I + D: Problema del Par Más Cercano

Santiago Andrés Mercado Barandica y David Salgado Cortes

Abstract

En clase hemos analizado la complejidad de tiempo de diferentes algoritmos y cómo se relaciona esta complejidad con el número de conditional checks que realiza el algoritmo. El objetivo de este laboratorio es analizar la complejidad del algoritmo junto con los conditional checks que este realiza y analizar cómo cambia la complejidad implementando una lista menos eficiente siendo esta una LinkedList (menos eficiente) que un ArrayList. El algoritmo crea una lista con N coordenadas enteras aleatorias, luego organiza las coordenadas según su posición en x. El programa utiliza la estrategia de Divide and Conquer con recursividad para particionar el set de coordenadas en grupos más pequeños. Una vez el grupo de coordenadas sea de tamaño 3 o menor, utiliza el algortimo de fuerza bruta para hallar el par de puntos más cercano de cada subgrupo de coordenadas. Luego, se queda con el par con la menor de todas las distancias y la utiliza para buscar pares de puntos cercanos de diferentes subgrupos y hallar la distancia mínima de los mismos con el algoritmo de fuerza bruta. Con esto el algoritmo devuelve el par de puntos más cercano de todo el set de coordenadas. Este proceso se realiza 256 veces para cada set de coordenadas para encontrar el tiempo de ejecución promedio para el programa dado. No obstante, diferente al anterior en este algoritmo fue necesario implementar las determinadas LinkedList en lugar de ArrayList, nosotros sabemos de que para acceder a cierto dato es necesario recorrer toda la lista hasta llegar al lugar deseado, ya que no contienen la propiedad del índice para acceder a un elemento directamente. Con esto se comprobó que la complejidad del algoritmo recursivo que resuelve el Closest Pair Problem es de O(n^2) cuando se divide por mitades el set de coordenadas.

I. Introducción

Actualmente, se ha hecho más visible la necesidad de contar con algoritmos más eficientes para nuestras aplicaciones, ya sea por la cantidad de datos procesados o por la necesidad de respuestas rápidas. [1]. El análisis de los algoritmos y su eficiencia respecto a memoria usada y tiempo de ejecución es algo extremadamente importante hoy en día. Tener la posibilidad de mejorar tu algoritmo es algo que siempre debemos buscar como ingenieros de sistemas para proporcionar a nuestros clientes los mejores resultados. Por esa razón, es necesario analizar en que situaciones nuestros algoritmos presentan una eficiencia mayor que en otros casos en promedio y poder proporcionar diferentes casos y saber el comportamiento de nuestro algoritmo. La complejidad de un algoritmo es una medida de su tiempo de ejecución basado en el número de operaciones que ejecuta para resolver un problema. [2]. Por esta razón, se busca que un algoritmo tenga menor complejidad; es decir que tarde menos tiempo en completar su tarea. Al ser un recurso tan importante, optimizar el tiempo que se tarda el algoritmo es crucial para la productividad, en especial cuando se trata de operaciones largas y complejas.

II. Definición del Problema

Implementar el algoritmo que encuentra el par de puntos más cercano de un set de coordenadas de forma recursiva dividiendo el set de coordenadas a subsets más pequeños y aplicando fuerza bruta a estos subsets. Estos sets y subsets son creados utilizando LinkedLists. Por ende, el algoritmo se hace de forma que al final devuelva la cantidad de comparaciones y el tiempo transcurrido necesario para lograrlo y poder analizar la complejidad del algoritmo. Posteriormente es necesario graficar los resultados del tiempo y comparaciones en gráficas para poder observar y analizar el comportamiento de nuestro algoritmo con respecto al tiempo. Los resultados esperados son que el algoritmo tenga una complejidad de O(n^2) al dividir por mitades.

Algoritmo 1 ilustra el algoritmo de fuerza bruta para hallar el par más cercano usado para resolver el problema.

