Estrategias de Programación y Estructuras de Datos
Tema 1: Estructuras de Datos Básicas
Ejercicios propuestos

1. Diseñar un interfaz `ListIPIF<E>` que defina las operaciones de las Listas con Punto de Interés (Interest Point). Recuerde que en este tipo de listas existe un puntero que puede desplazarse hacia adelante y hacia atrás y que todas las operaciones de consulta y modificación se realizan en la posición del puntero. ¿Dónde situaría este interfaz en el mapa de TAD de la asignatura?

2. Diseñar un interfaz `ListHTIF<E>` que defina las operaciones de las Listas Primero-Siguiente (o Cabeza-Cola: Head-Tail) en las que el acceso y la modificación sólo puede realizarse sobre el primer elemento de la lista. Todas las operaciones, además, deberán devolver un valor (es decir, su tipo de salida no puede ser `void`). ¿Dónde situaría este interfaz en el mapa de TAD de la asignatura?

3. Diseñar un interfaz `SequenceMS<E>` que defina las operaciones sobre secuencias con un tamaño máximo limitado (Max Size). ¿Dónde situaría este interfaz en el mapa de TAD de la asignatura?

4. ¿Qué habría que modificar en los interfaces `ListIF<E>`, `StackIF<E>` y `QueueIF<E>` para crear los interfaces `ListMSIF<E>`, `StackMSIF<E>` y `QueueMSIF<E>` que extendiesen a `SequenceMS<E>`?

Estrategias de Programación y Estructuras de Datos

Tema 2 – Recursividad. Ejercicios Propuestos

A continuación, se proponen ejercicios para complementar el estudio del tema 2 de la asignatura de Estrategias de Programación y Estructuras de Datos. Los ejercicios 5 y 6 utilizan estructuras de datos que se estudian a fondo en temas posteriores; recomendamos que se vuelva sobre ellos una vez que se hayan estudiado las listas y los árboles.

1. Función factorial

El factorial de un número natural n se define como

- 1 , si n=0
- n * factorial(n-1), en caso contrario.

Identifíquense las partes de esta definición recursiva:

- cuál sería el caso no recursivo, y qué haría el programa en este caso.
- Cuál sería el caso recursivo, cuál sería el argumento de la llamada recursiva, y cómo se obtiene el resultado final a partir del resultado de la llamada recursiva.

2. Serie de Fibonacci

El elemento n-ésimo de la serie de Fibonacci se define como

- 1, si n=0 o n=1
- Fibonacci(n-1) + Fibonacci(n-2) si n>1

Identifíquense las partes de esta definición recursiva como en el ejercicio anterior. ¿Por qué no sería eficiente una implementación directa de la serie de Fibonacci según esa definición? ¿Qué alternativas se le ocurren?

3. Exponencial de un número

Considérese la siguiente definición recursiva de la exponenciación de enteros a^b:

$a^b = 1$, si b=0
$= a^{b/2} * a^{b/2}$, si b>0

Identifíquense las partes de esta definición recursiva como en los ejercicios anteriores. ¿Es correcta? ¿Cómo se convertiría en un programa recursivo? ¿Sería eficiente? ¿Qué problemas habría que solucionar?

4 Más funciones numéricas recursivas

Explica qué calculan, y qué problemas tienen en su caso, cada una de las siguientes definiciones recursivas de funciones sobre enteros:

$f_1(n) = 2$ si $n=1$
$\quad 2+f_1(n-1)$ si $n>1$

$f_2(n) = 1$ si $n=0$
$\quad e*f_2(n-1)$ si $n>0$

$f_3(n) = 1$ si $n=3$
$\quad = 2*f_3(n-2)$ si $n\neq 3$

$f_4(n) = 0$ si $n=0$
$\quad = 3*f_4(n-1)$ si $n>0$

$f_5(n) = f_5(n-1)$ si $n > 5$
$\quad = 3*f_5(n-2)$ si $n < 5$

5. Lista módulo

La entrada del programa es una lista de números enteros, y la salida es el resultado de sumarlos todos.

Utilícese la definición recursiva de una lista, es decir:

Una lista de enteros es, o bien una lista vacía, o bien un primer elemento y una lista con el resto de elementos. Supónganse disponibles los métodos isEmpty? (devuelve cierto si la lista está vacía y falso en caso contrario), first (devuelve el primer elemento de la lista) y rest (devuelve una lista con todos los elementos menos el primero).

