]) dep[src] = n + 1;
  ++gap[++dep[u]];
  return used;
}

int isap() {
  bfs();
  int res = 0;
  while (dep[src] < n) {
    memcpy(cur, h, sizeof h);
    res += dfs(src, inf);
  }
  return res;
}
```

数学与数论

自适应Simpson积分

$\int_a^b F(x)dx \Rightarrow$ `asr(a, b, eps, simpson(a, b))`

```
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);
}
```

BM推公式大法

```
struct BM {
  static const int MAXN = 10005;
  int n, pn, fail[MAXN];
  double delta[MAXN];
  vector<double> ps[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("%lf%c", ps[pn][i], (i == ps[pn].size() - 1) ? '\n' : ' ');
    }
  }
} B;
double x[BM::MAXN];
int main() {
  for (int n; ~scanf("%d", &n); ) {
    for (int i = 1; i <= n; i++) {
      scanf("%lf", &x[i]);
    }
    B.Solve(x, n), B.print();
  }
}
```

线性基

```
struct LinearBasis {
  const static int MAXL = 50;
  long long a[MAXL + 1];
  LinearBasis() {
    memset(a, 0, sizeof a);
  }
  void insert(long long t) {
    for (int j = MAXL; j >= 0; j--) {
      if (!(t & (1ll << j))) continue;
      if (a[j]) t ^= a[j];
      else {
        for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
        for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;
        a[j] = t;
        return;
      }
    }
  }
};
```

扩展欧几里得

```
pll exgcd(const long long x, const long long y) {
  if (!y) return make_pair(1, 0);
  pll cur = exgcd(y, x % y);
  return make_pair(cur.second, cur.first - (x / y) * cur.second);
}
```

中国剩余定理

```
//v里每个pll中first为被模数, second为模数
pll crt(const vector<pll> & v) {
  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);
}
```

扩展卢卡斯

```
ll C(ll n, ll m, ll p) {
  if(m > n) return 0;
  ll ret = 1;
  for(ll i = 1; i <= m; i++) {
    ll a = (n + 1 - i) % p, b = mod(exgcd(i % p, p).first, p);
    ret = ret * a % p * b % p;
  }
  return ret;
}

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

ll cal(ll n, ll a, ll b, ll p) {
  if(!n) return 1;
  ll y = n / p, tmp = 1;
  for(ll i = 1; i <= p; i++) {
    if(i % a) {
      tmp = tmp * i % p;
    }
  }
  ll ans = fpow(tmp, y, p);
  for(ll i = y * p + 1; i <= n; i++) {
    if(i % a) {
      ans = ans * (i % p) % p;
    }
  }
  return ans * cal(n / a, a, b, p) % p;
}

ll multilucas(ll n, ll m, ll a, ll b, ll p) {
  ll s = 0;
  for(ll i = n; i; i /= a) s += i / a;
  for(ll i = m; i; i /= a) s -= i / a;
  for(ll i = n - m; i; i /= a) s -= i / a;
  ll tmp = fpow(a, s, p);
  ll t1 = cal(n, a, b, p), t2 = cal(m, a, b, p), t3 = cal(n - m, a, b, p);
  return tmp * t1 % p * mod(exgcd(t2, p).first, p) % p * mod(exgcd(t3, p).first,
p) % p;
}

ll exlucas(ll n, ll m, ll p) {
  vector<ll>q, a;
  for(ll i = 2; i * i <= p; i++) {
    if(p % i == 0) {
      q.push_back(1);
      ll t = 0;
      while(p % i == 0) {
        p /= i;
        q.back() *= i;
        t++;
      }
      a.push_back(q.back() == i ? lucas(n, m, q.back()) : multilucas(n, m, i, t,
q.back()));
    }
  }
  if(p > 1) {
    q.push_back(p);
    a.push_back(lucas(n, m, p));
  }
  const int e = q.size();
  for(ll i = 1; i < e; i++) {
    pll d = exgcd(q[0], q[i]);
    ll c = a[i] - a[0], g = d.first * q[0] + d.second * q[i];
    if(c % g) exit(-1);
    a[0] = q[0] * mod(c / g * d.first, q[i] / g) + a[0];
    q[0] = q[0] * q[i] / g;
  }
  return a[0];
}
```

快速乘

