Team Notebook

Kotlin Enjoyers - Universitas Indonesia

November 25, 2022

Contents

1 CRT

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 20;
long long GCD(long long a, long long b) { return (b == 0) ? a : GCD(b, a % b); }
inline long long LCM(long long a, long long b) { return a / GCD(a, b) * b; }
inline long long normalize(long long x, long long mod) { x %= mod; if (x < 0) x += mod; return x; }
struct GCD_type { long long x, y, d; };
GCD_type ex_GCD(long long a, long long 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};
}
int testCases;
int t;
long long a[N], n[N], ans, lcm;

// format input :
//  x dan MOD

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> t;
    for(int i = 1; i <= t; i++) cin >> a[i] >> n[i], normalize(a[i], n[i]);
    ans = a[1];
    lcm = n[1];
    for(int i = 2; i <= t; i++)
    {
        auto pom = ex_GCD(lcm, n[i]);
        int x1 = pom.x;
        int d = pom.d;
        if((a[i] - ans) % d != 0) return cerr << "No solutions" << endl, 0;
        ans = normalize(ans + x1 * (a[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
    }
    cout << ans << " " << lcm << endl;

    return 0;
}
```

2 Dinic

```cpp
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;
    }
};
```

3 Dynamic CHT

```cpp
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable 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;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will
     maintain upper hull for maximum
    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;

        // **** May need long double typecasting here
        return (long double)(x->b - y->b)*(z->m - y->m) >= (
             long double)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll 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));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

4 FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const
     & b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end())
         ;
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

5 Geometry

```
//Proyeksi segitiga: BC^2 = AC^2 + AB^2 - 2AD.AC
#define EPS 1E-9
#define PI acos(-1)

// >>>> Constructor of point
struct point {
  double x,y;
  point() { x = y = 0.0; }
  point(double _x, double _y) : x(_x), y(_y) {}
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) <
        EPS));
  }
};
// >>>> Constructor of vector
struct vec {
  double x, y;
  vec(double _x, double _y) : x(_x), y(_y) {}
};
// >>>> Constructor of line (ax + by = c)
struct line {
  double a,b,c;
};
// Distance of two points
double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y);
}
double DEG_to_RAD(double theta) {
  return theta * PI / 180.0;
}
// Rotate a point THETA degrees
point rotate(point p, double theta) {
  double rad = DEG_to_RAD(theta);
  return point(p.x * cos(rad) - p.y * sin(rad),
      p.x * sin(rad) + p.y * cos(rad));
}
// Make a line l from 2 given points
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {
   l.a = 1.0 ; l.b = 0.0 ; l.c = -p1.x;
  } else {
   l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
   l.b = 1.0;
   l.c = -(double)(l.a * p1.x) - p1.y;
  }
}
// Check if two lines are parallel
bool areParallel(line l1, line l2) {
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);
}
// Check if two lines are same
bool areSame(line l1, line l2) {
  return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);
}
```

```
// Check if two lines are intersect (at point P)
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a *
       l2.b);
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c); return true;
}
// Convert 2 points to vector A -> B
vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y);
}
// Scale a vector
vec scale(vec v, double s) {
  return vec(v.x * s, v.y * s);
}
// Translate P according to v
point translate(point p, vec v) {
  return point(p.x + v.x, p.y + v.y);
}
// Dot product of two vectors
double dot(vec a, vec b) {
  return a.x * b.x + a.y * b.y;
}
// Cross product of two vectors
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
double norm_sq(vec v) {
  return v.x * v.x + v.y * v.y;
}
// Get the minimum distance of point P and line AB
// Line PC is the minimum distance
double distToLine(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));
  return dist(p,c);
}
// Get the minimum distance of point P and line segment AB
// Line PC is the minimum distance
double distToLineSegment(point p, point a, point b, point &c
    ) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) {
   c = point(a.x, a.y);
   return dist(p,a);
  }
  if (u > 1.0) {
    c = point(b.x, b.y);
    return dist(p, b);
  }
  return distToLine(p, a, b, c);
}
// Returns angle AOB in RADIANS
double angle(point a, point o, point b) {
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa,ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}
// Heron's Formula : Find the area of triangle double
heronsFormula(double a, double b, double c) {
  double s = perimeter(a, b, c) * 0.5;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}
// Find the radius incircle of triangle ABC (lengths)
double rInCircle(double ab, double bc, double ca) {
  return heronsFormula(ab,bc,ca) / (0.5 * perimeter(ab, bc,
       ca));
}
// Find the radius incircle of triangle ABC (points)
double rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c), dist (c, a));
}
// Returns 1 if there is an incircle center, return 0
     otherwise
// ctr will be the incircle center
// r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr,
     double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0;

  line l1, l2;
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3), ratio / (1 +
       ratio)));
  pointsToLine(p1, p, l1);
  ratio = dist(p2, p1) / dist(p2 , p3);
  p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)
       ));
  pointsToLine(p2, p, l2);

  areIntersect(l1, l2, ctr);
  return 1;
}
// Find the radius circumcircle of triangle ABC (lengths)
double rCircumCircle(double ab, double bc, double ca) {
  return ab * bc * ca / (4.0 * heronsFormula(ab, bc, ca));
}
// Find the radius circumcircle of triangle ABC (points)
double rCircumCircle(point a, point b, point c) {
  return rCircumCircle(dist(a, b), dist(b, c), dist(c , a));
}
// Polygon Representation :
// 4 points, entered in counter clockwise order, 0-based
     indexing
// vector<point> P;
// P.push_back(point(1,1)); // P[0]
// P.push_back(point(3,3)); // P[1]
// P.push_back(point(9,7)); // P[2]
// P.push_back(point(1,7)); // P[3]
// P.push_back(P[0]); // P[n-1] = P[0]
// Checks if a polygon is convex or not
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
    bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i > sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft
        )
     return false;
  return true;
}
// Line segment PQ intersect with line AB at this point
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u + v),
      (p.y * v + q.y * u) / (u + v));
}
// Cuts polygon Q along the line AB
vector<point> cutPolygon(point a, point b, const vector<
     point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
   double left1 = cross(toVec(a,b), toVec(a, Q[i])), left2 =
        0;
   if (i != (int)Q.size()-1) left2 = cross(toVec(a, b),
       toVec(a, Q[i+1]));
   // Q[i] is on the left of AB
   // edge(Q[i], Q[i+1]) crosses line AB
   if (left1 > -EPS) P.push_back(Q[i]);
   if (left1 * left2 < -EPS)
     P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
```

```
    P.push_back(P.front());
  return P;
}
//-- Line Segment Intersection
int pyt(PII a, PII b){
    int dx=a.x-b.x;
    int dy=a.y-b.y;
    return (dx*dx + dy*dy);
}
int det(PII a, PII b, PII c){
    return ((a.x*b.y)+(b.x*c.y)+(c.x*a.y)
          -(a.x*c.y)-(b.x*a.y)-(c.x*b.y));
}
bool insec(pair<PII,PII> t1, pair<PII,PII> t2){
    bool hsl;
    h1=det(t1.F,t1.S, t2.F);
    h2=det(t1.F,t1.S, t2.S);
    h3=det(t2.F,t2.S, t1.F);
    h4=det(t2.F,t2.S, t1.S);
    hsl=false;
    if ((h1*h2<=0) && (h3*h4<=0) && !((h1==0) && (h2==0) && (
        h3==0) && (h4==0))){
       hsl=true;
    }
    return hasil;
}
...
//sg1 dan sg2 adalah pair<PII,PII>
if (insec(sg1,sg2)){
   le=sqrt((double)pyt(sg2.x, sg2.y));
   r1=fabs(crosp(MP(sg2.x, sg1.x),sg2)/le);
   r2=fabs(crosp(MP(sg2.x, sg1.y),sg2)/le);
   r2=r1+r2;
   dix=sg1.x.x + (r1/r2)*(sg1.y.x - sg1.x.x);
   diy=sg1.x.y + (r1/r2)*(sg1.y.y - sg1.x.y);
   //intersect here
   return MP(dix,diy);
}
// returns the area, which is half the determinant
// works for both convex and concave polygons
double area(vector<point> P) {
 double result = 0.0, x1, y1, x2, y2;
 for (int i = 0; i < P.size() - 1; i++) {
  x1 = P[i].x;
  x2 = P[i + 1].x;
  y1 = P[i].y;
  y2 = P[i + 1].y;
  result += (x1 * y2 - x2 * y1);
 }
 return fabs(result) / 2.0;
}
// returns true if point p is in either convex/concave
    polygon P
bool inPolygon(point p, const vector<point> &P) {
 if ((int) P.size() == 0) return false;
 double sum = 0; // assume first vertex = last vertex
 for (int i = 0; i < (int) P.size() - 1; i++) {
  if (ccw(p, P[i], P[i + 1]))
   sum += angle(P[i], p, P[i + 1]); // left turn/ccw
  else
   sum -= angle(P[i], p, P[i + 1]);
 } // right turn/cw
 return fabs(fabs(sum) - 2 * PI) < EPS;
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
} // compute distance between point (x,y,z) and plane ax+by+
    cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double
                              d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
//circle-circle intersect
for(int i = 1; i < n; i++) {
 for(int j = i + 1; j <= n; j++) {
  double d = dist(P[i], P[j]);
  double r0 = P[i].r, x0 = P[i].x, y0 = P[i].y
  double r1 = P[j].r, x1 = P[j].x, y1 = P[j].y;
  point center;
  if (d > r0 + r1) continue;
  if (d < fabs(r0 - r1) || fabs(d) < EPS) {
   if (r0 < r1) center = P[i];
   else center = P[j];
  } else {
   double a = (r0*r0 - r1*r1 + d*d)/(2*d);
   double h = sqrt(r0*r0 - a*a);
   double x2 = x0 + a*(x1 - x0)/d;
   double y2 = y0 + a*(y1 - y0)/d;
   double translationY = h*(y1 - y0)/d;
   double translationX = h*(x1 - x0)/d;
   center.x = x2 + translationY;
   center.y = y2 - translationX;
   ans = max(ans, go(center));
   center.x = x2 - translationY;
   center.y = y2 + translationX;
  }
  ans = max(ans, go(center));
 }
}
// line segment with circle intersect
private int FindLineCircleIntersections(
    float cx, float cy, float radius,
    PointF point1, PointF point2,
    out PointF intersection1, out PointF intersection2)
{
    float dx, dy, A, B, C, det, t;
    dx = point2.X - point1.X;
    dy = point2.Y - point1.Y;
    A = dx * dx + dy * dy;
    B = 2 * (dx * (point1.X - cx) + dy * (point1.Y - cy));
    C = (point1.X - cx) * (point1.X - cx) +
        (point1.Y - cy) * (point1.Y - cy) -
        radius * radius;
    det = B * B - 4 * A * C;
    if ((A <= 0.0000001) || (det < 0)) {
        // No real solutions.
        intersection1 = new PointF(float.NaN, float.NaN);
        intersection2 = new PointF(float.NaN, float.NaN);
        return 0;
    }   else if (det == 0)  {
        // One solution.
        t = -B / (2 * A);
        intersection1 =
            new PointF(point1.X + t * dx, point1.Y + t * dy);
        intersection2 = new PointF(float.NaN, float.NaN);
        return 1;
    } else {
      // Two solutions.
      t = (float)((-B + Math.Sqrt(det)) / (2 * A));
      intersection1 = new PointF(point1.X + t * dx, point1.Y
          + t * dy);
      t = (float)((-B - Math.Sqrt(det)) / (2 * A));
      intersection2 = new PointF(point1.X + t * dx, point1.Y
          + t * dy);
      return 2;
    }
 }
```

