Contents

1 Flow

1.1 Dinic's

```
struct Dinic {
  int n, s, t;
  vector<int> level;
  struct Edge {
    int to, rev, cap;
    Edge() {}
    Edge(int a, int b, int c): to(a), cap(b), rev(c) {}
  };
  vector<Edge> G[maxn];
  bool bfs() {
    level.assign(n, -1);
    level[s] = 0;
    queue<int> que; que.push(s);
    while (que.size()) {
      int tmp = que.front(); que.pop();
      for (auto e : G[tmp]) {
        if (e.cap > 0 && level[e.to] == -1) {
          level[e.to] = level[tmp] + 1;
          que.push(e.to);
        }
      }
    }
    return level[t] != -1;
  }
  int flow(int now, int low) {
    if (now == t) return low;
    int ret = 0;
    for (auto &e : G[now]) {
      if (e.cap > 0 && level[e.to] == level[now] + 1) {
        int tmp = flow(e.to, min(e.cap, low - ret));
        e.cap -= tmp; G[e.to][e.rev].cap += tmp;
        ret += tmp;
      }
    }
    if (ret == 0) level[now] = -1;
    return ret;
  }
  Dinic(int _n, int _s, int _t): n(_n), s(_s), t(_t) {
    fill(G, G + maxn, vector<Edge>());
  }
  void add_edge(int a, int b, int c) {
    G[a].push_back(Edge(b, c, G[b].size()));
    G[b].push_back(Edge(a, 0, G[a].size() - 1));
  }
  int maxflow() {
    int ret = 0;
    while (bfs()) ret += flow(s, inf);
    return ret;
  }
};
```

1.2 MinCostMaxFlow

```
struct MincostMaxflow {
  struct Edge {
    int to, rev, cap, w;
    Edge() {}
    Edge(int a, int b, int c, int d): to(a), cap(b), w(
    c), rev(d) {}
  };
  int n, s, t, p[maxn], id[maxn];
  int d[maxn];
  bool inque[maxn];
  vector<Edge> G[maxn];
  pair<int, int> spfa() {
    memset(p, -1, sizeof(-1));
    fill(d, d + maxn, inf);
    memset(id, -1, sizeof(id));
    d[s] = 0; p[s] = s;
    queue<int> que; que.push(s); inque[s] = true;
    while (que.size()) {
      int tmp = que.front(); que.pop();
      inque[tmp] = false;
      int i = 0;
```

```
      for (auto e : G[tmp]) {
        if (e.cap > 0 && d[e.to] > d[tmp] + e.w) {
          d[e.to] = d[tmp] + e.w;
          p[e.to] = tmp;
          id[e.to] = i;
          if (!inque[e.to]) que.push(e.to), inque[e.to]
     = true;
        }
        ++i;
      }
    }
    if (d[t] == inf) return make_pair(-1, -1);
    int a = inf;
    for (int i = t; i != s; i = p[i]) {
      a = min(a, G[p[i]][id[i]].cap);
    }
    for (int i = t; i != s; i = p[i]) {
      Edge &e = G[p[i]][id[i]];
      e.cap -= a; G[e.to][e.rev].cap += a;
    }
    return make_pair(a, d[t]);
  }
  MincostMaxflow(int _n, int _s, int _t): n(_n), s(_s),
     t(_t) {
    fill(G, G + maxn, vector<Edge>());
  }
  void add_edge(int a, int b, int cap, int w) {
    G[a].push_back(Edge(b, cap, w, (int)G[b].size()));
    G[b].push_back(Edge(a, 0, -w, (int)G[a].size() - 1)
    );
  }
  pair<int, int> maxflow() {
    int mxf = 0, mnc = 0;
    while (true) {
      pair<int, int> res = spfa();
      if (res.first == -1) break;
      mxf += res.first; mnc += res.first * res.second;
    }
    return make_pair(mxf, mnc);
  }
};
```

1.3 Hungarian

```
struct Hungarian {
  vector<vector<int>> w;
  bitset<maxn> s, t;
  vector<int> lx, ly, mx, my, slack, prv;
  int n, matched;
  Hungarian() {}
  Hungarian(int _n): n(_n) {
    w = vector<vector<int>>(n, vector<int>(n));
    lx.resize(n); ly.resize(n); mx.assign(n, -1); my.
    assign(n, -1);
    slack.resize(n); prv.resize(n);
  }
  void add_edge(int a, int b, int c) {
    w[a][b] = c;
  }
  void add(int x) {
    s[x] = true;
    for (int i = 0; i < n; ++i) {
      if (lx[x] + ly[i] - w[x][i] < slack[i]) {
        slack[i] = lx[x] + ly[i] - w[x][i];
        prv[i] = x;
      }
    }
  }
  void augment(int now) {
    int x = prv[now], y = now;
    ++matched;
    while (true) {
      int tmp = mx[x]; mx[x] = y; my[y] = x; y = tmp;
      if (y == -1) return;
      x = prv[y];
    }
  }
  void relabel() {
    int delta = inf;
    for (int i = 0; i < n; ++i) if (!t[i]) delta = min(
    delta, slack[i]);
    for (int i = 0; i < n; ++i) if (s[i]) lx[i] -=
    delta;
    for (int i = 0; i < n; ++i) {
      if (t[i]) ly[i] += delta;
      else slack[i] -= delta;
    }
  }
  void go() {
    s.reset(); t.reset();
    fill(slack.begin(), slack.end(), inf);
    int root = 0;
    for (; root < n && mx[root] != -1; ++root);
    add(root);
    while (true) {
      relabel();
      int y = 0;
      for (; y < n; ++y) if (!t[y] && slack[y] == 0)
    break;
      if (my[y] == -1) return augment(y), void();
      add(my[y]); t[y] = true;
    }
  }
  int matching() {
    int ret = 0;
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) lx[i] = max(lx[i], w[
    i][j]);
    }
    for (int i = 0; i < n; ++i) go();
    for (int i = 0; i < n; ++i) ret += w[i][mx[i]];
    return ret;
  }
};
```

