Algorithms and Complexity(I+D)

Juan Sebastian Neira Davila-200124788

November 2022

Abstract

El objetivo de este informe es analizar la complejidad del algoritmo par más cercano o closet pair, el cual consta de hallar la distancia más corta entre dos coordenadas en un plano dividido en dos, por lo tanto, en el presente trabajo hablaremos de las diferencias entre la implementación de este algoritmo en diferentes estructuras de datos (Arrays y listas enlazadas), se comparará el tiempo y la eficiencia de estos para así saber su complejidad.

1 Introducción

Dentro de las ciencias de la computación, el análisis de los algoritmos toma un papel importante, ya que se pueden tener muchas soluciones para resolver un problema, sin embargo, es relevante escoger la solución mas eficiente, de no ser así los recursos requeridos pueden llegar a ser un problema o un limitante.

El time complexity o la complejidad temporal, es una de las herramientas que tenemos para reconocer que algoritmo es mas eficiente, este funciona como una escala de medición para los algoritmos, se basa en la cantidad de comparaciones que realiza un algoritmo para resolver un problema, por lo tanto, el algoritmo que necesite el menor de número de operaciones es considerado el más eficiente (Aditya, 2019). En este trabajo usaremos la notación Big O para representar la complejidad temporal, existen diferentes notaciones para representarla, las cuales son: La notación constante O(1), Logarítmica O(log n), lineal O(n), N-log-N O(n log n), cuadrática $O(n^2), exponencial O(c^n), entre otras.$

Big O Notation Summary

Notation	Type	Examples	Description
O(1)	Constant	Hash table access	Remains constant regardless of the size of the data set
O(log n)	Logarithmic	Binary search of a sorted table	Increases by a constant. If n doubles, the time to perform increases by a constant, smaller than n amount
O(< n)	Sublinear	Search using parallel processing	Performs at less than linear and more than logarithmic levels
O(n)	Linear	Finding an item in an unsorted list	Increases in proportion to n. If n doubles, the time to perform doubles
O(n log(n))	n log(n)	Quicksort, Merge Sort	Increases at a multiple of a constant
$O(n^2)$	Quadratic	Bubble sort	Increases in proportion to the product of n*n
$O(c^n)$	Exponential	Travelling salesman problem solved using dynamic programming	Increases based on the exponent n of a constant c
O(n!)	Factorial	Travelling salesman problem solved using brute force	Increases in proportion to the product of all numbers included (e.g., 1*2*3*4...)

Figure 1: Big O Notation Summary

2 Metodologia

Para resolver el problema del par mas cercano, se desarrollo un algoritmo que primero calculaba las distancias mediante una función llamada getDistance(), se guardaba en un arraylist y mediante el método de fuerza bruta se comparaban las distancias, luego de dividir el plano en dos cuadrantes, se procede a contrastar las distancias de ambos lados y el centro del plano mediante la función mindistance para hallar la menor distancia de todas, también definimos una función llamada GetRandoms(), la cual se encarga de generar al azar los puntos que se usaron en las coordenadas, además hacemos uso de otra función llamada Punto para almacenar las coordenadas "X" y "Y" y posteriormente usarlo en los métodos anteriormente descritos. Luego, se realizó una modificación en las estructuras de datos, por lo cual se cambia el lugar donde se almacenan los datos de un arraylist a una lista enlazada, sin embargo, se siguió con la misma lógica empleada, esto se hace con el fin de analizar la complejidad temporal al trabajar este problema con diferentes estructuras de datos. A la hora de pasar este algoritmo a su version recursiva se realiza un cambio en el método que recorre la matriz y ambas funciones para hallar la distancia (getDistance y mindistance), porque se requiere reproducir los recorridos que se realizan iterativamente de otra forma, por lo cual hacemos uso de condicionales y llamados a la misma

función.

3 Desarrollo

Primero se desarrolló el código de closet pair hallándose mediante el método de fuerza bruta, al principio se le ingresaban todos los datos al código, ya que el fin solo era saber si se estaba hallando correctamente la distancia y distancia mínima de un par de puntos. Luego implementamos la función GetRandoms que se encarga de generar los puntos de una manera aleatoria, se procedió a modificar un poco la función de mindistance para poder determinar la distancia mínima de una manera correcta, ya que al hacerlos con números fijos se le asignaba el tamaño y los puntos. Por ultimo al hacer el cambio a listas enlazadas se hizo uso de la clase LinkedList de java la cual facilito mucho el cambio de estructura de datos, la distancia que hay entre los 2 puntos ((a.X,a.Y)y(.X,b.Y)) se asignan con la posición de los iteradores o variables de control.

4 Resultados

Table 1: El número de operaciones y el tiempo transcurrido (nanosegundos) en función del tamaño de entrada. (ClosetPairConArray)

Tamaño	Operaciones	Elapsed Time
10	45	27500
20	190	88200
30	435	96700
408	780	104300
50	1225	203800
60	1770	224300
70	2415	308900
80	3160	390500
90	4005	549500
100	4950	632600
110	5995	751100
120	7140	899100

Gráficos del número promedio de operaciones y el tiempo de ejecución, Figura 2,3,4 y 5.

