

1 ComputationalGeometry

1.1 2D

```

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) {
52     ang = atan2(v.y, v.x);
53   }
54   Point point(double a) {
55     return p + (v * a);
56   }
57   bool operator<(const Line &L) const {
58     return ang < L.ang;
59   }
60 };
61 bool OnLeft(const Line &L, const Point &p) {
62   return Cross(L.v, p - L.P) > 0;
63 }
64 vector<Point> HalfplaneIntersection(vector<Line> L) {
65   int n = L.size();
66   sort(L.begin(), L.end());
67   int first, last;
68   vector<Point> p(n);
69   vector<Line> q(n);
70   vector<Point> ans;
71   q[first = last = 0] = L[0];
72   for (int i = 1; i < n; i++) {
73     while (first < last && !OnLeft(L[i], p[last - 1]))
74       last--;
75     while (first < last && !OnLeft(L[i], p[first])) first++;
76     q[++last] = L[i];
77     if (fabs(Cross(q[last].v, q[last - 1].v)) < eps) {
78       last--;
79       if (OnLeft(q[last], L[i].P)) q[last] = L[i];
80     }
81     if (first < last)
82       p[last - 1] = GetLineIntersection(q[last - 1],
83         q[last]);
84   }
85   while (first < last && !OnLeft(q[first], p[last - 1]))
86     last--;
87   if (last - first <= 1) return ans;
88   p[last] = GetLineIntersection(q[last], q[first]);
89   for (int i = first; i <= last; i++) ans.push_back(p[i]);
90   return ans;
91 }
92 Point PolyGravity(Point *p, int n) {
93   Point tmp, g = Point(0, 0);
94   double sumArea = 0, area;
95   for (int i = 2; i < n; ++i) {
96     area = Cross(p[i - 1] - p[0], p[i] - p[0]);
97     sumArea += area;
98     tmp.x = p[0].x + p[i - 1].x + p[i].x;
99     tmp.y = p[0].y + p[i - 1].y + p[i].y;
100     g.x += tmp.x * area;
101     g.y += tmp.y * area;
102   }
103   g.x /= (sumArea * 3.0);
104   g.y /= (sumArea * 3.0);
105   return g;
106 }
107 vector<Point> ConvexHull(vector<Point> &p) {
108   sort(p.begin(), p.end());
109   p.erase(unique(p.begin(), p.end()), p.end());
110   int n = p.size();
111   int m = 0;
112   vector<Point> ch(n + 1);
113   for (int i = 0; i < n; i++) {
114     while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] -
115       ch[m - 2]) <= 0)
116       m--;
117     ch[m++] = p[i];
118   }
119   int k = m;
120   for (int i = n - 2; i >= 0; i--) {
121     while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] -
122       ch[m - 2]) <= 0)
123       m--;
124     ch[m++] = p[i];
125   }
126   if (n > 1) m--;
127   ch.resize(m);
128   return ch;
129 }
130 int isPointInPolygon(Point p, Polygon poly) {
131   int wn = 0;
132   int n = poly.size();
133   for (int i = 0; i < n; i++) {
134     if (!OnSegment(p, poly[i], poly[(i + 1) % n])) return -1;
135     int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p -
136       poly[i]));
137     int d1 = dcmp(poly[i].y - p.y);
138     int d2 = dcmp(poly[(i + 1) % n].y - p.y);
139     if (k > 0 && d1 <= 0 && d2 > 0) wn++;
140     if (k < 0 && d2 <= 0 && d1 > 0) wn--;
141   }
142   if (wn != 0) return 1;
143   return 0;
144 }
145 int diameter2(vector<Point> &points) {
146   vector<Point> p = ConvexHull(points);
147   int n = p.size();
148   if (n == 1) return 0;
149   if (n == 2) return Dist2(p[0], p[1]);
150   p.push_back(p[0]);

```

```

142 int ans = 0;
143 for (int u = 0, v = 1; u < n; u++) {
144     for (;) {
145         int diff = Cross(p[u + 1] - p[u], p[v + 1] - p[v]);
146         if (diff <= 0) {
147             ans = max(ans, Dist2(p[u], p[v]));
148             if (diff == 0) ans = max(ans, Dist2(p[u], p[v + 1]));
149             break;
150         }
151         v = (v + 1) % n;
152     }
153 }
154 return ans;
155 }
156 double RC_Distance(Point *ch1, Point *ch2, int n, int m) {
157     int q = 0, p = 0;
158     REP(i, n) if (ch1[i].y - ch1[p].y < -eps) p = i;
159     REP(i, m) if (ch2[i].y - ch2[q].y > eps) q = i;
160     ch1[n] = ch1[0];
161     ch2[m] = ch2[0];
162     double tmp, ans = 1e100;
163     REP(i, n) {
164         while ((tmp = Cross(ch1[p + 1] - ch1[p], ch2[q + 1] -
165             ch1[p]) - Cross(ch1[p + 1] - ch1[p], ch2[q] -
166             ch1[p])) > eps)
167             q = (q + 1) % m;
168         if (tmp < -eps)
169             ans = min(ans, DistanceToSegment(ch2[q], ch1[p],
170             ch1[p + 1]));
171         else
172             ans = min(ans, dis_pair_seg(ch1[p], ch1[p + 1], ch2[q],
173             ch2[q + 1]));
174         p = (p + 1) % n;
175     }
176     return ans;
177 }
178 double RC_Triangle(Point *res, int n) {
179     if (n < 3) return 0;
180     double ans = 0, tmp;
181     res[n] = res[0];
182     int j, k;
183     REP(i, n) {
184         j = (i + 1) % n;
185         k = (j + 1) % n;
186         while ((j != k) && (k != i)) {
187             while (Cross(res[j] - res[i], res[k + 1] - res[i]) >
188                 Cross(res[j] - res[i], res[k] - res[i]))
189                 k = (k + 1) % n;
190             tmp = Cross(res[j] - res[i], res[k] - res[i]);
191             if (tmp > ans) ans = tmp;
192             j = (j + 1) % n;
193         }
194     }
195     return ans;
196 }
197 double fermtat_point(Point *pt, int n, Point &ptres) {
198     Point u, v;
199     double step = 0.0, curlen, explen, minlen;
200     int i, j, k, idx;
201     bool flag;
202     u.x = u.y = v.x = v.y = 0.0;
203     REP(i, n) {
204         step += fabs(pt[i].x) + fabs(pt[i].y);
205         u.x += pt[i].x;
206         u.y += pt[i].y;
207     }
208     u.x /= n;
209     u.y /= n;
210     flag = 0;
211     while (step > eps) {
212         for (k = 0; k < 10; step /= 2, ++k)
213             for (i = -1; i <= 1; ++i)
214                 for (j = -1; j <= 1; ++j) {
215                     v.x = u.x + step * i;
216                     v.y = u.y + step * j;
217                     curlen = explen = 0.0;
218                     REP(idx, n) {
219                         curlen += dist(u, pt[idx]);
220                         explen += dist(v, pt[idx]);
221                     }
222                     if (curlen > explen) {
223                         u = v;
224                         minlen = explen;
225                         flag = 1;
226                     }
227                 }
228     }
229     ptres = u;
230     return flag ? minlen : curlen;
231 }
232 bool cmpy(const int &a, const int &b) {
233     return point[a].y < point[b].y;
234 }
235 double Closest_Pair(int left, int right) {
236     double d = INF;
237     if (left == right) return d;
238     if (left + 1 == right) return dis(left, right);
239     int mid = (left + right) >> 1;
240     double d1 = Closest_Pair(left, mid);
241     double d2 = Closest_Pair(mid + 1, right);
242     d = min(d1, d2);
243     int i, j, k = 0;
244     for (i = left; i <= right; i++) {
245         if (fabs(point[mid].x - point[i].x) <= d) tmp[k++] = i;
246     }
247     sort(tmp, tmp + k, cmpy);
248     for (i = 0; i < k; i++) {
249         for (j = i + 1; j < k && point[tmp[j]].y -
250             point[tmp[i]].y < d; j++) {
251             double d3 = dis(tmp[i], tmp[j]);
252             if (d > d3) d = d3;
253         }
254     }
255     return d;
256 }
257 int getLineCircleIntersection(Line L, Circle C, double &t1,
258     double &t2, vector<Point> &sol) {
259     double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y
260         - C.c.y;
261     double e = a * a + c * c, f = 2 * (a * b + c * d),
262         g = b * b + d * d - C.r * C.r;
263     double delta = f * f - 4 * e * g;
264     if (dcmp(delta) < 0) return 0;
265     if (dcmp(delta) == 0) {
266         t1 = t2 = -f / (2 * e);
267         sol.push_back(L.point(t1));
268         return 1;
269     }
270     t1 = (-f - sqrt(delta)) / (2 * e);
271     sol.push_back(L.point(t1));
272     t2 = (-f + sqrt(delta)) / (2 * e);
273     sol.push_back(L.point(t2));
274     return 2;
275 }
276 int getCircleCircleIntersection(Circle C1, Circle C2,
277     vector<Point> &sol) {
278     double d = Length(C1.c - C2.c);
279     if (dcmp(d) == 0) {
280         if (dcmp(C1.r - C2.r) == 0) return -1;
281         return 0;
282     }
283     if (dcmp(C1.r + C2.r - d) < 0) return 0;
284     if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
285     double a = angle(C2.c - C1.c);
286     double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2
287         * C1.r * d));
288     Point p1 = C1.point(a - da), p2 = C1.point(a + da);
289     sol.push_back(p1);
290     if (p1 == p2) return 1;
291     sol.push_back(p2);
292     return 2;
293 }
294 int getTangents(Point p, Circle C, Vector *v) {
295     Vector u = C.c - p;
296     double dist = Length(u);
297     if (dist < C.r)
298         return 0;
299     else if (dcmp(dist - C.r) == 0) {
300         v[0] = Rotate(u, PI / 2);
301     }

```

```

294     return 1;
295 } else {
296     double ang = asin(C.r / dist);
297     v[0] = Rotate(u, -ang);
298     v[1] = Rotate(u, +ang);
299     return 2;
300 }
301 }
302 int getTangents(Circle A, Circle B, Point *a, Point *b) {
303     int cnt = 0;
304     if (A.r < B.r) swap(A, B), swap(a, b);
305     int d2 =
306         (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) *
307         (A.c.y - B.c.y);
308     int rdif = A.r - B.r;
309     int rsum = A.r + B.r;
310     if (d2 < rdif * rdif) return 0;
311     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
312     if (d2 == 0 && A.r == B.r) return -1;
313     if (d2 == rdif * rdif) {
314         a[cnt] = A.point(base);
315         b[cnt] = B.point(base);
316         cnt++;
317         return 1;
318     }
319     double ang = acos((A.r - B.r) / sqrt(d2));
320     a[cnt] = A.point(base + ang);
321     b[cnt] = B.point(base + ang);
322     cnt++;
323     a[cnt] = A.point(base - ang);
324     b[cnt] = B.point(base - ang);
325     cnt++;
326     if (d2 == rsum * rsum) {
327         a[cnt] = A.point(base);
328         b[cnt] = B.point(Pi + base);
329         cnt++;
330     } else if (d2 > rsum * rsum) {
331         double ang = acos((A.r + B.r) / sqrt(d2));
332         a[cnt] = A.point(base + ang);
333         b[cnt] = B.point(Pi + base + ang);
334         cnt++;
335         a[cnt] = A.point(base - ang);
336         b[cnt] = B.point(Pi + base - ang);
337         cnt++;
338     }
339     return cnt;
340 }
341 Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
342     double Bx = p2.x - p1.x, By = p2.y - p1.y;
343     double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
344     double D = 2 * (Bx * Cy - By * Cx);
345     double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
346     double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
347     Point p = Point(cx, cy);
348     return Circle(p, Length(p1 - p));
349 }
350 Circle InscribedCircle(Point p1, Point p2, Point p3) {
351     double a = Length(p2 - p3);
352     double b = Length(p3 - p1);
353     double c = Length(p1 - p2);
354     Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
355     return Circle(p, DistanceToLine(p, p1, p2));
356 }
357 vector<Point>
358     CircleThroughPointTangentToLineGivenRadius(Point p,
359     Line L, double r) {
360     vector<Point> ans;
361     double t1, t2;
362     getLineCircleIntersection(L.move(-r), Circle(p, r), t1,
363     t2, ans);
364     getLineCircleIntersection(L.move(r), Circle(p, r), t1,
365     t2, ans);
366     return ans;
367 }
368 vector<Point> CircleTangentToLinesGivenRadius(Line a, Line
369     b, double r) {
370     vector<Point> ans;
371     Line L1 = a.move(-r), L2 = a.move(r);
372     Line L3 = b.move(-r), L4 = b.move(r);
373     ans.push_back(GetLineIntersection(L1, L3));
374     ans.push_back(GetLineIntersection(L1, L4));
375     ans.push_back(GetLineIntersection(L2, L3));
376     ans.push_back(GetLineIntersection(L2, L4));
377     return ans;
378 }
379 vector<Point>
380     CircleTangentToTwoDisjointCirclesWithRadius(Circle
381     c1, Circle c2, double r) {
382     vector<Point> ans;
383     Vector v = c2.c - c1.c;
384     double dist = Length(v);
385     int d = dcmp(dist - c1.r - c2.r - r * 2);
386     if (d > 0) return ans;
387     getCircleCircleIntersection(Circle(c1.c, c1.r + r),
388     Circle(c2.c, c2.r + r), ans);
389     return ans;
390 }
391 int getSegCircleIntersection(Line L, Circle C, Point *sol) {
392     Vector nor = normal(L.v);
393     Line pl = Line(C.c, nor);
394     Point ip = GetIntersection(pl, L);
395     double dis = Length(ip - C.c);
396     if (dcmp(dis - C.r) > 0) return 0;
397     Point dxy = vecunit(L.v) * sqrt(sqr(C.r) - sqr(dis));
398     int ret = 0;
399     sol[ret] = ip + dxy;
400     if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
401     sol[ret] = ip - dxy;
402     if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
403     return ret;
404 }
405 double SegCircleArea(Circle C, Point a, Point b) {
406     double a1 = angle(a - C.c);
407     double a2 = angle(b - C.c);
408     double da = fabs(a1 - a2);
409     if (da > Pi) da = Pi * 2.0 - da;
410     return dcmp(Cross(b - C.c, a - C.c)) * da * sqr(C.r) /
411     2.0;
412 }
413 double PolyCiclrArea(Circle C, Point *p, int n) {
414     double ret = 0.0;
415     Point sol[2];
416     p[n] = p[0];
417     REP(i, n) {
418         double t1, t2;
419         int cnt = getSegCircleIntersection(Line(p[i], p[i + 1]
420         - p[i]), C, sol);
421         if (cnt == 0) {
422             if (!On0rInCircle(p[i], C) || !On0rInCircle(p[i + 1],
423             C))
424                 ret += SegCircleArea(C, p[i], p[i + 1]);
425             else
426                 ret += Cross(p[i + 1] - C.c, p[i] - C.c) / 2.0;
427         }
428         if (cnt == 1) {
429             if (On0rInCircle(p[i], C) && !On0rInCircle(p[i + 1],
430             C))
431                 ret += Cross(sol[0] - C.c, p[i] - C.c) / 2.0,
432                 ret += SegCircleArea(C, sol[0], p[i + 1]);
433             else
434                 ret += SegCircleArea(C, p[i], sol[0]),
435                 ret += Cross(p[i + 1] - C.c, sol[0] - C.c) /
436                 2.0;
437         }
438         if (cnt == 2) {
439             if ((p[i] < p[i + 1]) ^ (sol[0] < sol[1]))
440                 swap(sol[0], sol[1]);
441             ret += SegCircleArea(C, p[i], sol[0]);
442             ret += Cross(sol[1] - C.c, sol[0] - C.c) / 2.0;
443             ret += SegCircleArea(C, sol[1], p[i + 1]);
444         }
445     }
446     return fabs(ret);
447 }
448 double area[N];
449 int n;
450 struct cp {
451     double x, y, r, angle;
452     int d;
453     cp() {}
454 }

