Universidad Nacional Autónoma de México

Facultad de Ciencias

Introducción a Ciencias de la Computación

Teoria

- ¿Qué es un ADT (*Abstract Data Type*)?
  Podemos pensar en un ADT como un concepto. Supongamos que tenemos una estructura de datos que queremos escribir y usarla para agregar, eliminar, ordenar e imprimir información. Es puramente conceptual. En Java escribimos una interfaz que contiene todos los encabezados de métodos relacionados con las operaciones que queremos para el ADT y luego se incluye esa interfaz cuando se escribe la clase para la estructura de datos, la clase en este caso es la implementación siendo que si tu tienes una clase, escribes código para lo que hace el adt.

- ¿Define qué es una Lista Simplemente Ligada?
  Podemos pensar que es como un tren. Un tren tiene una secuencia de vagones. Cada vagón contiene algo de carga y está conectado al siguiente vagón. Para llegar desde el vagón delantero del tren a cualquiera de los otros vagones del tren, tenemos que pasar por todos los vagones entre esos dos.
  Se utiliza para crear otras estructuras de datos. Con la explicacion previa podemos decir que esta formada por una serie de nodos que contienen información y uno o dos enlaces que los conectan al nodo anterior y/o siguiente. En Java usariamos una clase "Nodo", que tiene una variable para contener datos y una variable que apunta al siguiente nodo de la lista.

- ¿Define qué es un Arreglo?
  Es una lista de elementos almacenados en un área particular de la memoria con un nombre. Sin un orden particular, lo que entra primero ocupa el primer lugar y así sucesivamente.Es como una variable que se puede dividir en compartimentos.

- ¿Qué ventajas tiene una Lista Simplemente Ligada?
  A diferencia de arreglos, es más predecible porque no necesita perder tiempo periódicamente copiando todos sus datos en una nueva lista haciendolo mejor opcion para programas con requisitos de sincronización, cosas como software que controla máquinas industriales o aviones donde un breve corte de flujo podría tener consecuencias muy negativas.

- ¿Qué desventaja tiene una Lista Simplemente Ligada?
  Resulta que en la mayoría de los casos, es mejor usar arreglos o listas de arreglos en lugar de listas ligadas porque aunque estas permiten insertar datos, la mayoría de los programas del mundo real en realidad no insertan valores en listas.

- ¿Qué ventajas tiene un Arreglo?
  Cómo está diseñado el hardware informático moderno: dar un pequeño salto al siguiente elemento de la memoria es más barato que seguir un puntero y dar un salto potencialmente grande a otro lugar de la memoria.
- ¿Qué desventaja tiene un Arreglo?
  Una lista puede crecer a diferencia de los arreglos. La inserción y recuperación de datos al principio o al final es fácil y muy rápida. Eliminar elementos también es mucho más fácil de una lista que en un arreglo por ejemplo porque los elementos que siguen al elemento eliminado avanza una posición simplemente.

Práctica

1. **Problema:** Dada una cadena `s` que consiste en palabras con espacio, regresa la longitud de la última palabra en la cadena.
   - **Entrada:** `s = "Hola Mundo"`
   - **Salida:** `5`
   - **Explicación:** La última palabra es `Mundo` y su longitud es `5`.

```
    //  Creamos una clase llamada Main que contiene los
    metodos longitudUltimaPalabra y main.
    public class Main {

    // En el metodo longitudUltimaPalabra, tenemos una cadena
    's'para calcular la longitud de la ultima palabra en la
    cadena.
    public int longitudUltimaPalabra(String s) {
        s = s.trim();
        String[] palabras = s.split(" ");

//  Dentro del metodo longitudUltimaPalabra:
// - Utilizamos 's.trim()' para eliminar los espacios en
    blanco al principio y al final de la cadena para que no
    haya espacios adicionales.

// Luego, dividimos la cadena en palabras utilizando 's.split
    (" ")', que divide la cadena en palabras separadas por
    espacios en blanco. El resultado es un arreglo de cadenas
    que llamamos 'palabras'.

        if (palabras.length > 0) {
            String ultimaPalabra = palabras[palabras.length -
    1];
            int longitud = ultimaPalabra.length();
            System.out.println("La ultima palabra es: " +
    ultimaPalabra + " y su longitud es: " + longitud);
            return longitud;
        } else {
            System.out.println("No hay palabras");
            return 0;
```

```
        }
    }
//  Si la longitud de 'palabras' es mayor que 0, entonces hay
    al menos una palabra en la cadena y se obtiene la ultima
    palabra.
//  Tomamos la ultima palabra encontrada en 'palabras'
    utilizando 'palabras[palabras.length - 1]'.
//  Calculamos la longitud de la ultima palabra utilizando el
    metodo 'length()'
// Si la cadena esta vacia o no contiene palabras (es decir,
    la longitud de 'palabras' es 0), entonces regrsa 0.

    public static void main(String[] args) {
        Main main = new Main();
        String s = "Hola Mundo";
        int longitud = main.longitudUltimaPalabra(s);
    }
}

// En el metodo 'main', creamos una instancia de la clase '
    Main' para llamar a 'longitudUltimaPalabra'.
// Luego, definimos la cadena 's' que contiene "Hola Mundo".
// Llamamos al metodo 'longitudUltimaPalabra' en 'Main' para
    obtener la longitud de la ultima palabra en la cadena 's'.
// Y ya para terminar se imprimie el resultado usando 'System.
    out.println()' para mostrar la longitud de la ultima
    palabra. Imprime "La longitud de la ultima palabra en la
    cadena es: 5", siendo esta pues la palabra mundo.


