UNIVERSIDAD DE LAS FUERZAS ARMADAS

Nombre: Madellyn Tasipanta

NRC: 28952

Ejercicios Búsqueda Binaria y Hash

Problema 1:

Se cuenta con un archivo directo llamado ALUMNOS. DAT con información de alumnos de una universidad. El archivo se abre para lectura y se indexa por medio de su clave primaria (#legajo). El índice se mantiene en memoria principal. Se ofrece la posibilidad al usuario de buscar alumnos por legajo. La búsqueda binaria se aplica sobre el índice y se accede en forma directa (seek) al archivo para recuperar el registro. Se asume que el archivo está ordenado de manera ascendente por su clave primaria (#legajo). Si éste no fuese el caso, el índice debería ordenarse antes de aplicar una búsqueda binaria sobre él. Aclaración: no se provee el archivo ALUMNOS.DAT (debido a que es binario). Deberá ser generado antes de la ejecución de esta pro- grama).

Fórmula del punto medio:

medio = (inicio + fin) / 2

Complejidad temporal:

$T(n) = O(\log_2 n)$

Número máximo de iteraciones:

$k = \lceil \log_2(n) \rceil$

Donde:

- n = número de elementos
- k = iteraciones máximas

EJEMPLO

INPUT:

Array ordenado (legajos de alumnos):

A = [1001, 1005, 1010, 1015, 1020, 1025, 1030, 1035]

Índices: 0 1 2 3 4 5 6 7

n = 8 (cantidad de elementos)

Valor a buscar: x = 1025

ITERACIÓN 1:

inicio = 0

fin = 7

Cálculo del medio:

medio = ⌊(inicio + fin) / 2⌋

medio = ⌊(0 + 7) / 2⌋

medio = ⌊7 / 2⌋ medio = ⌊3.5⌋

medio = 3

Comparación:

A[medio] = A[3] = 1015 1015 < 1025

Conclusión:

El valor está en la mitad DERECHA

inicio = medio + 1 = 3 + 1 = 4

Nuevo rango: [4, 7]

Elementos restantes: [1020, 1025, 1030, 1035]

ITERACIÓN 2:

inicio = 4

fin = 7

Cálculo del medio:

medio = ⌊(inicio + fin) / 2⌋

medio = ⌊(4 + 7) / 2⌋

medio = ⌊11 / 2⌋

medio = ⌊5.5⌋

medio = 5

Comparación:

A[medio] = A[5] = 1025

1025 = 1025

OUTPUT

Valor encontrado: 1025

Posición (índice): 5

Número de iteraciones: 2

Posición en archivo (bytes): 5 × 58 = 290 bytes

Problema 2:

Diseñe una variación de Búsqueda Binaria (algoritmo 1.4) que efectúe sólo una comparación binaria (es decir, la comparación devuelve un resultado booleano) de K con un elemento del arreglo cada vez que se invoca la función. Pueden hacerse comparaciones adicionales con varia- bles de intervalo. Analice la corrección de su procedimiento. Sugerencia: ¿Cuándo deberá ser de igualdad (==) la única comparación que se hace?

- La búsqueda binaria tradicional hace **2 comparaciones** por iteración:

```
if (A[medio] == x)     // Comparación 1
else if (A[medio] < x) // Comparación 2
```

- La **variación optimizada** hace **1 sola comparación** por iteración:

if (A[medio] < x) // ÚNICA comparación

EJEMPLO

INPUT:

A = [1001, 1005, 1010, 1015, 1020, 1025, 1030, 1035]

x = 1018c

ITERACIÓN 1:

inicio = 0, fin = 7

medio = 3

¿1015 < 1018? SÍ

inicio = 4

ITERACIÓN 2:

inicio = 4, fin = 7

medio = 5

¿1025 < 1018? NO

fin = 5

ITERACIÓN 3:

inicio = 4, fin = 5

medio = 4

¿1020 < 1018? NO

fin = 4

SALIDA DEL BUCLE:

inicio = 4, fin = 4

VERIFICACIÓN FINAL:

¿A[4] == 1018?

