Yerevan State University ACM ICPC Team Notebook

Contents

1 Dynamic Programming

1.1 Convex Hull Trick

```
const int N = 100007;
const LD eps = 1e-9;
struct Line {
   LD a, b;
   Line(LD a = 0.0, LD b = 0.0) : a(a), b(b) {}
   bool operator < (const Line &o) const {
      return a > o.a;
   }
} st[N], mas[N];
int top, n;

inline LD cross(const Line &u, const Line &v) {
   return (v.b - u.b) / (u.a - v.a);
}

void add(const Line &L) {
   while (top > 1 && cross(st[top - 2], st[top - 1]) > cross(st[
         top - 1], L) - eps) {
      --top;
   }
   st[top++] = L;
}

LD query(LD x) {
   int l = 0;
   int r = top - 1;
   LD lx, rx;
   int pos = -1;
   while (true) {
      int m = (l + r) / 2;
      LD x1 = -1e18, x2 = 1e18;
      if (m != 0) {
         x1 = cross(st[m - 1], st[m]);
      }
      if (m != top - 1) {
         x2 = cross(st[m], st[m + 1]);
      }
      if (x > x1 - eps && x < x2 + eps) {
         pos = m;
         lx = x1;
         rx = x2;
         break;
      }
      else if (x < x1 - eps) {
         r = m - 1;
      }
      else {
         l = m + 1;
      }
   }
   assert(pos != -1);
   Line &L = st[pos];
   return L.a * x + L.b;
}
```

1.2 ConvexHull Hull Trick (Integers)

```
struct convex_hull {
    int top, pointer;

    struct uxix{
        long long k,b;
    } mas[N];

    void init() {
        top = 0;
        pointer = 0;
    }

    void add(long long k,long long b){
        k = -k; b = -b;
        uxix cur;
        cur.k=k;
        cur.b=b;
        while(top>=2){
            uxix u, v;
            u = mas[top-1];
            v = mas[top-2];
            assert(k >= u.k);
            assert(u.k >= v.k);
            if ((LD)(v.b - u.b) / (u.k - v.k) >  (LD)(u.b - b) /
                (k - u.k))
                --top;
            else
                break;
        }
        mas[top++] = cur;
    }
    /*
    // O(logN) general case
    long long get(long long x){
        if (top == 0) return INF;
        int ina=0,inb=top-1,answ;
        while(ina<=inb){
            int mid=(ina+inb)/2;
            long long p, q;
            if(mid==0) {
                p = -INF;
                q = 1;
            }
            else {
                p = mas[mid-1].b - mas[mid].b;
                q = mas[mid].k - mas[mid-1].k;
            }
            if((LD)p / q <= x){
                answ=mid;
                ina=mid+1;
            }
            else
                inb=mid-1;
        }
        long long pat=mas[answ].k * x + mas[answ].b;
        return -pat;
    }*/

    // Amortized O(1), if queries are sorted
    long long get(long long x){
        if (top == 0) return INF;
        pointer = min(pointer , top-1);
        while(pointer < top-1){
            long long p,q;
            q = mas[pointer+1].k - mas[pointer].k;
            p = mas[pointer].b - mas[pointer+1].b;
            if((LD)p / q <= x)
                ++pointer;
            else
                break;
        }
        long long pat=mas[pointer].k * x + mas[pointer].b;
        return -pat;
    }

} hull;
```

1.3 Divide and Conquer Optimization

```
// dp[i][j] = min_{k<j}{ dp[i-1][k] + cost(k,j) }
// if opt[i][j] <= opt[i][j+1]

int dp[N][N];

int cost(int l, int r) {
    // ...
}

void calcdp(int k, int l, int r, int optl, int optr) {
    if (l > r) return;

    int m = (l+r) / 2;
    int best = INF;
    int pos = -1;

    for (int j = optl; j <= min(m, optr); ++j) {
        int val = dp[k-1][j-1] + cost(j, m);
        if (val < best) {
            best = val;
            pos = j;
        }
    }

    dp[k][m] = best;
```

```
    calcdp(k, l, m-1, optl, pos);
    calcdp(k, m+1, r, pos, optr);
}

scanf("%d", &n);
for (int i = 1; i <= n; ++i) {
    scanf("%d", &a[i]);
}
}
// init
for (int k = 1; k <= n; ++k) {
    // init ?
    calcdp(k, 1, n, 1, n);
}
```

1.4 Knuth Optimization (Type 1)

```
// dp[i][j] = Min_{k < j}{dp[i-1][k] + cost(k, j)}
// if opt[i-1][j] <= opt[i][j] <= opt[i][j+1]

int dp[N][N];
int opt[N][N];

int cost(int l, int r) {
    // ...
}

void solve() {

    FOR(k, N) FOR(i, N) {
        dp[k][i] = INF;
    }
    dp[0][0] = 0;
    for (int i = 0; i <= n; ++i) {
        opt[0][i] = 1;
    }
    for (int k = 1; k <= n; ++k) {
        dp[k][0] = 0;
        opt[k][0] = 1;
        opt[k][n+1] = n;
        for (int i = n; i >= 1; --i) {
            for (int j = opt[k-1][i]; j <= opt[k][i+1]; ++j) {
                int val = dp[k-1][j-1] + cost(j, i);
                if (val < dp[k][i]) {
                    dp[k][i] = val;
                    opt[k][i] = j;
                }
            }
        }
    }
}
```

1.5 Knuth Optimization (Type 2)

```
// dp[i][j] = Min_{i < k < j} { dp[i][k] + dp[k+1][j] + cost(i, j
    ) }
// if opt[l][r-1] <= opt[l][r] <= opt[l+1][r]

LL dp[N][N];
int opt[N][N];


int cost(int l, int r) {
    // ...
}

void solve() {

    for (int l = n-1; l >= 0; --l) {
        dp[l][l] = 0;
        opt[l][l] = l;

        if (l+1 < n) {
            dp[l][l+1] = cost(l, l+1);
            opt[l][l+1] = l;
        }

        for (int r = l+2; r < n; ++r) {
            dp[l][r] = INF;

            int L = opt[l][r-1];
            int R = opt[l+1][r];

            for (int j = L; j <= R; ++j) {
                int val = dp[l][j] + dp[j+1][r] + getsum(l, r);
                if (val < dp[l][r]) {
                    dp[l][r] = val;
                    opt[l][r] = j;
                }
            }
        }
    }
}
```

2 Data Structures

2.1 2D Dynamic Segment Tree

```
struct intNode {
    intNode *l, *r;
    int tl, tr;
    LL g;

    intNode(int tl = 0, int tr = 0, LL g = 0, intNode *l = NULL,
          intNode *r = NULL):
        tl(tl), tr(tr), g(g), l(l), r(r) { }
};
typedef intNode* intPnode;

void intCreate(intPnode &T, int tl, int tr) {
    if (!T) T = new intNode(tl, tr);
}

LL intValue(intPnode &T) {
    return T == NULL ? 0 : T->g;
}

LL intGet(intPnode &T, int tl, int tr, int l, int r) {
    if (T == NULL) return 0;
    //intCreate(T, tl, tr); // eats more useless memory
    if (tl == l && tr == r) return intValue(T);
    int tm = (tl + tr) / 2;
    if (r <= tm) return intGet(T->l, tl, tm, l, r);
    if (l > tm) return intGet(T->r, tm + 1, tr, l, r);
    return gcd(intGet(T->l, tl, tm, l, tm), intGet(T->r, tm + 1,
          tr, tm + 1, r));
}

void intUpdate(intPnode &T, int tl, int tr, int pos, LL val, bool
      multipleLines = false, intPnode lson = NULL, intPnode rson
      = NULL) {
    intCreate(T, tl, tr);
    if (tl == tr) {
        if (!multipleLines) T->g = val;
        else {
            LL left = intGet(lson, 0, N, tl, tr);
            LL right = intGet(rson, 0, N, tl, tr);
            T->g = gcd(left, right);
        }
        return;
    }
    int tm = (tl + tr) / 2;
    if (pos <= tm) intUpdate(T->l, tl, tm, pos, val,
          multipleLines, lson, rson);
    else intUpdate(T->r, tm + 1, tr, pos, val, multipleLines,
          lson, rson);
    T->g = gcd(intValue(T->l), intValue(T->r));
}

struct extNode {
    extNode *l, *r;
    int tl, tr;
    intNode *root;

    extNode(int tl = 0, int tr = 0, intNode *root = NULL, extNode
          *l = NULL, extNode *r = NULL) :
        tl(tl), tr(tr), root(root), l(l), r(r) { }
};
typedef extNode* extPnode;
extPnode root;

void extCreate(extPnode &T, int tl, int tr) {
    if (!T) T = new extNode(tl, tr);
}

LL extGet(extPnode &T, int tl, int tr, int lx, int rx, int ly,
      int ry) {
    if (T == NULL) return 0;
    //extCreate(T, tl, tr); // eats more useless memory
    if (tl == lx && tr == rx) return intGet(T->root, 0, N, ly, ry
          );
    int tm = (tl + tr) / 2;
    if (rx <= tm) return extGet(T->l, tl, tm, lx, rx, ly, ry);
    if (lx > tm) return extGet(T->r, tm + 1, tr, lx, rx, ly, ry);
    return gcd(extGet(T->l, tl, tm, lx, tm, ly, ry), extGet(T->r,
          tm + 1, tr, tm + 1, rx, ly, ry));
}

void extUpdate(extPnode &T, int tl, int tr, int x, int y, LL val)
      {
    extCreate(T, tl, tr);
    if (tl == tr) {
        intUpdate(T->root, 0, N, y, val);
        return;
    }
    int tm = (tl + tr) / 2;
    if (x <= tm) extUpdate(T->l, tl, tm, x, y, val);
    else extUpdate(T->r, tm + 1, tr, x, y, val);
    intUpdate(T->root, 0, N, y, val, true, T->l ? T->l->root :
          NULL, T->r ? T->r->root : NULL);
}
```

2.2 Centroid Decomposition

```
const int N = 1000007;

struct CentroidDecomposition {
   vector<int> G[N];
   bool used[N];
   int size[N]; // subtree size
   int maxi[N]; // max subtree size

   struct node {
      int v;
      vector<node*> to;
      unordered_map<int, int> id; // child id of a vertex in
            subtree
      vector<int> nodes; // nodes in subtree

      // keep additional data here

      node(int v = 0) : v(v) {}
   };
   typedef node* pnode;
   pnode root;

   void dfs(int u, int par, vector<int> &mas) {
      mas.pb(u);
      size[u] = 1;
      maxi[u] = 1;
      for (int to : G[u]) {
         if (to != par && !used[to]) {
            dfs(to, u, mas);
            size[u] += size[to];
            maxi[u] = max(maxi[u], size[to]);
         }
      }
   }

   void buildData(pnode root) {
      root->nodes.pb(root->v);
      for (int i = 0; i < sz(root->to); ++i) {
         pnode to = root->to[i];
         for (int u : to->nodes) {
            root->nodes.pb(u);
            root->id[u] = i;
         }
      }
   }

   pnode build(int u) {
      vector<int> mas;
      dfs(u, -1, mas);
      int n = sz(mas);
      int best = N, pos = -1;
      for (int u : mas) {
         maxi[u] = max(maxi[u], n - size[u]);
         if (maxi[u] < best) {
            best = maxi[u];
            pos = u;
         }
      }
      u = pos;
      used[u] = true;
```

```
        node *root = new node(u);
        for (int to : G[u])
           if (!used[to]) root->to.pb(build(to));
        buildData(root);
        return root;
     }

} cd;


// build
for (int i = 0; i < n - 1; ++i) {
   int u, v;
   scanf("%d%d", &u, &v);
   cd.G[u].pb(v);
   cd.G[v].pb(u);
}
cd.root = cd.build(1);
```

