My Awesome Library

Mohaned El-Haddad

Computational Geometry

```cpp
//points must be sorted anti-clockwise

typedef complex<double> point;
#define vec(a, b) ((b)-(a))
#define cross(a, b) ((conj(a)*(b)).imag())

bool pointInConvexPolygon(vector<point> &poly, point p) {
    //points must be sorted anti-clockwise
    int n = poly.size();
    int l = 1, r = n - 1;
    while (l < r - 1) {
        int mid = (l + r) >> 1;
        ll ok = cross(vec(poly[0], p), vec(poly[mid], p));
        if (ok < 0) r = mid;
        else l = mid;
    }
    if (cross(vec(poly[0], p), vec(poly[l], p)) < 0) return false;
    if (cross(vec(poly[l], p), vec(poly[r], p)) < 0) return false;
    if (cross(vec(poly[r], p), vec(poly[0], p)) < 0) return false;
    return true;
}
```

```cpp
#include<bits/stdc++.h>
#include<ext/numeric>
#define ll long long
#define S second
#define F first

using namespace __gnu_cxx;
using namespace std;

const ll N = 2e3 + 5, M = 3e7 + 5;

const long double EPS = 1e-12, twoPi = 2 * acosl(-1);

/*
*       u
*      /|\
*     / | \
*   a/ | \b
*   / |m \
* /____|____\
* p h c-h q
*/

typedef complex<long double> point;

#define X real()
#define Y imag()
#define angle(v) (atan2((v).Y,(v).X))
#define vec(a,b) ((b)-(a))
#define length(v) ((long double)hypot((v).Y,(v).X))
#define dot(a,b) ((conj(a)*(b)).real())
#define cross(a,b) ((conj(a)*(b)).imag())
#define normalize(p) ((p)/length(p))
#define perp(a) (point(-(a).Y,(a).X))

point points[N];

int dcmp(const double &a, const double &b) {
    if (fabs(a - b) < EPS)
        return 0;
    return ((a > b) << 1) - 1;
}

int triangleThirdPoint(const point &p, const point &q, const double &a,
                       const double &b, point &u1, point &u2) {
    point pq = vec(p, q);
    double c = length(pq);
    double arr[] = {a, b, c};
    sort(arr, arr + 3);
    if (dcmp(arr[0] + arr[1], arr[2]) < 0)
        return false;
    //m^2=a^2-h^2
    //m^2=b^2-(c-h)^2
    //m^2=b^2-(c^2-2ch+h^2)
    //m^2=b^2-c^2+2ch-h^2
    //a^2-h^2=b^2-c^2+2ch-h^2
    //0=b^2-c^2+2ch-h^2-a^2+h^2
    //0=b^2-c^2+2ch-a^2
    //2ch=a^2-b^2+c^2
    //h=(a^2-b^2+c^2)/2c
    long double h = (a * a - b * b + c * c) / (2.0 * c);
    long double sq = a * a - h * h;
    if (!dcmp(sq, 0))sq = 0;
    long double m = sqrt(sq);
    point npq = normalize(pq);
    point prp = perp(npq);
    u1 = p + (npq * h) + m * prp;
    u2 = p + (npq * h) - m * prp;
    return 1 + (dcmp(arr[0] + arr[1], arr[2]) != 0);
```

```
}

long double zabat(long double x)
{
    return fmod(x + twoPi, twoPi);
}

int main()
{
//    freopen("input.in", "rt", stdin);
    int n;
    double r;
    int x, y;
    while(scanf("%d%lf", &n, &r), n or r)
    {
        for(int i = 0 ; i < n ; i++)
        {
            scanf("%d%d", &x, &y);
            points[i] = point(x, y);
        }
        int mx = 1;
        for(int i = 0 ; i < n ; i++)
        {
            point pv = points[i];
            vector<pair<long double, int> > events;
            for(int j = 0 ; j < n ; j++)
            {
                if(j == i) continue;
                point a, b;
                if(triangleThirdPoint(pv, points[j], r, r, a, b))
                {
                    double a1 = zabat(angle(vec(pv, b)));
                    double a2 = zabat(angle(vec(pv, a)));
                    if(a2 < a1) a2 += twoPi;
                    events.emplace_back(a1, -1);
                    events.emplace_back(a2, 1);
                }
            }
            sort(events.begin(), events.end());
            int cnt = 1;
            for(auto &e:events)
            {
                cnt -= e.S;
                mx = max(mx, cnt);
            }
        }
        printf("It is possible to cover %d points.\n", mx);
    }
    return 0;
}
```

```cpp
/*
 * Given n points on the plane, each represented by (x, y) coordinates,
 * find a pair of points with the smallest distance between them.
 */
#include<bits/stdc++.h>
#include<ext/numeric>
#define ll long long
#define S second
#define F first
#define Y second
#define X first

using namespace __gnu_cxx;
using namespace std;

const ll N = 2e5 + 5, M = 3e7 + 5;

int main()
{
//    freopen("input.in", "rt", stdin);
    int n;
    double x, y;
    scanf("%d", &n);
    map<double, multiset<double> > points;
    double mn = 1/0.0;
    while(n--)
    {
        scanf("%lf%lf", &x, &y);
        points[x].emplace(y);
    }
    for(auto &it:points)
    {
        auto &x = it.F;
        for(auto &y:it.S)
        {
            auto endX = points.upper_bound(x + mn);
            for(auto it1 = points.lower_bound(x) ; it1 != endX ; it1++)
            {
                auto &x1 = it1->F;
                auto &ys = it1->S;
                auto endY = ys.upper_bound(y + mn);
                for(auto it2 = ys.lower_bound(y - mn) ; it2 != endY ; it2++)
                {
                    auto &y1 = *it2;
                    if(&x == &x1 and &y == &y1) continue;
                    mn = min(mn, hypot(x - x1, y - y1));
                }
            }
        }
    }
    printf("%.6lf\n", mn);
    return 0;
}
```

Computational Geometry

```cpp
typedef complex<double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define X real()
#define Y imag()
#define vec(a, b) ((b)-(a))
#define cross(a, b) ((conj(a)*(b)).imag())

struct cmp {
    point about;

    cmp(point c) : about(c) {}

    bool operator()(const point &p, const point &q) const {
        double cr = cross(vec(about, p), vec(about, q));
        if (fabs(cr) < EPS)//collinear point
            return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);//return point closer
to pivot
        return cr > 0;//return true i p is on the right of q
    }
};

void sortAntiClockWise(vector <point> &pnts) {
    point mn(1 / 0.0, 1 / 0.0);//OO,OO
    //get lowest point if tie than leftmost one of them
    for (int i = 0; i < sz(pnts); i++)
        if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))
            mn = pnts[i];
    sort(all(pnts), cmp(mn));
}

void convexHull(vector <point> pnts, vector <point> &convex) {
    sortAntiClockWise(pnts);
    convex.clear();
    convex.push_back(pnts[0]);
    if (sz(pnts) == 1) return;
    convex.push_back(pnts[1]);
    for (int i = 2; i <= sz(pnts); i++) {
        point c = pnts[i % sz(pnts)];
        while (sz(convex) > 1) {
            point b = convex.back();
            point a = convex[sz(convex) - 2];
            //if top 2 points and next point can form left turn then it's okay to
insert c
            if (cross(vec(b, a), vec(b, c)) < -EPS) break;
            convex.pop_back();
        }
        if (i < sz(pnts))
            convex.push_back(pnts[i]);
    }
}
```

```
%%prettify
{
{
{
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <complex>
#include <iostream>

using namespace std;

typedef complex<long double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define OO 1e9
#define X real()
#define Y imag()
#define vec(a, b) ((b)-(a))
#define polar(r, t) ((r)*exp(point(0,(t))))
#define angle(v) (atan2((v).Y,(v).X))
#define length(v) ((long double)hypot((v).Y,(v).X))
#define lengthSqr(v) (dot(v,v))
#define dot(a, b) ((conj(a)*(b)).real())
#define cross(a, b) ((conj(a)*(b)).imag())
#define rotate(v, t) (polar(v,t))
#define rotateabout(v, t, a)  (rotate(vec(a,v),t)+(a))
#define reflect(p, m) ((conj((p)/(m)))*(m))
#define normalize(p) ((p)/length(p))
#define same(a, b) (lengthSqr(vec(a,b))<EPS)
#define mid(a, b) (((a)+(b))/point(2,0))
#define perp(a) (point(-(a).Y,(a).X))
#define colliner pointOnLine

namespace std {
    bool operator<(const point &a, const point &b) {
        return real(a) != real(b) ? real(a) < real(b) : imag(a) < imag(b);
    }
}
enum STATE {
    IN, OUT, BOUNDRY
};

bool intersect(const point &a, const point &b, const point &p, const point &q,
               point &ret) {

    //handle degenerate cases (2 parallel lines, 2 identical lines,   line is 1
point)

    double d1 = cross(p - a, b - a);
    double d2 = cross(q - a, b - a);
    ret = (d1 * q - d2 * p) / (d1 - d2);
    if (fabs(d1 - d2) > EPS) return 1;
    return 0;
}


bool pointOnLine(const point &a, const point &b, const point &p) {
    if (same(a, b))return same(a, p);//line is point
    return fabs(cross(vec(a, b), vec(a, p))) < EPS;
}

bool pointOnRay(const point &a, const point &b, const point &p) {
    //IMP NOTE: a,b,p must be collinear
    return dot(vec(a, p), vec(a, b)) > -EPS;
}

bool pointOnSegment(const point &a, const point &b, const point &p) {
```

```
    if (!colliner(a, b, p)) return 0;
    return pointOnRay(a, b, p) && pointOnRay(b, a, p);
}

long double pointLineDist(const point &a, const point &b, const point &p) {
    if (same(a, b))return length(vec(a, p));//line is point
    return fabs(cross(vec(a, b), vec(a, p)) / length(vec(a, b)));
}

long double pointSegmentDist(const point &a, const point &b, const point &p) {
    if (dot(vec(a, b), vec(a, p)) < EPS)
        return length(vec(a, p));
    if (dot(vec(b, a), vec(b, p)) < EPS)
        return length(vec(b, p));
    return pointLineDist(a, b, p);
}

int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
    return abs(__gcd(x1 - x2, y1 - y2)) + 1;
}

long double triangleAreaBH(long double b, long double h) {
    return b * h / 2;
}

long double triangleArea2sidesAngle(long double a, long double b, long double t) {
    return fabs(a * b * sin(t) / 2);
}

long double triangleArea2anglesSide(long double t1, long double t2,
                                    long double s) {
    return fabs(s * s * sin(t1) * sin(t2) / (2 * sin(t1 + t2)));
}

long double triangleArea3sides(long double a, long double b, long double c) {
    long double s((a + b + c) / 2);
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

long double triangleArea3points(const point &a, const point &b, const point &c) {
    return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;
}

//count interior
int picksTheorm(int a, int b) {
    return a - b / 2 + 1;
}

//get angle opposite to side a
long double cosRule(long double a, long double b, long double c) {
    // Handle denom = 0
    long double res = (b * b + c * c - a * a) / (2 * b * c);
    if (fabs(res - 1) < EPS)
        res = 1;
    if (fabs(res + 1) < EPS)
        res = -1;
    return acos(res);
}

long double sinRuleAngle(long double s1, long double s2, long double a1) {
    // Handle denom = 0
    long double res = s2 * sin(a1) / s1;
    if (fabs(res - 1) < EPS)
        res = 1;
    if (fabs(res + 1) < EPS)
        res = -1;
    return asin(res);
}
```

```
long double sinRuleSide(long double s1, long double a1, long double a2) {
    // Handle denom = 0
    long double res = s1 * sin(a2) / sin(a1);
    return fabs(res);
}

int circleLineIntersection(const point &p0, const point &p1, const point &cen,
                           long double rad, point &r1, point &r2) {

    // handle degenerate case if p0 == p1
    long double a, b, c, t1, t2;
    a = dot(p1 - p0, p1 - p0);
    b = 2 * dot(p1 - p0, p0 - cen);
    c = dot(p0 - cen, p0 - cen) - rad * rad;
    double det = b * b - 4 * a * c;
    int res;
    if (fabs(det) < EPS)
        det = 0, res = 1;
    else if (det < 0)
        res = 0;
    else
        res = 2;
    det = sqrt(det);
    t1 = (-b + det) / (2 * a);
    t2 = (-b - det) / (2 * a);
    r1 = p0 + t1 * (p1 - p0);
    r2 = p0 + t2 * (p1 - p0);
    return res;
}

int circleCircleIntersection(const point &c1, const long double &r1,
                             const point &c2, const long double &r2, point &res1,
point &res2) {
    if (same(c1, c2) && fabs(r1 - r2) < EPS) {
        res1 = res2 = c1;
        return fabs(r1) < EPS ? 1 : OO;
    }
    long double len = length(vec(c1, c2));
    if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
        point d, c;
        long double r;
        if (r1 > r2)
            d = vec(c1, c2), c = c1, r = r1;
        else
            d = vec(c2, c1), c = c2, r = r2;
        res1 = res2 = normalize(d) * r + c;
        return 1;
    }
    if (len > r1 + r2 || len < fabs(r1 - r2))
        return 0;
    long double a = cosRule(r2, r1, len);
    point c1c2 = normalize(vec(c1, c2)) * r1;
    res1 = rotate(c1c2, a) + c1;
    res2 = rotate(c1c2, -a) + c1;
    return 2;
}

void circle2(const point &p1, const point &p2, point &cen, long double &r) {
    cen = mid(p1, p2);
    r = length(vec(p1, p2)) / 2;
}

bool circle3(const point &p1, const point &p2, const point &p3, point &cen,
             long double &r) {
    point m1 = mid(p1, p2);
    point m2 = mid(p2, p3);
    point perp1 = perp(vec(p1, p2));
    point perp2 = perp(vec(p2, p3));
    bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
```

```
    r = length(vec(cen, p1));
    return res;
}

STATE circlePoint(const point &cen, const long double &r, const point &p) {
    long double lensqr = lengthSqr(vec(cen, p));
    if (fabs(lensqr - r * r) < EPS)
        return BOUNDRY;
    if (lensqr < r * r)
        return IN;
    return OUT;
}

int tangentPoints(const point &cen, const long double &r, const point &p,
                  point &r1, point &r2) {
    STATE s = circlePoint(cen, r, p);
    if (s != OUT) {
        r1 = r2 = p;
        return s == BOUNDRY;
    }
    point cp = vec(cen, p);
    long double h = length(cp);
    long double a = acos(r / h);
    cp = normalize(cp) * r;
    r1 = rotate(cp, a) + cen;
    r2 = rotate(cp, -a) + cen;
    return 2;
}

typedef pair <point, point> segment;

void getCommonTangents(point c1, double r1, point c2, double r2, vector <segment>
&res) {
    if (r1 < r2) swap(r1, r2), swap(c1, c2);
    double d = length(c1 - c2);
    double theta = acos((r1 - r2) / d);
    point v = c2 - c1;
    v = v / hypot(v.imag(), v.real());
    point v1 = v * exp(point(0, theta));
    point v2 = v * exp(point(0, -theta));
    res.clear();
    res.push_back(segment(c1 + v1 * r1, c2 + v1 * r2));
    res.push_back(segment(c1 + v2 * r1, c2 + v2 * r2));
    theta = acos((r1 + r2) / d);
    v1 = v * exp(point(0, theta));
    v2 = v * exp(point(0, -theta));
    res.push_back(segment(c1 + v1 * r1, c2 - v1 * r2));
    res.push_back(segment(c1 + v2 * r1, c2 - v2 * r2));
}

// minimum enclosing circle
//init p array with the points and ps with the number of points
//cen and rad are result circle
//you must call random_shuffle(p,p+ps); before you call mec
#define MAXPOINTS 100000
point p[MAXPOINTS], r[3], cen;
int ps, rs;
long double rad;

void mec() {
    if (rs == 3) {
        circle3(r[0], r[1], r[2], cen, rad);
        return;
    }
    if (rs == 2 && ps == 0) {
        circle2(r[0], r[1], cen, rad);
        return;
    }
    if (!ps) {
```

```
            cen = r[0];
            rad = 0;
            return;
        }
        ps--;
        mec();
        if (circlePoint(cen, rad, p[ps]) == OUT) {
            r[rs++] = p[ps];
            mec();
            rs--;
        }
        ps++;

    }

    //to check if the points are sorted anti-clockwise or clockwise
    //remove the fabs at the end and it will return -ve value if clockwise
    long double polygonArea(const vector <point> &p) {
        long double res = 0;
        for (int i = 0; i < sz(p); i++) {
            int j = (i + 1) % sz(p);
            res += cross(p[i], p[j]);
        }
        return fabs(res) / 2;
    }

    // return the centroid point of the polygon
    // The centroid is also known as the "centre of gravity" or the "center of mass".
    The position of the centroid
    // assuming the polygon to be made of a material of uniform density.
    point polyginCentroid(vector <point> &polygon) {
        long double a = 0;
        long double x = 0.0, y = 0.0;
        for (int i = 0; i < (int) polygon.size(); i++) {
            int j = (i + 1) % polygon.size();

            x += (polygon[i].X + polygon[j].X) * (polygon[i].X * polygon[j].Y
                                                  - polygon[j].X * polygon[i].Y);

            y += (polygon[i].Y + polygon[j].Y) * (polygon[i].X * polygon[j].Y
                                                  - polygon[j].X * polygon[i].Y);

            a += polygon[i].X * polygon[j].Y - polygon[i].Y * polygon[j].X;
        }

        a *= 0.5;
        x /= 6 * a;
        y /= 6 * a;

        return point(x, y);
    }

    int picksTheorm(vector <point> &p) {
        long double area = 0;
        int bound = 0;
        for (int i = 0; i < sz(p); i++) {
            int j = (i + 1) % sz(p);
            area += cross(p[i], p[j]);
            point v = vec(p[i], p[j]);
            bound += abs(__gcd((int) v.X, (int) v.Y));
        }
        area /= 2;
        area = fabs(area);
        return round(area - bound / 2 + 1);
    }

    void polygonCut(const vector <point> &p, const point &a, const point &b, vector <
    point> &res) {
        res.clear();
```

```
    for (int i = 0; i < sz(p); i++) {
        int j = (i + 1) % sz(p);
        bool in1 = cross(vec(a, b), vec(a, p[i])) > EPS;
        bool in2 = cross(vec(a, b), vec(a, p[j])) > EPS;
        if (in1)
            res.push_back(p[i]);
        if (in1 ^ in2) {
            point r;
            intersect(a, b, p[i], p[j], r);
            res.push_back(r);
        }
    }
}

//assume that both are anti-clockwise
void convexPolygonIntersect(const vector <point> &p, const vector <point> &q,
                            vector <point> &res) {
    res = q;
    for (int i = 0; i < sz(p); i++) {
        int j = (i + 1) % sz(p);
        vector <point> temp;
        polygonCut(res, p[i], p[j], temp);
        res = temp;
        if (res.empty())
            return;
    }
}

void voronoi(const vector <point> &pnts, const vector <point> &rect, vector <
vector<point>> &res) {
    res.clear();
    for (int i = 0; i < sz(pnts); i++) {
        res.push_back(rect);
        for (int j = 0; j < sz(pnts); j++) {
            if (j == i)
                continue;
            point p = perp(vec(pnts[i], pnts[j]));
            point m = mid(pnts[i], pnts[j]);
            vector <point> temp;
            polygonCut(res.back(), m, m + p, temp);
            res.back() = temp;
        }
    }
}

STATE pointInPolygon(const vector <point> &p, const point &pnt) {
    point p2 = pnt + point(1, 0);
    int cnt = 0;
    for (int i = 0; i < sz(p); i++) {
        int j = (i + 1) % sz(p);
        if (pointOnSegment(p[i], p[j], pnt))
            return BOUNDRY;
        point r;
        if (!intersect(pnt, p2, p[i], p[j], r))
            continue;
        if (!pointOnRay(pnt, p2, r))
            continue;
        if (same(r, p[i]) || same(r, p[j]))
            if (fabs(r.Y - min(p[i].Y, p[j].Y)) < EPS)
                continue;
        if (!pointOnSegment(p[i], p[j], r))
            continue;
        cnt++;
    }
    return cnt & 1 ? IN : OUT;
}

struct cmp {
    point about;
```

```
    cmp(point c) {
        about = c;
    }

    bool operator()(const point &p, const point &q) const {
        double cr = cross(vec(about, p), vec(about, q));
        if (fabs(cr) < EPS)
            return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);
        return cr > 0;
    }
};

void sortAntiClockWise(vector <point> &pnts) {
    point mn(1 / 0.0, 1 / 0.0);
    for (int i = 0; i < sz(pnts); i++)
        if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))
            mn = pnts[i];

    sort(all(pnts), cmp(mn));
}

void convexHull(vector <point> pnts, vector <point> &convex) {
    sortAntiClockWise(pnts);
    convex.clear();
    convex.push_back(pnts[0]);
    if (sz(pnts) == 1)
        return;
    convex.push_back(pnts[1]);
    if (sz(pnts) == 2) {
        if (same(pnts[0], pnts[1]))
            convex.pop_back();
        return;
    }
    for (int i = 2; i <= sz(pnts); i++) {
        point c = pnts[i % sz(pnts)];
        while (sz(convex) > 1) {
            point b = convex.back();
            point a = convex[sz(convex) - 2];
            if (cross(vec(b, a), vec(b, c)) < -EPS)
                break;
            convex.pop_back();
        }
        if (i < sz(pnts))
            convex.push_back(pnts[i]);
    }
}

}}}
```

```
double triangleArea3points(const point &a, const point &b,
                           const point &c) {
    return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;
}

bool pointInTriangle(point a, point b, point c, point p) {
    double s1 = triangleArea3points(a, b, c);
    double s2 = triangleArea3points(p, a, b) + triangleArea3points(p, a, c) +
triangleArea3points(p, b, c);
    return fabs(s1 - s2) < EPS;
}
```

```
typedef complex<double> point;
#define vec(a, b) ((b)-(a))
#define cross(a, b) ((conj(a)*(b)).imag())
#define EPS 1e-9
#define lengthSqr(v) (dot(v,v))
#define dot(a, b) ((conj(a)*(b)).real())
#define colliner pointOnLine
#define same(a, b) (lengthSqr(vec(a,b))<EPS)


double triangleArea3points(const point &a, const point &b, const point &c) {
    return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;
}

bool pointOnLine(const point &a, const point &b, const point &p) {
    // degenerate case: line is a point
    if (same(a, b))return same(a, p);
    return fabs(cross(vec(a, b), vec(a, p))) < EPS;
}

bool pointOnRay(const point &a, const point &b, const point &p) {
    //IMP NOTE: a,b,p must be collinear
    return dot(vec(a, p), vec(a, b)) > -EPS;
}

bool pointOnSegment(const point &a, const point &b, const point &p) {
    if (!colliner(a, b, p)) return 0;
    return pointOnRay(a, b, p) && pointOnRay(b, a, p);
}

bool pointInTriangle(point a, point b, point c, point p) {
    double s1 = triangleArea3points(a, b, c);
    double s2 = triangleArea3points(p, a, b) + triangleArea3points(p, a, c) +
triangleArea3points(p, b, c);
    if (fabs(s1) > EPS) return fabs(s1 - s2) < EPS;
    else return pointOnSegment(a, b, p) || pointOnSegment(a, c, p) ||
pointOnSegment(b, c, p);
}
```

```cpp
double triangleArea3sides(long double a, long double b, long double c) {
    long double s((a + b + c) / 2);
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

double get_rad_circumscribed_triangle(double a, double b, double c) {
    //given triangle abc return radius of circle in it
    double k = triangleArea3sides(a, b, c);
    if (a + b + c < EPS) return 0;
    return 2.0 * k / (a + b + c);
}
```

```cpp
typedef complex<double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define X real()
#define Y imag()
#define vec(a, b) ((b)-(a))
#define cross(a, b) ((conj(a)*(b)).imag())

void sortAntiClockWise(vector <point> &pnts) {
    point mn(1 / 0.0, 1 / 0.0);//OO,OO
    //get lowest point if tie than leftmost one of them
    for (int i = 0; i < sz(pnts); i++)
        if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))
            mn = pnts[i];
    //compare with respect to mn
    auto cmp = [mn](const point &p, const point &q) {
        double cr = cross(vec(mn, p), vec(mn, q));
        if (fabs(cr) < EPS)//collinear point
            return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);//return point closer
to pivot
        return cr > 0;//return true i p is on the right of q
    };
    sort(all(pnts), cmp);
}
```

```
typedef complex<double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define X real()
#define Y imag()
#define vec(a, b) ((b)-(a))
#define cross(a, b) ((conj(a)*(b)).imag())

struct cmp {
    point about;

    cmp(point c) : about(c) {}

    bool operator()(const point &p, const point &q) const {
        double cr = cross(vec(about, p), vec(about, q));
        if (fabs(cr) < EPS)//collinear point
            return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);//return point closer
to pivot
        return cr > 0;//return true i p is on the right of q
    }
};

void sortAntiClockWise(vector <point> &pnts) {
    point mn(1 / 0.0, 1 / 0.0);//OO,OO
    //get lowest point if tie than leftmost one of them
    for (int i = 0; i < sz(pnts); i++)
        if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))
            mn = pnts[i];
    sort(all(pnts), cmp(mn));
}
```

```
/*




/*
 *        u
 *       /|\
 *      / | \
 *    a/  |  \b
 *    /   |m  \
 *   /____|____\
 *  p  h   c-h  q
 */

int dcmp(const double &a, const double &b) {
    if (fabs(a - b) < EPS)
        return 0;
    return ((a > b) << 1) - 1;
}

int triangleThirdPoint(const point &p, const point &q, const double &a,
                       const double &b, point &u1, point &u2) {
    point pq = vec(p, q);
    double c = length(pq);
    double arr[] = { a, b, c };
    sort(arr, arr + 3);
    if (dcmp(arr[0] + arr[1], arr[2]) < 0)
        return false;

    //m^2=a^2-h^2
    //m^2=b^2-(c-h)^2
    //m^2=b^2-(c^2-2ch+h^2)
    //m^2=b^2-c^2+2ch-h^2
    //a^2-h^2=b^2-c^2+2ch-h^2
    //0=b^2-c^2+2ch-h^2-a^2+h^2
    //0=b^2-c^2+2ch-a^2
    //2ch=a^2-b^2+c^2
    //h=(a^2-b^2+c^2)/2c
    double h = (a * a - b * b + c * c) / (2.0 * c);
    double sq=a * a - h * h;
    if(!dcmp(sq,0))sq=0;
    double m = sqrt(sq);
    point npq = normalize(pq);
    point prp = perp(npq);
    u1 = p + (npq * h) + m * prp;
    u2 = p + (npq * h) - m * prp;
    return 1 + (dcmp(arr[0] + arr[1], arr[2]) != 0);
}



*/
```

```cpp
#include<bits/stdc++.h>
#include<ext/numeric>
#define ll long long
#define S second
#define F first
#define Y second
#define X first

using namespace __gnu_cxx;
using namespace std;

const ll N = 2e5 + 5, M = 3e7 + 5;

typedef pair<int, int> point;
typedef pair<point, point> segment;
typedef tuple<int, int, ll> line;

line getLine(const segment &seg)
{
    int dx = seg.S.X - seg.F.X;
    int dy = seg.S.Y - seg.F.Y;
    ll a = dy, b = -dx, c = 1LL * -a * seg.F.X - 1LL * b * seg.F.Y;
    ll g = llabs(__gcd(__gcd(a, b), c));
    if(a < 0 or (!a and b < 0)) g = -g;
    a /= g, b /= g, c /= g;
    return tie(a, b, c);
}

int getCoord()
{
    double x;
    scanf("%lf", &x);
    return x * 100 + 0.5;
}

int main()
{
//    freopen("input.in", "rt", stdin);
    int n, x, y;
    while(scanf("%d", &n), n)
    {
        int res = 0;
        map<line, vector<pair<point, int> > > lines;
        while(n--)
        {
            x = getCoord(), y = getCoord();
            point p1(x, y);
            x = getCoord(), y = getCoord();
            point p2(x, y);
            if(p1 > p2) swap(p1, p2);
            line l = getLine({p1, p2});
            auto &v = lines[l];
            v.emplace_back(p1, -1);
            v.emplace_back(p2, 1);
        }
        for(auto &l:lines)
        {
            auto &v = l.S;
            sort(v.begin(), v.end());
            int c = 0;
            for(auto &i:v)
            {
                c -= i.S;
                res += !c;
            }
        }
        printf("%d\n", res);
    }
    return 0;
```

```
}
```