6. Árbol de suma nula

Diseñar un programa cuya entrada es un árbol binario de enteros, y cuya salida es cierto si la suma de todos los enteros del árbol es cero, y falso en caso contrario.

Para diseñar el programa, utilícese la siguiente definición recursiva de un árbol binario de enteros:

Un árbol binario es, o bien un árbol vacío, o bien un nodo raíz (un valor entero) y dos árboles (que llamaremos árbol izquierdo y árbol derecho).

Supónganse disponibles los métodos *isEmpty?* (devuelve cierto si el árbol es vacío y falso en caso contrario), *root* (devuelve el valor del nodo raíz), *leftChild* (devuelve el subárbol izquierdo) y *rightChild* (devuelve el subárbol derecho).

Estrategias de Programación y Estructuras de Datos
Tema 3: Análisis Básico de Algoritmos
Ejercicios propuestos

1. Calcular el coste del Algoritmo de Euclides:

```
public int mcd(int A, int B) {
  if ( B == 0 ) { return (A); }
  else { return (mcd(B, A % B)); }
}
```

2. Calcular el coste del algoritmo “multiplicación rusa”:

```
public int mult_rusa(int A, int B) {
  if( A == 1 ){ return (B); }
  if ( A%2 != 0 ) { return(B+mult_rusa( A/2 , B*2)); }
  else { return(mult_rusa( A/2 , B*2)); }
}
```

3. Calcular el coste del algoritmo “potencia”:

```
public int potencia(int B, int N) {
  if( N == 0 ){ return (1); }
  else { return(B*potencia(B,N-1)); }
}
```

4. Calcular el coste del algoritmo “potencia optimizada”:

```
public int potencia2(int B, int N) {
  if( N == 0 ){ return (1); }
  int rec = potencia2(B,N/2);
  if ( N%2 == 0 ) { return(rec*rec); }
  else { return(B*rec*rec); }
}
```

5. Calcular el coste de invertir un número:

```
public int invertir(int n) { return invertirAux(0,n); }
public int invertirAux(int ac, int n) {
  if (n == 0 ) { return ac; }
  return invertirAux(ac*10+(n % 10), n / 10);
}
```

Estrategias de Programación y Estructuras de Datos
Tema 4: Listas
Ejercicios propuestos

1. Calcular el coste de todas las operaciones públicas (no abstractas) de `Collection<E>`.

2. Calcular el coste de todas las operaciones públicas de `Sequence<E>`.

3. Calcular el coste de todas las operaciones públicas de `List<E>`.

4. Realizar una implementación de `ListIPIF<E>` (Listas con Punto de Interés) utilizando dos pilas como Estructura de Datos de soporte. Calcular el coste de todas las operaciones públicas de esta implementación.

5. Programar dos versiones de un método:

   ```
   ListIF<E> invierte(ListIF<E> l)
   ```

   que invierta la lista dada por parámetro.

   a) Primera versión: utilizando iteradores.
   b) Segunda versión: sin utilizar iteradores.

   Compare el coste asintótico temporal en el caso peor de ambas implementaciones.

6. Programar un método que determine si una lista está ordenada o no.

7. Programar un método que determine si en una lista existe un rellano (definido como un cierto número de elementos contiguos que son iguales), bajo los siguientes supuestos:

   a) Sabemos que la lista está ordenada.
   b) No sabemos si la lista está ordenada.

8. Programar un método:

   ```
   ListIF<E> mezclar(ListIF<E> a, ListIF<E> b)
   ```

   que recibe dos listas ordenadas crecientemente y devuelve el resultado de mezclar ambas listas manteniendo el orden.

9. Implementar una clase `SequenceMS<E>` que implemente el interfaz `SequenceMSIF<E>` (ver ejercicio 3 del tema 1), de secuencias con tamaño máximo limitado, utilizando un Array como estructura de datos de soporte. Calcule el coste asintótico temporal en el caso peor de todos sus métodos públicos y compárelo con el coste de los métodos equivalentes de `Sequence<E>`.

