Chuyên đề
Các thuật toán về hình học phẳng

Hoàng Văn Thiên
THPT Chuyên Lý Tự Trọng – Cần Thơ

Tháng 8, 2016

Tóm tắt nội dung

Hình học tính toán (*computational geometry*) là một nhánh nhỏ của ngành khoa học máy tính. Nó là nguồn gốc sinh ra các bài toán thuần hình học đang được giảng dạy ở chương trình phổ thông và cao cấp. Khác với *hình học thuần*, bộ môn mà ở đó các bài toán có thể giải bằng giấy bút, hình học tính toán áp dụng các thuật toán và máy vi tính để đưa ra đáp án.

Trong hình học tính toán, người ta chia thành hai nhóm:

Toán hình học mô hình	:	Đưa các bài toán thực tế về mô hình toán học thích hợp mà có thể giải bằng máy vi tính.
Hình học thuật toán	:	Nghiên cứu và phát triển các giải thuật, cấu trúc dữ liệu để xử lý các yếu tố cơ bản về hình học.

Ở phạm vi tài liệu này, chúng ta sẽ bàn về hình học thuật toán, đặc biệt là hai vấn đề quan trọng sau:

- Cách lưu trữ các đối tượng hình học cơ bản: điểm, đường thẳng, vector, đường tròn, đa giác, v.v..
- Thao tác với đa giác và tập điểm: Tìm bao lồi, kiểm tra đa giác lồi, vị trí tương đối của một điểm và đa giác, cắt đa giác, v.v..

Hình học là bài toán rất phổ biến trong các kỳ thi lập trình. Hầu hết mọi cuộc thi ICPC[1] đều có ít nhất một bài về hình học. Trong những năm gần đây, kỳ thi IOI[2] cũng có xu hướng tương tự. Bài toán hình học thường rất dễ vướng vào các bộ test đặc biệt như ba điểm thẳng hàng, đa giác lõm. Nhìn chung, người lập trình phải đặc biệt cẩn thận đối với dạng bài này.

Trong khoa học máy tính, hiểu được vấn đề và cách giải rất khác so với việc thể hiện và xử lý nó bằng code. Do đó, chuyên đề sẽ giản lược các khái niệm sơ khai, không cần thiết, và dễ tiếp cận qua sách vở, internet. Vì để giải quyết một bài toán với kích thước dữ liệu lớn, độ phức tạp $O(n \log n)$ sẽ cho thấy sự vượt trội đáng kể so với $O(n^2)$. Chuyên đề tập trung vào thuật toán – yếu tố cần được lưu tâm nhất, đặc biệt là các đối tượng thí sinh của các cuộc thi lập trình lại cần giải thuật hiệu quả được thực hiện bằng một đoạn code ngắn gọn và dễ tùy biến.

Code minh họa trong chuyên đề được viết bằng ngôn ngữ C++, tuy nhiên chúng tôi

[1] International Collegiate Programming Contest
[2] International Olympiad of Informatics

Mục lục

1 Các yếu tố hình học

1.1 Điểm

Điểm là một khái niệm sơ khai đến mức không thể định nghĩa được trong hình học Euclid[3].

1.1.1 Lưu trữ một điểm

Trên mặt phẳng tọa độ, điểm (x, y) có thông tin là hoành độ x, tung độ y. Như vậy ta dễ dàng lưu một điểm bằng `record` trong Pascal (`struct` trong C/C++, `class` trong Java), với các trường `x,y`[4] là số nguyên hoặc số thực (tùy đề bài yêu cầu).

Nếu có thể, khuyến khích sử dụng số nguyên để tránh sai số thập phân xảy ra trên số thực, khó kiểm soát đối với người lập trình chưa có kinh nghiệm.

Sẽ có lúc chúng ta cần sắp xếp lại các điểm, nên khi so sánh hai điểm, ta quy ước lấy hoành độ làm khóa chính, tung độ làm khóa phụ. Cụ thể hơn, với hai điểm $P_1(x_1, y_1), P_2(x_2, y_2)$:

$$P_1 < P_2 \Leftrightarrow \left[\begin{array}{l} x_1 < x_2 \\ \begin{cases} x_1 = x_2 \\ y_1 < y_2 \end{cases} \end{array}\right., \quad P_1 = P_2 \Leftrightarrow \begin{cases} x_1 = x_2 \\ y_1 = y_2 \end{cases}$$

```
1  struct Point {
2      double x,y;
3      Point() {x = 0; y = 0;}
4      Point(double _x, double _y) {x = _x; y = _y}
5      bool operator < (Point B) const {
6          if (fabs(x - B.x) < EPS)
7              return y < B.y
8          else return x < B.x;
9      }
10     bool operator == (Point B) const {
11         return (fabs(x - B.x) < EPS && fabs(y - B.y) < EPS);
12     }
13 }
```

Dòng 6 tương đương với việc kiểm tra x có bằng $B.x$ hay không. **EPS**ilon $= 10^{-9}$ là một số rất nhỏ. Nếu độ chênh lệch giữa hai số là $|x - B.x| < 10^{-9}$ nhỏ như thế thì có thể xem chúng bằng nhau. So sánh hai số thực cần được thực hiện như thế này để tránh sai số thập phân khi tính toán.

