REPORTE I+D

ANÁLISIS DE COMPLEJIDAD DEL ALGORITMO CLOSEST PAIR

DEPARTAMENTO INGENIERÍA DE SISTEMAS

IST 4310 ALGORITMIA Y COMPLEJIDAD

PROF. M. DIAZ MALDONADO

DAVID ESCORCIA JIMENEZ
200123749

HORACIO SERPA RIVERA
200127769

UNIVERSIDAD DEL NORTE

NOVIEMBRE 18/2022, BARRANQUILLA

I. Abstract

El presente reporte tiene como fin presentar el análisis de complejidad del algoritmo de fuerza bruta o closest pair, que sirve para hallar la distancia mínima entre coordenadas o puntos establecidos en un plano de 2 cuadrantes, comparando la eficiencia y el tiempo de ejecución al usar dos estructuras de datos diferentes tales como Arrays convencionales y LinkedList, con el fin de analizar la complejidad de forma general.

II. Introduction

Primero se creó un algoritmo que calcule la distancia más cercana entre múltiples puntos que tienen coordenadas en (x,y) guardandolos en una matriz y usando un método de fuerza bruta para comparar distancias entre cierta cantidad de puntos, ya que se deben dividir los puntos en secciones como si estuvieran dispersos sobre un plano de dos cuadrantes, para entonces comparar todas las distancias del lado izquierdo, del cual se obtiene una distancia mínima, al mismo tiempo para el lado derecho se haría lo mismo y se obtiene otra distancia mínima, y por último a los puntos que están más cerca de la línea que divide a la mitad también se les analizaría la distancia, para al final entre estas 3 distancias (distancia izquierda, derecha y central) comparar y definir cuál es la menor de todas. Después se realiza una modificación en la que se cambia la matriz donde estaban guardadas las coordenadas por una estructura que consta de LinkedList, con el fin de realizar la comparación de la complejidad/ eficiencia al trabajar este problema.

El cambio en el funcionamiento del algoritmo recursivo reside en el método encargado de realizar el recorrido en la matriz de coordenadas y su consiguiente comparación de puntos mediante la función distancia(), ya que el camino iterativo consiste en ciclos anidados para recorrer los respectivos índices de cada array anidado para hacer el recorrido de la matriz, la novedad consiste en emular estos recorridos mediante condicionales y llamados recursivos a la misma función.

III. Methodology

Se definió un tamaño N el cual se genera al azar, además de las coordenadas x,y que también se guardan en la matriz de N filas y dos columnas. Para el cambio con linkedlist, nuestro N sigue siendo al azar, sólo que ahora tenemos una linkedlist para las coordenadas en “x“ y una para las coordenadas en “y” el cual también tiene coordenadas que se generan al azar al igual que cuando usamos la matriz, luego se usó una función que encuentra la distancia entre dos puntos y dependiendo del lado en el que esta, va comparando con la actual distancia mínima, hasta que termine de recorrer toda la matriz lo cual supone que encontró

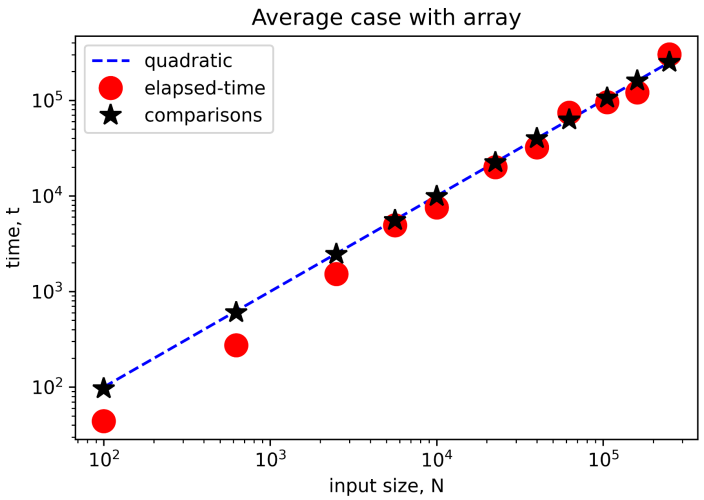
tanto la distancia mínima del lado izquierdo como la del lado derecho y la central, y a su vez la distancia mínima entre esas tres.

