Algoritmo divide y vencerás: Aplicación al problema del par más cercano con estructuras de datos distintas

Somil Sandoval Diaz

Universidad del Norte
Ingeniería de Sistemas
Profesor. Misael Diaz Maldonado

Barranquilla, Colombia
Noviembre del 2022

I. RESUMEN

El presente trabajo tiene como objetivo aplicar los métodos teóricos actuales para el análisis de algoritmos con el fin último de evaluar la efectividad temporal de un algoritmo que calcula la distancia entre pares de puntos con el objetivo de identificar cual par es el que se encuentra más cercano el uno con el otro y retornar dicha distancia mínima. El problema en cuestión tiene interés un practico en la actualidad y es posible encontrarle aplicaciones en los sistemas de control del tráfico aéreo, el par más cercano nos informa por ejemplo del mayor riesgo de colisión entre dos aviones, también para identificar cuáles son las dos ciudades más cercanas, o dada una ciudad determinar mediante una relación de distancias y costos o algunos otros factores, cual es la ciudad a la que se pueda llegar con el menor costo, entre otras. De entrada, estos problemas pudieran solucionarse, comparando el punto dado con cada uno de los puntos que se encuentren dentro de un conjunto n de puntos, pero si el conjunto es muy extenso y grande, entonces, los cálculos que se realicen podrían volverse excesivos. Sin embargo, existe otro método que utiliza la estrategia "divide y vencerás" que reduce notablemente esta cantidad excesiva de cálculos. Iremos analizando las técnicas de análisis de algoritmos para calcular complejidad de ambas formas de resolver el problema.

II. INTRODUCCION

El análisis de algoritmos es sin duda alguna una parte importante de las complejidades computacionales que le permite al diseñador de un algoritmo evaluar la capacidad de resolución de problemas de su algoritmo en términos de tiempo y memoria. Sin embargo, la principal preocupación en el análisis del algoritmo es el tiempo requerido o el rendimiento para dar solución al problema. La teoría de la complejidad algorítmica proporciona estimaciones teóricas de los recursos que necesita un algoritmo para resolver cualquier tarea computacional y el presente trabajo fue realizado justo con el propósito de aplicar los métodos teóricos actuales para el análisis de algoritmos para evaluar con esto la efectividad de un algoritmo que mide la distancia entre el par de puntos con el fin último de identificar cual es el par que se encuentra más cercano el uno con el otro y retorna dicha distancia mínima, en principio compararemos 2 métodos, el método de fuerza fruta y el metodología divide y vencerás este último tendrá un variante con el tipo de

estructura de dato utilizada, el objetivo será también descubrir el impacto del uso de una estructura y otra.

III. DEFINICION DEL PROBLEMA

Dado una lista que contiene n puntos en un plano se tiene como objetivo encontrar el par más cercano, es decir, la distancia entre los dos puntos más cercanos. El problema en cuestión tiene interés un práctico y es posible encontrarle aplicaciones en los sistemas de control del tráfico aéreo, el par más cercano nos informa del mayor riesgo de colisión entre dos aviones.

Dados dos puntos p(x, y), q(x1, y1) su distancia euclídea viene dada por la ecuación euclidiana. El algoritmo de "fuerza bruta" calcularía la distancia entre todos los posibles pares de puntos y se retornaría la distancia mínima entre todas ellas, sin embargo, el coste resultante sería cuadrático. En cambio, el enfoque divide y vencerás trataría de encontrar el par más cercano a partir de los pares más cercanos de conjuntos de puntos que sean una fracción del original, aplicando la estrategia de dividir y crear dos subconjuntos de puntos de un tamaño correspondiente a la mitad del original, con la metodología de divide y vencerás se aplicará una variante con respecto a la estructura de datos a utilizar, en este caso, utilizaremos inicialmente un arreglo y posteriormente una lista enlazada, calcularemos de forma experimental y teórica ambas complejidades temporales obtenidas y se tomaran decisiones para identificar cual es el método mas optimo.

IV. METODOLOGIA

Podemos seguir el enfoque de fuerza bruta e iterar sobre todos los pares de puntos, calculando la distancia entre cada dos y devolviendo el par más cercano. Sin embargo, la cantidad de cálculos necesarios para obtener la respuesta es excesivo para conjuntos de puntos muy grandes.

Aquí observamos el método "FuerzaBruta" que tiene como entrada una lista que contiene objetos de tipo "punto" que tiene a su vez los atributos asociados a una coordenada en el espacio.