2 Data Structure

2.1 Disjoint Set

```
struct DisjointSet {
  int p[maxn], sz[maxn], n, cc;
  vector<pair<int*, int>> his;
  vector<int> sh;
  void init(int _n) {
    n = _n; cc = n;
    for (int i = 0; i < n; ++i) sz[i] = 1, p[i] = i;
    sh.clear(); his.clear();
  }
  void assign(int *k, int v) {
    his.emplace_back(k, *k);
    *k = v;
  }
  void save() {
    sh.push_back((int)his.size());
  }
  void undo() {
    int last = sh.back(); sh.pop_back();
    while (his.size() != last) {
      int *k, v;
      tie(k, v) = his.back(); his.pop_back();
      *k = v;
    }
  }
  int find(int x) {
    if (x == p[x]) return x;
    return find(p[x]);
  }
  void merge(int x, int y) {
    x = find(x); y = find(y);
    if (x == y) return;
    if (sz[x] > sz[y]) swap(x, y);
    assign(&sz[y], sz[x] + sz[y]);
    assign(&p[x], y);
    assign(&cc, cc - 1);
  }
} dsu;
```

2.2 Splay Tree

```
struct node {
  static node nil;
  node *ch[2], *fa;
  int val, sz, tag;
  node(): sz(0), tag(0), val(-1) { fa = ch[0] = ch[1] =
     &nil; }
  node(int v): val(v), sz(1), tag(0) { fa = ch[0] = ch
    [1] = &nil; }
  bool r() { return fa->ch[0] != this && fa->ch[1] !=
    this; }
  int dir() { return fa->ch[0] == this ? 0 : 1; }
  void pull() {
    sz = ch[0]->sz + ch[1]->sz + 1;
    if (ch[0] != &nil) ch[0]->fa = this;
    if (ch[1] != &nil) ch[1]->fa = this;
  }
  void push() {
    if (tag == 0) return;
    swap(ch[0], ch[1]);
    if (ch[0] != &nil) ch[0]->tag ^= 1;
    if (ch[1] != &nil) ch[1]->tag ^= 1;
    tag = 0;
  }
  void addch(node *c, int d) {
    ch[d] = c;
    if (c != &nil) c->fa = this;
    pull();
  }
} node::nil;

node *nil = &node::nil;

void rotate(node *s) {
  node *p = s->fa;
  int d = s->dir();
  if (!p->r()) p->fa->addch(s, p->dir());
  else s->fa = p->fa;
  p->addch(s->ch[d ^ 1], d);
  s->addch(p, d ^ 1);
  p->pull(); s->pull();
}

void splay(node *s) {
  vector<node*> vec;
  for (node *n = s; ; n = n->fa) {
    vec.push_back(n);
    if (n->r()) break;
  }
  reverse(vec.begin(), vec.end());
  for (auto it : vec) it->push();
  while (!s->r()) {
    if (s->fa->r()) rotate(s);
    else if (s->dir() == s->fa->dir()) rotate(s->fa),
    rotate(s);
    else rotate(s), rotate(s);
  }
}
```

2.3 Link-Cut Tree

```
node *access(node *s) {
  node *n = nil;
  for (; s != nil; s = s->fa) {
    splay(s);
    s->addch(n, 1);
    n = s;
  }
  return n;
}

void evert(node *s) {
  access(s); splay(s);
  s->tag ^= 1;
  s->push(); s->pull();
}

void link(node *a, node *b) {
  access(a); splay(a);
  evert(b);
  a->addch(b, 1);
}

void cut(node *a, node *b) {
  access(b); splay(b);
  b->push();
  b->ch[0] = b->ch[0]->fa = nil;
}

node *find(node *s) {
  s = access(s);
  while (s->ch[0] != nil) s = s->ch[0];
  splay(s);
  return s;
}

int query(node *a, node *b) {
  access(a); access(b);
  splay(a);
  int ret = a->fa->val;
  if (ret == -1) ret = a->val;
  return ret;
}
```

2.4 <ext/pbds>

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef tree<int, null_type, std::less<int>,
    rb_tree_tag, tree_order_statistics_node_update>
    tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
  // rb tree
  tree_set s;
  s.insert(71); s.insert(22);
  assert(*s.find_by_order(0) == 22); assert(*s.
    find_by_order(1) == 71);
  assert(s.order_of_key(22) == 0); assert(s.
    order_of_key(71) == 1);
  s.erase(22);
  assert(*s.find_by_order(0) == 71); assert(s.
    order_of_key(71) == 0);
  // mergable heap
  heap a, b; a.join(b);
  // persistant
  rope<char> r[2];
  r[1] = r[0];
  std::string st = "abc";
  r[1].insert(0, st.c_str());
  r[1].erase(1, 1);
  std::cout << r[1].substr(0, 2) << std::endl;
  return 0;
}
```

3 Graph

3.1 Heavy-Light Decomposition

