Table Of Contents

Table Of Contents Of Geometry

```
 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 !=
       neigh[v].end(); it++)
79         if (*it != p && mark[*it] && dfi[*it] < low[v])
80             low[v] = dfi[*it];
81         else if (!mark[*it])
82         {
83             dfsList(*it, v);
84             if (low[*it] < low[v])
85                 low[v] = low[*it];
86             if (low[*it] == dfi[*it])
87                 cut.push_back(make_pair(v, *it));
88         }
89 }
90
91 void cutEdgeList()
92 {
93     t = 0;
94     memset(mark, false, n * sizeof(bool));
95     cut = vector<pair<int, int> >();
96     for (int i = 0; i < n; i++)
97         if (!mark[i])
98             dfsList(i, -1);
99 }
100
101 // ---------- Matrix CutEdge
102 int n, graph[NODES][NODES], dfi[NODES], t, low[NODES];
103 bool mark[NODES];
104 vector<pair<int, int> > cut;
105
106 void dfsMatrix(int v, int p)
107 {
108     mark[v] = true;
109     dfi[v] = t++;
110     low[v] = dfi[v];
111     for (int i = 0; i < n; i++)
112         if (i != p && graph[v][i] && mark[i] && dfi[i] < low[v])
113             low[v] = dfi[i];
```

```
        else if (graph[v][i] && !mark[i])
        {
            dfsMatrix(i, v);
            if (low[i] < low[v])
                low[v] = low[i];
            if (low[i] == dfi[i])
                cut.push_back(make_pair(v, i));
        }
}

void cutEdgeMatrix()
{
    t = 0;
    memset(mark, false, n * sizeof(bool));
    cut = vector<pair<int, int> >();
    for (int i = 0; i < n; i++)
        if (!mark[i])
            dfsMatrix(i, -1);
}
========================================================================
CUT VERTEX - BICONNECTED COMPONENT
========================================================================
#include <iostream>
#include <vector>
#include <set>

using namespace std;

#define NODES 100

vector<int> st;
vector<vector<int> > components;

void componentFound(int v)
{
    components.push_back(vector<int>());
    while (st.back() != v)
    {
        components.back().push_back(st.back());
        st.pop_back();
    }
    components.back().push_back(v);
}

// ---------- Matrix CutVertex

int n, graph[NODES][NODES], dfi[NODES], t, low[NODES], f;
bool mark[NODES];
set<int> cut;

void dfsMatrix(int v)
{
    st.push_back(v);
    int branches = 0;
    mark[v] = true;
    dfi[v] = t++;
    low[v] = dfi[v] - 1;
```



```
    for (int i = 0; i < n; i++)
        if (graph[v][i] && mark[i] && dfi[i] < low[v])
            low[v] = dfi[i];
        else if (graph[v][i] && !mark[i])
        {
            branches++;
            dfsMatrix(i);
            if (low[i] < low[v])
                low[v] = low[i];
            if (low[i] == dfi[v])
            {
                if (v != f)
                    cut.insert(v);
                componentFound(v);
            }
        }
    if (v == f && branches > 1)
        cut.insert(f);
}

void cutVertexMatrix()
{
    t = 0;
    memset(mark, false, n * sizeof(bool));
    cut = set<int>();
    st = vector<int>();
    components = vector<vector<int> >();
    for (int i = 0; i < n; i++)
        if (!mark[i])
        {
            dfsMatrix(f = i);
            st.pop_back();
        }
}

// ---------- List CutVertex

vector<int> neigh[NODES];
int n, t, f, dfi[NODES], low[NODES];
bool mark[NODES];
set<int> cut;

void dfsList(int v)
{
    st.push_back(v);
    int branches = 0;
    mark[v] = true;
    dfi[v] = t++;
    low[v] = dfi[v] - 1;
    for (vector<int>::iterator it = neigh[v].begin(); it !=
    neigh[v].end(); it++)
        if (mark[*it] && dfi[*it] < low[v])
            low[v] = dfi[*it];
        else if (!mark[*it])
        {
            branches++;
            dfsList(*it);
```

```
            if (low[*it] < low[v])
                low[v] = low[*it];
            if (low[*it] == dfi[v])
            {
                if (v != f)
                    cut.insert(v);
                componentFound(v);
            }
        }
    if (v == f && branches > 1)
        cut.insert(f);
}

void cutVertexList()
{
    t = 0;
    memset(mark, false, n * sizeof(bool));
    cut.clear();
    st.clear();
    components = vector<vector<int> >();
    for (int i = 0; i < n; i++)
        if (!mark[i])
        {
            dfsList(f = i);
            st.pop_back();
        }
}
========================================================================
EQUATOIN
========================================================================
/*-----------------------------------------------------------------------*
 * Linear Equation Solver
 *  * Solves a system of linear equations using Gauss Method.
 *  * You should :
 *      a) put equation matrix in mat[0..n - 1][0..n - 1]
 *      b) put array of answers in mat[n][0..n - 1]
 *      c) call solve(n)
 *
 *  * Solve(n) returns :
 *      a) 0 : if system has a unique answer
 *      b) 1 : if system has infinite answers
 *      c) 2 : if system has no answers
 *
 *  * It also fills mark[0..n - 1] with above flags (0, 1, 2) to
 *    indicate that the system had unique/infinite/no answer(s)
 *    for parameter[0..n - 1]
 *
 *  * If the system has a unique answer for parameter[i] then after
 *    calling solve(n) the answer for parameter[i] equals:
 *    (mat[i][i] / mat[i][n])
 *
 *-----------------------------------------------------------------------*/
#include <fstream>
#include <cmath>
#include <cstring>

using namespace std;
```



```

const double epsillon = 1e-6
;

double mat[100][101];
int mark[100];

int ZeroRow(int r, int n)
{
    for (int i = 0; i < n; i++)
        if (fabs(mat[r][i]) >= epsillon)
            return 0;
    return 1;
}

void AddRow(double a, int source, int target, int n)
{
    for (int i = 0; i < n + 1; i++)
        mat[target][i] += a * mat[source][i];
}

void ChangeRow(int source, int target, int n)
{
    double temp;
    for (int i = 0; i < n + 1; i++)
    {
        temp = mat[target][i];
        mat[target][i] = mat[source][i];
        mat[source][i] = temp;
    }
}

int Solve(int n)
{
    int i, j;
    int flag = 0;
    memset(mark, 0, sizeof(mark));
    for (i = 0; i < n; i++)
    {
        j = i;
        for (j = i; (fabs(mat[j][i]) < epsillon) && (j < n); j++);
            if ((j == n) && mark[i-1])
                j = i - 1;

        if (j == n)
        {
            flag = 1;
            mark[i] = 1;
        }

        if ((j != i) && (j != n))
            ChangeRow(j, i, n);

        if (j != n)
            for (j = 0; j < n; j++)
                if (j != i)
                    AddRow((-mat[j][i] / mat[i][i]), i, j, n);