```

```

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

```

```

591 if (b >= m) {
592     return (n - 1) * n / 2 * (b / m) + count(n, a, b % m,
593         m);
594 }
595 return count((a + b * n) / m, (a + b * n) % m, m, b);
596 }
597 bool TriSegIntersection(Point3 P0, Point3 P1, Point3 P2,
598     Point3 A, Point3 B, Point3 &P) {
599     Vector3 n = Cross(P1 - P0, P2 - P0);
600     if (dcmp(Dot(n, B - A)) == 0) return false;
601     double t = Dot(n, P0 - A) / Dot(n, B - A);
602     if (dcmp(t) < 0 || dcmp(t - 1) > 0) return false;
603     P = A + (B - A) * t;
604     return PointInTri(P, P0, P1, P2);

```

1.2 3D

```

1 struct Face {
2     int v[3];
3     Vector3 normal(Point3 *P) const {
4         return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);
5     }
6     int cansee(Point3 *p, int i) const {
7         return Dot(P[i] - P[v[0]], normal(P)) > 0 ? 1 : 0;
8     }
9 };
10 vector<Face> CH3D(Point3 *P, int n) {
11     vector<Face> cur;
12     cur.push_back((Face) {{0, 1, 2}});
13     cur.push_back((Face) {{2, 1, 0}});
14     for (int i = 3; i < n; ++i) {
15         vector<Face> next;
16         for (int j = 0; j < cur.size(); ++j) {
17             Face &f = cur[j];
18             int res = f.cansee(P, i);
19             if (!res) next.push_back(f);
20             for (int k = 0; k < 3; ++k) vis[f.v[k]][f.v[(k + 1) % 3]] = res;
21         }
22         for (int j = 0; j < cur.size(); ++j)
23             for (int k = 0; k < 3; ++k) {
24                 int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];
25                 if (vis[a][b] != vis[b][a] && vis[a][b])
26                     next.push_back((Face) {{
27                         a, b, i
28                     }});
29             }
30         cur = next;
31     }
32     return cur;
33 }
34 }

```

2 DataStructure

2.1 BIT

```

1 int Kth(int K) {
2     int ret = 0, i;
3     for(i = 21; i >= 0; i--) {
4         ret |= (1 << i);
5         if(ret >= 200000 || tree[ret] >= K) ret ^= (1 << i);
6         else K -= tree[ret];
7     }
8     return ret + 1;
9 }

```

2.2 Segment Tree

```

1 struct Tree {
2     int left, right, val, len[11];
3 };

```

```

4 Tree tree[8 * N];
5 void PushUp(int ind) {
6     int i;
7     for(i = 1; i <= K; i++) {
8         if(tree[ind].val >= i)
9             tree[ind].len[i] = Record[tree[ind].right + 1] -
10                 Record[tree[ind].left];
11         else if(tree[ind].left != tree[ind].right)
12             tree[ind].len[i] = tree[L(ind)].len[i -
13                 tree[ind].val] + tree[R(ind)].len[i -
14                 tree[ind].val];
15         else tree[ind].len[i] = 0;
16     }
17 }
18 void Update(int ind, int l, int r, int val) {
19     if(tree[ind].left == l && tree[ind].right == r) {
20         tree[ind].val += val;
21         PushUp(ind);
22         return;
23     }
24     int mid = (tree[ind].left + tree[ind].right) / 2;
25     if(r <= mid) Update(L(ind), l, r, val);
26     else if(l > mid) Update(R(ind), l, r, val);
27     else Update(L(ind), l, mid, val), Update(R(ind), mid + 1,
28         r, val);
29     PushUp(ind);
30 }

```

2.3 Scanning Line

```

1 const int UP = 0; const int DOWN = 1;
2 const int IN = 0; const int OUT = 1;
3 vector<int> g[N];
4 double TT;
5 struct Circle {
6     int x, y, r, w;
7     int getX(int flag) {
8         return flag == IN? x - r: x + r;
9     }
10    double getY(int flag) {
11        double ret = sqrt(1.0 * r * r - 1.0 * (TT - x) * (TT -
12            x));
13        return flag == UP? y + ret: y - ret;
14    }
15 } cir[N];
16 int sgn(double x) {
17     if(fabs(x) < 1e-10) return 0;
18     return x < 0? -1: 1;
19 }
20 struct Line {
21     int x, y, idx, flag;
22     Line() {}
23     Line(int x, int y, int idx, int flag): x(x), y(y),
24         idx(idx), flag(flag) {}
25     bool operator<(const Line& l) const {
26         if(x == l.x) return y > l.y;
27         return x < l.x;
28     }
29 } line[N << 1];
30 struct Node {
31     int idx, flag;
32     Node() {}
33     Node(int idx, int flag): idx(idx), flag(flag) {}
34     bool operator<(const Node& n) const {
35         double y1 = cir[idx].getY(flag);
36         double y2 = cir[n.idx].getY(n.flag);
37         int cmp = sgn(y1 - y2);
38         if(0 == cmp) return flag < n.flag;
39         return cmp > 0;
40     }
41 };
42 int n, totl;
43 int pre[N];
44 void solve() {
45     set<Node> re;
46     for(int i = 0; i < totl; i++) {
47         TT = line[i].x;
48         if(OUT == line[i].flag) {
49             re.erase(Node(line[i].idx, UP));

```

```

48     re.erase(Node(line[i].idx, DOWN));
49 } else {
50     set<Node>::iterator it = re.insert(Node(line[i].idx,
51                                         UP)).first;
52     set<Node>::iterator e = it, f = it;
53     e++;
54     int idx = it -> idx;
55     if(it == re.begin() || e == re.end()) {
56         cir[idx].w = 1;
57     } else {
58         f--;
59         if(f -> idx == e -> idx) {
60             g[f -> idx].push_back(idx);
61             cir[idx].w = cir[f -> idx].w + 1;
62             pre[idx] = f -> idx;
63         } else {
64             cir[idx].w = max(cir[f -> idx].w, cir[e -> idx].w);
65             if(cir[f -> idx].w >= cir[e -> idx].w && pre[f ->
66                 idx] {
67                 g[pre[f -> idx]].push_back(idx);
68                 pre[idx] = pre[f -> idx];
69             } else if(pre[e -> idx] {
70                 g[pre[e -> idx]].push_back(idx);
71                 pre[idx] = pre[e -> idx];
72             }
73         }
74     }
75     re.insert(Node(line[i].idx, DOWN));
76 }
77 }
78 totl = 0;
79 for(int i = 0; i < n; i++) {
80     scanf("%d%d%d", &cir[i].x, &cir[i].y, &cir[i].r);
81     cir[i].w = 0;
82     line[totl++] = Line(cir[i].getX(IN), cir[i].y, i, IN);
83     line[totl++] = Line(cir[i].getX(OUT), cir[i].y, i, OUT);
84 }
85 sort(line, line + totl);
86 solve();

```

2.4 2D SegmentTree

```

1  const int INF = (1 << 30);
2  struct IntervalTree2D {
3      int Max[maxn][maxn], Min[maxn][maxn], n, m;
4      int xo, xleaf, x1, y1, x2, y2, x, y, v, vmax, vmin;
5      void query1D(int o, int L, int R) {
6          if(y1 <= L && R <= y2) {
7              vmax = max(Max[xo][o], vmax);
8              vmin = min(Min[xo][o], vmin);
9          } else {
10             int M = L + (R - L) / 2;
11             if(y1 <= M) query1D(o * 2, L, M);
12             if(M < y2) query1D(o * 2 + 1, M + 1, R);
13         }
14     }
15     void query2D(int o, int L, int R) {
16         if(x1 <= L && R <= x2) {
17             xo = o;
18             query1D(1, 1, m);
19         } else {
20             int M = L + (R - L) / 2;
21             if(x1 <= M) query2D(o * 2, L, M);
22             if(M < x2) query2D(o * 2 + 1, M + 1, R);
23         }
24     }
25     void modify1D(int o, int L, int R) {
26         if(L == R) {
27             if(xleaf) {
28                 Max[xo][o] = Min[xo][o] = v;
29                 return;
30             }
31             Max[xo][o] = max(Max[xo * 2][o], Max[xo * 2 + 1][o]);
32             Min[xo][o] = min(Min[xo * 2][o], Min[xo * 2 + 1][o]);
33         } else {
34             int M = L + (R - L) / 2;
35             if(y <= M) modify1D(o * 2, L, M);
36             else modify1D(o * 2 + 1, M + 1, R);

```

```

37         Max[xo][o] = max(Max[xo][o * 2], Max[xo][o * 2 + 1]);
38         Min[xo][o] = min(Min[xo][o * 2], Min[xo][o * 2 + 1]);
39     }
40 }
41 void modify2D(int o, int L, int R) {
42     if(L == R) {
43         xo = o;
44         xleaf = 1;
45         modify1D(1, 1, m);
46     } else {
47         int M = L + (R - L) / 2;
48         if(x <= M) {
49             modify2D(o * 2, L, M);
50         } else {
51             modify2D(o * 2 + 1, M + 1, R);
52         }
53         xo = o;
54         xleaf = 0;
55         modify1D(1, 1, m);
56     }
57 }
58 void query() {
59     vmax = -INF;
60     vmin = INF;
61     query2D(1, 1, n);
62 }
63 void modify() {
64     modify2D(1, 1, n);
65 }
66 void init(int n, int m) {
67     this -> n = n;
68     this -> m = m;
69 }
70 } t;
71 t.x1 = max(1, x - 1 / 2); t.y1 = max(1, y - 1 / 2);
72 t.x2 = min(n, x + 1 / 2); t.y2 = min(n, y + 1 / 2);
73 t.query();
74 t.x = x; t.y = y; t.v = val;
75 t.modify();

```

2.5 SBT

```

1  Node *create(long long v) {
2      Node *p = &memo[top++];
3      p -> s = 1; p -> sum = p -> key = v;
4      p -> ch[0] = p -> ch[1] = nul; return p;
5  }
6  struct SBT {
7      #define L(x) ((x)->ch[0])
8      #define R(x) ((x)->ch[1])
9      Node *root;
10     void rot(Node *&t, int f) {
11         Node *k = t -> ch[f ^ 1];
12         t -> ch[f ^ 1] = k -> ch[f]; k -> ch[f] = t;
13         k -> s = t -> s; k -> sum = t -> sum;
14         t -> s = L(t) -> s + R(t) -> s + 1;
15         t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
16         t = k;
17     }
18     void maintain(Node *&t, int f) {
19         if(t -> ch[f] -> ch[f] -> s > t -> ch[f ^ 1] -> s) {
20             rot(t, f ^ 1);
21         } else if(t -> ch[f] -> ch[f ^ 1] -> s > t -> ch[f ^ 1]
22             -> s) {
23             rot(t -> ch[f], f);
24             rot(t, f ^ 1);
25         } else return;
26         for(int i = 0; i < 2; i++) maintain(t -> ch[i], i);
27         for(int i = 0; i < 2; i++) maintain(t, i);
28     }
29     void ins(Node *&t, long long v) {
30         if(t == nul) t = create(v);
31         else {
32             t -> s++;
33             t -> sum += v;
34             ins(t -> ch[v >= t -> key], v);
35             maintain(t, v >= t -> key);
36         }

```



```

37 Node *del(Node *&t, long long v) {
38     if(t == nul) return nul;
39     t -> s--;
40     Node *p = nul;
41     if(v == t -> key || t -> ch[v > t -> key] == nul) {
42         if(L(t) != nul && R(t) != nul) {
43             p = del(L(t), v + 1), t -> key = p -> key;
44         } else {
45             p = t; t = t -> ch[L(t) == nul];
46         }
47         t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
48         return p;
49     }
50     p = del(t -> ch[v > t -> key], v);
51     t -> sum = L(t) -> sum + R(t) -> sum + t -> key;
52     return p;
53 }
54 } sbt;
55 int KthL(int t, int k) { //largest
56     if(t == 0 || k <= 0 || k > sbt[t].s) return 0;;
57     int s = (sbt[t].ch[1] == 0? 0: sbt[sbt[t].ch[1]].s);
58     if(k == s + 1) {
59         return sbt[t].key;
60     } else if(k <= s) {
61         return KthL(sbt[t].ch[1], k);
62     }
63     return KthL(sbt[t].ch[0], k - s - 1);
64 }
65 int KthS(int t, int k) { //smallest
66     if(t == 0 || k <= 0 || k > sbt[t].s) return 0;
67     if(k <= sbt[sbt[t].ch[0]].s) {
68         return KthS(sbt[t].ch[0], k);
69     } else if(k > sbt[sbt[t].ch[0]].s + 1) {
70         return KthS(sbt[t].ch[1], k - sbt[sbt[t].ch[0]].s - 1);
71     }
72     return sbt[t].key;
73 }
74 nul = &memo[top++];
75 nul -> ch[0] = nul -> ch[1] = nul;
76 nul -> s = 0;

```

2.6 Splay

```

1 struct Node {
2     Node *ch[2], *pre;
3     int s, v, lmi, rmi, lma, rma, sum;
4     int rep, sw, inv;
5 };
6 char str[N];
7 struct Splay {
8     int top;
9     Node *nul, *root, memo[N];
10    Node *create(int v) {
11        Node* p = &memo[top++];
12        p -> pre = p -> ch[0] = p -> ch[1] = nul;
13        p -> s = 1; p -> v = v; p -> sum = v;
14        if(v > 0) {
15            p -> lmi = p -> rmi = 0; p -> lma = p -> rma = v;
16        } else {
17            p -> lmi = p -> rmi = v; p -> lma = p -> rma = 0;
18        }
19        p -> rep = p -> sw = p -> inv = 0;
20        return p; /**/
21    }
22    void init() {
23        top = 0;
24        nul = &memo[top++];
25        nul -> ch[0] = nul -> ch[1] = nul -> pre = nul;
26        nul -> s = 0; nul -> v = nul -> sum = 0;
27        nul -> lmi = nul -> rmi = oo; nul -> lma = nul -> rma = -oo;
28        nul -> rep = nul -> sw = nul -> inv = 0;
29    }
30    void pushup(Node* p) {
31    }
32    void repupd(Node* p, int v) {
33        if(p == nul) {
34            return;
35        }
36        p -> rep = v;

