Contents

1 Basic

1.1 vimrc

```
syn on
colo desert
se ai nu ru mouse=a
se cin et ts=4 sw=4 sts=4
set backspace=indent,eol,start
inoremap {<ENTER> {<ENTER>}<UP><END><ENTER>
```

1.2 Fast Integer Input

```
#define getchar gtx

inline int gtx() {
  const int N = 1048576;
  static char buffer[N];
  static char *p = buffer, *end = buffer;
  if (p == end) {
    if ((end = buffer + fread(buffer, 1, N, stdin)) ==
    buffer) return EOF;
    p = buffer;
  }
  return *p++;
}

template <typename T>
inline bool rit(T& x) {
  char c = 0; bool flag = false;
  while (c = getchar(), (c < '0' && c != '-') || c > '9
    ') if (c == -1) return false;
  c == '-' ? (flag = true, x = 0) : (x = c - '0');
  while (c = getchar(), c >= '0' && c <= '9') x = x *
    10 + c - '0';
  if (flag) x = -x;
  return true;
}

template <typename T, typename ...Args>
inline bool rit(T& x, Args& ...args) { return rit(x) &&
     rit(args...); }
```

2 Flow

2.1 Dinic

```
struct dinic {
  static const int inf = 1e9;
  struct edge {
    int dest, cap, rev;
    edge(int d, int c, int r): dest(d), cap(c), rev(r)
    {}
  };
  vector<edge> g[maxn];
  int qu[maxn], ql, qr;
  int lev[maxn];
  void init() {
    for (int i = 0; i < maxn; ++i)
      g[i].clear();
  }
  void add_edge(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size() - 0);
    g[b].emplace_back(a, 0, g[a].size() - 1);
  }
  bool bfs(int s, int t) {
    memset(lev, -1, sizeof(lev));
    lev[s] = 0;
    ql = qr = 0;
    qu[qr++] = s;
    while (ql < qr) {
      int x = qu[ql++];
      for (edge &e : g[x]) if (lev[e.dest] == -1 && e.
    cap > 0) {
        lev[e.dest] = lev[x] + 1;
```

```
        qu[qr++] = e.dest;
      }
    }
    return lev[t] != -1;
  }
  int dfs(int x, int t, int flow) {
    if (x == t) return flow;
    int res = 0;
    for (edge &e : g[x]) if (e.cap > 0 && lev[e.dest]
    == lev[x] + 1) {
      int f = dfs(e.dest, t, min(e.cap, flow - res));
      res += f;
      e.cap -= f;
      g[e.dest][e.rev].cap += f;
    }
    if (res == 0) lev[x] = -1;
    return res;
  }
  int operator()(int s, int t) {
    int flow = 0;
    for (; bfs(s, t); flow += dfs(s, t, inf));
    return flow;
  }
};
```

2.2 ISAP

```
struct isap {
  static const int inf = 1e9;
  struct edge {
    int dest, cap, rev;
    edge(int a, int b, int c): dest(a), cap(b), rev(c)
    {}
  };
  vector<edge> g[maxn];
  int it[maxn], gap[maxn], d[maxn];
  void add_edge(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size() - 0);
    g[b].emplace_back(a, 0, g[a].size() - 1);
  }
  int dfs(int x, int t, int tot, int flow) {
    if (x == t) return flow;
    for (int &i = it[x]; i < g[x].size(); ++i) {
      edge &e = g[x][i];
      if (e.cap > 0 && d[e.dest] == d[x] - 1) {
        int f = dfs(e.dest, t, tot, min(flow, e.cap));
        if (f) {
          e.cap -= f;
          g[e.dest][e.rev].cap += f;
          return f;
        }
      }
    }
    if ((--gap[d[x]]) == 0) d[x] = tot;
    else d[x]++, it[x] = 0, ++gap[d[x]];
    return 0;
  }
  int operator()(int s, int t, int tot) {
    memset(it, 0, sizeof(it));
    memset(gap, 0, sizeof(gap));
    memset(d, 0, sizeof(d));
    int r = 0;
    gap[0] = tot;
    for (; d[s] < tot; r += dfs(s, t, tot, inf));
    return r;
  }
};
```

2.3 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);
  }
};
```

2.4 Hungarian ($O(n^3)$)

```
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;
  }
};
```

2.5 Hungarian ($O(n^4)$)