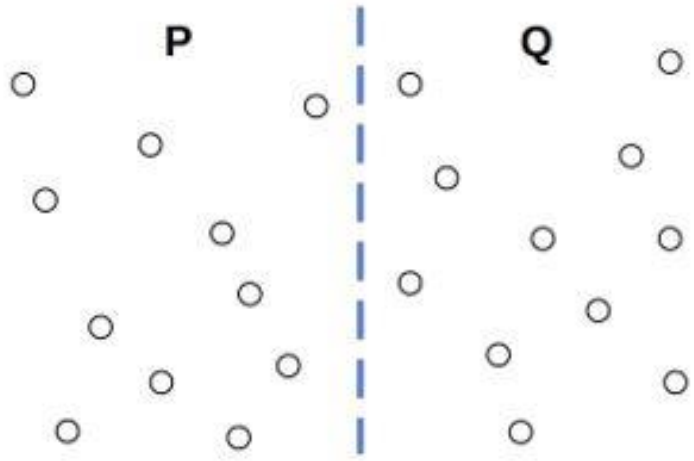
El método “fuerzaBruta” está conformado por dos ciclos “for”(para) aniñados lo que sugiera una complejidad temporal cuadrática que evidentemente para tamaños de conjuntos muy grandes este método es completamente ineficiente.

```
public double fuerzabruta(ListaSimple list) {
    double min = MIN_VAL;
    ListaSimple.Nodo p;
    ListaSimple.Nodo q;
    p = list.head;
    q = list.head.next;
    while (p.next != null) {
        while (q.next != null) {
            double dist = punto.distancia(p.data, q.data);
            if (dist < min) {
                min = dist;
            }
            min = min(min, dist);
        }
    }
    return min;
}
```

Sin embargo, podríamos diseñar un algoritmo más eficiente utilizando la estrategia divide y vencerás, obteniendo dos subconjuntos de puntos como subconjunto de la izquierda I, y la segunda como como subconjunto de la derecha D.

```
int mitad = inicio + (ancho - inicio) / 2;
punto coord_mitad = find(ordenX, mitad).data;
```

Dicha línea vertical imaginaria es tal que todos los puntos de I estén a su izquierda, y todos los de D están su derecha.



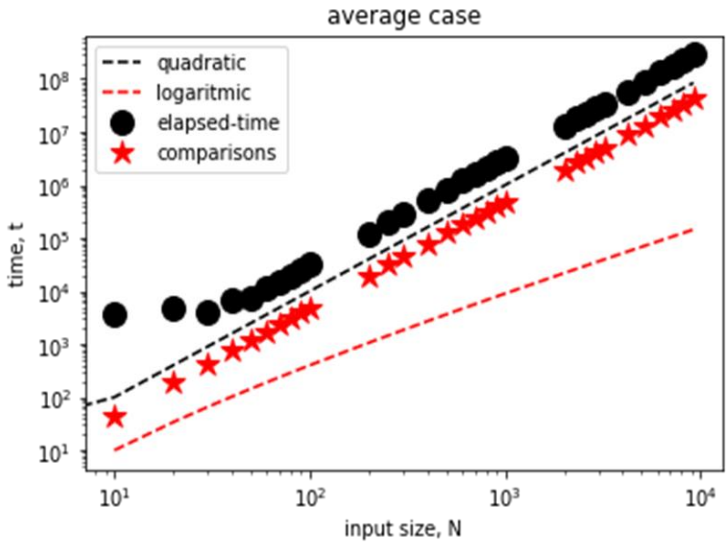
Sin embargo, podría existir la posibilidad de que la distancia mínima se encuentre en un punto intermedio entre la parte derecha e izquierda, para este caso se crea una franja de distancia

```
double distanciaMin_franja = distancia_minFranja(ordenY, franjaIZQ, franjaDER);
return min(distanciaMin, distanciaMin_franja);
```