```

```

37     p -> sw = p -> inv = 0;
38     if(p -> rep > 0) {
39         p -> lmi = p -> rmi = 0;
40         p -> lma = p -> rma = p -> s * p -> rep;
41     } else {
42         p -> lma = p -> rma = 0;
43         p -> lmi = p -> rmi = p -> s * p -> rep;
44     }
45     p -> sum = p -> s * p -> rep;
46     p -> v = p -> rep;
47 }
48 void swupd(Node* p) {
49     if(p == nul) return;
50     p -> sw ^= 1;
51     swap(L(p), R(p));
52     swap(p -> lmi, p -> rmi);
53     swap(p -> lma, p -> rma);
54 }
55 void invupd(Node* p) {
56     if(p == nul) return;
57     p -> inv ^= 1;
58     swap(p -> lmi, p -> lma);
59     p -> lmi = -p -> lmi; p -> lma = -p -> lma;
60     swap(p -> rmi, p -> rma);
61     p -> rmi = -p -> rmi; p -> rma = -p -> rma;
62     p -> sum = -p -> sum; p -> v = -p -> v;
63 }
64 void pushdown(Node* p) {
65     if(p -> rep) {
66         repupd(L(p), p -> rep); repupd(R(p), p -> rep);
67         p -> rep = 0;
68     }
69     if(p -> sw) {
70         swupd(L(p)); swupd(R(p)); p -> sw = 0;
71     }
72     if(p -> inv) {
73         invupd(L(p)); invupd(R(p)); p -> inv = 0;
74     }
75 }
76 Node* build(int l, int r, Node* fa) {
77     if(l > r) return nul;
78     int mid = (l + r) / 2;
79     Node* p;
80     if(0 == str[mid]) {
81         p = create(0);
82     } else if('(' == str[mid]) {
83         p = create(1);
84     } else {
85         p = create(-1);
86     }
87     p -> ch[0] = build(l, mid - 1, p);
88     p -> ch[1] = build(mid + 1, r, p);
89     p -> pre = fa;
90     pushup(p);
91     return p; /**/
92 }
93 void rotate(Node* x, int t) { //K>=1
94     Node* y = x -> pre;
95     pushdown(y); pushdown(x);
96     y -> ch[t ^ 1] = x -> ch[t]; x -> pre = y -> pre;
97     if(x -> ch[t] != nul) x -> ch[t] -> pre = y;
98     if(y -> pre != nul) y -> pre -> ch[R(y -> pre) == y] =
99         x;
100    x -> ch[t] = y; y -> pre = x;
101    pushup(y); /**/
102 }
103 void splay(Node* x, Node* f) {
104     pushdown(x);
105     while(x -> pre != f) {
106         Node* y = x -> pre; Node* z = y -> pre;
107         pushdown(z); pushdown(y); pushdown(x);
108         if(z == f) {
109             rotate(x, L(y) == x);
110         } else {
111             int t = (L(z) == y);
112             if(y -> ch[t] == x) {
113                 rotate(x, t ^ 1); rotate(x, t);
114             } else {
115                 rotate(y, t); rotate(x, t);
116             }
117         }
118     }
119 }

```

```

117     }
118     pushup(x);
119     if(f == nul) {
120         root = x;
121     }
122 }
123 void selectk(int k, Node* f) {
124     Node* p = root;
125     while(1) {
126         pushdown(p);
127         int tmp = L(p) -> s + 1;
128         if(tmp == k) {
129             break;
130         }
131         if(tmp > k) {
132             p = L(p);
133         } else {
134             p = R(p); k -= tmp;
135         }
136     }
137     splay(p, f);
138 }
139 void Rep(int l, int r, int c) {
140     selectk(l, nul); selectk(r + 2, root);
141     repupd(L(R(root)), c);
142     pushup(R(root)); pushup(root);
143 }
144 int Q(int l, int r) {
145     selectk(l, nul); selectk(r + 2, root);
146     int x = L(R(root)) -> lmi;
147     int y = L(R(root)) -> sum;
148     return (abs(y - x) + 1) / 2 + (abs(x) + 1) / 2;
149 }
150 Node *aa_bound(Node* p, int val) { //>=
151     if(p == nul) return nul;
152     if(p -> v < val) return aa_bound(R(p), val);
153     else {
154         Node* tmp = aa_bound(L(p), val);
155         if(tmp == nul || p -> v < val) return p;
156         return tmp;
157     }
158 }
159 Node *bb_bound(Node* p, int val) { //<
160     if(p == nul) return nul;
161     if(p -> v >= val) return bb_bound(L(p), val);
162     else {
163         Node *tmp = bb_bound(R(p), val);
164         if(tmp == nul || p -> v >= val) return p;
165         return tmp;
166     }
167 }
168 Node *cc_bound(Node* p, int val) { //<=
169     if(p == nul) return nul;
170     if(p -> v <= val) return cc_bound(R(p), val);
171     else {
172         Node* tmp = cc_bound(L(p), val);
173         if(tmp == nul || tmp -> v <= val) return p;
174         return tmp;
175     }
176 }
177 void ins(Node* q) {
178     Node *bb = bb_bound(root, q -> v);
179     Node *aa = aa_bound(root, q -> v);
180     splay(bb, nul); splay(aa, bb);
181     L(R(root)) = q; q->pre = R(root);
182     pushup(R(root)); pushup(root);
183 }
184 void del(Node *p) {
185     splay(p, nul);
186     if(R(p) != nul) {
187         root = R(p);
188         selectK(1, nul);
189         L(root) = L(p);
190         if(L(root) != nul) {
191             L(root) -> pre = root;
192         }
193     } else root = L(p);
194     if(root != nul) {
195         root->pre = nul; /*!*/
196         pushup(root);
197     }

```

```

198     }
199 };
200 Splay tree;
201 tree.init();
202 tree.root = tree.build(0, n + 1, tree.nul);

```

2.7 Persistent Binary Tree

```

1 int ls[N], rs[N], c[N], tree[N], idx, mx;
2 int build(int l, int r) {
3     int rt = idx++;
4     c[rt] = 0;
5     if(l != r) {
6         int mid = (l + r) >> 1;
7         ls[rt] = build(l, mid);
8         rs[rt] = build(mid + 1, r);
9     }
10    return rt;
11 }
12 int update(int rt, int pos, int val) {
13     int nrt = idx++, ret = nrt;
14     c[nrt] = c[rt] + val;
15     int l = 1, r = mx;
16     while(l != r) {
17         int mid = (l + r) >> 1;
18         if(pos <= mid) {
19             ls[nrt] = idx++; rs[nrt] = rs[rt];
20             nrt = ls[nrt]; rt = ls[rt];
21             r = mid;
22         } else {
23             rs[nrt] = idx++; ls[nrt] = ls[rt];
24             nrt = rs[nrt]; rt = rs[rt];
25             l = mid + 1;
26         }
27         c[nrt] = c[rt] + val;
28     }
29     return ret;
30 }
31 int query(int lrt, int rrt, int k) {
32     int l = 1, r = n;
33     while(l != r) {
34         int mid = (l + r) >> 1;
35         if(c[ls[lrt]] - c[ls[lrt]] < k) {
36             k -= (c[ls[lrt]] - c[ls[lrt]]);
37             lrt = rs[lrt]; rrt = rs[rrt];
38             l = mid + 1;
39         } else {
40             lrt = ls[lrt]; rrt = ls[rrt];
41             r = mid;
42         }
43     }
44     return l;
45 }
46 sort(b + 1, b + n + 1);
47 n = unique(b + 1, b + n + 1) - b - 1;
48 tree[0] = build(1, n);
49 tree[i] = update(tree[i - 1], f(a[i]), 1);
50 query(tree[l - 1], tree[r], k)
51 b[query(tree[l - 1], tree[r], k)]
52
53 struct Node {/**lazy**/
54     int ls, rs;
55     long long lazy, sum;
56     Node() {}
57     Node(int ls, int rs, int lazy, int sum): ls(ls), rs(rs),
58         lazy(lazy), sum(sum) {}
59 };
60 int a[100100];
61 Node node[N];
62 int tree[100100], idx, mx;
63 void pushup(int rt) {
64     node[rt].sum = node[node[rt].ls].sum +
65         node[node[rt].rs].sum;
66 }
67 int build(int l, int r) {
68     int rt = idx++;
69     node[rt].sum = node[rt].lazy = 0;
70     if(l != r) {
71         int mid = (l + r) >> 1;

```



```

70     node[rt].ls = build(1, mid);
71     node[rt].rs = build(mid + 1, r);
72     pushup(rt);
73 } else node[rt].sum = a[1];
74 return rt;
75 }
76 int update(int rt, int ll, int rr, int l, int r, long long
    val) {
77     int nrt = idx++;
78     node[nrt] = node[rt];
79     node[nrt].sum += val * (r - l + 1);
80     if(ll == l && rr == r) {
81         node[nrt].lazy += val;
82         return nrt;
83     }
84     int mid = (ll + rr) >> 1;
85     if(r <= mid) {
86         node[nrt].ls = update(node[rt].ls, ll, mid, l, r, val);
87     } else if(l > mid) {
88         node[nrt].rs = update(node[rt].rs, mid + 1, rr, l, r,
            val);
89     } else {
90         node[nrt].ls = update(node[rt].ls, ll, mid, l, mid,
            val);
91         node[nrt].rs = update(node[rt].rs, mid + 1, rr, mid +
            1, r, val);
92     }
93     return nrt;
94 }
95 long long query(int rt, int ll, int rr, int l, int r) {
96     if(ll == l && rr == r) return node[rt].sum;
97     long long ret = 0;
98     ret += (r - l + 1) * node[rt].lazy;
99     int mid = (ll + rr) >> 1;
100    if(r <= mid) {
101        ret += query(node[rt].ls, ll, mid, l, r);
102    } else if(l > mid) {
103        ret += query(node[rt].rs, mid + 1, rr, l, r);
104    } else {
105        ret += query(node[rt].ls, ll, mid, l, mid);
106        ret += query(node[rt].rs, mid + 1, rr, mid + 1, r);
107    }
108    return ret;
109 }
110 void init(int n) {
111     mx = n;
112     idx = 0;
113 }

```

2.8 LCT

```

1 struct Node {
2     Node *ch[2], *pre;
3     int v, sz, mx[2], tot[2];
4     bool isroot;
5     int rep, add, sw;
6 };
7 struct LCT {
8     int top;
9     Node* nul, memo[N];
10    Node* create(int v) {
11        Node* p = &memo[top++];
12        p->pre = p->ch[0] = p->ch[1] = nul;
13        p->v = v; p->sz = 1;
14        p->rep = oo; p->add = p->sw = 0;
15        p->isroot = 1;
16        return p; /*!*/
17    }
18    void init() {
19        top = 0;
20        nul = &memo[top++];
21        nul->ch[0] = nul->ch[1] = nul->pre = nul;
22        nul->v = oo; nul->sz = 0;
23        nul->rep = oo; nul->add = nul->sw = 0;
24        nul->isroot = 0;
25    }
26    void pushup(Node* p) {
27        p->sz = L(p) ->sz + R(p) ->sz + 1;
28    }

```

```

29    void repupd(Node* p, int v) {
30        if(p == nul) return;
31        p->v = v; p->rep = v; p->add = 0;
32    }
33    void addupd(Node* p, int add) {
34        if(p == nul) return;
35        p->v += add; p->add += add;
36    }
37    void swupd(Node* p) {
38        if(p == nul) return;
39        swap(L(p), R(p)); p->sw ^= 1;
40    }
41    void pushdown(Node* p) {
42        if(p->rep != oo) {
43            repupd(L(p), p->rep);
44            repupd(R(p), p->rep);
45            p->rep = oo;
46        }
47        if(p->sw) {
48            swupd(L(p)); swupd(R(p));
49            p->sw = 0;
50        }
51        if(p->add) {
52            addupd(L(p), p->add);
53            addupd(R(p), p->add);
54            p->add = 0;
55        }
56    }
57    void down(Node* p) {
58        if(!p->isroot) down(p->pre);
59        pushdown(p);
60    }
61    void rot(Node* x) {
62        Node *y = x->pre, *z = y->pre;
63        int t = (x == y->ch[0]);
64        y->ch[t] = x->ch[t];
65        x->ch[t] ->pre = y; x->ch[t] = y;
66        y->pre = x; x->pre = z;
67        if(y->isroot) {
68            y->isroot = 0; x->isroot = 1;
69        } else {
70            z->ch[y == z->ch[1]] = x;
71        }
72        pushup(y);
73    }
74    void splay(Node* x) {
75        down(x);
76        while(!x->isroot) {
77            Node* y = x->pre, *z = y->pre;
78            if(!y->isroot) {
79                Node* tmp = ((x == y->ch[0]) == (y == z->
                    ch[0]))? y: x;
80                rot(tmp); /*!*/
81            }
82            rot(x);
83        }
84        pushup(x);
85    }
86    void access(Node* p) {
87        for(Node* q = nul; p != nul; p = p->pre) {
88            splay(p);
89            p->ch[1] ->isroot = 1; q->isroot = 0;
90            p->ch[1] = q; q = p;
91            pushup(p);
92        }
93    }
94    void change_root(Node* p) {
95        access(p); splay(p); swupd(p);
96    }
97    void cut(Node* a, Node* b) {
98        change_root(a); access(a); splay(b);
99        b->pre = nul;
100    }
101    void link(Node* a, Node* b) {
102        change_root(b);
103        b->pre = a;
104    }
105    void gao1(Node* a, Node* b, Node* c, Node* d) {
106        cut(a, b); link(c, d);
107    }
108    void gao2(Node* a, Node* b, int c) {

```

```

109     change_root(a); access(b); splay(b);
110     repupd(b, c);
111 }
112 bool judge(Node* p, Node* q) {
113     while(p -> pre != nul)p = p -> pre;
114     while(q -> pre != nul)q = q -> pre;
115     return p == q;
116 }
117 };
118 for(int i = 1; i <= n; i++) {
119     int x;
120     scanf("%d", &x);
121     pos[i] = tree.create(x);
122 }
123 for(int i = 1; i < n; i++) {
124     int x, y;
125     scanf("%d%d", &x, &y);
126     tree.link(pos[x], pos[y]);
127 }
128 lct.change_root(re[x]); lct.access(re[x]);
129 re[x]->val=y; lct.pushup(re[x]); //w[x]=y;
130
131 void access(Node *p) {
132     for(Node *q=nul; p!=nul; p=p->pre) {
133         splay(p);
134         (p->s[0] += R(p) ->s[1]) %= mod;
135         (p->s[0] += mod - q ->s[1]) %= mod;
136         (p->s[2] += R(p) ->s[1] * R(p) ->s[1] % mod) %= mod;
137         (p->s[2] += mod - q ->s[1] * q ->s[1] % mod) %= mod;
138         R(p) ->isroot = 1;
139         q ->isroot = 0;
140         p ->ch[1] = q;
141         q = p;
142         pushup(p);
143     }
144 }
145 void link(Node *a, Node *b) {
146     change_root(b);
147     (a ->s[0] += b ->s[1]) %= mod;
148     (a ->s[1] += b ->s[1]) %= mod;
149     (a ->s[2] += b ->s[1] * b ->s[1] % mod) %= mod;
150     (a ->s[3] += b ->s[1] * b ->s[1] % mod) %= mod;
151     b ->pre = a;
152 }
153 for(int i=1; i<=n; i++)
154     for(int j=0; j<g[i].size(); j++)
155         lct.link(re[i], re[g[i][j]]);