```
struct hungarian {
  static const int inf = 1e9;
  int lx[maxn], ly[maxn], w[maxn][maxn];
  int match[maxn];
  bool vx[maxn], vy[maxn];
  void init() {
    for (int i = 0; i < maxn; ++i) for (int j = 0; j <
    maxn; ++j) w[i][j] = -inf;
    for (int i = 0; i < maxn; ++i) w[i][i] = 0;
  }
  void add_edge(int a, int b, int c) {
    w[a][b] = max(w[a][b], c);
  }
  bool dfs(int now) {
    vx[now] = true;
    for (int i = 0; i < maxn; ++i) if (lx[now] + ly[i]
    == w[now][i] && !vy[i]) {
      vy[i] = true;
      if (!match[i] || dfs(match[i])) {
        match[i] = now;
        return true;
      }
    }
    return false;
  }
  void relabel() {
    int dlt = inf;
    for (int i = 0; i < maxn; ++i) if (vx[i]) {
      for (int j = 0; j < maxn; ++j) if (!vy[j]) dlt =
    min(dlt, lx[i] + ly[j] - w[i][j]);
    }
    for (int i = 0; i < maxn; ++i) if (vx[i]) lx[i] -=
    dlt;
    for (int i = 0; i < maxn; ++i) if (vy[i]) ly[i] +=
    dlt;
  }
  int operator()() {
    fill(lx, lx + maxn, -inf); fill(ly, ly + maxn, 0);
    for (int i = 0; i < maxn; ++i) {
      for (int j = 0; j < maxn; ++j) lx[i] = max(lx[i],
     w[i][j]);
    }
    memset(match, 0, sizeof(match));
    for (int i = 0; i < maxn; ++i) {
      while (true) {
        memset(vx, false, sizeof(vx));
        memset(vy, false, sizeof(vy));
        if (dfs(i)) break;
        relabel();
      }
    }
    int r = 0;
    for (int i = 0; i < maxn; ++i) if (w[match[i]][i] >
     0) r += w[match[i]][i];
    return r;
  }
};
```

3 Data Structure

3.1 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;
```

3.2 <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;
}
```

4 Graph

4.1 Link-Cut Tree

```
struct node {
  node *ch[2], *fa, *pfa;
  int sum, v, rev;
  node(int s): v(s), sum(s), rev(0), fa(nullptr), pfa(
    nullptr) {
    ch[0] = nullptr;
    ch[1] = nullptr;
  }
  int relation() {
    return this == fa->ch[0] ? 0 : 1;
  }
  void push() {
    if (!rev) return;
    swap(ch[0], ch[1]);
    if (ch[0]) ch[0]->rev ^= 1;
    if (ch[1]) ch[1]->rev ^= 1;
    rev = 0;
  }
  void pull() {
    sum = v;
    if (ch[0]) sum += ch[0]->sum;
    if (ch[1]) sum += ch[1]->sum;
  }
  void rotate() {
    if (fa->fa) fa->fa->push();
    fa->push(), push();
    swap(pfa, fa->pfa);
    int d = relation();
    node *t = fa;
    if (t->fa) t->fa->ch[t->relation()] = this;
    fa = t->fa;
    t->ch[d] = ch[d ^ 1];
    if (ch[d ^ 1]) ch[d ^ 1]->fa = t;
    ch[d ^ 1] = t;
    t->fa = this;
    t->pull(), pull();
  }
  void splay() {
    while (fa) {
      if (!fa->fa) {
        rotate();
        continue;
      }
      fa->fa->push(), fa->push();
      if (relation() == fa->relation()) fa->rotate(),
    rotate();
      else rotate(), rotate();
    }
  }
  void evert() {
    access();
    splay();
    rev ^= 1;
  }
  void expose() {
    splay(), push();
    if (ch[1]) {
      ch[1]->fa = nullptr;
      ch[1]->pfa = this;
      ch[1] = nullptr;
      pull();
    }
  }
  bool splice() {
    splay();
    if (!pfa) return false;
    pfa->expose();
    pfa->ch[1] = this;
    fa = pfa;
    pfa = nullptr;
    fa->pull();
    return true;
  }
  void access() {
    expose();
    while (splice());
  }
  int query() {
    return sum;
  }
};

namespace lct {
  node *sp[maxn];
  void make(int u, int v) {
    // create node with id u and value v
    sp[u] = new node(v, u);
  }
  void link(int u, int v) {
    // u become v's parent
    sp[v]->evert();
    sp[v]->pfa = sp[u];
  }
  void cut(int u, int v) {
    // u was v's parent
    sp[u]->evert();
    sp[v]->access(), sp[v]->splay(), sp[v]->push();
    sp[v]->ch[0]->fa = nullptr;
    sp[v]->ch[0] = nullptr;
    sp[v]->pull();
  }
  void modify(int u, int v) {
    sp[u]->splay();
    sp[u]->v = v;
    sp[u]->pull();
  }
  int query(int u, int v) {
    sp[u]->evert(), sp[v]->access(), sp[v]->splay();
    return sp[v]->query();
  }
}
```

4.2 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;
  }
};
```

4.3 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);
  }
}
```

4.4 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;
    }
  }
};
```

