11

GEOMETRÍA COMPUTACIONAL

Sin embargo, demostré sin sombra de duda y con una lógica geométrica que existía una clave.

—de The Caine Mutiny

La **geometría computacional** se encarga del diseño y análisis de algoritmos para resolver problemas geométricos. Los algoritmos geométricos eficientes son muy útiles en campos como la graficación por computadora, la estadística, el procesamiento de imágenes y en el diseño con integración a muy grande escala (very-large-scale-integration, VLSI). En este capítulo presentaremos una introducción a este fascinante tema.

El problema del par más cercano proporciona un ejemplo de problema en geometría computacional: Dados n puntos en el plano, determinar el par más cercano. Además de este problema, consideraremos el problema de determinar la cubierta convexa.

11.1 EL PROBLEMA DEL PAR MÁS CERCANO

El **problema del par más cercano** se puede enunciar fácilmente: Dados n puntos en el plano, determinar un par más cercano (véase la figura 11.1.1). (Decimos un par más cercano pues es posible que varios pares tengan la misma distancia mínima.) Nuestra medida de distancia es la distancia euclidiana ordinaria.

Una forma de resolver este problema es enumerar la distancia entre cada par y elegir el mínimo en esta lista de distancias. Como existen $C(n, 2) = n(n-1)/2 =$

† Esta sección puede omitirse sin pérdida de continuidad.

$\Theta(n^2)$ pares, el tiempo necesario para este algoritmo "enumera todo" es $\Theta(n^2)$. Podemos hacer algo mejor: daremos un algoritmo "divide y vencerás" para obtener el par más cercano, cuyo tiempo en el peor de los casos es $\Theta(n \lg n)$. Primero analizaremos el algoritmo y luego daremos una descripción más precisa de éste mediante un seudocódigo.

Nuestro algoritmo comienza determinando una recta vertical l que divide a los puntos en dos conjuntos aproximadamente iguales (véase la figura 11.1.1). [Si n es par, dividimos los puntos en dos partes, cada una de las cuales tiene $n/2$ puntos. Si n es impar, separamos los puntos en dos partes, una con $(n+1)/2$ puntos y la otra con $(n-1)/2$ puntos.]

Ahora, resolvemos el problema de manera recursiva para cada una de las partes. Sea δ_L la distancia entre un par más cercano en la parte izquierda; sea δ_R la distancia entre un par más cercano en la parte derecha; y sea

$$\delta = \text{mín}\{\delta_L, \delta_R\}.$$

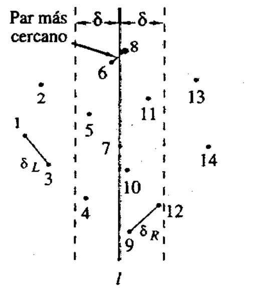

FIGURA 11.1.1 n puntos en el plano. El problema consiste en determinar un par más cercano. Para este conjunto, el par más cercano es 6 y 8. La línea$l$ divide a los puntos en dos partes aproximadamente iguales. El par más cercano en la mitad izquierda es 1 y 3, que están a una distancia δ_L. El par más cercano en la mitad derecha es 9 y 12, que están a una distancia δ_R. Cualquier par (por ejemplo, 6 y 8) cuya distancia sea menor que $\delta = \text{mín}\{\delta_L, \delta_R\}$ debe estar en la franja vertical de ancho 2δ con centro en l.

Por desgracia, podría ocurrir que δ no fuese la distancia entre un par más cercano en el conjunto original de puntos, pues un par de puntos, uno de la parte izquierda y el otro de la parte derecha, podrían estar a una distancia menor que δ (véase la figura 11.1.1). Así, debemos considerar las distancias entre los puntos en lados opuestos de la recta l.

Observemos primero que si la distancia entre un par de puntos es menor que δ, entonces los puntos deben estar en la franja vertical de ancho 2δ con centro en l (véase la figura 11.1.1). (Cualquier punto que no esté en esta franja estará al menos a distancia δ de cualquier punto del otro lado de l.) Así, podemos restringir nuestra búsqueda de un par a una distancia menor que δ a los puntos de esta franja.

Si existen n puntos en la franja y verificamos *todos* los pares de la franja, el tiempo necesario para procesar los puntos de la franja en el peor de los casos es $\Theta(n^2)$. En este caso, el tiempo de nuestro algoritmo en el peor de los casos es $\Omega(n^2)$, que al menos es tan malo como el de la búsqueda exhaustiva; así, debemos evitar la verificación de todos los pares de la franja.

Ordenamos los puntos de la franja en orden creciente de sus coordenadas y y luego analizamos los puntos en este orden. Al examinar un punto p de esta franja, cualquier punto q posterior a p cuya distancia a p sea menor que δ debe estar estrictamente dentro o en la base del rectángulo de altura δ cuya base contiene a p y cuyos lados verticales están a una distancia δ de l (véase la figura 11.1.2). (No necesitamos calcular la distancia entre p y los puntos por debajo de p. Estas distancias ya se habrán considerado anteriormente, pues estamos examinando los puntos en orden creciente de sus coordenadas y.) Mostraremos que este rectángulo contiene a lo más ocho puntos, incluyendo a p, de modo que si calculamos las distancias entre p y los siguientes siete puntos en la franja, podremos estar seguros de que calcularemos las distancias entre p y todos los puntos del rectángulo. Por supuesto, si existen menos de siete puntos después de p en la lista, calculamos las distancias entre p y los puntos restantes. Al restringir la búsqueda en la franja de esta forma, el tiempo necesario para procesar los puntos de la franja es $O(n)$. (Como a lo más existen n puntos en la franja, el tiempo necesario para procesar los puntos de la franja es a lo más $7n$.)

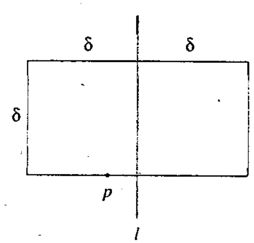

FIGURA 11.1.2 Cualquier punto q posterior a p cuya distancia a p sea menor que δ debe estar dentro del rectángulo.

Mostraremos que el rectángulo de la figura 11.1.2 contiene a lo más ocho puntos. La figura 11.1.3 muestra el rectángulo de la figura 11.1.2 dividido en ocho cuadrados iguales. Observe que la longitud de una diagonal de un cuadrado es

$$\left(\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2\right)^{1/2} = \frac{\delta}{\sqrt{2}} < \delta;$$

así, cada cuadrado contiene a lo más un punto. Por tanto, el rectángulo $2\delta \times \delta$ contiene a lo más ocho puntos.

EJEMPLO 11.1.1

Ahora mostraremos la forma en que el algoritmo del par más cercano determina un par más cercano para los datos de la figura 11.1.1.

El algoritmo comienza determinando una recta vertical l que separa a los puntos en dos partes iguales,

$$S_1 = \{1, 2, 3, 4, 5, 6, 7\}, \qquad S_2 = \{8, 9, 10, 11, 12, 13, 14\}.$$

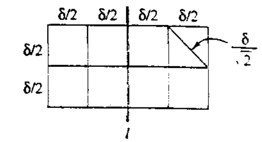

FIGURA 11.1.3 El rectángulo grande contiene a lo más ocho puntos, pues cada cuadrado contiene a lo más un punto.

