ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

THUẬT TOÁN ỨNG DỤNG

THUẬT TOÁN HÌNH HỌC

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Công thức cơ bản
- Tìm bao lồi
- Kiểm tra 1 điểm nằm trong đa giác lồi

Công thức cơ bản

- Điểm

```
struct Point {
    double x, y;
};
```

- Đường thẳng $ax + by + c = 0$

```
struct Line {
    double a, b, c;
};
```

- Vector $\overrightarrow{AB}$ của hai điểm $A(x_A, y_A)$ và $B(x_B, y_B)$

$$\overrightarrow{AB} = (x_B - x_A, y_B - y_A)$$

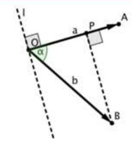
Công thức cơ bản

- 3 điểm $A(x_A, y_A)$, $B(x_B, y_B)$ và $C(x_C, y_C)$ thẳng hàng khi:
- $\overrightarrow{AB} = k \times \overrightarrow{AC}$
- $x_B - x_A = k \times (x_C - x_A)$
- $y_B - y_A = k \times (y_C - y_A)$
- Để tránh phép chia cho 0: $(x_A - x_B) \times (y_A - y_C) = (x_A - x_C) \times (y_A - y_B)$

Công thức cơ bản

- Tích vô hướng của $\overrightarrow{OA}(x_a, y_a)$ và $\overrightarrow{OB}(x_b, y_b)$
- $\overrightarrow{OA} \cdot \overrightarrow{OB} = x_a x_b + y_a y_b = \left|\overrightarrow{OA}\right| \left|\overrightarrow{OB}\right| \cos\alpha = \sqrt{x_a^2 + y_a^2}\sqrt{x_b^2 + y_b^2} \cos\alpha$,

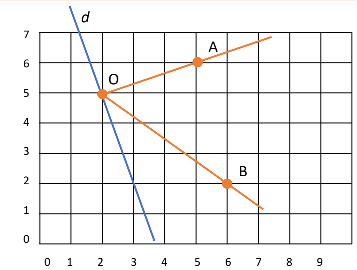
→ $\cos\alpha = \frac{x_a x_b + y_a y_b}{\sqrt{x_a^2+y_a^2}\sqrt{x_b^2+y_b^2}}$



- Vẽ đường thẳng d vuông góc với $\overrightarrow{OA}$, dựa vào $\cos\alpha$ ta có:
  - Nếu $\overrightarrow{OA} \cdot \overrightarrow{OB} > 0$ thì A và B cùng phía so với đường thẳng d
  - Nếu $\overrightarrow{OA} \cdot \overrightarrow{OB} = 0$ thì B nằm trên đường thẳng d
  - Nếu $\overrightarrow{OA} \cdot \overrightarrow{OB} < 0$ thì A và B khác phía so với đường thẳng d

Công thức cơ bản

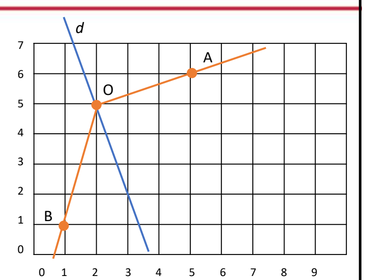
```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
double dist(Point &a, Point &b) {
    long long x = a.x - b.x; long long y = a.y - b.y;
    return sqrt(1LL * x*x + 1LL * y*y);
}
long long dot_product(Point &O, Point &A, Point &B) {
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * xb + 1LL * ya * yb;
}
int main(){
    Point O(2,5); Point A(5,6);  Point B(6,2);
    double cos = dot_product(O,A,B)*1.0/(dist(O,A)*dist(O,B));
    cout << "cos = " << cos << endl;
}
```




```
cos = 0.56921
```


Công thức cơ bản

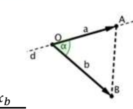
```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
double dist(Point &a, Point &b) {
    long long x = a.x - b.x; long long y = a.y - b.y;
    return sqrt(1LL * x*x + 1LL * y*y);
}
long long dot_product(Point &O, Point &A, Point &B) {
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * xb + 1LL * ya * yb;
}
int main(){
    Point O(2,5); Point A(5,6);  Point B(1,1);
    double cos = dot_product(O,A,B)*1.0/(dist(O,A)*dist(O,B));
    cout << "cos = " << cos << endl;
}
```




```
cos = -0.536875
```


