

# 1 ComputationalGeometry

## 1.1 Geometry Hezhu

```

1 Point GetLineIntersection(Point p, Vector v, Point q,
  Vector w) {
2   Vector u = p - q;
3   double t = Cross(w, u) / Cross(v, w);
4   return p + v * t;
5 }
6 Point GetLineProjection(Point P, Point A, Point B) {
7   Vector v = B - A;
8   return A + v * (Dot(v, P - A) / Dot(v, v));
9 }
10 double DistanceToLine(Point P, Point A, Point B) {
11   Vector v1 = B - A, v2 = P - A;
12   return fabs(Cross(v1, v2)) / Length(v1);
13 }
14 bool OnSegment(Point p, Point a1, Point a2) {
15   return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 -
    p, a2 - p)) < 0;
16 }
17 void getLineGeneralEquation(const Point &p1, const Point
    &p2, double &a,
18   double &b, double &c) {
19   a = p2.y - p1.y;
20   b = p1.x - p2.x;
21   c = -a * p1.x - b * p1.y;
22 }
23 double DistanceToSegment(Point p, Point a, Point b) {
24   if (a == b) return Length(p - a);
25   Vector v1 = b - a, v2 = p - a, v3 = p - b;
26   if (dcmp(Dot(v1, v2)) < 0)
27     return Length(v2);
28   else if (dcmp(Dot(v1, v3)) > 0)
29     return Length(v3);
30   else
31     return fabs(Cross(v1, v2)) / Length(v1);
32 }
33 double dis_pair_seg(Point p1, Point p2, Point p3, Point p4)
34 {
35   return min(
36     min(DistanceToSegment(p1, p3, p4),
37       DistanceToSegment(p2, p3, p4)),
38     min(DistanceToSegment(p3, p1, p2),
39       DistanceToSegment(p4, p1, p2)));
40 }
41 bool SegmentIntersection(Point a1, Point a2, Point b1,
    Point b2) {
42   double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1,
    b2 - a1),
43   c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2
    - b1);
44   return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
45 }
46 struct Line {
47   Point P;
48   Vector v;
49   double ang;
50   Line() {}
51   Line(Point P, Vector v) : P(P), v(v) { ang = atan2(v.y,
    v.x); }
52   Point point(double a) { return p + (v * a); }
53   bool operator<(const Line &L) const { return ang < L.ang;
    }
54 }
55 bool OnLeft(const Line &L, const Point &p) { return
    Cross(L.v, p - L.P) > 0; }
56 vector<Point> HalfplaneIntersection(vector<Line> L) {
57   int n = L.size();
58   sort(L.begin(), L.end());
59   int first, last;
60   vector<Point> p(n);
61   vector<Line> q(n);
62   vector<Point> ans;
63   q[first = last = 0] = L[0];
64   for (int i = 1; i < n; i++) {
65     while (first < last && !OnLeft(L[i], p[last - 1]))
66       last--;
67     while (first < last && !OnLeft(L[i], p[first])) first++;
68     q[++last] = L[i];
69     if (fabs(Cross(q[last].v, q[last - 1].v)) < eps) {
70       last--;
71       if (OnLeft(q[last], L[i].P)) q[last] = L[i];
72     }
73     if (first < last)
74       p[last - 1] = GetLineIntersection(q[last - 1],
75         q[last]);
76   }
77   while (first < last && !OnLeft(q[first], p[last - 1]))
78     last--;
79   if (last - first <= 1) return ans;
80   p[last] = GetLineIntersection(q[last], q[first]);
81   for (int i = first; i <= last; i++) ans.push_back(p[i]);
82   return ans;
83 }
84 Point PolyGravity(Point *p, int n) {
85   Point tmp, g = Point(0, 0);
86   double sumArea = 0, area;
87   for (int i = 2; i < n; ++i) {
88     area = Cross(p[i - 1] - p[0], p[i] - p[0]);
89     sumArea += area;
90     tmp.x = p[0].x + p[i - 1].x + p[i].x;
91     tmp.y = p[0].y + p[i - 1].y + p[i].y;
92     g.x += tmp.x * area;
93     g.y += tmp.y * area;
94   }
95   g.x /= (sumArea * 3.0);
96   g.y /= (sumArea * 3.0);
97   return g;
98 }
99 vector<Point> ConvexHull(vector<Point> &p) {
100   sort(p.begin(), p.end());
101   p.erase(unique(p.begin(), p.end(), p.end()));
102   int n = p.size();
103   int m = 0;
104   vector<Point> ch(n + 1);
105   for (int i = 0; i < n; i++) {
106     while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] -
107       ch[m - 2]) <= 0)
108       m--;
109     ch[m++] = p[i];
110   }
111   int k = m;
112   for (int i = n - 2; i >= 0; i--) {
113     while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] -
114       ch[m - 2]) <= 0)
115       m--;
116     ch[m++] = p[i];
117   }
118   if (n > 1) m--;
119   ch.resize(m);
120   return ch;
121 }
122 int isPointInPolygon(Point p, Polygon poly) {
123   int wn = 0;
124   int n = poly.size();
125   for (int i = 0; i < n; i++) {
126     if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1;
127     int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p -
128       poly[i]));
129     int d1 = dcmp(poly[i].y - p.y);
130     int d2 = dcmp(poly[(i + 1) % n].y - p.y);
131     if (k > 0 && d1 <= 0 && d2 > 0) wn++;
132     if (k < 0 && d2 <= 0 && d1 > 0) wn--;
133   }
134   if (wn != 0) return 1;
135   return 0;
136 }
137 int diameter2(vector<Point> &points) {
138   vector<Point> p = ConvexHull(points);
139   int n = p.size();
140   if (n == 1) return 0;
141   if (n == 2) return Dist2(p[0], p[1]);
142   p.push_back(p[0]);
143   int ans = 0;
144   for (int u = 0, v = 1; u < n; u++) {
145     for (; v) {
146       int diff = Cross(p[u + 1] - p[u], p[v + 1] - p[v]);
147       if (diff <= 0) {

```

```

139     ans = max(ans, Dist2(p[u], p[v]));
140     if (diff == 0) ans = max(ans, Dist2(p[u], p[v +
141         1]));
142     break;
143 }
144 v = (v + 1) % n;
145 }
146 return ans;
147 }
148 double RC_Distance(Point *ch1, Point *ch2, int n, int m) {
149     int q = 0, p = 0;
150     REP(i, n) if (ch1[i].y - ch1[p].y < -eps) p = i;
151     REP(i, m) if (ch2[i].y - ch2[q].y > eps) q = i;
152     ch1[n] = ch1[0];
153     ch2[m] = ch2[0];
154     double tmp, ans = 1e100;
155     REP(i, n) {
156         while ((tmp = Cross(ch1[p + 1] - ch1[p], ch2[q + 1] -
157             ch1[p]) -
158             Cross(ch1[p + 1] - ch1[p], ch2[q] - ch1[p])) >
159             eps)
160             q = (q + 1) % m;
161         if (tmp < -eps)
162             ans = min(ans, DistanceToSegment(ch2[q], ch1[p],
163                 ch1[p + 1]));
164         else
165             ans =
166                 min(ans, dis_pair_seg(ch1[p], ch1[p + 1], ch2[q],
167                     ch2[q + 1]));
168         p = (p + 1) % n;
169     }
170     return ans;
171 }
172 double RC_Triangle(Point *res, int n) {
173     if (n < 3) return 0;
174     double ans = 0, tmp;
175     res[n] = res[0];
176     int j, k;
177     REP(i, n) {
178         j = (i + 1) % n;
179         k = (j + 1) % n;
180         while ((j != k) && (k != i)) {
181             while (Cross(res[j] - res[i], res[k + 1] - res[i]) >
182                 Cross(res[j] - res[i], res[k] - res[i]))
183                 k = (k + 1) % n;
184             tmp = Cross(res[j] - res[i], res[k] - res[i]);
185             if (tmp > ans) ans = tmp;
186             j = (j + 1) % n;
187         }
188     }
189     return ans;
190 }
191 double fermat_point(Point *pt, int n, Point &ptres) {
192     Point u, v;
193     double step = 0.0, curlen, explen, minlen;
194     int i, j, k, idx;
195     bool flag;
196     u.x = u.y = v.x = v.y = 0.0;
197     REP(i, n) {
198         step += fabs(pt[i].x) + fabs(pt[i].y);
199         u.x += pt[i].x;
200         u.y += pt[i].y;
201     }
202     u.x /= n;
203     u.y /= n;
204     flag = 0;
205     while (step > eps) {
206         for (k = 0; k < 10; step /= 2, ++k)
207             for (i = -1; i <= 1; ++i)
208                 for (j = -1; j <= 1; ++j) {
209                     v.x = u.x + step * i;
210                     v.y = u.y + step * j;
211                     curlen = explen = 0.0;
212                     REP(idx, n) {
213                         curlen += dist(u, pt[idx]);
214                         explen += dist(v, pt[idx]);
215                     }
216                     if (curlen > explen) {
217                         u = v;
218                         minlen = explen;
219                     }
220                 }
221     }
222     return ans;
223 }
224 double Closest_Pair(int left, int right) {
225     double d = INF;
226     if (left == right) return d;
227     if (left + 1 == right) return dis(left, right);
228     int mid = (left + right) >> 1;
229     double d1 = Closest_Pair(left, mid);
230     double d2 = Closest_Pair(mid + 1, right);
231     d = min(d1, d2);
232     int i, j, k = 0;
233     for (i = left; i <= right; i++) {
234         if (fabs(point[mid].x - point[i].x) <= d) tmp[k++] = i;
235     }
236     sort(tmp, tmp + k, cmpy);
237     for (i = 0; i < k; i++) {
238         for (j = i + 1; j < k && point[tmp[j]].y -
239             point[tmp[i]].y < d; j++) {
240             double d3 = dis(tmp[i], tmp[j]);
241             if (d > d3) d = d3;
242         }
243     }
244     return d;
245 }
246 int getLineCircleIntersection(Line L, Circle C, double &t1,
247     double &t2, vector<Point> &sol) {
248     double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y
249         - C.c.y;
250     double e = a * a + c * c, f = 2 * (a * b + c * d),
251         g = b * b + d * d - C.r * C.r;
252     double delta = f * f - 4 * e * g;
253     if (dcmp(delta) < 0) return 0;
254     if (dcmp(delta) == 0) {
255         t1 = t2 = -f / (2 * e);
256         sol.push_back(L.point(t1));
257         return 1;
258     }
259     t1 = (-f - sqrt(delta)) / (2 * e);
260     t2 = (-f + sqrt(delta)) / (2 * e);
261     sol.push_back(L.point(t1));
262     sol.push_back(L.point(t2));
263     return 2;
264 }
265 int getCircleCircleIntersection(Circle C1, Circle C2,
266     vector<Point> &sol) {
267     double d = Length(C1.c - C2.c);
268     if (dcmp(d) == 0) {
269         if (dcmp(C1.r - C2.r) == 0) return -1;
270         return 0;
271     }
272     if (dcmp(C1.r + C2.r - d) < 0) return 0;
273     if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
274     double a = angle(C2.c - C1.c);
275     double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2
276         * C1.r * d));
277     Point p1 = C1.point(a - da), p2 = C1.point(a + da);
278     sol.push_back(p1);
279     if (p1 == p2) return 1;
280     sol.push_back(p2);
281     return 2;
282 }
283 int getTangents(Point p, Circle C, Vector *v) {
284     Vector u = C.c - p;
285     double dist = Length(u);
286     if (dist < C.r)
287         return 0;
288     else if (dcmp(dist - C.r) == 0) {
289         v[0] = Rotate(u, PI / 2);
290         return 1;
291     }
292     else {
293         double ang = asin(C.r / dist);
294         v[0] = Rotate(u, -ang);
295         v[1] = Rotate(u, +ang);
296         return 2;
297     }
298 }

```

```

290 }
291 }
292 int getTangents(Circle A, Circle B, Point *a, Point *b) {
293     int cnt = 0;
294     if (A.r < B.r) swap(A, B), swap(a, b);
295     int d2 =
296         (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) *
297         (A.c.y - B.c.y);
298     int rdif = A.r - B.r;
299     int rsum = A.r + B.r;
300     if (d2 < rdif * rdif) return 0;
301     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
302     if (d2 == 0 && A.r == B.r) return -1;
303     if (d2 == rdif * rdif) {
304         a[cnt] = A.point(base);
305         b[cnt] = B.point(base);
306         cnt++;
307         return 1;
308     }
309     double ang = acos((A.r - B.r) / sqrt(d2));
310     a[cnt] = A.point(base + ang);
311     b[cnt] = B.point(base + ang);
312     cnt++;
313     a[cnt] = A.point(base - ang);
314     b[cnt] = B.point(base - ang);
315     cnt++;
316     if (d2 == rsum * rsum) {
317         a[cnt] = A.point(base);
318         b[cnt] = B.point(Pi + base);
319         cnt++;
320     } else if (d2 > rsum * rsum) {
321         double ang = acos((A.r + B.r) / sqrt(d2));
322         a[cnt] = A.point(base + ang);
323         b[cnt] = B.point(Pi + base + ang);
324         cnt++;
325         a[cnt] = A.point(base - ang);
326         b[cnt] = B.point(Pi + base - ang);
327         cnt++;
328     }
329     return cnt;
330 }
331 Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
332     double Bx = p2.x - p1.x, By = p2.y - p1.y;
333     double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
334     double D = 2 * (Bx * Cy - By * Cx);
335     double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
336     double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
337     Point p = Point(cx, cy);
338     return Circle(p, Length(p1 - p));
339 }
340 Circle InscribedCircle(Point p1, Point p2, Point p3) {
341     double a = Length(p2 - p3);
342     double b = Length(p3 - p1);
343     double c = Length(p1 - p2);
344     Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
345     return Circle(p, DistanceToLine(p, p1, p2));
346 }
347 vector<Point>
348     CircleThroughPointTangentToLineGivenRadius(Point p,
349     Line L, double r) {
350     vector<Point> ans;
351     double t1, t2;
352     getLineCircleIntersection(L.move(-r), Circle(p, r), t1,
353     t2, ans);
354     getLineCircleIntersection(L.move(r), Circle(p, r), t1,
355     t2, ans);
356     return ans;
357 }
358 vector<Point> CircleTangentToLinesGivenRadius(Line a, Line
359     b, double r) {
360     vector<Point> ans;
361     Line L1 = a.move(-r), L2 = a.move(r);
362     Line L3 = b.move(-r), L4 = b.move(r);
363     ans.push_back(GetLineIntersection(L1, L3));
364     ans.push_back(GetLineIntersection(L1, L4));
365     ans.push_back(GetLineIntersection(L2, L3));
366     ans.push_back(GetLineIntersection(L2, L4));
367     return ans;
368 }
369 vector<Point>
370     CircleTangentToTwoDisjointCirclesWithRadius(Circle
371     c1, Circle c2, double r) {
372     vector<Point> ans;
373     Vector v = c2.c - c1.c;
374     double dist = Length(v);
375     int d = dcmp(dist - c1.r - c2.r - r * 2);
376     if (d > 0) return ans;
377     getCircleCircleIntersection(Circle(c1.c, c1.r + r),
378     Circle(c2.c, c2.r + r), ans);
379     return ans;
380 }
381 int getSegCircleIntersection(Line L, Circle C, Point *sol) {
382     Vector nor = normal(L.v);
383     Line pl = Line(C.c, nor);
384     Point ip = GetIntersection(pl, L);
385     double dis = Length(ip - C.c);
386     if (dcmp(dis - C.r) > 0) return 0;
387     Point dxy = vecunit(L.v) * sqrt(sqr(C.r) - sqr(dis));
388     int ret = 0;
389     sol[ret] = ip + dxy;
390     if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
391     sol[ret] = ip - dxy;
392     if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
393     return ret;
394 }
395 double SegCircleArea(Circle C, Point a, Point b) {
396     double a1 = angle(a - C.c);
397     double a2 = angle(b - C.c);
398     double da = fabs(a1 - a2);
399     if (da > Pi) da = Pi * 2.0 - da;
400     return dcmp(Cross(b - C.c, a - C.c)) * da * sqr(C.r) /
401     2.0;
402 }
403 double PolyCiclrArea(Circle C, Point *p, int n) {
404     double ret = 0.0;
405     Point sol[2];
406     p[n] = p[0];
407     REP(i, n) {
408         double t1, t2;
409         int cnt = getSegCircleIntersection(Line(p[i], p[i + 1]
410         - p[i]), C, sol);
411         if (cnt == 0) {
412             if (!On0rInCircle(p[i], C) || !On0rInCircle(p[i + 1],
413             C))
414                 ret += SegCircleArea(C, p[i], p[i + 1]);
415             else
416                 ret += Cross(p[i + 1] - C.c, p[i] - C.c) / 2.0;
417         }
418         if (cnt == 1) {
419             if (On0rInCircle(p[i], C) && !On0rInCircle(p[i + 1],
420             C))
421                 ret += Cross(sol[0] - C.c, p[i] - C.c) / 2.0,
422                 ret += SegCircleArea(C, sol[0], p[i + 1]);
423             else
424                 ret += SegCircleArea(C, p[i], sol[0]),
425                 ret += Cross(p[i + 1] - C.c, sol[0] - C.c) / 2.0;
426         }
427         if (cnt == 2) {
428             if ((p[i] < p[i + 1]) ^ (sol[0] < sol[1]))
429                 swap(sol[0], sol[1]);
430             ret += SegCircleArea(C, p[i], sol[0]);
431             ret += Cross(sol[1] - C.c, sol[0] - C.c) / 2.0;
432             ret += SegCircleArea(C, sol[1], p[i + 1]);
433         }
434     }
435     return fabs(ret);
436 }
437 double area[N];
438 int n;
439 struct cp {
440     double x, y, r, angle;
441     int d;
442     cp() {}
443     cp(double xx, double yy, double ang = 0, int t = 0) {
444         x = xx;
445         y = yy;
446         angle = ang;
447         d = t;
448     }
449     void get() {

