Análisis de Algoritmos
Tarea 5: Búsquedas.

Profesora: María de Luz Gasca Soto
Ayudantes: Rodrigo Fernando Velázquez Cruz
Teresa Becerril Torres
Nombre: Alvaro Ramirez Lopez
N°. de Cuenta.: 316276355
Correo: alvaro@ciencias.unam.mx

1. Problema 1

Consideremos el siguiente juego, entre dos personas: El jugador J_A piensa un número entero en un rango. El jugador J_B intenta encontrar tal número haciendo preguntas de la forma:

¿Es el número menor que x? o ¿Es mayor que y?

El objetivo es realizar el menor número de preguntas. Se supone que nadie hace trampa.

a) Diseñar una buena estrategia para el juego... cuando el jugador J_A indica un rango especifico, digamos de 1 a N. En este caso, ¿Cual resulta ser la complejidad del algoritmo? **Justificar**.
b) Diseñar una buena estrategia para el juego... cuando el jugador J_A no indica el rango del número que pensó. ¿Cual es la complejidad del algoritmo propuesto? **Justificar**.

Solución:

a) Diseñar una buena estrategia para el juego cuando el jugador J_A indica un rango especifico, digamos de 1 a N. En este caso, ¿Cual resulta ser la complejidad del algoritmo?.

Una buena estrategia seria la siguiente, dado a que conocemos el valor de N, donde N es el rango de números que el jugador J_A puede pensar, podemos sacar la mitad de N y en el primer turno podemos preguntar si el numero que pensó es mayor o menor al la mitad de N que previamente calculamos.

Si el jugador J_A responde que el numero que pensó no es ni mayor ni menor a la mitad de N, entonces podemos determinar que N es el numero que pensó, ya que es el único numero que queda y no existe la posibilidad de que el jugador J_A haya pensado otro numero que no este en el rango de 1 a N.

Si el jugador J_A responde que el numero que pensó es mayor a la mitad de N, entonces podemos descartar la mitad de los números que están a la izquierda de la mitad de N y repetimos el proceso de arriba, pero ahora con la mitad de los números que quedaron.

Si el jugador J_A responde que el numero que pensó es menor a la mitad de N, entonces podemos descartar la mitad de los números que están a la derecha de la mitad de N y repetimos el proceso de arriba, pero ahora con la mitad de los números que quedaron.

La complejidad del algoritmo propuesto es de $O(\log n)$ en el peor de los casos, ya que en cada iteración, el rango de búsqueda se reduce a la mitad, lo que proporciona una convergencia rápida hacia el elemento buscado.

b) Diseñar una buena estrategia para el juego cuando el jugador J_A no indica el rango del número que pensó. ¿Cual es la complejidad del algoritmo propuesto?

Para resolver este problema, vamos a inspirarnos en 2 algoritmos de búsqueda, la búsqueda binaria y la búsqueda exponencial. Primero vamos a usar la búsqueda exponencial para acotar en el rango en el que se encuentra el numero que pensó el jugador J_A y después vamos a usar la búsqueda binaria para encontrar el numero exacto que pensó el jugador J_A.

1. **Establecer un límite inferior**: Comienza con un límite inferior (por ejemplo, 1) y un límite superior (por ejemplo, 2). Inicialmente, el límite superior es menor que el número a adivinar.
2. **Duplicar el límite superior**: Incrementa el límite superior multiplicándolo por 2 en cada iteración hasta que se alcance un límite superior que sea mayor o igual al número objetivo.
3. **Realizar preguntas**: Para cada iteración, pregunta si el número objetivo está en el rango entre el límite inferior y el límite superior. Dependiendo de la respuesta, actualiza los límites para restringir la búsqueda.
   - Si el número objetivo es menor que el límite superior actual, aplica búsqueda binaria en el rango entre el límite inferior y el límite superior para encontrar el número.
   - Si el número objetivo es mayor o igual que el límite superior actual, duplica nuevamente el límite superior y continúa.

Este enfoque combina la búsqueda exponencial para encontrar un rango aproximado y luego aplica la búsqueda binaria dentro de ese rango para adivinar el número exacto.