Công thức cơ bản

- Tích có hướng của $\overrightarrow{OA}(x_a, y_a)$ và $\overrightarrow{OB}(x_b, y_b)$
- $\overrightarrow{OA} \times \overrightarrow{OB} = x_a y_b - y_a x_b = \left|\overrightarrow{OA}\right| \left|\overrightarrow{OB}\right| \sin\alpha$

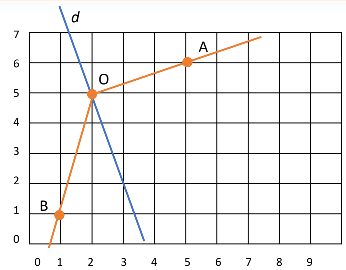


$$\overrightarrow{OA} \times \overrightarrow{OB} = x_a y_b - y_a x_b = \sqrt{x_a^2 + y_a^2}\sqrt{x_b^2 + y_b^2} \sin\alpha , \qquad \sin\alpha = \frac{x_a y_b - y_a x_b}{\sqrt{x_a^2+y_a^2}\sqrt{x_b^2+y_b^2}}$$

- Vẽ đường thẳng d trùng với $\overrightarrow{OA}$, dựa vào $\sin\alpha$ ta có:
  - Nếu $\overrightarrow{OA} \times \overrightarrow{OB} > 0$ thì B ở bên trái so với đường thẳng d (hướng xoay từ tia $\overrightarrow{OA}$ đến tia $\overrightarrow{OB}$ là **ngược** chiều kim đồng hồ)
  - Nếu $\overrightarrow{OA} \times \overrightarrow{OB} = 0$ thì B nằm trên đường thẳng d
  - Nếu $\overrightarrow{OA} \times \overrightarrow{OB} < 0$ thì B nằm bên phải so đường thẳng d (hướng xoay từ tia $\overrightarrow{OA}$ đến tia $\overrightarrow{OB}$ là **cùng** chiều kim đồng hồ)
- Trị tuyệt đối của tích có hướng của hai vector $\overrightarrow{OA}$ và $\overrightarrow{OB}$ bằng hai lần diện tích tam giác OAB.

Công thức cơ bản

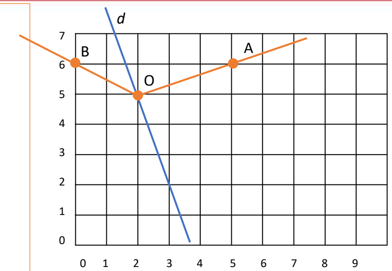
```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
double dist(Point &a, Point &b) {
    long long x = a.x - b.x; long long y = a.y - b.y;
    return sqrt(1LL * x*x + 1LL * y*y);
}
long long cross_product(Point &O, Point &A, Point &B) {
    //tich vo huong 2 vector (O,A).(O,B)
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * yb - 1LL * ya * xb;
}
int main(){
    Point O(2,5); Point A(5,6); Point B(1,1);
    double sin = cross_product(O,A,B)*1.0/(dist(O,A)*dist(O,B));
    cout << "sin = " << sin << endl;
}
```




```
sin = -0.843661
```


Công thức cơ bản

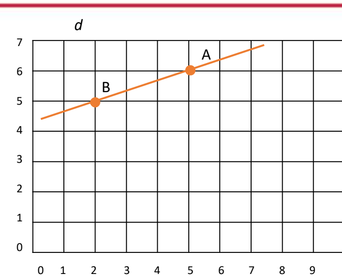
```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
double dist(Point &a, Point &b) {
    long long x = a.x - b.x; long long y = a.y - b.y;
    return sqrt(1LL * x*x + 1LL * y*y);
}
long long cross_product(Point &O, Point &A, Point &B) {
    //tich vo huong 2 vector (O,A).(O,B)
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * yb - 1LL * ya * xb;
}
int main(){
    Point O(2,5); Point A(5,6); Point B(0,6);
    double sin = cross_product(O,A,B)*1.0/(dist(O,A)*dist(O,B));
    cout << "sin = " << sin << endl;
}
```




```
sin =  0.707107
```