Para estos datos, existen muchas opciones posibles para la línea divisoria. La recta particular aquí elegida pasa por el punto 7.

Ahora, resolvemos el problema de manera recursiva para S_1 y S_2. El par más cercano de puntos en S_1 es 1 y 3. Sea δ_L la distancia entre los puntos 1 y 3. El par más cercano de puntos en S_2 es 9 y 12. Sea δ_R la distancia entre los puntos 9 y 12. Sea

$$\delta = \text{mín}\{\delta_L, \delta_R\} = \delta_L.$$

Ahora ordenamos los puntos de la franja vertical de ancho 2δ con centro en l en orden creciente de sus coordenadas y:

$$9, \quad 12, \quad 4, \quad 10, \quad 7, \quad 5, \quad 11, \quad 6, \quad 8.$$

Ahora analizamos los puntos en este orden. Calculamos las distancias entre cada punto y los siguientes siete puntos, o entre cada punto y los puntos restantes si existen menos de siete puntos después de él.

Primero calculamos las distancias de 9 a cada uno de los puntos 12, 4, 10, 7, 5, 11 y 6. Como cada una de estas distancias es mayor que δ, en este punto no hemos encontrado un par más cercano.

Ahora calculamos las distancias de 12 a cada uno de los puntos 4, 10, 7, 5, 11, 6 y 8. Como cada una de estas distancias es mayor que δ, en este punto aún no hemos encontrado un par más cercano.

Luego calculamos las distancias de 4 a cada uno de los puntos 10, 7, 5, 11, 6 y 8. Como cada una de estas distancias es mayor que δ, en este punto aún no hemos encontrado un par más cercano.

Ahora calculamos las distancias de 10 a cada uno de los puntos 7, 5, 11, 6 y 8. Como la distancia entre 10 y 7 es menor que δ, hemos descubierto un par más cercano. Actualizamos δ como la distancia entre los puntos 10 y 7.

Ahora calculamos las distancias de 7 a cada uno de los puntos 5, 11, 6 y 8. Como cada una de estas distancias es mayor que δ, no hemos encontrado un par más cercano.

Ahora calculamos las distancias de 5 a cada uno de los puntos 11, 6 y 8. Como cada una de estas distancias es mayor que δ, no hemos encontrado un par más cercano.

Ahora calculamos las distancias de 11 a cada uno de los puntos 6 y 8. Como cada una de estas distancias es mayor que δ, no hemos encontrado un par más cercano.

Ahora calculamos la distancia de 6 a 8. Como la distancia entre 6 y 8 es menor que δ, hemos descubierto un par más cercano. Actualizamos δ como la distancia entre los puntos 6 y 8. Como ya no hay más puntos en la franja por considerar, el algoritmo termina. El par más cercano es 6 y 8 y la distancia entre ellos es δ. □

Antes de enunciar formalmente el algoritmo del par más cercano, debemos resolver varios detalles técnicos.

Para terminar la recursión, verificamos la cantidad de puntos en la entrada y si existen tres o menos puntos, determinamos un par más cercano en forma directa. La separación de los datos y el uso de la recursión sólo cuando existen cuatro o más puntos nos garantiza que cada una de las dos partes contiene al menos un par de puntos y, por tanto, que existe un par más cercano en cada parte.

Antes de llamar al procedimiento recursivo, ordenamos todo el conjunto de puntos mediante sus abscisas. Esto facilita la separación de los puntos en dos partes casi iguales.

Utilizamos el ordenamiento por fusión (véase la sección 5.3) para ordenar según las coordenadas *y*. Sin embargo, en vez de ordenar cada vez, examinamos los puntos en la franja vertical, y suponemos, como en el ordenamiento por fusión, que cada mitad está ordenada según sus coordenadas *y*. Luego, basta realizar la fusión de las dos mitades para ordenar todos los puntos según sus coordenadas *y*.

Ahora enunciaremos formalmente el algoritmo del par más cercano. Para que nuestra descripción sea más sencilla, nuestra versión produce como salida la distancia entre un par más cercano, pero no el propio par más cercano. Dejaremos esta mejora como ejercicio (ejercicio 5).

ALGORITMO 11.1.2 ***Determinación de la distancia entre un par de puntos más cercanos***

Entrada: $p_1, \ldots, p_n$ ($n \geq 2$ puntos en el plano)

Salida: δ, la distancia entre un par de puntos más cercanos

```
procedure closest_pair(p, n)
  ordenar p_1, ..., p_n por su abscisa
  return (rec_cl_pair(p, 1, n))
end closest_pair
procedure rec_cl_pair(p, i, j)
  // La entrada es la sucesión p_i, ..., p_j de puntos en el plano
  // ordenados según su abscisa.
  // Al concluir rec_cl_pair, la sucesión queda ordenada
  // según su coordenada y.
  // rec_cl_pair regresa la distancia entre un par más cercano
  // en la entrada.
  // Denotemos la abscisa del punto p como p.x.
  // caso trivial (3 o menos puntos)
  if j − i < 3 then
    begin
    ordenar p_i, ..., p_j según su coordenada y
    determinar de manera directa la distancia δ entre un par más cercano
    return(δ)
    end
  // dividir
  k := ⌊(i + j)/2⌋
  l := p_k.x
  δ_L := rec_cl_pair(p, i, k)
  δ_R := rec_cl_pair(p, k + 1, j)
  δ := mín{δ_L, δ_R}
  // ahora, p_i, ..., p_k están ordenados según su coordenada y
  // p_{k+1}, ..., p_j están ordenados según su coordenada y
  se realiza la fusión de p_i, ..., p_k y p_{k+1}, ..., p_j según su coordenada y
  // suponga que el resultado de la fusión se guarda de nuevo en
  // p_i, ..., p_j
  // ahora, p_i, ..., p_j están ordenados según su coordenada y
  // se guardan los puntos de la franja vertical en v
  t := 0
  for k := i to j do
    if p_k.x > l − δ and p_k.x < l + δ then
      begin
      t := t + 1
      v_t := p_k
      end
  // los puntos en la franja son v_1, ..., v_t
  // se busca el par más cercano en la franja
  // se compara cada uno con los siguientes siete puntos
  for k := 1 to t − 1 do
    for s := k + 1 to min {t, k + 7} do
      δ := min {δ, dist(v_k, v_s)}
  return (δ)
end rec_cl_pair
```


Mostraremos que el tiempo del algoritmo del par más cercano en el peor de los casos es $\Theta(n \lg n)$. El procedimiento *closest_pair* comienza ordenando los puntos según su abscisa. Si utilizamos un ordenamiento óptimo (por ejemplo, ordenamiento por fusión), el tiempo de ordenamiento en el peor de los casos será $\Theta(n \lg n)$. A continuación, *closest_pair* llama a *rec_cl_pair*. Sea a_n el tiempo en el peor de los casos de *rec_cl_pair* para una entrada de tamaño n. Si $n > 3$, *rec_cl_pair* se llama a sí mismo, con una entrada de tamaño $\lfloor n/2 \rfloor$ y $\lfloor (n+1)/2 \rfloor$. Cada una de las fusiones, localización de los puntos en la franja, y la verificación de las distancias en la franja tarda un tiempo $O(n)$. Así, obtenemos la recurrencia

$$a_n \leq a_{\lfloor n/2 \rfloor} + a_{\lfloor (n+1)/2 \rfloor} + cn, \qquad n > 3.$$

Ésta es la misma recurrencia satisfecha por el ordenamiento por fusión, de modo que podemos concluir que *rec_cl_pair* tiene el mismo tiempo $O(n \lg n)$ que el ordenamiento por fusión, en el peor de los casos. Como el tiempo en el peor de los casos para el ordenamiento de los puntos según su abscisa es $\Theta(n \lg n)$ y el tiempo en el peor de los casos de *rec_cl_pair* es $O(n \lg n)$, el tiempo en el peor de los casos de *closest_pair* es $\Theta(n \lg n)$. En la sección 11.2 mostraremos que cualquier algoritmo que determine un par de puntos más cercanos en el plano tiene un tiempo $\Omega(n \lg n)$ en el peor de los casos; así, nuestro algoritmo es asintóticamente óptimo.

