

Table Of Contents

bitprime-mammad	1	1
bitprime-saman	1	32
cutedge	2	58
cutvertex	3	133
equation	5	254
eulerian	8	427
geometry	10	517
grid	23	1251
huge	24	1311
line-polygon-inter	27	1477
match-haamed	29	1568
mat-inverse	30	1615
maxflow	32	1728
maxflow-arash	34	1844
maxflow-haamed	38	2069
polygon-similarity	39	2131
stronglyconnected	41	2218
trie-haamed	42	2301
wmatching-babak	43	2359

Table Of Contents Of Geometry

equal	10	529
less	10	534
greater	10	539
lessEqual	10	544
greaterEqual	10	549
round	10	554
sign	10	561
vectorAngle	11	566
angle	11	578
squaredDistance	11	593
distance	11	598
segmentToLine	11	603
lineThroughPoint	11	611
segmentsIntersect	12	619
linesIntersectionPoint	12	646
segmentsIntersectionPoint	12	669
segmentsIntersectionPoint2	14	744
pointOnLine	15	795
multiply	15	800
pointInSegment	15	808
rotate	15	815
circlesIntersect	15	827
pointToLineDistance	16	847
circleLineIntersect	16	853
circleLineIntersectionPoints	16	870
circlesIntersectionPoints	17	912
polygonArea	17	930
convexHullGW	17	943
convexHullGS	19	1018
polygonOrientation	20	1077
generalPolygonInclusion	20	1091
angleInclusion	21	1113
convexPolygonInclusion	21	1127
perpendicularProjection	22	1175
circleThroughPoints	22	1196
cross	23	1218
dot	23	1222
ang	23	1227
rotate	23	1231
rotate	23	1239

```

1 =====
2 BIT PRIME - MAMMAD
3 =====
4 #include<iostream>
5 #include<string>
6 using namespace std;
7
8 #define MAXSIEVE 100000000
9 #define MAXSIEVEHALF (MAXSIEVE/2)
10 #define MAXSQRT 5000
11 #define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7)))
12 #define max(a,b) ((a)<(b)?(b):(a))
13 #define min(a,b) ((a)>(b)?(b):(a))
14 #define isp(n) ((n)==1?0:(n&1?isprime(n):(n)==2))
15
16 char a[MAXSIEVE/16+2];
17
18 main()
19 {
20     int i,j;
21     memset(a,255,sizeof(a));
22     a[0]=0xFE;
23     for(i=1;i<MAXSQRT;i++)
24         if (a[i>>3]&(1<<(i&7)))
25             for(j=(i<<1)+i+1;j<MAXSIEVEHALF;j+=(i<<1)+1)
26                 a[j>>3]&=~(1<<(j&7));
27     int n;
28     while(cin>>n)
29         cout<<(isp(n)?"prime":"not prime");
30     return 0;
31 }
32 =====
33 BIT PRIME - SAMAN
34 =====
35 #include<iostream>
36 #include<string>
37
38 using namespace std;
39
40 char primes[12500002];
41
42 main()
43 {
44     //computes the primes numbers between 1 to 100,000,000
45     memset(primes,85,sizeof primes);
46     primes[0]=83;
47     for(int i=3;i<10000;i+=2)
48         if(!((primes[i>>3])&(1<<(i&7))))
49             for(int j=i*i;j<100000000;j+=i)
50                 primes[j>>3]|=1<<(j&7);
51     //end
52     //how to check if some number is prime
53     int x;//the number we want to see if that's prime or not
54     if(!((primes[x>>3])&(1<<(x&7)))) cout<<"x is prime"<<endl;
55     //end
56     return 0;
57 }

```

```

58 =====
59 CUT EDGE
60 =====
61 #include <vector>
62
63 using namespace std;
64
65 #define NODES 100
66
67 // ----- List CutEdge
68 vector<int> neigh[NODES];
69 vector<pair<int, int> > cut;
70 int n, dfi[NODES], low[NODES], t;
71 bool mark[NODES];
72
73 void dfsList(int v, int p)
74 {
75     mark[v] = true;
76     dfi[v] = t++;
77     low[v] = dfi[v];
78     for (vector<int>::iterator it = neigh[v].begin(); it !=
79         neigh[v].end(); it++)
80         if (*it != p && mark[*it] && dfi[*it] < low[v])
81             low[v] = dfi[*it];
82         else if (!mark[*it])
83             {
84                 dfsList(*it, v);
85                 if (low[*it] < low[v])
86                     low[v] = low[*it];
87                 if (low[*it] == dfi[*it])
88                     cut.push_back(make_pair(v, *it));
89             }
90 }
91
92 void cutEdgeList()
93 {
94     t = 0;
95     memset(mark, false, n * sizeof(bool));
96     cut = vector<pair<int, int> >();
97     for (int i = 0; i < n; i++)
98         if (!mark[i])
99             dfsList(i, -1);
100 }
101
102 // ----- Matrix CutEdge
103 int n, graph[NODES][NODES], dfi[NODES], t, low[NODES];
104 bool mark[NODES];
105 vector<pair<int, int> > cut;
106
107 void dfsMatrix(int v, int p)
108 {
109     mark[v] = true;
110     dfi[v] = t++;
111     low[v] = dfi[v];
112     for (int i = 0; i < n; i++)
113         if (i != p && graph[v][i] && mark[i] && dfi[i] < low[v])
114             low[v] = dfi[i];

```

```

114     else if (graph[v][i] && !mark[i])
115     {
116         dfsMatrix(i, v);
117         if (low[i] < low[v])
118             low[v] = low[i];
119         if (low[i] == dfi[i])
120             cut.push_back(make_pair(v, i));
121     }
122 }
123
124 void cutEdgeMatrix()
125 {
126     t = 0;
127     memset(mark, false, n * sizeof(bool));
128     cut = vector<pair<int, int>> >();
129     for (int i = 0; i < n; i++)
130         if (!mark[i])
131             dfsMatrix(i, -1);
132 }
133
134 CUT VERTEX - BICONNECTED COMPONENT
135
136 #include <iostream>
137 #include <vector>
138 #include <set>
139
140 using namespace std;
141
142 #define NODES 100
143
144 vector<int> st;
145 vector<vector<int>> > components;
146
147 void componentFound(int v)
148 {
149     components.push_back(vector<int>());
150     while (st.back() != v)
151     {
152         components.back().push_back(st.back());
153         st.pop_back();
154     }
155     components.back().push_back(v);
156 }
157
158 // ----- Matrix CutVertex
159
160 int n, graph[NODES][NODES], dfi[NODES], t, low[NODES], f;
161 bool mark[NODES];
162 set<int> cut;
163
164 void dfsMatrix(int v)
165 {
166     st.push_back(v);
167     int branches = 0;
168     mark[v] = true;
169     dfi[v] = t++;
170     low[v] = dfi[v] - 1;

```

```

171     for (int i = 0; i < n; i++)
172         if (graph[v][i] && mark[i] && dfi[i] < low[v])
173             low[v] = dfi[i];
174     else if (graph[v][i] && !mark[i])
175     {
176         branches++;
177         dfsMatrix(i);
178         if (low[i] < low[v])
179             low[v] = low[i];
180         if (low[i] == dfi[v])
181         {
182             if (v != f)
183                 cut.insert(v);
184             componentFound(v);
185         }
186     }
187     if (v == f && branches > 1)
188         cut.insert(f);
189 }
190
191 void cutVertexMatrix()
192 {
193     t = 0;
194     memset(mark, false, n * sizeof(bool));
195     cut = set<int>();
196     st = vector<int>();
197     components = vector<vector<int>> >();
198     for (int i = 0; i < n; i++)
199         if (!mark[i])
200         {
201             dfsMatrix(f = i);
202             st.pop_back();
203         }
204 }
205
206 // ----- List CutVertex
207
208 vector<int> neigh[NODES];
209 int n, t, f, dfi[NODES], low[NODES];
210 bool mark[NODES];
211 set<int> cut;
212
213 void dfsList(int v)
214 {
215     st.push_back(v);
216     int branches = 0;
217     mark[v] = true;
218     dfi[v] = t++;
219     low[v] = dfi[v] - 1;
220     for (vector<int>::iterator it = neigh[v].begin(); it !=
221         neigh[v].end(); it++)
222         if (mark[*it] && dfi[*it] < low[v])
223             low[v] = dfi[*it];
224         else if (!mark[*it])
225         {
226             branches++;
227             dfsList(*it);

```

```

227         if (low[*it] < low[v])
228             low[v] = low[*it];
229         if (low[*it] == dfi[v])
230         {
231             if (v != f)
232                 cut.insert(v);
233             componentFound(v);
234         }
235     }
236     if (v == f && branches > 1)
237         cut.insert(f);
238 }
239
240 void cutVertexList()
241 {
242     t = 0;
243     memset(mark, false, n * sizeof(bool));
244     cut.clear();
245     st.clear();
246     components = vector<vector<int>> >();
247     for (int i = 0; i < n; i++)
248         if (!mark[i])
249         {
250             dfsList(f = i);
251             st.pop_back();
252         }
253 }
254
255 EQUATOIN
256
257 /*-----*
258 * Linear Equation Solver
259 * * Solves a system of linear equations using Gauss Method.
260 * * You should :
261 *     a) put equation matrix in mat[0..n - 1][0..n - 1]
262 *     b) put array of answers in mat[n][0..n - 1]
263 *     c) call solve(n)
264 *
265 * * Solve(n) returns :
266 *     a) 0 : if system has a unique answer
267 *     b) 1 : if system has infinite answers
268 *     c) 2 : if system has no answers
269 *
270 * * It also fills mark[0..n - 1] with above flags (0, 1, 2) to
271 *     indicate that the system had unique/infinite/no answer(s)
272 *     for parameter[0..n - 1]
273 *
274 * * If the system has a unique answer for parameter[i] then after
275 *     calling solve(n) the answer for parameter[i] equals:
276 *     (mat[i][i] / mat[i][n])
277 *
278 *-----*/
279 #include <fstream>
280 #include <cmath>
281 #include <cstring>
282
283 using namespace std;

```

```

284
285 const double epsilon = 1e-6
286 ;
287
288 double mat[100][101];
289 int mark[100];
290
291 int ZeroRow(int r, int n)
292 {
293     for (int i = 0; i < n; i++)
294         if (fabs(mat[r][i]) >= epsilon)
295             return 0;
296     return 1;
297 }
298
299 void AddRow(double a, int source, int target, int n)
300 {
301     for (int i = 0; i < n + 1; i++)
302         mat[target][i] += a * mat[source][i];
303 }
304
305 void ChangeRow(int source, int target, int n)
306 {
307     double temp;
308     for (int i = 0; i < n + 1; i++)
309     {
310         temp = mat[target][i];
311         mat[target][i] = mat[source][i];
312         mat[source][i] = temp;
313     }
314 }
315
316 int Solve(int n)
317 {
318     int i, j;
319     int flag = 0;
320     memset(mark, 0, sizeof(mark));
321     for (i = 0; i < n; i++)
322     {
323         j = i;
324         for (j = i; (fabs(mat[j][i]) < epsilon) && (j < n); j++);
325         if ((j == n) && mark[i-1])
326             j = i - 1;
327
328         if (j == n)
329         {
330             flag = 1;
331             mark[i] = 1;
332         }
333
334         if ((j != i) && (j != n))
335             ChangeRow(j, i, n);
336
337         if (j != n)
338             for (j = 0; j < n; j++)
339                 if (j != i)
340                     AddRow((-mat[j][i] / mat[i][i]), i, j, n);

