EDAA: Boletín 1

Estudiante: Álvaro Guzmán
Asignatura: Estructura de Datos y Algoritmos Avanzados

1 Resumen

Los algoritmos de búsqueda son cumplen un rol fundamental en informática: encontrar un elemento dentro de una colección es el pilar de consultas en bases de datos. Cuando la colección está **ordenada**, esto se puede explotar para reducir drásticamente el coste de localizar un elemento.

En este informe se analizan experimentalmente tres algoritmos de búsqueda en arreglos ordenados: la **búsqueda secuencial**, la **búsqueda binaria** y la **búsqueda exponencial**. La búsqueda secuencial recorre los elementos del arreglo uno a uno, resultando en una complejidad temporal $O(n)$ en el peor caso), y puede ser muy eficiente si el elemento está cerca del inicio o si el tamaño es pequeño, además, puede aplicarse incluso si el arreglo no está ordenado. La búsqueda binaria divide el espacio de búsqueda por la mitad en cada paso y tiene coste $O(\log n)$, la forma en que se realizan las divisiones asumen que el arreglo está ordenado y que se puede acceder a posiciones intermedias del arreglo en tiempo constante. La búsqueda exponencial (o galopante) busca el intervalo $[2^i, 2^{i+1}]$ de posiciones en el arreglo en el que se encuentra el elemento buscado, para ello avanza exponencialmente hasta encontrar el límite correcto. Finalmente aplica búsqueda binaria en el intervalo encontrado, es por esta razón que su complejidad temporal en peor caso sigue siendo de $O(\log n)$, tiene la ventaja de no necesitar conocer el tamaño del arreglo de antemano (no así búsqueda binaria).

El objetivo de este boletín es evaluar experimantalmente cómo se comportan estas tres estrategias en dos cosas: al variar el tamaño del arreglo y al variar la posición del elemento buscado. Para ello se implementaran los algoritmos correspondientes y se medirá el tiempo de ejecución de cada caso, para finalmente presentar y analizar los resultados.

En las secciones siguientes se discute el marco teórico, las decisiones de implementación, se presentan y analizan los resultados, y finalmente se destacan las conclusiones.

2 Análisis teórico

El problema a enfrentar corresponde a buscar un elemento dentro de un arreglo $A[1, ..., n]$ ordenado de $n \in \mathbb{N}$ elementos. En particular, se estudairán tres búsquedas: secuencial, binaria y exponencial.

2.1 Busqueda secuencial

La búsqueda secuencial es la forma *naive* de enfrentar este problema. Simplemente se revisa elemento a elemento hasta encontrar el buscado (Ver Algoritmo 1). En el peor de los casos, cuando el elemento no se encuentra, se deben consultar los n elementos de A. Esto corresponde a una complejidad temporal de $O(n)$ en peor caso. El único caso en que funciona en poco tiempo es cuando el elemento está al inicio del arreglo, sin embargo esto no logra compensar el tiempo de los casos cuando el elemento a encontrar está al final.

Algoritmo 1 Búsqueda secuencial

Entrada: Arreglo A ordenado de tamaño n y elemento e a buscar.

```
1: for i = 1, ..., n do
2:     if e < A[i] then          ▷ Si los elementos que quedan son mayores a e, retornamos que no está.
3:         return −1.
4:     end if
5:     if A[i] = e then
6:         return i.
7:     end if
8: end for
9: return −1.
```

Esta estrategia no requiere que los datos estén ordenados, sin embargo, su rendimiento se degrada rápidamente cuando el tamaño del arreglo crece, por lo que no es escalable.

2.2 Busqueda binaria

La búsqueda binaria aprovecha el hecho de que los elementos están ordenados. El algoritmo compara el valor buscado con el elemento en la posición central, descartando la mitad del arreglo en la que el valor no puede estar. Este proceso se repite sobre la mitad restante hasta encontrar el elemento o no queden más mitades válidas (intervalos incoherentes) que revisar (Ver Algoritmo 2). En el peor caso se deben revisar todas las posiciones asociadas a cada mitad de cada iteración, resultando en una complejidad temporal $O(\log n)$.

Algoritmo 2 Búsqueda binaria

Entrada: Arreglo A ordenado de tamaño n y elemento e a buscar.

```
1:  start = 1
2:  end = n
3:  while start ≤ end do
4:      mid = (start + end)/2
5:      if A[mid] == e then
6:          return mid
7:      end if
8:      if A[mid] < e then
9:          start = mid + 1
10:     else
11:         end = mid − 1
12:     end if
13: end while
14: return −1.
```

Esta estrategia requiere que los datos estén ordenados y que se pueda acceder a posiciones intermedias del arreglo en tiempo constante. A cambio, mejora considerablemente el comportamiento asintótico temporal, de lineal a logarítmico.