10. Implementar una clase `ListMS<E>` que extienda la clase `SequenceMS<E>` y que implemente el interfaz `ListMSIF<E>` (ver ejercicio 4 del tema 1). Compare el coste asintótico temporal en el caso peor de todos sus métodos públicos con los correspondientes de `List<E>`.

Estrategias de Programación y Estructuras de Datos
Tema 5: Pilas y Colas
Ejercicios propuestos

1. Calcular el coste de todas las operaciones públicas de `Stack<E>` y `Queue<E>`.

2. Programar tres versiones de un método:

   ```
   StackIF<E> invierte(StackIF<E> s)
   ```

   que invierta la pila dada por parámetro. Dicho método deberá realizarse fuera de la clase `Stack<E>`.

   a) Primera versión: utilizando iteradores.
   b) Segunda versión: sin utilizar iteradores y de forma iterativa.
   c) Tercera versión: sin utilizar iteradores y de forma recursiva.

   Compare el coste asintótico temporal en el caso peor de las implementaciones.

3. Realice el ejercicio anterior sobre colas.

4. Programar un método `rotateQ` que rote una cola a derecha e izquierda:

- El método recibe un entero (sin restricciones).
- Si el entero es positivo, la cola se rotará hacia la izquierda y si es negativo, hacia la derecha. A continuación podemos ver un ejemplo:

  - Cola original: `primero → 1 , 2 , 3 , 4 , 5 ← último`
  - Cola tras `rotateQ(4)`: `primero → 5 , 1 , 2 , 3 , 4 ← último`
  - Cola tras `rotateQ(-2)`: `primero → 4 , 5 , 1 , 2 , 3 ← último`

  a) Realice el método de manera externa a la clase `Queue<E>`, de forma que devuelva una cola con el resultado pedido.
  b) Añada el método a la clase `Queue<E>`, de forma que modifique la cola llamante.

  Calcule el coste asintótico temporal en el caso peor de ambas versiones y compárelo. Procure que el coste real (no asintótico) sea el menor posible.

5. Programar un método `rotateS` que rote una pila hacia arriba y hacia abajo:

- El método recibe un entero (sin restricciones).
- Si el entero es positivo, la pila se rotará hacia arriba y si es negativo, hacia abajo. A continuación podemos ver un ejemplo:

  - Pila original: `cima → 1 , 2 , 3 , 4 , 5`
  - Pila tras `rotateS(3)`: `cima → 4 , 5 , 1 , 2 , 3`
  - Pila tras `rotateS(-1)`: `cima → 5 , 1 , 2 , 3 , 4`

  a) Realice el método de manera externa a la clase `Stack<E>`, de forma que devuelva una pila con el resultado pedido.
  b) Añada el método a la clase `Stack<E>`, de forma que modifique la pila llamante.

  Calcule el coste asintótico temporal en el caso peor de ambas versiones y compárelo. Procure que el coste real (no asintótico) sea el menor posible.

6. Implementar una clase `StackMS<E>` que extienda la clase `SequenceMS<E>` (ver ejercicio 9 del tema 4) y que implemente el interfaz `StackMSIF<E>` (ver ejercicio 4 del tema 1). Compare el coste asintótico temporal en el caso peor de todos sus métodos públicos con los correspondientes de `Stack<E>`.

7. Implementar una clase `QueueMS<E>` que extienda la clase `SequenceMS<E>` (ver ejercicio 9 del tema 4) y que implemente el interfaz `QueueMSIF<E>` (ver ejercicio 4 del tema 1). Compare el coste asintótico temporal en el caso peor de todos sus métodos públicos con los correspondientes de `Queue<E>`.

8. Enriquezca las clases `List<E>`, `Stack<E>` y `Queue<E>` con constructores por copia que permitan realizar conversiones entre esos tipos de datos. Por ejemplo, en la clase `List<E>` se debería poder construir una lista con la misma secuencia que la cola o la pila que se reciba por parámetro.

Estrategias de Programación y Estructuras de Datos
Tema 6: Árboles
Ejercicios propuestos

1. Calcular el coste de todas las operaciones públicas de `Tree<E>`, `GTree<E>` y `Btree<E>`.

2. Programar un método:

   `getMax()`

   dentro de la clase `GTree` que devuelva el valor máximo de un árbol general. Consideremos los siguientes supuestos:

   a) Los nodos del árbol no están ordenados según su valor.
   b) El valor de la raíz es, siempre, menor que el de todos sus hijos.