La complejidad de este algoritmo está dominada por la fase de búsqueda binaria dentro del rango encontrado por la búsqueda exponencial. La búsqueda exponencial tiene una complejidad de $O(\log n)$, donde n es la distancia entre el límite inferior y el límite superior en la última iteración de la búsqueda exponencial. Luego, la búsqueda binaria en este rango tiene una complejidad adicional de $O(\log n)$ para adivinar el número exacto. En conjunto, la complejidad total es $O(\log n) + O(\log n) = O(\log n)$.

2. Problema 2

Dada un arreglo de n enteros $A[0, ..., n-1]$, tal que $\forall i, 0 \leq i \leq n$, se tiene que $|A[i] - A[i+1]| \leq 1$; si $A[0] = x$ y $A[n-1] = y$, se tiene que $x < y$.

Diseña un algoritmo de búsqueda eficiente, de orden logarítmico, que localice al indice j tal que $a[j] = z$, para un valor dado de z, $x \leq z \leq y$. **Justificar.**

Solución:

Lo primero que se debe de realizar, es verificar la forma que están distribuidos los elementos en el arreglo A, basándonos en la condición $|A[i] - A[i+1]| \leq 1$; si $A[0] = x$ y $A[n-1] = y$, se tiene que $x < y$, podemos ver que el arreglo esta ordenado de forma monótona y creciente, por lo que podemos continuar con el algoritmo a diseñar.

A continuación, se describe el algoritmo de búsqueda que se pensó para resolver el problema:

```
Inicializar dos índices, izquierda = 0 y derecha = n - 1, y una variable medio = 0.

Mientras (izquierda <= derecha):
  Calcular el índice medio como: medio = (izquierda + derecha) / 2.

   Si A[medio] es igual a z,entonces devolver medio ya que hemos encontrado el elemento buscado.

   Si A[medio] es menor que z, actualizar izquierda = medio + 1 para buscar en la mitad derecho
```

```
del arreglo.

    Si A[medio] es mayor que z, actualizar derecha = medio - 1 para buscar en la mitad izquierda
del arreglo.
```

El algoritmo de búsqueda logarítmica propuesto tiene una complejidad de $O(\log n)$ en el peor caso. Esto se debe a que en cada iteración, el rango de búsqueda se reduce a la mitad, lo que proporciona una convergencia rápida hacia el elemento buscado.

3. Problema 3

Se tiene un conjunto de N rocas, todas ellas de diferente tamaño, forma y consistencia. Rocas que se ven del mismo tamaño pueden tener peso muy diferente.

Un Experto desea convencer a un Jurado que una roca especial, de entre las N rocas dadas, es la de menor peso. El Experto si sabe los pesos de cada una de las rocas y el Jurado no. El Experto quiere mostrar que la roca especial es la de menor peso usando una balanza menos de $(N-1)$ veces. ¿Eso es posible? **Justifica.**

Solución:

Sí, es posible demostrar que una roca especial es la de menor peso utilizando menos de $(N-1)$ veces una balanza.

Para lograrlo, se puede utilizar una estrategia conocida como "división binaria" o "división en grupos", a continuación se describe esta estrategia:

a) **Paso 1: Divide las rocas en dos grupos aproximadamente iguales.** Divide las N rocas en dos grupos de aproximadamente N/2 rocas cada uno. Es fundamental que estos grupos estén lo más equilibrados posible en términos de cantidad de rocas y, por lo tanto, de peso total.

b) **Paso 2: Pesa los dos grupos.** Utiliza la balanza para comparar el peso del primer grupo de rocas con el peso del segundo grupo de rocas.

- Si la balanza está equilibrada, entonces la roca especial está en el grupo que no se pesó y es la de menor peso. Pasamos al paso 3.
- Si la balanza no está equilibrada, la roca especial está en el grupo más ligero. Pasamos al paso 3.

c) **Paso 3: Divide nuevamente el grupo de rocas más ligero.** Toma el grupo de rocas en el que se encuentra la roca especial y repite el proceso de dividirlo en dos grupos de aproximadamente igual tamaño.

d) **Paso 4: Pesa uno de los nuevos grupos con el grupo ya identificado como más ligero.** Utiliza la balanza para comparar el peso del grupo recién dividido con el grupo que previamente identificaste como más ligero.

- Si la balanza está equilibrada, entonces la roca especial está en el grupo que no se pesó y es la de menor peso. Hemos demostrado que la roca especial es la más ligera utilizando 2 pesadas en total.
- Si la balanza no está equilibrada, la roca especial está en el grupo más ligero. Hemos demostrado que la roca especial es la más ligera utilizando 2 pesadas en total.

