Problem
Chalk has been actively traveling the world with his friends taking pictures in all the coolest places. Most recently, h
made his way to Europe, where he studied the history of napkin folding. Ever since, Chalk has been collecting a wi
e variety of napkins to practice the art of napkin folding.

Chalk's napkins can be defined as simple polygons. A simple polygon is a polygon in which no edges intersect exce
t for adjacent edges which meet at their shared vertex. Each vertex of the polygon is on exactly two edges.

Chalk folds his napkins by first drawing a folding pattern on them. A folding pattern is a set of K-1 line segments w
ich are drawn on the napkin. Each line segment connects two points with rational coordinates on the border of the po
ygon defining the napkin and is fully contained in the polygon. No two line segments in a folding pattern may touch
r overlap, except possibly at common endpoints. A folding pattern of K-1 line segments splits the napkin into K poly
onal regions. Two points are in the same region if there exists some continuous line (not necessarily straight) betwee
them which does not intersect any edge of the polygon or any line segment in the folding pattern — even at endpoin
s.

Chalk is only interested in neat folding patterns. A folding pattern is neat if any two regions that are adjacent to the s
me folding line segment F are symmetric with respect to F. This means that folding the napkin along that line segme
t would result in the two regions lining up perfectly.

The following picture illustrates a neat folding pattern with K=8 regions.

Chalk has been successfully folding his collection of napkins using neat folding patterns. But he has some napkins i
his collection that he has not been able to find a neat folding pattern for. For each of those napkins, Chalk needs you
help to find a neat folding pattern with K regions or determine that no such neat folding pattern exists.

Input
The first line of the input gives the number of test cases, T. T test cases follow. Each test case starts with a line conta
ning two integers N and K: the number of points in the polygon defining Chalk's napkin and the number of regions t
split the napkin into with a neat folding pattern containing K-1 line segments.

The polygon defining the napkin is represented as a list of the N vertices, as encountered when traveling along the p
rimeter of the polygon in the clockwise direction, with the first vertex being chosen arbitrarily. The next N lines repr
sent that list. The i-th of these contains two integers Xi and Yi, indicating that the i-th point is located at (Xi, Yi) in t
o-dimensional space.

Output
For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is P
SSIBLE if it is possible to make a neat folding pattern with K regions and IMPOSSIBLE otherwise.

If it is possible to make a neat folding pattern with K regions, output K-1 more lines listing the segments of a neat fo
ding pattern with K regions, in any order. Each line should represent a different segment as Ax Ay Bx By, where (A
, Ay) and (Bx, By) are the two endpoints of the segment, in any order. Each of Ax, Ay, Bx, and By should be in the
orm N/D where N and D are positive integers (written with no leading zeroes) sharing no common prime factors, an
representing the rational number N/D. There must be no whitespace between N and /, or between / and D.

Limits
Time limit: 60 seconds per test set.
Memory limit: 1GB.
$1 \leq T \leq 100$.
$3 \leq N \leq 200$.

1 ≤ Xi ≤ 1000, for all i.
1 ≤ Yi ≤ 1000, for all i.
The N points are given in clockwise order.
No two adjacent edges of the polygon are collinear.
The polygon is a simple polygon with strictly positive area.
No two edges intersect except for adjacent edges at their shared endpoint.

Test set 1 (Visible)
K = 2.

Test set 2 (Hidden)
2 ≤ K ≤ 10.

Sample

Input 1

Output 1

```
4
4 2
1 1
1 2
2 2
2 1
3 2
1 1
1 2
2 1
8 2
1 3
3 5
5 5
4 4
7 3
5 1
4 2
3 1
8 2
1 3
3 5
4 4
5 5
7 3
5 1
4 2
3 1
```

```
Case #1: POSSIBLE
1/1 2/1 2/1 1/1
Case #2: POSSIBLE
1/1 1/1 3/2 3/2
```

```
Case #3: IMPOSSIBLE
Case #4: POSSIBLE
1/1 3/1 7/1 3/1
```

Input 2

Output 2

```
1
10 8
4 1
3 1
2 2
2 3
1 3
1 4
2 4
3 3
3 2
4 2
```

```
Case #1: POSSIBLE
3/1 1/1 4/1 2/1
3/1 1/1 3/1 2/1
2/1 2/1 3/1 2/1
2/1 2/1 3/1 3/1
2/1 3/1 3/1 3/1
2/1 3/1 2/1 4/1
1/1 3/1 2/1 4/1
```

Note: Sample 2 is not valid for Test set 1. Only Sample 1 will be tested prior to running Test set 1 (the same way sa ples normally are). Moreover, Sample 2 will not be tested prior to running Test set 2.