Data Strucure

```cpp
struct BIT {
    vector <ll> tree;

    BIT(int n) {
        tree.resize(n + 1);
    }

    BIT(vector <ll> &arr) {
        tree.resize(arr.size() + 1);
        for (int i = 1; i < (int) tree.size(); i++) {
            tree[i] += arr[i - 1];
            int parent = i + LSB(i);
            if (parent < (int) tree.size())
                tree[parent] += tree[i];
        }
    }

    void add(int i, ll v) {
        while (i < (int) tree.size()) {
            tree[i] += v;
            i += LSB(i);
        }
    }

    ll PrefixSum(int i) {
        ll sum = 0;
        while (i > 0) {
            sum += tree[i];
            i -= LSB(i);
        }
        return sum;
    }

    ll SumRange(int left, int right) {
        return PrefixSum(right) - PrefixSum(left - 1);
    }

    int LSB(int i) {
        return i & -i;
    }

    void SetPoint(int i, ll v) {
        add(i, v - SumRange(i, i));
    }
};


template<typename T>
struct BIT {
    int n;
    vector <T> a;

    BIT(int n) : n(n + 1), a(n + 1) {}

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i)a[i - 1] += v;
    }

    T sum(int x) {
        T ret = 0;
        for (int i = x + 1; i > 0; i -= i & -i)ret += a[i - 1];
        return ret;
    }
};
```

```cpp
//timus 1470 3d fenwick tree

//from (x0, y0, z0) to (x, y, z)
// value1 = sum(x,y,z)- sum(x0-1,y,z)  - sum(x,y0-1,z) + sum(x0-1,y0-1,z)
// value2 = sum(x,y,z0-1) - sum(x0-1,y,z0-1) - sum(x,y0-1,z0-1)  +
sum(x0-1,y0-1,z0-1)
// final answer = value1 - value2
// Time complexity of code is 8 (logn)^3

// 1> assume it as a cube.
// 2> value 1 gives answer for upper square layer of cube, but it include sum upto
0 in z-axis.
// You only want upto z0, so you have subtract that extra part(from z0 to 0, and
that is value2).
#include <bits/stdc++.h>
#define ll long long
#define ii pair<int,int>

using namespace std;

const int N=130;
ll tree[N][N][N];
void add(int x,int y,int z,ll value)
{
    while(x<N)
    {
        int y1 = y;
        while(y1<N)
        {
            int z1 = z;
            while(z1<N)
            {
                tree[x][y1][z1]+=value;
                z1+= (z1&-z1);
            }
            y1+= (y1&-y1);

        }
        x+= (x&-x);
    }
}
ll sum(int x,int y,int z)
{
    ll ret =0;
    while(x)
    {
        int y1 = y;
        while(y1)
        {
            int z1 = z;
            while(z1)
            {
                ret+=tree[x][y1][z1];
                z1-= (z1&-z1);
            }
            y1-= (y1&-y1);

        }
        x-= (x&-x);
    }
    return ret;
}


ll query(int x1,int y1,int z1,int x2,int y2,int z2)
{
    // from z = 0 to z2  (z1<=z2)
    ll value1 = sum(x2,y2,z2)-sum(x2,y1-1,z2)-sum(x1-1,y2,z2) + sum(x1-1,y1-1,z2);
    // from z = 0 to z1-1
```

```
    ll value2 = sum(x2,y2,z1-1)-sum(x2,y1-1,z1-1)-sum(x1-1,y2,z1-1) +
sum(x1-1,y1-1,z1-1);
    return value1 - value2;
}
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
//    freopen("input.txt","r",stdin);
    //freopen("output.txt","w",stdout);
    int n;
    cin>>n;
    int m;
    while(cin>>m&&m!=3)
    {
        if(m==1)
        {
            int x,y,z,v;
            cin>>x>>y>>z>>v;
            add(x+1,y+1,z+1,v);
        }
        else
        {
            int x1,y1,z1,x2,y2,z2;
            cin>>x1>>y1>>z1>>x2>>y2>>z2;
            cout<<query(x1+1,y1+1,z1+1,x2+1,y2+1,z2+1)<<endl;
        }
    }
}
```

```
/*
 *  count number of integers from set that make (a[i] xor num) >=l
 */
struct bTrie {
    struct node {
        int nxt[2] = {-1, -1};
        int cnt = 0;
    };
    vector<node> t;

    bTrie() : t(1) {}

    void update(int num, int val) {//val =1 add,-1 remove
        int cur = 0;
        for (int i = 30; i > -1; i--) {
            int bt = (num >> i) & 1;
            if (t[cur].nxt[bt] == -1)t[cur].nxt[bt] = t.size(), t.emplace_back();
            cur = t[cur].nxt[bt];
            t[cur].cnt += val;
        }
    }

    int query(int num, int l) {
        int cur = 0, ans = 0;
        for (int i = 30; i > -1 && cur != -1; i--) {
            int btP = (num >> i) & 1, btL = (l >> i) & 1;
            if (btL) {
                cur = t[cur].nxt[!btP];
            } else {
                ans += t[cur].nxt[!btP] == -1 ? 0 : t[t[cur].nxt[!btP]].cnt;//
hy5lone anwr
                cur = t[cur].nxt[btP];
            }
        }
        return ans + (cur != -1 ? t[cur].cnt : 0);
    }

};
```

```cpp
/*
 * "? x" — you are given integer x and need to compute the value ,
 * i.e. the maximum value of bitwise exclusive OR (also know as XOR)
 * of integer x and some integer y from the multiset A.
 */
#include <bits/stdc++.h>

#define ll long long
#define ii pair<int,int>
using namespace std;


struct trie {
    struct node {
        int nxt[2] = {-1, -1};
        int cnt = 0;
    };
    vector<node> t;

    trie() {
        t.emplace_back();
    }

    void add(int x) {
        int cur = 0;
        t[cur].cnt++;
        for (int i = 31; i > -1; i--) {
            int bt = (x >> i) & 1;
            if (t[cur].nxt[bt] == -1)t[cur].nxt[bt] = t.size(), t.emplace_back();
            cur = t[cur].nxt[bt];
            t[cur].cnt++;
        }
    }

    void remove(int x) {
        int cur = 0;
        t[cur].cnt--;
        for (int i = 31; i > -1; i--) {
            int bt = (x >> i) & 1;
            cur = t[cur].nxt[bt];
            t[cur].cnt--;
        }
    }

    int query(int x) {
        int ans = 0, cur = 0;
        for (int i = 31; i > -1; i--) {
            int bt = (x >> i) & 1;
            if (t[cur].nxt[!bt] != -1 && t[t[cur].nxt[!bt]].cnt) {
                cur = t[cur].nxt[!bt];
                ans |= (1 << i);
            } else if (t[cur].nxt[bt] != -1) {
                cur = t[cur].nxt[bt];
            }
        }
        return ans;
    }
};

int main() {
    //    freopen("input.in", "r", stdin);
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int q;
    cin >> q;
    trie t;
    t.add(0);
    while (q--) {
        char op;
```

```
        int x;
        cin >> op >> x;
        if (op == '+')t.add(x);
        else if (op == '-')t.remove(x);
        else cout << t.query(x) << endl;
    }

}
```

```
struct DSU
{
    vector<ii>parent;
    vector<int>GroupSize,bipartite;
    DSU(int sz)
    {
        parent.resize(sz);
        GroupSize.resize(sz);
        bipartite.resize(sz);
        for(int i = 0; i < sz; i++)
        {
            parent[i].first = i;
            parent[i].second=0;
            GroupSize[i] = 1;
            bipartite[i]=1;
        }
    }
    ii FindLeader(int i)
    {
        if(parent[i].first != i)
        {
            int parity = parent[i].second;
            parent[i] = FindLeader(parent[i].first);
            parent[i].second ^= parity;
        }
        return parent[i];
    }

    bool SameGroup(int x, int y)
    {
        ii leader1 = FindLeader(x);
        ii leader2 = FindLeader(y);

        return leader1.first == leader2.first;
    }

    void MergeGroups(int n1, int n2)
    {
        ii pa = FindLeader(n1);
        int a = pa.first;
        int x = pa.second;

        ii pb = FindLeader(n2);
        int b =pb.first;
        int y = pb.second;

        if(a==b)
        {
            if(x==y)
            {
                bipartite[a]=false;
            }
                return ;
        }
        else
        {
            if(GroupSize[a] < GroupSize[b])
                swap(a,b);
            //a is always bigger
            parent[b] =  {a,x^y^1};
            bipartite[a] &= bipartite[b];
            GroupSize[a] += GroupSize[b];
        }
    }

    int GetSize(int x)
    {
        ii leader = FindLeader(x);
        return GroupSize[leader.first];
```

```
    }

    bool IsBipartite(int v)
    {
        return bipartite[FindLeader(v).first];
    }
};
```

```
struct BIT{
    vector<long long> m,c;
    BIT(int n){
        m.resize(n);
        c.resize(n);
    }

    void add(int i,long long dm,long long dc){
        for(i++;i<=m.size();i+=(i&-i))
            m[i-1]+=dm , c[i-1]+=dc;

    }
    long long get(int i){
        long long res=0;
        int i2=i;
        for(i++;i;i-=(i&-i))
            res+=m[i-1]*i2 + c[i-1];
        return res;
    }
    void addRange(int st,int en,long long val){
        add(st,val,-val*(st-1));
        add(en+1,-val,val*en);
    }
};
```

```cpp
/*
 *  count number of integers from set that make a[i] xor num < l
 */
#include <bits/stdc++.h>

#define ll long long
#define ii pair<int,int>
using namespace std;


struct bTrie {
    struct node {
        int nxt[2] = {-1, -1};
        int cnt = 0;
    };
    vector<node> t;

    bTrie() : t(1) {}

    void update(int num, int val) {
        int cur = 0;
        for (int i = 30; i > -1; i--) {
            int bt = (num >> i) & 1;
            if (t[cur].nxt[bt] == -1)t[cur].nxt[bt] = t.size(), t.emplace_back();
            cur = t[cur].nxt[bt];
            t[cur].cnt += val;
        }
    }
    //count number of integers that make a^num<l
    int query(int num, int l) {
        int cur = 0, ans = 0;
        for (int i = 30; i > -1 && cur != -1; i--) {
            int btP = (num >> i) & 1, btL = (l >> i) & 1;
            if (btL) {
                ans += t[cur].nxt[btP] == -1 ? 0 : t[t[cur].nxt[btP]].cnt;
                cur = t[cur].nxt[!btP];
            } else {
                cur = t[cur].nxt[btP];
            }
        }
        return ans;
    }

};

int main() {
//    freopen("input.in", "r", stdin);
//    freopen("output.txt", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int q;
    cin >> q;
    bTrie t;
    while (q--) {
        int typ, x, y;
        cin >> typ >> x;
        if (typ <= 2)t.update(x, typ == 1 ? 1 : -1);
        else {
            cin >> y;
            cout << t.query(x, y) << endl;
        }
    }

}
```

Data Structure

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long
#define ii pair<ll,ll>
#define vi vector<int>
const int N = 1e5 + 10, MOD = 1000000007LL;
int n, q, arr[N], block_size = 325;
ll mp[N * 30];
int k;

struct Query {
    int l, r, idx;

    bool operator<(const Query &other) const {
        return make_pair(l / block_size, r) < make_pair(other.l / block_size,
other.r);
    }
};


ll sum = 0;

void add(int idx) {
    int v = arr[idx];
    sum += mp[k ^ v];
    mp[v]++;
}

void remove(int idx) {
    int v = arr[idx];
    mp[v]--;
    sum -= mp[k ^ v];
}


vector<ll> mo_s(vector<Query> &queries) {
    vector<ll> ans(queries.size());
    block_size = sqrt(n) + 1;
    sort(queries.begin(), queries.end());
    int curL = 0, curR = -1;
    for (auto &qe:queries) {
        while (curL > qe.l) add(--curL);
        while (curR < qe.r)add(++curR);
        while (curL < qe.l) remove(curL++);
        while (curR > qe.r)remove(curR--);
        ans[qe.idx] = sum;
    }
    return ans;
}

int main() {
//    freopen("input.in", "r", stdin);
//    freopen("output.txt", "w", stdout);
    scanf("%d%d%d", &n, &q, &k);
    for (int i = 1; i <= n; i++) scanf("%d", arr + i), arr[i] ^= arr[i - 1];


    vector<Query> queries(q);
    for (int i = 0; i < q; i++) {
        int l, r;
        scanf("%d%d", &l, &r);
        queries[i].idx = i, queries[i].l = l - 1, queries[i].r = r;
    }

    vector<ll> ans = mo_s(queries);
    for (auto a:ans) {
        printf("%lld\n", a);
```

```
        }
    }
```

```
struct modifedQueue {
    stack <pair<ll, ll>> s1, s2;

    void push(ll x) {
        ll newMin = s1.empty() ? x : max(x, s1.top().second);
        s1.push({x, newMin});
    }

    ll findMax() {
        ll mx;
        if (s1.empty() || s2.empty())
            mx = s1.empty() ? s2.top().second : s1.top().second;
        else
            mx = max(s1.top().second, s2.top().second);
        return mx;
    }

    void pop() {
        if (s2.empty()) {
            while (!s1.empty()) {
                pair <ll, ll> p = s1.top();
                s1.pop();
                ll newMin = s2.empty() ? p.first : max(s2.top().second, p.first);
                s2.push({p.first, newMin});
            }
        }
        s2.pop();
    }

    bool isEmpty() {
        return (s1.empty() && s2.empty());
    }

    int size() {
        return (s1.size() + s2.size());
    }
};


struct monoQueue {
    deque <pair<int, int>> q;

    void push(int val, int idx) {
        while (q.size() && q.back().first >= val)q.pop_back();
        q.push_back({val, idx});
    }

    void pop(int idx) {
        if (q.size() && q.front().second == idx)q.pop_front();
    }

    int getMin() {
        return q.front().first;
    }
};
```

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 40005;
const int MAXM = 100005;
const int LN = 19;

int N, M, K, cur, A[MAXN], LVL[MAXN], DP[LN][MAXN];
int BL[MAXN << 1], ID[MAXN << 1], VAL[MAXN], ANS[MAXM];
int d[MAXN], l[MAXN], r[MAXN];
bool VIS[MAXN];
vector < int > adjList[MAXN];

struct query{
    int id, l, r, lc;
    bool operator < (const query& rhs){
        return (BL[l] == BL[rhs.l]) ? (r < rhs.r) : (BL[l] < BL[rhs.l]);
    }
}Q[MAXM];

// Set up Stuff
void dfs(int u, int par){
    l[u] = ++cur;
    ID[cur] = u;
    for (int i = 1; i < LN; i++) DP[i][u] = DP[i - 1][DP[i - 1][u]];
    for (int i = 0; i < adjList[u].size(); i++){
        int v = adjList[u][i];
        if (v == par) continue;
        LVL[v] = LVL[u] + 1;
        DP[0][v] = u;
        dfs(v, u);
    }
    r[u] = ++cur; ID[cur] = u;
}

// Function returns lca of (u) and (v)
inline int lca(int u, int v){
    if (LVL[u] > LVL[v]) swap(u, v);
    for (int i = LN - 1; i >= 0; i--)
        if (LVL[v] - (1 << i) >= LVL[u]) v = DP[i][v];
    if (u == v) return u;
    for (int i = LN - 1; i >= 0; i--){
        if (DP[i][u] != DP[i][v]){
            u = DP[i][u];
            v = DP[i][v];
        }
    }
    return DP[0][u];
}

inline void check(int x, int& res){
    // If (x) occurs twice, then don't consider it's value
    if ( (VIS[x]) and (--VAL[A[x]] == 0) ) res--;
    else if ( (!VIS[x]) and (VAL[A[x]]++ == 0) ) res++;
    VIS[x] ^= 1;
}

void compute(){

    // Perform standard Mo's Algorithm
    int curL = Q[0].l, curR = Q[0].l - 1, res = 0;

    for (int i = 0; i < M; i++){

        while (curL < Q[i].l) check(ID[curL++], res);
        while (curL > Q[i].l) check(ID[--curL], res);
        while (curR < Q[i].r) check(ID[++curR], res);
        while (curR > Q[i].r) check(ID[curR--], res);
```

```
        int u = ID[curL], v = ID[curR];

        // Case 2
        if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);

        ANS[Q[i].id] = res;

        if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);
    }

    for (int i = 0; i < M; i++) printf("%d\n", ANS[i]);
}

int main(){

    int u, v, x;

    while (scanf("%d %d", &N, &M) != EOF){

        // Cleanup
        cur = 0;
        memset(VIS, 0, sizeof(VIS));
        memset(VAL, 0, sizeof(VAL));
        for (int i = 1; i <= N; i++) adjList[i].clear();

        // Inputting Values
        for (int i = 1; i <= N; i++) scanf("%d", &A[i]);
        memcpy(d + 1, A + 1, sizeof(int) * N);

        // Compressing Coordinates
        sort(d + 1, d + N + 1);
        K = unique(d + 1, d + N + 1) - d - 1;
        for (int i = 1; i <= N; i++) A[i] = lower_bound(d + 1, d + K + 1, A[i]) -
d;

        // Inputting Tree
        for (int i = 1; i < N; i++){
            scanf("%d %d", &u, &v);
            adjList[u].push_back(v);
            adjList[v].push_back(u);
        }

        // Preprocess
        DP[0][1] = 1;
        dfs(1, -1);
        int size = sqrt(cur);

        for (int i = 1; i <= cur; i++) BL[i] = (i - 1) / size + 1;

        for (int i = 0; i < M; i++){
            scanf("%d %d", &u, &v);
            Q[i].lc = lca(u, v);
            if (l[u] > l[v]) swap(u, v);
            if (Q[i].lc == u) Q[i].l = l[u], Q[i].r = l[v];
            else Q[i].l = r[u], Q[i].r = l[v];
            Q[i].id = i;
        }

        sort(Q, Q + M);
        compute();
    }
}
```

```cpp
// my code
#include <bits/stdc++.h>

using namespace std;

#define ll long long
#define ii pair<ll,ll>
#define vi vector<int>
const int N = 1e5 + 10, MOD = 1000000007LL;
int n, q, arr[N], block_size = 325, t_in[N], t_out[N], tour[2 * N], ln = 19,
timer, up[N][22];
vector<int> adj[N];
int occurence[N], colors[N];

struct Query {
    int l, r, idx, lc, color;

    bool operator<(const Query &other) const {
        return make_pair(l / block_size, r) < make_pair(other.l / block_size,
other.r);
    }
};


void init() {
    timer = 0;
    for (int i = 0; i <= n; i++)adj[i].clear(), colors[i] = 0, occurence[i] = 0;
}

void dfs(int node, int p) {
    t_in[node] = timer;
    tour[timer++] = node;
    up[node][0] = p;
    for (int i = 1; i <= ln; i++)up[node][i] = up[up[node][i - 1]][i - 1];
    for (auto child:adj[node]) {
        if (child != p)
            dfs(child, node);
    }
    t_out[node] = timer;
    tour[timer++] = node;
}

bool is_ancestor(int u, int v) {
    return t_in[u] <= t_in[v] && t_out[v] <= t_out[u];
}

int lca(int u, int v) {
    if (is_ancestor(u, v))return u;
    if (is_ancestor(v, u))return v;
    for (int i = ln; i >= 0; i--) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}


void update(int idx) {
    int node = tour[idx];
    occurence[node] ^= 1;
    if (occurence[node]) colors[arr[node]]++;
    else colors[arr[node]]--;
}


vector<bool> mo_s(vector<Query> &queries) {
    vector<bool> ans(queries.size());
    block_size = sqrt(n * 2) + 1;
    sort(queries.begin(), queries.end());
```

```
    int curL = 0, curR = -1;
//    for (int i = 0; i < 2 * n;i++)cout << tour[i]+1  << " ";
//    cout << endl;
    for (auto &qe:queries) {
//        cout << qe.idx << "  " << qe.l << "  " << qe.r << endl;
        while (curL > qe.l) update(--curL);
        while (curR < qe.r)update(++curR);
        while (curL < qe.l) update(curL++);
        while (curR > qe.r)update(curR--);

        if (qe.lc != qe.l) colors[arr[qe.lc]]++;

        ans[qe.idx] = colors[qe.color] != 0;

        if (qe.lc != qe.l) colors[arr[qe.lc]]--;

    }
    return ans;
}

int main() {
    //   freopen("input.in", "r", stdin);
//    freopen("output.txt", "w", stdout);
    while (~scanf("%d%d", &n, &q)) {
        init();
        for (int i = 0; i < n; i++) scanf("%d", arr + i);
        for (int i = 0; i < n - 1; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            adj[u - 1].push_back(v - 1);
            adj[v - 1].push_back(u - 1);
        }

        dfs(0, 0);

        vector<Query> queries(q);
        for (int i = 0; i < q; i++) {
            int l, r, c;
            scanf("%d%d%d", &l, &r, &c);
            l--;
            r--;
            if (t_in[l] > t_in[r])swap(l, r);
            int lc = lca(l, r);
            if (lc == l) queries[i].l = t_in[l], queries[i].r = t_in[r];
            else queries[i].l = t_out[l], queries[i].r = t_in[r];
            queries[i].idx = i, queries[i].color = c, queries[i].lc = lc;
        }

        vector<bool> ans = mo_s(queries);
        for (auto a:ans) {
            puts(a ? "Find" : "NotFind");
        }
        puts("");
    }
    return 0;
}
```

animeshf's blog

Mo's Algorithm on Trees [Tutorial]

By animeshf, history, 5 years ago,

Introduction

Mo's Algorithm has become pretty popular in the past few years and is now considered as a pretty standard technique in the world of Competitive Programming. This blog will describe a method to generalize Mo's algorithm to maintain information about paths between nodes in a tree.

Prerequisites

Mo's Algorithm — If you do not know this yet, read this amazing article before continuing with this blog.

Preorder Traversal or DFS Order of the Tree.

Problem 1 — Handling Subtree Queries

Consider the following problem. You will be given a rooted Tree T of N nodes where each node is associated with a value $A[node]$. You need to handle Q queries, each comprising one integer u. In each query you must report the number of distinct values in the subtree rooted at u. In other words, if you store all the values in the subtree rooted at u in a set, what would be the size of this set?

Constraints

$1 \le N, Q \le 10^5$

$1 \le A[node] \le 10^9$

Solution(s)

Seems pretty simple, doesn't it? One easy way to solve this is to flatten the tree into an array by doing a Preorder traversal and then implement Mo's Algorithm. Maintain a lookup table which maintains the frequency of each value in the current window. By maintaining this, the answer can be updated easily. The complexity of this algorithm would be $O(Q\sqrt{N})$

Note that you can also solve this in $O(N \log^2 N)$ by maintaining a set in each node and merging the smaller set into the larger ones.

Problem 2 — Handling Path Queries

Now let's modify Problem 1 a little. Instead of computing the number of distinct values in a subtree, compute the number of distinct values in the unique path from u to v. I recommend you to pause here and try solving the problem for a while. The constraints of this problem are the same as Problem 1.

The Issue

An important reason why Problem (1) worked beautifully was because the dfs-order traversal made it possible to represent any subtree as a contiguous range in an array. Thus the problem was reduced to "finding number of distinct values in a subarray $[L, R]$ of $A[]$. Note that it is not possible to do so for path queries, as nodes which are $O(N)$ distance apart in the

tree might be $O(1)$ distance apart in the flattened tree (represented by Array $A[]$). So doing a normal dfs-order would not work out.