1.1.2 Khoảng cách giữa hai điểm

Ta tính khoảng cách giữa hai điểm P_1 và P_2 nói trên bằng công thức quen thuộc:

$$P_1P_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

[3]Trong chuyên đề này, mọi khía cạnh đều xét trên hình học Euclid

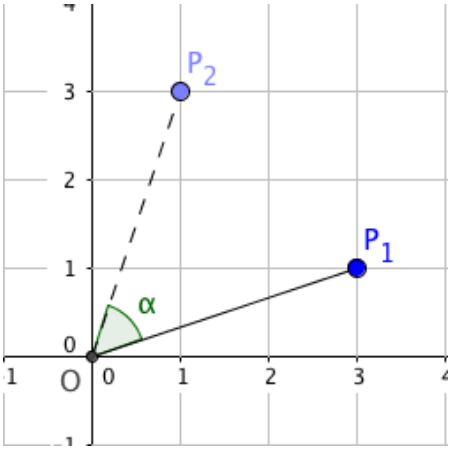
[4]Nếu xét trong hệ tọa độ $Oxyz$, chỉ cần thêm một trường z

```
1 double dist(Point P1, Point P2) {
2     return hypot(P1.x - P2.x, P1.y - P2.y);
3 }
```

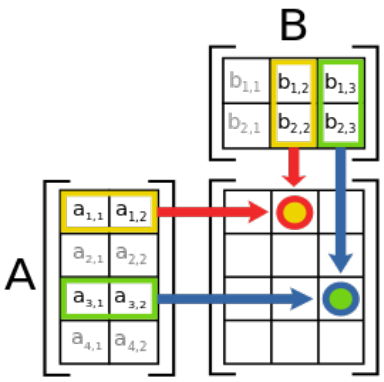
Hàm `hypot(double b, double c)` trả về độ dài cạnh huyền của tam giác vuông có hai cạnh góc vuông là $|b|$ và $|c|$, tức là $\sqrt{b^2 + c^2}$.

1.1.3 Ảnh của một điểm qua phép quay

Ta có thể tìm điểm P_2 là ảnh của P_1 qua phép quay Q quanh điểm O một góc $\alpha > 0$ (ngược chiều kim đồng hồ).



Hình 1: Phép quay



Hình 2: Phép nhân ma trận

Ta có:

$$\begin{cases} x_2 = x_1 \cos\alpha - y_1 \sin\alpha \\ y_2 = x_1 \sin\alpha + y_1 \cos\alpha \end{cases} \Leftrightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

```
1 Point rotation(Point P, double alpha) {
2     return Point(P.x * cos(alpha) - P.y * sin(alpha),
3                  P.x * sin(alpha) + P.y * cos(alpha));
4 }
```

Tất nhiên, để tiện tính toán, người ta vẫn thường dùng radian thay cho độ. Có thể ghi nhớ bằng công thức hoặc phép nhân ma trận.

1.2 Đường thẳng, vector

1.2.1 Lưu trữ đường thẳng

Trong mặt phẳng tọa độ, đường thẳng $(d) : ax + by + c = 0$ là một kiến thức phổ thông quen thuộc.

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

1.2.2 Vị trí tương đối hai đường thẳng

Khi nói đến đường thẳng, người ta sẽ liên tưởng đến khái niệm vị trí tương đối của chúng.

Thông thường, khi xét hai đường $(d_1) : a_1x + b_1y + c_1 = 0$ và $(d_2) : a_2x + b_2y + c_2 = 0$ ta phải biện luận về số nghiệm của hệ $\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$

- d_1 cắt d_2, hệ có một nghiệm duy nhất $\Leftrightarrow \frac{a_1}{a_2} \neq \frac{b_1}{b_2}$
- d_1 song song với d_2, hệ vô nghiệm $\Leftrightarrow \frac{a_1}{a_2} = \frac{b_1}{b_2} \neq \frac{c_1}{c_2}$
- d_1 trùng với d_2, hệ vô số nghiệm $\Leftrightarrow \frac{a_1}{a_2} = \frac{b_1}{b_2} = \frac{c_1}{c_2}$

Tuy nhiên, hàng loạt các công thức này khi đưa vào máy tính cần được kiểm soát chặt chẽ để tránh mẫu số bằng 0 (division by zero). Nếu áp dụng những phương pháp truyền thống thì sẽ mất thời gian và tiềm năng lỗ hổng cao. Do đó, ta chỉ nên sử dụng hai loại $(d) : \begin{cases} y = ax + b & (1) \\ x = a & (2) \end{cases}$. Một loại chỉ dùng cho các đường thẳng không song song với Oy, và loại kia dùng cho trường hợp ngược lại.

Tóm lại, thông tin về một đường thẳng vẫn được lưu bằng ba trường `a, b, c`, mô tả đường thẳng $(d) : ax + by + c = 0$; **trong đó `b` chỉ có hai giá trị là 0 hoặc 1.**

- `areParallel(Line d1, Line d2)`: d_1 cùng phương[5] với $d_2 \Leftrightarrow \begin{cases} a_1 = a_2 \\ b_1 = b_2 \end{cases}$ (chúng phải cùng có dạng (1) hoặc cùng có dạng (2) mới *có khả năng* cùng phương).
- `areTheSame(Line d1, Line d2)`: d_1 trùng với d_2 khi chúng đã cùng phương và $c_1 = c_2$.
- `areIntersect(Line d1, Line d2)`: d_1 cắt d_2 khi chúng không cùng phương.[6]

[5] Hai đường thẳng cùng phương có thể song song hoặc trùng nhau

[6] Dùng khái niệm này để hình thành nên khái niệm kia tạo ra điểm mạnh trong thuật toán: chia nhỏ công việc.

