Team Notebook

University of Maryland: Keivan Rezaei, Mohammad Mahdavi, and Colin Galen

May 24, 2023

Contents

1 Data Structures

1.1 Dynamic Convex Hull Trick

```
const ld is_query = -(1LL << 62);
struct Line {
    ld m, b;
    mutable std::function<const Line *()> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ld x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // dynamic
     upper hull + max value query
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b)
              * (y->m - x->m);
    }
    void insert_line(ld m, ld b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(
            y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y
            ));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ld best(ld x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

1.2 Heavy Light

```
const int N = 2000*100 + 10;
const int L = 20;
int par[N][L], h[N], fath[N], st[N], en[N], sz[N];
vector<int> c[N]; //Adjacency List
int dsz(int s, int p) {
 sz[s] = 1;
 for(int xt = 0; xt < (int)c[s].size(); xt++) {
  int x = c[s][xt];
  if( x != p ) {
   sz[s] += dsz( x , s );
   if( sz[x] > sz[c[s][0]] )
    swap( c[s][0], c[s][xt] );
  }
 }
 return sz[s];
}
void dfs(int s, int p) {
 static int ind = 0;
 st[s] = ind++;
 for(int k = 1; k < L; k++)
  par[s][k] = par[par[s][k-1]][k-1];
 for(int xt = 0; xt < (int)c[s].size(); xt++) {
  int x = c[s][xt];
  if( x == p ) continue;
  fath[x] = x;
  if( xt == 0 ) fath[x] = fath[s];
  h[x] = h[s] + 1;
  par[x][0] = s;
  dfs(x, s);
 }
 en[s] = ind;
}
int n, q;
void upset(int u, int w, int qv) {
 int stL = max( st[w] , st[fath[u]] );
 set( stL, st[u] + 1 , qv , 0, n , 1 ); //l,r,val,s,e,id
 if( stL == st[w] ) return;
 upset( par[fath[u]][0] , w , qv );
}
```

1.3 Link-Cut tree

```
Node x[N];
struct Node {
 int sz, label; /* size, label */
 Node *p, *pp, *l, *r; /* parent, path-parent, left, right
      pointers */
 Node() { p = pp = l = r = 0; }
};
void update(Node *x) {
 x->sz = 1;
 if(x->l) x->sz += x->l->sz;
 if(x->r) x->sz += x->r->sz;
}
void rotr(Node *x) {
 Node *y, *z;
 y = x->p, z = y->p;
 if((y->l = x->r)) y->l->p = y;
 x->r = y, y->p = x;
 if((x->p = z)) {
  if(y == z->l) z->l = x;
  else z->r = x;
 }
 x->pp = y->pp;
 y->pp = 0;
 update(y);
}
void rotl(Node *x) {
 Node *y, *z;
 y = x->p, z = y->p;
 if((y->r = x->l)) y->r->p = y;
 x->l = y, y->p = x;
 if((x->p = z)) {
  if(y == z->l) z->l = x;
  else z->r = x;
 }
 x->pp = y->pp;
 y->pp = 0;
 update(y);
}
void splay(Node *x) {
 Node *y, *z;
 while(x->p) {
  y = x->p;
  if(y->p == 0) {
   if(x == y->l) rotr(x);
   else rotl(x);
  }
  else {
   z = y->p;
   if(y == z->l) {
    if(x == y->l) rotr(y), rotr(x);
    else rotl(x), rotr(x);
   }
   else {  if(x == y->r) rotl(y), rotl(x);
    else rotr(x), rotl(x);
   }
  }
 }
 update(x);
}
Node *access(Node *x) {
 splay(x);
 if(x->r) {
  x->r->pp = x;
  x->r->p = 0;
  x->r = 0;
  update(x);
 }
 Node *last = x;
 while(x->pp) {
  Node *y = x->pp;
  last = y;
  splay(y);
  if(y->r) {
   y->r->pp = y;
   y->r->p = 0;
  }
  y->r = x;
  x->p = y;
  x->pp = 0;
  update(y);
  splay(x);
 }
 return last;
}
Node *root(Node *x) {
 access(x);
 while(x->l) x = x->l;
 splay(x);
 return x;
}
void cut(Node *x) {
 access(x);
 x->l->p = 0;
 x->l = 0;
 update(x);
}
void link(Node *x, Node *y) {
 access(x);
 access(y);
 x->l = y;
 y->p = x;
 update(x);
}
Node *lca(Node *x, Node *y) {
 access(x);
 return access(y);
}
int depth(Node *x) {
 access(x);
 return x->sz - 1;
}
void init(int n) {
 for(int i = 0; i < n; i++) {
  x[i].label = i;
  update(&x[i]);
 }
}
```

1.4 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
     tree_order_statistics_node_update
using namespace std;
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;
// find_by_order = A[x], order_of_key = index of x
```

1.5 Treap

```
struct item {
 int key, prior;
 item * l, * r;
 item() { }
 item (int key, int prior) : key(key), prior(prior), l(NULL)
      , r(NULL) { }
};
typedef item * pitem;
void split (pitem t, int key, pitem & l, pitem & r) {
 if (!t)
  l = r = NULL;
 else if (key < t->key)
  split (t->l, key, l, t->l), r = t;
 else
  split (t->r, key, t->r, r), l = t;
}
void insert (pitem & t, pitem it) {
 if (!t)
  t = it;
 else if (it->prior > t->prior)
  split (t, it->key, it->l, it->r), t = it;
 else
  insert (it->key < t->key ? t->l : t->r, it);
}
void merge (pitem & t, pitem l, pitem r) {
 if (!l || !r)
  t = l ? l : r;
 else if (l->prior > r->prior)
  merge (l->r, l->r, r), t = l;
 else
  merge (r->l, l, r->l), t = r;
}
void erase (pitem & t, int key) {
 if (t->key == key)
  merge (t, t->l, t->r);
 else
  erase (key < t->key ? t->l : t->r, key);
}
pitem unite (pitem l, pitem r) {
 if (!l || !r) return l ? l : r;
 if (l->prior < r->prior) swap (l, r);
 pitem lt, rt;
 split (r, l->key, lt, rt);
 l->l = unite (l->l, lt);
 l->r = unite (l->r, rt);
 return l;
}
pitem root = NULL;
int main()
{
 ios_base::sync_with_stdio(false),cin.tie(0);
 item a = item(10,20);
 item b = item(10,20);
 insert(root, &a);
 insert(root, &b);
 return 0;
}
```

2 Dp Optimizations

2.1 Convex Hull Trick

```
#define F first
#define S second
#define pii pair <int, int>
#define pb psh_back

typedef long long ll;
```

```
vector <pair <ll, ll> > cv;

ll barkhord(pair<ll, ll> p1, pair<ll, ll> p2) { //Make sure
     m1 > m2;
 return (p2.S - p1.S + p1.F - p2.F - 1) / (p1.F - p2.F);
}

ll get(ll t)
{
 int lo = -1, hi = cv.size() - 1;
 while(hi - lo > 1)
 {
  int mid = (lo + hi)/2;
  if(barkhord(cv[mid + 1], cv[mid]) <= t) lo = mid;
  else hi = mid;
 }
 return t * cv[hi].F + cv[hi].S;
}

//{m, h} in points.
void build(vector <pair <ll, ll> > points) {

 sort(points.begin(), points.end(), cmp); //Make them
      increasing in m and decreasing in h.

 for (auto X : points)
 {
  while((cv.size() >= 1 and cv.back().F == X.F) or
    (cv.size() >= 2 and barkhord(X, cv.back()) <= barkhord(cv
         .back(), cv[cv.size() - 2])))
   cv.pop_back();
  cv.pb(X);
 }
 //cv is convex hull.
}
```

2.2 Knuth

Knuth Optimization is applicable if $C_{i,j}$ satisfied the following 2 conditions:

1- Quadrangle Inequality: $C_{a,c} + C_{b,d} \leq C_{a,d} + C_{b,c}$ for $a \leq b \leq c \leq d$

2- Monotonicity: $C_{b,c} \leq C_{a,d}$ for $a \leq b \leq c \leq d$

Then if the smallest k that gives optimal answer in $dp_{i,j} = dp_{i-1,k} + C_{k,j}$ equals to $A_{i,j}$ we have:

$$A_{i,j-1} \leq A_{i,j} \leq A_{i+1,j}$$

3 Geometry

3.1 Convex Hull 3D

```
struct pt{
 ld X,Y,Z;
 pt(ld x=0,ld y=0,ld z=0){X=x; Y=y; Z=z;}

 bool operator==(const pt& rhs) const {
  return (rhs.X==this->X && rhs.Y==this->Y && rhs.Z==this->Z
       );
 }
 bool operator<(const pt& rhs) const {
  return rhs.X > this->X || (rhs.X == this->X && rhs.Y >
       this->Y) || (rhs.X==this->X && rhs.Y==this->Y && rhs.Z
       >this->Z);
 }
};

pt operator -(pt p,pt q){return pt(p.X-q.X,p.Y-q.Y,p.Z-q.Z);
      }
ld cross2d(pt p,pt q){return p.X*q.Y-p.Y*q.X;}
pt _cross(pt u,pt v){return pt(u.Y*v.Z-u.Z*v.Y,u.Z*v.X-u.X*v
     .Z,u.X*v.Y-u.Y*v.X); }
pt cross(pt o,pt p,pt q){return _cross(p-o,q-o);}
ld dot(pt p,pt q){return p.X*q.X+p.Y*q.Y+p.Z*q.Z;}
pt shift(pt p) {return pt(p.Y,p.Z,p.X);}
pt norm(pt p)
{
 if(p.Y<p.X || p.Z<p.X) p=shift(p);
 if(p.Y<p.X) p=shift(p);
 return p;
}

const int MAX=1000;

int n;
pt P[MAX];
vector<pt>ans;
queue<pair<int,int> >Q;
set<pair<int,int> >mark;

int main()
{
 cin>>n;
 int mn=0;
 for(int i=0;i<n;i++)
 {
  cin>>P[i].X>>P[i].Y>>P[i].Z;
  if(P[i]<P[mn]) mn=i;
 }
 int nx=(mn==0);
 for(int i=0;i<n;i++)
  if(i!=mn && i!=nx && cross2d(P[nx]-P[mn],P[i]-P[mn])>0)
   nx=i;
 Q.push({mn,nx});
 while(!Q.empty())
 {
  int v=Q.front().first,u=Q.front().second;
  Q.pop();
  if(mark.count({v,u})) continue;
  mark.insert({v,u});
  int p=-1;
  for(int q=0;q<n;q++)
   if(q!=v && q!=u)
    if(p==-1 || dot(cross(P[v],P[u],P[p]),P[q]-P[v])<0)
     p=q;
  ans.push_back(norm(pt(v,u,p)));
  Q.push({p,u});
  Q.push({v,p});
 }
 sort(ans.begin(),ans.end());
 ans.resize(unique(ans.begin(),ans.end())-ans.begin());
 for(int i=0;i<ans.size();i++)
  cout<<ans[i].X<<" "<<ans[i].Y<<" "<<ans[i].Z<<endl;
}
```