```

```

436     scanf("%lf%lf%lf", &x, &y, &r);
437     d = 1;
438 }
439 } cir[N], tp[N * 2];
440 double dis(cp a, cp b) { return sqrt(sqr(a.x - b.x) +
441     sqr(a.y - b.y)); }
442 double cross(cp p0, cp p1, cp p2) {
443     return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) *
444         (p2.x - p0.x);
445 }
446 bool circmp(const cp &u, const cp &v) { return dcmp(u.r -
447     v.r) < 0; }
448 bool cmp(const cp &u, const cp &v) {
449     if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
450     return u.d > v.d;
451 }
452 double calc(cp cir, cp cp1, cp cp2) {
453     double ans = (cp2.angle - cp1.angle) * sqr(cir.r) -
454         cross(cir, cp1, cp2) +
455         cross(cp(0, 0), cp1, cp2);
456     return ans / 2;
457 }
458 void CirUnion(cp cir[], int n) {
459     cp cp1, cp2;
460     sort(cir, cir + n, circmp);
461     for (int i = 0; i < n; ++i)
462         for (int j = i + 1; j < n; ++j)
463             if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r)
464                 <= 0)
465                 cir[i].d++;
466     for (int i = 0; i < n; ++i) {
467         int tn = 0, cnt = 0;
468         for (int j = 0; j < n; ++j) {
469             if (i == j) continue;
470             if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
471                 cp2, cp1) < 2)
472                 continue;
473             cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
474             cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
475             cp1.d = 1;
476             tp[tn++] = cp1;
477             cp2.d = -1;
478             tp[tn++] = cp2;
479             if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
480         }
481         tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
482         tp[tn++] = cp(cir[i].x + cir[i].r, cir[i].y, -pi, cnt);
483         sort(tp, tp + tn, cmp);
484         int p, s = cir[i].d + tp[0].d;
485         for (int j = 1; j < tn; ++j) {
486             p = s;
487             s += tp[j].d;
488             area[p] += calc(cir[i], tp[j - 1], tp[j]);
489         }
490     }
491 }
492 void solve() {
493     scanf("%d", &n);
494     for (int i = 0; i < n; ++i) cir[i].get();
495     memset(area, 0, sizeof(area));
496     CirUnion(cir, n);
497     for (int i = 1; i <= n; ++i) { area[i] -= area[i + 1]; }
498     double tot = 0;
499     for (int i = 1; i <= n; ++i) tot += area[i];
500     printf("%f\n", tot);
501 }
502 inline double cross(point o, point a, point b) { return (a
503     - o) * (b - o); }
504 PDI s[maxN * maxp * 2];
505 Polygon P[maxN];
506 double S, ts;
507 int N;
508 inline double seg(point o, point a, point b) {
509     if (cmp(b.x - a.x) == 0) return (o.y - a.y) / (b.y - a.y);
510     return (o.x - a.x) / (b.x - a.x);
511 }
512 double PolygonUnion() {
513     int M, c1, c2;
514     double s1, s2, ret = 0;
515     for (int i = 0; i < N; ++i)
516         for (int ii = 0; ii < P[i].n; ++ii) {
517             M = 0;
518             s[M++] = mp(0.00, 0);
519             s[M++] = mp(1.00, 0);
520             for (int j = 0; j < N; ++j)
521                 if (i != j)
522                     for (int jj = 0; jj < P[j].n; ++jj) {
523                         c1 = cmp(cross(P[i][ii], P[i][ii + 1],
524                             P[j][jj]));
525                         c2 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj +
526                             1]));
527                         if (c1 == 0 && c2 == 0) {
528                             if (((P[i][ii + 1] - P[i][ii]) ^
529                                 (P[j][jj + 1] - P[j][jj])) > 0 &&
530                                 i > j) {
531                                 s[M++] = mp(
532                                     seg(P[j][jj], P[i][ii], P[i][ii + 1]), 1);
533                                 s[M++] = mp(
534                                     seg(P[j][jj + 1], P[i][ii], P[i][ii + 1]),
535                                     -1);
536                             }
537                         } else {
538                             s1 = cross(P[j][jj], P[j][jj + 1], P[i][ii]);
539                             s2 = cross(P[j][jj], P[j][jj + 1], P[i][ii +
540                                 1]);
541                             if (c1 >= 0 && c2 < 0)
542                                 s[M++] = mp(s1 / (s1 - s2), 1);
543                             else if (c1 < 0 && c2 >= 0)
544                                 s[M++] = mp(s1 / (s1 - s2), -1);
545                         }
546                     }
547             sort(s, s + M);
548             double pre = min(max(s[0].x, 0.0), 1.0), now;
549             double sum = 0;
550             int cov = s[0].y;
551             for (int j = 1; j < M; ++j) {
552                 now = min(max(s[j].x, 0.0), 1.0);
553                 if (!cov) sum += now - pre;
554                 cov += s[j].y;
555                 pre = now;
556             }
557             ret += P[i][ii] * P[i][ii + 1] * sum;
558         }
559     }
560     return ret / 2;
561 }
562 int main() {
563     for (int i = 0; i < N; ++i) {
564         P[i].n = 4;
565         P[i].input();
566         ts = P[i].Area();
567         if (cmp(ts < 0)) {
568             reverse(P[i].p, P[i].p + P[i].n);
569             P[i][P[i].n] = P[i][0];
570             ts = -ts;
571         }
572         S += ts;
573     }
574     printf("%.9lf\n", S / PolygonUnion());
575 }
576 // count(c / a + 1, c % a, a, b) + c / a + 1
577 long long count(long long a, long long b, long
578     long m) {
579     if (b == 0) { return n * (a / m); }
580     if (a >= m) { return n * (a / m) + count(n, a % m, b, m); }
581     if (b >= m) { return (n - 1) * n / 2 * (b / m) + count(n,
582         a, b % m, m); }
583     return count((a + b * n) / m, (a + b * n) % m, m, b);
584 }
585 bool TriSegIntersection(Point3 P0, Point3 P1, Point3 P2,
586     Point3 A, Point3 B, Point3 &P) {
587     Vector3 n = Cross(P1 - P0, P2 - P0);
588     if (dcmp(Dot(n, B - A)) == 0) return false;
589     double t = Dot(n, P0 - A) / Dot(n, B - A);
590     if (dcmp(t) < 0 || dcmp(t - 1) > 0) return false;
591     P = A + (B - A) * t;
592     return PointInTri(P, P0, P1, P2);
593 }
594 struct Face {
595     int v[3];
596     Vector3 normal(Point3 *P) const {
597         return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);
598     }
599 }

```

```

584 }
585 int cansee(Point3 *p, int i) const {
586     return Dot(P[i] - P[v[0]], normal(P)) > 0 ? 1 : 0;
587 }
588 };
589 vector<Face> CH3D(Point3 *P, int n) {
590     vector<Face> cur;
591     cur.push_back((Face){0, 1, 2});
592     cur.push_back((Face){2, 1, 0});
593     for (int i = 3; i < n; ++i) {
594         vector<Face> next;
595         for (int j = 0; j < cur.size(); ++j) {
596             Face &f = cur[j];
597             int res = f.cansee(P, i);
598             if (!res) next.push_back(f);
599             for (int k = 0; k < 3; ++k) vis[f.v[k]][f.v[(k + 1) % 3]] = res;
600         }
601         for (int j = 0; j < cur.size(); ++j)
602             for (int k = 0; k < 3; ++k) {
603                 int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];
604                 if (vis[a][b] != vis[b][a] && vis[a][b])
605                     next.push_back((Face){a, b, i});
606             }
607         cur = next;
608     }
609     return cur;
610 }

```

## 1.2 3D Convex

```

1  const int MAXN = 100;
2  const double EPS = 1e-8;
3  struct Point {
4      double x, y, z;
5      Point() {}
6      Point(double xx, double yy, double zz): x(xx), y(yy),
7          z(zz) {}
8      Point operator -(const Point p1) {
9          return Point(x - p1.x, y - p1.y, z - p1.z);
10     }
11     Point operator *(Point p) {
12         return Point(y * p.z - z * p.y, z * p.x - x * p.z,
13             x * p.y - y * p.x);
14     }
15     double operator ^(Point p) {
16         return (x * p.x + y * p.y + z * p.z);
17     }
18 };
19 struct CH3D {
20     struct face {
21         int a, b, c;
22         bool ok;
23     };
24     int n;
25     Point P[MAXN];
26     int num;
27     face F[8 * MAXN];
28     int g[MAXN][MAXN];
29     double vlen(Point a) {
30         return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
31     }
32     Point cross(const Point &a, const Point &b, const Point
33         &c) {
34         return Point((b.y - a.y) * (c.z - a.z) - (b.z -
35             a.z) * (c.y - a.y), -(b.x - a.x) * (c.z - a.z)
36             - (b.z - a.z) * (c.x - a.x), (b.x -
37             a.x) * (c.y - a.y) - (b.y - a.y) *
38             (c.x - a.x));
39     }
40     double area(Point a, Point b, Point c) {
41         return vlen((b - a) * (c - a)) * 0.5;
42     }
43     double volume(Point a, Point b, Point c, Point d) {
44         return (b - a) * (c - a) ^ (d - a);
45     }
46     double dblcmp(Point &p, face &f) {

```

```

41     Point m = P[f.b] - P[f.a];
42     Point n = P[f.c] - P[f.a];
43     Point t = p - P[f.a];
44     return (m * n) ^ t;
45 }
46 void deal(int p, int a, int b) {
47     int f = g[a][b];
48     face add;
49     if (F[f].ok) {
50         if (dblcmp(P[p], F[f]) > EPS)
51             dfs(p, f);
52         else {
53             add.a = b;
54             add.b = a;
55             add.c = p;
56             add.ok = 1;
57             g[p][b] = g[a][p] = g[b][a] = num;
58             F[num++] = add;
59         }
60     }
61 }
62 void dfs(int p, int now) {
63     F[now].ok = 0;
64     deal(p, F[now].b, F[now].a);
65     deal(p, F[now].c, F[now].b);
66     deal(p, F[now].a, F[now].c);
67 }
68 bool same(int s, int t) {
69     Point &a = P[F[s].a];
70     Point &b = P[F[s].b];
71     Point &c = P[F[s].c];
72     return fabs(volume(a, b, c, P[F[t].a])) < EPS &&
73         fabs(volume(a, b, c, P[F[t].b])) < EPS
74         && fabs(volume(a, b, c, P[F[t].c])) < EPS;
75 }
76 void solve() {
77     int i, j, tmp;
78     face add;
79     bool flag = true;
80     num = 0;
81     if (n < 4)
82         return;
83     for (i = 1; i < n; ++i) {
84         if (vlen(P[0] - P[i]) > EPS) {
85             swap(P[1], P[i]);
86             flag = false;
87             break;
88         }
89     }
90     if (flag)
91         return;
92     flag = true;
93     for (i = 2; i < n; ++i) {
94         if (vlen((P[0] - P[i]) * (P[1] - P[i])) > EPS) {
95             swap(P[2], P[i]);
96             flag = false;
97             break;
98         }
99     }
100     if (flag)
101         return;
102     flag = true;
103     for (i = 3; i < n; ++i) {
104         if (fabs((P[0] - P[i]) * (P[1] - P[2]) ^ (P[0] -
105             P[i])) > EPS) {
106             swap(P[3], P[i]);
107             flag = false;
108             break;
109         }
110     }
111     if (flag)
112         return;
113     for (i = 0; i < 4; ++i) {
114         add.a = (i + 1) % 4;
115         add.b = (i + 2) % 4;
116         add.c = (i + 3) % 4;
117         add.ok = true;
118         if (dblcmp(P[i], add) > 0)
119             swap(add.b, add.c);
120         g[add.a][add.b] = g[add.b][add.c] =
121             g[add.c][add.a] = num;

```

```

119     F[num++] = add;
120 }
121 for(i = 4; i < n; i++) {
122     for(j = 0; j < num; j++) {
123         if(F[j].ok && dblcmp(P[i], F[j]) > EPS) {
124             dfs(i, j);
125             break;
126         }
127     }
128 }
129 tmp = num;
130 for(i = num = 0; i < tmp; i++)
131     if(F[i].ok) {
132         F[num++] = F[i];
133     }
134 }
135 double area() {
136     double res = 0.0;
137     if(n == 3) {
138         Point p = cross(P[0], P[1], P[2]);
139         res = vlen(p) / 2.0;
140         return res;
141     }
142     for(int i = 0; i < num; i++)
143         res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
144     return res / 2.0;
145 }
146 double volume() {
147     double res = 0.0;
148     Point tmp(0, 0, 0);
149     for(int i = 0; i < num; i++)
150         res += volume(tmp, P[F[i].a], P[F[i].b],
151             P[F[i].c]);
152     return fabs(res / 6.0);
153 }
154 int triangle() {
155     return num;
156 }
157 int polygon() {
158     int i, j, res, flag;
159     for(i = res = 0; i < num; i++) {
160         flag = 1;
161         for(j = 0; j < i; j++)
162             if(same(i, j)) {
163                 flag = 0;
164                 break;
165             }
166         res += flag;
167     }
168     return res;
169 }
170 Point getcent() {
171     Point ans(0, 0, 0), temp = P[F[0].a];
172     double v = 0.0, t2;
173     for(int i = 0; i < num; i++) {
174         if(F[i].ok == true) {
175             Point p1 = P[F[i].a], p2 = P[F[i].b], p3 =
176                 P[F[i].c];
177             t2 = volume(temp, p1, p2, p3) / 6.0;
178             if(t2 > 0) {
179                 ans.x += (p1.x + p2.x + p3.x + temp.x) *
180                     t2;
181                 ans.y += (p1.y + p2.y + p3.y + temp.y) *
182                     t2;
183                 ans.z += (p1.z + p2.z + p3.z + temp.z) *
184                     t2;
185                 v += t2;
186             }
187         }
188     }
189     ans.x /= (4 * v);
190     ans.y /= (4 * v);
191     ans.z /= (4 * v);
192     return ans;
193 }
194 double function(Point fuck) {
195     double min = 99999999;
196     for(int i = 0; i < num; i++) {
197         if(F[i].ok == true) {
198             Point p1 = P[F[i].a], p2 = P[F[i].b], p3 =
199                 P[F[i].c];