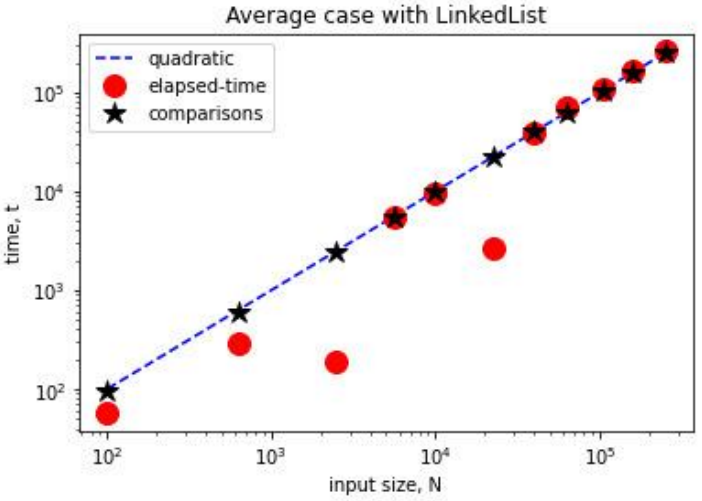
Para el algoritmo recursivo con uso de LinkedList en vez de Arrays, se siguió la misma lógica, la diferencia fue el recorrido de estas listas enlazadas, ya que su funcionamiento interno permite moverse sobre ambas listas aprovechando el cambio de un mismo índice, así se establece una lista para todas las coordenadas en X de los puntos, y otra para las coordenadas en Y, siendo que cada par de elementos de ambas listas con el mismo índice sea un punto.