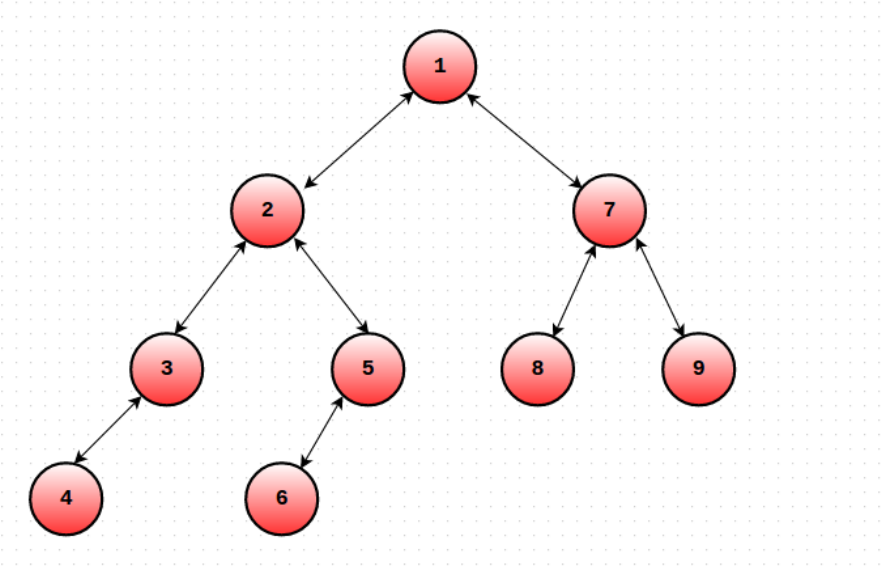
Observation(s)

Let a node u have k children. Let us number them as $v_1,v_2...v_k$. Let $S(u)$ denote the subtree rooted at u.

Let us assume that $dfs()$ will visit u's children in the order $v_1,v_2...v_k$. Let x be any node in $S(v_i)$ and y be any node in $S(v_j)$ and let $i < j$. Notice that $dfs(y)$ will be called only after $dfs(x)$ has been completed and $S(x)$ has been explored. Thus, before we call $dfs(y)$, we would have entered and exited $S(x)$. We will exploit this seemingly obvious property of $dfs()$ to modify our existing algorithm and try to represent each query as a contiguous range in a flattened array.

Modified DFS-Order

Let us modify the dfs order as follows. For each node u, maintain the Start and End time of $S(u)$. Let's call them $ST(u)$ and $EN(u)$. The only change you need to make is that you must increment the global timekeeping variable even when you finish traversing some subtree ($EN(u)$ = $++cur$). In short, we will maintain 2 values for each node u. One will denote the time when you entered $S(u)$ and the other would denote the time when you exited $S(u)$. Consider the tree in the picture. Given below are the $ST()$ and $EN()$ values of the nodes.



$ST(1) = 1$ $EN(1) = 18$

$ST(2) = 2$ $EN(2) = 11$

$ST(3) = 3$ $EN(3) = 6$

$ST(4) = 4$ $EN(4) = 5$

$ST(5) = 7$ $EN(5) = 10$

$ST(6) = 8$ $EN(6) = 9$

$ST(7) = 12$ $EN(7) = 17$

$ST(8) = 13$ $EN(8) = 14$

$ST(9) = 15$ $EN(9) = 16$

$A[] = \{1, 2, 3, 4, 4, 3, 5, 6, 6, 5, 2, 7, 8, 8, 9, 9, 7, 1\}$

The Algorithm

Now that we're equipped with the necessary weapons, let's understand how to process the queries.

Let a query be (u, v). We will try to map each query to a range in the flattened array. Let $ST(u) \leq ST(v)$ where $ST(u)$ denotes visit time of node u in T. Let $P = LCA(u, v)$ denote the lowest common ancestor of nodes u and v. There are 2 possible cases:

$Case\ 1: P = u$

In this case, our query range would be $[ST(u), ST(v)]$. Why will this work?

Consider any node x that does not lie in the (u, v) path.
Notice that x occurs twice or zero times in our specified query range.
Therefore, the nodes which occur exactly once in this range are precisely those that are on the (u, v) path! (Try to convince yourself of why this is true : It's all because of $dfs()$ properties.)

This forms the crux of our algorithm. While implementing Mo's, our add/remove function needs to check the number of times a particular node appears in a range. If it occurs twice (or zero times), then we don't take it's value into account! This can be easily implemented while moving the left and right pointers.

$Case\ 2: P \neq u$

In this case, our query range would be $[EN(u), ST(v)] + [ST(P), ST(P)]$.

The same logic as Case 1 applies here as well. The only difference is that we need to consider the value of P i.e the LCA separately, as it would not be counted in the query range.

This same problem is available on SPOJ.

If you aren't sure about some elements of this algorithm, take a look at this neat code.

Conclusion

We have effectively managed to reduce problem (2) to number of distinct values in a subarray by doing some careful bookkeeping. Now we can solve the problem in $O(Q\sqrt{N})$. This modified DFS order works brilliantly to handle any type path queries and works well with Mo's algo. We can use a similar approach to solve many types of path query problems.

For example, consider the question of finding number of inversions in a (u, v) path in a Tree T, where each node has a value associated with it. This can now be solved in $O(Q\sqrt{N}\log N)$ by using the above technique and maintaining a BIT or Segment Tree.

This is my first blog and I apologize for any mistakes that I may have made. I would like to thank **sidhant** for helping me understand this technique.

Sample Problems

1) Count on a Tree II
2) Frank Sinatra — Problem F
3) Vasya and Little Bear

Thanks a lot for reading!

Original Post
Related Blog

+407 ☆ animeshf 5 years ago 149

Comments (149)

Write comment?

5 years ago, # | ☆ ← Rev. 2 +8

In this case, our query range would be [EN(u), ST(v)] + [ST(P), ST(P)]

```cpp
// C++ program to demonstrate the
// ordered set in GNU C++
#include <iostream>
using namespace std;

// Header files, namespaces,
// macros as defined above
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>,
rb_tree_tag,tree_order_statistics_node_update>

// Driver program to test above functions
int main()
{
    // Ordered set declared with name o_set
    ordered_set o_set;

    // insert function to insert in
    // ordered set same as SET STL
    o_set.insert(5);
    o_set.insert(1);
    o_set.insert(2);

    // Finding the second smallest element
    // in the set using * because
    //  find_by_order returns an iterator
    cout << *(o_set.find_by_order(1))
         << endl;

    // Finding the number of elements
    // strictly less than k=4
    cout << o_set.order_of_key(4)
         << endl;

    // Finding the count of elements less
    // than or equal to 4 i.e. strictly less
    // than 5 if integers are present
    cout << o_set.order_of_key(5)
         << endl;

    // Deleting 2 from the set if it exists
    if (o_set.find(2) != o_set.end())
        o_set.erase(o_set.find(2));

    // Now after deleting 2 from the set
    // Finding the second smallest element in the set
    cout << *(o_set.find_by_order(1))
         << endl;

    // Finding the number of
    // elements strictly less than k=4
    cout << o_set.order_of_key(4)
         << endl;

    return 0;
}
```

```cpp
#include<bits/stdc++.h>
#define ll long long

using namespace std;

struct node;
extern node* empty;

struct node
{
    int sum;
    node *lf, *rt;
    node():sum(0), lf(this), rt(this){}
    node(int val):sum(val), lf(empty), rt(empty){}
};

node* empty = new node();

void add(int ind, int val, node*& nd, int ns = -1e9, int ne = 1e9)
{
    if(ind < ns or ind > ne) return;
    if(nd == empty) nd = new node(0);
    if(ns == ne)
    {
        nd->sum += val;
        return;
    }
    int mid = ns + (ne - ns) / 2;
    add(ind, val, nd->lf, ns, mid);
    add(ind, val, nd->rt, mid + 1, ne);
    nd->sum = nd->lf->sum + nd->rt->sum;
}

int get(int qe, int qs, node*& nd, int ns = -1e9, int ne = 1e9)
{
    if(qe < ns or qs > ne) return 0;
    if(ns >= qs and ne <= qe) return nd->sum;
    int mid = ns + (ne - ns) / 2;
    return get(qe, qs, nd->lf, ns, mid) +
           get(qe, qs, nd->rt, mid + 1, ne);
}

int find(int val, node*& nd, int ns = -1e9, int ne = 1e9)
{
    if(ns == ne) return ns;
    int szLf = nd->lf->sum;
    int mid = ns + (ne - ns) / 2;
    if(val < szLf) return find(val, nd->lf, ns, mid);
    return find(val - szLf, nd->rt, mid + 1, ne);
}

struct mySet{
    node* root;
    mySet():root(empty){}

    void insert(int x){
        add(x, 1, root);
    }
    int count(int x){
        return get(x, x, root);
    }
    int operator[](int i){
        return find(i, root);
    }
    int size(){
        return root->sum;
    }
    void erase(int val){
        add(val, -count(val), root);
```

```
    }
    // delete cnt items of value val
    void erase(int val, int cnt){
        add(val, -min(cnt,count(val)), root);
    }
    int lowerBound(int val)
    {
        return get(val - 1, -1e9, root);
    }
};

int main()
{
    mySet s;
    int q, x;
    scanf("%d", &q);
    char c;
    while(q--)
    {
        scanf(" %c %d", &c, &x);
        switch(c)
        {
            case 'I': if(!s.count(x)) s.insert(x); break;
            case 'D': s.erase(x); break;
            case 'K': if(x > s.size()) puts("invalid"); else printf("%d\n", s[--
x]); break;
            case 'C': printf("%d\n", s.lowerBound(x)); break;
        }
    }
    return 0;
}
```

```cpp
struct DSU
{
    vector<int>parent,GroupSize;
    DSU(int sz)
    {
        parent.resize(sz);
        GroupSize.resize(sz);
        for(int i = 0; i < sz; i++)
        {
            parent[i] = i;
            GroupSize[i] = 1;
        }
    }
    int FindLeader(int i)
    {
        if(parent[i] == i)
            return i;

        return parent[i] = FindLeader(parent[i]);
    }

    bool SameGroup(int x, int y)
    {
        int leader1 = FindLeader(x);
        int leader2 = FindLeader(y);

        return leader1 == leader2;
    }

    void MergeGroups(int x, int y)
    {
        int leader1 = FindLeader(x);
        int leader2 = FindLeader(y);

        if(leader1 == leader2)
            return;

        if(GroupSize[leader1] < GroupSize[leader2])
            swap(leader1,leader2);

        //leader1 is always bigger
        parent[leader2] =  leader1;
        GroupSize[leader1] += GroupSize[leader2];

    }

    int GetSize(int x)
    {
        int leader = FindLeader(x);
        return GroupSize[leader];
        }
};
```

```cpp
#include <bits/stdc++.h>

using namespace std;
#define ll long long
const int N = 3e5 + 4;

struct BIT {
    vector<ll> tree;

    BIT(int n) {
        tree.resize(n + 1);
    }

    BIT(vector<ll> &arr) {
        tree.resize(arr.size() + 1);
        for (int i = 1; i < (int) tree.size(); i++) {
            tree[i] += arr[i - 1];
            int parent = i + LSB(i);
            if (parent < (int) tree.size())
                tree[parent] += tree[i];
        }
    }

    void add(int i, ll v) {
        while (i < (int) tree.size()) {
            tree[i] += v;
            i += LSB(i);
        }
    }

    ll PrefixSum(int i) {
        ll sum = 0;
        i = min(i, (int) tree.size() - 1);
        while (i > 0) {
            sum += tree[i];
            i -= LSB(i);
        }
        return sum;
    }

    ll SumRange(int left, int right) {
        return PrefixSum(right) - PrefixSum(left - 1);
    }

    static int LSB(int i) {
        return i & -i;
    }

    void SetPoint(int i, ll v) {
        add(i, v - SumRange(i, i));
    }
};

int main() {
//    freopen("input.in", "r", stdin);
//    freopen("output.txt", "w", stdout);
    int n, m;
    while (scanf("%d%d", &n, &m) != -1) {
        vector<int> arr(n), pos(n + 1);
        for (int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
            pos[arr[i]] = i;
        }

        int root = 2000;
        int count = (n + root - 1) / root;
        vector<vector<ll>> buckets(count);
        for (int i = 0; i < n; i++)
            buckets[i / root].emplace_back(arr[i]);
```

```
        for (int i = 0; i < count; i++)sort(buckets[i].begin(), buckets[i].end());

        BIT bit = BIT(n + 2);
        ll total = 0;//n*log(n)
        for (int i = n - 1; i > -1; i--) {
            total += bit.PrefixSum(arr[i]);
            bit.add(arr[i], 1);
        }
        //5*sqrt(n)*log(sqrt)*m
        while (m--) {
            int x, idx;
            cin >> x;
            printf("%lld\n", total);
            idx = pos[x];
            for (int i = 0; i < idx;) {
                if (i % root == 0 && i + root - 1 < idx) {
                    total -=
                            buckets[i / root].end() -
                            upper_bound(buckets[i / root].begin(), buckets[i /
root].end(), x);
                    i += root;
                } else {
                    total -= (arr[i] > x);
                    i++;
                }
            }
            for (int i = idx + 1; i < n;) {
                if (i % root == 0 && i + root - 1 < n) {
                    total -=
                            upper_bound(buckets[i / root].begin(), buckets[i /
root].end(), x) -
                            buckets[i / root].begin();
                    i += root;
                } else {
                    total -= (arr[i] < x && arr[i] != -1);
                    i++;
                }
            }

            int b;
            b= idx / root;
            buckets[b].erase(find(buckets[b].begin(), buckets[b].end(), x));
            arr[idx] = -1;
        }
    }
}
```

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

unordered_map<int, int, custom_hash> mp;
```

Graph Theory

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;
const int N = 26 * 4;
const ll mod = 1e9 + 7;
vector<int> adj[N], adj2[N], gAdj[N], gAdj2[N];

/*
 *
 * if AB = 2
 * A =true , B =true
 *
 * if AB =0
 * A = false,B =false
 *
 * if AB = 1
 * A ^ B
 *
*/

int get_idx(int u, int player) {
    return (2 * u) + (52 * (player == 2));
}

void add_or(int a, int b) {
    adj[a ^ 1].emplace_back(b);
    adj[b ^ 1].emplace_back(a);

    gAdj[b].emplace_back(a ^ 1);
    gAdj[a].emplace_back(b ^ 1);
}

void add_xor(int a, int b) {
    add_or(a, b);
    add_or(a ^ 1, b ^ 1);
}

int vis[N], cmp[N];
vector<int> order;

void dfs1(int node) {
    vis[node] = true;
    for (auto child:adj[node])if (!vis[child])dfs1(child);
    order.push_back(node);
}

void dfs2(int node, int cmpId) {
    cmp[node] = cmpId;
    for (auto child:gAdj[node])if (cmp[child] == -1)dfs2(child, cmpId);
}

bool solve_2SAT() {
    memset(vis, 0, sizeof vis);
    memset(cmp, -1, sizeof cmp);
    order.clear();
    for (int i = 0; i < N; i++) if (!vis[i])dfs1(i);

    for (int i = 0, cmpId = 0; i < N; i++) {
        int v = order[N - i - 1];
        if (cmp[v] == -1)dfs2(v, cmpId++);
    }
    for (int i = 0; i < N; i += 2) {
        if (cmp[i] == cmp[i + 1]) return false;
    }
    return true;
}

/*
```

```
 * a1^a2
 */
void test() {
    int m, p, score;
    cin >> m;
    for (int i = 0; i < m; i++) {
        string tmp;
        cin >> tmp >> p >> score;
        int u = get_idx(tmp[0] - 'A', p), v = get_idx(tmp[1] - 'A', p);
        if (score == 0) {
            add_or(u ^ 1, u ^ 1);//u is false
            add_or(v ^ 1, v ^ 1);//v is false
        } else if (score == 1) {
            add_xor(u, v);//a^b
        } else {
            add_or(u, u);//u is true
            add_or(v, v);//v is true
        }
    }
    for (int i = 0; i < N; i++)adj2[i] = adj[i], gAdj2[i] = gAdj[i];
    int ans = 0;
    for (int i = 0; i < 26; i++) {
        for (int j = i + 1; j < 26; j++) {
            for (int k = j + 1; k < 26; k++) {
                //revert graph
                for (int w = 0; w < N; w++) adj[w] = adj2[w], gAdj[w] = gAdj2[w];
                for (int w = 0; w < 26; w++)
                    if (w != i && w != j && w != k)
                        add_xor(get_idx(w, 1), get_idx(w, 2));

                //set i j k to false
                int a1 = get_idx(i, 1), a2 = get_idx(i, 2);
                int b1 = get_idx(j, 1), b2 = get_idx(j, 2);
                int c1 = get_idx(k, 1), c2 = get_idx(k, 2);
                add_or(a1 ^ 1, a1 ^ 1);
                add_or(a2 ^ 1, a2 ^ 1);
                add_or(b1 ^ 1, b1 ^ 1);
                add_or(b2 ^ 1, b2 ^ 1);
                add_or(c1 ^ 1, c1 ^ 1);
                add_or(c2 ^ 1, c2 ^ 1);
                ans += solve_2SAT();
            }
        }
    }
    printf("%d\n", ans);
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    //    freopen("input.in", "r", stdin);
    int t = 1;
//    cin >> t;
//    scanf("%d", &t);
    for (int cas = 1; cas <= t; cas++) {
        //        printf("Case %d: ", cas);
        test();
    }
}
```

Definitions

The **CONJUNCTION** of two propositions p and q is written $p \wedge q$. It denotes "p and q".

The **DISJUNCTION** of two propositions p and q is written $p \vee q$. It denotes "p or q".

The **CONDITIONAL PROPOSITION** is written $p \rightarrow q$, is the proposition "if p then q".
p is the **HYPOTHESIS** or **PREMISE** and q is the **CONCLUSION** or **CONSEQUENCE**

p and q are **EQUIVALENT** and write $p \equiv q$ if they always have the same truth value.

The **BICONDITIONAL PROPOSITION** $p \leftrightarrow q$ is "p if, and only if, q". It is true when p and q have the same truth values and is false otherwise (exactly the same)

Initial	$p \rightarrow q$	$\equiv$	$\neg p \vee q$
Contrapositive	$\neg q \rightarrow \neg p$	$\equiv$	$\neg p \vee q$
Converse	$q \rightarrow p$	$\equiv$	$\neg p \vee q$
Inverse	$\neg p \rightarrow \neg q$	$\equiv$	$\neg p \vee q$

Commutative laws:

$$p \wedge q \equiv q \wedge p; \qquad p \vee q \equiv q \vee p$$

Associative laws:

$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r); \quad (p \vee q) \vee r \equiv p \vee (q \vee r)$$

Distributive laws:

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$
$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

Identity laws:

$$p \wedge \mathbf{T} \equiv p; \qquad p \vee \mathbf{C} \equiv p$$

Negation laws:

$$p \wedge \neg p = \mathbf{C}; \qquad p \vee \neg p = \mathbf{T}$$

Double negation laws:

$$\neg(\neg p) \equiv p$$

Idempotent laws:

$$p \wedge p \equiv p; \qquad p \vee p \equiv p$$

Universal bound laws:

$$p \wedge \mathbf{C} \equiv \mathbf{C}; \qquad p \vee \mathbf{T} \equiv \mathbf{T}$$

De Morgan's laws:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q; \quad \neg(p \vee q) \equiv \neg p \wedge \neg q$$

Absorption laws:

$$p \wedge (p \vee q) \equiv p; \quad p \vee (p \wedge q) \equiv p$$

Negations of **T** and **C**:

$$\neg \mathbf{C} \equiv \mathbf{T}; \qquad \neg \mathbf{T} \equiv \mathbf{C}$$

We imagine the actions suggested by the quantifiers as being performed in the order in which the quantifiers occur. Reversing the order of the quantifiers can change the truth value of a statement **unless** the quantifiers are all universal or all existential. For example, "everyone loves someone" is translated into:

$\forall x \exists y\, P(x,y)$ where $P(x,y)$ is "x loves y."

However

$$\exists y \forall x\, P(x,y)$$

is "there is someone who is loved by everyone".