1.2.3 Tìm giao điểm hai đường thẳng không cùng phương

Giao điểm của d_1 và d_2 là nghiệm của hệ hai phương trình bậc nhất hai ẩn:

$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$$

Ta có các định thức:

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}, \quad D_x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}, \quad D_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

Vậy nghiệm của hệ là: $\begin{cases} x = \frac{D_x}{D} \\ y = \frac{D_y}{D} \end{cases}$

```
bool areIntersect(Line d1, Line d2, Point & I) {
    if (areParallel(d1, d2)) return false;
    I.x = (d1.c * d2.b - d1.b * d2.c) / (d1.a * d2.b - d1.b * d2.a);
    I.y = (d1.a * d2.c - d1.c * d2.a) / (d1.a * d2.b - d1.b * d2.a);
}
```

1.2.4 Dựng đường thẳng đi qua hai điểm

Cách cơ bản để hình thành đường thẳng d đó là bắt đầu từ hai điểm P_1 và P_2 cho trước.

- Nếu $x_1 = x_2$ thì $(d) : x = x_1$.
- Ngược lại, ta xây dựng vector pháp tuyến và chuyển về để `b = 1`:

$$\overrightarrow{P_1P_2} = (x_2 - x_1,\ y_2 - y_1) \Rightarrow \overrightarrow{n} = (y_2 - y_1,\ x_1 - x_2) = (\frac{y_2 - y_1}{x_1 - x_2},\ 1) \Rightarrow \begin{cases} a = \frac{y_2 - y_1}{x_1 - x_2} \\ b = 1 \\ c = -ax_1 - y_1 \end{cases}$$

Nếu đường thẳng d đi qua điểm $M(x_0,\ y_0)$, có vector pháp tuyến $\overrightarrow{n} = (a, b)$ thì có phương trình $(d) : ax + by - ax_0 - by_0 = 0$

```
Line makeLine(Point P1, Point P2) {
    double a, b, c;
    if (P1.x == P2.y) {
        a = 1;
        b = 0;
        c = –P1.x;
    } else {
        a = (P2.y – P1.y) / (P1.x – P2.x);
        b = 1;
        c = –a*P1.x – P1.y; }
    return Line(a, b, c); }
```

1.2.5 Vector và các phép toán liên quan

Các phép toán này rất cơ bản và được giảng dạy nhiều trong chương trình phổ thông, tuy nhiên vẫn nên nhắc đến vì sự cần thiết để giải các vấn đề sắp được bàn luận.

1. Mỗi vector trong hệ trục tọa độ Oxy đều có hoành độ và tung độ.

```
struct Vect {
    double x, y;
    Vect(double _x, double _y) { x = _x; y = _y; } }
```

2. Từ hai điểm đầu và cuối có thể vẽ được một vector:

```
Vect getVect(Point P1, Point P2) {
    return Vect(P2.x - P1.x, P2.y - P1.y); }
```

3. Nhân vector với một số nguyên $k \neq 0$

```
Vect scale(Vect v, double k) {
    return Vect(v.x * k, v.y * k); }
```

4. Tích vô hướng hai vector

```
double scalarProduct(Vect v1, Vect v2) {
    return v1.x * v2.x + v1.y * v2.y; }
```

5. Tịnh tiến một điểm theo vector

```
Point translate(Point P, Vect v) {
    return Point(P.x + v.x, P.y + v.y); }
```

6. Độ lớn (độ dài) của vector

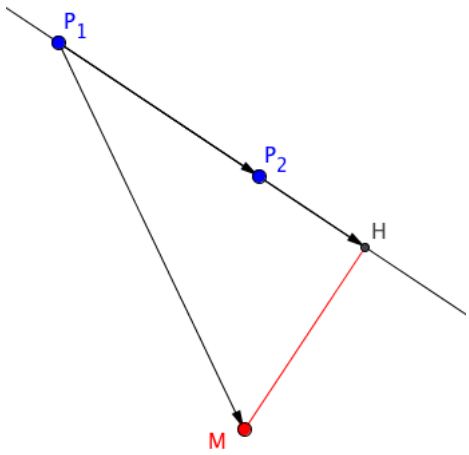
```
double getLength(Vect v) {
    return hypot(v.x, v.y); }
```

Để tiện trong một vài phép tính, ta cũng định nghĩa một hàm tính bình phương độ dài của vector.

```
double getLength_sq(Vect v) {
    return v.x * v.x + v.y * v.y; }
```

1.2.6 Tính khoảng cách từ một điểm đến một đường thẳng

Có hai dạng tính khoảng cách MH từ một điểm M đến một đường thẳng d:



Hình 3: Minh họa cách tìm khoảng cách từ một điểm đến đường thẳng

1. **Cho biết d đi qua hai điểm P_1 và P_2**

   Có nhiều cách tiếp cận bài toán này, nếu là *hình học thuần*, người ta sẽ xét các tính chất của hình vẽ để tìm đáp án dễ dàng hơn, tránh sai sót khi cộng trừ. Tuy nhiên, lợi thế của chiếc máy vi tính là... tính toán cực kỳ chính xác, nên hãy phát huy nó. Sau đây là một trong những cách đơn giản và hiệu quả:

   Nhận thấy rằng, hình chiếu H của điểm M có thể tính ra được bằng cách tịnh tiến điểm P_1 theo vector $\overrightarrow{P_1H}$. Lấy chiều của $\overrightarrow{P_1P_2}$ làm chuẩn, dễ thấy $\overrightarrow{P_1H}$ cùng chiều hay ngược chiều so với chiều chuẩn sẽ phụ thuộc vào góc lượng giác $(\overrightarrow{P_1P_2}, \overrightarrow{P_1M})$. [7]. Đặt $\overrightarrow{P_1H} = k\overrightarrow{P_1P_2}$

$$\Rightarrow k = \frac{P_1M.\cos(\overrightarrow{P_1P_2}, \overrightarrow{P_1M})}{P_1P_2} = \frac{\overrightarrow{P_1M}.\overrightarrow{P_1P_2}}{{P_1P_2}^2}$$

```
1 double distToLine(Point M, Point P1, Point P2) {
2     double k = scalarProduct(Vect(P1, M), Vect(P1, P2))
3                / getLength_sq(P1, P2);
4     Point H = translate(P1, scale(Vect(P1, P2), k));
5     return dist(M, H); }
```

2. **Cho biết phương trình của d**

   - Nếu d song song với Oy, đáp án chính bằng `fabs(-c - M.x)`.
   - Ngược lại, ta tìm hai điểm phân biệt từ hai hoành độ khác nhau. Sau đó đưa về bài toán dạng 1.