```

```
341
342         for (j = 0; j < n; j++)
343             if (fabs(mat[j][i]) < epsillon)
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]) >= epsillon))
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;
        mat[0][4] = 0;
381     mat[1][0] = 1; mat[1][1] = 2; mat[1][2] = 0; mat[1][3] = 2;
        mat[1][4] = 0;
382     mat[2][0] = 1; mat[2][1] = 3; mat[2][2] = 0; mat[2][3] = 3;
        mat[2][4] = 0;
383
384     int n = 3;
385     int res = Solve(n);
386     int flag = 0;
387
388     for(int i = 0; i < n; i++)
389     {
390         switch (mark[i])
391         {
392         case 0:
393             cout << "X(" << i << ") = " << mat[i][n] / mat[i][i] <<
                endl;
```



```
394         break;
395
396         case 1:
397             cout << "X(" << i << ") has infinite Answers . . ." <<
                endl;
398             if (flag < 1)
399                 flag = 1;
400         break;
401
402         case 2:
403             cout << "X(" << i << ") has no answers . . ." << endl;
404             if (flag < 2)
405                 flag = 2;
406         break;
407         }
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;
```

```
    int flag=0,ret=v;
    for(int i=0;i<n;i++)
        if(graph[v][i])
        {
            graph[v][i]--;
            graph[i][v]--;
            local.push_back(_p(v,i));
            v=i;
            while(v!=ret)
            {
                for(int j=0;j<n;j++)
                    if(graph[v][j])
                    {
                        graph[v][j]--;
                        graph[j][v]--;
                        local.push_back(_p(v,j));
                        v=j;
                        break;
                    }
            }
        }
    for(list<_p>::iterator i=local.begin();i!=local.end();i++)
    {
        list<_p> temp=euler(i->first);
        local.insert(i,temp.begin(),temp.end());
    }
    return local;
}
main()
{
    int N,m,a,b,flag;
    cin>>N;
    for(int z=1;z<=N;z++)
    {
        if(z>1)cout<<endl;
        cout<<"Case #"<<z<<endl;
        cin>>m;
        flag=n=0;
        memset(graph,0,sizeof graph);
        for(int i=0;i<m;i++)
        {
            cin>>a>>b;
            graph[a-1][b-1]++;
            graph[b-1][a-1]++;
            n=max(n,max(a,b));
        }
        dfs(n-1);
        for(int i=0;i<n;i++)
        {
            int d=0;
            for(int j=0;j<n;j++)
                d+=graph[i][j];
            if(d&1||!mark[i]&&d)
            {
                cout<<"some beads may be lost"<<endl;
                flag=1;break;
            }
```



```
        }
        if(!flag)
        {
            list<_p> l=euler(n-1);
            for(list<_p>::iterator i=l.begin();i!=l.end();i++)
                cout<<i->first+1<<' '<<i->second+1<<endl;
        }
    }
    return 0;
}
```

```
==========================================================================
GEOMETRY
==========================================================================
#include <cmath>
#include <algorithm>
#include <vector>

using namespace std;

const double PI = 3.14159265358979323 85;//2 * acos(0.)
const double EPS = 1e-15;//5e-15 ?
const int PRECISION = 14;
inline bool equal(double x, double y)
{
    return fabs(x - y) < EPS;
}

inline bool less(double x, double y)
{
    return x <= y - EPS;
}

inline bool greater(double x, double y)
{
    return x >= y + EPS;
}

inline bool lessEqual(double x, double y)
{
    return x < y + EPS;
}

inline bool greaterEqual(double x, double y)
{
    return x > y - EPS;
}

inline double round(double x, int p)
{
    if (less(x, 0))
        return floor(x * pow(10., p) - .5) / pow(10., p);
    return floor(x * pow(10., p) + .5) / pow(10., p);
}

inline int sign(double x)
{
    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,
604                           double &a, double &b, double &c)
605 {
606     a = y2 - y1;
607     b = x1 - x2;
608     c = -y1 * b - x1 * a;
609 }
610
611 inline void lineThroughPoint(double x, double y, double m,
612                              double &a, double &b, double &c)
613 {
614     a = -m;
615     b = 1;
616     c = m * x - y;
617 }
```



```
618
619 bool segmentsIntersect(double x1, double y1, double x2, double y2,
620                        double x3, double y3, double x4, double y4)
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))||
630        greater(min(y1,y2),max(y3,y4))||greater(min(y3,y4),max(y1,y2)))
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))*
637                        ((x3-x2)*(y1-y2)-(y3-y2)*(x1-x2)),0))&&
638             (lessEqual(((x1-x4)*(y3-y4)-(y1-y4)*(x3-x4))*
639                        ((x2-x4)*(y3-y4)-(y2-y4)*(x3-x4)),0)));
640 /*
641     'lessEqual' : intersections at endpoints are valid
642     'less'      :       //      //     //     // invalid
643 */
644 }
645
646 int linesIntersectionPoint(double a1, double b1, double c1,
647                            double a2, double b2, double c2,
648                            double &x, double &y)
649 /*
650     return values:
651         0: lines are parallel
652         1: lines are intersecting at point (x, y)
653         2: lines are the same
654 */
655 {
656     if (equal(a1 * b2, a2 * b1))
657         if(equal(a1 , 0))
658             return equal(b1 * c2, b2 * c1) * 2;
659         else
660             return equal(a1 * c2, a2 * c1) * 2;
661     y = -(a1 * c2 - a2 * c1) / (a1 * b2 - a2 * b1);
662     if (equal(a1, 0))
663         x = -(b2 * y + c2) / a2;
664     else
665         x = -(b1 * y + c1) / a1;
666     return 1;
667 }
668
669 int segmentsIntersectionPoint(double x1, double y1, double x2, double
    y2,
670                               double x3, double y3, double x4, double
                                  y4,
671                               double &x5, double &y5)
672 /*
```

```
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;
```

```
    }
    double r = rnum / den;
    double s = snum / den;
    x5 = x1 + r * (x2 - x1);
    y5 = y1 + r * (y2 - y1);
    if (greater(r, 1)) res |= 8;
    if (less(r, 0)) res |= 16;
    if (greater(s, 1)) res |= 32;
    if (less(s, 0)) res |= 64;
    return (res ? res : 1);
}

int pointOnLine(double x, double y, double a, double b, double c)
{
    return equal(a * x + b * y + c, 0);
}

double multiply(double x1, double y1, double x2, double y2, double x3,
double y3)
/*
    cross product of lines p1-p2 and p1-p3
*/
{
    return (x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1);
}

int pointInSegment(double xp, double yp, double x1, double y1, double