$\forall x \exists y\, P(x, y)$ – For all x, there exist y such that P is true
$\forall y \exists x\, P(x, y)$ – For all y, there exist x such that P is true
$\exists x \forall y\, P(x, y)$ – There exist x such that for all y, P is true
$\exists y \forall x\, P(x, y)$ – There exist y such that for all x, P is true

```cpp
const int OO = 1e9 + 9;
const int N = 5e6 + 9;
vector <pair<int, int>> adj[N];
ll parent[N], score[N];
int parentChar[N];

struct Node {
    int from, to, score, alpha;

    Node(int from, int to, int score, int alpha) : from(from), to(to),
score(score), alpha(alpha) {}

    bool operator<(const Node &rhs) const {
        if (score != rhs.score)return score < rhs.score;
        return alpha < rhs.alpha;
    }
};

void bfs(int src, int target, int n) {
    queue<int> q;
    q.push(src);
    for (int i = 0; i < n; i++) sort(adj[i].begin(), adj[i].end(), [](ii l, ii r)
{ return l.second < r.second; });

    fill(score, score + n, OO);
    fill(parent, parent + n, -1);

    score[src] = 0;
    int counter = 0;

    while (!q.empty()) {
        int sz = q.size();
        vector <Node> v;
        while (sz--) {
            int node = q.front();
            q.pop();
            for (auto[to, ch]:adj[node]) {
                if (score[to] == OO) {
                    Node nd(node, to, score[node], ch);
                    v.push_back(nd);
                }
            }
        }
        sort(v.begin(), v.end());
        pair<int, int> prev{-1, -1};
        for (auto &nd : v) {
            if (make_pair(nd.score, nd.alpha) != prev)counter++;
            if (score[nd.to] == OO) {
                score[nd.to] = counter;
                q.push(nd.to);
                parentChar[nd.to] = nd.alpha;
                parent[nd.to] = nd.from;
            }
            prev = {nd.score, nd.alpha};
        }
    }


    string ans;
    vector<int> path;
    while (parent[target] != -1) {
        path.push_back(target);
        ans += char(parentChar[target]);
        target = parent[target];
    }
    path.push_back(src);
    reverse(ans.begin(), ans.end());
    printf("%zu\n", ans.size());
```

```
    for (int i = path.size() - 1; i > -1; i--) printf("%d%c", path[i] + 1, " \n"[i
== 0]);
    printf("%s\n", ans.c_str());

}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

#define neig(a, u, e, v) for(int v, e = a.head[u] ; e and (v = a.to[e]) ; e =
a.nxt[e])

const int N = 5e4 + 5, M = 2e5 + 5;

struct Adj {
    int head[N], nxt[M], to[M], ne, n;
    void init(int n) {
        memset(head + 1, 0, n * sizeof(head[0]));
        ne = 2;
        this->n = n;
    }
    void addEdge(int f, int t) {
        nxt[ne] = head[f];
        to[ne] = t;
        head[f] = ne++;
    }
    void addBiEdge(int f, int t) {
        addEdge(f, t);
        addEdge(t, f);
    }
    int addNode() {
        head[++n] = 0;
        return n;
    }
} adj, comp;

int vis[N], dfsT[N], low[N], stk[N], isArticl[N], isBridge[M], isFirst, sz, vid,
        curT;

void extComp(int u, int v) {
    int id = comp.addNode();
    comp.addEdge(id, u);
    do {
        comp.addEdge(id, stk[--sz]);
    } while (stk[sz] != v);
}

void tarjanDfs(int u, int p) {
    low[u] = dfsT[u] = curT++;
    vis[u] = vid;
    stk[sz++] = u;
    neig(adj, u, e, v)
    {
        if (e == p)
            continue;
        if (vis[v] != vid) {
            tarjanDfs(v, e ^ 1);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfsT[u]) {
                if (u != isFirst || dfsT[v] > dfsT[u] + 1)
                    isArticl[u] = vid;
                extComp(u, v);
            }
            if (low[v] > dfsT[u])
                isBridge[e] = isBridge[e ^ 1] = vid;
        } else
            low[u] = min(low[u], dfsT[v]);
    }
}

void biConnected() {
    comp.init(0);
    ++vid;
    int timeBef;
    for (int i = 1; i <= adj.n; i++) {
```

```
            if (vis[i] != vid) {
                isFirst = i;
                timeBef = curT;
                sz = 0;
                tarjanDfs(i, -1);
                if (timeBef + 1 == curT) {
                    int id = comp.addNode();
                    comp.addEdge(id, i);
                }
            }
        }
    }

    int main() {
        //  freopen("input.txt", "r", stdin);
        int m, u, v, test = 1;
        while (scanf("%d", &m) && m) {
            printf("Case %d:", test++);
            int n = 0;
            adj.ne = 2;
            memset(adj.head, 0, sizeof(adj.head));
            for (int i = 0; i < m; ++i) {
                scanf("%d%d", &u, &v);
                n = max( { n, u, v });
                adj.addBiEdge(v, u);
            }
            adj.n = n;
            biConnected();
            if (comp.n == 1) {
                printf(" 2 %lld\n", n * (n - 1LL) / 2);
                continue;
            }

            int ndsCnt = 0;
            long long numSl = 1;
            for (int i = 1; i <= comp.n; i++) {
                int cnt = 0, comSz = 0;
                neig(comp,i,e,w)
                {
                    comSz++;
                    if (isArticl[w] == vid)
                        cnt++;
                }
                if (cnt == 1) {
                    ndsCnt++;
                    numSl *= (comSz - 1); // all nodes except articulation node
                }
            }
            printf(" %d %lld\n", ndsCnt, numSl);
        }
        return 0;
    }
```

```cpp
/*
 * returns the block-cut tree of graph G, such that each node in tree represents
either
 * a biconnected component or cut vertex of G.
 * A node representing a cut vertex is connected
 * to all nodes representing biconnected components that contain that cut vertex.
 */

//#pragma GCC optimize ("O3")
#include "bits/stdc++.h"
#include "ext/numeric"

using namespace std;
using namespace __gnu_cxx;

#define neig(adj, u, e, v) for(int e = adj.head[u], v ; e and (v = adj.to[e]) ; e
= adj.nxt[e])
#define rep2(i, s, e) for(int i=s;i<=(int)e;++i)
#define rep(i, n) for(int i=0;i<(int)n;++i)

const int N = 4e5 + 5, M = 2 * N;

struct Adj {
    int head[N], to[M], nxt[M], ne = 2, n;

    Adj() : n(0) {
        init();
    }

    void init() {
        memset(head + 1, 0, n * sizeof(head[0]));
        ne = 2;
    }

    void addEdge(int f, int t) {
        nxt[ne] = head[f];
        to[ne] = t;
        head[f] = ne++;
    }

    void addBiEdge(int f, int t) {
        addEdge(f, t);
        addEdge(t, f);
    }

    int addNode() {
        head[++n] = 0;
        return n;
    }
} adj, comps, blktree;


int vis[N], dfsT[N], low[N], stk[M], isArticl[N], isBridge[M], isFirst, sz, vid,
curT, cntFirst;
int roots[N], rootsCnt;

void extComp(int u, int v) {
    int id = comps.addNode();
    comps.addEdge(id, u);
    while (comps.to[comps.head[id]] != v)
        comps.addEdge(id, stk[--sz]);

}

void tarjanDfs(int u, int p) {
    low[u] = dfsT[u] = curT++;
    vis[u] = vid;
    stk[sz++] = u;
    neig(adj, u, e, v) {
```

```
        if (e == p) continue;
        if (vis[v] != vid) {
            // stk[sz++] = e;
            tarjanDfs(v, e ^ 1);
            low[u] = min(low[u], low[v]);
            if (u == isFirst) {
                if (cntFirst++) isArticl[u] = vid;
                extComp(u, v);
            } else if (low[v] >= dfsT[u]) {
                isArticl[u] = vid;
                extComp(u, v);
            }
            if (low[v] > dfsT[u]) isBridge[e] = isBridge[e ^ 1] = vid;
        } else {
            //    if (dfsT[u] > dfsT[v]) stk[sz++] = e;
            low[u] = min(low[u], dfsT[v]);
        }
    }
}

void biConnected() {
    // Isolated nodes should be handled seperately
    ++vid;
    comps.n = 0;
    sz = 0;
    rootsCnt = 0;
    for (int i = 1; i <= adj.n; i++) {
        if (vis[i] != vid) {
            if (!adj.head[i]) {
                int id = comps.addNode();
                comps.addEdge(id, i);
            }
            cntFirst = 0;
            isFirst = i;
            roots[rootsCnt++] = i;
            tarjanDfs(i, -1);
        }
    }
}

int id[N], cntArt, artIdx[N];

void buildBlockCut() {
    biConnected();
    cntArt = blktree.n = 0;
    blktree.init();
    rep2(i, 1, adj.n)if (isArticl[i] == vid)
            artIdx[id[i] = blktree.addNode()] = i, cntArt++;
    rep2(c, 1, comps.n) {
        int cmpId = blktree.addNode();
        neig(comps, c, e, u) {
            if (isArticl[u] == vid)
                blktree.addBiEdge(id[u], cmpId);
            else
                id[u] = cmpId;
        }
    }
}

void printTree() {
    rep2(i, 1, blktree.n)printf("%d\n", i);
    for (int i = 2; i < blktree.ne; i += 2) {
        printf("%d %d\n", blktree.to[i ^ 1], blktree.to[i]);
    }
}

int ri() {
    char c = getchar();
    int sign = 1, ret = 0;
```

```
    while (isspace(c)) c = getchar();
    if (c == '-') sign = -1, c = getchar();
    while (isdigit(c)) ret *= 10, ret += c - '0', c = getchar();
    return ret * sign;
}

int g[N];
long long ans[N];
unordered_map<int, int> cnt[N], colCnt;
long long initAns = 0;

void update(int u, unordered_map<int, int> &all) {
    static int vis[N] = {0};
    if (vis[u] == vid)return;
    vis[u] = vid;
    all[g[u]]++;
    initAns += colCnt[g[u]];
    neig(adj, u, e, v)update(v, all);
}


void add(int u, unordered_map<int, int> &all) {
    static int vis[N] = {0};
    if (vis[u] == vid)return;
    vis[u] = vid;
    colCnt[g[u]]++;
    if(isArticl[u]!=vid)
        ans[u] = all[g[u]] - 1;
    neig(adj, u, e, v)add(v, all);
}

int subSz[N], dfsOrder[N], st[N], en[N], tim;

void calcSz(int u, int p = -1) {
    subSz[u] = cnt[u].size();
    dfsOrder[st[u] = tim++] = u;
    neig(blktree, u, e, v)if (v != p)calcSz(v, u), subSz[u] += subSz[v];
    en[u] = tim - 1;
}

unordered_map<int, int> all;

pair<long long, unordered_map<int, int>> smtoBg(int u, int p) {
    int bg = -1;
    pair<long long, unordered_map<int, int>> ret(0, {});
    vector<unordered_map<int, int>> sm;
    neig(blktree, u, e, v)if (v != p)
            if (subSz[v] * 2 > subSz[u])
                ret = smtoBg(v, u);
            else sm.emplace_back(smtoBg(v, u).second);

    for (auto &s:sm)
        for (auto &pr:s)
            ret.first -= pr.second * 1ll * ret.second[pr.first];
    for (auto &pr:cnt[u])
        ret.first -= pr.second * 1ll * ret.second[pr.first];
    long long sub = 0;
    for (auto &s:sm) {
        for (auto &pr:s)
            sub += pr.second * 1ll * ret.second[pr.first];
        for (auto &pr:s)
            ret.second[pr.first] += pr.second;
    }
    for (auto &pr:cnt[u]) {
        sub += pr.second * 1ll * ret.second[pr.first];
        ret.second[pr.first] += pr.second;
    }

    for (auto &s:sm)
```

```
        for (auto &pr:s)
            ret.first += (all[pr.first] - ret.second[pr.first]) * 1ll * pr.second;
    for (auto &pr:cnt[u])
        ret.first += (all[pr.first] - ret.second[pr.first]) * 1ll * pr.second;
    if(u<=cntArt)
        ans[artIdx[u]]=sub+ret.first;
    return ret;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("test.in", "r", stdin);
    //  freopen("test.txt", "w", stdout);
#endif
    adj.n = ri();
    int m = ri();
    rep2(i, 1, adj.n) {
        g[i] = ri();
    }
    while (m--) {
        int u = ri(), v = ri();
        adj.addBiEdge(u, v);
        //  printf("%d,%d %d,%d %d,%d\n", u, g[u], v, g[v], adj.ne - 2, adj.ne -
1);
    }
    buildBlockCut();
//      printTree();
    rep2(i, 1, adj.n) {
        //    printf("%d %d %d\n", id[i], i,g[i]);
        cnt[id[i]][g[i]]++;
    }
//    rep2(i, 1, blktree.n) {
//        printf("%d:", i);
//        for (auto &p:cnt[i])
//            printf(" (%d,%d)", p.first, p.second);
//        puts("");
//    }
    rep(i, rootsCnt) {
        all.clear();
        update(roots[i], all);
        add(roots[i], all);
        calcSz(id[roots[i]]);
        smtoBg(id[roots[i]], -1);
    }
    rep2(i, 1, adj.n)ans[i] += initAns, printf("%lld\n", ans[i]);
    return 0;
}
```

```cpp
const int N = 2e5 + 9;

typedef vector<vector<pair<int, int>>> graph;
struct bridges {
    int tin[N], low[N], timer;

    void init(int n) {
        memset(tin, -1, n * sizeof(tin[0]));
        timer = 0;
    }

    void dfs(int node, graph &adj, vector<bool> &is_bridge, int p = -1) {
        tin[node] = low[node] = timer++;
        for (auto[child, idx]:adj[node]) {
            if (child != p) {
                if (tin[child] == -1) {
                    dfs(child, adj, is_bridge, node);
                    low[node] = min(low[node], low[child]);
                    if (low[child] > tin[node])
                        is_bridge[idx] = true;
                } else {
                    low[node] = min(low[node], tin[child]);
                }
            } else p = -1;//handle multiple edges
        }
    }

    vector<bool> get_bridges(graph &adj, int m) {
        //vector<pair> {node ,index in input}
        vector<bool> is_bridge(m);
        for (int i = 0; i < adj.size(); i++) {
            if (tin[i] == -1)dfs(i, adj, is_bridge);
        }
        return is_bridge;
    }
};
```

```cpp
/*
 *
    Bridge Component : A bridge component of a given graph is the maximal
connected subgraph which does not contain any bridge edges.

    Bridge Tree : If each bridge component of a given graph is shrinked into/
represented as a single node, and these nodes are connected to each
    other by the bridge edges which separated these components, then the resulting
tree formed is called a Bridge Tree.

    Properties of the Bridge Tree

    Each edge in the bridge tree is the one of the bridge edges in the original
graph.
    Since each node in the bridge tree is formed by shrinking the bridge
components of original graph, therefore the bridge tree of a graph with N vertices
can have at most N nodes (and N-1 edges).
    From the above point, it directly follows that a graph with N vertices can
have at most N-1 bridges (why ?? )

 */

const int N = 2e5 + 9;

typedef vector <vector<pair < int, int>>> graph;

struct bridge_tree {
    int tin[N], low[N], timer, cmpId[N], cntCmp;
    graph &p_adj;//private adj
    vector<bool> is_bridge;

    bridge_tree(graph &pAdj) : p_adj(pAdj) {}

    void init(int n) {
        memset(tin, -1, n * sizeof(tin[0]));
        timer = 0;
        cntCmp = 0;
    }

    void dfs(int node, int p = -1, bool block_bridges = false) {
        tin[node] = low[node] = timer++;
        cmpId[node] = cntCmp;
        for (auto[child, idx]:p_adj[node]) {
            if (block_bridges && is_bridge[idx])continue;
            if (child != p) {
                if (tin[child] == -1) {
                    dfs(child, node, block_bridges);
                    low[node] = min(low[node], low[child]);
                    if (low[child] > tin[node])
                        is_bridge[idx] = true;
                } else {
                    low[node] = min(low[node], tin[child]);
                }
            } else p = -1;//handle multiple edges
        }
    }

    vector<bool> get_bridges(graph &adj, int m) {
        //vector<pair> {node ,index in input}
        init(adj.size());
        is_bridge.clear();
        is_bridge.resize(m);
        p_adj = adj;
        for (int i = 0; i < adj.size(); i++) {
            if (tin[i] == -1)dfs(i);
        }
        return is_bridge;
    }
```

```
    vector <vector<int>> get_bridge_tree(graph &adj, int m) {
        vector<bool> bridges = get_bridges(adj, m);
        int n = adj.size();
        init(n);
        for (int i = 0; i < n; i++) {
            if (tin[i] == -1) {
                dfs(i, -1, true);
                cntCmp++;
            }
        }
        vector <vector<int>> ret(cntCmp);
        for (int i = 0; i < n; i++) {
            for (auto[child, idx]:adj[i]) {
                if (bridges[idx]) {
                    int c1 = cmpId[i], c2 = cmpId[child];
                    if (c1 != c2) {
                        ret[c1].emplace_back(c2);
                    }
                }
            }
        }
        return ret;
    }
};
```

```cpp
#include <bits/stdc++.h>

#define ll long long
#define ii pair<int,int>
using namespace std;
const int OO = 1e8 + 9, N = 2e5 + 9;
const ll mod = 1e9 + 7;

typedef vector <vector<pair < int, int>>>
graph;

struct bridge_tree {
    int tin[N], low[N], timer, cmpId[N], cntCmp;
    graph &p_adj;//private adj
    vector<bool> is_bridge;

    bridge_tree(graph &pAdj) : p_adj(pAdj) {}

    void init(int n) {
        memset(tin, -1, n * sizeof(tin[0]));
        timer = 0;
        cntCmp = 0;
    }

    void dfs(int node, int p = -1, bool block_bridges = false) {
        tin[node] = low[node] = timer++;
        cmpId[node] = cntCmp;
        for (auto[child, idx]:p_adj[node]) {
            if (block_bridges && is_bridge[idx])continue;
            if (child != p) {
                if (tin[child] == -1) {
                    dfs(child, node, block_bridges);
                    low[node] = min(low[node], low[child]);
                    if (low[child] > tin[node])
                        is_bridge[idx] = true;
                } else {
                    low[node] = min(low[node], tin[child]);
                }
            } else p = -1;//handle multiple edges
        }
    }

    vector<bool> get_bridges(graph &adj, int m) {
        //vector<pair> {node ,index in input}
        init(adj.size());
        is_bridge.clear();
        is_bridge.resize(m);
        p_adj = adj;
        for (int i = 0; i < adj.size(); i++) {
            if (tin[i] == -1)dfs(i);
        }
        return is_bridge;
    }

    vector <vector<int>> get_bridge_tree(graph &adj, vector<int>
&NodeToBridgeComp, int m) {
        vector<bool> bridges = get_bridges(adj, m);
        int n = adj.size();
        init(n);
        for (int i = 0; i < n; i++) {
            if (tin[i] == -1) {
                dfs(i, -1, true);
                cntCmp++;
            }
        }
        vector <vector<int>> ret(cntCmp);
        NodeToBridgeComp = vector<int>(n);
        for (int i = 0; i < n; i++) {
            NodeToBridgeComp[i] = cmpId[i];
```

```
            for (auto[child, idx]:adj[i]) {
                if (bridges[idx]) {
                    int c1 = cmpId[i], c2 = cmpId[child];
                    if (c1 != c2) {
                        ret[c1].emplace_back(c2);
                    }
                }
            }
        }
        return ret;
    }
};

vector <vector<int>> adj, up;
int tin[N], tout[N], timer, l, depth[N];

void dfs(int v, int p) {
    tin[v] = ++timer;
    up[v][0] = p, depth[v] = depth[p] + 1;
    for (int i = 1; i <= l; ++i) up[v][i] = up[up[v][i - 1]][i - 1];
    for (int u : adj[v]) if (u != p) dfs(u, v);
    tout[v] = ++timer;
}

bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v) {
    if (is_ancestor(u, v))return u;
    if (is_ancestor(v, u)) return v;
    for (int i = l; i >= 0; --i) if (!is_ancestor(up[u][i], v)) u = up[u][i];
    return up[u][0];
}

void preprocess(int root, int n) {
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    depth[root] = -1;
    dfs(root, root);
}

int dist(int u, int v) { return u == -1 ? 0 : depth[u] + depth[v] - 2 *
depth[lca(u, v)]; }


ii getCommonPath(int a, int b, int p, int q) {
    // common path between two chains a->b and p->q
    if (depth[a] < depth[b])swap(a, b);
    if (depth[p] < depth[q])swap(p, q);

    if (!is_ancestor(q, a))return {-1, -1};// max(p,q) < min(a,b)

    int lca_of_min = lca(a, p);
    if (is_ancestor(q, b)) {
        // q is higher than b
        if (is_ancestor(b, lca_of_min)) {
            //  lca_of_min  b  q
            return {lca_of_min, b};
        }
    } else {
        // b is higher than q
        if (is_ancestor(q, lca_of_min)) {
            //  lca_of_min q b
            return {lca_of_min, q};
        }
    }
    return {-1, -1};
```

```
}

void test() {
    int n, m, q, u, v;
    scanf("%d%d%d", &n, &m, &q);
    graph g(n);
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        g[u - 1].emplace_back(v - 1, i);
        g[v - 1].emplace_back(u - 1, i);
    }
    bridge_tree bt(g);
    vector<int> nodeToIdx;
    adj = bt.get_bridge_tree(g, nodeToIdx, m);
    preprocess(0, adj.size());
    for (int i = 0; i < q; i++) {
        int a, b, c, d;
        scanf("%d%d%d%d", &c, &d, &a, &b);
        a = nodeToIdx[a - 1], b = nodeToIdx[b - 1], c = nodeToIdx[c - 1], d =
nodeToIdx[d - 1];
        u = lca(a, b), v = lca(c, d);
        int ans = dist(a, b);
        ii x;
        x = getCommonPath(a, u, c, v), ans -= dist(x.first, x.second);
        x = getCommonPath(a, u, d, v), ans -= dist(x.first, x.second);
        x = getCommonPath(b, u, c, v), ans -= dist(x.first, x.second);
        x = getCommonPath(b, u, d, v), ans -= dist(x.first, x.second);
        printf("%d\n", ans);
    }
}

int main() {
//    ios_base::sync_with_stdio(0), cin.tie(0);
//    freopen("input.in", "r", stdin);
//    freopen("input.out", "w", stdout);
    int t = 1;
//    scanf("%d", &t);
//    cin >> t;
    for (int cas = 1; cas <= t; cas++)test();
}
```

```cpp
const int N = 2e5 + 9;

typedef vector <vector<pair < int, int>>>
graph;// {node ,index of edge in input}

struct bridgesAndCutPoints {
    int tin[N], low[N], timer;

    void dfs(int node, graph &adj, vector<bool> &is_bridge, vector<bool>
&is_cutpoint, int p = -1) {
        tin[node] = low[node] = timer++;
        bool isRoot = p == -1;
        int children = 0;
        for (auto[child, idx]:adj[node]) {
            if (child != p) {
                if (tin[child] == -1) {
                    children++;
                    dfs(child, adj, is_bridge, is_cutpoint, node);
                    low[node] = min(low[node], low[child]);
                    if (low[child] > tin[node]) is_bridge[idx] = true;
                    if (low[child] >= tin[node] && !isRoot) is_cutpoint[node] =
true;
                } else {
                    low[node] = min(low[node], tin[child]);
                }
            } else p = -1;//handle multiple edges
        }
        if (isRoot && children > 1)is_cutpoint[node] = true;
    }

    pair <vector<bool>, vector<bool>> get_bridges_and_cut_points(graph &adj, int
m) {
        //adj -> vector<pair> {node ,index of edge in input}
        memset(tin, -1, adj.size() * sizeof(tin[0]));
        timer = 0;
        vector<bool> is_bridge(m);
        vector<bool> is_cutpoint(adj.size());
        for (int i = 0; i < adj.size(); i++) {
            if (tin[i] == -1)dfs(i, adj, is_bridge, is_cutpoint);
        }
        return {is_bridge, is_cutpoint};
    }
};


int main() {
    int n, m;
    while (cin >> n >> m && n + m) {
        graph adj(n);
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            adj[u - 1].push_back({v - 1, i});
            adj[v - 1].push_back({u - 1, i});
        }
        bridgesAndCutPoints solver;
        auto cut_point = solver.get_bridges_and_cut_points(adj, m).second;
        cout << count(cut_point.begin(), cut_point.end(), 1) << endl;
    }
}
```

```cpp
#define ii pair<int,int>

ii diameter(int node, int p = -1) {
    int mx[3] = {0}, diam = 0;
    for (auto child:adj[node]) {
        if (child == p)continue;
        ii tmp = diameter(child, node);
        diam = max(diam, tmp.first);
        mx[0] = tmp.second + 1;
        sort(mx, mx + 3);
    }
    return {max(diam, mx[1] + mx[2]), mx[2]};
}
```

```cpp
struct FlowEdge {
    int v, u;
    ll cap, flow = 0;

    FlowEdge(int v, int u, ll cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const ll flow_inf = 1e18;
    vector <FlowEdge> edges;
    vector <vector<int>> adj;
    int n, m = 0, 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, ll cap, ll cap2 = 0) {
        edges.emplace_back(v, u, cap), edges.emplace_back(u, v, cap2);
        adj[v].push_back(m), adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        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;
    }

    ll dfs(int v, ll mnCap) {
        if (mnCap == 0)return 0;
        if (v == t) return mnCap;

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

    ll flow() {
        ll f = 0;
        while (bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
```

```
            while (ll cur_flow = dfs(s, flow_inf)) f += cur_flow;
        }
        return f;
    }
};
```

```cpp
#include <bits/stdc++.h>

using namespace std;
const int N = 1e5 + 7, M = 1e5 + 7;

struct edge {
    int f, t, c;
} edges[M];

int delC[M];

int n, m;

struct DSU {
    int par[N], sz[N], ncmp, sets[N], pos[N];
    int tail[N], nxt[N];

    void init(int n) {
        iota(par, par + n, 0);
        iota(sets, sets + n, 0);
        iota(pos, pos + n, 0);
        iota(tail, tail + n, 0);
        ncmp = n;
        fill(sz, sz + n, 1);
        memset(nxt, -1, n * sizeof nxt[0]);
    }

    int operator[](int u) {
        return par[u] = par[u] == u ? u : (*this)[par[u]];
    }

    bool operator()(int u, int v) {
        u = (*this)[u];
        v = (*this)[v];
        if (u == v) return 0;

        if (sz[u] < sz[v]) swap(u, v);
        par[v] = u;
        sz[u] += sz[v];

        int &t = tail[u];
        nxt[t] = v;
        t = tail[v];

        int p = pos[v];
        sets[p] = sets[--ncmp];
        pos[sets[p]] = p;

        return 1;
    }
};

//inId stores node parent

int remN[N], remE[M], mnCst[N], par[N], used[M], inId[N], vis[N], vid;
int uid, prevTo[M];
DSU dsu;

vector<int> mustDelete[M];

int chu(int ne, int root) {
    int orgNe = ne;
    ++uid;
    dsu.init(n);
    iota(remE, remE + ne, 0);

    bool notFinish = 1;
    int res = 0;
```

```
    memset(par, -1, n * sizeof par[0]);
    while (notFinish) {
        notFinish = 0;

        vis[root] = ++vid;
        mnCst[root] = 0;
        inId[root] = -1;


        for (int i = 0; i < ne; ++i) {
            int e = remE[i];
            int f = dsu[edges[e].f];
            int t = dsu[edges[e].t];

            if (t == root || f == t)
                continue;
            int realT = prevTo[e];

            if (~par[realT] && par[realT] != e)
                mustDelete[e].push_back(par[realT]);
        }

        //Select minimum incoming edge for each node
        for (int i = 0; i < ne; ++i) {
            int e = remE[i];
            int f = dsu[edges[e].f];
            int t = dsu[edges[e].t];
            if (t == root /*don't care about root*/ || f == t /*same component*/) {
                swap(remE[i--], remE[--ne]);
                continue;
            }
            int realT = prevTo[e];


            int c = edges[e].c - delC[e];
            if (vis[t] != vid || c < mnCst[t]) {
                vis[t] = vid;
                mnCst[t] = c;
                inId[t] = f;

                par[t] = e;

            }
            //because t could change in dsu
            prevTo[e] = t;
        }

        for (int i = 0; i < ne; ++i) {
            int e = remE[i];
            int t = dsu[edges[e].t];
            delC[e] += mnCst[t];
        }

        for (int i = 0; i < dsu.ncmp; ++i) {
            int id = dsu.sets[i];
            remN[i] = id;
            //node exists without incoming edge
            if (vis[id] != vid) return -1;

            res += mnCst[id];

            vis[id] = 0;
            if (id == root) continue;
            used[par[id]] = uid;
        }

        //Contract cycles
        int curncmp = dsu.ncmp;
        for (int i = 0; i < curncmp; ++i) {
```

```
            ++vid;
            int id = remN[i], u;
            for (u = id; u != root && !vis[u]; u = inId[u])
                vis[u] = vid;
            if (u != root && vis[u] == vid) {    //visited a node already visited
in the same iteration
                for (; dsu(u, inId[u]); u = inId[u]);
                notFinish = 1;
            }
        }
    }
    for (int i = 0; i < orgNe; ++i) {
        int e = remE[i];
        if (used[e] == uid)
            for (auto &v : mustDelete[e])
                used[v] = 0;
    }

    return res;
}

int main() {

#ifdef CLION
    freopen("input.in", "rt", stdin);
#else
    freopen("input.txt", "rt", stdin);
    freopen("output.txt", "wt", stdout);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        edges[i] = {--a, --b, c};
        prevTo[i] = b;
    }
    int v = chu(m, 0);
    printf("%d\n", v);
    int usedCnt = 0;
    if (~v) {
        for (int i = 0; i < m; ++i) {

            if (used[i] == uid && edges[i].c == 1) {
                usedCnt++;
                printf("%d\n", i + 1);
            }
            /*if (used[i] == uid) {
                printf("%d %d %d\n", edges[i].f + 1, edges[i].t + 1, edges[i].c);
            }*/
        }
        //assert(usedCnt == v);
    }
}
```

```cpp
//lexo smallest euler tour
#include <bits/stdc++.h>
using namespace std;


const int MAXN = 128, MAXM = 1003;

int head[MAXN], to[MAXM], nxt[MAXM], deg[MAXN], len[MAXM], sorted[MAXM], ne, n;
char arr[MAXM][21];


void addEdge(int u, int v) {
    to[ne] = v;
    nxt[ne] = head[u];
    head[u] = ne++;
}

int res[MAXM], resSZ;
void euler(int u) {
    for (int &e = head[u]; ~e; ) {
        int v = to[e];
        int tmp = e;
        e = nxt[e];
        euler(v);
        res[resSZ++] = tmp;
    }
}

void init() {
    memset(head, -1, n * sizeof head[0]);
    memset(deg, 0, n * sizeof deg[0]);
    ne = 0;
}

void print(int arr[], int s, string name) {
    for (int i = 0; i < s; ++i) {
        cout << name << "[" << i << "] = " << arr[i] << endl;
    }
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
//    freopen("fence.out", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        n = 128;
        init();
        int m;
        cin >> m;
        for (int i = 0; i < m; ++i) {
            cin >> arr[i];
            len[i] = strlen(arr[i]);
            sorted[i] = i;
        }
        sort(sorted, sorted + m, [](int a, int b) {
            return strcmp(arr[a], arr[b]) < 0;
        });
        while (m--) {
            int i = sorted[m];
            int u, v;
            addEdge(u = arr[i][0], v = arr[i][len[i] - 1]);
            ++deg[u];
            --deg[v];
        }
        if (any_of(deg, deg + n, [](int a){
```

```
            return abs(a) > 1;
        }) || count(deg, deg + n, 1) > 1) {
            cout << "***\n";
            continue;
        }
        int st = find(deg, deg + n, 1) - deg;
        if (st == n) {
            st = arr[sorted[0]][0];
        }
        resSZ = 0;
        euler(st);
        if (resSZ != ne) {
            cout << "***\n";
            continue;
        }
        while (resSZ--) {
            int i = res[resSZ];
            i = ne - i - 1;
            i = sorted[i];
            cout << arr[i] << ".\n" [!resSZ];
        }
    }

    return 0;
}
```

```cpp
const int N = 2e5 + 9;
vector <vector<int>> adj, up;
int tin[N], tout[N], timer, n, l;

void dfs(int v, int p) {
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i) up[v][i] = up[up[v][i - 1]][i - 1];
    for (int u : adj[v]) if (u != p) dfs(u, v);
    tout[v] = ++timer;
}

bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v) {
    if (is_ancestor(u, v))return u;
    if (is_ancestor(v, u)) return v;
    for (int i = l; i >= 0; --i) if (!is_ancestor(up[u][i], v)) u = up[u][i];
    return up[u][0];
}

void preprocess(int root) {
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