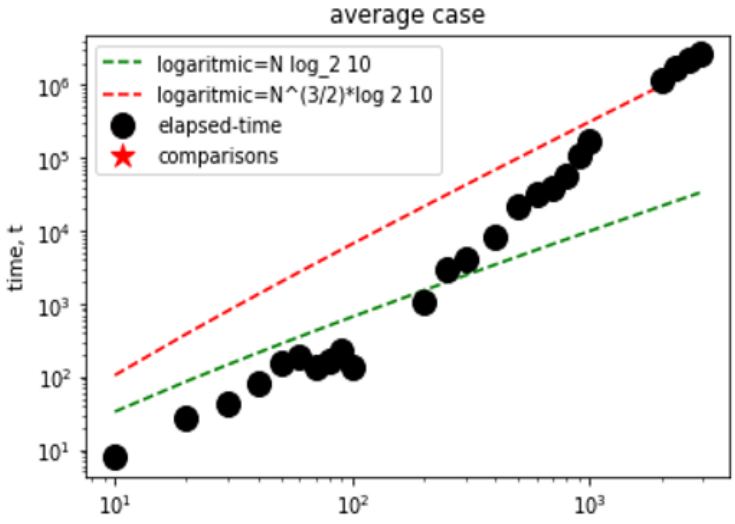
igual a la mínima distancia hallada hasta el momento e identificamos si los puntos ubicados en esta franja se encuentran a una distancia menor a la ya obtenida.

V. RESULTADOS

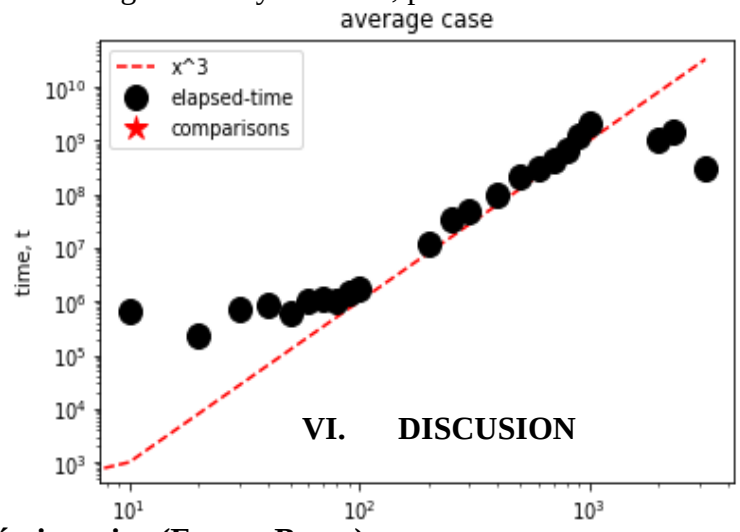
Usando la metodología de fuerza bruta



Usando la metodología divide y vencerás, pero usando arreglos unidimensionales.



Usando la metodología divide y vencerás, pero usando listas enlazadas.



VI. DISCUSION

1. Solución iterativa (Fuerza Bruta)

El método básico para la solución del problema de encontrar los dos puntos más cercanos en el plano, cae dentro de los conocidos como métodos de fuerza bruta, y consiste en comparar cada punto con el resto de los puntos del conjunto, a fin de ir calculando la distancia entre estos, e ir guardando los puntos que resulten con la distancia mínima.

Se ve claramente que este algoritmo, se puede calcular:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 1$$

$$\sum_{i=1}^{n} n$$

$$n \cdot n$$

$$T_n = n^2$$

2. Divide y Vencerás

El algoritmo diseñado con la lógica dividir y vencerás divide el conjunto de puntos por la mitad, es decir N/2 posteriormente se llama a sí misma recursivamente en cada mitad (Tanto a la izquierda como a la derecha) y luego realiza algunas operaciones para combinar las respuestas.

Por lo que se realizan las llamadas recursivas N/2 (Para el conjunto izquierdo) + N/2 Para el conjunto derecho) = 2[T(N/2)] construyendo la siguiente ecuación de recurrencia:

2.1.Usando arreglos como estructura de datos

T(N) = 1 para todo N=1
$T(N) = 2T(\frac{N}{2}) + N$ para todo $N \geq 2$

$$T(N) = 2T(\frac{N}{2}) + N$$

$$T(\frac{N}{2}) = 2T(\frac{N}{2^2}) + \frac{N}{2}$$

k=3

$$T(N) = 2^3 \cdot T(\frac{N}{2^3}) + 2^2 \cdot \frac{N}{2^2}) + 2 \cdot \frac{N}{2}) + n$$

$$T(N,k) = 2^k \cdot T(\frac{N}{2^k}) + \sum_{i=1}^{k} N$$

Pero, $T(1) = 1$

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

Aplicando propiedades del logaritmo para despejar k;