x2, double y2)
{
    return (equal(multiply(xp, yp, x1, y1, x2, y2), 0) &&
        greaterEqual(xp, min(x1, x2)) && lessEqual(xp, max(x1, x2)) &&
        greaterEqual(yp, min(y1, y2)) && lessEqual(yp, max(y1, y2)));
}

void rotate(double &x, double &y, double angle)
/*
    rotate (x, y) around (0, 0)
    angle should be in radians
*/
{
    double tx = x * cos(angle) - y * sin(angle);
    double ty = x * sin(angle) + y * cos(angle);
    x = tx;
    y = ty;
}

int circlesIntersect(double x1, double y1, double r1,
                     double x2, double y2, double r2)
/*
    return values:
        0: no intersection
        1: one intersection
        2: two intersections
        3: the circles are the same
*/
{
    if (equal(x1, x2) && equal(y1, y2) && equal(r1, r2))
```



```
        return 3;
    double d = distance(x1, y1, x2, y2);
    if (greater(d, r1 + r2) || less(d, fabs(r1 - r2)))
        return 0;
    if (equal(d, r1 + r2))
        return 1;
    return 2;
}

double pointToLineDistance(double x, double y,
                           double a, double b, double c)
{
    return fabs(a * x + b * y + c) / sqrt(a * a + b * b);
}

int circleLineIntersect(double x, double y, double r,
                        double a, double b, double c)
/*
    return values:
        0: no intersection
        1: one intersection
        2: two intersections
*/
{
    double delta;
    if (equal(b, 0))
        delta = a * a * r * r - (a * x + c) * (a * x + c);
    else
        delta = b * b * ((a * a + b * b) * r * r - (a * x + b * y + c)
        * (a * x + b * y + c));
    return (less(delta, 0) ? 0 : (equal(delta, 0) ? 1 : 2));
}

int circleLineIntersectionPoints(double x, double y, double r,
                                 double a, double b, double c,
                                 double &x1, double &y1, double &x2,
                                 double &y2)
/*
    return values:
        0: no intersection
        1: one intersection at (x1, y1)
        2: two intersections at (x1, y1) and (x2, y2)
*/
{
    double delta;
    if (equal(b, 0))
        delta = a * a * r * r - (a * x + c) * (a * x + c);
    else
        delta = b * b * ((a * a + b * b) * r * r - (a * x + b * y + c)
        * (a * x + b * y + c));
    if (less(delta, 0))
        return 0;
    if (equal(delta, 0))
    {
        x1 = (-a * (a * x + b * y + c)) / (a * a + b * b);
        y1 = (-b * (a * x + b * y + c)) / (a * a + b * b);
        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)&&
976                 greaterEqual(x[i], min(x[p1], x[p2])) &&
                    lessEqual(x[i], max(x[p1], x[p2])) &&
977                 greaterEqual(y[i], min(y[p1], y[p2])) &&
                    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     }
```

```
}

struct CompCHGS
{
    int m;
    double *x, *y;
    CompCHGS(int m, double *x, double *y): m(m), x(x), y(y) {}
    bool operator()(const int i1, const int i2) const
    {
        return (greater((x[i1] - x[m]) * (y[i2] - y[m]), (x[i2] - x[m])
        * (y[i1] - y[m])) ||
/*
    'greater' : counter-clockwise
    'less'    : clockwise
*/
            (equal((x[i1] - x[m]) * (y[i2] - y[m]), (x[i2] - x[m]) *
            (y[i1] - y[m])) &&
            greaterEqual(x[i1], min(x[i2], x[m])) && lessEqual(x[i1],
            max(x[i2], x[m])) &&
            greaterEqual(y[i1], min(y[i2], y[m])) && lessEqual(y[i1],
            max(y[i2], y[m])));
    }
};

void convexHullGS(int n, double x[], double y[], vector<int> &ch)
/*
    Convex Hull finding, Graham Scan algorithm, O(n*log(n))
*/
{
/*
    note the array size
*/
    int sorted[1000], i, m = 0;
    for (i = 0; i < n; i++)
    {
        sorted[i] = i;
        if (less(y[i], y[m]) || (equal(y[i], y[m]) && less(x[i],
        x[m])))
            m = i;
    }
    sorted[0] = m;
    sorted[m] = 0;
/*
    the angles 1 to n - 1 should be sorted in ascending order. for each
    two equal angles,
    the angle with enpoint nearer to origin should become first.
*/
    sort(sorted + 1, sorted + n, CompCHGS(m, x, y));
/*
    after the sort, order of the last equal angles should be reversed.
*/
    for (i = n - 1; i > 1 &&
        equal((x[sorted[i]] - x[m]) * (y[sorted[i - 1]] - y[m]),
        (x[sorted[i - 1]] - x[m]) * (y[sorted[i]] - y[m])); i--);
    if (i > 1) reverse(sorted + i, sorted + n);

    ch = vector<int>();
```



```
    ch.push_back(sorted[0]);
    if (n > 1)
    {
        ch.push_back(sorted[1]);
        for (i = 2; i < n; i++)
        {
            while (ch.size() > 1 &&
                greaterEqual(multiply(x[sorted[i]], y[sorted[i]],
                x[ch.back()], y[ch.back()],
                x[ch[ch.size() - 2]], y[ch[ch.size() - 2]]), 0))
/*
    'greater'      : counter-clockwise, maximum number of points
    'greaterEqual' : counter-clockwise, minimum number of points
    'less'         : clockwise, maximum number of points
    'lessEqual'    : clockwise, minimum number of points
*/
                ch.pop_back();
            ch.push_back(sorted[i]);
        }
/*
    maximum number of points : exclude the following 'if' statement
    minimum number of points : include the following 'if' statement
*/
        if (ch.size() > 2 && equal(multiply(x[ch.front()],
        y[ch.front()], x[ch.back()],
            y[ch.back()],x[ch[ch.size()-2]],y[ch[ch.size()-2]]),0))
            ch.pop_back();
    }
}

int polygonOrientation(int n, double x[], double y[])
/*
    result values:
         1 : counter-clockwise
        -1 : clockwise
*/
{
    int m = 0;
    for (int i = 1; i < n; i++)
        if(less(y[i],y[m])||(equal(y[i],y[m])&&less(x[i],x[m])))
            m = i;
    return sign(multiply(x[m], y[m], x[(m + n - 1) % n], y[(m + n - 1)
    % n], x[(m + 1) % n], y[(m + 1) % n]));
}

int generalPolygonInclusion(int n, double x[], double y[], double xp,
double yp)
/*
    return values:
        0: (xp, yp) is outside the polygon
        1: (xp, yp) is inside the polygon
        2: (xp, yp) is on the polygon side
*/
{
    int i;
    for (i = 1; i <= n; i++)
        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
      'a-o-b' from 'a' to 'b'
1117  */
1118  {
1119      double ab = multiply(xo, yo, xa, ya, xb, yb);
1120      double ap = multiply(xo, yo, xa, ya, xp, yp);
1121      double bp = multiply(xo, yo, xb, yb, xp, yp);
1122      return (greaterEqual(ab, 0) && greaterEqual(ap, 0) && lessEqual(bp,
      0)) ||
1123          (less(ab, 0) && (greaterEqual(ap, 0) || lessEqual(bp, 0)));
1124  }
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
      same line.