```

2. **Problema:** Dada la `cabeza` de una lista simplemente ligada, verifica si la lista es un palíndromo.

   - **Entrada:** `cabeza = [1,2,2,1]`
   - **Salida:** `true`
   - **Explicación:** La lista es un palíndromo.

```
      // Importamos la clase java.util.Stack para utilizarla

import java.util.Stack;

//  Bueno esto seria ponerlo en su propio archivo,
    definiriamos la clase Nodo que representa un nodo de la
    lista simplemente ligada.
public class Nodo {
    int valor;
    Nodo siguiente;

    public Nodo(int valor) {
        this.valor = valor;
    }
}


```

```java
// Definimos la clase Main que contiene el metodo para verificar si la lista es un palindromo.

public class Main {
    //  En esPalindromo, recibimos la cabeza de la lista como entrada y verificamos si si es.

    public boolean esPalindromo(Nodo cabeza) {
        if (cabeza == null || cabeza.siguiente == null) {
            // Una lista vacia o con un solo elemento siempre es un palindromo.
            return true;
        }

        Nodo slow = cabeza;
        Nodo fast = cabeza;
        Stack<Integer> pila = new Stack<>();

        // Movemos slow a la mitad de la lista y agregamos los valores en la pila.
        while (fast != null && fast.siguiente != null) {
            pila.push(slow.valor);
            slow = slow.siguiente;
            fast = fast.siguiente.siguiente;
        }

        // Si la lista tiene una longitud impar, omitimos el elemento del medio.
        if (fast != null) {
            slow = slow.siguiente;
        }

        // Comparamos la mitad original con la mitad revertida utilizando la pila.
        while (slow != null) {
            if (slow.valor != pila.pop()) {
                return false;
            }
            slow = slow.siguiente;
        }

        // Si las mitades coinciden en todos los elementos, retornamos true, indicando que la lista es un palindromo.
        return true;
    }

    public static void main(String[] args) {
        Main main = new Main();
        Nodo nodo1 = new Nodo(1);
        Nodo nodo2 = new Nodo(2);
        Nodo nodo3 = new Nodo(2);
        Nodo nodo4 = new Nodo(1);

        nodo1.siguiente = nodo2;
        nodo2.siguiente = nodo3;
        nodo3.siguiente = nodo4;

```

```
        boolean resultado = main.esPalindromo(nodo1);

        // 10. Imprimimos el resultado
        System.out.println("Entrada: cabeza = [1, 2, 2, 1]");
        System.out.println("Salida: " + resultado);
        if (resultado) {
            System.out.println("Explicacion: La lista si es palindromo.");
        } else {
            System.out.println("Explicacion: La lista no es un palindromo.");
        }
    }
}
```

3. **Problema:** Dado un arreglo de enteros **ordenados** en orden ascendente, existe exactamente un entero en el arreglo que aparece más del 25 % de las veces, regresa el valor de dicho entero.

