Laboratorio Closest Pair

Alan Daniel Florez Cerro, *Ingenieria de sistemas*

Abstract

El siguiente laboratorio se enfoca en analizar el comportamiento de un algoritmo que compara las distancias de un conjunto de puntos generados aleatoriamente y determina que par de puntos es el más cercano. Se determinara el par más cercano usando dos estrategias (algoritmo de fuerza bruta y algoritmo empleando ”divide and conquer”). Se observará el tiempo de ejecución de cada estrategia en distintos conjuntos de puntos aleatorios, donde la mitad tendrá un x menor a cierto valor y la otra mitad tendrá un x mayor a dicho valor. El algoritmo se implementará en java.

Index Terms

divide and conquer, fuerza bruta. complejidad.

I. Introducción

EL objetivo principal de este informe es determinar la complejidad de ambas estrategias poniéndolas a prueba con distintos casos para así poder elaborar una gráfica con los resultados obtenidos de los tiempos de ejecución que ilustre la complejidad de cada una.

Esto es importante porque permite ver como la estrategia de ”divide and conquer” es capaz de hacer la solución de problemas complejos de tal forma que el algoritmo sea más eficiente a la hora de resolver problemas que conllevan una gran cantidad de operaciones.

mds
Octubre 26, 2022

II. Definición del problema

Cuando codificamos un algoritmo que deba comparar valores en un conjunto de datos usualmente el primer aproximamiento que nos viene a la mente es el de fuerza bruta considerando que es lo más sencillo de implementar y el poder de computo de los equipos actuales les permite hacer muchas operaciones. Sin embargo, si tuviéramos un conjunto de datos de gran tamaño y recursos de computo limitados usar fuerza bruta resultará en fallos o en desperdicio de recursos.

Para poder reducir el uso de recursos incluso cuando el conjunto de datos es grande podemos usar la estrategia de divide y vencerás definida por Programiz(s. f.) como una estrategia que consiste en dividir el problema en problemas más pequeños, resolvelos y de sus resultados obtener el resultado buscado.

III. Metodología

Se tomará el tiempo de ejecución del algoritmo de fuerza bruta y de ”divide y vencerás” en un mismo conjunto de puntos generados aleatoriamente sobre una cantidad de puntos que aumentara en potencias de 2.

El tiempo en nanosegundos se medirá usando la función System.nanotime() de java antes y después de las lineas de código que llaman a cada algoritmo, la resta de ambas permite hallar el tiempo de ejecución del algoritmo.

IV. Resultados

Puntos	Iteraciones	Tiempo (ms)
4	6	622600
8	28	144300
16	120	129900
32	496	233000
64	2016	1754700
128	8128	1498600
256	32640	7091000
512	130816	15373200
1024	523776	15469900
2048	2096128	21580900
4096	8386560	56996600
8192	33550336	459652300
16384	134209536	1083541700
32768	536854528	4046464200
65536	2147450880	22597086800
131072	8589869056	135702551400
262144	34359607296	802027289500