¿1020 == 1018? NO

OUTPUT

Valor NO encontrado: 1018

Posición: -1

Iteraciones: 3

Comparaciones: 4

Problema 3:

¿Cómo podría modificar Búsqueda Binaria (algoritmo 1.4) para eliminar trabajo innecesario si se tiene la certeza de que K está en el arreglo? Dibuje un árbol de decisión para el algoritmo modificado con n = 7. Efecto análisis de comportamiento promedio y de peor caso. (Para el promedio, puede suponerse que n = 2^k − 1 para alguna k.)

ENTRADA:

- Array ordenado A[0...n-1]

- Valor x a buscar

PROCESO:

inicio ← 0

```
  fin ← n - 1
  MIENTRAS inicio < fin:  // Cambio clave: < en vez de ≤
    medio ← ⌊(inicio + fin) / 2⌋
    SI A[medio] < x:  // ÚNICA comparación booleana
      inicio ← medio + 1
    SINO:
      fin ← medio
  // Al terminar: inicio == fin
  SI A[inicio] == x:  // ÚNICA comparación de igualdad
    RETORNAR inicio
  SINO:
    RETORNAR -1
```

EJEMPLO

INPUT:

Array: [1001, 1005, 1010, 1015, 1020, 1025, 1030, 1035]

Buscar: x = 1025

ITERACIÓN 1:

inicio=0, fin=7, medio=3

A[3]=1015 < 1025? SÍ → inicio=4

ITERACIÓN 2:

inicio=4, fin=7, medio=5

A[5]=1025 < 1025? NO → fin=5

ITERACIÓN 3:

inicio=4, fin=5, medio=4

A[4]=1020 < 1025? SÍ → inicio=5

SALIDA: inicio=5, fin=5

VERIFICACIÓN:

A[5] == 1025?

OUTPUT:

Posición: 5

Iteraciones: 3

Comparaciones: 4 (3 en bucle + 1 final)

Problema 4:

Se cuenta con una lista de Pokémons, de cada uno de estos se sabe su nombre, número y tipo (solo considerar uno el principal), con los cuales deberá resolver las siguientes tareas: determinar si existe el Pokémon Cobalion y mostrar toda su información.

ENTRADA:

- Lista de Pokémons P[0...n-1]
- Nombre a buscar: "Cobalion"

PROCESO:

```
PARA i = 0 HASTA n-1:
  SI P[i].nombre == "Cobalion":
    RETORNAR i
RETORNAR -1 (no encontrado)
```

COMPLEJIDAD:

Mejor caso: O(1) - está en la primera posición

Peor caso: O(n) - está al final o no existe

Caso promedio: O(n/2) = O(n)

```
INPUT:
Lista de Pokémons (n = 10):
[0] Número: 638, Nombre: "Cobalion",   Tipo: "Acero"
[1] Número: 643, Nombre: "Reshiram",   Tipo: "Fuego"
[2] Número: 644, Nombre: "Zekrom",     Tipo: "Eléctrico"
[3] Número: 150, Nombre: "Mewtwo",     Tipo: "Psíquico"
[4] Número: 25,  Nombre: "Pikachu",    Tipo: "Eléctrico"
Buscar: "Cobalion"
```

ITERACIÓN 1:

i = 0

P[0].nombre = "Cobalion"

¿"Cobalion" == "Cobalion"?

Comparación carácter por carácter:

C == C ✓

o == o ✓

b == b ✓

a == a ✓

l == l ✓

i == i ✓

o == o ✓

n == n ✓

¡ENCONTRADO en posición 0!

Total de comparaciones: 1

Problema 5:

Se dispone de la lista de superhéroes y villanos de la saga de Marvel Cinematic Universe (MCU) de los que contamos con la información de nombre del personaje y año de la primera película en la que apareció; a partir de estos resolver las siguientes actividades: indicar quien fue el primer y el último personaje en aparecer en una película sin realizar un recorrido de la lista (podrían ser más de uno tanto el primero como el último).