```

```

341
342     for (j = 0; j < n; j++)
343         if (fabs(mat[j][i]) < epsilon)
344             mat[j][i] = 0;
345
346     }
347
348     if (flag == 1)
349     {
350         for (i = 0; i < n; i++)
351             if (ZeroRow(i, n))
352             {
353                 if ((fabs(mat[i][n]) >= epsilon))
354                 {
355                     mark[i]++;
356                     for(int m = 0; m < n; m++)
357                         if(!mark[m] && mat[m][i])
358                             mark[m] = 2;
359                     flag = 2;
360                 }
361                 else
362                 {
363                     for (int m = 0; m < n; m++)
364                         if (!mark[m] && mat[m][i])
365                             mark[m] = 1;
366                 }
367             }
368
369     return flag;
370 }
371
372 return 0;
373 }
374
375 // A Test for Equation Solver
376 int main()
377 {
378     memset(mat, 0, sizeof(mat));
379
380     mat[0][0] = 1; mat[0][1] = 1; mat[0][2] = 0; mat[0][3] = 1;
381     mat[0][4] = 0;
382     mat[1][0] = 1; mat[1][1] = 2; mat[1][2] = 0; mat[1][3] = 2;
383     mat[1][4] = 0;
384     mat[2][0] = 1; mat[2][1] = 3; mat[2][2] = 0; mat[2][3] = 3;
385     mat[2][4] = 0;
386
387     int n = 3;
388     int res = Solve(n);
389     int flag = 0;
390
391     for(int i = 0; i < n; i++)
392     {
393         switch (mark[i])
394         {
395             case 0:
396                 cout << "X(" << i << ") = " << mat[i][n] / mat[i][i] <<
397                     endl;

```

```

394         break;
395
396     case 1:
397         cout << "X(" << i << ") has infinite Answers . . ." <<
398             endl;
399         if (flag < 1)
400             flag = 1;
401         break;
402
403     case 2:
404         cout << "X(" << i << ") has no answers . . ." << endl;
405         if (flag < 2)
406             flag = 2;
407         break;
408     }
409
410     cout << "So the equation has ";
411     switch (flag)
412     {
413     case 0:
414         cout << "a Unique answer" << endl;
415         break;
416
417     case 1:
418         cout << "infinite answers" << endl;
419         break;
420
421     case 3:
422         cout << "no answers" << endl;
423         break;
424     }
425     return 0;
426 }
427
428 EULERIAN
429
430 #include<iostream>
431 #include<algorithm>
432 #include<string>
433 #include<list>
434 using namespace std;
435 typedef pair<int,int> _p;
436 int mark[50];
437 int graph[50][50];
438 int n;
439 void dfs(int v)
440 {
441     mark[v]=1;
442     for(int i=0;i<n;i++)
443         if(!mark[i]&&graph[v][i])dfs(i);
444 }
445 list<_p> euler(int v)
446 {
447     list<_p> local;
448     while(graph[v][v])
449         local.push_back(_p(v,v)),graph[v][v]--=2;

```

```

450 int flag=0,ret=v;
451 for(int i=0;i<n;i++)
452     if(graph[v][i])
453     {
454         graph[v][i]--;
455         graph[i][v]--;
456         local.push_back(_p(v,i));
457         v=i;
458         while(v!=ret)
459         {
460             for(int j=0;j<n;j++)
461                 if(graph[v][j])
462                 {
463                     graph[v][j]--;
464                     graph[j][v]--;
465                     local.push_back(_p(v,j));
466                     v=j;
467                     break;
468                 }
469             }
470         }
471     for(list<p>::iterator i=local.begin();i!=local.end();i++)
472     {
473         list<p> temp=euler(i->first);
474         local.insert(i,temp.begin(),temp.end());
475     }
476     return local;
477 }
478 main()
479 {
480     int N,m,a,b,flag;
481     cin>>N;
482     for(int z=1;z<=N;z++)
483     {
484         if(z>1)cout<<endl;
485         cout<<"Case #"<<z<<endl;
486         cin>>m;
487         flag=n=0;
488         memset(graph,0,sizeof graph);
489         for(int i=0;i<m;i++)
490         {
491             cin>>a>>b;
492             graph[a-1][b-1]++;
493             graph[b-1][a-1]++;
494             n=max(n,max(a,b));
495         }
496         dfs(n-1);
497         for(int i=0;i<n;i++)
498         {
499             int d=0;
500             for(int j=0;j<n;j++)
501                 d+=graph[i][j];
502             if(d&1||!mark[i]&&d)
503             {
504                 cout<<"some beads may be lost"<<endl;
505                 flag=1;break;
506             }

```

```

507     }
508     if(!flag)
509     {
510         list<p> l=euler(n-1);
511         for(list<p>::iterator i=l.begin();i!=l.end();i++)
512             cout<<i->first+1<<' '<<i->second+1<<endl;
513     }
514 }
515 return 0;
516 }
517 =====
518 GEOMETRY
519 =====
520 #include <cmath>
521 #include <algorithm>
522 #include <vector>
523
524 using namespace std;
525
526 const double PI = 3.1415926535897932385;//2 * acos(0.)
527 const double EPS = 1e-15;//5e-15 ?
528 const int PRECISION = 14;
529 inline bool equal(double x, double y)
530 {
531     return fabs(x - y) < EPS;
532 }
533
534 inline bool less(double x, double y)
535 {
536     return x <= y - EPS;
537 }
538
539 inline bool greater(double x, double y)
540 {
541     return x >= y + EPS;
542 }
543
544 inline bool lessEqual(double x, double y)
545 {
546     return x < y + EPS;
547 }
548
549 inline bool greaterEqual(double x, double y)
550 {
551     return x > y - EPS;
552 }
553
554 inline double round(double x, int p)
555 {
556     if (less(x, 0))
557         return floor(x * pow(10., p) - .5) / pow(10., p);
558     return floor(x * pow(10., p) + .5) / pow(10., p);
559 }
560
561 inline int sign(double x)
562 {
563     return (less(x, 0) ? -1 : (greater(x, 0) ? 1 : 0));

```

```

564 }
565
566 inline double vectorAngle(double x, double y)
567 {
568     double r = atan2(y, x);
569 /*
570     (-Pi < r <= Pi): the following 'if' statement should be excluded
571     (0 <= r < 2 * Pi): the following 'if' statement should be included
572 */
573     if (less(r, 0))
574         r += 2 * PI;
575     return r;
576 }
577
578 inline double angle(double x1, double y1, double x2, double y2, double
x3, double y3)
579 /*
580     angle (x1,y1)-(x2,y2)-(x3,y3)
581 */
582 {
583     double r = vectorAngle(x3 - x2, y3 - y2) - vectorAngle(x1 - x2, y1
- y2);
584 /*
585     (-Pi < r <= Pi): the following 'if' statement should be excluded
586     (0 <= r < 2 * Pi): the following 'if' statement should be included
587 */
588     if (less(r, 0))
589         r += 2 * PI;
590     return r;
591 }
592
593 inline double squaredDistance(double x1, double y1, double x2, double
y2)
594 {
595     return (x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1);
596 }
597
598 inline double distance(double x1, double y1, double x2, double y2)
599 {
600     return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
601 }
602
603 inline void segmentToLine(double x1, double y1, double x2, double y2,
double &a, double &b, double &c)
604 {
605     a = y2 - y1;
606     b = x1 - x2;
607     c = -y1 * b - x1 * a;
608 }
609
610
611 inline void lineThroughPoint(double x, double y, double m,
double &a, double &b, double &c)
612 {
613     a = -m;
614     b = 1;
615     c = m * x - y;
616 }
617

```

```

618
619 bool segmentsIntersect(double x1, double y1, double x2, double y2,
double x3, double y3, double x4, double y4)
620 {
621 /*
622     NOTE: when intersections at endpoints are valid, two intersecting
623     coincident segments are considered intersecting.
624     when intersections at endpoints are invalid, two intersecting
625     coincident segments are considered not intersecting.
626     it could be handled seperately!
627 */
628 {
629     if(greater(min(x1,x2),max(x3,x4)) || greater(min(x3,x4),max(x1,x2)) ||
greater(min(y1,y2),max(y3,y4)) || greater(min(y3,y4),max(y1,y2)))
630         return false;
631 /*
632     'greater' : intersections at endpoints are valid
633     'greaterEqual' : // // // // invalid
634 */
635     return false;
636     return ((lessEqual((x4-x2)*(y1-y2)-(y4-y2)*(x1-x2))*
((x3-x2)*(y1-y2)-(y3-y2)*(x1-x2)),0))&&
637         (lessEqual((x1-x4)*(y3-y4)-(y1-y4)*(x3-x4))*
((x2-x4)*(y3-y4)-(y2-y4)*(x3-x4)),0));
638 /*
639     'lessEqual' : intersections at endpoints are valid
640     'less' : // // // // invalid
641 */
642 }
643
644 int linesIntersectionPoint(double a1, double b1, double c1,
double a2, double b2, double c2,
double &x, double &y)
645 {
646 /*
647     return values:
648     0: lines are parallel
649     1: lines are intersecting at point (x, y)
650     2: lines are the same
651 */
652 {
653     if (equal(a1 * b2, a2 * b1))
654         if(equal(a1, 0))
655             return equal(b1 * c2, b2 * c1) * 2;
656         else
657             return equal(a1 * c2, a2 * c1) * 2;
658     y = -(a1 * c2 - a2 * c1) / (a1 * b2 - a2 * b1);
659     if (equal(a1, 0))
660         x = -(b2 * y + c2) / a2;
661     else
662         x = -(b1 * y + c1) / a1;
663     return 1;
664 }
665
666 int segmentsIntersectionPoint(double x1, double y1, double x2, double
y2,
double x3, double y3, double x4, double
y4,
double &x5, double &y5)
667 /*