```

```

194     double a = ( (p2.y - p1.y) * (p3.z - p1.z) -
195         (p2.z - p1.z) * (p3.y - p1.y) );
196     double b = ( (p2.z - p1.z) * (p3.x - p1.x) -
197         (p2.x - p1.x) * (p3.z - p1.z) );
198     double c = ( (p2.x - p1.x) * (p3.y - p1.y) -
199         (p2.y - p1.y) * (p3.x - p1.x) );
200     double d = ( 0 - (a * p1.x + b * p1.y + c *
201         p1.z) );
202     double temp = fabs(a * fuck.x + b * fuck.y +
203         c * fuck.z + d) / sqrt(a * a + b * b +
204         c * c);
205     if(temp < min) min = temp;
206 }
207 }
208 return min;
209 }
210 }
211 };

```

### 1.3 Closet Point Pair

```

1 const int Max = 100005;
2 struct Point {
3     double x, y;
4 };
5 Point p[Max], *px[Max], *py[Max];
6 inline int Cmp_x(Point *a, Point *b) {
7     return a->x < b->x;
8 }
9 inline int Cmp_y(Point *a, Point *b) {
10     return a->y < b->y;
11 }
12 inline double Dis(Point *a, Point *b) {
13     return sqrt((a->x - b->x) * (a->x - b->x) + (a->y -
14         b->y) * (a->y - b->y));
15 }
16 double Close(int l, int r) {
17     if (l + 1 == r)
18         return Dis(px[l], px[r]);
19     else if (l + 2 == r)
20         return min(min(Dis(px[l], px[l + 1]), Dis(px[l +
21         1], px[r])), Dis(
22             px[l], px[r]));
23     int mid = (l + r) / 2;
24     double ans = min(Close(l, mid), Close(mid + 1, r));
25     int i, j, cnt;
26     for (i = l, cnt = 0; i <= r; i++)
27         if (px[i]->x >= px[mid]->x - ans && px[i]->x <=
28             px[mid]->x + ans)
29             py[cnt++] = px[i];
30     sort(py, py + cnt, Cmp_y);
31     for (i = 0; i < cnt; i++)
32         for (j = i + 1; j < cnt; j++) {
33             if (py[j]->y - py[i]->y >= ans)
34                 break;
35             ans = min(ans, Dis(py[i], py[j]));
36         }
37     return ans;
38 }

```

## 2 DataStructure

### 2.1 BIT

```

1 int findkth(int k) {
2     int idx = 0;
3     for(int i = 20; i >= 0; --i) {
4         idx ^= 1 << i;
5         if(idx <= N && bit[idx] < k) k -= bit[idx];
6         else idx ^= 1 << i;
7     }
8     return idx + 1;
9 }

```

### 2.2 SBT



```

1  const int N = 100005;
2  struct SBT {
3      int lc, rc, sz, key;
4      void init(int k) {
5          lc = rc = 0;
6          sz = 1;
7          key = k;
8      }
9  } T[N];
10 int tot;
11 inline void push_up(int x) {
12     T[x].sz = T[T[x].lc].sz + T[T[x].rc].sz + 1;
13 }
14 void L_rotate(int &x) {
15     int k = T[x].rc;
16     T[x].rc = T[k].lc;
17     T[k].lc = x;
18     push_up(x);
19     push_up(k);
20     x = k;
21 }
22 void R_rotate(int &x) {
23     int k = T[x].lc;
24     T[x].lc = T[k].rc;
25     T[k].rc = x;
26     push_up(x);
27     push_up(k);
28     x = k;
29 }
30 void Maintain(int &x, bool fg) {
31     if(fg) {
32         if(T[T[x].rc].rc.sz > T[T[x].lc].sz) L_rotate(x);
33         else if(T[T[x].rc].lc.sz > T[T[x].lc].sz) {
34             R_rotate(T[x].rc);
35             L_rotate(x);
36         }
37         else return;
38     }
39     else {
40         if(T[T[x].lc].lc.sz > T[T[x].rc].sz) R_rotate(x);
41         else if(T[T[x].lc].rc.sz > T[T[x].rc].sz) {
42             L_rotate(T[x].lc);
43             R_rotate(x);
44         }
45         else return;
46     }
47     Maintain(T[x].lc, 0);
48     Maintain(T[x].rc, 1);
49     Maintain(x, 0);
50     Maintain(x, 1);
51 }
52 void Insert(int &x, int k) {
53     if(!x) {
54         x = ++tot;
55         T[x].init(k);
56     }
57     else {
58         Insert(k < T[x].key ? T[x].lc : T[x].rc, k);
59         push_up(x);
60         Maintain(x, k >= T[x].key);
61     }
62 }
63 int d_key;
64 void Delete(int &x, int k) {
65     if(T[x].key == k || (k < T[x].key && !T[x].lc) || (k >
66         T[x].key && !T[x].rc)) {
67         if(!T[x].lc || !T[x].rc) {
68             d_key = T[x].key;
69             x = T[x].lc + T[x].rc;
70         }
71         else {
72             Delete(T[x].lc, k + 1);
73             T[x].key = d_key;
74         }
75     }
76     else Delete(k < T[x].key ? T[x].lc : T[x].rc, k);
77     if(x) push_up(x);
78 }
79 int get_rank(int x, int k) {
80     int res = 1;
81     while(x) {
82         if(T[x].key < k) {
83             res += T[T[x].lc].sz + 1;
84             x = T[x].rc;
85         }
86         else x = T[x].lc;
87     }
88     return res;
89 }
90 int get_kth(int x, int k) {
91     while(T[T[x].lc].sz + 1 != k) {
92         if(T[T[x].lc].sz + 1 < k) {
93             k -= T[T[x].lc].sz + 1;
94             x = T[x].rc;
95         }
96         else x = T[x].lc;
97     }
98     return T[x].key;
99 }
100 int get_pre(int x, int k) {
101     int res;
102     while(x) {
103         if(T[x].key < k) {
104             res = T[x].key;
105             x = T[x].rc;
106         }
107         else x = T[x].lc;
108     }
109     return res;
110 }
111 int get_nxt(int x, int k) {
112     int res;
113     while(x) {
114         if(T[x].key > k) {
115             res = T[x].key;
116             x = T[x].lc;
117         }
118         else x = T[x].rc;
119     }
120     return res;
121 }

```

## 2.3 Splay

```

1  #define keyTree (ch[ch[root][1]][0])
2  const int N = 100005;
3  int pre[N], ch[N][2], sz[N];
4  int val[N], mx[N], add[N];
5  int tot, root;
6  void init() {
7      root = tot = 0;
8      ch[0][0] = ch[0][1] = pre[0] = 0;
9      sz[0] = val[0] = mx[0] = 0;
10     add[0] = 0;
11 }
12 inline void push_up(int x) {
13     sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1;
14     mx[x] = max(val[x], max(mx[ch[x][0]], mx[ch[x][1]]));
15 }
16 inline void Add(int x, int v) {
17     if(!x) return;
18     val[x] += v;
19     mx[x] += v;
20     add[x] += v;
21 }
22 inline void push_down(int x) {
23     if(!add[x]) return;
24     Add(ch[x][0], add[x]);
25     Add(ch[x][1], add[x]);
26     add[x] = 0;
27 }
28 inline void newnode(int &x, int v, int fa) {
29     x = ++tot;
30     pre[x] = fa;
31     ch[x][0] = ch[x][1] = 0;
32     sz[x] = 1;
33     mx[x] = val[x] = v;
34     add[x] = 0;
35 }
36 inline void Rotate(int x, bool kind) {

```

```

37     int y = pre[x];
38     ch[y][!kind] = ch[x][kind];
39     pre[ch[x][kind]] = y;
40     pre[x] = pre[y];
41     if(pre[x]) ch[pre[x]][ch[pre[x]][1] == y] = x;
42     ch[x][kind] = y;
43     pre[y] = x;
44     push_up(y);
45 }
46 void P(int x) {
47     if(!x) return;
48     P(pre[x]);
49     push_down(x);
50 }
51 void Splay(int x, int goal) {
52     P(x);
53     while(pre[x] != goal) {
54         if(pre[pre[x]] == goal) Rotate(x, ch[pre[x]][0] ==
55             x);
56         else {
57             int y = pre[x];
58             bool kind = (ch[pre[y]][0] == y);
59             if(ch[pre[x]][kind] == x)
60                 Rotate(x, !kind), Rotate(x, kind);
61             else
62                 Rotate(y, kind), Rotate(x, kind);
63         }
64         push_up(x);
65         if(!goal) root = x;
66     }
67     int get_kth(int k) {
68         int x = root;
69         push_down(x);
70         while(sz[ch[x][0]] + 1 != k) {
71             if(sz[ch[x][0]] + 1 < k) {
72                 k -= sz[ch[x][0]] + 1;
73                 x = ch[x][1];
74             }
75             else x = ch[x][0];
76             push_down(x);
77         }
78         return x;
79     }
80     void RotateTo(int k, int goal) {
81         int t = get_kth(k);
82         Splay(t, goal);
83     }
84     void update(int l, int r, int v) {
85         RotateTo(l, 0);
86         RotateTo(r + 2, root);
87         Add(keyTree, v);
88         push_up(ch[root][1]);
89         push_up(root);
90     }
91     void Delete(int x) {
92         RotateTo(x, 0);
93         RotateTo(x + 2, root);
94         keyTree = 0;
95         push_up(ch[root][1]);
96         push_up(root);
97     }
98     void Insert(int x, int v) {
99         RotateTo(x + 1, 0);
100        RotateTo(x + 2, root);
101        newnode(keyTree, v, ch[root][1]);
102        push_up(ch[root][1]);
103        push_up(root);
104    }

8     srand(0);
9     root = nill = pool;
10    nill->sz = 0;
11    tot = 0;
12 }
13 Treap* newnode(int v) {
14     Treap *t = pool + (++tot);
15     t->val = v;
16     t->sz = 1;
17     t->key = (rand() << 16) | rand();
18     t->lc = t->rc = nill;
19     return t;
20 }
21 inline void push_up(Treap *p) {
22     p->sz = p->lc->sz + p->rc->sz + 1;
23 }
24 Treap* Merge(Treap *a, Treap *b) {
25     if(a == nill) return b;
26     if(b == nill) return a;
27     if(a->key < b->key) {
28         a->rc = Merge(a->rc, b);
29         push_up(a);
30         return a;
31     }
32     else {
33         b->lc = Merge(a, b->lc);
34         push_up(b);
35         return b;
36     }
37 }
38 typedef pair <Treap*, Treap*> pii;
39 pii Split(Treap *a, int k) {
40     if(!k) return make_pair(nill, a);
41     int cnt = a->lc->sz;
42     if(cnt >= k) {
43         pii u = Split(a->lc, k);
44         a->lc = u.SE;
45         push_up(a);
46         return make_pair(u.FI, a);
47     }
48     else {
49         pii u = Split(a->rc, k - cnt - 1);
50         a->rc = u.FI;
51         push_up(a);
52         return make_pair(a, u.SE);
53     }
54 }
55 int get_rank(int k) {
56     Treap *p = root;
57     int res = 1;
58     while(p != nill) {
59         if(p->val < k) {
60             res += p->lc->sz + 1;
61             p = p->rc;
62         }
63         else p = p->lc;
64     }
65     return res;
66 }
67 int get_kth(int k) {
68     Treap *p = root;
69     while(p->lc->sz + 1 != k) {
70         if(p->lc->sz + 1 < k) {
71             k -= p->lc->sz + 1;
72             p = p->rc;
73         }
74         else p = p->lc;
75     }
76     return p->val;
77 }
78 int get_pre(int k) {
79     int res;
80     Treap *p = root;
81     while(p != nill) {
82         if(p->val < k) {
83             res = p->val;
84             p = p->rc;
85         }
86         else p = p->lc;
87     }
88     return res;

```

## 2.4 Treap

```

1  const int N = 100005;
2  struct Treap {
3      int key, val, sz;
4      Treap *lc, *rc;
5  } pool[N], *nill, *root;
6  int tot;
7  void init() {

```



```

89 }
90 int get_nxt(int k) {
91     int res;
92     Treap *p = root;
93     while(p != null) {
94         if(p->val > k) {
95             res = p->val;
96             p = p->lc;
97         }
98         else p = p->rc;
99     }
100     return res;
101 }
102 void Insert(int k) {
103     Treap *t = newnode(k);
104     pii u = Split(root, get_rank(k) - 1);
105     root = Merge(u.FI, t);
106     root = Merge(root, u.SE);
107 }
108 void Delete(int k) {
109     int p = get_rank(k);
110     pii a = Split(root, p - 1);
111     pii b = Split(a.SE, 1);
112     root = Merge(a.FI, b.SE);
113 }

```

## 2.5 LCT

```

1  const int N = 100005;
2  struct Link_Cut_Tree {
3      int pre[N], ch[N][2], bef[N];
4      bool rev[N];
5      inline void init() {
6
7      }
8      inline void Rev(int x) {
9          if(!x) return;
10         swap(ch[x][0], ch[x][1]);
11         rev[x] ^= 1;
12     }
13     inline void push_down(int x) {
14
15     }
16     inline void P(int x) {
17         if(pre[x]) P(pre[x]);
18         push_down(x);
19     }
20     inline void push_up(int x) {
21
22     }
23     inline void Rotate(int x, bool kind) {
24         int y = pre[x];
25         ch[y][!kind] = ch[x][kind];
26         pre[ch[x][kind]] = y;
27         pre[x] = pre[y];
28         if(pre[x]) ch[pre[x]][ch[pre[x]][1] == y] = x;
29         ch[x][kind] = y;
30         pre[y] = x;
31         //push_up(y);
32     }
33     inline void Splay(int x) {
34         P(x);
35         int r = x;
36         while(pre[r]) r = pre[r];
37         if(r != x) bef[x] = bef[r], bef[r] = 0;
38         while(pre[x]) {
39             if(pre[pre[x]] == 0) Rotate(x, ch[pre[x]][0] ==
40                 x);
41             else {
42                 int y = pre[x], k = ch[pre[y]][0] == y;
43                 if(ch[pre[x]][k] == x)
44                     Rotate(x, !k), Rotate(x, k);
45                 else
46                     Rotate(y, k), Rotate(x, k);
47             }
48             push_up(x);
49         }
50     inline void Access(int x) {

```

```

51     int fa = 0;
52     for(; x; x = bef[x]) {
53         Splay(x);
54         bef[ch[x][1]] = x;
55         bef[fa] = 0;
56         pre[ch[x][1]] = 0;
57         ch[x][1] = fa;
58         pre[fa] = x;
59         fa = x;
60         //push_up(x);
61     }
62 }
63 inline int get_rt(int x) {
64     Access(x);
65     Splay(x);
66     //push_down(x);
67     while(ch[x][0]) {
68         x = ch[x][0];
69         //push_down(x);
70     }
71     Splay(x);
72     return x;
73 }
74 inline void make_rt(int x) {
75     Access(x);
76     Splay(x);
77     Rev(x);
78 }
79 inline void Cut(int u, int v) {
80     make_rt(u);
81     Access(v);
82     Splay(v);
83     pre[ch[v][0]] = 0;
84     ch[v][0] = 0;
85     //push_up(v);
86 }
87 inline void Link(int u, int v) {
88     make_rt(v);
89     bef[v] = u;
90     Access(v);
91     /*make_rt(v);
92     push_down(v);
93     Access(u);
94     Splay(u);
95     pre[u] = v;
96     ch[v][0] = u;
97     push_up(v);*/
98 }
99 inline int Query(int x, int y) {
100     Access(y);
101     for(y = 0; x; x = bef[x]) {
102         Splay(x);
103         if(!bef[x]) return max(mx[y], mx[ch[x][1]]);
104         bef[ch[x][1]] = x;
105         bef[y] = 0;
106         pre[ch[x][1]] = 0;
107         ch[x][1] = y;
108         pre[y] = x;
109         y = x;
110         push_up(x);
111     }
112 }
113 } lct;

```

## 2.6 Leftist Tree

```

1  const int N = 100005;
2  struct LHeap {
3      int dis, key;
4      LHeap *lc, *rc;
5  } pool[N], *null;
6  int tot;
7  inline void init() {
8      tot = 0;
9      null = pool;
10     null->dis = -1;
11 }
12 inline LHeap* MakeTree(int v) {
13     LHeap *t = pool + (++tot);