Se puede mostrar (ejercicio 10) que existen a lo más seis puntos en el rectángulo de la figura 11.1.2 al incluir la base y excluir los otros lados. Este resultado es el mejor posible, pues podemos colocar seis puntos en el rectángulo (ejercicio 8). Al considerar las posibles posiciones de los puntos en el rectángulo, D. Lerner y R. Johnsonbaugh han demostrado que basta comparar cada punto en la franja con los siguientes tres puntos (en vez de los siguientes siete). Este resultado es el mejor posible, pues la verificación de los dos puntos siguientes no conduce a un algoritmo correcto (ejercicio 7).

Ejercicios

1. Describa la forma en que el algoritmo del par más cercano determina el par más cercano de puntos si la entrada es (8, 4), (3 11), (12, 10), (5, 4), (1, 2), (17, 10), (8, 7), (8, 9), (11, 3), (1, 5), (11, 7), (5, 9), (1, 9), (7, 6), (3, 7), (14, 7).
2. ¿Qué podría concluir acerca de la entrada del algoritmo del par más cercano si la salida es cero para la distancia entre un par más cercano?
3. Dé un ejemplo de entrada para la cual el algoritmo del par más cercano coloca algunos puntos sobre la línea divisoria l en la mitad izquierda y otros puntos sobre l en la mitad derecha.
4. Explique por qué en ciertos casos, al separar un conjunto de puntos mediante una recta vertical en dos partes casi iguales, es necesario que la línea contenga algunos de los puntos.
5. Escriba un algoritmo del par más cercano que determine un par más cercano, así como la distancia entre el par de puntos.
6. Escriba un algoritmo que determina la distancia entre un par de puntos más cercanos sobre una línea (recta).
7. Dé un ejemplo de entrada para la cual la comparación de cada punto en la franja con los siguientes dos puntos produzca una salida incorrecta.
8. Dé un ejemplo para mostrar que es posible colocar seis puntos en el rectángulo de la figura 11.1.2 al incluir la base y excluir los otros lados.
9. Al calcular las distancias entre un punto p de la franja y los puntos siguientes a él, ¿podemos dejar de calcular las distancias a p si encontramos un punto q tal que la distancia entre p y q sea mayor que δ? Explique.
10. ★ Muestre que existen a lo más seis puntos en el rectángulo de la figura 11.1.2 al incluir la base y excluir los otros lados.
11. Escriba un algoritmo de tiempo $\Theta(n \lg n)$ para determinar la distancia δ entre un par más cercano, de modo que si $\delta > 0$ también determine *todos* los pares que están a distancia δ.
12. Escriba un algoritmo de tiempo $\Theta(n \lg n)$ para determinar la distancia δ entre un par más cercano, y *todos* los pares que están a distancia menor que 2δ.

†11.2 UNA COTA INFERIOR PARA EL PROBLEMA DEL PAR MÁS CERCANO

En la sección 11.1 dimos un algoritmo $\Theta(n \lg n)$ para determinar la distancia entre un par más cercano entre n elementos del plano. En esta sección mostraremos que este resultado es óptimo; es decir, que cualquier algoritmo que determine la distancia entre un par más cercano entre n elementos del plano tiene un tiempo $\Omega(n \lg n)$ en el peor de los casos.

Primero mostraremos que un problema relacionado con el anterior, el de determinar si n elementos son distintos, tiene una cota inferior $\Omega(n \lg n)$ para el tiempo en el peor de los casos. Como un algoritmo del par más cercano se puede utilizar para determinar si n elementos son distintos (n elementos son distintos si y sólo si la distancia entre el par más cercano es distinta de cero), su tiempo en el peor de los casos debe ser al menos tan grande como $\Omega(n \lg n)$, la cota inferior para el problema de los elementos distintos.

† Esta sección puede omitirse sin pérdida de continuidad.

TEOREMA 11.2.1

El tiempo en el peor de los casos para un algoritmo que resuelva el problema de determinar si n números reales son distintos es $\Omega(n \lg n)$.

Demostración. Demostraremos que cualquier algoritmo que resuelva el problema de determinar si n números reales son distintos debe ordenar los números. Como el ordenamiento tiene un tiempo $\Omega(n \lg n)$ en el peor de los casos (teorema 7.7.3), cualquier algoritmo que resuelva el problema de determinar si n números reales son distintos también debe tener un tiempo $\Omega(n \lg n)$ en el peor de los casos.

Supongamos que un algoritmo que resuelva el problema de determinar si n números reales son distintos recibe la entrada

$$x_1, \quad \ldots, \quad x_n$$

donde los x_i son distintos. La salida será "Distintos". Supongamos que los elementos ordenados son

$$x_{k_1}, \quad x_{k_2}, \quad \ldots, \quad x_{k_n}. \tag{11.2.1}$$

Afirmamos que el algoritmo debe comparar x_{k_i} y $x_{k_{i+1}}$ para $i = 1, \ldots, n-1$, de modo que el algoritmo debe "conocer" el orden de la entrada. Estableceremos esta afirmación argumentando por contradicción.

Supongamos que el algoritmo no compara x_{k_j} y $x_{k_{j+1}}$ para alguna j. Alteramos la entrada original $x_1, \ldots, x_n$ cambiando el valor de x_{k_j} para $x_{k_{j+1}}$, pero dejando invariantes las demás x_i. Volvemos a ejecutar el algoritmo. El resultado de cada comparación será igual al de la ejecución original, pues la única comparación cuyo resultado cambiaría implica a x_{k_j} y $x_{k_{j+1}}$ y el algoritmo no compara este par. Así, la salida es nuevamente "Distintos". Ésta es una contradicción, pues ahora la entrada tiene números duplicados. Así, cualquier algoritmo que resuelva el problema de determinar si n números reales son distintos debe comparar x_{k_i} y $x_{k_{i+1}}$ para $i = 1, \ldots, n-1$.

Para completar la demostración, ahora mostraremos la forma de convertir un algoritmo que resuelva el problema de determinar si n números reales son distintos en un algoritmo de ordenamiento. Como el ordenamiento tiene una cota inferior $\Omega(n \lg n)$ en el peor de los casos, esto completará la demostración del teorema.

