Team Notebook

Swarthmore College

September 27, 2021

Contents

1 3dHull

```
#include "Point3D.h"

typedef Point3D<double> P3;

struct PR {
 void ins(int x) { (a == -1 ? a : b) = x; }
 void rem(int x) { (a == x ? a : b) = -1; }
 int cnt() { return (a != -1) + (b != -1); }
 int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
 assert(sz(A) >= 4);
 vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
 vector<F> FS;
 auto mf = [&](int i, int j, int k, int l) {
  P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
  if (q.dot(A[l]) > q.dot(A[i]))
   q = q * -1;
  F f{q, i, j, k};
  E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
  FS.push_back(f);
 };
 rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
  mf(i, j, k, 6 - i - j - k);

 rep(i,4,sz(A)) {
  rep(j,0,sz(FS)) {
   F f = FS[j];
   if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
    E(a,b).rem(f.c);
    E(a,c).rem(f.b);
    E(b,c).rem(f.a);
    swap(FS[j--], FS.back());
    FS.pop_back();
   }
  }
  int nw = sz(FS);
  rep(j,0,nw) {
   F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.
     c);
   C(a, b, c); C(a, c, b); C(b, c, a);
  }
 }
 trav(it, FS) if ((A[it.b] - A[it.a]).cross(
  A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
 return FS;
};
```

2 AdvancedHash

```
ll base1[MX], base2[MX];


struct hsh {
    ll base, p1, p2;

    ll val1, val2;
    ll inv1, inv2;

  vl val1s, val2s;

  vl inVal1s, inVal2s;

 ll modExp(ll power, ll prime) {
        if (power == 0) {
            return 1;
        } else {
            ll cur = modExp(power / 2, prime); cur = cur *
                 cur; cur = cur % prime;
            if (power % 2 == 1) cur = cur * base;
            cur = cur % prime;
            return cur;
        }
    }

    hsh() {
    }


    hsh(ll b, string S) {
        p1 = 1000000007;
        p2 = 1000000009;
        base = b;
        val1 = 0;
        val2 = 0;

        inv2 = modExp(p2-2, p2);
        inv1 = modExp(p1-2, p1);
    val1s.pb(0); val2s.pb(0);
        FOR(i, sz(S)) {
            push_back(S[i] - 'a');
        }

    }

    void push_back(ll v) {
        val1 *= base;
        val1 %= p1;
        val1 += v;
        val1 %= p1;
        val2 *= base;
        val2 %= p2;
        val2 += v;
        val2 %= p2;

  val1s.pb(val1);
  val2s.pb(val2);
    }


  pl get(int L, int R) {
   ll A = (val1s[R+1] - (val1s[L] * base1[R-L+1]) % p1 + p1)
         % p1;
   ll B = (val2s[R+1] - (val2s[L] * base2[R-L+1]) % p2 + p2)
         % p2;
   return {A, B};
  }



};

  int base = uniform_int_distribution<int>(1000, MOD-2)(rng)
      ;

 base1[0] = 1; base2[0] = 1;
 FOR(i, 1, MX) {
  base1[i] = base1[i-1] * base; baseInv1[i] %= MOD;
  base2[i] = base2[i-1] * base; baseInv2[i] %= (MOD+2);
 }
```

3 AhoCorasick

```
struct AhoCorasick {
    static const int K = 26;

    struct Vertex {
        int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];
    ll value = -1;
    ll num = 0;

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t;

void init() {
    Vertex v; t.pb(v);
}



void add_string(string const& s, int count) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
    t[v].num = count;
}

ll dfs(int v) {
    if (t[v].value != -1) {
        return t[v].value;
    }
    ll ans = t[v].num;
    ans += dfs(get_link(v)); // is this right?
    return t[v].value = ans;
}

void compute() {
    t[0].value = 0;
    FOR(i, 1, sz(t)) {
        dfs(i);
    }
```

```
    }

    int get_link(int v) {
        if (t[v].link == -1) {
            if (v == 0 || t[v].p == 0)
                t[v].link = 0;
            else
                t[v].link = go(get_link(t[v].p), t[v].pch);
        }
        return t[v].link;
    }

    int go(int v, char ch) {
        int c = ch - 'a';
        if (t[v].go[c] == -1) {
            if (t[v].next[c] != -1)
                t[v].go[c] = t[v].next[c];
            else
                t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
        }
        return t[v].go[c];
    }
};
```

4 Angle

```
struct Angle {
 int x, y;
 int t;
 Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
 Angle operator-(Angle b) const { return {x-b.x, y-b.y, t};
      }
 int half() const {
  assert(x || y);
  return y < 0 || (y == 0 && x < 0);
 }
 Angle t90() const { return {-y, x, t + (half() && x >= 0)};
       }
 Angle t180() const { return {-x, -y, t + half()}; }
 Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
 // add a.dist2() and b.dist2() to also compare distances
 return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
        make_tuple(b.t, b.half(), a.x * (ll)b.y);
}
```

```
// Given two points, this calculates the smallest angle
     between
// them, i.e., the angle that covers the defined line
     segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
 if (b < a) swap(a, b);
 return (b < a.t180() ?
         make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
 Angle r(a.x + b.x, a.y + b.y, a.t);
 if (a.t180() < r) r.t--;
 return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
 int tu = b.t - a.t; a.t = b.t;
 return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)
      };
}
```

5 BIT

```
const int maxn = 30005;
int n, bit[maxn];
void add(int i, int x) {
    for (; i <= n; i += i & (-i))
        bit[i] += x;
}

int sum(int i) {
    int r = 0;
    for (; i; i -= i & (-i)) {
        r += bit[i];
    }
    return r;
}
```

6 BinSearchSegtree

```
const ll identity = 0;
const ll SZ = 131072;

ll sum[2*SZ], lazy[2*SZ];

ll combine(ll A, ll B) {
    return A+B;
}

ll combineUpd(ll A, ll B) {
    return A+B;
}

void push(int index, ll L, ll R) {
    sum[index] = combineUpd(sum[index], lazy[index]);
    if (L != R) lazy[2*index] = combineUpd(lazy[2*index],
         lazy[index]), lazy[2*index+1] = combineUpd(lazy[2*
         index+1], lazy[index]);
    lazy[index] = identity;
}

void pull(int index) {
    sum[index] = combine(sum[2*index], sum[2*index+1]);
}

bool checkCondition(int index) {
    //FILL THIS IN
}

ll query(int lo = 0, int hi = SZ-1, int index = 1, ll L = 0,
      ll R = SZ-1) { //returns first node satisfying con
    push(index, L, R);
    if (lo > R || L > hi) return -1;
    bool condition = checkCondition(index);
    if (L == R) {
        return (condition ? L : -1);
    }
    int M = (L+R) / 2;
    if (checkCondition(2*index+1)) {
        return query(lo, hi, 2*index+1, M+1, R);
    }
    return query(lo, hi, 2*index, L, M);
}

void update(int lo, int hi, ll increase, int index = 1, ll L
      = 0, ll R = SZ-1) {
    push(index, L, R);
    if (hi < L || R < lo) return;
    if (lo <= L && R <= hi) {
        lazy[index] = increase;
        push(index, L, R);
        return;
    }

    int M = (L+R) / 2;
    update(lo, hi, increase, 2*index, L, M); update(lo, hi,
         increase, 2*index+1, M+1, R);
    pull(index);
}
```

7 BipartiteMatching

```
//Storing the graph
vector<int> g[maxn];
//Storing whether we have visited a node
bool vis[maxn];
//Storing the vertex matched to
int match[maxn];

bool hungarian(int u){
  for (int i = 0;i < g[u].size();++i){
    int v = g[u][i];
    if (!vis[v]){
   vis[v] = true;
   if (!match[v] || hungarian(match[v])){
    match[u] = v; match[v] = u; return true;
   }
  }
 }
 return false;
}
//in main: call hungarian for each vertex on one side
for (int i = 1;i <= n1;++i){
  memset(vis,false,sizeof vis);
  if (hungarian(i)) ans++; //if we can match i
}
```

8 BipartiteMatchingWithWeights

```
ll g[maxn][maxn];
ll fx[maxn], fy[maxn], a[maxn], b[maxn], slack[maxn],pre[
     maxn];
bool visx[maxn], visy[maxn];
int q[maxn];
int n;

void augment(int v){
 if (!v) return; fy[v] = pre[v]; augment(fx[pre[v]]); fx[fy[
      v]] = v;
}

void bfs(int source){
 memset(visx, 0, sizeof visx);
 memset(visy, 0, sizeof visy);
 memset(slack, 127, sizeof slack);
 int head, tail; head = tail = 1;
 q[tail] = source;
 while (true){
  while (head <= tail){
   int u = q[head++];
   visx[u] = true;
   for (int v = 1;v <= n;++v){
    if (!visy[v]){
     if (a[u] + b[v] == g[u][v])
     {
      visy[v] = true; pre[v] = u;
      if (!fy[v]){
       augment(v); return;
      }
      q[++tail] = fy[v];continue;
     }
     if (slack[v] > a[u] + b[v] - g[u][v]){
      slack[v] = a[u] + b[v] - g[u][v];
      pre[v] = u;
     }
    }
   }
  }
  ll d = inf;
  for (int i = 1;i <= n;++i){
   if (!visy[i]) d = min(d, slack[i]);
  }
  for (int i = 1;i <= n;++i){
   if (visx[i]) a[i] -= d;
   if (visy[i]) b[i] += d;
   else slack[i] -= d;
  }
  for (int v = 1;v <= n;++v){
   if (!visy[v] && !slack[v]){
    visy[v] = true;
    if (!fy[v]){
     augment(v);
     return;
    }
    q[++tail] = fy[v];
   }
  }
 }
}
```

```
ll km(){
 for (int i = 1;i <= n;++i){
  a[i] = -inf;
  b[i] = 0;
  for (int j = 1;j <= n;++j) a[i] = max(a[i], g[i][j]);
 }
 memset(fx, 0, sizeof fx);
 memset(fy, 0, sizeof fy);
 for (int i = 1;i <= n;++i) bfs(i);
 ll ans = 0;
 for (int i = 1;i <= n;++i) ans += a[i] + b[i];
 //vertex i on left is matched to g2[i][fx[i]] * fx[i]
 //g2[a][b]=1 iff exists edge ab
 return ans;
}
```

