Facultad de Informática
UNIVERSIDAD NACIONAL DEL COMAHUE

Análisis de Algoritmos

Resolución TP0 - Consideraciones

21/08/2023

Alumno

- Comeron, Nicolás | FAI-1152
- Leiva, Julián | FAI-2318

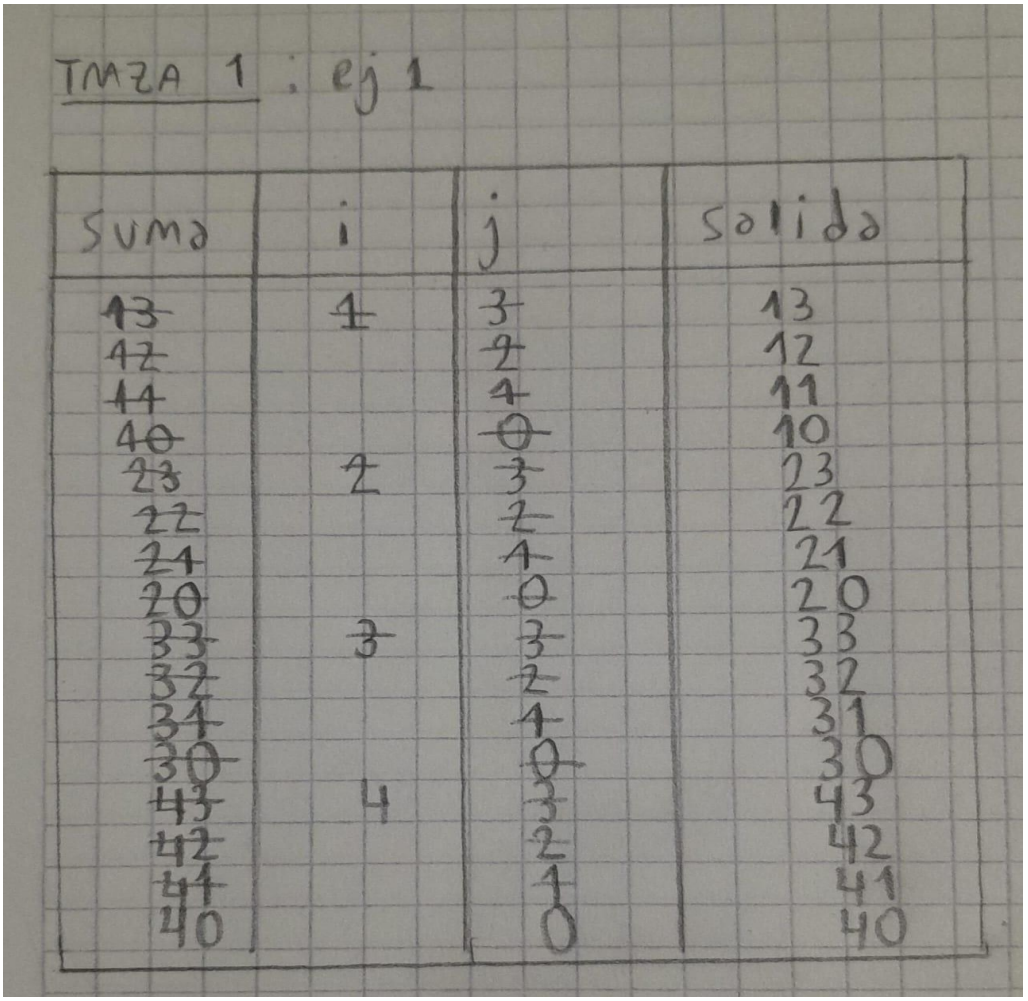
Cátedra

- Martínez Carod, Nadina
- Baeza, Natalia

Repaso de Algoritmia

1. Realiza detenidamente una traza al siguiente programa y muestra cuál será la salida por pantalla:

```
ALGORITMO ej1
    VARIABLES
    suma,i, j:ENTERO
    PARA i ← 1 HASTA 4 HACER
        PARA j ← 3 HASTA 0 PASO -1 HACER
            suma ← i * 10 + j
            escribir(suma)
        FIN PARA
    FIN PARA
FIN ALGORITMO
```



TRAZA 1 : ej 1

suma	i	j	salida
~~13~~	~~1~~	~~3~~	13
~~12~~		~~2~~	12
~~11~~		~~1~~	11
~~10~~		~~0~~	10
~~23~~	~~2~~	~~3~~	23
~~22~~		~~2~~	22
~~21~~		~~1~~	21
~~20~~		~~0~~	20
~~33~~	~~3~~	~~3~~	33
~~32~~		~~2~~	32
~~31~~		~~1~~	31
~~30~~		~~0~~	30
~~43~~	4	~~3~~	43
~~42~~		~~2~~	42
~~41~~		~~1~~	41
40		0	40