```



```

673     uses:
674         segmentToLine(double, double, double, double, double, double,
        double)
675         segmentsIntersect(double, double, double, double, double,
        double, double, double)
676 */
677 /*
678     return values:
679         0: no intersection
680         1: intersection at point (x5, y5)
681         2: intersection is a line segment
682 */
683 {
684
685     if (!segmentsIntersect(x1, y1, x2, y2, x3, y3, x4, y4))
686 /*
687     intersections at endpoints should be valid
688 */
689         return 0;
690     double a1, b1, c1, a2, b2, c2;
691     segmentToLine(x1, y1, x2, y2, a1, b1, c1);
692     segmentToLine(x3, y3, x4, y4, a2, b2, c2);
693     if (equal(a1 * b2, a2 * b1))
694 /*
695     the segments are coincident and intersecting
696 */
697     {
698         if (less(x1, x2))
699         {
700             if (equal(x2, x3) && equal(y2, y3) && greaterEqual(x4, x2) ||
701                 equal(x2, x4) && equal(y2, y4) && greaterEqual(x3, x2))
702             {
703                 x5 = x2;
704                 y5 = y2;
705                 return 1;
706             }
707             if (equal(x1, x3) && equal(y1, y3) && lessEqual(x4, x1) ||
708                 equal(x1, x4) && equal(y1, y4) && lessEqual(x3, x1))
709             {
710                 x5 = x1;
711                 y5 = y1;
712                 return 1;
713             }
714         } else
715         {
716             if (equal(x2, x3) && equal(y2, y3) && lessEqual(x4, x2) ||
717                 equal(x2, x4) && equal(y2, y4) && lessEqual(x3, x2))
718             {
719                 x5 = x2;
720                 y5 = y2;
721                 return 1;
722             }
723             if (equal(x1, x3) && equal(y1, y3) && greaterEqual(x4, x1) ||
724                 equal(x1, x4) && equal(y1, y4) && greaterEqual(x3, x1))
725             {
726                 x5 = x1;
727                 y5 = y1;

```

```

728         return 1;
729     }
730 }
731 /*
732     one of the segments is lied on the other one
733 */
734     return 2;
735 }
736 y5 = -(a1 * c2 - a2 * c1) / (a1 * b2 - a2 * b1);
737 if (equal(a1, 0))
738     x5 = -(b2 * y5 + c2) / a2;
739 else
740     x5 = -(b1 * y5 + c1) / a1;
741     return 1;
742 }
743
744 int segmentsIntersectionPoint2(double x1, double y1, double x2, double
y2,
745                               double x3, double y3, double x4, double
y4,
746                               double &x5, double &y5)
747 /*
748     return values:
749         7 important bits of the result value are:
750         bit 0: 1 : segments are intersecting at (x5, y5)
751                0 : segments are not intersecting
752         bit 1: 1 : containing lines are parallel
753                0 : containing lines are not parallel
754         bit 2: 1 : containing lines are coincident
755                0 : containing lines are not coincident
756         bit 3: 1 : containing lines are intersecting on
757                the extension of segment 1-2 at (x5, y5)
758                0 : containing lines are not intersecting
759                on the extension of segment 1-2
760         bit 4: 1 : containing lines are intersecting on
761                the extension of segment 2-1 at (x5, y5)
762                0 : containing lines are not intersecting
763                on the extension of segment 2-1
764         bit 5: 1 : containing lines are intersecting on
765                the extension of segment 3-4 at (x5, y5)
766                0 : containing lines are not intersecting
767                on the extension of segment 3-4
768         bit 6: 1 : containing lines are intersecting on
769                the extension of segment 4-3 at (x5, y5)
770                0 : containing lines are not intersecting
771                on the extension of segment 4-3
772 */
773 {
774     double rnum = (y1 - y3) * (x4 - x3) - (x1 - x3) * (y4 - y3);
775     double den = (x2 - x1) * (y4 - y3) - (y2 - y1) * (x4 - x3);
776     double snum = (y1 - y3) * (x2 - x1) - (x1 - x3) * (y2 - y1);
777     int res = 0;
778     if (equal(den, 0))
779     {
780         res |= 2;
781         if (equal(rnum, 0)) res |= 4;
782         return res;

```

```

783     }
784     double r = rnum / den;
785     double s = snum / den;
786     x5 = x1 + r * (x2 - x1);
787     y5 = y1 + r * (y2 - y1);
788     if (greater(r, 1)) res |= 8;
789     if (less(r, 0)) res |= 16;
790     if (greater(s, 1)) res |= 32;
791     if (less(s, 0)) res |= 64;
792     return (res ? res : 1);
793 }
794
795 int pointOnLine(double x, double y, double a, double b, double c)
796 {
797     return equal(a * x + b * y + c, 0);
798 }
799
800 double multiply(double x1, double y1, double x2, double y2, double x3,
801                double y3)
802 /*
803    cross product of lines p1-p2 and p1-p3
804 */
805 {
806     return (x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1);
807 }
808
809 int pointInSegment(double xp, double yp, double x1, double y1, double
810                    x2, double y2)
811 {
812     return (equal(multiply(xp, yp, x1, y1, x2, y2), 0) &&
813             greaterEqual(xp, min(x1, x2)) && lessEqual(xp, max(x1, x2)) &&
814             greaterEqual(yp, min(y1, y2)) && lessEqual(yp, max(y1, y2)));
815 }
816
817 void rotate(double &x, double &y, double angle)
818 /*
819    rotate (x, y) around (0, 0)
820    angle should be in radians
821 */
822 {
823     double tx = x * cos(angle) - y * sin(angle);
824     double ty = x * sin(angle) + y * cos(angle);
825     x = tx;
826     y = ty;
827 }
828
829 int circlesIntersect(double x1, double y1, double r1,
830                     double x2, double y2, double r2)
831 /*
832    return values:
833    0: no intersection
834    1: one intersection
835    2: two intersections
836    3: the circles are the same
837 */
838 {
839     if (equal(x1, x2) && equal(y1, y2) && equal(r1, r2))

```

```

838     return 3;
839     double d = distance(x1, y1, x2, y2);
840     if (greater(d, r1 + r2) || less(d, fabs(r1 - r2)))
841         return 0;
842     if (equal(d, r1 + r2))
843         return 1;
844     return 2;
845 }
846
847 double pointToLineDistance(double x, double y,
848                             double a, double b, double c)
849 {
850     return fabs(a * x + b * y + c) / sqrt(a * a + b * b);
851 }
852
853 int circleLineIntersect(double x, double y, double r,
854                         double a, double b, double c)
855 /*
856    return values:
857    0: no intersection
858    1: one intersection
859    2: two intersections
860 */
861 {
862     double delta;
863     if (equal(b, 0))
864         delta = a * a * r * r - (a * x + c) * (a * x + c);
865     else
866         delta = b * b * ((a * a + b * b) * r * r - (a * x + b * y + c)
867                          * (a * x + b * y + c));
868     return (less(delta, 0) ? 0 : (equal(delta, 0) ? 1 : 2));
869 }
870
871 int circleLineIntersectionPoints(double x, double y, double r,
872                                  double a, double b, double c,
873                                  double &x1, double &y1, double &x2,
874                                  double &y2)
875 /*
876    return values:
877    0: no intersection
878    1: one intersection at (x1, y1)
879    2: two intersections at (x1, y1) and (x2, y2)
880 */
881 {
882     double delta;
883     if (equal(b, 0))
884         delta = a * a * r * r - (a * x + c) * (a * x + c);
885     else
886         delta = b * b * ((a * a + b * b) * r * r - (a * x + b * y + c)
887                          * (a * x + b * y + c));
888     if (less(delta, 0))
889         return 0;
890     if (equal(delta, 0))
891     {
892         x1 = (-a * (a * x + b * y + c)) / (a * a + b * b);
893         y1 = (-b * (a * x + b * y + c)) / (a * a + b * b);
894         x1 += x;

```

```

892     y1 += y;
893     return 1;
894 }
895 if (equal(b, 0))
896 {
897     y1 = sqrt(delta) / a;
898     y2 = -sqrt(delta) / a;
899     x1 = x2 = -(a * x + c) / a;
900 } else
901 {
902     x1 = (-a * (a * x + b * y + c) + sqrt(delta)) / (a*a+b*b);
903     y1 = -(a * (x1 + x) + b * y + c) / b;
904     x2 = (-a * (a * x + b * y + c) - sqrt(delta)) / (a*a+b*b);
905     y2 = -(a * (x2 + x) + b * y + c) / b;
906 }
907 x1 += x; y1 += y;
908 x2 += x; y2 += y;
909 return 2;
910 }
911
912 int circlesIntersectionPoints(double x1,double y1,double r1,double
x2,double y2,double r2,
913     double &xp1,double &yp1,double &xp2,double &yp2)
914 {
915     double a=atan2(y2 - y1, x2 - x1);
916     double s=hypot(x1 - x2, y1 - y2);
917     if(s > r1 + r2 + eps)return 0;
918     double b=acos((r1 * r1 + s * s - r2 * r2) / (2 * r1 * s));
919     xp1=cos(a+b)*r1+x1;
920     xp2=cos(a-b)*r1+x1;
921     yp1=sin(a+b)*r1+y1;
922     yp2=sin(a-b)*r1+y1;
923     if(abs(s-r1-r2)<eps || abs(s-abs(r1-r2))<eps)
924         return 1;
925     if(s<abs(r1-r2)-eps)return 0;
926     return 2;
927 }
928
929
930 double polygonArea(int n, double x[], double y[])
931 /*
932     return value:
933         positive: points are ordered counter-clockwise
934         negative: // // // clockwise
935 */
936 {
937     double a = 0;
938     for (int i = 1; i <= n; i++)
939         a += (x[i - 1] * y[i % n] - x[i % n] * y[i - 1]);
940     return a / 2;
941 }
942
943 void convexHullGW(int n, double x[], double y[], vector<int> &ch)
944 /*
945     Convex Hull finding, Gift-Wrapping algorithm, O(n^2)
946 */
947 {

```

```

948 /*
949     note the array size
950 */
951     bool mark[1000];
952     int i, p0, p1, p2;
953 /*
954     find the first point, which is the lowest, leftmost point.
955 */
956     p0 = 0;
957     for (i = 0; i < n; i++)
958         if(less(y[i],y[p0]) || equal(y[i],y[p0])&&less(x[i],x[p0]))
959             p0 = i;
960     p1 = p0;
961     fill(mark, mark + n, false);
962     ch = vector<int>();
963     while (true)
964     {
965         mark[p1] = true;
966         ch.push_back(p1);
967         p2 = -1;
968         for (i = 0; i < n; i++)
969             if (!mark[i] &&
970                 (p2 == -1 || less(multiply(x[p1], y[p1], x[p2], y[p2],
x[i], y[i]), 0) ||
971 /*
972         'less' : counter-clockwise
973         'greater' : clockwise
974 */
975         equal(multiply(x[p1], y[p1], x[p2], y[p2], x[i], y[i]), 0)&&
greaterEqual(x[i], min(x[p1], x[p2])) &&
976         lessEqual(x[i], max(x[p1], x[p2])) &&
greaterEqual(y[i], min(y[p1], y[p2])) &&
977         lessEqual(y[i], max(y[p1], y[p2]))))
978 /*
979         maximum number of points : i, p1, p2, i, p1, p2
980         minimum number of points : p2, p1, i, p2, p1, i
981 */
982         p2 = i;
983         if (p2 == -1 || less(multiply(x[p0], y[p0], x[p1], y[p1],
x[p2], y[p2]), 0)
984 /*
985         'less' : counter-clockwise
986         'greater' : clockwise
987 */
988         /*|| (equal(multiply(x[p0], y[p0], x[p1], y[p1], x[p2],
y[p2]), 0) &&
989         greaterEqual(x[p2], min(x[p1], x[p0])) &&
lessEqual(x[p2], max(x[p1], x[p0])) &&
990         greaterEqual(y[p2], min(y[p1], y[p0])) &&
lessEqual(y[p2], max(y[p1], y[p0]))))*/)
991 /*
992         maximum number of points : exclude the above 3 lines
993         minimum number of points : include the above 3 lines
994 */
995         break;
996         p1 = p2;
997     }