```cpp
struct Dinic {
    const ll flow_inf = 1e18;
    vector <FlowEdge> edges;
    vector <vector<int>> adj;
    int n, m = 0, 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 u, int v, ll cap, ll cap2 = 0) {
        edges.emplace_back(u, v, cap), edges.emplace_back(v, u, cap2);
        adj[u].push_back(m), adj[v].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        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;
    }

    ll dfs(int v, ll mnCap) {
        if (mnCap == 0)return 0;
        if (v == t) return mnCap;

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

    ll flow() {
        ll f = 0;
        while (bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
            while (ll cur_flow = dfs(s, flow_inf)) f += cur_flow;
        }
        return f;
    }


    void print() {
        bfs();
```

```
        for (int i = 0; i < edges.size(); i++) {
            ll from = edges[i].v, to = edges[i].u, rem = edges[i].cap -
edges[i].flow;
            if (level[from] != -1 && level[to] == -1 && rem < 1)printf("%lld
%lld\n", from + 1, to + 1);
        }
    }
};
```

```cpp
struct FlowEdge {
    int u, v;
    ll cap, flow = 0, cost;

    FlowEdge(int u, int v, ll cap, ll cost) : u(u), v(v), cap(cap), cost(cost) {}
};

struct MinCostMaxFlow {
    const ll flow_inf = 1e18;
    vector <FlowEdge> edges;
    vector <vector<int>> adj;
    vector <ll> dist, flw, vis, par;

    int n, m = 0, src, targ;

    MinCostMaxFlow(int n, int src, int targ) : n(n), src(src), targ(targ) {
        adj.resize(n), dist.resize(n), flw.resize(n), vis.resize(n), par.resize(n);
    }

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

    ll Bellman() {
        int u = src;
        for (int i = 0; i < n; i++)dist[i] = OO, flw[i] = flow_inf, vis[i] = 0;
        flw[u] = flow_inf, flw[targ] = 0, dist[u] = 0, vis[u] = 1;
        queue<int> Q;
        Q.emplace(u);
        int cnt = n;
        while (!Q.empty() and cnt--) {
            int sz = Q.size();
            while (sz--) {
                u = Q.front(), Q.pop();
                vis[u] = 0;
                for (int id : adj[u]) {
                    ll mnCap = edges[id].cap - edges[id].flow;
                    if (mnCap < 1)continue;
                    ll dd = dist[u] + edges[id].cost, v = edges[id].v;
                    if (dd < dist[v]) {
                        dist[v] = dd;
                        flw[v] = min(flw[u], mnCap);
                        par[v] = id;
                        if (!vis[v]) {
                            vis[v] = 1;
                            Q.emplace(v);
                        }
                    }
                }

            }
        }
        if (flw[targ]) {
            for (int i = targ, id; id = par[i], i != src; i = edges[id].u) {
                edges[id].flow += flw[targ];
                edges[id ^ 1].flow -= flw[targ];
            }
        }
        return flw[targ];
    }

    pair <ll, ll> flow() {
        ll res = 0, f, cost = 0;
        while ((f = Bellman())) {
            res += f;
            cost += f * dist[targ];
        }
```

```
            return make_pair(res, cost);
        }
    };
```

```cpp
#include<bits/stdc++.h>
#include<ext/numeric>
#define S second
#define F first
#define neig(a, u, e, v) for(int v, e = (a).head[u] ; e and (v = (a).to[e], 1) ; e
= (a).nxt[e])

using namespace std;

const int N = 1e5 + 5, M = N * 2, NM1 = N - 1;

int k[N];
char names[N][22];

struct Adj {
    int head[N], nxt[M], to[M], ne, n;
    void init(int n)
    {
        memset(head + 1, 0, n * sizeof(head[0]));
        ne = 2;
        this->n = n;
    }
    void addEdge(int f, int t)
    {
        nxt[ne] = head[f];
        to[ne] = t;
        head[f] = ne++;
    }
    void addBiEdge(int f, int t)
    {
        addEdge(f, t);
        addEdge(t, f);
    }
    int addNode()
    {
        head[++n] = 0;
        return n;
    }
}adj, Q;

int par[N], sz[N], dfsOrder[N], sTime[N], eTime[N], dep[N], ans[N], curT;

void calcSz(int u)
{
    sz[u] = 1;
    dfsOrder[sTime[u] = curT++] = u;
    neig(adj, u, e, v)
    {
        if(v == par[u]) continue;
        par[v] = u;
        dep[v] = dep[u] + 1;
        calcSz(v);
        sz[u] += sz[v];
    }
    eTime[u] = curT - 1;
}

deque<unordered_set<string> > smallToLarge(int u)
{
    deque<unordered_set<string> > ret;
    int bg = -1;
    neig(adj, u, e, v)
    {
        if(v == par[u]) continue;
        if(sz[v] * 2 > sz[u]) bg = v;
        else smallToLarge(v);
    }
    if(~bg) ret = smallToLarge(bg);
    ret.emplace_front(unordered_set<string>({names[u]}));
```

```
    neig(adj, u, e, v)
    {
        if(v == par[u]) continue;
        if(sz[v] * 2 > sz[u]) continue;
        for(int i = sTime[v] ; i <= eTime[v] ; i++)
        {
            int w = dfsOrder[i];
            int idx = dep[w] - dep[u];
            if(idx >= ret.size()) ret.resize(idx + 1);
            ret[idx].emplace(names[w]);
        }
    }
    neig(Q, u, e, q) ans[q] = k[q] < ret.size() ? ret[k[q]].size() : 0;
    return ret;
}

int main()
{
    // freopen("in.txt", "r", stdin);
    int t, n, m, p, u;
    scanf("%d", &n);
    adj.init(n);
    Q.init(n);
    vector<int> roots;
    for(int i = 1 ; i <= n ; i++)
    {
        scanf(" %s%d", names[i], &p);
        if(p) adj.addBiEdge(p, i);
        else roots.emplace_back(i);
    }
    scanf("%d", &m);
    for(int i = 0 ; i < m ; i++)
    {
        scanf("%d%d", &u, k + i);
        Q.addEdge(u, i);
    }
    for(auto i:roots)
    {
        calcSz(i);
        smallToLarge(i);
    }
    for(int i = 0 ; i < m ; i++) printf("%d\n", ans[i]);
    return 0;
}
```

```cpp
#include<cstdio>
#include <algorithm>
using namespace std;

#define adj(head, u, e, v) for(int v, e = head[u] ; e and (v = to[e]) ; e = nxt[e])

const int N = 1e2 + 5, M = 2e4 + 5;

int head1[N], head2[N], to[M], nxt[M], vis[N], res[N], cmpId[N], sz, ne = 1, n,
vid, cmpCnt;

void addEdge(int f, int t, int head[])
{
    nxt[ne] = head[f];
    to[ne] = t;
    head[f] = ne++;
}

void dfs1(int u)
{
    vis[u] = vid;
    adj(head1, u, e, v) if(vis[v] != vid) dfs1(v);
    res[sz++] = u;
}

void dfs2(int u, int cId)
{
    //printf("%d\n", u);
    cmpId[u] = cId;
    vis[u] = vid;
    adj(head2, u, e, v)
    {
        if(vis[v] != vid) dfs2(v, cId);
    }
}

void kosaraju()
{
    cmpCnt = 0;
    ++vid;
    sz = 0;
    for(int i = 1 ; i <= n ; i++) if(vis[i] != vid) dfs1(i);
    // for(int i = 0 ; i < n ; i++) printf("%d%c", res[i], " \n"[i == n - 1]);
    ++vid;
    for(int i = n - 1 ; ~i ; i--)
    {
        int u = res[i];
        // printf("%d %d %d\n", u, vis[u], vid);
        // exit(0);

        if(vis[u] != vid) dfs2(u, cmpCnt++);
    }
}
bool notSource[N],notSink[N];
int main()
{
    //  freopen("input.txt", "r", stdin);
    int  u, v;
    scanf("%d", &n);
    for(int i = 1 ; i <= n ; i++)
    {
        while(scanf("%d", &u), u)
        {
            addEdge(i, u, head1);
            addEdge(u, i, head2);
        }
    }
    kosaraju();
    for (int i = 1; i <=n; ++i)
```

```
    {
        adj(head1,i,e,j)
        {
            if(cmpId[i]!=cmpId[j])
            {
                notSource[cmpId[j]]=1;
                notSink[cmpId[i]]=1;
            }
        }
    }
    int cntSrcs=count(notSource,notSource+cmpCnt,0);
    int cntSnks=count(notSink,notSink+cmpCnt,0);
    printf("%d\n",cntSrcs);
    if(cmpCnt==1)puts("0");
    else printf("%d\n",max(cntSrcs,cntSnks));
    return 0;
}
```

```cpp
vector<int> tree_center(vector<vector<int> > &adj, int src) {
    //src to handle 1 node tree
    int n = adj.size();
    vector<int> deg(n, -1);
    queue<int> q;
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        if (!adj[i].empty())cnt++;
        if (adj[i].size() == 1) q.push(i);
        else deg[i] = adj[i].size();
    }
    int rem = cnt;
    while (rem > 2 && !q.empty()) {
        int sz = q.size();
        while (sz--) {
            int cur = q.front();
            q.pop();
            for (int child: adj[cur]) {
                if (--deg[child] == 1) {
                    q.push(child);
                }
            }
            rem--;
        }
    }
    vector<int> ret;
    if (q.empty()) return {src};

    ret.push_back(q.front()), q.pop();
    if (!q.empty())ret.push_back(q.front());
    return ret;
}
```

Mathematics

```cpp
#include<bits/stdc++.h>
#define S second
#define F first

using namespace std;

const int N = 1e5 + 15, M = 2 * N;

#define arr(t, n, s, e) t _##n[e-s+1], *n= _##n+(-s);

typedef long long ll;

vector<int> fact(int x)
{
    vector<int> ret;
    for(int i = 2 ; i <= x / i ; i += 1 + (i&1))
    {
        if(x % i == 0) ret.emplace_back(i);
        while(x % i == 0) x /= i;
    }
    if(x > 1) ret.emplace_back(x);
    return ret;
}

int main()
{
//    freopen("input.in", "r", stdin);
//    freopen("out.out", "w", stdout);
    int t, tc = 0, n;
    ll a, b;
    scanf("%d", &t);
    while(t--)
    {
        ll ans = 0;
        scanf("%lld%lld%d", &a, &b, &n);
        printf("Case #%d: ", ++tc);
        auto v = fact(n);
        for(int mask = 1 ; mask < (1<<v.size()) ; mask++)
        {
            int p = 1;
            for(int i = 0 ; i < v.size() ; i++)
            {
                if((mask>>i)&1) p *= v[i];
            }
            bool isOdd = __builtin_popcount(mask)&1;
            ll st = (a + p - 1) / p, en = b / p;
            ll sz = en - st + 1;
            ans += (isOdd * 2 - 1) * sz;
        }
        printf("%lld\n", b - a + 1 - ans);
    }
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long

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

vector<ll> v1;
ll n;
ll inc_excl (ll num , ll index = 0 , int sign = -1 , ll ans = 1 , int chosen = 0) {
    if ( index == 5 ) {
        if ( chosen == 0 )
            return 0;
        else
            return sign * num / ans;
    }
    ll ret = inc_excl(num , index + 1 , sign , ans , chosen);//not chosen
    ret += inc_excl(num , index + 1 , sign * -1 , lcm(ans , v1[ index ]) , chosen
+ 1);//chosen
    return ret;

}
ll stresstest (ll mini , int maxi) {
    ll counter = 0;
    for ( int i = mini ; i <= maxi ; i++ ) {
        if ( i % v1[ 0 ]    && i % v1[ 1 ]&& i % v1[ 2 ] && i % v1[ 3 ]  && i %
v1[ 4 ] )
            counter++;
    }
    return counter ;
}
int main () {

    //freopen("D:\\cs\\problem solving\\Competitive-Programming\\input.txt" ,
"r" , stdin);
    int T;
    int counter =0;

    T =0;
    cin>>T;
    while (  T--) {
        ll a , b , x , y;

        cin >> a >> b >> x >> y;
        v1 = vector<ll>{x , x + y , x + (2 * y) , x + (3 * y) , x + (4 * y)};

        ll ans = inc_excl(b) - inc_excl(a - 1);
        ans = (b-a+1 ) - ans;
        //ll num = stresstest(a,b) ;
//        if(num != ans){
//            cout<<a<< "     "<<b<<"     "<<x<<"         "<<y<<endl;
//            cout<<"     "<<num<<"            "<<ans<<endl;
//        }
//        if ( ans )
//            num++;
        cout << ans << endl;
    }
}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
#define pb push_back
using namespace std;

const int N = 50 + 2;
const double OO = 1e18;
ll n, m;


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

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0, ll &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt) {
    x += cnt * b;
    y -= cnt * a;
}

ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy) {
    //ax+by=c
    ll x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    ll sign_a = a > 0 ? +1 : -1;
    ll sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    ll lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    ll rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
```

```
        return 0;
    ll lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    ll rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    ll lx = max(lx1, lx2);
    ll rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}

int main() {
    int t;
    scanf("%d", &t);
    for (int cas = 1; cas <= t; cas++) {
        ll a, b, c, x, y, g;
        scanf("%lld%lld%lld", &a, &b, &c);
        printf("Case %d: ", cas);
        puts(find_any_solution(a, b, c, x, y, g) ? "Yes" : "No");
    }
}
```

Theoretical Computer Science Cheat Sheet

Definitions

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf_{n\to\infty} a_n$	$\lim_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup_{n\to\infty} a_n$	$\lim_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\left[{n \atop 1}\right] = (n-1)!$, **15.** $\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\left[{n \atop n}\right] = 1$, **17.** $\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\left\langle{0 \atop k}\right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0$ for $n \ne 0$,

34. $\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^{n} \begin{bmatrix} k \\ m \end{bmatrix} n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix}$,

39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^{n} \left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle \binom{x+k}{2n}$,

40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{n}{k} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k}$,

41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}$,

42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^{m} k \begin{Bmatrix} n+k \\ k \end{Bmatrix}$,

43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^{m} k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix}$,

44. $\binom{n}{m} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k}$,

45. $(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{m-k}$, for $n \geq m$,

46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix}$,

47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix}$,

48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$,

49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:

$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i / 2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{aligned} &1\big(T(n) - 3T(n/2) = n\big) \\ &3\big(T(n/2) - 3T(n/4) = n/2\big) \\ &\quad \vdots \quad \vdots \quad \vdots \\ &3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big) \end{aligned}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find

$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$B_0 = 1$, $B_1 = -\frac{1}{2}$, $B_2 = \frac{1}{6}$, $B_4 = -\frac{1}{30}$, $B_6 = \frac{1}{42}$, $B_8 = -\frac{1}{30}$, $B_{10} = \frac{5}{66}$.

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \ldots$

$$\ln n < H_n < \ln n + 1,$$

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1,2) & j = 1 \\ a(i-1, a(i, j-1)) & i,j \geq 2 \end{cases}$$

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$

$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \Pr[X_i] + \sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

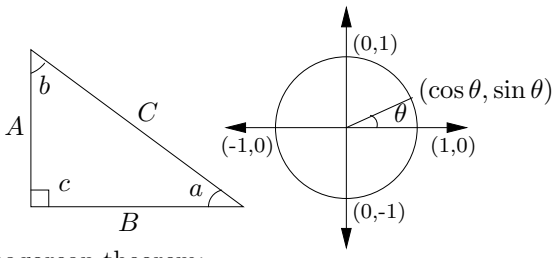
Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

Theoretical Computer Science Cheat Sheet

Trigonometry



Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2\sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$
$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i\sin x, \qquad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g\begin{vmatrix} b & c \\ e & f \end{vmatrix} - h\begin{vmatrix} a & c \\ d & f \end{vmatrix} + i\begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= aei + bfg + cdh - ceg - fha - ibd.$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

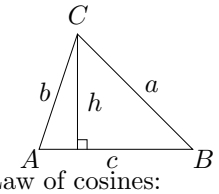
Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2\sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2\tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2\tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

. . . in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Number Theory

The Chinese remainder theorem: There exists a number C such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^{n} p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\Big(\frac{a}{d}\Big).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n\frac{\ln \ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

Graph Theory

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq \lvert S \rvert$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k, \ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

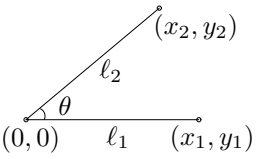
Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$
$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\frac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$
$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$
$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Calculus

Derivatives:

1. $\frac{d(cu)}{dx} = c\frac{du}{dx}$, **2.** $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$, **3.** $\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$,

4. $\frac{d(u^n)}{dx} = nu^{n-1}\frac{du}{dx}$, **5.** $\frac{d(u/v)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\frac{d(e^{cu})}{dx} = ce^{cu}\frac{du}{dx}$,

7. $\frac{d(c^u)}{dx} = (\ln c)c^u\frac{du}{dx}$, **8.** $\frac{d(\ln u)}{dx} = \frac{1}{u}\frac{du}{dx}$,

9. $\frac{d(\sin u)}{dx} = \cos u\frac{du}{dx}$, **10.** $\frac{d(\cos u)}{dx} = -\sin u\frac{du}{dx}$,

11. $\frac{d(\tan u)}{dx} = \sec^2 u\frac{du}{dx}$, **12.** $\frac{d(\cot u)}{dx} = \csc^2 u\frac{du}{dx}$,

13. $\frac{d(\sec u)}{dx} = \tan u\,\sec u\frac{du}{dx}$, **14.** $\frac{d(\csc u)}{dx} = -\cot u\,\csc u\frac{du}{dx}$,

15. $\frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}}\frac{du}{dx}$, **16.** $\frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}}\frac{du}{dx}$,

17. $\frac{d(\arctan u)}{dx} = \frac{1}{1+u^2}\frac{du}{dx}$, **18.** $\frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2}\frac{du}{dx}$,

19. $\frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}}\frac{du}{dx}$, **20.** $\frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$,

21. $\frac{d(\sinh u)}{dx} = \cosh u\frac{du}{dx}$, **22.** $\frac{d(\cosh u)}{dx} = \sinh u\frac{du}{dx}$,

23. $\frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\frac{du}{dx}$, **24.** $\frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\frac{du}{dx}$,

25. $\frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u\,\tanh u\frac{du}{dx}$, **26.** $\frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u\,\coth u\frac{du}{dx}$,

27. $\frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}}\frac{du}{dx}$, **28.** $\frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}}\frac{du}{dx}$,

29. $\frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2}\frac{du}{dx}$, **30.** $\frac{d(\operatorname{arccoth} u)}{dx} = \frac{1}{u^2-1}\frac{du}{dx}$,

31. $\frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}}\frac{du}{dx}$, **32.** $\frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{|u|\sqrt{1+u^2}}\frac{du}{dx}$.

Integrals:

1. $\int cu\,dx = c\int u\,dx$, **2.** $\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\int \frac{1}{x}dx = \ln x$, **5.** $\int e^x\,dx = e^x$,

6. $\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\int \sin x\,dx = -\cos x$, **9.** $\int \cos x\,dx = \sin x$,

10. $\int \tan x\,dx = -\ln|\cos x|$, **11.** $\int \cot x\,dx = \ln|\cos x|$,

12. $\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\int \arcsin\frac{x}{a}dx = \arcsin\frac{x}{a} + \sqrt{a^2-x^2}, \quad a > 0$,

15. $\int \arccos\frac{x}{a}dx = \arccos\frac{x}{a} - \sqrt{a^2-x^2}, \quad a>0,$

16. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2), \quad a>0,$

17. $\int \sin^2(ax)dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax)dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\,dx = \tan x,$

20. $\int \csc^2 x\,dx = -\cot x,$

21. $\int \sin^n x\,dx = -\frac{\sin^{n-1}x\cos x}{n} + \frac{n-1}{n}\int \sin^{n-2}x\,dx,$

22. $\int \cos^n x\,dx = \frac{\cos^{n-1}x\sin x}{n} + \frac{n-1}{n}\int \cos^{n-2}x\,dx,$

23. $\int \tan^n x\,dx = \frac{\tan^{n-1}x}{n-1} - \int \tan^{n-2}x\,dx, \quad n\neq 1,$

24. $\int \cot^n x\,dx = -\frac{\cot^{n-1}x}{n-1} - \int \cot^{n-2}x\,dx, \quad n\neq 1,$

25. $\int \sec^n x\,dx = \frac{\tan x\sec^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \sec^{n-2}x\,dx, \quad n\neq 1,$

26. $\int \csc^n x\,dx = -\frac{\cot x\csc^{n-1}x}{n-1} + \frac{n-2}{n-1}\int \csc^{n-2}x\,dx, \quad n\neq 1,$

27. $\int \sinh x\,dx = \cosh x,$

28. $\int \cosh x\,dx = \sinh x,$

29. $\int \tanh x\,dx = \ln|\cosh x|,$

30. $\int \coth x\,dx = \ln|\sinh x|,$

31. $\int \operatorname{sech} x\,dx = \arctan\sinh x,$

32. $\int \operatorname{csch} x\,dx = \ln\left|\tanh\frac{x}{2}\right|,$

33. $\int \sinh^2 x\,dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x\,dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x\,dx = \tanh x,$

36. $\int \operatorname{arcsinh}\frac{x}{a}dx = x\operatorname{arcsinh}\frac{x}{a} - \sqrt{x^2+a^2}, \quad a>0,$

37. $\int \operatorname{arctanh}\frac{x}{a}dx = x\operatorname{arctanh}\frac{x}{a} + \frac{a}{2}\ln|a^2-x^2|,$

38. $\int \operatorname{arccosh}\frac{x}{a}dx = \begin{cases} x\operatorname{arccosh}\dfrac{x}{a} - \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a}>0 \text{ and } a>0, \\ x\operatorname{arccosh}\dfrac{x}{a} + \sqrt{x^2+a^2}, & \text{if } \operatorname{arccosh}\frac{x}{a}<0 \text{ and } a>0, \end{cases}$

39. $\int \frac{dx}{\sqrt{a^2+x^2}} = \ln\left(x+\sqrt{a^2+x^2}\right), \quad a>0,$

40. $\int \frac{dx}{a^2+x^2} = \frac{1}{a}\arctan\frac{x}{a}, \quad a>0,$

41. $\int \sqrt{a^2-x^2}\,dx = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

42. $\int (a^2-x^2)^{3/2}dx = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

43. $\int \frac{dx}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a}, \quad a>0,$

44. $\int \frac{dx}{a^2-x^2} = \frac{1}{2a}\ln\left|\frac{a+x}{a-x}\right|,$

45. $\int \frac{dx}{(a^2-x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2-x^2}},$

46. $\int \sqrt{a^2\pm x^2}\,dx = \frac{x}{2}\sqrt{a^2\pm x^2} \pm \frac{a^2}{2}\ln\left|x+\sqrt{a^2\pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2-a^2}} = \ln\left|x+\sqrt{x^2-a^2}\right|, \quad a>0,$

48. $\int \frac{dx}{ax^2+bx} = \frac{1}{a}\ln\left|\frac{x}{a+bx}\right|,$

49. $\int x\sqrt{a+bx}\,dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a+bx}}{x}\,dx = 2\sqrt{a+bx} + a\int \frac{1}{x\sqrt{a+bx}}dx,$

51. $\int \frac{x}{\sqrt{a+bx}}\,dx = \frac{1}{\sqrt{2}}\ln\left|\frac{\sqrt{a+bx}-\sqrt{a}}{\sqrt{a+bx}+\sqrt{a}}\right|, \quad a>0,$

52. $\int \frac{\sqrt{a^2-x^2}}{x}\,dx = \sqrt{a^2-x^2} - a\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

53. $\int x\sqrt{a^2-x^2}\,dx = -\frac{1}{3}(a^2-x^2)^{3/2},$

54. $\int x^2\sqrt{a^2-x^2}\,dx = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a}, \quad a>0,$

55. $\int \frac{dx}{\sqrt{a^2-x^2}} = -\frac{1}{a}\ln\left|\frac{a+\sqrt{a^2-x^2}}{x}\right|,$

56. $\int \frac{x\,dx}{\sqrt{a^2-x^2}} = -\sqrt{a^2-x^2},$

57. $\int \frac{x^2\,dx}{\sqrt{a^2-x^2}} = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a}, \quad a>0,$

58. $\int \frac{\sqrt{a^2+x^2}}{x}\,dx = \sqrt{a^2+x^2} - a\ln\left|\frac{a+\sqrt{a^2+x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2-a^2}}{x}\,dx = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|}, \quad a>0,$

60. $\int x\sqrt{x^2\pm a^2}\,dx = \frac{1}{3}(x^2\pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2+a^2}} = \frac{1}{a}\ln\left|\frac{x}{a+\sqrt{a^2+x^2}}\right|,$

Calculus Cont.

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

65. $\displaystyle\int \frac{\sqrt{x^2+a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac, \\ \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac, \end{cases}$$

67. $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0, \\ \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0, \end{cases}$$

68. $\displaystyle\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

69. $\displaystyle\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$

70. $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0, \\ \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0, \end{cases}$$

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax) + \tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax) - \tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

74. $\displaystyle\int x^n e^{ax}\,dx = \frac{x^n e^{ax}}{a} - \tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

75. $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2}\right),$

76. $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

$$
\begin{aligned}
x^1 &= x^{\underline{1}} &&= x^{\overline{1}} \\
x^2 &= x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{2}} - x^{\overline{1}} \\
x^3 &= x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}} \\
x^4 &= x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}} \\
x^5 &= x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}} &&= x^{\underline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}
\end{aligned}
$$

$$
\begin{aligned}
x^{\overline{1}} &= x^1 & x^{\underline{1}} &= x^1 \\
x^{\overline{2}} &= x^2 + x^1 & x^{\underline{2}} &= x^2 - x^1 \\
x^{\overline{3}} &= x^3 + 3x^2 + 2x^1 & x^{\underline{3}} &= x^3 - 3x^2 + 2x^1 \\
x^{\overline{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\underline{4}} &= x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\overline{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\underline{5}} &= x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}
$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$
$$\mathrm{E}\,f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u + \Delta v,$

$\Delta(uv) = u\Delta v + \mathrm{E}\,v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum(u+v)\,\delta x = \sum u\,\delta x + \sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv - \sum \mathrm{E}\,v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum\binom{x}{m}\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$$
$$x^{\underline{0}} = 1,$$
$$x^{\underline{n}} = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$$
$$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$$
$$x^{\overline{0}} = 1,$$
$$x^{\overline{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$$
$$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$$

Conversion:

$$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}} = 1/(x+1)^{\overline{-n}},$$
$$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}} = 1/(x-1)^{\underline{-n}},$$
$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\}(-1)^{n-k}x^{\overline{k}},$$
$$x^{\underline{n}} = \sum_{k=1}^n \left[n \atop k \right](-1)^{n-k}x^k,$$
$$x^{\overline{n}} = \sum_{k=1}^n \left[n \atop k \right]x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \cdots &&= \sum_{i=0}^{\infty} x^i, \\
\frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \cdots &&= \sum_{i=0}^{\infty} c^i x^i, \\
\frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \cdots &&= \sum_{i=0}^{\infty} x^{ni}, \\
\frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \cdots &&= \sum_{i=0}^{\infty} ix^i, \\
\sum_{k=0}^{n} \left\{ n \atop k \right\} \frac{k!z^k}{(1-z)^{k+1}} &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots &&= \sum_{i=0}^{\infty} i^n x^i, \\
e^x &= 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\
\ln(1+x) &= x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots &&= \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\
\ln \frac{1}{1-x} &= x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots &&= \sum_{i=1}^{\infty} \frac{x^i}{i}, \\
\sin x &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\
\cos x &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\
\tan^{-1} x &= x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\
(1+x)^n &= 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\
\frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\
\frac{x}{e^x - 1} &= 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots &&= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\
\frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}} &= 1 + 2x + 6x^2 + 20x^3 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}} \left(\frac{1 - \sqrt{1-4x}}{2x} \right)^n &= 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\
\frac{1}{1-x} \ln \frac{1}{1-x} &= x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots &&= \sum_{i=1}^{\infty} H_i x^i, \\
\frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2 &= \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots &&= \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i}, \\
\frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \cdots &&= \sum_{i=0}^{\infty} F_i x^i, \\
\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} &= F_n x + F_{2n} x^2 + F_{3n} x^3 + \cdots &&= \sum_{i=0}^{\infty} F_{ni} x^i.
\end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\begin{aligned}
\alpha A(x) + \beta B(x) &= \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i, \\
x^k A(x) &= \sum_{i=k}^{\infty} a_{i-k} x^i, \\
\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} &= \sum_{i=0}^{\infty} a_{i+k} x^i, \\
A(cx) &= \sum_{i=0}^{\infty} c^i a_i x^i, \\
A'(x) &= \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i, \\
xA'(x) &= \sum_{i=1}^{\infty} i a_i x^i, \\
\int A(x)\,dx &= \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i, \\
\frac{A(x) + A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i} x^{2i}, \\
\frac{A(x) - A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.
\end{aligned}$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

Series

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i, \qquad \left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\left[{n \atop i} \right]x^i, \qquad (e^x-1)^n = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}\frac{n!x^i}{i!},$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n} \right]\frac{n!x^i}{i!}, \qquad x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^iB_{2i}x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!}, \qquad \zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x}, \qquad \frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

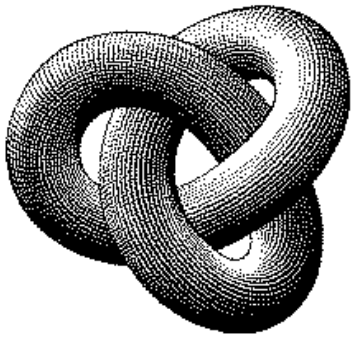
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^ii!^2}{(i+1)(2i+1)!}x^{2i}.$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a\le b\le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots \qquad\quad & \qquad \vdots \qquad\quad \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A=(a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A\neq 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1}+2$ for all i, $1\le i<m$ and $k_m\ge 2$.

Fibonacci Numbers

$1,1,2,3,5,8,13,21,34,55,89,\ldots$

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0 = F_1 = 1,$$

$$F_{-i} = (-1)^{i-1}F_i,$$

$$F_i = \tfrac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i>0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$$

$$F_{2n} = F_nF_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