3.2 Delaunay Triangulation NlogN

```
typedef long long ll;

bool ge(const ll& a, const ll& b) { return a >= b; }
bool le(const ll& a, const ll& b) { return a <= b; }
bool eq(const ll& a, const ll& b) { return a == b; }
bool gt(const ll& a, const ll& b) { return a > b; }
bool lt(const ll& a, const ll& b) { return a < b; }
int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }

struct pt {
    ll x, y;
    pt() { }
    pt(ll _x, ll _y) : x(_x), y(_y) { }
    pt operator-(const pt& p) const {
        return pt(x - p.x, y - p.y);
    }
    ll cross(const pt& p) const {
        return x * p.y - y * p.x;
    }
    ll cross(const pt& a, const pt& b) const {
        return (a - *this).cross(b - *this);
    }
    ll dot(const pt& p) const {
        return x * p.x + y * p.y;
    }
    ll dot(const pt& a, const pt& b) const {
        return (a - *this).dot(b - *this);
    }
    ll sqrLength() const {
        return this->dot(*this);
    }
    bool operator==(const pt& p) const {
        return eq(x, p.x) && eq(y, p.y);
    }
};

const pt inf_pt = pt(1e18, 1e18);

struct QuadEdge {
    pt origin;
    QuadEdge* rot = nullptr;
    QuadEdge* onext = nullptr;
    bool used = false;
    QuadEdge* rev() const {
        return rot->rot;
    }
    QuadEdge* lnext() const {
        return rot->rev()->onext->rot;
    }
    QuadEdge* oprev() const {
        return rot->onext->rot;
    }
    pt dest() const {
        return rev()->origin;
    }
};

QuadEdge* make_edge(pt from, pt to) {
    QuadEdge* e1 = new QuadEdge;
    QuadEdge* e2 = new QuadEdge;
    QuadEdge* e3 = new QuadEdge;
    QuadEdge* e4 = new QuadEdge;
    e1->origin = from;
    e2->origin = to;
    e3->origin = e4->origin = inf_pt;
    e1->rot = e3;
    e2->rot = e4;
    e3->rot = e2;
    e4->rot = e1;
    e1->onext = e1;
    e2->onext = e2;
    e3->onext = e4;
    e4->onext = e3;
    return e1;
}

void splice(QuadEdge* a, QuadEdge* b) {
    swap(a->onext->rot->onext, b->onext->rot->onext);
    swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e) {
    splice(e, e->oprev());
    splice(e->rev(), e->rev()->oprev());
    delete e->rev()->rot;
    delete e->rev();
    delete e->rot;
    delete e;
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
    QuadEdge* e = make_edge(a->dest(), b->origin);
    splice(e, a->lnext());
    splice(e->rev(), b);
    return e;
}

bool left_of(pt p, QuadEdge* e) {
    return gt(p.cross(e->origin, e->dest()), 0);
}

bool right_of(pt p, QuadEdge* e) {
    return lt(p.cross(e->origin, e->dest()), 0);
}

template <class T>
T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3)
     {
    return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3
         ) +
           a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(pt a, pt b, pt c, pt d) {
// If there is __int128, calculate directly.
// Otherwise, calculate angles.
#if defined(__LP64__) || defined(_WIN64)
    __int128 det = -det3<__int128>(b.x, b.y, b.sqrLength(), c
         .x, c.y,
                                   c.sqrLength(), d.x, d.y, d.
                                        sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), c.x, c.y,
         c.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det -= det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y,
         b.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y,
         b.sqrLength(), c.x,
                          c.y, c.sqrLength());
    return det > 0;
#else
    auto ang = [](pt l, pt mid, pt r) {
        ll x = mid.dot(l, r);
        ll y = mid.cross(l, r);
        long double res = atan2((long double)x, (long double)
             y);
        return res;
    };
    long double kek = ang(a, b, c) + ang(c, d, a) - ang(b, c,
         d) - ang(d, a, b);
    if (kek > 1e-8)
        return true;
    else
        return false;
#endif
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt
     >& p) {
    if (r - l + 1 == 2) {
        QuadEdge* res = make_edge(p[l], p[r]);
        return make_pair(res, res->rev());
    }
    if (r - l + 1 == 3) {
        QuadEdge *a = make_edge(p[l], p[l + 1]), *b =
             make_edge(p[l + 1], p[r]);
        splice(a->rev(), b);
        int sg = sgn(p[l].cross(p[l + 1], p[r]));
        if (sg == 0)
            return make_pair(a, b->rev());
        QuadEdge* c = connect(b, a);
        if (sg == 1)
            return make_pair(a, b->rev());
        else
            return make_pair(c->rev(), c);
    }
    int mid = (l + r) / 2;
    QuadEdge *ldo, *ldi, *rdo, *rdi;
    tie(ldo, ldi) = build_tr(l, mid, p);
    tie(rdi, rdo) = build_tr(mid + 1, r, p);
    while (true) {
        if (left_of(rdi->origin, ldi)) {
            ldi = ldi->lnext();
            continue;
        }
        if (right_of(ldi->origin, rdi)) {
            rdi = rdi->rev()->onext;
            continue;
        }
        break;
    }
    QuadEdge* basel = connect(rdi->rev(), ldi);
    auto valid = [&basel](QuadEdge* e) { return right_of(e->
        dest(), basel); };
    if (ldi->origin == ldo->origin)
        ldo = basel->rev();
    if (rdi->origin == rdo->origin)
        rdo = basel;
    while (true) {
        QuadEdge* lcand = basel->rev()->onext;
        if (valid(lcand)) {
            while (in_circle(basel->dest(), basel->origin,
                 lcand->dest(),
                            lcand->onext->dest())) {
                QuadEdge* t = lcand->onext;
                delete_edge(lcand);
                lcand = t;
            }
        }
        QuadEdge* rcand = basel->oprev();
        if (valid(rcand)) {
            while (in_circle(basel->dest(), basel->origin,
                 rcand->dest(),
                            rcand->oprev()->dest())) {
                QuadEdge* t = rcand->oprev();
                delete_edge(rcand);
                rcand = t;
            }
        }
        if (!valid(lcand) && !valid(rcand))
            break;
        if (!valid(lcand) ||
                (valid(rcand) && in_circle(lcand->dest(), lcand->
                    origin,
                                           rcand->origin, rcand->
                                               dest())))
                basel = connect(rcand, basel->rev());
            else
                basel = connect(basel->rev(), lcand->rev());
        }
        return make_pair(ldo, rdo);
}

vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
    sort(p.begin(), p.end(), [](const pt& a, const pt& b) {
        return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.y, b.y))
            ;
    });
    auto res = build_tr(0, (int)p.size() - 1, p);
    QuadEdge* e = res.first;
    vector<QuadEdge*> edges = {e};
    while (lt(e->onext->dest().cross(e->dest(), e->origin),
         0))
        e = e->onext;
    auto add = [&p, &e, &edges]() {
        QuadEdge* curr = e;
        do {
            curr->used = true;
            p.push_back(curr->origin);
            edges.push_back(curr->rev());
            curr = curr->lnext();
        } while (curr != e);
    };
    add();
    p.clear();
    int kek = 0;
    while (kek < (int)edges.size()) {
        if (!(e = edges[kek++])->used)
            add();
    }
    vector<tuple<pt, pt, pt>> ans;
    for (int i = 0; i < (int)p.size(); i += 3) {
        ans.push_back(make_tuple(p[i], p[i + 1], p[i + 2]));
    }
    return ans;
}
```

3.3 Find Polynomial from it's Points

$P(x) = \sum_{i=1}^{n} y_i \prod_{j=1, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$

3.4 Geometry Duality

duality of point (a, b) is y = ax - b and duality of line y = ax + b is (a, -b)
Properties:

1. p is on l iff l* is in p*
2. p is in intersection of l1 and l2 iff l1* and l2* lie on p*
3. Duality preserve vertical distance
4. Translating a line in primal to moving vertically in dual
5. Rotating a line in primal to moving a point along a non-vertical line
6. $li \cap lj$ is a vertex of lower envelope $\iff$ (li*, lj*) is an edge of upper hull in dual