2.3 Busqueda exponencial

La búsqueda exponencial, también llamada búsqueda galopante, consiste en dos fases. En la primera fase, se duplica de forma exponencial el índice de búsqueda hasta que se encuentra un valor mayor o igual al buscado, determinando así un intervalo acotado de la forma $A[2^i, ..., 2^{i+1}]$ (el extremo superior puede ser n si es que el intervalo acotado está en la última sección del arreglo) donde se debe encontrar el elemento. En la segunda fase,se aplica una búsqueda binaria dentro de intervalo acotado determinado en la primera fase. El Algoritmo 3 detalla este procedimiento.

Algoritmo 3 Búsqueda exponencial

Entrada: Arreglo A ordenado de tamaño n y elemento e a buscar.

```
1:  if A[1] then
2:  end if
3:  index = 1
4:  while index ≤ n do
5:      if e < A[index] then
6:          break
7:      end if
8:      index = 2 ∗ index
9:  end while
10: return −1.
```

Los requerimientos y ventajas de esta estrategia son los mismos que los de la búsqueda binaria, con la ventaja de funcionar más rápido cuando el elemento está cerca del inicio y que puede aplicarse sin necesidad de conocer el tamaño del arreglo de antemano.

Cabe recalcar que para el análisis teórico se utilizó indexación desde 1 para referirse a las posiciones en el arreglo. Además, por simplicidad, se utilizó la convención de que los elementos son no negativos, por lo que los algoritmos retornan -1 para indicad que elemento no existe, esta convención se mantuvo para lo que es la implementación, esto podría evitarse agregando una salida con error cuando el elemento no existe.

3 Implementaciones

Todas la implementaciones se encuentran adjuntas junto al informe en un comprimido .zip. Se utilizó como base el helper *uhr* del repositorio del curso (`https://github.com/jfuentess/edaa/tree/main/experimentos`), adaptado para correr los distintos experimentos. En contraste con el análisis teórico, desde ahora se considerará indexación desde 0 para el arreglo. Los códigos se encuentran en carpetas según el experimento que se trabaja (*tamaño* o *posición*), en cada caso un archivo *run.sh* indica la configuración utilizada para correr los experimentos.

4 Resultados experimentales

Se utilizó el servidor **chome** de la Universidad de Concepción para correr los experimentos, el cual tiene las siguientes caraterísticas:

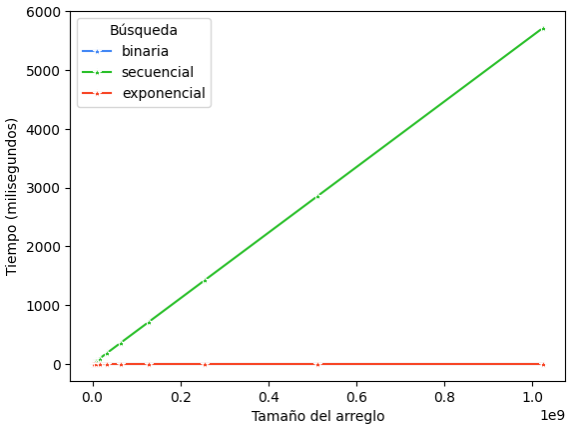
1. **Procesador:** Intel(R) Xeon(R) Gold 5320T CPU 2.30GHz.
2. **Memoria RAM:** 251 GB.

Se utilizó g++ para compilar los programas con las opciones `-std=c++11` y `-O0`, tal como es indicado en el repositorio del curso. Los resultados fueron gráficados utilizando Python 3.12.3.

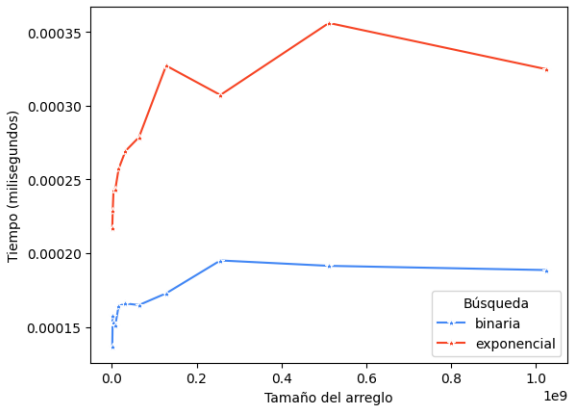
Considerarémos un arreglo $A[0, ..., n-1]$ de tamaño $n \in \mathbb{N}$ y un elemento e (que puede estar o no en A). A continuación se presentan los resultados obtenidos al utilizar los tres algoritmos de búsqueda en cada caso.