```
struct HeavyLightDecomp {
  vector<int> G[maxn];
  int tin[maxn], top[maxn], dep[maxn], maxson[maxn], sz
    [maxn], p[maxn], n, clk;
  void dfs(int now, int fa, int d) {
    dep[now] = d;
    maxson[now] = -1;
    sz[now] = 1;
    p[now] = fa;
```

```
    for (int u : G[now]) if (u != fa) {
      dfs(u, now, d + 1);
      sz[now] += sz[u];
      if (maxson[now] == -1 || sz[u] > sz[maxson[now]])
     maxson[now] = u;
    }
  }
  void link(int now, int t) {
    top[now] = t;
    tin[now] = ++clk;
    if (maxson[now] == -1) return;
    link(maxson[now], t);
    for (int u : G[now]) if (u != p[now]) {
      if (u == maxson[now]) continue;
      link(u, u);
    }
  }
  HeavyLightDecomp(int n): n(n) {
    clk = 0;
    memset(tin, 0, sizeof(tin)); memset(top, 0, sizeof(
    top)); memset(dep, 0, sizeof(dep));
    memset(maxson, 0, sizeof(maxson)); memset(sz, 0,
    sizeof(sz)); memset(p, 0, sizeof(p));
  }
  void add_edge(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
  }
  void solve() {
    dfs(0, -1, 0);
    link(0, 0);
  }
  int lca(int a, int b) {
    int ta = top[a], tb = top[b];
    while (ta != tb) {
      if (dep[ta] < dep[tb]) {
        swap(ta, tb); swap(a, b);
      }
      a = p[ta]; ta = top[a];
    }
    if (a == b) return a;
    return dep[a] < dep[b] ? a : b;
  }
  vector<pair<int, int>> get_path(int a, int b) {
    int ta = top[a], tb = top[b];
    vector<pair<int, int>> ret;
    while (ta != tb) {
      if (dep[ta] < dep[tb]) {
        swap(ta, tb); swap(a, b);
      }
      ret.push_back(make_pair(tin[ta], tin[a]));
      a = p[ta]; ta = top[a];
    }
    ret.push_back(make_pair(min(tin[a], tin[b]), max(
    tin[a], tin[b])));
    return ret;
  }
};
```

3.2 Centroid Decomposition

```
vector<pair<int, int>> G[maxn];
int sz[maxn], mx[maxn];
bool v[maxn];
vector<int> vtx;

void get_center(int now) {
  v[now] = true; vtx.push_back(now);
  sz[now] = 1; mx[now] = 0;
  for (int u : G[now]) if (!v[u]) {
    get_center(u);
    mx[now] = max(mx[now], sz[u]);
    sz[now] += sz[u];
  }
}

void get_dis(int now, int d, int len) {
  dis[d][now] = cnt;
  v[now] = true;
  for (auto u : G[now]) if (!v[u.first]) {
    get_dis(u, d, len + u.second);
  }
}

void dfs(int now, int fa, int d) {
  get_center(now);
  int c = -1;
  for (int i : vtx) {
    if (max(mx[i], (int)vtx.size() - sz[i]) <= (int)vtx
    .size() / 2) c = i;
    v[i] = false;
  }
  get_dis(c, d, 0);
  for (int i : vtx) v[i] = false;
  v[c] = true; vtx.clear();
  dep[c] = d; p[c] = fa;
  for (auto u : G[c]) if (u.first != fa && !v[u.first])
     {
    dfs(u.first, c, d + 1);
  }
}
```

3.3 Maximum Clique

```
struct MaxClique {
  int n, deg[maxn], ans;
  bitset<maxn> adj[maxn];
  vector<pair<int, int>> edge;
  void init(int _n) {
    _n = n;
    for (int i = 0; i < n; ++i) adj[i].reset();
  }
  void add_edge(int a, int b) {
    edge.emplace_back(a, b);
    ++deg[a]; ++deg[b];
  }
  int solve() {
    vector<int> ord;
    for (int i = 0; i < n; ++i) ord.push_back(i);
    sort(ord.begin(), ord.end(), [&](const int &a,
    const int &b) { return deg[a] < deg[b]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
      int u = id[e.first], v = id[e.second];
      adj[u][v] = adj[v][u] = true;
    }
    bitset<maxn> r, p;
    for (int i = 0; i < n; ++i) p[i] = true;
    dfs(r, p);
    return ans;
  }
  void go(bitset<maxn> r, bitset<maxn> p) {
    if (1.0 * clock() / CLOCKS_PER_SEC >= time_limit)
    return;
    if (p.count() == 0) return ans = max(ans, (int)r.
    count()), void();
    if ((r | p).count() <= ans) return;
    int now = p._Find_first();
    bitset<maxn> cur = p & ~adj[now];
    for (now = cur._Find_first(); now < n; now = cur.
    _Find_next(now)) {
      r[now] = true;
      go(r, p & adj[now]);
      r[now] = false;
      p[now] = false;
    }
  }
};
```

3.4 Tarjan's