4.5 Tarjan's articulation point

```
vector<pair<int, int>> g[maxn];
int low[maxn], tin[maxn], t;
int bcc[maxn], sz;
int a[maxn], b[maxn], deg[maxn];
bool cut[maxn], ins[maxn];
```

```
vector<int> ed[maxn];

stack<int> st;

void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  int ch = 0;
  for (auto u : g[x]) if (u.first != p) {
    if (!ins[u.second]) st.push(u.second), ins[u.second
    ] = true;
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    ++ch;
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] >= tin[x]) {
      cut[x] = true;
      ++sz;
      while (true) {
        int e = st.top(); st.pop();
        bcc[e] = sz;
        if (e == u.second) break;
      }
    }
  }
  if (ch == 1 && p == -1) cut[x] = false;
}
```

4.6 Tarjan's bridge

```
vector<pair<int, int>> g[maxn];
int tin[maxn], low[maxn], t;
int a[maxn], b[maxn];
int bcc[maxn], sz;
bool br[maxn];

stack<int> st;

void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  st.push(x);
  for (auto u : g[x]) if (u.first != p) {
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] == tin[u.first]) br[u.second] =
    true;
  }
  if (tin[x] == low[x]) {
    ++sz;
    while (st.size()) {
      int u = st.top(); st.pop();
      bcc[u] = sz;
      if (u == x) break;
    }
  }
}
```

5 String

5.1 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;
}
```

5.2 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];
    }
  }
}
```

5.3 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;
}
```

5.4 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]];
  }
};
```

5.5 Suffix Array

```
struct SuffixArray {
  int sa[maxn], tmp[2][maxn], c[maxn], _lcp[maxn], r[
    maxn], n;
  string s;
  SparseTable st;
  void suffixarray() {
     int* rank = tmp[0];
    int* nRank = tmp[1];
    int A = 128;
    for (int i = 0; i < A; ++i) c[i] = 0;
    for (int i = 0; i < s.length(); ++i) c[rank[i] = s[
    i]]++;
    for (int i = 1; i < A; ++i) c[i] += c[i - 1];
    for (int i = s.length() - 1; i >= 0; --i) sa[--c[s[
    i]]] = i;
    for (int n = 1; n < s.length(); n *= 2) {
      for (int i = 0; i < A; ++i) c[i] = 0;
      for (int i = 0; i < s.length(); ++i) c[rank[i
    ]]++;
      for (int i = 1; i < A; ++i) c[i] += c[i - 1];
      int* sa2 = nRank;
      int r = 0;
      for (int i = s.length() - n; i < s.length(); ++i)
     sa2[r++] = i;
      for (int i = 0; i < s.length(); ++i) if (sa[i] >=
     n) sa2[r++] = sa[i] - n;
      for (int i = s.length() - 1; i >= 0; --i) sa[--c[
    rank[sa2[i]]]] = sa2[i];
      nRank[sa[0]] = r = 0;
      for (int i = 1; i < s.length(); ++i) {
        if (!(rank[sa[i - 1]] == rank[sa[i]] && sa[i -
    1] + n < s.length() && rank[sa[i - 1] + n] == rank[
    sa[i] + n])) r++;
        nRank[sa[i]] = r;
      }
      swap(rank, nRank);
      if (r == s.length() - 1) break;
      A = r + 1;
    }
  }
  void solve() {
    suffixarray();
    for (int i = 0; i < n; ++i) r[sa[i]] = i;
    int ind = 0; _lcp[0] = 0;
    for (int i = 0; i < n; ++i) {
      if (!r[i]) { ind = 0; continue; }
      while (i + ind < n && s[i + ind] == s[sa[r[i] -
    1] + ind]) ++ind;
      _lcp[r[i]] = ind ? ind-- : 0;
    }
    st = SparseTable(n, _lcp);
  }
  int lcp(int L, int R) {
    if (L == R) return n - L - 1;
    L = r[L]; R = r[R];
    if (L > R) swap(L, R);
    ++L;
    return st.query(L, R);
  }
  SuffixArray(string s): s(s), n(s.length()) {}
  SuffixArray() {}
};
```