2.3 DSU With Rollbacks

```
struct DSU_withRollbacks {
   int *par, *size;
   int n, ncomps;

   enum { PAR, SIZE, NCOMPS };

   struct Change {
      int type, index, oldValue;
      Change() {}
      Change(int type, int index, int oldValue) : type(type),
            index(index), oldValue(oldValue) {}
   } *changes;
   int top;

   void init(int n) {
      this->n = n;
      par = new int [n];
      size = new int [n];
      changes = new Change[2 * n];
      ncomps = n;
      top = 0;
      FOR(i, n) {
         par[i] = i;
         size[i] = 1;
      }
   }

   int get(int i) {
      if (par[i] == i) return i;
      return get(par[i]);
   }

   void unite(int i, int j) {
      i = get(i);
      j = get(j);
      if (i == j) return;
      if (size[i] < size[j]) swap(i, j);

      changes[top++] = Change(PAR, j, par[j]);
      changes[top++] = Change(SIZE, i, size[i]);
      changes[top++] = Change(NCOMPS, -1, ncomps);

      size[i] += size[j];
      par[j] = i;
      --ncomps;
   }

   int snapshot() {
      return top;
   }

   void toSnapshot(int snap) {
      while (top != snap) {
         --top;
         if (changes[top].type == PAR) {
            par[changes[top].index] = changes[top].oldValue;
         }
         if (changes[top].type == SIZE) {
            size[changes[top].index] = changes[top].oldValue;
         }
         if (changes[top].type == NCOMPS) {
            ncomps = changes[top].oldValue;
         }
      }
   }
};
```

2.4 Heavy-Light Decomposition

```
const int N = 100007;
int n, dep[N], par[N], size[N];
vector<int> G[N];
bool heavy[N];

vector<vector<int> > chains;
int id[N], pos[N];

void dfs(int u, int p = 0, int d = 0) {
   par[u] = p;
   dep[u] = d;
   size[u] = 1;
   int maxi = 0, pos = 0;
   for (int i = 0; i < sz(G[u]); ++i) {
      int to = G[u][i];
      if (to != p) {
         dfs(to, u, d + 1);
         if (size[to] > maxi) {
            maxi = size[to];
            pos = to;
         }
         size[u] += size[to];
      }
   }
   heavy[pos] = true;
}

int makeHLD(int u) {
   bool f = false;
   for (int i = 0; i < sz(G[u]); ++i) {
      int to = G[u][i].first;
      if (to == par[u]) continue;
      int ind = makeHLD(to);
      if (ind != -1) {
         chains[ind].pb(u);
         id[u] = ind;
         f = true;
      }
   }
   if (!f) {
      id[u] = sz(chains);
      chains.pb(vector<int>());
      chains[id[u]].pb(u);
   }
   if (heavy[u]) return id[u];
   return -1;
}

struct segTree {

   int n, *T;

   void init(int s) {
      n = s;
      T = new int[4 * s + 2];
      memset(T, 0, 4 * (4 * s + 2));
   }

   // ...
};
vector<segTree> S;

int getLca(int u, int v) {
   while (id[u] != id[v]) {
      if (dep[chains[id[u]][0]] < dep[chains[id[v]][0]]) v = par[
            chains[id[v]][0]];
      else u = par[chains[id[u]][0]];
   }
   return dep[u] < dep[v] ? u : v;
}

int go(int u, int v) {
   int ret = -1000000000;
   while (id[u] != id[v]) {
      ret = max(ret, S[id[u]].get(1, 0, S[id[u]].n - 1, 0, pos[u
            ]));
      u = par[chains[id[u]][0]];
   }
   if (u == v) return ret;
   ret = max(ret, S[id[u]].get(1, 0, S[id[u]].n - 1, pos[v] + 1,
         pos[u]));
   return ret;
}

// build
dfs(1);
chains.clear();
makeHLD(1);
for (int i = 0; i < sz(chains); ++i) reverse(all(chains[i]));
for (int i = 0; i < sz(chains); ++i)
  for (int j = 0; j < sz(chains[i]); ++j)
     pos[chains[i][j]] = dep[chains[i][j]] - dep[chains[i][0]];

S.clear();
S.resize(sz(chains));
for (int i = 0; i < sz(chains); ++i) {
  S[i].init(sz(chains[i]));
  for (int j = 0; j < S[i].n; ++j)
     S[i].update(1, 0, S[i].n - 1, j, w[chains[i][j]]);
}
```

2.5 Persistent Segment Tree With Range Updates

```
struct node
{
    node *l, *r;
    int sum, add;

    node(int sum = 0, int add = 0) : sum(sum), add(add), l(NULL),
           r(NULL) { } // creates a leaf

    node(node *left, node *right) // creates a node
    {
        add = 0;
        sum = 0;
        l = left;
        r = right;
    }
};
typedef node* pnode;

inline int get(pnode T, int tl, int tr)
{
    if (!T)
    {
        return 0;
    }
    return T->add * (tr - tl + 1) + T->sum;
}

inline void fix(pnode &T, int tl, int tr)
{
    int tm = (tl + tr) / 2;
    T->sum = get(T->l, tl, tm) + get(T->r, tm + 1, tr);
}

pnode push(pnode &T, int tl, int tr)
{
    pnode ret = new node();
    if (T->l)
    {
        ret->l = new node(T->l->l, T->l->r);
        ret->l->add = T->l->add;
        ret->l->sum = T->l->sum;
        ret->l->add += T->add;
    }
    if (T->r)
    {
        ret->r = new node(T->r->l, T->r->r);
        ret->r->add = T->r->add;
        ret->r->sum = T->r->sum;
        ret->r->add += T->add;
    }
    ret->sum = T->sum + (tr - tl + 1) * T->add;
    ret->add = 0;
    return ret;
}

pnode build(int a[], int tl, int tr)
{
    if (tl == tr)
    {
        return new node(a[tl], 0);
    }
    int tm = (tl + tr) / 2;
    pnode ret = new node(build(a, tl, tm), build(a, tm + 1, tr));
    fix(ret, tl, tr);
    return ret;
}
```

```
pnode rsq(pnode &T, int tl, int tr, int l, int r, int &s)
{
    pnode ret = push(T, tl, tr);
    if (tl == l && tr == r)
    {
        s = ret->sum;
        return ret;
    }
    int tm = (tl + tr) / 2;
    if (r <= tm)
    {
        ret->l = rsq(ret->l, tl, tm, l, r, s);
    }
    else if (l > tm)
    {
        ret->r = rsq(ret->r, tm + 1, tr, l, r, s);
    }
    else
    {
        int s1, s2;
        ret->l = rsq(ret->l, tl, tm, l, tm, s1);
        ret->r = rsq(ret->r, tm + 1, tr, tm + 1, r, s2);
        s = s1 + s2;
    }
    fix(T, tl, tm);
    return ret;
}

pnode update(pnode &T, int tl, int tr, int l, int r, int delta)
{
    if (tl == l && tr == r)
    {
        pnode ret = new node(T->l, T->r);
        ret->add = T->add + delta;
        ret->sum = T->sum;
        return ret;
    }
    int tm = (tl + tr) / 2;
    pnode ret = push(T, tl, tr);
    if (r <= tm)
    {
        ret = new node(update(ret->l, tl, tm, l, r, delta), ret->
            r);
    }
    else if (l > tm)
    {
        ret = new node(ret->l, update(ret->r, tm + 1, tr, l, r,
            delta));
    }
    else
    {
        ret = new node(update(ret->l, tl, tm, l, tm, delta),
            update(ret->r, tm + 1, tr, tm + 1, r, delta));
    }
    fix(ret, tl, tr);
    return ret;
}
```

2.6 Treap With Implicit Key

```
const int N = 1000007;
struct node {
   int pr, size;
   int val;
   node *l, *r;

   node(int pr = 0, int val = 0, node *l = NULL, node *r = NULL)
         :
      pr(pr), l(l), r(r), size(1), val(val) {
   }
} *root;
typedef node* pnode;
int cur, prio[N];

inline int getSize(pnode T) {
   return (T ? T->size : 0);
}

inline void fix(pnode &T) {
   if (!T) {
      return;
   }
   T->size = getSize(T->l) + getSize(T->r) + 1;
}

void merge(pnode A, pnode B, pnode &T) {
   if (!A || !B) {
      T = (A ? A : B);
   }
   else if (A->pr > B->pr) {
      merge(A->r, B, A->r);
      T = A;
   }
   else {
      merge(A, B->l, B->l);
      T = B;
   }
   fix(T);
}

void split(pnode T, int x, pnode &LT, pnode &RT, int add = 0) {
   if (!T) {
      LT = RT = 0;
      return;
   }
   int curx = add + getSize(T->l);
   if (x <= curx) {
      split(T->l, x, LT, T->l, add);
      RT = T;
   }
   else {
      split(T->r, x, T->r, RT, add + 1 + getSize(T->l));
      LT = T;
   }
   fix(LT);
   fix(RT);
}
```

2.7 Persistent Treap

```
struct node {
    node *l, *r;
    bool rev;
    int size;
    char data;

    node(char data = 'a') : data(data) {
        l = r = NULL;
        rev = false;
        size = 1;
    }
};
typedef node* pnode;

const int N = 200007;
char st[N];

pnode copyNode(pnode other) {
    if (other == NULL) return NULL;
    pnode ret = new node();
    ret->l = other->l;
    ret->r = other->r;
    ret->rev = other->rev;
    ret->size = other->size;
    ret->data = other->data;
    return ret;
}

int getSize(pnode T) {
    return T ? T->size : 0;
}

void fix(pnode T) {
    if (T) T->size = 1 + getSize(T->l) + getSize(T->r);
}

void push(pnode &T) {
    if (!T->rev) return;
    T->l = copyNode(T->l);
    T->r = copyNode(T->r);
    swap(T->l, T->r);
    T->rev = false;
    if (T->l) T->l->rev ^= 1;
    if (T->r) T->r->rev ^= 1;
}

void split(pnode T, int cnt, pnode &L, pnode &R) {
    if (!T) {
        L = R = NULL;
        return;
    }
    pnode cur = copyNode(T);
    push(cur);
    if (getSize(cur->l) >= cnt) {
        split(cur->l, cnt, L, cur->l);
        R = cur;
    }
    else {
        split(cur->r, cnt - 1 - getSize(cur->l), cur->r, R);
        L = cur;
    }
    fix(L);
    fix(R);
}

void merge(pnode L, pnode R, pnode &T) {
    if (!L || !R) {
        T = copyNode((L ? L : R));
        return;
    }
    int lsize = getSize(L);
    int rsize = getSize(R);
    if (rand() % (lsize + rsize) < lsize) {
        T = copyNode(L);
        push(T);
        merge(T->r, R, T->r);
    }
    else {
        T = copyNode(R);
        push(T);
        merge(L, T->l, T->l);
    }
    fix(T);
}

void print(pnode T) {
    if (!T) return;
    push(T);
    print(T->l);
    putchar(T->data);
    print(T->r);
}
```

2.8 Splay Tree

```
struct _node{
        int in;
        node l, r;
        node p;
};
bool is_root(node v){
        return v->p==NULL || (v->p->l!=v && v->p->r!=v);
}
bool Type(node v){
        return v->p->r == v;
}
void connect(node v,node p,bool ty){

        (ty?p->r:p->l) = v;
        if(v)
                v->p = p;
}
void rotate(node n){
        node p = n->p;
        node g = p->p;
        bool t = Type(n);
        connect(t?n->l:n->r,p,t);
        if(!is_root(p))
                connect(n,g,Type(p));
        else
                n->p = g;
        connect(p,n,t^1);
        update(p);
        update(n);
}
stack< node > st;
void splay(node n){
        node u = n;
        while(1){
                st.push( u);
                if(is_root(u))
                        break;
                u = u->p;
        }
        while(!st.empty()){
                push(st.top());st.pop();
        }
        while(!is_root(n)){
                node p = n->p;
                if(!is_root(p))
                        rotate(  ( Type(n)^Type(p) ) ? n:p );
```

```
                rotate(n);
        }
}
node expose(node n){
        node last = NULL;
        for(node m = n;m;m = m->p){
                splay(m);
                m->r = last;
                last = m;
                update(m);
        }
        splay(n);
        return last;
}

node Par(node v){
        v = v->l;
        while(v){
                push(v);
                if(v->r)
                        v = v->r;
                else
                        break;
        }
        splay(v);
        return v;
}
```

