Algoritmos

Geometría Computacional

MSc Edson Ticona Zegarra

Preparación ICPC Regionales

Contenido

Puntos

Listing 1: Puntos en C++

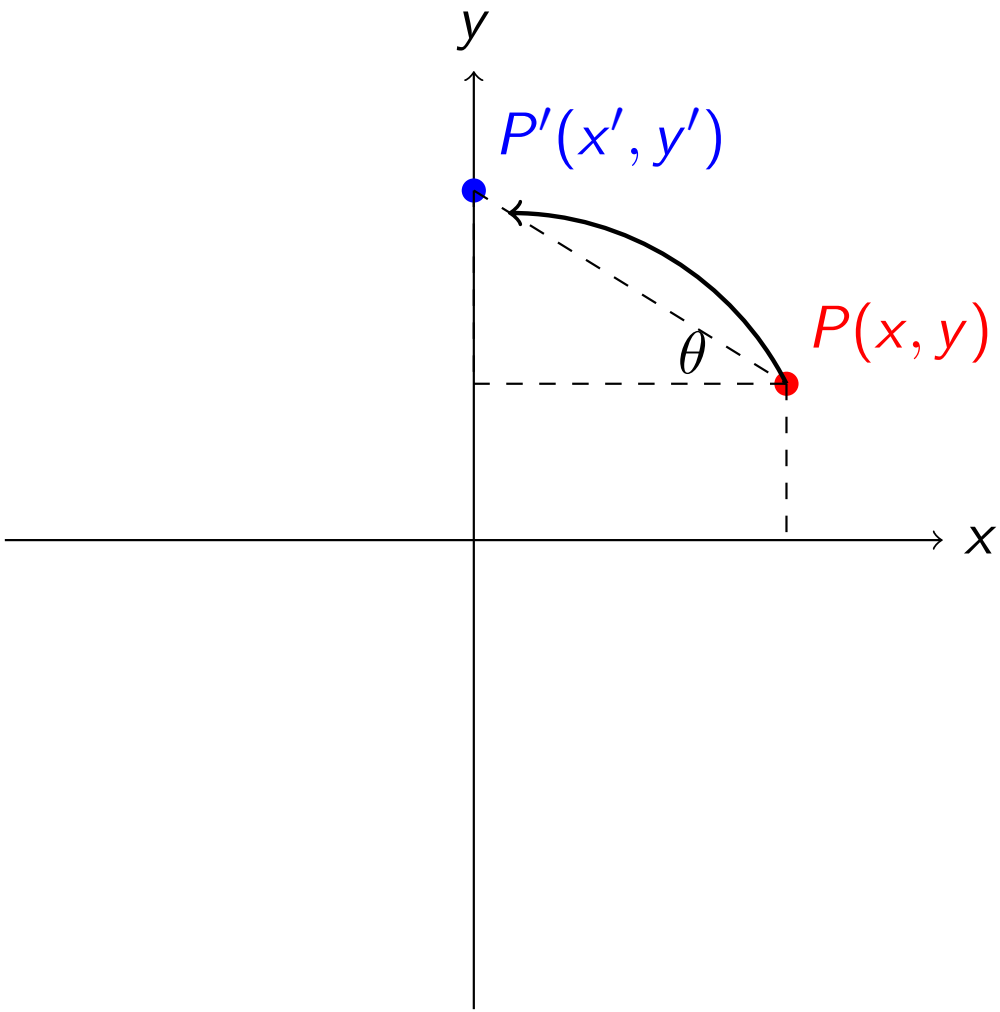
```
#include <iostream>
#include <cmath>
#define EPS 1e-6

using namespace std;

struct point_i {
  int x, y;
  point_i() { x = y = 0; }
  point_i(int _x, int _y) : x(_x), y(_y) {}
};
```

Puntos

Listing 2: Puntos double

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#define EPS 1e-6 // 10^-6 => 0.000001

using namespace std;

struct point {
  double x, y;
  point() { x = y = 0; }
  point(double _x, double _y) : x(_x), y(_y) {}

  bool operator < (point p2) const {
    if ( fabs(x-p2.x) > EPS ) // no son "iguales"
      return x < p2.x;
    return y < p2.y;
  }