```
int tin[maxn], low[maxn], t, bccsz;
stack<int> st;
vector<int> bcc[maxn];

void dfs(int now, int fa) {
  tin[now] = ++t; low[now] = tin[now];
```

```
  st.push(now);
  for (int u : G[now]) if (u != fa) {
    if (!tin[u]) {
      dfs(u, now);
      low[now] = min(low[now], low[u]);
      if (low[u] >= tin[now]) {
        int v;
        ++bccsz;
        do {
          v = st.top(); st.pop();
          bcc[bccsz].push_back(v);
        } while (v != u);
        bcc[bccsz].push_back(now);
      }
    } else {
      low[now] = min(low[now], tin[u]);
    }
  }
}
```

4 String

4.1 KMP

```
int f[maxn];

int kmp(const string& a, const string& b) {
  f[0] = -1; f[1] = 0;
  for (int i = 1, j = 0; i < b.size() - 1; f[++i] = ++j
    ) {
    if (b[i] == b[j]) f[i] = f[j];
    while (j != -1 && b[i] != b[j]) j = f[j];
  }
  for (int i = 0, j = 0; i - j + b.size() <= a.size();
    ++i, ++j) {
    while (j != -1 && a[i] != b[j]) j = f[j];
    if (j == b.size() - 1) return i - j;
  }
  return -1;
}
```

4.2 Z algorithm

```
int z[maxn];

void z_function(const string& s) {
  memset(z, 0, sizeof(z));
  z[0] = (int)s.length();
  int l = 0, r = 0;
  for (int i = 1; i < s.length(); ++i) {
    z[i] = max(0, min(z[i - l], r - i + 1));
    while (i + z[i] < s.length() && s[z[i]] == s[i + z[
    i]]) {
      l = i; r = i + z[i];
      ++z[i];
    }
  }
}
```

4.3 Manacher's

```
int z[maxn];

int manacher(const string& s) {
  string t = ".";
  for (int i = 0; i < s.length(); ++i) t += s[i], t +=
    '.';
  int l = 0, r = 0;
  for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length() && t[
    i - z[i]] == t[i + z[i]]) ++z[i];
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  int ans = 0;
  for (int i = 1; i < t.length(); ++i) ans = max(ans, z
    [i] - 1);
  return ans;
}
```

4.4 Aho-Corasick

```
struct AC {
  int ptr, ql, qr, root;
  vector<int> cnt, q, ed, el, ch[sigma], f;
  void clear(int p) { for (int i = 0; i < sigma; ++i)
    ch[i][p] = 0; }
  int newnode() { clear(ptr); ed[ptr] = 0; return ptr
    ++; }
  void init() {
    ptr = 1; cnt.resize(maxn); q.resize(maxn);
    ed.resize(maxn); el.resize(maxn); f.resize(maxn);
    for (int i = 0; i < sigma; ++i) ch[i].resize(maxn);
    root = newnode();
  }
  int add(const string &s) {
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
      if (ch[s[i]][now] == 0) ch[s[i]][now] = newnode()
      ;
      now = ch[s[i]][now];
    }
    ed[now] = 1;
    return now;
  }
  void build_fail() {
    ql = qr = 0; q[qr++] = root;
    while (ql < qr) {
      int now = q[ql++];
      for (int i = 0; i < sigma; ++i) if (ch[i][now]) {
        int p = ch[i][now], fp = f[now];
        while (fp && !ch[i][fp]) fp = f[fp];
        int pd = fp ? ch[i][fp] : root;
        f[p] = pd;
        el[p] = ed[pd] ? pd : el[pd];
        q[qr++] = p;
      }
    }
  }
  void build(const string &s) {
    build_fail();
    int now = 1;
    for (int i = 0; i < s.length(); ++i) {
      while (now && !ch[s[i]][now]) now = f[now];
      now = now ? ch[s[i]][now] : root;
      ++cnt[now];
    }
    for (int i = qr - 1; i >= 0; --i) cnt[f[q[i]]] +=
    cnt[q[i]];
  }
};
```

4.5 Primes (hasing)

```
const int mod[] = { 479001599, 433494437, 1073807359,
    1442968193, 715827883 }
const int p[] = { 101, 233, 457, 173, 211 }
```

5 Math

5.1 FFT

```
const double pi = acos(-1);
const complex<double> I(0, 1);
complex<double> omega[maxn + 1];

void prefft() {
```

```
  for (int i = 0; i <= maxn; ++i) omega[i] = exp(i * 2
    * pi / maxn * I);
}

void fft(vector<complex<double>>& a, int n, bool inv =
    false) {
  int basic = maxn / n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int h = m >> 1;
    for (int i = 0; i < h; ++i) {
      complex<double> w = omega[inv ? maxn - (i * theta
     % maxn) : i * theta % maxn];
      for (int j = i; j < n; j += m) {
        int k = j + h;
        complex<double> x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
    theta = (theta * 2) % maxn;
  }
  int i = 0;
  for (int j = 1; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  if (inv) for (int i = 0; i < n; ++i) a[i] /= (double)
    n;
}
```

5.2 NTT

