Array Lists & Linked Lists: Closest Pair

David Henriquez, Estudiante UNINORTE

Abstract

Existen diferentes tipos de estructuras de datos que permiten almacenar y actualizar una lista de elementos de diversas maneras, cada una teniendo sus ventajas y desventajas. La elección de una de estas estructuras puede ser un trabajo difícil de realizar pues esta a criterio del programador priorizar la complejidad temporal, el dinamismo del espacio o la facilidad en la comprensión del código, además de que la eficiencia de la estructura depende del problema a solucionar. Con el fin de facilitar el aprendizaje de como tomar esta decisión, realizamos un experimento en el cual solucionamos un problema usando dos tipos de estructuras de datos distintas y comparamos sus resultados, para posteriormente determinar cual de ambas es mas eficiente y por ende la estructura que se debe tomar para el programa.

Index Terms

Estructura de datos, eficiencia algoritmica, complejidad temporal, complejidad espacial, programación.

I. Introducción

LAS estructuras de datos son formas de organizar de manera eficiente una lista de elementos que el usuario necesite y se utilizan para almacenarlos de tal forma que se pueda acceder a ellos y actualizarlos de manera sencilla. Además, se utilizan para realizar diversos procesos con los elementos eficientemente, teniendo al alcance todos los que hacen parte de la estructura. Las estructuras de datos están divididas en estructuras estáticas, como las *Array Lists*, estructuras dinámicas, como las pilas, las colas y las *Linked Lists*, y estructuras no lineales como los árboles y los grafos. Dos de las estructuras mas comunes son las *Array Lists* y las *Linked Lists*. [1]

Las *Array Lists* almacenan elementos en ubicaciones de memoria contiguas, lo que causa que las direcciones de estos elementos sean fácilmente calculadas y esto permite un acceso más rápido a un elemento con un índice especifico, sin embargo, dado que los datos solo se pueden almacenar en bloques contiguos de memoria en una matriz, el tamaño de esta lista no se puede modificar en tiempo de ejecución, pues se corre el riesgo de sobrescribir otros datos. La asignación de memoria ocurre en tiempo de compilación. [2]

Las *Linked Lists* son estructuras de almacenamiento dinámicas y los elementos generalmente no se almacenan en ubicaciones contiguas, por lo que deben ser almacenados con un apuntador que haga referencia al siguiente elemento, por esta razón es que es necesario recorrer todos los elementos previos de la lista hasta tener acceso a un elemento en específico, sin embargo, dado que cada elemento apunta al siguiente, los datos pueden existir en direcciones dispersas, lo que permite un tamaño dinámico que puede cambiar durante la ejecución del programa, es decir que la asignación de memoria ocurre en tiempo de ejecución. [3]

En este experimento vamos a determinar cual de estos dos tipos de estructura de datos es mas eficiente para resolver el problema de encontrar el par de puntos mas cercanos en una lista.

II. Definición del problema

Se debe realizar un programa en java capaz de crear una lista de N nodos con coordenadas (X, Y) usando tanto *Array Lists* como *Linked Lists* y ordenarlos de forma ascendente, para posteriormente encontrar cual es el par de coordenadas más cercanas usando dos técnicas de diseño de algoritmos distintas, la técnica *Divide y vencerás* y la técnica *Fuerza bruta*. Por último, se deben tomar los datos del número de iteraciones y el tiempo de ejecución dado diferentes N para ambas técnicas usando los dos tipos de estructura de datos, y de esta manera poder graficarlos y determinar las diferencias en el comportamiento de ambas técnicas dado el tipo de estructura implementado.

III. Metodologia

Para llevar a cabo este programa, primero diseñamos un algoritmo para realizar la búsqueda del par más cercano por medio de la técnica de *Fuerza bruta* y posteriormente diseñamos un algoritmo recursivo que usa la técnica de *Divide y vencerás* que realiza la misma función que el método anterior.

