Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2011

Práctico 1: Tipos concretos y Tipos Abstractos de Datos

Docentes: Daniel Fridlender, Renato Cherini, Diego Lis, y Juan Cruz Rodriguez

1. Escribí un programa que inicialice cada componente de un arreglo a de tamaño N con el valor 0. Especificá su comportamiento dando una pre y postcondición, y un invariante del ciclo.

2. Analizá el siguiente programa y especificá su comportamiento dando una pre y postcondición, y un invariante del ciclo:

   var a : **array**$[1..N]$ **of nat**
   var n, m : **nat**

   $\{Pre:\ ?\ \}$
   $n, m := 1, 0;$
   do $n \leq N$
   $\quad \{Inv:\ ?\ \}$
   $\quad$ **if** $a[n] \leq m$ **then skip**;
   $\quad$ **else** $m := a[n]$;
   $\quad n := n + 1;$
   od
   $\{Post:\ ?\ \}$

3. Completá la precondición para que el siguiente programa no dé lugar a un error en *tiempo de ejecución*:

   var a : **array**$[1..N]$ **of nat**
   con X, I, M : **nat**

   $\{Pre:\ ?\ \}$
   for $i := I$ **to** M **do**
   $\quad \{Inv:\ \langle \forall j\ :\ I \leq j < i\ :\ a[j] = X \rangle\ \}$
   $\quad a[i] := X;$
   $\{Post:\ \langle \forall j\ :\ I \leq j \leq M\ :\ a[j] = X \rangle\ \}$

4. Dado el tipo

   type *person* = **tuple**
   $\quad$ *name* : **string**
   $\quad$ *age* : **nat**
   $\quad$ *weight* : **nat**

   escribí un programa que calcule la edad y peso promedio de un arreglo a : **array**$[1..N]$ **of** *person*.

5. Traducí el siguiente programa a su *equivalente* usando el comando **do** :

   for $i :=$ *fact*(5) **to** *fib*(30) **do** C **od**

   Luego, suponiendo que *fact*(n) y *fib*(n) son funciones que calculan el factorial de n, y el n-ésimo número de Fibonacci respectivamente:

   a) Escribí un programa en Haskell que evalúe *fib*(n). Probá ejecutarlo para $n = 3, 6, 10, 15, 20, 25, 30$.

   b) ¿Cúantas veces se evalúa *fact*(5)? ¿Cúantas veces *fib*(30)?

   c) ¿Cómo se puede realizar una diferente traducción del **for** para que evalúe una única vez *fact*(5) y *fib*(30)?

6. Dada una lista de enteros xs, escribí un programa que calcule, utilizando la notacion **for** $e \in xs$ **do** C **od** , la suma de xs.

7. Dado a : **array**$[M..N]$ **of int**, escribí un programa que calcule, utilizando la notacion **for** $e \in a$ **do** C **od** , la suma de a. Comparalo con el programa que se obtiene utilizando la notación **for** $i := M$ **to** N **do** C **od** .

8. Completá el siguiente programa para que guarde en la variable xs la lista XS de la cual se han eliminado todas las ocurrencias del valor X.

   var $xs, ys : [A]$
   var $x : A$
   con $XS : [A]$
   con $X : A$

   $xs := [\,]$;
   $ys := XS$;
   do $ys \neq [\,]$
   $x := head(ys)$;
   ...
   od

   ¿Cómo puede especificarse formalmente este programa?

9. El programa del ejercicio 2 computa el máximo valor de un arreglo. Modificalo para que compute *al mismo tiempo* el valor mínimo.

10. Analizá el siguiente programa y especificá su comportamiento dando una pre y postcondición:

    var a : **array**$[1..N, 1..M]$ **of int**
    var i, j : **nat**
    con K : **int**

    {*Pre:* ? }
    for $i = 1$ **to** N **do**
    for $j = 1$ **to** M **do**
    $a[i, j] := a[i, j] * K$
    od
    od
    {*Post:* ? }

    (Más difícil) Dá un invariante para cada ciclo.

11. Escribí un programa que compute la suma de dos matrices a, b : **array**$[M..N, P..Q]$ **of int**.

12. Escribí un programa que compute la multiplicación de una matriz a : **array**$[1..M, 1..N]$ **of int** por un vector v : **array**$[1..N]$ **of int**. Luego modificalo para que el programa compute la multiplicación (general) de matrices a : **array**$[1..M, 1..P]$ y b : **array**$[1..P, 1..N]$.

