Memoria Trabajo Individual

Algoritmo Divide y Vencerás: Par de Puntos Más Próximos

Álvaro Berrío Galindo

Algoritmos y Computación

Curso 2018-2019

Problema

El tema que se trata en este trabajo es el problema del par de puntos más cercanos. Este pertenece al grupo de problemas de geometría computacional, que es una rama de la informática dedicada al estudio de algoritmos basados en términos geométricos. El problema de par de puntos más cercanos fue de los primeros que se usaron para el análisis de la complejidad de algorimtos geométricos y fueron Shamos y Hoey quienes lo realizaron en la década de los 70.

El problema consiste en hallar la distancia mínima entre dos puntos dado un conjunto de puntos. Se realiza usando algoritmos Divide y Vencerás pero no es el único tipo, también se puede resolver mediante Programación Dinámica o algoritmos probabilísticos. En este caso, me centro en el algoritmo Divide y Vencerás cuya idea fundamental es dividir recursivamente el programa en fragmentos más pequeños hasta poder trabajar con una masa de datos más fácil de manipular y después combinar las soluciones obtenidas para cada subproblema.

Conocer la distancia mínima entre dos puntos de un conjunto es una información útil en ámbitos como la geografía, las finanzas, la predicción meteorológica o las simulaciones biológicas de manera que se puedan conocer las observaciones más parecidas. Esto indica que se usa en labores con una cantidad grande de datos, por ello, aumentar la eficiencia de estos algoritmos puede repercutir en un ahorro de tiempo y recursos considerable.

Descripción del Algoritmo

El problema resuelto mediante la fuerza bruta consiste en comparar las distancias de cada punto con el resto y encontrar la que sea mínima de todas ellas. Esta manera de resolver el problema es muy costosa, especialmente cuando la masa de puntos es muy extensa. Es por esto que se usan algoritmos más eficientes.

Como ya se ha mencionado, el problema de los puntos más cercanos se puede resolver con un algoritmo Divide y Vencerás. La implemetación del algoritmo apenas varía según la dimensionalidad del problema aunque sí existen diferencias en el trasfondo teórico y geométrico.

- Antes de comenzar el algoritmo, es necesario ordenar el vector de puntos según las coordenadas x e y independientemente de la dimensión en la que se está trabajando.

- El procedimiento del algoritmo comienza con la comprobación del número de puntos en el conjunto. Si hay solo uno, la distancia es infinita, si hay dos o tres elementos, la distancia mínima se obtiene mediante la fuerza bruta entre los puntos, este será el caso base.

- Si hay más puntos en el conjunto, se divide la masa de puntos en dos subconjuntos según la mediana de la coordenada x, trazando una línea en el caso bidimensional y un plano en el caso tridimensional. La división izquierda la conforman los vectores *s1_x* y *s1_y* que contendrán los puntos ordenados por la coordenada x e y respectivamente mientras que los vectores que forman la derecha se nombran como *s2_x* y *s2_y*. Al dividir por la mediana quedan el mismo número de puntos en cada lado.

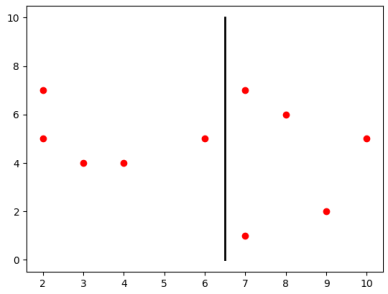

Figure 1: División para 2 dimensiones

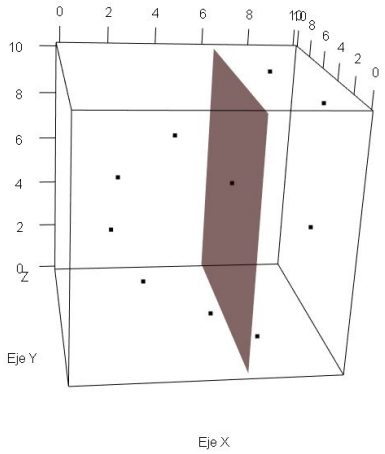

Figure 2: División para 3 dimensiones

- Se resuelve recursivamente cada lado de la división obteniendo la distancia mínima en cada uno, *d1* para el lado izquierdo y *d2* para el lado derecho. Comparándolas, se escoge la distancia mínima *d*.

- Hecho esto se tiene que contemplar la posibilidad de que la distancia mínima del conjunto de puntos se dé entre un punto que se sitúa en el lado derecho y otro que esté en el izquierdo de la división. Para tener en cuenta esta tesitura, se establece una banda a ambos lados de la división cuyo lado será la distancia mínima hallada previamente (d). Esta banda es un plano en dos dimensiones y un ortoedro en tres. Se calcula la distancia menor, *db*.