```

```

14     t->lc = t->rc = null;
15     t->dis = 0;
16     t->key = v;
17     return t;
18 }
19 LHeap* Merge(LHeap *a, LHeap *b) {
20     if(a == null) return b;
21     if(b == null) return a;
22     if(a->key > b->key) swap(a, b);
23     a->rc = Merge(a->rc, b);
24     if(a->rc->dis > a->lc->dis) swap(a->rc, a->lc);
25     a->dis = a->rc->dis + 1;
26     return a;
27 }
28 inline void Insert(LHeap* &a, int v) {
29     LHeap *b = MakeTree(v);
30     a = Merge(a, b);
31 }
32 inline int DeleteMin(LHeap* &a) {
33     int t = a->key;
34     a = Merge(a->lc, a->rc);
35     return t;
36 }

```

## 2.7 Neighbors for Tree Path

```

1 inline int query(int u, int v) {
2     int ans = 0;
3     int f1 = top[u], f2 = top[v];
4     while(f1 ^ f2) {
5         if(dep[f1] < dep[f2]) {
6             swap(f1, f2);
7             swap(u, v);
8         }
9         //Heavy son of the end of this chain
10        if(son[u]) add(ans, sqr(bt1.query(L[son[u]],
11            R[son[u]])));
12        //All the light sons on this chain
13        add(ans, bt2.query(L[f1], L[u]));
14        //Subtract the value of the top of the chain, since
15        //we will count it when count the light sons on
16        //the above chain
17        add(ans, -sqr(bt1.query(L[f1], R[f1])));
18        u = fa[f1]; f1 = top[u];
19    }
20    if(dep[u] > dep[v]) swap(u, v);
21    //All the light sons on the last chain
22    add(ans, bt2.query(L[u], L[v]));
23    //Heavy son of the bottom of the last chain
24    if(son[v]) add(ans, sqr(bt1.query(L[son[v]],
25        R[son[v]])));
26    //Subtree above the LCA
27    if(u != 1) add(ans, sqr(sum - bt1.query(L[u], R[u])));
28    return ans;
29 }

```

## 2.8 RMQ

```

1 void initRMQ(int n) {
2     Lg[1] = 0;
3     for(int i = 2; i <= n; ++i) Lg[i] = Lg[i >> 1] + 1;
4     for(int j = 1; j < 20; ++j) {
5         for(int i = 1; i <= n; ++i) {
6             if(i + (1 << j) - 1 > n) break;
7             minx[i][j] = min(minx[i][j - 1], minx[i + (1 <<
8                 (j - 1))] [j - 1]);
9         }
10    }
11 }
12 inline int query(int l, int r) {
13     int t = Lg[r - l + 1];
14     return min(minx[l][t], minx[r - (1 << t) + 1][t]);
15 }

```

## 2.9 Manhattan Distance MST

```

1 struct Point {
2     int x, y, id;
3 } po[10005];
4 int data[10005], cc;
5 struct Edge {
6     int u, v, l;
7 } ed[50005];
8 int ecnt = 0;
9 inline int Find( int x ) {
10     return lower_bound( data, data + cc, x ) - data + 1;
11 }
12 inline bool cmp( Point a, Point b ) {
13     return a.x > b.x || ( a.x == b.x && a.y > b.y );
14 }
15 inline int AB( int x ) {
16     return x > 0 ? x : -x;
17 }
18 inline int Dis( Point a, Point b ) {
19     return AB( a.x - b.x ) + AB( a.y - b.y );
20 }
21 inline void addedge( int u, int v, int l ) {
22     ed[ecnt].u = u;
23     ed[ecnt].v = v;
24     ed[ecnt++].l = l;
25 }
26 int bitv[10005], bitid[10005];
27 inline void add( int x, int v, int id ) {
28     x = cc - x + 1;
29     for( ; x <= cc; x += x & -x ) if( bitv[x] > v ) {
30         bitv[x] = v;
31         bitid[x] = id;
32     }
33 }
34 inline int read( int x ) {
35     int v = INF, id = -1;
36     x = cc - x + 1;
37     for( ; x >= 1; x -= x & -x ) if( bitv[x] < v ) {
38         v = bitv[x];
39         id = bitid[x];
40     }
41     return id;
42 }
43 inline bool ecmp( Edge a, Edge b ) {
44     return a.l < b.l;
45 }
46 int F[10005];
47 inline int findroot( int x ) {
48     return F[x] == x ? x : F[x] = findroot( F[x] );
49 }
50 int main() {
51     int n, K;
52     while( ~scanf( "%d%d", &n, &K ) ) {
53         for( int i = 0; i < n; ++i ) {
54             scanf( "%d%d", &po[i].x, &po[i].y );
55             po[i].id = i;
56         }
57         for( int dir = 0; dir < 4; ++dir ) {
58             if( dir == 1 || dir == 3 ) {
59                 for( int i = 0; i < n; ++i ) swap( po[i].x,
60                     po[i].y );
61             } else if( dir == 2 ) {
62                 for( int i = 0; i < n; ++i ) po[i].x *= -1;
63             }
64             cc = 0;
65             for( int i = 0; i < n; ++i ) data[cc++] =
66                 po[i].y - po[i].x;
67             sort( data, data + cc );
68             cc = unique( data, data + cc ) - data;
69             sort( po, po + n, cmp );
70             memset( bitv, 0x3f, sizeof( bitv ) );
71             for( int i = 0; i < n; ++i ) {
72                 int v = Find( po[i].y - po[i].x );
73                 int id = read( v );
74                 if( id != -1 ) addedge( po[i].id, po[id].id,
75                     Dis( po[i], po[id] ) );
76                 add( v, po[i].x + po[i].y, i );
77             }
78             sort( ed, ed + ecnt, ecmp );
79             for( int i = 0; i < n; ++i ) F[i] = i;
80             int cnt = 0;

```

```

79     for( int i = 0; i < ecnt; ++i ) {
80         int fu = findroot( ed[i].u ), fv = findroot(
            ed[i].v );
81         if( fu == fv ) continue;
82         ++cnt;
83         if( cnt == n - K ) {
84             printf( "%d\n", ed[i].l );
85             break;
86         }
87         F[fu] = fv;
88     }
89     return 0;
90 }
91

```

## 2.10 Scanline Circle

```

1  const int N = 150005;
2  int x[N], y[N], r[N], id[N];
3  int op[N];
4  struct Event {
5      int x, tp, id;
6      bool operator < (const Event &a) const {
7          if( x != a.x ) return x < a.x;
8          return tp > a.tp;
9      }
10 } C[N * 2];
11 int tot = 0;
12 void add(int x, int tp, int id) {
13     C[tot].x = x;
14     C[tot].tp = tp;
15     C[tot++].id = id;
16 }
17 inline double sqr(double x) {
18     return x * x;
19 }
20 inline int sgn(double x) {
21     if( x < -eps ) return -1;
22     return x > eps;
23 }
24 double X;
25 struct HC {
26     int id, up;
27     double get_y() const {
28         double v = sqr(r[id]) - sqr(x[id] - X);
29         v = sqrt(max(v, 0.0));
30         return up ? y[id] + v : y[id] - v;
31     }
32     bool operator < (const HC &a) const {
33         int ck = sgn(get_y() - a.get_y());
34         if( ck ) return ck > 0;
35         return up > a.up;
36     }
37 };
38 inline bool OnCircle(int c, int p) {
39     int dx = (x[c] - x[p]), dy = (y[c] - y[p]);
40     return r[c] * r[c] == dx * dx + dy * dy;
41 }
42 set <HC> st;
43 int fa[N], belong[N];
44 void get_fa(int &x, int up, int down) {
45     if( up == down ) x = up;
46     else if( fa[down] == up ) x = up;
47     else if( fa[up] == down ) x = down;
48     else x = fa[up];
49 }
50 int main() {
51     int n, m;
52     scanf("%d%d", &n, &m);
53     for( int i = 0; i < n; ++i ) {
54         op[i] = 1;
55         scanf("%d", &id[i]);
56         scanf("%d%d%d", &x[id[i]], &y[id[i]], &r[id[i]]);
57         add(x[id[i]] - r[id[i]], -1, id[i]);
58         add(x[id[i]] + r[id[i]], 1, id[i]);
59     }
60     for( int i = n; i < n + m; ++i ) {
61         op[i] = 2;
62         scanf("%d", &id[i]);

```

```

63         scanf("%d%d", &x[id[i]], &y[id[i]]);
64         add(x[id[i]], 0, id[i]);
65     }
66     int q;
67     scanf("%d", &q);
68     q += n + m;
69     for( int i = n + m; i < q; ++i ) {
70         scanf("%d%d", &op[i], &id[i]);
71         if( op[i] == 1 ) {
72             scanf("%d%d%d", &x[id[i]], &y[id[i]], &r[id[i]]);
73             add(x[id[i]] - r[id[i]], -1, id[i]);
74             add(x[id[i]] + r[id[i]], 1, id[i]);
75         } else if( op[i] == 2 ) {
76             scanf("%d%d", &x[id[i]], &y[id[i]]);
77             add(x[id[i]], 0, id[i]);
78         }
79     }
80     sort(C, C + tot);
81     HC t;
82     for( int i = 0; i < tot; ++i ) {
83         X = C[i].x;
84         if( C[i].tp == -1 ) {
85             t.id = C[i].id;
86             t.up = 0;
87             set <HC>::iterator it = st.upper_bound(t);
88             int up = 0, down = 0;
89             if( it != st.end() ) down = it->id;
90             if( it != st.begin() ) up = (--it)->id;
91             get_fa(fa[t.id], up, down);
92             st.insert(t);
93             t.up = 1;
94             st.insert(t);
95         } else if( C[i].tp == 0 ) {
96             t.id = C[i].id;
97             t.up = 1;
98             set <HC>::iterator it = st.lower_bound(t);
99             if( it == st.end() ) continue;
100             if( OnCircle(it->id, t.id) ) {
101                 belong[t.id] = fa[it->id];
102                 continue;
103             }
104             if( it == st.begin() ) continue;
105             int down = it->id, up = (--it)->id;
106             get_fa(belong[t.id], up, down);
107         } else {
108             t.id = C[i].id;
109             t.up = 0;
110             st.erase(t);
111             t.up = 1;
112             st.erase(t);
113         }
114     }
115     return 0;
116 }

```

## 3 Graph

### 3.1 Blossom Tree

```

1  const int N = 250;
2  int belong[N];
3  int findb(int x) {
4      return belong[x] == x ? x : belong[x] =
        findb(belong[x]);
5  }
6  void unit(int a, int b) {
7      a = findb(a);
8      b = findb(b);
9      if( a != b ) belong[a] = b;
10 }
11 int n, match[N];
12 vector<int> e[N];
13 int Q[N], rear;
14 int next[N], mark[N], vis[N];
15 int LCA(int x, int y) {
16     static int t = 0;
17     t++;

```

```

18 while (true) {
19     if (x != -1) {
20         x = findb(x);
21         if (vis[x] == t) return x;
22         vis[x] = t;
23         if (match[x] != -1) x = next[match[x]];
24         else x = -1;
25     }
26     swap(x, y);
27 }
28 }
29 void group(int a, int p) {
30     while (a != p) {
31         int b = match[a], c = next[b];
32         if (findb(c) != p) next[c] = b;
33         if (mark[b] == 2) mark[Q[rear++]] = b;
34         if (mark[c] == 2) mark[Q[rear++]] = c;
35         unit(a, b);
36         unit(b, c);
37         a = c;
38     }
39 }
40 void aug(int s) {
41     for (int i = 0; i < n; i++)
42         next[i] = -1, belong[i] = i, mark[i] = 0, vis[i] =
43             -1;
44     mark[s] = 1;
45     Q[0] = s;
46     rear = 1;
47     for (int front = 0; match[s] == -1 && front < rear;
48         front++) {
49         int x = Q[front];
50         for (int i = 0; i < (int)e[x].size(); i++) {
51             int y = e[x][i];
52             if (match[x] == y) continue;
53             if (findb(x) == findb(y)) continue;
54             if (mark[y] == 2) continue;
55             if (mark[y] == 1) {
56                 int r = LCA(x, y);
57                 if (findb(x) != r) next[x] = y;
58                 if (findb(y) != r) next[y] = x;
59                 group(x, r);
60                 group(y, r);
61             } else if (match[y] == -1) {
62                 next[y] = x;
63                 for (int u = y; u != -1; ) {
64                     int v = next[u];
65                     int mv = match[v];
66                     match[v] = u, match[u] = v;
67                     u = mv;
68                 }
69                 break;
70             } else {
71                 next[y] = x;
72                 mark[Q[rear++]] = match[y] = 1;
73                 mark[y] = 2;
74             }
75         }
76     }
77 }
78 bool g[N][N];
79 int main() {
80     scanf("%d", &n);
81     for (int i = 0; i < n; i++)
82         for (int j = 0; j < n; j++) g[i][j] = false;
83     int x, y;
84     while (scanf("%d%d", &x, &y) != EOF) {
85         x--, y--;
86         if (x != y && !g[x][y])
87             e[x].push_back(y), e[y].push_back(x);
88         g[x][y] = g[y][x] = true;
89     }
90
91     for (int i = 0; i < n; i++) match[i] = -1;
92     for (int i = 0; i < n; i++) if (match[i] == -1) aug(i);
93
94     int tot = 0;
95     for (int i = 0; i < n; i++) if (match[i] != -1) tot++;
96     printf("%d\n", tot);

```

```

97     for (int i = 0; i < n; i++) if (match[i] > i)
98         printf("%d %d\n", i + 1, match[i] + 1);
99     return 0;
100 }

```

## 3.2 Directed MST

```

1 #define M 600
2 #define type int
3 const type inf = (1) << 30;
4 struct Node {
5     int u, v;
6     type cost;
7 } E[M * M + 5];
8 int pre[M], ID[M], vis[M];
9 type In[M];
10
11 type Directed_MST(int root, int NV, int NE) {
12     type ret = 0;
13     while(true) {
14         for(int i = 0; i < NV; i++) In[i] = inf;
15         for(int i = 0; i < NE; i++) {
16             int u = E[i].u;
17             int v = E[i].v;
18             if(E[i].cost < In[v] && u != v) {
19                 pre[v] = u;
20                 In[v] = E[i].cost;
21             }
22         }
23         for(int i = 0; i < NV; i++) {
24             if(i == root) continue;
25             if(In[i] == inf) return -1;
26         }
27         int cntnode = 0;
28         memset(ID, -1, sizeof(ID));
29         memset(vis, -1, sizeof(vis));
30         In[root] = 0;
31         for(int i = 0; i < NV; i++) {
32             ret += In[i];
33             int v = i;
34             while(vis[v] != i && ID[v] == -1 && v != root) {
35                 vis[v] = i;
36                 v = pre[v];
37             }
38             if(v != root && ID[v] == -1) {
39                 for(int u = pre[v]; u != v; u = pre[u]) {
40                     ID[u] = cntnode;
41                 }
42                 ID[v] = cntnode++;
43             }
44         }
45         if(cntnode == 0) break;
46         for(int i = 0; i < NV; i++) if(ID[i] == -1) {
47             ID[i] = cntnode++;
48         }
49         for(int i = 0; i < NE; i++) {
50             int v = E[i].v;
51             E[i].u = ID[E[i].u];
52             E[i].v = ID[E[i].v];
53             if(E[i].u != E[i].v) {
54                 E[i].cost -= In[v];
55             }
56         }
57         NV = cntnode;
58         root = ID[root];
59     }
60     return ret;
61 }

```

## 3.3 Directed MST SOL

```

1 const int N = 505;
2 const int DN = N << 1;
3 const int M = N * N + 5;
4 const int inf = 1 << 30;
5 struct EDGE { int u, v, w; };
6 struct D_MST {