For Sample Case #1, a neat folding pattern with K=2 can be drawn using any of the 4 dashed lines:

For Sample Case #2, a neat folding pattern with K=2 can be drawn as follows:

For Sample Case #3, there are no neat folding patterns:

For Sample Case #4, there are two possible neat folding patterns with K=2:

For the test set 2 sample case, a neat folding pattern with K=8 can be drawn as follows:

Solution:

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <bits/stdc++.h>

using namespace std;

typedef long long int64;
typedef unsigned long long uint64;
#define two(X) (1<<(X))
#define twoL(X) (((int64)(1))<<(X))
#define contain(S,X) (((S)&two(X))!=0)
#define containL(S,X) (((S)&twoL(X))!=0)
const double pi=acos(-1.0);
const double eps=1e-11;
template<class T> inline void ckmin(T &a,T b){if(b<a) a=b;}
template<class T> inline void ckmax(T &a,T b){if(b>a) a=b;}
template<class T> inline T sqr(T x){return x*x;}
typedef pair<int,int> ipair;
#define SIZE(A) ((int)A.size())
#define LENGTH(A) ((int)A.length())
#define MP(A,B) make_pair(A,B)
#define PB(X) push_back(X)
#define FOR(i,a,b) for(int i=(a);i<(b);++i)
#define REP(i,a) for(int i=0;i<(a);++i)
#define ALL(A) A.begin(),A.end()
using VI=vector<int>;

struct Point
{
 Point(int x = 0, int y = 0) : x(x), y(y) {}
 int x = 0;
 int y = 0;
};

int outer(const Point& a,const Point& b,const Point& c)
{
 return (b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y);
}

int inner(const Point& a,const Point& b,const Point& c)
{
 return (b.x-a.x)*(c.x-a.x)+(b.y-a.y)*(c.y-a.y);
}

int gcd(int a,int b)
{
 return b==0?a:gcd(b,a%b);
}
void reduce(int &a,int &b)
```

```
{
 int d=gcd(a,b);
 a/=d;
 b/=d;
}
void output(int a,int b)
{
 reduce(a,b);
 printf("%d/%d",a,b);
}

void solve_easy(vector<Point> a)
{
 int n=SIZE(a);
 while (1)
 {
  n=SIZE(a);
  int ok=1;
  REP(i,n) if (outer(a[(i+n-1)%n],a[i],a[(i+1)%n])==0) { a.erase(a.begin()+i); ok=0; break; }
  if (ok) break;
 }
 REP(i,n) a[i].x*=2,a[i].y*=2;
 vector<Point> b=a;
 REP(i,n) b.push_back(Point((b[i].x+b[(i+1)%n].x)/2,(b[i].y+b[(i+1)%n].y)/2));
 for (const Point& p1:b) for (const Point& p2:b) if (p1.x<p2.x || p1.x==p2.x && p1.y<p2.y)
 {
  vector<Point> w;
  REP(i,n)
  {
   int c=outer(p1,p2,a[i]);
   int d=inner(p1,p2,a[i]);
   w.push_back(Point(c,d));
  }
  set<pair<ipair,ipair>> s1,s2;
  int ok=1;
  REP(i,n)
  {
   const auto& w1=w[i];
   const auto& w2=w[(i+1)%n];
   if (w1.x>0 && w2.x<0 || w1.x<0 && w2.x>0)
   {
    if (w1.y!=w2.y) { ok=0; break; }
    continue;
   }
   if (w1.x>=0 && w2.x>=0)
   {
    ipair p1=MP(w1.x,w1.y),p2=MP(w2.x,w2.y);
    if (p1>p2) swap(p1,p2);
    s1.insert(MP(p1,p2));
   }
   if (w1.x<=0 && w2.x<=0)
   {
    ipair p1=MP(-w1.x,w1.y),p2=MP(-w2.x,w2.y);
    if (p1>p2) swap(p1,p2);
    s2.insert(MP(p1,p2));
   }
  }
  if (ok && s1==s2)
  {
   printf("POSSIBLE\n");
   output(p1.x,2);
   printf(" ");
   output(p1.y,2);
   printf(" ");
   output(p2.x,2);
   printf(" ");
   output(p2.y,2);
   printf("\n");
   return;
  }
 }
 printf("IMPOSSIBLE\n");
}

void solve_hard()
{
 printf("SKIP HARD\n");
}

int main()
{
#ifdef _MSC_VER
 freopen("input.txt","r",stdin);
#endif
 std::ios::sync_with_stdio(false);
 int testcase;
 cin>>testcase;
 for (int case_id=1;case_id<=testcase;case_id++)
 {
  printf("Case #%d: ",case_id);
  int n,k;
  cin>>n>>k;
  vector<Point> a(n);
  REP(i,n) cin>>a[i].x>>a[i].y;
  if (k==2)
   solve_easy(a);
  else
   solve_hard();
 }
 return 0;
}
```