2.9 2D Fenwick Tree Range Updates

```
int get(int x, int y){
        if (x <= 0 || y <= 0)
                return 0;
        int val = mul(Q.get(x, y), mul(x + 1, y + 1)) + Fij.get(x
              , y) - mul(Fi.get(x, y), y + 1) - mul(Fj.get(x, y),
               x + 1);
        return (val%MO + MO) % MO;
}
void update(int x, int y,int val){
        if (x > n || y > m)
                return;
//      cout << x << " " << y<<" "<<val << endl;
        Q.update(x, y, val);
        Fi.update(x, y, mul(val,x));
        Fj.update(x, y, mul(val,y));
        Fij.update(x, y, mul(val, mul(x, y)));
}
void add_val(int sx, int sy, int fx, int fy, int val){//val==1
        update(sx, sy, val);
        update(sx, fy+1, (MO-val)%MO);
        update(fx+1, sy , (MO-val)%MO);
        update(fx + 1, fy+1, val);
}
int get_sum(int sx, int sy, int fx, int fy){
        //cout << sx<<" "<<sy<<" "<<fx<<" "<<fy<<" "<<Q.get(fx,
              fy) << endl;
        int val = get(fx, fy) - get(sx - 1, fy) - get(fx, sy - 1)
               + get(sx - 1, sy - 1);
        val = (val%MO + MO) % MO;
        return val;
}
```

3 Geometry

3.1 3D Convex Hull

```
const int N = 1007;
const LD pi = acosl(-1.0);
const LD eps = 1e-9;
const LD INF = 1e18;

struct pt {
    LD x, y, z;

    pt (LD x=0, LD y=0, LD z=0): x(x), y(y), z(z) {
    }

    void read() {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        x = a;
        y = b;
        z = c;
    }

    void print() {
        cerr << "( " << x << " " << y << " " << z << " ) ";
    }

    pt operator + (const pt &o) const {
        return pt(x+o.x, y+o.y, z+o.z);
    }

    LD dist2(const pt &o) const {
        return sqrt((x-o.x)*(x-o.x) + (y-o.y)*(y-o.y));
    }

    pt operator - (const pt &o) const {
        return pt(x-o.x, y-o.y, z-o.z);
    }

    pt operator * (LD c) const {
        return pt(x*c, y*c, z*c);
    }

    pt operator / (LD c) const {
        return pt(x/c, y/c, z/c);
    }

    LD len() const {
        return sqrtl(x*x + y*y + z*z);
    }

    pt normalize() const {
        return *this * (1.0 / len());
    }

    pt to_len(LD need_len) const {
        return this->normalize() * need_len;
    }

    bool operator < (const pt &o) const {
        if (fabsl(x-o.x) > eps) return x < o.x;
        if (fabsl(y-o.y) > eps) return y < o.y;
        return z < o.z - eps;
    }
};

LD det(LD a, LD b, LD c, LD d) {
    return a*d - b*c;
}

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
}

pt cross(pt u, pt v) {
    return pt(det(u.y, u.z, v.y, v.z),
              -det(u.x, u.z, v.x, v.z),
               det(u.x, u.y, v.x, v.y));
}

vector<pt> p;

struct plane {
    LD a, b, c, d;
    pt A, B, C;
    int i, j, k;

    plane(LD a=0, LD b=0, LD c=0, LD d=0): a(a), b(b), c(c), d(d)
          {
        // note A, B, C stay undefined
    }

    plane(int i, int j, int k): A(p[i]), B(p[j]), C(p[k]), i(i),
          j(j), k(k) {
        pt norm = cross(B-A, C-A).normalize();
        a = norm.x;
        b = norm.y;
        c = norm.z;
        d = -a * A.x - b * A.y - c * A.z;
    }
};


LD dist_from_plane(pt A, plane& t) {
    return (A.x * t.a + A.y * t.b + A.z * t.c + t.d) / sqrtl(t.a*
          t.a + t.b*t.b + t.c*t.c);
}

LD sign(LD x) {
    if (x < -eps) return -1;
    if (x > eps) return 1;
    return 0;
}

bool inside(pt A, vector<plane>& faces, pt O) {
    for (auto t : faces) {
        int need_sign = sign(dist_from_plane(O, t));
        int cur_sign = sign(dist_from_plane(A, t));
        if (need_sign * cur_sign == -1) return false;
    }
    return true;
}

void dfs(int u, vector<vector<int> >& G, vector<bool>& used,
      vector<int>& order) {
    order.pb(u);
    used[u] = true;
    for (auto to : G[u]) {
        if (!used[to]) {
            dfs(to, G, used, order);
        }
    }
}

void solve() {
    int n;
    scanf("%d", &n);

    p.resize(n);
    FOR(i, n) {
        p[i].read();
    }

    vector<plane> faces;
    FOR(i, 4) FOR(j, i) FOR(k, j) {
        faces.pb(plane(i, j, k));
    }

    pt O = (p[0] + p[1] + p[2] + p[3]) / 4.0; // point strictly
          inside

    for (int ptr = 4; ptr < n; ++ptr) {
        auto P = p[ptr];
        if (inside(P, faces, O)) continue;

        vector<plane> new_faces;
        vector<plane> to_delete;

        FOR(i, sz(faces)) {
            int need_sign = sign(dist_from_plane(O, faces[i]));
            int cur_sign = sign(dist_from_plane(P, faces[i]));
            if (cur_sign * need_sign != -1) {
                new_faces.pb(faces[i]);
            }
            else {
                to_delete.pb(faces[i]);
            }
        }

        unordered_map<int, int> M;

        for (auto t : to_delete) {
            int i = t.i;
            int j = t.j;
            int k = t.k;
            M[min(i, j) * N + max(i,j)] += 1;
            M[min(i, k) * N + max(i,k)] += 1;
            M[min(k, j) * N + max(k,j)] += 1;
        }

        vector<vector<int> > G(n);
        int some_vertex = -1;
        for (auto pr : M) {
            if (pr.second >= 2) continue;
            int i = pr.first / N;
            int j = pr.first % N;
            some_vertex = i;
            G[i].pb(j);
            G[j].pb(i);
        }
        assert(some_vertex != -1);
        vector<bool> used(ptr, false);
        vector<int> order;
        dfs(some_vertex, G, used, order);

        vector<plane> to_add;
        FOR(i, sz(order)) {
            int j = (i+1)%sz(order);
            int u = order[i];
            int v = order[j];
            to_add.pb(plane(ptr, u, v));
        }
```

```
        for (auto t : to_add) {
            new_faces.pb(t);
        }
        faces = new_faces;
    }

}
```

3.2 Closest Point Pair

```
struct point {
    double x, y;
    int ind;
    double operator + (const point & a) const
    {
        return sqrt((a.x - x) * (a.x - x) + (a.y - y) * (a.y - y));
    }
} p[100007], temp[100007]; // temp for merge
double best = 1e18;
int n, ind1, ind2;

bool cmp1(const point & a, const point & b)  // sort by x
{
    return (a.x < b.x || (a.x == b.x && a.y < b.y));
}

bool cmp2(const point & a, const point & b)  // sort by y
{
    return (a.y < b.y || (a.y == b.y && a.x < b.x));
}

void merge(int l, int r) // merge for merge_sort by y
{
    int m = (l + r) / 2, i = l, j = m + 1, k;
    for (k = l; k <= r; ++k) {
        if (i > m) {
            temp[k] = p[j++];
        }
        else if (j > r) {
            temp[k] = p[i++];
        }
        else if (cmp2(p[i], p[j])) {
            temp[k] = p[i++];
        }
        else {
            temp[k] = p[j++];
        }
    }
    for (k = l; k <= r; ++k) {
        p[k] = temp[k];
    }
}

void update(int i, int j) {
    double dis = p[i] + p[j];
    if (dis < best) {
        best = dis;
        ind1 = p[i].ind;
        ind2 = p[j].ind;
        if (ind1 > ind2) {
            swap(ind1, ind2);
        }
    }
}

void solve(int l, int r) {
    double midx;
    int i, j, m;
    if (r - l <= 3)
    {
        for (i = l; i <= r; ++i) {
            for (j = i + 1; j <= r; ++j) {
                update(i, j);
            }
        }
        sort(p + l, p + r + 1, cmp2);
        return;
    }
    m = (l + r) / 2;
    midx = p[m].x;
    solve(l, m);
    solve(m + 1, r);
    merge(l, r);

    static int t[100007];
    int tsz = 0;
    for (i = l; i <= r; ++i) {
        if (fabs(p[i].x - midx) < best) {
            j = tsz - 1;
            while (j >= 0 && p[i].y - p[t[j]].y < best) {
                update(i, t[j]);
                j--;
            }
            t[tsz++] = i;
        }
    }

}

// run
scanf("%d", &n);
sort(p, p + n, cmp1);
solve(0, n - 1);
```

3.3 Dynamic Upper Convex Hull

```
const int N = 100007;
int n, x[N], c[N], t[N];
LL dp[N];

const LD eps = 1e-12;
struct segment {
    LL k, b; // y = kx + b
    LD x1, x2; // x1 < x2

    segment(LL k = 0, LL b = 0, LD x1 = -1e7, LD x2 = +1e7) : k(k)
        , b(b), x1(x1), x2(x2) {}

    bool operator < (const segment &o) const {
        if (k != o.k) return k < o.k;
        return b < o.b;
    }
};

struct segment2 {
    LL k, b;
    LD x;

    segment2(LL k = 0, LL b = 0, LD x = 0.0) : k(k), b(b), x(x) {}

    bool operator < (const segment2 &o) const {
        if (fabsl(x - o.x) < eps) return k < o.k;
        return x < o.x;
    }
};
set&lt;segment2&gt; hull2;
set&lt;segment&gt; hull;

int cmp(LL k, LL b, const segment &s) {
    LD y1line = k * s.x1 + b;
    LD y2line = k * s.x2 + b;
    LD y1seg = s.k * s.x1 + s.b;
    LD y2seg = s.k * s.x2 + s.b;
    if (y1line < y1seg + eps && y2line < y2seg + eps) return -1;
    if (y1line + eps > y1seg && y2line + eps > y2seg) return +1;
    return 0;
}

inline void addSegment(segment s) {
    hull.insert(s);
    hull2.insert(segment2(s.k, s.b, s.x1));
}

inline void deleteSegment(segment s) {
    hull.erase(s);
    hull2.erase(segment2(s.k, s.b, s.x1));
}

void addLine(LL k, LL b) {
    //lines.pb(mp(k, b));

    segment cur(k, b);
    if (hull.empty()) {
        addSegment(cur);
        return;
    }
    if (hull.count(cur)) return;
    auto it = hull.lower_bound(cur);
    // go left
    while (!hull.empty() && it != hull.begin()) {
        --it;
        if (cmp(k, b, *it) == 1) deleteSegment(*it);
        else break;
        it = hull.lower_bound(cur);
    }
    it = hull.lower_bound(cur);
    // go right
    while (!hull.empty() && it != hull.end()) {
        if (cmp(k, b, *it) == 1) deleteSegment(*it);
        else break;
        it = hull.lower_bound(cur);
    }
    if (hull.empty()) {
        addSegment(cur);
        return;
    }
    // intersect with left and right
    it = hull.lower_bound(cur);
    if (it == hull.begin()) {
        if (cmp(k, b, *it) == -1) return;
        else {
            LD x = (LD)(b - it->b) / (it->k - k);
            segment tmp = *it;
            tmp.x1 = x;
            deleteSegment(*it);
            addSegment(tmp);
            cur.x2 = x;
            addSegment(cur);
        }
    }
    else if (it == hull.end()) {
        --it;
        if (cmp(k, b, *it) == -1) return;
        LD x = (LD)(b - it->b) / (it->k - k);
        segment tmp = *it;
        tmp.x2 = x;
        deleteSegment(*it);
        addSegment(tmp);
        cur.x1 = x;
        addSegment(cur);
    }
    else {
        auto lit = it; --lit;
        auto rit = it;
        if (cmp(k, b, *lit) == -1 || cmp(k, b, *rit) == -1) return;
        LD x1 = (LD)(b - lit->b) / (lit->k - k);
        LD x2 = (LD)(b - rit->b) / (rit->k - k);
        segment t1 = *lit;
        segment t2 = *rit;
        deleteSegment(t1);
        deleteSegment(t2);
        t1.x2 = x1;
        t2.x1 = x2;
        cur.x1 = x1;
        cur.x2 = x2;
        addSegment(t1);
        addSegment(cur);
        addSegment(t2);
    }
}

LL getMax(LL x) {
    segment2 t(-(1LL << 58), 0, x);
    auto it = hull2.lower_bound(t);
    --it;
    return x * it->k + it->b;
}
```