Algorithm 1 Fuerza Bruta

```
dmin ← INF
for i = [1, N − 1] do
  for j = [i+1, N - 1] do
    d ← distance(coords, i, j)
    if d < d_min then
      first ← i
      second ← j
      dmin ← d
    end if
  end for
end for
return(first, second, dmin)
```

III. Metodología

Para desarrollar el problema, lo primero que hace el algoritmo es crear un set con n coordenadas enteras aleatorias que se almacenan en una lista que después es transformada a Linkedlist. Posteriormente, se toma esta lista y se ordena de forma ascendente según las posiciones en x. Esta lista ordenada es luego utilizada como parámetro de entrada para la subrutina recursiva que se encarga de encontrar el par más cercano. Esta subrutina verifica el tamaño del set de coordenadas, y si es mayor a 3 lo divide en 2 subsets de coordenadas. Esta división se hace de forma recursiva llamandose a sí misma 2 veces, una recibe como parámetro de entrada la primera mitad del set de coordenadas y la otra recibe la segunda mitad. Cuando el set de muestra es de tamaño menor o igual a 3, se aplica el algoritmo de fuerza bruta para hallar el par más cercano de estos.

Luego, se compara la distancia mínima obtenida de ambos subsets y se queda con el par de coordenadas con la menor distancia mínima entre ambos. Se utiliza una subrutina llamada FindCandidates que recorre el set de coordenadas y compara la distancia en x y y entre sus puntos con la distancia mínima dada para hallar posibles puntos cercanos. Si la distancia entre las posiciones de x y y es menor que la distancia mínima, se añaden ambas coordenadas a una lista de candidatos. Esta búsqueda comienza con el par más cercano a la división del set y se va alejando de forma cíclica. El ciclo se detiene en el momento en el que un par de coordenadas excede la distancia mínima. A los candidatos se les aplica también el algoritmo de fuerza bruta y se comparan las distancias mínimas. El programa se devuelve la menor de ellas y las coordenadas asociadas a ella. Todos los sets y subsets son Linkedlists, siendo la clase creada por nosotros.

Para determinar el comportamiento de la gráfica tiempo vs n^2 para cada caso, se ejecuta el proceso anterior contando cada condicional (Si) que utiliza la subrutina recursiva y el tiempo de ejecución de la misma. No se tiene en cuenta los procesos de creación y ordenamineto del set de coordenadas para el análisis. Este procedimiento se realiza 256 veces con un mismo set de coordenadas, y se halla un promedio del tiempo de ejecución y complejidad (uso de condicionales). Luego, los datos obtenidos se guardan en un archivo con 3 columnas: n, comparaciones, tiempo de ejecución. Este proceso se realiza con un set de coordenadas de de tamaño n=100 que aumenta en factor de 4/3 hasta llegar a n=50000. Tras la ejecución del algoritmo, se obtiene un archivo con los resultados necesarios para hallar la complejidad del algoritmo y los cuales son usados para graficar los resultados de n^2 vs tiempo con la ayuda de Python y MatLab. Por último, se hace un ajuste de la gráfica para determinar su complejidad y sacar las conclusiones del experimento.

IV. Resultados

Para los resultados fue necesario recolectar dentro un TXT los datos obtenidos que eran necesarios para el análisis de complejidad. Dentro de este TXT de resultados se presentan 3 columnas. La primera columna hace referencia a la cantidad de números que se encuentran dentro del TXT que estamos leyendo. La segunda columna hace referencia a la cantidad de comparaciones realizadas durante todo el algoritmo y finalmente la tercera columna hace referencia al tiempo que se demoró el algoritmo en realizar todo el trabajo solicitado. Los resultados del experimento son los siguientes:

TABLE I
Resultados obtenidos y almacenados en el archivo TXT

Tamaño	Operaciones	Tiempo transcurrido
100	283	25784
133	458	31079
177	530	38315
236	580	36428
314	1338	102542
418	1147	114688
557	1915	203819
742	1959	368371
989	2581	591506
1318	4388	998987
1757	4891	1760155
2342	9391	3169843
3122	9443	5635615
4162	18044	10123861
5549	15611	17926071
7398	18680	31907792
9864	31867	56930489
13152	41973	101362355
17536	75221	180858082
23381	74385	321426699
31174	78275	573974156
41565	195568	1030328865