```cpp
include <bits/stdc++.h>

#define ll long long
using namespace std;
const ll mod = 1e9 + 7;

typedef vector<ll> row;
typedef vector<row> mat;

mat operator*(const mat &a, const mat &b) {
    int r1 = a.size(), r2 = b.size(), c2 = b[0].size();
    mat ret(r1, row(c2));
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c2; j++)
            for (int k = 0; k < r2; k++)
                ret[i][j] += a[i][k] * b[k][j], ret[i][j] %= mod;
    return ret;
}

mat operator^(const mat &a, ll k) {
    mat ret(a.size(), row(a.size()));
    for (int i = 0; i < a.size(); i++)ret[i][i] = 1;
    for (mat tmp = a; k; tmp = tmp * tmp, k /= 2)if (k & 1)ret = ret * tmp;
    return ret;
}
```

```
/*
 * steps to solve matrix power problems
 * write iterative dp with memory reduction
 * lets say i want to write
 * new_dp[x] = 3*dp[i-1]
 * int the transition matrix this is equal to
 * trans[i-1][x] =3;
 *
 * to keep old value of a number make a cycle for example let
 * we want to get sum of all number at most k
 * we add extra row let it's index be n and answer of f(n) at row(n-1)
 * so we want each time increase dp[n] by dp[n-1]
 * new_dp[n] = dp[n-1] +dp[n]
 * so we can write it like this
 * trans[n][n] =1 ,this mean add the old of dp[n],
 * trans[n-1][n]= 1 --> new_dp[n]+=dp[n-1]
 *
 *
 * to add constant number p to dp[n-1]
 * answer is at dp[n-1] and p is at n
 * trans[n][n] = 1;//dp[n]=1
 * trans[n][n - 1] = p;//dp[n-1]+=p
 *
 * to add i*q for loop in matrix
 *  trans[n + 1][n + 1] = 1;//new_dp[n+1]=dp[n+1]
 *  trans[n][n] = 1;//dp[n]=1
 *  trans[n][n + 1] = 1;//new_dp[n+1]+=1
 *  trans[n + 1][n - 1] = q;//new_dp[n-1]+=i*q
 *
 *  //i*i*r
 *  trans[n + 2][n + 2] = 1;
 *  trans[n + 1][n + 2] = 2;
 *  trans[n][n + 2] = 1;
 *  trans[n + 2][n - 1] = r;
 */
```

```cpp
/*
 * ai=(c1·ai−1+c2·ai−2+⋯+cn·ai−n)+p+i·q+i2·r
 * Find the value of the kth a  mod 1e9+7.
 */
#include <bits/stdc++.h>

#define ll long long
#define REP(i, n) for(int i=0;i<(n);i++)
using namespace std;
const ll mod = 1e9 + 7;

typedef vector<ll> row;
typedef vector<row> mat;

mat operator*(const mat &a, const mat &b) {
    int r1 = a.size(), r2 = b.size(), c2 = b[0].size();
    mat ret(r1, row(c2));
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c2; j++)
            for (int k = 0; k < r2; k++)
                ret[i][j] += a[i][k] * b[k][j], ret[i][j] %= mod;
    return ret;
}

mat operator^(const mat &a, ll k) {
    mat ret(a.size(), row(a.size()));
    for (int i = 0; i < a.size(); i++)ret[i][i] = 1;
    for (mat tmp = a; k; tmp = tmp * tmp, k /= 2)if (k & 1)ret = ret * tmp;
    return ret;
}

//void test() {
//    ll n, k, p, q, r;
//    cin >> n >> k;
//    vector<ll> dp(n), c(n);
//    for (int i = 0; i < n; i++)cin >> dp[i];
//    for (int i = 0; i < n; i++)cin >> c[i];
//    cin >> p >> q >> r;
//    if (k < n)cout << dp[k] << endl;
//    else {
//        k -= (n - 1);
//        for (int i = 0; i < k; i++) {
//            vector<ll> new_dp(n);
//            for (int j = 0; j < n - 1; j++) {
//                new_dp[j] = dp[j + 1];
//            }
//            for (int j = 0; j < n; j++) {
//                new_dp.back() += dp[n - 1 - j] * c[j];
//            }
//            new_dp.back() += p;
//            dp = new_dp;
//        }
//        cout << dp.back() << endl;
//    }
//
//}
/*
 * 62
 * 11
 */
void test() {
    ll n, k, p, q, r;
    cin >> n >> k;
    vector<ll> c(n);
    mat init(1, row(n + 3));

    for (int i = 0; i < n; i++)cin >> init[0][i];
    for (int i = 0; i < n; i++)cin >> c[i];
    cin >> p >> q >> r;
```

```
    if (k < n)cout << init[0][k] << endl;
    else {
        k -= (n - 1);
        mat trans(n + 3, row(n + 3));
        for (int j = 0; j < n - 1; j++) {
            trans[j + 1][j] = 1;
        }
        for (int j = 0; j < n; j++) {
            trans[n - 1 - j][n - 1] = c[j];
        }
        //p
        trans[n][n] = 1;//dp[n]=1
        trans[n][n - 1] = p;//dp[n-1]+=p
        //i * q
        trans[n + 1][n + 1] = 1;//new_dp[n+1]=dp[n+1]
        trans[n][n + 1] = 1;//new_dp[n+1]+=1
        trans[n + 1][n - 1] = q;//new_dp[n-1]+=i
        //i*i*r
        trans[n + 2][n + 2] = 1;
        trans[n + 1][n + 2] = 2;
        trans[n][n + 2] = 1;
        trans[n + 2][n - 1] = r;


        trans = trans ^ k;
        init[0][n] = 1;
        init[0][n + 1] = n;
        init[0][n + 2] = n * n;
        cout << (init * trans)[0][n - 1] << endl;
    }

}

int main() {
    ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
//    freopen("input.in", "r", stdin);
//    freopen("output.txt", "w", stdout);
    int t;
//    cin >> t;
//    for (int cas = 1; cas <= t; cas++) {
    test();
//    }
}
```

```cpp
/*
 * compute all power of twos transtions matrix
 * then do vector matrix multiplication in o(n^2) per each query
 * total complexity O(n^2*q*log(k))
 */
#include <bits/stdc++.h>

#define ll long long
#define REP(i, n) for(int i=0;i<(n);i++)
using namespace std;
ll mod = 1e9L + 7;
const int N = 1e5 + 9;

typedef vector<ll> row;
typedef vector<row> mat;

mat operator*(const mat &a, const mat &b) {
    int r1 = a.size(), c1 = a[0].size(), c2 = b[0].size();
    mat ret(r1, row(c2));
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c2; j++)
            for (int k = 0; k < c1; k++)
                ret[i][j] += a[i][k] * b[k][j], ret[i][j] %= mod;
    return ret;
}

mat pre[35];

mat operator^(const mat &a, ll k) {
    mat ret(a.size(), row(a.size()));
    for (int i = 0; i < a.size(); i++)ret[i][i] = 1;
    int count = 0;
    for (mat tmp = a; k; tmp = tmp * tmp, k /= 2, count++) {
        pre[count] = tmp;
        if (k & 1)ret = ret * tmp;
    }
    return ret;
}

mat adj;
map<int, int> mp;

void test() {
    int n;
    cin >> n;
    int u = 0;
    for (int i = 6; i >= 0; i--) {
        int x;
        cin >> x;
        if (x)u |= (1 << i);
    }
    u = mp[u];
    mat init(1, row(mp.size(), 0));
    n--;
    init[0][u] = 1;
    for (int count = 0; n; n /= 2, count++) {
        if (n & 1)
            init = init * pre[count];
    }
    ll c = 0;
    for (int i = 0; i < init[0].size(); i++)c += init[0][i], c %= mod;
    cout << c << endl;
}

set<int> get_next(int mask) {
    set<int> ret;
    vector<vector<int>> arr;
    for (int i = 0; i < 7; i++) {
        if ((mask >> i) & 1) {
```

```
            arr.push_back({i - 1, i + 1});
        }
    }
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                for (int w = 0; w < 2; w++) {
                    int b1 = arr[0][i], b2 = arr[1][j], b3 = arr[2][k], b4 = arr[3]
[w];
                    if (min({b1, b2, b3, b4}) < 0 || max({b1, b2, b3, b4}) ==
7)continue;
                    int new_mask = (1 << b1) | (1 << b2) | (1 << b3) | (1 << b4);
                    if (__builtin_popcount(new_mask) == 4)ret.insert(new_mask);
                }
            }
        }
    }
    return ret;
}

int main() {
    ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
//    freopen("input.in", "r", stdin);
    //  freopen("output.txt", "w", stdout);
    int t;
    cin >> t;
    for (int i = 0; i < (1 << 8); i++) {
        if (__builtin_popcount(i) == 4) mp[i] = mp.size();
    }
    adj = mat(mp.size(), row(mp.size()));
    for (int i = 0; i < (1 << 8); i++) {
        if (__builtin_popcount(i) == 4) {
            int u = mp[i];
            auto arr = get_next(i);
            for (auto ele:arr) {
                adj[u][mp[ele]] += 1;
            }
        }
    }
    adj ^ (1LL << 33);
    for (int cas = 1; cas <= t; cas++) {
//        cout << "Case " << cas << ": ";
        test();
    }
}
```

Strings Algorithms

```cpp
struct hashing {
    ll BASE = 197ll;
    ll MOD = 2000000011ll;
    ll BASE_INV = 1283018875ll;  // pow(BASE, MOD-2)%MOD
    ll base_pow[N] = {0};//max string size

    explicit hashing(ll _mod = 2000000011ll, ll base = 197ll) {
        this->MOD = _mod;
        this->BASE = base;
        base_pow[0] = 1;
        for (int i = 1; i < N; i++) base_pow[i] = (base_pow[i - 1] * BASE) % MOD;
    }
    //string abcd with length l is represented as
    //a*base^(l-1) + b *base^(l-2) + c*base^(1) + d*base^0
    ll remove_at(ll code, int idx, int val) {
        //to remove left char from string with length L use idx = L-1
        //to remove right char use shiftRight
        return (code - (val * base_pow[idx]) % MOD + MOD) % MOD;
    }

    ll addAt(ll code, int idx, int val) {
        //to add char to the left first shiftLeft than add using idx = 0
        //to add char to the right add directly using idx = L where L is the old
string Length
        return (code + (val * base_pow[idx]) % MOD) % MOD;
    }

    ll shift_left(ll code) {
        return (code * BASE) % MOD;
    }

    ll shift_right(ll code) {
        //delete right char from string
        return (code * BASE_INV) % MOD;
    }
};
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

const int N = 300 + 2;
const double OO = 1e18;

vector<int> suffix_array(string s) {
    s += '$';
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }

    int k = 0;
    while ((1 << k) < n) {
        vector<pair<int, int>> a(n);
        for (int i = 0; i < n; i++)a[i] = {c[i], c[(i + (1 << k)) % n]};
        sort(suff.begin(), suff.end(), [&a](int i, int j) { return a[i] < a[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (a[curr] != a[prev]);
        }
        k++;
    }
    return suff;
}

bool bs(vector<int> &suff, string &s, string &sub_string) {
    int p = sub_string.size(), n = s.size();
    int l = 0, r = suff.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        string tmp = s.substr(suff[mid], min(n - suff[mid], p));
        if (tmp == sub_string)return true;
        if (tmp < sub_string) {
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return false;
}

int upper_bound(vector<int> &suff, string &s, string &sub_string) {
    int p = sub_string.size(), n = s.size();
    int l = 0, r = suff.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        string tmp = s.substr(suff[mid], min(n - suff[mid], p));
        if (tmp <= sub_string)l = mid + 1;
        else {
            if (l == r)break;
            r = mid;
        }
    }
    return l;
}
```

```cpp
int lower_bound(vector<int> &suff, string &s, string &sub_string) {
    int p = sub_string.size(), n = s.size();
    int l = 0, r = suff.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        string tmp = s.substr(suff[mid], min(n - suff[mid], p));
        if (tmp < sub_string) {
            l = mid + 1;
        } else {
            if (l == r)break;
            r = mid;
        }
    }
    return l;
}

int main() {
    string s;
    cin >> s;
    vector<int> ans = suffix_array(s);
    int q;
    cin >> q;
    while (q--) {
        string tmp;
        cin >> tmp;
        int a = upper_bound(ans, s, tmp);
        int b = lower_bound(ans, s, tmp);
        cout << a - b << endl;
    }
}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
#define ii pair<int,int>
#define pll pair<ll,ll>
using namespace std;

const int N = 1e5 + 9;
ll fast_power(ll base, ll power, ll mod) {
    ll ret = 1;
    while (power) {
        if (power % 2)ret = (ret * base) % mod;
        base = (base * base) % mod;
        power /= 2;
    }
    return ret;
}


struct hashing {
    ll BASE, MOD, BASE_INV;
    ll base_pow[N] = {0};//max string size
    ll inv[N] = {0};

    explicit hashing(ll _mod = 2000000011ll, ll base = 197ll) {
        this->MOD = _mod;
        this->BASE = base;
        this->BASE_INV = fast_power(BASE, MOD - 2, MOD);
        base_pow[0] = 1;
        inv[0] = BASE_INV;
        for (int i = 1; i < N; i++) {
            base_pow[i] = (base_pow[i - 1] * BASE) % MOD;
//            inv[i] = fast_power(base_pow[i], MOD - 2, MOD);
        }
    }
    //string abcd with length l is represented as
    //a*base^(l-1) + b *base^(l-2) + c*base^(1) + d*base^0
    ll remove_at(ll code, int idx, int val) {
        //to remove left char from string with length L use idx = L-1
        //to remove right char use shiftRight
        return (code - (val * base_pow[idx]) % MOD + MOD) % MOD;
    }

    ll addAt(ll code, int idx, int val) {
        //to add char to the left first shiftLeft than add using idx = 0
        //to add char to the right add directly using idx = L where L is the old
string Length
        return (code + (val * base_pow[idx]) % MOD) % MOD;
    }

    ll shift_left(ll code) {
        return (code * BASE) % MOD;
    }

    ll shift_right(ll code) {
        //delete right char from string
        return (code * BASE_INV) % MOD;
    }
};

struct double_hashing {
    hashing h = hashing();
    hashing h2 = hashing(1e9 + 7LL, 53LL);

    pll remove_at(pll code, int idx, int val) {
        return {h.remove_at(code.first, idx, val), h2.remove_at(code.second, idx,
val)};
    }
```

```
    pll addAt(pll code, int idx, int val) {
        return {h.addAt(code.first, idx, val), h2.addAt(code.second, idx, val)};
    }

    pll shift_left(pll code) {
        return {h.shift_left(code.first), h2.shift_left(code.second)};
    }

    pll shift_right(pll code) {
        return {h.shift_right(code.first), h2.shift_right(code.second)};
    }

    ll rabin_karp(string const &pattern, string const &text, int P) {
        pll pat_code;
//        int P = pattern.size();
        for (int i = 0; i < P; i++) {
            pat_code = shift_left(pat_code);
            pat_code = addAt(pat_code, 0, pattern[i]);
        }
        pll sub_code;
        ll ans = 0;
        for (int i = 0; i < text.size(); i++) {
            if (i >= P) {
                sub_code = remove_at(sub_code, P - 1, text[i - P]);
            }
            sub_code = shift_left(sub_code);
            sub_code = addAt(sub_code, 0, text[i]);
            ans += (pat_code == sub_code);
        }
        return ans;
    }
};



//other version

struct hashing {
    vector<ll> pow;
    ll BASE, MOD, BASE_INV;

    hashing(int mxLen, ll _mod = 1e9 + 7LL, ll base = 27) : pow(mxLen),
BASE(base), MOD(_mod) {
        pow[0] = 1;
        BASE_INV = fast_power(base, _mod - 2, mod);
        for (int i = 1; i < mxLen; i++)
            pow[i] = (pow[i - 1] * BASE) % MOD;
    }

    ll push_front(ll code, int val) {
        return ((code * BASE) + val) % MOD;
    }

    ll push_back(ll code, int val, int new_string_len) {
        return (code + val * pow[new_string_len - 1]) % MOD;
    }

    ll pop_back(ll code, int val, int new_string_len) {
        return (((code - val * pow[new_string_len]) % MOD) + MOD) % MOD;
    }

    ll pop_front(ll code, int val) {
        code = (((code - val) % MOD) + MOD) % MOD;
        return (code * BASE_INV) % MOD;
    }
};

typedef pair<ll, ll> hash_c;
```

```
struct doubleHashing {
    hashing h1, h2;

    doubleHashing(int mxLen, ll mod1 = 1e9 + 7LL, ll mod2 = 1e9 + 9LL, ll base1 =
27, ll base2 = 54)
            : h1(mxLen, mod1, base1), h2(mxLen, mod2, base2) {}

    hash_c push_front(hash_c code, int val) {
        return {h1.push_front(code.first, val), h2.push_front(code.second, val)};
    }

    hash_c push_back(hash_c code, int val, int new_string_len) {
        return {h1.push_back(code.first, val, new_string_len),
h2.push_back(code.second, val, new_string_len)};
    }

    hash_c pop_back(hash_c code, int val, int new_string_len) {
        return {h1.pop_back(code.first, val, new_string_len),
h2.pop_back(code.second, val, new_string_len)};
    }

    hash_c pop_front(hash_c code, int val) {
        return {h1.pop_front(code.first, val), h2.pop_front(code.second, val)};
    }

    template<typename Iterator>
    hash_c hash_it(Iterator begin, Iterator end) {
        hash_c ret;
        for (auto it = begin; it != end; it++) ret = push_back(ret, *it, (it -
begin) + 1);
        return ret;
    }
};
```

```cpp
//
// Created by mohaned on ٢٠٢٠/٣/٢٧.
//

#include <bits/stdc++.h>
using namespace std;

#define all(v)                  ((v).begin()), ((v).end())
#define sz(v)                   ((int)((v).size()))
#define clr(v, d)               memset(v, d, sizeof(v))
#define rep(i, v)               for(int i=0;i<sz(v);++i)
#define lp(i, n)                for(int i=0;i<(int)(n);++i)
#define lpi(i, j, n)    for(int i=(j);i<(int)(n);++i)
#define lpd(i, j, n)    for(int i=(j);i>=(int)(n);--i)
typedef long long ll;

/*
 Another base, in case needed.

 #define BASE 53ll
 #define BASE_INV 682323657ll    // pow(BASE, MOD-2)%MOD
 System.out.println(BigInteger.probablePrime(35, new Random()));

 (MOD-1)^2 and (MOD-1)*BASE_INV MUST fit in your data type(E.g. ll)


 bases 33, 37, 39, or 41 are pretty good for dictionary strings (< 7 collisions)
 */

#define MOD 2000000011ll
#define BASE 53ll
#define BASE_INV 1283018875ll   // pow(BASE, MOD-2)%MOD

ll fastpow(ll num, ll p) {
    if (p == 0)
        return 1;
    if (p % 2)
        return (num % MOD * fastpow(num, p - 1)) % MOD;
    ll a = fastpow(num, p / 2);
    return (a * a) % MOD;
}

ll removeAt(ll code, int idx, int val) {
    return (code - (val * fastpow(BASE, idx)) % MOD + MOD) % MOD;
}

ll addAt(ll code, int idx, int val) {
    return (code + (val * fastpow(BASE, idx)) % MOD) % MOD;
}

ll shiftLeft(ll code) {
    return (code * BASE) % MOD;
}

ll shiftRight(ll code) {
    return (code * BASE_INV) % MOD;
}

////////////////////////////////////////////////////////////////////////
// better reduce chars values by reindexing
int val(char c) {
    return c;   // no reduction
    return 1 + islower(c) ? c - 'a' : c - 'A' + 26;
}

ll getHashValue1(string pat) {
    ll patCode = 0;
    for (int i = 0; i < (int) pat.size(); ++i) {
        patCode = patCode * BASE;  // left shift
```

```
        patCode += val(pat[i]);    // add value
        patCode %= MOD;            // mod to avoid overflow
    }
    return patCode;
}

ll getHashValue2(string pat) {
    ll patCode = 0;
    for (int i = 0; i < (int) pat.size(); ++i) {
        patCode = shiftLeft(patCode);
        patCode = addAt(patCode, 0, val(pat[i]));
    }
    return patCode;
}

////////////////////////////////////////////////////////////////////////////

void pattern_search(string main, string pat) {
    int n = pat.size();
    ll patCode = 0;
    for (int i = 0; i < (int) pat.size(); ++i) {
        patCode = shiftLeft(patCode);
        patCode = addAt(patCode, 0, val(pat[i]));
    }

    ll subCode = 0;
    string subStr;
    for (int i = 0; i < (int) main.size(); ++i) {
        if (i - n >= 0) {
            subCode = removeAt(subCode, n - 1, val(main[i - n]));
            subStr.erase(subStr.end() - 1);
        }
        subCode = shiftLeft(subCode);
        subCode = addAt(subCode, 0, val(main[i]));
        subStr.insert(subStr.begin(), main[i]);
        if (patCode == subCode)
            cout << subCode << "\t" << subStr << "\n";
    }
}

// return lowest suffix index that is palindrome
int longestSuffixPalindrome(string str) {
    int n = str.size(), longestSuffix = 0;
    ll strCode = 0, strRevCode = 0;

    // start from end, find longest suffix = reverse of suffix
    for (int i = n - 1, len = 0; i >= 0; --i, ++len) {
        strCode = shiftLeft(strCode);
        strCode = addAt(strCode, 0, str[i]);
        //Note, next is not so efficient, as we compute pow each step
        strRevCode = addAt(strRevCode, len, str[i]);

        if (strCode == strRevCode)
            longestSuffix = n - i;
    }
    return longestSuffix;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("test.txt", "rt", stdin);
#endif

    string str;
    while (getline(cin, str)) {

        int longestSuffix = longestSuffixPalindrome(str);

        cout<<str;
```

```
            // print the first few letters reversed
            for (int i = (str.size() - longestSuffix) - 1; i >= 0; i--)
                cout<<str[i];
            cout<<"\n";
        }
        return 0;
    }
```