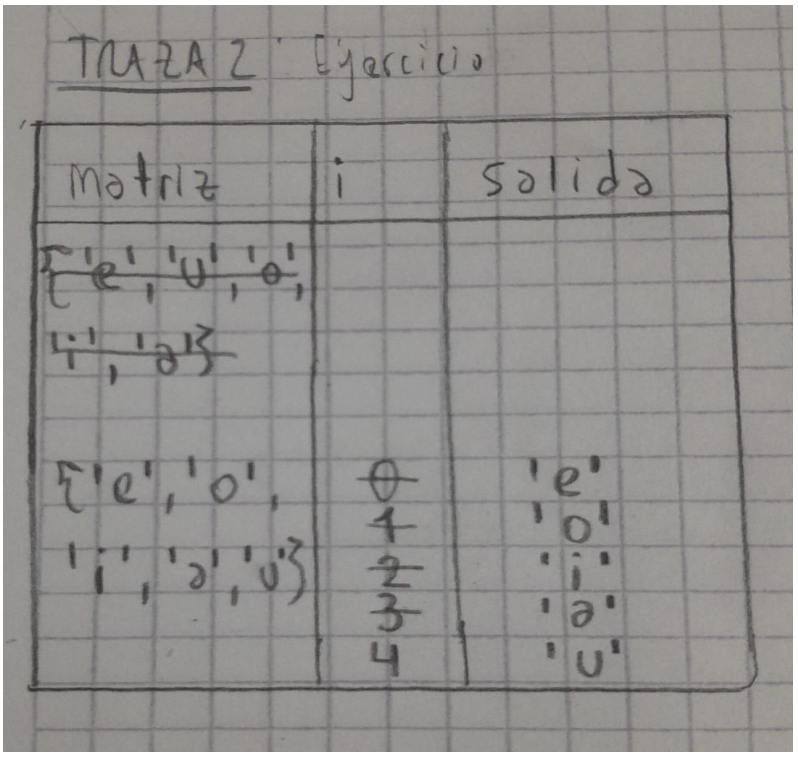
2. ¿Que imprime el siguiente programa?

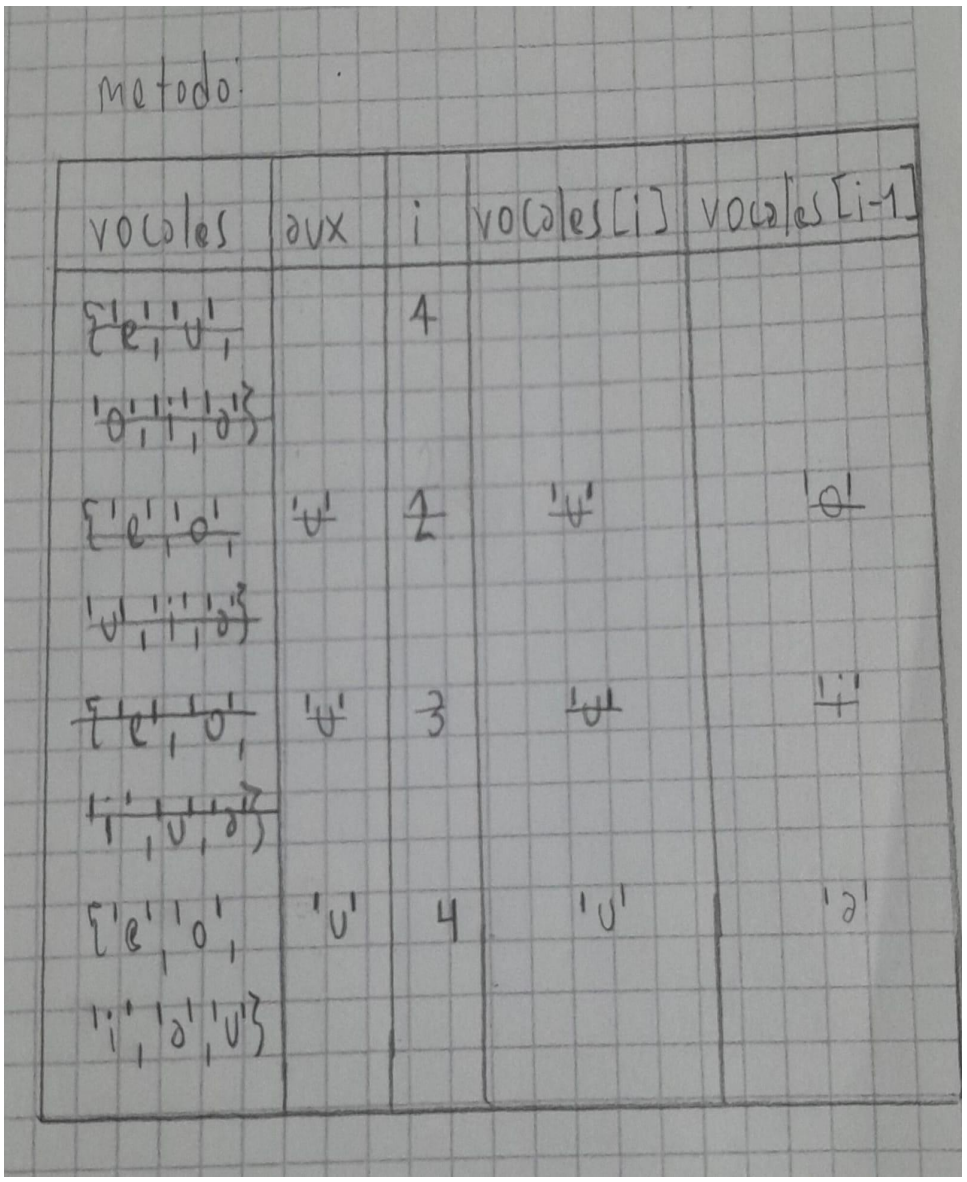
```
class Ejercicio {
    public static void main (String [] args){
        char [] matriz←{'e','u','o','i','a'};
        metodo(matriz);
        for (int i←0;i<matriz.length;i++){
            System.out.print(matriz[i];
        }
    }
```

```
    public static void metodo (char [] vocales){
        char aux;

        for (int i←1;i<vocales.length;i++){
            if (vocales[i-1]>vocales[i]){
                aux←vocales[i-1];
                vocales[i-1]←vocales[i];
                vocales[i]←aux;
            }
        }
    }
}
```