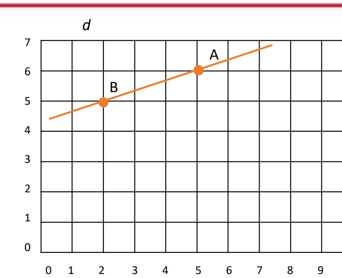

Công thức cơ bản

- Gọi đường thẳng l_{AB} đi qua hai điểm $A(x_A, y_A)$ và $B(x_B, y_B)$
  - Vector $\overrightarrow{AB} = (x_B - x_A, y_B - y_A)$
  - Vector vuông góc $\vec{v} = (y_A - y_B, x_B - x_A)$
  - Mọi điểm $P(x, y)$ nằm trên đường thẳng l_{AB} thì có $\overrightarrow{AP} \cdot \vec{v} = 0$
    $$(x - x_A)(y_A - y_B) + (y - y_A)(x_B - x_A) = 0$$
  - Phương trình đường thẳng l_{AB} là $ax + by + c = 0$
    $$a = y_A - y_B$$
    $$b = x_B - x_A$$
    $$c = x_A y_B - y_A x_B$$
  - Ví dụ: đường thẳng đi qua A(5, 6) và B(2, 5) có phương trình: (6-5)x + (2-5)y + (5x5 – 6x2) = 0 hay x – 3y + 13 = 0



Công thức cơ bản

```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
struct Line{
    int a,b,c;
};
void makeLine(Point& A, Point& B, Line& L){
    L.a = A.y - B.y;
    L.b = B.x - A.x;
    L.c = A.x*B.y - A.y*B.x;
}
int main(){
    Point A(5,6);  Point B(2,5);  Line L;
    makeLine(A,B,L);
    cout << L.a << "x" << " + " << L.b << "y + " << L.c << " = 0“;
}
```

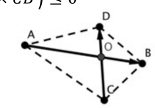


Công thức cơ bản

- Đường thẳng l_{AB} đi qua hai điểm $A(x_A, y_A)$ và $B(x_B, y_B)$
  - $(y_A - y_B)x + (x_B - x_A)y + (x_A y_B - y_A x_B) = 0$ (1)
- Khoảng cách từ điểm $C(x_C, y_C)$ đến đường thẳng l_{AB} là d
- Diện tích tam giác ABC là:
  - $S_{ABC} = \frac{|\overrightarrow{AB} \times \overrightarrow{AC}|}{2} = \frac{|\overrightarrow{AB}| \times d}{2} \Rightarrow d = \frac{|\overrightarrow{AB} \times \overrightarrow{AC}|}{|\overrightarrow{AB}|}$
- $\frac{|\overrightarrow{AB} \times \overrightarrow{AC}|}{|\overrightarrow{AB}|} = \frac{|(x_B - x_A)(y_C - y_A) - (y_B - y_A)(x_C - x_A)|}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}}$ (2)
- Từ (1) và (2), nếu đường thẳng l có phương trình là $ax + by + c = 0$ thì khoảng cách từ điểm $P(x_P, y_P)$ xuống đường thẳng l sẽ là:
- $dist(l, P) = \frac{|ax_P + by_P + c|}{\sqrt{a^2 + b^2}}$

Công thức cơ bản

- Hai đoạn thẳng AB và CD cắt nhau khi:
  - C và D không nằm cùng phía so với đường thẳng l_{AB}: $(\overrightarrow{AB} \times \overrightarrow{AC})\left(\overrightarrow{AB} \times \overrightarrow{AD}\right) \leq 0$
  - A và B không nằm cùng phía so với đường thẳng l_{CD}: $(\overrightarrow{CD} \times \overrightarrow{CA})\left(\overrightarrow{CD} \times \overrightarrow{CB}\right) \leq 0$
- Tọa độ giao điểm $O(x_O, y_O)$ của hai đường thẳng:
  - Phương trình đường thẳng l_{AB}: $a_1 x + b_1 y + c_1 = 0$
  - Phương trình đường thẳng l_{CD}: $a_2 x + b_2 y + c_2 = 0$