Para cada personaje con año y:

- Si y < año_mínimo → actualizar año_mínimo y reiniciar lista de primeros
- Si y == año_mínimo → agregar a lista de primeros
- Si y > año_máximo → actualizar año_máximo y reiniciar lista de últimos
- Si y == año_máximo → agregar a lista de últimos

Invariante: Siempre mantenemos el año mínimo, máximo y sus personajes correspondientes.

EJEMPLO

INPUT:
5
Iron Man, 2008
Thanos, 2012
Captain America, 2011
Hulk, 2008
Black Widow, 2010

OUTPUT:
Primeros personajes en aparecer (2008):
- Iron Man
- Hulk
Últimos personajes en aparecer (2012):
- Thanos

Problema 6:

Se dispone de una lista de películas con la siguiente información: título, año de estreno, recaudación y valoración del público (de 1 a 5), los cuales debemos procesar contemplando las siguientes tareas: determinar si la película “Avengers: Infinity War” está en la lista y mostrar toda su información; indicar en qué posición se encuentra la película “Star Wars: The Return of Jedi”.

Búsqueda Lineal Secuencial:

- Comparar cada elemento con el valor buscado
- Complejidad: O(n) en el peor caso
- Retornar cuando se encuentra coincidencia exacta

Algoritmo de comparación de cadenas:

- Comparar carácter por carácter
- Si todos coinciden → encontrado
- Si alguno difiere → continuar búsqueda

INPUT:

4

Iron Man, 2008, 585200000, 4

Avengers: Infinity War, 2018, 2048000000, 5

The Dark Knight, 2008, 1005000000, 5

Star Wars: The Return of Jedi, 1983, 475100000, 4

OUTPUT:

Película encontrada: Avengers: Infinity War

Título: Avengers: Infinity War

Año: 2018

Recaudación: $2048000000

Valoración: 5/5

Posición de "Star Wars: The Return of Jedi": 4

Problema 7:

Encontrar el algoritmo de búsqueda binaria para encontrar un elemento K en una lista de elemen-tos $$X_1, X_2, ..., X_n$$ previamente clasificados en orden ascendente.El array o vector X se supone ordenado en orden creciente si los datos son numéricos, o alfabéti-camente si son caracteres. Las variables BAJO, CENTRAL, ALTO indican los límites inferior, cen-tral y superior del intervalo de búsqueda.algoritmo busqueda_binaria//declaracionesinicio//1lenar (X,N)//ordenar (X,N)leer(K)//inicializar variablesBAJO ← 1ALTO ← NCENTRAL ← ent ((BAJO + ALTO) / 2)mientras (BAJO <= ALTO) y (X[CENTRAL] <> K) hacer si K < X[CENTRAL] entonces ALTO ← CENTRAL - 1 si_no BAJO ← CENTRAL + 1 fin_si CENTRAL ← ent ((BAJO + ALTO) / 2)fin_mientrassi K = X[CENTRAL] entonces escribir('Valor encontrado en', CENTRAL)si_no escribir('Valor no encontrado')fin_sifin

En cada iteración, el espacio de búsqueda se reduce a la mitad:

- Iteración 1: n elementos
- Iteración 2: n/2 elementos
- Iteración 3: n/4 elementos

- Iteración k: n/2^k elementos

Complejidad: $O(\log_2 n)$

El algoritmo termina cuando $2^k \geq n$, por lo tanto $k = \lceil \log_2 n \rceil$

EJEMPLO

INPUT:

Array ordenado: [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]

Buscar: K = 23

Traza de ejecución:

Iter	BAJO	ALTO	CENTRAL	X[CENTRAL]	Comparación	Acción
1	1	11	6	23	K = 23	¡Encontrado!