6 Great Circle Distance

```
double dist3d(double lat1, double lon1, double lat2, double
    lon2){
  double dlat = lat2 - lat1;
  double dlon = lon2 - lon1;
  double a = pow(sin(dlat/2),2) + cos(lat1) * cos(lat2)* pow
      (sin(dlon/2),2);
  return (R*2*atan2(sqrt(a), sqrt(1-a)));
}
```

7 HLD

```
#include "bits/stdc++.h"
using namespace std;

const int N = 2e5+5;
const int D = 19;
const int S = (1<<D);

int n, q, v[N];
vector<int> adj[N];

int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];

void update(int idx, int val) {
 st[idx += n] = val;
 for (idx /= 2; idx; idx /= 2)
  st[idx] = max(st[2 * idx], st[2 * idx + 1]);
}

int query(int lo, int hi) {
 int ra = 0, rb = 0;
 for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
  if (lo & 1)
   ra = max(ra, st[lo++]);
  if (hi & 1)
   rb = max(rb, st[--hi]);
 }
 return max(ra, rb);
}

int dfs_sz(int cur, int par) {
 sz[cur] = 1;
 p[cur] = par;
 for(int chi : adj[cur]) {
  if(chi == par) continue;
  dep[chi] = dep[cur] + 1;
  p[chi] = cur;
  sz[cur] += dfs_sz(chi, cur);
 }
 return sz[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
 id[cur] = ct++;
 tp[cur] = top;
 update(id[cur], v[cur]);
 int h_chi = -1, h_sz = -1;
 for(int chi : adj[cur]) {
  if(chi == par) continue;
  if(sz[chi] > h_sz) {
   h_sz = sz[chi];
   h_chi = chi;
  }
 }
 if(h_chi == -1) return;
 dfs_hld(h_chi, cur, top);
 for(int chi : adj[cur]) {
  if(chi == par || chi == h_chi) continue;
  dfs_hld(chi, cur, chi);
 }
}

int path(int x, int y){
 int ret = 0;
 while(tp[x] != tp[y]){
  if(dep[tp[x]] < dep[tp[y]])swap(x,y);
  ret = max(ret, query(id[tp[x]],id[x]));
  x = p[tp[x]];
 }
 if(dep[x] > dep[y])swap(x,y);
 ret = max(ret, query(id[x],id[y]));
 return ret;
}

// Tiap edge punya value.
// Query 1: ubah value suatu node
// Query 2: cari max value di path a ke b

int main() {
 scanf("%d%d", &n, &q);
 for(int i=1; i<=n; i++) scanf("%d", &v[i]);
 for(int i=2; i<=n; i++) {
  int a, b;
  scanf("%d%d", &a, &b);
  adj[a].push_back(b);
  adj[b].push_back(a);
 }
 dfs_sz(1, 1);
 dfs_hld(1, 1, 1);
 while(q--) {
  int t;
  scanf("%d", &t);
  if(t == 1) {
   int s, x;
   scanf("%d%d", &s, &x);
   v[s] = x;
   update(id[s], v[s]);
  } else {
   int a, b;
   scanf("%d%d", &a, &b);
   int res = path(a,b);
   printf("%d ", res);
  }
 }
}
```