3.5 Half Planes

```
typedef int T;
typedef long long T2;
typedef long long T4; // maybe int128_t

const int MAXLINES = 100 * 1000 + 10;
const int INF = 20 * 1000 * 1000;

typedef pair<T, T> point;
typedef pair<point, point> line;

#define X first
#define Y second
#define A first
#define B second

// REPLACE ZERO WITH EPS FOR DOUBLE

point operator - (const point &a, const point &b) {
 return point(a.X - b.X, a.Y - b.Y);
}


T2 cross(point a, point b) {
 return ((T2)a.X * b.Y - (T2)a.Y * b.X);
}

bool cmp(line a, line b) {
 bool aa = a.A < a.B;
 bool bb = b.A < b.B;
 if (aa == bb) {
  point v1 = a.B - a.A;
  point v2 = b.B - b.A;
  if (cross(v1, v2) == 0)
   return cross(b.B - b.A, a.A - b.A) > 0;
  else
   return cross(v1, v2) > 0;
 }
 else
  return aa;
}

bool parallel(line a, line b) {
 return cross(a.B - a.A, b.B - b.A) == 0;
}

pair<T2, T2> alpha(line a, line b) {
 return pair<T2, T2>(cross(b.A - a.A, b.B - b.A),
   cross(a.B - a.A, b.B - b.A));
}

bool fcmp(T4 f1t, T4 f1b, T4 f2t, T4 f2b) {
 if (f1b < 0) {
  f1t *= -1;
  f1b *= -1;
 }
 if (f2b < 0) {
  f2t *= -1;
  f2b *= -1;
 }
 return f1t * f2b < f2t * f1b; // check with eps
}

bool check(line a, line b, line c) {
 bool crs = cross(c.B - c.A, a.B - a.A) > 0;
 pair<T2, T2> a1 = alpha(a, b);
 pair<T2, T2> a2 = alpha(a, c);
 bool alp = fcmp(a1.A, a1.B, a2.A, a2.B);
 return (crs ^ alp);
}

bool notin(line a, line b, line c) { // is intersection of a
     and b in ccw direction of c?
 if (parallel(a, b))
  return false;
 if (parallel(a, c))
  return cross(c.B - c.A, a.A - c.A) < 0;
 if (parallel(b, c))
  return cross(c.B - c.A, b.A - c.A) < 0;
 return !(check(a, b, c) && check(b, a, c));
}

void print(vector<line> lines) {
 cerr << " @ @ @ " << endl;
 for (int i = 0; i < lines.size(); i++)
  cerr << lines[i].A.X << " " << lines[i].A.Y << " -> " <<
       lines[i].B.X << " " << lines[i].B.Y << endl;
 cerr << " @ @ @ " << endl<< endl;
}

line dq[MAXLINES];

vector<line> half_plane(vector<line> lines) {
 lines.push_back(line(point(INF, -INF), point(INF, INF)));
 lines.push_back(line(point(-INF, INF), point(-INF, -INF)));
 lines.push_back(line(point(-INF, -INF), point(INF, -INF)));
 lines.push_back(line(point(INF, INF), point(-INF, INF)));
 sort(lines.begin(), lines.end(), cmp);
 int ptr = 0;
 for (int i = 0; i < lines.size(); i++)
  if (i > 0 &&
    (lines[i - 1].A < lines[i - 1].B) == (lines[i].A < lines[
         i].B) &&
    parallel(lines[i - 1], lines[i]))
   continue;
  else
   lines[ptr++] = lines[i];
 lines.resize(ptr);
 if (lines.size() < 2)
  return lines;
 //print(lines);
 int f = 0, e = 0;
 dq[e++] = lines[0];
 dq[e++] = lines[1];
 for (int i = 2; i < lines.size(); i++) {
  while (f < e - 1 && notin(dq[e - 2], dq[e - 1], lines[i]))
   e--;
  //print(vector<line>(dq + f, dq + e));
  if (e == f + 1) {
   T2 crs = cross(dq[f].B - dq[f].A, lines[i].B - lines[i].A
        ) ;
   if (crs < 0)
    return vector<line>();
   else if (crs == 0 && cross(lines[i].B - lines[i].A, dq[f
        ].B - lines[i].A) < 0)
    return vector<line>();
  }
  while (f < e - 1 && notin(dq[f], dq[f + 1], lines[i]))
   f++;
  dq[e++] = lines[i];
 }
 while (f < e - 1 && notin(dq[e - 2], dq[e - 1], dq[f]))
  e--;
 while (f < e - 1 && notin(dq[f], dq[f + 1], dq[e - 1]))
  f++;
 vector<line> res;
 res.resize(e - f);
 for (int i = f; i < e; i++)
  res[i - f] = dq[i];
 return res;
}


int main() {
 int n;
 cin >> n;
 vector<line> lines;
 for (int i = 0; i < n; i++) {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  lines.push_back(line(point(x1, y1), point(x2, y2)));
 }
 lines = half_plane(lines);
 cout << lines.size() << endl;
 for (int i = 0; i < lines.size(); i++)
  cout << lines[i].A.X << " " << lines[i].A.Y << " " <<
       lines[i].B.X << " " << lines[i].B.Y << endl;
}
```

3.6 Minimum Enclosing Circle

```
const int N = 1000*100 + 10;
struct point {
    ll x, y, z;
};
typedef vector<point> circle;
bool ccw(point a, point b, point c) {
    return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a
         .y) >= 0;
}
bool incircle( circle a , point p ) {
    if( sz(a) == 0 ) return false;
    if( sz(a) == 1 )
        return a[0].x == p.x && a[0].y == p.y;
    if( sz(a) == 2 ) {
        point mid = {a[0].x+a[1].x, a[0].y+a[1].y};
        return sq(2*p.x-mid.x) + sq( 2*p.y-mid.y) <= sq(2*a
            [0].x-mid.x) + sq(2*a[0].y-mid.y);
    }
    if( !ccw(a[0], a[1], a[2]) )
        swap(a[0], a[2]);
    return incircle(a[0],a[1],a[2], p) >= 0;
}
point a[N];
circle solve(int i, circle curr) {
    assert(curr.size() <= 3);
    if( i == 0 )
        return curr;
    circle ret = solve(i-1, curr);
    if( incircle(ret, a[i-1]) )
        return ret;
    curr.pb(a[i-1]);
    return solve(i-1, curr);
}
int n;
void gg(circle c) {
    if( sz(c) == 1 ) {
        cout << ld(a[0].x) << " " << ld(a[0].y) << endl;
        cout << 0.1 << endl;
        return;
    }
    if( sz(c) == 2 ) {
        point mid = {c[0].x+c[1].x, c[0].y+c[1].y};
        ld ret = sqrt(sq(2*c[0].x-mid.x) + sq(2*c[0].y-mid.y)
            )/2;
        cout << ld(mid.x) / 2 << " " << ld(mid.y) /2 << endl;
        cout << ret << endl;
    } else {
        lpt a[3];
        for(int i = 0; i < 3; i++)
            a[i] = lpt(c[i].x, c[i].y);
        lpt A = ld(0.5) * (a[0] + a[1]), C = ld(0.5) * (a[1]
            + a[2]);
        lpt B = A + (a[1] - a[0]) * lpt(0, 1), D = C + (a[2]
            - a[1]) * lpt(0, 1);
        lpt center = intersection( A , B , C , D );
        ld ret = abs(a[0] - center);
        cout << center.real() << " " << center.imag() << endl
            ;
        cout << ret << endl;
    }
}
int main(){
    cin >> n;
    for(int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y;
        a[i].z = sq(a[i].x) + sq(a[i].y);
    }
    srand(time(NULL));
    for(int i = 1; i < n; i++)
        swap(a[i], a[rand()%(i+1)]);
    circle ans = solve(n, circle());
    cout << fixed << setprecision(3) ;
    gg(ans);
    return 0;
}
```