5.6 SAIS

```
namespace SAIS {
  enum type { L, S, LMS };
  const int maxn = 1e5 + 5;
  int bkt[maxn], cnt[maxn], lptr[maxn], rptr[maxn],
    tptr[maxn];
  int rev[maxn];
  void pre(const vector<int> &s, int sigma) {
    fill(bkt, bkt + s.size(), -1);
    fill(cnt, cnt + sigma, 0);
    for (int i = 0; i < s.size(); ++i) ++cnt[s[i]];
    int last = 0;
    for (int i = 0; i < sigma; ++i) {
      lptr[i] = last;
      last += cnt[i];
      rptr[i] = tptr[i] = last - 1;
    }
  }
  void induce(const vector<int> &s, const vector<type>
    &v) {
    for (int i = 0; i < s.size(); ++i) if (bkt[i] > 0)
    {
      if (v[bkt[i] - 1] == L) bkt[lptr[s[bkt[i] -
    1]]++] = bkt[i] - 1;
    }
    for (int i = s.size() - 1; i >= 0; --i) if (bkt[i]
    > 0) {
      if (v[bkt[i] - 1] != L) bkt[rptr[s[bkt[i] -
    1]]--] = bkt[i] - 1;
    }
  }
  bool equal(int l, int r, const vector<int> &s, const
    vector<type> &v) {
    do { if (s[l] != s[r]) return false; ++l, ++r; }
    while (v[l] != LMS && v[r] != LMS);
    return s[l] == s[r];
  }
  vector<int> radix_sort(const vector<int> &lms, const
    vector<int> &s, const vector<type> &v, int sigma) {
    pre(s, sigma);
    for (int i = 0; i < lms.size(); ++i) bkt[tptr[s[lms
    [i]]]--] = lms[i];
    induce(s, v);
    vector<int> rt(lms.size());
    for (int i = 0; i < lms.size(); ++i) rev[lms[i]] =
    i;
    int prv = -1, rnk = 0;
    for (int i = 0; i < s.size(); ++i) {
      int x = bkt[i];
      if (v[x] != LMS) continue;
      if (prv == -1) {
        rt[rev[x]] = rnk;
        prv = x;
        continue;
      }
      if (!equal(prv, x, s, v)) ++rnk;
      rt[rev[x]] = rnk;
      prv = x;
    }
    return rt;
  }
  vector<int> counting_sort(const vector<int> &s) {
    vector<int> o(s.size());
    for (int i = 0; i < s.size(); ++i) o[s[i]] = i;
    return o;
  }
  vector<int> reconstruct(const vector<int> &sa, const
    vector<int> &s, const vector<type> &v) {
    vector<int> pos;
```

```
    for (int i = 0; i < s.size(); ++i) if (v[i] == LMS)
     pos.push_back(i);
    vector<int> rev(sa.size());
    for (int i = 0; i < sa.size(); ++i) rev[i] = pos[sa
    [i]];
    return rev;
  }
  vector<int> sais(const vector<int> &s, int sigma) {
    vector<type> v(s.size());
    v[s.size() - 1] = S;
    for (int i = s.size() - 2; i >= 0; --i) {
      if (s[i] < s[i + 1] || s[i] == s[i + 1] && v[i +
    1] == S) v[i] = S;
      else v[i] = L;
    }
    for (int i = s.size() - 1; i >= 1; --i) {
      if (v[i] == S && v[i - 1] == L) v[i] = LMS;
    }
    vector<int> lms;
    for (int i = 0; i < s.size(); ++i) if (v[i] == LMS)
     lms.push_back(i);
    vector<int> r = radix_sort(lms, s, v, sigma);
    vector<int> sa;
    if (*max_element(r.begin(), r.end()) == r.size() -
    1) sa = counting_sort(r);
    else sa = sais(r, *max_element(r.begin(), r.end())
    + 1);
    sa = reconstruct(sa, s, v);
    pre(s, sigma);
    for (int i = sa.size() - 1; i >= 0; --i) bkt[tptr[s
    [sa[i]]]--] = sa[i];
    induce(s, v);
    return vector<int>(bkt, bkt + s.size());
  }
  vector<int> build(const string &s) {
    vector<int> v(s.size() + 1);
    for (int i = 0; i < s.size(); ++i) v[i] = s[i];
    v[v.size() - 1] = 0;
    vector<int> sa = sais(v, 256);
    return vector<int>(sa.begin() + 1, sa.end());
  }
}
```