4.1 Variando el tamaño del arreglo

En este primer experimento se considerarán arreglos A ordenados de tamaño $n = 2^k \cdot 10^6$, con $k \in \{1, 2, ..., 10\}$, donde los elementos de A serán $A[i] = i$. En cada caso, se ejecutará la búsqueda de un elemento $e > n$ que no esté en el arreglo, así estarémos considerando el peor caso de búsqueda. Además, cada experimento se ejecutará 256 veces para tener un promedio lo más estable posible.



(a) Tiempo de búsqueda de los tres algoritmos, en milisegundos.



(b) Tiempo de búsqueda para búsqueda binaria y exponencial, en milisegundos.

Figura 1: Tiempos de ejecución, en milisegundos, de los experimentos de búsqueda variando el tamaño n del arreglo. Buscando un elemento mayor a todos los de A (peor caso para todos los algoritmos

En la Figura 1 se presenta el resultado de las mediciones de tiempo, en milisegundos, al variar el tamaño del arreglo A. Los algoritmos de búsqueda secuencial, binaria y exponencial están graficados con color verde, azúl y rojo, respectivamente. En la Figura 1a se presentan los resultados para los tres algoritmos, donde destaca el algoritmo de búsqueda secuencial por su comportamiento lineal además de tomar considerablemente más tiempo en ejecutarse. Para observar mejor el comportamiento de los otros dos algoritmos se realizó un segundo gráfico,

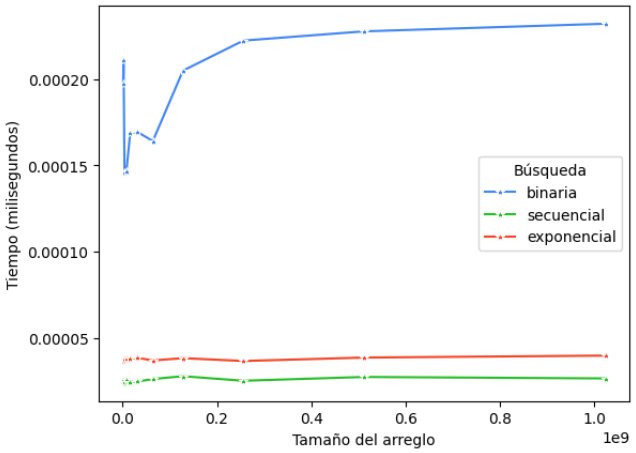


Figura 2: Tiempo de búsqueda para búsqueda binaria y exponencial, en milisegundos. Buscando un elemento menor a todos los del arreglo A.

presentado en la Figura 1b, donde vemos que ambos algoritmos presentan un comportamiento similar. La forma de la gráfica para ambos algoritmos respalda la teória, el comportamiento es logarítmico.

También vemos que la búsqueda binaria tarda menos que la búsqueda exponencial. Esto se debe al diseño del experimento: dado que el elemento no se encuentra en el arreglo (es mayor a todos los elementos de este), la búsqueda binaria comienza de inmediato a realizarse en la segunda mitad del arreglo, mientras que la búsqueda exponencial recorrerá todos sus posibles intervalos para finalmente realizar búsqueda binaria en el último, el es exactamente igual a la segunda mitad del arreglo (pues los tamaños de los arreglos son potencia de 2). De esta forma, ambos algoritmos realizan búsqueda binaria sobre la segunda mitad del arreglo, pero la búsqueda exponencial realiza pasos adicionales para llegar ahí, lo cual se ve reflejado en el desplazamiento vertical de la gráfica.

Al repetir el experimento, pero considerando un elemento que no esté en el arreglo y sea menor a todos los elementos de esto (en específico, -1), se debería observar un comportamiento muy diferente. En la Figura 2 podemos ver que efectivamente si el elemento está hacia la izquierda (menor a todos los elementos de A), la búsqueda secuencial y la exponencial detectan inmadiatamente que el elemento no está, por lo que se de tienen rápidamente independiente del tamaño del arreglo. Por otro lado, la búsqueda binaria está obligada a realizar todas sus iteraciones, comportándose como en su peor caso: logarítmico.

4.2 Variando la posición del elemento en el arreglo

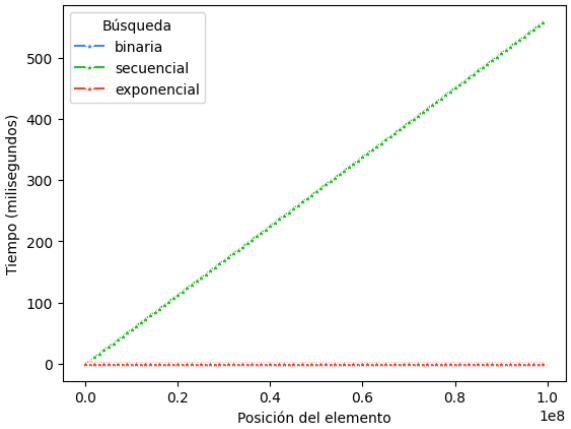
En este experimento se considerará un arreglo A ordenados de tamaño $n = 1 \cdot 10^8$, donde los elementos de A serán $A[i] = i$. Se utilizarán los tres algoritmo de búsqueda para buscar elementos en las posiciones $i = 1 + k \cdot 10^6$, con $k \in \{0, 1, ..., 99\}$, con el fin de ver el impacto de la posición del elemento en el tiempo de ejecución. Se utilizarán posiciones equiespaciadas para abarcar de uniformemente el dominio, y así no sesgar los experiemtnos para aquellos algoritmos que dividen o multiplican en su búsqueda. Cada experimento se ejecutará 64 veces para tener un promedio y eliminar, en lo posible, la variabilidad.