```
// mod <= 1e12
inline ll mul(ll a, ll b, ll mod) {
  return (((a * (b >> 20) % mod) << 20) + (a * (b & ((1 << 20) - 1)))) % mod;
}
// mod <= 1e18
inline ll mul(ll a, ll b, ll mod) {
  ll d = (ll)floor(a * (long double)b / mod + 0.5);
  ll ret = (a * b - d * mod) % mod;
  if (ret < 0) ret += mod;
  return ret;
}
```

exbsgs

```
ll bsgs(ll a, ll b, ll c, ll q = 1, ll d = 0) {
  unordered_map<ll, ll> x;
  ll m = sqrt(c) + 1;
  ll v = 1;
  if(d > 0) {
    for(int i = 1; i <= m; i++) {
      v = fmul(v, a, c);
      x[fmul(v, b, c)] = i;
    }
  } else {
    for(int i = 0; i < m; i++) {
      x[fmul(v, b, c)] = i;
      v = fmul(v, a, c);
    }
  }
  for(int i = 1; i <= m; i++) {
    q = fmul(q, v, c);
    auto it = x.find(q);
    if(it != x.end()) {
      return i * m - it->second + d;
    }
  }
  return -1;
}
// 返回最小正整数n使得 a^n mod m = b; O(sqrt(m))
ll exbsgs(ll a, ll b, ll m) {
  a = mod(a, m), b = mod(b, m);
  if(a == 0) {
    return b > 1 ? -1 : b == 0 && m > 1;
  }
  if(b == 1 && gcd(a, m) != 1) { // b为1时随机应变吧。
    return -1;
  }
  ll g, c = 0, q = 1;
  while((g = gcd(a, m)) != 1) {
    if(b == 1) return c;
    if(b % g) return -1;
    c++;
    b /= g, m /= g;
    q = fmul(a / g, q, m);
  }
  return bsgs(a, b, m, q, c);
}
```

polysum

```
namespace polysum {
ll mod = 998244353LL;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
const int D=200005;
ll a[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h[D][2],C[D];
ll powmod(ll a,ll b) {
  ll res=1;
  a%=mod;
  assert(b>=0);
  for(; b; b>>=1) {
    if(b&1)res=res*a%mod;
    a=a*a%mod;
  }
  return res;
}
//函数用途：给出数列的（d+1）项，其中d为最高次方项
//求出数列的第n项，数组下标从0开始
ll calcn(int d,ll *a,ll n) { // a[0].. a[d]  a[n]
  if (n<=d) return a[n];
  p1[0]=p2[0]=1;
  rep(i,0,d+1) {
    ll t=(n-i+mod)%mod;
    p1[i+1]=p1[i]*t%mod;
  }
  rep(i,0,d+1) {
    ll t=(n-d+i+mod)%mod;
    p2[i+1]=p2[i]*t%mod;
  }
  ll ans=0;
  rep(i,0,d+1) {
    ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
    if ((d-i)&1) ans=(ans-t+mod)%mod;
    else ans=(ans+t)%mod;
  }
  return ans;
}
void init(int M) {
  f[0]=f[1]=g[0]=g[1]=1;
  rep(i,2,M+5) f[i]=f[i-1]*i%mod;
  g[M+4]=powmod(f[M+4],mod-2);
  per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
}
//函数用途：给出数列的（m+1）项，其中m为最高次方
//求出数列的前（n-1）项的和
ll polysum(ll m,ll *a,ll n) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
  ll b[D];
  for(int i=0; i<=m; i++) b[i]=a[i];
  b[m+1]=calcn(m,b,m+1);
  rep(i,1,m+2) b[i]=(b[i-1]+b[i])%mod;
  return calcn(m+1,b,n-1);
}
ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
  if (R==1) return polysum(n,a,m);
  a[m+1]=calcn(m,a,m+1);
  ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
  h[0][0]=0;
  h[0][1]=1;
  rep(i,1,m+2) {
    h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
    h[i][1]=h[i-1][1]*r%mod;
  }
  rep(i,0,m+2) {
    ll t=g[i]*g[m+1-i]%mod;
    if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,p4=((p4-h[i][1]*t)%mod+mod)%mod;
    else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i][1]*t)%mod;
  }
  c=powmod(p4,mod-2)*(mod-p3)%mod;
  rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
  rep(i,0,m+2) C[i]=h[i][0];
  ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
  if (ans<0) ans+=mod;
  return ans;
}
} // polysum::init();
```

线性筛