5.7 DC3

```
namespace DC3{
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-compare"

#define SG(v,i) ((i)>=int(v.size())?0:v[i])
  inline bool smaller(int a, int b, vector<int> &r){
    if(SG(r,a+0) != SG(r,b+0)) return SG(r,a+0)<SG(r,b
    +0);
    if(SG(r,a+1) != SG(r,b+1)) return SG(r,a+1)<SG(r,b
    +1);
    return SG(r,a+2)<SG(r,b+2);
  }

  int cc[100005];
  inline vector<int> sort(vector<int> &r, int o, vector
    <int> &ix, int m){
    vector<int> rt(ix.size());
    for(int z=0;z<o;++z) r.push_back(0);
    for(int i=0;i<=m;++i) cc[i] = 0;
    for(int i=0;i<ix.size();++i) ++cc[r[ix[i]+o]];
    for(int i=0;i<=m;++i) cc[i+1] += cc[i];
    for(int i=ix.size()-1;i>=0;--i) rt[--cc[r[ix[i]+o
    ]]] = ix[i];
    for(int z=0;z<o;++z) r.pop_back();
    return rt;
  }

  vector<int> dc3(vector<int> &v, int n, int m){
    int c1 = (n+1)/3;
    vector<int> i12;
    for(int i=0;i<n;++i){
      if(i%3==0)continue;
      i12.push_back(i);
    }
    i12 = sort(v, 2, i12, m);
    i12 = sort(v, 1, i12, m);
    i12 = sort(v, 0, i12, m);

    int nr = 1;
    vector<int> r12(i12.size());
#define GRI(x) ((x)/3 + ((x)%3==2?c1:0))
    r12[GRI(i12[0])] = 1;
    for(int i=1;i<i12.size();++i){
      if(smaller(i12[i-1], i12[i], v)) r12[GRI(i12[i])]
       = ++nr;
      else r12[GRI(i12[i])] = nr;
    }

#define GEI(x) ((x)<c1?(x)*3+1:(x-c1)*3+2)
    if(nr != i12.size()){
      i12 = dc3(r12, i12.size(), nr);

      for(int i=0;i<i12.size();++i) r12[i12[i]] = i+1;
      for(int &i: i12) i = GEI(i);
    }

    vector<int> i0;
    if(n%3==1) i0.push_back(n-1);
    for(int i=0;i<i12.size();++i) if(i12[i]%3 == 1) i0.
    push_back(i12[i]-1);
    i0 = sort(v, 0, i0, m);

    vector<int> ret(v.size());
    int ptr12=0, ptr0=0, ptr=0;
    while(ptr12<i12.size() && ptr0<i0.size()){
      if(i12[ptr12]%3 == 1){
        if([&](int i, int j) -> bool{
          if(SG(v,i) != SG(v,j)) return SG(v,i)<SG(v,j)
    ;
          return SG(r12,GRI(i+1))<SG(r12,GRI(j+1));
        }(i12[ptr12], i0[ptr0]))ret[ptr++] = i12[ptr12
    ++];
        else ret[ptr++] = i0[ptr0++];
      }
      else{
        if([&](int i, int j) -> bool{
          if(SG(v,i+0) != SG(v,j+0)) return SG(v,i+0)<
    SG(v,j+0);
          if(SG(v,i+1) != SG(v,j+1)) return SG(v,i+1)<
    SG(v,j+1);
          return SG(r12,GRI(i+2))<SG(r12,GRI(j+2));
        }(i12[ptr12], i0[ptr0]))ret[ptr++] = i12[ptr12
    ++];
        else ret[ptr++] = i0[ptr0++];
      }
    }
    while(ptr12<i12.size()) ret[ptr++] = i12[ptr12++];
    while(ptr0<i0.size()) ret[ptr++] = i0[ptr0++];

    return ret;
  }
  vector<int> build(string str){
    vector<int> val(str.size()+1, 0);
    for(int i=0;i<str.size();++i) val[i] = str[i];
    return dc3(val, val.size(), 255);
  }
#pragma GCC diagnostic pop
}
```

5.8 Smallest Rotation

```
string rotate(const string &s) {
  int n = s.length();
  string t = s + s;
  int i = 0, j = 1;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[i + k] == s[j + k]) ++k;
    if (s[i + k] <= s[j + k]) j += k + 1;
    else i += k + 1;
    if (i == j) ++j;
  }
  int pos = (i < n ? i : j);
  return s.substr(pos, n);
}
```

6 Math

6.1 Fast Fourier transform

```
const int maxn = 131072;
using cplx = complex<double>;
const cplx I = cplx(0, 1);
const double pi = acos(-1);
cplx omega[maxn + 1];

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

void bin(vector<cplx> &a, int n) {
  int lg;
  for (lg = 0; (1 << lg) < n; ++lg); --lg;
  vector<cplx> tmp(n);
  for (int i = 0; i < n; ++i) {
    int to = 0;
    for (int j = 0; (1 << j) < n; ++j) to |= (((i >> j)
     & 1) << (lg - j));
    tmp[to] = a[i];
  }
  for (int i = 0; i < n; ++i) a[i] = tmp[i];
}

void fft(vector<cplx> &a, int n) {
  bin(a, n);
  for (int step = 2; step <= n; step <<= 1) {
    int to = step >> 1;
    for (int i = 0; i < n; i += step) {
      for (int k = 0; k < to; ++k) {
        cplx x = a[i + to + k] * omega[maxn / step * k
    ];
        a[i + to + k] = a[i + k] - x;
        a[i + k] += x;
      }
    }
  }
}

void ifft(vector<cplx> &a, int n) {
  fft(a, n);
  reverse(a.begin() + 1, a.end());
  for (int i = 0; i < n; ++i) a[i] /= n;
}

vector<int> multiply(const vector<int> &a, const vector
    <int> &b, bool trim = false) {
  int d = 1;
  while (d < max(a.size(), b.size())) d <<= 1; d <<= 1;
  vector<cplx> pa(d), pb(d);
  for (int i = 0; i < a.size(); ++i) pa[i] = cplx(a[i],
     0);
  for (int i = 0; i < b.size(); ++i) pb[i] = cplx(b[i],
     0);
  fft(pa, d); fft(pb, d);
  for (int i = 0; i < d; ++i) pa[i] *= pb[i];
  ifft(pa, d);
  vector<int> r(d);
  for (int i = 0; i < d; ++i) r[i] = round(pa[i].real()
    );
  if (trim) while (r.size() && r.back() == 0) r.
    pop_back();
  return r;
}
```