Resultado: Valor encontrado en posición 6

Problema 8:

Un vector T tiene cien posiciones, 0..100. Supongamos que las claves de búsqueda de los elementos de la tabla son enteros positivos (por ejemplo, número del DNI).
Una función de conversión h debe tomar un número arbitrario entero positivo x y convertirlo en un entero en el rango 0..100, esto es, h es una función tal que para un entero positivo x.

h(x)=n, donde n es entero en el rango 0…100

El método del modulo, tomando 101 sera

h(x) = x mod 101

Si se tiene el DNI número 234661234, por ejemplo, se tendrá la posición 56:

234661234 mod 101 = 56

Función Hash (Dispersión)

Una **función hash** transforma una clave en un índice válido del array:

h(x) = x mod m

Donde:

- **x**: clave de búsqueda (ej: DNI)

- **m**: tamaño de la tabla (101 en este caso)
- **h(x)**: posición en el rango [0, 100]

Manejo de Colisiones

Cuando $h(x_1) = h(x_2)$ con $x_1 \neq x_2$:

- **Direccionamiento abierto**: Buscar siguiente posición libre
  - Lineal: $h'(x) = (h(x) + i) \bmod 101$
  - Cuadrático: $h'(x) = (h(x) + i^2) \bmod 101$

EJEMPLO

DNI: 234661234 h(234661234) = 234661234 mod 101 = 56 Posición en tabla: 56

Problema 9:

La clave de búsqueda es una cadena de caracteres -tal como un nombre-. Obtener las direcciones de conversión.
El método más simple es asignar a cada carácter de la cadena un valor entero (por ejemplo, A =1, B = 2, ...) y sumar los valores de los caracteres en la cadena. Al resultado se le aplica entonces el
módulo 101, por ejemplo.
Si el nombre fuese JONAS, esta clave se convertiría en el entero
10+15+14+1+19= 63
63 mod 101=63

Paso 1: Asignar valor numérico a cada carácter

- A=1, B=2, C=3, ..., Z=26

Paso 2: Sumar todos los valores

- h'(cadena) = Σ valor(carácter_i)

Paso 3: Aplicar módulo

- h(cadena) = h'(cadena) mod m

Fórmula Matemática

Para una cadena $S = c_1c_2c_3...c_n$:

$$h(S) = \left(\sum_{i=1}^{n} valor(c_i)\right) \mod 101$$

Donde:

- **valor(c_i)**: valor numérico del carácter i
- **n**: longitud de la cadena
- **101**: tamaño de la tabla (número primo)

EJEMPLO “JONAS”

```
J = 10

O = 15

N = 14

A = 1

S = 19

-------

Suma = 10 + 15 + 14 + 1 + 19 = 59
```

```
h("JONAS") = 59 mod 101 = 59

Posición en tabla: 59
```

Problema 10:

Un entero de ocho dígitos se puede dividir en grupos de tres, tres y dos dígitos, los grupos se suman juntos y se truncan si es necesario para que estén en el rango adecuado de índices. Por consiguiente, si la clave es:

62538194

y el número de direcciones es 100, la función de conversión será

625 + 381 + 94 = 1100

que se truncará a 100 y que será la dirección deseada.

Paso 1: Dividir la clave en grupos

- Para 8 dígitos: 3 + 3 + 2 dígitos
- Ejemplo: 62538194 → 625 | 381 | 94

Paso 2: Sumar los grupos

- suma = 625 + 381 + 94 = 1100

Paso 3: Truncar al rango válido

- Si suma ≥ tamaño_tabla, aplicar módulo

- h(62538194) = 1100 mod 100 = 0

Fórmula General

Para una clave de n dígitos dividida en grupos g_1, g_2, ..., g_k:

$$h(x) = \left(\sum_{i=1}^{k} g_i \right) \mod m$$

Donde m es el tamaño de la tabla.

EJEMPLO

Clave: 62538194

División: 625 | 381 | 94

Suma: 625 + 381 + 94 = 1100

Truncamiento: 1100 mod 100 = 0

Posición en tabla: 0Suma: 625 + 381 + 94 = 1100

Truncamiento: 1100 mod 100 = 0

Posición en tabla: 0