1130  */
1131  /*
1132      return values:
1133          0: (xp, yp) is outside the polygon
1134          1: (xp, yp) is inside the polygon
1135          2: (xp, yp) is on the polygon side
1136  */
1137  {
1138      double xm, ym;
1139      int i;
1140  /*
1141      the points are checked to be in counter-clockwise order
1142  */
1143      if (polygonOrientation(n, x, y) > 0)
1144      {
1145          reverse(x, x + n);
1146          reverse(y, y + n);
1147      }
1148  /*
1149      find an internal point
1150  */
1151      for (i = 2; i < n; i++)
1152          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
1197          , double x3, double y3,double &x, double &y, double &r)
1198  /*
1199      result values:
1200          false : the points are on a same extension
1201          true  : circle is (x, y, r)
1202  */
1203  {
1204      double d = 2*(y1*x3+y2*x1-y2*x3-y1*x2-y3*x1+y3*x2);
1205      if (equal(d, 0))
1206          return false;
1207      x = (y2 * x1 * x1 - y3 * x1 * x1 - y2 * y2 * y1 + y3 * y3 * y1 +
1208           x2 * x2 * y3 + y1 * y1 * y2 + x3 * x3 * y1 - y3 * y3 * y2 -
1209           x3 * x3 * y2 - x2 * x2 * y1 + y2 * y2 * y3 - y1 * y1 * y3)/d;
```

```
    y = (x1 * x1 * x3 + y1 * y1 * x3 + x2 * x2 * x1 - x2 * x2 * x3 +
         y2 * y2 * x1 - y2 * y2 * x3 - x1 * x1 * x2 - y1 * y1 * x2 -
         x3 * x3 * x1 + x3 * x3 * x2 - y3 * y3 * x1 + y3 * y3 * x2)/d;
    r = sqrt((x1 - x) * (x1 - x) + (y1 - y) * (y1 - y));
    return true;
}


double cross(double ax,double ay,double bx,double by)
{
       return ax*by-ay*bx;
}
double dot(double ax,double ay,double bx,double by)
{
       return ax*bx+ay*by;
}

double ang(double ax,double ay,double bx,double by)
{
       return atan2(cross(ax,ay,bx,by),dot(ax,ay,bx,by));
}
void rotate(double &px,double &py,double t)
{
    double x,y;
    x=px;
    y=py;
    px=x*cos(t)-y*sin(t);
    py=x*sin(t)+y*cos(t);
}
void rotate(double &px,double &py,double &pz,double vx,double vy,double
vz,double t)
{
    double ty,tx;
    ty=ang(vz,vx,1,0);
    rotate(pz,px,ty);
    rotate(vz,vx,ty);
    tx=ang(vy,vz,0,1);
    rotate(py,pz,tx);
    rotate(px,py,t);
    rotate(py,pz,-tx);
    rotate(pz,px,-ty);
}
================================================================================
GRID
================================================================================
#include<iostream>

using namespace std;

int m,n;
int dir[4][2]={0,-1,-1,0,0,1,1,0};
int edge[2][200][200];
int mark[200][200];
int cc,len,best;

int g(int a,int b,int k)
{
```



```
       return k<2?edge[1-k][a][b]:edge[1-(k%2)][a+(k%2)][b+1-(k%2)];
}

void dfs(int a,int b)
{
       if(a<0||b<0||a>=m+n+4||b>=m+n+4||mark[a][b])return;
       mark[a][b]=1;
       len++;
       for(int k=0;k<4;k++)
              if(!g(a,b,k))
                  dfs(a+dir[k][0],b+dir[k][1]);
}

main()
{
       for(int N=0;cin>>n>>m,m||n;N++)
       {
              memset(mark,0,sizeof mark);
              memset(edge,0,sizeof edge);
              for(int i=0;i<m;i++)
                  for(int j=0;j<n;j++)
                  {
                      char c;
                      cin>>c;

                      edge[c=='\\'][1+i+j+1][m-i+j+1]=edge[c=='\\'][1
                      +i+j-(c=='\\')+1][m-i+j-(c=='/')+1]=1;
                  }
              cc=best=0;
              dfs(0,0);
              for(int i=0;i<m+n+4;i++)
                  for(int j=0;j<m+n+4;j++)
                      if(!mark[i][j])
                      {
                              len=0;
                              cc++;
                              dfs(i,j);
                              best>?=len;
                      }
              cout<<"Maze #"<<N+1<<":"<<endl;
              if(cc)
                      cout<<cc<<" Cycles; the longest has length
                      "<<best<<"."<<endl;
              else
                      cout<<"There are no cycles."<<endl;
              cout<<endl;
       }
}
================================================================================
HUGE NUMBERS
================================================================================
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
#define MAXLEN 10000
struct BigNum{
```

```
    short n[MAXLEN];
    int len;
} temp,one,zero,ans,res,up,xx;
void print(BigNum &a)
{
    for(int i=a.len-1;i+1;i--)cout<<a.n[i];
    cout<<endl;
}
void assignInt(BigNum &a,int n)
{
    int i=0;
    do {a.n[i++]=n%10;n/=10;}while(n);
    a.n[i]=-1;
    a.len=i;
}
void assignStr(BigNum &a,string n)
{
    int i=0,len=n.length();
    for(i=len-1;i+1;i--)
        a.n[len-1-i]=n[i]-'0';
    a.n[len]=-1;
    a.len=len;
}
void copyNum(BigNum &des,BigNum &src)
{
    int i;
    for(i=0;src.n[i]+1;i++)des.n[i]=src.n[i];
    des.n[i]=-1;
    des.len=i;
}
int numcmp(BigNum &a,BigNum &b)
{
    int l1=a.len,l2=b.len;
    if(l1!=l2)return l1>l2?1:-1;
    for(int i=l1-1;i+1;i--)
        if(a.n[i]!=b.n[i])return a.n[i]<b.n[i]?-1:1;
    return 0;
}
void add(BigNum &a,BigNum &b,BigNum &c)//c=a+b
{
    int i,t=0;
    for(i=0;a.n[i]>=0&&b.n[i]>=0;i++)
        c.n[i]=(t=(a.n[i]+b.n[i]+t))%10,t/=10;
    for(short *p=(a.n[i]==-1?&b.n[i]:&a.n[i]);*p!=-1;p++,i++)
        c.n[i]=(t=(*p+t))%10,t/=10;
    if(t)c.n[i++]=t;
    c.n[i]=-1;
    c.len=i;
}
void mul(BigNum &a,BigNum &b,BigNum &c)//c=a*b
{
    int t=0,tt,i,j;
    memset(c.n,0,sizeof(short)*MAXLEN);
    for(i=0;a.n[i]+1;i++)
    {
        for(j=0;b.n[j]+1;j++)
            tt=c.n[i+j]+a.n[i]*b.n[j]+t,c.n[i+j]=tt%10,t=tt/10;
```



```
        for(;t;j++)
            tt=c.n[i+j]+t,c.n[i+j]=tt%10,t=tt/10;
    }
    for(i+=j;!c.n[i]&&i;i--);
    c.n[i+1]=-1;
    c.len=i+1;
}
void div2(BigNum &a,BigNum &b)
{
    int i=a.len,t,j;
    b.n[t=i]=-1;
    b.len=i;
    for(i--,j=i;i+1;i--,j--)
        if((a.n[i]&1)&&i)
        {
            b.n[j]=a.n[i]/2;
            a.n[i-1]+=10;
        }else
            b.n[j]=a.n[i]/2;
    for(i=t-1;!b.n[i]&&i;b.len=i,b.n[i--]=-1);
}
void sub(BigNum &a,BigNum &b,BigNum &c)//c=a-b && should a>=b,a maybe
changed
{
    int i,j;
    for(i=0;a.n[i]+1&&b.n[i]+1;i++)
        if(a.n[i]-b.n[i]>=0)c.n[i]=a.n[i]-b.n[i];
        else
        {
            c.n[i]=10-b.n[i]+a.n[i];
            for(j=i+1;!a.n[j];j++)a.n[j]=9;
            a.n[j]--;
        }
    for(;a.n[i]+1;i++)c.n[i]=a.n[i];
    c.len=i;
    c.n[i--]=-1;
    while(i+1&&!c.n[i])c.n[i--]=-1,c.len--;
    if(i==-1)c.n[0]=0,c.len=1;
}
void removeZero(BigNum &n)//for use in div
{
    int i=n.len-1;
    while(!n.n[i]&&i)n.len--,n.n[i--]=-1;
}
void div(BigNum &a,BigNum &b,BigNum &r,BigNum &q)// a=b*Q+R