TABLE I
RESULTADOS FUERZA BRUTA

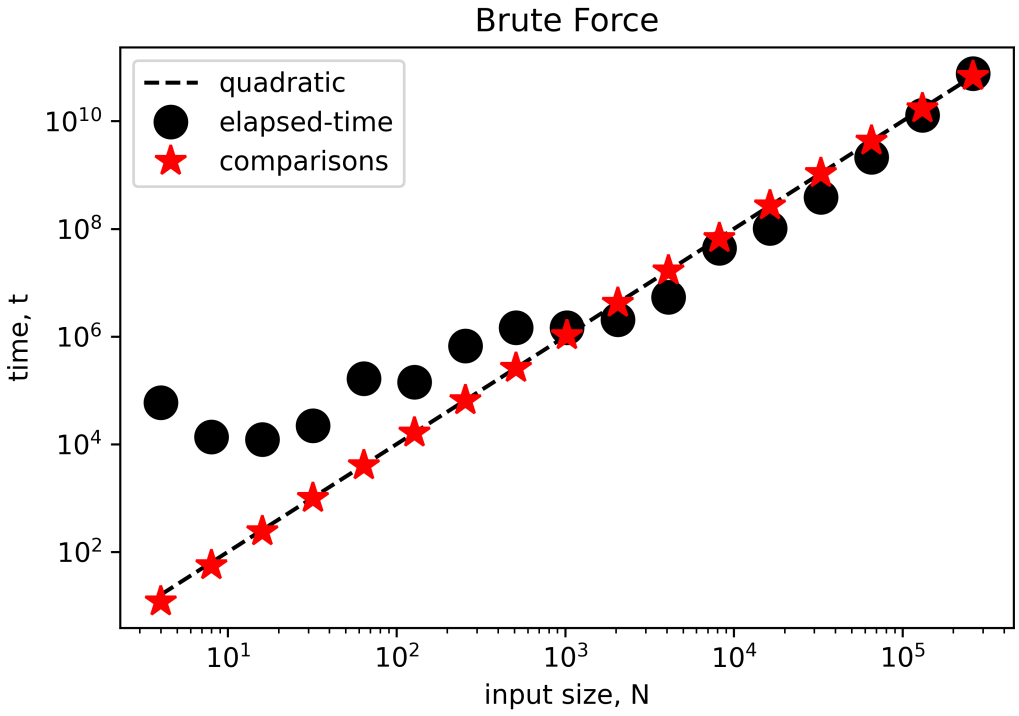

Fig. 1. Resultados algoritmo iterativo

Puntos	Iteraciones	Tiempo (ms)
4	3	10487775
8	16	758975
16	72	661400
32	240	486275
64	992	601150
128	4032	1163550
256	16256	2936900
512	65280	4203275
1024	261632	3493450
2048	1047552	7773700
4096	4192256	25683525
8192	16773120	102718075
16384	67100672	445447950
32768	268419072	1575060025
65536	1073709056	6736732150
131072	4294901760	33748572725
262144	17179738112	270556875925

TABLE II
RESULTADOS DIVIDE AND CONQUER

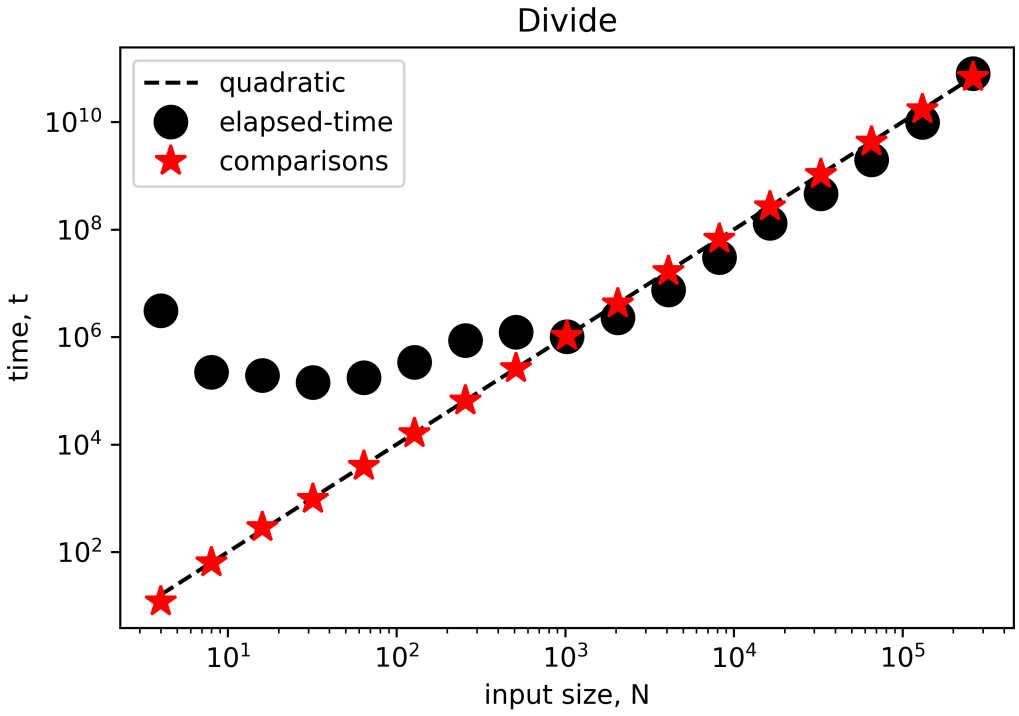

Fig. 2. Resultados algoritmo divide and conquer

V. Discusión

Si bien podemos ver que el número promedio de iteraciones que le tomo al algoritmo usando divide and conquer es menor al numero de iteraciones que requiere usar el algoritmo de fuerza bruta, las gráficas siguen apuntando a que el aproximamiento de divide and conquer implementado sigue manejando una complejidad cuadrática al igual qu el usar el algoritmo de fuerza bruta.

Ciertamente el resultado esperado para esta actividad era que la complejidad del algoritmos que usa divide and conquer fuera lineal.

VI. Conclusión

Reconocimiento

Agradecimientos al profesor Misael por su contribución a la mejora de la implementación del algoritmo para este laboratorio.

References

[1] Programiz. (s. f.). Divide and Conquer Algorithm. Recuperado 26 de octubre de 2022, de https://www.programiz.com/dsa/divide-and-conquer
[2] Diaz Maldonado, M. (8 de Agosto de 2022). loglogPlot.py. Github. Recuperado el 26 de octubre de 2022, de https://github.com/misael-diaz/computer-programming/blob/main/src/io/java/loglogPlot.py
[3] How to increase the Java stack size? (13 de septiembre de 2010). Stack Overflow. Recuperado el 4 de noviembre de 2022, de https://stackoverflow.com/questions/3700459/how-to-increase-the-java-stack-size

Alan Florez Estdudiante de ingenieria de sistemas en la Universidad del norte.