[7] Trong tài liệu này, mọi ký hiệu $(\overrightarrow{a}, \overrightarrow{b})$ được ngầm hiểu là **góc lượng giác** giữa hai vector $\overrightarrow{a}$ và $\overrightarrow{b}$

1.3 Góc

1.3.1 Tính góc lượng giác giữa hai vector

$$\cos(\overrightarrow{OA}, \overrightarrow{OB}) = \frac{\overrightarrow{OA}.\overrightarrow{OB}}{\sqrt{OA^2 + OB^2}}$$

```
1 double getAngle(Point O, Point A, Point B) {
2     Vect OA = Vect(O,A), OB = Vect(O,B);
3     return acos(scalarProduct(OA, OB) /
4             sqrt(getLength_sq(OA), getLength_sq(OB))); }
```

Trong C++, hàm `acos` tương ứng với arccos trong toán học, và $\cos^{-1}$ trong máy tính cầm tay.

1.3.2 Tích có hướng của hai vector

Tích có hướng là khái niệm chỉ có nghĩa trong không gian 3 chiều hoặc 7 chiều. Tích có hướng còn có tên khác là tích ngoài. Tên tiếng Anh: cross product, vector product, outer product.

Trong hệ tọa độ $Oxyz$, hai vector $\overrightarrow{u} = (x, y, z)$ và $\overrightarrow{v} = (x', y', z')$ cùng thuộc mặt phẳng Oxy.[8] Tích có hướng của hai vector này là $\overrightarrow{w} = [\overrightarrow{u}, \overrightarrow{v}]$.[9] Vector $\overrightarrow{w}$ lần lượt vuông góc với $\overrightarrow{u}$ và $\overrightarrow{v}$, có *độ lớn đại số* bằng $|\overrightarrow{u}|.|\overrightarrow{v}|.\sin(\overrightarrow{u}, \overrightarrow{v}) \in \mathbb{R}$. Nhiều tài liệu gọi đây là giá trị của tích ngoài hai vector và ký hiệu là $\overrightarrow{u} \wedge \overrightarrow{v}$.

$$\overrightarrow{w} = (\begin{vmatrix} y & z \\ y' & z' \end{vmatrix}; \begin{vmatrix} z & x \\ z' & x' \end{vmatrix}; \begin{vmatrix} x & y \\ x' & y' \end{vmatrix}) = (0;\ 0;\ xy' - x'y)$$

Vậy:

$$|\overrightarrow{u}|.|\overrightarrow{v}|.\sin(\overrightarrow{u}, \overrightarrow{v}) = xy' - x'y = \overrightarrow{u} \wedge \overrightarrow{v}$$

Ta chỉ quan tâm đến cao độ (z) của $\overrightarrow{w}$.

```
1 double crossProduct(Vect u, Vect v) {
2     return u.x*v.y - u.y*v.x; }
```

Lưu ý tính phản giao hoán của phép nhân có hướng: $[\overrightarrow{u}, \overrightarrow{v}] = -[\overrightarrow{v}, \overrightarrow{u}]$ và $\overrightarrow{u} \wedge \overrightarrow{v} = -(\overrightarrow{v} \wedge \overrightarrow{u})$

1.3.3 CCW

Cho ba điểm $O,\ A,\ B$ thuộc **mặt phẳng** Oxy, kiểm tra xem đường đi $O \to A \to B$ (hướng tam giác OAB) là đường đi rẽ trái (**C**ounter **C**lock**W**ise – ngược chiều kim đồng hồ), hay rẽ phải (ClockWise – thuận chiều kim đồng hồ), hay ba điểm thẳng hàng.

Đặt $\overrightarrow{u} = \overrightarrow{OA},\ \overrightarrow{v} = \overrightarrow{OB}$.

[8] Tọa độ z của hai vector này bằng 0

[9] Tọa độ x, y của $\overrightarrow{w}$ bằng 0

Diện tích đại số của tam giác OAB, được định nghĩa và ký hiệu

$$S[OAB] = \frac{1}{2}(\overrightarrow{u} \wedge \overrightarrow{v})$$

Định lý: Diện tích đại số dương (bằng diện tích hình học) khi tam giác OAB hướng dương, âm khi tam giác OAB hướng âm. Như vậy,

- O, A, B có thứ tự **ngược chiều kim đồng hồ** $\Leftrightarrow xy' - x'y > 0$
- O, A, B thẳng hàng $\Leftrightarrow \overrightarrow{u}, \overrightarrow{v}$ cùng phương $\Leftrightarrow xy' - x'y = 0$.
- O, A, B có thứ tự cùng chiều kim đồng hồ $\Leftrightarrow xy' - x'y < 0$.

```
1 bool ccw(Point O, Point A, Point B) {
2     Vect OA = Vect(O,A), OB = Vect(O,B);
3     return (crossProduct(OA, OB) > 0); }
4
5 bool collinear(Point O, Point A, Point B) {
6     Vect OA = Vect(O,A), OB = Vect(O,B);
7     return (fabs(crossProduct(OA, OB)) < EPS); }
```

2 Các dạng hình cơ bản

Hình tròn, hình tam giác, hình tứ giác, v.v.. là các hình đơn giản và không yêu cầu quá cao về giải thuật khi tính toán với chúng. Do đó, tôi sẽ chỉ nhắc lại sơ lược những kiến thức cần nắm chứ không đầu tư phân tích.