8 KMP

```
ll kmpt[100050];

void kmp(string s){
 kmpt[0] = -1; kmpt[1] = 0;

 ll cnd = 0;
 FOR(i, 2, s.length()){
  if(s[i-1] == s[cnd]){
   kmpt[i] = ++cnd;
  }
  else{
   while(cnd > 0 && s[i-1] != cnd) cnd = kmpt[cnd];
  }
 }
}
```

9 LineDistance

```
/*
Returns the signed distance between point p and the line con
    -
```

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

10 LineHullIntersection

```
/*
Line-convex polygon intersection. The polygon must be ccw
and have no colinear points. lineHull(line, poly) returns a
    pair describing the
intersection of a line with the polygon:  (-1, -1) if no
    collision,  (i, -1) if
touching the corner i,  (i, i) if along side (i, i + 1),  (i
    , j) if crossing sides
(i, i + 1) and (j, j + 1). In the last case, if a corner i
    is crossed, this is treated
as happening on side (i, i + 1). The points are returned in
    the same order as
the line hits the polygon. extrVertex returns the point of a
     hull with the
max projection onto a line.
*/
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;
 FOR(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;
}
```

11 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

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;
}
```

12 NTT

```
// TEMPLATE FFT/NTT AWOKWOK
const int mod = 998244353;

ll pang(ll x,ll y){
 if(x==0)return 0;
 if(y==0)return 1;
 if(y==1)return x;
 ll z=pang(x,y/2);
 return z*z%mod*pang(x,y%2)%mod;
}

const int root = pang(3,119);
const int root_1 = pang(root,mod-2);
const int root_pw = 1 << 23;

ll inv[300005],fact[300005],ifact[300005];

void fft(vector<ll> & a, bool invert) {
    int n = a.size();
```

```
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] *
                      w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v +
                     mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = inv[n];
        for (ll & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

13 OnSegment

```
/*
Returns true iff p lies on the line segment from s to e. Use
(segDist(s,e,p)<=epsilon) instead when using Point<double>.
*/
template<class P> bool onSegment(P s, P e, P p) {
 return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

14 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 d i s t ()
     =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 << ")"; }
};
```

15 PointInsideHull

```
/*
Determine whether a point t lies inside a convex hull (CCW
order, with no colinear points). Returns true if point lies
    within the hull. If
strict is true, points on the boundary arent included.
*/
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;
}
```

16 PolygonCenter

```
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;
}
```

17 Push Relabel

```
const int inf = 1000000000;

int n;
vector<vector<int>> capacity, flow;
vector<int> height, excess;

void push(int u, int v)
{
    int d = min(excess[u], capacity[u][v] - flow[u][v]);
    flow[u][v] += d;
    flow[v][u] -= d;
    excess[u] -= d;
    excess[v] += d;
}

void relabel(int u)
{
    int d = inf;
    for (int i = 0; i < n; i++) {
```

```
        if (capacity[u][i] - flow[u][i] > 0)
            d = min(d, height[i]);
    }
    if (d < inf)
        height[u] = d + 1;
}

vector<int> find_max_height_vertices(int s, int t) {
    vector<int> max_height;
    for (int i = 0; i < n; i++) {
        if (i != s && i != t && excess[i] > 0) {
            if (!max_height.empty() && height[i] > height[
                 max_height[0]])
                max_height.clear();
            if (max_height.empty() || height[i] == height[
                 max_height[0]])
                max_height.push_back(i);
        }
    }
    return max_height;
}

int max_flow(int s, int t)
{
    height.assign(n, 0);
    height[s] = n;
    flow.assign(n, vector<int>(n, 0));
    excess.assign(n, 0);
    excess[s] = inf;
    for (int i = 0; i < n; i++) {
        if (i != s)
            push(s, i);
    }

    vector<int> current;
    while (!(current = find_max_height_vertices(s, t)).empty
         ()) {
        for (int i : current) {
            bool pushed = false;
            for (int j = 0; j < n && excess[i]; j++) {
                if (capacity[i][j] - flow[i][j] > 0 && height[
                     i] == height[j] + 1) {
                    push(i, j);
                    pushed = true;
                }
            }
            if (!pushed) {
                relabel(i);
                break;
            }
        }
    }

    int max_flow = 0;
    for (int i = 0; i < n; i++)
        max_flow += flow[i][t];
    return max_flow;
}
```

18 SegmentDistance

```
/*
Returns the shortest distance between point p and the line
segment from point s to e.
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
*/
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;
}
```

19 SideOf

```
/*
Returns where p is as seen from s towards e. 1/0/-1 ? left/
    on
line/right. If the optional argument eps is given 0 is
    returned if p is within
distance eps from the line. P is supposed to be Point<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.
Usage: bool left = sideOf(p1,p2,q)==1;
*/
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);
}
```

