Analysis of Best, Worst, Average cases in an algorithm of Closets Pairs with Linked list (17 Noviembre de 2022) David enrique Calderon Bonilla, Universidad del Norte

Resumen - Mediante la implementación de un algoritmo que tiene como objetivo calcular la distancia entre dos puntos 'X' e 'Y' para encontrar el par de coordenadas más cercano con el fin de establecer las descripciones de estos tres casos y el análisis de cada uno de ellos, además de la importancia de esto y cómo puede afectar el uso de dos estructuras diferentes para los arreglos a implementar.

Abstract – By implementing an algorithm that aims to calculate the distance between two points 'X' and 'Y' to find the nearest pair of coordinates in order to establish the descriptions of these three cases and the analysis of each of them, in addition to the importance of this and how it can affect the use of two different structures for the arrangements to be implemented.

Índice de Términos - Best Cases, Worst Cases, Avrage Cases, point, coordinates

1 INTRODUCCIÓN

En el estudio sistemático de la complejidad computacional de algoritmos geométricos, el algoritmo del par más cercano o "close pairs" fue uno de los primeros problemas que fue tratado en estos estudios el cual consiste dado n puntos en el espacio métrico, encuentra un par de puntos con la distancia más pequeña entre ellos. Esté algoritmo se puede implementar de dos maneras la primera es por fuerza bruta y la segundo utilizando el método de "divide y vencerás", además de implementar también listas enlazadas debido a esto el análisis de estas implementaciones podrían servir para determinar su rendimiento en los tres casos y en evidenciar como se ve afectado el uso de un arraylist o una lista enlazada.

David Enrique Calderon Bonilla

E-mail addresses: edcalderon@uninorte.edu.co

2 PLANTEAMIENTO DEL PROBLEMA

Implemente los algoritmos de fuerza bruta y por el método "Divide y Vencerás" que encuentran el par más cercano en un conjunto de datos de coordenadas (x, y) de seis elementos en Java. Cree el conjunto de datos muestreando números aleatorios uniformemente distribuidos de tipo integral. Para simplificar, no utilice números de punto flotante o flotantes para definir el conjunto de datos de coordenadas. Para esta actividad, puede usar un ArrayList o cualquier otro contenedor adecuado que proporcione un método de clasificación. Sin embargo, le recomendamos que

implemente su propio algoritmo de clasificación si no está seguro de qué contenedor usar y cómo invocar su método de clasificación.

3 ALGORITMO

El algoritmo implementado está estructurado por varios métodos:

1) Método Fuerza Bruta:

Podemos resolver esto de manera ingenua simplemente calculando la distancia entre cada punto y el origen, luego comparándolos, encontrando la distancia más corta y regresando. El problema con este enfoque es que es ineficiente con grandes cantidades de datos porque calculamos cada distancia, lo que toma O(n) tiempo, y hay n elementos que debemos hacer para llegar a donde estamos en $O(n)^2$.

a) Implementación:

```
private static double bruteForce(Point[] p, int start, int end) {
    double min = Double.MAX_VALUE;
    if (p != null && end - start > 0) {
for (int i = start; i < end - 1; i++) {
for (int j = i + 1; j < end; j++) {
                min = Math.min(min, dist(p[i], p[j]));
            }
        }
    }
return min;
  }
```

2) Método Divide y Vencerás:

Podemos reducir la complejidad temporal de nuestro método implementando un algoritmo divide y vencerás para reducir el número de puntos buscados simultáneamente. Esto está en tiempo O(nlogn^2), El procedimiento para este método es el siguiente:

b) Implementación:

```
    public double[] divideAndConquer(LinkedList<Point> list) {
      LinkedList<Point> firstHalf = sub(list,0, list.size()/2-1);
LinkedList<Point> secondHalf = sub(list, list.size()/2, list.size()-1);
double[] first= new double[5];        first = bruteForce(firstHalf);
System.out.println("First Pair");
    System.out.println("(" + first[1] + ", " + first[2] + ") and (" + first[3] + ", " +    first[4] + ")");
     System.out.println("Distance**2: " + first[0]+"\n");
double[] second= new double[5];        second =
bruteForce(secondHalf);
System.out.println("Second Pair");
```

```
        System.out.println("(" + second[1] + ", " + second[2] + ") and (" + second[3] + ",
"
        + second[4] + ")");
        System.out.println("Distance**2: " + second[0]+"\n");
        if (first[0] > second[0]) {
            bruteForce(firstHalf,secondHalf,second);
            return second;
        }           else           if           (first[0]           ==           second[0])           {
bruteForce(firstHalf,secondHalf,second);
            return    second;
} else {
            bruteForce(firstHalf,secondHalf,first);
            return first;
        }
    }
```