3.7 Points Inside Polygon

S = I + B / 2 - 1

3.8 Primitives

```
typedef long double ld;
typedef complex<ld> pt;
typedef vector<pt> poly;
#define x real()
#define y imag()

typedef pair<pt, pt> line;
// +, -, * scalar well defined
const ld EPS = 1e-12;
const ld PI = acos(-1);
const int ON = 0, LEFT = 1, RIGHT = -1, BACK = -2, FRONT =
    2, IN = 3, OUT = -3;

inline bool Lss(ld a, ld b){ return a - b < -EPS; }
inline bool Grt(ld a, ld b){ return a - b > +EPS; }
inline bool Leq(ld a, ld b){ return a - b < +EPS; }
inline bool Geq(ld a, ld b){ return a - b > -EPS; }
inline bool Equ(ld a, ld b){ return abs(a-b) < EPS; }

bool byX(const pt &a, const pt &b)
{
 if (Equ(a.x, b.x)) return Lss(a.y, b.y);
 return Lss(a.x, b.x);
}
bool byY(const pt &a, const pt &b){
 if (Equ(a.y, b.y)) return Lss(a.x, b.x);
 return Lss(a.y, b.y);
}
struct cmpXY{ inline bool operator ()(const pt &a, const pt
    &b)const { return byX(a, b); } };
struct cmpYX{ inline bool operator ()(const pt &a, const pt
    &b)const { return byY(a, b); } };
bool operator < (const pt &a, const pt &b){ return byX(a, b)
    ; }

istream& operator >> (istream& in, pt p){ld valx,valy; in>>
    valx>>valy; p={valx,valy}; return in;}
ostream& operator << (ostream& out, pt p){out<<p.x<<' '<<p.y
    ; return out;}

ld dot(pt a, pt b){return (conj(a) * b).x;}
ld cross(pt a, pt b){return (conj(a) * b).y;}
ld disSQ(pt a, pt b){return norm(a - b);}
ld dis(pt a, pt b){return abs(a - b);}
ld angleX(pt a, pt b){return arg(b - a);}
ld slope(pt a, pt b){return tan(angleX(a,b));}
//polar(r,theta) -> cartesian
pt rotate(pt a, ld theta){return a * polar((ld)1, theta);}
pt rotatePiv(pt a, ld theta, pt piv){return (a - piv) *
    polar((ld)1, theta) + piv;}
ld angleABC(pt a, pt b, pt c){return abs(remainder(arg(a-b)
    - arg(c-b), 2.0 * PI));}
pt proj(pt p, pt v){return v * dot(p,v) / norm(v);}
pt projPtLine(pt a, line l){return proj(a - l.first,l.second
    -l.first)+l.first;}
ld disPtLine(pt p, line l){return dis(p-l.first, proj(p-l.
    first,l.second-l.first));}

int relpos(pt a, pt b, pt c) //c to a-b
{
 b = b-a, c= c-a;
 if (Grt(cross(b,c), 0)) return LEFT;
 if (Lss(cross(b,c), 0)) return RIGHT;
 if (Lss(dot(b,c), 0)) return BACK;
 if (Lss(dot(b,c), abs(b))) return FRONT;
 return ON;
}
int relpos(line l, pt b){return relpos(l.first, l.second, b)
    ;}

pair<pt,bool> intersection(line a, line b)
{
 ld c1 = cross(b.first - a.first, a.second - a.first);
 ld c2 = cross(b.second - a.first, a.second - a.first);
 if (Equ(c1,c2))
  return {{-1,-1},false};
 return {(c1 * b.second - c2 * b.first) / (c1 - c2), true};
}
```

```
bool intersect(line a, line b)
{
 pair<pt, bool> ret = intersection(a,b);
 if (!ret.second) return false;
 if (relpos(a, ret.first) == ON and relpos(b, ret.first) ==
      ON)
  return true;
 return false;
}
bool isconvex(poly &pl)
{
 int n = pl.size();
 bool neg = false, pos = false;
 for (int i=0;i<n;i++)
 {
  int rpos = relpos(pl[i], pl[(i+1)%n], pl[(i+2)%n]);
  if (rpos == LEFT) pos = true;
  if (rpos == RIGHT) neg = true;
 }
 return !(neg&pos);
}
int crossingN(poly &pl, pt a)
{
 int n = pl.size();
 pt b = a;
 for (pt p:pl)
  b.real(max(b.x,p.y));
 int cn = 0;
 for (int i=0;i<n;i++)
 {
  pt p = pl[i], q=pl[(i+1)%n];
  if (intersect({a,b},{p,q}) && (relpos({p,q},a)!= RIGHT ||
       relpos({p,q},b) != RIGHT))
   cn ++;
 }
 return cn;
}
int pointInPoly(poly &pl, pt p)
{
 int n = pl.size();
 for (int i=0;i<n;i++)
  if (relpos(pl[i], pl[(i+1)%n], p) == ON)
   return ON;
 return crossingN(pl,p)%2? IN : OUT;
}

poly getHull(poly &pl, bool lower)
{
 sort(pl.begin(), pl.end(), byX);
 poly res;
 int n = res.size();
 for (auto p : pl)
 {
  while (n >= 2 && relpos(res[n-2], res[n-1], p) == (lower?
       RIGHT : LEFT))
   res.pop_back(), n--;
  res.push_back(p), n++;
 }
 return res;
}

pair<pt, pt> nearestPair(poly &pl)
{
 int n = pl.size();
 sort(pl.begin(), pl.end(), byX);
 multiset<pt, cmpYX> s;
 ld rad = abs(pl[1] - pl[0]);
 pair<pt, pt> res = {pl[0], pl[1]};
 int l = 0, r = 0;
 for (int i=0;i<n;i++)
 {
  while (l<r && Geq(pl[i].x - pl[l].x, rad))
   s.erase(pl[l++]);
  while (r<l && Leq(pl[r].x, pl[i].x))
   s.insert(pl[r++]);
  for (auto it = s.lower_bound(pt(pl[i].x, pl[i].y-rad)); it
        != s.end(); it++)
  {
   if (Grt(it->y, pl[i].y+rad))
    break;
   ld cur = abs(pl[i] - (*it));
   if (Lss(cur, rad))
    rad = cur, res = {*it, pl[i]};
  }
 }
 return res;
}

typedef struct circle{
 pt c;
 ld r;
} cir;

//number of common tangent lines
int tangentCnt(cir c1, cir c2)
{
 ld d= abs(c1.c-c2.c);
 if (Grt(d, c1.r+c2.r)) return 4; //outside
 if (Equ(d, c1.r+c2.r)) return 3; //tangent outside
 if (Lss(d, c1.r+c2.r) && Grt(d, abs(c1.r-c2.r))) return 2;
      //interfere
 if (Equ(d, abs(c1.r-c2.r))) return 1; //tangent inside
 return 0;//inside
}

line intersection(line l, cir c)
{
 ld dis = disPtLine(c.c, l);
 ld d = sqrt(c.r*c.r - dis*dis);
 pt p = projPtLine(c.c, l);
 pt vec = (l.second-l.first)/abs(l.second - l.first);
 return {p + d * vec, p - d * vec};
}

/*
   0 = other is inside this, zero point
   1 = other is tangent inisde of this, one point
   2 = other is intersect with this, two point
   3 = other is tangent outside of this, one point
   4 = other is outside of this, zero point
 */
pair<int, vector<pt> > intersect(cir c, cir other) {
 ld r = c.r;
 pt o = c.c;
 vector<pt> v;
 ld sumr = other.r + r;
 ld rr = r - other.r;
 ld d = dis(o, other.c);
 ld a = (r*r - other.r*other.r + d*d)/(2*d);
 ld h = sqrt(r*r-a*a);
 pt p2 = a * (other.c - o) / d;
 if(Equ(sumr - d, 0)) {
  v.push_back(p2);
  return make_pair(3, v);
 }
 if(Equ(rr - d, 0)) {
  v.push_back(p2);
  return make_pair(1, v);
 }
 if(d <= rr)
  return make_pair(0, v);
 if(d >= sumr)
  return make_pair(4, v);
 pt p3(p2.x + h*(other.c.y - o.y)/d, p2.y - h*(other.c.x - o
      .x)/d);
 pt p4(p2.x - h*(other.c.y - o.y)/d, p2.y + h*(other.c.x - o
      .x)/d);
 v.push_back(p3);
 v.push_back(p4);
 return make_pair(2, v);
}
ld arcarea(ld l, ld r, ld R){//circle with radius(r)
     intersect with circle with radius (R) and distance
     between centers equal to (d)
 ld cosa = (l*l + r*r - R*R)/(2.0*r*l);
 ld a = acos(cosa);
 return r*r*(a - sin(2*a)/2);
}
```

3.9 Rotating Calipers

```
vector<pair<pt, pt>> get_antipodals(poly &p)
{
 int n = p.size();
 sort(p.begin(), p.end(), byX);
 vector <pt> U, L;
 for (int i = 0; i < n; i++){
  while (U.size() > 1 && relpos(U[U.size()-2], U[U.size()
       -1], p[i]) != LEFT)
   U.pop_back();
  while (L.size() > 1 && relpos(L[L.size()-2], L[L.size()
       -1], p[i]) != RIGHT)
   L.pop_back();
  U.push_back(p[i]);
  L.push_back(p[i]);
 }
 vector <pair<pt, pt>> res;
 int i = 0, j = L.size()-1;
 while (i+1 < (int)U.size() || j > 0){
  res.push_back({U[i], L[j]});
  if (i+1 == (int)U.size())
   j--;
  else if (j == 0)
   i++;
  else if (cross(L[j]-L[j-1], U[i+1]-U[i]) >= 0) i++;
  else
   j--;
 }
 return res;
}
```

3.10 Triangles

```
pt bary(pt A, pt B, pt C, ld a, ld b, ld c) {
    return (A*a + B*b + C*c) / (a + b + c);
}
```

```
pt centroid(pt A, pt B, pt C) {
    // geometric center of mass
    return bary(A, B, C, 1, 1, 1);
}
pt circumcenter(pt A, pt B, pt C) {
    // intersection of perpendicular bisectors
    double a = norm(B - C), b = norm(C - A), c = norm(A - B);
    return bary(A, B, C, a*(b+c-a), b*(c+a-b), c*(a+b-c));
}

pt incenter(pt A, pt B, pt C) {
    // intersection of internal angle bisectors
    return bary(A, B, C, abs(B-C), abs(A-C), abs(A-B));
}

pt orthocenter(pt A, pt B, pt C) {
    // intersection of altitudes
    double a = norm(B - C), b = norm(C - A), c = norm(A - B);
    return bary(A, B, C, (a+b-c)*(c+a-b), (b+c-a)*(a+b-c), (c
         +a-b)*(b+c-a));
}

pt excenter(pt A, pt B, pt C) {
    // intersection of two external angle bisectors
    double a = abs(B - C), b = abs(A - C), c = abs(A - B);
    return bary(A, B, C, -a, b, c);

    //// NOTE: there are three excenters
    // return bary(A, B, C, a, -b, c);
    // return bary(A, B, C, a, b, -c);
}
```

3.11 Useful Geometry Facts

```
Area of triangle with sides a, b, c: sqrt(S *(S-a)*(S-b)*(S-
     c)) where S = (a+b+c)/2
Area of equilateral triangle: s^2 * sqrt(3) / 4 where is
     side lenght
Pyramid and cones volume: 1/3 area(base) * height

if p1=(x1, x2), p2=(x2, y2), p3=(x3, y3) are points on
     circle, the center is
x = -((x2^2 - x1^2 + y2^2 - y1^2)*(y3 - y2) - (x2^2 - x3^2 +
      y2^2 - y3^2)*(y1 - y2)) / (2*(x1 - x2)*(y3 - y2) - 2*(
     x3 - x2)*(y1 - y2))
y = -((y2^2 - y1^2 + x2^2 - x1^2)*(x3 - x2) - (y2^2 - y3^3 +
      x2^2 - x3^2)*(x1 - x2)) / (2*(y1 - y2)*(x3 - x2) - 2*(
     y3 - y2)*(x1 - x2))
```