   Calcule el coste asintótico temporal en el caso peor de este método.

3. Programar un método:

   `getMax()`

   dentro de la clase `BTree` que devuelva el valor máximo de un árbol general. Consideremos los siguientes supuestos:

   a) Los nodos del árbol no están ordenados según su valor.
   b) El valor de la raíz es, siempre, menor que el de todos sus hijos.
   c) El valor de la raíz es, siempre, menor que el de todos los nodos contenidos en uno de sus hijos y mayor que el de todos los nodos contenidos en el otro.

4. Debido a la estructura de datos utilizada, no es posible mantener los atributos de tamaño de los árboles (`size`) de manera que podamos utilizar el método `size()` de `Collection` para devolver el tamaño de un árbol (tanto general como binario).

   En este ejercicio proponemos una solución para este problema realizando los siguientes pasos:

   a) Añada un atributo `parent` que apunte al padre del árbol. ¿En qué clase lo añadiría y con qué tipo?

b) Identifique todos los métodos de las clases `GTree` y `BTree` que se vean afectados por la inclusión de este atributo, de manera que siempre se mantenga actualizado.
c) Realice las modificaciones oportunas en estos métodos y compare su coste asintótico temporal en el caso peor con el calculado en el ejercicio 1.

5. Al igual que en el ejercicio anterior, es posible enriquecer la Estructura de Datos con atributos que almacenen la altura del árbol o su fan-out.

   a) Añada un atributo `height` que almacene la altura del árbol. ¿En qué clase lo añadiría? Realice los apartados b y c del ejercicio anterior sobre este atributo, de forma que el método `getHeight()` consista, simplemente, en devolver el valor de ese atributo.
   b) Añada un atributo `fanout` que almacene el fanout del árbol. ¿En qué clase lo añadiría? Realice los apartados b y c del ejercicio anterior sobre este atributo, de forma que el método `getFanOut()` consista, simplemente, en devolver el valor de ese atributo.

6. Dado que los árboles binarios sólo tienen dos hijos y los árboles generales pueden tener cualquier número de hijos, es posible implementar los árboles binarios utilizando un árbol general como estructura de datos.

   Implemente una clase `BTreeGT` que extienda `Collection` e implemente el interfaz `BTreeIF` empleando un árbol general como estructura de datos en lugar de los atributos añadidos por las clases `Tree` y `BTree`. Utilice los métodos de `GTree` para programar los de `BTree`.

7. Aunque en un principio puede resultar chocante, es posible representar un árbol general empleando un árbol binario como estructura de datos. Para ello hay que realizar la siguiente transformación:

   - El hijo izquierdo de un árbol apunta a su primer hijo.
   - El hijo derecho de un árbol apunta a su siguiente hermano.

   Implemente una clase `GTreeBT` que extienda `Collection` e implemente el interfaz `GTreeIF` empleando un árbol binario como estructura de datos en lugar de los atributos añadidos por las clases `Tree` y `GTree`. Utilice los métodos de `BTree` para programar los de `Gtree`.

   A continuación se puede ver una imagen con un ejemplo de representación de un árbol general mediante un árbol binario. Los enlaces en negro representan una relación padre-hijo, mientras que los enlaces en azul representan una relación entre hermanos.