6.2 Number theoretic transform

```
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;
}
```

6.3 Fast Walsh-Hadamard transform

```
void xorfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  xorfwt(v, l, m), xorfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = v[i] + v[j];
    v[j] = v[i] - v[j], v[i] = x;
  }
}

void xorifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = (v[i] + v[j]) / 2;
    v[j] = (v[i] - v[j]) / 2, v[i] = x;
  }
  xorifwt(v, l, m), xorifwt(v, m, r);
}

void andfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andfwt(v, l, m), andfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] += v[j];
}

void andifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andifwt(v, l, m), andifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] -= v[j];
}

void orfwt(int v[], int l, int r) {
  if (r - l == 1) return;
```

```
  int m = l + r >> 1;
  orfwt(v, l, m), orfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] += v[i];
}

void orifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  orifwt(v, l, m), orifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] -= v[i];
}
```

6.4 Lagrange Interpolation

```
namespace lagrange {
  long long pf[maxn], nf[maxn];
  void init() {
    pf[0] = nf[0] = 1;
    for (int i = 1; i < maxn; ++i) {
      pf[i] = pf[i - 1] * i % mod;
      nf[i] = nf[i - 1] * (mod - i) % mod;
    }
  }
  // given y: value of f(a), a = [0, n], find f(x)
  long long solve(int n, vector<long long> y, long long
      x) {
    if (x <= n) return y[x];
    long long all = 1;
    for (int i = 0; i <= n; ++i) (all *= (x - i + mod))
     %= mod;
    long long ans = 0;
    for (int i = 0; i <= n; ++i) {
      long long z = all * fpow(x - i, -1) % mod;
      long long l = pf[i], r = nf[n - i];
      (ans += y[i] * z % mod * fpow(l * r, -1)) %= mod;
    }
    return ans;
  }
}
```

6.5 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]
//
vector<long long> chk = { 2, 325, 9375, 28178, 450775,
    9780504, 1795265022 };

long long fmul(long long a, long long n, long long mod)
     {
  long long ret = 0;
  for (; n; n >>= 1) {
    if (n & 1) (ret += a) %= mod;
    (a += a) %= mod;
  }
  return ret;
}

long long fpow(long long a, long long n, long long mod)
     {
  long long ret = 1LL;
  for (; n; n >>= 1) {
    if (n & 1) ret = fmul(ret, a, mod);
    a = fmul(a, 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 = fmul(a, 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;
}
```

6.6 Pollard's rho

```
long long f(long long x, long long n, int p) { return (
    fmul(x, x, n) + p) % n; }

map<long long, int> cnt;

void pollard_rho(long long n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2 == 0) return pollard_rho(n / 2), ++cnt[2],
    void();
  long long x = 2, y = 2, d = 1, p = 1;
  while (true) {
    if (d != n && d != 1) {
      pollard_rho(n / d);
      pollard_rho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p); y = f(f(y, n, p), n, p);
    d = __gcd(abs(x - y), n);
  }
}
```

6.7 Gaussian Elimination

```
void gauss(vector<vector<double>> &d) {
  int n = d.size(), m = d[0].size();
  for (int i = 0; i < m; ++i) {
    int p = -1;
    for (int j = i; j < n; ++j) {
      if (fabs(d[j][i]) < eps) continue;
      if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p =
     j;
    }
    if (p == -1) continue;
    for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      double z = d[j][i] / d[i][i];
      for (int k = 0; k < m; ++k) d[j][k] -= z * d[i][k
    ];
    }
  }
}
```

6.8 Linear Equations (full pivoting)

```
void linear_equation(vector<vector<double>> &d, vector<
    double> &aug, vector<double> &sol) {
  int n = d.size(), m = d[0].size();
  vector<int> r(n), c(m);
  iota(r.begin(), r.end(), 0);
  iota(c.begin(), c.end(), 0);
  for (int i = 0; i < m; ++i) {
    int p = -1, z = -1;
    for (int j = i; j < n; ++j) {
      for (int k = i; k < m; ++k) {
        if (fabs(d[r[j]][c[k]]) < eps) continue;
        if (p == -1 || fabs(d[r[j]][c[k]]) > fabs(d[r[p
    ]][c[z]])) p = j, z = k;
```

```
      }
    }
    if (p == -1) continue;
    swap(r[p], r[i]), swap(c[z], c[i]);
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      double z = d[r[j]][c[i]] / d[r[i]][c[i]];
      for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z *
    d[r[i]][c[k]];
      aug[r[j]] -= z * aug[r[i]];
    }
  }
  vector<vector<double>> fd(n, vector<double>(m));
  vector<double> faug(n), x(n);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j
    ]];
    faug[i] = aug[r[i]];
  }
  d = fd, aug = faug;
  for (int i = n - 1; i >= 0; --i) {
    double p = 0.0;
    for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j
    ];
    x[i] = (aug[i] - p) / d[i][i];
  }
  for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}
```

6.9 μ function