```

```

7  EDGE E[M];
8  int pre[DN], ID[DN], vis[DN];
9  int In[DN], inE[DN], ring;
10 int nV[DN], nnV[DN];
11 vector < pair <int, int> > R[DN];
12 bool ans[M], newR[DN];
13 map <int, int> dirE[DN];
14 void del(int u, int e) {
15     if(R[u].empty()) return;
16     int pu = dirE[u][e];
17     for(auto &x : R[u]) {
18         if(x.first == pu) {
19             ans[x.second] = false;
20             del(x.first, e);
21         } else {
22             del(x.first, x.second);
23         }
24     }
25 }
26 int Directed_MST(int rt, int n, int m) {
27     while(true) {
28         for(int i = 0; i < n; ++i) In[nV[i]] = inf;
29         for(int i = 0; i < m; ++i) {
30             int u = E[i].u, v = E[i].v;
31             if(E[i].w < In[v] && u != v) {
32                 pre[v] = u;
33                 In[v] = E[i].w;
34                 inE[v] = i;
35             }
36         }
37         for(int i = 0; i < n; ++i) {
38             if(nV[i] != rt && In[nV[i]] == inf)
39                 return -1;
40         }
41         int cntnode = 0;
42         memset(ID, -1, sizeof(ID));
43         memset(vis, -1, sizeof(vis));
44         memset(newR, 0, sizeof(newR));
45         In[rt] = 0;
46         for(int i = 0; i < n; ++i) {
47             int v = nV[i], s = v;
48             while(vis[v] != s && ID[v] == -1 && v != rt) {
49                 vis[v] = s;
50                 v = pre[v];
51             }
52             if(v != rt && ID[v] == -1) {
53                 for(int u = pre[v]; ; u = pre[u]) {
54                     ID[u] = ring;
55                     R[ring].push_back( {u, inE[u]} );
56                     ans[inE[u]] = true;
57                     newR[u] = true;
58                     if(u == v) break;
59                 }
60                 nnV[cntnode++] = ring++;
61             }
62         }
63         if(cntnode == 0) {
64             for(int i = 0; i < n; ++i) {
65                 if(nV[i] == rt) continue;
66                 ans[inE[nV[i]]] = true;
67                 del(nV[i], inE[nV[i]]);
68             }
69             return 0;
70         }
71         for(int i = 0; i < n; ++i) {
72             int v = nV[i];
73             if(ID[v] != -1) continue;
74             ID[v] = v;
75             nnV[cntnode++] = v;
76         }
77         for(int i = 0; i < m; ++i) {
78             int v = E[i].v;
79             E[i].u = ID[E[i].u];
80             E[i].v = ID[E[i].v];
81             if(!newR[v]) continue;
82             if(E[i].u != E[i].v) {
83                 E[i].w -= In[v];
84                 dirE[E[i].v][i] = v;
85             }
86         }

```

```

87         n = cntnode; rt = ID[rt];
88         for(int i = 0; i < n; ++i) nV[i] = nnV[i];
89     }
90 }
91 vector <int> solve(int rt, int n, int m, EDGE *e) {
92     for(int i = 0; i < m; ++i) E[i] = e[i];
93     memset(ans, 0, m * sizeof(bool));
94     for(int i = 0; i < n; ++i) nV[i] = i;
95     ring = n; vector <int> ret;
96     if(Directed_MST(rt, n, m) == -1) return ret;
97     for(int i = 0; i < ring; ++i)
98         dirE[i].clear(), R[i].clear();
99     for(int i = 0; i < m; ++i) if(ans[i])
100         ret.push_back(i);
101     return ret;
102 } mst;

```

### 3.4 Global Minimum Cut

```

1  const int maxn = 510;
2  int G[maxn][maxn];
3  int n, m;
4  void contract(int x, int y) {
5      for(int i = 0; i < n; ++i) if(i != x) G[x][i] +=
6          G[y][i], G[i][x] += G[i][y];
7      for(int i = y + 1; i < n; ++i) for(int j = 0; j < n;
8          ++j) {
9          G[i - 1][j] = G[i][j];
10         G[j][i - 1] = G[j][i];
11     }
12     n--;
13 }
14 int w[maxn], c[maxn];
15 int sx, tx;
16 int mincut() {
17     int t, k;
18     memset(c, 0, sizeof(c));
19     c[0] = 1;
20     for(int i = 0; i < n; ++i) w[i] = G[0][i];
21     for(int i = 1; i < n; ++i) {
22         t = k = -1;
23         for(int j = 0; j < n; ++j) if(c[j] == 0 && w[j] >
24             k) k = w[t = j];
25         c[sx = t] = 1;
26         for(int j = 0; j < n; ++j) w[j] += G[t][j];
27     }
28     for(int i = 0; i < n; ++i) if(c[i] == 0) return w[tx =
29         i];
30 }
31 int main() {
32     while(~scanf("%d%d", &n, &m)) {
33         memset(G, 0, sizeof(G));
34         while(m--) {
35             int u, v, c;
36             scanf("%d%d%d", &u, &v, &c);
37             G[u][v] += c;
38             G[v][u] += c;
39         }
40         int mint = INF;
41         while(n > 1) {
42             int t = mincut();
43             mint = min(mint, t);
44             contract(sx, tx);
45         }
46         printf("%d\n", mint);
47     }
48     return 0;
49 }

```

### 3.5 Dominator Tree

```

1  const int vector_num = 50000;
2  vector<int> succ[vector_num + 10], prod[vector_num + 10],
3      bucket[vector_num + 10], dom_t[vector_num + 10];
4  int semi[vector_num + 10], anc[vector_num + 10],
5      idom[vector_num + 10], best[vector_num + 10],

```

```

    fa[vector_num + 10];
4  int dfn[vector_num + 10], redfn[vector_num + 10];
5  int child[vector_num + 10], size[vector_num + 10];
6  int timestamp;
7  void dfs(int now) {
8      dfn[now] = ++timestamp;
9      redfn[timestamp] = now;
10     anc[timestamp] = idom[timestamp] = child[timestamp] =
        size[timestamp] = 0;
11     semi[timestamp] = best[timestamp] = timestamp;
12     int sz = succ[now].size();
13     for(int i = 0; i < sz; ++i) {
14         if(dfn[succ[now][i]] == -1) {
15             dfs(succ[now][i]);
16             fa[dfn[succ[now][i]]] = dfn[now];
17         }
18         prod[dfn[succ[now][i]]].push_back(dfn[now]);
19     }
20 }
21 void compress(int now) {
22     if(anc[anc[now]] != 0) {
23         compress(anc[now]);
24         if(semi[best[now]] > semi[best[anc[now]]])
25             best[now] = best[anc[now]];
26         anc[now] = anc[anc[now]];
27     }
28 }
29 inline int eval(int now) {
30     if(anc[now] == 0)
31         return now;
32     else {
33         compress(now);
34         return semi[best[anc[now]]] >= semi[best[now]] ?
            best[now] : best[anc[now]];
35     }
36 }
37 }
38 inline void link(int v, int w) {
39     int s = w;
40     while(semi[best[w]] < semi[best[child[w]]]) {
41         if(size[s] + size[child[s]] >= 2 *
            size[child[s]]) {
42             anc[child[s]] = s;
43             child[s] = child[child[s]];
44         } else {
45             size[child[s]] = size[s];
46             s = anc[s] = child[s];
47         }
48     }
49     best[s] = best[w];
50     size[v] += size[w];
51     if(size[v] < 2 * size[w])
52         swap(s, child[v]);
53     while(s != 0) {
54         anc[s] = v;
55         s = child[s];
56     }
57 }
58 void lengauer_tarjan(int n) { // n is the vertices' number
59     memset(dfn, -1, sizeof dfn);
60     memset(fa, -1, sizeof fa);
61     timestamp = 0;
62     dfs(n);
63     fa[1] = 0;
64     for(int w = timestamp; w > 1; --w) {
65         int sz = prod[w].size();
66         for(int i = 0; i < sz; ++i) {
67             int u = eval(prod[w][i]);
68             if(semi[w] > semi[u])
69                 semi[w] = semi[u];
70         }
71         bucket[semi[w]].push_back(w);
72         //anc[w] = fa[w]; link operation for o(mlogm)
        version
73         link(fa[w], w);
74         if(fa[w] == 0)
75             continue;
76         sz = bucket[fa[w]].size();
77         for(int i = 0; i < sz; ++i) {
78             int u = eval(bucket[fa[w]][i]);
79             if(semi[u] < fa[w])

```

```

80         idom[bucket[fa[w]][i]] = u;
81     else
82         idom[bucket[fa[w]][i]] = fa[w];
83 }
84 bucket[fa[w]].clear();
85 }
86 for(int w = 2; w <= timestamp; ++w) {
87     if(idom[w] != semi[w])
88         idom[w] = idom[idom[w]];
89 }
90 idom[1] = 0;
91 for(int i = timestamp; i > 1; --i) {
92     if(fa[i] == -1)
93         continue;
94     dom_t[idom[i]].push_back(i);
95 }
96 }
97 long long ans[50010];
98 void get_ans(int now) {
99     ans[redfn[now]] += redfn[now];
100    int sz = dom_t[now].size();
101    for(int i = 0; i < sz; ++i) {
102        ans[redfn[dom_t[now][i]]] += ans[redfn[now]];
103        get_ans(dom_t[now][i]);
104    }
105 }
106 void init(int n, int m) {
107     for(int i = 0; i <= n; i++)
108         succ[i].clear(), prod[i].clear(),
            bucket[i].clear(), dom_t[i].clear();
109     memset(ans, 0, sizeof(*ans) * (n + 3));
110 }
111 int main() {
112     int n, m;
113     while(scanf("%d%d", &n, &m) != EOF) {
114         init(n, m);
115         for(int i = 0, u, v; i < m; i++) {
116             scanf("%d%d", &u, &v);
117             succ[u].push_back(v);
118         }
119         lengauer_tarjan(n);
120         get_ans(1);
121         for(int i = 1; i <= n; i++)
122             printf("%I64d%c", ans[i], i == n ? '\n' : ' ');
123     }
124     return 0;
125 }

```

### 3.6 KM

```

1  /*****
2  Bipartite Graph Maximum Weighted Matching
3  (kuhn munkras algorithm O(m*m*n))
4  adjacent matrix: mat
5  notice: m <= n
6  init: for(i=0;i<MAXN;i++)
7         for(j=0;j<MAXN;j++) mat[i][j]=-inf;
8  for existing edges: mat[i][j]=val;
9  *****/
10 #define MAXN 310
11 #define inf 1000000000
12 #define _clr(x) memset(x,-1,sizeof(int)*MAXN)
13 int KM(int m, int n, int mat[][MAXN], int *match1, int
    *match2) {
14     int s[MAXN], t[MAXN], l1[MAXN], l2[MAXN];
15     int p, q, i, j, k, ret = 0;
16     for(i = 0; i < m; i++) {
17         l1[i] = -inf;
18         for(j = 0; j < n; j++)
19             l1[i] = mat[i][j] > l1[i] ? mat[i][j] : l1[i];
20         if(l1[i] == -inf) return -1;
21     }
22     for(i = 0; i < n; i++)
23         l2[i] = 0;
24     _clr(match1);
25     _clr(match2);
26     for(i = 0; i < m; i++) {
27         _clr(t);
28         p = 0;

```

```

29     q = 0;
30     for(s[0] = i; p <= q && match1[i] < 0; p++) {
31         for(k = s[p], j = 0; j < n && match1[i] < 0;
32             j++) {
33             if(l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
34                 s[++q] = match2[j];
35                 t[j] = k;
36                 if(s[q] < 0) {
37                     for(p = j; p >= 0; j = p) {
38                         match2[j] = k = t[j];
39                         p = match1[k];
40                         match1[k] = j;
41                     }
42                 }
43             }
44         }
45         if(match1[i] < 0) {
46             i--;
47             p = inf;
48             for(k = 0; k <= q; k++) {
49                 for(j = 0; j < n; j++) {
50                     if(t[j] < 0 && l1[s[k]] + l2[j] -
51                         mat[s[k]][j] < p)
52                         p = l1[s[k]] + l2[j] - mat[s[k]][j];
53                 }
54                 for(j = 0; j < n; j++)
55                     l2[j] += t[j] < 0 ? 0 : p;
56                 for(k = 0; k <= q; k++)
57                     l1[s[k]] -= p;
58             }
59         }
60         for(i = 0; i < m; i++)
61             ret += mat[i][match1[i]];
62         return ret;
63     }

```

### 3.7 SAP

```

1  const int MAXN = 20010;
2  const int MAXM = 880010;
3  const int INF = 0x3f3f3f3f;
4  struct Node {
5      int from, to, next;
6      int cap;
7  } edge[MAXN];
8  int tol;
9  int head[MAXN];
10 int dep[MAXN];
11 int gap[MAXN];
12 int n;
13 void init() {
14     tol = 0;
15     memset(head, -1, sizeof(head));
16 }
17 void addedge(int u, int v, int w) {
18     edge[tol].from = u;
19     edge[tol].to = v;
20     edge[tol].cap = w;
21     edge[tol].next = head[u];
22     head[u] = tol++;
23     edge[tol].from = v;
24     edge[tol].to = u;
25     edge[tol].cap = 0;
26     edge[tol].next = head[v];
27     head[v] = tol++;
28 }
29 void BFS(int start, int end) {
30     memset(dep, -1, sizeof(dep));
31     memset(gap, 0, sizeof(gap));
32     gap[0] = 1;
33     int que[MAXN];
34     int front, rear;
35     front = rear = 0;
36     dep[end] = 0;
37     que[rear++] = end;
38     while(front != rear) {
39         int u = que[front++];

```

```

40         if(front == MAXN) front = 0;
41         for(int i = head[u]; i != -1; i = edge[i].next) {
42             int v = edge[i].to;
43             if(dep[v] != -1) continue;
44             que[rear++] = v;
45             if(rear == MAXN) rear = 0;
46             dep[v] = dep[u] + 1;
47             ++gap[dep[v]];
48         }
49     }
50 }
51 int SAP(int start, int end) {
52     int res = 0;
53     BFS(start, end);
54     int cur[MAXN];
55     int S[MAXN];
56     int top = 0;
57     memcpy(cur, head, sizeof(head));
58     int u = start;
59     int i;
60     while(dep[start] < n) {
61         if(u == end) {
62             int temp = INF;
63             int inser;
64             for(i = 0; i < top; i++)
65                 if(temp > edge[S[i]].cap) {
66                     temp = edge[S[i]].cap;
67                     inser = i;
68                 }
69             for(i = 0; i < top; i++) {
70                 edge[S[i]].cap -= temp;
71                 edge[S[i] ^ 1].cap += temp;
72             }
73             res += temp;
74             top = inser;
75             u = edge[S[top]].from;
76         }
77         if(u != end && gap[dep[u] - 1] == 0)
78             break;
79         for(i = cur[u]; i != -1; i = edge[i].next)
80             if(edge[i].cap != 0 && dep[u] == dep[edge[i].to]
81                 + 1)
82                 break;
83         if(i != -1) {
84             cur[u] = i;
85             S[top++] = i;
86             u = edge[i].to;
87         } else {
88             int min = n;
89             for(i = head[u]; i != -1; i = edge[i].next) {
90                 if(edge[i].cap == 0) continue;
91                 if(min > dep[edge[i].to]) {
92                     min = dep[edge[i].to];
93                     cur[u] = i;
94                 }
95             }
96             --gap[dep[u]];
97             dep[u] = min + 1;
98             ++gap[dep[u]];
99             if(u != start) u = edge[S[--top]].from;
100         }
101     }
102     return res;

```

### 3.8 MCMF

```

1  const int N = 305, M = 100005;
2  int head[N];
3  struct Edge {
4      int nxt, to, cow, cost;
5  } Edge();
6  Edge(int _nxt, int _to, int _cow, int _cost) {
7      nxt = _nxt; to = _to; cow = _cow; cost = _cost;
8  }
9  } ed[M];
10 int ecnt, mx_flow, mi_cost;
11 void init() {
12     mx_flow = mi_cost = ecnt = 0;

```



```

13     memset(head, -1, sizeof(head));
14 }
15 void addedge(int u, int v, int cow, int cost) {
16     ed[ecnt] = Edge(head[u], v, cow, cost);
17     head[u] = ecnt++;
18     ed[ecnt] = Edge(head[v], u, 0, -cost);
19     head[v] = ecnt++;
20 }
21 queue<int> Q;
22 int dis[N], pre[N], inq[N];
23 bool Spfa(int S, int T) {
24     memset(dis, 0x3f, sizeof(dis));
25     dis[S] = 0;
26     Q.push(S);
27     while(!Q.empty()) {
28         int u = Q.front(); Q.pop();
29         inq[u] = 0;
30         for(int e = head[u]; ~e; e = ed[e].nxt) {
31             if(!ed[e].cow) continue;
32             int v = ed[e].to;
33             if(dis[v] > dis[u] + ed[e].cost) {
34                 dis[v] = dis[u] + ed[e].cost;
35                 pre[v] = e;
36                 if(!inq[v]) {
37                     inq[v] = 1;
38                     Q.push(v);
39                 }
40             }
41         }
42     }
43     return dis[T] != INF;
44 }
45 void End(int S, int T) {
46     int flow = INF;
47     for(int u = T; u != S; u = ed[pre[u] ^ 1].to) {
48         flow = min(flow, ed[pre[u]].cow);
49     }
50     for(int u = T; u != S; u = ed[pre[u] ^ 1].to) {
51         ed[pre[u]].cow -= flow;
52         ed[pre[u] ^ 1].cow += flow;
53         mi_cost += flow * ed[pre[u]].cost;
54     }
55     mx_flow += flow;
56 }