Sea A un algoritmo que resuelva el problema de determinar si n números reales son distintos. Modificamos A de la siguiente manera. Primero construimos los vértices $1, \ldots, n$. Cada vez que el algoritmo A compare x_j con x_k, colocamos una arista dirigida de j a k, si $x_j < x_k$. Para entradas distintas, si el orden es (11.2.1), hemos mostrado que A debe comparar x_{k_i} y $x_{k_{i+1}}$ para $i = 1, \ldots, n-1$. Así, existe un camino de x_{k_1} y x_{k_n} que proporciona el orden deseado. Podemos determinar este camino de la siguiente forma. Primero localizamos el único vértice sin aristas de entrada. Éste es el vértice k_1 correspondiente a x_{k_1}, el menor elemento de la lista. Eliminamos todas las aristas de salida de k_1. Repetimos este proceso; es decir, localizamos el único vértice sin aristas de entrada. Éste es el vértice k_2 que corresponde a x_{k_2}, el segundo menor elemento de la lista. Continuamos de esta forma para obtener el orden deseado. Como el ordenamiento requiere al menos $C\, n \lg n$ comparaciones (teorema 7.7.3), concluimos que nuestro algoritmo modificado realiza al menos $C\, n \lg n$ comparaciones. Como las modificaciones al algoritmo A no implican la comparación de elementos, este algoritmo modificado tiene exactamente el mismo número de comparaciones que A. Así, el algoritmo A necesita al menos $C\, n \lg n$ comparaciones. Esto concluye la demostración. ■

COROLARIO 11.2.2

El tiempo en el peor de los casos para cualquier algoritmo que determine la distancia entre un par más cercano entre n elementos en el plano es $\Omega(n \lg n)$.

Demostración. Sea t_n el tiempo en el peor de los casos para un algoritmo CP (par más cercano, por sus siglas en inglés) que regrese la distancia entre un par más cercano de n puntos en el plano. Consideremos el siguiente algoritmo que resuelve el problema de determinar si existen duplicados entre n números:

```
procedure dup (x, n)
 // La entrada es x_1, ..., x_n.
 // Se transforma la entrada en puntos del plano.
 for i := 1 to n do
  a_i := (x_i, 0)
 if CP (a, n) = 0 then
  return ("Con duplicados")
 else
  return ("Sin duplicados")
end dup
```

El tiempo t'_n en el peor de los casos para *dup* es el tiempo necesario en el ciclo for, más el tiempo en el peor de los casos para CP; es decir,

$$t'_n = n + t_n.$$

Por el teorema 11.2.1,

$$Cn \lg n \le t'_n.$$

Al combinar estos dos enunciados, obtenemos

$$\Omega(n \lg n) = C\, n \lg n - \le t'_n - n = t_n. \quad ■$$

Ejercicios

1. Escriba un algoritmo que resuelva el problema de determinar si n números reales son distintos. Haga su algoritmo lo más eficiente posible.
2. El *problema del vecino más cercano* es: Dados n puntos S del plano, uno de los cuales se designa como p, y las distancias ordenadas de p a q para todo $q \neq p$, determinar un punto s en S, $s \neq p$, más cercano a p. Muestre que el tiempo en el peor de los casos para un algoritmo que resuelva este problema es $\Omega(\lg n)$.
3. El problema de todos los vecinos más cercanos es: Dados n puntos S del plano, y las distancias de p a q para todo $q \neq p$, para cada punto p en S, determinar un punto q en S, $q \neq p$, más cercano a p. Muestre que el tiempo en el peor de los casos para un algoritmo que resuelva este problema es $\Omega(n \lg n)$.
4. Suponga dada una gráfica dirigida con vértices $1, \ldots, n$, la cual contiene las aristas (p_i, p_{i+1}), $i = 1, \ldots, n-1$, para cierta permutación $p_1, \ldots, p_n$ de $1, \ldots, n$. Suponga también que si (p_i, p_j) es una arista, entonces $i < j$. (Esta situación es análoga a la de la demostración del teorema 11.2.1.) Dé un algoritmo cuya salida sea $p_1, \ldots, p_n$. [Existe un algoritmo cuyo tiempo en el peor de los casos es $\Theta(e + n)$, donde e es el número de aristas en la gráfica.]

11.3 UN ALGORITMO PARA CALCULAR LA CUBIERTA CONVEXA

Un problema fundamental en la geometría computacional es el de calcular los puntos que "acotan" a un conjunto finito de **puntos en el plano**, formalmente llamados cubierta convexa. (Véase la figura 11.3.1, donde se indican los puntos que forman la cubierta convexa.) La cubierta convexa tiene aplicaciones en muchas áreas, incluyendo la estadística, la graficación por computadora y el procesamiento de imágenes. Por ejemplo, en estadística, los puntos de un conjunto de datos que determinen la cubierta convexa podrían ser extraños, puntos poco representativos de los datos, por lo que podrían descartarse. En esta sección presentamos el algoritmo de Graham para calcular la cubierta convexa. En esta sección, un "conjunto de puntos" será un "conjunto de puntos *distintos*". Comenzamos con las definiciones.

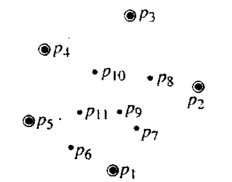

FIGURA 11.3.1
La cubierta convexa de los puntos $p_1, \ldots, p_{11}$ es p_1, p_2, p_3, p_4, p_5.

DEFINICIÓN 11.3.1

Dado un conjunto finito de puntos S del plano, un punto p en S es un *punto de la cubierta* si existe una línea (recta) L que pasa por p de modo que todos los puntos de S, excepto p, están en un lado de L (y excepto p, ninguno está en L).

EJEMPLO 11.3.2

En la figura 11.3.2, p_1 es un punto de la cubierta, pues podemos determinar una línea L_1 que pasa por p_1 tal que los demás puntos están estrictamente en un lado de L_1. El punto p_8 no es un punto de la cubierta, pues cada línea L por p_8 tiene puntos en ambos lados de L. El punto p_6 tampoco es un punto de la cubierta. Como muestra la figura 11.3.2, es posible trazar una línea L_2 por p_6 tal que un lado de L_2 no tenga puntos del conjunto, pero L_2 no cumple las condiciones de la definición 11.3.1, pues contiene puntos distintos de p_6. □

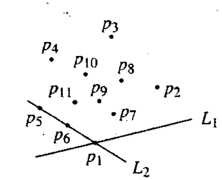

FIGURA 11.3.2
Los puntos de la cubierta son p_1, p_2, p_3, p_4, p_5. Los demás puntos no son puntos la cubierta.

La cubierta convexa de un conjunto finito de puntos S del plano consta de los puntos de la cubierta, enumerados en orden al recorrer la frontera de S. En la figura 11.3.2, la cubierta convexa es la sucesión de puntos p_1, p_2, p_3, p_4, p_5. La siguiente definición precisa el concepto de orden de los puntos de la cubierta.

DEFINICIÓN 11.3.3

La *cubierta convexa* de un conjunto finito de puntos S del plano es la sucesión $p_1, p_2, \ldots, p_n$ de puntos de la cubierta de S, enumerados en el siguiente orden. El punto p_1 es el punto con coordenada y mínima. Si existen varios puntos con la misma coordenada y mínima, p_1 es el que tiene abscisa mínima. (Observe que p_1 es un punto de la cubierta.) Para $i \ge 2$, sea α_i el ángulo que forma la horizontal con el segmento de recta p_1, p_i (véase la figura 11.3.3). Los puntos $p_2, p_3, \ldots, p_n$ se ordenan de modo que $\alpha_2, \alpha_3, \ldots, \alpha_n$ sea una sucesión creciente.