3.4 Farthest Point Pair

```
struct pt {

    LD x, y;

    pt(LD x=0, LD y=0): x(x), y(y) {

    }

    void read() {
        int a, b;
        scanf("%d%d", &a, &b);
        x = a;
        y = b;
    }

    pt operator + (const pt &o) const {
        return pt(x+o.x, y+o.y);
    }

    pt operator - (const pt &o) const {
        return pt(x-o.x, y-o.y);
    }
```

```
    LD dist(const pt &o) const {
        return (x-o.x)*(x-o.x) + (y-o.y)*(y-o.y);
    }

    pt prod(LD c) const {
        return pt(c*x, c*y);
    }

    bool operator < (const pt &o) const {
        static const LD eps=1e-5;
        if (fabsl(x-o.x) > eps) return x < o.x;
        return y < o.y - eps;
    }

} hull[N];
int n, nhull;

LD cross(pt u, pt v) {
    return u.x * v.y - u.y * v.x;
}

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y;
}

bool ccw(pt A, pt B, pt C) {
    static const LD eps = 1e-5;
    return cross(B-A, C-B) > eps;
}

void convex_hull(pt *A, int n) {
    // build convex hull, write answer in (hull, nhull)
}

LD dist_from_line(pt A, pt B, pt C) {
    pt u = B-A;
    swap(u.x, u.y);
    u.x = -u.x;
    return fabsl(dot(u, C-A));
}

LD get_distance(pt *A, int n) {
    convex_hull(A, n);

    LD ret = 0.0;
    int n = nhull;
    pt *A = hull;

    if (n <= 3) {
        FOR(i, n) FOR(j, i) {
            ret = max(ret, A[i].dist(A[j]));
        }
        return ret;
    }

    int r = 1;
    while(dist_from_line(A[0], A[1], A[(r+1)%n]) > dist_from_line
          (A[0], A[1], A[r])) r = (r+1) % n;
    ret = max(ret, A[r].dist(A[0]));

    for (int i = 1; i < n; ++i) {
        while(dist_from_line(A[i], A[(i+1)%n], A[(r+1)%n]) >
              dist_from_line(A[i], A[(i+1)%n], A[r])) r = (r+1) %
                n;
        ret = max(ret, A[r].dist(A[i]));
    }
    return ret;
}
```

3.5 Half Plane Intersection (Randomized Incremental Algorithm)

```
const LD eps = 1e-8;

struct pt {
    LD x, y;
    pt (LD x=0, LD y=0): x(x), y(y) {
    }

    void read() {
        int a, b;
        scanf("%d%d", &a, &b);
        x=a;
        y=b;
    }

    pt operator + (const pt &o) const {
        return pt(x+o.x, y+o.y);
    }

    pt operator - (const pt &o) const {
        return pt(x-o.x, y-o.y);
    }

} p[N];
int n;

struct Plane {
    LD a, b, c;
    int s;

    Plane(LD aa=0, LD bb=0, LD cc=0, int s=0): s(s) {
        LD norm = sqrtl(aa*aa+bb*bb);
        a = aa / norm;
        b = bb / norm;
        c = cc / norm;
    }
};

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y;
}

LD sign(LD x) {
    if (x < -eps) return -1;
    if (x > +eps) return 1;
    return 0;
}

int get_sign(pt A, Plane p) {
    return sign(p.a * A.x + p.b * A.y + p.c);
}

struct HalfPlaneIntersection {

    vector<Plane> planes;

    void init() {
        planes.clear();
        planes.resize(0);
    }

    void add(Plane p) {
        if (p.s == 1) {
            p.c -= eps;
        }
        else {
            p.c += eps;
        }
        planes.pb(p);
    }

    bool check() {
        random_shuffle(all(planes));
        planes.pb(Plane(1, 0, INF, 1));
        planes.pb(Plane(1, 0, -INF, -1));
        planes.pb(Plane(0, 1, INF, 1));
        planes.pb(Plane(0, 1, -INF, -1));
        reverse(all(planes));

        pt cur(-INF, 0);
        for (int iter = 4; iter < sz(planes); ++iter) {
            auto p = planes[iter];

            if (get_sign(cur, p) == -p.s) {
                pt dir = pt(-p.b, p.a);
                vector<pt> A, B;

                FOR(i, iter) {
                    auto q = planes[i];

                    LD det = p.a * q.b - p.b * q.a;
                    if (fabsl(det) < eps) continue;
                    LD detx = q.c * p.b - p.c * q.b;
                    LD dety = p.c * q.a - q.c * p.a;
                    pt X = pt(detx/det, dety/det);

                    if (get_sign(X + dir, q) == q.s) {
                        A.pb(X);
                    }
                    else {
                        B.pb(X);
                    }
                }

                // A and B are not empty (note 4 initial half
                    planes)
                pt amax = A[0];
                for (auto P : A) {
                    if (dot(P, dir) > dot(amax, dir)) amax = P;
                }

                pt bmax = B[0];
                for (auto P : B) {
                    if (dot(P, dir) < dot(bmax, dir)) bmax = P;
                }

                if (amax.x < bmax.x) cur = amax;
                else cur = bmax;

                FOR(i, iter+1) {
                    if (get_sign(cur, planes[i]) == -planes[i].s)
                        return false;
                }
            }
        }

        return true;
    }

} hp;


bool check(int k) {
    hp.init();
    FOR(i, n) {
        int j=(i+k)%n;

        LD a = p[i].y - p[j].y;
        LD b = p[j].x - p[i].x;
        LD c = -a * p[i].x - b * p[i].y;

        int z=(i-1+n)%n;
        int s = sign(a*p[z].x + b*p[z].y + c);

        hp.add(Plane(a, b, c, s));
    }
    return hp.check();
}
```

3.6 Minkowski Sum of Convex Polygons

```
const LD pi = acosl(-1.0);
const LD eps = 1e-11;
const int N = 250007;

#define vect pt
struct pt {
    LL x, y;
    pt() : x(0), y(0) { }
    pt(LL x, LL y) : x(x), y(y) {}

    pt operator + (const pt &o) const {
        return pt(x + o.x, y + o.y);
    }

    pt operator - (const pt &o) const {
        return pt(x - o.x, y - o.y);
    }

    bool operator < (const pt &o) const {
        if (x != o.x) return x < o.x;
        return y < o.y;
    }

    LD len() const {
        return sqrtl((LD)x * x + (LD)y * y);
    }

    pt rotate() const {
        return pt(y, -x);
    }
};
vector<pt> mas[N];
LL ans = 0;

LD get_angle(LD ux, LD uy, LD x, LD y) {
    LD l1 = sqrtl(ux * ux + uy * uy);
    LD l2 = sqrtl(x * x + y * y);
    if (l1 < eps || l2 < eps) return 2 * pi;
    LD t = (ux * x + uy * y) / l1 / l2;
    if (t < -1.0 + eps) return pi;
```

```
    if (t > 1.0 - eps) return 0;
    return acosl(t);
}

void minkowski_sum(int na, pt *A, int nb, pt *B, int &nc, pt *C) {
    int posa = 0;
    FOR(i, na) {
        if (A[i].x > A[posa].x || (A[i].x == A[posa].x && A[i].y > A[posa].y)) posa = i;
    }
    int posb = 0;
    FOR(i, nb) {
        if (B[i].x > B[posb].x || (B[i].x == B[posb].x && B[i].y > B[posb].y)) posb = i;
    }
    nc = 0;
    C[nc++] = A[posa] + B[posb];
    LD rot = 0.0;
    while (true) {
        int nposa = (posa + 1) % na;
        int nposb = (posb + 1) % nb;
        vect ua = vect(A[nposa].x - A[posa].x, A[nposa].y - A[posa].y).rotate();
        vect ub = vect(B[nposb].x - B[posb].x, B[nposb].y - B[posb].y).rotate();

        if (na == 3 && nb == 2 && rot > 0) {
            cerr << ua.x << " " << ua.y << " " << cosl(rot) << sinl(rot) << endl;
        }
        LD anga = get_angle(0.0 + ua.x, 0.0 + ua.y, cosl(rot), sinl(rot));
        LD angb = get_angle(0.0 + ub.x, 0.0 + ub.y, cosl(rot), sinl(rot));

        if (anga < angb) {
            rot += anga;
            posa = nposa;
        }
        else {
            rot += angb;
            posb = nposb;
        }
        if (rot > 2 * pi - eps) break;
        C[nc++] = A[posa] + B[posb];
    }
}

pt tmp[N], H[N];
pt A[N], B[N], C[N];
int na, nb, nc;

LL cross(vect a, vect b) {
    return a.x * b.y - a.y * b.x;
}

bool ccw(pt A, pt B, pt C) {
    return cross(B - A, C - B) > 0;
}

void convex_hull(int &na, pt *A) {
    if (na <= 2) return;
    int nh = 0;
    sort(A, A + na);
    int ptr = 0;
    FOR(i, na) {
        while (ptr >= 2 && !ccw(tmp[ptr - 2], tmp[ptr - 1], A[i])) --ptr;
        tmp[ptr++] = A[i];
    }
    FOR(i, ptr - 1) H[nh++] = tmp[i];
    ptr = 0;
    for (int i = na - 1; i >= 0; --i) {
        while (ptr >= 2 && !ccw(tmp[ptr - 2], tmp[ptr - 1], A[i])) --ptr;
        tmp[ptr++] = A[i];
    }
    FOR(i, ptr - 1) H[nh++] = tmp[i];
    na = nh;
    FOR(i, na) A[i] = H[i];
}

void solvefor(int l, int r) {
    if (l == r) return;

    int m = (l + r) / 2;
    solvefor(l, m);
    solvefor(m + 1, r);

    na = 0;
    for (int i = l; i <= m; ++i) {
        for (pt P : mas[i]) A[na++] = P;
    }
    nb = 0;
    for (int i = m + 1; i <= r; ++i) {
        for (pt P : mas[i]) B[nb++] = pt(-P.x, -P.y);
    }
    convex_hull(na, A);
    convex_hull(nb, B);

    if (na == 0 || nb == 0) return;

    minkowski_sum(na, A, nb, B, nc, C);
    FOR(i, nc) {
        ans = max(ans, C[i].x * C[i].x + C[i].y * C[i].y);
    }
}
```