```
int mu[maxn], pi[maxn];
vector<int> prime;

void sieve() {
  mu[1] = pi[1] = 1;
  for (int i = 2; i < maxn; ++i) {
    if (!pi[i]) {
      pi[i] = i;
      prime.push_back(i);
      mu[i] = -1;
    }
    for (int j = 0; i * prime[j] < maxn; ++j) {
      pi[i * prime[j]] = prime[j];
      mu[i * prime[j]] = -mu[i];
      if (i % prime[j] == 0) {
        mu[i * prime[j]] = 0;
        break;
      }
    }
  }
}
```

6.10 $\lfloor \frac{n}{i} \rfloor$ Enumeration

```
vector<int> solve(int n) {
  vector<int> vec;
  for (int t = 1; t < n; t = (n / (n / (t + 1)))) vec.
    push_back(t);
  vec.push_back(n);
  vec.resize(unique(vec.begin(), vec.end()) - vec.begin
    ());
  return vec;
}
```

6.11 Extended GCD

```
template <typename T> tuple<T, T, T> extgcd(T a, T b) {
  if (!b) return make_tuple(a, 1, 0);
  T d, x, y;
  tie(d, x, y) = extgcd(b, a % b);
  return make_tuple(d, y, x - (a / b) * y);
}
```

6.12 Chinese remainder theorem

Given $x \equiv a_i \mod n_i \forall 1 \leq i \leq k$, where n_i are pairwise coprime, find x.
Let $N = \prod_{i=1}^{k} n_i$ and $N_i = N/n_i$, there exist integer M_i and m_i such that $M_i N_i + m_i n_i = 1$.
A solution to the system of congruence is $x = \sum_{i=1}^{k} a_i M_i N_i$.

6.13 Lucas's theorem

For non-negative integers m and n and prime p,
$\binom{m}{n} = \prod_{i=0}^{k} \binom{m_i}{n_i} \mod p$
where
$m = m_k p^k + m_{k-1} p^{k-1} + \ldots + m_1 p + m_0$,
$m = n_k p^k + n_{k-1} p^{k-1} + \ldots + n_1 p + n_0$.

6.14 Primes

97, 101, 131, 487, 593, 877, 1087, 1187, 1487, 1787, 3187, 12721, 13331, 14341, 75577, 123457, 222557, 556679, 999983, 1097774749, 1076767633, 100102021, 999997771, 1001010013, 1000512343, 987654361, 999991231, 999888733, 98789101, 987777733, 999991921, 1000000007, 1000000087, 1000000123, 1010101333, 1010102101, 1000000000039, 1000000000000037, 2305843009213693951, 4611686018427387847, 9223372036854775783, 18446744073709551557

7 Dynamic Programming

7.1 Convex Hull (monotone)

```
struct line {
  double a, b;
  inline double operator()(const double &x) const {
    return a * x + b; }
  inline bool checkfront(const line &l, const double &x
    ) const { return (*this)(x) < l(x); }
  inline double intersect(const line &l) const { return
     (l.b - b) / (a - l.a); }
  inline bool checkback(const line &l, const line &
    pivot) const { return pivot.intersect((*this)) <=
    pivot.intersect(l); }
};

void solve() {
  for (int i = 1; i < maxn; ++i) dp[0][i] = inf;
  for (int i = 1; i <= k; ++i) {
    deque<line> dq; dq.push_back((line){ 0.0, dp[i -
    1][0] });
    for (int j = 1; j <= n; ++j) {
      while (dq.size() >= 2 && dq[1].checkfront(dq[0],
    invt[j])) dq.pop_front();
      dp[i][j] = st[j] + dq.front()(invt[j]);
      line nl = (line){ -s[j], dp[i - 1][j] - st[j] + s
    [j] * invt[j] };
      while (dq.size() >= 2 && nl.checkback(dq[dq.size
    () - 1], dq[dq.size() - 2])) dq.pop_back();
      dq.push_back(nl);
    }
  }
}
```

7.2 Convex Hull (non-monotone)

```
struct line {
  int m, y;
  int l, r;
  line(int m = 0,int y = 0, int l = -5, int r =
    1000000009): m(m), y(y), l(l), r(r) {}
  int get(int x) const { return m * x + y; }
```