    Ayuda: En el producto ab de matrices a de tamaño $m \times p$ y b de tamaño $p \times n$, la entrada i, j es de la forma:

    $$(ab)_{ij} = \Sigma_{k=1}^{p} a_{ik} * b_{kj}$$

13. Escribí un programa que dado dos punteros p, q : **pointer to int**, intercambie los valores referidos (notá que no es lo mismo que intercambiar sus valores).

14. Escribí un programa similar al del ejercicio 2, pero que devuelva la posición (relativa) del máximo valor en lugar del valor mismo. ¿Cómo se puede devolver la *posición absoluta* (en la memoria)? Notá que para esto último es necesario apelar a operaciones que están fuera del formalismo utilizado en clase.

15. En muchos lenguajes de programación un arreglo a : **array**$[1..N]$ **of** A se implementa como una sucesión de N celdas de memoria de tipo A, y a un puntero a la primera de ellas. De esta manera, la expresión usual para referenciar la celda i de un arreglo, $a[i]$, es una abreviación del dereferenciamiento $*(a+i-1)$. En este ejercicio trabajaremos según este modelo que se escapa de lo admitido por el formalismo que utilizamos normalmente. Ademas supondremos que existe una operación $\&v$ que devuelve la dirección de memoria de la celda utilizada para almacenar v.

    - Considerá el siguiente programa:

      var a, b : **array**$[1..2]$ **of nat**
      var n, m : **nat**

      $n := 5;$
      $m := 4;$
      $b[1] := \&n;$
      $b[2] := m;$
      $a[1] := \&a[2];$
      $a[2] := \&b[1];$

      ¿Cuál es el valor final de $*(*a[1] + 1)$?

    - En la siguiente porción de código, tratamos a los *strings* como arreglos de caracteres:

      var s : **array**$[1..6]$ **of char**
      $\ldots$
      $s :=$ "$qwerty$";
      $p := s + 2;$
      $r := *p;$
      $*p := *s;$
      $*s := r;$

      Inferí el tipo de las variables p y r. ¿Cuál es el valor final del string que comienza en s?

16. El tipo de los valores de verdad es muy elemental pero no siempre es incluido como tipo concreto en los lenguajes de programación. Especificá el TAD *booleanos* incluyendo constructores (*verdadero* y *falso*), las operaciones usuales $(\wedge, \vee, \Rightarrow, \equiv, \neg)$, y las ecuaciones que las describen.

17. Muchos lenguajes de programación actual no incluyen el tipo *lista* como tipo concreto, y por lo tanto es necesario definirlo como un TAD. Especificá el TAD *lista* de valores de tipo A a partir de los constructores usuales:

    TAD $lista_A$
    constructores
    $[\,]$: $lista_A$
    $\triangleright$: $A \times lista_A \rightarrow lista_A$

    Las operaciones a incluir son: $\triangleleft$ para insertar un elemento por detrás, *tamaño* para observar la longitud de la lista, *cabeza* para observar el primer elemento de la lista, *cola* para eliminar el primer elemento de la lista, *es_vacia* para observar si la lista es vacia, *concat* para concatenar dos listas, y . para observar el elemento que se encuentra en cierta posición dada.

    Tené en cuenta que existen ciertas restricciones para la aplicación de algunas operaciones.

18. El TAD *cola* es ampliamente usado para resolver una gran variedad de problemas. Este uso se puede extender aún mas si se incluyen operaciones para agregar, observar y eliminar elementos por ambos extremos. Una variante con estas características se denomina *cola doble* o *bicola*. Especificá este TAD que además de las 5 operaciones habituales del TAD *cola*, incluye operaciones para: devolver el último elemento, eliminar el último elemento, y agregar un elemento al comienzo de la bicola. Detallá los constructores, operaciones y ecuaciones necesarias.