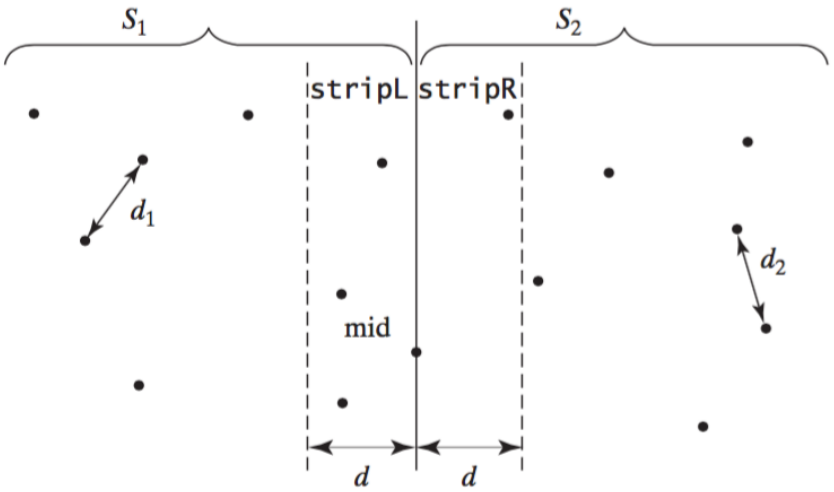

Figure 3: Banda formada en dos dimensiones

- Finalmente, se compara *db* con *d*, de manera que el algoritmo devolverá la más pequeña de las dos.

Es necesario profundizar en la manera de hallar la distancia mínima de la banda mencionada en el penúltimo punto ya que es un punto crucial del problema y además es el distintivo entre las dos y las tres dimensiones. Para cada punto en la banda se establece una caja aumentando y disminuyendo la distancia d en la coordenada y y sumando esa misma distancia en la coordenada x , siendo la distancia d la mitad de la longitud de la banda. Es decir, para cada punto p, la caja será [p.x,p.x+d], [p.y-d,p.y+d]. Esta probado teóricamente que en esta caja entran solo 6 elementos además del propio punto. A la hora de hacerlo en tres dimensiones, lo único que hay que tener en cuenta es la tercera dimensión z. De esta manera, para cada punto la caja tiene la forma: [p.x,p.x+d], [p.y-d,p.y+d], [p.z-d,p.z+d]. En tres dimensiones son 12 los puntos que entran en la caja. Teniendo esto en cuenta y que se tiene un vector con los puntos ordenados por la coordeanda y, ya solo faltaría hallar las distancias de cada punto con los seis siguientes en dos dimensiones y con los dieciocho siguiente en tres dimensiones y devolver la menor, que se ha llamado *db*.

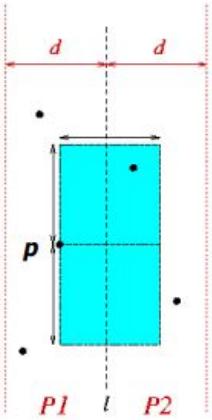

Figure 4: Formación de la caja de cada punto en la banda

Pseudocódigo

En el pseudocódigo se pueden apreciar los diferentes pasos anteriores.

```
puntos_x = vector de puntos ordenados por la coordenada x
puntos_y = vector de puntos ordenados por la coordenada y

función puntos_cercanos (puntos_x, puntos_y)
        if n=1
                return infinito
        elseif 2<=n<=3
                return caso_base(puntos)
        elseif n>=4
                med=mediana de la coordenada x de puntos
                s1_x = vector de puntos en el lado izquierdo ordenados por x
                s1_y =  vector de puntos en el lado izquierdo ordenados por y
                s2_x = vector de puntos en el lado derecho ordenados por x
                s2_y = vector de puntos en el lado derecho ordenados por y
                d1=puntos_cercanos(s1_x, s1_y)
                d2=puntos_cercanos(s2_x, s2_y)
                d=min(d1,d2)
                candidatos = par de puntos cuya distancia es d
                db=dist_banda(puntos_y, med, d)
                dtotal=min(d,db)
                candidatos = par de puntos cuya distancia es dtotal
                return dTotal y candidatos
```

```
función dist_banda(puntos,mediana,dist)
    for each i in puntos
        banda = vector de puntos p tal que (mediana-dist < p.x < mediana+dist)
    if 2D
        for i from 0 to length(banda)
            for j from i+1 to i+6
                if j<length(banda) & dist < |i − j|
                    dist = |i-j|
                    Actualizar candidatos
    elseif 3D
        for i from 0 to length(banda)
            for j from i+1 to i+18
                if j<length(banda) & dist < |i − j| & j.z pertenece a (i.z-d, i.z+d)
                    dist = |i-j|
                    Actualizar candidatos

    return dist y candidatos
```