Algorithm 1 FuerzaBruta

```
for i = 1, Lista.size - 1 do
  for j = 1, Lista.size - 1 do
    NodoA = Lista.get(i)
    NodoB = Lista.get(j)
    distX = NodoB.x - NodoA.x
    distY = NodoB.y - NodoA.y
    dist = distX * distX + distY * distY
    if dist < DistanciaMinima then
      PrimerNodo = NodoA
      SegundoNodo = NodoB
      DistanciaMinima = dist
    end if
  end for
end for
```

Algorithm 2 DivideYVenceras

```
if Lista.size <= 3 then
  FuerzaBruta(Lista)
else
  DivideYVenceras(Primera mitad de la lista)
  DivideYVenceras(Segunda mitad de la lista)
  FuerzaBruta(Medio de la lista)
end if
```

Posteriormente implementamos estos algoritmos en un programa en java usando tanto *Array Lists* como *Linked Lists* y además le agregamos la funcionalidad de contar el número de iteraciones que le toma a cada uno de los métodos realizar su función y el tiempo de ejecución requerido para este. Para la técnica de *Divide y vencerás* usando ambos tipos de estructura de datos, el tiempo de ejecución y las iteraciones son calculados a partir del promedio de 6 ejecuciones con el mismo valor de N, pues los resultados son variables dependiendo de los datos y es adecuado realizar este promedio para que los resultados sean más precisos.

Por último, con todas las funcionalidades ya implementadas, realizamos pruebas con diferentes puntos aleatorios para cada uno de los cuatro métodos, guardamos los resultados y repetimos el proceso aumentando el tamaño N para visualizar el comportamiento de estos métodos cuando N tiende a infinito.

IV. Resultados

Despues de procesar los resultados del experimento en un script de Python obtuvimos las siguientes graficas:

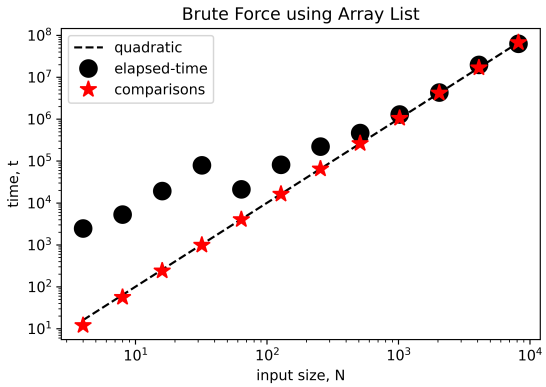

Fig. 1: Comportamiento de la tecnica de *Fuerza Bruta* usando *Array Lists*

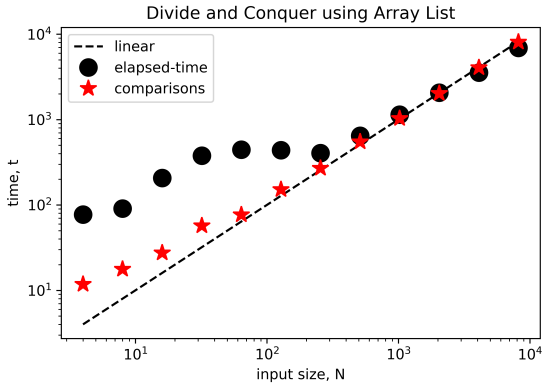

Fig. 2: Comportamiento de la tecnica de *Divide y Venceras* usando *Array Lists*

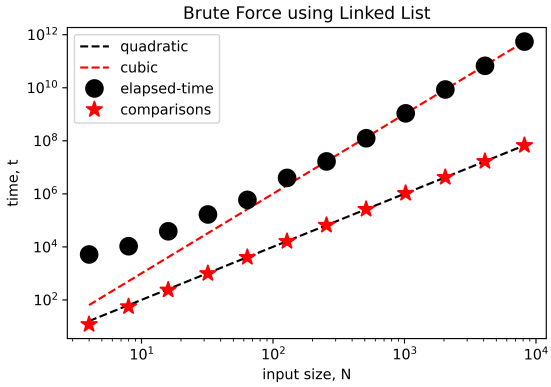

Fig. 3: Comportamiento de la tecnica de *Fuerza Bruta* usando *Linked Lists*

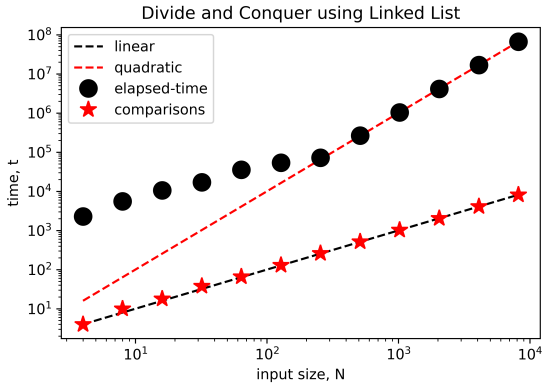

Fig. 4: Comportamiento de la tecnica de *Divide y Venceras* usando *Linked Lists*