```cpp
#include <bits/stdc++.h>

#define ll long long
#define ld long double
#define X first
#define Y second
using namespace std;
int const N = 5e4 + 5;
vector<int> adj[N][2];
int sz[N][2], tin[N][2], tout[N][2];
char ch[N][2];
string s[2];

void dfs(int node, int p, int cur) {
    tin[node][cur] = s[cur].size();

    s[cur].push_back('(');
    s[cur].push_back(ch[node][cur]);

    sz[node][cur] = 1;
    for (auto child:adj[node][cur]) {
        if (p == child)continue;
        dfs(child, node, cur);
        sz[node][cur] += sz[child][cur];
    }
    tout[node][cur] = s[cur].size();
    s[cur].push_back(')');
}

const int cnt_hash = 2;

ll hash1[N][cnt_hash], hash2[N][cnt_hash], mod[cnt_hash] = {static_cast<long
long>(1e9 + 7LL),
                                                           static_cast<long
long>(1e9 + 9LL)}, base[cnt_hash] = {201,

157};
ll basePowers[N][cnt_hash];


void pre() {
    for (int j = 0; j < cnt_hash; j++) {
        basePowers[0][j] = 1;
        for (int i = 1; i < N; i++)basePowers[i][j] = (base[j] * basePowers[i - 1]
[j]) % mod[j];
    }
}

void init(int n) {
    for (int j = 0; j < 2; j++)for (int i = 0; i < n; i++)adj[i][j].clear(), sz[i]
[j] = 0;
    s[0].clear(), s[1].clear();
}


void test() {
    int n, m;
    cin >> n >> m;
    init(n);
    for (int i = 0; i < n; i++) {
        char a;
        int prt;
        cin >> a >> prt;
        ch[i][0] = a;
        if (prt == -1) continue;
        adj[prt][0].push_back(i);

    }
    for (int i = 0; i < m; i++) {
```

```
        char a;
        int prt;
        cin >> a >> prt;
        ch[i][1] = a;
        if (prt == -1)continue;
        adj[prt][1].push_back(i);
    }
    dfs(0, -1, 0);
    dfs(0, -1, 1);

    for (int j = 0; j < cnt_hash; j++) {
        ll p = 1;
        for (int i = 0; i < s[0].size(); i++) {
            if (i)hash1[i][j] = hash1[i - 1][j];
            hash1[i][j] = (hash1[i][j] + (s[0][i] * p) % mod[j]) % mod[j];
            p = (p * base[j]) % mod[j];
        }
        p = 1;
        for (int i = 0; i < s[1].size(); i++) {
            if (i)hash2[i][j] = hash2[i - 1][j];
            hash2[i][j] = (hash2[i][j] + (s[1][i] * p) % mod[j]) % mod[j];
            p = (p * base[j]) % mod[j];
        }
    }
    set<vector<ll>> tree1;
    for (int i = 0; i < n; i++) {
        int st = tin[i][0], en = tout[i][0], siz = sz[i][0], len = en - st + 1;
        vector<ll> hh(cnt_hash);
        for (int j = 0; j < cnt_hash; j++) {
            hh[j] = ((((hash1[en][j] - (st == 0 ? 0 : hash1[st - 1][j])) % mod[j])
+ mod[j]) % mod[j]);
            hh[j] = (basePowers[N - st - len + 1][j] * hh[j]) % mod[j];
        }
        tree1.insert(hh);
    }
    int ans = 0;
    for (int i = 0; i < n; i++) {
        int st = tin[i][1], en = tout[i][1], siz = sz[i][1], len = en - st + 1;
        vector<ll> hh(cnt_hash);
        for (int j = 0; j < cnt_hash; j++) {
            hh[j] = ((((hash2[en][j] - (st == 0 ? 0 : hash2[st - 1][j])) % mod[j])
+ mod[j]) % mod[j]);
            hh[j] = (basePowers[N - st - len + 1][j] * hh[j]) % mod[j];
        }
        if (tree1.count(hh))ans = max(ans, siz);
    }
    cout << ans << '\n';
}

int main() {
    pre();
    int t = 1;
    cin >> t;
    for (int i = 1; i <= t; i++) {
        test();
    }
}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

vector<int> lcp;

void count_sort(vector<int> &suff, vector<int> &c) {
    int n = suff.size();
    vector<int> cnt(n), pos(n), suff_new(n);
    for (auto x:c)cnt[x]++;
    pos[0] = 0;
    for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];
    for (auto x:suff) {
        int cost = c[x];
        suff_new[pos[cost]] = x;
        pos[cost]++;
    }
    suff = suff_new;
}

vector<int> suffix_array(string &s) {
    s += '$';//empty suffix
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }
    int k = 0;
    while ((1 << k) < n) {
        // k -> k+1
        for (int &ele:suff)ele = (ele - (1 << k) + n) % n;
        count_sort(suff, c);

        c[suff[0]] = 0;
        vector<int> c_new(n);
        for (int i = 1; i < n; i++) {
            auto curr = make_pair(c[suff[i]], c[(suff[i] + (1 << k)) % n]);
            auto prev = make_pair(c[suff[i - 1]], c[(suff[i - 1] + (1 << k)) % n]);
            c_new[suff[i]] = c_new[suff[i - 1]] + (curr != prev);
        }
        k++;
        c = c_new;
    }

    //lcp of adjacent strings
    k = 0;
    lcp.resize(n - 1);
    for (int i = 0; i < n - 1; i++) {
        int pi = c[i];//pos of i in suffix array
        int j = suff[pi - 1];//previous suffix before suffix i in suffix array
        while (s[i + k] == s[j + k])k++;
        lcp[pi - 1] = k;
        k = max(0, k - 1);
    }
    return suff;
}


int main() {
    string s;
    cin >> s;
```

```
    vector<int> ans = suffix_array(s);
    for (int i = 0; i < ans.size(); i++) printf("%d%c", ans[i], " \n"[i + 1 ==
ans.size()]);
    for (int i = 0; i < lcp.size(); i++) printf("%d%c", lcp[i], " \n"[i + 1 ==
lcp.size()]);

}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

vector<int> lcp;

void count_sort(vector<int> &suff, vector<int> &c) {
    int n = suff.size();
    vector<int> cnt(n), pos(n), suff_new(n);
    for (auto x:c)cnt[x]++;
    pos[0] = 0;
    for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];
    for (auto x:suff) {
        int cost = c[x];
        suff_new[pos[cost]] = x;
        pos[cost]++;
    }
    suff = suff_new;
}

vector<int> suffix_array(string s) {
    s += '$';//empty suffix
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }
    int k = 0;
    while ((1 << k) < n) {
        // k -> k+1
        for (int &ele:suff)ele = (ele - (1 << k) + n) % n;
        count_sort(suff, c);

        c[suff[0]] = 0;
        vector<int> c_new(n);
        for (int i = 1; i < n; i++) {
            auto curr = make_pair(c[suff[i]], c[(suff[i] + (1 << k)) % n]);
            auto prev = make_pair(c[suff[i - 1]], c[(suff[i - 1] + (1 << k)) % n]);
            c_new[suff[i]] = c_new[suff[i - 1]] + (curr != prev);
        }
        k++;
        c = c_new;
    }

    //lcp of adjacent strings
    k = 0;
    lcp.resize(n);
    for (int i = 0; i < n - 1; i++) {
        int pi = c[i];//pos of i in suffix array
        int j = suff[pi - 1];//previous suffix before suffix i in suffix array
        while (s[i + k] == s[j + k])k++;
        lcp[pi] = k;// b7ot ans 3nd alrkm altany m4 al2wel y3ny lcp bta3 (3,4)
hktboh 3nd 4 // dymn mkan rkm 0 we mkan rkm 1 hyb2o equal 0
        k = max(0, k - 1);
    }
    return suff;
}


int main() {
    string s, s1, s2;
```

```
    cin >> s1 >> s2;
    s = s1 + "." + s2;
    vector<int> ans = suffix_array(s);
    int mx = 0, mxIndex = 0;
    for (int i = 1; i < lcp.size(); i++) {
        int t1 = (ans[i] < s1.size()), t2 = (ans[i - 1] < s1.size());
        if (t1 != t2 && lcp[i] > mx) {
            mx = lcp[i];
            mxIndex = ans[i];
        }
    }
    cout << s.substr(mxIndex, mx) << endl;
}
```

```cpp
string getStringAtIdx(const string &s, int idx, int ansLen) {
    string tmp;
    int n = s.size();
    for (int i = idx - ansLen + 1, counter = 0; counter < ansLen; counter++, i +=
2) {
        tmp.push_back(s[i]);
    }
    return tmp;
}

int mancherAlg(const string &s) {
    if (s.empty())
        return 0;

    string t = "#";
    for (auto &ch:s)t.push_back(ch), t.push_back('#');

    int n = t.size();
    vector<int> ans(n);

    int C = 0, R = 0;
    for (int i = 1; i < n - 1; i++) {
        int mirror = 2 * C - i;
        int rad = (i < R) ? min(ans[mirror], R - i) : 0;
        while (i + 1 + rad < n && i + 1 + rad > -1 && t[i + 1 + rad] == t[i - rad
- 1])
            rad++;

        ans[i] = rad;
        if (i + ans[i] > R) {
            C = i;
            R = i + ans[i];
        }
    }

    //uncomment to print longest palindromic substring
//    int idx = max_element(ans.begin(), ans.end()) - ans.begin();
//    return getStringAtIdx(t, idx, ans[idx]);
    return *max_element(ans.begin(), ans.end());
}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

vector<int> lcp;

void count_sort(vector<int> &suff, vector<int> &c) {
    int n = suff.size();
    vector<int> cnt(n), pos(n), suff_new(n);
    for (auto x:c)cnt[x]++;
    pos[0] = 0;
    for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];
    for (auto x:suff) {
        int cost = c[x];
        suff_new[pos[cost]] = x;
        pos[cost]++;
    }
    suff = suff_new;
}

vector<int> suffix_array(string &s) {
    s += '$';//empty suffix
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }
    int k = 0;
    while ((1 << k) < n) {
        // k -> k+1
        for (int &ele:suff)ele = (ele - (1 << k) + n) % n;
        count_sort(suff, c);

        c[suff[0]] = 0;
        vector<int> c_new(n);
        for (int i = 1; i < n; i++) {
            auto curr = make_pair(c[suff[i]], c[(suff[i] + (1 << k)) % n]);
            auto prev = make_pair(c[suff[i - 1]], c[(suff[i - 1] + (1 << k)) % n]);
            c_new[suff[i]] = c_new[suff[i - 1]] + (curr != prev);
        }
        k++;
        c = c_new;
    }

    //lcp of adjacent strings
    k = 0;
    lcp.resize(n - 1);
    for (int i = 0; i < n - 1; i++) {
        int pi = c[i];//pos of i in suffix array
        int j = suff[pi - 1];//previous suffix before suffix i in suffix array
        while (s[i + k] == s[j + k])k++;
        lcp[pi - 1] = k;
        k = max(0, k - 1);
    }
    return suff;
}


int main() {
    string s;
    cin >> s;
```

```
    vector<int> ans = suffix_array(s);
    int n = s.size() - 1;
    ll aa = 0;
    for (int i = 1; i < ans.size(); i++) {
        aa += (n - ans[i]) - lcp[i - 1];
    }
    printf("%lld\n", aa);
}
```

```cpp
//Problem: Given two strings - a pattern s and a text t,
//determine if the pattern appears in the text and if it does
//, enumerate all its occurrences in O(|s|+|t|) time.

//Algorithm: Calculate the hash for the pattern s. Calculate hash values for all
the prefixes of the text t.
//Now, we can compare a substring of length |s| with s in constant time using the
calculated hashes.
//So, compare each substring of length |s| with the pattern. This will take a
total of O(|t|) time.
//Hence the final complexity of the algorithm is O(|t|+|s|): O(|s|) is required
for calculating the hash of the pattern
//and O(|t|) for comparing each substring of length |s| with the pattern

vector<int> rabin_karp(string const& s, string const& t) {
    //pattern s and text t

    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

```cpp
struct hashing {
    const ll BASE = 53ll;
    const ll MOD = 2000000011ll;
    const ll BASE_INV = 1283018875ll;  // pow(BASE, MOD-2)%MOD
    vector<ll> base_pow;

    explicit hashing(int mxLength) {
        base_pow.resize(mxLength + 2);
        base_pow[0] = 1;
        for (int i = 1; i <= mxLength; i++) base_pow[i] = (base_pow[i - 1] * BASE)
% MOD;
    }
    //string abcd with length l is represented as
    //a*base^(l-1) + b *base^(l-2) + c*base^(1) + d*base^0
    ll remove_at(ll code, int idx, int val) {
        //to remove left char from string with length L use idx = L-1
        //to remove right char use shiftRight
        return (code - (val * base_pow[idx]) % MOD + MOD) % MOD;
    }

    ll addAt(ll code, int idx, int val) {
        //to add char to the left first shiftLeft than add using idx = 0
        //to add char to the right add directly using idx = L where L is the old
string Length
        return (code + (val * base_pow[idx]) % MOD) % MOD;
    }

    ll shift_left(ll code) {
        return (code * BASE) % MOD;
    }

    ll shift_right(ll code) {
        //delete right char from string
        return (code * BASE_INV) % MOD;
    }

    bool rabin_karp(string const &p, string const &t) {
        //search for pattern s int text t
        int P = p.size(), T = t.size();
        ll pattern_h = 0;
        for (int i = 0; i < P; i++) {
            pattern_h = shift_left(pattern_h);
            pattern_h = addAt(pattern_h, 0, p[i] - 'a' + 1);
        }

        ll text_h = 0;
        for (int i = 0; i < T; i++) {
            if (i >= P) {
                text_h = remove_at(text_h, i - 1, t[i - P] - 'a' + 1);
            }
            text_h = shift_left(text_h);
            text_h = addAt(text_h, 0, t[i] - 'a' + 1);
            if (text_h == pattern_h)return true;
        }
        return false;
    }

    bool rabin_karp(ll pattern_h, int pattern_size, string const &t) {
        // given hash of string s and len of s return true if it occurred int
string t
        int P = pattern_size, T = t.size();
        ll text_h = 0;
        for (int i = 0; i < T; i++) {
            if (i >= P) {
                text_h = remove_at(text_h, P - 1, t[i - P] - 'a' + 1);
            }
            text_h = shift_left(text_h);
            text_h = addAt(text_h, 0, t[i] - 'a' + 1);
            if (text_h == pattern_h)return true;
```

```
            }
            return false;
        }
    };
```

```cpp
#include <bits/stdc++.h>

#define ll long long
    using namespace std;

vector<int> lcp;

void count_sort(vector<int> &suff, vector<int> &c) {
    int n = suff.size();
    vector<int> cnt(n), pos(n), suff_new(n);
    for (auto x:c)cnt[x]++;
    pos[0] = 0;
    for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];
    for (auto x:suff) {
        int cost = c[x];
        suff_new[pos[cost]] = x;
        pos[cost]++;
    }
    suff = suff_new;
}

vector<int> suffix_array(string s) {
    s += char(32);//empty suffix
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }
    int k = 0;
    while ((1 << k) < n) {
        // k -> k+1
        for (int &ele:suff)ele = (ele - (1 << k) + n) % n;
        count_sort(suff, c);

        c[suff[0]] = 0;
        vector<int> c_new(n);
        for (int i = 1; i < n; i++) {
            auto curr = make_pair(c[suff[i]], c[(suff[i] + (1 << k)) % n]);
            auto prev = make_pair(c[suff[i - 1]], c[(suff[i - 1] + (1 << k)) % n]);
            c_new[suff[i]] = c_new[suff[i - 1]] + (curr != prev);
        }
        k++;
        c = c_new;
    }

    //lcp of adjacent strings
    k = 0;
    lcp.resize(n);
    for (int i = 0; i < n - 1; i++) {
        int pi = c[i];//pos of i in suffix array
        int j = suff[pi - 1];//previous suffix before suffix i in suffix array
        while (s[i + k] == s[j + k])k++;
        lcp[pi - 1] = k;
        k = max(0, k - 1);
    }
    return suff;
}

const int MAXN = 4e5 + 9, K = 19;
int st[MAXN][K + 1];
int logg[MAXN + 1];
```

```
void build_sparse_table(vector<int> &arr) {
    for (int i = 0; i < arr.size(); i++)
        st[i][0] = arr[i];
    for (int j = 1; j <= K; j++)
        for (int i = 0; i + (1 << j) <= arr.size(); i++)
            st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);

    logg[1] = 0;
    for (int i = 2; i <= MAXN; i++)
        logg[i] = logg[i / 2] + 1;
}

int rmq(int L, int R) {
    int j = logg[R - L + 1];
    return min(st[L][j], st[R - (1 << j) + 1][j]);
}

int longest_common_prefix(int i, int j) {
    if (i > j)swap(i, j);
    return rmq(i, j - 1);
}

int random(int mn, int mx) {
    return (rand() % (mx + 1 - mn)) + mn;
}

char arr[MAXN];

int main() {
    string s;
    int n;
    scanf("%s%d", arr, &n);
    s = string(arr);
    vector<int> ans = suffix_array(s);
    vector<int> pos(ans.size());
    for (int i = 0; i < ans.size(); i++) {
        pos[ans[i]] = i;
    }
    build_sparse_table(lcp);
    vector <pair<int, int>> v(n);
    //v[i] = L ,R of substring
    for (int i = 0; i < n; i++)scanf("%d%d", &v[i].first, &v[i].second);
    sort(v.begin(), v.end(), [&](pair<int, int> lf, pair<int, int> rt) {
        int l1 = lf.second - lf.first + 1, l2 = rt.second - rt.first + 1;
        if (lf.first == rt.first)return lf.second < rt.second;
        if (longest_common_prefix(pos[lf.first - 1], pos[rt.first - 1]) >= min(l1,
l2)) {
            if (l1 == l2)
                return lf < rt;
            else return l1 < l2;
        } else
            return pos[lf.first - 1] < pos[rt.first - 1];
    });

    for (int j = 0; j < n; j++) printf("%d %d\n", v[j].first, v[j].second);


}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

const int N = 300 + 2;
const double OO = 1e18;

vector<int> suffix_array(string s) {
    s += '$';
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }

    int k = 0;
    while ((1 << k) < n) {
        vector <pair<int, int>> a(n);
        for (int i = 0; i < n; i++)a[i] = {c[i], c[(i + (1 << k)) % n]};
        sort(suff.begin(), suff.end(), [&a](int i, int j) { return a[i] < a[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (a[curr] != a[prev]);
        }
        k++;
    }
    return suff;
}

bool bs(vector<int> &suff, string &s, string &sub_string) {
    int p = sub_string.size(), n = s.size();
    int l = 0, r = suff.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        string tmp = s.substr(suff[mid], min(n - suff[mid], p));
        if (tmp == sub_string)return true;
        if (tmp < sub_string) {
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return false;
}

int main() {
    string s;
    cin >> s;
    vector<int> ans = suffix_array(s);
    int q;
    cin >> q;
    while (q--) {
        string tmp;
        cin >> tmp;
        puts(bs(ans, s, tmp) ? "Yes" : "No");

    }
}
```

```cpp
#include <bits/stdc++.h>

#define ll long long
using namespace std;

const int N = 300 + 2;
const double OO = 1e18;

void count_sort(vector<int> &suff, vector<int> &c) {
    int n = suff.size();
    vector<int> cnt(n), pos(n), suff_new(n);
    for (auto x:c)cnt[x]++;
    pos[0] = 0;
    for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];
    for (auto x:suff) {
        int cost = c[x];
        suff_new[pos[cost]] = x;
        pos[cost]++;
    }
    suff = suff_new;
}

vector<int> suffix_array(string &s) {
    s += '$';//empty suffix
    int n = s.size();
    vector<int> suff(n), c(n);
    {
        //k =0
        iota(suff.begin(), suff.end(), 0);
        sort(suff.begin(), suff.end(), [&s](int i, int j) { return s[i] < s[j]; });
        c[suff[0]] = 0;
        for (int i = 1; i < n; i++) {
            int curr = suff[i], prev = suff[i - 1];
            c[curr] = c[prev] + (s[curr] != s[prev]);
        }
    }

    int k = 0;
    while ((1 << k) < n) {
        // k -> k+1
        for (int &ele:suff)ele = (ele - (1 << k) + n) % n;
        count_sort(suff, c);

        c[suff[0]] = 0;
        vector<int> c_new(n);
        for (int i = 1; i < n; i++) {
            auto curr = make_pair(c[suff[i]], c[(suff[i] + (1 << k)) % n]);
            auto prev = make_pair(c[suff[i - 1]], c[(suff[i - 1] + (1 << k)) % n]);
            c_new[suff[i]] = c_new[suff[i - 1]] + (curr != prev);
        }
        k++;
        c = c_new;
    }
    return suff;
}

int main() {
    string s;
    cin >> s;
    vector<int> ans = suffix_array(s);
    for (auto i:ans) printf("%d%c", i, " \n"[i == ans.back()]);
}
```

```cpp
struct trie {
    struct node {
        map<char, int> nxt;
        int isEnd = 0, count = 0;
        pair<int, int> dp = {-1, -1};
    };
    vector<node> t;

    trie() { t.emplace_back(); }

    void add(string &s, int k) {
        int curNode = 0;
        for (char ch : s) {
            if (!t[curNode].nxt.count(ch)) {
                t[curNode].nxt[ch] = t.size();
                t.emplace_back();
            }
            curNode = t[curNode].nxt[ch];
            t[curNode].dp = {-1, -1};
            t[curNode].count++;
        }
        t[curNode].isEnd = k;
    }

    void remove(string &s) {
        int curNode = 0, prev = 0;
        for (char ch : s) {
            prev = curNode;
            curNode = t[curNode].nxt[ch];
            t[curNode].dp = {-1, -1};
            if (!--t[curNode].count)t[prev].nxt.erase(ch);
        }
        t[curNode].isEnd = 0;
    }

    pair<int, int> dfs(int node) {
        int mnLen = (1 << 30), mnInd = 0;
        if (t[node].isEnd)return {t[node].isEnd, 0};
        if (t[node].dp != make_pair(-1, -1))return t[node].dp;
        for (auto &child:t[node].nxt) {
            auto p = dfs(child.second);
            if (p.second < mnLen)mnLen = p.second, mnInd = p.first;
        }
        return t[node].dp = {mnInd, mnLen + 1};
    }

    int query(string &s) {
        int curNode = 0;
        for (char ch : s) {
            if (t[curNode].nxt.count(ch))
                curNode = t[curNode].nxt[ch];
            else
                return -1;
        }
        return dfs(curNode).first;
    }
};
```

```cpp
struct trie {
    struct node {
        map<char, int> adj;
        int cnt;
        bool isEnd;

        node() : cnt(0), isEnd(false) {}
    };

    vector<node> t;

    trie() { t.emplace_back(); }

    void add(string &s) {
        int idx = 0;
        t[idx].cnt++;
        for (char ch:s) {
            if (!t[idx].adj.count(ch))t[idx].adj[ch] = t.size(), t.emplace_back();
            idx = t[idx].adj[ch];
            t[idx].cnt++;
        }
        t[idx].isEnd = true;
    }

    ii dfs(string &s, int idx = 0, int node = 0) {
        if (idx == s.size())return {s.size(), 0};//lma aktb string kaml
        char ch = s[idx];
        int cntLessThanMe = t[node].isEnd;
        for (auto child:t[node].adj) {
            if (child.first == ch)break;
            int nxt = child.second;
            cntLessThanMe += t[nxt].cnt;
        }
        ii tmp = dfs(s, idx + 1, t[node].adj[ch]);
        cntLessThanMe += tmp.second;
        int cntGreaterThanMe = t[node].cnt - cntLessThanMe - 1;
        return {
                min({tmp.first, cntLessThanMe + idx, cntGreaterThanMe + 1 + idx}),
                cntLessThanMe
        };
    }
};
```

dmkozyrev's blog

[Tutorial] Rolling hash and 8 interesting problems [Editorial]

By dmkozyrev, history, 2 years ago, translation,

UPD: while I was translating this post from Russian to English, **dacin21** wrote his post, more advanced, link. I hope that my post will help beginners, but in my post more rough estimates.

And in Russia we call **rolling hashes** as a **polynomial hashes**.

Hello, codeforces! This blogpost is written for all those who want to understand and use polynomial hashes and learn how to apply them in solving various problems. I will briefly write the theoretical material, consider the features of the implementation and consider some problems, among them:

1. Searching all occurrences of one string of length n in another string length m in $O(n + m)$ time
2. Searching for the largest common substring of two strings of lengths n and m $(n \geq m)$ in $O((n + m \cdot log(n)) \cdot log(m))$ and $O(n \cdot log(m))$ time
3. Finding the lexicographically minimal cyclic shift of a string of length n in $O(n \cdot log(n))$ time
4. Sorting of all cyclic shifts of a string of length n in lexicographic order in $O(n \cdot log(n)^2)$ time
5. Finding the number of sub-palindromes of a string of length n in $O(n \cdot log(n))$ time
6. The number of substrings of string of length n that are cyclic shifts of the another string length m in $O((n + m) \cdot log(n))$ time
7. The number of suffixes of a string of length n, the infinite extension of which coincides with the infinite extension of the given string for $O(n \cdot log(n))$ (extension is a duplicate string an infinite number of times).
8. Largest common prefix of two strings length n with swapping two chars in one of them $O(n \cdot log(n))$

Note 1. It is possible that some problems can be solved more quickly by other methods, for example, sorting the cyclic shifts — this is exactly what happens when constructing a suffix array, to search for all occurrences of one string in another will allow the Knut-Morris-Pratt algorithm, the Manaker algorithm works well with the sub-palindromes, and for own suffixes there is a prefix function.

Note 2. In the problems above, an estimate is made when a hash search is performed by sorting and binary searching. If you have your own hash table with open mixing or overflow chains, then you — lucky, boldly replace the hash search for a search in your hash table, but do not try to use `std::unordered_set`, as in practice the search in `std::unordered_set` loses sorting and binary search in connection with the fact that this piece obeys the C++ standard and has to guarantee a lot to the user, which is useful for industrial coding and, often, is useless in the competitive programming, so sorting and binary search for simple structures gain absolute primacy in C++ in speed of work , if not used additional own structures.

Note 3. In cases where comparison of elements is slow (for example, comparison by hash in $O(log(n))$ time), in the worst case `std::random_shuffle` + `std::sort` always loses `std::stable_sort`, because `std::stable_sort` guarantees the minimum number of comparisons among all sorts (based on comparisons) for the worst case.

The solution of the listed tasks will be given below, the source codes also.

As **advantages of polynomial hashing** I can notice that you often do not need to think, you can immediately take and write a naive algorithm to solve the problem and speed it up with polynomial hashing. Personally, firstly, I think about solution with a polynomial hash, perhaps that's why I'm blue.