```

```

998 }
999
1000 struct CompCHGS
1001 {
1002     int m;
1003     double *x, *y;
1004     CompCHGS(int m, double *x, double *y): m(m), x(x), y(y) {}
1005     bool operator()(const int i1, const int i2) const
1006     {
1007         return (greater((x[i1] - x[m]) * (y[i2] - y[m]), (x[i2] - x[m])
1008             * (y[i1] - y[m])) ||
1009             'greater' : counter-clockwise
1010             'less'    : clockwise
1011         */
1012         (equal((x[i1] - x[m]) * (y[i2] - y[m]), (x[i2] - x[m]) *
1013             (y[i1] - y[m])) &&
1014             greaterEqual(x[i1], min(x[i2], x[m])) && lessEqual(x[i1],
1015             max(x[i2], x[m])) &&
1016             greaterEqual(y[i1], min(y[i2], y[m])) && lessEqual(y[i1],
1017             max(y[i2], y[m]))));
1018     }
1019 };
1020
1021 void convexHullGS(int n, double x[], double y[], vector<int> &ch)
1022 /*
1023     Convex Hull finding, Graham Scan algorithm, O(n*log(n))
1024 */
1025 {
1026     note the array size
1027     int sorted[1000], i, m = 0;
1028     for (i = 0; i < n; i++)
1029     {
1030         sorted[i] = i;
1031         if (less(y[i], y[m]) || (equal(y[i], y[m]) && less(x[i],
1032             x[m])))
1033             m = i;
1034     }
1035     sorted[0] = m;
1036     sorted[m] = 0;
1037
1038     /*
1039         the angles 1 to n - 1 should be sorted in ascending order. for each
1040         two equal angles,
1041         the angle with endpoint nearer to origin should become first.
1042     */
1043     sort(sorted + 1, sorted + n, CompCHGS(m, x, y));
1044
1045     /*
1046         after the sort, order of the last equal angles should be reversed.
1047     */
1048     for (i = n - 1; i > 1 &&
1049         equal((x[sorted[i]] - x[m]) * (y[sorted[i - 1]] - y[m]),
1050             (x[sorted[i - 1]] - x[m]) * (y[sorted[i]] - y[m])); i--);
1051     if (i > 1) reverse(sorted + i, sorted + n);
1052
1053     ch = vector<int>();

```

```

1049     ch.push_back(sorted[0]);
1050     if (n > 1)
1051     {
1052         ch.push_back(sorted[1]);
1053         for (i = 2; i < n; i++)
1054         {
1055             while (ch.size() > 1 &&
1056                 greaterEqual(multiply(x[sorted[i]], y[sorted[i]],
1057                     x[ch.back()], y[ch.back()],
1058                     x[ch[ch.size() - 2]], y[ch[ch.size() - 2]]), 0))
1059             /*
1060                 'greater'      : counter-clockwise, maximum number of points
1061                 'greaterEqual' : counter-clockwise, minimum number of points
1062                 'less'         : clockwise, maximum number of points
1063                 'lessEqual'    : clockwise, minimum number of points
1064             */
1065             ch.pop_back();
1066             ch.push_back(sorted[i]);
1067         }
1068         /*
1069             maximum number of points : exclude the following 'if' statement
1070             minimum number of points : include the following 'if' statement
1071         */
1072         if (ch.size() > 2 && equal(multiply(x[ch.front()],
1073             y[ch.front()], x[ch.back()],
1074             y[ch.back()], x[ch[ch.size() - 2]], y[ch[ch.size() - 2]]), 0))
1075             ch.pop_back();
1076     }
1077
1078 int polygonOrientation(int n, double x[], double y[])
1079 /*
1080     result values:
1081     1 : counter-clockwise
1082     -1 : clockwise
1083 */
1084 {
1085     int m = 0;
1086     for (int i = 1; i < n; i++)
1087         if (less(y[i], y[m]) || (equal(y[i], y[m]) && less(x[i], x[m])))
1088             m = i;
1089     return sign(multiply(x[m], y[m], x[(m + n - 1) % n], y[(m + n - 1)
1090         % n], x[(m + 1) % n], y[(m + 1) % n]));
1091 }
1092
1093 int generalPolygonInclusion(int n, double x[], double y[], double xp,
1094     double yp)
1095 /*
1096     return values:
1097     0: (xp, yp) is outside the polygon
1098     1: (xp, yp) is inside the polygon
1099     2: (xp, yp) is on the polygon side
1100 */
1101 {
1102     int i;
1103     for (i = 1; i <= n; i++)
1104         if (pointInSegment(xp, yp, x[i - 1], y[i - 1], x[i % n], y[i %

```

```

n)))
1102     return 2;
1103 int r = 0;
1104 for (i = 1; i <= n; i++)
1105     if ((lessEqual(y[i - 1], yp) && greater(y[i % n], yp)
1106         || lessEqual(y[i % n], yp) && greater(y[i - 1], yp)) &&
1107         less(xp, ((x[i % n] - x[i - 1]) * (yp - y[i - 1]) /
1108             (y[i % n] - y[i - 1]) + x[i - 1])))
1109         r = !r;
1110 return r;
1111 }
1112
1113 bool angleInclusion(double xo, double yo, double xa, double ya, double
xb, double yb,
1114     double xp, double yp)
1115 /*
1116     checks to see if we encounter point 'p' while sweeping angle
1117     'a-o-b' from 'a' to 'b'
1118 */
1119 {
1120     double ab = multiply(xo, yo, xa, ya, xb, yb);
1121     double ap = multiply(xo, yo, xa, ya, xp, yp);
1122     double bp = multiply(xo, yo, xb, yb, xp, yp);
1123     return (greaterEqual(ab, 0) && greaterEqual(ap, 0) && lessEqual(bp,
0)) ||
1124         (less(ab, 0) && (greaterEqual(ap, 0) || lessEqual(bp, 0)));
1125 }
1126
1127 int convexPolygonInclusion(int n, double x[], double y[], double xp,
double yp)
1128 /*
1129     the polygon should have at least three points which are not on the
1130     same line.
1131 */
1132 /*
1133     return values:
1134     0: (xp, yp) is outside the polygon
1135     1: (xp, yp) is inside the polygon
1136     2: (xp, yp) is on the polygon side
1137 */
1138 {
1139     double xm, ym;
1140     int i;
1141     /*
1142     the points are checked to be in counter-clockwise order
1143     */
1144     if (polygonOrientation(n, x, y) > 0)
1145     {
1146         reverse(x, x + n);
1147         reverse(y, y + n);
1148     }
1149     /*
1150     find an internal point
1151     */
1152     for (i = 2; i < n; i++)
1153         if (!equal(multiply(x[0], y[0], x[1], y[1], x[i], y[i]), 0))

```

```

1153     {
1154         xm = (x[0] + x[1] + x[i]) / 3;
1155         ym = (y[0] + y[1] + y[i]) / 3;
1156         break;
1157     }
1158 /*
1159     NOTE: as you see, this algorithm is of O(log(n)) if the above
1160     O(n) computations are done before!
1161 */
1162 int f = 0, l = n;
1163 while (f < l - 1)
1164 {
1165     int m = (f + l) / 2;
1166     if (angleInclusion(xm, ym, x[f], y[f], x[m], y[m], xp, yp))
1167         l = m;
1168     else
1169         f = m;
1170 }
1171 double r = multiply(x[f], y[f], x[l % n], y[l % n], xp, yp);
1172 return (equal(r, 0) ? 2 : greater(r, 0));
1173 }
1174
1175 int perpendicularProjection(double xp, double yp,
1176     double x1, double y1, double x2, double y2,
1177     double &x, double &y)
1178 /*
1179     computes the point (x, y) of perpendicular projection of point
1180     (xp, yp) onto segment (x1, y1)-(x2, y2)
1181 */
1182 /*
1183     results:
1184     -1 : the point is on backward extension of the line
1185     0 : the point is in the line including endpoints
1186     1 : the point is on ahead extension of the line
1187 */
1188 {
1189     double l = squaredDistance(x1, y1, x2, y2);
1190     double r = ((y1 - yp) * (y1 - y2) - (x1 - xp) * (x2 - x1)) / l;
1191     x = x1 + r * (x2 - x1);
1192     y = y1 + r * (y2 - y1);
1193     return (less(r, 0) ? -1 : greater(r, 1));
1194 }
1195
1196 bool circleThroughPoints(double x1, double y1, double x2, double y2
, double x3, double y3, double &x, double &y, double &r)
1197 /*
1198     result values:
1199     false : the points are on a same extension
1200     true : circle is (x, y, r)
1201 */
1202 {
1203     double d = 2*(y1*x3+y2*x1-y2*x3-y1*x2-y3*x1+y3*x2);
1204     if (equal(d, 0))
1205         return false;
1206     x = (y2 * x1 * x1 - y3 * x1 * x1 - y2 * y2 * y1 + y3 * y3 * y1 +
1207         x2 * x2 * y3 + y1 * y1 * y2 + x3 * x3 * y1 - y3 * y3 * y2 -
1208         x3 * x3 * y2 - x2 * x2 * y1 + y2 * y2 * y3 - y1 * y1 * y3)/d;

```

```

1210     y = (x1 * x1 * x3 + y1 * y1 * x3 + x2 * x2 * x1 - x2 * x2 * x3 +
1211           y2 * y2 * x1 - y2 * y2 * x3 - x1 * x1 * x2 - y1 * y1 * x2 -
1212           x3 * x3 * x1 + x3 * x3 * x2 - y3 * y3 * x1 + y3 * y3 * x2)/d;
1213     r = sqrt((x1 - x) * (x1 - x) + (y1 - y) * (y1 - y));
1214     return true;
1215 }
1216
1217 double cross(double ax,double ay,double bx,double by)
1218 {
1219     return ax*by-ay*bx;
1220 }
1221 double dot(double ax,double ay,double bx,double by)
1222 {
1223     return ax*bx+ay*by;
1224 }
1225
1226 double ang(double ax,double ay,double bx,double by)
1227 {
1228     return atan2(cross(ax,ay,bx,by),dot(ax,ay,bx,by));
1229 }
1230 void rotate(double &px,double &py,double t)
1231 {
1232     double x,y;
1233     x=px;
1234     y=py;
1235     px=x*cos(t)-y*sin(t);
1236     py=x*sin(t)+y*cos(t);
1237 }
1238 void rotate(double &px,double &py,double &pz,double vx,double vy,double
1239            vz,double t)
1240 {
1241     double ty,tx;
1242     ty=ang(vz,vx,1,0);
1243     rotate(pz,px,ty);
1244     rotate(vz,vx,ty);
1245     tx=ang(vy,vz,0,1);
1246     rotate(py,pz,tx);
1247     rotate(px,py,t);
1248     rotate(py,pz,-tx);
1249     rotate(pz,px,-ty);
1250 }
1251 =====
1252 GRID
1253 =====
1254 #include<iostream>
1255
1256 using namespace std;
1257
1258 int m,n;
1259 int dir[4][2]={0,-1,-1,0,0,1,1,0};
1260 int edge[2][200][200];
1261 int mark[200][200];
1262 int cc,len,best;
1263
1264 int g(int a,int b,int k)
1265 {

