Pontok összekötése zárt poligonná

Adott a síkon n pont, amelyek nem esnek egy egyenesre. Kössünk össze pontpárokat egyenenes szakaszokkal úgy, hogy egy zárt, nem-metsző törtvonalat, poligont kapjunk. Egy ilyen poligon megadható a pontok egy felsorolásával, a felsorolásban az egymást követő pontpárokat, továbbá az utolsót az elsővel kötjük össze.

Bemenet

A `standard bemenet` első sora egy egész számot tartalmaz, a pontok n számát ($1 \leq n \leq 100\,000$). A következő n sor mindegyike két egész számot tartalmaz, egy pont **x** és **y** koordinátáját ($-100\,000 \leq x, y \leq 100\,000$). A pontokat az $1, \ldots, n$ számokkal azonosítjuk, az állomány $i+1$-edik sora tartalmazza az i-edik pont koordinátáit.

Kimenet

A `standard kimenet` első sora a pontok azonosítóinak egy olyan felsorolását tartalmazza (egy-egy szóközzel elválasztva), hogy ha az egymást követő pontpárokat, és az utolsót az elsővel összekötjük egyenes szakasszal, akkor zárt, nem-metsző poligont kapunk. Több megoldás esetén bármelyik megadható.

Példa

Bemenet

```
6
2 0
1 4
0 2
3 2
2 4
2 6
```

Kimenet

```
3 1 4 5 6 2
```

Korlátok

Időlimit: 0.5 mp.
Memórilimit: 32 MiB
Pontozás: a tesztesetek 40%-ában $n < 200$

Megoldás

Három lépésben határozhatjuk meg a pontok megfelelő sorrendetjét:

1. lépés

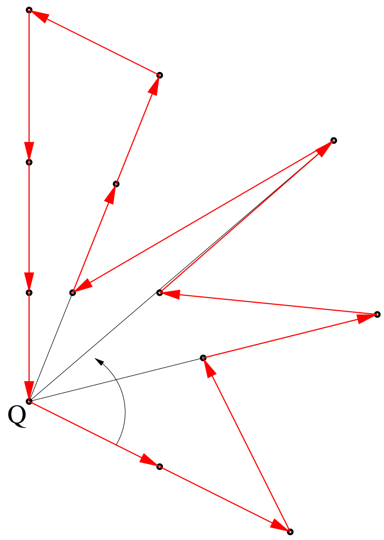
Határozzuk meg a ponthalmaz bal alsó Q sarokpontját.
A $P = \{p_1, \ldots, p_n\}$ ponthalmaz (bal alsó) sarokpontja a legkisebb x-koordinátájú pont, ha több ilyen van, akkor ezek közül a legkisebb y-koordinátájú pont.

2. lépés

Rendezzük a ponthalmazt a Q sarokpontra vonatkozó szög szerint. Ha az A és B pont azonos szögben látszik Q-ból, akkor az legyen előbb, amelyik közelebb van Q-hoz.
Az így rendezett sorrendet jelölje $q_1, \ldots, q_n$

3. lépés

Legyen j a legnagyobb olyan index, hogy a Q, q_j és q_n pontok nem esnek egy egyenesre. Ilyen biztosan van, mert a bemeneti feltétel szerint a pontok nem esnek egy egyenesre. Ekkor egy helyes sorrend az alábbi:
$q_1, q_2, \ldots, q_j, q_n, \ldots, q_{j+1}$
Hogyan valósítható meg a fenti három lépés?



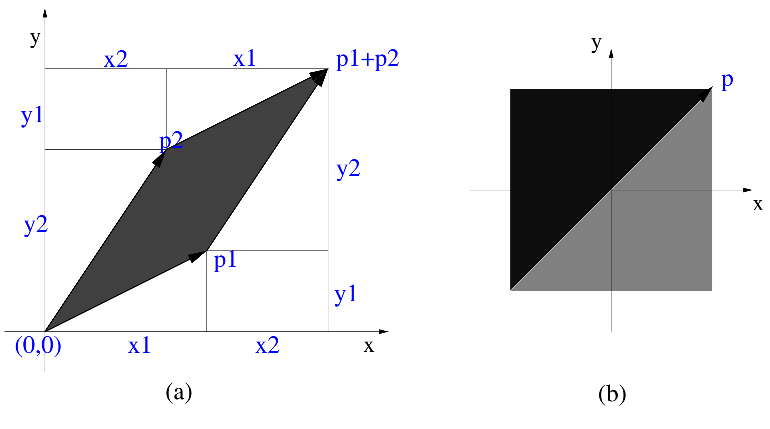
1. ábra.