Podemos calcular la complejidad temporal del algoritmo de la siguiente manera:

Dividimos los puntos en 2 sus arreglos, luego buscamos el punto más cercano según los límites calculados y lo ponemos en un único arreglo: O(n)
Ordenar la nueva matriz: O(nlogn) Encuentra el punto más cercano: O(n)
como resultado tendremos que la complejidad temporal nos da: O(nlogn^2)

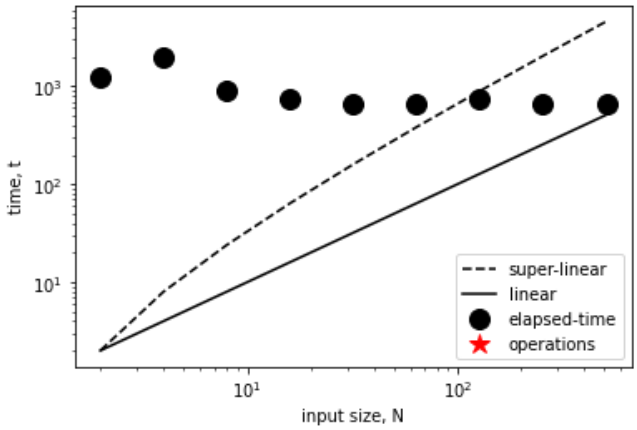
4 RESULTADOS DEL PROBLEMA

Para obtener estos resultados se creó un archivo.txt en el cual guardamos tres valores en forma de tabla mediante un arreglo que son el tamaño es decir la cantidad de puntos generados, el tiempo que se demoró el algoritmo en ejecutarse y las comparaciones realizadas durante todo el algoritmo con las cuales medir su rendimiento y analizar la complejidad, los resultados fueron los siguiente:

Tamaño	Tiempo	Comparaciones
2.00000000e+00	7.00000000e+02	0.00000000e+00
4.00000000e+00	5.00000000e+02	0.00000000e+00
8.00000000e+00	5.00000000e+02	0.00000000e+00
1.60000000e+01	5.00000000e+02	0.00000000e+00
3.20000000e+01	5.00000000e+02	0.00000000e+00
6.40000000e+01	5.00000000e+02	0.00000000e+00
1.28000000e+02	4.00000000e+02	0.00000000e+00
2.56000000e+02	4.00000000e+02	0.00000000e+00

5.12000000e+02	4.00000000e+02	0.00000000e+00
1.02400000e+03	4.00000000e+02	0.00000000e+00

Como se puede observar, el número de operaciones son cero esto puede deberse a un error en el experimento o incluso a un posible suma que genera cero en el arreglo sin embargo sabemos que claramente deben ser lineales debido a que la complejidad es n^2. La grafica obtenida con estos datos es la siguiente:



Ahora bien, la complejidad del algoritmo 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 es de la 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 obviamente O(n2), por lo que parece ser la complejidad de todo el programa. Sin embargo, no es así y esto se evidencia porque los operandos están definidos por:

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

Para resolver la complejidad debemos tener en cuenta que ente caso se tendrá una complejidad O (n^2) en lugar de O (1) como nos fue propuesto en el laboratorio anterior, entonces la ecuación que tenemos será la siguiente:

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

Ahora, con base a este ecuación aplicamos la "Master ecuation" y el resultado sería que las variable a,b,d son iguales a 2 y por tanto al remitirnos a lo que nos dice la master ecuation $a<b^d=2<4$, así debido a esto tenemos que:

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

En conclusión, podemos decir que la mayoría de los objetivos de este laboratorio se cumplieron de manera exitosa al poder observar y analizar el comportamiento de los datos para el problema planteado y se logra evidenciar en los resultados de todos los casos, sea mayor el tamaño así mismo será mayor 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 matemáticamente la razón de esta complejidad cuadrática con lo que nos enseñó en clase y se pudo 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, menciono que fueron la mayoría ya que hubo un contra tiempo con el proceso de generar las gráficas ya que no se pudo lograr que las comparaciones se vieran en el arreglo pero se puedo llevar acabo el desarrollo del laboratorio y por último se evidencio que al compararlo con el anterior laboratorio en el cual implementamos arraylist, estas estructuras tienen sus pro y sus contras en la eficiencia pero ente caso es más eficiente el uso arraylist.

5 REFERENCES

[1] Best, worst and average case - Wikipedia. (s. f.). Pagina principal - Main Page abcdef.wiki. https://es.abcdef.wiki/wiki/Best,_worst and average case (s. f.)

[2] (s. f.). Universitat de València. https://www.uv.es/afuertes/Informatica/Libro/PDFs/CAPI7.pdf