```

2.9 Union Set

```

1  int f(int x) {
2      int o = x;
3      while ( p[o] != o ) o = p[o];
4      int ans = o;
5      while ( p[x] != x ) {
6          int tmp = p[x]; p[x] = tmp; x = tmp;
7      }
8      return ans;
9  }
10 void Union(int uu, int vv, bool in) {
11     int u = f(uu), v = f(vv);
12     if(u == v) return;
13     if(in) {
14         st[++top] = Re(u, v, cnt[u], cnt[v], rk[u], rk[v]);
15         ret += 1LL * cnt[u] * cnt[v];
16     }
17     if(rk[u] <= rk[v]) {
18         rk[v] = max(rk[v], rk[u] + 1);
19         p[u] = v;
20         cnt[v] += cnt[u];
21     } else {
22         rk[u] = max(rk[u], rk[v] + 1);
23         p[v] = u;
24         cnt[u] += cnt[v];
25     }
26 }

```

2.10 Cdq

```

1  struct Re {
2      int x, y, z, val, idx;
3      Re() {}
4      Re(int x, int y, int z, int val, int idx) : x(x), y(y),
5          z(z), val(val), idx(idx) {}
6  } re[N];
7  bool cmpx(const Re& a, const Re& b) {
8      if(a.x ^ b.x) return a.x < b.x;
9      return a.idx < b.idx;
10 }
11 bool cmpy(const Re& a, const Re& b) {
12     if(a.y ^ b.y) return a.y < b.y;
13     return a.idx < b.idx;
14 }
15 void upd(int l, int r) {
16     if(l == r) return;
17     int mid = (l + r) >> 1;
18     upd(l, mid);
19     upd(mid + 1, r);
20     aa = 0, bb = 0;
21     for(int i = 1; i <= mid; i++) {
22         mema[aa++] = pool[i];
23     }
24     for(int i = mid + 1; i <= r; i++) {
25         memb[bb++] = pool[i];
26     }
27     sort(mema, mema + aa, cmpy);
28     sort(memb, memb + bb, cmpy);
29     for(int i = 0, j = 0; j < bb; j++) {
30         for(; i < aa && mema[i].y <= memb[j].y; i++) {
31             if(!mema[i].val) update(mema[i].z, 1);
32         }
33         if(memb[j].val) ans[memb[j].idx] += memb[j].val *
34             query(memb[j].z);
35     }
36     for(int i = 0; i < aa; i++) clc(mema[i].z);
37 }
38 void solve(int l, int r) {
39     if(l == r) return;
40     int mid = (l + r) >> 1;
41     solve(l, mid);
42     solve(mid + 1, r);
43     tt = 0;
44     for(int i = 1; i <= mid; i++) {
45         if(!re[i].val) pool[tt++] = re[i];
46     }
47     for(int i = mid + 1; i <= r; i++) {
48         if(re[i].val) pool[tt++] = re[i];
49     }
50     sort(pool, pool + tt, cmpx);
51     if(tt) upd(0, tt - 1);
52 }

```

2.11 Binary Search

```

1  void solve(int head, int tail, int l, int r) {
2      if (head > tail) return;
3      if (l == r) {
4          for (int i = head; i <= tail; ++i) {
5              if (q[i].qt == 3) ans[q[i].index] = 1;
6          }
7          return;
8      }
9      int mid = (l + r) >> 1;
10     for (int i = head; i <= tail; ++i) {
11         if (q[i].qt == 1 && q[i].y <= mid) {
12             add(q[i].x, 1);
13         } else if (q[i].qt == 2 && q[i].y <= mid) {
14             add(q[i].x, -1);
15         } else {
16             tmp[i] = query(q[i].y) - query(q[i].x - 1);
17         }
18     }
19     for (int i = head; i <= tail; ++i) {
20         if (q[i].qt == 1 && q[i].y <= mid) {
21             add(q[i].x, -1);
22         }
23     }
24 }

```

```

22     } else if (q[i].qt == 2 && q[i].y <= mid) {
23         add(q[i].x, 1);
24     }
25 }
26 int l1=0, l2 = 0;
27 for (int i = head; i <= tail; ++i) {
28     if (q[i].qt == 3) {
29         if (q[i].cur + tmp[i] >= q[i].k) {
30             q1[++l1] = q[i];
31         } else {
32             q[i].cur += tmp[i];
33             q2[++l2] = q[i];
34         }
35     } else {
36         if (q[i].y <= mid) q1[++l1] = q[i];
37         else q2[++l2] = q[i];
38     }
39 }
40
41 for (int i = 1; i <= l1; ++i) {
42     q[head + i - 1] = q1[i];
43 }
44 for (int i = 1; i <= l2; ++i) {
45     q[head + l1 + i - 1] = q2[i];
46 }
47 solve(head, head + l1 - 1, l, mid);
48 solve(head + l1, tail, mid + 1, r);
49 }
50 int main() {
51     while (scanf("%d", &n) != EOF) {
52         qtop = 0;
53         tot = 0;
54         for (int i = 1; i <= n; ++i) {
55             scanf("%d", &a + i);
56             b[tot++] = a[i];
57             q[++qtop].qt = 1;
58             q[qtop].x = i;
59             q[qtop].y = a[i];
60         }
61         int cmd;
62         int xi, yi, ki;
63         ansid = 0;
64         scanf("%d", &cmd);
65         for (int i = 0; i < m; ++i) {
66             scanf("%d", &cmd);
67             if (2==cmd) {
68                 scanf("%d%d", &xi, &yi, &ki);
69                 q[++qtop].x = xi; q[qtop].y = yi;
70                 q[qtop].k = ki; q[qtop].qt = 3;
71                 q[qtop].cur = 0;
72                 q[qtop].index = ++ansid;
73             } else {
74                 scanf("%d%d", &xi, &yi);
75                 q[++qtop].x = xi; q[qtop].y = a[xi];
76                 q[qtop].qt = 2;
77                 q[++qtop].x = xi; q[qtop].y = yi;
78                 q[qtop].qt = 1;
79                 a[xi] = yi;
80                 b[tot++] = yi;
81             }
82         }
83         sort(b, b + tot);
84         bsize = unique(b, b + tot) - b;
85         for (int i = 1; i <= qtop; ++i) {
86             if (q[i].qt == 1 || q[i].qt == 2) {
87                 q[i].y = lower_bound(b, b + bsize, q[i].y) - b + 1;
88             }
89         }
90         solve(1, qtop, 1, bsize);
91         for (int i = 1; i <= ansid; ++i) {
92             printf("%d\n", b[ans[i]-1]);
93         }
94     }
95     return 0;
96 }

```

2.12 Dominator Tree

```
1 | const int vector_num = 50000;
```

```

2 vector<int> succ[vector_num + 10], prod[vector_num + 10],
   bucket[vector_num + 10], dom_t[vector_num + 10];
3 int semi[vector_num + 10], anc[vector_num + 10],
   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[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(),
109         dom_t[i].clear();
110     memset(ans,0,sizeof(*ans)*(n+3));
111 }
112 while(scanf("%d%d",&n,&m)!=EOF) {
113     init(n,m);
114     for(int i=0,u,v; i<m; i++) {
115         scanf("%d%d",&u,&v);
116         succ[u].push_back(v);
117     }
118     lengauer_tarjan(n);
119     get_ans(1);
120     for(int i=1; i<=n; i++)
121         printf("%I64d%c", ans[i], i == n ? '\n' : ' ');
122 }

```

3 DP

3.1 2D-LIS

```

1  set<pair<int,int>> re[100100];
2  int cc;
3  const int oo = (1 << 30);
4  bool judge(int idx, pair<int, int> x) {
5      if(re[idx].empty())return true;
6      set<pair<int,int>>::iterator it = re[idx].lower_bound(
7          {x.first - 1, oo});
8      if(it == re[idx].begin())return false;
9      it--;
10     if((*it).first < x.first && (*it).second <
11         x.second)return true;
12     return false;
13 }
14 void update(pair<int, int> x) {
15     if(0 == cc || judge(cc - 1, x)) {
16         re[cc++].insert(x);
17         return ;
18     }
19     int l = -1, r = cc - 1;
20     while(l + 1 < r) {
21         int mid = (l + r) >> 1;
22         if(judge(mid, x)) {
23             l = mid;
24         } else {
25             r = mid;
26         }
27     }
28 }

```

```

25 }
26 set<pair<int,int>>::iterator it = re[r].lower_bound(x);
27 if(it != re[r].begin()) {
28     it--;
29     if((*it).second <= x.second)return ;
30     it++;
31 }
32 while(it != re[r].end() && x.second <=
33     (*it).second)re[r].erase(it++);
34 re[r].insert(x);

```

3.2 2D-RMQ

```

1  void init() {
2      for(int i = 0; i < line; i++)
3          for(j = 0; j < row; j++)
4              dp[i][j][0][0] = mat[i][j];
5      for(int i = 0; i < 10; i++) {
6          for(int j = 0; j < 10; j++) {
7              if(i == 0 && j == 0)continue;
8              for(int l = 0; l + C(i) <= line; l++) {
9                  for(int r = 0; r + C(j) <= row; r++) {
10                     if(i == 0)dp[l][r][i][j] = max(dp[l][r][i][j - 1],
11                         dp[l][r + C(j - 1)][i][j - 1]);
12                     else dp[l][r][i][j] = max(dp[l][r][i - 1][j], dp[l
13                         + C(i - 1)][r][i - 1][j]);
14                 }
15             }
16         }
17     }
18     int Q(int x1, int y1, int x2, int y2) {
19         int lx = Log[x2 - x1 + 1];
20         int ly = Log[y2 - y1 + 1];
21         int ret = max(dp[x1][y1][lx][ly], dp[x2 - C(lx) +
22             1][y1][lx][ly]);
23         return max(ret, max(dp[x1][y2 - C(ly) + 1][lx][ly], dp[x2
24             - C(lx) + 1][y2 - C(ly) + 1][lx][ly]));
25     }
26 }

```

3.3 LCA

```

1  void update(int u, int fa) {
2      dep[u] = dep[fa] + 1;
3      dp[u][0] = fa;
4      for(int i = 1; i < 17; i++) {
5          dp[u][i] = dp[dp[u][i - 1]][i - 1];
6      }
7  }
8  int lca(int u, int v) {
9      if(dep[u] < dep[v]) swap(u, v);
10     for(int df = dep[u] - dep[v], t = 0; df; df >= 1, t++)
11         if(df & 1) u = dp[u][t];
12     if(u == v) return u;
13     for(int i = 16; i >= 0; i--) {
14         if(dp[u][i] != dp[v][i]) {
15             u = dp[u][i];
16             v = dp[v][i];
17         }
18     }
19     return dp[u][0];
20 }
21
22 int F[MAXN*2]; //Euler Sequence
23 int P[MAXN]; //first time point i appears in F
24 void dfs(int u, int pre, int dep) {
25     F[++cnt] = u;
26     rmq[cnt] = dep;
27     P[u] = cnt;
28     for(int i = head[u]; i != -1; i = edge[i].next) {
29         int v = edge[i].to;
30         if(v == pre)continue;
31         dfs(v, u, dep+1);
32     }
33     F[++cnt] = u;
34     rmq[cnt] = dep;
35 }

```

```

35 }
36 dp[i][j] = rmq[dp[i][j-1]] <
    rmq[dp[i+(1<<(j-1))][j-1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];
37 return rmq[dp[a][k]] <=
    rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[b-(1<<k)+1][k];
38 return F[st.query(P[u],P[v])];

```

3.4 Steiner Tree

```

1 for (int i = 1; i <= n; ++i)
2   for (int j = 0; j < 1 << 4; ++j)
3     if (dis[i][j] == 0) que.push((Opt) {0, i, j});
4 while (!que.empty()) {
5   int x = que.top().x, mask = que.top().mask;
6   que.pop();
7   if (bo[x][mask]) continue;
8   bo[x][mask] = 1;
9   for (int k = first[x]; k != -1; k = edge[k].nex) {
10    int y = edge[k].y;
11    if (dis[y][mask] > dis[x][mask] + edge[k].s) {
12      dis[y][mask] = dis[x][mask] + edge[k].s;
13      que.push((Opt) {dis[y][mask], y, mask});
14    }
15  }
16  for (int mask1 = 0; mask1 < 1 << 4; ++mask1)
17    if (dis[x][mask | mask1] > dis[x][mask] +
        dis[x][mask1]) {
18      dis[x][mask | mask1] = dis[x][mask] + dis[x][mask1];
19      que.push((Opt) {dis[x][mask | mask1], x, mask |
        mask1});
20    }
21 }

```

4 Graph

4.1 KM

```

1 const int INF = 1000000000;
2 int n;
3 int w[N][N], u[N], v[N], p[N], minv[N], fa[N];
4 bool used[N];
5 void km(int lev) {
6   int i = lev;
7   for (int j = 0; j <= n; ++j) {
8     minv[j] = INF; used[j] = false;
9   }
10  p[n] = i;
11  int j0 = n;
12  do {
13    used[j0] = true;
14    int i0 = p[j0], delta = INF, j1;
15    for (int j = 0; j < n; ++j) {
16      if (!used[j]) {
17        int cur = w[i0][j] - u[i0] - v[j];
18        if (cur < minv[j]) {
19          minv[j] = cur; fa[j] = j0;
20        }
21        if (minv[j] < delta) {
22          delta = minv[j]; j1 = j;
23        }
24      }
25    }
26    for (int j = 0; j <= n; ++j) {
27      if (used[j]) {
28        u[p[j]] += delta, v[j] -= delta;
29      } else {
30        minv[j] -= delta;
31      }
32    }
33    j0 = j1;
34  } while (p[j0] != -1);
35  do {
36    int j1 = fa[j0]; p[j0] = p[j1]; j0 = j1;
37  } while (j0 != n);
38 }
39 void solve() {

```

```

40 for(int i = 0; i < n; i++) {
41   km(i);
42 }
43 printf("%d\n", v[n]);
44 }
45 mst(p, -1); mst(u, 0); mst(v, 0);
46 for(int i = 0; i < n; i++) {
47   for(int j = 0; j < n; j++) {
48     scanf("%d", &w[i][j]);
49     w[i][j] = -w[i][j];
50   }
51 }
52 solve();

```

4.2 Hungary

```

1 int vis[N * N], match[N * N];
2 int fi(int u) {
3   for(int temp = head[u]; temp != -1; temp =
    edge[temp].next) {
4     int v = edge[temp].to;
5     if(vis[v])continue;
6     vis[v] = true;
7     if(-1 == match[v] || fi(match[v])) {
8       match[v] = u;
9       return true;
10    }
11  }
12  return false;
13 }
14 int Hungary() {
15   int ret = 0;
16   memset(match, -1, sizeof(match));
17   for(int i = 0; i < n * m; i++) {
18     memset(vis, 0, sizeof(vis));
19     if(fi(i))ret++;
20   }
21   return ret;
22 }