2.1 Hình tam giác

1. Tam giác ABC có ba cạnh độ dài $BC = a,\ AC = b,\ AB = c$. Chu vi tam giác $P = \frac{P}{2}$, nửa chu vi $p = a + b + c$. Bán kính đường tròn ngoại tiếp là R, bán kính đường tròn nội tiếp là r.

2. Diện tích tam giác

$$S = \frac{1}{2}ah = \sqrt{p(p-a)(p-b)(p-c)} = \frac{abc}{4R} = pr = \frac{1}{2}ab\sin\widehat{C} = \dots$$

   Các hình đặc biệt như tam giác đều, tam giác vuông có cách tính gọn hơn, nhưng đều có bản chất là những công thức này.

3. Bất đẳng thức tam giác: tổng hai cạnh bất kỳ lớn hơn hai cạnh còn lại.

4. Định lý cosine:

$$c^2 = a^2 + b^2 - 2ab\cos C$$

   Định lý sine:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

5. Định lý Pytago (Pythagorean Theorem) trong tam giác vuông: bình phương cạnh huyền bằng tổng bình phương hai cạnh góc vuông.
   Xem thêm định lý đảo Pytago.

6. Bộ ba số Pytago (Pythagorean Triple) gồm số đo ba cạnh của một tam giác vuông. VD, $(3, 4, 5)$.

2.2 Hình tứ giác

1. Các hình đặc biệt: hình bình hành, hình chữ nhật, hình vuông, hình thoi, hình thang. Mỗi hình có cách tính diện tích khác nhau.

2. Tứ giác nội tiếp có tổng hai góc đối nhau bằng π.

3 Thuật toán trên đa giác

3.1 Lưu trữ đa giác

Dùng một dãy các `Point` (có thể lưu trong `vector`) để lưu các đỉnh của đa giác. **Quan trọng:** Các điểm phải đi theo một chiều nhất định[10] (thuận chiều hay ngược chiều kim đồng hồ). Ví dụ:

```
vector<Point> polygon;
polygon.push_back(Point(0,0));
polygon.push_back(Point(2,1));
polygon.push_back(Point(1,3));
polygon.push_back(Point(-5,2));
```

3.2 Chu vi đa giác

Đơn giản, chỉ cần tính tổng độ dài các cạnh của đa giác đó.

```
double perimeter(const vector<Point> &p) {
    int n = p.size();
    double res = 0;
    for(int i=0; i<n; ++i) {
        int next = (i+1)%n;
        res += dist(p[i], p[next]); }
    return res; }
```

3.3 Diện tích đa giác

Xin được nhắc lại, nếu O, A, B ngược chiều kim đồng hồ thì $S[OAB] > 0$; nếu O, A, B cùng chiều kim đồng hồ thì $S[OAB] < 0$.

[10] Giống như cách đọc tên đa giác

Với tính chất này, ta rút ra được

$$S[P_1P_2P_3\dots P_n] = S[MP_1P_2] + S[MP_2P_3] + \dots + S[MP_{n-1}P_n] + S[MP_nP_1](n \le 3)(*)$$

Biểu thức trên có thể dễ dàng chứng minh bằng phương pháp quy nạp và hệ thức Chasles.

Chứng minh:

Với $n = 3$, xét tam giác ABC và điểm M bất kỳ, ta có:

$$S[ABC] = \frac{1}{2}\left(\overrightarrow{AB} \wedge \overrightarrow{AC}\right) = \frac{1}{2}\left[\left(\overrightarrow{MB} - \overrightarrow{MA}\right) \wedge \left(\overrightarrow{MC} - \overrightarrow{MA}\right)\right]$$

$$= \frac{1}{2}\left(\overrightarrow{MB} \wedge \overrightarrow{MC} - \overrightarrow{MA} \wedge \overrightarrow{MC} - \overrightarrow{MB} \wedge \overrightarrow{MA} + \overrightarrow{MA} \wedge \overrightarrow{MA}\right)$$

$$= \frac{1}{2}\left(\overrightarrow{MB} \wedge \overrightarrow{MC} + \overrightarrow{MC} \wedge \overrightarrow{MA} + \overrightarrow{MA} \wedge \overrightarrow{MB}\right)$$

$$= S[MBC] + S[MCA] + S[MAB]$$

Giả sử (*) đúng với $n = k$, tức là:

$$S[P_1P_2\dots P_k] = S[MP_1P_2] + S[MP_2P_3] + \dots + S[MP_{k-1}P_k] + S[MP_kP_1]$$

Ta cần chứng minh (*) đúng với $n = k + 1$.

$$S[P_1P_2\dots P_{k+1}] = S[P_1P_2\dots P_k] + S[P_kPk+1P_1]$$

$$= S[MP_1P_2]+S[MP_2P_3]+\dots+S[MP_{k-1}P_k]+S[MP_kP_1]+S[MP_kP_{k+1}]+S[MP_{k+1}P_1]+S[MP_1P_k]$$

$$= S[MP_1P_2] + S[MP_2P_3] + \dots + S[MP_{k-1}P_k] + S[MP_kP_{k+1}] + S[MP_{k+1}P_1]$$

Lưu ý: $S[MP_kP_1] + S[MP_1P_k] = 0$

Vậy (*) đúng với $n = k + 1$. Theo nguyên lý quy nạp, (*) đúng với mọi $n \le 3$. Thay điểm M bằng gốc tọa độ O để tiện tính toán: tọa độ vector $\overrightarrow{MP}$ cũng chính bằng tọa độ đỉnh P.

```
1 double getArea(const vector<Point> &p) {
2     int n = p.size();
3     double ans = 0;
4     for(int i=0; i<n; ++i) {
5         int next = (i+1)%n
6         ans += p[i].x*p[i+1].y - p[i].y*p[i+1].x;
7     }
8     return fabs(ans)/2; }
```

3.4 Kiểm tra tính lồi của đa giác

Xét một đa giác $P_1P_2\dots P_n$. Giả sử, ta ký hiệu:

- P'_i là đỉnh liền sau đỉnh P_i trong tên gọi đa giác. Đặc biệt, $P'_n = P_1$.