3.7 Pair of Intersecting Segments

```
const LD pi = acosl(-1.0);
const LD eps = 1e-7;
struct pt {
   LD x, y;
   pt(LD x = 0.0, LD y = 0.0) : x(x), y(y) {}

   void rotate(LD ang) {
      LD fi = atan2l(y, x);
      LD len = sqrtl(x * x + y * y);
      fi += ang;
      x = len * cosl(fi);
      y = len * sinl(fi);
   }
};
vector<pair<pt, pt> > seg;
int n;

bool inSegment(pt &M, pt &A, pt &B) {
   return min(A.x, B.x) < M.x + eps && M.x < max(A.x, B.x) + eps &&
      min(A.y, B.y) < M.y + eps && M.y < max(A.y, B.y) + eps;
}

bool intersects(LD a, LD b, LD c, LD d) {
   if (a > b) swap(a, b);
   if (c > d) swap(c, d);
   return max(a, c) < min(b, d) + eps;
}

bool intersects(pt &A, pt &B, pt &C, pt &D) {
   LD a1 = A.y - B.y;
   LD b1 = B.x - A.x;
   LD c1 = -a1 * A.x - b1 * A.y;

   LD a2 = C.y - D.y;
   LD b2 = D.x - C.x;
   LD c2 = -a2 * C.x - b2 * C.y;

   LD det = a1 * b2 - a2 * b1;

   if (fabsl(det) < eps) {
      if (fabsl(c1 * b2 - c2 * b1) < eps && fabsl(c1 * a2 - c2 * a1) < eps)
         return intersects(A.x, B.x, C.x, D.x) && intersects(A.y, B.y, C.y, D.y);
      return false;
   }
   else {
      pt M;
      M.x = (b1 * c2 - b2 * c1) / det;
      M.y = (a2 * c1 - a1 * c2) / det;
      return inSegment(M, A, B) && inSegment(M, C, D);
   }
}

inline bool intersects(int i, int j) {
   return intersects(seg[i].first, seg[i].second, seg[j].first, seg[j].second);
}

LD sweepX;
struct segment {
   int id;
   segment(int id = 0) : id(id) {}
   bool operator < (const segment &o) const {
      pt& A = seg[id].first;
      pt& B = seg[id].second;
      pt& C = seg[o.id].first;
      pt& D = seg[o.id].second;
      /*
      ete uneng vertical hatvacner, hetevyal kerpov enq y hashvum, aglorithmy jisht e ashxatum naev
      vertical uxixneri depqum, vorpes y yntrum enq kamayakan, orinak hatvaci araji keti Y
      bolor verticalnery skzbic insert en arvum, heto deleta TESTED
      ete ayspes anenq, nerqevum(144-159) petq che pttel ketery
      LD y1 = fabsl(B.x - A.x) > eps ? A.y + (B.y - A.y) * (sweepX - A.x) / (B.x - A.x) : A.y;
      LD y2 = fabsl(D.x - C.x) > eps ? C.y + (D.y - C.y) * (sweepX - C.x) / (D.x - C.x) : C.y;
      */
      LD y1 = A.y + (B.y - A.y) * (sweepX - A.x) / (B.x - A.x);
      LD y2 = C.y + (D.y - C.y) * (sweepX - C.x) / (D.x - C.x);
      if (fabsl(y1 - y2) < eps) return id < o.id;
      return y1 < y2;
   }
};
set&lt;segment&gt; status;

// main
scanf("%d", &n);
for (int i = 0; i < n; ++i) {
  int ax, ay, bx, by;
  scanf("%d%d%d%d", &ax, &ay, &bx, &by);
  seg.pb(mp(pt(ax, ay), pt(bx, by)));
}

while (true) {
  LD rotAngle = rand() / (LD)RAND_MAX * 2.0 * pi;
  bool ok = true;
  for (int i = 0; i < n; ++i) {
     seg[i].first.rotate(rotAngle);
     seg[i].second.rotate(rotAngle);
     if (fabsl(seg[i].first.x - seg[i].second.x) < eps) {
        ok = false;
        break;
     }
  }
  if (ok) break;
}
vector<pair<LD, int> > events;

for (int i = 0; i < n; ++i) {
  if (seg[i].first.x > seg[i].second.x) swap(seg[i].first, seg[i].second);

  events.pb(mp(seg[i].first.x, -i - 1));
  events.pb(mp(seg[i].second.x, +i + 1));
}
sort(all(events));
for (int i = 0; i < 2 * n; ++i) {
  int ty = events[i].second;
  sweepX = events[i].first;
  if (ty < 0) // open
  {
     int id = -ty - 1;
     auto it = status.lower_bound(segment(id));
     if (it != status.begin()) {
        auto oit = it;
        --oit;
        if (intersects(oit->id, id)) {
           printf("YES\n%d %d\n", id + 1, oit->id + 1);
           return 0;
        }
     }
     if (it != status.end() && intersects(id, it->id)) {
        printf("YES\n%d %d\n", id + 1, it->id + 1);
        return 0;
     }
     status.insert(segment(id));
  }
  else // close
  {
     int id = ty - 1;
     auto it = status.lower_bound(segment(id));
     if (it != status.begin() && it != status.end()) {
        auto lit = it; --lit;
        auto rit = it; ++rit;
        if (rit != status.end() && intersects(lit->id, rit->id)) {
           printf("YES\n%d %d\n", lit->id + 1, rit->id + 1);
           return 0;
        }
     }
     status.erase(segment(id));
  }
}
puts("NO");
```

3.8 Minimum Enclosing Circle (Randomized Incremental Algorithm)

```
// One can do 2 nested ternary searches: O(N * (logN)^2)
struct MEC {

    bool inside_circle(pt A, pt C, LD R) {
        return A.dist2(C) < R + eps;
    }

    LD cross(pt u, pt v) {
        return u.x * v.y - u.y * v.x;
    }

    pt intersect_bisectors(pt A, pt B, pt C, pt D) {
        pt AB = (A+B) / 2.0;
        pt CD = (C+D) / 2.0;

        pt n1 = A - B;
        pt n2 = C - D;

        LD a1 = n1.x;
        LD b1 = n1.y;
        LD c1 = -a1 * AB.x - b1 * AB.y;

        LD a2 = n2.x;
        LD b2 = n2.y;
        LD c2 = -a2 * CD.x - b2 * CD.y;

        LD det = a1 * b2 - a2 * b1;
        assert(fabsl(det) > eps);

        LD detx = c2 * b1 - c1 * b2;
        LD dety = c1 * a2 - c2 * a1;

        return pt(detx/det, dety/det);
    }

    void update_ans(pt O, vector<pt>& p, pt A, pt B, pt& C, LD& R
          ) {
        LD cur = O.dist2(A);
        if (cur > R) return;
        for (auto E : p) {
            if (!inside_circle(E, O, cur)) return;
        }
        R = cur;
        C = O;
    }

    void consider(vector<pt> &p, vector<pt>& side, pt A, pt B, pt
          & C, LD& R) {
        pt mini(INF,INF,INF);
        pt maxi(-INF, -INF, -INF);

        for (auto E : side) {
            pt X = intersect_bisectors(A, E, A, B);
            if (X < mini) mini = X;
            if (maxi < X) maxi = X;
        }

        if (mini.x < INF/2) {
            update_ans(mini, p, A, B, C, R);
            update_ans(maxi, p, A, B, C, R);
        }
    }

    void two_points_known(vector<pt>& p, pt A, pt B, pt& C, LD& R
          ) {
        pt dir = B-A;

        R=INF;
        update_ans((A+B)/2.0, p, A, B, C, R);

        vector<pt> lside, rside;
        for (auto E : p) {
            if (cross(dir, E-A) > +eps) lside.pb(E);
            if (cross(dir, E-A) < -eps) rside.pb(E);
        }
        consider(p, lside, A, B, C, R);
        consider(p, rside, A, B, C, R);
    }

    void one_point_known(vector<pt>& p, pt A, pt& C, LD& R) {
        random_shuffle(all(p));

        C = (p[0] + A) / 2.0;
        R = p[0].dist2(A) / 2.0;

        for (int i = 1; i < sz(p); ++i) {
            if (inside_circle(p[i], C, R)) continue;
            vector<pt> mas(p.begin(), p.begin()+i);
            two_points_known(mas, A, p[i], C, R);
        }
    }

    LD get(vector<pt> &p) {
        random_shuffle(all(p));
        pt C = p[0];
        LD R = 0.0;

        for (int i = 1; i < sz(p); ++i) {
            if (inside_circle(p[i], C, R)) continue;
            vector<pt> mas(p.begin(), p.begin()+i);
            one_point_known(mas, p[i], C, R);
        }

        return R;
    }

} mec;
```

4 Graphs

4.1 Facet Finding

```
const LD eps = 1e-12;
struct pt {
   LD x, y;
   pt(LD x = 0.0, LD y = 0.0) : x(x), y(y) {};
};
typedef pt vect;
vector<pair<pt, pt> > lines;
vector<vector<pt> > in;
vector<vector<int> > G;
vector<pt> mas;
int n, nlines;
map<LL, LD> M;

bool cmp(const pt &A, const pt &B) {
   if (fabsl(A.x - B.x) < eps) return A.y + eps < B.y;
   return A.x + eps < B.x;
}

void unique(vector<pt> &mas) {
   if (sz(mas) == 0) return;
   sort(all(mas), cmp);
   vector<pt> tmp;
   tmp.pb(mas[0]);
   for (int i = 1; i < sz(mas); ++i)
      if (cmp(mas[i], mas[i - 1]) || cmp(mas[i - 1], mas[i])) tmp
            .pb(mas[i]);
   mas = tmp;
}

int find(pt &A) {
   for (int i = 0; i < sz(mas); ++i)
      if (!cmp(mas[i], A) && !cmp(A, mas[i])) return i;
   throw "no such point finded";
}

bool ccw(vect &u, vect &v) {
   return u.x * v.y - u.y * v.x > eps;
}

void dfs(int i, vect &u, vector<int> &facet) {
   if (i == facet.front()) return;
   facet.pb(i);
   vect best = u;
   int pos = -1;
   for (int j : G[i]) {
      vect v(mas[j].x - mas[i].x, mas[j].y - mas[i].y);
      if (ccw(best, v) && ccw(u, v)) best = v, pos = j;
   }
   if (pos == -1) facet.clear();
   else dfs(pos, best, facet);
}

int main() {
#ifdef harhro94
   freopen("input.txt", "r", stdin);
   //freopen("output.txt", "w", stdout);
#else
#define task "areas"
   freopen(task".in", "r", stdin);
   freopen(task".out", "w", stdout);
#endif

   int nlines;
   cin >> nlines;
   lines.resize(nlines);
   for (int i = 0; i < nlines; ++i)
      cin >> lines[i].first.x >> lines[i].first.y >> lines[i].
            second.x >> lines[i].second.y;
   in.resize(nlines);

   for (int i = 0; i < nlines; ++i) {
      for (int j = i + 1; j < nlines; ++j) {
         pt A = lines[i].first;
         pt B = lines[i].second;
         pt C = lines[j].first;
         pt D = lines[j].second;

         LD a1 = B.y - A.y;
         LD b1 = A.x - B.x;
         LD c1 = B.x * A.y - B.y * A.x;

         LD a2 = D.y - C.y;
         LD b2 = C.x - D.x;
         LD c2 = D.x * C.y - D.y * C.x;

         LD det = a1 * b2 - a2 * b1;
         if (fabsl(det) < eps) continue;

         LD detx = -c1 * b2 + c2 * b1;
         LD dety = -a1 * c2 + a2 * c1;
         mas.pb(pt(detx / det, dety / det));
         in[i].pb(mas.back());
         in[j].pb(mas.back());
      }
   }
   unique(mas);
   for (int i = 0; i < nlines; ++i) unique(in[i]);
   int n = sz(mas);
   G.resize(n);
   for (int i = 0; i < nlines; ++i) {
      int cnt = sz(in[i]);
      for (int j = 1; j < cnt; ++j) {
         int u = find(in[i][j]);
         int v = find(in[i][j - 1]);
         G[u].pb(v);
         G[v].pb(u);
      }
   }
   for (int i = 0; i < n; ++i) {
      for (int j : G[i]) {
         vect u(mas[j].x - mas[i].x, mas[j].y - mas[i].y);
         vector<int> facet;
         facet.pb(i);
         dfs(j, u, facet);
         if (sz(facet) == 0) continue;
         //cerr << "Facet finded !!!\n";
         //for (int p : facet) cerr << "\t" << mas[p].x << " " <<
               mas[p].y << endl;
         //cerr << endl;
         LD S = 0.0;
         for (int u = 1; u < sz(facet); ++u) {
            pt B = mas[facet[u]];
            pt A = mas[facet[u - 1]];
            S += (A.y + B.y) * (B.x - A.x);
         }
         pt A = mas[facet.back()];
         pt B = mas[facet.front()];
         S += (A.y + B.y) * (B.x - A.x);
         S = fabsl(S) / 2.0;
         sort(all(facet));
         LL h = 0;
         for (int p : facet) h = 1000000007 * h + p;
         if (S > 1e-9) M[h] = S;
      }
   }
   vector<LD> S;
   for (pair<LL, LD> p : M) S.pb(p.second);
   cout << sz(S) << endl;
   sort(all(S));
   for (LD s : S) cout << fixed << setprecision(9) << s << endl;

#ifdef harhro94
   cerr << fixed << setprecision(3) << "\nExecution time = " <<
         clock() / 1000.0 << "s\n";
#endif
   return 0;
}
```

4.2 LCA O(1)

```
const int N = 100007;
int n, timer, tin[N], id[N], ptr, mas[N + N], pos[N], mlog[N + N
    ], mini[20][N + N];
vector<int> G[N];

void dfs(int u, int p = -1) {
   id[timer] = u;
   tin[u] = timer++;
   pos[u] = ptr;
   mas[ptr++] = tin[u];
   for (int to : G[u]) {
      if (to != p) {
         dfs(to, u);
         mas[ptr++] = tin[u];
      }
   }
}

int lca(int u, int v) {
   int l = pos[u], r = pos[v];
   if (l > r) swap(l, r);
   int k = mlog[r - l + 1];
   int w = id[min(mini[k][l], mini[k][r - (1 << k) + 1])];
   return w;
}


dfs(0);
mlog[1] = 0;
for (int i = 2; i < ptr; ++i) mlog[i] = 1 + mlog[i >> 1];
for (int i = 0; i < ptr; ++i) mini[0][i] = mas[i];
for (int k = 0; k < 19; ++k) {
   for (int i = 0; i < ptr; ++i) {
      if (i + (1 << (k + 1)) - 1 >= ptr) break;
      mini[k + 1][i] = min(mini[k][i], mini[k][i + (1 << k)]);
   }
}
```