4 Graph

4.1 2-SAT

```
vector<int> adj[2 * N], jda[2 * N], top;
bool mark[2 * N];
int c[2 * N];
void add_clause(int x, int y) {
 adj[x ^ 1].pb(y);
 adj[y ^ 1].pb(x);
 jda[y].pb(x ^ 1);
 jda[x].pb(y ^ 1);
}
void dfs(int u) {
 mark[u] = 1;
 for(auto v : adj[u]) if(!mark[v]) dfs(v);
 top.pb(u);
}

void sfd(int u, int col) {
 c[u] = col;
 for(auto v : jda[u]) if(!c[v]) sfd(v, col);
}

vector<int> two_sat(int n) {
 memset(mark, 0, sizeof mark);
 memset(c, 0, sizeof c);
 top.clear();
 for(int i = 2; i < 2 * n + 2; i++) if(!mark[i]) dfs(i);
 int cnt = 1;
 while(top.size()) {
  int x = top.back(); top.pop_back();
  if(!c[x]) sfd(x, cnt++);
 }
 vector<int> ans, ans1;
 ans1.pb(-1);
 for(int i = 1; i <= n; i++) {
  if(c[2 * i] == c[2 * i + 1]) return ans1;
  if(c[2 * i] > c[2 * i + 1]) ans.pb(i);
 }
 return ans;
}
```

4.2 Biconnected-Component

```
vector<int> adj[N];
bool vis[N];
int dep[N], par[N], lowlink[N];
```

```
vector<vector<int> > comp;
stack<int> st;
void dfs(int u, int depth = 0, int parent = -1){
 vis[u] = true;
 dep[u] = depth;
 par[u] = parent;
 lowlink[u] = depth;
 st.push(u);
 for (int i = 0; i < adj[u].size(); i++){
  int v = adj[u][i];
  if (!vis[v])
  {
   dfs(v, depth + 1, u);
   lowlink[u] = min(lowlink[u], lowlink[v]);
  }
  else
   lowlink[u] = min(lowlink[u], dep[v]);
 }
 if (lowlink[u] == dep[u] - 1){
  comp.push_back(vector<int>());
  while (st.top() != u)
  {
   comp.back().push_back(st.top());
   st.pop();
  }
  comp.back().push_back(u);
  st.pop();
  comp.back().push_back(par[u]);
 }
}
void bicon(int n){
 for (int i = 0; i < n; i++)
  if (!vis[i])
   dfs(i);
}
```

4.3 Directed Minimum Spanning Tree MlogN

```
/*
GETS:
call make_graph(n) at first
you should use add_edge(u,v,w) and
add pair of vertices as edges (vertices are 0..n-1)
GIVES:
output of dmst(v) is the minimum arborescence with root v in
      directed graph
(INF if it hasn't a spanning arborescence with root v)
O(mlogn)
 */
const int INF = 2e7;
struct MinimumAborescense{
 struct edge {
  int src, dst, weight;
 };
 struct union_find {
  vector<int> p;
  union_find(int n) : p(n, -1) { };
  bool unite(int u, int v) {
   if ((u = root(u)) == (v = root(v))) return false;
   if (p[u] > p[v]) swap(u, v);
   p[u] += p[v]; p[v] = u;
   return true;
  }
  bool find(int u, int v) { return root(u) == root(v); }
  int root(int u) { return p[u] < 0 ? u : p[u] = root(p[u]);
       }
  int size(int u) { return -p[root(u)]; }
 };
 struct skew_heap {
  struct node {
   node *ch[2];
   edge key;
   int delta;
  } *root;
  skew_heap() : root(0) { }
  void propagate(node *a) {
   a->key.weight += a->delta;
   if (a->ch[0]) a->ch[0]->delta += a->delta;
   if (a->ch[1]) a->ch[1]->delta += a->delta;
   a->delta = 0;
  }
  node *merge(node *a, node *b) {
   if (!a || !b) return a ? a : b;
   propagate(a); propagate(b);
   if (a->key.weight > b->key.weight) swap(a, b);
   a->ch[1] = merge(b, a->ch[1]);
   swap(a->ch[0], a->ch[1]);
   return a;
  }
  void push(edge key) {
   node *n = new node();
   n->ch[0] = n->ch[1] = 0;
   n->key = key; n->delta = 0;
   root = merge(root, n);
  }
  void pop() {
   propagate(root);
   node *temp = root;
   root = merge(root->ch[0], root->ch[1]);
  }
  edge top() {
   propagate(root);
   return root->key;
  }
  bool empty() {
   return !root;
  }
  void add(int delta) {
   root->delta += delta;
  }
  void merge(skew_heap x) {
   root = merge(root, x.root);
  }
 };
 vector<edge> edges;
 void add_edge(int src, int dst, int weight) {
  edges.push_back({src, dst, weight});
 }
 int n;
 void make_graph(int _n) {
  n = _n;
  edges.clear();
 }
 int dmst(int r) {
  union_find uf(n);
  vector<skew_heap> heap(n);
  for (auto e: edges)
   heap[e.dst].push(e);
  double score = 0;
  vector<int> seen(n, -1);
  seen[r] = r;
  for (int s = 0; s < n; ++s) {
   vector<int> path;
   for (int u = s; seen[u] < 0;) {
    path.push_back(u);
    seen[u] = s;
    if (heap[u].empty()) return INF;
    edge min_e = heap[u].top();
    score += min_e.weight;
    heap[u].add(-min_e.weight);
    heap[u].pop();
    int v = uf.root(min_e.src);
    if (seen[v] == s) {
     skew_heap new_heap;
     while (1) {
      int w = path.back();
      path.pop_back();
      new_heap.merge(heap[w]);
      if (!uf.unite(v, w)) break;
     }
     heap[uf.root(v)] = new_heap;
     seen[uf.root(v)] = -1;
    }
    u = uf.root(v);
   }
  }
  return score;
 }
};
```

4.4 Ear Decomposition

Solution:

1- Find a spanning tree of the given graph and choose a root for the tree.

2- Determine, for each edge uv that is not part of the tree, the distance between the root and the lowest common ancestor of u and v.

3- For each edge uv that is part of the tree, find the corresponding "master edge", a non-tree edge wx such that the cycle formed by adding wx to the tree passes through uv and such that, among such edges, w and x have a lowest common ancestor that is as close to the root as possible (with ties broken by edge identifiers).

4- Form an ear for each non-tree edge, consisting of it and the tree edges for which it is the master, and order the ears by their master edges' distance from the root (with the same tie-breaking rule).

4.5 Edmond-Blossom

```
// Order: M * Sqrt(N)
// Edges of 1-based. add_edge for adding edges and calc for
     calculating matching
// output is in match array (match[i] = 0 if i isn't in
     matching)
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
template<int SZ> struct UnweightedMatch {
 int match[SZ], N;
 vector<int> adj[SZ];

void add_edge(int u, int v) {
 adj[u].pb(v);
 adj[v].pb(u);
}

queue<int> q;
int par[SZ], vis[SZ], orig[SZ], aux[SZ];

void augment(int u, int v) { // toggle edges on u-v path
 while (1) { // one more matched pair
  int pv = par[v], nv = match[pv];
  match[v] = pv; match[pv] = v;
  v = nv; if (u == pv) return;
 }
}

int lca(int u, int v) { // find LCA of supernodes in O(dist
     )
 static int t = 0;
 for (++t;;swap(u,v)) {
  if (!u) continue;
  if (aux[u] == t) return u; // found LCA
  aux[u] = t; u = orig[par[match[u]]];
 }
}

void blossom(int u, int v, int a) { // go other way
 for (; orig[u] != a; u = par[v]) { // around cycle
  par[u] = v; v = match[u]; // treat u as if vis[u] = 1
  if (vis[v] == 1) vis[v] = 0, q.push(v);
  orig[u] = orig[v] = a; // merge into supernode
 }
}

bool bfs(int u) { // u is initially unmatched
 for(int i = 0; i < N + 1; i++)
  par[i] = 0, vis[i] = -1, orig[i] = i;
 q = queue<int>();
 vis[u] = 0;
 q.push(u);
 while (q.size()) { // each node is pushed to q at most
      once
  int v = q.front(); q.pop(); // 0 -> unmatched vertex
  for (int x : adj[v]) {
   if (vis[x] == -1) { // neither of x, match[x] visited
    vis[x] = 1; par[x] = v;
    if (!match[x])
     return augment(u,x),1;
    vis[match[x]] = 0;
    q.push(match[x]);
   } else if (vis[x] == 0 && orig[v] != orig[x]) {
    int a = lca(orig[v],orig[x]); // odd cycle
    blossom(x,v,a), blossom(v,x,a);
   } // contract O(n) times
  }
 }
 return 0;
}

int calc(int _N) { // rand matching -> constant improvement
 N = _N;
 for(int i = 0; i <= N; i++)
  match[i] = aux[i] = 0;
 int ans = 0; vector<int> V(N); iota(V.begin(), V.end(),1);
 shuffle(V.begin(), V.end(),rng); // find rand matching
 for (int x : V) {
  if (!match[x]) {
   for (int y : adj[x]) {
    if (!match[y]) {
     match[x] = y, match[y] = x; ++ans;
     break;
    }
   }
  }
 }
 for(int i = 1; i <= N; i++)
  if (!match[i] && bfs(i))
   ++ans;
 return ans;
 }
};
```