9 Bridge

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

10 CRT

```
//each is x mod p_i = a_i
ll p[maxn], a[maxn];
//for quickmult see pollard rho
ll exgcd(ll x, ll y,ll & a, ll & b){
    if (y == 0){
        a = 1;b = 0;return x;
    }
    ll d = exgcd(y, x%y, a, b);
    ll temp = a; a = b; b = temp - (x / y) * b;
    return d;
}

int first_nontrivial = 0;
ll current_p ;
ll sol = 0; //this is the solution
for (int i = 1;i <= n;i++){
  if (p[i] != 1){
    first_nontrivial = i;
    current_p = p[i]; sol = a[i];
    break;
  }
}
for (int i = first_nontrivial+1;i <= n;i++){
  ll x,y;
  if (p[i] == 1) continue;
  ll d = exgcd(current_p, p[i], x, y);
  ll r = ((a[i] - sol) % p[i] + p[i])%p[i];
  ll temp = quickmult(x, r / d,p[i] / d);
  sol = sol + current_p * temp;
  current_p = current_p / d * p[i];
  sol = (sol % current_p + current_p) % current_p;
}
```

11 CentroidDecomp

```
struct CentroidDecomposition {
 vector<set<int>> tree; // it's not vector<vector<int>>!
 vector<int> dad;
 vector<int> sub;
 vector<int> dep;

CentroidDecomposition(vector<set<int>> &tree) : tree(tree)
     {
 int n = tree.size();
 dad.resize(n);
 sub.resize(n);
       dep.resize(n);
 build(0, -1);
}

void build(int u, int p) {
 int n = dfs(u, p); // find the size of each subtree
 int centroid = dfs(u, p, n); // find the centroid
      if (p == -1) {
          dep[centroid] = 0;
      } else {
          dep[centroid] = dep[p] + 1;
      }
 if (p == -1) p = centroid; // dad of root is the root
     itself
 dad[centroid] = p;
 // for each tree resulting from the removal of the
     centroid
      while (!tree[centroid].empty()) {
          int v = *(tree[centroid].begin());
  tree[centroid].erase(v); // remove the edge to disconnect
  tree[v].erase(centroid); // the component from the tree
  build(v, centroid);
      }

}

int dfs(int u, int p) {
 sub[u] = 1;

 for (auto v : tree[u])
  if (v != p) sub[u] += dfs(v, u);

 return sub[u];
}

int dfs(int u, int p, int n) {
 for (auto v : tree[u])
  if (v != p and sub[v] > n/2) return dfs(v, u, n);

 return u;
}

int operator[](int i) {
 return dad[i];
 }
};
```

12 CircleIntersection

```
#include "Point.h"

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out
     ) {
 if (a == b) { assert(r1 != r2); return false; }
 P vec = b - a;
 double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
       p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
 if (sum*sum < d2 || dif*dif > d2) return false;
 P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2
      );
 *out = {mid + per, mid - per};
 return true;
}
```

13 CircleLine

```
#include "Point.h"
#include "lineDistance.h"
#include "LineProjectionReflection.h"

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
 double h2 = r*r - a.cross(b,c)*a.cross(b,c)/(b-a).dist2();
 if (h2 < 0) return {};
 P p = lineProj(a, b, c), h = (b-a).unit() * sqrt(h2);
 if (h2 == 0) return {p};
 return {p - h, p + h};
}
```

14 CirclePolygonIntersection

```
#include "../../content/geometry/Point.h"
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
 auto tri = [&](P p, P q) {
  auto r2 = r * r / 2;
  P d = q - p;
  auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2()
      ;
  auto det = a * a - b;
  if (det <= 0) return arg(p, q) * r2;
  auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
  if (t < 0 || 1 <= s) return arg(p, q) * r2;
  P u = p + d * s, v = p + d * t;
  return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
 };
 auto sum = 0.0;
 rep(i,0,sz(ps))
  sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
 return sum;
}
```

15 CircleTangents

```
#include "Point.h"

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2
     ) {
 P d = c2 - c1;
 double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
 if (d2 == 0 || h2 < 0) return {};
 vector<pair<P, P>> out;
 for (double sign : {-1, 1}) {
  P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
  out.push_back({c1 + v * r1, c2 + v * r2});
 }
 if (h2 == 0) out.pop_back();
 return out;
}
```

16 Circumcircle

```
#include "Point.h"

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
 return (B-A).dist()*(C-B).dist()*(A-C).dist()/
   abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
 P b = C-A, c = B-A;
 return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

17 ClosestPair

```
#include "Point.h"

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
 assert(sz(v) > 1);
 set<P> S;
 sort(all(v), [](P a, P b) { return a.y < b.y; });
 pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
 int j = 0;
 trav(p, v) {
  P d{1 + (ll)sqrt(ret.first), 0};
  while (v[j].y <= p.y - d.x) S.erase(v[j++]);
  auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
  for (; lo != hi; ++lo)
   ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
  S.insert(p);
 }
 return ret.second;
}
```

18 ConvexHull

```
#include "Point.h"

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
 if (sz(pts) <= 1) return pts;
 sort(all(pts));
 vector<P> h(sz(pts)+1);
 int s = 0, t = 0;
 for (int it = 2; it--; s = --t, reverse(all(pts)))
  trav(p, pts) {
   while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
   h[t++] = p;
  }
 return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])
     };
}
```

19 ConvexHullTrick

```
//This represents those relying on j only;i.e intercept
inline ll y_axis(int j){
    return dp[j] + a * s[j] * s[j] - b * s[j];
}
//This represents those relying on both i and j; i.e slope
inline ll x_axis(int j){
    return s[j];
}
inline ld getSlope(int j,int k){
    ld y = y_axis(k) - y_axis(j);
    ld x = x_axis(k) - x_axis(j);
    return y / x;
}
//s stores prefix sum
int head = 1;int tail = 1;
q[head] = 0;
for (int i = 1;i <= n;i++){
  dp[i] = a*s[i]*s[i] + b*s[i] + c;
  while (head < tail && getSlope(q[head],q[head+1]) >= 2*a*s
       [i]) head++;
  if (head <= tail){
    int j = q[head];
    dp[i] = max(dp[i], dp[j] + a *(s[i] - s[j]) * (s[i] - s[j
        ]) + b*(s[i] - s[j]) + c);
  }
  while (head < tail && getSlope(q[tail],i) >= getSlope(q[
       tail-1],q[tail])) tail--;
  q[++tail] = i;
}
```

20 Cutpoints

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

21 DSU

```
template<int SZ> struct DSU {
    int parent[SZ], size[SZ];

    DSU() {
        FOR(i, SZ) parent[i] = i, size[i] = 0;
    }

    int get(int x) {
        if (parent[x] != x) parent[x] = get(parent[x]);
        return parent[x];
    }

    void unify(int x, int y) {
        x = get(x); y = get(y);
        if (x == y) return;
        if (size[x] < size[y]) swap(x, y);
        if (size[x] == size[y]) size[x]++;
        parent[y] = x;

    }
};
```

22 DelaunayTriangulation

```
#include "Point.h"
#include "3dHull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
 if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
  trifun(0,1+d,2-d); }
 vector<P3> p3;
 trav(p, ps) p3.emplace_back(p.x, p.y, p.dist2());
 if (sz(ps) > 3) trav(t, hull3d(p3)) if ((p3[t.b]-p3[t.a]).
   cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
  trifun(t.a, t.c, t.b);
}
```

23 Dinic

```
//from https://cp-algorithms.com/graph/dinic.html
//Complexity: O(E*V^2)
struct FlowEdge {
   int v, u;
   long long cap, flow = 0;
   FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
        cap) {}
};

struct Dinic {
   const long long flow_inf = 1e18;
   vector<FlowEdge> edges;
   vector<vector<int>> adj;
   int n, m = 0;
   int s, t;
   vector<int> level, ptr;
   queue<int> q;

   Dinic(int n, int s, int t) : n(n), s(s), t(t) {
       adj.resize(n);
       level.resize(n);
       ptr.resize(n);
}

void add_edge(int v, int u, long long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
}

bool bfs() {
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap - edges[id].flow < 1)
                continue;
            if (level[edges[id].u] != -1)
                continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}

long long dfs(int v, long long pushed) {
    if (pushed == 0)
        return 0;
    if (v == t)
        return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
         ++) {
        int id = adj[v][cid];
        int u = edges[id].u;
        if (level[v] + 1 != level[u] || edges[id].cap -
             edges[id].flow < 1)
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap -
             edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}

long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

24 EulerPath

```
int N, M;
vector<vpi> graph(MX); //{ed, edNum}
vector<vpi::iterator> its(MX);
vector<bool> used(MX);

vpi eulerPath(int r) {
    FOR(i, N) its[i] = begin(graph[i]);
    FOR(i, M) used[i] = false;
    vpi ans, s{{r, -1}};
    int lst = -1;
    while (sz(s)) {
        int x = s.back().f; auto &it = its[x], en = end(graph
             [x]);
        while (it != en && used[it->s]) it++;

        if (it == en) {
            if (lst != -1 && lst != x) return {};
            ans.pb(s.back()); s.pop_back(); if (sz(s)) lst =
                 s.back().f;
        } else {
            s.pb(*it);
            used[it->s] = 1;
        }
    }
    if (sz(ans) != M+1) return {};
    return ans;
}
```