4.3 Min Cost Max Flow (Dijkstra with Potentials)

```
struct MCMF {
    vector<Edge> E;
    vector<int> G[N];
    int n, s, t;
    int Q[N], pre[N], preId[N];
    LL pi[N], dist[N];
    bool is[N];
    LL ansFlow, ansCost;

    void addEdge(int u, int v, LL cap, LL cost) {
        G[u].pb(sz(E));
        E.pb(Edge(v, cap, cost));
        G[v].pb(sz(E));
        E.pb(Edge(u, 0, -cost));
    }

    void init(int n, int s, int t) {
        this->n = n;
        this->s = s;
        this->t = t;
        assert(t < n);
        FOR(i, n) G[i].clear();
        E.clear();
        ansFlow = ansCost = 0;
    }

    void fordBellman() {
        FOR(i, n) {
            is[i] = false;
            dist[i] = INF;
        }
        int l = 0, r = 0;
        Q[r++] = s;
        dist[s] = 0;
        is[s] = true;
        while (l != r) {
            int u = Q[l++];
            if (l == N) l = 0;
            is[u] = false;
            for (int id : G[u]) {
                int to = E[id].to;
                if (E[id].cap - E[id].flow > 0 && dist[to] > dist
                    [u] + E[id].cost) {
                    dist[to] = dist[u] + E[id].cost;
                    if (!is[to]) {
                        is[to] = true;
                        Q[r++] = to;
                        if (r == N) r = 0;
                    }
                }
            }
        }
    }

    bool dijkstra() {
        FOR(i, n) dist[i] = INF;
        dist[s] = 0;
        set<pair<LL, int> > S;
        S.insert(mp(0, s));
        while (!S.empty()) {
            int u = S.begin()->second;
            S.erase(S.begin());
            for (int id : G[u]) {
                int to = E[id].to;
                LL cost = E[id].cost + pi[u] - pi[to];
                if (E[id].cap - E[id].flow > 0 && dist[to] > dist
                    [u] + cost) {
                    S.erase(mp(dist[to], to));
                    pre[to] = u;
                    preId[to] = id;
                    dist[to] = dist[u] + cost;
                    S.insert(mp(dist[to], to));
                }
            }
        }
        // new_pi = pi + dist, easy to prove
        FOR(i, n) {
            if (dist[i] != INF) pi[i] += dist[i];
        }
        return dist[t] != INF;
    }

    void augment() {
        int u = t;
        LL minCap = INF;
        LL totalCost = 0;
        while (u != s) {
            int id = preId[u];
            minCap = min(minCap, E[id].cap - E[id].flow);
            totalCost += E[id].cost;
            u = pre[u];
        }

        ansFlow += minCap;
        ansCost += minCap * totalCost;

        u = t;
        while (u != s) {
            int id = preId[u];
            E[id].flow += minCap;
            E[id ^ 1].flow -= minCap;
            u = pre[u];
        }
    }

    pair<LL, LL> mcmf() {
        fordBellman();
        FOR(i, n) pi[i] = dist[i];
        while (dijkstra()) {
            augment();
        }
        return mp(ansFlow, ansCost);
    }

} mf;
```

4.4 Offline LCA (Tarjan)

```
int n, m, timer, par[N], cand[N], u1[M], u2[M], pos1[M], pos2[M],
    lca[M];
vector<pair<int, int> > query[M];
vector<int> G[N];
bool used[N];

int getPar(int u) {
   return (par[u] == u ? u : (par[u] = getPar(par[u])));
}

void unite(int u, int v, int newCand) {
   u = getPar(u);
   v = getPar(v);
   if (rand() & 1) {
      swap(u, v);
   }
   par[u] = v;
   cand[v] = newCand;
}

void dfs(int u) {
   used[u] = true;
   par[u] = u;
   cand[u] = u;
   for (int to : G[u]) {
      if (!used[to]) {
         dfs(to);
         unite(to, u, u);
      }
   }
   for (int i = 0; i < sz(query[u]); ++i) {
      int v = query[u][i].first;
      if (used[v]) {
         query[u][i].second = cand[getPar(v)];
      }
   }
}

void process() {
   for (int i = 0; i < m; ++i) {
      int u = u1[i];
      int v = u2[i];
      pos1[i] = sz(query[u]);
      query[u].pb(mp(v, -1));
      pos2[i] = sz(query[v]);
      query[v].pb(mp(u, -1));
   }
   dfs(1);
   for (int i = 0; i < m; ++i) {
      int p1 = pos1[i];
      int p2 = pos2[i];
      int u = u1[i];
      int v = u2[i];
      if (query[u][p1].second != -1)
         lca[i] = query[u][p1].second;
      else if (query[v][p2].second != -1)
         lca[i] = query[v][p2].second;
      else
         assert(false);
   }
}
```

4.5 Online LCA

```
int n, m, par[N], dep[N], up[20][N];
vector<int> G[N];

void dfs(int u, int p = 1, int d = 0) {
   par[u] = p;
   dep[u] = d;
   for (int i = 0; i < sz(G[u]); ++i) {
      int to = G[u][i];
      if (to != p) dfs(to, u, d + 1);
   }
}

int lca(int u, int v) {
   if (dep[u] < dep[v]) swap(u, v);
   int diff = dep[u] - dep[v];
   for (int i = 0; (1 << i) <= diff; ++i)
      if (((diff >> i) & 1) == 1) u = up[i][u];
   if (u == v) return u;
   for (int i = 19; i >= 0; --i)
      if (up[i][u] != up[i][v])
         u = up[i][u], v = up[i][v];
   return up[0][u];
}

// preprocessing

dfs(1);
for (int i = 1; i <= n; ++i) up[0][i] = par[i];
for (int k = 1; k < 20; ++k)
  for (int i = 1; i <= n; ++i)
     up[k][i] = up[k - 1][up[k - 1][i]];
```

4.6 Rooted Tree Isomorphism

```
// Slow string implementation

string dfsSlow(int root, int par, vector<int> *G) {
    vector<string> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfsSlow(to, root, G));
    }
    sort(all(childs));
    string cur = "(";
    for (auto s : childs) cur += s;
    cur += ")";
    return cur;
}

// fast algorithm, DETERMINISTIC

map<vector<int>, int> ID;
int curid = 0;

int getId(vector<int> &v) {
    if (ID.count(v)) return ID[v];
    ID[v] = curid++;
    return curid - 1;
}

int dfs(int root, int par, set<int> *G) {
    vector<int> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfs(to, root, G));
    }
    sort(all(childs));
    return getId(childs);
}

// faster algorithm: vector<int>'s in map is slow
// we will hash that vectors then add hashes to the map

map<LL, int> ID;
int curid = 0;

LL getVectorHash(vector<int> &v) {
    static const LL P = 1000003;
    LL h = 1;
    for (int u : v) h = h * P + u;
    return h;
}

int getId(vector<int> &v) {
    LL h = getVectorHash(v);
    if (ID.count(h)) return ID[h];
    ID[h] = curid++;
    return curid - 1;
}

int dfs(int root, int par, set<int> *G) {
    vector<int> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfs(to, root, G));
    }
    sort(all(childs));
    return getId(childs);
}
```

4.7 Vertex Cover (Types of Vertices)

```
const int N = 2007;
int n, m, k, mt[N];
int ptr[N], deg[N], G[N][N];
bool used[N];
char ans[N];
queue<int> Q;

bool kuhn(int u)
{
   used[u] = true;
   for (int i = 0; i < ptr[u]; ++i)
   {
      int to = G[u][i];
      if (mt[to] == -1 || (!used[mt[to]] && kuhn(mt[to])))
      {
         mt[to] = u;
         mt[u] = to;
         return true;
      }
   }
   return false;
}

// init some matching
for (int i = 1; i <= n + m; ++i)
  mt[i] = -1;

for (int i = 1; i <= n; ++i) deg[i] = ptr[i];

int size = 0;
FOR(iter, n)
{
  int u = -1;
  int best = N * N;
  for (int i = 1; i <= n; ++i)
  {
     if (deg[i] >= 1 && deg[i] < best)
     {
        best = deg[i];
        u = i;
     }
  }
  if (u == -1) break;
  deg[u] = 0;
  FOR(i, ptr[u])
  {
     int to = G[u][i];
     if (mt[to] == -1)
     {
        mt[to] = u;
        mt[u] = to;
        ++size;
        FOR(j, ptr[to]) --deg[G[to][j]];
        break;
     }
  }
}

// kuhn
for (int i = 1; i <= n; ++i)
{
  if (mt[i] == -1)
  {
     for (int j = 1; j <= n; ++j) used[j] = false;
     size += kuhn(i);
  }
}

// bfs from 2 unmatched sides
for (int i = 1; i <= n + m; ++i) ans[i] = 'E';
for (int i = 1; i <= n + m; ++i)
{
  if (mt[i] == -1)
  {
     ans[i] = 'N';
     Q.push(i);
  }
}
while (!Q.empty())
{
  int u = Q.front();
  Q.pop();
  if (ans[u] == 'N')
  {
     FOR(i, ptr[u])
     {
        int to = G[u][i];
        if (ans[to] == 'E')
        {
           ans[to] = 'A';
           Q.push(to);
        }
     }
  }
  else
  {
     int to = mt[u];
     if (ans[to] == 'E')
     {
        ans[to] = 'N';
        Q.push(to);
     }
  }
}
```

4.8 Hungarian Algorithm

```
vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
        p[0] = i;
        int j0 = 0;
        vector<int> minv (m+1, INF);
        vector<char> used (m+1, false);
        do {
                used[j0] = true;
                int i0 = p[j0],  delta = INF,  j1;
                for (int j=1; j<=m; ++j)
                        if (!used[j]) {
                                int cur = a[i0][j]-u[i0]-v[j];
                                if (cur < minv[j])
                                        minv[j] = cur,  way[j] =
                                                j0;
                                if (minv[j] < delta)
                                        delta = minv[j],  j1 = j;
                        }
                for (int j=0; j<=m; ++j)
                        if (used[j])
                                u[p[j]] += delta,  v[j] -= delta;
                        else
                                minv[j] -= delta;
                j0 = j1;
        } while (p[j0] != 0);
        do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
        } while (j0);
}
```

4.9 Cactus

```
int topc, tmr, tin[ N ], fup[ N ], p[ N ];
vi gr[ N ], cycles[ N ], graph[ N ];
vpi c[ N ];
void add_cycle(int v,int to){
        int cr = v;
        ++ topc;
        while(1){
                c[ cr ].push_back( mp(topc,sz(cycles[topc])) );
                cycles[ topc ].push_back( cr );
                if(cr==to)
                        break;
                cr = p[cr];
        }
}
void dfs(int v,int par){

        p[v] = par;
        tin[v] = fup[v] = ++ tmr;
        for(auto to:graph[v]){
                if(to==par)
                        continue;
                if(!tin[to]){
                        dfs(to,v);
                        fup[v] = min(fup[v],fup[to]);

                        if(fup[to] > tin[v])
                                add_cycle(to,v);
                }else
                if(tin[to] < tin[v]){
                        fup[v] = min(fup[v],tin[to]);
                        add_cycle(v,to);
                }
        }
}

void update(vi & a,vi & b,int d){
        if(sz(b)+d > sz(a))a.resize(sz(b)+d,0);
        forn(i,sz(b))
                a[i+d] = add(a[i+d],b[i]);
}
map< pii, vi > memo;
vi & solvefor(int x,int dad){
        if(memo.count(mp(x,dad)))return memo[ mp(x,dad) ];
        memo[ mp(x,dad) ] = {0,1};
        auto & ans = memo[ mp(x,dad) ];
        for(auto p:c[x]){
                int i = p.first;
                int pos = p.second;
```

```
                int c_len = sz(cycles[i]);
                if(i==dad)continue;
                forn(j,c_len){
                        int d = (j-pos+c_len)%c_len;
                        int y = cycles[i][j];
                        if(d==0)continue;
                        update(ans,solvefor(y,i),d);
                        if(c_len > 2)
                                update(ans,solvefor(y,i),c_len-d)
                                        ;
                }
        }
        return ans;
}
```