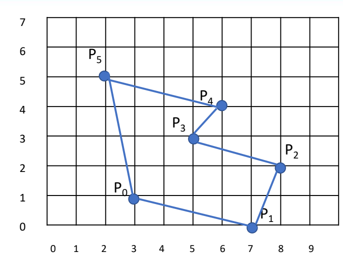
$$x_O = \frac{c_2 b_1 - c_1 b_2}{a_1 b_2 - a_2 b_1}$$
$$y_O = \frac{c_1 a_2 - c_2 a_1}{a_1 b_2 - a_2 b_1}$$

Công thức cơ bản

```
struct Point {
    int x, y;
    Point(int x, int y) : x(x), y(y) {}
};
struct Line{
    int a,b,c;
};
void makeLine(Point& A, Point& B, Line& L){
    L.a = A.y - B.y;    L.b = B.x - A.x;    L.c = A.x*B.y - A.y*B.x;
}
void intersection(Line& L1, Line& L2){
    double x = (L2.c*L1.b - L1.c*L2.b)*1.0/(L1.a*L2.b - L2.a*L1.b);
    double y = (L1.c*L2.a - L2.c*L1.a)*1.0/(L1.a*L2.b - L2.a*L1.b);
    cout << "Giao diem = (" << x << "," << y << ")" << endl;
}
int main(){
    Point A(3,1);  Point B(6,4);  Line LAB;
    Point C(2,5); Point D(7,0); Line LCD;
    makeLine(A,B,LAB); makeLine(C,D,LCD);    intersection(LAB,LCD);
}
```


Công thức cơ bản

- Một đa giác được tạo thành bởi 1 đường gấp khúc không tự cắt với các cạnh P_0P_1, P_1P_2, P_2P_3, ..., $P_{n-1}P_0$
- Trong đó đỉnh P_i có tọa độ (x_i, y_i)
  - Cố định một đỉnh P_0
  - Tính tổng S:
  - $S = \sum_{i=1}^{n-2} \overrightarrow{P_0P_i} \times \overrightarrow{P_0P_{i+1}}$
  - Diện tích của đa giác là $\frac{|S|}{2}$



Tìm bao lồi (P.08.13.05)

- Cho một tập n điểm P_i, tìm đa giác lồi có diện tích nhỏ nhất chứa tất cả các điểm đã cho.
- Dữ liệu
  - Dòng 1: chứa số nguyên dương *n* (3 <= *n* <= 100000)
  - Dòng *i*+1 (*i* = 1, 2, …, *n*): chứa 2 số nguyên x_i, y_i là tọa độ của điểm P_i (-1000 <= x_i, y_i <= 1000)
- Kết quả
  - Dòng 1: ghi số nguyên dương *m* là số điểm (đỉnh của đa giác) trên bao lồi tìm được
  - Dòng *i* + 1 (*i* = 1, 2, …, *m*): ghi 2 số nguyên là tọa độ của điểm thứ *i* của bao lồi tìm được

stdin	stdout
6 4 5 5 3 5 6 2 5 8 7 3 7	4 5 3 8 7 3 7 2 5

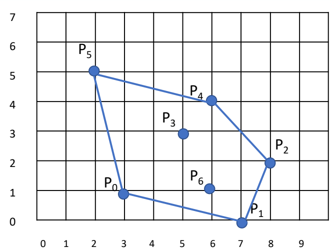
Tìm bao lồi (P.08.13.05)

- Cho một tập n điểm P_i, tìm đa giác lồi có diện tích nhỏ nhất chứa tất cả các điểm đã cho.
- Thuật toán **Graham Scan**
  - Tìm điểm bên trái dưới nhất – là điểm chắc chắn thuộc bao lồi. Cho điểm này thành điểm P_0
  - Sắp xếp $n - 1$ điểm còn lại theo góc với gốc là điểm P_0.
  - Tạo một stack rỗng S và thêm P_0 và P_1 vào S
  - Với $n - 2$ điểm còn lại lặp lại các bước sau với từng điểm P_i:
    - Lặp đi lặp lại việc xóa điểm ở đỉnh của stack S chừng nào **CCW** của 3 điểm sau không **dương**:
      - (a) Điểm kề (trong stack) với điểm ở đỉnh của stack S
      - (b) Điểm ở đỉnh của stack S
      - (c) Điểm P_i
    - Thêm P_i vào stack S
  - Kết thúc ta được bao lồi là các đỉnh theo thứ tự cùng chiều kim đồng hồ khi lấy từ stack S ra.