```
const long long p = 2013265921, root = 31;
long long omega[maxn + 1];

long long fpow(long long a, long long n) {
  long long ret = 1ll;
  for (; n; n >>= 1) {
    if (n & 1) ret = ret * a % p;
    a = a * a % p;
  }
  return ret;
}

void prentt() {
  omega[0] = 1;
  long long r = fpow(root, (p - 1) / maxn);
  for (int i = 1; i <= maxn; ++i) omega[i] = omega[i -
    1] * r % p;
}

void ntt(vector<long long>& a, int n, bool inv = false)
     {
  int basic = maxn / n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i = 0; i < mh; ++i) {
      long long w = omega[i * theta % maxn];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        long long x = a[j] - a[k];
        if (x < 0) x += p;
        a[j] += a[k];
        if (a[j] > p) a[j] -= p;
        a[k] = w * x % p;
      }
    }
    theta = theta * 2 % maxn;
  }
  int i = 0;
  for (int j = 1; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  if (!inv) return;
  long long ni = fpow(n, p - 2);
  reverse(a.begin() + 1, a.end());
  for (int i = 0; i < n; ++i) a[i] = a[i] * ni % p;
}
```

5.3 Miller Rabin

```
// n < 4759123141   chk = [2, 7, 61]
// n < 1122004669633 chk = [2, 13, 23, 1662803]
// n < 2^64      chk = [2, 325, 9375, 28178, 450775,
    9780504, 1795265022]

long long fpow(long long a, long long n, long long mod)
     {
  long long ret = 1LL;
  for (; n; n >>= 1) {
    if (n & 1) ret = (__int128)ret * (__int128)a % mod;
    a = (__int128)a * (__int128)a % mod;
  }
  return ret;
}
bool check(long long a, long long u, long long n, int t
    ) {
  a = fpow(a, u, n);
  if (a == 0) return true;
  if (a == 1 || a == n - 1) return true;
  for (int i = 0; i < t; ++i) {
    a = (__int128)a * (__int128)a % n;
    if (a == 1) return false;
    if (a == n - 1) return true;
  }
  return false;
}
bool is_prime(long long n) {
  if (n < 2) return false;
  if (n % 2 == 0) return n == 2;
  long long u = n - 1; int t = 0;
  for (; u & 1; u >>= 1, ++t);
  for (long long i : chk) {
    if (!check(i, u, n, t)) return false;
  }
  return true;
}
```

6 Geometry

6.1 Basic

```
const double eps = 1e-8;
const double pi = acos(-1);

struct Point {
  double x, y;
  Point(double a = 0, double b = 0): x(a), y(b) {}
};

typedef Point Vector;

// L:ax+by+c=0
struct Line {
  double a, b, c, angle;
  Point p1, p2;
  Line() {}
  Line(Point s, Point e) {
    a = s.y - e.y, b = e.x - s.x;
    c = s.x * e.y - e.x * s.y;
    angle = atan2(e.y - s.y, e.x - s.x);
    p1 = s, p2 = e;
  }
};

struct Segment {
  Point s, e;
  Segment() {}
  Segment(Point a, Point b): s(a), e(b) {}
  Segment(double x1, double y1, double x2, double y2) {
    s = Point(x1, y1);
    e = Point(x2, y2);
  }
```

```
};

Vector operator+(Point a, Point b) { return Vector(a.x
    + b.x, a.y + b.y); }
Vector operator-(Point a, Point b) { return Vector(a.x
    - b.x, a.y - b.y); }
Vector operator*(Point a, double k) { return Vector(a.x
     * k, a.y * k); }
Vector operator/(Point a, double k) { return Vector(a.x
     / k, a.y / k); }
double len(Vector a) { return sqrt(a.x * a.x + a.y * a.
    y); }

// <0 when ep at opsp clockwise
double Cross(Point &sp, Point &ep, Point &op) { return
    (sp.x - op.x) * (ep.y - op.y) - (ep.x - op.x) * (sp
    .y - op.y); }
double Cross(Vector a, Vector b) { return a.x * b.y - b
    .x * a.y; }
double Dot(Vector a, Vector b) { return a.x * b.x + a.y
     * b.y; }

int epssgn(double x) {
  if (fabs(x) < eps) return 0;
  else return x < 0 ? -1 : 1;
}

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

bool Parallel(Line l1, Line l2) { return fabs(l1.a * l2
    .b - l2.a * l1.b) < eps; }
bool LineEqual(Line l1, Line l2) { return Parallel(l1,
    l2) && fabs(l1.a * l2.c - l2.a * l1.c) < eps &&
    fabs(l1.b * l2.c - l2.b * l1.c) < eps; }

double PointToSegDist(Point A, Point B, Point C) {
  if (dis(A, B) < eps) return dis(B, C);
  if (epssgn(Dot(B - A, C - A)) < 0) return dis(A, C);
  if (epssgn(Dot(A - B, C - B)) < 0) return dis(B, C);
  return fabs(Cross(B - A, C - A)) / dis(B, A);
}

double TwoSegMinDist(Point A, Point B, Point C, Point D
    ) { return min(min(PointToSegDist(A, B, C),
    PointToSegDist(A, B, D)), min(PointToSegDist(C, D,
    A), PointToSegDist(C, D, B))); }