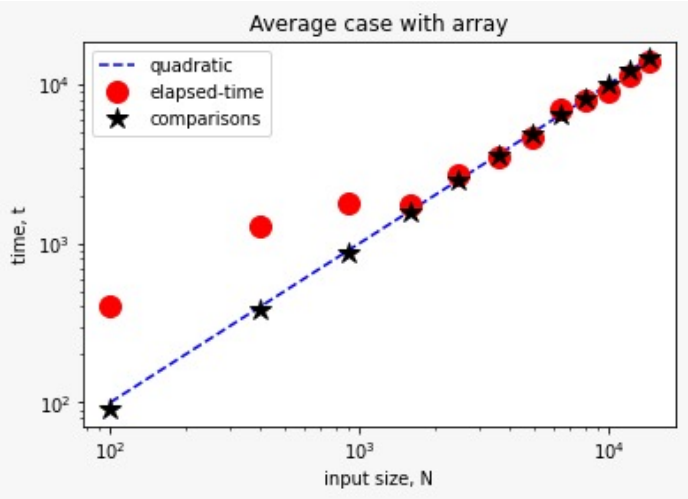

Figure 2: average case with non-recursive array.

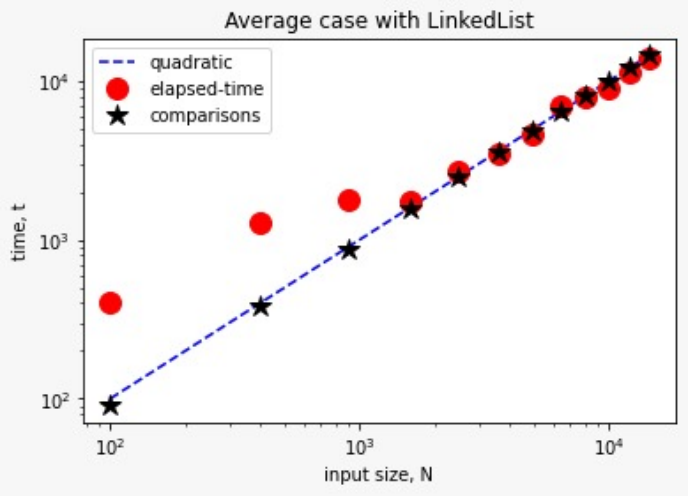

Figure 3: average case with non-recursive LinkedList.

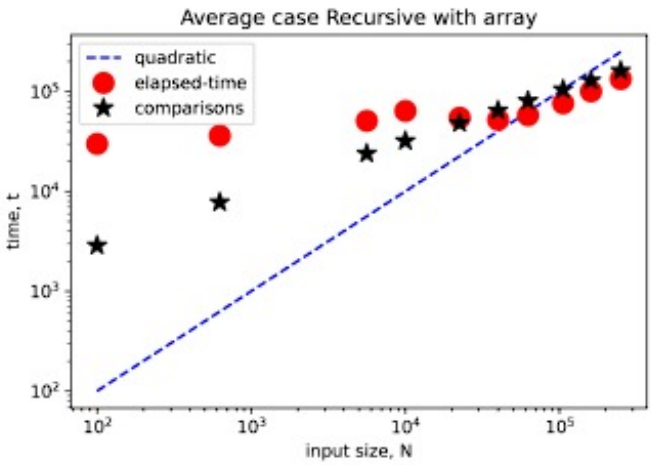

Figure 4: average case with recursive array.

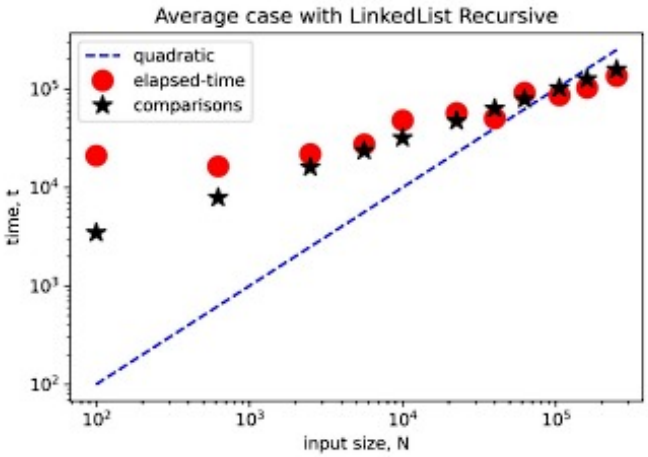

Figure 5: average case with recursive LinkedList.

5 Conclusiones

Al observar los resultados de los algoritmos de listas enlazadas y arrays no recursivos, notamos que no importa el tamaño n ya que siempre se recorrerá el todo el array por lo cual tiene un orden n elevado a la 2,sin embargo la complejidad de estos cuatro algoritmos es correspondiente a O(n log n),siendo casi lineales, como se puede observar en los gráficos, también se puede observar que el numero de comparaciones permanece igual en ambos algoritmos, además notamos que al

usar listas enlazadas en vez de los arrays hay un ligero incremento en el tiempo transcurrido, lo cual con N de pequeños valores puede llegar a ser irrelevante pero entre mas incrementos dicho tamaño mayor será la diferencia entre estos, esta diferencia se debe a que la lista enlazada necesita de más información para recorrerlo, mientras que el array se hace con un simple ciclo for.
Con respecto a los algoritmos recursivos notamos el mismo patrón que en los no recursivos dando una complejidad O(nlogn), también se ve un comportamiento menos eficiente por parte del algoritmo de las listas enlazadas, dada su naturaleza de necesitar punteros para recorrerse. A pesar de tener la misma complejidad tienen una menor eficiencia, lo cual se debe a la misma recursividad, ya que se usan pilas y colas para resultados momentáneos y también gracias a las llamadas pendientes del mismo método recursivo.

6 Referencias

Figura1.donkcowan, donkcowan., 11 5 2013. [En línea]. Available: https://www.donkcowan.com/blog/2013/5/1
o-notation. [Último acceso: 20 11 2022].
Aditya, freecodecamp, 10 junio 2019. [En línea]. Available: https://www.freecodecamp.org/news/time-
complexity-of-algorithms/. [Último acceso: 20 noviembre 2022].