```

4.3 HK

```

1 int dist[N << 1], mx[N], my[N], m, n;
2 vector<int> map[N];
3 int que[N << 1], head, tail;
4 int bfs() {
5   int i;
6   head = 0; tail = -1;
7   for(i = 1; i <= n; i++)
8     if(mx[i] == -1) que[++tail] = i;
9   for(i = 0; i <= m + n; i++) dist[i] = 0;
10  int flag = 0;
11  while(head <= tail) {
12    int u = que[head++];
13    for(i = 0; i < map[u].size(); i++) {
14      int v = map[u][i];
15      if(dist[n + v] == 0) {
16        dist[n + v] = dist[u] + 1;
17        if(my[v] != -1) {
18          dist[my[v]] = dist[n + v] + 1;
19          que[++tail] = my[v];
20        } else flag = 1;
21      }
22    }
23  }
24  return flag;
25 }
26 int dfs(int u) {
27   for(int i = 0; i < map[u].size(); i++) {
28     int v = map[u][i];
29     if(dist[u] + 1 == dist[v + n]) {
30       int t = my[v]; dist[v + n] = 0;
31       if(t == -1 || dfs(t)) {
32         my[v] = u; mx[u] = v;
33         return 1;
34       }
35     }

```

```

36     }
37     return 0;
38 }
39 int H_K() {
40     int i;
41     for(i = 0; i <= n; i++) mx[i] = -1;
42     for(i = 0; i <= m; i++) my[i] = -1;
43     int ans = 0;
44     while(bfs()) {
45         for(i = 1; i <= n; i++)
46             if(mx[i] == -1 && dfs(i)) ans++;
47     }
48     return ans;
49 }

```

4.4 2-SAT

```

1 bool Dfs(int u) {
2     if(mark[u ^ 1]) return false;
3     if(mark[u]) return true;
4     mark[u] = true;
5     stack[++top] = u;
6     int temp;
7     for(temp = head[u]; temp != -1; temp = edge[temp].next)
8         if(!Dfs(edge[temp].to)) return false;
9     return true;
10 }
11 bool Solve() {
12     memset(mark, 0, sizeof(mark));
13     for(int i = 0; i < 2 * n; i += 2) {
14         if(!mark[i] && !mark[i+1]) {
15             top = 0;
16             if(!Dfs(i)) {
17                 while(top) mark[stack[top--]] = false;
18                 if(!Dfs(i + 1)) return false;
19             }
20         }
21     }
22     return true;
23 }

```

4.5 SAP

```

1 struct F {
2     void init(int N) {
3         this->N = N;
4         mst(head, -1); countedge = 0;
5     }
6     //Edge(head[s], t, cap); Edge(head[t], s, 0);
7     int sap(const int& s, const int& t) {
8         mst(numh, 0); mst(h, 0); mst(pre, -1);
9         int i;
10        for(i = 0; i < N; i++) curedge[i] = head[i];
11        numh[0] = N;
12        int u = s, curflow, neck, ret = 0;
13        while(h[s] < N) {
14            if(u == t) {
15                curflow = oo;
16                for(i = s; i != t; i = edge[curedge[i]].to) {
17                    if(curflow > edge[curedge[i]].cap) {
18                        curflow = edge[curedge[i]].cap;
19                        neck = i;
20                    }
21                }
22                for(int i = s; i != t; i = edge[curedge[i]].to) {
23                    int e = curedge[i];
24                    edge[e].cap -= curflow;
25                    edge[e ^ 1].cap += curflow;
26                }
27                ret += curflow;
28                u = neck;
29            }
30            for(i = curedge[u]; i != -1; i = edge[i].next) {
31                if(edge[i].cap && h[edge[i].to] + 1 == h[u]) break;
32            }
33            if(i != -1) {
34                curedge[u] = i;

```

```

35        pre[edge[i].to] = u;
36        u = edge[i].to;
37    } else {
38        if(0 == --numh[h[u]]) break;
39        curedge[u] = head[u];
40        int H = N;
41        for(i = head[u]; i != -1; i = edge[i].next) {
42            if(edge[i].cap) H = min(H, h[edge[i].to] + 1);
43        }
44        h[u] = H;
45        numh[H]++;
46        if(u != s) u = pre[u];
47    }
48 }
49 return ret;
50 }
51 bool s_side[MAXN];
52 void bfs(const int& s) {
53     mst(s_side, 0);
54     queue<int> que; que.push(s);
55     while(!que.empty()) {
56         int u = que.front();
57         que.pop();
58         for(int tmp = head[u]; tmp != -1; tmp =
59             edge[tmp].next) {
60             int v = edge[tmp].to;
61             if(edge[tmp].cap && !s_side[v]) {
62                 s_side[v] = true;
63                 que.push(v);
64             }
65         }
66     }
67 }
68 int p[MAXN], c[MAXN][MAXN];
69 void gusfield() {
70     mst(p, 0); mst(c, 63);
71     for(int i = 1; i < N; i++) {
72         for(int j = 0; j < countedge; j += 2) {
73             edge[j].cap += edge[j ^ 1].cap;
74             edge[j ^ 1].cap = 0;
75         }
76         int f = sap(i, p[i]);
77         bfs(i);
78         for(int j = i + 1; j < N; j++) {
79             if(s_side[j] && p[j] == p[i]) {
80                 p[j] = i;
81             }
82         }
83         c[i][p[i]] = c[p[i]][i] = f;
84         for(int j = 0; j < i; j++) {
85             c[i][j] = c[j][i] = min(f, c[p[i]][j]);
86         }
87     }
88     for(int i = 0; i < N; i++)
89         c[i][i] = 0;
90 } f;
91 mst(h, -1);
92 h[t] = 0; numh[0] = 1;
93 queue<int> que; que.push(t);
94 while(!que.empty()) {
95     int u = que.front(); que.pop();
96     for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
97         int v = edge[tmp].to;
98         if(h[v] != -1) continue;
99         que.push(v); h[v] = h[u] + 1; ++numh[h[v]];
100    }
101 }
102 }
103 c(u,v) = 1; c(s,v) = oo; c(v,t) = oo + 2 * g - du(v)
104 c(u,v) = we; c(s,v) = oo; c(v,t) = oo + 2 * g -
105     sumofedge(v) - 2 * p(v)
106 h(g) = (m * n - cut(S,T)) / 2
107
108 for(e: E){
109     cap(u, v) = up(u, v) - low(u, v)
110     cap(S', v) = low(u, v)
111     cap(v, T') = low(u, v)
112     tflow += low(u, v)
113 }

```



```

114 int ans1 = f.sap(S', T');
115 cap(T, S) = oo;
116 int ans2 = f.sap(S', T');
117 if(ans1 + ans2 == sum) ans2;
118 else puts("Impossible");

```

4.6 MCMF

```

1 struct MCMF {
2     Edge edge[MAXM];
3     int head[MAXN], countedge;
4     void Init(int N) {
5         this->N = N;
6         memset(head, -1, sizeof(head));
7         countedge = 0;
8     }
9     int N;
10    int inq[MAXN], dis[MAXN], pre[MAXN], ad[MAXN];
11    bool BF(int s, int t, int& cost) {
12        int i;
13        for(i = 0; i < N; i++) dis[i] = INF;
14        memset(inq, 0, sizeof(inq));
15        dis[s] = 0; inq[s] = 1; pre[s] = -1; ad[s] = INF;
16        queue<int> que; que.push(s);
17        int u, v, temp;
18        while(!que.empty()) {
19            u = que.front(); que.pop();
20            inq[u] = 0;
21            for(temp = head[u]; temp != -1; temp =
22                edge[temp].next) {
23                v = edge[temp].to;
24                if(edge[temp].cap && dis[v] > dis[u] +
25                    edge[temp].cost) {
26                    dis[v] = dis[u] + edge[temp].cost; pre[v] = temp;
27                    ad[v] = min(ad[u], edge[temp].cap);
28                    if(!inq[v]) {
29                        inq[v] = 1; que.push(v);
30                    }
31                }
32            }
33            if(dis[t] == INF) return false;
34            cost += dis[t] * ad[t];
35            u = t;
36            while(u != s) {
37                edge[pre[u]].cap -= ad[t]; edge[pre[u] ^ 1].cap +=
38                    ad[t];
39                u = edge[pre[u]].from;
40            }
41            return true;
42        }
43        int MinC(int s, int t) {
44            int cost = 0;
45            while(BF(s, t, cost));
46            return cost;
47        }
48    };
49    /*****zkw*****/
50    struct MaxFlow {
51        int N;
52        int st, en, maxflow, mincost;
53        bool vis[MAXN];
54        Edge edge[MAXM];
55        int head[MAXN], countedge, pre[MAXN], cur[MAXN],
56            dis[MAXN];
57        std::queue<int> Q;
58        void init(int N) {
59            this->N = N;
60            memset(head, -1, sizeof(head));
61            countedge = 0;
62        }
63        bool modell() {
64            int v, min = oo;
65            for(int i = 0; i <= N; i++) {
66                if(!vis[i])
67                    continue;
68                for(int j = head[i]; v = edge[j].v, j != -1; j =

```

```

68                edge[j].next)
69                    if(edge[j].cap
70                        if(!vis[v] && dis[v]-dis[i]+edge[j].cost < min)
71                            min = dis[v] - dis[i] + edge[j].cost;
72            }
73            if(min == oo)
74                return false;
75            for(int i = 0; i <= N; i++)
76                if(vis[i])
77                    cur[i] = head[i], vis[i] = false, dis[i] += min;
78            return true;
79        }
80        int augment(int i, int flow) {
81            if(i == en) {
82                mincost += dis[st] * flow; maxflow += flow;
83                return flow;
84            }
85            vis[i] = true;
86            for(int j = cur[i], v; v = edge[j].v, j != -1; j =
87                edge[j].next) {
88                if(!edge[j].cap
89                    continue;
90                if(vis[v] || dis[v]+edge[j].cost != dis[i])
91                    continue;
92                int delta = augment(v, min(flow, edge[j].cap));
93                if(delta) {
94                    edge[j].cap -= delta; edge[j^1].cap += delta;
95                    cur[i] = j;
96                    return delta;
97                }
98            }
99            return 0;
100        }
101        void spfa() {
102            int u, v;
103            for(int i = 0; i <= N; i++)
104                vis[i] = false, dis[i] = oo;
105            dis[st] = 0; Q.push(st); vis[st] = true;
106            while(!Q.empty()) {
107                u = Q.front(), Q.pop(); vis[u] = false;
108                for(int i = head[u]; v = edge[i].v, i != -1; i =
109                    edge[i].next) {
110                    if(!edge[i].cap || dis[v] <= dis[u] + edge[i].cost)
111                        continue;
112                    dis[v] = dis[u] + edge[i].cost;
113                    if(!vis[v]) {
114                        vis[v] = true; Q.push(v);
115                    }
116                }
117            }
118        }
119        for(int i = 0; i <= N; i++)
120            dis[i] = dis[en] - dis[i];
121    }
122    int zkw(int s, int t) {
123        st = s, en = t;
124        spfa();
125        mincost = maxflow = 0;
126        for(int i = 0; i <= N; i++)
127            vis[i] = false, cur[i] = head[i];
128        do {
129            while(augment(st, oo))
130                memset(vis, false, sizeof(vis));
131        } while(modell());
132        return mincost;
133    }
134    } f;

```

4.7 BCC

```

1 void dfs(int u, int fa) {
2     dfn[u] = low[u] = ++idx;
3     int child = 0;
4     for(int temp = head[u]; temp != -1; temp =
5         edge[temp].next) {
6         int v = edge[temp].to;
7         pair<int,int> e = make_pair(u, v);
8         if(!dfn[v]) {
9             st.push(e);
10            child++;

```

```

10     dfs(v, u);
11     low[u] = min(low[u], low[v]);
12     if(low[v] >= dfn[u]) {
13         iscut[u] = true;
14         bcc_cnt++;
15         while(1) {
16             pair<int,int> x = st.top();
17             st.pop();
18             if(bccno[x.first] != bcc_cnt)
19                 bccno[x.first] = bcc_cnt;
20             if(bccno[x.second] != bcc_cnt)
21                 bccno[x.second] = bcc_cnt;
22             if(x.first == u && x.second == v) break;
23         }
24     } else if(dfn[v] < dfn[u] && v != fa) {
25         st.push(e);
26         low[u] = min(low[u], dfn[v]);
27     }
28 }
29 if(fa < 0 && child == 1) {
30     iscut[u] = 0;
31 }
32 }
33 }
34 void find_bcc(int n) {
35     pre = 0;
36     mst(dfn, 0); mst(iscut, 0); mst(bccno, 0);
37     idx = bcc_cnt = 0;
38     while(!st.empty()) st.pop();
39     for(int i = 0; i < n; i++) {
40         if(!dfn[i]) {
41             dfs(i, -1);
42             pre++;
43         }
44     }
45 }

```

4.8 SCC

```

1  int dfn[N], low[N], sccno[N], scc_cnt, idx;
2  stack<int> st;
3  void dfs(int u) {
4      dfn[u] = low[u] = ++idx;
5      st.push(u);
6      for(int temp = head[u]; temp != -1; temp =
7          edge[temp].next) {
8          int v = edge[temp].to;
9          if(!dfn[v]) {
10             dfs(v);
11             low[u] = min(low[u], low[v]);
12         } else if(!sccno[v]) {
13             low[u] = min(low[u], dfn[v]);
14         }
15     }
16     if(low[u] == dfn[u]) {
17         scc_cnt++;
18         while(1) {
19             int x = st.top();
20             st.pop();
21             sccno[x] = scc_cnt;
22             W[scc_cnt] += w[x];
23             if(x == u) {
24                 break;
25             }
26         }
27     }
28 }
29 void find_scc(int n) {
30     idx = scc_cnt = 0;
31     while(!st.empty()) {
32         st.pop();
33     }
34     memset(sccno, 0, sizeof(sccno));
35     memset(dfn, 0, sizeof(dfn));
36     for(int i = 1; i <= n; i++) {
37         if(!dfn[i]) {
38             dfs(i);
39         }
40     }
41 }