20 String Automaton

```
struct state {
    int len, link;
    map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

```
// OP STRING ALGO AMORGOS
```

21 Treap

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

using namespace std;

typedef long long ll;
const ll LLINF = 2e16, LLBOUND = 2e15;

struct Node {
 ll val, mx, mn, mdiff;
 int size, priority;
 Node *l, *r;
 Node(ll _val) : val(_val), mx(_val), mn(_val), mdiff(LLINF)
      , size(1) {
  priority = rand();
 }
};

int size(Node *p) { return p == NULL ? 0 : p->size; }
ll getmax(Node *p) { return p == NULL ? -LLINF : p->mx; }
ll getmin(Node *p) { return p == NULL ? LLINF : p->mn; }
ll getmdiff(Node *p) { return p == NULL ? LLINF : p->mdiff;
    }
void update(Node *p) {
 if (p == NULL) return;
 p->size = 1 + size(p->l) + size(p->r);
 p->mx = max(p->val, max(getmax(p->l), getmax(p->r)));
 p->mn = min(p->val, min(getmin(p->l), getmin(p->r)));
 p->mdiff = LLINF;
 if (p->l != NULL)
  p->mdiff = min(p->mdiff, min(getmdiff(p->l), p->val -
       getmax(p->l)));
 if (p->r != NULL)
  p->mdiff = min(p->mdiff, min(getmdiff(p->r), getmin(p->r)
       - p->val));
}
void merge(Node *&t, Node *l, Node *r) {
 if (l == NULL)    { t = r; }
 else if (r == NULL) { t = l; }
 else if (l->priority > r->priority) {
  merge(l->r, l->r, r);
  t = l;
 } else {
  merge(r->l, l, r->l);
  t = r;
 }
 update(t);
}
void splitat(Node *t, Node *&l, Node *&r, int at) {
 if (t == NULL) { l = r = NULL; return; }
 int id = size(t->l);
 if (id > at) {
  splitat(t->l, l, t->l, at);
  r = t;
 } else {
  splitat(t->r, t->r, r, at - id - 1);
  l = t;
 }
 update(t);
}
ll Nquery(Node *t, int i, int j) {
 Node *l, *r;
 splitat(t, l, t, i - 1);
 splitat(t, t, r, j - i);
 ll ret = getmdiff(t);
 merge(t, l, t);
 merge(t, t, r);
 return (ret <= 0 || ret > LLBOUND ? -1 : ret);
}
ll Xquery(Node *t, int i, int j) {
 Node *l, *r;
 splitat(t, l, t, i - 1);
 splitat(t, t, r, j - i);
 ll ret = getmax(t) - getmin(t);
 merge(t, l, t);
 merge(t, t, r);
 return (ret <= 0 || ret > LLBOUND ? -1 : ret);
}
void split(Node *t, Node *&l, Node *&r, ll val) {
 if (t == NULL) { l = r = NULL; return; }
 if (t->val >= val) {
  split(t->l, l, t->l, val);
  r = t;
 } else {
  split(t->r, t->r, r, val);
  l = t;
 }
 update(t);
}
void insert(Node *&t, ll val) {
 Node *n = new Node(val), *l, *r;
 split(t, l, t, val);
 split(t, t, r, val + 1);
 merge(t, l, n);
 merge(t, t, r);
}
void erase(Node *&t, ll val, bool del = true) {
 Node *L, *rm;
 split(t, t, L, val);
 split(L, rm, L, val + 1);
 merge(t, t, L);
 if (del && rm != NULL) delete rm;
}
void inorder(Node *p) {
 if (p == NULL) return;
 inorder(p->l);
 cout << p->val << ' ';
 inorder(p->r);
}
void cleanup(Node *p) {
 if (p == NULL) return;
 cleanup(p->l); cleanup(p->r);
 delete p;
}

int main() {
 ios::sync_with_stdio(false);
 cin.tie(NULL);

 Node *tree = NULL;

 srand(time(NULL));

 int Q;
 cin >> Q;
 for (int q = 1; q <= Q; ++q) {
  char c;
  cin >> c;
  switch (c) {
   case 'I':
    ll k;
    cin >> k;
    insert(tree, k);
    break;
   case 'D':
    ll kd;
    cin >> kd;
    erase(tree, kd);
    break;
   case 'X':
    int l, r;
```

```
    cin >> l >> r;
    if (r - l < 1) cout << -1 << '\n';
    else cout << Xquery(tree, l, r) << '\n';
    break;
   case 'N':
    int ll, rr;
    cin >> ll >> rr;
    if (rr - ll < 1) cout << -1 << '\n';
    else cout << Nquery(tree, ll, rr) << '\n';
    break;
  }
//  cout << " ";
//  inorder(tree); cout << endl;
 }
 cout << flush;
 cleanup(tree);

 return 0;
}
```

22 aho corasick

```
#include <bits/stdc++.h>
using namespace std;
#define ff first
#define ss second
#define pb push_back
#define debug(val) cerr << "The value of " << #val << " is =
      " << val << '\n';
typedef long double ld;
typedef long long ll;
typedef unsigned long long ull;
const ld PI = 4*atan((ld)1);
const ll mod = 1e9 + 7;
const ll inf = 922337203685477;
const ll nax = 1e3 + 5;

const int K = 105;

struct Vertex {
    int next[K];
    vector<ll> leaf;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    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(1);

void add_string(string const& s, ll idx) {
    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.pb(idx);
}

int go(int v, char ch);

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];
}

ll tc, q;
string s, a[nax];
bool cek[nax], vis[nax];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie
        (NULL);
    //freopen("test.in", "r", stdin);
    //freopen("test.out", "w", stdout);

    cin >> tc;
    while(tc--){
  cin >> s;
  cin >> q;
  // reset
  memset(vis, 0, sizeof(vis));
  memset(cek, 0, sizeof(cek));
  t.clear();
  t.emplace_back();

  for(ll i = 1; i <= q; i++){
   cin >> a[i];
   add_string(a[i], i);
   cek[i] = 0;
  }
  ll cur = 0;
  for(auto it : s){
   cur = go(cur, it);
   if(!vis[cur]){
    for(auto each : t[cur].leaf){
     cek[each] = 1;
    }
    vis[cur] = 1;
   }
  }
  for(ll i = 1; i <= q; i++){
   if(cek[i]) cout << "y\n";
   else cout << "n\n";
  }
 }
}
```

23 bridgearticulation

```
int time;

void dfs(int u, int parent) {
    disc[u] = low[u] = time++;
    for (int v: adj[u]) {
        if (disc[v] == -1) {
            ++child[u];
            dfs(v, u);
            if (low[v] > disc[u]) {
                // (u, v) adalah bridge
            }
```

```
            if (low[v] >= disc[u]) {
                // u adalah articulation point
            }
            low[u] = min(low[u], low[v]);
        }
        else if (v != parent) {
            low[u] = min(low[u], disc[v]);
        }
    }
}

dfs(root, -1);
// Special case
if (child[root] < 2) {
    // root bukan articulation point
} else {
    // root adalah articulation point
}
```

24 centroid

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define ff first
#define ss second
#define pb push_back
const ll nax = 2e5 + 5;
const ll inf = 1e10;

ll n, m;
ll par[nax], removed[nax], sub[nax];
vector<vector<ll>> v(nax);

// Centroid

void get_sz(ll idx, ll bfr){
 sub[idx] = 1;
 for(auto y : v[idx]){
  if(y != bfr && !removed[y]){
   get_sz(y, idx);
   sub[idx] += sub[y];
  }
 }
}

ll find_centroid(ll idx){
 get_sz(idx, -1);
 ll tree = sub[idx];

 ll cek = 0;
 while(!cek){
  cek = 1;
  for(auto y : v[idx]){
   if(removed[y] || sub[y] > sub[idx]) continue;
   if(sub[y] > tree / 2){
    cek = 0;
    idx = y;
    break;
   }
  }
 }
 return idx;
}

void solve(ll idx){
  // Do smth here
}

ll built_centroid(ll idx){
 idx = find_centroid(idx);
  // Do smth here
 solve(idx);
 removed[idx] = 1;

 for(auto y : v[idx]){
  if(!removed[y]){
   ll nxt = built_centroid(y);
   par[nxt] = idx;
  }
 }
 return idx;
}