  // operator overloading: sobrecarga de operadores
```

Rotación



Matriz de rotación

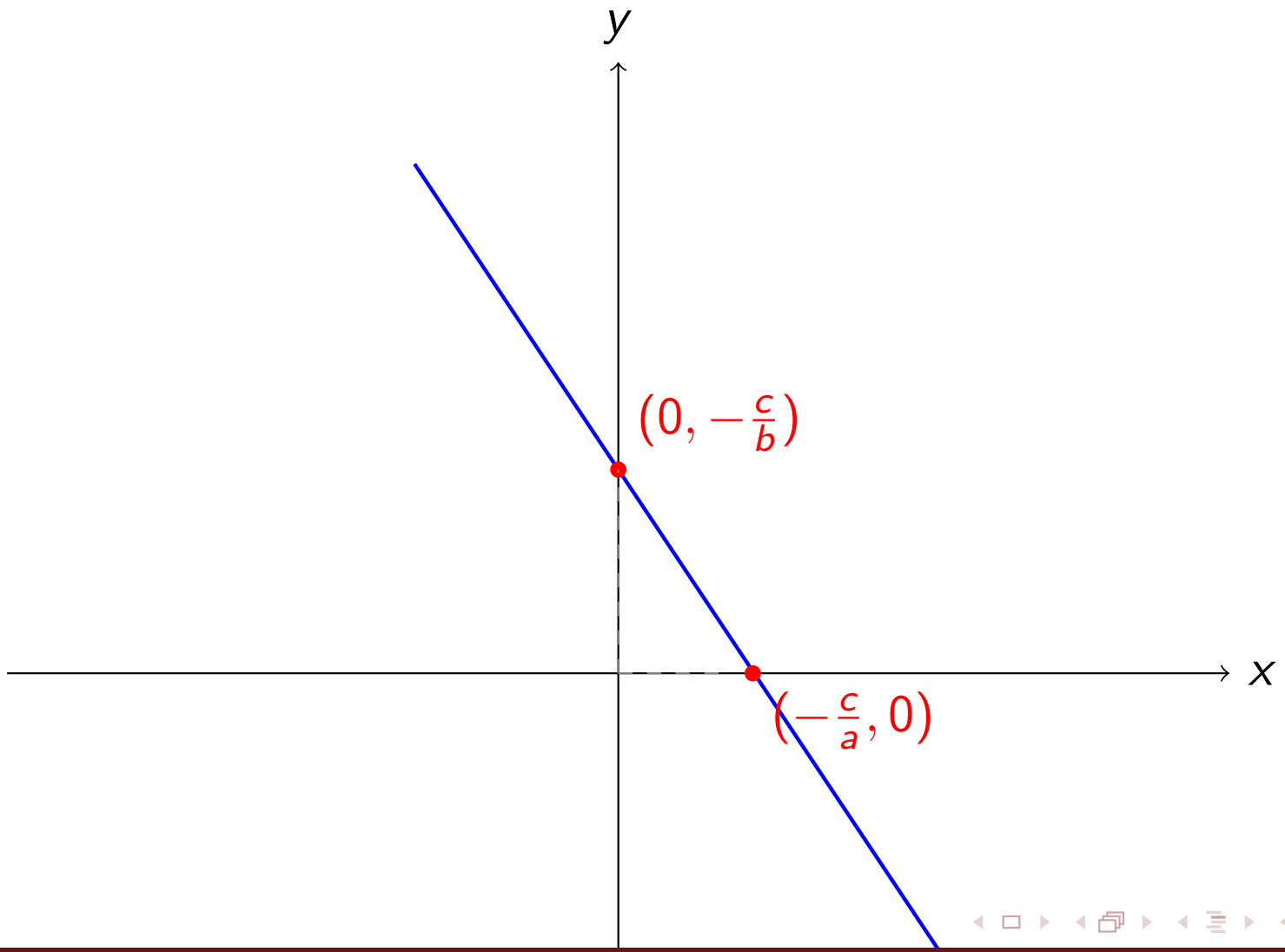
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotación

Listing 3: Rotación de puntos

```
#include <iostream>
#include <cmath>
#define EPS 1e-6 // 10^-6 => 0.000001

// cpp math library uses radians
double degrees2radians(double d) {
  return d*acos(-1.0) / 180;
}

point rotation(const point &p, double theta){
  double rad = degrees2radians(theta);
  return point(p.x*cos(rad) - p.y*sin(rad), p.x*sin(rad) + p.y*cos
}

using namespace std;
struct point {
  double x, y;
  point() { x = y = 0; }
  point(double _x, double _y) : x(_x), y(_y) {}
```

Linea: $ax + by + c = 0$



Lineas

Listing 4: Lineas en C++