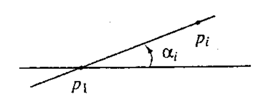

FIGURA 11.3.3
α_i es el ángulo que forma la horizontal con el segmento de recta p_1, p_i.

EJEMPLO 11.3.4

En la figura 11.3.4, el punto p_1 tiene la coordenada y mínima, de modo que es el primer punto enumerado en la cubierta convexa. Como se muestra, los ángulos que forma la horizontal con los segmentos de recta $p_1, p_2; p_1, p_3; p_1, p_4; p_1, p_5$ van creciendo. Así, la cubierta convexa del conjunto de puntos de la figura 11.3.4 es p_1, p_2, p_3, p_4, p_5. □

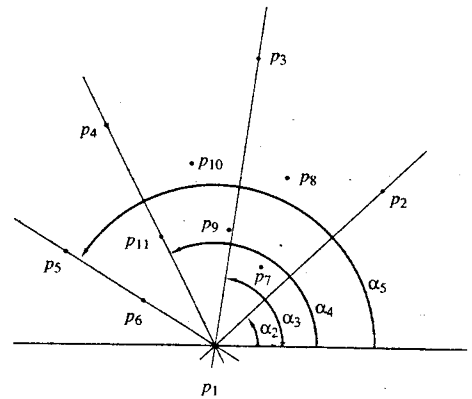

FIGURA 11.3.4 La cubierta convexa es p_1, p_2, p_3, p_4, p_5 debido a que estos son puntos de la cubierta convexa y que los ángulos correspondientes $\alpha_2, \alpha_3, \alpha_4, \alpha_5$ van creciendo, donde α_i es el ángulo que forma la horizontal con el segmento de recta p_1, p_i.

La definición 11.3.3 sugiere un algoritmo para calcular la cubierta convexa de un conjunto finito de puntos S del plano. Primero se determina el punto p_1 con coordenada y mínima. Si existen varios puntos con la misma coordenada y mínima, se elige el punto con abscisa mínima. A continuación se ordenan *todos* los puntos p en S de acuerdo con el ángulo que forma la horizontal con el segmento de recta p_1, p. Por último, se examinan los puntos, en orden, y se descartan aquellos que no están en la cubierta convexa. El resultado será la cubierta convexa. Ésta es la estrategia utilizada por el algoritmo de Graham. Para convertir esta idea en un algoritmo, hay que resolver dos aspectos fundamentales. Debemos describir una forma para comparar ángulos, y debemos desarrollar un método para verificar si los puntos están en la cubierta convexa. Primero veremos el problema de la comparación de ángulos.

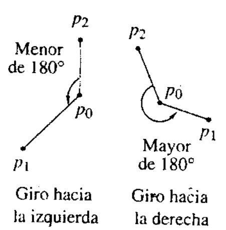

FIGURA 11.3.5 El primer giro es hacia la izquierda, pues el ángulo de p_0, p_2 a p_0, p_1 es menor de 180°. El segundo giro es hacia la derecha, pues el ángulo de p_0, p_2 a p_0, p_1 es mayor de 180°.

Supongamos que visitamos los puntos distintos p_1, p_0, p_2 en el plano en este orden. Si después de salir de p_0 nos movemos hacia la izquierda, decimos que los puntos p_1, p_0, p_2 *forman un giro hacia la izquierda* (véase la figura 11.3.5). Más precisamente, los puntos p_1, p_0, p_2 forman un giro hacia la izquierda si el ángulo del segmento de recta p_0, p_2 al segmento de recta p_0, p_1, medido en el sentido contrario al de las manecillas del reloj, es menor de 180°. De manera análoga, si al salir de p_0 nos movemos hacia la derecha, decimos que los puntos p_1, p_0, p_2 *forman un giro hacia la derecha* (véase la figura 11.3.5); es decir, los puntos p_1, p_0, p_2 forman un giro hacia la derecha si el ángulo del segmento de recta p_0, p_2 al segmento de recta p_0, p_1, medido en el sentido contrario al de las manecillas del reloj, es mayor de 180°.

Podemos utilizar los métodos de la geometría analítica para deducir un criterio que permita decidir si los puntos p_1, p_0, p_2 forman un giro hacia la izquierda o hacia la derecha. Sean (x_i, y_i) las coordenadas del punto p_i, $i = 0, 1, 2$ (véase la figura 11.3.6). Supongamos primero que $x_1 < x_0$. La ecuación de la recta L que pasa por p_0 y p_1 es

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0).$$

Ahora, p_1, p_0, p_2 forman un giro hacia la izquierda precisamente cuando p_2 está arriba de L, o cuando

$$y_2 > y' = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0).$$

Podemos reescribir esta última desigualdad como

$$y_2 - y_0 > \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0).$$

Al multiplicar por $x_1 - x_0$, que es negativo, y pasar todos los términos a un lado de la desigualdad se obtiene

$$(y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0) < 0.$$

Si definimos el **producto cruz** de los puntos p_0, p_1, p_2 como

$$\text{cruz}(p_0, p_1, p_2) = (y_2 - y_0)(x_1 - x_0) - (y_1 - y_0)(x_2 - x_0),$$

hemos demostrado lo siguiente:

Si los puntos p_1, p_0, p_2 forman un giro hacia la izquierda, entonces $\text{cruz}(p_0, p_1, p_2) < 0$.

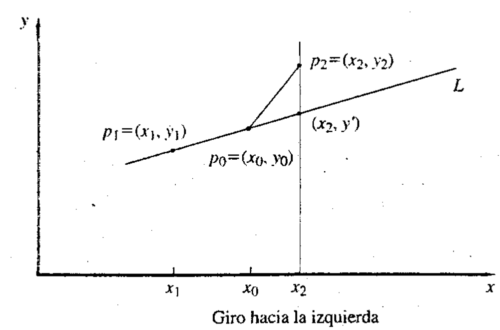

FIGURA 11.3.6 Criterio para decidir si los puntos p_1, p_0, p_2 forman un giro hacia la izquierda o hacia la derecha. La figura supone que $x_1 < x_0$. L es la recta que pasa por p_0 y p_1. Como se muestra, en este caso ocurre un giro hacia la izquierda precisamente cuando p_2 está arriba de L.

De manera análoga, podemos mostrar que

Si los puntos p_1, p_0, p_2 forman un giro hacia la derecha, entonces $\text{cruz}(p_0, p_1, p_2) > 0$,

y

Si los puntos p_1, p_0, p_2 son colineales, entonces $\text{cruz}(p_0, p_1, p_2) = 0$.

También podemos demostrar los recíprocos de estas afirmaciones. Por ejemplo, para demostrar que si cruz $(p_0, p_1, p_2) < 0$, entonces los puntos p_1, p_0, p_2 forman un giro hacia la izquierda, podemos argumentar como sigue. Supongamos que $\text{cruz}(p_0, p_1, p_2) < 0$. Como los puntos no forman un giro hacia la derecha [ya que en ese caso, $\text{cruz}(p_0, p_1, p_2)$ sería positivo] y no son colineales [ya que en ese caso, $\text{cruz}(p_0, p_1, p_2) = 0$], deben formar un giro hacia la izquierda. Así,

Los puntos p_1, p_0, p_2 forman un giro hacia la izquierda si y sólo si $\text{cruz}(p_0, p_1, p_2) < 0$.