{
    BigNum sum[11],temp,temp1;
    int cmp=numcmp(a,b);
    if(!cmp)
    {
        assignInt(r,0);
        assignInt(q,1);
        return;
    }else if(cmp==-1)
    {
        copyNum(r,a);
        assignInt(q,0);
```

```
        return;
    }
    copyNum(sum[0],b);
    for(int i=1;i<11;i++)
        add(sum[i-1],b,sum[i]);
    int index=0,k;
    int first=b.len;
    copy(a.n+a.len-first,a.n+a.len,temp.n);
    temp.n[temp.len=first]=-1;
    if(numcmp(temp,b)<0)
        first++;
    copy(a.n+a.len-first,a.n+a.len,temp.n);
    temp.n[temp.len=first]=-1;
    int firstLen=a.len-first;
    while(1)
    {
        for(k=0;k<11;k++)
            if(numcmp(temp,sum[k])<0)
                break;
        q.n[index++]=k;
        sub(temp,sum[k-1],temp1);
        if(firstLen<1)break;
        for(int i=temp1.len+1;i;i--)
            temp1.n[i]=temp1.n[i-1];
        temp1.n[0]=a.n[--firstLen];
        temp1.len++;
        removeZero(temp1);
        while(numcmp(temp1,b)<0)
        {
            q.n[index++]=0;
            if(firstLen<1)break;
            for(int i=temp1.len+1;i;i--)
                temp1.n[i]=temp1.n[i-1];
            temp1.n[0]=a.n[--firstLen];
            temp1.len++;
            removeZero(temp1);
        }
        if(numcmp(temp1,b)<0)break;
        copyNum(temp,temp1);
    }
    copyNum(r,temp1);
    q.n[q.len=index]=-1;
    reverse(q.n,q.n+index);
}
========================================================================
LINE POLYGON INTERSECTION
========================================================================
#include<iostream>
#include<algorithm>
#include<cstdio>
using namespace std;

int n;
double list[200][2]={0,0,10,0,10,10,0,10};
int mark[200];
double a[2],b[2],m[2],c,coef;
```



```
void cross(double aa[],double p1[],double p2[])
{
    double ap,bp,cp;
    bp=p1[0]-p2[0];
    ap=-(p1[1]-p2[1]);
    cp=-bp*p1[1]-ap*p1[0];

    double det=-(m[0]*bp-m[1]*ap);
    aa[0]=(c*bp-cp*m[1])/det;
    aa[1]=(m[0]*cp-c*ap)/det;
}

double area()
{
    double ans=0;
    for (int i=1;i<n;i++)
        ans+=(list[i-1][0]*list[i][1]-list[i][0]*list[i-1][1]);
    ans+=(list[n-1][0]*list[0][1]-list[0][0]*list[n-1][1]);
    return ans/2;
}

int main()
{
    n=4;
    b[0]=b[1]=0;
    int ff=0;
    while(cin>>a[0]>>a[1])
    {
        char txt[1024];
        cin>>txt;
        if(!strcmp(txt,"Same")^(a[0]==b[0]&&a[1]==b[1]))
            ff=1;
        if(ff)
        {
            cout<<"0.00"<<endl;
            continue;
        }
        if(!strcmp(txt,"Same"))
        {
            sprintf(txt,"%.2lf",area());
            cout<<txt<<endl;
            continue;
        }
        m[0]=a[0]-b[0];
        m[1]=a[1]-b[1];
        c=-(m[0]*(a[0]+b[0])/2+m[1]*(a[1]+b[1])/2);
        int mod=(a[0]*m[0]+a[1]*m[1]+c>=0);
        coef=(mod^(strcmp(txt,"Hotter")!=0))?1:-1;
        int j=-1;
        for(int i=0;i<n;i++)
           if(!(mark[i]=(coef*(list[i][0]*m[0]+list[i][1]*m[1]+c)>=0)))
               j=i;
        if(j+1)
        {
            int first,last;
            for(first=j;!mark[first];first=(first-1+n)%n);
            for(last=j;!mark[last];last=(last+1)%n);
```

```
            double aa[2],bb[2];
            cross(aa,list[first],list[(first+1)%n]);
            cross(bb,list[(last-1+n)%n],list[last]);
            double _list[200][2];
            int _n=n;
            memcpy(_list,list,sizeof(list));
            n=0;
            for(int i=last;i!=(first+1)%_n;i=(i+1)%_n)
            {
                list[n][0]=_list[i][0];
                list[n++][1]=_list[i][1];
            }
            list[n][0]=aa[0];list[n++][1]=aa[1];
            list[n][0]=bb[0];list[n++][1]=bb[1];
        }
        sprintf(txt,"%.2lf",area());
        cout<<txt<<endl;
        b[0]=a[0];b[1]=a[1];
    }
    return 0;
}
========================================================================
MATCH HAAMED
========================================================================
// Bipartite Matching - Haamed

#include<iostream.h>
#include<cstring>

#define MAXNODE 1000

int m,n;
int graph[MAXNODE][MAXNODE];
int mark[MAXNODE],match[MAXNODE];

int dfs(int v)
{
    if(v==-1)return 1;
    mark[v]=1;
    for(int i=0;i<n;i++)
        if(graph[v][i]&&(match[i]==-1||!mark[match[i]])&&dfs(match[i]))
        {
            match[i]=v;
            return 1;
        }
    return 0;
}

int main()
{
    //init
    memset(mark,0,sizeof mark);
    memset(match,-1,sizeof match);
    //input
    cin>>m>>n;
    for(int i=0;i<m;i++)
        for(int j=0;j<n;j++)
```



```
            cin>>graph[i][j];
    //body
    for(int i=0;i<m;i++)
        if(!mark[i]&&dfs(i))
            memset(mark,0,sizeof mark);
    //output
    for(int j=0;j<n;j++)
        if(match[j]+1)
            cout<<match[j]<<' '<<j<<endl;
    return 0;
}
========================================================================
MATRIX INVERSION
========================================================================
#include <iostream>
#include <string>
#include <vector>
#include <cmath>

using namespace std;
#define double int

typedef vector<double> Row;
typedef vector<Row> Matrix;
int p;

void swapRows(Matrix &matrix, int m, int n)
{
        Row tmp = matrix[m];
        matrix[m] = matrix[n];
        matrix[n] = tmp;
}

void addRow(Matrix &matrix, double coef, int m, int n)
{
        for (unsigned i = 0; i < matrix[m].size();i++)
        {
                matrix[n][i] += coef * matrix[m][i];
                matrix[n][i]%=p;
        }
}

void multiplyRow(Matrix &matrix, int n, double coef)
{
        for (unsigned i = 0; i < matrix[n].size();i++)
        {
                matrix[n][i] *= coef;
                matrix[n][i]%=p;
        }
}

int invt[30001];
int inv(int a)
{
        if(invt[a%p])return invt[a%p];
        for(int i=1;i<p;i++)
                if((a*i)%p==1)return invt[a%p]=i;
```

```
}

Matrix inverse(Matrix matrix)
{
        Matrix result(matrix.size(),Row(matrix.size()));
        unsigned i, j, k;

        for (i = 0; i < matrix.size(); i++)
                result[i][i] = 1;

        for (i = 0; i < matrix.size(); i++)
        {
                for (k = i; k < matrix.size() && !matrix[k][i]; k++);
                if (k == matrix.size())
                        cout<<string("Given matrix is not invertible");
                swapRows(matrix, i, k);
                swapRows(result, i, k);
                for (j = 0; j < matrix.size(); j++)
                {
                        if (j == i)
                                continue;