```
#include <cmath>
#include <iostream>
#define EPS 1e-6 // 10^-6 => 0.000001

using namespace std;

struct line {
  double a, b, c; // ax + by + c = 0
};

struct point {
  double x, y;
  point() { x = y = 0; }
  point(double _x, double _y) : x(_x), y(_y) {}

  bool operator < (point p2) const {
    if ( fabs(x-p2.x) > EPS ) // no son "iguales"
      return x < p2.x;
    return y < p2.y;
  }
```

Vectores

Listing 5: Vectores

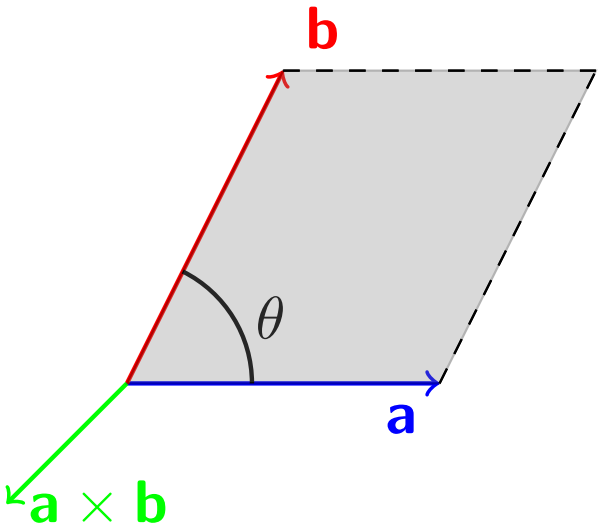
```
struct vec {
  double x, y; // magnitudes
  vec(double _x, double _y) : x(_x), y(_y) {}
};

vec toVec(const point &a, const point &b) {
  return vec(b.x-a.x, b.y-a.y); // vector a->b
}

vec scale(const vec &v, double s) {
  return vec(v.x*s, v.y*s);
}

point translate(const point &p, const vec &v) {
  return point(p.x+v.x, p.y+v.y);
}

double norm_sq(vec v) {
  return v.x*v.x + v.y*v.y;
}
```

Producto punto

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \sum_{i=1}^{n} a_i b_i$$

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos\theta,$$

Producto cruz



$$\text{Area} = \|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\|\|\mathbf{b}\| \sin\theta$$

Producto cruz

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

$$\mathbf{a} \times \mathbf{b} = (a_2 b_3 - a_3 b_2)\mathbf{i} - (a_1 b_3 - a_3 b_1)\mathbf{j} + (a_1 b_2 - a_2 b_1)\mathbf{k}.$$

Producto punto y cruz

Listing 6: Producto punto y cruz

```
#include <cmath>

double dot(const vec &a, const vec &b) {
  return a.x*b.x + a.y*b.y;
}

double cross(const vec &a, const vec &b) {
  return a.x*b.y - a.y*b.x;
}

double angle(const point &a, const point &b, const point &c) {
  vec ab = toVec(a,b);
  vec bc = toVec(b,c);
  return acos(dot(ab, bc) / sqrt(norm_sq(ab) * norm_sq(bc)));
}
```

CCW Test

Listing 7: Test CCW

```
#include <iostream>
#include <cmath>
#define EPS 1e-6 // 10^-6 => 0.000001

using namespace std;

struct point {
  double x, y;
  point() { x = y = 0; }
  point(double _x, double _y) : x(_x), y(_y) {}

  bool operator < (point p2) const {
    if ( fabs(x-p2.x) > EPS ) // no son "iguales"
      return x < p2.x;
    return y < p2.y;
  }