25 FastDelaunay

```
#include "Point.h"

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
 bool mark; Q o, rot; P p;
 P F() { return r()->p; }
 Q r() { return rot->rot; }
 Q prev() { return rot->o->rot; }
 Q next() { return r()->prev(); }
};

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
 lll p2 = p.dist2(), A = a.dist2()-p2,
     B = b.dist2()-p2, C = c.dist2()-p2;
 return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
      0;
}
Q makeEdge(P orig, P dest) {
 Q q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
          new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
 rep(i,0,4)
  q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
 return *q;
}
void splice(Q a, Q b) {
 swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
 Q q = makeEdge(a->F(), b->p);
 splice(q, a->next());
 splice(q->r(), b);
 return q;
}

pair<Q,Q> rec(const vector<P>& s) {
 if (sz(s) <= 3) {
  Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
  if (sz(s) == 2) return { a, a->r() };
  splice(a->r(), b);
  auto side = s[0].cross(s[1], s[2]);
  Q c = side ? connect(b, a) : 0;
  return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
 }
```

```cpp
#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
 Q A, B, ra, rb;
 int half = sz(s) / 2;
 tie(ra, A) = rec({all(s) - half});
 tie(B, rb) = rec({sz(s) - half + all(s)});
 while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
 Q base = connect(B->r(), A);
 if (A->p == ra->p) ra = base->r();
 if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
  while (circ(e->dir->F(), H(base), e->F())) { \
   Q t = e->dir; \
   splice(e, e->prev()); \
   splice(e->r(), e->r()->prev()); \
   e = t; \
  }
 for (;;) {
  DEL(LC, base->r(), o); DEL(RC, base, prev());
  if (!valid(LC) && !valid(RC)) break;
  if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
   base = connect(RC, base->r());
  else
   base = connect(base->r(), LC->r());
 }
 return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
 sort(all(pts)); assert(unique(all(pts)) == pts.end());
 if (sz(pts) < 2) return {};
 Q e = rec(pts).first;
 vector<Q> q = {e};
 int qi = 0;
 while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p)
     ; \
 q.push_back(c->r()); c = c->next(); } while (c != e); }
 ADD; pts.clear();
 while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
 return pts;
}
```

26 FastHashTable

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
```

27 GeneralMatching

```cpp
//belong is a DSU; unit = union
int n, match[N], next[N], mark[N], vis[N], Q[N];
std::vector<int> e[N];
int rear;

int LCA(int x, int y){
  static int t = 0; t++;
  while (true) {
    if (x != -1) {
      x = findb(x);
      if (vis[x] == t) return x;
      vis[x] = t;
      if (match[x] != -1) x = next[match[x]];
      else x = -1;
    }
    std::swap(x, y);
  }
}

void group(int a, int p){
    while (a != p){
      int b = match[a], c = next[b];
      if (findb(c) != p) next[c] = b;
      if (mark[b] == 2) mark[Q[rear++] = b] = 1;
      if (mark[c] == 2) mark[Q[rear++] = c] = 1;
      unit(a, b); unit(b, c);
      a = c;
    }
}

void aug(int s){
  for (int i = 0; i < n; i++)
    next[i] = -1, belong[i] = i, mark[i] = 0, vis[i] = -1;
  mark[s] = 1;
  Q[0] = s; rear = 1;
  for (int front = 0; match[s] == -1 && front < rear; front
       ++){
    int x = Q[front];
    for (int i = 0; i < (int)e[x].size(); i++){
      int y = e[x][i];
      if (match[x] == y) continue;
      if (findb(x) == findb(y)) continue;
      if (mark[y] == 2) continue;
      if (mark[y] == 1){
        int r = LCA(x, y);
        if (findb(x) != r) next[x] = y;
        if (findb(y) != r) next[y] = x;
        group(x, r);
        group(y, r);
      }
      else if (match[y] == -1){
        next[y] = x;
        for (int u = y; u != -1; ){
          int v = next[u];
          int mv = match[v];
          match[v] = u, match[u] = v;u = mv;
        }
        break;
      }
      else{
        next[y] = x;
        mark[Q[rear++] = match[y]] = 1;
        mark[y] = 2;
      }
    }
  }
}
//the graph is stored as e[N] and g[N]
for (int i = 0; i < n; i++) match[i] = -1;
for (int i = 0; i < n; i++) if (match[i] == -1) aug(i);
int tot = 0;
for (int i = 0; i < n; i++){
  if (match[i] != -1) tot++;
}
//matched pairs = tot/2
printf("%d\n", tot/2);
for (int i = 0; i < n; i++){
    printf("%d ", match[i] + 1);
}
```

28 GeometrySnippets

```cpp
/*
They are from KACTL, KTH's Team Reference Document
*/

//check point in convex hull
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
 int a = 1, b = sz(l) - 1, r = !strict;
 if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
 if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
 if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -
     r)
  return false;
 while (abs(a - b) > 1) {
  int c = (a + b) / 2;
  (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
 }
 return sgn(l[a].cross(l[b], p)) < r;
}

//center of mass of polygon
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
 P res(0, 0); double A = 0;
 for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
  res = res + (v[i] + v[j]) * v[j].cross(v[i]);
  A += v[j].cross(v[i]);
 }
 return res / A / 3;
}

// Returns a vector with the vertices of a polygon with
     everything to the left
// of the line going from s to e cut away.
ypedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
 vector<P> res;
 rep(i,0,sz(poly)) {
  P cur = poly[i], prev = i ? poly[i-1] : poly.back();
  bool side = s.cross(e, cur) < 0;
  if (side != (s.cross(e, prev) < 0))
   res.push_back(lineInter(s, e, cur, prev).second);
  if (side)
   res.push_back(cur);
 }
 return res;
}

//volumn of polyhedron, with face outwards
template<class V, class L>
double signed_poly_volume(const V& p, const L& trilist) {
 double v = 0;
 trav(i, trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
 return v / 6;
}

//intersection of two lines
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
 auto d = (e1 - s1).cross(e2 - s2);
 if (d == 0) // if parallel
  return {-(s1.cross(e1, s2) == 0), P(0, 0)};
 auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
 return {1, (s1 * p + e1 * q) / d};
}

// Returns where $p$ is as seen from $s$ towards $e$. 1/0/-1
      $\Leftrightarrow$ left/on line/right.

template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
 auto a = (e-s).cross(p-s);
 double l = (e-s).dist()*eps;
 return (a > l) - (a < -l);
}

//f is longitude, t is latitude
double sphericalDistance(double f1, double t1,
 double f2, double t2, double radius) {
 double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
 double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
 double dz = cos(t2) - cos(t1);
 double d = sqrt(dx*dx + dy*dy + dz*dz);
 return radius*2*asin(d/2);
}
```

29 HullDiameter

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
 int n = sz(S), j = n < 2 ? 0 : 1;
 pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
 rep(i,0,j)
  for (;; j = (j + 1) % n) {
   res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
   if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
    break;
  }
 return res.second;
}
```

30 InsidePolygon

```
#include "Point.h"
#include "OnSegment.h"
#include "SegmentDistance.h"

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
 int cnt = 0, n = sz(p);
 rep(i,0,n) {
  P q = p[(i + 1) % n];
  if (onSegment(p[i], q, a)) return !strict;
  //or: if (segDist(p[i], q, a) <= eps) return !strict;
  cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
 }
 return cnt;
}
```

31 KMP

```
int nxt[maxn]; //next array
/*Core KMP @param nxt: the resulting "next" array*/
void build_nxt(char * str,int * nxt, int length){
 int k = 0;
 nxt[0] = 0;
 for (int i = 1;i < length;i++){
  while (k > 0 && str[k] != str[i]){
   k = nxt[k - 1];
  }
  if (str[k] == str[i]){
   k++;
  }
  nxt[i] = k;
 }
}
/*Matching the string with the pattern
@return number of occurences of pattern string in the
     original string
*/
int match(char * str, char * pattern, int length_str, int
     length_pattern){
 int total = 0;
 int p = 0;
 for (int i = 0;i < length_str;i++){
  while (p > 0 && pattern[p] != str[i]){
   p = nxt[p - 1];
  }
  if (pattern[p] == str[i]){
   p++;
  }
  if (p == length_pattern){
   total++;
   p = nxt[p - 1];
  }
 }
 return total;
}
//build nxt for pattern
build_nxt(pattern, nxt, len_pattern);
```