                         double coef = ((-matrix[j][i] *
                         inv(matrix[i][i]))%p+p)%p;
                        addRow(matrix, coef, i, j);
                        addRow(result, coef, i, j);
                }
                 double revCoef = inv(matrix[i][i]);
                multiplyRow(matrix, i, revCoef);
                multiplyRow(result, i, revCoef);
        }
         return result;
}
int pp(int a,int n)
{
        if(!n)return 1;
        int t=pp(a,n/2);
        t=(t*t)%p;
        if(n&1)
                t=(t*a)%p;
        return t;
}

int main()
{
        int N;
        char str[100];
        for(cin>>N;N--;)
        {
                cin>>p>>str;
                memset(invt,0,sizeof invt);
                int n=strlen(str);
                vector<Row>mat(n);
                for(int i=0;i<n;i++)
                    for(int j=0;j<n;j++)
                        mat[i].push_back(pp(i+1,j));
                mat=inverse(mat);
                for(int i=0;i<n;i++)
```



```
                {
                    int ans=0;
                    for(int j=0;j<n;j++)
                        ans+=(str[j]=='*'?0:str[j]-'a'+1)*mat[i][j];
                    ans=((p+ans)%p+p)%p;
                    cout<<ans<<' ';
                }
                cout<<endl;
        }
        return 0;
}
```

```
=========================================================================
MAXFLOW
=========================================================================
#include <iostream>
#include <cstring>
#include <list>
#include <vector>

using namespace std;

const int NODES = 300;

int relabelToFront(int n, int c[NODES][NODES], int s, int t, int
f[NODES][NODES])
{
    int h[NODES], e[NODES];

    // initialize preflow
    memset(h, 0, n * sizeof(int));
    memset(e, 0, n * sizeof(int));
    memset(f, 0, NODES * NODES * sizeof(int));
    h[s] = n;
    int value = 0;
    for (int i = 0; i < n; i++)
        if (c[s][i])
        {
            f[s][i] = c[s][i];
            f[i][s] = -c[s][i];
            e[i] = c[s][i];
            value += c[s][i];
        }

    // constructing list of vertices
    list<int> ver;
    for (int i = 0; i < n; i++)
        if (i != s && i != t)
            ver.push_back(i);

    // constructing list of neighbors
    vector<int> neigh[NODES];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (c[i][j] || c[j][i])
                neigh[i].push_back(j);

    vector<int>::iterator cur[NODES];
```

```
    for (int i = 0; i < n; i++)
        cur[i] = neigh[i].begin();

    // body
    list<int>::iterator u = ver.begin();
    while (u != ver.end())
    {
        int uu = *u;
        int oldH = h[uu];

        // discharge(u)
        while (e[uu] > 0)
        {
            if (cur[uu] == neigh[uu].end())
            {
                // relabel(u)
                int minh = 2 * n;
                for (int i = 0; i < n; i++)
                    if (c[uu][i] - f[uu][i] > 0 && h[i] < minh)
                        minh = h[i];
                h[uu] = minh + 1;

                cur[uu] = neigh[uu].begin();
            } else
            {
                int vv = *cur[uu];
                if (c[uu][vv] - f[uu][vv] > 0 && h[uu] == h[vv] + 1)
                {
                    // push(u)
                    int d = min(e[uu], c[uu][vv] - f[uu][vv]);
                    f[uu][vv] += d;
                    f[vv][uu] = -f[uu][vv];
                    e[uu] -= d;
                    e[vv] += d;
                    if (vv == s)
                        value -= d;
                }
                else
                    cur[uu]++;
            }
        }

        if (h[uu] > oldH)
            ver.splice(ver.begin(), ver, u);
        u++;
    }

    return value;
}

int main()
{
    int n, c[NODES][NODES], f[NODES][NODES];
    while (cin >> n, n)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
```



```
                cin >> c[i][j];
        int s, t;
        cin >> s >> t;
        cout << "-> " << relabelToFront(n, c, s, t, f) << endl;
        /*for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
                cout << f[i][j] << '/' << c[i][j] << ' ';
            cout << endl;
        }
        cout << endl;*/
    }
    return 0;
}
=============================================================================
MAXFLOW ARASH
=============================================================================
/*---------------------------------------------------------------------------*
 * Maximum Flow
 * Implemented Algorithms :
 *
 *      - Ford Fulkerson Method (Edmonds-Karp Algorithm)
 *          Runs in O(V * E^2) time
 *      - Generic Preflow-Push Algorithm
 *          Runs in O(E * V^2) time
 *      - Maximum Bipartite Matching using Maximum-Flow
 *---------------------------------------------------------------------------*/

#include <fstream>
#include <vector>
#include <queue>
#include <cstring>
#include <set>
using namespace std;

#define NODE_NUM 100


int capacity[NODE_NUM][NODE_NUM];
int flow[NODE_NUM][NODE_NUM];
int n;


struct Path
{
    int capacity;
    vector<int> path;

    Path() : path(), capacity() {}
    Path(const Path &copy):capacity(copy.capacity),path(copy.path){}
};



// This function finds and returns augmenting paths in the Residual
Network
Path bfs(int src, int tgt)
```

```
{
    bool mark[NODE_NUM];
    queue<Path> q;

    memset(mark, false, sizeof(mark));
    while (!q.empty())
        q.pop();

    Path first;
    first.capacity = 10000000;
    first.path.push_back(src);
    q.push(first);

    while (!q.empty())
    {
        Path cur = q.front();
        q.pop();
        int last = *(cur.path.end() - 1);
        if (last == tgt)
            return cur;

        for (int i = 0; i < n; i++)
            if (!mark[i] && (capacity[last][i] - flow[last][i] > 0))
            {
                Path child;
                child.capacity = min(cur.capacity, capacity[last][i] -
                flow[last][i]);
                child.path = cur.path;
                child.path.push_back(i);
                mark[i] = true;
                q.push(child);
            }
    }

    first.capacity = 0;
    return first;
}


// Implementation of Edmonds-Karp Algorithm : uses bfs(src, tgt)
int maxFlow(int src, int tgt)
{
    memset(flow, 0, sizeof(flow));

    Path p;
    int answer = 0;
    while (p = bfs(src, tgt), p.capacity)
    {
        for (vector<int>::iterator it = p.path.begin() + 1; it !=
        p.path.end(); it++)
        {
            flow[*(it - 1)][*it] += p.capacity;
            flow[*it][*(it - 1)] -= p.capacity;