// Centroid

// Full Code Prob : CF 342E

int main(){
 ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(
     NULL);

 cin >> n >> m;
 for(ll i = 1; i < n; i++){
  ll x, y;
  cin >> x >> y;
  v[x].pb(y);
  v[y].pb(x);
 }
 built_centroid(1);
}
```

25 closest$_{p}air$

```
long long ClosestPair(vector<pair<int, int>> pts) {
    int n = pts.size();
    sort(pts.begin(), pts.end());
    set<pair<int, int>> s;

    long long best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; ++i) {
        int d = ceil(sqrt(best_dist));
        while (pts[i].first - pts[j].first >= d) {
            s.erase({pts[j].second, pts[j].first});
            j += 1;
        }

        auto it1 = s.lower_bound({pts[i].second - d, pts[i].
            first});
        auto it2 = s.upper_bound({pts[i].second + d, pts[i].
            first});

        for (auto it = it1; it != it2; ++it) {
            int dx = pts[i].first - it->second;
            int dy = pts[i].second - it->first;
            best_dist = min(best_dist, 1LL * dx * dx + 1LL *
                dy * dy);
        }
        s.insert({pts[i].second, pts[i].first});
    }
    return best_dist;
}
```

26 directed MST

```
/**
 * Author: chilli, Takanori MAEHARA, Benq, Simon Lindholm
 * Date: 2019-05-10
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/
     blob/master/graph/arborescence.cc
```

```
 * and https://github.com/bqi343/USACO/blob/42
     d177dfb9d6ce350389583cfa71484eb8ae614c/Implementations/
     content/graphs%20(12)/Advanced/DirectedMST.h for the
     reconstruction
 * Description: Finds a minimum spanning
 * tree/arborescence of a directed graph, given a root node.
      If no MST exists, returns -1.
 * Time: O(E \log V)
 * Status: Stress-tested, also tested on NWERC 2018
     fastestspeedrun
 */
#pragma once

#include "../data-structures/UnionFindRollback.h"

struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
 Edge key;
 Node *l, *r;
 ll delta;
 void prop() {
  key.w += delta;
  if (l) l->delta += delta;
  if (r) r->delta += delta;
  delta = 0;
 }
 Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
 if (!a || !b) return a ?: b;
 a->prop(), b->prop();
 if (a->key.w > b->key.w) swap(a, b);
 swap(a->l, (a->r = merge(b, a->r)));
 return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
 RollbackUF uf(n);
 vector<Node*> heap(n);
 for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
 ll res = 0;
 vi seen(n, -1), path(n), par(n);
 seen[r] = r;
 vector<Edge> Q(n), in(n, {-1,-1}), comp;
 deque<tuple<int, int, vector<Edge>>> cycs;
 rep(s,0,n) {
  int u = s, qi = 0, w;
  while (seen[u] < 0) {
   if (!heap[u]) return {-1,{}};
   Edge e = heap[u]->top();
   heap[u]->delta -= e.w, pop(heap[u]);
   Q[qi] = e, path[qi++] = u, seen[u] = s;
   res += e.w, u = uf.find(e.a);
   if (seen[u] == s) { /// found cycle, contract
    Node* cyc = 0;
    int end = qi, time = uf.time();
    do cyc = merge(cyc, heap[w = path[--qi]]);
    while (uf.join(u, w));
    u = uf.find(u), heap[u] = cyc, seen[u] = -1;
    cycs.push_front({u, time, {&Q[qi], &Q[end]}});
   }
  }
  rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
 }

 for (auto& [u,t,comp] : cycs) { // restore sol (optional)
  uf.rollback(t);
  Edge inEdge = in[u];
  for (auto& e : comp) in[uf.find(e.b)] = e;
  in[uf.find(inEdge.b)] = inEdge;
 }
 rep(i,0,n) par[i] = in[i].a;
 return {res, par};
}
```

27 dpcht

```
pii tpot(pii satu,pii dua){
    pii jwb;
    jwb.first=dua.second-satu.second;
    jwb.second=satu.first-dua.first;
    return jwb;
}
bool cmp(pii a,pii b){
    if(a.fi/a.se==b.fi/b.se){
        a.fi%=a.se;
        b.fi%=b.se;
        return a.fi*b.se<=b.fi*a.se;
    }
    return a.fi/a.se<=b.fi/b.se;
}
line.push_back({1,-pref[0]}); //cari maksimum , gradien non
     decreasing //m and c
//cari minimum gradien non increasing
for(LL i=2;i<=n;i++){
    LL x=a[i];
    LL ki=1,ka=(LL)line.size()-1,add=-1e18;
    while(ki<=ka){
        LL mid=(ki+ka)/2;
        LL l=line[mid-1].first*x+line[mid-1].second;
        LL r=line[mid].first*x+line[mid].second;
        add=max(add,max(l,r));
        if(l>r)ka=mid-1;
        else ki=mid+1;
        /*
        Minimum
        add=min(add,min(l,r))
        if(l>r)ki=mid+1;
        else ka=mid-1;
        */
    }
    if(line.size()==1)add=line[0].first*x+line[0].second;
    ans=max(ans,ret+add-a[i]*i+pref[i-1]); //tambahin
         constant
    pii now={i,-pref[i-1]};
    LL skg=line.size()-1,prev=line.size()-2;
    while(skg>0 && cmp(tpot(now,line[skg]),tpot(now,line[prev
         ])){
        //hapus yang gamasuk hull
        line.pop_back();
        skg--;
        prev--;
    }
    line.push_back(now);
}
```

28 dpdnc

```
void calc(int L,int R,int optL,int optR,int j){
    if(L>R)return;
    int mid=(L+R)/2;
    int res=2e9;
    int opt=-1;
    for(int i=optL;i<=min(optR,mid-1);i++){
        if(dp[j-1][i]+cost(i+1,mid)<res){
            res=dp[j-1][i]+cost(i+1,mid);
            opt=i;
        }
    }
    dp[j][mid]=res;
    calc(L,mid-1,optL,opt,j);
    calc(mid+1,R,opt,optR,j);
}
for(int i=1;i<=n;i++)dp[1][i]=cost(1,i);
for(int i=2;i<=k;i++){
```

```
    calc(i,n,i-1,n,i); //mau ngisi dp[i][...] dengan ... dari
          i sampai n karena dengan k gondola minimal k orang
}
```

29 dynsegtree

```
int tree[3000005],lazy[3000005],ki[3000005],ka[3000005],node
    =2;