32 KdTree

```
/**
 * Author: Stanford
 * Date: Unknown
 * Source: Stanford Notebook
 * Description: KD-tree (2d, can be extended to 3d)
 * Status: Untested, but works for Stanford
 */
#pragma once

#include "Point.h"

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
 P pt; // if this is a leaf, the single point in it
 T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
 Node *first = 0, *second = 0;

 T distance(const P& p) { // min squared distance to a point
  T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
  T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
  return (P(x,y) - p).dist2();
 }

 Node(vector<P>&& vp) : pt(vp[0]) {
  for (P p : vp) {
   x0 = min(x0, p.x); x1 = max(x1, p.x);
   y0 = min(y0, p.y); y1 = max(y1, p.y);
  }
  if (vp.size() > 1) {
   // split on x if width >= height (not ideal...)
   sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
   // divide by taking half the array for each child (not
   // best performance with many duplicates in the middle)
   int half = sz(vp)/2;
   first = new Node({vp.begin(), vp.begin() + half});
   second = new Node({vp.begin() + half, vp.end()});
  }
 }
};

struct KDTree {
 Node* root;
 KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

 pair<T, P> search(Node *node, const P& p) {
  if (!node->first) {
   // uncomment if we should not find the point itself:
   // if (p == node->pt) return {INF, P()};
   return make_pair((p - node->pt).dist2(), node->pt);
  }

  Node *f = node->first, *s = node->second;
  T bfirst = f->distance(p), bsec = s->distance(p);
  if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

  // search closest side first, other side if needed
  auto best = search(f, p);
  if (bsec < best.first)
   best = min(best, search(s, p));
  return best;
 }

 // find nearest point to a point, and its squared distance
 // (requires an arbitrary operator< for Point)
 pair<T, P> nearest(const P& p) {
  return search(root, p);
 }
};
```

33 LCA

```
int L; //SET THIS TO CEIL(LOG(MX_N))
int anc[MX][L];
int depth[MX];
int parent[MX];

int LCA(int a, int b) {
    if (depth[a] < depth[b]) {
        int c = b;
        b = a;
        a = c;
    }

    int dist = depth[a] - depth[b];
    while (dist > 0) {
        FOR(i, L) {
            if (dist & 1 << i) {
                a = anc[a][i];
                dist -= 1 << i;
            }
        }
    }

    if (a == b) return a;

    FORd(j, L) {
        if (anc[a][j] != -1 && anc[a][j] != anc[b][j]) {
            a = anc[a][j]; b = anc[b][j];
        }
    }
    return parent[a];
}

void preprocess() {
    //make sure to compute parents and depths
    FOR(i, N) FOR(j, L) anc[i][j] = -1;
    FOR(i, N) anc[i][0] = parent[i];
    FOR(j, 1, L) {
        FOR(i, N) {
            if (anc[i][j-1] != -1) {
                anc[i][j] = anc[anc[i][j-1]][j-1];
            }
        }
    }
}
```

34 LCT

```
/*
LCT is short for link-cut tree.
It is a powerful data structure that supports dynamic trees
     and sometimes dynamic graphs.
```

```
(it supports adding an edge to a forest or removing an edge
     from a forest)
LCT is based on splay tree, so you will find many
     similarities between these two data structures.
*/

#include <cstdio>
#include <cstdlib>
#include <algorithm>

using namespace std;

typedef unsigned int uint;
const int maxn = 110000; //maximum number of vertices in the
      tree
const uint modi = 51061;

struct rec
{
    int ls, rs, p; //ls = left son; rs = right son; p =
         parent
    uint siz; //siz = size of the subtree
    uint key, sum; //sum: sum of weights in the subtree
    uint mult, add; //two lazy tags
    bool rev; //denote whether this segment has been reverted
};
rec splay[maxn];

void clear(){
    splay[0].p = splay[0].ls = splay[0].rs = splay[0].rev =
         splay[0].key = splay[0].sum = 0;
    splay[0].siz = 0;
}

void update(int x){
    clear();
    splay[x].sum = splay[splay[x].ls].sum + splay[splay[x].rs
         ].sum + splay[x].key;
    splay[x].sum %= modi;
    splay[x].siz = splay[splay[x].ls].siz + splay[splay[x].rs
         ].siz + 1;
    splay[x].siz %= modi;
}

void zig(int x){
    int y = splay[x].p, z = splay[y].p;
    if (y == splay[z].ls) splay[z].ls = x;
    else if (y == splay[z].rs) splay[z].rs = x;
    splay[x].p = z;
    if (splay[x].rs) splay[splay[x].rs].p = y;
    splay[y].ls = splay[x].rs;
    splay[x].rs = y;
    splay[y].p = x;
    update(y);
}

void zag(int x){
    int y = splay[x].p, z = splay[y].p;
    if (y == splay[z].ls) splay[z].ls = x;
    else if (y == splay[z].rs) splay[z].rs = x;
    splay[x].p = z;
    if (splay[x].ls) splay[splay[x].ls].p = y;
    splay[y].rs = splay[x].ls;
    splay[x].ls = y;
    splay[y].p = x;
    update(y);
}

bool is_root(int x){
    return x != splay[splay[x].p].ls && x != splay[splay[x].p
         ].rs;
}

void rev(int x){
    if (!x)return;
    swap(splay[x].ls, splay[x].rs);
    splay[x].rev ^= true;
}

void pushdown(int x){
    if (splay[x].rev){
        rev(splay[x].ls);
        rev(splay[x].rs);
        splay[x].rev = false;
    }
    //Todo: Push lazy tags here.
}
void set_root(int x){
    static int q[maxn];
    static int top;
    int i;
    for (i = x; !is_root(i); i = splay[i].p){
        q[++top] = i;
    }
    q[++top] = i;
    while (top){
        pushdown(q[top--]);
    }
    while (!is_root(x)){
        int y = splay[x].p;
        if (is_root(y)){
            if (x == splay[y].ls) zig(x); else zag(x);
        }
        else{
            int z = splay[y].p;
            if (y == splay[z].ls){
                if (x == splay[y].ls) zig(y), zig(x);
                else zag(x), zig(x);
            }
            else{
                if (x == splay[y].rs) zag(y), zag(x);
                else zig(x), zag(x);
            }
        }
    }
    update(x);
}

//this is a special operation on LCT
void access(int x)
{
    for (int t = 0; x; t = x, x = splay[x].p){
        set_root(x);
        splay[x].rs = t;
        update(x);
    }
}

//we will make x be the new root of the tree it belongs to
void makeroot(int x){
    access(x);
    set_root(x);
    rev(x);
}

void split(int x, int y){
    makeroot(x);
    access(y);
    set_root(y);
}

//link vertex x and vertex y
void link(int x, int y){
    makeroot(x);
    makeroot(y);
    splay[x].p = y;
}

//cut the edge between x and y
void cut(int x, int y){
    split(x, y);
    splay[y].ls = splay[x].p = 0;
    update(y);
}

//Adding edge between u and v: link(u, v);
//Removing edge between u and v: link(u1, v1);
//Adding vertices on route between u and v by c :
/* split(u, v);
   calc(v, 1, c);*/
//Query the sum on route from u to v: split(u1,v1) print(
     splay[v1].sum);
```

35 LazySegtree

```
const ll identity = 0;
const ll SZ = 131072;

ll sum[2*SZ], lazy[2*SZ];

ll combine(ll A, ll B) {
    return A+B;
}

ll combineUpd(ll A, ll B) {
    return A+B;
}

void push(int index, ll L, ll R) {
    sum[index] = combineUpd(sum[index], lazy[index]);
    if (L != R) lazy[2*index] = combineUpd(lazy[2*index],
         lazy[index]), lazy[2*index+1] = combineUpd(lazy[2*
         index+1], lazy[index]);
    lazy[index] = identity;
}

void pull(int index) {
    sum[index] = combine(sum[2*index], sum[2*index+1]);
}

ll query(int lo, int hi, int index = 1, ll L = 0, ll R = SZ
     -1) {
    push(index, L, R);
    if (lo > R || L > hi) return 0;
    if (lo <= L && R <= hi) return sum[index];

    int M = (L+R) / 2;
    return combine(query(lo, hi, 2*index, L, M), query(lo, hi
         , 2*index+1, M+1, R));
}

void update(int lo, int hi, ll increase, int index = 1, ll L
      = 0, ll R = SZ-1) {
    push(index, L, R);
    if (hi < L || R < lo) return;
    if (lo <= L && R <= hi) {
        lazy[index] = increase;
        push(index, L, R);
        return;
    }

    int M = (L+R) / 2;
    update(lo, hi, increase, 2*index, L, M); update(lo, hi,
         increase, 2*index+1, M+1, R);
    pull(index);
}
```

36 LeftistTree

```
struct node{
    node *l,*r;
    //key is the priority
    int key,id;
    //distant to the leftist child - it is used to maintain
         the properties of the lefitst tree
    int dist;
    int rdist(){return (r==NULL)?0:r->dist;}
    int ldist(){return (l==NULL)?0:l->dist;}
};
node* merge(node*l,node*r)
{
    if (l == NULL) return r;
    if (r == NULL) return l;
    //we want to make sure the root has the smallest key
    if (l->key > r->key) swap(l,r);
    l->r = merge(l->r,r);
    //maintain the properties of the leftist tree
    if (l->ldist() < l->rdist()) swap(l->l,l->r);
    l->dist = l->rdist()+1;
    return l;
}
```

37 LineDistance

```
/**
Returns the signed distance between point p and the line
     containing points a and b. Positive value on left side
     and negative on right as seen from a towards b. a==b
     gives nan. P is supposed to be Point<T> or Point3D<T>
     where T is e.g. double or long long. It uses products
     in intermediate steps so watch out for overflow if
     using int or long long. Using Point3D will always give
     a non-negative distance. For Point3D, call .dist on the
      result of the cross product.
#include "Point.h"
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
 return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

38 LineHullIntersection