```

### 3.9 ZKW MCMF

```

1  const int maxn = 105, maxm = 10005;
2  struct MaxFlow {
3      int size, n;
4      int st, en, maxflow, mincost;
5      bool vis[maxn];
6      int net[maxn], pre[maxn], cur[maxn], dis[maxn];
7      std::queue<int> Q;
8      struct EDGE {
9          int v, cap, cost, next;
10         EDGE() {}
11         EDGE(int a, int b, int c, int d) {
12             v = a, cap = b, cost = c, next = d;
13         }
14     } E[maxn];
15     void init(int _n) {
16         n = _n, size = 0;
17         memset(net, -1, sizeof(net));
18     }
19     void add(int u, int v, int cap, int cost) {
20         E[size] = EDGE(v, cap, cost, net[u]);
21         net[u] = size++;
22         E[size] = EDGE(u, 0, -cost, net[v]);
23         net[v] = size++;
24     }
25     bool modell() {
26         int v, min = INF;
27         for(int i = 0; i <= n; i++) {
28             if(!vis[i])
29                 continue;
30             for(int j = net[i]; v = E[j].v, j != -1; j =
31                 E[j].next)
32                 if(E[j].cap

```

```

32                 if(!vis[v] && dis[v] - dis[i] + E[j].cost
33                     < min)
34                     min = dis[v] - dis[i] + E[j].cost;
35             }
36             if(min == INF)
37                 return false;
38             for(int i = 0; i <= n; i++)
39                 if(vis[i])
40                     cur[i] = net[i], vis[i] = false, dis[i] +=
41                     min;
42             return true;
43         }
44     }
45     int augment(int i, int flow) {
46         if(i == en) {
47             mincost += dis[st] * flow;
48             maxflow += flow;
49             return flow;
50         }
51         vis[i] = true;
52         for(int j = cur[i], v = E[j].v, j != -1; j =
53             E[j].next) {
54             if(!E[j].cap)
55                 continue;
56             if(vis[v] || dis[v] + E[j].cost != dis[i])
57                 continue;
58             int delta = augment(v, std::min(flow, E[j].cap));
59             if(delta) {
60                 E[j].cap -= delta;
61                 E[j ^ 1].cap += delta;
62                 cur[i] = j;
63                 return delta;
64             }
65         }
66     }
67     void spfa() {
68         int u, v;
69         for(int i = 0; i <= n; i++)
70             vis[i] = false, dis[i] = INF;
71         dis[st] = 0;
72         Q.push(st);
73         vis[st] = true;
74         while(!Q.empty()) {
75             u = Q.front(); Q.pop();
76             vis[u] = false;
77             for(int i = net[u]; v = E[i].v, i != -1; i =
78                 E[i].next) {
79                 if(!E[i].cap || dis[v] <= dis[u] + E[i].cost)
80                     continue;
81                 dis[v] = dis[u] + E[i].cost;
82                 if(!vis[v]) {
83                     vis[v] = true;
84                     Q.push(v);
85                 }
86             }
87         }
88         for(int i = 0; i <= n; i++)
89             dis[i] = dis[en] - dis[i];
90     }
91     int zkw(int s, int t) {
92         st = s, en = t;
93         spfa();
94         mincost = maxflow = 0;
95         for(int i = 0; i <= n; i++)
96             vis[i] = false, cur[i] = net[i];
97         do {
98             while(augment(st, INF))
99                 mincost += maxflow * dis[st];
100             while(modell());
101         } while(modell());
102         return mincost;
103     }
104 } cf;

```

## 4 Math

### 4.1 EX GCD

```

1 void gcd(int a, int b, int &d, int &x, int &y) {
2     if(!b) {
3         d = a;
4         x = 1;
5         y = 0;
6     } else {
7         gcd(b, a % b, d, y, x);
8         y -= x * (a / b);
9     }
10 }
11 //ax + by = gcd(a, b)
12 //when gcd(a, b) = 1, ax = 1 (mod b)

```

## 4.2 FFT

```

1 using namespace std;
2 const int mod = 1e9 + 7;
3 const int max0 = 1 << 17;
4 struct comp {
5     double x, y;
6     comp() : x(0), y(0) {}
7     comp(const double &x, const double &y) : x(_x), y(_y)
8     {}
9 };
10 inline comp operator+(const comp &a, const comp &b) {
11     return comp(a.x + b.x, a.y + b.y);
12 }
13 inline comp operator-(const comp &a, const comp &b) {
14     return comp(a.x - b.x, a.y - b.y);
15 }
16 inline comp operator*(const comp &a, const comp &b) {
17     return comp(a.x * b.x - a.y * b.y, a.x * b.y + a.y *
18     b.x);
19 }
20 inline comp conj(const comp &a) {
21     return comp(a.x, -a.y);
22 }
23 const double PI = acos(-1);
24 int N, L;
25 comp w[max0 + 5];
26 int bitrev[max0 + 5];
27 void fft(comp *a, const int &n) {
28     for (int i = 0; i < n; ++i)
29         if (i < bitrev[i])
30             swap(a[i], a[bitrev[i]]);
31     for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>=
32     1)
33         for (int j = 0; j < n; j += i) {
34             comp *l = a + j, *r = a + j + (i >> 1), *p = w;
35             for (int k = 0; k < i >> 1; ++k) {
36                 comp tmp = *r * *p;
37                 *r = *l - tmp, *l = *l + tmp;
38                 ++l, ++r, p += lyc;
39             }
40         }
41     inline void fft_prepare() {
42         for (int i = 0; i < N; ++i)
43             bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (L -
44             1));
45         for (int i = 0; i < N; ++i)
46             w[i] = comp(cos(2 * PI * i / N), sin(2 * PI * i /
47             N));
48     }
49     inline vector<int> conv(const vector<int> &x, const
50     vector<int> &y) {
51         static comp a[max0 + 5], b[max0 + 5];
52         static comp dfta[max0 + 5], dftb[max0 + 5], dftc[max0 +
53         5], dftd[max0 + 5];
54         L = 0;
55         while ((1 << L) < x.size() + y.size() - 1)
56             ++L;
57         N = 1 << L;
58         fft_prepare();
59         for (int i = 0; i < N; ++i)
60             a[i] = b[i] = comp(0, 0);
61         for (int i = 0; i < x.size(); ++i)
62             a[i] = comp(x[i] & 32767, x[i] >> 15);
63         for (int i = 0; i < y.size(); ++i)

```

```

58         b[i] = comp(y[i] & 32767, y[i] >> 15);
59         fft(a, N), fft(b, N);
60         for (int i = 0; i < N; ++i) {
61             int j = (N - i) & (N - 1);
62             static comp da, db, dc, dd;
63             da = (a[i] + conj(a[j])) * comp(0.5, 0);
64             db = (a[i] - conj(a[j])) * comp(0, -0.5);
65             dc = (b[i] + conj(b[j])) * comp(0.5, 0);
66             dd = (b[i] - conj(b[j])) * comp(0, -0.5);
67             dfta[j] = da * dc;
68             dftb[j] = da * dd;
69             dftc[j] = db * dc;
70             dftd[j] = db * dd;
71         }
72         for (int i = 0; i < N; ++i)
73             a[i] = dfta[i] + dftb[i] * comp(0, 1);
74         for (int i = 0; i < N; ++i)
75             b[i] = dftc[i] + dftd[i] * comp(0, 1);
76         fft(a, N), fft(b, N);
77         vector<int> z(x.size() + y.size() - 1);
78         for (int i = 0; i < x.size() + y.size() - 1; ++i) {
79             int da = (long long)(a[i].x / N + 0.5) % mod;
80             int db = (long long)(a[i].y / N + 0.5) % mod;
81             int dc = (long long)(b[i].x / N + 0.5) % mod;
82             int dd = (long long)(b[i].y / N + 0.5) % mod;
83             z[i] = (da + ((long long)(db + dc) << 15) + ((long
84             long)dd << 30)) % mod;
85         }
86         return z;
87     }
88 }

```

## 4.3 FFT Normal

```

1 const double PI = acos(-1.0);
2 struct Virt {
3     double r, i;
4     Virt(double r = 0.0, double i = 0.0) {
5         this->r = r;
6         this->i = i;
7     }
8     Virt operator + (const Virt &x) {
9         return Virt(r + x.r, i + x.i);
10    }
11    Virt operator - (const Virt &x) {
12        return Virt(r - x.r, i - x.i);
13    }
14    Virt operator * (const Virt &x) {
15        return Virt(r * x.r - i * x.i, i * x.r + r * x.i);
16    }
17 };
18 void Rader(Virt F[], int len) {
19     int i, j, k;
20     for(i = 1, j = len / 2; i < len - 1; i++) {
21         if(i < j) swap(F[i], F[j]);
22         k = len / 2;
23         while(j >= k) {
24             j -= k;
25             k >>= 1;
26         }
27         if(j < k) j += k;
28     }
29 }
30 void FFT(Virt F[], int len, int on) {
31     Rader(F, len);
32     for(int h = 2; h <= len; h <<= 1) {
33         Virt wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI /
34         h));
35         for(int j = 0; j < len; j += h) {
36             Virt w(1, 0);
37             for(int k = j; k < j + h / 2; k++) {
38                 Virt u = F[k];
39                 Virt t = w * F[k + h / 2];
40                 F[k] = u + t;
41                 F[k + h / 2] = u - t;
42                 w = w * wn;
43             }
44         }
45         if(on == -1)

```

```

46     for(int i = 0; i < len; i++)
47         F[i].r /= len;
48 }

```

## 4.4 FNT All

```

1  const int P = 258280327, G = 5;
2  struct Number_Theory_Transform {
3      #define size 531441
4      int N, *W, w[2][size], tmp[size];
5      int pow_mod(ll a, int b) {
6          int c = 1;
7          while(b) {
8              if(b & 1) c = c * a % P;
9              b >>= 1;
10             a = a * a % P;
11         }
12         return c;
13     }
14     void prepare(int n) {
15         if(N == n) return;
16         N = n;
17         int x = pow_mod(G, (P - 1) / n);
18         int y = pow_mod(x, P - 2);
19         w[0][0] = w[1][0] = 1;
20         for(int i = 1; i < n; ++i) {
21             w[0][i] = (ll)w[0][i - 1] * x % P;
22             w[1][i] = (ll)w[1][i - 1] * y % P;
23         }
24     }
25     void work(int *A, int n) {
26         if(n == 1) return;
27         int i, j, k = 0, l, x, m, u = W[N / n], w = 1;
28         for(x = 2; n % x; ++x); m = n / x;
29         for(i = 0; i < x; ++i) {
30             for(j = i; j < n; j += x) {
31                 tmp[k++] = A[j];
32             }
33         }
34         for(i = 0; i < n; ++i) A[i] = tmp[i];
35         for(i = 1; i < x; ++i, l += m) work(A + l, m);
36         for(i = j = 0; i < n; ++i) {
37             for(l = j + x * m - m, k = x, tmp[i] = 0; k --
38                 k, l -= m) {
39                 tmp[i] = ((ll)w * tmp[i] + A[l]) % P;
40             }
41             w = (ll) w * u % P;
42             if(++j == m) j = 0;
43         }
44         for(int i = 0; i < n; ++i) A[i] = tmp[i];
45     }
46     void DFT(int *A, int n) {
47         prepare(n);
48         W = w[0];
49         work(A, n);
50     }
51     void IDFT(int *A, int n) {
52         prepare(n);
53         W = w[1];
54         work(A, n);
55         for(int i = 0, x = pow_mod(n, P - 2); i < n; ++i) {
56             A[i] = (ll)A[i] * x % P;
57         }
58     }
59 #undef size
60 } NTT;

```

## 4.5 FNT Base2

```

1  //1004535809 3
2  //211812353 3
3  //10000000025100289 22
4  //11000000009994241 17
5  //100000000135659521 3
6  //mod:prime g:prime root
7  const int mod = 998244353, g = 3;
8  int _g[30], _ig[30];

```

```

9  LL invl;
10 int pow_mod(LL a, int b) {
11     LL c = 1;
12     while(b) {
13         if(b & 1) c = c * a % mod;
14         b >>= 1;
15         a = a * a % mod;
16     }
17     return c;
18 }
19 void init() {
20     for(int i = 0; i < 30; ++i) {
21         _g[i] = pow_mod(g, (mod - 1) / (1 << i));
22         _ig[i] = pow_mod(_g[i], mod - 2);
23     }
24 }
25 void FNT(int F[], int len, int f) {
26     int i, j, k, cnt = 1;
27     LL x, y, w = 1, wn;
28     for(i = 1, j = len >> 1; i < len - 1; ++i) {
29         if(i < j) swap(F[i], F[j]);
30         k = len >> 1;
31         while(j >= k) j ^= k, k >>= 1;
32         j |= k;
33     }
34     for(i = 1; i < len; i <= 1) {
35         wn = f ? _g[cnt++] : _ig[cnt++];
36         for(j = 0; j < len; j += i << 1, w = 1) {
37             for(k = j; k < j + i; ++k, w = w * wn % mod) {
38                 x = F[k]; y = w * F[k + i] % mod;
39                 F[k] = x + y;
40                 F[k + i] = x - y;
41                 if(F[k] >= mod) F[k] -= mod;
42                 if(F[k + i] < 0) F[k + i] += mod;
43             }
44         }
45     }
46     if(!f) {
47         for(i = 0; i < len; ++i) {
48             F[i] = F[i] * invl % mod;
49         }
50     }
51 }
52 int a[150000], b[150000];
53 void conv(int ca[], int l1, int cb[], int l2, int c[], int
54     &l) {
55     l = 1; while(l < l1 + l2) l <= 1;
56     init(); invl = pow_mod(l, mod - 2);
57     for(int i = 0; i < l; ++i) {
58         a[i] = i < l1 ? ca[i] : 0;
59         b[i] = i < l2 ? cb[i] : 0;
60     }
61     FNT(a, l, 1); FNT(b, l, 1);
62     for(int i = 0; i < l; ++i) a[i] = (ll)a[i] * b[i] % mod;
63     FNT(a, l, 0);
64     for(int i = 0; i < l; ++i) c[i] = a[i];
65 }

```

## 4.6 FNT Base3

```

1  const int P = 258280327, G = 5, B = 3;
2  const int N = 531441;
3  int pow_mod(ll a, int b) {
4      int c = 1;
5      while(b) {
6          if(b & 1) c = c * a % P;
7          b >>= 1;
8          a = a * a % P;
9      }
10     return c;
11 }
12 int w[2][N], rev[N];
13 void init() {
14     ll t = pow_mod(G, (P - 1) / N);
15     w[0][0] = w[1][0] = 1;
16     for(int i = 1; i < N; ++i) {
17         w[0][i] = w[0][i - 1] * t % P;
18     }
19     for(int i = 1; i < N; ++i) {

```

```

20     w[1][i] = w[0][N - i];
21 }
22 for(int i = 0; i < N; ++ i) {
23     for(int j = 1; j < N; j *= B) {
24         (rev[i] *= B) += i / j % B;
25     }
26 }
27 }
28 void ntt(int *a, int n, int o) {
29     int tt = N / n, d = N / B;
30     for(int i = 0; i < n; ++ i) {
31         int j = rev[i] / tt;
32         if(i < j) swap(a[i], a[j]);
33     }
34     for(int i = 1; i < n; i *= B) {
35         for(int j = 0, t = N / (i * B); j < n; j += i * B) {
36             for(int k = 0, l = 0; k < i; ++ k, l += t) {
37                 int x = a[j + k], y = a[j + k + i], z = a[j
38                     + k + i + i];
39                 a[j + k] = (x + (ll)y * w[o][l] + (ll)z *
40                     w[o][l + l]) % P;
41                 a[j + k + i] = (x + (ll)y * w[o][l + d] +
42                     (ll)z * w[o][(l + l + d + d) % N]) % P;
43                 a[j + k + i + i] = (x + (ll)y * w[o][l + d +
44                     d] + (ll)z * w[o][(l + d + d) * 2 - N])
45                     % P;
46             }
47         }
48     }
49     if(o == 1) {
50         ll inv = pow_mod(n, P - 2);
51         for(int i = 0; i < n; ++ i) {
52             a[i] = a[i] * inv % P;
53         }
54     }
55 }
56 int getlen(int n) {
57     int r = 1;
58     while(r < n) r *= B;
59     return r;
60 }