//update x sampai y, jadiin 1 semua, query dari x sampe y (
    bisa sampe 1e9)


void pushdown(int now,int L,int R){
 int mid=(L+R)/2;
 if(ki[now]==0){
  ki[now]=node;
  node++;
 }
 if(ka[now]==0){
  ka[now]=node;
  node++;
 }
 tree[ki[now]]=mid-L+1;
 lazy[ki[now]]=1;
 tree[ka[now]]=R-mid;
 lazy[ka[now]]=1;
 lazy[now]=0;
}

void update(int now,int L,int R,int x,int y){
       if(tree[now]==R-L+1)return;
 if(L>=x && R<=y){
  tree[now]=R-L+1;
  lazy[now]=1;
  return;
 }
 if(L>y || R<x)return;
 int mid=(L+R)/2;
 if(lazy[now])pushdown(now,L,R);
 if(ki[now]==0){
  ki[now]=node;
  node++;
 }
 if(ka[now]==0){
  ka[now]=node;
  node++;
 }
 update(ki[now],L,mid,x,y);
 update(ka[now],mid+1,R,x,y);
 tree[now]=tree[ki[now]]+tree[ka[now]];
}

int query(int now,int L,int R,int x,int y){
 if(L>=x && R<=y)
 {
  return tree[now];
 }
 if(L>y || R<x || now==0)return 0;
 if(lazy[now])pushdown(now,L,R);
 int mid=(L+R)/2;
 return query(ki[now],L,mid,x,y)+query(ka[now],mid+1,R,x,y);
}
```

30 eulerian

```
void eulerian_path(int cur){
    stack<int> st;
    vector<int> ans;
    st.push(cur);
    //V is multiset
    while(!st.empty()){
        int cur = st.top();
        if(V[cur].size()){
            auto it = V[cur].begin();
            st.push(*it);
            V[cur].erase(it);
            //use this for bidirectional graph
            //if(V[*it].count(cur)){
            // V[*it].erase(V[*it].find(cur));
            //}
        }else{
            ans.pb(cur);
            st.pop();
        }
    }
}
```

31 fordfulkerson

```
LL bneck,adj[5005][5005],source,sink,ans=0,n;
bool visited[5005];

void dfs(LL node,LL bottleneck){
 if(node==sink){
  ans+=bottleneck;
  sudah=true;
  bneck=bottleneck;
  return;
 }
 if(!visited[node]){
  visited[node]=true;
  for(LL i=1;i<=n;i++){
   if(adj[node][i]>0){
    dfs(i,min(adj[node][i],bottleneck));
    if(sudah){
     adj[node][i]-=bneck;
     adj[i][node]+=bneck;
     return;
    }
   }
  }
 }
}

int main(){
 source=1,sink=n;
 sudah=true;
 while(sudah){
  memset(visited,false,sizeof(visited));
  sudah=false;
  dfs(source,1e18);
 }
 cout << ans << endl;
}
```

32 graham scan

```
/* Quick Note :
 * Jangan Mikir Lama - lama, sampahin dulu aja kalo OI
 * Always Try to reset
*/
#include <bits/stdc++.h>
using namespace std;
#define ff first
#define ss second
#define pb push_back
#define debug(val) cerr << "The value of " << #val << " is =
      " << val << '\n';
typedef long double ld;
typedef long long ll;
```

```
typedef unsigned long long ull;
const ll mod = 1e9 + 7;
const ll inf = 922337203685477;
const ll nax = 0;

struct point{
 ll x, y;
};

ll t, n;
vector<point> a;

ll cross(point p, point q, point r){
 ll val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q
      .y);
 if(val == 0){
  return 0;
 }
 else if(val > 0){
  return 1;
 }
 else{
  return -1;
 }
}

ll dist(point p, point q){
 ll dx = p.x - q.x, dy = p.y - q.y;
 return dx * dx + dy * dy;
}

bool cmp(point p, point q){
 ll order = cross(a[0], p, q);
 if(order == 0){
  return dist(a[0], p) < dist(a[0], q);
 }
 else{
  return (order == -1);
 }
}

// Problem : 681 - Convex Hull Finding - UVA

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie
         (NULL);
    //freopen("test.in", "r", stdin);
    //freopen("test.out", "w", stdout);

    cin >> t;
    cout << t << '\n';
    while(t--){
  a.clear();
  cin >> n;
  ll mini = 0;
  for(ll i = 0; i < n; i++){
   ll x, y;
   cin >> x >> y;
   a.pb({x, y});
   if(y < a[mini].y){
    mini = i;
   }
  }
  if(t){
   ll gbg;
   cin >> gbg;
  }
  // Jadiin satu titik sebagai titik acuan / pivot, titik
       yang dipakai adalah titik yang paling bawah
  swap(a[0], a[mini]);
  // Sort by polar angel
  sort(a.begin() + 1, a.end(), cmp);
  vector<point> v;
  for(ll i = 0; i < n; i++){
   if(v.size() < 2){
    v.pb(a[i]);
   }
   else{
    // Kalau Cross product nya tidak Counter Clockwise
         pop_back();
    while(v.size() >= 2 && cross(v[v.size()-2], v[v.size()
         -1], a[i]) != -1){
     v.pop_back();
    }
    v.pb(a[i]);
   }
  }
  cout << v.size() + 1 << '\n';
  for(auto p : v){
   cout << p.x << " " << p.y << '\n';
  }
  cout << a[0].x << " " << a[0].y << '\n';
  if(t){
   cout << "-1\n";
  }
 }
}
```

33 hungarian

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
#define pb push_back
#define ff first
#define ss second
const ld PI = 4 * atan((ld)1);
const ll nax = 25;
const ll inf = 1e16;

ll n;
ll ans;

ll dist(pair<ll,ll> x, pair<ll,ll> y){
 return abs(x.ff - y.ff) + abs(x.ss - y.ss);
}

ll hungarian(vector<pair<ll,ll>>&a, vector<pair<ll,ll>>&b){
  // pairing a ke b
 vector<ll> u(n + 1), v(n + 1), p(n + 1), way(n + 1);
 for(ll i = 1; i <= n; i++){
  p[0] = i;
  ll curM = 0;
  vector<ll> minv(n + 1, inf);
  vector<bool> used(n + 1, 0);
  while(p[curM] != 0){
   used[curM] = 1;
   ll curN = p[curM], delta = inf;
   ll nexM;
   for(int j = 1; j <= n; ++j){
    if(!used[j]){
     int cur = dist(a[curN-1], b[j-1]) - u[curN] - v[j];
     if(cur < minv[j]){
      minv[j] = cur, way[j] = curM;
     }
     if(minv[j] < delta){
      delta = minv[j], nexM = j;
     }
    }
   }
   for(int j = 0; j <= n; j++){
    if(used[j]){
     u[p[j]] += delta, v[j] -= delta;
    }
    else{
     minv[j] -= delta;
    }
```

```
   }
   curM = nexM;
  }
  do{
   ll nexM = way[curM];
   p[curM] = p[nexM];
   curM = nexM;
  }while(curM != 0);
 }
 return (-v[0]);
}

void make_diagonal(vector<pair<ll,ll>>&a){
 vector<pair<ll,ll>> b;
 for(ll i = 1; i <= n; i++){
  b.pb({i, i});
 }
 ans = min(ans, hungarian(a, b));
 b.clear();
 ll cnt = 1;
 for(ll i = n; i >= 1; i--){
  b.pb({cnt, i});
  cnt++;
 }
 ans = min(ans, hungarian(a, b));
}

void make_horizontal(vector<pair<ll,ll>> &a){
 vector<pair<ll,ll>> b;
 for(ll i = 1; i <= n; i++){
  for(ll j = 1; j <= n; j++){
   b.pb({i, j});
  }
  ans = min(ans, hungarian(a, b));
  b.clear();
 }
}

void make_vertical(vector<pair<ll,ll>> &a){
 vector<pair<ll,ll>> b;
 for(ll i = 1; i <= n; i++){
  for(ll j = 1; j <= n; j++){
   b.pb({j, i});
  }
  ans = min(ans, hungarian(a, b));
  b.clear();
 }
}

int main(){
 ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(
     NULL);

 ll ct = 0;
 while(1){
  cin >> n;
  ans = inf;
  if(n == 0){
   break;
  }
  vector<pair<ll,ll>> a;
  for(ll i = 1; i <= n; i++){
   ll x, y;
   cin >> x >> y;
   a.pb({x, y});
  }
  make_diagonal(a);
  make_horizontal(a);
  make_vertical(a);
  cout << "Board " << ++ct << ": " << ans << " moves
      required." << "\n\n";
 }

}
```