- P''_i là đỉnh liền sau đỉnh P'_i trong tên gọi đa giác. Đặc biệt, $P''_n = P'_1$.

  Ví dụ, đa giác có 5 đỉnh. Khi đó, $P'_3 = P_4,\ P''_3 = P'_4 = P_5,\ P'_5 = P_1,\ P''_5 = P'_1 = P_2,\ P''_4 = P'_5 = P_1$.

Đa giác này được gọi là lồi khi và chỉ khi tất cả bộ ba $P_i,\ P'_i,\ P''_i$ $(\forall i = \overline{1..n})$ đều có cùng chiều (CCW hoặc CW; nhưng **phải bảo đảm không được thẳng hàng trước khi đưa vào kiểm tra**).

```
1  bool checkConvex(const vector<Point> &p) {
2      int n = p.size();
3      if (n < 3) return false;
4      bool sample = ccw(p[0], p[1], p[2]);
5      for(int i=0; i<n; ++i) {
6          int next = (i+1)%n, later = (i+2)%n;
7          if (ccw(p[i], p[next], p[later]) != sample) return false;
8      }
9      return true; }
```

3.5 Kiểm tra một điểm nằm bên trong đa giác

Điểm M nằm trong đa giác $P_1P_2 \dots P_n$ khi và chỉ khi tổng các góc lượng giác

$$(\overrightarrow{MP_1}, \overrightarrow{MP_2}),\ (\overrightarrow{MP_2}, \overrightarrow{MP_3}),\ \dots,\ (\overrightarrow{MP_n}, \overrightarrow{MP_1})$$

bằng 2π.

```
1  bool insidePolygon(Point M, const vector<int> &p) {
2      int n = p.size();
3      if (n < 3) return false;
4      double sum = 0;
5      for(int i=0; i<n; ++i) {
6          int next = (i+1)%n;
7          sum += getAngle(M, p[i], p[next]); }
8      return fabs(fabs(sum) - 2*PI) < EPS; }
```

3.6 Thuật toán quét Graham

Một vấn đề thường gặp với các bài toán hình học là tìm *bao lồi* của tập điểm P gồm n điểm $P_0,\ P_1,\ \dots,\ P_{n-1}$ trên mặt phẳng Oxy. Đường bao lồi là đường gấp khúc kín, có hình dạng của một đa giác lồi, sao cho **tất cả mọi điểm** thuộc P đều nằm *bên trong hoặc ngay trên* đường này. Thuật toán Graham, với độ phức tạp $O(n \log n)$ sẽ giải quyết hiệu quả dạng bài này.

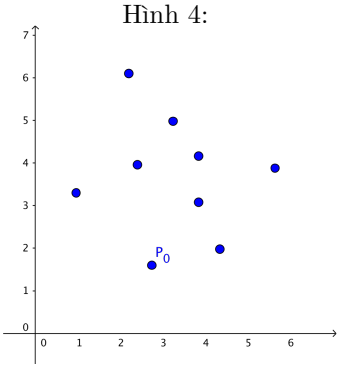
Nếu $n \leq 3$, bao lồi là chính tập điểm đã cho.

Nếu $n \geq 3$, thuật toán bao gồm các bước:

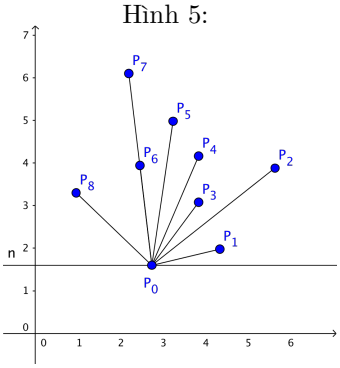
1. Chọn một điểm P_0 có tọa độ $y = y_{min}$ nhỏ nhất. Nếu có nhiều điểm như thế, chọn điểm có tọa độ x lớn nhất trong số đó.[11] (Hình 4)

[11] Lowest y and rightmost x

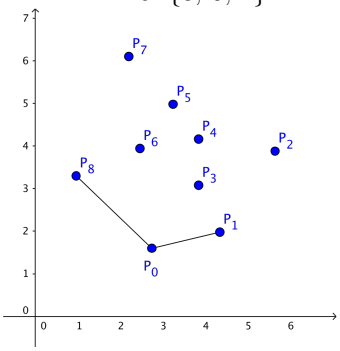
Hình 4:



Hình 5:



Hình 6: {8, 0, 1}



2. Sắp xếp $n-1$ điểm còn lại theo thứ tự tăng dần của góc lượng giác $\alpha = (\overrightarrow{i},\ \overrightarrow{P_0P_i})$ $(0 \le \alpha \le 2\pi,\ i = \overline{1..n-1})$ (Hình 5)

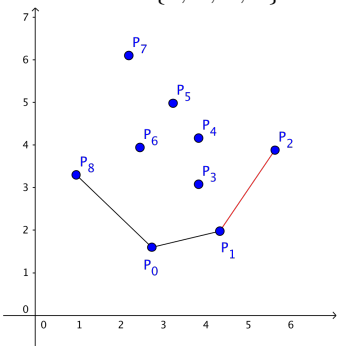
   $\overrightarrow{i} = (1,0)$ là vector đơn vị theo trục Ox

3. Dùng cấu trúc `stack<Point>`. Lần lượt đẩy vào stack các điểm $P_n,\ P_1,\ P_2$. (Hình 6) Chồng điểm này phải luôn đảm bảo rằng, 3 điểm trên cùng, từ dưới lên, phải luôn đi ngược chiều kim đồng hồ (tạo đường rẽ trái).