Observando las gráficas podemos notar que usando la técnica de *Divide y vencerás* el número de iteraciones crece de manera lineal $O(N)$ tanto para el método que usa *Array Lists* como para el que usa *Linked Lists*, aunque no de manera totalmente precisa, pues el número de iteraciones depende de la lista de coordenadas, por lo que los datos no están alineados en su totalidad al ajuste lineal, pero se comportan de manera parecida. Por otro lado, sin importar el tipo de estructura de datos que se use, el número de iteraciones de la técnica de *Fuerza bruta* crece de manera exponencial $O(N^2)$, y sin importar los datos del arreglo el número de iteraciones dado un mismo N será el mismo y se ajusta de manera perfecta al ajuste exponencial. La diferencia está en que usando la técnica de *Divide y vencerás*, el programa no compara cada par de puntos en la lista, sino que compara solo los que están más cercanos, lo que ahorra una gran cantidad de comparaciones y permite que el número de iteraciones aumenten de manera parecida a N, es decir de manera lineal. Mientras que usando la técnica de *Fuerza bruta*, sin importar que las coordenadas estén muy lejanas se calcula la distancia entre ellas, lo que causa que se tengan que realizar

muchas más comparaciones de las necesarias, y provocan que el número de iteraciones aumenten de manera exponencial a medida que aumente N.

Sin embargo, a pesar de que notamos que el crecimiento en el número de iteraciones dada la misma técnica de desarrollo de algoritmos se comporta de manera similar sin importar el tipo de estructura de datos, lo que si varia de manera significativa es la complejidad temporal, pues notamos que los tiempos de ejecución para los métodos que usaron *Array Lists* fueron considerablemente menores que los tiempos de ejecución de los métodos que usaron *Linked Lists*. Pues la complejidad temporal cuando se aplica la técnica de *Divide y vencerás* usando *Array Lists* es lineal $O(N)$, mientras que cuando se usan *Linked Lists* la complejidad es cuadrática $O(N^2)$. Y cuando se aplica la técnica de *Fuerza bruta* usando *Array Lists* la complejidad temporal es cuadrática $O(N^2)$, mientras que cuando se usan *Linked Lists* la complejidad es cubica $O(N^3)$. Esto es debido a que se puede acceder a los elementos de las *Array Lists* directamente usando su *index*, mientras que para las *Linked Lists* es necesario recorrer todos los elementos previos para acceder al elemento que se necesita. Por esta razón es que los métodos que usan *Array Lists* son más rápidos que los que usan *Linked Lists*.

V. Conclusiones

Con la realización de este experimento pudimos determinar la importancia de considerar la implementación de diferentes tipos de estructuras de datos para nuestros programas, con el fin de escoger el que tenga mejor complejidad espacial y temporal, es decir que tenga una mayor eficiencia algorítmica, pues notamos en los resultados del experimento que, a pesar de que cuando se usa una misma técnica para el diseño de algoritmos el número de iteraciones es similar sin importar el tipo de estructura que se haya utilizado, el tiempo de ejecución si varía mucho entre ellas, lo que nos permite elegir el tipo de estructura que mejor nos convenga.

Además, pudimos observar que, cuando el número de elementos crece considerablemente, los programas que usan *Array Lists* son más eficientes temporalmente que los programas que usan *Linked Lists*, pues notamos que las *Array Lists* pueden acceder a un elemento especifico de la lista directamente con su índice, mientras que las *Linked Lists* necesitan recorrer todos los elementos previos para acceder al dato que se necesita. Esto lo pudimos comprobar en las graficas realizadas a partir de los resultados del experimento, pues notamos que los métodos que usaron *Array Lists* tuvieron una complejidad temporal de $O(N)$ y $O(N^2)$, mientras que los métodos que usaron *Linked Lists* tuvieron una complejidad temporal de $O(N^2)$ y $O(N^3)$.

Por último, a partir de los resultados del experimento podemos concluir que los programas que usan la estructura estatica *Array List* tienen una menor complejidad temporal, y por ende una mayor eficiencia algorítmica en la solución del problema de encontrar el par de puntos mas cercanos en una lista, que los programas que usan la estructura dinamica *Linked List*.

References

[1] G. for Geeks, "Data structures." [Online]. Available: https://www.geeksforgeeks.org/data-structures/
[2] ——, "Linked list vs array." [Online]. Available: https://www.geeksforgeeks.org/linked-list-vs-array/
[3] N. Parlante, "Linked list basics," *Stanford CS Education Library*, vol. 1, p. 25, 2001.