        }

        answer += p.capacity;
    }
```



```

    return answer;
}


// Implementation of Generic Preflow-Push Algorithm
int maxFlowPP(int src, int tgt)
{
    int h[NODE_NUM], e[NODE_NUM];

    memset(flow, 0, sizeof(flow));
    memset(h, 0, n * sizeof(int));
    memset(e, 0, n * sizeof(int));

    h[src] = n;

    for (int i = 0; i < n; i++)
        if (capacity[src][i])
        {
            flow[src][i] = capacity[src][i];
            flow[i][src] = -capacity[src][i];
            e[i] = capacity[src][i];
        }


    bool flag = true;

    while (flag)
    {
        flag = false;
        for (int i = 0; i < n; i++)
            if (i != src && i != tgt && e[i])
            {
                bool liftNeeded = false;
                int minHeight = 2 * n;
                for (int j = 0; j < n; j++)
                    if ((capacity[i][j] - flow[i][j]) > 0)
                    {
                        if (h[i] == h[j] + 1)
                        {
                            int d = min(e[i], capacity[i][j] -
                            flow[i][j]);
                            flag = true;
                            flow[i][j] += d;
                            flow[j][i] = -flow[i][j];
                            e[i] -= d;
                            e[j] += d;
                        }
                        else if (h[i] <= h[j])
                        {
                            minHeight = min(minHeight, h[j]);
                            liftNeeded = true;
                        }
                    }


                if (e[i] && liftNeeded)
```

```
                {
                    h[i] = 1 + minHeight;
                    flag = true;
                }
            }

    }

    return e[tgt];
}

// An Example : Maximum Bipartite Matching using Maximum Flow
// Returns : Cardinality of Maximum Bipartite Matching
int bipartiteMatch(set<int> first, set<int> second)
{
    for (set<int>::iterator it=first.begin();it!=first.end();it++)
        capacity[n][*it] = 1;

    for (set<int>::iterator it=second.begin();it!=second.end();it++)
        capacity[*it][n + 1] = 1;

    n += 2;

    return maxFlowPP(n - 2, n - 1);
}


int main()
{
    int v;
    memset(capacity, 0, sizeof(capacity));
    cin >> n >> v;

    for (int i = 0; i < v; i++)
    {
        int s, t;
        cin >> s >> t;
        capacity[s][t] = 1;
    }

    int n1, n2;
    cin >> n1;
    set<int> first, second;

    for (int i = 0; i < n1; i++)
    {
        int tmp;
        cin >> tmp;
        first.insert(tmp);
    }

    cin >> n2;

    for (int i = 0; i < n2; i++)
    {
        int tmp;
        cin >> tmp;
```



```
        second.insert(tmp);
    }

    cout << endl << "Maximum Matching Cardinality : " <<
    bipartiteMatch(first, second) << endl;
    cout << "Matching Graph : " << endl;

    for (int i = 0; i < n - 2; i++)
    {
        for (int j = 0; j < n - 2; j++)
            cout << flow[i][j] << ' ';
        cout << endl;
    }

    return 0;
}
========================================================================
MAXFLOW HAAMED
========================================================================
// Maximum Flow - Haamed

#include<iostream.h>
#include<cstring>
#define MAXNODE 100
#define INF 1000000000
#define MIN(a,b) (a<?b)

int n,s,t;
int cap[MAXNODE][MAXNODE];
int flow[MAXNODE][MAXNODE];
int mark[MAXNODE];

int dfs(int v,int val)
{
    if(v==t)return val;
    mark[v]=1;
    int f;
    for(int i=0;i<n;i++)

        if(!mark[i]&&flow[v][i]<cap[v][i]&&(f=dfs(i,MIN(val,cap[v][i]-f
        low[v][i]))))
        {
            flow[v][i]+=f;
            flow[i][v]-=f;
            return f;
        }
    return 0;
}

int main()
{
    //init
    memset(cap,0,sizeof cap);
    memset(flow,0,sizeof flow);
    memset(mark,0,sizeof mark);
    //input
    cin>>n>>s>>t;
```

```
    int m,a,b;
    for(cin>>m;m--;)
    {
        int c;
        cin>>a>>b>>c;
        cap[a][b]=c;
    }
    //body
    int f,sum=0;
    while(f=dfs(s,INF))
    {
        sum+=f;
        memset(mark,0,sizeof mark);
    }
    //output
    cout<<sum<<endl;
    for(int i=0;i<n;i++)
        if(mark[i])
            for(int j=0;j<n;j++)
                if(!mark[j]&&cap[i][j])
                    cout<<i<<' '<<j<<' '<<cap[i][j]<<endl;
    return 0;
}
========================================================================
POLYGON SIMILARITY
========================================================================
#include<iostream>
#include<cmath>
#include<string>
#include<algorithm>
#define EPS 1e-10
#define N(i) ((i+1)%n)
#define P(i) ((i-1+n)%n)
#define p2(a) ((a)*(a))
using namespace std;
const double PI = 3.1415926535897932385;//2 * acos(0.)
inline bool dless(double x, double y)
{
    return x <= y - EPS;
}
inline double vectorAngle(double x, double y)
{
    double r = atan2(y, x);
    if (dless(r, 0))
        r += 2 * PI;
    return r;
}

inline double angle(double x1, double y1, double x2, double y2, double
x3, double y3)
{
    double r=vectorAngle(x3-x2,y3-y2)-vectorAngle(x1-x2,y1-y2);
    if (dless(r, 0))
        r += 2 * PI;
    return r;
}
main()
```



```
{
    int n;
    double xf[10],xs[10],yf[10],ys[10];
    while(cin>>n,n)
    {
        for(int i=0;i<n;i++)
            cin>>xf[i]>>yf[i];
        for(int i=0;i<n;i++)
            cin>>xs[i]>>ys[i];
        double max1=0,max2=0;
        double len1[20],len2[10];
        double an1[20],an2[10];
        for(int i=0;i<n;i++)
        {
            double
            b1=sqrt(p2(xf[(i+1)%n]-xf[i])+p2(yf[(i+1)%n]-yf[i]));
            double
            b2=sqrt(p2(xs[(i+1)%n]-xs[i])+p2(ys[(i+1)%n]-ys[i]));
            len1[i]=b1;
            len2[i]=b2;
            max1=max(max1,b1);
            max2=max(max2,b2);

            an1[i]=angle(xf[P(i)],yf[P(i)],xf[i],yf[i],xf[N(i)],yf[N(i)
            ]);

            an2[i]=angle(xs[P(i)],ys[P(i)],xs[i],ys[i],xs[N(i)],ys[N(i)
            ]);
        }
        for(int i=0;i<n;i++)
        {
            len1[i]/=max1;
            len2[i]/=max2;
            len1[i+10]=len1[i];
            an1[i+10]=an1[i];
        }
        bool f=false;
        for(int i=0;i<n;i++)
        {
            bool f1=true;
            for(int j=0;j<n;j++)
                if(abs(len1[i+j]-len2[j])>=EPS
                    ||abs(an1[i+j]-an2[j])>=EPS)
                {
                    f1=false;
                    break;
                }
            if (f1)
            {
                f=true;
                break;
            }
        }