Como se puede observar, el número de operaciones claramente son lineales por lo cual, como estamos analizando en una gráfica de n^2 no fueron incluidas dentro de la gráfica. La gráfica obtenida con estos datos es la siguiente:

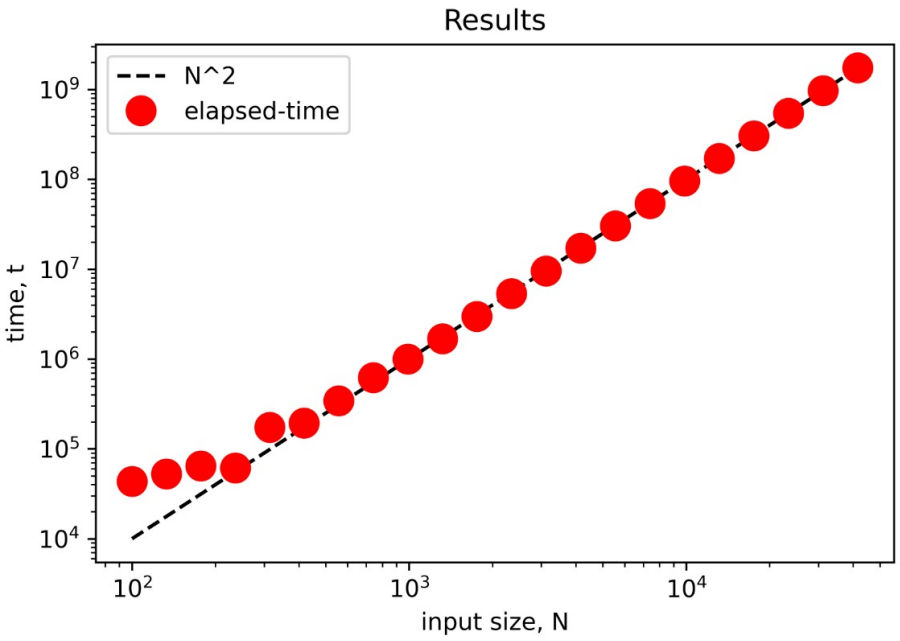

Fig. 1. Número promedio de operaciones y tiempo de ejecución como función del tamaño del set de coordenadas.

Como se puede observar en la gráfica, el comportamiento de los datos es claramente cuadráticos. La línea punteada representa el comportamiento esperado para el experimento, el cuál es cuadrático. Los resultados obtenidos siguen el comportamiento esperado, puesto que la pendiente también corresponde a un comportamiento cuadrático. Los primeros datos de tiempo transcurrido difieren de lo esperado, pero estos no representan el comportamiento real del algoritmo puesto que son solo 4 datos atípicos que se deben a que las primeras ejecuciones pueden tardar más de lo esperado en lo que se normaliza su comportamiento. Otra observación importante es que algunos puntos se encuentran ligeramente por encima de la pendiente. Esto puede deberse a que la lista de posibles candidatos es la que mayor complejidad y tiempo de ejecución añade al algoritmo puesto que utiliza la función de búsqueda por fuerza bruta a todos los candidatos que pueden ser muchos más de 3. De esta forma, si la Linkedlist de candidatos es muy grande el tiempo que se tarda en completar la tarea puede aumentar y viceversa. Esto podría ser una posible explicación para dicha variación.
Ahora bien, la complejidad del algortimo resulta O(n^2) debido a que se usa la recursividad en listas enlazadas. El algoritmo tiene 2 procesos principales, aplicar el algoritmo de fuerza bruta y encontrar los posibles puntos más cercanos (candidatos). Normalmente, el algoritmo de fuerza bruta tendría la mayor complejidad, pues el número de comparaciones sería de la forma:

$$\sum_{j=i+1}^{N}\left(\sum_{i=0}^{N}1\right)=\sum_{i=0}^{N}N-i=\sum_{i=0}^{N}i=\frac{N(N+1)}{2}$$