Tìm bao lồi (P.08.13.05)

- Hàm **counterclockwise** xác định chiều quay tia OA đến OB
- **CCW**(O, A, B) được định nghĩa bằng

$$\begin{cases} 0, & \text{nếu } \overrightarrow{OA} \times \overrightarrow{OB} = 0 \\ -1, & \text{nếu } \overrightarrow{OA} \times \overrightarrow{OB} < 0 \\ +1, & \text{nếu } \overrightarrow{OA} \times \overrightarrow{OB} > 0 \end{cases}$$



Tìm bao lồi (P.08.13.05) – MÃ GIẢ

```
struct Point {
    int x, y;
};
Point P[N];
int n;
vector<Point> C;
void input(){
    read n;
    for(int i = 0; i < n; i++)
      read P[i].x, P[i].y;
}
```

```
dist2(Point a, Point b) {
    x = a.x - b.x;
    y = a.y - b.y;
    return x*x + y*y;
}
cross_product(Point O, Point A, Point B) {
    //tich vo huong 2 vector (O,A).(O,B)
    xa = A.x - O.x; ya = A.y - O.y;
    xb = B.x - O.x; yb = B.y - O.y;
    return xa * yb - ya * xb;
}
cmp(Point A, Point B){
    cp = cross_product(P[0],A,B);
    return cp == 0 ? dist2(P[0],A) < dist2(P[0],B) : cp > 0;
}
ccw(Point a, Point b, Point c) {
    cp = cross_product(a, b, c);
    return cp == 0 ? 0 : (cp < 0 ? -1 : 1);
}
```

Tìm bao lồi (P.08.13.05) – MÃ GIẢ

```
solve(){
    // find lowest point
    k = 0;
    for i = 1 to n – 1 do {
        if(P[i].y < P[k].y or P[i].y == P[k].y and P[i].x < P[k].x) k = i;
    }
    swap(P[0],P[k]);// let P[0] be the lowest point
    sort(P+1,P+n,cmp);
    C.push_back(P[0]);    C.push_back(P[1]);
    for i = 2 to n-1 do {
        while(C.size() > 1 and ccw(C[C.size()-2], C[C.size()-1],P[i]) <= 0)
            C.pop_back();
        C.push_back(P[i]);
    }
    print C[0], C[1], . . ., C[C.size-1];
}
```

```
main(){
    input();
    solve();
}
```


Tìm bao lồi (P.08.13.05) - CODE

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5;
struct Point {
    int x, y;
    Point():x(0),y(0){}
    Point(int x, int y) : x(x), y(y) {}
};
Point P[N];
int n;
vector<Point> C;
void input(){
    cin >> n;
    for(int i = 0; i < n; i++)
    cin >> P[i].x >> P[i].y;
}
```

```
long long dist2(Point &a, Point &b) {
    long long x = a.x - b.x;
    long long y = a.y - b.y;
    return 1LL * x*x + 1LL * y*y;
}
long long cross_product(Point &O, Point &A, Point &B) {
    //tich vo huong 2 vector (O,A).(O,B)
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * yb - 1LL * ya * xb;
}
bool cmp(Point& A, Point& B){
    long long cp = cross_product(P[0],A,B);
    return cp == 0 ? dist2(P[0],A) < dist2(P[0],B) : cp > 0;
}
int ccw(Point &a, Point &b, Point &c) {
    long long cp = cross_product(a, b, c);
    return cp == 0 ? 0 : (cp < 0 ? -1 : 1);
}
```


Tìm bao lồi (P.08.13.05) - CODE