4.6 Flow-Dinic

```
//Order : General: mn^2, Bipartite: mn^0.5, Zero-One: mn
     ^(2/3)

const int maxN = 1000, maxE = 2 * 1e5 + 10;

int from[maxE], to[maxE], cap[maxE], prv[maxE], head[maxN],
    pt[maxN], ec;
void addEdge(int u, int v, int uv, int vu = 0){
 from[ec] = u, to[ec] = v, cap[ec] = uv, prv[ec] = head[u],
 head[u] = ec++;
 from[ec] = v, to[ec] = u, cap[ec] = vu, prv[ec] = head[v],
 head[v] = ec++;
}
int lv[maxN], q[maxN];
```

```
bool bfs(int source, int sink){
 memset(lv, 31, sizeof(lv));
 int h = 0, t = 0;
 lv[source] = 0;
 q[t++] = source;
 while (t-h){
  int v = q[h++];
  for (int e = head[v]; ~e; e = prv[e])
  {
   if (cap[e] && lv[v] + 1 < lv[to[e]]){
    lv[to[e]] = lv[v] + 1;
    q[t++] = to[e];
   }
  }
 }
 return lv[sink] < 1e8;
}
int dfs(int v, int sink, int f = 1e9){
 if (v == sink || f == 0)
  return f;
 int ret = 0;
 for (int &e = pt[v]; ~e; e = prv[e])
  if (lv[v]+1 == lv[to[e]]){
   int x = dfs(to[e], sink, min(f, cap[e]));
   cap[e] -= x;
   cap[e^1] += x;
   ret += x;
   f -= x;
   if (!f)
    break;
  }
 return ret;
}
int dinic(int source, int sink){
 memset(prv, -1, sizeof prv);
 memset(head, -1, sizeof head);

 int ret = 0;
 while (bfs(source, sink)){
  memcpy(pt, head, sizeof(head));
  ret += dfs(source, sink);
 }
 return ret;
}
```

4.7 Gomory-Hu

```
bool mark[N];
int p[N], w[N];
void gfs(int u) {
 mark[u] = 1;
 for(int e = head[u]; e != -1; e = prv[e])
  if(!mark[to[e]] && cap[e])
   gfs(to[e]);
}
//edges is one-directed. Order: O(n * flow)
vector<pair<int, pii>> gomory_hu(int n, vector<pair<int, pii
     >> edges) {
 for(int i = 1; i <= n; i++) p[i] = 1;
 memset(w, 0, sizeof w);
 p[1] = 0;
 for(int i = 2; i <= n; i++) {
  memset(head, -1, sizeof head);
  ec = 0;
  for(auto u : edges) add_edge(u.S.F, u.S.S, u.F);
  w[i] = dinic(i, p[i]);
  memset(mark, 0, sizeof mark);
  gfs(i);
  for(int j = i + 1; j <= n; j++)
   if(mark[j] && p[j] == p[i])
    p[j] = i;
  if(p[p[i]] && mark[p[p[i]]]) {
   int pi = p[i];
   swap(w[i], w[pi]);
   p[i] = p[pi];
   p[pi] = i;
  }
 }
 vector<pair<int, pii>> tree;
 for(int i = 1; i <= n; i++) if(p[i]) tree.pb({w[i], {i, p[i
      ]}});
 return tree;
}
```

4.8 Hungarian

```
const int N = 2002;
const int INF = 1e9;
int hn, weight[N][N]; //hn should contain number of vertices
      in each part. weight must be positive.
int x[N], y[N]; //initial value doesn't matter.

int hungarian() // maximum weighted perfect matching O(n^3)
{
 int n = hn;
 int p, q;
 vector<int> fx(n, -INF), fy(n, 0);
 fill(x, x + n, -1);
 fill(y, y + n, -1);

 for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
   fx[i] = max(fx[i], weight[i][j]);

 for (int i = 0; i < n; ) {
  vector<int> t(n, -1), s(n+1, i);
  for (p = 0, q = 1; p < q && x[i] < 0; ++p) {
   int k = s[p];
   for (int j = 0; j < n && x[i] < 0; ++j)
    if (fx[k] + fy[j] == weight[k][j] && t[j] < 0) {
     s[q++] = y[j], t[j] = k;
     if (y[j] < 0) // match found!
      for (int p = j; p >=0; j = p)
       y[j] = k = t[j], p = x[k], x[k] = j;
    }
  }
  if (x[i] < 0) {
   int d = INF;
   for (int k = 0; k < q; ++k)
    for (int j = 0; j < n; ++j)
     if (t[j] < 0) d = min(d, fx[s[k]] + fy[j] - weight[s[k
          ]][j]);
   for (int j = 0; j < n; ++j) fy[j] += (t[j] <0? 0: d);
   for (int k = 0; k < q; ++k) fx[s[k]] -= d;
  } else ++i;
 }
 int ret = 0;
 for (int i = 0; i < n; ++i) ret += weight[i][x[i]];
 return ret;
}

int main() {
 int n, e; cin >> n >> e;
 for (int i=0; i<e; i++)
 {
  int u, v; cin >> u >> v;
  --u; --v;
  cin >> weight[u][v];
 }
 hn = n;
 cout << hungarian() << '\n';
 return 0;
}
```

4.9 Min-Cost-Max-Flow

```
const int N = 810, E = N * N, INF = 1e9;
```

```
int n, ed = 0, from[E], to[E], cap[E], head[N], nex[E], par[
    N];
ld dis[N], cost[E];

void add_edge(int u, int v, int c, ld co)
{
 from[ed] = u, to[ed] = v, cap[ed] = c, cost[ed] = co , nex[
     ed] = head[u], head[u] = ed ++;
 from[ed] = v, to[ed] = u, cap[ed] = 0, cost[ed] = -co, nex[
     ed] = head[v], head[v] = ed ++;
}

pair<int, ld> spfa(int sink, int source)
{
 for(int i=0; i<N; i++)dis[i] = INF;
 memset(mark, 0, sizeof mark);
 memset(par, -1, sizeof par);

 queue<int> q;
 dis[source] = 0, mark[source] = true;
 q.push(source);

 while(q.size())
 {
  int v = q.front(); q.pop();
  mark[v] = false;

  for(int e = head[v]; e != -1; e = nex[e])
  {
   if(cap[e] && dis[to[e]] > dis[v] + cost[e])
   {
    dis[to[e]] = dis[v] + cost[e];
    par[to[e]] = e;
    if(!mark[to[e]])q.push(to[e]), mark[to[e]] = true;
   }
  }
 }

 int curr = sink;
 if(dis[curr] == INF)return make_pair(0, 0);

 ld res = 0;

 int flow = INF;

 while(curr != source)
 {
  flow = min(flow, cap[par[curr]]);
  curr = from[par[curr]];
 }

 curr = sink;
 while(curr != source)
 {
  res += cost[par[curr]];
  cap[par[curr]] -= flow;
  cap[par[curr] ^ 1] += flow;
  curr = from[par[curr]];
 }

 return make_pair(flow, res);
}

pair<int, ld> MinCostMaxFlow(int sink, int source)
{
 int flow = 0;
 pair<int, ld> f = {INF, 0};
 ld Cost = 0;

 while(f.F)
 {
  f = spfa(sink, source);
  flow += f.F;
  Cost += f.F * f.S;
 }

 return make_pair(flow, Cost);
}
```

5 Number Theory

5.1 Chineese Reminder Theorem

```
#define lcm LLLCCM

ll GCD(ll a, ll b) { return (b == 0) ? a : GCD(b, a % b); }
inline ll LCM(ll a, ll b) { return a / GCD(a, b) * b; }
inline ll normalize(ll x, ll mod) { x %= mod; if (x < 0) x
     += mod; return x; }

struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b){
 if (b == 0) return {1, 0, a};
 GCD_type pom = ex_GCD(b, a % b);
 return {pom.y, pom.x - a / b * pom.y, pom.d};
}
const int N = 2;
ll r[N], n[N], ans, lcm;
// t: number of equations,
// r: reminder array, n: mod array
// returns {reminder, lcm}

pair <ll, ll> CRT(ll* r, ll *n, int t) {
 for(int i = 0; i < t; i++)
  normalize(r[i], n[i]);
 ans = r[0];
 lcm = n[0];

 for(int i = 1; i < t; i++){
  auto pom = ex_GCD(lcm, n[i]);
  ll x1 = pom.x;
  ll d = pom.d;
  if((r[i] - ans) % d != 0) {
   return {-1, -1}; //No Solution
  }
  ans = normalize(ans + x1 * (r[i] - ans) / d % (n[i] / d) *
       lcm, lcm * n[i] / d);
  lcm = LCM(lcm, n[i]); // you can save time by replacing
      above lcm * n[i] /d by lcm = lcm * n[i] / d
 }
 return {ans, lcm};
}
```

5.2 Miller Robin

```
//with probability (1/4) iter, we might make mistake in our
    guess.
//we have false positive here.
using u64 = uint64_t;
using u128 = __uint128_t;

using namespace std;

u64 binpower(u64 base, u64 e, u64 mod) {
 u64 result = 1;
 base %= mod;
 while (e) {
  if (e & 1)
   result = (u128)result * base % mod;
  base = (u128)base * base % mod;
  e >>= 1;
 }
 return result;
}
```

```
bool check_composite(u64 n, u64 a, u64 d, int s) {
 u64 x = binpower(a, d, n);
 if (x == 1 || x == n - 1)
  return false;
 for (int r = 1; r < s; r++) {
  x = (u128)x * x % n;
  if (x == n - 1)
   return false;
 }
 return true;
};

bool MillerRabin(u64 n, int iter=5) { // returns true if n
     is probably prime, else returns false.
 if (n < 4)
  return n == 2 || n == 3;

 int s = 0;
 u64 d = n - 1;
 while ((d & 1) == 0) {
  d >>= 1;
  s++;
 }

 for (int i = 0; i < iter; i++) {
  int a = 2 + rand() % (n - 3);
  if (check_composite(n, a, d, s))
   return false;
 }
 return true;
}
```