La complejidad del método de Fuerza Bruta es claramente de $O(n^2)$ por lo que parecería que esta es la complejidad de todo el programa. Sin embargo, al dividirse el set de datos en subsets de tamaño 3 o menor, el algoritmo de fuerza bruta tendrá un N de como máximo 3. Es decir que solo hará entre 1 y 6 comparaciones, que no representan practicamente nada de complejidad tomando en cuenta que nuestro N es de más de 100. Por ende, podemos tomar esta complejidad como constante O(1). Por otro lado, la subrutina que encuentra los candidatos es una búsqueda lineal y del set de coordenadas. La complejidad de una búsqueda lineal es, como su nombre lo indica, lineal; O(n). Esto se demuestra ya que el número de operaciones está dado por la fórmula:

$$\sum_{i=0}^{N/2-1}1+\sum_{i=N/2}^{N}1=\sum_{i=0}^{N}1=N+1$$

Sin embargo, al tener que recorrer los sets y subsets ya no es tan fácil como lo era con las ArrayList, he aquí donde varía la complejidad de este algoritmo ya que al tener listas enlazadas, siempre es necesario recorrer la lista enlazada desde el primer elemento hasta el que se esté buscando, ya que no se puede acceder a cierto elemento a través de su indice. Por otro lado, tambien mientras se va recorriendo elemento por elemento tambien se hace el proceso de comparación. Con lo cual, en

resumen, hallar y resolver la lista de candidatos tendría una complejidad de O(n^2) en lugar de O(1) como fue planteado con las ArrayList en el laboratorio anterior Por tanto, la ecuación de complejidad de nuestro código sería

$$T(N) = 2T(N/2) + N^2$$

Donde N^2 representa la complejidad de la lista de candidatos como fue explicado anteriormente. Luego, se resuelve esta ecuación con la ecuación maestra y se obtiene que a=2, b=2 y d=2. Luego la ecuación maestra dicta que $a < b^d = 2 < 4$ por lo que la complejidad del algoritmo sería

$$T(N) = O(N^d) = O(N^2)$$

V. Conclusión

En conclusión, podemos decir que todos los objetivos de este laboratorio se cumplieron de manera exitosa al poder observar y analizar el comportamiento de los datos (input size, comparaciones y tiempo de ejecución) para el problema planteado. Como se logra evidenciar en los resultados de todos los casos, a mayor es el input size (n) mayor es el tiempo de ejecución y también es mayor el número de comparaciones. Además, la complejidad del algoritmo obtenida es la misma complejidad esperada: O(n^2). También se logró demostrar matematicamente la razón de esta complejidad cuadrática con lo aprendido en clase. Podemos concluir que con el aumento de los elementos se aprecia un comportamiento cuadrático, demostrando que el número de comparaciones es proporcional al tiempo de ejecución del algoritmo, tal como se vió en la gráfica resultante.No obstante, tambien podemos analizar la diferencia en el tiempo de complejidad entre O(N) y O(nN^2) con al laboratorio anterior donde se utilizó ArrayList en lugar de LinkedList como fue solicitado en este laboratorio. Podemos observar que las listas enlazadas presentan una mayor complejidad para el algoritmo por el simple hecho de que si queremos llegar a cierto elemento siempre es necesario recorrer toda la lista y no por indices. Todas estas conclusiones entre el uso de las 2 diferentes listas fueron comprobadas matemáticamente logrando así el resultado esperado para este laboratorio. Por otro lado, durante este laboratorio no se presentaron inconvenientes específicos. Para el desarrollo del algoritmo la única dificultad fue desarrollar el método de buscar los pares de coordenadas cercanos a las divisiones de subsets. El resto simplemente fue reutilizar el algoritmo para crear, editar y leer archivos .TXT y el algoritmo otorgado por el profesor en Python para graficar nuestros resultados y lograr nuestros objetivos. Para el análisis de resultados, tampoco se presentaron inconvenientes ya que se obtuvieron los resultados esperados. Con esto concluye así el laboratorio satisfactoriamente.

References

[1] W. Rojas, “Análisis de complejidad del algoritmo: ¿cuál es la importancia?” 2022.
[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, ser. Computer Science. MIT Press, 2009.