19. Se quiere diseñar un sistema para montar en los colectivos del sistema público de transporte de alguna ciudad, que lleve la cuenta de la cantidad de pasajes emitidos y la cantidad de monedas utilizadas para abonarlos. Se cuenta con dos dipositivos: una *tiqueteadora* que emite pasajes y un *monedero* que almacena monedas de 10, 25 y 50 centavos. El chofer opera ambos dispositivos de manera manual: introduciendo cada moneda de manera clasificada según su valor, y pulsando un botón para emitir el tique. Se quiere resolver dos problemas: (1) verificar al terminar cada ronda que el monto de dinero acumulado en el monedero se corresponde con la cantidad de tickets emitidos; y (2) emitir un tique cuando el monto introducido es suficiente (habiendo introducido las monedas utilizadas para pagar y posiblemente habiendo sacado monedas para dar vuelto).

    a) Especificá el monedero de la siguiente manera. Tendrá como constructores uno para crear el monedero vacío, y tres que introducen una moneda de cada tipo. Además dispondrá de operaciones para: devolver el número de monedas dentro del monedero para cada tipo, y sacar una moneda para cada tipo. Describí el comportamiento incluyendo las ecuaciones que sean necesarias, asumiendo que el monedero tiene capacidad infinita y que las operaciones para sacar monedas sólo pueden llevarse a cabo cuando existen monedas del tipo adecuado.

    b) Escribí un programa que resuelva cada uno de los problemas (1) y (2), asumiendo que el valor de un pasaje es N.

    Ayuda: Para llevar la cuenta de los tiques emitidos podes utilizar el TAD contador visto en clase.

20. Implementá el TAD *booleanos* del ejercicio 16 utilizando un número natural.

21. Estamos programando un sistema para la manipulación algebraica de expresiones. Nos interesa en particular diseñar un TAD para la manipulación de *polinomios con coeficientes enteros*. Los constructores del tipo son los siguientes:

    TAD *polinomio*
    constructores
    $0 : polinomio$
    $\_\, x^{-} + : \mathbf{nat} \times \mathbf{nat} \times polinomio \to polinomio$

    El constructor *0* crea el polinomio cuyos coeficientes son todos ceros. El constructor $n\, x^k +$ agrega el factor con coeficiente n y grado k. Notá que no hay restricciones en la forma de construir un polinomio, por lo que puede aparecer mas de un factor con mismo grado.

    a) Especificá las operaciones: evaluar en X, devolver la suma de todos los coeficientes de los factores de grado k, eliminar todos los factores con grado k.

    b) Luego escribí un programa que *normalice* un polinomio, es decir, devuelva un polinomio equivalente pero con, a lo sumo, un factor de cada grado.

    Ayuda: Puede ser necesario introducir operaciones auxiliares para programar este procedimiento.

    c) Implementá el TAD *polinomio* utilizando el TAD *lista* especificado en el ejercicio 17.

22. Implementá el TAD *bicola* del ejercicio 18 utilizando la idea de *arreglo circular*.

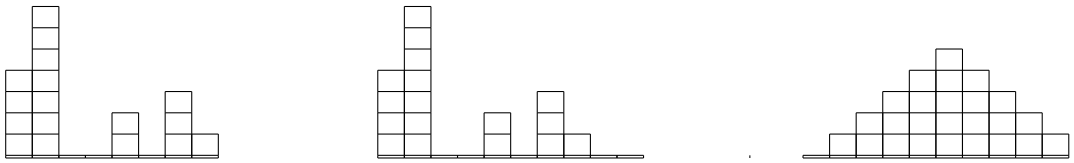
23. Considerá un TAD muy parecido al tipo *pila*, que además de las 5 operaciones características, dispone una operación que invierte el orden de los elementos en la pila.

    a) Especificá el TAD *pila reversible* que tienen los mismos constructores y operaciones que el TAD *pila*, y además la operación *invertir*.

    Ayuda: Puede ser conveniente introducir operaciones auxiliares para caracterizar la operación *invertir*.

    b) Implementá este TAD utilizando arreglos. ¿Se te ocurre una implementación de manera que todas las operaciones sean de orden constante?

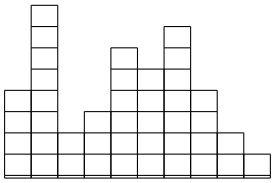
24. Los histogramas son gráficos en forma de barras verticales como, por ejemplo, los siguientes tres:



Para crear tales histogramas, se pueden utilizar tres constructores: uno para construir el histograma inicial (consistente de 1 sola columna vacía), otro para para agregar una nueva columna vacía a (la derecha de) un histograma existente, y una tercera para agregar un cuadrado a la última columna (la que está más a la derecha).