```
struct Seive {
  int maxn;
  vector<bool> isp;
  vector<int> p, phi, mu;
  Seive(int n = 0) : maxn(n), isp(n + 5, true), phi(n + 5, 0), mu(n + 5, 0) {
solve(); }
  void solve() {
    isp[0] = isp[1] = false;
    phi[1] = 1;
    mu[1] = 1;
    for (int i = 2; i <= maxn; i++) {
      if (isp[i]) {
        p.push_back(i);
        phi[i] = i - 1;
        mu[i] = -1;
      }
      for (int j = 0; j < (int)p.size() && i * p[j] <= maxn; j++) {
        const int cur = i * p[j];
        isp[cur] = false;
        if (i % p[j]) {
          phi[cur] = phi[i] * (p[j] - 1);
          mu[cur] = -mu[i];
        } else {
          phi[cur] = phi[i] * p[j];
          mu[cur] = 0;
          break;
        }
      }
    }
  }
};
```

MillerRabin素性测试

```
const int psize = 1010000;
bool isp[psize];
int prime[psize], tot;
void prime_table() {
  register int i, j;
  for (i = 2, tot = 0; i < psize; i++) {
    if (!isp[i]) prime[tot++] = i;
    for (j = 0; j < tot && prime[j] * i < psize; j++) {
      isp[prime[j] * i] = true;
      if (i % prime[j] == 0) break;
    }
  }
}
bool witness(ll a, ll n) {
  int t = 0;
  ll u = n - 1;
  for (; ~u & 1; u >>= 1) t++;
  ll x = qpow(a, u, n), _x = 0;
  while (t--) {
    _x = mul(x, x, n);
    if (_x == 1 && x != 1 && x != n - 1) return true;
    x = _x;
  }
  return _x != 1;
}
bool Miller(ll n) {
  if (n < 2) return false;
  if (n < psize) return !isp[n];
  if (~n & 1) return false;
  for (int j = 0; j <= 7; j++) {
    if (witness(rand() % (n - 1) + 1, n)) {
      return false;
    }
  }
  return true;
}
```

pollard_rho分解质因数

```
int tot;
long long factor[10000];
long long pollard_rho(long long x, long long c) {
  long long i = 1, k = 2;
  long long x0 = rand() % x, y = x0;
  while (true) {
    i++;
    x0 = (mul(x0, x0, x) + c) % x;
    long long d = __gcd(y - x0, x);
    if (d != 1 && d != x) return d;
    if (y == x0) return x;
    if (i == k) {
      y = x0, k <<= 1;
    }
  }
}
void findfac(long long n) {
  if (Miller(n)) {
    factor[tot++] = n;
    return;
  }
  long long p = n;
  while (p >= n) p = pollard_rho(p, rand() % (n - 1) + 1);
  findfac(p), findfac(n / p);
}
```

fft

```
namespace fft {
```

```
const double pi = acos(-1.0);
struct Complex {
  double r, i;
  Complex(double x = 0, double y = 0) : r(x), i(y) {}
  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);
  }
};
Complex conj(Complex a) { return Complex(a.r, -a.i); }

int base = 1;
vector<int> rev = { 0, 1 };
vector<Complex> roots = { { 0, 0 }, { 1, 0 } };

void ensure_base(int nbase) {
  if (nbase <= base) return;
  rev.resize(1 << nbase);
  for (int i = 0; i < (1 << nbase); i++) {
    rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
  }
  roots.resize(1 << nbase);
  while (base < nbase) {
    double angle = 2 * pi / (1 << (base + 1));
    for (int i = 1 << (base - 1); i < (1 << base); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << base));
      roots[(i << 1) + 1] = Complex(cos(angle_i), sin(angle_i));
    }
    base++;
  }
}
void fft(vector<Complex> &a, int n = -1) {
  if (n == -1) {
    n = a.size();
  }
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for (int i = 0; i < n; i++) {
    if (i < (rev[i] >> shift)) {
      swap(a[i], a[rev[i] >> shift]);
    }
  }
  for (int k = 1; k < n; k <<= 1) {
      for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; j++) {
          Complex z = a[i + j + k] * roots[j + k];
          a[i + j + k] = a[i + j] - z;
          a[i + j] = a[i + j] + z;
        }
      }
    }
  }