En la Figura 3 se reportan los gráficos para este experimento. Los algoritmos de búsqueda secuencial, binaria y exponencial están graficados con color verde, azúl y rojo, respectivamente. En la Figura 3a se presentan los resultados para los tres algoritmos, donde, al igual que en el experimento anterior, destaca el algoritmo de búsqueda secuencial por su comportamiento lineal y el mayor tiempo de ejecución en comparación a los otros dos algoritmos.

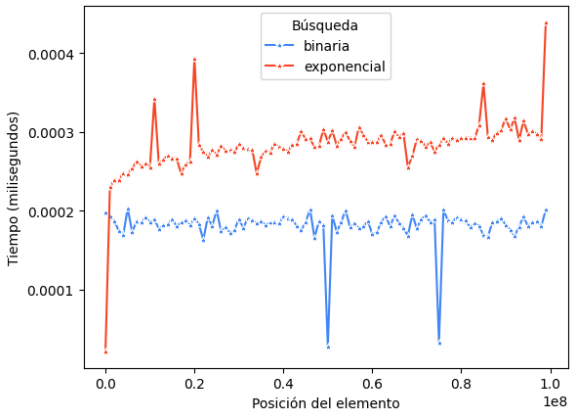
De igual forma que en el experimento antrior, se presenta en la Figura 1b el comportamiento de los algoritmos de búsqueda binaria y exponencial, donde vemos que ambos algoritmos presentan un comportamiento similar. Sólo que la forma ahora no logarítmica. La búsqueda binaria parece mostrar un comportamiento constante, independiente de la posición, excepto en el caso evidente de cuando el elemento se encuentra justo al medio (primera iteración). Por otro lado, la búsqueda exponencial muestra un comportamiento similar, pero es considerablemente más rápida encontrando el elemento cuando está al inicio, aunque también es más lenta cuando está al final.

5 Conclusiones

Los resultados experimentales respladan el comportamiento asintótico de peor caso para los algoritmos de búsqueda secuencial, búsqueda binaria y búsqueda exponencial. En general, la búsqueda binaria es la que menor tiempo toma,



(a) Tiempo de búsqueda de los tres algoritmos, en milisegundos.



(b) Tiempo de búsqueda para búsqueda binaria y exponencial, en milisegundos.

Figura 3: Tiempos de ejecución, en milisegundos, de los experimentos de búsqueda variando la posición i del elemento a buscar en un arreglo de tamaño 2^8.

aunque existen casos donde la secuencial o la exponencial es mejor, en particular cuando el elemento a buscar está al inicio del arreglo. Considerando el peor caso de búsqueda cuando el arreglo es de tamaño 10^7 (el mayor tamaño del primer experimento), la búsqueda binaria es $6.7 \cdot 10^6$ veces más rápida que la búsqueda secuencial, por otro lado, la búsqueda exponencial es $1.9 \cdot 10^6$ veces más rápida que la búsqueda secuencial. Quedando en evidencia la gran ventaja de explotar el ordenamiento del arreglo para realizar la búsqueda.

La ventaja que puede tener la búsqueda exponencial sobre la búsqueda binaria es su rapidez para encontrar elementos al inicio del arreglo (aunque también tarda más al final del arreglo), o incluso para detectar que elementos pequeños (relativos a los valores del arreglo) no están en el arreglo, tal como reveló la Figura 2. Además de no necesitar conocer el tamaño completo del arreglo para comenzar con sus iteraciones, es posible adaptarlo para que expanda los índices de busqueda hasta llegar al último elemento.

Los algoritmos de búsqueda binaria y exponencial son capáces de encontrar rápidamente elementos en arreglos ordenados de tamaños considerables ($\approx 4 \cdot 10^8 bytes$), a diferencia del algoritmo secuencial. Los experiemntos permiten evidenciar la considerable defierencia entre tiempos lineales en comparación a tiempos logarítmicos, y cómo estos escalan.

6 Referencias

La implementación de los algoritmos de búsqueda es de autoría propia, los datasets fueron generados como se describió en el Boletín, además de haber incluído el enlace a las referencias correspondientes.