5.3 Most Divisors

```
<= 1e2: 60 with 12 divisors
<= 1e3: 840 with 32 divisors
<= 1e4: 7560 with 64 divisors
<= 1e5: 83160 with 128 divisors
<= 1e6: 720720 with 240 divisors
<= 1e7: 8648640 with 448 divisors
<= 1e8: 73513440 with 768 divisors
<= 1e9: 735134400 with 1344 divisors
<= 1e10: 6983776800 with 2304 divisors
<= 1e11: 97772875200 with 4032 divisors
<= 1e12: 963761198400 with 6720 divisors
<= 1e13: 9316358251200 with 10752 divisors
<= 1e14: 97821761637600 with 17280 divisors
<= 1e15: 866421317361600 with 26880 divisors
<= 1e16: 8086598962041600 with 41472 divisors
<= 1e17: 74801040398884800 with 64512 divisors
<= 1e18: 897612484786617600 with 103680 divisors
```

5.4 Number of Primes

```
30: 10
60: 17
100: 25
1000: 168
10000: 1229
100000: 9592
1000000: 78498
10000000: 664579
```

6 Numerical

6.1 Base Vector Z2

```
const int maxL = 61;

struct Base{
 ll a[maxL] = {};
 ll eliminate(ll x){
  for(int i=maxL-1; i>=0; --i) if(x >> i & 1) x ^= a[i];
  return x;
 }
 void add(ll x){
  x = eliminate(x);
  if(x == 0) return ;
  for(int i=maxL-1; i>=0; --i) if(x >> i & 1) {
   a[i] = x;
   return ;
  }
 }
 int size(){
  int cnt = 0;
  for(int i=0; i<maxL; ++i) if(a[i]) ++cnt;
  return cnt;
 }
 ll get_mx() {
  ll x = 0;
  for (int i=maxL-1; i>=0; i--) {
   if(x & (1LL << i)) continue ;
   else x ^= a[i];
  }
  return x;
 }
};
```

6.2 Extended Catalan

number of ways for going from 0 to A with k moves without going to -B:

$$\binom{k}{\frac{A+k}{2}} - \binom{k}{\frac{2B+A+k}{2}}$$

6.3 FFT

```
const int LG = 20; // IF YOU WANT TO CONVOLVE TWO ARRAYS OF
     LENGTH N AND M CHOOSE LG IN SUCH A WAY THAT 2LG > n + m
const int MAX = 1 << LG;

#define M_PI acos(-1)

struct point{
 double real, imag;
 point(double _real = 0.0, double _imag = 0.0){
  real = _real;
  imag = _imag;
 }
};
point operator + (point a, point b){
 return point(a.real + b.real, a.imag + b.imag);
}
point operator - (point a, point b){
 return point(a.real - b.real, a.imag - b.imag);
}
point operator * (point a, point b){
 return point(a.real * b.real - a.imag * b.imag, a.real * b.
   imag + a.imag * b.real);
}
void fft(point *a, bool inv){
 for (int mask = 0; mask < MAX; mask++){
  int rev = 0;
  for (int i = 0; i < LG; i++)
   if ((1 << i) & mask)
    rev |= (1 << (LG - 1 - i));
  if (mask < rev)
   swap(a[mask], a[rev]);
 }
 for (int len = 2; len <= MAX; len *= 2){
  double ang = 2.0 * M_PI / len;
  if (inv)
```

```
   ang *= -1.0;
  point wn(cos(ang), sin(ang));
  for (int i = 0; i < MAX; i += len){
   point w(1.0, 0.0);
   for (int j = 0; j < len / 2; j++){
    point t1 = a[i + j] + w * a[i + j +
     len / 2];
    point t2 = a[i + j] - w * a[i + j +
     len / 2];
    a[i + j] = t1;
    a[i + j + len / 2] = t2;
    w = w * wn;
   }
  }
 }
 if (inv)
  for (int i = 0; i < MAX; i++){
   a[i].real /= MAX;
   a[i].imag /= MAX;
  }
}
```

6.4 Gaussian Elimination

```
const int N = 505, MOD = 1e9 + 7;
typedef vector <ll> vec;

ll pw(ll a, ll b) {
 if(!b)
  return 1;
 ll x = pw(a, b/2);
 return x * x % MOD * (b % 2 ? a : 1) % MOD;
}
ll inv(ll x) { return pw(x, MOD - 2); }

//matrix * x = ans
vec solve(vector<vec> matrix, vec ans) {
 int n = matrix.size(), m = matrix[0].size();
 for (int i=0; i<n; i++)
  matrix[i].pb(ans[i]);

 vector <int> ptr;
 ptr.resize(n);

 int i = 0, j =0;
 while(i < n and j < m) {
  int ind = -1;
  for(int row = i; row < n; row++)
   if(matrix[row][j])
    ind = row;
  if(ind == -1) {
   j++;
   continue ;
  }

  matrix[i].swap(matrix[ind]);
  ll inverse = inv(matrix[i][j]);
  for(int row = i + 1; row < n; row++) {
   ll z = matrix[row][j] * inverse % MOD;
   for(int k = 0; k <= m; k++)
    matrix[row][k] = (matrix[row][k] % MOD - matrix[i][k]*z %
          MOD + MOD) % MOD;
  }

  ptr[i] = j;
  i ++;
  j ++;
 }

 vector <ll> sol;

 if(i != n) {
  for (int row=i; row<n; row++)
   if(matrix[row][m] != 0)
    return sol; //without answer;
 }
 sol.resize(m);
 for (int j=0; j<m; j++)
  sol[j] = 0;

 for (int row=i-1; row>=0; row--){
  int j = ptr[row];
  sol[j] = matrix[row][m] * inv(matrix[row][j]) % MOD;
  for (int c=row-1; c>=0; c--)
   matrix[c][m] += (MOD - sol[j] * matrix[c][j] % MOD),
        matrix[c][m] %= MOD;
 }
 return sol;
}

int main() {
 int n, m; cin >> n >> m;
 vector <vec> A;
 for (int i=0; i<n; i++)
 {
  vec B;
  for (int j=0; j<m; j++)
  {
   ll x; cin >> x;
   B.push_back(x);
  }
  A.push_back(B);
 }

 vec ans;
 for (int i=0; i<n; i++)
 {
  ll y; cin >> y;
  ans.pb(y);
 }

 vec sol = solve(A, ans);
 for (auto X : sol)
  cout << X << ' ';
 cout << endl;
}
```

6.5 General Linear Recursion

```
const int maxL = 20; // IF YOU WANT TO CONVOLVE TWO ARRAYS
     OF LENGTH N AND M CHOOSE LG IN SUCH A WAY THAT 2LG > n
     + m
const int maxN = 1 << maxL, MOD = 998244353;

#define M_PI acos(-1)

int root[maxL + 2] = {0,998244352,86583718,372528824,
 69212480,87557064,15053575,57475946,15032460,
 4097924,1762757,752127,299814,730033,227806,
 42058,44759,8996,2192,1847,646,42};

int bpow(int a, int b){
 int ans = 1;
 while (b){
  if (b & 1)
   ans = 1LL * ans * a % MOD;
  b >>= 1;
  a = 1LL * a * a % MOD;
 }
 return ans;
}

void ntt(vector<int> &a, bool inv){
 int LG = 0, z = 1, MAX = a.size();
 while(z != MAX) z *= 2, LG ++;
 int ROOT = root[LG];

 for (int mask = 0; mask < MAX; mask++){
  int rev = 0;
  for (int i = 0; i < LG; i++)
   if ((1 << i) & mask)
    rev |= (1 << (LG - 1 - i));
  if (mask < rev)
   swap(a[mask], a[rev]);
 }
 for (int len = 2; len <= MAX; len *= 2){
  int wn = bpow(ROOT, MAX / len);
  if (inv)
   wn = bpow(wn, MOD - 2);
  for (int i = 0; i < MAX; i += len){
   int w = 1;
   for (int j = 0; j < len / 2; j++){
    int l = a[i + j];
    int r = 1LL * w * a[i + j + len / 2] %
     MOD;
    a[i + j] = (l + r);
    a[i + j + len / 2] = l - r + MOD;
    if (a[i + j] >= MOD)
     a[i + j] -= MOD;
    if (a[i + j + len / 2] >= MOD)
     a[i + j + len / 2] -= MOD;
    w = 1LL * w * wn % MOD;
   }
  }
 }
 if (inv){
  int x = bpow(MAX, MOD - 2);
  for (int i = 0; i < MAX; i++)
   a[i] = 1LL * a[i] * x % MOD;
 }
}


int ans[maxN], bb[maxN];

//ans[i] = sum_j=1^i b_j * ans[i - j], ans[0] = 1;
void solve(int l, int r) {
 if(r - l == 1) return ;
 int mid = (l + r)/2;

 solve(l, mid);

 vector <int> a, b;
 for (int i=l; i<r; i++) {
  if(i < mid) a.pb(ans[i]);
  else a.pb(0);

  b.pb(bb[i-l+1]);
 }

 for (int i=l; i<r; i++) {
  a.pb(0);
  b.pb(0);
 }

 ntt(a, false);
 ntt(b, false);

 vector <int> c;
 c.resize(a.size());

 for (int i=0; i<2*r-2*l; i++)
  c[i] = 1LL * a[i] * b[i] % MOD;

 ntt(c, true);
 for (int i=0; i<r-mid; i++)
  ans[mid + i] += c[mid - l - 1 + i], ans[mid + i] %= MOD;

 solve(mid, r);
}

int main() {
 int n, m; cin >> n >> m;
 for (int i=1; i<=m; i++)
  cin >> bb[i];
 int k = 1;
 while(k < n) k = 2 * k;

 ans[0] = 1;
 solve(0, k);
 for (int i=0; i<n; i++)
  cout << ans[i] << ' ';
 cout << endl;
}
```

6.6 LP Duality

primal: Maximize $c^T x$ subject to $Ax \le b, x \ge 0$
dual: Minimize $b^T y$ subject to $A^T y \ge c, y \ge 0$