Among the **disadvantages of polynomial hashing**: a) Too many operations getting remainder from the integer division, sometimes on the border with TLE for large problems, and b) on the codeforces in C++ programs are often small guarantees against hacking due to MinGW: `std::random_device` generates the same number every time, `std::chrono::high_resolution_clock` ticks in microseconds instead of nanoseconds. (The Cygwin compiler for windows wins against MinGW).

`UPD`: Solved a)

`UPD`: Solved b)

What is polynomial hashing?

Hash-function must assign to the object a **certain value** (hash) and possess the following properties:

1. If two objects are equal, then their hashes are equal.
2. If two hashes are equal, then the objects are equal with a high probability.

A collision is the very unpleasant situation of equality of two hashes for not equal objects. Ideally, when you choose a hash function, you need to ensure that **probability of collision lowest of possibles**. In practice — just a probability to successfully pass a set of tests to the task.

There are **two approaches** in choosing a function of a polynomial hash that depend on **directions**: from left to right and from right to left. To begin with, consider the option from left to right, and below, after describing the problems that arise in connection with the choice of the first option, consider the second.

Consider the sequence $\{a_0, a_1, ..., a_{n-1}\}$. Under the polynomial hash from left to right for this sequence, we have in mind the result of calculating the following expression:

$$\text{hash}(a, p, m) = (a_0 + a_1 \cdot p + a_2 \cdot p^2 + ... + a_{n-1} \cdot p^{n-1}) \bmod m$$

Here p and m — point (or base) and a hash module, respectively.

The conditions that we will impose: $\max(a_i) < p < m, \gcd(p, m) = 1$.

Note. If you think about interpreting the expression, then we match the sequences $\{a_0, a_1, ..., a_{n-1}\}$ number of length n in the number in system with base p and take the remainder from its division by the number m, or the value of the polynomial $(n-1)$-th power with coefficients a_i at the point p modulo m. We'll talk about the choice of p and m later.

Note. If the value of $\text{hash}(a, p, m)$ (not by modulo), is placed in an integer data type (for example, a 64-bit type), then each sequence can be associated with this number. Then the comparison by greater / less / equal can be performed in $O(1)$ time.

Comparison by equal in $O(1)$ time

Now let's answer the question, how to compare arbitrary segments of sequence for $O(1)$? We show that to compare the segments of given sequence $\{a_0, a_1, ..., a_{n-1}\}$, it is sufficient to compute the polynomial hash on **each prefix** of the original sequence.

Define a polynomial hash on the prefix as:

$$\text{pref}(k, a, p, m) = (a_0 + a_1 \cdot p + a_2 \cdot p^2 + ... + a_{k-1} \cdot p^{k-1}) \bmod m$$

Briefly denote $\text{pref}(k, a, p, m)$ as $\text{pref}(k)$ and keep in mind that the final value is taken modulo m. Then:

$$\text{pref}(0) = 0,\ \text{pref}(1) = a_0,\ \text{pref}(2) = a_0 + a_1 \cdot p,\ \text{pref}(3) = a_0 + a_1 \cdot p + a_2 \cdot p^2$$

General form:

$$\text{pref}(n) = a_0 + a_1 \cdot p + a_2 \cdot p^2 + ... + a_{n-1} \cdot p^{n-1}$$

The polynomial hash on each prefix can be calculated in $O(n)$ time, using recurrence relations:

$$p^k = p^{k-1} \cdot p,\ \text{pref}(k+1) = \text{pref}(k) + a_k \cdot p^k$$

Let's say we need to compare two substrings that begin with i and j and have the length len, for equality:

$$a_i, a_{i+1}, ..., a_{i+len-1} =? \ a_j, a_{j+1}, ..., a_{j+len-1}$$

Consider the differences $\text{pref}(i+len) - \text{pref}(i)$ and $\text{pref}(j+len) - \text{pref}(j)$. It's not difficult to see that:

$$\begin{aligned}\text{pref}(i+len) - \text{pref}(i) &= a_i \cdot p^i + a_{i+1} \cdot p^{i+1} + ... + a_{i+len-1} \cdot p^{i+len-1} \\ \text{pref}(j+len) - \text{pref}(j) &= a_j \cdot p^j + a_{j+1} \cdot p^{j+1} + ... + a_{j+len-1} \cdot p^{j+len-1}\end{aligned}$$

We multiply the first equation by p^j, and the second by p^i. We get:

$$\begin{aligned}p^j \cdot (\text{pref}(i+len) - \text{pref}(i)) &= p^{i+j} \cdot (a_i + a_{i+1} \cdot p + ... + a_{i+len-1} \cdot p^{len-1}) \\ p^i \cdot (\text{pref}(j+len) - \text{pref}(j)) &= p^{i+j} \cdot (a_j + a_{j+1} \cdot p + ... + a_{j+len-1} \cdot p^{len-1})\end{aligned}$$

We see that on the right-hand side of the expressions in brackets polynomial hashes were obtained from the segments of sequence:

$$a_i, a_{i+1}, ..., a_{i+len-1} \text{ and } a_j, a_{j+1}, ..., a_{j+len-1}$$

Thus, in order to determine whether the required segments of sequence have coincided, we need to check the following equality:

$$p^j \cdot (\text{pref}(i+len) - \text{pref}(i)) = p^i \cdot (\text{pref}(j+len) - \text{pref}(j))$$

One such comparison can be performed in $O(1)$ time, assuming the degree of p modulo precalculated. With the module m, we have:

$$p^j \cdot (\text{pref}(i+len) - \text{pref}(i)) \bmod m = p^i \cdot (\text{pref}(j+len) - \text{pref}(j)) \bmod m$$

Problem: Comparing one segment of sequence depends on the parameters of the other segment of sequence (from j).

The first solution of this problem (given by **veschii_nevstrui**) is based on multiplying the first equation by p^{-i}, and the second by p^{-j}. Then we get:

$$\begin{aligned}p^{-i} \cdot (\text{pref}(i+len) - \text{pref}(i)) &= a_i + a_{i+1} \cdot p + ... + a_{i+len-1} \cdot p^{len-1} \\ p^{-j} \cdot (\text{pref}(j+len) - \text{pref}(j)) &= a_j + a_{j+1} \cdot p + ... + a_{j+len-1} \cdot p^{len-1}\end{aligned}$$

We can see that in the right-hand parts of equations we get a polynomial hash from the needed segments of sequence. Then, the equality is checked as:

$$p^{-i} \cdot (\text{pref}(i+len) - \text{pref}(i)) = p^{-j} \cdot (\text{pref}(j+len) - \text{pref}(j))$$

To implement this, we need to find the inverse element for p modulo m. From the condition $gcd(p, m) = 1$, the inverse element always exists. To do this, we need calculate or just know the value of the Euler function for the selected module $\varphi(m)$ and get power $\varphi(m) - 1$ for p. If we precalculate the powers of the inverse element for the selected module, then the comparison can be performed in $O(1)$ time.

The second solution we can use if we know the maximum lengths of compared segments of sequences. Let's denote the maximum length of compared lines as mxPow. We multiply 1-th equation by power $mxPow - i - len + 1$ of p, and 2-nd equation by $mxPow - j - len + 1$ power of p. We get:

$$p^{mxPow-i-len+1} \cdot (\mathrm{pref}(i+len) - \mathrm{pref}(i)) = p^{mxPow-len+1} \cdot (a_i + a_{i+1} \cdot p + ... + a_{i+len-1} \cdot p^{len-1})$$

$$p^{mxPow-j-len+1} \cdot (\mathrm{pref}(j+len) - \mathrm{pref}(j)) = p^{mxPow-len+1} \cdot (a_j + a_{j+1} \cdot p + ... + a_{j+len-1} \cdot p^{len-1})$$

We can note that on the right-hand sides of equals a polynomial hash of segments of sequence. Then, the equality is checked as follows:

$$p^{mxPow-i-len+1} \cdot (\mathrm{pref}(i+len) - \mathrm{pref}(i)) \bmod m = p^{mxPow-j-len+1} \cdot (\mathrm{pref}(j+len) - \mathrm{pref}(j)) \bmod m$$

This approach allows you to compare **one substring** of length len with **all substrings** of length len by **equality**, including **substrings of another string**, since the expression $p^{mxPow-i-len+1} \cdot (\mathrm{pref}(i+len) - \mathrm{pref}(i)) \bmod m$ for the substring of the length len starting at the position i, depends only on **the parameters of the current substring** i, len and **constant** $mxPow$, and not from the parameters of another substring.

Now consider **another approach** for choosing polynomial hash function. Define a polynomial hash on the prefix as:

$$\mathrm{pref}(k, a, p, m) = (a_0 \cdot p^{k-1} + a_1 \cdot p^{k-2} + ... + a_{k-2} \cdot p + a_{k-1}) \bmod m$$

Briefly denote $\mathrm{pref}(k, a, p, m)$ as $\mathrm{pref}(k)$ and keep in mind that the final value is taken modulo m. Then:

$$\mathrm{pref}(0) = 0,\ \mathrm{pref}(1) = a_0,\ \mathrm{pref}(2) = a_0 \cdot p + a_1,\ \mathrm{pref}(3) = a_0 \cdot p^2 + a_1 \cdot p + a_2$$

The polynomial hash on each prefix can be calculated in $O(n)$ time, using recurrence relations:

$$p^k = p^{k-1} \cdot p,\ \mathrm{pref}(k+1) = \mathrm{pref}(k) \cdot p + a_k$$

Let's say we need to compare two substrings that begin with i and j and have the length len, for equality:

$$a_i, a_{i+1}, ..., a_{i+len-1} =? a_j, a_{j+1}, ..., a_{j+len-1}$$

Consider the differences $\mathrm{pref}(i+len) - \mathrm{pref}(i) \cdot p^{len}$ and $\mathrm{pref}(j+len) - \mathrm{pref}(j) \cdot p^{len}$. It's not difficult to see that:

$$\mathrm{pref}(i+len) - \mathrm{pref}(i) \cdot p^{len} = a_i \cdot p^{len-1} + ... + a_{i+len-2} \cdot p + a_{i+len-1}$$
$$\mathrm{pref}(j+len) - \mathrm{pref}(j) \cdot p^{len} = a_j \cdot p^{len-1} + ... + a_{j+len-2} \cdot p + a_{j+len-1}$$

We see that on the right-hand side of the expressions in brackets polynomial hashes were obtained from the segments of sequence:

$a_i, a_{i+1}, ..., a_{i+len-1}$ и $a_j, a_{j+1}, ..., a_{j+len-1}$

Thus, in order to determine whether the required segments of sequence have coincided, we need to check the following equality:

$$(\text{pref}(i+len) - \text{pref}(i) \cdot p^{len}) \bmod m = (\text{pref}(j+len) - \text{pref}(j) \cdot p^{len}) \bmod m$$

One such comparison can be performed in $O(1)$ time, assuming the degree of p modulo m precalculated.

Comparison by greater / less in $O(log(n))$ time

Consider two substrings of (possibly) different strings of lengths $len1$ and $len2$, ($len1 \le len2$), starting in the positions i and j respectively. Note that the ratio greater / less is determined **by the first non-equal symbol** in these substrings, and before this position strings are equal. Thus, we need to find the **position of the first non-equal symbol** by the **binary search method**, and then compare the found symbols. By comparing substrings to equality in $O(1)$ time, we can solve the problem of comparing substrings by greater / less in $O(log(len1))$ time:

Pseudocode

Minimizing the probability of collision

Using approximations in birthday problem, we get (perhaps a rough) estimate of the probability of collision. Suppose we compute a polynomial hash modulo m and, during the program, we need to compare n strings. Then the probability that the collision will occur:

$$p \approx 1 - \exp\left(-\frac{n^2}{2m}\right)$$

Hence it is obvious that m needs to be taken much more than n^2. Then, approximating the exponential as Taylor series, we get the probability of collision on one test:

$$p \approx 1 - \exp\left(-\frac{n^2}{2m}\right) \approx \frac{n^2}{2m}$$

If we look at the problem of searching of occurrences of all cyclic shifts of one row in another string of lengths to 10^5, then we can get 10^{15} comparisons of strings.

If we take a prime modulo of the order 10^9, then we will not go through any of the maximum tests.

If we take a module of the order 10^{18}, then the probability of collision on one test is ≈ 0.001. If the maximum tests are 100, the probability of collision in one of the tests is ≈ 0.1, that is 10%.

If we take the module of the order 10^{27}, then on the 100 maximum tests the probability of collision is $\approx 10^{-10}$.

Conclusion: the higher the value of module, the more likely that we must pass the test. This probability not includes estimate probability of hack your solution.

Double polynomial hash

Of course, in real programs we can not take modules of the order 10^{27}. How to be? To the aid comes the Chinese theorem on the remainders. If we take two mutually simple modules m_1 and m_2, then the ring of residues modulo $m = m_1 \cdot m_2$ is equivalent to the product of rings modulo m_1 and m_2, i.e. there is biection between them, based on the idempotents of the residue ring modulo m. In other words, if you calculate hash_1 modulo m_1 and hash_2 modulo m_2, and then compare two segments of sequence with hash_1 and hash_2 simultaneously, then this is equivalent to comparing a polynomial hash modulo m. Similarly, we can take three mutually prime modules m_1, m_2, m_3.

Features of the implementation

So, we came to the implementation of the above. Goal — **the minimum of the number of the remainder calculation from the integer division**, i.e. get two multiplications in a 64-bit type and one take the remainder from division in 64-bit type for one calculation of a double polynomial hash, **get a hash modulo about 10^27** and **protect the code from hacking** on codeforces

Selection of modules. It is advantageous to use a double polynomial hash on the modules `m1 = 1000000123` and `m2 = 2^64`. If you do not like this choice of `m1`, you can select `1000000321` or `2^31-1`, the main thing is to choose such a prime number so that the difference of the two residues lies within the signed 32-bit type (int). A prime number is more convenient, since the conditions `gcd(m1, m2) = 1` and `gcd(m1, p) = 1` are automatically provided. The choice of `m2 = 2^64` is not accidental. The C++ standard ensures that all calculations in the `unsigned long long` are executed modulo `2^64` automatically. Separately, the module `2^64` can not be taken, because there is an anti-hash test, which does not depend on the choice of the hash point `p`. The module `m1` should be specified as **constant** to speed up the taking the remainder (the compiler (not MinGW) optimizes, replacing by multiplication and bitwise shifting).

Sequence encoding. If given a sequence of characters, consisting, for example, of small Latin letters, then you not need to encode anything, because each character already corresponds to its code. If a sequence of integers is given that is reasonable for a representation in memory of length, then it is possible to collect all the occurring numbers into one array, sort, delete the repeats and assign to each number in the sequence its index in sorted set. **Code zero is forbidden**: all sequences of the form `0,0,0, .., 0` **of different length** will have **the same** polynomial hash.

Choosing of the base. As the base `p` it suffices to take any odd number satisfying the condition `max(a[i]) < p < m1`. (odd, because then `gcd(p, 2 ^ 64) = 1`). If you **can be hacked**, then it is necessary to generate `p` randomly with every new program start, and generation with `std::srand(std::time(0))` and `std::rand()` is a bad idea, because `std::time(0)` ticks very slowly, and `std::rand()` does not provide enough uniformity. If the compiler **is not MINGW** (unfortunately, MinGW is installed on the codeforces), then you can use `std::random_device`, `std::mt19937`, `std::uniform_int_distribution<int>` (in cygwin on windows and gnu gcc on linux this set provides almost absolute randomness). If you were unlucky and you use only MinGW, then there is nothing left to do but replace `std::random_device` with `std::chrono::high_resolution_clock` and hope for the best (or is there a way to get some counter from the processor?). On MinGW, this timer ticks in microseconds, on cygwin and gnu gcc in nanoseconds.

Warranties against hacking. Odd numbers up to a module of the order of `10^9` are also of the order of `10^9`. The cracker will need to generate an anti-hash test for each odd number so that there is a collision in the space to `10^27`, compile all the tests into one big test and hack you! This is if you do not use MinGW on Windows. On MinGW, the timer is ticking, as already mentioned, in microseconds. Knowing **when solution was started** on the server with an accuracy of seconds, it is possible for each of the `10^6` microseconds to calculate what random `p` was generated, and then the number of variants in the `1000` times less. If `10^9` is some cosmic number, then `10^6` already seems not so safe. If you use `std::time (0)` only `10^3` number of variants (milliseconds) — can be hacked! In the comments I saw that grandmasters know how to break a polynomial hash modulo ot order of `10^36`.

Ease of use. It is convenient to write a **universal object** for a polynomial hash and **copy** it to the task where it might be needed. It is better to write independently for your needs and goals in the style in which you write to understand the source code if necessary. All problems in this post are solved by copying the one class object. It is possible that there are specific tasks in which this does not work.

UPD: To speed up programs, you can quickly calculate the remainder of the divisions by modules $2^{31} - 1$ and $2^{61} - 1$. The main difficulty is multiplication. To understand the principle, look at this post by **dacin21** in **Larger modulo** and his comment.

Mult mod `2^61-1`

Problem 1. Searching all occurrences of one string of length n in another string length m in $O(n + m)$ time

Problem 1 statement in English

Link on problen on acmp.ru.
Solution and code

Problem 2. Searching for the largest common substring of two strings of lengths n and m ($n \geq m$) in $O((n + m \cdot log(n)) \cdot log(m))$ and $O(n \cdot log(m))$ time

Link on problem on acm.timus.ru with length 10^5.

Link on problem on spoj.com with length 10^6.
Solution and code

Problem 3. Finding the lexicographically minimal cyclic shift of a string of length n in $O(n \cdot log(n))$ time
Problem 3 statement in English

Link on problem 3
Solution and code

Problem 4. Sorting of all cyclic shifts of a string of length n in lexicographic order in $O(n \cdot log(n)^2)$ time
Problem 4 statement in English

Link on problem 4
Note
Solution and code

Problem 5. Finding the number of sub-palindromes of a string of length n in $O(n \cdot log(n))$ time
Problem 5 statement in English

Link on problem 5 with length 10^5

Link on problem 5 with length 10^6
Solution and code

Problem 6. The number of substrings of string of length n that are cyclic shifts of the another string length m in $O((n + m) \cdot log(n))$ time
Problem 6 statement in English

Link on problem 6
Solution and code

Problem 7. The number of suffixes of a string of length n, the infinite extension of which coincides with the infinite extension of the given string for $O(n \cdot log(n))$ (extension is a duplicate string an infinite number of times).
Problem 7 statement in English

Link on problem 7
Solution and code

Problem 8. Largest common prefix of two strings length n with swapping two chars in one of them in $O(n \cdot log(n))$ time

Link on problem

You can check the editorial written by **a_kk** and **smokescreen** on this site for getting solution without hashes in `O(n)` time.
Solution and code

MISC

neal's blog

Don't use rand(): a guide to random number generators in C++

By **neal**, 2 years ago,

Don't use `rand()`. Why? Let's jump right into some code. What value will the following code print, approximately?

```
#include <cstdlib>
#include <iostream>
using namespace std;

const int ITERATIONS = 1e7;

int main() {
    double sum = 0;

    for (int i = 0; i < ITERATIONS; i++)
        sum += rand() % 1000000;

    cout << "Average value: " << sum / ITERATIONS << '\n';
}
```

Should be about 500,000, right? Turns out it depends on the compiler, and on Codeforces it prints 16382, which isn't even close. Try it out yourself.

What's happening here?

If you look up C++ documentation on `rand()`, you'll see that it returns "a pseudo-random integral value between `0` and `RAND_MAX`." Click again on `RAND_MAX` and you'll see that "This value is implementation dependent. It's guaranteed that this value is at least 32767." On the Codeforces machines, it turns out `RAND_MAX` is exactly 32767. That's so small!

It doesn't stop there though; `random_shuffle()` also uses `rand()`. Recall that in order to perform a random shuffle, we need to generate random indices up to n, the size of the array. But if `rand()` only goes up to 32767, what happens if we call `random_shuffle()` on an array with significantly more elements than that? Time for some more code. What would you expect the following code to print?

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

const int N = 3000000;

double average_distance(const vector<int> &permutation) {
    double distance_sum = 0;

    for (int i = 0; i < N; i++)
        distance_sum += abs(permutation[i] - i);

    return distance_sum / N;
```

```
}

int main() {
    vector<int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    random_shuffle(permutation.begin(), permutation.end());
    cout << average_distance(permutation) << '\n';
}
```

This computes the average distance that each value moves in the random shuffle. If you work out a bit of math, you'll find that the answer on a perfectly random shuffle should be $\frac{N}{3}$ = 1,000,000. Even if you don't want to do the math, you can observe that the answer is between $\frac{N}{2}$ = 1,500,000, the average distance for index 0, and $\frac{N}{4}$ = 750,000, the average distance for index $\frac{N}{2}$.

Well, once again the code above disappoints; it prints out 64463. Try it yourself. In other words, `random_shuffle()` moved each element a distance of 2% of the length of the array on average. Based on my testing, the implementation of `random_shuffle()` on Codeforces matches the following exactly:

```
    for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[rand() % (i + 1)]);
```

So naturally if `RAND_MAX` is much less than N, this shuffle will be problematic.

`rand()` itself has more quality problems than just `RAND_MAX` being small though; it is typically implemented as a relatively simple linear congruential generator. On the Codeforces compiler, it looks like this:

```
static long holdrand = 1L;

void srand(unsigned int seed) {
  holdrand = (long) seed;
}

int rand() {
  return (((holdrand = holdrand * 214013L + 2531011L) >> 16) & 0x7fff);
}
```

In particular, linear congruential generators (LCGs) suffer from extreme predictability in the lower bits. The k-th bit (starting from k = 0, the lowest bit) has a period of at most 2^{k+1} (i.e., how long until the sequence takes to repeat). So the lowest bit has a period of just **2**, the second lowest a period of 4, etc. This is why the function above discards the lowest 16 bits, and the resulting output is at most 32767.

What's the solution?

Don't worry, as of C++11 there are **much better** random number generators available in C++. The only thing you need to remember is to use `mt19937`, included in the `<random>` header. This is a Mersenne Twister based on the prime $2^{19937} - 1$, which also happens to be its period. It's a much higher-quality RNG than `rand()`, in addition to being much faster (389 ms to generate and add 10^8 numbers from `mt19937` in Custom Invocation, vs. 1170 ms for `rand()`). It also produces full 32-bit unsigned outputs between 0 and $2^{32} - 1$ = 4294967295, rather than maxing out at a measly 32767.

To replace `random_shuffle()`, you can now call `shuffle()` and pass in your `mt19937` as the third argument; the shuffle algorithm will use your provided generator for shuffling.

C++11 also gives you some nifty distributions. `uniform_int_distribution` gives you perfectly uniform numbers, without the bias of mod -- i.e., `rand() % 10000` is more likely to give you a number between 0 and 999 than a number between 9000 and 9999, since

32767 is not a perfect multiple of 10000. There are many other fun distributions as well including `normal_distribution` and `exponential_distribution`.

To give you a more concrete idea, here's some code using several of the tools mentioned above. Note that the code seeds the random number generator using a high-precision clock. This is important for avoiding hacks specifically tailored to your code, since using a fixed seed means that anyone can determine what your RNG will output. For more details, see How randomized solutions can be hacked, and how to make your solution unhackable.

One last thing: if you want 64-bit random numbers, just use `mt19937_64` instead.

```
#include <algorithm>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>
using namespace std;

const int N = 3000000;

double average_distance(const vector<int> &permutation) {
    double distance_sum = 0;

    for (int i = 0; i < N; i++)
        distance_sum += abs(permutation[i] - i);

    return distance_sum / N;
}

int main() {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    vector<int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    shuffle(permutation.begin(), permutation.end(), rng);
    cout << average_distance(permutation) << '\n';

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[uniform_int_distribution<int>(0,
i)(rng)]);

    cout << average_distance(permutation) << '\n';
}
```

Both shuffles result in almost exactly 10^6 average distance, like we originally expected.

Additional References

This post was inspired in part by Stephan T. Lavavej's talk "rand() Considered Harmful": https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful

If you want even faster, higher-quality random number generators, take a look at this site by Sebastiano Vigna.

c++, c++11, random, random_shuffle

+891 ☆ neal 2 years ago 39

```cpp
int next_int() {
    char c;
    do { c = getchar(); } while( c != '-' && !isdigit(c) );
    bool neg = (c == '-');
    int result = neg ? 0 : c - '0';
    while( isdigit(c = getchar()) )
        result = 10 * result + (c - '0');
    return neg ? -result : result;
}
```

```cpp
//#pragma GCC optimize ("O3")
#include<bits/stdc++.h>
#define ll long long
#define S second
#define F first

using namespace std;

const int N = 1e6 + 5, M = 2 * N;

int arr[N];

int LIS(int i)
{
    static int dp[N];
    static int dummy = [](){memset(dp, -1, sizeof dp); return 0;}();
    int &ret = dp[i];
    if(~ret) return ret;
    ret = 1;
    for(int p = 0 ; p < i ; p++)
    {
        if(arr[p] < arr[i]) ret = max(ret, LIS(p) + 1);
    }
    return ret;
}

void LISBuild(int i)
{
    int ret = LIS(i);
    for(int p = 0 ; p < i ; p++)
    {
        if(arr[p] < arr[i] and ret == LIS(p) + 1)
        {
            LISBuild(p);
            break;
        }
    }
    printf("%d ", i);
    return;
}

int best[N], n;
int prv[N];

int LIS()
{
    arr[n] = INT_MIN;
    arr[n + 1] = INT_MAX;
    int mx = 0;
//    memset(best, '?', (n + 1) * sizeof best[0]);
    best[0] = n;
    best[1] = n + 1;
    for(int i = 0 ; i < n ; i++)
    {
        int idx = lower_bound(best, best + mx + 2, i, [](int a, int b){return
arr[a] < arr[b];}) - best;
        best[idx] = i;
        prv[i] = best[idx - 1];
        mx = max(mx, idx);
        best[mx + 1] = n + 1;
    }
    return mx;
}

void print(int i)
{
    if(i == n) return;
    print(prv[i]);
    printf("%d\n", arr[i]);
```

```
    return;
}

int main()
{
//    freopen("input.in", "r", stdin);
    n = 0;
    while(~scanf("%d", arr + n)) n++;
    int mx = LIS();
    printf("%d\n-\n", mx);
    print(best[mx]);
    return 0;
}
```