  vector<Complex> fa, fb;
  vector<int> multiply(const vector<int> &a, const vector<int> &b) {
    int need = a.size() + b.size() - 1;
    int nbase = 32 - __builtin_clz(need) - (need - need & (-need) == 0);
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < sz; i++) {
      int x = (i < (int) a.size() ? a[i] : 0);
      int y = (i < (int) b.size() ? b[i] : 0);
      fa[i] = Complex(x, y);
    }
    fft(fa, sz);
    Complex r(0, -0.25 / sz);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      Complex z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
      if (i != j) {
        fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
      }
      fa[i] = z;
    }
    fft(fa, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      res[i] = fa[i].r + 0.5;
    }
    return res;
  }
  vector<int> multiply_mod(const vector<int> &a, const vector<int> &b, int m,
int eq = 0) {
    int need = a.size() + b.size() - 1;
    int nbase = 32 - __builtin_clz(need) - (need - need & (-need) == 0);
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < (int) a.size(); i++) {
```

```
    int x = (a[i] % m + m) % m;
    fa[i] = Complex(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fa.begin() + a.size(), fa.begin() + sz, Complex {0, 0});
  fft(fa, sz);
  if (sz > (int) fb.size()) {
    fb.resize(sz);
  }
  if (eq) {
    copy(fa.begin(), fa.begin() + sz, fb.begin());
  } else {
    for (int i = 0; i < (int) b.size(); i++) {
      int x = (b[i] % m + m) % m;
      fb[i] = Complex(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fb.begin() + b.size(), fb.begin() + sz, Complex {0, 0});
    fft(fb, sz);
  }
  double ratio = 0.25 / sz;
  Complex r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0, 1);
  for (int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    Complex a1 = (fa[i] + conj(fa[j]));
    Complex a2 = (fa[i] - conj(fa[j])) * r2;
    Complex b1 = (fb[i] + conj(fb[j])) * r3;
    Complex b2 = (fb[i] - conj(fb[j])) * r4;
    if (i != j) {
      Complex c1 = (fa[j] + conj(fa[i]));
      Complex c2 = (fa[j] - conj(fa[i])) * r2;
      Complex d1 = (fb[j] + conj(fb[i])) * r3;
      Complex d2 = (fb[j] - conj(fb[i])) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz);
  fft(fb, sz);
  vector<int> res(need);
  for (int i = 0; i < need; i++) {
    long long aa = fa[i].r + 0.5;
    long long bb = fb[i].r + 0.5;
    long long cc = fa[i].i + 0.5;
    res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
  }
  return res;
}
vector<int> square_mod(const vector<int> &a, int m) {
  return multiply_mod(a, a, m, 1);
}
```

```
}
```

ntt

```
namespace ntt {
  int qpow(int a, int t, int mod) {
    ll b = 1;
    for (; t; t >>= 1, a = (ll)a * a % mod) {
      if (t & 1) b = b * a % mod;
    }
    return b;
  }
  int revv(int x, int bits) {
    int ret = 0;
    for (int i = 0; i < bits; i++) {
      ret <<= 1, ret |= x & 1, x >>= 1;
    }
    return ret;
  }
  void ntt(vector<int> &a, bool rev, int mod, int root) {
    int n = (int)a.size(), bits = 31 - __builtin_clz(n);
    for (int i = 0; i < n; i++) {
      int j = revv(i, bits);
      if (i < j) swap(a[i], a[j]);
    }
    for (int k = 1; k < n; k <<= 1) {
      int e = qpow(root, (mod - 1) / 2 / k, mod);
      if (rev) e = qpow(e, mod - 2, mod); // exgcd is better
      for (int i = 0; i < n; i += 2 * k) {
        ll w = 1;
        for (int j = 0; j < k; j++, w = w * e % mod) {
          int x = a[i + j], y = w * a[i + j + k] % mod;
          a[i + j] = (x + y) % mod, a[i + j + k] = (x - y + mod) % mod;
        }
      }
    }
    if (rev) {
      int inv = qpow(n, mod - 2, mod); // exgcd is better
      for (int i = 0; i < n; i++) a[i] = 1ll * a[i] * inv % mod;
    }
  }
  // mod = 998244353 = (119 << 23) + 1, root = 3, // = (119 << 23, 3)
  // For p < 2^30, (5 << 25, 3), (7 << 26, 3),
  // (479 << 21, 3) and (483 << 21, 5), last two are > 10^9
  vector<int> conv(const vector<int>& a, const vector<int>& b, const int mod =
(119 << 23) + 1, int root = 3) {
    int sz = (int)a.size() + (int)b.size() - 1;
    int L = sz > 1 ? (32 - __builtin_clz(sz - 1)) : 0, n = 1 << L;
    vector<int> av(n), bv(n);
    copy(a.begin(), a.end(), av.begin());
    copy(b.begin(), b.end(), bv.begin());
    ntt(av, false, mod, root), ntt(bv, false, mod, root);
    for (int i = 0; i < n; i++) {
      av[i] = 1ll * av[i] * bv[i] % mod;
    }
    ntt(av, true, mod, root);
    av.resize(sz);
    return av;
  }
}
```

linear_seq