Los puntos p_1, p_0, p_2 forman un giro hacia la derecha si y sólo si $\text{cruz}(p_0, p_1, p_2) > 0$.

Los puntos p_1, p_0, p_2 son colineales si y sólo si $\text{cruz}(p_0, p_1, p_2) = 0$. (11.3.1)

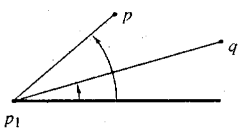

FIGURA 11.3.7
El caso $p > q$ ocurre cuando p, p_1, q forman un giro hacia la izquierda.

Hemos demostrado (11.3.1) suponiendo que $x_1 < x_0$. Si $x_1 = x_0$ o $x_1 > x_0$, la conclusión (11.3.1) sigue siendo válida (véanse los ejercicios 2 y 3). Resumimos estas conclusiones como un teorema.

TEOREMA 11.3.5

Si p_0, p_1, p_2 son puntos distintos en el plano,

(a) p_1, p_0, p_2 forman un giro hacia la izquierda si y sólo si $\text{cruz}(p_0, p_1, p_2) < 0$.

(b) p_1, p_0, p_2 forman un giro hacia la derecha si y sólo si $\text{cruz}(p_0, p_1, p_2) > 0$.

(c) p_1, p_0, p_2 son colineales si y sólo si $\text{cruz}(p_0, p_1, p_2) = 0$.

Demostración. La demostración aparece antes del enunciado del teorema. ■

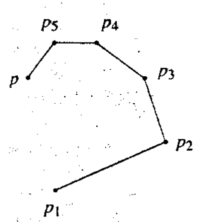

FIGURA 11.3.8
Una situación en el algoritmo de la cubierta convexa al examinar el punto p. Antes de examinar a p, la cubierta convexa de los puntos analizados hasta ese momento es p_1, p_2, p_3, p_4, p_5. Como p_4, p_5, p forman un giro hacia la izquierda, p_5 sigue estando en la cubierta convexa de los puntos analizados hasta ese momento, por lo que se le retiene. La cubierta convexa actual es $p_1, p_2, p_3, p_4, p_5, p$. El algoritmo continúa examinando al punto posterior a p.

El algoritmo de Graham comienza determinando el punto p_1 con coordenada y mínima. Si existen varios puntos con la misma coordenada y mínima, el algoritmo elige el punto con abscisa mínima. A continuación, el algoritmo ordena *todos* los puntos p en S de acuerdo con el ángulo que forma la horizontal con el segmento de recta p_1, p. Por último, se examinan los puntos, en orden, y se descartan aquellos que no están en la cubierta convexa.

Podemos utilizar el producto cruz para comparar puntos $p \neq q$ en el ordenamiento. Para comparar los puntos p y q, calculamos $\text{cruz}(p_1, p, q)$. Si $\text{cruz}(p_1, p, q) < 0$, entonces p, p_1, q forman un giro hacia la izquierda. Con respecto de los ángulos con la horizontal, esto significa que $p > q$ (véase la figura 11.3.7). Si $\text{cruz}(p_1, p, q) > 0$, entonces $p < q$. Si $\text{cruz}(p_1, p, q) = 0$, entonces p, p_1, q son colineales. En este último caso, definimos $p > q$ si p está más lejos de p_1 que q, y $p < q$ si q está más lejos de p_1 que p.

También podemos utilizar el producto cruz para determinar si un punto no está en la cubierta convexa y que, por tanto, puede ser descartado. Al examinar los puntos en orden, conservamos los puntos que estarían en la cubierta convexa si no hay más puntos por examinar. Luego analizamos el siguiente punto p. Por ejemplo, en la figura 11.3.8, suponga que hemos conservado $p_1, \ldots, p_5$ y que el siguiente punto por examinar es p. Como p_4, p_5, p forman un giro hacia la izquierda, conservamos a p_5. Luego continuamos analizando el punto posterior a p.

Como otro ejemplo, suponga que hemos conservado a $p_1, \ldots, p_5$ (véase la figura 11.3.9). Esta vez, como p_4, p_5, p forman un giro hacia la derecha, descartamos p_5. Ahora regresamos para examinar p_3, p_4, p. Como estos puntos también forman un giro hacia la derecha, descartamos p_4. Ahora regresamos para examinar p_2, p_3, p. Como estos puntos forman un giro hacia la izquierda, conservamos a p_3. Continuamos examinando el punto posterior a p. El seudocódigo para el algoritmo de Graham aparece como el algoritmo 11.3.6.

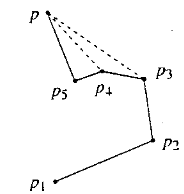

FIGURA 11.3.9
Una situación en el algoritmo de la cubierta convexa al examinar el punto p. Antes de examinar a p, la cubierta convexa de los puntos analizados hasta ese momento es p_1, p_2, p_3, p_4, p_5. Como p_4, p_5, p forman un giro hacia la derecha, descartamos p_5. Esto deja los puntos p_3, p_4, p, que también forman un giro hacia la derecha; así, también descartamos p_4. Esto deja los puntos p_2, p_3, p, que forman un giro hacia la izquierda, por lo que conservamos a p_3. La cubierta convexa actual es p_1, p_2, p_3, p. El algoritmo continúa examinando al punto posterior a p.

ALGORITMO 11.3.6 ***Algoritmo de Graham para calcular la cubierta convexa***

Este algoritmo calcula la cubierta convexa de los puntos $p_1, \ldots, p_n$ del plano. Las coordenadas x y y del punto p se denotan $p.x$ y $p.y$, respectivamente.

Entrada: $p_1, \ldots, p_n$ y n

Salida: $p_1, \ldots, p_k$ (la cubierta convexa de $p_1, \ldots, p_n$) y k

```
procedure graham_scan (p, n, k)
 // caso trivial
 if n = 1 then
  begin
  k := 1
  return
  end
 // determinar el punto con coordenada y mínima
 min := 1
 for i := 2 to n do
  if p_i.y < p_min.y then
   min := 1
 // Entre todos estos puntos, determinar aquél con abscisa mínima
 // coordenada x
 for i := 1 to n do
  if p_i.y = p_min.y and p_i.x < p_min.x then
   min := i
 swap(p_1, p_min)
 // ordenar según el ángulo de la horizontal a p_1, p_i
 sort p_2, ..., p_n
 // p_0 es un punto adicional, agregado con el fin de evitar que
 // el algoritmo se repita indefinidamente
 p_0 := p_n
 // descartar los puntos que no están en la cubierta convexa
 k := 2
 for i := 3 to n do
  begin
   while p_{k-1}, p_k, p_i no giran a la izquierda do
    // descartar p_k
    k := k - 1
   k := k + 1
   swap(p_i, p_k)
  end
end graham_scan
```

EJEMPLO 11.3.7

La figura 11.3.10 muestra al algoritmo de Graham en acción.