34 li chao

```
typedef long long ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) {
    return (conj(a) * b).x();
}

ftype f(point a, ftype x) {
    return dot(a, {x, 1});
}

const int maxn = 2e5;

point line[4 * maxn];

void add_line(point nw, int v = 1, int l = 0, int r = maxn)
     {
    int m = (l + r) / 2;
    bool lef = f(nw, l) < f(line[v], l);
    bool mid = f(nw, m) < f(line[v], m);
    if(mid) {
        swap(line[v], nw);
    }
    if(r - l == 1) {
        return;
    } else if(lef != mid) {
        add_line(nw, 2 * v, l, m);
    } else {
        add_line(nw, 2 * v + 1, m, r);
    }
}

ftype get(int x, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    if(r - l == 1) {
        return f(line[v], x);
    } else if(x < m) {
        return min(f(line[v], x), get(x, 2 * v, l, m));
    } else {
        return min(f(line[v], x), get(x, 2 * v + 1, m, r));
    }
}
```

35 mcbm

```
bool dfs(int now){
    if(visited[now])return false;
    visited[now]=true;
    for(auto nxt : adj[now]){
        if(match[nxt]==-1 || dfs(match[nxt])){
            match[nxt]=now;
            return true;
        }
    }
    return false;
}

memset(match,-1,sizeof(match));
for(int i=0;i<n;i++){
    memset(visited,0,sizeof(visited));
    if(dfs(i))matching++;
}
```

36 mo's

```
bool cmp(pair<pii,LL> a,pair<pii,LL> b){
 if(a.first.first/SQRT==b.first.first/SQRT)return a.first.
     second<b.first.second;
 return a.first.first/SQRT<b.first.first/SQRT;
}

sort(que.begin(),que.end(),cmp);
LL L=1,R=1;
for(auto isi : que){
 LL ki=isi.first.first,ka=isi.first.second;
 while(R<=ka)update(R++);
 while(L-1>=ki)update(--L);
   while(R-1>ka)remove(--R);
   while(L<ki)remove(L++);
   ans[isi.second]=ret;
}
```

37 pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<
int,
null_type,
less<int>, // mau multiset ganti jadi less_equal
rb_tree_tag,
tree_order_statistics_node_update>ordered_set;
ordered_set X;
*X.find_by_order(v) // elemen ke-v zero based
X.order_of_key(v) //banyaknya elemen yang < v
```

38 perssegtree

```
int tree[2*MAXN*LOG],ki[2*MAXN*LOG],ka[2*MAXN*LOG],a[MAXN],
    root[MAXN],idx,MAX,balik[MAXN];
map<int,int> mp;

int build(int L,int R){
 idx++;
 int no=idx;
 tree[no]=0;
 if(L==R)return no;
 int mid=(L+R)/2;
 ki[no]=build(L,mid);
 ka[no]=build(mid+1,R);
 return no;
}

int update(int bef,int L,int R,int x){
 idx++;
 int no=idx;
 tree[no]=tree[bef]+1;
 ki[no]=ki[bef];
 ka[no]=ka[bef];
 if(L==R)return no;
 int mid=(L+R)/2;
 if(x<=mid)ki[no]=update(ki[no],L,mid,x);
 else ka[no]=update(ka[no],mid+1,R,x);
 return no;
}

int query(int a,int b,int L,int R,int k){
 if(L==R)return L;
 int mid=(L+R)/2;
 int brp=tree[ki[b]]-tree[ki[a]];
 if(brp>=k)return query(ki[a],ki[b],L,mid,k);
 else return query(ka[a],ka[b],mid+1,R,k-brp);
}

root[0]=build(1,MAX); //seperti null
 for(int i=1;i<=n;i++)root[i]=update(root[i-1],1,MAX,mp[a[i
     ]]);
 while(q--){
  cin >> l >> r >> k;
  l++;k++;
  cout << balik[query(root[l-1],root[r],1,MAX,k)] << '\n';
 }
```

39 segment tree lazy

```
/* Quick Note :
 * Jangan Mikir Lama - lama, sampahin dulu aja kalo OI
 * Always Try to reset
*/
#include <bits/stdc++.h>
using namespace std;
#define ff first
#define ss second
#define pb push_back
#define debug(val) cerr << "The value of " << #val << " is =
      " << val << '\n';
typedef long double ld;
typedef int ll;
typedef unsigned long long ull;
const ld PI = 4*atan((ld)1);
const ll mod = 1e9 + 7;
const ll inf = 1e9;
const ll nax = 1e6 + 5;

struct info{
 ll four, sev, inc, dec;
};

ll n, m;
ll prop[4*nax];
info seg[4*nax];
string s;

info merge(info x, info y){
 info ret;
 ret.four = x.four + y.four;
 ret.sev = x.sev + y.sev;
 ret.inc = max({x.four + y.four, x.sev + y.sev, x.four + y.
      inc, x.inc + y.sev});
 ret.dec = max({x.four + y.four, x.sev + y.sev, x.sev + y.
      dec, x.dec + y.four});
 return ret;
}

void rev(ll x){
 swap(seg[x].four, seg[x].sev);
 swap(seg[x].inc, seg[x].dec);
}

void lazy(ll x){
 if(prop[x]){
  rev(2*x), rev(2*x+1);
  prop[2*x] ^= 1, prop[2*x+1] ^= 1;
  prop[x] = 0;
 }
}

void built(ll l, ll r, ll pos){
 if(l == r){
  seg[pos] = {s[l-1] == '4', s[l-1] == '7', 1, 1};
 }
 else{
  ll mid = (l + r) / 2;
  built(l, mid, 2*pos);
```