El primer histograma de ejemplo se construye a partir del histograma inicial, utilizando 4 veces el constructor que agrega un cuadrado, una vez el que agrega una columna, 7 veces el que agrega un cuadrado, 3 veces el que agrega una columna, 2 veces el que agrega un cuadrado, 2 veces el que agrega una columna, 3 veces el que agrega un cuadrado, una vez el que agrega una columna y finalmente una vez el que agrega un cuadrado.

a) Especificá el TAD *hist*, con sus 3 constructores, las ecuaciones entre ellos que sean necesarias y al menos tres operaciones: una para observar el número de columnas, otra para observar el número de cuadrados en una columna particular, y finalmente otra que aplicada a histogramas con igual número de columnas devuelva un nuevo histograma con la suma de cuadrados columna a columna. Por ejemplo, si se suman el segundo y tercer histogramas mostrados más arriba, se obtiene:



Ayuda: Es posible que necesites definir operaciones auxiliares para enunciar las ecuaciones que describen las operaciones mencionadas.

b) Implementá el TAD *hist* de la manera que creas adecuada. Justificá tu elección mencionando sus ventajas frente a otras posibles implementaciones.

25. Implementá el TAD $listas_A$ del ejercicio 17 con punteros, utilizando la estructura de datos de *lista ligada*. Considerá diferentes variaciones de esta estructura que permitan implementar las operaciones de manera mas o menos eficiente.

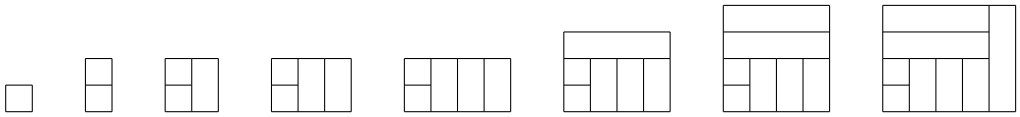
26. La siguiente es una especificación del TAD *conjunto* de elementos del tipo A:

TAD $conjunto_A$
constructores
$\emptyset : conjunto_A$
$\{\_\} : A \to conjunto_A$
$\cup : conjunto_A \times conjunto_A \to conjunto_A$
operaciones
$\in : A \times conjunto_A \to booleano$
$\cap : conjunto_A \times conjunto_A \to conjunto_A$
ecuaciones
$$e \in \emptyset = falso$$
$$e \in \{e'\} = (e = e')$$
$$e \in C \cup C' = (e \in C) \vee (e \in C)$$

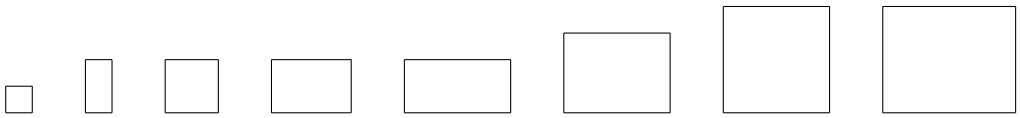
$$
\begin{aligned}
\emptyset \cap C &= C \\
\{e\} \cap C &= (e \in C \rightarrow \{e\} \\
&\quad \Box \neg e \in C \rightarrow \emptyset \\
&\quad) \\
(C'' \cup C') \cap C &= (C'' \cap C) \cup (C' \cap C)
\end{aligned}
$$

Implementá el TAD *conjunto* de enteros utilizando como estructura de datos listas (abstractas), manteniendo sus elementos ordenados y sin repetición.

27. El TAD *rect* de los rectángulos cuyos lados tienen longitud entera, tiene tres constructores: uno para crear un rectángulo de 1×1; otro para agregar una fila (de alto 1) a un rectángulo ya creado; y el último para agregar una columna (de ancho 1) a un rectángulo ya creado. Por ejemplo, la siguiente secuencia de rectángulos



se crea a partir del rectángulo inicial (de 1x1) uno de 4x5 agregando sucesivamente una fila, una columna, una columna, una columna, una fila, una fila y una columna. Las líneas internas fueron dibujadas sólo para ilustrar lo que fue ocurriendo, en realidad lo que importa son los rectángulos:



Observá que hay diferentes formas de construir un mismo rectángulo. Por ejemplo, la secuencia que comienza con el rectángulo inicial y agrega sucesivamente una columna, una columna, una fila, una fila, una columna, una fila y una columna construye el mismo rectángulo que la secuencia de arriba.

   a) Especificá el TAD *rect*, con sus 3 constructores, las ecuaciones entre ellos que sean necesarias y al menos cuatro operaciones que se aplican sobre un rectángulo y devuelven: la altura, el ancho, su perímetro, y su superficie respectivamente.

   b) Implementar el TAD *rect* de la manera que creas conveniente.