5 Math

5.1 Chinese Remainder Theorem

```
int power(int a, int b, int mod) {
    if (b == 0) return 1;
    if (b & 1) return (a * power(a, b - 1, mod)) % mod;
    int ret = power(a, b >> 1, mod);
    return (ret * ret) % mod;
}

int getInverse(int a, int mod) {
    a %= mod;
    return power(a, mod - 2, mod);
}

LL solve(vector<int> p, vector<int> r) {
    LL M = 1;
    int n = sz(p);
    for (int i = 0; i < n; ++i) M *= p[i];
    vector<vector<int> > inv(n, vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j) inv[i][j] = getInverse(p[i], p[j]);
    vector<LL> x(n);
    for (int i = 0; i < n; ++i) {
        x[i] = r[i];
        for (int j = 0; j < i; ++j) {
            x[i] = (x[i] - x[j]) * inv[j][i];
            x[i] %= p[i];
            if (x[i] < 0) x[i] += p[i];
        }
    }
    LL ret = 0;
    LL cur = 1;
    for (int i = 0; i < n; ++i) {
        ret += x[i] * cur;
        cur *= p[i];
        ret %= M;
        cur %= M;
    }
    return ret;
}
```

5.2 Discrete Logarithm and Square Root

```
// g ^ x = a (mod p)
int discreteLog(int g, int a, int p) {
    int s = (int)sqrt(p + 0.0);
    vector<pair<int, int> > A, B;
    int gs = power(g, s, p), cur = 1;
    for (int t = 0; t * s < p - 1; ++t, cur = cur * gs % p) {
        A.pb(mp(cur, t));
    }
    int invg = power(g, p - 2, p);
    cur = a;
    for (int i = 0; i < s; ++i, cur = cur * invg % p) {
        B.pb(mp(cur, i));
    }
    sort(all(A));
    sort(all(B));
    int i = 0, j = 0;
    while (i < sz(A) && j < sz(B)) {
        if (A[i].first == B[j].first) return A[i].second * s + B[
            j].second;
        if (A[i].first < B[j].first) ++i;
        else ++j;
    }
    assert(false);
}

// to find square root

int g = findGenerator(p);
int r = discreteLog(g, a, p);
if (r & 1) puts("No root");
else {
    int x1 = power(g, r / 2, p);
    int x2 = p - x1;
    if (x1 > x2) swap(x1, x2);
    if (x1 == x2) {
        assert(p == 2);
        puts("1");
    }
    else printf("%d %d\n", x1, x2);
}
```

5.3 Extended Euclid Algorithm

```
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int g = gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return g;
}
```

5.4 Fast Fourier Transform

```
void DFT(comp *a, int n) {
   int bitlen = 0;
   while ((1 << bitlen) < n) ++bitlen;
   for (int i = 0; i < n; ++i) {
      int r = rev(i, bitlen);
      if (i < r) swap(a[i], a[r]);
   }
   for (int len = 2; len <= n; len <<= 1) {
      int half = (len >> 1);
      comp wlen(cos(2 * pi / len), sin(2 * pi / len));
      for (int i = 0; i < n; i += len) {
         comp power = 1;
         for (int j = i, l = i, r = i + half; j < i + half; ++j,
               ++l, ++r, power *= wlen) {
            comp u = a[l], v = power * a[r];
            a[j] = u + v;
            a[j + half] = u - v;
         }
      }
   }
}

void inverseDFT(comp *a, int n) {
   DFT(a, n);
   for (int i = 0; i < n; ++i) a[i] /= n;
   reverse(a + 1, a + n);
}
```

5.5 Generator of Zp

```
ector<int> d[N];
int cur, used[N];
bool is[N];
int gen[N];

void sieve() {
    fill(is + 2, is + N, true);
    for (int i = 2; i < 200; ++i)
        if (is[i]) for (int j = i * i; j < N; j += i)
            is[j] = false;
    for (int i = 0; i < N; ++i)
        if (is[i]) for (int j = i; j < N; j += i) d[j].pb(i);
}

int power(int a, int n, int MOD) {
    int res = 1;
    while (n) {
        if (n & 1) res = res * a % MOD;
        a = a * a % MOD;
        n >>= 1;
    }
    return res;
}

int findGenerator(int p) {
    if (gen[p]) return gen[p];
    ++cur;
    while (true) {
        int g = rand() % (p - 1) + 1;
        if (used[g] == cur) continue;
        used[g] = cur;
        bool ok = true;
        for (int i = 0; i < sz(d[p - 1]); ++i) {
            int t = d[p - 1][i];
            if (power(g, (p - 1) / t, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return (gen[p] = g);
    }
}
```

5.6 Linear Time Precalculation of Inverses

```
/***  O(MOD) methods to precalculate all inverses
a)
    r[1] = 1;
    for (int i=2; i<m; ++i)
        r[i] = (m - (m/i) * r[m%i] % m) % m;

b)
    r[a*b] = r[a] * r[b] // O(1)
    r[p] = power(p, MOD-2) // O(log(MOD))
    total: O(MOD)
***/

const LL MOD = 1000000000 + 7LL;
const int N = 2000007;

LL f[N], invf[N];
int lp[N];
int pr_cnt, pr[N];

void sieve() {
    lp[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (lp[i] == 0) {
            pr[pr_cnt++] = i;
            lp[i] = i;
        }
        for (int j = 0; j < pr_cnt && pr[j] <= lp[i] && pr[j] * i
                < N; ++j) {
            lp[pr[j] * i] = pr[j];
        }
    }
}

LL power(LL a, LL b) {
    if (b == 0) return 1;
    if (b & 1) return a * power(a, b-1) % MOD;
    LL r = power(a, b >> 1);
    return r * r % MOD;
}

inline LL inv(LL a) {
    return power(a, MOD-2);
}

inline LL C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return f[n] * invf[k] % MOD * invf[n-k] % MOD;
```

```
}

void precalc() {
    f[0] = 1;
    for (int i = 1; i < N; ++i) {
        f[i] = i * f[i-1] % MOD;
    }
    sieve();
    for (int i = 1; i < N; ++i) {
        if (lp[i] == i) invf[i] = inv(i);
        else invf[i] = invf[i / lp[i]] * invf[lp[i]] % MOD;
    }
    invf[0] = 1;
    for (int i = 1; i < N; ++i) {
        invf[i] = invf[i] * invf[i-1] % MOD;
    }
}
```

5.7 Simplex Algorithm

```
namespace simplex{
        const LD eps=1e-10;
        const int N=3000+10,M=3000+10;
        int n,m;
        int Left[M],Down[N],idx[N],va[N];
        LD a[M][N],b[M],c[N],v;
        //maximize cTx, subject to Ax <= b and x >= 0
        void init(int p,int q) {
                n=p; m=q;
                for1(i,m) for1(j,n) a[i][j]=0;
                for1(j,m) b[j]=0; for1(i,n) c[i]=0;
                for1(i,n) idx[i]=0;
                v=0;
        }
        void pivot(int x,int y) {
                swap(Left[x],Down[y]);
                LD k=a[x][y];
                a[x][y]=1; b[x]/=k;
                int t=0;
                for1(j,n) {
                        a[x][j]/=k;
                        if (abs(a[x][j])>eps) va[++t]=j;
                }
                for1(i,m) if(i!=x&&abs(a[i][y])>eps) {
                        k=a[i][y];
                        a[i][y]=0;
                        b[i]-=k*b[x];
                        for1(j,t) a[i][va[j]]-=k*a[x][va[j]];
                }
                k=c[y];
                c[y]=0;
                v+=k*b[x];
                for1(j,t) c[va[j]]-=k*a[x][va[j]];
        }

        LD x[ N ];
        LD ans;
        int solve() {
                for1(i,n) Down[i]=i;
                for1(i,m) Left[i]=n+i;
                while(1) {
                        int x=0;
                        for1(i,m) if (b[i]<-eps&&(x==0||b[i]<b[x
                                ])) x=i;
                        if(x==0) break;
                        int y=0;
                        for1(j,n) if (a[x][j]<-eps) if (y==0||a[x
                                ][j]<a[x][y]) y=j;
                        if(y==0) {  return -1; } //Infeasible
                        pivot(x,y);
                }
                while(1) {
                        int y=0;
                        for1(i,n) if (c[i]>eps&&(y==0||c[i]>c[y])
                                ) y=i;
                        if(y==0) break;
                        int x=0;
                        for1(j,m) if (a[j][y]>eps) if (x==0||b[j
                                ]/a[j][y]<b[x]/a[x][y]) x=j;
                        if(x==0) {  return -2; } // Unbounded
                        pivot(x,y);
                }
                ans = v;
                for1(i,m) if(Left[i]<=n) idx[Left[i]]=i;
                for1(i,n) x[i] = b[idx[i]];
                return 1;
        }
}
```

6 Strings

6.1 Aho-Corasick

```
int code(char ch) {
    if (ch >= 'a' && ch <= 'z') return ch - 'a';
    if (ch >= 'A' && ch <= 'Z') return ch - 'A' + 26;
    assert(ch >= '0' && ch <= '9');
    return ch - '0' + 52;
}

const int A = 64;
const int N = 100007;

struct AC {

    int to[64][N], go[64][N];
    int par[N], dep[N], suff[N];
    int cur_node;
    char pch[N];

    // additional information
    int next_term[N], min_len[N];
    bool term[N];

    AC() {
        memset(suff, -1, sizeof suff);
        memset(go, -1, sizeof go);
        memset(to, -1, sizeof to);
        cur_node = 1;
    }

    void add(string st) {
        int cur = 0;
        for (char symbol : st) {
            int ch = code(symbol);
            if (to[ch][cur] == -1) {
                to[ch][cur] = cur_node++;
            }
            int nxt = to[ch][cur];
            pch[nxt] = ch;
            par[nxt] = cur;
            dep[nxt] = dep[cur] + 1;
            cur = nxt;
        }
        term[cur] = true;
    }

    int get_suff(int node) {
        if (node == 0) return 0;
        if (par[node] == 0) return 0;
        if (suff[node] != -1) return suff[node];

        int ret = get_go(pch[node], get_suff(par[node]));
        suff[node] = ret;
        return ret;
    }

    int get_go(int ch, int node) {
        if (to[ch][node] != -1) return to[ch][node];
        if (go[ch][node] != -1) return go[ch][node];
        if (node == 0) return 0;
        int ret = get_go(ch, get_suff(node));
        go[ch][node] = ret;
        return ret;
    }

    void calc_term() {

        vector<pair<int, int> > mas;
        FOR(i, cur_node) {
            mas.pb(mp(dep[i], i));
        }
        sort(all(mas));
        for (auto p : mas) {
            int i = p.second;
            next_term[i] = -1;
            int to = get_suff(i);
            if (term[to]) next_term[i] = to;
            else next_term[i] = next_term[to];
        }

        for (auto p : mas) {
            int i = p.second;
            min_len[i] = N;
            if (term[i]) min_len[i] = dep[i];
            if (next_term[i] != -1) {
                min_len[i] = min(min_len[i], min_len[next_term[i
                    ]]);
            }
        }
    }

} ac;
```

6.2 Discrete Manacer

```
n = strlen(st);
   int l = 0, r = -1;
   for (int i = 0; i < n; ++i) {
      d1[i] = 1;
      if (i <= r) d1[i] = min(d1[l + (r - i)], r - i + 1);
      while (i + d1[i] < n && i - d1[i] >= 0 && st[i + d1[i]] ==
          st[i - d1[i]]) ++d1[i];
      if (i + d1[i] - 1 > r) {
         l = i - d1[i] + 1;
         r = i + d1[i] - 1;
      }
   }
   l = 0, r = -1;
   for (int i = 0; i < n; ++i) {
      d2[i] = 0;
      if (i <= r) d2[i] = min(d2[l + (r - i) + 1], r - i + 1);
      while (i + d2[i] < n && i - d2[i] - 1 >= 0 && st[i + d2[i]]
          == st[i - d2[i] - 1]) ++d2[i];
      if (i + d2[i] - 1 > r) {
         l = i - d2[i] + 1 - 1;
         r = i + d2[i] - 1;
      }
   }
```