```
  built(mid + 1, r, 2*pos+1);
  seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
 }
}

void upd(ll l, ll r, ll pos, ll fl, ll fr){
 if(fl <= l && fr >= r){
  rev(pos);
  prop[pos] ^= 1;
 }
 else if(fl > r || fr < l){
  return;
 }
 else{
  lazy(pos);
  ll mid = (l + r) / 2;
  upd(l, mid, 2*pos, fl, fr);
  upd(mid + 1, r, 2*pos+1, fl, fr);
  seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
 }
}
```

40 sos

```
//DP SOS (Sum over submask)
for(int i=0;i<m;i++){
 for(int mask=(1<<m)-1;mask>=0;mask--){
  if(mask & (1<<i))dp[mask]+=dp[mask^(1<<i)];
 }
}
```

41 suffix array

```
#include <bits/stdc++.h>
using namespace std;
#define ff first
#define ss second
#define pb push_back
#define debug(val) cerr << "The value of " << #val << " is =
      " << val << '\n';
typedef long double ld;
typedef long long ll;
typedef unsigned long long ull;
const ld PI = 4*atan((ld)1);
const ll mod = 1e9 + 7;
const ll inf = 922337203685477;
const ll nax = 5e5 + 5;

ll n;
ll sa[nax], ra[nax];
ll tempSA[nax], tempRA[nax];
ll freq_radix[nax];
string s;

void radixSort(ll k){
 ll maxi = max(300ll, n);
 memset(freq_radix, 0, sizeof(freq_radix));
 for(ll i = 0; i < n; i++){
  if(i + k < n){
   freq_radix[ra[i+k]]++;
  }
  else{
   freq_radix[0]++;
  }
 }
 ll sum = 0;
 for(ll i = 0; i < maxi; i++){
  ll temp = freq_radix[i];
  freq_radix[i] = sum;
  sum += temp;
 }
 for(ll i = 0; i < n; i++){
  ll temp = sa[i] + k;
  if(temp < n){
   tempSA[freq_radix[ra[temp]]++] = sa[i];
  }
  else{
   tempSA[freq_radix[0]++] = sa[i];
  }
 }
 for(ll i = 0; i < n; i++){
  sa[i] = tempSA[i];
 }
}

void builtSA(){
 for(ll i = 0; i < n; i++){
  ra[i] = s[i];
  sa[i] = i;
 }
 for(ll k = 1; k < n; k *= 2){
  radixSort(k);
  radixSort(0);
  tempRA[sa[0]] = 0;
  ll r = 0;
  for(ll i = 1; i < n; i++){
   if(ra[sa[i]] == ra[sa[i-1]] && ra[sa[i]+k] == ra[sa[i-1]+
        k]){
    tempRA[sa[i]] = r;
   }
   else{
    tempRA[sa[i]] = ++r;
   }
  }
  for(ll i = 0; i < n; i++){
   ra[i] = tempRA[i];
  }
  if (ra[sa[n-1]] == n-1) break; // nice optimization trick
 }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie
         (NULL);
    //freopen("test.in", "r", stdin);
    //freopen("test.out", "w", stdout);

    /*
 contoh input
 qwedasd

 contoh output
 asd: URUTAN KE 1
 d: URUTAN KE 2
 dasd: URUTAN KE 3
 edasd: URUTAN KE 4
 qwedasd: URUTAN KE 5
 sd: URUTAN KE 6
 wedasd: URUTAN KE 7
 */

    cin >> s;
    s += '$';
    n = s.size();
    builtSA();
    for(ll i = 1; i < n; i++){
  for(ll j = sa[i]; j < n - 1; j++){
   cout << s[j];
  }
  cout << ": URUTAN KE " << i << '\n';
 }
}
```

42 unionrectangle

```
struct Edge {
    bool open;
    int x, yMin, yMax;
    Edge(int x, int y1, int y2, bool op) {
        this->x = x;
        yMin = y1, yMax = y2;
        open = op;
    }
    bool operator < (const Edge &e) const {
        return (x < e.x);
    }
};

int m, h[maxN << 1];
int sum[maxN << 5], counter[maxN << 5];
vector<Edge> edges;

void update(int p, int l, int r, int yMin, int yMax, bool
     open) {
  if (h[r] < yMin || yMax < h[l]) return;
    int c = p << 1, mid = (l + r) >> 1;
    if (yMin <= h[l] && h[r] <= yMax) {     // ymin --- h[l]
         --- h[r] --- ymax
        counter[p] += open ? 1 : -1;
        if (counter[p]) sum[p] = h[r] - h[l]; //if there is a
             rectangle at that posn that is bw h[l] and h[r]
             we will add that to length
        else sum[p] = sum[c] + sum[c + 1]; // else we will
             just sumup of lengths above and beloew this
             region
        return;
    }
    if (l + 1 >= r) return;
    update(c, l, mid, yMin, yMax, open);
    update(c + 1, mid, r, yMin, yMax, open);
    if (counter[p]) sum[p] = h[r] - h[l];
    else sum[p] = sum[c] + sum[c + 1];
}

long long solve() {
                                // process height for horzntl.
                                    sweep line
    sort(h + 1, h + m + 1); //  Sorting the hieght according
         to the y coordinates
    int k = 1;
    for(int i=2;i<=m;i++) if (h[i] != h[k]) // Deleting the
         same horizontal sweeplines
       h[++k] = h[i];              //     as they are redundant
          m = k;

    for (int i = 0, lm = (int)edges.size() << 4; i < lm; i++)
         // This is the initialization step of segment tree
       sum[i] = 0, counter[i] = 0;

    long long area = 0LL; // Initializing the Area
    sort(edges.begin(),edges.end()); // Sorting according to
         x coordinates for ver. swp line
    update(1, 1, m, edges[0].yMin, edges[0].yMax, edges[0].
         open);
    for (int i = 1; i < edges.size(); i++) {
        area += sum[1] * (long long)(edges[i].x - edges[i -
             1].x);
        update(1, 1, m, edges[i].yMin, edges[i].yMax, edges[i
             ].open);
    }
    return area;
}

int main(){
    edges.pb(Edge(x1, y1, y2, true)); // Inserting the Left
         edge
    edges.pb(Edge(x2, y1, y2, false)); // Inserting the Right
          Edge
    /*(x1,y2)    (x2,y2)
               _________
              |         |
   LeftEdge <- |       |-> Right Edge
              |_________|
           (x1,y1)    (x2,y1)
    */
    h[++m] = y1; // Inserting the Lower y Coordinate 1 based
         inddexiing
    h[++m] = y2; // Inserting the Upper y Coordinate
    solve();
}
```

43 xor 1 to n

```
int computeXOR(int n)
{

// If n is a multiple of 4
if (n % 4 == 0)
 return n;

// If n%4 gives remainder 1
if (n % 4 == 1)
 return 1;

// If n%4 gives remainder 2
if (n % 4 == 2)
 return n + 1;

// If n%4 gives remainder 3
return 0;
}
```