```

## 4.7 FWT XOR

```

1 inline int ck(int x) {
2     if(x < 0) x += mod;
3     if(x >= mod) x -= mod;
4     return x;
5 }
6 int tmp[1 << 16];
7 void tf(int a[], int ta[], int n) {
8     if(n == 1) {
9         ta[0] = a[0];
10        return;
11    }
12    int x = n >> 1;
13    tf(a, ta, x);
14    tf(a + x, ta + x, x);
15    for(int i = 0; i < x; ++ i) {
16        tmp[i] = ck(ta[i] - ta[x + i]);
17        tmp[x + i] = ck(ta[i] + ta[x + i]);
18    }
19    for(int i = 0; i < n; ++ i) ta[i] = tmp[i];
20 }
21 LL inv2;
22 void utf(int a[], int ta[], int n) {
23     if(n == 1) {
24         ta[0] = a[0];
25         return;
26     }
27     int x = n >> 1;
28     for(int i = 0; i < x; ++ i) {
29         tmp[i] = (a[i] + a[i + x]) * inv2 % mod;
30         tmp[i + x] = (a[i + x] - a[i] + mod) * inv2 % mod;
31     }
32     for(int i = 0; i < n; ++ i) a[i] = tmp[i];
33     utf(a, ta, x);
34     utf(a + x, ta + x, x);
35 }

```

## 4.8 Miller Rabin-Pollard rho

```

1 const int S = 20;
2 long long mult_mod(long long a, long long b, long long c) {
3     a %= c;
4     b %= c;
5     long long ret = 0;
6     while(b) {
7         if(b & 1) {
8             ret += a;
9             ret %= c;
10        }
11        a <<= 1;
12        if(a >= c) a %= c;
13        b >>= 1;
14    }
15    return ret;
16 }
17 long long pow_mod(long long x, long long n, long long mod)
18 { //x^n%c
19     if(n == 1) return x % mod;
20     x %= mod;
21     long long tmp = x;
22     long long ret = 1;
23     while(n) {
24         if(n & 1) ret = mult_mod(ret, tmp, mod);
25         tmp = mult_mod(tmp, tmp, mod);
26         n >>= 1;
27     }
28     return ret;
29 }
30 bool check(long long a, long long n, long long x, long long
31     t) {
32     long long ret = pow_mod(a, x, n);
33     long long last = ret;
34     for(int i = 1; i <= t; i++) {
35         ret = mult_mod(ret, ret, n);
36         if(ret == 1 && last != 1 && last != n - 1) return
37             true;
38         last = ret;
39     }
40     return false;
41 }
42 bool Miller_Rabin(long long n) {
43     if(n < 2) return false;
44     if(n == 2) return true;
45     if((n & 1) == 0) return false; // even number
46     long long x = n - 1;
47     long long t = 0;
48     while((x & 1) == 0) {
49         x >>= 1;
50         t++;
51     }
52     for(int i = 0; i < S; i++) {
53         long long a = rand() % (n - 1) + 1;
54         if(check(a, n, x, t))
55             return false;
56     }
57     return true;
58 }
59 long long factor[100];
60 int tol;
61 long long gcd(long long a, long long b) {
62     if(a == 0) return 1;
63     if(a < 0) return gcd(-a, b);
64     while(b) {
65         long long t = a % b;
66         a = b;
67         b = t;
68     }
69     return a;
70 }
71 long long Pollard_rho(long long x, long long c) {
72     long long i = 1, k = 2;
73     long long x0 = rand() % x;
74     long long y = x0;
75     while(1) {
76         i++;
77         x0 = (mult_mod(x0, x0, x) + c) % x;
78         y = (mult_mod(y, y, x) + c) % x;
79         long long d = gcd(x0 - y, x);
80         if(d > 1) return d;
81         if(x0 == y) return x0;
82         k++;
83     }
84 }

```

```

76     long long d = gcd(y - x0, x);
77     if(d != 1 && d != x) return d;
78     if(y == x0) return x;
79     if(i == k) {
80         y = x0;
81         k += k;
82     }
83 }
84 }
85 void findfac(long long n) {
86     if(n == 1) return;
87     if(Miller_Rabin(n)) {
88         factor[tol++] = n;
89         return;
90     }
91     long long p = n;
92     while(p >= n)p = Pollard_rho(p, rand() % (n - 1) + 1);
93     findfac(p);
94     findfac(n / p);
95 }
96 int main() {
97     srand(time(NULL));
98     long long n;
99     while(scanf("%I64d", &n) != EOF) {
100         tol = 0;
101         findfac(n);
102         for(int i = 0; i < tol; i++)printf("%I64d ",
103             factor[i]);
104         printf("\n");
105         if(Miller_Rabin(n))printf("Yes\n");
106         else printf("No\n");
107     }
108     return 0;
109 }

```

## 4.9 Romberg

```

1  const int MAX = 18;
2  double f(double x) {
3
4  }
5  double Romberg (double a, double b) {
6      #define MAX_N 18
7      int i, j, temp2, min;
8      double h, R[2][MAX_N], temp4;
9      for (i = 0; i < MAX_N; i++) {
10         R[0][i] = 0.0;
11         R[1][i] = 0.0;
12     }
13     h = b - a;
14     min = (int)(log(h * 10.0) / log(2.0)); //h should be at
15         most 0.1
16     R[0][0] = (f(a) + f(b)) * h * 0.50;
17     i = 1;
18     temp2 = 1;
19     while (i < MAX_N) {
20         i++;
21         R[1][0] = 0.0;
22         for (j = 1; j <= temp2; j++)
23             R[1][j] += f(a + h * ((double)j - 0.50));
24         R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
25         temp4 = 4.0;
26         for (j = 1; j < i; j++) {
27             R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
28             temp4 *= 4.0;
29         }
30         if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min))
31             return R[1][i - 1];
32         h *= 0.50;
33         temp2 *= 2;
34         for (j = 0; j < i; j++)
35             R[0][j] = R[1][j];
36     }
37     return R[1][MAX_N - 1];
38 }

```

## 4.10 Simplex

```

1  //a[i][0]*x[0] + a[i][1]*x[1] + ... <= a[i][n]
2  //max(a[m][0]*x[0] + a[m][1]*x[1] + ... + a[m][n-1]*x[n-1]
3  //x[i] >= 0
4  const int maxm = 500; // st
5  const int maxn = 500; // var
6  const double INF = 1e100;
7  const double eps = 1e-10;
8
9  struct Simplex {
10     int n; // var
11     int m; // st
12     double a[maxm][maxn]; // input matrix
13     int B[maxm], N[maxn];
14
15     void pivot(int r, int c) {
16         swap(N[c], B[r]);
17         a[r][c] = 1 / a[r][c];
18         for(int j = 0; j <= n; j++) if(j != c) a[r][j] *=
19             a[r][c];
20         for(int i = 0; i <= m; i++) if(i != r) {
21             for(int j = 0; j <= n; j++) if(j != c) a[i][j] -=
22                 a[i][c] * a[r][j];
23             a[i][c] = -a[i][c] * a[r][c];
24         }
25     }
26     bool feasible() {
27         for(;;) {
28             int r, c;
29             double p = INF;
30             for(int i = 0; i < m; i++) if(a[i][n] < p) p = a[r = i][n];
31             if(p > -eps) return true;
32             p = 0;
33             for(int i = 0; i < n; i++) if(a[r][i] < p) p = a[r][c = i];
34             if(p > -eps) return false;
35             p = a[r][n] / a[r][c];
36             for(int i = r+1; i < m; i++) if(a[i][c] > eps) {
37                 double v = a[i][n] / a[i][c];
38                 if(v < p) { r = i; p = v; }
39             }
40             pivot(r, c);
41         }
42     }
43     // 0: no solution, -1: unlimit solution, 1: limit solution
44     // b[i] = x[i], ret: opt value
45     int simplex(int n, int m, double x[maxn], double& ret) {
46         this->n = n;
47         this->m = m;
48         for(int i = 0; i < n; i++) N[i] = i;
49         for(int i = 0; i < m; i++) B[i] = n+i;
50         if(!feasible()) return 0;
51         for(;;) {
52             int r, c;
53             double p = 0;
54             for(int i = 0; i < n; i++) if(a[m][i] > p) p = a[m][c = i];
55             if(p < eps) {
56                 for(int i = 0; i < n; i++) if(N[i] < n) x[N[i]] = 0;
57                 for(int i = 0; i < m; i++) if(B[i] < n) x[B[i]] =
58                     a[i][n];
59                 ret = -a[m][n];
60                 return 1;
61             }
62             p = INF;
63             for(int i = 0; i < m; i++) if(a[i][c] > eps) {
64                 double v = a[i][n] / a[i][c];
65                 if(v < p) { r = i; p = v; }
66             }
67             if(p == INF) return -1;
68             pivot(r, c);
69         }
70     }
71 };

```

## 5 String

### 5.1 KMP

```

1 void getNext(char *p, int *next) {
2     int j = 0, k = -1;
3     next[0] = -1;
4     int len = strlen(p);
5     while(j < len) {
6         if(k == -1 || p[j] == p[k]) {
7             ++j; ++k;
8             next[j] = k;
9         }
10        else k = next[k];
11    }
12 }
13 int KMPMatch(char *s, char *p) {
14     int i = 0, j = 0;
15     int len = strlen(s), lenp = strlen(p);
16     while(i < len) {
17         if(j == -1 || s[i] == p[j]) {
18             ++i;
19             ++j;
20         }
21         else j = next[j];
22         if(j == lenp) return i - lenp;
23     }
24     return -1;
25 }

```

### 5.2 EX KMP

```

1 const int N = 1e5 + 5;
2 int next[N], extand[N];
3 void getNext(char *T) {
4     int i, length = strlen(T);
5     next[0] = length;
6     for(i = 0; i < length - 1; i++) {
7         next[i] = i;
8         int a = 1;
9         for(int k = 2; k < length; k++) {
10            int p = a + next[a] - 1, L = next[k - a];
11            if((k - 1) + L >= p) {
12                int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
13                while(k + j < length && T[k + j] == T[j]) j++;
14                next[k] = j, a = k;
15            }
16            else next[k] = L;
17        }
18    }
19 void getextand(char *S, char *T) {
20     memset(next, 0, sizeof(next));
21     getNext(T);
22     int Slen = strlen(S), Tlen = strlen(T), a = 0;
23     int MinLen = Slen > Tlen ? Tlen : Slen;
24     while(a < MinLen && S[a] == T[a]) a++;
25     extand[0] = a, a = 0;
26     for(int k = 1; k < Slen; k++) {
27         int p = a + extand[a] - 1, L = next[k - a];
28         if((k - 1) + L >= p) {
29             int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
30             while(k + j < Slen && j < Tlen && S[k + j] == T[j]) j++;
31             extand[k] = j, a = k;
32         }
33         else extand[k] = L;
34     }
35 }

```

### 5.3 Suffix Array

```

1 struct Suffix_Array {
2     int wa[N], wb[N], wv[N], wd[N];
3     inline int cmp(int *r, int a, int b, int l) {
4         return r[a] == r[b] && r[a + l] == r[b + l];

```

```

5     }
6     void da(int *r, int *sa, int n, int m) {
7         int i, j, p, *x = wa, *y = wb, *t;
8         for(i = 0; i < m; ++i) wd[i] = 0;
9         for(i = 0; i < n; ++i) wd[x[i]] = r[i]++;
10        for(i = 1; i < m; ++i) wd[i] += wd[i - 1];
11        for(i = n - 1; i >= 0; --i) sa[--wd[x[i]]] = i;
12        for(j = 1, p = 1; p < n; j *= 2, m = p) {
13            for(p = 0, i = n - j; i < n; ++i) y[p++] = i;
14            for(i = 0; i < n; ++i) if(sa[i] >= j) y[p++] = sa[i] - j;
15            for(i = 0; i < n; ++i) wv[i] = x[y[i]];
16            for(i = 0; i < m; ++i) wd[i] = 0;
17            for(i = 0; i < n; ++i) wd[wv[i]]++;
18            for(i = 1; i < m; ++i) wd[i] += wd[i - 1];
19            for(i = n - 1; i >= 0; --i) sa[--wd[wv[i]]] = y[i];
20            for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; ++i)
21                x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
22        }
23    }
24    int rank[N], height[N], data[N], sa[N];
25    void calheight(int *r, int *sa, int n) {
26        int i, j, k = 0;
27        for(i = 1; i <= n; ++i) rank[sa[i]] = i;
28        for(i = 0; i < n; height[rank[i++]] = k)
29            for(k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; ++k);
30    }
31    int minx[N][20], Lg[N];
32    void initRMQ(int n) {
33        Lg[1] = 0;
34        for(int i = 2; i <= n; ++i) Lg[i] = Lg[i >> 1] + 1;
35        for(int j = 1; j < 20; ++j) {
36            for(int i = 1; i <= n; ++i) {
37                if(i + (1 << j) - 1 > n) break;
38                minx[i][j] = min(minx[i][j - 1], minx[i + (1 << (j - 1))] [j - 1]);
39            }
40        }
41    }
42    inline int lcp(int l, int r) {
43        int t = Lg[r - l + 1];
44        return min(minx[l][t], minx[r - (1 << t) + 1][t]);
45    }
46    int len;
47    void work(char *s) {
48        len = strlen(s);
49        for(int i = 0; i < len; ++i) data[i] = s[i];
50        data[len] = 0;
51        da(data, sa, len + 1, 128);
52        calheight(data, sa, len);
53        for(int i = 1; i <= len; ++i) minx[i][0] = height[i];
54        initRMQ(len);
55    }
56 } sa;

```

### 5.4 Aho-Corasick Automaton

```

1 const int N = 1e5 + 5;
2 struct Trie {
3     Trie *next[26];
4     Trie *fail;
5 } pool[N], *root;
6 int tot;
7 Trie* newnode() {
8     Trie *t = pool + (tot++);
9     memset(t->next, 0, sizeof(t->next));
10    t->fail = NULL;
11    return t;
12 }
13 void init() {
14     tot = 0;
15     root = newnode();
16 }
17 void Insert(char *s) {

```

```

18     Trie *p = root;
19     for(int i = 0; s[i]; ++i) {
20         int k = s[i] - 'a';
21         if(!p->next[k]) p->next[k] = newnode();
22         p = p->next[k];
23     }
24 }
25 queue <Trie*> Q;
26 void Build_Ac() {
27     Trie *p, *temp;
28     Q.push(root);
29     while(!Q.empty()) {
30         temp = Q.front(); Q.pop();
31         for(int i = 0; i < 26; ++i) if(temp->next[i]) {
32             p = temp->fail;
33             while(p) {
34                 if(p->next[i]) {
35                     temp->next[i]->fail = p->next[i];
36                     break;
37                 }
38                 p = p->fail;
39             }
40             if(!p) temp->next[i]->fail = root;
41             Q.push(temp->next[i]);
42         }
43     }
44     temp->next[i] = temp->fail ? temp->fail->next[i]
45         : root;
46 }
47 }

```

## 5.5 Minimum Representation

```

1 int minP(char s[])
2 {
3     int l=strlen(s);
4     int i = 0, j = 1, k = 0;
5     while (1)
6     {
7         if (i + k >= l || j + k >= l) break;
8         if (s[i + k] == s[j + k])
9         {
10             k++;
11             continue;
12         }
13         else
14         {
15             if (s[j + k] > s[i + k]) j += k + 1;
16             else i += k + 1;
17             k = 0;
18             if (i == j) j++;
19         }
20     }
21     return min(i, j);
22 }