```

```

1266     return k<2?edge[1-k][a][b]:edge[1-(k%2)][a+(k%2)][b+1-(k%2)];
1267 }
1268
1269 void dfs(int a,int b)
1270 {
1271     if(a<0||b<0||a>=m+n+4||b>=m+n+4||mark[a][b])return;
1272     mark[a][b]=1;
1273     len++;
1274     for(int k=0;k<4;k++)
1275         if(!g(a,b,k))
1276             dfs(a+dir[k][0],b+dir[k][1]);
1277 }
1278
1279 main()
1280 {
1281     for(int N=0;cin>>n>>m,m||n;N++)
1282     {
1283         memset(mark,0,sizeof mark);
1284         memset(edge,0,sizeof edge);
1285         for(int i=0;i<m;i++)
1286             for(int j=0;j<n;j++)
1287             {
1288                 char c;
1289                 cin>>c;
1290
1291                 edge[c=='\\'[1+i+j+1][m-i+j+1]=edge[c=='\\'[1
1292                    +i+j-(c=='\\')+1][m-i+j-(c=='/')+1]=1;
1293             }
1294             cc=best=0;
1295             dfs(0,0);
1296             for(int i=0;i<m+n+4;i++)
1297                 for(int j=0;j<m+n+4;j++)
1298                     if(!mark[i][j])
1299                     {
1300                         len=0;
1301                         cc++;
1302                         dfs(i,j);
1303                         best>=len;
1304                     }
1305             cout<<"Maze #"<<N+1<<": "<<endl;
1306             if(cc)
1307                 cout<<cc<<" Cycles; the longest has length
1308                    "<<best<<". "<<endl;
1309             else
1310                 cout<<"There are no cycles."<<endl;
1311             cout<<endl;
1312         }
1313     }
1314
1315     =====
1316     #include<iostream>
1317     #include<algorithm>
1318     #include<string>
1319     using namespace std;
1320     #define MAXLEN 10000
1321     struct BigNum{

```

```

1320     short n[MAXLEN];
1321     int len;
1322 } temp,one,zero,ans,res,up,xx;
1323 void print(BigNum &a)
1324 {
1325     for(int i=a.len-1;i+1;i--)cout<<a.n[i];
1326     cout<<endl;
1327 }
1328 void assignInt(BigNum &a,int n)
1329 {
1330     int i=0;
1331     do {a.n[i++]=n%10;n/=10;}while(n);
1332     a.n[i]=-1;
1333     a.len=i;
1334 }
1335 void assignStr(BigNum &a,string n)
1336 {
1337     int i=0,len=n.length();
1338     for(i=len-1;i+1;i--)
1339         a.n[len-1-i]=n[i]-'0';
1340     a.n[len]=-1;
1341     a.len=len;
1342 }
1343 void copyNum(BigNum &des,BigNum &src)
1344 {
1345     int i;
1346     for(i=0;src.n[i]+1;i++)des.n[i]=src.n[i];
1347     des.n[i]=-1;
1348     des.len=i;
1349 }
1350 int numcmp(BigNum &a,BigNum &b)
1351 {
1352     int l1=a.len,l2=b.len;
1353     if(l1!=l2)return l1>l2?-1:1;
1354     for(int i=l1-1;i+1;i--)
1355         if(a.n[i]!=b.n[i])return a.n[i]<b.n[i]?-1:1;
1356     return 0;
1357 }
1358 void add(BigNum &a,BigNum &b,BigNum &c)//c=a+b
1359 {
1360     int i,t=0;
1361     for(i=0;a.n[i]>=0&&b.n[i]>=0;i++)
1362         c.n[i]=(t+(a.n[i]+b.n[i]+t))%10,t/=10;
1363     for(short *p=(a.n[i]==-1?&b.n[i]:&a.n[i]);*p!=-1;p++,i++)
1364         c.n[i]=(t+(*p+t))%10,t/=10;
1365     if(t)c.n[i++]=t;
1366     c.n[i]=-1;
1367     c.len=i;
1368 }
1369 void mul(BigNum &a,BigNum &b,BigNum &c)//c=a*b
1370 {
1371     int t=0,tt,i,j;
1372     memset(c.n,0,sizeof(short)*MAXLEN);
1373     for(i=0;a.n[i]+1;i++)
1374     {
1375         for(j=0;b.n[j]+1;j++)
1376             tt=c.n[i+j]+a.n[i]*b.n[j]+t,c.n[i+j]=tt%10,t=tt/10;

```

```

1377         for(;t;j++)
1378             tt=c.n[i+j]+t,c.n[i+j]=tt%10,t=tt/10;
1379     }
1380     for(i+=j;!c.n[i]&&i;i--);
1381     c.n[i+1]=-1;
1382     c.len=i+1;
1383 }
1384 void div2(BigNum &a,BigNum &b)
1385 {
1386     int i=a.len,t,j;
1387     b.n[t=i]=-1;
1388     b.len=i;
1389     for(i--,j=i+1;i--,j--)
1390         if((a.n[i]&1)&&i)
1391         {
1392             b.n[j]=a.n[i]/2;
1393             a.n[i-1]+=10;
1394         }else
1395             b.n[j]=a.n[i]/2;
1396     for(i=t-1;!b.n[i]&&i;b.len=i,b.n[i--]=-1);
1397 }
1398 void sub(BigNum &a,BigNum &b,BigNum &c)//c=a-b && should a>=b,a maybe
    changed
1399 {
1400     int i,j;
1401     for(i=0;a.n[i]+1&&b.n[i]+1;i++)
1402         if(a.n[i]-b.n[i]>=0)c.n[i]=a.n[i]-b.n[i];
1403     else
1404     {
1405         c.n[i]=10-b.n[i]+a.n[i];
1406         for(j=i+1;!a.n[j];j++)a.n[j]=9;
1407         a.n[j]--;
1408     }
1409     for(;a.n[i]+1;i++)c.n[i]=a.n[i];
1410     c.len=i;
1411     c.n[i--]=-1;
1412     while(i+1&&!c.n[i])c.n[i--]=-1,c.len--;
1413     if(i==0)c.n[0]=0,c.len=1;
1414 }
1415 void removeZero(BigNum &n)//for use in div
1416 {
1417     int i=n.len-1;
1418     while(!n.n[i]&&i)n.len--,n.n[i--]=-1;
1419 }
1420 void div(BigNum &a,BigNum &b,BigNum &r,BigNum &q)// a=b*Q+R
1421 {
1422     BigNum sum[l1],temp,templ;
1423     int cmp=numcmp(a,b);
1424     if(!cmp)
1425     {
1426         assignInt(r,0);
1427         assignInt(q,1);
1428         return;
1429     }else if(cmp==1)
1430     {
1431         copyNum(r,a);
1432         assignInt(q,0);

```

```

1433     return;
1434 }
1435 copyNum(sum[0],b);
1436 for(int i=1;i<11;i++)
1437     add(sum[i-1],b,sum[i]);
1438 int index=0,k;
1439 int first=b.len;
1440 copy(a.n+a.len-first,a.n+a.len,temp.n);
1441 temp.n[temp.len=first]=-1;
1442 if(numcmp(temp,b)<0)
1443     first++;
1444 copy(a.n+a.len-first,a.n+a.len,temp.n);
1445 temp.n[temp.len=first]=-1;
1446 int firstLen=a.len-first;
1447 while(1)
1448 {
1449     for(k=0;k<11;k++)
1450         if(numcmp(temp,sum[k])<0)
1451             break;
1452     q.n[index++]=k;
1453     sub(temp,sum[k-1],temp1);
1454     if(firstLen<1)break;
1455     for(int i=temp1.len+1;i;i--)
1456         temp1.n[i]=temp1.n[i-1];
1457     temp1.n[0]=a.n[--firstLen];
1458     temp1.len++;
1459     removeZero(temp1);
1460     while(numcmp(temp1,b)<0)
1461     {
1462         q.n[index++]=0;
1463         if(firstLen<1)break;
1464         for(int i=temp1.len+1;i;i--)
1465             temp1.n[i]=temp1.n[i-1];
1466         temp1.n[0]=a.n[--firstLen];
1467         temp1.len++;
1468         removeZero(temp1);
1469     }
1470     if(numcmp(temp1,b)<0)break;
1471     copyNum(temp,temp1);
1472 }
1473 copyNum(r,temp1);
1474 q.n[q.len=index]=-1;
1475 reverse(q.n,q.n+index);
1476 }
1477 =====
1478 LINE POLYGON INTERSECTION
1479 =====
1480 #include<iostream>
1481 #include<algorithm>
1482 #include<cstdio>
1483 using namespace std;
1484
1485 int n;
1486 double list[200][2]={0,0,10,0,10,10,0,10};
1487 int mark[200];
1488 double a[2],b[2],m[2],c,coef;
1489

```

```

1490 void cross(double aa[],double p1[],double p2[])
1491 {
1492     double ap,bp,cp;
1493     bp=p1[0]-p2[0];
1494     ap=-(p1[1]-p2[1]);
1495     cp=-bp*p1[1]-ap*p1[0];
1496
1497     double det=-(m[0]*bp-m[1]*ap);
1498     aa[0]=(c*bp-cp*m[1])/det;
1499     aa[1]=(m[0]*cp-c*ap)/det;
1500 }
1501
1502 double area()
1503 {
1504     double ans=0;
1505     for (int i=1;i<n;i++)
1506         ans+=(list[i-1][0]*list[i][1]-list[i][0]*list[i-1][1]);
1507     ans+=(list[n-1][0]*list[0][1]-list[0][0]*list[n-1][1]);
1508     return ans/2;
1509 }
1510
1511 int main()
1512 {
1513     n=4;
1514     b[0]=b[1]=0;
1515     int ff=0;
1516     while(cin>>a[0]>>a[1])
1517     {
1518         char txt[1024];
1519         cin>>txt;
1520         if(!strcmp(txt,"Same")^(a[0]==b[0]&&a[1]==b[1]))
1521             ff=1;
1522         if(ff)
1523         {
1524             cout<<"0.00"<<endl;
1525             continue;
1526         }
1527         if(!strcmp(txt,"Same"))
1528         {
1529             sprintf(txt,"%2lf",area());
1530             cout<<txt<<endl;
1531             continue;
1532         }
1533         m[0]=a[0]-b[0];
1534         m[1]=a[1]-b[1];
1535         c=-(m[0]*(a[0]+b[0])/2+m[1]*(a[1]+b[1])/2);
1536         int mod=(a[0]*m[0]+a[1]*m[1]+c>=0);
1537         coef=(mod^(strcmp(txt,"Hotter")!=0)?1:-1;
1538         int j=-1;
1539         for(int i=0;i<n;i++)
1540             if(!(mark[i]=(coef*(list[i][0]*m[0]+list[i][1]*m[1]+c)>=0)))
1541                 j=i;
1542         if(j+1)
1543         {
1544             int first,last;
1545             for(first=j;!mark[first];first=(first-1+n)%n);
1546             for(last=j;!mark[last];last=(last+1)%n);