```
void solve(){
    // find lowest point
    int k = 0;
    for(int i = 1; i < n; i++){
        if(P[i].y < P[k].y || P[i].y == P[k].y && P[i].x < P[k].x) k = i;
    }
    swap(P[0],P[k]);// let P[0] be the lowest point
    sort(P+1,P+n,cmp);
    C.push_back(P[0]);    C.push_back(P[1]);
    for(int i = 2; i < n; i++){
        while(C.size() > 1 && ccw(C[C.size()-2], C[C.size()-1],P[i]) <= 0)
            C.pop_back();
        C.push_back(P[i]);
    }
    for(int i = 0; i < C.size(); i++)
         cout << "(" << C[i].x << "," << C[i].y << ") ";
    cout << endl;
}
```

```
int main(){
    input();
    solve();
    return 0;
}
```

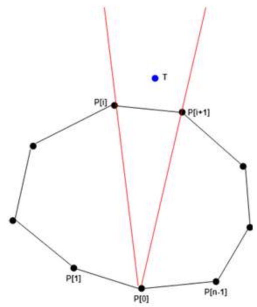

Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06)

- Cho n điểm $P_1, P_2, \ldots, P_n$ trên mặt phẳng, tọa độ nguyên. Cho K điểm $T_1, T_2, \ldots, T_k$. Hãy kiểm tra xem tập điểm $P_1, P_2, \ldots, P_n$ có tạo thành đa giác lồi hay không? Nếu có thì kiểm tra mỗi điểm trong số $T_1, T_2, \ldots, T_k$ có nằm trong (hoặc trên cạnh) của đa giác lồi đó hay không? (các điểm đều có tọa độ nguyên nằm trong khoảng từ -1000 đến 1000)
- Dữ liệu
  - Dòng 1: ghi số nguyên dương n (3 <= n <= 10000)
  - Dòng i+1 (i = 1, 2, ..., n): ghi 2 số nguyên là tọa độ x và y của điểm P_i
  - Dòng n+2: ghi số nguyên dương K (1 <= K <= 100000)
  - Dòng $k + n + 2$ (k = 1, 2, ..., K): ghi 2 số nguyên là tọa độ x và y của điểm T_k
- Kết quả
  - Dòng thứ k: ghi giá trị 1 nếu tập điểm $P_1, P_2, \ldots, P_n$ có tạo thành đa giác lồi và điểm Tk nằm trong trong trên cạnh đa giác đó; và ghi ra 0, nếu ngược lại

stdin	stdout
4 5 6 3 7 2 5 5 3 4 8 7 4 5 3 7 0 0	0 1 1 0

Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06)

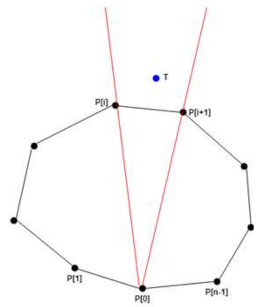
- Cho đa giác lồi n điểm $P[i]$, kiểm tra xem điểm T có nằm trong đa giác lồi đã cho hay không?
- Kiểm tra xem điểm T có nằm cùng phía với điểm $P[n-1]$ so với đường thẳng đi qua $P[0]$ và $P[1]$ hay không.
- Kiểm tra xem điểm T có nằm cùng phía với điểm $P[1]$ so với đường thẳng đi qua $P[0]$ và $P[n-1]$ hay không.
- Sử dụng tìm kiếm nhị phân để tìm điểm $P[i]$ thỏa mãn điểm $P[i+1]$ nằm khác phía với điểm $P[1]$ so với đường thẳng đi qua $P[0]$ và T, và điểm $P[i]$ nằm cùng phía với điểm $P[1]$ so với đường thẳng đi qua $P[0]$ và T.
- T nằm trong đa giác khi $P[0]$ và T nằm cùng phía so với đường thẳng đi qua $P[i]$ và $P[i+1]$.



Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06) – MÃ GIẢ

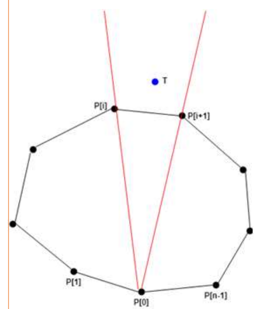
```
ccw(Point a, Point b, Point c) {
    cp = cross_product(a, b, c);
    return cp == 0 ? 0 : (cp < 0 ? -1 : 1);
}

same_side(Point a, Point b, Point c, Point d) {
    // return true if c and d are in the same side of the line (a,b)
    sc = ccw(a, b, c);
    sd = ccw(a, b, d);
    return sc * sd >= 0;
}
```



Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06) – MÃ GIẢ