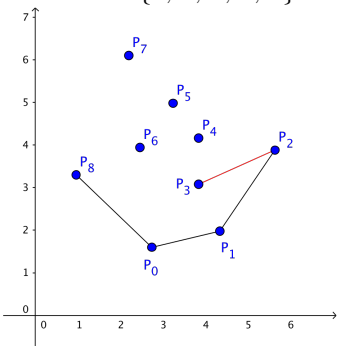
   Ta ký hiệu hai điểm S và S' là hai điểm trên cùng của stack (S dưới S'). Đây chỉ là ký hiệu viết tắt để tiện gọi tên, phụ thuộc vào trạng thái của stack, không phải là điểm cố định.
   Ví dụ: Nếu từ dưới lên, stack gồm điểm $\{A,\ B,\ D,\ C\}$ thì S chỉ điểm D, S' chỉ điểm C. Nhưng nếu trong quá trình thực hiện, stack thay đổi thành $\{A,\ B,\ D,\ E,\ F\}$ thì xin ngầm hiểu S chỉ điểm E, S' chỉ điểm F.

4. Bắt đầu với $i = 2$.

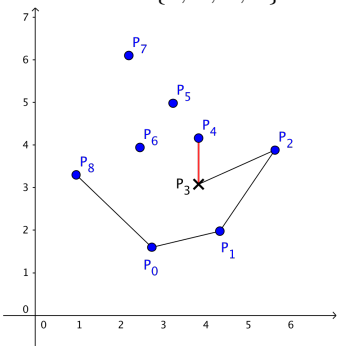
Hình 7: {8, 0, 1, 2}



Hình 8: {8, 0, 1, 2, 3}



Hình 9: {8, 0, 1, 2}



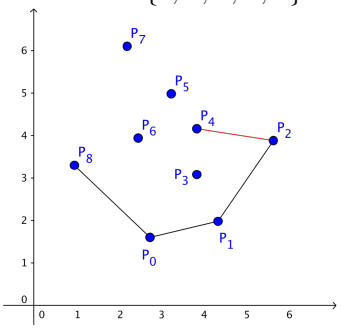
5. Nếu $S,\ S',\ P_i$ chạy theo hướng CCW (đường rẽ trái) (Hình 7 thì ta sẽ:
   - Đẩy thêm điểm P_i vào stack (tất nhiên, nằm trên cùng so với các điểm sẵn có).
   - Tăng i lên 1 đơn vị (để xét tiếp điểm P_{i+1}).
   - Nếu $i+1 < n$ (vẫn còn điểm để xét) thì quay lại từ đầu **bước này**. Nếu không, sang bước kế tiếp.

Nếu S, S', P_i không chạy theo hướng CCW (8), ta bỏ điểm S' ra khỏi stack (9). Quay lại từ đầu **bước này**.

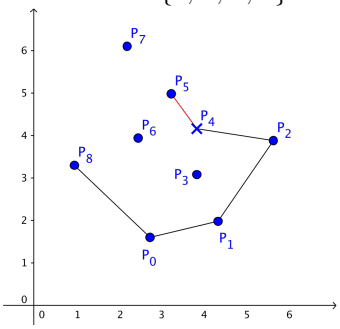
> Điều này chứng tỏ rằng S' là điểm "lõm", nên không có khả năng trở thành điểm thuộc bao lồi.

6. Kẻ các đường thẳng nối mọi cặp điểm nằm kề nhau trong stack, nối điểm trên cùng với điểm dưới cùng, sẽ tạo thành đường bao lồi của tập điểm. Kết thúc thuật toán.

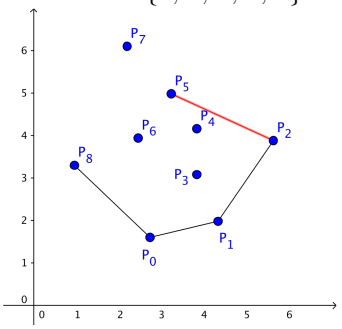
Hình 10: {8, 0, 1, 2, 4}



Hình 11: {8, 0, 1, 2}

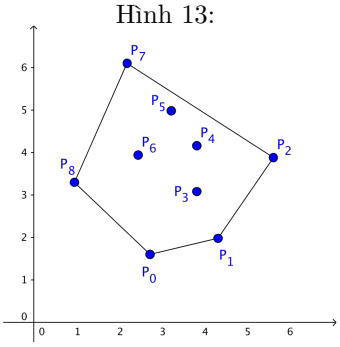


Hình 12: {8, 0, 1, 2, 5}



Tiếp tục thực hiện, cho ra kết quả:

Hình 13:



Định nghĩa cách so sánh hai điểm theo yếu tố góc lượng giác α nói ở bước 2:

```
1 Point pivot = Point(0, 0);
2 bool cmp(Point A, Point B) {
3     if (collinear(pivot, A, B)) return dist(pivot, A) < dist(pivot, B);
4     Vect OA = Vect(O, A), OB = Vect(O, B);
5     return atan2(OA.y, OA.x) < atan2(OB.y, OB.x);
6 }
```