```

```
40 | }
```

4.9 DLX

```

1  struct DLX {
2      int U[M], D[M], L[M], R[M], C[M];
3      int H[N], S[N], Q[N];
4      int size, ret;
5      void remove(const int& c) {
6          L[R[c]] = L[c]; R[L[c]] = R[c];
7          for(int i = D[c]; i != c; i = D[i]) {
8              for(int j = R[i]; j != i; j = R[j]) {
9                  U[D[j]] = U[j]; D[U[j]] = D[j]; --S[C[j]];
10             }
11         }
12     }
13     void resume(const int& c) {
14         for(int i = U[c]; i != c; i = U[i]) {
15             for(int j = L[i]; j != i; j = L[j]) {
16                 ++S[C[j]]; U[D[j]] = j; D[U[j]] = j;
17             }
18         }
19         L[R[c]] = c; R[L[c]] = c;
20     }
21     void dfs(const int& k) {
22         if(k >= ret) {
23             return;
24         }
25         if(!R[0]) {
26             ret = k;
27             return;
28         }
29         int c;
30         for(int tmp = M, i = R[0]; i; i = R[i]) {
31             if(S[i] < tmp) {
32                 tmp = S[c = i];
33             }
34         }
35         remove(c);
36         for(int i = D[c]; i != c; i = D[i]) {
37             for(int j = R[i]; j != i; j = R[j]) {
38                 remove(C[j]);
39             }
40             dfs(k + 1);
41             for(int j = L[i]; j != i; j = L[j]) {
42                 resume(C[j]);
43             }
44         }
45         resume(c);
46     }
47     void update(const int& r, const int& c) {
48         ++S[C[size] = c];
49         D[size] = D[c]; U[D[c]] = size;
50         U[size] = c; D[c] = size;
51         if(H[r] < 0) H[r] = L[size] = R[size] = size;
52         else {
53             R[size] = R[H[r]]; L[R[H[r]]] = size;
54             L[size] = H[r]; R[H[r]] = size;
55         }
56         size++;
57     }
58     void init(int col) {
59         ret = (1 << 30);
60         for(int i = 0; i <= col; i++) {
61             S[i] = 0; D[i] = U[i] = i;
62             L[i + 1] = i; R[i] = i + 1;
63         }
64         R[col] = 0; size = col + 1;
65         memset(H, -1, sizeof(H));
66     }
67 } dlx;
68
69 struct DLX {
70     int U[M], D[M], L[M], R[M], C[M], X[M];
71     int H[N], S[N], Q[N];
72     int size;
73     void remove(const int& c) {
74         for(int i = D[c]; i != c; i = D[i]) {
75             R[L[i]] = R[i]; L[R[i]] = L[i];

```

```

76     }
77 }
78 void resume(const int& c) {
79     for(int i = U[c]; i != c; i = U[i]) {
80         R[L[i]] = i; L[R[i]] = i;
81     }
82 }
83 int h() {
84     int ret = 0;
85     bool vis[60];
86     memset(vis, false, sizeof(vis));
87     for (int i = R[0]; i; i = R[i]) {
88         if(vis[i])continue;
89         ret++;
90         vis[i] = true;
91         for (int j = D[i]; j != i; j = D[j])
92             for (int k = R[j]; k != j; k = R[k])
93                 vis[C[k]] = true;
94     }
95     return ret;
96 }
97 int ret;
98 void dfs(const int& k) {
99     if(!R[0] || k + h() >= ret) {
100         if(!R[0]) ret = min(ret, k);
101         return ;
102     }
103     int c;
104     for(int tmp = M, i = R[0]; i; i = R[i]) {
105         if(S[i] < tmp) {
106             tmp = S[c = i];
107         }
108     }
109     for(int i = D[c]; i != c; i = D[i]) {
110         Q[k] = i;
111         remove(i);
112         for(int j = R[i]; j != i; j = R[j]) {
113             remove(j);
114         }
115         dfs(k + 1);
116         for(int j = L[i]; j != i; j = L[j]) {
117             resume(j);
118         }
119         resume(i);
120     }
121 }
122 void update(const int& r, const int& c) {
123     ++S[C[size] = c];
124     D[size] = D[c]; U[D[c]] = size;
125     U[size] = c; D[c] = size;
126     if(H[r] < 0) H[r] = L[size] = R[size] = size;
127     else {
128         R[size] = R[H[r]]; L[R[H[r]]] = size;
129         L[size] = H[r]; R[H[r]] = size;
130     }
131     X[size++] = r;
132 }
133 void init(int col) {
134     for(int i = 0; i <= col; i++) {
135         S[i] = 0; D[i] = U[i] = i;
136         L[i+1] = i; R[i] = i + 1;
137     }
138     R[col] = 0;
139     size = col + 1;
140     memset(H, -1, sizeof(H));
141     ret = (1 << 30);
142 }
143 }dlx;

```

4.10 Spanning Tree On Directed Graph

```

1 int zhuliu(int root, int n, int m, Edge e[]) {
2     int res = 0, u, v;
3     while (true) {
4         for (int i = 0; i < n; i++)
5             in[i] = inf;
6         for (int i = 0; i < m; i++)

```

```

7         if (e[i].u != e[i].v && e[i].cost < in[e[i].v]) {
8             pre[e[i].v] = e[i].u;
9             in[e[i].v] = e[i].cost;
10        }
11        for (int i = 0; i < n; i++)
12            if (i != root)
13                if (in[i] == inf) return -1;
14        int tn = 0;
15        mst(id, -1); mst(visit, -1);
16        in[root] = 0;
17        for (int i = 0; i < n; i++) {
18            res += in[i]; v = i;
19            while (visit[v] != i && id[v] == -1 && v != root) {
20                visit[v] = i; v = pre[v];
21            }
22            if (v != root && id[v] == -1) {
23                for(int u = pre[v] ; u != v ; u = pre[u])
24                    id[u] = tn;
25                id[v] = tn++;
26            }
27        }
28        if(tn == 0) break;
29        for (int i = 0; i < n; i++)
30            if (id[i] == -1)
31                id[i] = tn++;
32        for (int i = 0; i < m; i++) {
33            int v = e[i].v;
34            e[i].u = id[e[i].u]; e[i].v = id[e[i].v];
35            if (e[i].u != e[i].v)
36                e[i++].cost -= in[v];
37            else
38                swap(e[i], e[--m]);
39        }
40        n = tn; root = id[root];
41    }
42    return res;
43 }

```

4.11 Blossom

```

1 void Push(int u) {
2     Queue[Tail] = u; Tail++; InQueue[u] = true;
3 }
4 int Pop() {
5     int res = Queue[Head]; Head++; return res;
6 }
7 int FindCommonAncestor(int u, int v) {
8     memset(InPath, false, sizeof(InPath));
9     while(true) {
10         u = Base[u]; InPath[u] = true;
11         if(u == Start) break;
12         u = Father[Match[u]];
13     }
14     while(true) {
15         v = Base[v];
16         if(InPath[v])break;
17         v = Father[Match[v]];
18     }
19     return v;
20 }
21 void ResetTrace(int u) {
22     int v;
23     while(Base[u] != NewBase) {
24         v = Match[u];
25         InBlossom[Base[u]] = InBlossom[Base[v]] = true;
26         u = Father[v];
27         if(Base[u] != NewBase) Father[u] = v;
28     }
29 }
30 void BlossomContract(int u, int v) {
31     NewBase = FindCommonAncestor(u, v);
32     memset(InBlossom, false, sizeof(InBlossom));
33     ResetTrace(u); ResetTrace(v);
34     if(Base[u] != NewBase) Father[u] = v;
35     if(Base[v] != NewBase) Father[v] = u;
36     for(int tu = 1; tu <= N; tu++)
37         if(InBlossom[Base[tu]]) {
38             Base[tu] = NewBase;
39             if(!InQueue[tu]) Push(tu);

```

```

40     }
41 }
42 void FindAugmentingPath() {
43     mst(InQueue, 0); mst(Father, 0);
44     for(int i = 1; i <= N; i++)
45         Base[i] = i;
46     Head = Tail = 1; Push(Start); Finish = 0;
47     while(Head < Tail) {
48         int u = Pop();
49         for(int v = 1; v <= N; v++)
50             if(Graph[u][v] && (Base[u] != Base[v]) && (Match[u]
51                 != v)) {
52                 if((v == Start) || ((Match[v] > 0) &&
53                     Father[Match[v]] > 0))
54                     BlossomContract(u, v);
55                 else if(Father[v] == 0) {
56                     Father[v] = u;
57                     if(Match[v] > 0)
58                         Push(Match[v]);
59                     else {
60                         Finish = v;
61                         return;
62                     }
63                 }
64             }
65     }
66 }
67 void AugmentPath() {
68     int u, v, w;
69     u = Finish;
70     while(u > 0) {
71         v = Father[u]; w = Match[v];
72         Match[v] = u; Match[u] = v;
73         u = w;
74     }
75 }
76 void Edmonds() {
77     memset(Match, 0, sizeof(Match));
78     for(int u = 1; u <= N; u++)
79         if(Match[u] == 0) {
80             Start = u;
81             FindAugmentingPath();
82             if(Finish > 0) AugmentPath();
83         }
84 }
85 int getMatch() {
86     Edmonds();
87     Count = 0;
88     for(int u = 1; u <= N; u++)
89         if(Match[u] > 0)
90             Count++;
91     return Count / 2;
92 }
93 bool g[MAXN][MAXN];
94 int main() {
95     int m;
96     while(scanf("%d%d", &N, &m) == 2) {
97         mst(g, 0); mst(Graph, 0);
98         int u, v;
99         for(int i = 1; i <= m; i++) {
100             scanf("%d%d", &u, &v);
101             p[i] = make_pair(u, v);
102             g[u][v] = true; g[v][u] = true;
103             Graph[u][v] = true; Graph[v][u] = true;
104         }
105         int cnt0 = getMatch();
106     }
107     return 0;
108 }

```

4.12 Clique

```

1 bool array[maxnum][maxnum];
2 bool use[maxnum];
3 int cn, bestn, p, e;
4 void dfs(int i) {
5     int j;
6     bool flag;
7     if(i > p) {

```

```

8         bestn = cn;
9         printf("%d\n", bestn);
10        for(j = 1; j <= p; j++)
11            if(use[j])
12                printf("%d ", j);
13        printf("\n");
14        return ;
15    }
16    flag = true;
17    for(j = 1; j < i; j++)
18        if(use[j] && !array[j][i]) {
19            flag = false;
20            break;
21        }
22    if(flag) {
23        cn++; use[i] = true; dfs(i + 1); cn--;
24    }
25    if(cn + p - i > bestn) {
26        use[i] = false; dfs(i + 1);
27    }
28 }
29 scanf("%d%d", &u, &v);
30 array[u][v] = true; array[v][u] = true;
31 cn = bestn = 0; dfs(1);

```

4.13 Count The Number Of Spanning Tree

```

1 Matrix laplacian;
2 laplacian.clear();
3 for (int i = 0; i < n; i++)
4     for (int j = 0; j < n; j++)
5         if (i != j && G[i][j]) {
6             laplacian.a[i][j] = -1;
7             laplacian.a[i][i]++;
8         }
9 printf("%d\n", laplacian.det(n-1));

```

5 Math

5.1 Cantor

```

1 1324 0 * 3! + 1 * 2! + 0 * 1! = 2

```

5.2 FastWalshHadamard Transform

```

1 void tf(int a[], int l, int r) {
2     if(l + 1 == r) return ;
3     int len = (r - l) >> 1, mid = l + len;
4     tf(a, l, mid);
5     tf(a, mid, r);
6     for(int i = l; i < mid; i++) {
7         a[i] += a[i + len];
8         a[i] %= mod;
9     }
10 }
11 void utf(int a[], int l, int r) {
12     if(l + 1 == r) return ;
13     int len = (r - l) >> 1, mid = l + len;
14     for(int i = l; i < mid; i++) {
15         a[i] = a[i] + mod - a[i + len];
16         a[i] %= mod;
17     }
18     utf(a, l, mid);
19     utf(a, mid, r);
20 }
21
22 AND
23 tf(A) = (tf(A0) + tf(A1), tf(A1))
24 utf(A) = (utf(A0) - utf(A1), utf(A1))
25
26 XOR
27 tf(A) = (tf(A0) + tf(A1), tf(A0) - tf(A1))

```

```

28 | utf(A) = (utf((A0 + A1) / 2), utf((A0 - A1) / 2))
29 |
30 | OR
31 | tf(A) = (tf(A0), tf(A0) + tf(A1))
32 | utf(A) = (utf(A0), utf(A1) - utf(A0))

```

5.3 FFT

```

1 | complex<double> x1[N];
2 | void change(complex<double> y[], int len) {
3 |     for(int i = 1, j = len / 2; i < len - 1; i++) {
4 |         if(i < j) swap(y[i], y[j]);
5 |         int k = len / 2;
6 |         for(; k <= j; k >= 1) j -= k;
7 |         if(j < k) j += k;
8 |     }
9 | }
10 | void fft(complex<double> y[], int len, int on) {
11 |     change(y, len);
12 |     for(int h = 2; h <= len; h <= 1) {
13 |         complex<double> wn(cos(-on * 2 * PI / h), sin(-on * 2 *
14 |             PI / h));
15 |         for(int j = 0; j < len; j += h) {
16 |             complex<double> w(1, 0);
17 |             for(int k = j; k < j + h / 2; k++) {
18 |                 complex<double> u = y[k];
19 |                 complex<double> t = w * y[k + h / 2];
20 |                 y[k] = u + t;
21 |                 y[k + h / 2] = u - t;
22 |                 w = w * wn;
23 |             }
24 |         }
25 |     }
26 |     //x1[i]=complex<double>(b[i], 0); fft(x1, n, 1);
27 |     //x1[i]=complex<double>(1,0)/x1[i];fft(x1,n,-1);
28 |     //x1[i]/=n;c[i]=x1[i].real();//Cyclic matrix

```

5.4 Fibonacci Sequence

```

1 | long long f(long long x) {
2 |     if(Fib.count(x)) return Fib[x];
3 |     return Fib[x] = (f((x + 1) / 2) * f(x / 2) + f((x - 1) /
4 |         2) * f((x - 2) / 2)) % MOD;
5 | }

```

5.5 Integer Partition

```

1 | void init() {
2 |     f[0] = 1;
3 |     int i, j, k, flag = 1;
4 |     for(i = 1; i <= 100000; i++) {
5 |         f[i] = 0;
6 |         for(j = 1; ; j++) {
7 |             int t = i - j * (3 * j - 1) / 2;
8 |             int tt = i - j * (3 * j + 1) / 2;
9 |             if(j & 1) flag = 1;
10 |             else flag = -1;
11 |             if(t < 0 && tt < 0) break;
12 |             if(t >= 0) f[i] = (f[i] + flag * f[t]) % mod;
13 |             if(tt >= 0) f[i] = (f[i] + flag * f[tt]) % mod;
14 |         }
15 |         f[i] = (f[i] + mod) % mod;
16 |     }
17 | }

```

5.6 Miller-Rabin

```

1 | inline u32 fb(u32 n) {
2 |     u32 k = 31, c = n - 1;
3 |     while(!(c & (1U << k))) k--;
4 |     return k;
5 | }
6 | bool witness(u32 a, u32 nbits, u32 n) {

```

```

7 |     u32 d = 1, n1 = n - 1;
8 |     for(int i = nbits; i >= 0; i--) {
9 |         u32 x = d;
10 |        d = mul(d, d, n);
11 |        if (d == 1 && x != 1 && x != n1) return true;
12 |        if (n1 & (1U << i)) d = mul(a, d, n);
13 |    }
14 |    return d != 1;
15 | }
16 | bool isprime(u32 n) {
17 |     int nbits = fb(n);
18 |     if (witness(2, nbits, n)) return false;
19 |     if (n == 61) return true;
20 |     if (witness(61, nbits, n)) return false;
21 |     if (n < 916327) return true;
22 |     if (witness(7, nbits, n)) return false;
23 |     return true;
24 | }
25 | //2, 325, 9375, 28178, 450775, 9780504, 1795265022