Point SymPoint(Point p, Line l) {
  Point result;
  double a = l.p2.x - l.p1.x;
  double b = l.p2.y - l.p1.y;
  double t = ((p.x - l.p1.x) * a + (p.y - l.p1.y) * b)
    / (a * a + b * b);
  result.x = 2 * l.p1.x + 2 * a * t - p.x;
  result.y = 2 * l.p1.y + 2 * b * t - p.y;
  return result;
}

// without end points: <= -> <
bool IsSegmentIntersect(Point s1, Point e1, Point s2,
    Point e2) {
  if (min(s1.x, e1.x) <= max(s2.x, e2.x) &&
    min(s1.y, e1.y) <= max(s2.y, e2.y) &&
    min(s2.x, e2.x) <= max(s1.x, e1.x) &&
    min(s2.y, e2.y) <= max(s1.y, e1.y) &&
    Cross(s2, e2, s1) * Cross(s2, e2, e1) <= 0 &&
    Cross(s1, e1, s2) * Cross(s1, e1, e2) <= 0) return
    1;
  return 0;
}

int IsLineIntersectSegment(Point p1, Point p2, Point s,
     Point e){ return !Cross(p1, p2, s) * Cross(p1, p2,
     e) > eps; }
int IsLineIntersectSegment(Line l1, Point s, Point e) {
     return !Cross(l1.p1, l1.p2, s) * Cross(l1.p1, l1.
    p2, e) > eps; }

Point GetIntersect(Line l1, Line l2) {
  Point res;
  res.x = (l1.b * l2.c - l2.b * l1.c) / (l1.a * l2.b -
    l2.a * l1.b);
  res.y = (l1.c * l2.a - l2.c * l1.a) / (l1.a * l2.b -
    l2.a * l1.b);
  return res;
}
```

6.2 Triangle Center

```
Point TriangleCircumCenter(Point a, Point b, Point c) {
  Point res;
  double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
  double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
  double ax = (a.x + b.x) / 2;
  double ay = (a.y + b.y) / 2;
  double bx = (c.x + b.x) / 2;
  double by = (c.y + b.y) / 2;
  double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay
    )) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
  return Point(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Point TriangleMassCenter(Point a, Point b, Point c) {
  return (a + b + c) / 3.0;
}

Point TriangleOrthoCenter(Point a, Point b, Point c) {
  return TriangleMassCenter(a, b, c) * 3.0 -
    TriangleCircumCenter(a, b, c) * 2.0;
}

Point TriangleInnerCenter(Point a, Point b, Point c) {
  Point res;
  double la = len(b - c);
  double lb = len(a - c);
  double lc = len(a - b);
  res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb +
     lc);
  res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb +
     lc);
  return res;
}
```

6.3 Sector Area

```
// calc area of sector which include a, b
double SectorArea(Point a, Point b, double r) {
  double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
  while (theta <= 0) theta += 2 * pi;
  while (theta >= 2 * pi) theta -= 2 * pi;
  theta = min(theta, 2 * pi - theta);
  return r * r * theta / 2;
}
```

6.4 Polygon Area

```
// point sort in counterclockwise
double ConvexPolygonArea(vector<Point> &p, int n) {
  double area = 0;
  for (int i = 1; i < p.size() - 1; i++) area += Cross(
    p[i] - p[0], p[i + 1] - p[0]);
  return area / 2;
}
```

6.5 Half Plane Intersection

```
int cmp(const Line &l1, const Line &l2) {
  int d = epssgn(l1.angle - l2.angle);
  if (!d) return (epssgn(Cross(l2.p1 - l1.p1, l2.p2 -
    l1.p1)) > 0);
  return d < 0;
}
```

```
void QSort(Line L[], int l, int r) {
  int i = l, j = r;
  Line swap, mid = L[(l+r) / 2];
  while (i <= j) {
    while (cmp(L[i], mid)) ++i;
    while (cmp(mid, L[j])) --j;
    if (i <= j) {
      swap = L[i];
      L[i] = L[j];
      L[j] = swap;
      ++i, --j;
    }
  }
  if (i < r) QSort(L, i, r);
  if (l < j) QSort(L, l, j);
}

int IntersectionOutOfHalfPlane(Line &hpl, Line &l1,
    Line &l2) {
   Point p = GetIntersect(l1, l2);
   return epssgn(Cross(hpl.p1 - p, hpl.p2 - p)) < 0;
}

// move hpl for dis
Line HalfPlaneMoveIn(Line &hpl, double &dis) {
  double dx = hpl.p1.x - hpl.p2.x;
  double dy = hpl.p1.y - hpl.p2.y;
  double ll = len(hpl.p1 - hpl.p2);
  Point pa = Point(dis * dy / ll + hpl.p1.x, hpl.p1.y -
     dis * dx / ll);
  Point pb = Point(dis * dy / ll + hpl.p2.x, hpl.p2.y -
     dis * dx / ll);
  return Line(pa, pb);
}