28. Sea *bintree* una implementación (cualquiera) del TAD *arbol binario*. Escribí programas para cada uno de los siguientes problemas:

   a) **fun** *height*($t : bintree$) **ret** $h : Nat$, que devuelve la *altura* del arbol t.

   b) **fun** *ancestor*($t : bintree, e_1 : elem, e_2 : elem$) **ret** $res : Bool$, que devuelve *true* si y solo si e_1 es *ancestro* de e_2 en el árbol t.

   c) **fun** *descendant*($t : bintree, e_1 : elem, e_2 : elem$) **ret** $res : Bool$, que devuelve *true* si y solo si e_1 es *descendiente* de e_2 en el árbol t.

   Ayuda: Las soluciones recursivas son más sencillas que las iterativas.

29. Implementá el TAD *arbol binario* utilizando la siguiente estructura de datos:

```
type tnode = tuple
      left : pointer to tnode
      value : elem
      right : pointer to ntode

type binTree = pointer to tnode
```

con la que se representa un *árbol binario* como un puntero a un nodo, que a su vez posee punteros a los subárboles izquierdo y derecho (*left* y *right* respectivamente).

30. Investigá las diferentes maneras de iterar sobre un árbol binario: *in-order*, *pre-order*, *post-order*. Escribí un procedimiento para cada una de estas formas, suponiendo que se quiere modificar el valor de cada nodo del árbol con la función **fun** *process*$(e : elem)$ **ret** $f : elem$.

31. Con las valores $\{1, 4, 5, 10, 16, 17, 21\}$ construí *árboles binarios de búsqueda* de alturas 2, 3, 4, 5 y 6.

32. Dado un ABB con cuyos nodos poseen valores entre 1 y 1000, suponé que te interesa encontrar el número 363. ¿Cuáles de las siguientes secuencias no puede ser una secuencia de nodos examinados según el algoritmo de búsqueda?

    a) $2, 252, 401, 398, 330, 344, 397, 363$.

    b) $924, 220, 911, 244, 898, 258, 362, 363$.

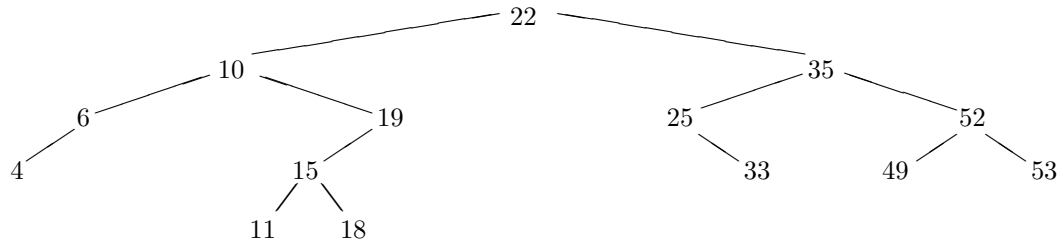
    c) $925, 202, 911, 240, 912, 245, 363$.

    d) $2, 399, 387, 219, 266, 382, 381, 278, 363$.

    e) $935, 278, 347, 621, 299, 392, 358, 363$.

33. Dada la secuencia de números $23, 35, 49, 51, 41, 25, 50, 43, 55, 15, 47$ y 37, determiná el ABB resultante de realizar las inserciones de dichos números exactamente en ese orden a partir del ABB vacío.

34. Determiná una secuencia de inserciones que dé lugar al siguiente ABB:



35. Reimplementá la función *search* de un ABB sin utilizar recursión.

36. Un *multiconjunto* es una colección desordenada de valores en la que puede haber duplicados, que se puede especificar como sigue:

TAD $multiconjunto_A$
constructores
$\emptyset : multiconjunto_A$
$ins : A \times multiconjunto_A \to multiconjunto_A$
operaciones
$mult : A \times multiconjunto_A \to Nat$
$elim : A \times multiconjunto_A \to multiconjunto_A$
ecuaciones

$$
\begin{aligned}
mult(e, \emptyset) &= 0 \\
mult(e, ins(e', M)) &= (e = e' \to 1 + mult(e, M) \\
&\quad \square \neg e = e' \to mult(e, M) \\
&\quad)
\end{aligned}
$$

$$
\begin{aligned}
elim(e, \emptyset) &= \emptyset \\
elim(e, ins(e', M)) &= (e = e' \to elim(e, M) \\
&\quad \square \neg e = e' \to add(e', elim(e, M)) \\
&\quad)
\end{aligned}
$$

    a) Extendé la especificación agregando tres operaciones: una para observar si un multiconjunto es vacío; otra para unir dos multiconjuntos; y otra que dados dos multiconjuntos elimina del primero la cantidad de ocurrencias en el segundo de cierto valor v (esto es, la resta de multiconjuntos).

b) Implementá este TAD utilizando como estructura de datos un árbol binario de búsqueda (abstracto).

Ayuda: Tené en cuenta que con el invariante de ordenación de un árbol binario de búsqueda no es posible tener dos nodos con un mismo valor.

37. Investigá las tácticas de búsqueda *Depth-first search* (DFS) y *Breadth-first search* (BFS). Extendé la especificación del TAD *bintree* con una operación para cada una de las búsquedas. Luego implementálas utilizando la estructura de datos del ejercicio 29.

38. Reimplementá el TAD *multiconjunto* del ejercicio 36 utilizando como estructura de datos un *arbol binario de búsqueda* implementado con punteros.

39. Un *heap binario*, además de la propiedad de ordenación, debe cumplir la propiedad de *forma*: el árbol debe ser (casi) completo, esto es, los nodos de todos los niveles, excepto posiblemente el último, tienen tanto hijo izquierdo como derecho. En el último nivel, si un nodo no tiene hijos, entonces todos sus *hermanos* a la derecha no pueden tener hijos.

    Teniendo en cuenta estas dos propiedades características de un *heap*:

    a) ¿Cuál es el número máximo y mínimo de nodos que puede tener un *heap binario* de altura h?

    b) ¿Dónde está ubicado el elemento mínimo de un *heap*? ¿Y el máximo?

    c) ¿Un arreglo ordenado de forma descendente implementa un *heap*? ¿Un *heap* implementado con un arreglo, siempre da lugar a un arreglo ordenado de manera descendente?

    d) El arreglo $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ ¿es un *heap*?

40. Cuando implementamos un *heap binario* con un arreglo A, el procedimiento $sink(A, i)$ "hunde" el elemento $A[i]$ a través de sus hijos $A[2 * i]$ y $A[2 * i + 1]$ de manera de que la estructura resultante mantenga la propiedad de ordenación del *heap*. Representá gráficamente la evolución, paso a paso, del *heap* al ejecutar $sink(A, 3)$, cuando $A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]$.

41. ¿Qué sucede si llamamos a $sink(A, i)$ cuando $A[i]$ es mas grande que sus hijos? ¿Y cuando $i > \#A/2$, donde $\#A$ es el tamaño de A?

42. Considerá una *cola de prioridades* Q representada con el *heap binario* $[15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]$. Graficá el *heap* paso a paso cuando:

    a) se ejecuta *dequeue(Q)*,

    b) se ejecuta *enqueue(Q,10)*.

43. Un *heap n-ario* es un árbol cuyos nodos tienen n hijos que mantiene las propiedad de ordenación y forma del *heap binario*.

    a) ¿Cómo se representa un *heap n-ario* con un arreglo?

    b) ¿Cuál es la altura de un *heap n-ario* con m nodos?

    c) Implementá los procedimientos *sink* y *float* (y posiblemente sus procedimientos auxiliares) para esta clase de *heaps*.

44. Un *árbol AVL* es un ABB auto balanceado. En un AVL las alturas de los subárboles de cierto nodo difieren a lo sumo en uno. Manteniendo este invariante (extra) se obtiene buena eficiencia en las operaciones asociadas, a costa de que la inserción y el borrado de nodos puede requerir rebalancear el árbol, aplicando una o mas operaciones de *rotación*.

    a) Investigá como funcionan los AVL.

    b) Implementa las operaciones de *search*, *insert* y *delete* sobre un AVL implementado con punteros.

    Ayuda: Puede resultar útil pensar primero cómo resolver el problema sobre arboles abstractos. Además una buena modularización de estas operaciones *abstractas* y otras auxiliares como las de rotación puede resultar de mucha ayuda.