```

5.7 Mul Mod

```

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

```

5.8 NTT

```

1 | const int P = 880803841;
2 | const int G = 26;
3 | const int NUM = 24;
4 | struct solve {
5 |     long long wn[NUM];
6 |     void GetWn() {
7 |         for(int i = 0; i < NUM; i++) {
8 |             int t = 1 << i;
9 |             wn[i] = quick_mod(G, (P - 1) / t, P);
10 |        }
11 |    }
12 |    void Rader(long long a[], int len) {
13 |        int j = len >> 1;
14 |        for(int i = 1; i < len - 1; i++) {
15 |            if(i < j) swap(a[i], a[j]);
16 |            int k = len >> 1;
17 |            while(j >= k) {
18 |                j -= k;
19 |                k >>= 1;
20 |            }
21 |            if(j < k) j += k;
22 |        }
23 |    }
24 |    void NTT(long long a[], int len, int on) {
25 |        Rader(a, len);
26 |        int id = 0;
27 |        for(int h = 2; h <= len; h <= 1) {
28 |            id++;
29 |            for(int j = 0; j < len; j += h) {
30 |                long long w = 1;
31 |                for(int k = j; k < j + h / 2; k++) {
32 |                    long long u = a[k] % P;
33 |                    long long t = w * (a[k + h / 2] % P) % P;
34 |                    a[k] = (u + t) % P;
35 |                    a[k + h / 2] = ((u - t) % P + P) % P;
36 |                    w = w * wn[id] % P;
37 |                }

```

```

38     }
39 }
40 if(on == -1) {
41     for(int i = 1; i < len / 2; i++)
42         swap(a[i], a[len - i]);
43     long long Inv = quick_mod(len, P - 2, P);
44     for(int i = 0; i < len; i++)
45         a[i] = a[i] % P * Inv % P;
46 }
47 }
48 void Conv(long long a[], long long b[], int n) {
49     NTT(a, n, 1);
50     NTT(b, n, 1);
51     for(int i = 0; i < n; i++)
52         a[i] = a[i] * b[i] % P;
53     NTT(a, n, -1);
54 }
55 } ntt;
56 ntt.GetWn();
57 ntt.Conv(A, B, 1 << 18); //twice length

```

5.9 Pollard-Rho

```

1 LL Pollard_Rho(LL n, LL c = 1) { // get a factor of n in
2     log(n)
3     LL i = 1, k = 2, x = rand() % (n - 1) + 1, y = x, d;
4     while(1) {
5         i++;
6         x = (mul_mod(x, x, n) + c) % n;
7         d = __gcd(n, y - x);
8         if(d > 1 && d < n) return d;
9         if(y == x) return n;
10        if(i == k) {
11            k <= 1;
12            y = x;
13        }
14    }

```

5.10 Primitive Root

```

1 struct PR {
2     int divs[N + 5];
3     int primitive_root(const int p) {
4         if (p == 2) return 1;
5         int cnt = 0, m = p - 1;
6         for (int i = 2; i * i <= m; ++i) if (m%i == 0) {
7             divs[cnt++] = i;
8             if (i * i < m) divs[cnt++] = m / i;
9         }
10        int r = 2, j = 0;
11        while (1) {
12            for (j = 0; j < cnt; ++j) {
13                if (fastpow(r, divs[j], p) == 1) break;
14            }
15            if (j >= cnt) return r;
16            r++;
17        }
18        return -1;
19    }
20 } pr_solver;

```

5.11 Romberg

```

1 double f(double x) {
2     return x * x;
3 }
4 const int MAXREPT = 10;
5 const double eps = 1e-5;
6 double y[MAXREPT];
7 double Romberg(double aa, double bb) {
8     int m, n;
9     double h, x, s, q, ep, p;
10    h = bb - aa;
11    y[0] = h * (f(aa) + f(bb)) / 2.0;

```

```

12    m = n = 1;
13    ep = eps + 1.0;
14    while ((ep > eps) && (m < MAXREPT)) {
15        p = 0.0;
16        for(int i = 0; i < n; i++)
17            x = aa + (i + 0.5) * h, p += f(x);
18        p = (y[0] + h * p) / 2.0;
19        s = 1.0;
20        for(int k = 1; k <= m; k++) {
21            s *= 4.0;
22            q = (s * p - y[k - 1]) / (s - 1.0);
23            y[k - 1] = p;
24            p = q;
25        }
26        p = fabs(q - y[m - 1]);
27        y[m++] = q;
28        n <= 1;
29        h /= 2.0;
30    }
31    return q;
32 }

```

6 nsqrt(n)

6.1 example

```

1 struct Re {
2     int l, r, idx;
3     Re() {}
4     Re(int l, int r, int idx): l(l), r(r), idx(idx) {}
5     bool operator < (const Re& re) const {
6         if(l / 173 == re.l / 173) return r < re.r;
7         return l / 173 < re.l / 173;
8     }
9 };
10
11 for(int i = 0; i < n; i += limit) {
12     ll[i / limit] = i;
13     rr[i / limit] = min(i + limit - 1, n - 1);
14     trie[i / limit].query();
15 }
16 l--;
17 r--;
18 int L = l / limit, R = r / limit;
19 for(int i = L; i <= R; i++) {
20     if(l <= ll[i] && rr[i] <= r);
21     else {
22         int x = max(l, ll[i]), y = min(r, rr[i]);
23         for(int j = x; j <= y; j++);
24     }
25 }
26
27 int dfs(int u, int fa) {
28     dfn[u] = ++dfs_idx;
29     int sz = 0;
30     for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
31         int v = edge[tmp].to;
32         if(v == fa) continue;
33         update(v, u);
34         C[v] = edge[tmp].w;
35         sz += dfs(v, u);
36         if(sz >= limit) {
37             lab_idx++;
38             for(int i = 0; i < sz; i++)
39                 lab[st[st_top--]] = lab_idx;
40             sz = 0;
41         }
42     }
43     st[++st_top] = u;
44     return sz + 1;
45 }

```


7 String

7.1 KMP

```

1 f[0] = f[1] = 0;
2 for(i = 1; i < 11; i++) {
3     j = f[i];
4     while(j && P[i] != P[j]) j = f[j];
5     f[i + 1] = (P[i] == P[j] ? j + 1 : 0);
6 }
7 for(i = 0; i < 12; i++) {
8     while(j && T[i] != P[j]) j = f[j];
9     if(T[i] == P[j]) j++;
10    if(j == 11) {
11        ret++;
12        j = f[j];
13    }
14 }
```

7.2 ExKMP

```

1 void exkmp(char s[], char t[]) {
2     int i, j, p, L;
3     int lens = strlen(s); int lent = strlen(t);
4     next[0] = lent;
5     j = 0;
6     while(j + 1 < lent && t[j] == t[j + 1]) j++;
7     next[1] = j;
8     int a = 1;
9     for(i = 2; i < lent; i++) {
10        p = next[a] + a - 1; L = next[i - a];
11        if(i + L < p + 1) next[i] = L;
12        else {
13            j = max(0, p - i + 1);
14            while(i + j < lent && t[i + j] == t[j]) j++;
15            next[i] = j; a = i;
16        }
17    }
18    j = 0;
19    while(j < lens && j < lent && s[j] == t[j]) j++;
20    extend[0] = j; a = 0;
21    for(i = 1; i < lens; i++) {
22        p = extend[a] + a - 1; L = next[i - a];
23        if(L + i < p + 1) extend[i] = L;
24        else {
25            j = max(0, p - i + 1);
26            while(i + j < lens && j < lent && s[i + j] == t[j]) j++;
27            extend[i] = j; a = i;
28        }
29    }
30 } //lcp(A[1..lenA-1], B)
```

7.3 Suffix Array

```

1 struct SuffixArray {
2     int s[N], sa[N], rk[N], height[N], t[N], t2[N], c[N], n;
3     void clear() {
4         n = 0;
5     }
6     void build_sa(int m) {
7         int *x = t, *y = t2;
8         for(int i = 0; i < m; i++) c[i] = 0;
9         for(int i = 0; i < n; i++) c[x[i]] = s[i]++;
10        for(int i = 1; i < m; i++) c[i] += c[i - 1];
11        for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
12        for(int k = 1; k <= n; k <= 1) {
13            int p = 0;
14            for(int i = n - k; i < n; i++) y[p++] = i;
15            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
16            for(int i = 0; i < m; i++) c[i] = 0;
17            for(int i = 0; i < n; i++) c[x[y[i]]]++;
18            for(int i = 1; i < m; i++) c[i] += c[i - 1];
19            for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] =
```

```

20            y[i];
21            swap(x, y);
22            p = 1;
23            x[sa[0]] = 0;
24            for(int i = 1; i < n; i++)
25                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i] - 1]
26                    + k == y[sa[i] + k]? p - 1: p++;
27            if(p >= n) break;
28            m = p;
29        }
30        void build_height() {
31            for(int i = 0; i < n; i++) rk[sa[i]] = i;
32            for(int i = 0, k = 0; i < n - 1; i++) {
33                if(k) k--;
34                int j = sa[rk[i] - 1];
35                while(s[i + k] == s[j + k]) k++;
36                height[rk[i]] = k;
37            }
38            void add(int ch) {
39                s[n++] = ch;
40            }
41        };
42        SuffixArray sa;
43        sa.clear();
44        int len = strlen(str);
45        for(int i = 0; i < len; i++) {
46            sa.add(str[i] - 'a' + 1);
47        }
48        sa.add(0);
49        sa.build_sa(27);
50        int callo(int x, int len) {
51            int l = 0, r = x;
52            while(l + 1 < r) {
53                int mid = (l + r) >> 1;
54                if(callcp(mid + 1, x) >= len) r = mid;
55                else l = mid;
56            }
57            return r;
58        }
59        int calup(int x, int len) {
60            int l = x, r = sa.n;
61            while(l + 1 < r) {
62                int mid = (l + r) >> 1;
63                if(callcp(x + 1, mid) >= len) l = mid;
64                else r = mid;
65            }
66            return l;
67        }
68        for(int i = 1; i < sa.n; i++) {
69            lcp[i][0] = sa.height[i];
70        }
71        for(int j = 1; j < 20; j++) {
72            for(int i = 1; i + (1 << j) <= sa.n; i++) {
73                lcp[i][j] = min(lcp[i][j - 1], lcp[i + (1 << (j - 1))][j - 1]);
74            }
75        }
76        int callcp(int a, int b) {
77            int tt = b - a + 1, len = Log[tt];
78            return min(lcp[a][len], lcp[a + tt - (1 << len)][len]);
79        }
80    }
```

7.4 Aho-Corasick Automaton

```

1 void insert(char str[]) {
2     int len = strlen(str);
3     int now = 0;
4     for(int i = 0; i < len; i++) {
5         int ind = str[i] - 'a';
6         if(!trie[now][ind]) {
7             total++;
8             memset(trie[total], 0, sizeof(trie[total]));
9             trie[now][ind] = total;
10        }
11        now = trie[now][ind];
12    }
13    flag[now]++;
```

```

14 }
15 void AC() {
16     front = rear = 0;
17     for(int i = 0; i < 26; i++) {
18         if(trie[0][i]) {
19             que[rear++] = trie[0][i];
20             f[trie[0][i]] = 0;
21         }
22     }
23     while(front < rear) {
24         int u = que[front++];
25         for(i = 0; i < 26; i++) {
26             if(trie[u][i]) {
27                 int v = trie[u][i];
28                 que[rear++] = v;
29                 f[v] = trie[f[u]][i];
30             } else trie[u][i] = trie[f[u]][i];
31         }
32     }
33 }
34 void AC() {
35     queue<int> que;
36     for(map<int,int>::iterator it = trie[0].begin(); it !=
37         trie[0].end(); it++) {
38         que.push(it->second);
39         f[it->second] = 0;
40     }
41     while(!que.empty()) {
42         int u = que.front();
43         que.pop();
44         for(map<int,int>::iterator it = trie[u].begin(); it !=
45             trie[u].end(); it++) {
46             int v = it->second;
47             que.push(v);
48             int x = f[u];
49             for(; x && !trie[x].count(it->first); x = f[x]);
50             f[v] = trie[x][it->first];
51         }
52     }
53 }

```

7.5 Minimum Representation

```

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

```

7.6 Manacher

```

1 int add(int l, int len) {
2     if(len <= 0) return 0;
3     long long x = (h[l] - h[l + len] * p[len] % mod + mod) %
4         mod;
5     if(vis[x]) return vis[x];
6     else {
7         int now = ++idx;
8         int fa = add(l + 1, len - 2);
9         g[fa].push_back(now);
10        return vis[x] = now;
11    }
12 }
13 void Manacher(int len) {
14     tot = 0;

```

```

14 a[tot++] = -1; a[tot++] = 0;
15 int len = strlen(s);
16 for(int i = 0; i < len; i++) {
17     a[tot++] = s[i] - 'a' + 1; a[tot++] = 0;
18 }
19 a[tot] = 0;
20 for(int i = 1, mx = 0, id; i < tot; i++) {
21     if(mx > i) cc[i] = min(cc[2 * id - i], mx - i);
22     else cc[i] = 1;
23     for(; a[i + cc[i]] == a[i - cc[i]]; cc[i]++);
24     if(cc[i] + i > mx) mx = cc[i] + i, id = i;
25 }
26 }
27 for(int i = 1; i < tot; i++) {
28     int len = cc[i] - 1, l = (i - 1) / 2 - len / 2;
29     cnt[add(l, len)]++;
30 }

```

7.7 Palindromic Tree

```

1 struct Palindromic_Tree {
2     int next[MAXN][N], fail[MAXN], cnt[MAXN], num[MAXN],
3         len[MAXN], S[MAXN];
4     int last, n, p;
5     int newnode(int l) {
6         for(int i = 0; i < N; ++i) next[p][i] = 0;
7         cnt[p] = 0; num[p] = 0; len[p] = l;
8         return p++;
9     }
10    void init() {
11        p = 0; newnode(0); newnode(-1);
12        last = 0; n = 0; S[n] = -1; fail[0] = 1;
13    }
14    int get_fail(int x) {
15        while(S[n - len[x] - 1] != S[n]) x = fail[x];
16        return x;
17    }
18    void add(int c) {
19        c -= 'a';
20        S[++n] = c;
21        int cur = get_fail(last);
22        if(!next[cur][c]) {
23            int now = newnode(len[cur] + 2);
24            fail[now] = next[get_fail(fail[cur])][c];
25            next[cur][c] = now;
26            num[now] = num[fail[now]] + 1;
27        }
28        last = next[cur][c];
29        cnt[last]++;
30    }
31    void count() {
32        for(int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
33    }
34 } T;
35 void init ( int n ) {
36     p = 0 ; newnode ( 0 ) ; newnode ( -1 ) ;
37     clr ( S , -1 ) ; L = n ; R = n - 1 ;
38     last[0] = last[1] = 1 ; fail[0] = 1 ; tot = 0 ;
39 }
40 int get_fail ( int v , bool d ) {
41     if ( d ) while ( S[R - len[v] - 1] != S[R] ) v = fail[v] ;
42     else while ( S[L + len[v] + 1] != S[L] ) v = fail[v] ;
43     return v ;
44 }
45 void add ( int c , bool d ) {
46     if ( d ) S[++R] = c ;
47     else S[--L] = c ;
48     int cur = get_fail ( last[d] , d ) ;
49     if ( !nxt[cur][c] ) {
50         int now = newnode ( len[cur] + 2 ) ;
51         fail[now] = nxt[get_fail ( fail[cur] , d )][c] ;
52         nxt[cur][c] = now ;
53         num[now] = num[fail[now]] + 1 ;
54     }
55     last[d] = nxt[cur][c] ;
56     if ( len[last[d]] == R - L + 1 ) last[d ^ 1] = last[d] ;
57     tot += num[last[d]] ;
58 }