6.7 Popular LP

BellmanFord:

maximize X_n
$X_1 = 0$
and for eache edge ($v- > u$ and weight w):
$X_u - X_v \le w$

Flow:

maximize Σf_{out} (where *out* is output edges of vertex 1)
for each vertex (except 1 and n):
$\Sigma f_{in} - \Sigma f_{out} = 0$ (where *in* is input edges of v and *out* is output edges of v)

Dijkstra(IP):

minimize $\Sigma z_i * w_i$
for eache edge ($v- > u$ and weight w):
$0 \le z_i \le 1$
and for each ST-cut which vertex 1 is in S and vertex n is in T:
$\Sigma z_e \ge 1$ (for each edge e from S to T)

6.8 Simplex

```
typedef vector <ld> vd;
typedef vector <int> vi;
const ld Eps = 1e-9;

// ax <= b, max(cTx), x >= 0
// O(nm^2)

vd simplex(vector <vd> a, vd b, vd c) {
 int n = a.size(), m = a[0].size() + 1, r = n, s = m - 1;
 vector <vd> d(n + 2, vd(m + 1, 0)); vd x(m - 1);
 vi ix(n + m); iota(ix.begin(), ix.end(), 0);
 for(int i = 0; i < n; i ++) {
  for(int j = 0; j < m - 1; j ++) d[i][j] = -a[i][j];
  d[i][m - 1] = 1;
  d[i][m] = b[i];
  if(d[r][m] > d[i][m])
   r = i;
 }
 for(int j = 0; j < m - 1; j ++) d[n][j] = c[j];
 d[n + 1][m - 1] = -1;
 while(true) {
  if(r < n) {
   vd su;
   swap(ix[s], ix[r + m]); d[r][s] = 1 / d[r][s];
   for(int j = 0; j <=m; j ++) if(j != s) {
    d[r][j] *= -d[r][s]; if(d[r][j]) su.pb(j);
   }
   for(int i = 0; i <= n + 1; i ++) if(i != r) {
    for(int j = 0; j < su.size(); j ++)
     d[i][su[j]] += d[r][su[j]] * d[i][s];
    d[i][s] *= d[r][s];
   }
  }
  r = s = -1;
  for(int j = 0; j < m; j ++) if(s < 0 || ix[s] > ix[j])
   if(d[n + 1][j] > Eps || d[n + 1][j] > -Eps &&
     d[n][j] > Eps) s = j; if(s < 0) break;
  for(int i = 0; i < n; i ++) if(d[i][s] < -Eps) {
   if(r < 0) {
    r = i;
    continue;
   }
   double e = d[r][m] / d[r][s] - d[i][m] / d[i][s];
   if(e < -Eps || e < Eps && ix[r + m] > ix[i + m]) r = i;
  }
  if(r < 0)
  {return vd();} // Unbounded
 }
 if(d[n + 1][m] < -Eps) {return vd();}// No solution
 for(int i = m; i < n + m; i ++)
  if(ix[i] <m - 1) x[ix[i]] = d[i - m][m];
 return x;
}
```

6.9 Stirling

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

7 String

7.1 Aho Corasick

```
int nxt[N][C];
int f[N], q[N], vcnt;
vector<int> adj[N];

int add(string s)
{
 int cur = 0;
 for(auto ch : s)
 {
  ch -= 'a';
  if(!nxt[cur][ch]) nxt[cur][ch] = ++vcnt;
  cur = nxt[cur][ch];
 }
 return cur;
}

void aho()
{
 int hi = 0, lo = 0;
 for(int i = 0; i < C; i++) if(nxt[0][i]) q[hi++] = nxt[0][i
     ];
 while(hi != lo)
 {
  int x = q[lo++];
  adj[f[x]].pb(x);
  for(int i = 0; i < C; i++)
  {
   if(nxt[x][i])
   {
    q[hi++] = nxt[x][i];
    f[nxt[x][i]] = nxt[f[x]][i];
   }
   else nxt[x][i] = nxt[f[x]][i];
  }
 }
}
```

7.2 Palindromic

```
int n, last, sz;
char s[N];
int len[N], link[N], cnt[N];
map<short, int> to[N];
void init() {
 n = 0; last = 0;
 for(int i = 0; i < N; i++) to[i].clear();
 s[n++] = -1;
 link[0] = 1;
 len[1] = -1;
 sz = 2;
}
int get_link(int v) {
 while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
 return v;
}
void add_letter(int c) {
 s[n++] = c;
 last = get_link(last);
 if(!to[last][c]) {
  len [sz] = len[last] + 2;
  link[sz] = to[get_link(link[last])][c];
  to[last][c] = sz++;
 }
 last = to[last][c];
 cnt[last] = cnt[link[last]] + 1;
}
```

7.3 Suffix Array

```
string s;
int rank[LOG][N], n, lg;
pair<pair<int, int>, int> sec[N];
int sa[N];
int lc[N];

int lcp(int a, int b)
{
 int _a = a;
 for(int w = lg - 1; ~w && max(a, b) < n; w--)
  if(max(a, b) + (1 << w) <= n && rank[w][a] == rank[w][b])
   a += 1 << w, b += 1 << w;
 return a - _a;
}

int cnt[N];
pair<pii, int> gec[N];
void srt()
{
 memset(cnt, 0, sizeof cnt);
 for(int i = 0; i < n; i++) cnt[sec[i].F.S+1]++;
 for(int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
 for(int i = 0; i < n; i++) gec[--cnt[sec[i].F.S+1]] = sec[i
     ];
 memset(cnt, 0, sizeof cnt);
 for(int i = 0; i < n; i++) cnt[gec[i].F.F+1]++;
 for(int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
 for(int i = n - 1; ~i; i--) sec[--cnt[gec[i].F.F+1]] = gec[
     i];
}

void build()
{
 n = s.size();
 {
```

```
 int cur = 1; lg = 0;
 while(cur < n)
 {
  lg++;
  cur <<= 1;
 }
 lg++;
}

for(int i = 0; i < n; i++) rank[0][i] = s[i];
for(int w = 1; w < lg; w++)
{
 for(int i = 0; i < n; i++)
  if(i + (1 << w - 1) >= n)
   sec[i] = {{rank[w-1][i], -1}, i};
  else
   sec[i] = {{rank[w-1][i], rank[w-1][i+(1<<w-1)]}, i};
 srt();
 rank[w][sec[0].S] = 0;
 for(int i = 1; i < n; i++)
  if(sec[i].F == sec[i - 1].F)
   rank[w][sec[i].S] = rank[w][sec[i-1].S];
  else
   rank[w][sec[i].S] = i;
}

for(int i = 0; i < n; i++)
 sa[rank[lg-1][i]] = i;
for(int i = 0; i + 1 < n; i++)
 lc[i] = lcp(sa[i], sa[i + 1]);
}
```

7.4 Suffix Automata

```
const int maxn = 2 e5 + 42; // Maximum amount of states
map < char , int > to [ maxn ]; // Transitions
int link [ maxn ]; // Suffix links
int len [ maxn ]; // Lengthes of largest strings in states
int last = 0; // State corresponding to the whole string
int sz = 1; // Current amount of states
void add_letter ( char c ) { // Adding character to the end
 int p = last ; // State of string s
 last = sz ++; // Create state for string sc
 len [ last ] = len [ p ] + 1;
 for (; to [ p ][ c ] == 0; p = link [ p ]) // (1)
  to [ p ][ c ] = last ; // Jumps which add new suffixes
 if ( to [ p ][ c ] == last ) { // This is the first
      occurrence of
  c if we are here
   link [ last ] = 0;
  return ;
 }
 int q = to [ p ][ c ];
 if ( len [ q ] == len [ p ] + 1) {
  link [ last ] = q ;
  return ;
 }
 // We split off cl from q here
 int cl = sz ++;
 to [ cl ] = to [ q ]; // (2)
 link [ cl ] = link [ q ];
 len [ cl ] = len [ p ] + 1;
 link [ last ] = link [ q ] = cl ;
 for (; to [ p ][ c ] == q ; p = link [ p ]) // (3)
  to [ p ][ c ] = cl ; // Redirect transitions where needed
}
```

7.5 Suffix Tree

```
#define fpos adla
const int inf = 1e9;
const int maxn = 1e4; //maxn = number of states of suffix
     tree

char s[maxn];
map<int, int> to[maxn]; //edges of tree
int len[maxn], fpos[maxn], link[maxn];
//len[i] is the length of the inner edge of v
//fpos[i] is start position of inner edge in string s
int node, pos;
int sz = 1, n = 0;

int make_node(int _pos, int _len) {
 fpos[sz] = _pos;
 len [sz] = _len;
 return sz++;
}
void go_edge() {
 while(pos > len[to[node][s[n - pos]]]) {
  node = to[node][s[n - pos]];
  pos -= len[node];
 }
}
void add_letter(int c) {
 s[n++] = c;
 pos++;
 int last = 0;
 while(pos > 0) {
  go_edge();
  int edge = s[n - pos];
  int &v = to[node][edge];
  int t = s[fpos[v] + pos - 1];
  if(v == 0) {
   v = make_node(n - pos, inf);
   link[last] = node;
   last = 0;
  } else if(t == c) {
   link[last] = node;
   return;
  } else {
   int u = make_node(fpos[v], pos - 1);
   to[u][c] = make_node(n - 1, inf);
   to[u][t] = v;
   fpos[v] += pos - 1;
   len [v] -= pos - 1;
   v = u;
   link[last] = u;
   last = u;
  }
  if(node == 0)
   pos--;
  else
   node = link[node];
 }
}
```

8 Useful Fact and Constants

8.1 Long Long Long Integer

```
__int128 x;
unsigned __int128 y;
//Cin and Cout must be implemented
//Constants doesn't work
```