```
checkInSideConvexHull(vector<Point> C, Point q) {
    last = C.size() - 1;
    if (same_side(C[0], C[1], C[last], q) and // C[last] và q cùng phía đối với
                                             // C[0]C[1]
        same_side(C[0], C[last], C[1], q)) {// C[1] và q khác phía đối với
                                             // C[last]C[0]
      l = 1;    r = C.size() - 1;
      while (r - l > 1) do {
        mid = (l + r) / 2;
        if (same_side(C[0], C[mid], C[last], q)) l = mid; else r = mid;
      }
      if ( not same_side(C[l], C[r], C[0], q))
        return false;
      else return true;
    }
    return false;
}
```



Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06) – CODE

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+1;
struct Point {
    int x, y;
    Point():x(0),y(0){}
    Point(int x, int y) : x(x), y(y) {}
};
Point P[N];
int n;
vector<Point> C;
Point O; // goc
long long dist2(Point &a, Point &b) {
    long long x = a.x - b.x;
    long long y = a.y - b.y;
    return 1LL * x*x + 1LL * y*y;
}
```

```
long long cross_product(Point &O, Point &A, Point &B) {
    //tich co huong 2 vector (O,A) x (O,B)
    long long xa = A.x - O.x; long long ya = A.y - O.y;
    long long xb = B.x - O.x; long long yb = B.y - O.y;
    return 1LL * xa * yb - 1LL * ya * xb;
}
bool cmp(Point& A, Point& B){
    long long cp = cross_product(O,A,B);
    return cp == 0 ? dist2(O,A) < dist2(O,B) : cp > 0;
}
int ccw(Point &a, Point &b, Point &c) {
    long long cp = cross_product(a, b, c);
    return cp == 0 ? 0 : (cp < 0 ? -1 : 1);
}
bool same_side(Point &a, Point &b, Point &c, Point &d) {
    // return true if c and d are in the same side of the line (a,b)
    int sc = ccw(a, b, c);
    int sd = ccw(a, b, d);
    return sc * sd >= 0;
}
```

Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06) – CODE

```
void computeConvexHull(Point* P, int n){
  C.clear(); int k = 0;
  for(int i = 1; i < n; i++){
    if(P[i].y < P[k].y ||
      P[i].y == P[k].y && P[i].x < P[k].x) k = i;
  }
  swap(P[0],P[k]);// let P[0] be the lowest point
  O = P[0];// update goc
  sort(P+1,P+n,cmp);
  C.push_back(P[0]);    C.push_back(P[1]);
  for(int i = 2; i < n; i++){
    while(C.size() > 1
        && ccw(C[C.size()-2], C[C.size()-1],P[i]) <= 0)
       C.pop_back();
    C.push_back(P[i]);
  }
}
```

```
int checkInSideConvexHull(vector<Point> P, Point& T) {
  int last = P.size() - 1;
  if (same_side(P[0], P[1], P[last], T) &&
                   same_side(P[0], P[last], P[1], T)) {
    int l = 1;    int r = P.size() - 1;
    while (r - l > 1) {
      int mid = (l + r) >> 1;
      if (same_side(P[0], P[mid], P[last], T)) l = mid;
      else r = mid;
    }
    if (!same_side(P[l], P[r], P[0], T))  return 0;
    else return 1;
  }
  return 0;
}
```


Kiểm tra 1 điểm nằm trong đa giác lồi (P.08.13.06) – CODE

```
int main(){
  scanf("%d",&n);
  for(int i = 0; i < n; i++){
    scanf("%d%d",&(P[i].x),&(P[i].y));
  }
  computeConvexHull(P,n);
  int res = 1;
  if(C.size() != n) res = 0;
  int K;
  scanf("%d",&K);
  for(int k = 1; k <= K; k++){
    int x,y;        scanf("%d%d",&x,&y);
    Point p(x,y);
    if(res == 1)
      res = checkInSideConvexHull(C,p);
    printf("%d\n",res);
  }
  return 0;
}
```


HUST

THANK YOU !