```

## 5.6 Manacher

```

1 const int N = 2e5 + 5;
2 char s[N], str[N];
3 int p[N];
4 int Manacher(char *s) {
5     str[0] = '$';
6     int cc = 1;
7     for(int i = 0; s[i]; ++i) {
8         str[cc++] = '#';
9         str[cc++] = s[i];
10    }
11    str[cc++] = '#';
12    str[cc] = 0;
13    int mx = 0, id;
14    for(int i = 1; str[i]; ++i) {
15        if(mx > i) {
16            p[i] = min(p[2 * id - i], mx - i);
17        }
18        else p[i] = 1;

```

```

19        for(; str[i + p[i]] == str[i - p[i]]; ++p[i]);
20        if(p[i] + i > mx) {
21            mx = p[i] + i;
22            id = i;
23        }
24    }
25    return cc;
26 }

```

## 5.7 Palindromic Tree AF

```

1 const int MAXN = 200005;
2 const int base = 100002;
3 const int N = 26;
4
5 struct Palindromic_Tree {
6     int next[MAXN][N];
7     int fail[MAXN];
8     int num[MAXN];
9     int len[MAXN];
10    int S[MAXN];
11    int suflast, prelast;
12    int L, R;
13    int p;
14    int newnode(int l) {
15        for(int i = 0; i < N; ++i) next[p][i] = 0;
16        num[p] = 0;
17        len[p] = l;
18        return p++;
19    }
20    void init() {
21        p = 0;
22        newnode(0);
23        newnode(-1);
24        suflast = prelast = 0;
25        L = base + 1; R = base;
26        fail[0] = 1;
27    }
28    int get_back_fail(int x) {
29        while(R - len[x] - 1 < L || S[R - len[x] - 1] !=
30            S[R]) x = fail[x];
31        return x;
32    }
33    int get_front_fail(int x) {
34        while(L + len[x] + 1 > R || S[L + len[x] + 1] !=
35            S[L]) x = fail[x];
36        return x;
37    }
38    void add_back(int c) {
39        c -= 'a';
40        S[++R] = c;
41        int cur = get_back_fail(suflast);
42        if(!next[cur][c]) {
43            int now = newnode(len[cur] + 2);
44            fail[now] = next[get_back_fail(fail[cur])][c];
45            next[cur][c] = now;
46            num[now] = num[fail[now]] + 1;
47        }
48        suflast = next[cur][c];
49        if(len[suflast] == R - L + 1) prelast = suflast;
50    }
51    void add_front(int c) {
52        c -= 'a';
53        S[--L] = c;
54        int cur = get_front_fail(prelast);
55        if(!next[cur][c]) {
56            int now = newnode(len[cur] + 2);
57            fail[now] = next[get_front_fail(fail[cur])][c];
58            next[cur][c] = now;
59            num[now] = num[fail[now]] + 1;
60        }
61        prelast = next[cur][c];
62        if(len[prelast] == R - L + 1) suflast = prelast;
63    }
64 } T;

```

## 5.8 Suffix Automaton



```

1  const int N = 1e5 + 5;
2  struct Sam {
3      Sam *next[26], *par;
4      int step;
5  } pool[N * 2], *root, *last;
6  int tot;
7  Sam* newnode(int step) {
8      Sam *t = pool + (tot++);
9      memset(t->next, 0, sizeof(t->next));
10     t->par = NULL;
11     t->step = step;
12     return t;
13 }
14 void init() {
15     tot = 0;
16     last = root = newnode(0);
17 }
18 void Extend(int w) {
19     Sam *p = last;
20     Sam *neww = newnode(p->step + 1);
21     for(; p && !p->next[w]; p = p->par) p->next[w] = neww;
22     if(!p) neww->par = root;
23     else {
24         Sam *q = p->next[w];
25         if(q->step == p->step + 1) neww->par = q;
26         else {
27             Sam *nq = newnode(p->step + 1);
28             memcpy(nq->next, q->next, sizeof(q->next));
29             nq->par = q->par;
30             q->par = nq;
31             neww->par = nq;
32             for(; p && p->next[w] == q; p = p->par)
33                 p->next[w] = nq;
34         }
35     }
36     last = neww;

```

## 5.9 Palindromic Factorization

```

1  const int MAXN = 300005;
2  const int N = 26;
3  const int inf = 0x3f3f3f3f;
4  struct Palindromic_Tree {
5      int nxt[MAXN][N], fail[MAXN];
6      int occ[MAXN], num[MAXN], len[MAXN];
7      int S[MAXN], last, n, p;
8      int sfail[MAXN], diff[MAXN], dp[2][MAXN], ans[2][MAXN];
9      int newnode(int l) {
10         memset(nxt[p], 0, N * sizeof(int));
11         occ[p] = num[p] = 0;
12         len[p] = l;
13         return p++;
14     }
15     void init() {
16         p = 0;
17         newnode(0);
18         newnode(-1);
19         last = 0;
20         n = 0;
21         S[n] = -1;
22         fail[0] = 1;
23         ans[0][0] = 0;
24         ans[1][0] = inf;
25     }
26     int get_fail(int x) {
27         while(S[n - len[x] - 1] != S[n]) x = fail[x];
28         return x;
29     }
30     void add(int c) {
31         c -= 'a';
32         S[++n] = c;
33         int cur = get_fail(last);
34         if(!nxt[cur][c]) {
35             int v = newnode(len[cur] + 2);
36             fail[v] = nxt[get_fail(fail[cur])][c];
37             nxt[cur][c] = v;
38             num[v] = num[fail[v]] + 1;
39             diff[v] = len[v] - len[fail[v]];

```

```

40         sfail[v] = diff[v] ^ diff[fail[v]] ? fail[v] :
41             sfail[fail[v]];
42     }
43     last = nxt[cur][c];
44     occ[last]++;
45     update();
46 }
47 void update() {
48     ans[0][n] = ans[1][n] = inf;
49     for(int u = last; u; u = sfail[u]) {
50         dp[0][u] = ans[1][n - len[fsfail[u]] - diff[u]];
51         dp[1][u] = ans[0][n - len[fsfail[u]] - diff[u]];
52         if(diff[u] == diff[fail[u]]) {
53             dp[0][u] = min(dp[0][u], dp[0][fail[u]]);
54             dp[1][u] = min(dp[1][u], dp[1][fail[u]]);
55         }
56         ans[0][n] = min(ans[0][n], dp[0][u] + 1);
57         ans[1][n] = min(ans[1][n], dp[1][u] + 1);
58     }
59 }
60 void count() {
61     for(int i = p - 1; i >= 0; --i) occ[fail[i]] +=
62         occ[i];

```

## 5.10 Suffix Tree

```

1  const int SIGMA = 26;
2  const int N = 100005;
3  int alloc, curPos, actEdge, actLen, remaind;
4  struct node {
5      int l, r, son;
6      node *nxt[SIGMA], *slink, *fa;
7      inline int edgeLen() {
8          return min(r, curPos + 1) - l;
9      }
10 } S[N + N], *root, *actNode, *needSL;
11 inline node* newnode(int l, int r = INF) {
12     node *t = S + (alloc++);
13     t->l = l; t->r = r;
14     t->slink = t->fa = 0;
15     t->son = 0;
16     memset(t->nxt, 0, sizeof(t->nxt));
17     return t;
18 }
19 int text[N];
20 inline int actedge() {
21     return text[actEdge];
22 }
23 inline void addSL(node *p) {
24     if(needSL) needSL->slink = p;
25     needSL = p;
26 }
27 bool walkDown(node *p) {
28     if(actLen < p->edgeLen()) return false;
29     actEdge += p->edgeLen();
30     actLen -= p->edgeLen();
31     actNode = p;
32     return true;
33 }
34 void doneins() {
35     --remaind;
36     if(actNode == root && actLen > 0) {
37         --actLen;
38         actEdge = curPos - remaind + 1;
39     } else {
40         actNode = actNode->slink ? actNode->slink : root;
41     }
42 }
43 int head, tail;
44 node* leaves[N + N];
45 ll curAns;
46 void init() {
47     curAns = head = tail = 0;
48     needSL = 0; alloc = 0; curPos = -1;
49     remaind = actEdge = actLen = 0;
50     root = actNode = newnode(-1, -1);
51 }

```

```

52 void extend(int c) {
53     text[++ curPos] = c;
54     needSL = 0;
55     ++ remaind;
56     curAns += tail - head;
57     while(remaind > 0) {
58         if(actLen == 0) actEdge = curPos;
59         if(actNode->nxt[actEdge()] == 0) {
60             node* leaf = newnode(curPos);
61             actNode->nxt[actEdge()] = leaf;
62             leaf->fa = actNode;
63             ++ actNode->son;
64             addSL(actNode);
65             leaves[tail++] = leaf;
66         } else {
67             node* nt = actNode->nxt[actEdge()];
68             if(walkDown(nt)) continue;
69             if(text[nt->l + actLen] == c) {
70                 ++ actLen;
71                 addSL(actNode);
72                 break;
73             }
74             node* split = newnode(nt->l, nt->l + actLen);
75             actNode->nxt[actEdge()] = split;
76             split->fa = actNode;
77             node* leaf = newnode(curPos);
78             split->nxt[c] = leaf;
79             leaf->fa = split;
80             nt->l += actLen;
81             split->nxt[text[nt->l]] = nt;
82             nt->fa = split;
83             addSL(split);
84             split->son = 2;
85             leaves[tail++] = leaf;
86         }
87         doneins();
88         ++ curAns;
89     }
90 }
91 void erasefront() {
92     while(actLen > 0 && actNode->nxt[actEdge()] &&
93           walkDown(actNode->nxt[actEdge()]));
94     node* u = leaves[head++], *f = u->fa;
95     while(u != root && u->son == 0 && actNode != f) {
96         curAns -= u->edgeLen();
97         f->nxt[text[u->l]] = 0;
98         -- f->son;
99         u = f; f = u->fa;
100     }
101     if(u == root || u->son > 0) return;
102     if(actLen == 0 || f->nxt[actEdge()] != u) {
103         curAns -= u->edgeLen();
104         f->nxt[text[u->l]] = 0;
105         if(-- f->son) return;
106         if(remaind) doneins();
107         if(f != root) {
108             leaves[tail++] = f;
109             f->l = curPos - f->edgeLen() + 1;
110             f->r = INF;
111         }
112     } else {
113         curAns -= u->edgeLen() - actLen;
114         u->l = curPos - actLen + 1;
115         u->r = INF;
116         doneins();
117         leaves[tail++] = u;
118     }

```

## 6 Others

### 6.1 Big Number

```

1 #include<iostream>
2 #include<string>
3 #include<iomanip>
4 #include<algorithm>

```

```

5 using namespace std;
6
7 #define MAXN 9999
8 #define DLEN 4
9
10 class BigNum{
11 private:
12     int a[300]; //DLEN digs for a position
13     int len;
14 public:
15     BigNum(){len = 1; memset(a, 0, sizeof(a));}
16     BigNum(const int b);
17     BigNum(const BigNum & T);
18
19     bool Bigger(const BigNum &) const;
20     BigNum & operator=(const BigNum &);
21     BigNum & Add(const BigNum &);
22     BigNum & Sub(const BigNum &);
23     BigNum operator+(const BigNum &) const;
24     BigNum operator-(const BigNum &) const;
25     BigNum operator*(const BigNum &) const;
26     BigNum operator/(const int &) const;
27     void Print();
28 };
29 BigNum::BigNum(const int b)
30 {
31     int c, d = b;
32
33     len = 0;
34     memset(a, 0, sizeof(a));
35     while(d > MAXN){
36         c = d - d / (MAXN + 1) * (MAXN + 1);
37         d = d / (MAXN + 1);
38         a[len++] = c;
39     }
40     a[len++] = d;
41 }
42 BigNum::BigNum(const BigNum & T) : len(T.len)
43 {
44     int i;
45     memset(a, 0, sizeof(a));
46     for(i = 0; i < len; i++)
47         a[i] = T.a[i];
48 }
49 bool BigNum::Bigger(const BigNum & T) const
50 {
51     int ln;
52     if(len > T.len) return true;
53     else if(len == T.len){
54         ln = len - 1;
55         while(a[ln] == T.a[ln] && ln >= 0) ln--;
56         if(ln >= 0 && a[ln] > T.a[ln]) return true;
57         else return false;
58     }
59     else return false;
60 }
61 BigNum & BigNum::operator=(const BigNum & n)
62 {
63     len = n.len;
64     memset(a, 0, sizeof(a));
65     for(int i = 0; i < len; i++)
66         a[i] = n.a[i];
67     return *this;
68 }
69 BigNum & BigNum::Add(const BigNum & T)
70 {
71     int i, big;
72
73     big = T.len > len ? T.len : len;
74     for(i = 0; i < big; i++)
75     {
76         a[i] = a[i] + T.a[i];
77         if(a[i] > MAXN)
78         {
79             a[i + 1]++;
80             a[i] = a[i] - MAXN - 1;
81         }
82     }
83     if(a[big] != 0) len = big + 1;
84     else len = big;
85 }

```

```

86     return *this;
87 }
88 BigNum & BigNum::Sub(const BigNum & T)
89 {
90     int i,j,big;
91
92     big = T.len > len ? T.len : len;
93     for(i = 0 ; i < big ; i++){
94         if(a[i] < T.a[i]){
95             j = i + 1;
96             while(a[j] == 0) j++;
97             a[j--]--;
98             while(j > i) a[j--] += MAXN;
99             a[i] = a[i] + MAXN + 1 - T.a[i];
100         }
101         else a[i] -= T.a[i];
102     }
103     len = big;
104     while(a[len - 1] == 0 && len > 1) len--;
105     return *this;
106 }
107 BigNum BigNum::operator+(const BigNum & n) const
108 {
109     BigNum a = *this;
110
111     a.Add(n);
112     return a;
113 }
114 BigNum BigNum::operator-(const BigNum & T) const
115 {
116     BigNum b = *this;
117
118     b.Sub(T);
119     return b;
120 }
121 BigNum BigNum::operator*(const BigNum & T) const
122 {
123     BigNum ret;
124     int i,j,up;
125     int temp,temp1;
126
127     for(i = 0 ; i < len ; i++){
128         up = 0;
129         for(j = 0 ; j < T.len ; j++){
130             temp = a[i] * T.a[j] + ret.a[i + j] + up;
131             if(temp > MAXN){
132                 temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
133                 up = temp / (MAXN + 1);
134                 ret.a[i + j] = temp1;
135             }
136             else {
137                 up = 0;
138                 ret.a[i + j] = temp;
139             }
140         }
141         if(up != 0)
142             ret.a[i + j] = up;
143     }
144     ret.len = i + j;
145     while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
146     return ret;
147 }
148 BigNum BigNum::operator/(const int & b) const
149 {
150     BigNum ret;
151     int i,down = 0;
152
153     for(i = len - 1 ; i >= 0 ; i--){
154         ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
155         down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
156     }
157     ret.len = len;
158     while(ret.a[ret.len - 1] == 0) ret.len--;
159     return ret;
160 }
161 void BigNum::Print()
162 {
163     int i;
164
165     cout << a[len - 1];
166     for(i = len - 2 ; i >= 0 ; i--){

```

```

167         cout.width(DLEN);
168         cout.fill('0');
169         cout << a[i];
170     }
171     cout << endl;
172 }

```

## 6.2 Fast IO

```

1 inline void R(int &x) {
2     char c; bool sign = false;
3     for (c = getchar(); c < '0' || c > '9'; c = getchar()) if
4         (c == '-') sign = true;
5     for (x = 0; c >= '0' && c <= '9'; c = getchar()) x =
6         x*10+c-'0';
7     sign && (x=-x);
8 }

```

## 6.3 Long Long Mul

```

1 long long Mul(long long x, long long y) {
2     return (x * y - (long long)(x / (long double)P * y +
3         1e-3) * P + P) % P;
4 }
5 LL mul_mod(LL x, LL y, LL n) { // x*y % n
6     LL T = floor(sqrt(n) + 0.5);
7     LL t = T * T - n;
8     LL a = x / T, b = x % T;
9     LL c = y / T, d = y % T;
10    LL e = a * c / T, f = a * c % T;
11    LL v = ((a * d + b * c) % n + e * t) % n;
12    LL g = v / T, h = v % T;
13    LL ret = (((f + g) * t % n + b * d) % n + h * T) %
14        n;
15    return (ret % n + n) % n;
16 }

```

## 6.4 Java

```

1 Scanner cin = new Scanner(new File("derangements.in"));
2 PrintWriter cout = new PrintWriter(new
3     File("derangements.out"));

```