  // operator overloading: sobrecarga de operadores
  bool operator == (const point &p2) const {
    return (fabs(x-p2.x) < EPS) && (fabs(y-p2.y) < EPS);
```

Polygons

- Representación canónica: lista de vértices del polígono en orden CW o CCW.

Polygons

- Representación canónica: lista de vértices del polígono en orden CW o CCW.
- Se asume que el primer punto está conectado con el último punto

Perímetro

Listing 8: Perímetro del polígono

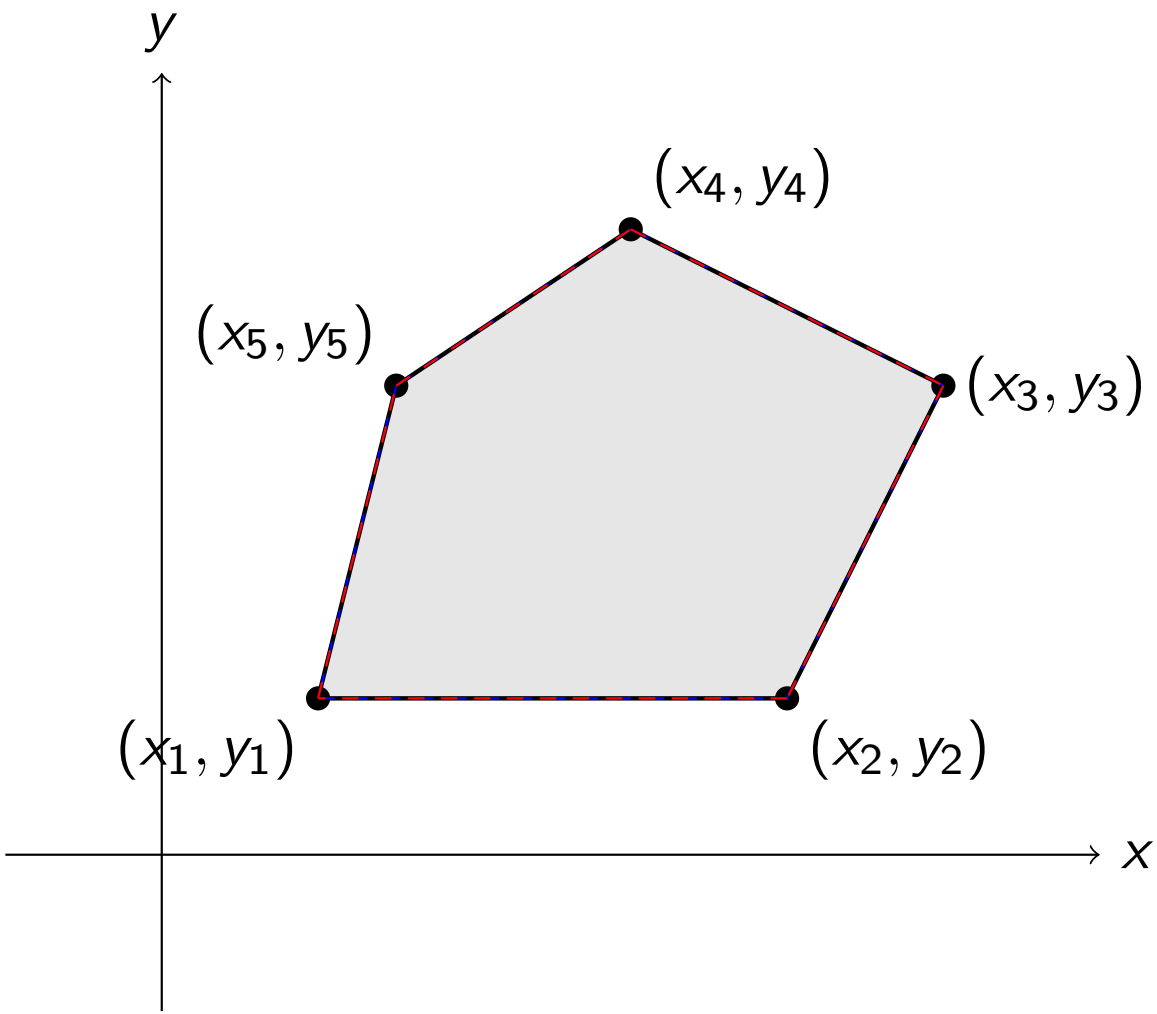
```
double perimeter(vector <point> &P) {
  double p = 0;
  int n = P.size();
  for(int i=0; i<n; i++)
    p += dist(P[i], P[(i+1)%n]);
  return p;
}
```

Área

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})\right|$$

Donde A es el área de un polígono definido por n vértices con coordenadas (x_i, y_i), con $i = 1, ..., n$

Área



Área

Listing 9: Área del polígono

```
double area(vector <point> &P) {
  double a = 0;
  int n = P.size();
  for(int i=0; i<n; i++)
    a += (P[i].x*P[(i+1)%n].y - P[(i+1)%n].x*P[i].y);
  return fabs(a)/2.0;
}
```

Convexidad

- Se dice que un polígono es convexo si para todo segmento formado por dos puntos interiores cualesquiera, el segmento es también interno al polígono

Convexidad

- Se dice que un polígono es convexo si para todo segmento formado por dos puntos interiores cualesquiera, el segmento es también interno al polígono
- Un polígono puede ser cóncavo o convexo

Convexidad

- Se dice que un polígono es convexo si para todo segmento formado por dos puntos interiores cualesquiera, el segmento es también interno al polígono
- Un polígono puede ser cóncavo o convexo
- Los polígonos convexos tienen propiedades interesantes y son ampliamente estudiantes en diversas áreas

Convexidad

Listing 10: Test de convexidad

```
bool isConvex(const vector<point> &P) {
  int n = P.size();
  bool firstTurn = ccw(P[0], P[1], P[2]);
  for (int i=1; i<n; i++)
    if (cww(P[i], P[(i+1)%n], P[(i+2)%n]) != firstTurn)
      return false;
  return true
}
```