A rendezés megvalósításához pontok viszonyának meghatározása alapján juthatunk, ami a legalapvetőbb algoritmikus-geometriai művelet.

Két pont viszonya

Először tekintsük két pont viszonyának meghatározását: adott a síkon két pont, $p1$ és $p2$. Döntsük el, hogy az alábbi 3 lehetőség közül melyik igaz?

1. az origóból nézve $p2$ az órajárással ellentétes irányban van $p1$-hez képest
2. az origó, $p1$ és $p2$ egy egyenesre esnek
3. az origóból nézve $p2$ az órajárással megegyező irányban van $p1$-hez képest



2. ábra.

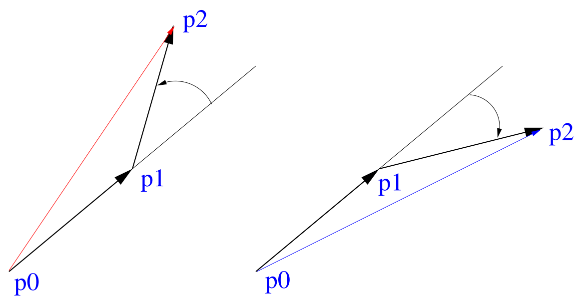
A kérdés megválaszolásához tekintsük a 2.a ábrán látható $(0,0)$, $p1$, $p2$ és $p1+p2$ pontok által meghatározott paralelogramma előjeles területét, amit $p1$ és $p2$ keresztszorzatának neveznek, és $p1 \times p2$-vel jelölünk. A terület:

$$\begin{aligned} T(x1+x2)(y1+y2) - x1y1 - x2y2 - x2y1 - x2y1 &= \\ x1y1 + x1y2 + x2y1 + x2y2 - x1y1 - x2y2 - x2y1 - x2y1 &= x1y2 - x2y1 \end{aligned}$$

$$p1 \times p2 = x1y2 - x2y1 = -p2 \times p1$$

$p1 \times p2 > 0 \Leftrightarrow p2$ az órajárással ellentétes irányban van $p1$-hez képest
$p1 \times p2 = 0 \Leftrightarrow$ a $(0,0)$, $p1$ és $p2$ pontok egy egyenesre estnek (kollineárisak)
$p1 \times p2 < 0 \Leftrightarrow p2$ az órajárással megegyező irányban van $p1$-hez képest.

Három pont viszonya



3. ábra. Csatlakozó szakaszok viszonya

Adott a síkon három pont, $p0$, $p1$ és $p2$. Döntsük el, hogy az alábbi három lehetőség közül melyik teljesül:

1. a $p0$ pontból nézve $p2$ az órajárással ellentétes igányban van $p1$-hez képest
2. a $p0$ $p1$ és $p2$ egy egyenesre esnek
3. a $p0$ pontból nézve $p2$ az órajárással megegyező igányban van $p1$-hez képest

Másképpen fogalmazva, ha $\overrightarrow{p0,p1}$ és a $\overrightarrow{p1,p2}$ csatlakozó irányított szakaszokat tekintjük, akkor az a kérdés, hogy $\overrightarrow{p1,p2}$ merre fordul $\overrightarrow{p0,p1}$-hez képest.
A válasz a $(p1-p0) \times (p2-p0)$ keresztszorzat előjele alapján megadható:
$(p1-p0) \times (p2-p0) > 0$ $\overrightarrow{p1,p2}$ balra fordul,
$(p1-p0) \times (p2-p0) = 0$ $\overrightarrow{p0,p1}$ és a $\overrightarrow{p1,p2}$ kollineárisak,
$(p1-p0) \times (p2-p0) < 0$ $\overrightarrow{p1,p2}$ jobbra fordul.
Definiáljuk a `Fordul(p0,p1,p2)` műveletet úgy, hogy annak értéke a $(p1-p0) \times (p2-p0)$ előjele legyen.