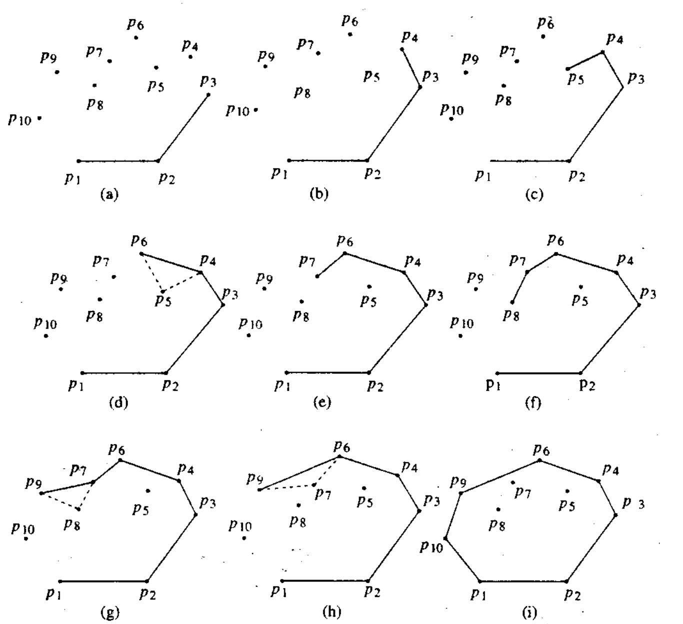

FIGURA 11.3.10 El algoritmo de Graham para calcular la cubierta convexa. El punto p_1 tiene la coordenada y mínima, y entre todos los puntos de este tipo, tiene la abscisa mínima. Los puntos ordenados según el ángulo de la horizontal con p_1, p_i son $p_2, p_3, \ldots, p_{10}$. En (a), el algoritmo comienza examinando p_1, p_2, p_3. p_1, p_2, p_3 forman un giro hacia la izquierda, de modo que se conserva a p_2. Luego en (b), se examinan p_2, p_3, p_4. Estas p_2, p_3, p_4 forman un giro hacia la izquierda, de modo que se conserva a p_3. Luego en (c), se examinan p_3, p_4, p_5. p_3, p_4, p_5 forman un giro hacia la izquierda, de modo que se conserva a p_4. Luego en (d), se examinan p_4, p_5, p_6. p_4, p_5, p_6 forman un giro hacia la derecha, de modo que se descarta a p_5. El algoritmo regresa a p_3, p_4, p_6. p_3, p_4, p_6 forman un giro hacia la izquierda, de modo que se conserva a p_4. Luego en (e), se examinan p_4, p_6, p_7. p_4, p_6, p_7 forman un giro hacia la izquierda, de modo que se conserva a p_6. Luego en (f), se examinan p_6, p_7, p_8. p_6, p_7, p_8 forman un giro hacia la izquierda, de modo que se conserva a p_7. Luego en (g), se examinan p_7, p_8, p_9. p_7, p_8, p_9 forman un giro hacia la derecha, de modo que se descarta a p_8. En (h), el algoritmo regresa a p_6, p_7, p_9. p_6, p_7, p_9 también forman un giro hacia la derecha, de modo que se descarta a p_7. El algoritmo regresa a p_4, p_6, p_9. p_4, p_6, p_9 forman un giro hacia la izquierda, de modo que se conserva a p_6. Finalmente, en (i), el algoritmo concluye examinando a p_6, p_9, p_{10}. p_6, p_9, p_{10} forman un giro hacia la izquierda, de modo que se conserva a p_9. La cubierta convexa es $p_1, p_2, p_3, p_4, p_6, p_9, p_{10}$. □

El algoritmo de Graham comienza con dos ciclos for, cada uno de los cuales tarda un tiempo $\Theta(n)$. Si utilizamos un ordenamiento óptimo, como aquél por fusión, el tiempo de ordenamiento en el peor de los casos será $\Theta(n \lg n)$.

El tiempo *total* de ejecución del último ciclo while es $O(n)$, pues un punto se puede descartar a lo más una vez y existen n puntos. El propio ciclo for se ejecuta $\Theta(n)$; así, el tiempo total para el último ciclo for y el ciclo while es $\Theta(n)$. Vemos que el tiempo de ordenamiento domina, de modo que el tiempo en el peor de los casos del algoritmo de Graham es $\Theta(n \lg n)$. El tiempo necesario después de ordenar los puntos es $\Theta(n)$, de modo que si los puntos vienen ordenados de antemano, el algoritmo de Graham puede calcular la cubierta convexa en un tiempo lineal.

Nuestro último teorema muestra que cualquier algoritmo que calcule la cubierta convexa de n puntos del plano tiene un tiempo $\Omega(n \lg n)$ en el peor de los casos, de modo que el algoritmo de Graham es óptimo.

TEOREMA 11.3.8

Cualquier algoritmo que calcule la cubierta convexa de n puntos del plano tiene un tiempo $\Omega(n \lg n)$ en el peor de los casos.

Demostración. Sea A un algoritmo que calcule la cubierta convexa de un conjunto finito de puntos del plano. Mostraremos que, en el peor de los casos, este algoritmo ocupa tanto tiempo como un algoritmo de ordenamiento y por tanto tiene la misma cota inferior $\Omega(n \lg n)$ del problema de ordenamiento.

Consideremos una sucesión arbitraria de números reales

$$y_1, \quad y_2, \quad \ldots, \quad y_n,$$

donde cada y_i está entre 0 y 1. Utilizamos el algoritmo de la cubierta convexa, el algoritmo A, para construir un algoritmo, el algoritmo B, que ordene a esta sucesión. En primer lugar, el algoritmo B proyecta estos números reales sobre el círculo unitario (véase la figura 11.3.11). A continuación, el algoritmo B llama al algoritmo A para determinar la cubierta convexa $h_1, h_2, \ldots, h_n$ de los puntos del círculo. Por último, el algoritmo B produce como salida las coordenadas y de $h_1, h_2, \ldots, h_n$ en este orden. Observe que la salida del algoritmo B es la sucesión de entrada ordenada:

$$y_1, \quad y_2, \quad \ldots, \quad y_n.$$

Como el algoritmo B es un algoritmo de ordenamiento, el teorema 7.7.3 implica que su tiempo en el peor de los casos, t_n, satisface

$$t_n \geq Cn \lg n.$$

Por otro lado, el algoritmo B consta de dos ciclos $\Theta(n)$ (uno para proyectar los puntos en el círculo unitario, y el otro para producir como salida las coordenadas y de la cubierta convexa) y la llamada al algoritmo A, que tarda un tiempo s_n, digamos, en el peor de los casos. Así,

$$t_n = 2n + s_n.$$

Por tanto,

$$s_n = t_n - 2n \geq Cn \lg n - 2n = \Omega(n \lg n)$$

y esto concluye la demostración. ■

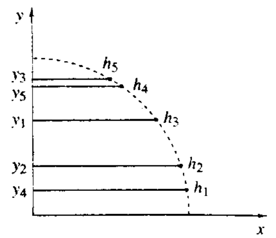

FIGURA 11.3.11 Cubos numerados sobre una mesa. Ordenamiento mediante el algoritmo de la cubierta convexa. Los puntos y_1, y_2, y_3, y_4, y_5 se proyectan primero sobre el círculo unitario. Los puntos resultantes sobre el círculo unitario se denotan h_i. Luego, se utiliza el algoritmo de la cubierta convexa para determinar la cubierta convexa h_1, h_2, h_3, h_4, h_5. Las coordenadas y de la cubierta convexa (en el orden h_1, h_2, h_3, h_4, h_5) son y_4, y_2, y_1, y_5, y_3, el orden requerido de las y.