```
/**
 * lineHull(line, poly) returns a pair describing the
      intersection of a line with the polygon:
 *    \item $(-1, -1)$ if no collision,
 *    \item $(i, -1)$ if touching the corner $i$,
 *    \item $(i, i)$ if along side $(i, i+1)$,
 *    \item $(i, j)$ if crossing sides $(i, i+1)$ and $(j, j
      +1)$.
 * Time: O(N + Q \log n)
 */
#include "Point.h"

typedef array<P, 2> Line;
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n
     ]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
int extrVertex(vector<P>& poly, P dir) {
 int n = sz(poly), lo = 0, hi = n;
 if (extr(0)) return 0;
 while (lo + 1 < hi) {
  int m = (lo + hi) / 2;
  if (extr(m)) return m;
  int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
  (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
 }
 return lo;
}
```

```
#define cmpL(i) sgn(line[0].cross(poly[i], line[1]))
array<int, 2> lineHull(Line line, vector<P> poly) {
 int endA = extrVertex(poly, (line[0] - line[1]).perp());
 int endB = extrVertex(poly, (line[1] - line[0]).perp());
 if (cmpL(endA) < 0 || cmpL(endB) > 0)
  return {-1, -1};
 array<int, 2> res;
 rep(i,0,2) {
  int lo = endB, hi = endA, n = sz(poly);
  while ((lo + 1) % n != hi) {
   int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
   (cmpL(m) == cmpL(endB) ? lo : hi) = m;
  }
  res[i] = (lo + !cmpL(hi)) % n;
  swap(endA, endB);
 }
 if (res[0] == res[1]) return {res[0], -1};
 if (!cmpL(res[0]) && !cmpL(res[1]))
  switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
   case 0: return {res[0], res[0]};
   case 2: return {res[1], res[1]};
  }
 return res;
}
```

39 LinearProgramming

```
/*solving linear programming with simplex algorithm.
Problem: Given the i-th condition to be \sum_{j=1}^n a_{ij}
     x_j \leq b_i
x_j \geq 0 (m conditions in total)
Find X-j that maximizes \sum_{j=1}^n c_jx_j*/
const double eps = 1e-8;
const double inf = 1e100;

double a[maxn][maxn], b[maxn], c[maxn], ans[maxn];
int id[maxn << 1];
int n, m; //n = num of variables; m = constraints

void pivot(int base,int base2){
 swap(id[base + n], id[base2]);
 for (int j = 0;j <= n;++j){
  if (j != base2){
   a[base][j] /= a[base][base2];
  }
 }
 a[base][base2] = 1 / a[base][base2];
 for (int i = 0;i <= m;++i){
  if (i != base && fabs(a[i][base2]) > eps){
   for (int j = 0;j <= n;++j){
    if (j != base2)
     a[i][j] -= a[base][j] * a[i][base2];
   }
   a[i][base2] *= -a[base][base2];
  }
 }
}

bool init(){
 for (int i = 1;i <= n + m;++i)
  id[i] = i;
 while (true){
  int x = 0, y = 0;
  double mini = 0;
  for (int i = 1;i <= m;++i){
   if (a[i][0] < mini - eps){
    x = i;
    mini = a[i][0];
   }
  }
  if (!x) break;
  for (int j = 1;j <= n;++j){
   if (a[x][j] < -eps) y = j;
  }
  if (!y) return false;
  pivot(x, y);
 }
 return true;
}

bool simplex(){
 while (true){
  int x = 0, y = 0;
  double mini = inf;
  for (int j = 1;j <= n;++j){
   if (a[0][j] > eps){
    y = j;
    break;
   }
  }
  if (!y) break;
  for (int i = 1;i <= m;++i){
   if (a[i][y] > eps && a[i][0] / a[i][y] < mini - eps){
    x = i;
    mini = a[i][0] / a[i][y];
   }
  }
  if (!x) return false;
  pivot(x, y);
 }
 return true;
}

//main program
for (int i = 1;i <= n;++i) a[0][i] = c[i];
for (int i = 1;i <= m;++i) a[i][0] = b[i];
if (!init()){
 printf("Infeasible\n");return 0;
}
else if (!simplex()){
 printf("Unbounded\n");return 0;
}
else{
 printf("%.10lf\n", -a[0][0]);
 for (int i = 1;i <= m;++i){
  if (id[n + i] <= n) ans[id[n + i]] = a[i][0];
 }
 if (t){
   for (int i = 1;i <= n;++i) printf("%.10lf ", ans[i]);
   printf("\n");
  }
 }
 return 0;
}
```

40 LinearTransformation

```
/**
 * Author: Per Austrin, Ulf Lundstrom
 * Date: 2009-04-09
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
 Apply the linear transformation (translation, rotation and
      scaling) which takes line p0-p1 to line q0-q1 to point
      r.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-8mm}
\includegraphics[width=\textwidth]{content/geometry/
     linearTransformation}
\vspace{-2mm}
\end{minipage}
 * Status: not tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
  const P& q0, const P& q1, const P& r) {
 P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
 return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2
     ();
}
```

41 Manacher

```
#include <set>
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

//you need twice the length of original string here, since
     you need space for adding $
const int maxn = 2100000;

char str[maxn], str2[maxn];
//p[i] is just "how long you can extend from i in both ways"
int p[maxn];

void manacher(int n)
{
    int mx = 0;
    int id = 0;
    for (int i = 1;i <= n;++i){
        if (mx >= i){
            p[i] = min(mx - i, p[2 * id - i]);
        }
        else{
            p[i] = 0;
        }
        for (;str[i + p[i] + 1] == str[i - p[i] - 1];){
            p[i]++;
        }
        if (p[i] + i > mx){
            id = i;
            mx = p[i] + i;
        }
    }
}
```

42 Matrix

```
template<class T> struct Matrix {
    int R, C;
    vector<vector<T>> data;

    Matrix(int r, int c) {
        R = r; C = c;
        FOR(i, R) {
            vector<T> blank(c);
            data.pb(blank);
        }
    }

    Matrix(vector<vector<T>> dat) {
        data = dat; R = sz(data); C = sz(data[0]);
    }

    Matrix operator+(const Matrix& M) {
        Matrix res(R, C);
        FOR(i, R) {
            FOR(j, C) {
                res[i][j] = data[i][j] + M.data[i][j];
            }
        }
        return res;
    }

    Matrix operator-(const Matrix& M) {
        Matrix res(R, C);
        FOR(i, R) {
            FOR(j, C) {
                res[i][j] = data[i][j] - M.data[i][j];
            }
        }
        return res;
    }

    Matrix operator*(const Matrix& M) {
        Matrix res(R, M.c);
        FOR(i, R) FOR(j, M.c) res[i][j] = 0;
        FOR(i, R) {
            FOR(j, C) {
                FOR(k, M.c) {
                    res[i][k] += data[i][j] + M.data[j][k];
                }
            }
        }
        return res;
    }

    Matrix& operator+=(const Matrix& M) {
        return *this = (*this)+M;
    }

    Matrix& operator-=(const Matrix& M) {
        return *this = (*this)-M;
    }

    Matrix& operator*=(const Matrix& M) {
        return *this = (*this)*M;
    }

    friend Matrix exp(Matrix M, ll P) {
        Matrix res(M.R, M.C);
        FOR(i, M.R) {
            R.data[i][i] = 1;
        }
        for (; P; p /= 2; M *= M) if (P & 1) R *= M;
        return res;
    }
};
```

43 MinCostFlow

```
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<
     int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v
                 ]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s,
     int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

44 MinimumEnclosingCircle

```
point a[maxn];

void getCenter2(point a,point b,point & c)
{
    c.x = (a.x+b.x)/2;
    c.y = (a.y+b.y)/2;
}

void getCenter3(point a,point b,point c,point &d)
{
    double a1 = b.x-a.x, b1 = b.y-a.y, c1 = (a1*a1+b1*b1)/2;
    double a2 = c.x-a.x, b2 = c.y-a.y, c2 = (a2*a2+b2*b2)/2;
    double de = (a1 * b2 - b1 * a2);
    d.x = a.x + (c1 * b2 - c2 * b1)/de;
    d.y = a.y + (a1 * c2 - a2 * c1)/de;
}


    //randomP_shuffle before using
    radius = 0;
    center = a[0];
    for (int i = 1;i < n;++i)
    {
        if (!isIn(a[i], center, radius))
        {
            radius = 0;
            center = a[i];
            for (int j = 0;j < i;++j)
                if (!isIn(a[j], center, radius))
                {
                    getCenter2(a[i], a[j], center);
                    radius = dis(a[i],center);
                    for (int k = 0;k < j;++k) if (!isIn(a[k],
                         center, radius))
                    {
                        getCenter3(a[i], a[j], a[k], center);
                        radius = dis(a[k],center);
                    }
                }
        }
    }
    //printf("%.2lf\n%.2lf %.2lf\n",radius,center.x,center.y)
        ;
    printf("%.3lf\n",radius);
    return 0;
}
```

45 ModOps

```
ll modExp(ll base, ll power) {
    if (power == 0) {
        return 1;
    } else {
        ll cur = modExp(base, power / 2); cur = cur * cur;
             cur = cur % MOD;
        if (power % 2 == 1) cur = cur * base;
        cur = cur % MOD;
        return cur;
    }
}

ll inv(ll base) {
    int g = MOD, r = base, x = 0, y = 1;
    while (r != 0) {
        int q = g / r;
        g %= r; swap(g, r);
        x -= q * y; swap(x, y);
    }
    return x < 0 ? x+MOD : x;
}

ll mul(ll A, ll B) {
 return (A*B)%MOD;
}

ll add(ll A, ll B) {
 return (A+B)%MOD;
}

ll dvd(ll A, ll B) {
    return mul(A, inv(B));
}

ll sub(ll A, ll B) {
    return (A-B+MOD)%MOD;
}
```