$$log_2 N = k$$

Remplezando k en las ecuacion de la complejidad temporal:

$$T(N, k = log_2 N) = 2^{log_2 N} \cdot T(\frac{N}{2^{log_2 N}}) + \sum_{i=1}^{log_2 N} N$$

$$T(N) = N \cdot T(1) + log_2 N * N$$

Se determina entonces que el algoritmo que encuentra el par más cercano usando la lógica de divide y vencerás con arreglos tiene una complejidad temporal de:

$$\Theta(N * log_2 N)$$

2.2.Usando listas simples como estructura de datos

Los arreglos a diferencia de las listas enlazadas admiten un acceso aleatorio, es decir que se puede acceder a los elementos del arreglo directamente usando su índice, como arr[0]para el primer elemento, arr[6]para el séptimo elemento, etc. Por lo tanto, acceder a los elementos de un arreglo es rápido con una complejidad de tiempo constante de O(1). Por el contrario, las listas enlazadas

admiten un acceso secuencial, lo que significa que para acceder a cualquier elemento/nodo en una lista enlazada, tenemos que atravesar secuencialmente la lista enlazada completa, hasta ese elemento. Para acceder al elemento n de una lista enlazada, la complejidad del tiempo es O(N).

$$T(N) = \sum_{i=1}^{N} 1$$
$$T(N) = N$$

Por lo que, teniendo en cuenta este número de operaciones adicionales realizadas por utilizar una lista simple como estructura de datos obtenemos esta nueva ecuación de recurrencia:

$T(N) = 2T(\frac{N}{2} + N^3$

$T(1) = 1$

Llamados recursivos

$$T(N) = 2 \cdot T\left(\frac{N}{2}\right) + N^3$$

$$T\,\frac{N}{2}\Big) = 2 \cdot T\left(\frac{N}{2^2}\right) + \left(\frac{N}{2}\right)^3$$

$$T\left(\frac{N}{2^2}\right) = 2 \cdot T\left(\frac{N}{2^3}\right) + \left(\frac{N}{2^2}\right)^3$$

Irteraciones
k=1

$$T(N) = 2 \cdot T\left(\frac{N}{2}\right) + N^3$$

k=2

$$T(N) = 2 \cdot \left[2 \cdot T\left(\frac{N}{2^2}\right) + \left(\frac{N}{2}\right)^3\right] + N^3$$

k=3

$$T(N) = 2 \cdot \left[2 \cdot \left(2 \cdot T\left(\frac{N}{2^3}\right) + \left(\frac{N}{2^2}\right)^3\right) + \left(\frac{N}{2}\right)^3\right] + N^3$$

Ecuacion parametrica

$$T(N) = 2^2 \cdot \left(2 \cdot T\left(\frac{N}{2^3}\right) + \left(\frac{N}{2^2}\right)^3\right] + \left(\frac{N}{3}\right) + N^3$$

$$T(N) = 2^3 \cdot T\left(\frac{N}{2^3}\right) + 2^2 \cdot \left(\frac{N}{2^2}\right)^3 + 2\left(\frac{N}{2}\right)^{\dot{3}} + N^3$$

$$T(N) = 2^3 \cdot T\left(\frac{N}{2^3}\right) + 2^2 \cdot \left(\frac{N}{2^2}\right)^3 + 2\left(\frac{N}{2}\right)^{\dot{3}} + N^3$$

$$T(N) = 2^3 \cdot T\left(\frac{N}{2^3}\right) + 2^2 \cdot \left(\frac{N}{2^6}\right) + 2\left(\frac{N}{2^3}\right)^{\cdot} + N^3$$

$$T(N) = 2^3 \cdot T\left(\frac{N}{2^3}\right) + \left(\frac{N}{2^4}\right) + \left(\frac{N}{2^2}\right) + N^3$$

$$T(N,k) = 2^k \cdot T\left(\frac{N}{2^k}\right) + \sum_{i=0}^{k} \frac{N^3}{2^i}$$

Aplicando propiedades de la sumatoria

$$T(N,k) = 2^k \cdot T\left(\frac{N}{2^k}\right) + N^3 \cdot \left(2 - \left(\frac{1}{2}\right)^k\right)$$