// get intersect of n halfplane l, intersect point in p
void HalfPlaneIntersect(Line l[], int n, Point p[], int
     &pn) {
  int i, j;
  int dq[maxn], top = 1, bot = 0;
  deque<int> dq;
  QSort(l, 0, n-1);
  for (i = j = 0; i < n; i++) if (epssgn(l[i].angle - l
    [j].angle) > 0) l[++j] = l[i];
  n = j + 1;
  dq.push_back(0); dq.push_back(1);
  for(i = 2; i < n; i++) {
    while (dq.size() >= 2 && IntersectionOutOfHalfPlane
    (l[i], l[dq[dq.size() - 1]], l[dq[dq.size() - 2]]))
     dq.pop_back();
    while (dq.size() >= 2 && IntersectionOutOfHalfPlane
    (l[i], l[dq[0]], l[dq[1]])) dq.pop_front();
    dq.push_back(i);
  }
  while (dq.size() >= 2 && IntersectionOutOfHalfPlane(l
    [dq[0]], l[dq[dq.size() - 1]], l[dq[dq.size() -
    2]])) dq.pop_back();
  while (dq.size() >= 2 && IntersectionOutOfHalfPlane(l
    [dq[dq.size() - 1]], l[dq[dq[0]]], l[dq[dq[1]]]))
    dq.pop_front();
  dq.push_back(dq.front());
  for (pn = 0, i = 0; i < dq.size() - 1; ++i, ++pn) p[
    pn] = GetIntersect(l[dq[i + 1]], l[dq[i]]);
}
```

6.6 Polygon Center

```
Point BaryCenter(vector<Point> &p, int n) {
  Point res(0, 0);
  double s = 0.0, t;
  for (int i = 1; i < p.size() - 1; i++) {
    t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
    s += t;
    res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
    res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
  }
  res.x /= (3 * s);
  res.y /= (3 * s);
  return res;
}
```

6.7 Maximum Triangle

```
double ConvexHullMaxTriangleArea(Point p[], int res[],
    int chnum) {
  double area = 0, tmp;
  res[chnum] = res[0];
  for (int i = 0, j = 1, k = 2; i < chnum; i++) {
    while (fabs(Cross(p[res[j]] - p[res[i]], p[res[(k +
     1) % chnum]] - p[res[i]])) > fabs(Cross(p[res[j]]
    - p[res[i]], p[res[k]] - p[res[i]]))) k = (k + 1) %
     chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] -
     p[res[i]]));
    if (tmp > area) area = tmp;
    while (fabs(Cross(p[res[(j + 1) % chnum]] - p[res[i
    ]], p[res[k]] - p[res[i]])) > fabs(Cross(p[res[j]]
    - p[res[i]], p[res[k]] - p[res[i]]))) j = (j + 1) %
     chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] -
     p[res[i]]));
    if (tmp > area) area = tmp;
  }
  return area / 2;
}
```

6.8 Point in Polygon

```
bool PointInConvexHull(Point p[], int res[], int chnum,
     Point x) {
  Point g = (p[res[0]] + p[res[chnum / 3]] + p[res[2 *
    chnum / 3]]) / 3.0;
  int l = 0, r = chnum, mid;
  while (l + 1 < r) {
    mid = (l + r) >> 1;
    if (epssgn(Cross(p[res[l]] - g, p[res[mid]] - g)) >
     0) {
      if (epssgn(Cross(p[res[l]] - g, x - g)) >=0 &&
    epssgn(Cross(p[res[mid]] - g ,x - g)) < 0) r = mid;
      else l = mid;
    } else {
      if (epssgn(Cross(p[res[l]] - g, x - g)) < 0 &&
    epssgn(Cross(p[res[mid]] - g, x - g)) >=0 ) l = mid
    ;
      else r = mid;
    }
  }
  r %= chnum;
  return epssgn(Cross(p[res[r]] - x, p[res[l]] - x)) ==
     -1;
}
```

6.9 Circle-Line Intersection

```
// remove second level if to get points for line (
    defalut: segment)
void CircleCrossLine(Point a, Point b, Point o, double
    r, Point ret[], int &num) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2 - x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 -
    y0) - r * r;
  double delta = B * B - 4 * A * C;
  num = 0;
  if (epssgn(delta) >= 0) {
    double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
    double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
    if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret[
    num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
    if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret[
    num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
  }
}
```

```
vector<Point> CircleCrossLine(Point a, Point b, Point o
    , double r) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2- x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 -
    y0) - r * r;
  double delta = B * B - 4 * A * C;
  vector<Point> ret;
  if (epssgn(delta) >=0){
    double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
    double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
    if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret.
    emplace_back(x1 + t1 * dx, y1 + t1 * dy);
    if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret.
    emplace_back(x1 + t2 * dx, y1 + t2 * dy);
  }
  return ret;
}
```

6.10 Circle-Triangle Intersection

```
// calc area intersect by circle with radius r and
    triangle OAB
double Calc(Point a, Point b, double r) {
  Point p[2];
  int num = 0;
  bool ina = epssgn(len(a) - r) < 0, inb = epssgn(len(b
    ) - r) < 0;
  if (ina) {
    if (inb) return fabs(Cross(a, b)) / 2.0; //
    triangle in circle
    else { // a point inside and another outside: calc
    sector and triangle area
      CircleCrossLine(a, b, Point(0, 0), r, p, num);
      return SectorArea(b, p[0], r) + fabs(Cross(a, p
    [0])) / 2.0;
    }
  } else {
    CircleCrossLine(a, b, Point(0, 0), r, p, num);
    if (inb) return SectorArea(p[0], a, r) + fabs(Cross
    (p[0], b)) / 2.0;
    else {
      if (num == 2) return SectorArea(a, p[0], r) +
    SectorArea(p[1], b, r) + fabs(Cross(p[0], p[1])) /
    2.0; // segment ab has 2 point intersect with
    circle
      else return SectorArea(a, b, r); // segment has
    no intersect point with circle
    }
  }
}
```

6.11 Polygon Diameter

