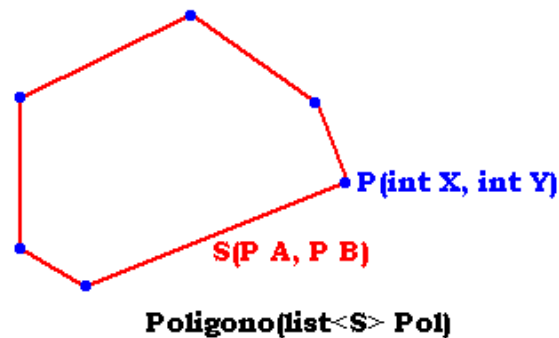
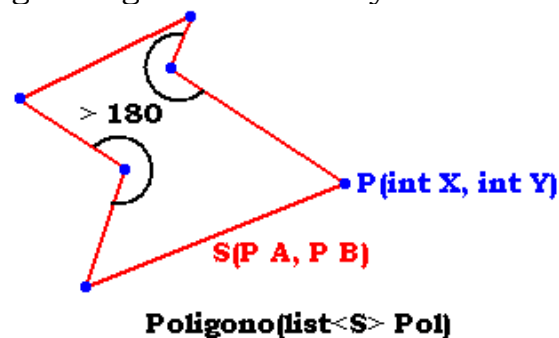


Análisis y Diseño de Algoritmos II – Algoritmos Geométricos

- Algoritmos Geométricos:
Además de resolver problemas geométricos, en la actualidad tiene otras aplicaciones como gráfica, robótica y diseño.
- Entidades Geométricas:
 - Punto: entidad más básica formada por una coordenada X y una Y.
 - Segmento o línea: Formado por 2 puntos A y B.
 - Polígono: Formado por n segmentos. Ejemplo: un cuadrado.
 - Poliedro: formado por n polígonos los cuales le dan profundidad a la figura. Ejemplo: un cubo.
- Polígono Convexo:
Es el que tiene todos sus ángulos internos menores a 180° .



- Polígono No Convexo:
Es el que tiene algún ángulo interno mayor a 180° .



- Problemas clásicos de Geometría:
 - Verificar si un punto se encuentra a izquierda o derecha de un segmento.
 - Verificar corte de segmentos y punto de corte.
 - Verificar si un punto se encuentra dentro de un polígono.
 - Calcular el área encerrada por un polígono.
 - Encontrar el convex hull. (menor polígono convexo que encierra un conjunto de puntos).
 - Encontrar el par de puntos cuya distancia es la menor de un conjunto.

➤ **Verificar si un punto se encuentra a izquierda o derecha de un segmento:**

Conociendo las propiedades del producto vectorial es muy simple saber si un punto esta a derecha o a izquierda de un segmento. Debemos seguir los siguientes pasos:

1. Trasladamos todo al origen, generando 2 vectores, uno del primer punto del segmento a segundo, y otro del primer punto del segmento al punto que quiero saber si se encuentra a izquierda o a derecha.

Dado un segmento y un punto creo los 2 vectores trasladandolos al origen

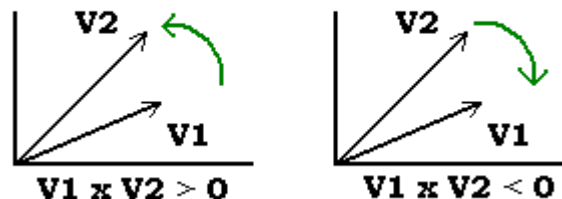


Para hacer el traslado y generar los vectores:

$$V1 = P - S.p1$$

$$V2 = S.p2 - S.p1$$

Aplicando producto vectorial



2. Aplicar producto vectorial, entre los vectores generados. Y sacar las conclusiones desde ahí, como muestra la figura.

En nuestro caso $V1 \times V2$, como el punto esta a la derecha nos da positivo, siguiendo la regla del tirabuzón. Si nos diese negativo el punto estaría a izquierda y si nos diese 0 el punto estaría sobre el segmento.

```
int izq_der(segmento S, punto P){
    punto V1;
    punto V2;
    V1.x = P.x - S.p1.x; V1.y = P.y - S.p1.y;
    V2.x = S.p2.x - S.p1.x; V2.y = S.p2.y - S.p1.y;
    int producto_vectorial = ((V1.x * V2.y)-(V2.x * V1.y));
    return producto_vectorial;// > 0 a derecha, < 0 izquierda, = 0 en el segmento
} //este producto vectorial me estaria dando solo la coordenada en z
//que es la que me interesa saber si es positiva o negativa.
```