Pero T(1)=1, por lo que remplezando tenemos que:

$$\frac{N}{2^k} = 1$$

$$2^k = N$$

Aplicando propiedades de los logaritmos para despejar k tenemos que:

$$log_2 N = k$$

Remplazando k de la ecuacion, tenemos que:

$$T(N, k = log_2 N) = 2^{log_2 N} \cdot T\left(\frac{N}{2^{log_2 N}}\right) + N^3 \cdot \left(2 - \left(\frac{1}{2}\right)^{log_2 N}\right)$$

$$T(N) = N \cdot 1 + N^3 \cdot \left(2 - \frac{1^{log_2 N}}{2^{log_2 N}}\right)$$

$$T(N) = N + N^3\left(2 - \frac{1}{N}\right)$$

$$T(N) = 2 \cdot N^3 - N^2 + N$$

Se determina entonces que el algoritmo que encuentra el par más cercano usando la lógica de divide y vencerás con listas simples tiene una complejidad temporal de:

$$\Theta(N^3)$$

VII. CONCLUSION

Finalmente se logró el objetivo de encontrar una solución mediante el diseño de un algoritmo que resuelva el problema del par de puntos más cercanos con una complejidad temporal menor que la obtenida usando el método de “fuerza bruta", es decir una complejidad menor que O(N^2) para así obtener resultados de forma eficientes y poder aplicar la misma solución a varios problemas de la actualidad que tienen una estrecha relación con el problema aquí planteado. Hemos verificado con los métodos teóricos para el análisis de algoritmos recursivos que el algoritmo recursivo de variante usando como estructura de datos un arreglo y cual fue explicado ampliamente en el pasado informe calcula la distancia entre pares de puntos con una complejidad: $\Theta(N * log_2 N)$

De entrada, esto es muy importante porque se logró mejorar la eficiencia y el problema ahora si puede admitir conjunto más extenso y grande que los que cabría esperar con el algoritmo de fuerza bruta, reduciendo el número de los cálculos que se realizando siendo menos excesivos y al final más eficiente.

Sin embargo, este resultado exitoso no fue únicamente consecuencia de una optimización en los procesos algorítmicos realizados por el algoritmo, sino también por las decisiones tomadas en cuanto a la utilización de que estructura de datos usar, ya que, en este trabajo de demostró teórica y empíricamente que incluso la estructura de datos que tomemos para almacenar el par de puntos es de vital importancia para obtener una mejor eficiencia. En este caso vimos que solo cambiar la estructura de datos usada y usar una lista simple como estructura de datos, la complejidad del algoritmo fue de: $\Theta(N^3)$

Es decir, fue incluso peor que la complejidad del algoritmo que encontraba la distancia mínima utilizando la metodología de fuerza bruta.

Esto nos deja una conclusión de vital importancia, y es que cuando de habla de complejidad temporal incluso las estructuras de datos que utilicemos serán de gran relevancia para obtener mas eficiencia, por lo que, en este caso(algoritmo recursivo que encuentra el par de puntos más cercanos), implementar el algoritmo usando lista simples es una muy mala decisión, puesto a que cualquier elemento/nodo en una lista enlazada, tenemos que atravesar secuencialmente la lista enlazada completa, dándonos un número de operaciones adicionales que usando la estructura de datos arreglos no ocurre. Por lo que tenemos que ser conscientes de las ventajas y desventajas que nos ofrecen las diferentes estructuras de datos que conocemos para sacar el mayor provecho (en cuanto a eficiencia) de nuestras implementaciones.

VIII. REFERENCIAS

Application of Divide and Conquer Method in Solving the "Closest Pair" Problem - Programmer Sought. (s. f.). Programmer Sought. https://www.programmersought.com/article/18615345367/

Closest Pair of Points - Divide and Conquer - Learn in 30 sec from Microsoft Awarded MVP. (s. f.). Wikitechy. https://www.wikitechy.com/technology/closest-pair-of-points/

How to Find the Two Points With the Minimal Manhattan Distance | Baeldung on Computer Science. (s. f.). Baeldung on Computer Science. https://www.baeldung.com/cs/minimal-manhattan-distance

Closest Pair of Points Problem. (s. f.). Recuperado 4 de noviembre de 2022, de https://www.tutorialspoint.com/Closest-Pair-of-Points-Problem

Finding the nearest pair of points - Algorithms for Competitive Programming. (s. f.). Recuperado 4 de noviembre de 2022, de https://cpalgorithms.com/geometry/nearest_points.html

GeeksforGeeks. (2022, 13 julio). Closest Pair of Points using Divide and Conquer algorithm. https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/