Ejercicios

1. Sea S un conjunto finito de puntos del plano. Sea p_1 el punto con coordenada y mínima. Si varios puntos tienen la misma coordenada y mínima, elija aquél con abscisa mínima. Demuestre que p_1 está en la cubierta convexa de S.
2. Demuestre el teorema 11.3.5 cuando $x_1 = x_0$.
3. Demuestre el teorema 11.3.5 cuando $x_1 > x_0$.
4. Utilice el algoritmo de Graham para determinar la cubierta convexa de los puntos (10, 1), (7, 7), (3, 13), (6, 10), (16, 4), (10, 5), (7, 13), (13, 8), (4, 4), (2, 2), (1, 8), (10 ,13), (7, 1), (4, 8), (12, 3), (16, 10), (14, 5), (10, 9).
5. Utilice el algoritmo de Graham para determinar la cubierta convexa de los puntos (7, 8), (9, 8), (3, 11), (5, 1), (7, 11), (9, 5), (9, 1), (6, 7), (4, 5), (2, 1), (10 ,17), (7, 3), (7, 14), (4, 8), (11, 3), (10 ,12).
6. Suponga que se ha utilizado el algoritmo de Graham para determinar la cubierta convexa de un conjunto de n puntos S. Muestre que si se agrega un punto S para obtener S', la cubierta convexa de S' se puede determinar en un tiempo $\Theta(n)$.

Los ejercicios 7-10 se refieren a la *marcha de Jarvis*, otro algoritmo que calcula la cubierta convexa de un conjunto finito de puntos del plano. Comienza, como el algoritmo de Graham, determinando el punto con coordenada *y* mínima. Si varios puntos tienen la misma coordenada *y* mínima, se elige aquél con abscisa mínima. A continuación, la marcha de Jarvis determina el punto p_2 tal que el ángulo de la horizontal con el segmento p_1, p_2 sea mínimo. (En el caso de empates, se elige el punto más lejano de p_1.) Después de determinar $p_1, \ldots, p_i$, la marcha de Jarvis determina el punto p_{i+1} tal que p_{i-1}, p_i, p_{i+1} forman el menor giro hacia la izquierda. (En el caso de empates, se elige el punto más lejano de p_i.)

7. Muestre que la marcha de Jarvis realmente determina la cubierta convexa.
8. Escriba un seudocódigo para la marcha de Jarvis.
9. Determine el tiempo en el peor de los casos para la marcha de Jarvis.
10. ¿Existen conjuntos para los cuales la marcha de Jarvis sea más rápida que el algoritmo de Graham? Explique.

NOTAS

[Preparata, 1985 y Edelsbrunner] son libros de geometría computacional.

El algoritmo del par más cercano de la sección 11.1 fue ideado por M. I. Shamos y aparece en [Preparata, 1985]. [Preparata, 1985] también proporciona un algoritmo $\Theta(n \lg n)$ para determinar el par más cercano en un número arbitrario de dimensiones.

El algoritmo de Graham (véase [Graham, 1972]) que apareció en 1972 fue uno de los primeros algoritmos $\Theta(n \lg n)$ para la cubierta convexa plana. La marcha de Jarvis aparece en [Jarvis]. El cálculo de la cubierta convexa de un conjunto de puntos en un espacio de más de dos dimensiones es mucho más complejo que el cálculo de la cubierta convexa en el plano. El primer algoritmo tridimensional óptimo fue dado en 1977 por [Preparata, 1977]. En 1981, [Seidel] dio un algoritmo n-dimensional para la cubierta convexa, que es óptimo para n par. La determinación de algoritmos más eficientes para determinar la cubierta convexa en espacios de varias dimensiones es una importante área de investigación en la actualidad.

CONCEPTOS BÁSICOS DEL CAPÍTULO

Sección 11.1

Geometría computacional
Problema del par más cercano
Algoritmo del par más cercano

Sección 11.2

El tiempo en el peor de los casos de cualquier algoritmo que resuelve el problema de determinar si n números reales son distintos es $\Omega(n \lg n)$ (teorema 11.2.1).
El tiempo en el peor de los casos de cualquier algoritmo que determine la distancia entre un par más cercano entre n elementos del plano es $\Omega(n \lg n)$ (corolario 11.2.2).

Sección 11.3

Punto de la cubierta
Cubierta convexa
Producto cruz
Algoritmo de Graham para calcular la cubierta convexa
Cualquier algoritmo que calcule la cubierta convexa de n puntos del plano tiene un tiempo en el peor de los casos de $\Omega(n \lg n)$ (teorema 11.3.8).

AUTOEVALUACIÓN DEL CAPÍTULO

Sección 11.1

1. Describa la forma en que el algoritmo del par más cercano determina el par más cercano de puntos, si la entrada es como la del ejercicio 4, sección 11.3.
2. Para concluir la recursión en el algoritmo del par más cercano, si existen tres o menos puntos en la entrada podemos determinar el par más cercano en forma directa. ¿Por qué no podemos reemplazar "tres" por "dos"?
3. Muestre que existen a lo más cuatro puntos en la mitad inferior del rectángulo de la figura 11.3.3.
4. ¿Cuál sería el tiempo en el peor de los casos del algoritmo del par más cercano (algoritmo 11.1.2) si en vez de realizar la fusión de $p_i, \ldots, p_k$ y $p_{k+1}, \ldots, p_j$ utilizamos el ordenamiento por fusión para ordenar $p_i, \ldots, p_j$?

Sección 11.2

5. Muestre que el tiempo en el peor de los casos de cualquier algoritmo que determine un par más cercano entre n elementos en el espacio de dimensión d es $\Omega(n \lg n)$.
6. Muestre que el tiempo en el peor de los casos de cualquier algoritmo que determine todos los pares más cercanos entre n elementos del plano es $\Omega(n \lg n)$.
7. Enuncie y demuestre una mejor cota inferior para el tiempo en el peor de los casos de cualquier algoritmo que resuelva el problema al determinar si n números reales son todos iguales.
8. Muestre que el tiempo en el peor de los casos de cualquier algoritmo que determine todos los pares a una distancia menor o igual a 2δ, donde δ es la distancia entre un par más cercano, es $\Omega(n \lg n)$.

Sección 11.3

9. Sea S un conjunto finito de puntos en el plano. Sea p el punto en S con abscisa máxima. Si varios puntos tienen la misma abscisa máxima, elija el que tiene coordenada y máxima. Demuestre que p está en la cubierta convexa de S.
10. Sea S un conjunto finito de puntos distintos en el plano. Suponga que S contiene al menos dos puntos. Sean p y q puntos en S que están a una distancia máxima. Demuestre que p y q están en la cubierta convexa de S.
11. Utilice el algoritmo de Graham para determinar la cubierta convexa del conjunto de puntos del ejercicio 1, sección 11.1.
12. Suponga que se ha utilizado el algoritmo de Graham para determinar la cubierta convexa de un conjunto de $n \geq 2$ puntos. Muestre que si un punto distinto de p_1 del algoritmo 11.3.6 se elimina de S para obtener S', la cubierta convexa de S' se puede determinar en un tiempo $\Theta(n)$.