46 NTT

```
namespace NTT { //Source: Codeforces user neal
    vector<ll> roots = {0, 1};
    vector<int> bit_reverse;
    int max_size = -1;
    ll root;

    bool is_power_of_two(int n) {
        return (n & (n - 1)) == 0;
    }

    int round_up_power_two(int n) {
        while (n & (n - 1))
            n = (n | (n - 1)) + 1;

        return max(n, 1);
    }

    // Given n (a power of two), finds k such that n == 1 <<
         k.
    int get_length(int n) {
        assert(is_power_of_two(n));
        return __builtin_ctz(n);
    }

    // Rearranges the indices to be sorted by lowest bit
         first, then second lowest, etc., rather than highest
         bit first.
    // This makes even-odd div-conquer much easier.
    void bit_reorder(int n, vector<ll> &values) {
        if ((int) bit_reverse.size() != n) {
            bit_reverse.assign(n, 0);
            int length = get_length(n);

            for (int i = 0; i < n; i++)
                bit_reverse[i] = (bit_reverse[i >> 1] >> 1) +
                     ((i & 1) << (length - 1));
        }

        for (int i = 0; i < n; i++)
            if (i < bit_reverse[i])
                swap(values[i], values[bit_reverse[i]]);
    }

    void find_root() {
        max_size = 1 << __builtin_ctz(MOD - 1);
        root = 2;

        // Find a max_size-th primitive root of MOD.
        while (!(modExp(root, max_size) == 1 && modExp(root,
             max_size/2) != 1))
            root++;
    }

    void prepare_roots(int n) {
        if (max_size < 0)
            find_root();

        assert(n <= max_size);

        if ((int) roots.size() >= n)
            return;

        int length = get_length(roots.size());
        roots.resize(n);

        // The roots array is set up such that for a given
             power of two n >= 2, roots[n / 2] through roots[
             n - 1] are
        // the first half of the n-th primitive roots of MOD.
        while (1 << length < n) {
            // z is a 2^(length + 1)-th primitive root of MOD
                 .
            ll z = modExp(root, max_size >> (length+1));

            for (int i = 1 << (length - 1); i < 1 << length;
                 i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = (roots[i] * z)%MOD;
            }

            length++;
        }
    }

    void fft_iterative(int N, vector<ll> &values) {
        assert(is_power_of_two(N));
        prepare_roots(N);
        bit_reorder(N, values);

        for (int n = 1; n < N; n *= 2)
            for (int start = 0; start < N; start += 2 * n)
                for (int i = 0; i < n; i++) {
                    ll even = values[start + i];
                    ll odd = values[start + n + i] * roots[n +
                         i];
                    odd %= MOD;
                    values[start + n + i] = even - odd + MOD;
                    values[start + i] = even + odd;
                    values[start + n + i] %= MOD;
                    values[start + i] %= MOD;
                }
    }

    const int FFT_CUTOFF = 150;

    vector<ll> mod_multiply(vector<ll> left, vector<ll> right
         ) {
        int n = left.size();
        int m = right.size();

        // Brute force when either n or m is small enough.
        if (min(n, m) < FFT_CUTOFF) {
            const uint64_t ULL_BOUND = numeric_limits<
                 uint64_t>::max() - (uint64_t) MOD * MOD;
            vector<uint64_t> result(n + m - 1, 0);

            for (int i = 0; i < n; i++)
                for (int j = 0; j < m; j++) {
                    result[i + j] += (uint64_t) ((int) left[i
                         ]) * ((int) right[j]);

                    if (result[i + j] > ULL_BOUND)
                        result[i + j] %= MOD;
                }

            for (uint64_t &x : result)
                if (x >= MOD)
                    x %= MOD;

            return vector<ll>(result.begin(), result.end());
        }

        int N = round_up_power_two(n + m - 1);
        left.resize(N);
        right.resize(N);

        bool equal = left == right;
        fft_iterative(N, left);
    if (equal)
        right = left;
    else
        fft_iterative(N, right);

    ll inv_N = inv(N);

    for (int i = 0; i < N; i++) {
        left[i] *= (right[i] * inv_N)%MOD;
        left[i] %= MOD;
    }

    reverse(left.begin() + 1, left.end());
    fft_iterative(N, left);
    left.resize(n + m - 1);
    return left;
}

vector<ll> mod_power(const vector<ll> &v, int exponent) {
    assert(exponent >= 0);
    vector<ll> result = {1};

    if (exponent == 0)
        return result;

    for (int k = 31 - __builtin_clz(exponent); k >= 0; k
         --) {
        result = mod_multiply(result, result);

        if (exponent >> k & 1)
            result = mod_multiply(result, v);
    }

    return result;
}

vector<ll> mod_multiply_all(const vector<vector<ll>> &
     polynomials) {
    if (polynomials.empty())
        return {1};

    struct compare_size {
        bool operator()(const vector<ll> &x, const vector
             <ll> &y) {
            return x.size() > y.size();
        }
    };

    priority_queue<vector<ll>, vector<vector<ll>>,
         compare_size> pq(polynomials.begin(),
             polynomials.end());

    while (pq.size() > 1) {
        vector<ll> a = pq.top(); pq.pop();
        vector<ll> b = pq.top(); pq.pop();
        pq.push(mod_multiply(a, b));
    }

    return pq.top();
  }
}
```

47 PersistentSegtree

```
//Define the node of a persistent segment tree
struct node{
    int l,r,sum;
};
//the persistent segment tree. Warning: Check memory limit
     before using persistent segment tree!
node tree[maxn*32];
//Storing the root of versions of segment tree
int head[maxn];
//allocate next position. You can implement in a way that
     support garbage collection.
int nextPos(){
    static int ct;return ++ct;
}

//Building the first version of our segmetn tree
void build(int cur,int l,int r){
  tree[cur].sum = 0;
  tree[cur].l = nextPos();
  tree[cur].r = nextPos();
  if (l == r){
    tree[cur].l = tree[cur].r = 0;
  }
  else{
    int mid = (l+r)>>1;
    build(tree[cur].l,l,mid);
    build(tree[cur].r,mid+1,r);
  }
}

//This function is: currently we are at node cur, which is a
      node in the latest version of segment tree
//we want to make modifications based on some past segment
     tree, and the corresponding node in the last version is
      at last
//we want to add 1 at position key
void modify(int cur,int last,int key,int l,int r){
  //this is creating the node for our latest version
  tree[cur].sum = tree[last].sum;
  tree[cur].l = nextPos();
  tree[cur].r = nextPos();
  if (l == r){
    //base case:add on current version of our segment tree
    tree[cur].sum++;
    tree[cur].l = tree[cur].r = 0;
  }
  else{
    int mid = (l+r)>>1;
    if (key <= mid){
      //we are going to modify in the left part, so we can
           reuse the right child
      tree[cur].r = tree[last].r;
      modify(tree[cur].l, tree[last].l, key,l, mid);
      //update information for the current version of segment
            tree
      tree[cur].sum++;
    }
    else
    {
      tree[cur].l = tree[last].l;
      modify(tree[cur].r, tree[last].r, key,mid+1, r);
      tree[cur].sum++;
    }
  }
}

int query(int cur,int last,int l,int r,int k){
  if (l == r) return l;
  int mid = (l+r) >> 1;
  //notice the subtraction here - we want tgo see the
       dfifference between today's version and old versions.
  int ct = tree[tree[cur].l].sum - tree[tree[last].l].sum ;
  //if there are to many larger than mid, the k-th element
       should be in the left
  if (ct >= k){
    return query(tree[cur].l,tree[last].l,l,mid,k);
  }
  //otherwise,the k-th element should be in the right
  else{
    return query(tree[cur].r, tree[last].r, mid+1, r, k-ct);
  }
}
```

```
//Build segment tree to support queres k-th element in a
     subinterval
void build(int n){
  for (int i = 0;i <= n;++i){
    head[i] = nextPos();
  }
  build(head[0], 1, n);
  for (int i = 1;i <= n;++i){
    modify(head[i], head[i-1],c[i], 1, n);
  }
}
/*Query the k-th element in [l,r]:
printf("%d\n",a[query(head[r], head[l-1], 1, n, k)].key);*/
```

48 Point

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0);
     }
template<class T>
struct Point {
 typedef Point P;
 T x, y;
 explicit Point(T x=0, T y=0) : x(x), y(y) {}
 bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y);
      }
 bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y);
      }
 P operator+(P p) const { return P(x+p.x, y+p.y); }
 P operator-(P p) const { return P(x-p.x, y-p.y); }
 P operator*(T d) const { return P(x*d, y*d); }
 P operator/(T d) const { return P(x/d, y/d); }
 T dot(P p) const { return x*p.x + y*p.y; }
 T cross(P p) const { return x*p.y - y*p.x; }
 T cross(P a, P b) const { return (a-*this).cross(b-*this);
     }
 T dist2() const { return x*x + y*y; }
 double dist() const { return sqrt((double)dist2()); }
 // angle to x-axis in interval [-pi, pi]
 double angle() const { return atan2(y, x); }
 P unit() const { return *this/dist(); } // makes dist()=1
 P perp() const { return P(-y, x); } // rotates +90 degrees
 P normal() const { return perp().unit(); }
 // returns point rotated 'a' radians ccw around the origin
 P rotate(double a) const {
  return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
 friend ostream& operator<<(ostream& os, P p) {
  return os << "(" << p.x << "," << p.y << ")"; }
};
```