   - **Entrada:** `arr = [1,2,2,6,6,6,6,7,10]`
   - **Salida:** `6`
   - **Entrada:** `arr=[1,1]`
   - **Salida:** `1`

```
 public class Main {
  public int encuentraEntero(int[] arr) {
      int n = arr.length;
      int frecuencia = n / 4; // El 25% de n

      // Recorremos el arreglo y mantenemos un registro del numero con mayor frecuencia.
      int actual = arr[0];
      int conteo = 1;
  //Recorremos el arreglo a partir del segundo elemento (i = 1) y comparamos cada elemento con el valor actual.

      for (int i = 1; i < n; i++) {
          if (arr[i] == actual) {
              conteo++;
              if (conteo > frecuencia) {
                  return actual;
              }
          } else {
              actual = arr[i];
              conteo = 1;
          }
      }

      return actual;
  }
  //Si el elemento es igual al valor actual, incrementamos el contador conteo. Si el contador supera la frecuencia, significa que hemos encontrado el num que buscamos y lo devolvemos como resultado
```

```
26
27      public static void main(String[] args) {
28          Main main = new Main();
29          //Creamos una instancia de la clase Main llamada main.
30          int[] arr1 = {1, 2, 2, 6, 6, 6, 6, 7, 10};
31          int resultado1 = main.encuentraEntero(arr1);
32          System.out.println("Entrada: arr = [1, 2, 2, 6, 6, 6,
    6, 7, 10]");
33          System.out.println("Salida: " + resultado1); //
    imprime 6
34
35          //Definimos dos arreglos de entrada (arr1 y arr2).
36          //Llamamos al metodo encuentraEntero en la instancia
    main para encontrar el num en cada uno de ellos
37
38          int[] arr2 = {1, 1};
39          int resultado2 = main.encuentraEntero(arr2);
40          System.out.println("Entrada: arr = [1, 1]");
41          System.out.println("Salida: " + resultado2); //
    imprime 1
42      }
43  }
44
45
46
```

4. Dada la `cabeza` de una lista simplemente ligada. El valor de cada nodo será `0` o `1`. La lista mantiene la representación binaria de un número. Regresa el valor decimal del número representado por la lista.

   Nota: El bit más significativo es la cabeza de la lista.

   - **Entrada:** `cabeza = [1,0,1]`
   - **Salida:** `5`
   - **Explicación:** (101) en base 2 = (5) en base 10.

```
1           // definimos la clase nod, si fuera en una ide seria
    en otro archivo, esto para representar un nodo
2   public class Nodo {
3       int valor;
4       Nodo siguiente;
5
6       public Nodo(int valor) {
7           this.valor = valor;
8       }
9   }
10
11  // definimos la clase Main que contiene el metodo para obtener
     el valor decimal del numero binario representado por la
    lista
12  public class Main {
13      public int obtenDecimal(Nodo cabeza) {
14          //  Inicializamos una variable para mantener el valor
    decimal en 0.
15          int decimal = 0;
```

```
        Nodo actual = cabeza;

        // Recorremos la lista desde la cabeza hasta el final.
        while (actual != null) {
            // Se multilplica el valor actual por 2 y sumamos el valor del nodo actual (0 o 1).
            decimal = decimal * 2 + actual.valor;
            actual = actual.siguiente;
        }

        // Devolvemos el valor decimal del num. binario representado por la lista.
        return decimal;
    }

    public static void main(String[] args) {
        Main main = new Main();
        Nodo nodo1 = new Nodo(1);
        Nodo nodo2 = new Nodo(0);
        Nodo nodo3 = new Nodo(1);

        nodo1.siguiente = nodo2;
        nodo2.siguiente = nodo3;

        //  Llamamos al metodo obtenDecimal para obtener el valor decimal.
        int resultado = main.obtenDecimal(nodo1);

        //  Se imprime
        System.out.println("Entrada: cabeza = [1, 0, 1]");
        System.out.println("Salida: " + resultado); // imprime 5
    }
}
```

5. **Problema:** Dada la `cabeza` de una lista simplemente ligada, regresa la lista desde la mitad de la lista.

   Si la lista tiene dos mitades, entonces regresa la lista desde el segundo elemento a la mitad.

   - **Entrada:** `cabeza = [1,2,3,4,5]`
   - **Salida:** `[3,4,5]`
   - **Explicación:** El nodo a la mitad es 3.
   - **Entrada:** `cabeza = [1,2,3,4,5,6]`
   - **Salida:** `[4,5,6]`
   - **Explicación:** Dado que tenemos dos nodos a la mitad con los valores 3 y 4, regresamos a partir del nodo 4.

```
        // si fuera un ide definimos la clase Nodo en su propio archivo
public class Nodo {
    int valor;
    Nodo siguiente;

    public Nodo(int valor) {
        this.valor = valor;
    }
}

// ahora si definimos la clase Main que contiene el metodo para obtener la lista desde la mitad de la lista.

public class Nodo {
    int valor;
    Nodo siguiente;

    public Nodo(int valor) {
        this.valor = valor;
    }
}

public class Main {
    public Nodo obtenMitad(Nodo cabeza) {
        if (cabeza == null || cabeza.siguiente == null) {
            return null; // No hay mitad en una lista vacia o con un solo elemento.
        }

        Nodo puntero1 = cabeza;
        Nodo puntero2 = cabeza;

        while (puntero2 != null && puntero2.siguiente != null) {
            puntero1 = puntero1.siguiente;
            puntero2 = puntero2.siguiente.siguiente;
        }