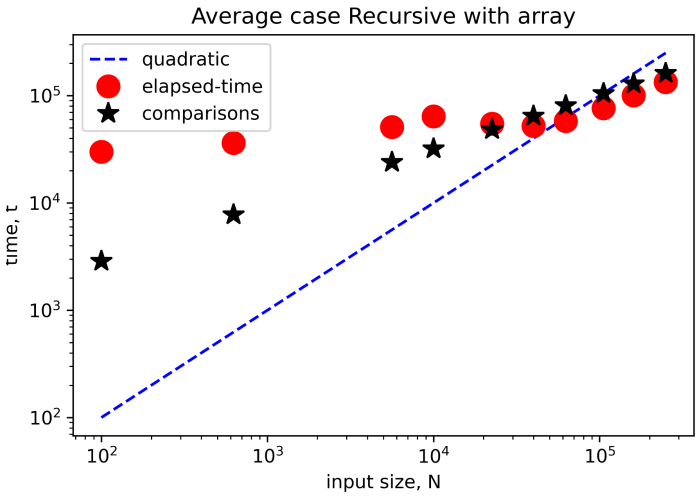
IV. Development History

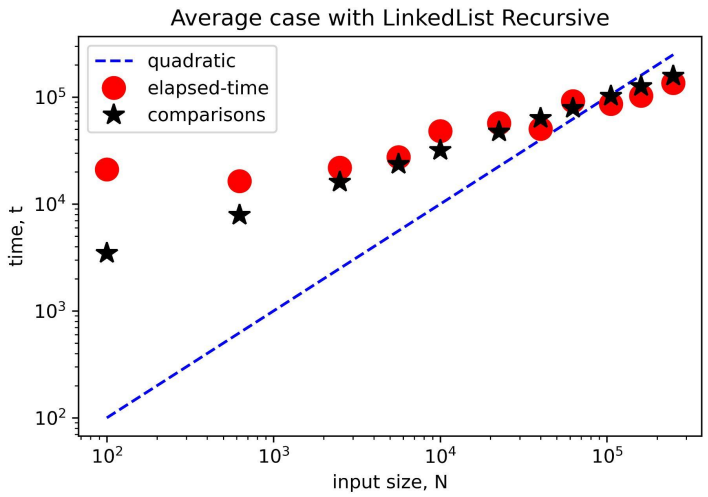
Se comenzó escribiendo el pseudocódigo de fuerza bruta, luego como referencia se definieron 6 puntos para saber si el algoritmo funcionaba bien. Al principio se dio con un error a la hora de realizar el llamado de la función que encontraba la distancia entre dos puntos, ya que se pasan como parámetros dos números fijos, pero luego se cambiaron esos parámetros por i y j que eran los índices del ciclo y funcionó exitosamente.. Lo siguiente fue agregar ciclos if para comparar las tres distancias mínimas que teníamos (la del lado izquierdo, lado derecho y los que estuvieran más al centro) y así determinar la distancia mínima, ya por último se definió el número de puntos con random(), al igual que la generación de coordenadas (x,y). El cambio a linkedlist, fue más rápido de lo esperado puesto es que solo se tuvieron que crear 2 nuevas LinkedList para “x” y “y” de tipo Integer, luego cambiar en las funciones donde recibía el parámetro de la matriz por las nuevas listas tipo linkedlist y también donde hace los llamados, por último en la función distance() para hallar la distancia entre los puntos 1 y 2 compuestos de (x1,y1) y (x2, y2) se asignan en vez de la matriz, los valores enteros de la linkedlist en la posición i y j respectivamente para x1,y1 y x2,y2.

V. Results









VI. Conclusion

Algoritmo No recursivo

Se observó en el algoritmo que usaba la matriz y las linkedlist para dividir el plano en dos lados no era suficiente para hallar la distancia mínima, puesto que en muchos casos habían distancias más pequeñas de puntos que estaban cerca al límite del otro lado pero al no estar en el mismo lado no se comparaban, es por esto que es importante tomar puntos medios y también hacer la comparación de su distancia. Otro dato encontrado es que no importa el tamaño de N porque sea muy grande o muy pequeño esté siempre recorre la matriz en su totalidad por ende este será de orden N^2 . En cuanto a lo observado de comparar el algoritmo en su forma no recursiva usando matriz y usando linkedlist, se

puede observar que lo único constante es el número de comparaciones que hace, pero si hablamos del tiempo transcurrido con las linkedlist es menor no en una gran diferencia pero si puede ser una diferencia considerable como cuando usas N muy grandes como por ejemplo 500 el cual el tiempo en la linkedlist fue de 19031201100 nanosegundos lo cual son 19.031 segundos, mientras que usando una matriz fue de 30599826800 nanosegundos lo cual son 30.599 segundos.

Algoritmo recursivo

El algoritmo de forma recursiva parece ser menos eficiente puesto que por la naturaleza del funcionamiento recursivo, se crea una pila que sirve para guardar resultados parciales y las llamadas pendientes del procedimiento recursivo; teniendo en consideración este hecho se aprecia que mientras mayor sea la entrada N(número de coordenadas), mayor es el tiempo que toma la ejecución puesto que al ejecutarse se van añadiendo llamados a la cola, y al terminar la lógica interna se toma la otra mitad del tiempo en sacar de la cola el primer elemento añadido. Este tecnicismo complicó el análisis del algoritmo pues se tuvo que medir el número de comparaciones teniendo en cuenta el mayor valor de dicho contador almacenado en la cola. Sobre la diferencia entre la complejidad temporal usando estas estructuras se deviene la observación de que; el uso de LinkedList provee un beneficio a la hora de insertar o eliminar elementos mediante iteradores pero implica un retraso al momento de acceder a los elementos pues se deben recorrer los anteriores de forma secuencial mediante la estructura (lista(pointer=siguiente nodo)) es decir con el uso de apuntadores, por lo que se obtiene una complejidad mayor mientras más grande sea la entrada N(teniendo en cuenta que el movimiento consistirá en comparaciones recursivas) respecto al uso de Arrays convencionales donde el movimiento a través de la lista es constante, sin embargo a groso modo la variación fue mínima en los análisis ejecutados debido a la sensibilidad del algoritmo en general