```



```

1547     double aa[2],bb[2];
1548     cross(aa,list[first],list[(first+1)%n]);
1549     cross(bb,list[(last-1+n)%n],list[last]);
1550     double _list[200][2];
1551     int _n=n;
1552     memcpy(_list,list,sizeof(list));
1553     n=0;
1554     for(int i=last;i!=(first+1)%_n;i=(i+1)%_n)
1555     {
1556         list[n][0]=_list[i][0];
1557         list[n++][1]=_list[i][1];
1558     }
1559     list[n][0]=aa[0];list[n++][1]=aa[1];
1560     list[n][0]=bb[0];list[n++][1]=bb[1];
1561 }
1562 sprintf(txt,"%0.2lf",area());
1563 cout<<txt<<endl;
1564 b[0]=a[0];b[1]=a[1];
1565 }
1566 return 0;
1567 }
1568 =====
1569 MATCH HAAMED
1570 =====
1571 // Bipartite Matching - Haamed
1572
1573 #include<iostream.h>
1574 #include<cstring>
1575
1576 #define MAXNODE 1000
1577
1578 int m,n;
1579 int graph[MAXNODE][MAXNODE];
1580 int mark[MAXNODE],match[MAXNODE];
1581
1582 int dfs(int v)
1583 {
1584     if(v==-1)return 1;
1585     mark[v]=1;
1586     for(int i=0;i<n;i++)
1587         if(graph[v][i]&&(match[i]==-1||!mark[match[i]]&&dfs(match[i])))
1588         {
1589             match[i]=v;
1590             return 1;
1591         }
1592     return 0;
1593 }
1594
1595 int main()
1596 {
1597     //init
1598     memset(mark,0,sizeof mark);
1599     memset(match,-1,sizeof match);
1600     //input
1601     cin>>m>>n;
1602     for(int i=0;i<m;i++)
1603         for(int j=0;j<n;j++)

```

```

1604         cin>>graph[i][j];
1605     //body
1606     for(int i=0;i<m;i++)
1607         if(!mark[i]&&dfs(i))
1608             memset(mark,0,sizeof mark);
1609     //output
1610     for(int j=0;j<n;j++)
1611         if(match[j]!=-1)
1612             cout<<match[j]<<" "<<j<<endl;
1613     return 0;
1614 }
1615 =====
1616 MATRIX INVERSION
1617 =====
1618 #include <iostream>
1619 #include <string>
1620 #include <vector>
1621 #include <cmath>
1622
1623 using namespace std;
1624 #define double int
1625
1626 typedef vector<double> Row;
1627 typedef vector<Row> Matrix;
1628 int p;
1629
1630 void swapRows(Matrix &matrix, int m, int n)
1631 {
1632     Row tmp = matrix[m];
1633     matrix[m] = matrix[n];
1634     matrix[n] = tmp;
1635 }
1636
1637 void addRow(Matrix &matrix, double coef, int m, int n)
1638 {
1639     for (unsigned i = 0; i < matrix[m].size();i++)
1640     {
1641         matrix[n][i] += coef * matrix[m][i];
1642         matrix[n][i]%=p;
1643     }
1644 }
1645
1646 void multiplyRow(Matrix &matrix, int n, double coef)
1647 {
1648     for (unsigned i = 0; i < matrix[n].size();i++)
1649     {
1650         matrix[n][i] *= coef;
1651         matrix[n][i]%=p;
1652     }
1653 }
1654
1655 int invt[30001];
1656 int inv(int a)
1657 {
1658     if(invt[a%p])return invt[a%p];
1659     for(int i=1;i<p;i++)
1660         if((a*i)%p==1)return invt[a%p]=i;

```

```

1661 }
1662
1663 Matrix inverse(Matrix matrix)
1664 {
1665     Matrix result(matrix.size(), Row(matrix.size()));
1666     unsigned i, j, k;
1667
1668     for (i = 0; i < matrix.size(); i++)
1669         result[i][i] = 1;
1670
1671     for (i = 0; i < matrix.size(); i++)
1672     {
1673         for (k = i; k < matrix.size() && !matrix[k][i]; k++);
1674         if (k == matrix.size())
1675             cout<<string("Given matrix is not invertible");
1676         swapRows(matrix, i, k);
1677         swapRows(result, i, k);
1678         for (j = 0; j < matrix.size(); j++)
1679         {
1680             if (j == i)
1681                 continue;
1682             double coef = ((-matrix[j][i] *
1683                             inv(matrix[i][i]))%p+p)%p;
1684             addRow(matrix, coef, i, j);
1685             addRow(result, coef, i, j);
1686         }
1687         double revCoef = inv(matrix[i][i]);
1688         multiplyRow(matrix, i, revCoef);
1689         multiplyRow(result, i, revCoef);
1690     }
1691     return result;
1692 }
1693 int pp(int a, int n)
1694 {
1695     if(!n) return 1;
1696     int t=pp(a, n/2);
1697     t=(t*t)%p;
1698     if(n&1)
1699         t=(t*a)%p;
1700     return t;
1701 }
1702 int main()
1703 {
1704     int N;
1705     char str[100];
1706     for(cin>>N; N--;)
1707     {
1708         cin>>p>>str;
1709         memset(invt, 0, sizeof invt);
1710         int n=strlen(str);
1711         vector<Row>mat(n);
1712         for(int i=0; i<n; i++)
1713             for(int j=0; j<n; j++)
1714                 mat[i].push_back(pp(i+1, j));
1715         mat=inverse(mat);
1716         for(int i=0; i<n; i++)

```

```

1717     {
1718         int ans=0;
1719         for(int j=0; j<n; j++)
1720             ans+=(str[j]!='?'?0:str[j]-'a'+1)*mat[i][j];
1721         ans=((p+ans)%p+p)%p;
1722         cout<<ans<<' ';
1723     }
1724     cout<<endl;
1725 }
1726 return 0;
1727 }
1728 =====
1729 MAXFLOW
1730 =====
1731 #include <iostream>
1732 #include <cstring>
1733 #include <list>
1734 #include <vector>
1735
1736 using namespace std;
1737
1738 const int NODES = 300;
1739
1740 int relabelToFront(int n, int c[NODES][NODES], int s, int t, int
1741 f[NODES][NODES])
1742 {
1743     int h[NODES], e[NODES];
1744
1745     // initialize preflow
1746     memset(h, 0, n * sizeof(int));
1747     memset(e, 0, n * sizeof(int));
1748     memset(f, 0, NODES * NODES * sizeof(int));
1749     h[s] = n;
1750     int value = 0;
1751     for (int i = 0; i < n; i++)
1752         if (c[s][i])
1753         {
1754             f[s][i] = c[s][i];
1755             f[i][s] = -c[s][i];
1756             e[i] = c[s][i];
1757             value += c[s][i];
1758         }
1759
1760     // constructing list of vertices
1761     list<int> ver;
1762     for (int i = 0; i < n; i++)
1763         if (i != s && i != t)
1764             ver.push_back(i);
1765
1766     // constructing list of neighbors
1767     vector<int> neigh[NODES];
1768     for (int i = 0; i < n; i++)
1769         for (int j = 0; j < n; j++)
1770             if (c[i][j] || c[j][i])
1771                 neigh[i].push_back(j);
1772
1773     vector<int>::iterator cur[NODES];

```

```

1773     for (int i = 0; i < n; i++)
1774         cur[i] = neigh[i].begin();
1775
1776     // body
1777     list<int>::iterator u = ver.begin();
1778     while (u != ver.end())
1779     {
1780         int uu = *u;
1781         int oldH = h[uu];
1782
1783         // discharge(u)
1784         while (e[uu] > 0)
1785         {
1786             if (cur[uu] == neigh[uu].end())
1787             {
1788                 // relabel(u)
1789                 int minh = 2 * n;
1790                 for (int i = 0; i < n; i++)
1791                     if (c[uu][i] - f[uu][i] > 0 && h[i] < minh)
1792                         minh = h[i];
1793                 h[uu] = minh + 1;
1794
1795                 cur[uu] = neigh[uu].begin();
1796             } else
1797             {
1798                 int vv = *cur[uu];
1799                 if (c[uu][vv] - f[uu][vv] > 0 && h[uu] == h[vv] + 1)
1800                 {
1801                     // push(u)
1802                     int d = min(e[uu], c[uu][vv] - f[uu][vv]);
1803                     f[uu][vv] += d;
1804                     f[vv][uu] = -f[uu][vv];
1805                     e[uu] -= d;
1806                     e[vv] += d;
1807                     if (vv == s)
1808                         value -= d;
1809                 }
1810                 else
1811                     cur[uu]++;
1812             }
1813         }
1814
1815         if (h[uu] > oldH)
1816             ver.splice(ver.begin(), ver, u);
1817         u++;
1818     }
1819
1820     return value;
1821 }
1822
1823 int main()
1824 {
1825     int n, c[NODES][NODES], f[NODES][NODES];
1826     while (cin >> n, n)
1827     {
1828         for (int i = 0; i < n; i++)
1829             for (int j = 0; j < n; j++)

```

```

1830         cin >> c[i][j];
1831     int s, t;
1832     cin >> s >> t;
1833     cout << "-> " << relabelToFront(n, c, s, t, f) << endl;
1834     /*for (int i = 0; i < n; i++)
1835     {
1836         for (int j = 0; j < n; j++)
1837             cout << f[i][j] << '/' << c[i][j] << ' ';
1838         cout << endl;
1839     }
1840     cout << endl;*/
1841 }
1842 return 0;
1843 }
1844
1845 MAXFLOW ARASH
1846
1847 /*-----*
1848 * Maximum Flow
1849 * Implemented Algorithms :
1850 *
1851 * - Ford Fulkerson Method (Edmonds-Karp Algorithm)
1852 *   Runs in  $O(V * E^2)$  time
1853 * - Generic Preflow-Push Algorithm
1854 *   Runs in  $O(E * V^2)$  time
1855 * - Maximum Bipartite Matching using Maximum-Flow
1856 *-----*/
1857
1858 #include <fstream>
1859 #include <vector>
1860 #include <queue>
1861 #include <cstring>
1862 #include <set>
1863 using namespace std;
1864
1865 #define NODE_NUM 100
1866
1867
1868 int capacity[NODE_NUM][NODE_NUM];
1869 int flow[NODE_NUM][NODE_NUM];
1870 int n;
1871
1872
1873 struct Path
1874 {
1875     int capacity;
1876     vector<int> path;
1877
1878     Path() : path(), capacity() {}
1879     Path(const Path &copy) : capacity(copy.capacity), path(copy.path) {}
1880 };
1881
1882
1883
1884 // This function finds and returns augmenting paths in the Residual
1885 // Network
1886 Path bfs(int src, int tgt)