        if (f)
            cout<<"similar"<<endl;
        else
            cout<<"dissimilar"<<endl;
```

```
    }
    return 0;
}
=====================================================================
STRONGLY CONNECTED
=====================================================================
// Strongly Connected Components

#include <iostream>
#include <vector>

using namespace std;

#define NODES 100

// --------------- StronglyConnected Matrix

int n, t, c, sorted[NODES], vc[NODES];
int graph[NODES][NODES];

void dfsMatrix(int v, bool rev)
{
    vc[v] = rev ? c : 1;
    for (int i = 0; i < n; i++)
        if (!rev && !vc[i] && graph[v][i] ||
            rev && vc[i] == -1 && graph[i][v])
            dfsMatrix(i, rev);
    if (!rev)
        sorted[n - ++t] = v;
}

void stronglyConnectedMatrix()
{
    memset(vc, 0, n * sizeof(int)); // dfs on G
    for (int i = 0; i < n; i++)
        if (!vc[i])
            dfsMatrix(i, false);

    memset(vc, -1, n * sizeof(int));// dfs on G'(reversed)
    c = 0;
    for (int i = 0; i < n; i++)
        if (vc[sorted[i]] == -1)
        {
            dfsMatrix(i, true);
            c++;
        }
}

// --------------- StronglyConnected List

int n, t, c, sorted[NODES], vc[NODES];
vector<int> neigh[NODES], rneigh[NODES];

void dfsList(int v)
{
    vc[v] = 1;
    for (vector<int>::iterator it = neigh[v].begin(); it !=
```



```
        neigh[v].end(); it++)
        if (!vc[*it])
            dfsList(*it);
    sorted[n - ++t] = v;
}

void rDfsList(int v)
{
    vc[v] = c;
    for (vector<int>::iterator it = rneigh[v].begin(); it !=
    rneigh[v].end(); it++)
        if (vc[*it] == -1)
            rDfsList(*it);
}

void stronglyConnectedList()
{
    memset(vc, 0, n * sizeof(int)); // dfs on G
    for (int i = 0; i < n; i++)
        if (!vc[i])
            dfsList(i);

    memset(vc, -1, n * sizeof(int));// dfs on G'(reversed)
    c = 0;
    for (int i = 0; i < n; i++)
        if (vc[sorted[i]] == -1)
        {
            rDfsList(i);
            c++;
        }
}
=====================================================================
TRIE HAAMED
=====================================================================
// Trie - Haamed

#include<iostream>
#include<string>

using namespace std;

int n;
string words[130000];

struct Trie
{
    Trie *next[26];
    int mark;
}zero,*head;

Trie *insert(Trie *ptr,const char *txt)
{
    if(!txt)
        return ptr;
    if(!ptr)
    {
        ptr=new Trie;
```

```
        *ptr=zero;
    }
    if(*txt)
        ptr->next[*txt-'a']=insert(ptr->next[*txt-'a'],txt+1);
    else
        ptr->mark=1;
    return ptr;
}

int search(Trie *ptr,const char *txt,int mod)
{
    if(!ptr||!txt)
        return 0;
    if(mod&&ptr->mark)
        if(search(head,txt,0))
            return 1;
    if(*txt)
        return search(ptr->next[*txt-'a'],txt+1,mod);
    return mod?0:ptr->mark;
}

int main()
{
    n=0;
    head=NULL;
    while(cin>>words[n])
        head=insert(head,words[n++].c_str());
    for(int i=0;i<n;i++)
        if(search(head,words[i].c_str(),1))
            cout<<words[i]<<endl;
    return 0;
}
```

```
======================================================================
WMATCHING BABAK
======================================================================
// Matching + Weighted Matching - Babak

//BEGIN OF SOURCE CODE
#include <cstring>
#include <iostream>

using namespace std;

#define MAXSIZE 100
#define INF 1000000000

int match[MAXSIZE];

bool dfs(int v, int n, int adj[MAXSIZE][MAXSIZE], int mark[], int s[],
int t[])
{
    int i;
    s[v] = 1;
    mark[v] = 1;
    for (i = 0; i < n; i++)
        if (adj[v][i])
            if (t[i] = 1, match[i] == -1 || (!mark[match[i]] &&
```



```
                dfs(match[i], n, adj, mark, s, t)))
                    return match[i] = v, true;
    return false;
}

bool matching(int n, int adj[MAXSIZE][MAXSIZE], int s[], int t[], int
sa[])
{
    int i;
    int max = 0;
    int mark[MAXSIZE];
    memset(mark, 0, sizeof(mark));
    for (i = 0; i < n; i++)
        if (!sa[i] && !mark[i] && dfs(i, n, adj, mark, s, t))
        {
            memset(mark, 0, sizeof(mark));
            sa[i] = 1;
        }
    for (i = 0; i < n; i++)
        if (!sa[i])
            return false;
    return true;
}

void weighted(int n, int m, int weight[MAXSIZE][MAXSIZE])
{
    int i, j;
    int size = 0;
    int sa[MAXSIZE], s[MAXSIZE], t[MAXSIZE];
    int cover[2][MAXSIZE];
    int adj[MAXSIZE][MAXSIZE];
    memset(match, -1, sizeof(match));
    memset(sa, 0, sizeof(sa));
    memset(adj, 0, sizeof(adj));
    if (m > n)
        n = m;
    for (i = 0; i < n; i++)
    {
        int index = 0;
        for (j = 1; j < n; j++)
            if (weight[i][index] < weight[i][j])
                index = j;
        cover[1][i] = 0;
        cover[0][i] = weight[i][index];
        adj[i][index] = 1;
    }

    while (!matching(n, adj, s, t, sa))
    {
        memset(s, 0, sizeof(s));
        memset(t, 0, sizeof(t));
        matching(n, adj, s, t, sa);
        int min = INF;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                if (s[i] && !t[j] && !adj[i][j])
                    if (cover[0][i] + cover[1][j] - weight[i][j] < min)
```

```
                        min = cover[0][i] + cover[1][j] - weight[i][j];
        for (i = 0; i < n; i++)
            if (s[i])
                cover[0][i] -= min;
        for (i = 0; i < n; i++)
            if (t[i])
                cover[1][i] += min;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                if ((s[i] && !t[j]) || (!s[i] && t[j]))
                    if ((cover[0][i] + cover[1][j]-weight[i][j])==0)
                        adj[i][j] = 1;
                    else
                        adj[i][j] = 0;
    }
}
//END OF SOURCE CODE

main()
{
    int i, j, tn;
    int n, m, sum;
    int weight[MAXSIZE][MAXSIZE];
    for (cin >> tn; tn--;)
    {
        memset(weight, 0, sizeof weight);
        cin >> n >> m;
        for (i = 0; i < n; i++)
            for (j = 0; j < m; j++)
                cin >> weight[i][j];

        weighted(n, m, weight);

        sum = 0;
        for (i = 0; i < max(n, m); i++)
        {
            if (i < m && match[i] < n)
            {
                cout << match[i] << ' ' << i << endl;
                sum += weight[match[i]][i];
            }
        }
        cout << sum << endl;
    }
    return 0;
}
```



```
pi =
3.141592653589793238462643383279502884197169399375105820974944592307816
40628620899862803482534211706798214808651328230664709384460955058223172
```