En resumen, utilizando esta estrategia de división binaria, puedes demostrar que una roca especial es la de menor peso usando como máximo 2 pesadas en total, independientemente del número de rocas N en el conjunto inicial.

4. Problema 4

Consideremos el Algoritmo de Búsqueda por Interpolación.

a) Presentar un ejemplo de al menos 400 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en pocas iteraciones.

b) Dar un ejemplo de, al menos, 400 datos para el cual el algoritmo termine la búsqueda (exitosa o no) en muchas iteraciones.

Solución:

El algoritmo de búsqueda por interpolación es utilizado para buscar un elemento en una lista ordenada aplicando una estimación basada en la distribución de los datos. La eficiencia de este algoritmo depende de la distribución de los datos y la posición del elemento buscado en la lista.

a) Ejemplo con pocas iteraciones:

Supongamos que tenemos una lista de 400 datos en la que la mayoría de los elementos están agrupados al principio y al final de la lista, y solo unos pocos están en el medio. Vamos a crear una lista siguiendo este patrón:

```
1, 2, 3, ..., 150, 151, 152, ..., 250, 251, 252, ..., 350, 351, 352, ..., 399, 400
```

Para buscar el número 100 en esta lista, el algoritmo de búsqueda por interpolación realizará pocas iteraciones ya que puede estimar rápidamente que el número 100 se encuentra en el rango 1-150.

b) Ejemplo con muchas iteraciones:

Supongamos que tenemos una lista de 400 datos en la que están distribuidos de manera uniforme. Vamos a crear una lista de esta forma:

```
1, 2, 3, 4, ..., 397, 398, 399, 400
```

Para buscar el número 100 en esta lista, el algoritmo de búsqueda por interpolación realizará muchas iteraciones ya que la estimación inicial será menos precisa debido a la distribución uniforme de los datos.

En ambos casos, el resultado de la búsqueda (exitosa o no) dependerá de si el elemento que estamos buscando está presente en la lista o no.

5. Problema Opcional

Supongamos que se tiene un programa que manipula textos muy grandes, como un procesador de palabras. El programa toma como entrada un texto, representado como una secuencia de caracteres y produce alguna salida.

Si en algún momento el programa encuentra un error del cual no puede recuperarse y ademas no puede indicar que error es o donde esta, entonces la única acción que el programa toma es escribir ERROR y abortar el proceso.

Supongamos que el error es local, esto es, se tiene una cadena en particular del texto la cual, por alguna extraña razón, al programa no le gusta. El error es independiente del contexto en el cual aparece la cadena “ofensiva”. Diseñar una estrategia logarítmica para localizar la fuente del error.

Solución:

Para localizar la fuente del error en un texto representado como una secuencia de caracteres de manera logarítmica, podemos aplicar un enfoque similar a la búsqueda binaria. Este enfoque implica dividir iterativamente el texto en mitades y buscar la cadena ofensiva en cada mitad, reduciendo así el espacio de búsqueda.

Aquí está la estrategia detallada:

1. **Inicializacion**:

- Definimos un rango inicial para la búsqueda que abarca todo el texto.
- Inicialmente, el límite inferior es 0 y el límite superior es la longitud total del texto.

2. **Búsqueda binaria para localizar la cadena ofensiva**:
   - En cada iteración, calculamos el punto medio del rango actual.
   - Buscamos la cadena ofensiva en la mitad del texto definida por el rango actual.
   - Si encontramos la cadena ofensiva en la mitad actual, la fuente del error está en esa mitad. Terminamos el proceso y reportamos la mitad actual como la fuente del error.
   - Si la cadena ofensiva está en la mitad inferior, actualizamos el límite superior al punto medio actual.
   - Si la cadena ofensiva está en la mitad superior, actualizamos el límite inferior al punto medio actual.
3. **Repetición de la búsqueda binaria**:
   - Repetimos el paso 2 hasta que el rango se reduzca a un solo carácter (es decir, límite inferior igual a límite superior). En este punto, habremos localizado la fuente del error.

Esta estrategia tiene una complejidad logarítmica $O(\log n)$, ya que en cada iteración estamos reduciendo a la mitad la parte del texto en la que estamos buscando la cadena ofensiva. Cada iteración se acerca rápidamente a la ubicación exacta de la cadena ofensiva, permitiendo una localización eficiente en términos de tiempo.