6.3 Palindromic Tree

```
struct node {
   int len, suff;
   int to[2]; // size of the alphabet

   node() {
      to[0] = to[1] = -1;
      suff = -1;
      len = -1;
   }

} T[N];
int size = 2; // 0 -> -1  |   1 -> e
int maxPal = 0;
char st[N], ans[N];

bool addChar(int i, int c) {
   bool ret = false;
   while (true) {
      int curLen = T[maxPal].len;
      if (st[i] == st[i - curLen - 1]) {
         int v;
         if (T[maxPal].to[c] == -1) {
            v = size++;
            T[maxPal].to[c] = v;
            T[v].len = curLen + 2;
            ret = true;
         }
         else v = T[maxPal].to[c];

         if (T[v].len == 1) T[v].suff = 1;
         else {
            while (true) {
               maxPal = T[maxPal].suff;
               if (st[i] == st[i - T[maxPal].len - 1]) {
                  T[v].suff = T[maxPal].to[c];
                  break;
               }
            }
         }
         maxPal = v;
         break;
      }
      maxPal = T[maxPal].suff;
```

```
    }
    return ret;
}


// init
T[1].len = 0;
T[1].suff = 0; // e -> "-1"
```

6.4 Prefix Function

```
p[0] = 0;
for (int i = 1; i < n + m + 1; i++) {
    int pos = p[i - 1];
    while (pos>0 && s[pos] != s[i]) pos = p[pos - 1];
    if (s[pos] == s[i]) pos++;
    p[i] = pos;
}
```

6.5 Suffix Array (Implementation 1)

```
const int N = 20007;
const int LOG = 16;
char st[N];
int n;
int cnt[N], p[N], tp[N], c[N], tc[N];
int lcp[N], mlog[N], mini[LOG][N];

void build() {
    FOR(i, n) ++cnt[st[i]];
    FOR(i, N) {
        if (i) cnt[i] += cnt[i - 1];
    }
    for (int i = n - 1; i >= 0; --i) {
        p[--cnt[st[i]]] = i;
    }
    c[p[0]] = 0;
    for (int i = 1, cur = 0; i < n; ++i) {
        if (st[p[i]] != st[p[i - 1]]) ++cur;
        c[p[i]] = cur;
    }
    for (int h = 0; (1 << h) < n; ++h) {
        FOR(i, n) {
            tp[i] = p[i] - (1 << h);
            if (tp[i] < 0) tp[i] += n;
        }

        memset(cnt, 0, sizeof cnt);
        FOR(i, n) ++cnt[c[i]];
        FOR(i, N) {
            if (i) cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            p[--cnt[c[tp[i]]]] = tp[i];
        }

        tc[p[0]] = 0;
        for (int i = 1, cur = 0; i < n; ++i) {
            int m1 = (p[i] + (1 << h)) % n;
            int m2 = (p[i - 1] + (1 << h)) % n;
            if (c[p[i]] != c[p[i - 1]] || c[m1] != c[m2]) ++cur;
            tc[p[i]] = cur;
        }
        FOR(i, n) c[i] = tc[i];
    }
}

void calc_lcps() {
    int *pos = new int[n];
    FOR(i, n) pos[p[i]] = i;
    int cur = 0;
    FOR(i, n) {
        int ind = pos[i];
        if (ind == n - 1) {
            cur = 0;
            continue;
        }
        cur = max(0, cur - 1);
        while (p[ind] + cur < n && p[ind + 1] + cur < n && st[p[ind
              ] + cur] == st[p[ind + 1] + cur]) {
            ++cur;
        }
        lcp[ind] = cur;
    }
}

void build_sparse_table() {
    mlog[1] = 0;
    for (int i = 2; i < N; ++i) mlog[i] = 1 + mlog[i >> 1];
    FOR(i, n - 1) mini[0][i] = lcp[i];
    FOR(h, LOG - 1) {
        FOR(i, n - 1) {
            if (i + (1 << (h + 1)) > n - 1) break;
            mini[h + 1][i] = min(mini[h][i], mini[h][i + (1 << h)]);
        }
    }
}

int get_min(int i, int j) {
    if (i == j) return N;
    --j;
    int k = mlog[j - i + 1];
    return min(mini[k][i], mini[k][j - (1 << k) + 1]);
}
```

6.6 Suffix Array (Implementation 2)

```
const int N = 100007;
const int A = 256;
char st[N];
int n;

struct SA {

    int C[20][N], p[N], tp[N], cnt[N];

    void build() {
        FOR(i, A) cnt[i] = 0;
        FOR(i, n) ++cnt[st[i]];
        FOR(i, A) {
            if (i) cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            p[--cnt[st[i]]] = i;
        }
        C[0][p[0]] = 0;
        for (int i = 1, cur = 0; i < n; ++i) {
            if (st[p[i]] != st[p[i - 1]]) ++cur;
            C[0][p[i]] = cur;
        }
        for (int h = 1, lev = 0; h < n; h <<= 1, ++lev) {
            FOR(i, n) tp[i] = (p[i] - h + n) % n;
            FOR(i, n) cnt[i] = 0;
            FOR(i, n) ++cnt[C[lev][i]];
            FOR(i, n) {
                if (i) cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i >= 0; --i) {
                int ind = tp[i];
                p[--cnt[C[lev][ind]]] = ind;
            }

            C[lev + 1][p[0]] = 0;
            for (int i = 1, cur = 0; i < n; ++i) {
                if (C[lev][p[i - 1]] != C[lev][p[i]] || C[lev][(p[i -
                      1] + h) % n] != C[lev][(p[i] + h) % n]) ++cur;
                C[lev + 1][p[i]] = cur;
            }
        }
    }
} sa;

int getlcp(int i, int j) {
    int ans = 0;
    for (int h = 19; h >= 0; --h) {
        if ((1 << h) > n) continue;
        if (sa.C[h][i] == sa.C[h][j]) {
            i += (1 << h);
            j += (1 << h);
            ans += (1 << h);
        }
    }
    return ans;
}

int compare(int i, int j, int len) {
    for (int h = 19; h >= 0; --h) {
        if ((1 << h) > len) continue;
        if (sa.C[h][i] < sa.C[h][j]) return -1;
        if (sa.C[h][i] > sa.C[h][j]) return 1;
        i = (i + (1 << h)) % n;
        j = (j + (1 << h)) % n;
        len -= (1 << h);
    }
    return 0;
}
```

6.7 Suffix Tree

```
const int N = 300007;
char A[N], B[N], C[N], S[N];
int na, nb, nc, n;

struct node {
    int l, r, par, suff;
    char pch;
    map<char, int> nxt;

    node(int l = 0, int r = 0, int par = 0, char pch = 0, int suff
          = -1) :
        l(l), r(r), par(par), pch(pch), suff(suff) {
    }

    int len() const {
        return r - l;
    }
} T[2 * N];

struct pos {
    int v, p;
    pos(int v = 0, int p = 0) : v(v), p(p) {}
} ptr;
int size = 1;

pos go(const pos &ptr, char c) {
    if (T[ptr.v].len() == ptr.p) {
        int v = ptr.v;
        if (T[v].nxt.count(c)) {
            int to = T[v].nxt[c];
            return pos(to, 1);
        }
        return pos(-1, -1);
    }
    if (S[T[ptr.v].l + ptr.p] == c) return pos(ptr.v, ptr.p + 1);
    return pos(-1, -1);
}

int split(const pos &ptr) {
    if (T[ptr.v].len() == ptr.p) return ptr.v;
    if (ptr.p == 0) return T[ptr.v].par;

    int mid = size++;
    int v = ptr.v;
    int par = T[v].par;
    char midch = S[T[v].l + ptr.p];

    //par
    T[par].nxt[T[v].pch] = mid;

    //mid
    T[mid].par = par;
    T[mid].pch = T[v].pch;
    T[mid].l = T[v].l;
    T[mid].r = T[v].l + ptr.p;
    T[mid].nxt[midch] = v;

    // v
    T[v].par = mid;
    T[v].pch = midch;
    T[v].l = T[mid].r;

    return mid;
}

pos fastGo(pos ptr, int l, int r) {
    while (l != r) {
        if (T[ptr.v].len() == ptr.p) {
            ptr.v = T[ptr.v].nxt[S[l]];
            ++l;
            ptr.p = 1;
            continue;
        }
        int step = min(T[ptr.v].len() - ptr.p, r - l);
        ptr.p += step;
        l += step;
    }
    if (ptr.p == 0) {
        ptr.p = T[ptr.v].len();
```

```
        ptr.v = T[ptr.v].par;
    }
    return ptr;
}

pos getSuff(int v) {
    if (T[v].suff != -1) return pos(T[v].suff, T[T[v].suff].len())
        ;
    pos ret;
    if (T[v].par == 0) ret = fastGo(0, T[v].l + 1, T[v].r);
    else ret = fastGo(getSuff(T[v].par), T[v].l, T[v].r);
    assert(ret.p == T[ret.v].len());
    T[v].suff = ret.v;
    return ret;
}

void extend(int i) {
    char c = S[i];
    while (true) {
        pos goPos = go(ptr, c);
        if (goPos.v == -1) {
            int mid = split(ptr);
            int leaf = size++;
            T[mid].nxt[c] = leaf;
            T[leaf] = node(i, n, mid, c);
            if (ptr.v == 0) {
                ptr = pos(0, 0);
                break;
            }
            int par = T[mid].par;
            if (par == 0) ptr = fastGo(pos(0, 0), T[mid].l + 1, T[
                mid].r);
            else ptr = fastGo(getSuff(par), T[mid].l, T[mid].r);
        }
        else {
            ptr = goPos;
            break;
        }
    }
}

void print(int u) {
    for (auto p : T[u].nxt) {
        int to = p.second;
        cerr << "edge from " << u << " " << " to " << to << " by
             string " << string(S + T[to].l, S + T[to].r) << endl;
        print(to);
    }
}

// build
T[0].suff = 0;
for (int i = 0; i < n; ++i) extend(i);
```

6.8 Z Function

```
n = (int)s.size();
z.resize(n, 0);
z[0] = 0;
for (i = 1; i < n; i++) {
  if (r >= i) {
    z[i] = min(z[i - l], r - i + 1);
  }
  while (z[i] + i < n && s[z[i] + i] == s[z[i]]) z[i]++;
  if (i + z[i] - 1 > r) {
    l = i;
    r = i + z[i] - 1;
  }
}
```

6.9 All Palindroms

```
const int K = 30;
typedef long long LL;
inline LL F(LL x)
{
        if(x&(1ll<<(K+1)))
                return x^(1ll<<(K+1));
        return x;
}
struct all_palind
{
        static const int N = 3*100000+5;
        LL pl[ N ];
        LL gpl[ N ];
        int n;
        char s[ N ];
        struct triple
        {
                int i, d, k;
                triple(int _i = 0,int _d = 0,int _k = 0):i(_i), d
                    (_d), k(_k){}
        };
        vector< triple > g;
        void init(int _n)
        {
                n = _n;
        }
        void phase(int j)
        {
                vector< triple > g1;
                for(int u = 0; u < (int)g.size(); ++ u)
                {
                        int i = g[u].i;
                        if(i > 1 && s[i-1]==s[j])
                                g1.push_back( triple(g[u].i-1, g[
                                    u].d, g[u].k) );
                }
                vector< triple > g2;
                int r = -j;
                for(int u = 0; u < (int)g1.size(); ++ u)
                {
                        int i = g1[u].i;
                        if(i-r!=g1[u].d)
                        {
                                g2.push_back( triple(i, i-r, 1) )
                                    ;
                                if(g1[u].k > 1)
                                        g2.push_back( triple(i+g1
                                            [u].d, g1[u].d, g1[
                                            u].k-1) );
                        }
                        else
                                g2.push_back( g1[u] );
                        r = i+(g1[u].k-1)*g1[u].d;
                }
                if(j > 1 && s[j-1]==s[j])
                {
                        g2.push_back( triple(j-1, j-1-r, 1) );
                        r = j-1;
                }
                g2.push_back( triple(j, j-r, 1) );

                g.clear();
                triple tr = g2[0];
                for(int u = 1; u < (int)g2.size(); ++ u)
                {
                        if(tr.d==g2[u].d)
                                tr.k+=g2[u].k;
                        else
                        {
                                g.push_back( tr );
                                tr = g2[u];
                        }
                }
                g.push_back( tr );

                for(int u = 0; u < (int)g.size(); ++ u)
                {
                        int i = g[u].i, d = g[u].d;
                        int r = i+(g[u].k-1)*d;
                        LL m = pl[ r - 1 ];
                        if(g[u].k > 1)
                                m = m|gpl[ i-d ];
                        if(d <= i)
                                gpl[ i-d ] = m;
                        pl[j]|=m;
                }
                pl[j] = F(pl[j]<<1);
                if(g[0].i==1)
                        pl[j]|=1;
        }
};
```