```
  int useful(line le) const {
    return (int)(get(l) >= le.get(l)) + (int)(get(r) >=
      le.get(r));
  }
};

int magic;
bool operator < (const line &a, const line &b) {
  if (magic) return a.m < b.m;
  return a.l < b.l;
}

set<line> st;

void addline(line l) {
  magic = 1;
  auto it = st.lower_bound(l);
  if (it != st.end() && it->useful(l) == 2) return;
  while (it != st.end() && it->useful(l) == 0) it = st.
    erase(it);
  if (it != st.end() && it->useful(l) == 1) {
    int L = it->l, R = it->r, M;
    while (R > L) {
      M = (L + R + 1) >> 1;
      if (it->get(M) >= l.get(M)) R = M - 1;
      else L = M;
    }
    line cp = *it;
    st.erase(it);
    cp.l = L + 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.r = L;
  }
  else if (it != st.end()) l.r = it->l - 1;
  it = st.lower_bound(l);
  while (it != st.begin() && prev(it)->useful(l) == 0)
    it = st.erase(prev(it));
  if (it != st.begin() && prev(it)->useful(l) == 1) {
    --it;
    int L = it->l, R = it->r, M;
    while (R > L) {
      M = (L + R) >> 1;
      if (it->get(M) >= l.get(M)) L = M + 1;
      else R = M;
    }
    line cp = *it;
    st.erase(it);
    cp.r = L - 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.l = L;
  }
  else if (it != st.begin()) l.l = prev(it)->r + 1;
  if (l.l <= l.r) st.insert(l);
}

int getval(int d) {
  magic = 0;
  return (--st.upper_bound(line(0, 0, d, 0)))->get(d);
}
```

7.3 $1D/1D$ Convex Optimization

```
struct segment {
  int i, l, r;
  segment() {}
  segment(int a, int b, int c): i(a), l(b), r(c) {}
};

inline long long f(int l, int r) {
  return dp[l] + w(l + 1, r);
}

void solve() {
  dp[0] = 0ll;
  deque&lt;segment&gt; deq; deq.push_back(segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
    dp[i] = f(deq.front().i, i);
    while (deq.size() && deq.front().r < i + 1) deq.
      pop_front();
    deq.front().l = i + 1;
    segment seg = segment(i, i + 1, n);
    while (deq.size() && df(i, deq.back().l) < df(deq.
    back().i, deq.back().l)) deq.pop_back();
    if (deq.size()) {
      int d = 1048576, c = deq.back().l;
      while (d >>= 1) if (c + d <= deq.back().r) {
        if (df(i, c + d) > df(deq.back().i, c + d)) c
    += d;
      }
      deq.back().r = c; seg.l = c + 1;
    }
    if (seg.l <= n) deq.push_back(seg);
  }
}
```

7.4 Conditon

7.4.1 concave totally monotone

$\forall i < i', j < j', B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$

7.4.2 convex totally monotone

$\forall i < i', j < j', B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$

7.4.3 concave monge condition

$\forall i < i', j < j', B[i][j] + B[i'][j'] \ge B[i][j'] + B[i'][j]$

7.4.4 convex monge condition

$\forall i < i', j < j', B[i][j] + B[i'][j'] \le B[i][j'] + B[i'][j]$

8 Geometry

8.1 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;
}
```

8.2 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;
}
```

8.3 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;
}
```

8.4 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;
}
```

8.5 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]]);
}
```

8.6 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;
}
```

8.7 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;
}
```

8.8 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;
}
```

8.9 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;
}
```

8.10 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
    }
  }
}
```

8.11 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;
}
```

8.12 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;
}
```

8.13 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;
}
```

8.14 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]
  }
}
```

8.15 Min Enclosing Circle

```
pt center(const pt &a, const pt &b, const pt &c) {
  pt p0 = b - a, p1 = c - a;
  double c1 = norm2(p0) * 0.5, c2 = norm2(p1) * 0.5;
  double d = p0 ^ p1;
  double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
  double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
  return pt(x, y);
}

circle min_enclosing(vector<pt> &p) {
  random_shuffle(p.begin(), p.end());
  double r = 0.0;
  pt cent;
  for (int i = 0; i < p.size(); ++i) {
    if (norm2(cent - p[i]) <= r) continue;
    cent = p[i];
    r = 0.0;
    for (int j = 0; j < i; ++j) {
      if (norm2(cent - p[j]) <= r) continue;
      cent = (p[i] + p[j]) / 2;
      r = norm2(p[j] - cent);
      for (int k = 0; k < j; ++k) {
        if (norm2(cent - p[k]) <= r) continue;
        cent = center(p[i], p[j], p[k]);
        r = norm2(p[k] - cent);
      }
    }
  }
  return circle(cent, sqrt(r));
}
```

8.16 Closest Pair

```
pt p[maxn];

double dis(const pt& a, const pt& b) {
  return sqrt((a - b) * (a - b));
}

double closest_pair(int l, int r) {
  if (l == r) return inf;
  if (r - l == 1) return dis(p[l], p[r]);
  int m = (l + r) >> 1;
  double d = min(closest_pair(l, m), closest_pair(m +
    1, r));
  vector<int> vec;
  for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d;
    --i) vec.push_back(i);
  for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) <
     d; ++i) vec.push_back(i);
  sort(vec.begin(), vec.end(), [=](const int& a, const
    int& b) { return p[a].y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
    for (int j = i + 1; j < vec.size() && fabs(p[vec[j
    ]].y - p[vec[i]].y) < d; ++j) {
      d = min(d, dis(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```