```

```

1886 {
1887     bool mark[NODE_NUM];
1888     queue<Path> q;
1889
1890     memset(mark, false, sizeof(mark));
1891     while (!q.empty())
1892         q.pop();
1893
1894     Path first;
1895     first.capacity = 10000000;
1896     first.path.push_back(src);
1897     q.push(first);
1898
1899     while (!q.empty())
1900     {
1901         Path cur = q.front();
1902         q.pop();
1903         int last = *(cur.path.end() - 1);
1904         if (last == tgt)
1905             return cur;
1906
1907         for (int i = 0; i < n; i++)
1908             if (!mark[i] && (capacity[last][i] - flow[last][i] > 0))
1909             {
1910                 Path child;
1911                 child.capacity = min(cur.capacity, capacity[last][i] -
1912                                     flow[last][i]);
1913                 child.path = cur.path;
1914                 child.path.push_back(i);
1915                 mark[i] = true;
1916                 q.push(child);
1917             }
1918
1919     first.capacity = 0;
1920     return first;
1921 }
1922
1923 // Implementation of Edmonds-Karp Algorithm : uses bfs(src, tgt)
1924 int maxFlow(int src, int tgt)
1925 {
1926     memset(flow, 0, sizeof(flow));
1927
1928     Path p;
1929     int answer = 0;
1930     while (p = bfs(src, tgt), p.capacity)
1931     {
1932         for (vector<int>::iterator it = p.path.begin() + 1; it !=
1933             p.path.end(); it++)
1934         {
1935             flow[*it - 1][*it] += p.capacity;
1936             flow[*it][*it - 1] -= p.capacity;
1937         }
1938
1939         answer += p.capacity;
1940     }

```

```

1941
1942     return answer;
1943 }
1944
1945 // Implementation of Generic Preflow-Push Algorithm
1946 int maxFlowPP(int src, int tgt)
1947 {
1948     int h[NODE_NUM], e[NODE_NUM];
1949
1950     memset(flow, 0, sizeof(flow));
1951     memset(h, 0, n * sizeof(int));
1952     memset(e, 0, n * sizeof(int));
1953
1954     h[src] = n;
1955
1956     for (int i = 0; i < n; i++)
1957         if (capacity[src][i])
1958         {
1959             flow[src][i] = capacity[src][i];
1960             flow[i][src] = -capacity[src][i];
1961             e[i] = capacity[src][i];
1962         }
1963
1964     bool flag = true;
1965
1966     while (flag)
1967     {
1968         flag = false;
1969         for (int i = 0; i < n; i++)
1970             if (i != src && i != tgt && e[i])
1971             {
1972                 bool liftNeeded = false;
1973                 int minHeight = 2 * n;
1974                 for (int j = 0; j < n; j++)
1975                     if ((capacity[i][j] - flow[i][j]) > 0)
1976                     {
1977                         if (h[i] == h[j] + 1)
1978                         {
1979                             int d = min(e[i], capacity[i][j] -
1980                                     flow[i][j]);
1981                             flag = true;
1982                             flow[i][j] += d;
1983                             flow[j][i] = -flow[i][j];
1984                             e[i] -= d;
1985                             e[j] += d;
1986                         }
1987                         else if (h[i] <= h[j])
1988                         {
1989                             minHeight = min(minHeight, h[j]);
1990                             liftNeeded = true;
1991                         }
1992                     }
1993             }
1994
1995     if (e[i] && liftNeeded)

```

```

1997         {
1998             h[i] = 1 + minHeight;
1999             flag = true;
2000         }
2001     }
2002 }
2003
2004
2005     return e[tgt];
2006 }
2007
2008 // An Example : Maximum Bipartite Matching using Maximum Flow
2009 // Returns : Cardinality of Maximum Bipartite Matching
2010 int bipartiteMatch(set<int> first, set<int> second)
2011 {
2012     for (set<int>::iterator it=first.begin();it!=first.end();it++)
2013         capacity[n][*it] = 1;
2014
2015     for (set<int>::iterator it=second.begin();it!=second.end();it++)
2016         capacity[*it][n + 1] = 1;
2017
2018     n += 2;
2019
2020     return maxFlowPP(n - 2, n - 1);
2021 }
2022
2023
2024 int main()
2025 {
2026     int v;
2027     memset(capacity, 0, sizeof(capacity));
2028     cin >> n >> v;
2029
2030     for (int i = 0; i < v; i++)
2031     {
2032         int s, t;
2033         cin >> s >> t;
2034         capacity[s][t] = 1;
2035     }
2036
2037     int n1, n2;
2038     cin >> n1;
2039     set<int> first, second;
2040
2041     for (int i = 0; i < n1; i++)
2042     {
2043         int tmp;
2044         cin >> tmp;
2045         first.insert(tmp);
2046     }
2047
2048     cin >> n2;
2049
2050     for (int i = 0; i < n2; i++)
2051     {
2052         int tmp;
2053         cin >> tmp;

```

```

2054         second.insert(tmp);
2055     }
2056
2057     cout << endl << "Maximum Matching Cardinality : " <<
bipartiteMatch(first, second) << endl;
2058     cout << "Matching Graph : " << endl;
2059
2060     for (int i = 0; i < n - 2; i++)
2061     {
2062         for (int j = 0; j < n - 2; j++)
2063             cout << flow[i][j] << ' ';
2064         cout << endl;
2065     }
2066
2067     return 0;
2068 }
2069 =====
2070 MAXFLOW HAAMED
2071 =====
2072 // Maximum Flow - Haamed
2073
2074 #include<iostream.h>
2075 #include<cstring>
2076 #define MAXNODE 100
2077 #define INF 1000000000
2078 #define MIN(a,b) (a<?b)
2079
2080 int n,s,t;
2081 int cap[MAXNODE][MAXNODE];
2082 int flow[MAXNODE][MAXNODE];
2083 int mark[MAXNODE];
2084
2085 int dfs(int v,int val)
2086 {
2087     if(v==t)return val;
2088     mark[v]=1;
2089     int f;
2090     for(int i=0;i<n;i++)
2091
2092         if(!mark[i]&&flow[v][i]<cap[v][i]&&(f=dfs(i,MIN(val,cap[v][i]-f
low[v][i]))))
2093     {
2094         flow[v][i]+=f;
2095         flow[i][v]-=f;
2096         return f;
2097     }
2098     return 0;
2099 }
2100 int main()
2101 {
2102     //init
2103     memset(cap,0,sizeof cap);
2104     memset(flow,0,sizeof flow);
2105     memset(mark,0,sizeof mark);
2106     //input
2107     cin>>n>>s>>t;

```

```

2108     int m,a,b;
2109     for(cin>>m;m--;)
2110     {
2111         int c;
2112         cin>>a>>b>>c;
2113         cap[a][b]=c;
2114     }
2115     //body
2116     int f,sum=0;
2117     while(f=dfs(s,INF))
2118     {
2119         sum+=f;
2120         memset(mark,0,sizeof mark);
2121     }
2122     //output
2123     cout<<sum<<endl;
2124     for(int i=0;i<n;i++)
2125         if(mark[i])
2126             for(int j=0;j<n;j++)
2127                 if(!mark[j]&&cap[i][j])
2128                     cout<<i<<' '<<j<<' '<<cap[i][j]<<endl;
2129     return 0;
2130 }
2131 =====
2132 POLYGON SIMILARITY
2133 =====
2134 #include<iostream>
2135 #include<cmath>
2136 #include<string>
2137 #include<algorithm>
2138 #define EPS 1e-10
2139 #define N(i) ((i+1)%n)
2140 #define P(i) ((i-1+n)%n)
2141 #define p2(a) ((a)*(a))
2142 using namespace std;
2143 const double PI = 3.1415926535897932385;//2 * acos(0.)
2144 inline bool dless(double x, double y)
2145 {
2146     return x <= y - EPS;
2147 }
2148 inline double vectorAngle(double x, double y)
2149 {
2150     double r = atan2(y, x);
2151     if (dless(r, 0))
2152         r += 2 * PI;
2153     return r;
2154 }
2155
2156 inline double angle(double x1, double y1, double x2, double y2, double
x3, double y3)
2157 {
2158     double r=vectorAngle(x3-x2,y3-y2)-vectorAngle(x1-x2,y1-y2);
2159     if (dless(r, 0))
2160         r += 2 * PI;
2161     return r;
2162 }
2163 main()

```

```

2164 {
2165     int n;
2166     double xf[10],xs[10],yf[10],ys[10];
2167     while(cin>>n,n)
2168     {
2169         for(int i=0;i<n;i++)
2170             cin>>xf[i]>>yf[i];
2171         for(int i=0;i<n;i++)
2172             cin>>xs[i]>>ys[i];
2173         double max1=0,max2=0;
2174         double len1[20],len2[10];
2175         double an1[20],an2[10];
2176         for(int i=0;i<n;i++)
2177         {
2178             double
2179             b1=sqrt(p2(xf[(i+1)%n]-xf[i])+p2(yf[(i+1)%n]-yf[i]));
2180             double
2181             b2=sqrt(p2(xs[(i+1)%n]-xs[i])+p2(ys[(i+1)%n]-ys[i]));
2182             len1[i]=b1;
2183             len2[i]=b2;
2184             max1=max(max1,b1);
2185             max2=max(max2,b2);
2186
2187             an1[i]=angle(xf[P(i)],yf[P(i)],xf[i],yf[i],xf[N(i)],yf[N(i)]);
2188
2189             an2[i]=angle(xs[P(i)],ys[P(i)],xs[i],ys[i],xs[N(i)],ys[N(i)]);
2190
2191         }
2192         for(int i=0;i<n;i++)
2193         {
2194             len1[i]/=max1;
2195             len2[i]/=max2;
2196             len1[i+10]=len1[i];
2197             an1[i+10]=an1[i];
2198         }
2199         bool f=false;
2200         for(int i=0;i<n;i++)
2201         {
2202             bool f1=true;
2203             for(int j=0;j<n;j++)
2204                 if(abs(len1[i+j]-len2[j])>=EPS
2205                     ||abs(an1[i+j]-an2[j])>=EPS)
2206                 {
2207                     f1=false;
2208                     break;
2209                 }
2210             if (f1)
2211             {
2212                 f=true;
2213                 break;
2214             }
2215         }
2216         if (f)
2217             cout<<"similar"<<endl;
2218         else
2219             cout<<"dissimilar"<<endl;
2220     }

```

```

2215     }
2216     return 0;
2217 }
2218 =====
2219 STRONGLY CONNECTED
2220 =====
2221 // Strongly Connected Components
2222
2223 #include <iostream>
2224 #include <vector>
2225
2226 using namespace std;
2227
2228 #define NODES 100
2229
2230 // ----- StronglyConnected Matrix
2231
2232 int n, t, c, sorted[NODES], vc[NODES];
2233 int graph[NODES][NODES];
2234
2235 void dfsMatrix(int v, bool rev)
2236 {
2237     vc[v] = rev ? c : 1;
2238     for (int i = 0; i < n; i++)
2239         if (!rev && !vc[i] && graph[v][i] ||
2240             rev && vc[i] == -1 && graph[i][v])
2241             dfsMatrix(i, rev);
2242     if (!rev)
2243         sorted[n - ++t] = v;
2244 }
2245
2246 void stronglyConnectedMatrix()
2247 {
2248     memset(vc, 0, n * sizeof(int)); // dfs on G
2249     for (int i = 0; i < n; i++)
2250         if (!vc[i])
2251             dfsMatrix(i, false);
2252
2253     memset(vc, -1, n * sizeof(int)); // dfs on G'(reversed)
2254     c = 0;
2255     for (int i = 0; i < n; i++)
2256         if (vc[sorted[i]] == -1)
2257         {
2258             dfsMatrix(i, true);
2259             c++;
2260         }
2261 }
2262
2263 // ----- StronglyConnected List
2264
2265 int n, t, c, sorted[NODES], vc[NODES];
2266 vector<int> neigh[NODES], rneigh[NODES];
2267
2268 void dfsList(int v)
2269 {
2270     vc[v] = 1;
2271     for (vector<int>::iterator it = neigh[v].begin(); it !=