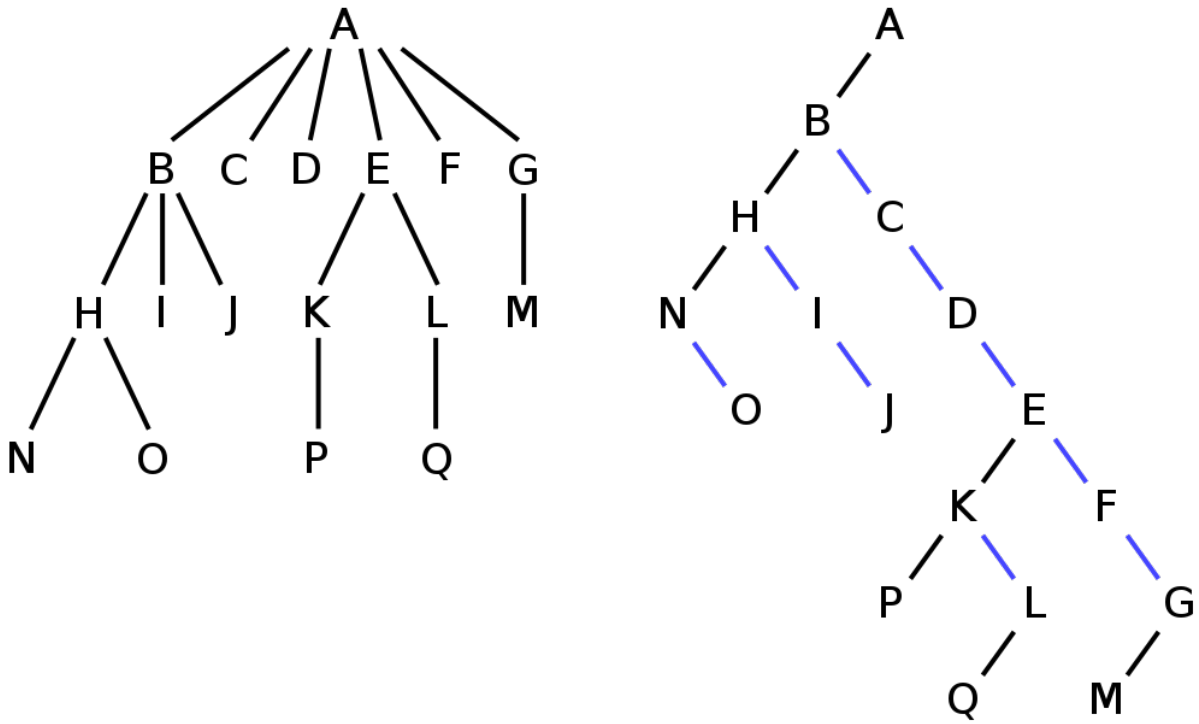

Imagen 1: Imagen extraída de Wikipedia. De CyHawk - Trabajo propio, Dominio público, https://commons.wikimedia.org/w/index.php?curid=4657190

Estrategias de Programación y Estructuras de Datos

Tema 7: Árboles de búsqueda binaria

Ejercicios Propuestos

1. Programe dos funciones que devuelvan el elemento máximo y mínimo de un árbol binario de búsqueda de enteros sin que se recorra todos los nodos necesariamente. Calcule el coste computacional de ambas funciones y compárelo con el coste computacional de las funciones análogas para árboles binarios (*BTree*). Razone en qué casos sería necesario recorrer todos los nodos del árbol.

2. Programe una función que reciba un árbol binario de búsqueda de enteros y dos enteros v_1 y v_2, y devuelva la lista de valores v del árbol tales que $v_1 \leq v \leq v_2$ sin que se recorra todos los nodos necesariamente. Calcule el coste computacional de la función.

3. Programe una función que reciba un árbol binario (*BTree*) de enteros y decida si es un árbol binario de búsqueda. Calcule el coste computacional de la función.

4. ¿Es posible generar unívocamente un árbol binario de búsqueda a partir de su preorden? ¿Y de su inorden? ¿Y de su postorden? Justifique la respuesta. Para los casos afirmativos: diseñe un método de generación del árbol cuya entrada sea el recorrido representado mediante una lista y calcule su coste computacional.

5. Programe una función que reciba un árbol binario de búsqueda y decida si está equilibrado en altura. Calcule el coste computacional de la función.

6. Dada la siguiente secuencia: 7, 3, 9, 2, 1, 6, 5, 10, 4, 11, 12, 13.

- **a.** Dibuje paso a paso el árbol binario de búsqueda desbalanceado resultante tras insertar consecutivamente los números de la secuencia.
- **b.** Dibuje paso a paso el árbol AVL de enteros resultante tras insertar consecutivamente los números de la secuencia. Indique para qué números se ha requerido aplicar alguna rotación.

7. Escriba una secuencia de números enteros de manera que al insertarlos consecutivamente en un árbol AVL vacío se realicen al menos dos rotaciones dobles. Dibuje dicho árbol AVL y el árbol binario de búsqueda desbalanceado resultante tras insertar consecutivamente los números de la secuencia.