> Hàm trên trả về `true` nếu điểm A "nhỏ hơn" điểm B

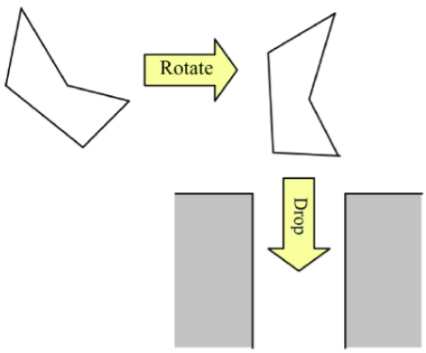
Cốt lõi của thuật toán:

```
1  vector<Point> convexHull(vector<Point> p) {
2      int n = p.size();
3      if (n <= 3) return p;
4      int t = 0;
5      for(int i = 1; i < n; ++i)
6          if (make_pair(p[i].x, p[i].y) < make_pair(p[t].x, p[t].y)) t = i;
7      swap(p[t], p[0]);
8      pivot = p[0];
9      sort(++p.begin(), p.end(), cmp);
10     vector<Point> stk;
11     stk.push_back(Point[n-1]);
12     stk.push_back(Point[0]);
13     stk.push_back(Point[1]);
14     int i = 2;
15     while (i < n) {
16         int j = stk.size();
17         if (ccw(stk[j-2], stk[j-1], Point[i])) {
18             stk.push_back(Point[i]);
19             ++i;
20         } else {
21             stk.pop_back();
22         }
23     }
24     return stk;
25 }
```

4 Bài tập vận dụng

Ống rác

Đây là bài toán rất thú vị và mang tính thực tiễn cao. Bài được trích từ bộ đề chung kết quốc tế ACM-ICPC năm 2011, tổ chức tại Orlando.



Tóm tắt đề bài:

Ống rác là thiết bị thường được lắp trong những tòa nhà cao tầng để đổ rác từ trên cao xuống thùng rác. Thiết kế ống rác thật sự rất quan trọng. Tùy vào loại rác, ống phải có kích thước phù hợp. Vì chi phí sản xuất tỉ lệ thuận với kích thước, công ty luôn muốn tối thiểu hóa độ rộng của ống.

Xét trên phương diện hình học phẳng – một khía cạnh đơn giản của vấn đề, ống rác hướng thẳng xuống mặt đất và có độ rộng không đổi. Rác cho vào ống có hình dạng của đa giác. Trước khi thả, rác có thể được xoay để có thể bỏ vừa vào ống. Trong quá trình rơi từ trên tầng xuống thùng rác, giả sử rằng rác rơi tự do, không bị lực cản không khí, không tự xoay.

Cách giải: Tìm bao lồi của đa giác. Với mỗi cạnh a của bao lồi, thử đặt cạnh đó dọc theo (song song) chiều cao ống rác. Trong phép thử này, chiều rộng ống rác tối thiểu phải bằng khoảng cách lớn nhất từ một điểm thuộc bao lồi đến cạnh a. Qua tất cả các lần thử, ta sẽ chọn chiều rộng nào nhỏ nhất để kết luận đáp án.

UVa 10927 - Bright Lights

Tóm tắt đề bài:

Có n cột laser với độ cao riêng đặt trên mặt phẳng tọa độ Oxy, tất cả tia laser đều có phương song song với mặt phẳng, hướng về cột thu tín hiệu ở gốc tọa độ. Đèn laser được đặt trên đỉnh cột, và có thể sẽ bị các cột đèn khác chắn nếu cột đó nằm trên đường đi của tia sáng. Tìm xem có bao nhiêu tia đến được gốc tọa độ?

Cách giải: Sắp xếp các điểm theo giống như trong thuật toán quét Graham, trong đó `pivot` là gốc tọa độ. Duyệt theo thứ tự đã sắp xếp, chúng ta nhận điểm P_0; và chỉ nhận điểm $P_i (i > 0)$ nào thỏa mãn $\alpha_i > \alpha_{i-1}$.

5 Kết luận

Chuyên đề này dựa trên công trình nghiên cứu [2]. Tuy nhiên thay vì trình bày ngắn gọn ý tưởng như quyển sổ tay, chuyên đề đã được đầu tư trong việc phân tích cặn kẽ các giải thuật: về bản chất và chứng minh tính đúng đắn của nó. Tôi tin rằng, đây là cách để người đọc trở nên tường tận một vấn đề trước khi đặt tay lên bàn phím.

Tài liệu

[1] Nguyễn Minh Hà, Nguyễn Xuân Bình – THPT Chuyên Đại học Sư phạm Hà Nội. *Bài tập nâng cao và một số chuyên đề hình học 10.*

[2] Steven Halim, Felix Halim – Đại học Quốc gia Singapore. *Competitive Programming 3 — The New Lower Bound of Programming Contests.*

[3] Cộng đồng LaTeX Ấn Độ. *LaTeX Tutorial.*

[4] Nguyễn Hoàng Phú – THPT Chuyên Lý Tự Trọng, Cần Thơ. *Giáo viên cố vấn lập trình.*

[5] Nguyễn Văn Thắng – THPT Lưu Hữu Phước, Cần Thơ. *Giáo viên cố vấn hình học.*

[6] Calvin Lin. "Why does cross product tell us about clockwise or anti-clockwise rotation?". *Diễn đàn Math Stack Exchange.* 23/01/2013. math.stackexchange.com

[7] Wikimedia Foundation. *Bách khoa toàn thư mở Wikipedia.* wikipedia.org