```

```
36      //En el metodo obtenMitad, comprobamos si la lista esta
    vacia o tiene un solo elemento. Si es asi, retornamos null
     ya que no hay mitad
37          if (puntero2 != null) {
38              // Hay un numero par de nodos, enlazamos el nodo
    anterior al nodo a la mitad al nodo nulo.
39              Nodo temp = puntero1.siguiente;
40              puntero1.siguiente = null;
41              return temp;
42          } else {
43              // Hay un numero impar de nodos, la mitad esta en
    puntero1.
44              return puntero1;
45          }
46      }
47
48      public static void main(String[] args) {
49          Main main = new Main();
50          Nodo nodo1 = new Nodo(1);
51          Nodo nodo2 = new Nodo(2);
52          Nodo nodo3 = new Nodo(3);
53          Nodo nodo4 = new Nodo(4);
54          Nodo nodo5 = new Nodo(5);
55
56          nodo1.siguiente = nodo2;
57          nodo2.siguiente = nodo3;
58          nodo3.siguiente = nodo4;
59          nodo4.siguiente = nodo5;
60
61          Nodo salida = main.obtenMitad(nodo1);
62
63          System.out.println("Entrada: cabeza = [1, 2, 3, 4, 5]"
    );
64          System.out.print("Salida: [");
65          while (salida != null) {
66              System.out.print(salida.valor);
67              salida = salida.siguiente;
68              if (salida != null) {
69                  System.out.print(",");
70              }
71          }
72          System.out.println("]");
73          System.out.println("Explicacion: El nodo a la mitad es
     3.");
74      }
75  }
76
77
```

6. **Problema:** Dado un arreglo de enteros `arr`, regresa el máximo común divisor del número más pequeño y el número más grande en el arreglo `arr`.

   Nota: El máximo común divisor de dos números es el número más grande que divide a ambos números sin dejar residuo.

   - **Entrada:** `arr = [2,5,6,9,10]`
   - **Salida:** 2
   - **Explicación:**
     - El número más pequeño en el arreglo es 2.
     - El número más grande es 10.
     - El MCD de 2 y 10 es 2.

```
    public class Main {
public int encuentraMCD(int[] arr) {
    // Comprobamos si el arreglo esta vacio o tiene un
solo elemento.
    if (arr == null || arr.length < 2) {
        return -1; // No se puede calcular el MCD en este
caso.
    }

    int min = arr[0];
    int max = arr[0];

    // Encontramos el numero mas peque y el numero mas
grande en el arreglo.
    for (int num : arr) {
        if (num < min) {
            min = num;
        }
        if (num > max) {
            max = num;
        }
    }

    // Calculamos el MCD de los numeros mas peques y mas
grandes
    int mcd = calcularMCD(min, max);
    return mcd;
}
private int calcularMCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

public static void main(String[] args) {
    Main main = new Main();
    int[] arr = {2, 5, 6, 9, 10};
```

```
        int resultado = main.encuentraMCD(arr);

        System.out.println("Entrada: arr = [2, 5, 6, 9, 10]");
        System.out.println("Salida: " + resultado);
        System.out.println("Explicacion: El numero mas peque
    es 2, el numero mas grande es 10 y el MCD de 2 y 10 es 2."
    );
    }
}
        }

```

7. **Problema:** Dado un arreglo de enteros `arr` y un número regresa el índice de la primera aparición del número, si no existe en el arreglo regresa `-1`.
   - **Entrada:** `arr = [1,2,3,4,5], num = 3`
   - **Salida:** 2
   - **Explicación:** El número 3 aparece por primera vez en el índice 2 del arreglo.

```

    public class Main {
    public int encuentraIndice(int[] arr, int num) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == num) {
                return i; // aparece, devolvemos el indice.
            }
        }
        return -1; // No se encontro el numero
    }

    public static void main(String[] args) {
        Main main = new Main();
        int[] arr = {1, 2, 3, 4, 5};
        int num = 3;
        int resultado = main.encuentraIndice(arr, num);

        System.out.println("Entrada: arr = [1, 2, 3, 4, 5],
    num = " + num);
        System.out.println("Salida: " + resultado);
        if (resultado != -1) {
            System.out.println("Explicacion: El numero " + num
     + " aparece por primera vez en el indice " + resultado +
    " del arreglo.");
        } else {
            System.out.println("Explicacion: El numero " + num
     + " no existe en el arreglo.");
        }
    }
}
    //En el main, se crea una instancia de la clase Main, se
    define el arreglo de entrada [1, 2, 3, 4, 5] y el nUM a
    buscar 3, se llama al metodo encuentraIndice y se imprime

        }

```