El vector de puntos original se ordena para las coordenadas x e y creando un vector para cada ordenación. La primera sirve para poder dividir los puntos en dos subconjuntos de mismo tamaño mientras que el vector ordenado según la coordenada$y$ es el que se usa para calcular la distancia mínima en la banda. La función **puntos_cercanos** es la función recursiva,

la principal. Como ya se indicó con anterioridad primero evalúa el número de elementos que tiene el vector recibido. Si tiene uno solo devolverá infinito, si hay dos o tres puntos se ejecuta el caso base basado en fuerza bruta. En cambio, si el tamaño del vector es mayor, este se divide en dos partes con el mismo número de puntos cada una si el tamaño es par y con un elemento más en el lado izquierdo si el tamaño es impar. El lado izquierdo es *s1* y el derecho *s2*. Acto seguido, se ejecuta recursivamente la propia función **puntos_cercanos** para ambos dos subconjuntos. La distancia mínima será entonces la menor de la dos devueltas. Esta distancia se compara después con la obtenida al ejecutar la función **dist_banda**, devolviendo la menor.

La última parte del pseudocódigo corresponde con la función **dist_banda**, que calcula la distancia mínima que se da en la banda previamente mencionada. Primero se construye la banda de manera que escoge a los puntos cuya coordenada x se encuentre en el intervalo (med-dist,med+dist) siendo med la mediana de las coordenadas x de los puntos recibidos y dist la distancia mínima entre las distancias del lado derecho y lado izquierdo. Se construye posteriormente la *caja* para cada punto [+d, +-d) en la que habrá a lo sumo seis puntos en dos dimensiones. Para tres dimensiones, la caja tiene la forma [+d, +-d, +-d) y entran dieciocho puntos. Se halla la distancia de cada punto con los otros cinco o diecisiete que entren en su caja y se devuelve la menor de todas.

Coste

Si se halla la solución del problema del par de puntos más cercano mediante un algoritmo de fuerza bruta en el que se adquiere la distancia mínima mediante la comparación de todos los puntos con los demás se llegaría a un coste correspondiente a O(n^2).

Para obtener la complejidad de nuestro algoritmo se hace uso del caso 2 del Método Maestro para resolver la recursividad $T(n) = a \cdot T(n/b) + f(n)$, después hay que tener en cuenta las instrucciones que no entran en la recursividad.

En la función general de recursión, la variable a indica el número de subproblemas en que se divide el problema original, n/b es el tamaño de cada problema y *f(n)* es el coste de las instrucciones de división y combinación. En este caso a y b son claramente 2 ya que el problema principal se divide en 2 subproblemas correspondientes a cada lado de la división y cada lado tiene un tamaño de *n/2*. Por lo tanto, queda la incógnita de *f(n)*. Este coste es O(n) ya que es el mayor coste de una instrucción dentro de la función de recursión o el cálculo de la distancia en la banda, que corresponde a la creación de los vectores para cada lado de la división o a hallar la distancia de cada punto en la banda con los que conforman la caja.

La ecuación general de la ecuación de recursión queda como $T(n) = 2 \cdot T(n/2) + O(n)$, mediante el caso dos del Método Maestro tenemos que hallar un k>=0 y un c=$log_b a$ de manera que $f(n) = O(n^c \cdot log^k(n)) = O(n)$. Es claro ver que k=0 y c=1, lo cual tenemos que aplicar a la ecuación de resolución de la recursión planteada en el Método Maestro además del valor de las variables a y b: $T(n) = O(n^{log_b a} \cdot log^{k+1}(n)) = O(n \cdot log(n))$.

Ahora hay que tener en cuenta el coste de las ordenaciones iniciales, que es lo único que no entra en la recursión. Se asume que ambas tienen coste $O(n \cdot log(n))$ ya que es la complejidad de los algoritmos de ordenación óptimos por lo que, al ser iguales que las de la recursividad el coste total del algoritmo es este mismo: $O(n \cdot log(n))$.

References

[1] https://arxiv.org/abs/1407.5609

[2] https://en.wikipedia.org/wiki/Closest_pair_of_points_problem

[3] Algorithm Design by Jon Kleinberg & Éva Tardos