Realizamos una traza para saber que imprime el programa:





El programa imprime por pantalla: ‘e’, ‘o’, ‘i’, ‘a’, ‘u’

5. Suponiendo n ≤ 1000000 y un usuario que siga de forma óptima la lógica del juego y que quiera dar la menor cantidad de pasos hasta adivinar el número: ¿Cuál es el máximo número de intentos que puede necesitar el jugador hasta encontrar un número dentro del intervalo?

La manera más óptima para poder resolver dicho escenario sería partiendo desde el valor que determina la mitad dentro del conjunto de valores. Es decir, dado que n podría llegar a ser 1.000.000, deberíamos comenzar consultando por el valor 500.000. Si el número es acertado, el juego finaliza en dicho momento. Pero en caso contrario, el juego nos indica si debemos consultar por un número menor o mayor. De esta manera, equilibramos la cantidad de valores a izquierda y derecha, y por lo tanto, reducimos el tiempo de búsqueda descartando aquellos valores que se encuentran en el sector opuesto. Por lo tanto, en el peor de los casos, se necesitarían 20 intentos hasta dar con el número correcto.

Esta estrategia es la que conocemos como Búsqueda Binaria, y basándonos en su aplicación, se logra obtener un orden de eficiencia logarítmico(log n). Cabe mencionar que para su correcta utilización, los números deberían estar ordenados. Esto no entra en consideración para este caso, dado que simplemente se indica si el número ingresado es igual, mayor o menor (no existe tal estructura con los valores).

6. Liste y describa claramente los algoritmos para la resolución del problema de búsqueda que conoce. (tip: recordar distintas implementaciones de interfaz TablaDeBusqueda).

Para resolver problemas de búsqueda nos podemos basar en algoritmos de búsqueda secuenciales (recorrer elemento a elemento), búsqueda binaria (dividir el problema a partir de la comparativa de un valor) y búsqueda por clave/índice (utilizando una función hash). Algunas de las estructuras conocidas que permiten almacenar y realizar dichos tipos de búsqueda son las siguientes:

- Árbol binario de búsqueda: Es un árbol binario donde todos los elementos del subárbol izquierdo de cualquier nodo (si no está vacío) son menores que el árbol de dicho nodo, y todos los elementos del subárbol derecho (si no está vacío) son mayores que él. Su principal característica es que la búsqueda de un elemento es eficiente en la mayoría de los casos, ya que se puede saber la posible ubicación de un elemento en el árbol sin necesidad de visitar todos los nodos del mismo. Para que el algoritmo de búsqueda se mantenga eficiente, no se permiten insertar elementos con igual valor en un ABB, es decir, se considera un error tratar de insertar elementos repetidos. Tiene eficiencia logarítmica.