```

7.8 SAM

```

1  if(p->go[x]) {
2      cc++; p = p->go[x];
3  } else {
4      while(p && !p->go[x]) p = p->pre;
5      if(0 == p) {
6          p = root; cc = 0;
7      } else {
8          cc = p->step + 1; p = p->go[x];
9      }
10 }
11 struct State {
12     State *pre, *go[26], *dp[20], *next[26];
13     int step, mxstep, val;
14     long long les;
15     void clear() {
16         pre = 0;
17         step = val = 0;
18         mst(go, 0); mst(dp, 0); mst(next, 0);
19     }
20 } *root, *last;
21 State statePool[N * 2], *b[2 * N], *cur;
22 int c[N];
23 void init() {
24     cur = statePool; root = last = cur++;
25     root->clear();
26 }
27 void extend(int w) {
28     State *p = last; State *np = cur++;
29     np->clear();
30     np->step = np->mxstep = p->step + 1;
31     np->val = 1;
32     while(p && !p->go[w])
33         p->go[w] = np, p = p->pre;
34     if(p == 0)
35         np->pre = root;
36     else {
37         State *q = p->go[w];
38         if(p->step + 1 == q->step)
39             np->pre = q;
40         else {
41             State *nq = cur++;
42             nq->clear();
43             memcpy(nq->go, q->go, sizeof(q->go));
44             nq->step = p->step + 1;
45             nq->pre = q->pre;
46             nq->mxstep = q->mxstep;
47             q->pre = nq;
48             np->pre = nq;
49             while(p && p->go[w] == q)
50                 p->go[w] = nq, p = p->pre;
51         }
52     }
53     last = np;
54 }
55 void dfs(State* p, long long& ret) {
56     if(p->pre) ret += 1LL * p->val * (p->step - p->pre->step);
57     p->les = ret;
58     for(int i = 0; i < 26; i++) {
59         if(p->next[i]) dfs(p->next[i], ret);
60     }
61 }
62 int main() {
63     scanf("%s", s);
64     int len = strlen(s);
65     reverse(s, s + len);
66     init();
67     for(int i = 0; i < len; i++) {
68         extend(s[i] - 'A'); accept[i] = last;
69     }
70     for(State *p = statePool; p < cur; p++) {
71         p->dp[0] = p->pre; c[p->step]++;
72     }
73     for(int i = 1; i <= len; i++) c[i] += c[i - 1];
74     for(State *p = cur - 1; p >= statePool; p--)
75         b[--c[p->step]] = p;
76     for(int i = 0; i < cur - statePool; i++) {
77         State *p = b[i];
78         for(int i = 1; i < 20; i++) {

```

```

78         p->dp[i] = (0 == p->dp[i - 1]? 0: p->dp[i - 1]->dp[i
79             - 1]);
80     }
81     for(int i = cur - statePool - 1; i; i--) {
82         State *p = b[i];
83         p->pre->val += p->val;
84         p->pre->next[s[p->mxstep - p->pre->step - 1] - 'A'] = p;
85     }
86     ret = 0;
87     dfs(root, ret);
88     int q;
89     scanf("%d", &q);
90     for(int i = 0; i < q; i++) {
91         int x, y;
92         scanf("%d%d", &x, &y);
93         x = len - x;
94         y = len - y;
95         swap(x, y);
96         State *now = accept[y];
97         int len = (y - x + 1);
98         for(int i = 19; i >= 0; i--) {
99             if(now->dp[i] && now->dp[i]->step >= len) {
100                 now = now->dp[i];
101             }
102         }
103         long long ans = now->les - 1LL * (now->step - len + 1)
104             * now->val;
105         printf("%I64d\n", ans);
106     }
107     return 0;
108 }

```

8 Tree

8.1 Divide And Conquer Tree

```

1  int sz[N], vis[N], fa[N], que[N];
2  int TT;
3  int n, q;
4  int getroot(int u, int& tot) {
5      int mi = n, root = 0;
6      int l = 0, r = 0;
7      que[r++] = u;
8      fa[u] = 0;
9      while(l < r) {
10         u = que[l++];
11         for(int i = 0; i < g[u].size(); i++) {
12             int v = g[u][i];
13             if(v == fa[u] || vis[v] == TT) continue;
14             que[r++] = v;
15             fa[v] = u;
16         }
17     }
18     tot = r;
19     for(--l; l >= 0; l--) {
20         int x = que[l], ma = 0;
21         sz[x] = 1;
22         for(int i = 0; i < g[x].size(); i++) {
23             int v = g[x][i];
24             if(v == fa[x] || vis[v] == TT) continue;
25             ma = max(ma, sz[v]);
26             sz[x] += sz[v];
27         }
28         ma = max(ma, r - sz[x]);
29         if(ma < mi) {
30             mi = ma, root = x;
31         }
32     }
33     return root;
34 }
35 int dis[N];
36 void go(int u, int rr) {
37     int l = 0, r = 0;
38     que[r++] = u;
39     fa[u] = 0;
40     dis[u] = 1;

```

```

41 while(l < r) {
42     u = que[l++];
43     re[u].push_back(Re(rr, idx_of_bit, dis[u]));
44     for(int i = 0; i < g[u].size(); i++) {
45         int v = g[u][i];
46         if(v == fa[u] || vis[v] == TT) continue;
47         que[r++] = v;
48         fa[v] = u;
49         dis[v] = dis[u] + 1;
50     }
51 }
52 bit[idx_of_bit].tree.resize(r + 1, 0);
53 for(int i = 0; i < r; i++) {
54     int u = que[i];
55     bit[idx_of_bit].update(dis[u], w[u]);
56     bit[rr].update(dis[u], w[u]);
57 }
58 }
59 void solve(int u) {
60     int tot;
61     int root = getroot(u, tot);
62     vis[root] = TT;
63     int r = ++idx_of_bit;
64     re[root].push_back(Re(r, 0, 0));
65     bit[r].tree.resize(tot + 1, 0);
66     bit[r].update(0, w[root]);
67     int tmp = root;
68     for(int i = 0; i < g[tmp].size(); i++) {
69         int v = g[tmp][i];
70         if(vis[v] == TT) continue;
71         ++idx_of_bit;
72         go(v, r);
73     }
74     for(int i = 0; i < g[tmp].size(); i++) {
75         int v = g[tmp][i];
76         if(vis[v] == TT) continue;
77         solve(v);
78     }
79 }
80 //dfs
81 void predfs(int u, int fa) {
82     sz[u] = 1;
83     for(int v: g[u]) {
84         if(v == fa || vis[v]) continue;
85         predfs(v, u);
86         sz[u] += sz[v];
87     }
88 }
89 int root;
90 int mx[N], mi;
91 void dfsroot(int u, int fa, int r) {
92     mx[u] = sz[r] - sz[u];
93     for(int v: g[u]) {
94         if(v == fa || vis[v]) continue;
95         dfsroot(v, u, r);
96         mx[u] = max(mx[u], sz[v]);
97     }
98     if(mx[u] < mi) {
99         mi = mx[u]; root = u;
100 }
101 }
102 void cal(int u, int fa, Re pre) {
103     long long s = 0, ss = 0;
104     for(int i = 0; i < K; i++) {
105         s += (6 - pre.s[i] - cnt[root][i]) % 3 * p3[i];
106         ss += pre.s[i] * p3[i];
107     }
108     ret += mp[s];
109     re.push_back(ss);
110     for(int v: g[u]) {
111         if(v == fa || vis[v]) continue;
112         Re now;
113         for(int j = 0; j < K; j++)
114             now.s[j] = (pre.s[j] + cnt[v][j]) % 3;
115         cal(v, u, now);
116     }
117 }
118 void dfs(int u) {
119     mp.clear(); mp[0] = 1;
120     for(int v: g[u]) {

```

```

122         if(vis[v]) continue;
123         re.clear();
124         Re now;
125         for(int j = 0; j < K; j++)
126             now.s[j] = cnt[v][j];
127         cal(v, u, now);
128         for(int j = 0; j < re.size(); j++)
129             mp[re[j]]++;
130     }
131 }
132 void gao(int u) {
133     mi = n; root = u;
134     predfs(u, 0); dfsroot(u, 0, u);
135     vis[root] = true; dfs(root);
136     int tmp = root;
137     for(int v: g[tmp]) {
138         if(vis[v]) continue;
139         gao(v);
140     }
141 }

```

8.2 Heavy-Light Decomposition

```

1 void init() {
2     memset(head, -1, sizeof(head));
3     countedge = 0;
4     loc = 0;
5     mst(tp, -1); mst(fa, -1); mst(son, -1); mst(w, -1);
6 }
7 void dfs(int u) {
8     sz[u] = 1;
9     for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
10         int v = edge[tmp].to;
11         if(v == fa[u]) continue;
12         fa[v] = u; dep[v] = dep[u] + 1;
13         dfs(v);
14         if(-1 == son[u] || sz[v] > sz[son[u]]) son[u] = v;
15         sz[u] += sz[v];
16     }
17 }
18 void build_tree(int u, int f) {
19     w[u] = ++loc; tp[u] = f;
20     if(son[u] != -1) {
21         build_tree(son[u], f);
22     }
23     for(int tmp = head[u]; tmp != -1; tmp = edge[tmp].next) {
24         int v = edge[tmp].to;
25         if(v == son[u] || v == fa[u]) {
26             continue;
27         }
28         build_tree(v, v);
29     }
30 }
31 int solve(int x, int y) {
32     int f1 = tp[x], f2 = tp[y], ret = 0;
33     while(f1 != f2) {
34         if(dep[f1] < dep[f2]) {
35             swap(f1, f2); swap(x, y);
36         }
37         ret = max(ret, query(1, 1, loc, w[f1], w[x]));
38         x = fa[f1]; f1 = tp[x];
39     }
40     if(x == y) return ret;
41     if(dep[x] > dep[y]) swap(x, y);
42     return max(ret, query(1, 1, loc, w[son[x]], w[y]));
43 }
44 dep[1] = 0;
45 dfs(1);
46 build_tree(1, 1);
47 for(int i = 1; i < n; i++) {
48     if(dep[re[i][0]] > dep[re[i][1]]) swap(re[i][0], re[i][1]);
49     update(1, 1, loc, w[re[i][1]], re[i][2]);
50 }
51 void solve(int x, int y, int val) {
52     int f1 = tp[x], f2 = tp[y], ret = 0;
53     while(f1 != f2) {
54         if(dep[f1] < dep[f2]) {
55             swap(f1, f2); swap(x, y);
56         }

```

```

57     update(1, 1, loc, w[f1], w[x], val);
58     x = fa[f1]; f1 = tp[x];
59 }
60 if(dep[x] > dep[y]) swap(x, y);
61 update(1, 1, loc, w[x], w[y], val);
62 }
63
64 void prepare() {/****bfs****/
65     dep[0] = 0; fa[1] = 0;
66     f = r = 0;
67     que[r++] = 1;
68     for(; f < r; ) {
69         int u = que[f++]; dep[u] = dep[fa[u]] + 1;
70         for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
71             int v = edge[tmp].to;
72             if(v != fa[u]) {
73                 fa[v] = u; que[r++] = v;
74             }
75         }
76     }
77     for(int i = r-1; i >= 0; i--) {
78         int u = que[i]; sz[u]++;
79         if(-1 == son[fa[u]] || sz[u] > sz[son[fa[u]]]) {
80             son[fa[u]] = u;
81         }
82         sz[fa[u]] += sz[u];
83     }
84 }
85 void build_tree() {
86     deque<pair<int, int>> que;
87     que.push_front( {1, 1} );
88     while(!que.empty()) {
89         pair<int, int> p = que.front();
90         que.pop_front();
91         int u = p.first, f = p.second;
92         w[u] = ++loc; tp[u] = f;
93         if(son[u] != -1) {
94             que.push_front( {son[u], f} );
95         }
96         for(int tmp = head[u]; ~tmp; tmp = edge[tmp].next) {
97             int v = edge[tmp].to;
98             if(v != fa[u] && v != son[u]) {
99                 que.push_back( {v, v} );
100             }
101         }
102     }
103 }

```

9 Others

9.1 bash Script

```

1 while true; do
2     ./gen > input
3     ./sol < input > output.sol
4     ./bf < input > output.bf
5
6     diff output.sol output.bf
7     if [ $? -ne 0 ] ; then break; fi
8 done

```

9.2 tool

```

1 int __size__ = 256 << 20;
2 char * __p__ = (char *) malloc(__size__) + __size__;
3 __asm__("movl %0,%esp\n:::r"(__p__));
4
5 template <class T>
6 inline bool scan_d(T &ret) {
7     char c;
8     int sgn;
9     if(c = getchar(), c == EOF) return 0; //EOF
10    while(c != '-' && (c < '0' || c > '9')) c = getchar();
11    sgn = (c == '-')? -1: 1;
12    ret = (c == '-')? 0: (c - '0');
13    while(c = getchar(), c >= '0' && c <= '9') ret = ret * 10

```

```

14     + (c - '0');
15     ret *= sgn;
16     return 1;
17 }
18 for(int y = 0; y < n; y++) D(x) += d(x, y);
19 (n - 1) * d(x, y) + D(y) - D(x)
20
21 (2k-1)^2 n*(4*n*n-1)/3
22 (2*k-1)^3 n*n*(2*n*n-1)
23 k*(k+1) n*(n+1)*(n+2)/3
24 k*(k+1)*(k+2) n*(n+1)*(n+2)*(n+3)/4
25 (1+...+n)1
26 (1+...+n^2)1/2 1/2
27 (1+...+n^3)1/6 1/2 1/3
28
29 set cindent autoindent number sts=4 sw=4 ts=4 et
30 set background=dark
31 set backspace=indent,eol,start
32 set autochdir
33 syntax on
34 map <F4> :!g++ -g -std=c++11 % -o %<<cr>
35 map <F5> :!%<<cr>

```

9.3 java

```

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

```

9.4 Minimum Representation

```

1 string str1, str2;
2 string min_pre(string str) {
3     vector<string> box;
4     string ret = "";
5     int equal = 0, st = 0;
6     for(int i = 0; i < str.size(); i++) {
7         if(str[i] == '0') equal++;
8         else equal--;
9         if(equal == 0) {
10            if(i - 1 > st + 1) {
11                box.push_back("0" + min_pre(str.substr(st + 1, i - 1
12                    - st)) + "1");
13            } else box.push_back("01");
14            st = i + 1;
15        }
16    }
17    sort(box.begin(), box.end());
18    for(int i = 0; i < box.size(); i++) ret += box[i];
19    return ret;
20 }
21 while(cin >> str1 >> str2) if(min_pre(str1) ==
    min_pre(str2)) printf("same\n");

```