➤ **Verificar corte de segmentos y punto de corte:**

Para el problema de corte de segmentos existen diferentes algoritmos como:
Box Check:

```
bool box_check(segmento S, segmento Q){
    punto Smax; punto Smin;
    punto Qmax; punto Qmin;
    Smax.x = max(S.p1.x, S.p2.x);
    Smax.y = max(S.p1.y, S.p2.y);
    Smin.x = min(S.p1.x, S.p2.x);
    Smin.y = min(S.p1.y, S.p2.y);
    Qmax.x = max(Q.p1.x, Q.p2.x);
    Qmax.y = max(Q.p1.y, Q.p2.y);
    Qmin.x = min(Q.p1.x, Q.p2.x);
    Qmin.y = min(Q.p1.y, Q.p2.y);
    bool hay_corte = ((Smax.x >= Qmin.x) && (Qmax.x >= Smin.x)
        && (Smax.y >= Qmin.y) && (Qmax.y >= Smin.y));
    return hay_corte;
} //retorna 1 en caso de que halla corte
```

Verificar mismo lado:

Verifica que ambos puntos del segmento se encuentren del mismo lado utilizando la función `der_izq`.

```
bool ver_se_cortan(segmento S, segmento Q){
    int lado_p1 = izq_der(S, Q.p1);
    int lado_p2 = izq_der(S, Q.p2);
    if((lado_p1 * lado_p2) <= 0){
        lado_p1 = izq_der(Q, S.p1);
        lado_p2 = izq_der(Q, S.p2);
        if((lado_p1 * lado_p2) <= 0)
            return 1; //se cortan
    }
    return 0; //no se cortan
} //cuando pregunto ((lado_p1 * lado_p2) <= 0) verifico si estan
//del mismo lado, es porque o son los 2 positivos o son los 2 negativos.
```

Verificar por la ecuación de la recta:

La ventaja de este algoritmo es que además de decidir si hay o no hay corte, en caso que lo haya devuelve el punto de corte.

```
/*calcula la distancia del punto, a ambos puntos del segmento, si la
suma de estas, es la misma a la distancia entre los puntos del
segmento entonces, el punto esta sobre este.*/
bool pertenece(segmento S, punto P){
    double distancia_P_1 = sqrt(pow(S.p1.x - P.x) + pow(S.p1.y - P.y));
    double distancia_P_2 = sqrt(pow(S.p2.x - P.x) + pow(S.p1.y - P.y));
    double distancia_1_2 = sqrt(pow(S.p1.x - S.p2.x) + pow(S.p1.y - S.p2.y));
    return ((distancia_P_1 + distancia_P_2) == distancia_1_2);
}

bool ver_por_ecuacion(segmento S, segmento Q, punto &X){//Y = MX + B
    double M1 = (S.p2.y - S.p1.y)/(S.p2.x - S.p1.x);//pendiente 1
    double M2 = (Q.p2.y - Q.p1.y)/(Q.p2.x - Q.p1.x);//pendiente 2
    double B1 = S.p1.y - M1 * S.p1.x;//ordenada 1
    double B2 = Q.p1.y - M2 * Q.p1.x;//ordenada 2
    double XE = (B2 - B1)/(M1 - M2);//X de encuentro, igualando las rectas
    if(pertenece(S,XE) && pertenece(Q,XE)){
        X.x = XE;
        X.y = M1 * XE + B1;
        return 1;
    }
    X.x = INFINITO;
    X.y = INFINITO;
    return 0;
}
```

➤ **Verificar si un punto se encuentra dentro de un polígono:**

Para este problema se plantean diferentes soluciones según la información que tengamos acerca del polígono.

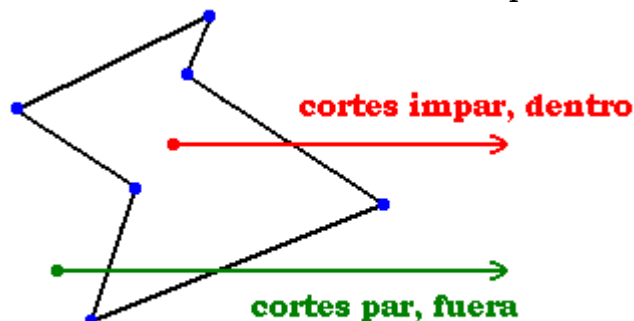
Verificando siempre a izquierda:

Dado un polígono convexo, y suponiendo que los segmentos vienen ordenados de manera tal que van girando en sentido contrario a las agujas del reloj, y donde termina un segmento es el principio del otro, hasta llegar al segmento desde el que partí, lo que tendría que verificar es si el punto se encuentra siempre a izquierda de los segmentos del polígono.

```
bool siempre_a_izquierda(list<segmento> Poligono, punto P){
    list<segmento>::iterator S;
    for(S = Poligono.begin(); S != Poligono.end(); S++)
        if(izq_der(*S,P) >= 0)//significa que esta a derecha o sobre el segmento
            return 0;
    return 1;
}
```

Test del Rayo:

Consiste en trazar una semirecta, generalmente horizontal, desde el punto hasta hasta el infinito, y contar la cantidad de veces que corta al polígono. Si la cantidad es par entonces se encuentra fuera del polígono, y si se encuentra dentro la cantidad de cortes será impar.