```

```

neigh[v].end(); it++)
2272         if (!vc[*it])
2273             dfsList(*it);
2274     sorted[n - ++t] = v;
2275 }
2276
2277 void rDfsList(int v)
2278 {
2279     vc[v] = c;
2280     for (vector<int>::iterator it = rneigh[v].begin(); it !=
2281         rneigh[v].end(); it++)
2282         if (vc[*it] == -1)
2283             rDfsList(*it);
2284 }
2285
2286 void stronglyConnectedList()
2287 {
2288     memset(vc, 0, n * sizeof(int)); // dfs on G
2289     for (int i = 0; i < n; i++)
2290         if (!vc[i])
2291             dfsList(i);
2292
2293     memset(vc, -1, n * sizeof(int)); // dfs on G'(reversed)
2294     c = 0;
2295     for (int i = 0; i < n; i++)
2296         if (vc[sorted[i]] == -1)
2297         {
2298             rDfsList(i);
2299             c++;
2300         }
2301 }
2302
2303 =====
2304 TRIE HAAMED
2305 =====
2306 // Trie - Haamed
2307
2308 #include<iostream>
2309 #include<string>
2310
2311 using namespace std;
2312
2313 int n;
2314 string words[130000];
2315
2316 struct Trie
2317 {
2318     Trie *next[26];
2319     int mark;
2320 }zero,*head;
2321
2322 Trie *insert(Trie *ptr,const char *txt)
2323 {
2324     if(!txt)
2325         return ptr;
2326     if(!ptr)
2327     {
2328         ptr=new Trie;

```

```

2327     *ptr=zero;
2328 }
2329 if(*txt)
2330     ptr->next[*txt-'a']=insert(ptr->next[*txt-'a'],txt+1);
2331 else
2332     ptr->mark=1;
2333 return ptr;
2334 }
2335
2336 int search(Trie *ptr,const char *txt,int mod)
2337 {
2338     if(!ptr||!txt)
2339         return 0;
2340     if(mod&&ptr->mark)
2341         if(search(head,txt,0))
2342             return 1;
2343     if(*txt)
2344         return search(ptr->next[*txt-'a'],txt+1,mod);
2345     return mod?0:ptr->mark;
2346 }
2347
2348 int main()
2349 {
2350     n=0;
2351     head=NULL;
2352     while(cin>>words[n])
2353         head=insert(head,words[n++].c_str());
2354     for(int i=0;i<n;i++)
2355         if(search(head,words[i].c_str(),1))
2356             cout<<words[i]<<endl;
2357     return 0;
2358 }
2359 =====
2360 WMATCHING BABAK
2361 =====
2362 // Matching + Weighted Matching - Babak
2363
2364 //BEGIN OF SOURCE CODE
2365 #include <cstring>
2366 #include <iostream>
2367 using namespace std;
2368
2369 #define MAXSIZE 100
2370 #define INF 1000000000
2371
2372 int match[MAXSIZE];
2373
2374 bool dfs(int v, int n, int adj[MAXSIZE][MAXSIZE], int mark[], int s[],
2375         int t[])
2376 {
2377     int i;
2378     s[v] = 1;
2379     mark[v] = 1;
2380     for (i = 0; i < n; i++)
2381         if (adj[v][i])
2382             if (t[i] == 1, match[i] == -1 || (!mark[match[i]] &&

```

```

2383         dfs(match[i], n, adj, mark, s, t)))
2384         return match[i] = v, true;
2385     }
2386     return false;
2387 }
2388
2389 bool matching(int n, int adj[MAXSIZE][MAXSIZE], int s[], int t[], int
2390 sa[])
2391 {
2392     int i;
2393     int max = 0;
2394     int mark[MAXSIZE];
2395     memset(mark, 0, sizeof(mark));
2396     for (i = 0; i < n; i++)
2397         if (!sa[i] && !mark[i] && dfs(i, n, adj, mark, s, t))
2398             {
2399                 memset(mark, 0, sizeof(mark));
2400                 sa[i] = 1;
2401             }
2402     for (i = 0; i < n; i++)
2403         if (!sa[i])
2404             return false;
2405     return true;
2406 }
2407
2408 void weighted(int n, int m, int weight[MAXSIZE][MAXSIZE])
2409 {
2410     int i, j;
2411     int size = 0;
2412     int sa[MAXSIZE], s[MAXSIZE], t[MAXSIZE];
2413     int cover[2][MAXSIZE];
2414     int adj[MAXSIZE][MAXSIZE];
2415     memset(match, -1, sizeof(match));
2416     memset(sa, 0, sizeof(sa));
2417     memset(adj, 0, sizeof(adj));
2418     if (m > n)
2419         n = m;
2420     for (i = 0; i < n; i++)
2421     {
2422         int index = 0;
2423         for (j = 1; j < n; j++)
2424             if (weight[i][index] < weight[i][j])
2425                 index = j;
2426         cover[1][i] = 0;
2427         cover[0][i] = weight[i][index];
2428         adj[i][index] = 1;
2429     }
2430
2431     while (!matching(n, adj, s, t, sa))
2432     {
2433         memset(s, 0, sizeof(s));
2434         memset(t, 0, sizeof(t));
2435         matching(n, adj, s, t, sa);
2436         int min = INF;
2437         for (i = 0; i < n; i++)
2438             for (j = 0; j < n; j++)
2439                 if (s[i] && !t[j] && !adj[i][j])
2440                     if (cover[0][i] + cover[1][j] - weight[i][j] < min)

```



```

2438         min = cover[0][i] + cover[1][j] - weight[i][j];
2439     for (i = 0; i < n; i++)
2440     if (s[i])
2441         cover[0][i] -= min;
2442     for (i = 0; i < n; i++)
2443     if (t[i])
2444         cover[1][i] += min;
2445     for (i = 0; i < n; i++)
2446     for (j = 0; j < n; j++)
2447     if ((s[i] && !t[j]) || (!s[i] && t[j]))
2448     if ((cover[0][i] + cover[1][j] - weight[i][j]) == 0)
2449         adj[i][j] = 1;
2450     else
2451         adj[i][j] = 0;
2452 }
2453 }
2454 //END OF SOURCE CODE
2455
2456 main()
2457 {
2458     int i, j, tn;
2459     int n, m, sum;
2460     int weight[MAXSIZE][MAXSIZE];
2461     for (cin >> tn; tn--;)
2462     {
2463         memset(weight, 0, sizeof weight);
2464         cin >> n >> m;
2465         for (i = 0; i < n; i++)
2466             for (j = 0; j < m; j++)
2467                 cin >> weight[i][j];
2468
2469         weighted(n, m, weight);
2470
2471         sum = 0;
2472         for (i = 0; i < max(n, m); i++)
2473         {
2474             if (i < m && match[i] < n)
2475             {
2476                 cout << match[i] << ' ' << i << endl;
2477                 sum += weight[match[i]][i];
2478             }
2479         }
2480         cout << sum << endl;
2481     }
2482     return 0;
2483 }
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494

```

```

2495 pi =
3.141592653589793238462643383279502884197169399375105820974944592307816
40628620899862803482534211706798214808651328230664709384460955058223172
53594081284811174502841027019385211055596446229489549303819644288109756
65933446128475648233786783165271201909145648566923460348610454326648213
39360726024914127372458700660631558817488152092096282925409171536436789
25903600113305305488204665213841469519415116094330572703657595919530921
86117381932611793105118548074462379962749567351885752724891227938183011
94912983367336244065664308602139494639522473719070217986094370277053921
71762931767523846748184676694051320005681271452635608277857713427577896
09173637178721468440901224953430146549585371050792279689258923542019956
11212902196086403441815981362977477130996051870721134999999837297804995
10597317328160963185950244594553469083026425223082533446850352619311881
71010003137838752886587533208381420617177669147303598253490428755468731
159562863882353759375195778185778053217122680661300192787661119590921
64201989380952572010654858632788659361533818279682303019520353018529689
95773622599413891249721775283479131515574857242454150695950829533116861
72785588907509838175463746493931925506040092770167113900984882401285836
16035637076601047101819429555961989467678374494482553797747268471040475
34646208046684259069491293313677028989152104752162056966024058038150193
51125338243003558764024749647326391419927260426992279678235478163600934
17216412199245863150302861829745557067498385054945885869269956909272107
97509302955321165344987202755960236480665499119881839179775366369807426
54252786255181841757467289097777279380008164706001614524919217321721477
23501414419735685481613611573525521334757418494684385233239073941433345
47762416862518983569485562099219222184272550254256887671790494601653466
8049886272327917860857843882796797668145410095388378636095068006422512
52051173929848960841284886269456042419652850222106611863067442786220391
94945047123713786960956364371917287467764657573962413890865832645995813
39047802759009946576407895126946839835259570982582262052248940772671947
82684826014769909026401363944374553050682034962524517493996514314298091
90659250937221696461515709858387410597885959772975498930161753928468138
26868386894277415599185592524595395943104997252468084598727364469584865
38367362226260991246080512438843904512441365497627807977156914359977001
2961608944169486855584840635342207225828488648158456028506016842739452
26746767889525213852254995466672782398645659611635488623057745649803559
36345681743241125150760694794510965960940252288797108931456691368672287
48940560101503308617928680920874760917824938589009714909675985261365549
78189312978482168299894872265880485756401427047755513237964145152374623
43645428584447952658678210511413547357395231134271661021359695362314429
52484937187110145765403590279934403742007310578539062198387447808478489
68332144571386875194350643021845319104848100537061468067491927819119793
99520614196634287544406437451237181921799983910159195618146751426912397
48940907186494231961567945208095146550225231603881930142093762137855956
63893778708303906979207734672218256259966150142150306803844773454920260
54146659252014974428507325186660021324340881907104863317346496514539057
96268561005508106658796998163574736384052571459102897064140110971206280
43903975951567715770042033786993600723055876317635942187312514712053292
81918261861258673215791984148488291644706095752706957220917567116722910
98169091528017350671274858322287183520935396572512108357915136988209144
42100675103346711031412671113699086585163983150197016515116851714376576
1835155650884909898599823455283316355076479185358932261854896321329
33089857064204675259070915481416549859461637180270981994309924488957571
28289059232332609729971208443357326548938239119325974636673058360414281
38830320382490375898524374417029132765618093773444030707469211201913020
33038019762110110044929321516084244485963766983895228684783123552658213
14495768572624334418930396864262434107732269780280731891544110104468232

```