```
//Line distance
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
 return (double)(b-a).cross(p-a)/(b-a).dist();
}

//LineProjectionReflection
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
 P v = b - a;
 return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}

//Line Segment Distance
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
 if (s==e) return (p-s).dist();
 auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
 return ((p-s)*d-(e-s)*t).dist()/d;
}

//On Segment
template<class P> bool onSegment(P s, P e, P p) {
 return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}

//Segment Intersection
/*If a unique intersection point between the line segments
     going from s1 to e1 and from s2 to e2 exists then it is
      returned.
If no intersection point exists an empty vector is returned.
      If infinitely many exist a vector with 2 elements is
     returned, containing the endpoints of the common line
     segment.
The wrong position will be returned if P is Point<ll> and
     the intersection point does not have integer
     coordinates.
Products of three coordinates are used in intermediate steps
      so watch out for overflow if using int or long long.*/
template<class P> vector<P> segInter(P a, P b, P c, P d) {
 auto oa = c.cross(d, a), ob = c.cross(d, b),
      oc = a.cross(b, c), od = a.cross(b, d);
 // Checks if intersection is single non-endpoint point.
 if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
  return {(a * ob - b * oa) / (ob - oa)};
 set<P> s;
 if (onSegment(c, d, a)) s.insert(a);
 if (onSegment(c, d, b)) s.insert(b);
 if (onSegment(a, b, c)) s.insert(c);
 if (onSegment(a, b, d)) s.insert(d);
 return {all(s)};
}
```

49 Point3d

```
template<class T> struct Point3D {
 typedef Point3D P;
 typedef const P& R;
 T x, y, z;
 explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
 bool operator<(R p) const {
  return tie(x, y, z) < tie(p.x, p.y, p.z); }
 bool operator==(R p) const {
  return tie(x, y, z) == tie(p.x, p.y, p.z); }
 P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
 P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
 P operator*(T d) const { return P(x*d, y*d, z*d); }
 P operator/(T d) const { return P(x/d, y/d, z/d); }
 T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
 P cross(R p) const {
  return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
 }
 T dist2() const { return x*x + y*y + z*z; }
 double dist() const { return sqrt((double)dist2()); }
 //Azimuthal angle (longitude) to x-axis in interval [-pi,
      pi]
 double phi() const { return atan2(y, x); }
 //Zenith angle (latitude) to the z-axis in interval [0, pi]
 double theta() const { return atan2(sqrt(x*x+y*y),z); }
 P unit() const { return *this/(T)dist(); } //makes dist()=1
 //returns unit vector normal to *this and p
 P normal(P p) const { return cross(p).unit(); }
 //returns point rotated 'angle' radians ccw around axis
 P rotate(double angle, P axis) const {
  double s = sin(angle), c = cos(angle); P u = axis.unit();
  return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
 }
};
```

50 PollardRho

```
#include <map>
#include <ctime>
#include <cstdlib>
```

```cpp
#include <iostream>

using namespace std;

typedef long long ll;

//Quick Multiplication - Calculate x * y mod modi
     efficiently
//where x and y is in long long range
ll quickmult(ll x, ll y, ll p){
    ll temp = x * y - ((ll)((long double)x / p * y + 0.5)) *
         p;
    return (temp < 0) ? temp + p : temp;
}

//Prime Test via Miller-Rabin

bool prime_test(ll p){
    static int tests[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
          31, 37};
    int r = 0;
    ll b = p - 1;
    if (p == 2) return true;
    if (p == 1 || (p & 1) == 0) return false;

    while ((b & 1) == 0){
        r++;
        b >>= 1;
    }
    ll d = (p - 1) / (1ll << r);
    for (int i = 0;i < 12;i++){
        if (p == tests[i]){
            return true;
        }
        ll x = quickpow2(tests[i], d, p);
        for (int j = 1;j <= r;j++){
            ll y = quickmult(x, x, p);
            if (y == 1 && x != 1 && x != p - 1) return false;
            x = y;
        }
        if (x != 1) return false;
    }
    return true;
}

//We will store factors in a global variable to save time
map<ll, int> factors;

ll abs(ll x){
    return (x < 0)? (-x) : x;
}

ll gcd(ll x, ll y){
    if (y == 0) return x;
    return gcd(y, x % y);
}

ll get_next(ll x, ll addi,ll modi){
    return (quickmult(x, x, modi) + addi);
}

//find a prime factor of n based on the seed, if we cannot
     find it return -1
ll rho_find(ll n, ll seed, ll addi){
    ll a = seed;
    ll b = get_next(seed, addi, n);
    while (a != b){
        ll p = gcd(abs(a - b), n);
        if (p > 1){
            if (p == n) return -1;
            return p;
        }
        a = get_next(a, addi, n);
        b = get_next(get_next(b, addi, n), addi, n);
    }
    return -1;
}

//factorizing n via pollard rho
void pollard_rho(ll n){
    if (n == 1){
        return;
    }
    if (prime_test(n)){
        factors[n]++;
        return;
    }
    ll p = -1;
    while (p == -1){
        ll addi = rand() % (n - 1) + 1;
        p = rho_find(n, rand() % (n - 1) + 1, addi);
        if (p != -1){
            pollard_rho(p);
            pollard_rho(n / p);
            return;
        }
    }
}

int main(int argc, const char * argv[]) {
    srand(0);
    int t;
    cin >> t;
    while (t--){
        ll n;
        cin >> n;
        if (prime_test(n)){
            cout << "Prime\n";
        }
        else{
            factors.clear();
            pollard_rho(n);
            cout << factors.begin()->first << endl;
        }
    }
    return 0;
}
```

51 QuasiExgcdSum

```cpp
/*
Using Quasi_Exgcd to sum
f(a,b,c,n) = sum_{i=0}^n \floor{(ai+b)/c}
g(a,b,c,n) = sum_{i=0}^n \floor i{(ai+b)/c}
h(a,b,c,n) = sum_{i=0}^n (\floor{(ai+b)/c})^2
all are done under mod p
*/

struct rec{
  ll f, g, h;
};

//add, sub, quickpow omitted
ll inv2 = quickpow(2, modi-2);
ll inv6 = quickpow(6, modi - 2);

rec solve(ll a, ll b, ll c, ll n){
  rec ans;
 if (a == 0){
   ans.f = (b / c) * (n + 1) % modi;
  ans.g = (b / c) * (n + 1) % modi * n % modi * inv2 % modi;
  ans.h = (b / c) * (b / c) % modi * (n+1) % modi;
  return ans;
 }
 ans.f = ans.g = ans.h = 0;
 if (a >= c || b >= c){
   rec temp = solve(a % c, b % c, c, n);
  add(ans.f, (a/c)*n%modi*(n+1)%modi*inv2%modi);
  add(ans.f, (b/c)*(n+1)%modi);
  add(ans.f, temp.f);
  add(ans.g, (a/c)*n%modi*(n+1)%modi*
  ((2*n+1)%modi)%modi*inv6%modi);
    add(ans.g, (b/c)*n%modi*(n+1)%modi*inv2 % modi);
    add(ans.g, temp.g);
    add(ans.h, (a/c)*(a/c)%modi*n%modi*
  (n+1)%modi*((2*n+1)%modi)%modi*inv6% modi);
    add(ans.h, (b/c)*(b/c)%modi*(n+1)%modi);
    add(ans.h, (a/c)*(b/c)%modi*n%modi*(n+1)%modi);
    add(ans.h, temp.h);
    add(ans.h, 2LL * (a/c)%modi*temp.g%modi);
    add(ans.h, 2LL * (b/c)%modi*temp.f%modi);
  return ans;
 }
 if (a < c && b < c){
  ll m = (a * n + b) / c;
    rec temp = solve(c, c - b - 1, a, m - 1);
    ans.f = n * m % modi;
    sub(ans.f, temp.f);
    ans.g = n * (n + 1) % modi * m % modi;
    sub(ans.g, temp.f);
    sub(ans.g, temp.h);
    ans.g = ans.g * inv2 % modi;
    ans.h = n * m % modi * (m + 1) % modi;
    sub(ans.h, 2LL * temp.g % modi);
    sub(ans.h, 2LL * temp.f % modi);
    sub(ans.h, ans.f);
  return ans;
 }
 return ans;
}
```

52 QuickPhiSum

```
/*
This algorithm concerns efficient evaluation of sum of
     number theoric functions like phi or mu.
We know that using Eulerian sieve, we can only archieve O(n)
      time complexity.
What we are doing is to archieve O(n^{2/3}) time complexity.
The example program shows how to evaluate sum of phi and sum
      of mu efficientyl.
For smaller n (n less than (N^{2/3}), we use calculate them
      as usual.
For larger n, see getphi and getmu
*/
//See Sieve for more technical details.
//When i is prime, phi[i] = i-1;mu[i] = -1; Otherwise,
     inside the inner loop, let p = primes[j].
//Then it follows phi[p*i] = phi[i] * (p-1); mu[p*i] = -mu[i
     ];
//finally, when i % p == 0, phi[p*i] = phi[i]*p and mu[p*i]
     = 0;

ll getphi(ll n)
{
  if (n <= m) return phi[n];
  if (phi_cheat.find(n) != phi_cheat.end()) return phi_cheat
      [n];
  ll ans = (ll)n*(n + 1) / 2; //this is \sum_{i = 1}^n \sum_
      {d | n} \phi(d)
  //when getting mu, ans = 1
  ll last;
  for (ll i = 2;i <= n;i = last + 1)
  {
    last = n / (n / i);
    ans -= (last-i+1)*getphi(n / i);
  }
  phi_cheat[n] = ans;
  return ans;
}
```

53 RootNonTree