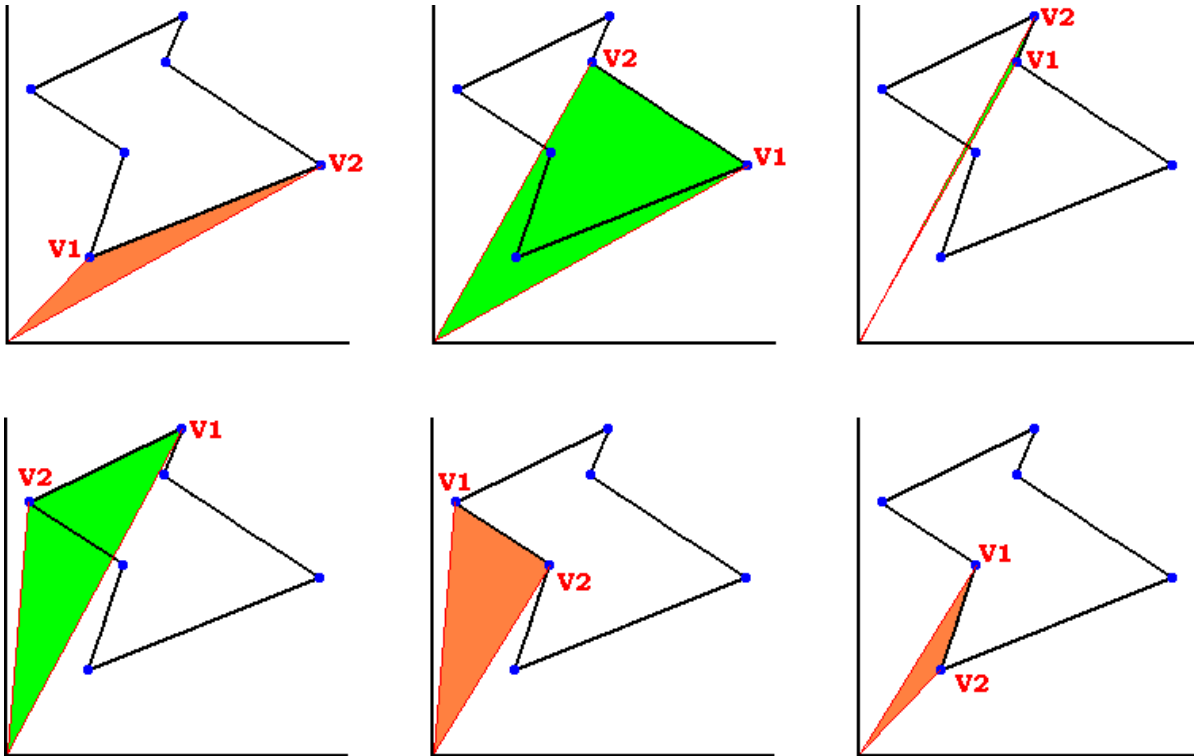


```
bool test_del_rayo(list<segmento> Poligono, punto P){
    int cortes = 0;
    segmento semirecta;
    semirecta.p1.x = P.x; semirecta.p1.y = P.y;
    semirecta.p2.x = INFINITO; semirecta.p1.y = P.y;
    list<segmento>::iterator S;
    for(S = Poligono.begin(); S != Poligono.end(); S++)
        if(max(*S.p1.x,*S.p2.x) > semirecta.p1.x)
            if(ver_se_cortan(*S,Q)
                cortes++;
    if(mod(cortes,2) == 0)//la cantidad de cortes es par
        return 0;//esta fuera
    return 1;//esta dentro
}
```

➤ **Calcular el área encerrada por un polígono:**

Para resolver este problema, lo que debemos hacer es triangular el polígono, calcular el área de cada triángulo y sumarlas.

El producto vectorial además de decirnos si un punto está a izquierda o a derecha nos da el área encerrada por el triángulo formado por los 2 vectores.



**Al sumar todas las áreas de los triángulos nos da el área del polígono.
 Los productos vectoriales de las imágenes verdes nos dan positivos.
 Los productos vectoriales de las imágenes naranja nos dan negativos.**

Por comodidad, calculamos los vectores desde el origen a cada punto del polígono. Pero podríamos calcularlos desde cualquier punto.

```
double area_triangulo(punto V1, punto V2){
    double producto_vectorial = ((V1.x * V2.y)-(V2.x * V1.y));
    return producto_vectorial;
}

double area_poligono(list<segmento> Poligono){
    punto origen; punto V1; punto V2;
    origen.x = 0; origen.y = 0;
    double area = 0;
    list<segmento>::iterator S;
    for(S = Poligono.begin(); S != Poligono.end(); S++){
        V1.x = (*S.p1.x - origen.x); V1.y = (*S.p1.y - origen.y);
        V2.x = (*S.p2.x - origen.x); V2.y = (*S.p2.y - origen.y);
        area += area_triangulo(V1,V2);
    }
}
```

- **Encontrar el convex hull. (menor polígono convexo que encierra un conjunto de puntos).**

Para este problema se plantean 2 algoritmos:

Graham's Scan