```
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define SZ(x) ((int)(x).size())

typedef vector<int> VI;
typedef pair<int, int> PII;

const ll mod = 1e9 + 7;

ll powmod(ll a, ll b) {
  ll res = 1; a %= mod;
  assert(b >= 0);
  for (; b; b >>= 1) {
    if (b & 1) res = res * a % mod;
    a = a * a % mod;
  }
  return res;
}

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

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

MeisselLehmer

Count the number of primes in [1, n].

```
namespace pcf {
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;
}
}
```

Geometry

多边形

```
#include<bits/stdc++.h>
#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?2:0))
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

```
#include<bits/stdc++.h>
#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?2:0))
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-l.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-l2.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_o
nline_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,u
2)||dot_online_in(v2,u1,u2);
}
//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
ponit 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;in;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 <cstdio>
#include <cmath>
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 perpencenter(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)/((l/R)-(2*pi));
            x = r*r*s/(R*R);
            ans = s - x + pi * r * r;
        }
        printf("Case %d: %.2lf\n",T,ans);
     }
     return 0;
 }
```

三维几何

```
//三维几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3{double x,y,z;};
struct line3{point3 a,b;};
struct plane3{point3 a,b,c;};
//计算cross product U x V
point3 xmult(point3 u,point3 v){
    point3 ret;
    ret.x=u.y*v.z-v.y*u.z;
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}
```

```
//计算dot product U . V
double dmult(point3 u,point3 v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}
//矢量差 U - V
point3 subt(point3 u,point3 v){
    point3 ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}
//取平面法向量
point3 pvec(plane3 s){
    return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
    return xmult(subt(s1,s2),subt(s2,s3));
}
//两点距离,单参数取向量大小
double distance(point3 p1,point3 p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-
p2.z)*(p1.z-p2.z));
}
//向量大小
double vlen(point3 p){
    return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}
//判三点共线
int dots_inline(point3 p1,point3 p2,point3 p3){
    return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
}
//判四点共面
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
    return zero(dmult(pvec(a,b,c),subt(d,a)));
}
//判点是否在线段上,包括端点和共线
int dot_online_in(point3 p,line3 l){
    return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*
(l.b.x-p.x)<eps&&
        (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
}
int dot_online_in(point3 p,point3 l1,point3 l2){
    return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-
p.x)<eps&&
        (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
}
//判点是否在线段上,不包括端点
int dot_online_ex(point3 p,line3 l){
    return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-
l.a.y)||!zero(p.z-l.a.z))&&
        (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2){
    return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-
l1.y)||!zero(p.z-l1.z))&&
        (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}
//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p,plane3 s){
    return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-
vlen(xmult(subt(p,s.a),subt(p,s.b)))-
        vlen(xmult(subt(p,s.b),subt(p,s.c)))-
vlen(xmult(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
    return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-
vlen(xmult(subt(p,s1),subt(p,s2)))-
        vlen(xmult(subt(p,s2),subt(p,s3)))-
vlen(xmult(subt(p,s3),subt(p,s1))));
}
//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p,plane3 s){
    return
dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&&

vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p
,s.a)))>eps;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
    return
dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&&

vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1
)))>eps;
}
//判两点在线段同侧,点在线段上返回0,不共面无意义
int same_side(point3 p1,point3 p2,line3 l){
    return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b
)))>eps;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>ep
s;
}
//判两点在线段异侧,点在线段上返回0,不共面无意义
int opposite_side(point3 p1,point3 p2,line3 l){
    return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b
)))<-eps;
```