```
void dfs(int u){
  static int top;
  stk[++top] = u;
  for (int i = 0;i < g[u].size();i++){
    int v = g[u][i];
    if (v != p[u]){
        p[v] = u;
        dfs(v);
        if (siz[u] + siz[v] >= magic){
          siz[u] = 0;
          tot_cols++;
          cap[tot_cols] = u;
          while (stk[top] != u){
            col[stk[top--]] = tot_cols;
          }
        }
        else siz[u] += siz[v];
    }
  }
  siz[u]++;
}

void paint(int u,int c){
    if (col[u]) c = col[u];
    else col[u] = c;
    for (int i = 0;i < g[u].size();i++){
        int v = g[u][i];
        if (v != p[u]){
            paint(v,c);
        }
    }
}
//actual blokcing; magic = block size
dfs(1);
if (!tot_cols){
    cap[++tot_cols] = 1;
}
paint(1,tot_cols);
```

54 SCC

```
vector < vector<int> > g, gr;
vector<bool> used;
vector<int> order, component;

void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i)
        if (!used[ g[v][i] ])
            dfs1 (g[v][i]);
    order.push_back (v);
}

void dfs2 (int v) {
    used[v] = true;
    component.push_back (v);
    for (size_t i=0; i<gr[v].size(); ++i)
        if (!used[ gr[v][i] ])
            dfs2 (gr[v][i]);
}

int main() {
    int n;
    ... reading n ...
    for (;;) {
        int a, b;
        ... reading next edge (a,b) ...
        g[a].push_back (b);
        gr[b].push_back (a);
    }

    used.assign (n, false);
    for (int i=0; i<n; ++i)
        if (!used[i])
            dfs1 (i);
    used.assign (n, false);
    for (int i=0; i<n; ++i) {
        int v = order[n-1-i];
        if (!used[v]) {
            dfs2 (v);
            ... printing next component ...
            component.clear();
        }
    }
}
```

55 Sam

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 510000;
const int sigma = 26;

struct edge
{
    int v, next;
};

edge g[maxn << 1];
int trie[maxn << 1][sigma], fa[maxn << 1], maxi[maxn << 1],
     sizia[maxn << 1];
char str[maxn];
int head[maxn << 1];
int siz,last;

void insert(int u, int v)
{
    static int id;
    g[++id].v = v;
    g[id].next = head[u];
    head[u] = id;
}

//This is the core of SAM
void add(int id)
{
    int p = last;
    int np = last = ++siz;
    sizia[np] = 1;
    maxi[np] = maxi[p] + 1;
    while (p && !trie[p][id]){
        trie[p][id] = np;
        p = fa[p];
    }
    if (!p){
        fa[np] = 1;
    }
    else{
        int q = trie[p][id];
        if (maxi[p] + 1 == maxi[q]){
            fa[np] = q;
        }
        else{
            int nq = ++siz;
            maxi[nq] = maxi[p] + 1;
            memcpy(trie[nq], trie[q], sizeof trie[q]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            while (trie[p][id] == q){
                trie[p][id] = nq;
                p = fa[p];
            }
        }
    }
}
```

56 Segtree

```
ll SZ = 262144;
ll* seg = new ll[2*SZ]; //segtree implementation by bqi343's
      Github


ll combine(ll a, ll b) { return max(a, b); }

void build() { FORd(i,SZ) seg[i] = combine(seg[2*i],seg[2*i
     +1]); }

void update(int p, ll value) {
    for (seg[p += SZ] = value; p > 1; p >>= 1)
        seg[p>>1] = combine(seg[(p|1)^1], seg[p|1]);
}

ll query(int l, int r) { // sum on interval [l, r]
    ll resL = 0, resR = 0; r++;
    for (l += SZ, r += SZ; l < r; l >>= 1, r >>= 1) {
        if (l&1) resL = combine(resL,seg[l++]);
        if (r&1) resR = combine(seg[--r],resR);
    }
    return combine(resL,resR);
}
```

57 Sieve

```
vi primes, leastFac;
void compPrimes(int N) {
 FOR(i, N) {
  leastFac.pb(0);
 }
 leastFac[0] = 1; leastFac[1] = 1;
 FOR(i, 2, N) {
  if (leastFac[i] == 0) {
   primes.pb(i);
   leastFac[i] = i;
  }
  for (int j = 0; j < sz(primes) && i*primes[j] < N &&
       primes[j] <= leastFac[i]; j++) {
   leastFac[i*primes[j]] = primes[j];
  }
 }
}
```

58 Simpson

```
/*
This is a template for solving simpson integration
We are going to integrate \frac{cx+d}{ax+b} over L and R
*/
ld simpson(ld lower, ld upper){
    ld mid = (lower + upper) / 2;
    return (f(lower) + 4 * f(mid) + f(upper)) * (upper-lower)
          / 6;
}

ld simpson_integration(ld lower, ld upper, ld target){
    ld mid = (lower + upper) / 2;
    ld left_sum = simpson(lower, mid);
    ld right_sum = simpson(mid, upper);
    if (fabs(left_sum + right_sum - target) < eps){
        return left_sum + right_sum;
    }
    return simpson_integration(lower, mid, left_sum) +
         simpson_integration(mid, upper, right_sum);
}

//Call: simpson_integration(lower, upper, simpson(lower,
     upper)) << endl;
```

59 SuffixArray

```
vector<vi> C[20][MX]; //log N

vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;

    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);

    FOR(i, N) C[0][i] = c[i];

    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)
                 ) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1
                 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        FOR(i, N) C[h+1][i] = cn[i];
        c.swap(cn);
    }
    return p;
}


vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

int lcp(int i, int j) {
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (C[k][i] == C[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}
```

60 Template

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef long long ll;
typedef long double ld;
typedef complex<ld> cd;

typedef pair<int, int> pi;
typedef pair<ll,ll> pl;
typedef pair<ld,ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
typedef vector<cd> vcd;

template <class T> using Tree = tree<T, null_type, less<T>,
     rb_tree_tag,tree_order_statistics_node_update>;

#define FOR(i, a, b) for (int i=a; i<(b); i++)
#define FOR(i, a) for (int i=0; i<(a); i++)
#define FORd(i,a,b) for (int i = (b)-1; i >= a; i--)
#define FORd(i,a) for (int i = (a)-1; i >= 0; i--)

#define sz(x) (int)(x).size()
#define mp make_pair
#define pb push_back
#define f first
#define s second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());

const int MOD = 1000000007;
const ll INF = 1e18;
const int MX = 100001; //check the limits, dummy

int main() {
 ios_base::sync_with_stdio(0); cin.tie(0);
 return 0;
}

// read the question correctly (ll vs int)
// template by bqi343
```

61 Treap

```
struct node
{
    node *ch[2]; //ch[0] = left child; ch[1] = right child;
    int ct,priority,size,key;
    int lsize(){return(ch[0] == NULL)?0:ch[0]->size;}
    int rsize(){return(ch[1] == NULL)?0:ch[1]->size;}
};
typedef node* tree;
void update(tree & o){//this part depends on the actual info
     to maintain
    o->size = o->ct; o->size += o->lsize(); o->size += o->
         rsize();
}
void rotate(tree & o,int dir){ //dir = 0: left rotate
    tree temp = o->ch[dir^1]; o->ch[dir^1] = temp->ch[dir];
         temp->ch[dir] = o;
    update(o); update(temp); o = temp;
}
void insert(tree & o,int key){
    if (o == NULL){
        o = new node;
        o->size = o->ct = 1;o->priority = rand();o->ch[0]=o->
             ch[1]=NULL;o->key=key;
    }
    else if (key == o->key){
        o->ct++;o->size++;
    }
    else{
        int dir = (key<o->key)?0:1;
        insert(o->ch[dir],key);
        if (o->ch[dir]->priority>o->priority) rotate(o,dir^1)
            ;
        update(o);
    }
}
void remove(tree & o,int key){
    if (key == o->key){
        if (o->ct > 1){
            o->ct--;o->size--;return;
        }
        else if (o->ch[0]==NULL||o->ch[1]==NULL){
            int d = (o->ch[0]==NULL)?0:1;
            tree temp = o; o = o->ch[d^1]; delete temp;
        }
        else{
            int d =(o->ch[0]->priority > o->ch[1]->priority)
                 ?1:0;
            rotate(o,d);remove(o,key);
        }
    }
    else{
        int d = (key<o->key)?0:1;
        remove(o->ch[d],key);
    }
    if (o) update(o);
}
```

62 Vimrc

```
source $VIMRUNTIME/defaults.vim
set tabstop=4
set nocompatible
set shiftwidth=4
set autoindent
"set smartindent
set cindent
set ruler
set showcmd
set incsearch
set number
set relativenumber
set cino+=L0 "Doesn't untab when typing colons
syntax on
filetype indent on

inoremap {<CR> {<CR>}<Esc>O
inoremap {}    {}
imap jk        <Esc>
set belloff=all
```

63 Z-algorithm

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

64 ordered$_set$

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
```