- Árbol auto-balanceado (AVL): Es un árbol binario de búsqueda que está siempre equilibrado. Como ventaja podríamos decir que el árbol AVL asegura una eficiencia logarítmica para la operación de búsqueda. Como desventaja se puede mencionar la complejidad de la implementación de las rotaciones, aunque es un costo menor, comparado con la ganancia en eficiencia al almacenar conjuntos de tamaño considerable.

- Árbol Heap: Es una estructura que permite mantener sus elementos parcialmente ordenados. Su propósito es permitir encontrar rápidamente el menor o mayor elementos de todos los elementos guardados en la estructura. Se debe destacar que la búsqueda de un elemento es de orden logarítmico. Esto está calculado para la implementación con arreglos, porque se asegura que el tiempo de acceso desde el padre a los hijos y de estos a su padre es siempre O(1). Como ventaja por ser árboles semi-completos, se pueden implementar usando arreglos, lo cual simplifica su codificación contando con acceso directo a los elementos de la estructura. Como desventaja se puede mencionar la búsqueda de cualquier otro elemento que no sea el que se encuentra en la cima del montículo (raíz) requiere recorrer la mayor parte del árbol.

- Tabla Hash: Como se sabe que la tabla hash es un arreglo de estructura estática que permite un tiempo de acceso constante a cualquier posición , por lo tanto, si la función hash es simple de calcular y produce muy pocas colisiones, el tiempo medio de recuperación de la información será constante, es decir que el tiempo para buscar un elemento no dependera del tamaño de la tabla ni del número de elementos almacenados en la misma. Sin embargo, esta eficiencia depende siempre de que la función hash distribuya uniformemente las claves. Si la función esta mal diseñada o no es la adecuada para el

conjunto de elementos a almacenar, se produciran muchas colisiones y la eficiencia sera mas pobre.

7. Al ordenar una lista de números enteros aplicando el algoritmo quicksort, como pivote se elige el primer elemento de la lista. ¿que pasaría si se selecciona otro pivote?

La elección del pivote es de suma importancia, ya que a raíz de este valor, se van a generar los subarreglos (o la estructura que utilicemos). Aquellos valores que son menores al pivote serán situados en uno de los subarreglos, mientras que aquellos que son mayores irán al otro. En este momento, el pivote ya se encontraría ordenado. Este proceso se repite de manera recursiva, mientras haya dos o más elementos en el subarreglo. Con lo cual, respondiendo a la pregunta, lo ideal sería que el pivote que elijamos sea aquel que determine el medio (numéricamente hablando, no la posición) equitativamente, para poder aprovechar la división recursiva de subestructuras. El problema radica en que si se recibe una estructura con valores aleatorios, no disponemos de tal conocimiento (eligiríamos a ciegas el pivote que determine el medio). Distinto es el caso si poseemos noción del conjunto de datos con el que se trabaja. A su vez, podría llegar a ocurrir que se reciba una estructura con datos ordenados (ascendentemente, por ejemplo), en donde ambos valores extremos generarían subestructuras vacías (a izquierda/a derecha). Y es para estos casos en donde se ve afectado mayormente el rendimiento del algoritmo.

8. Para la solución al problema, las siguientes son algunas de las consideraciones que fueron tenidas en cuenta:
- La lectura de datos desde el archivo de texto debe seguir el siguiente formato: nombreAlumno nota nota nota nota nota. Asumimos que los mismos ya fueron corroborados con anterioridad, es decir, son datos completos y precisos/correctos.
- Optamos por utilizar ArrayList como estructura para almacenar los datos de los alumnos, dado que, a diferencia de los Array tradicionales, no están atados a una capacidad predefinida (son dinámicos).
- Conformamos ArrayList de ArrayList, es decir, una “matriz”. A diferencia del array tradicional, permite la incorporación de nuevos alumnos a la estructura. También podrían añadirse nuevas materias para dichos alumnos, aunque este no es el caso (son 5 asignaturas).
- Entonces, un Alumno es un ArrayList con seis posiciones utilizadas:
    - Pos 0: indica su nombre.
    - Pos 1 a 5: indican sus notas
- Luego, otro ArrayList contendrá a cada Alumno en donde:
    - La longitud de este ArrayList es de la cantidad de alumnos, en principio.
    - Cada posición apunta a un ArrayList (Alumno) en su totalidad. Por ejemplo: Pos 0: [Juan, 10, 7, 9, 10, 9]
- Para el ordenamiento, se utilizaron los algoritmos de quicksort y selección. Si bien ambos suelen utilizarse con arreglos enteros, realizamos la adaptación a float correspondiente. También se modificó el ordenamiento, de ascendente a decreciente. Incluímos el algoritmo de selección ya que, si bien no es de los más eficientes, la cantidad de alumnos de un cursado no es tan significativa como para que afecte el rendimiento al ordenarlos.