```
// get diameter of p[res[]] store opposite points in
    app
double Diameter(Point p[], int res[], int chnum, int
    app[][2], int &appnum) {
  double ret = 0, nowlen;
  res[chnum] = res[0];
  appnum = 0;
  for (int i = 0, j = 1; i < chnum; ++i) {
    while (Cross(p[res[i]] - p[res[i + 1]], p[res[j +
    1]] - p[res[i + 1]]) < Cross(p[res[i]] - p[res[i +
    1]], p[res[j]] - p[res[i + 1]])) {
      ++j;
      j %= chnum;
    }
    app[appnum][0] = res[i];
    app[appnum][1] = res[j];
    ++appnum;
    nowlen = dis(p[res[i]], p[res[j]]);
    if (nowlen > ret) ret = nowlen;
    nowlen = dis(p[res[i + 1]], p[res[j + 1]]);
    if (nowlen > ret) ret = nowlen;
  }
  return ret;
}
```

6.12 Minimun Distance of 2 Polygons

```
// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n
    , int m) {
  int YMinP = 0, YMaxQ = 0;
  double tmp, ans = 999999999;
  for (i = 0; i < n; ++i) if(P[i].y < P[YMinP].y) YMinP
     = i;
  for (i = 0; i < m; ++i) if(Q[i].y > Q[YMaxQ].y) YMaxQ
     = i;
  P[n] = P[0], Q[m] = Q[0];
  for (int i = 0; i < n; ++i) {
    while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[
    YMinP] - P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP +
     1], P[YMinP] - P[YMinP + 1])) YMaxQ = (YMaxQ + 1)
    % m;
    if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP
    ], P[YMinP + 1], Q[YMaxQ]));
    else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP
     + 1], Q[YMaxQ], Q[YMaxQ + 1]));
    YMinP = (YMinP + 1) % n;
  }
  return ans;
}
```

6.13 Convex Hull

```
int Graham(Point p[], int n, int res[]) {
  int len, top;
  top = 1;
  sort(p, p + n, [](const Point &a, const Point &b) {
      return a.y == b.y ? a.x < b.x : a.y < b.y; }
  // QSort(p,0,n-1);
  for (int i = 0; i < 3; i++) res[i] = i;
  for (int i = 2; i < n; i++) {
    while (top && epssgn(Cross(p[i], p[res[top]], p[res
    [top - 1]])) >= 0) top--;
    res[++top] = i;
  }
  len = top;
  res[++top] = n - 2;
  for (int i = n-3; i>=0; i--) {
    while (top != len && epssgn(Cross(p[i], p[res[top
    ]], p[res[top - 1]])) >= 0) top--;
    res[++top] = i;
  }
  return top;
}
```

6.14 Rotating Caliper

```
struct pnt {
  int x, y;
  pnt(): x(0), y(0) {};
  pnt(int xx, int yy): x(xx), y(yy) {};
} p[maxn];

pnt operator-(const pnt &a, const pnt &b) { return pnt(
    b.x - a.x, b.y - a.y); }
int operator^(const pnt &a, const pnt &b) { return a.x
    * b.y - a.y * b.x; } //cross
int operator*(const pnt &a, const pnt &b) { return (a -
     b).x * (a - b).x + (a - b).y * (a - b).y; } //
    distance
int tb[maxn], tbz, rsd;

int dist(int n1, int n2){
  return p[n1] * p[n2];
}
int cross(int t1, int t2, int n1){
```

```
  return (p[t2] - p[t1]) ^ (p[n1] - p[t1]);
}
bool cmpx(const pnt &a, const pnt &b) { return a.x == b
    .x ? a.y < b.y : a.x < b.x; }

void RotatingCaliper() {
  sort(p, p + n, cmpx);
  for (int i = 0; i < n; ++i) {
    while (tbz > 1 && cross(tb[tbz - 2], tb[tbz - 1], i
    ) <= 0) --tbz;
    tb[tbz++] = i;
  }
  rsd = tbz - 1;
  for (int i = n - 2; i >= 0; --i) {
    while (tbz > rsd + 1 && cross(tb[tbz - 2], tb[tbz -
     1], i) <= 0) --tbz;
    tb[tbz++] = i;
  }
  --tbz;
  int lpr = 0, rpr = rsd;
  // tb[lpr], tb[rpr]
  while (lpr < rsd || rpr < tbz - 1) {
    if (lpr < rsd && rpr < tbz - 1) {
      pnt rvt = p[tb[rpr + 1]] - p[tb[rpr]];
      pnt lvt = p[tb[lpr + 1]] - p[tb[lpr]];
      if ((lvt ^ rvt) < 0) ++lpr;
      else ++rpr;
    }
    else if (lpr == rsd) ++rpr;
    else ++lpr;
    // tb[lpr], tb[rpr]
  }
}
```