Megvalósítás C++ nyelven

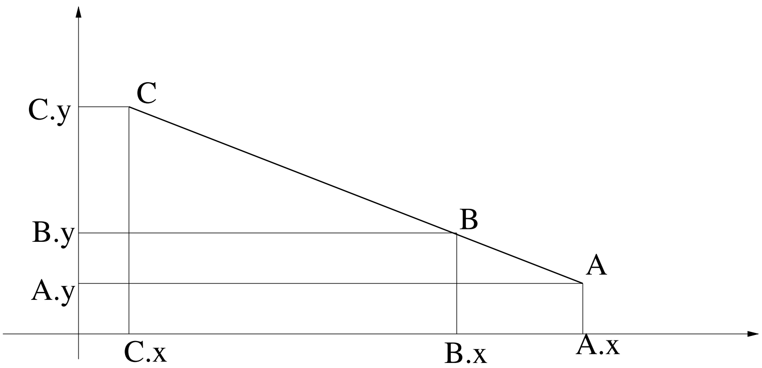
Mivel a keresztszorzat értéke nagy lehet, ezért a kiszámítást 64-bites aritmetikában kell végyezni. Ez automatikusan megvalósul, ha pontok koordinátai eleve 64-bites egész típusú.

```
typedef   struct {
   long long   x,y;
   long   azon;
} Pont;

int Fordul(Pont p0, Pont p1, Pont p2){
   long long Kereszt=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
   if (Kereszt <0)
      return -1;
   else if (Kereszt >0)
```

```
        return 1;
    else
        return  0;
}
```

A pontok rendezésének meghatározásához el kell tudni dönteni, hogy ha három pont egy egyenesre esik, akkor melyik van középpen. Tehát legyen `Kozte(A,B,C)` művelet értéke igaz akkor és csak akkor, a `B` pont az `A` és `C` pont között van. Feltesszük, hogy a három pont kollineáris.
A megvalósítás az alábbi ábrán szemléltethetően a koordináta értékek alapján egyszerűen kifejezhető a feltétel.



4. ábra. Három kollineáris pont viszonya

$$\begin{aligned}
|B.x - A.x| &\leq |C.x - A.x| \text{ és} \\
|C.x - B.x| &\leq |C.x - A.x| \text{ és} \\
|B.y - A.y| &\leq |C.y - A.y| \text{ és} \\
|C.y - B.y| &\leq |C.y - A.y|
\end{aligned}$$

Most már a sarokpont szerinti rendezés relációját meg tudjuk adni a `Fordul` és a `Kozte` műveletekkel: a $p1$ pont akkor és csak akkor előzi meg a $p2$ pontot, ha

$$Fordul(Q, p1, p2) > 0 \text{ vagy } Fordul(Q, p1, p2) = 0 \text{ és } Kozte(Q, p1, p2)$$

Fontos megjegyezni, hogy ha a pontok koordinátái egész számok, akkor mindkét alapművelet (Fordul és Kozte) kiszámítható egész aritmetikában, nem kell lebegőpontos aritmetikát hasznáélni (és nem is használjunk!).

Megvalósítás C++ nyelven

```
#include <iostream>
#include <algorithm>
#include <limits.h>
#define  maxN  100000
using  namespace  std ;

typedef  struct {
    long long  x,y;
    long  azon;
} Pont;
Pont  P[maxN];
Pont  Q;    // sarokpont
int Fordul(Pont A, Pont B, Pont C){
/*
```

```
Kimenet: +1 ha A->B->C balra fordul,
          0  ha A--B--C kollineárisak,
         -1  ha A->B->C jobbra fordul.
*/
   long long Kereszt=(B.x-A.x)*(C.y-A.y)-(C.x-A.x)*(B.y-A.y);
   if (Kereszt <0)
      return -1;
   else if (Kereszt >0)
      return 1;
   else
      return  0;
}
bool Kozte(Pont A, Pont B, Pont C){
//Bemenet: A-B-C kolineáris
//Kimenet: A és C között van B
   return abs(B.x-A.x) <= abs(C.x-A.x) &&
          abs(C.x-B.x) <= abs(C.x-A.x) &&
          abs(B.y-A.y) <= abs(C.y-A.y) &&
          abs(C.y-B.y) <= abs(C.y-A.y) ;
}
bool SarokRend(Pont A, Pont B){
//Globális: Q
   return Fordul(Q,A,B)>0 || Fordul(Q,A,B)==0 && Kozte(Q,A,B);
}

int   main(){
   int   n,x,y;
   cin>>n;
   Q.x=INT_MAX;
   for( int i=0; i<n; i++){
      cin>>x>>y;
      P[i].azon=i+1;
      P[i].x=x; P[i].y=y;
      if (x<Q.x || x==Q.x  &&  y<Q.y)
         Q=P[i];
   }
   sort(begin(P), begin(P)+n, SarokRend);
   int j=n-2;
   while (j>0  &&  Fordul(Q, P[n-1], P[j])==0) j--;
   for (int i=0; i<=j; i++)
      cout<<P[i].azon<<"␣";
   for (int i=n-1; i>j; i--)
      cout<<P[i].azon<<"␣";

   return 0;
}
```