```
}
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-
eps;
}
//判两点在平面同侧,点在平面上返回0
int same_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>eps;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>eps
;
}
//判两点在平面异侧,点在平面上返回0
int opposite_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-
eps;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-
eps;
}
//判两直线平行
int parallel(line3 u,line3 v){
    return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
    return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;
}
//判两平面平行
int parallel(plane3 u,plane3 v){
    return vlen(xmult(pvec(u),pvec(v)))<eps;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3
v3){
    return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
}
//判直线与平面平行
int parallel(line3 l,plane3 s){
    return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}
//判两直线垂直
int perpendicular(line3 u,line3 v){
    return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
}
```

```
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
    return zero(dmult(subt(u1,u2),subt(v1,v2)));
}
//判两平面垂直
int perpendicular(plane3 u,plane3 v){
    return zero(dmult(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
    return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}
//判直线与平面平行
int perpendicular(line3 l,plane3 s){
    return vlen(xmult(subt(l.a,l.b),pvec(s)))<eps;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<eps;
}
//判两线段相交,包括端点和部分重合
int intersect_in(line3 u,line3 v){
    if (!dots_onplane(u.a,u.b,v.a,v.b))
        return 0;
    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_o
nline_in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
    if (!dots_onplane(u1,u2,v1,v2))
        return 0;
    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,u
2)||dot_online_in(v2,u1,u2);
}
//判两线段相交,不包括端点和部分重合
int intersect_ex(line3 u,line3 v){
    return
dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(
v.a,v.b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
    return
dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1
,v2,u1,u2);
}
//判线段与空间三角形相交,包括交于边界和(部分)包含
int intersect_in(line3 l,plane3 s){
    return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
!same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
        !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}
//判线段与空间三角形相交,不包括交于边界和(部分)包含
int intersect_ex(line3 l,plane3 s){
    return
opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&

opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return
opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
        opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}
//计算两直线交点,注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point3 intersection(line3 u,line3 v){
    point3 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;
    ret.z+=(u.b.z-u.a.z)*t;
    return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 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;
    ret.z+=(u2.z-u1.z)*t;
    return ret;
}
//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
//线段和空间三角形交点请另外判断
point3 intersection(line3 l,plane3 s){
    point3 ret=pvec(s);
    double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-
l.a.z))/
        (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
    ret.x=l.a.x+(l.b.x-l.a.x)*t;
    ret.y=l.a.y+(l.b.y-l.a.y)*t;
    ret.z=l.a.z+(l.b.z-l.a.z)*t;
    return ret;
}
```

```
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    point3 ret=pvec(s1,s2,s3);
    double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
        (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
    ret.x=l1.x+(l2.x-l1.x)*t;
    ret.y=l1.y+(l2.y-l1.y)*t;
    ret.z=l1.z+(l2.z-l1.z)*t;
    return ret;
}
//计算两平面交线,注意事先判断是否平行,并保证三点不共线!
line3 intersection(plane3 u,plane3 v){
    line3 ret;
    ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?
intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.a,v.b,u.a,u.b,u.c);
    ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?
intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.c,v.a,u.a,u.b,u.c);
    return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
    line3 ret;
    ret.a=parallel(v1,v2,u1,u2,u3)?
intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,u2,u3);
    ret.b=parallel(v3,v1,u1,u2,u3)?
intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,u2,u3);
    return ret;
}
//点到直线距离
double ptoline(point3 p,line3 l){
    return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2){
    return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}
//点到平面距离
double ptoplane(point3 p,plane3 s){
    return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
    return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}
//直线到直线距离
double linetoline(line3 u,line3 v){
    point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
    return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 n=xmult(subt(u1,u2),subt(v1,v2));
    return fabs(dmult(subt(u1,v1),n))/vlen(n);
}
//两直线夹角cos值
```

```
double angle_cos(line3 u,line3 v){
    return
dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.
b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
    return
dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}
//两平面夹角cos值
double angle_cos(plane3 u,plane3 v){
    return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3
v2,point3 v3){
    return
dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,
v2,v3));
}
//直线平面夹角sin值
double angle_sin(line3 l,plane3 s){
    return
dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return
dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3)
);
}
```

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;
}
```

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 sqrx()                    //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.sqrx()/b.sqrx() )*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.sqrx()/b.sqrx() )*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.sqrx()/n.sqrx() )*r*r;
        double ret2 = acos( p1*p2/p1.sqrx()/p2.sqrx() )*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;
}
```

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;
}
//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;
}
```