Estructuras de Datos y Algoritmos
Grado en Ingeniería Informática
Universidad de Zaragoza – curso 2024-2025

Sesión de problemas 5
Árboles n-arios

Objetivos

- Implementación (en pseudocódigo) de operaciones con árboles n-arios.

1. Implementación de función `contarAridadPar`

Dado un árbol n-ario basado en la representación primogénito-siguiente hermano:

```
tipos árbol = ↑nodo;
       nodo = registro
                   dato: elemento;
                   primogénito, sigHermano: árbol
              freg
```

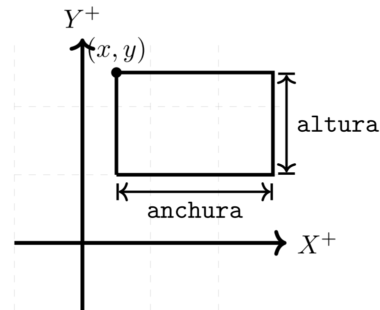
Se pide **implementar una función `contarAridadPar` que calcule y devuelva cuántos nodos tienen un número par de hijos** (asume que si un nodo tiene 0 hijos, entonces no tiene un número par de hijos). **La representación no debe almacenar el número de hijos de cada nodo**.

2. TAD `colección` de rectángulos

Se desea almacenar una colección de datos rectángulo que representen rectángulos de áreas no nulas y no solapadas (no se considerarán solapados si sólo se tocan sus lados), y donde cada uno de ellos se caracteriza por las coordenadas de su esquina superior izquierda, su altura y su anchura (véase dibujo esquemático y estructura de datos).

Distribuido bajo licencia CC BY-NC-SA 4.0. (© R. J. Rodríguez)
https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es

```
tipo rectángulo = registro
                      x, y: entero;
                      altura: entero;
                      anchura: entero;
                  freg
```



La colección debe almacenar los rectángulos de forma que se pueda comprobar y gestionar fácilmente qué rectángulos están incluidos dentro de otros. Por ello, se decide implementar la colección en base a los árboles n-arios y su implementación denominada "primogénito-siguiente hermano", estudiada en la asignatura, y de forma que:

- Cada nodo del árbol n-ario contendrá los datos de un rectángulo, y
- Los hijos de ese nodo deberán contener los datos de los rectángulos de la colección que estén "dentro" de ese rectángulo (es decir, si el rectángulo C está dentro del rectángulo B, y B está dentro del rectángulo A, entonces B será hijo de A, y C será hijo únicamente de B).

No se permitirá almacenar en la colección rectángulos repetidos ni rectángulos que se solapen con otros que ya estén en la colección.

Se pide:

a) **Define en pseudocódigo los tipos de datos necesarios** para definir la estructura de datos para la colección de rectángulos (TAD `colección`), **y todas las estructuras de datos que consideres necesarias**.

b) Utilizando las estructuras de datos propuestas en el apartado anterior, **implementa la operación `añadir` que, dado un rectángulo R, lo añada a la colección C indicando si ha sido posible añadirlo o no**. Su signatura será:

```
procedimiento añadir (ent R: rectángulo; e/s C: colección; sal añadido: booleano)
```

Para la implementación de la operación **`añadir`** se pueden considerar ya disponibles las siguientes operaciones para rectángulos:

```
función separados (A, B: rectángulo) devuelve booleano
{Devuelve verdad si y sólo si el rectángulo A y el rectángulo B no tienen ningún punto
interior en común.}
función  dentro (A, B: rectángulo) devuelve booleano
{Devolverá verdad si y sólo si el rectángulo A está dentro del rectángulo B, es decir,
todo punto interior de A es un punto interior de B.}
función  identidad (A, B: rectángulo) devuelve booleano
{Devuelve verdad si y sólo si el rectángulo A tiene las mismas coordenadas, altura y
anchura que B.}
función  solapados (A, B: rectángulo) devuelve booleano
{Devuelve verdad si y sólo si el rectángulo A y el rectángulo B tienen algún punto interior
en común, sin ser idénticos ni estar uno dentro del otro. Es decir: solapados(A,B)=
not(separados(A,B)) and not(identidad(A,B)) and not(dentro(A,B)) and not(dentro(B,A))}
```

© CC BY-NC-SA 4.0 – R. J. Rodríguez (DIIS, Universidad de Zaragoza)

3. Implementación del procedimiento `equipar`

Se quiere utilizar un árbol n-ario de equipamiento para organizar la colección de ítems que un jugador puede comprar para equipar a su avatar, y para gestionar el equipamiento ya adquirido y disponible para ser usado por el avatar (véase ejemplo en la figura inferior). Se podrá equipar con un ítem a un avatar si y sólo si, ese ítem aparece en el árbol de equipamiento permitido para el avatar, si el avatar aún no lo tiene disponible (ya adquirido), y si el dinero que se aporta para la compra es suficiente para: pagar el precio por dicho ítem, y pagar todos aquellos ítems que aún no se tengan disponibles y que se encuentren en el camino del árbol n-ario desde su raíz a dicho ítem (y que también deben comprarse para poder adquirirlo).

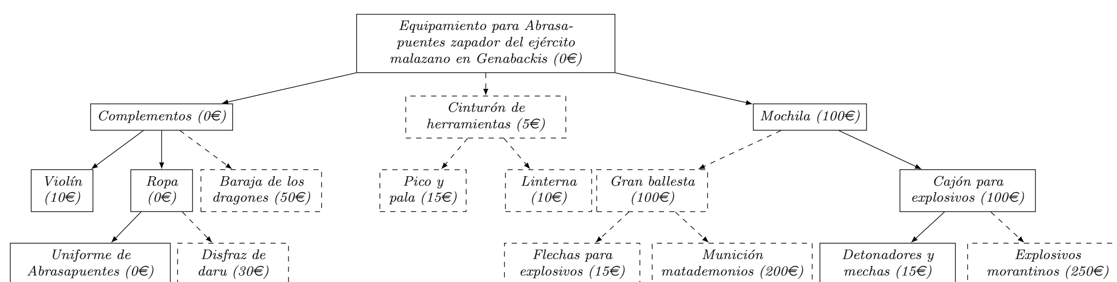
Se pide **implementar el procedimiento `equipar`** que, dado el árbol de equipamiento del avatar, una determinada aportación de dinero, y la cadena que identifica el ítem que se quiere adquirir para el avatar, **comprobará si es posible adquirir dicho ítem para el avatar, y en caso de ser posible lo adquirirá, adquiriendo además todos los ítems necesarios para ello, actualizando adecuadamente el árbol de equipamiento, informando del dinero sobrante y de si la compra se ha podido realizar o no**.

```
procedimiento equipar(e/s a: arbEquipamiento; ent IDitem: cadena;
                      e/s dinero: natural; sal adquirido: booleano)
```

Con los siguientes tipos de datos que definen el árbol n-ario de equipamiento:

```
tipos
    ítem = registro
              IDnombre: cadena;
              descripción: cadena;
              precio: natural;
              disponible: booleano
              {verdad si el jugador ha adquirido el ítem para su avatar}
           freg;
    arbEquipamiento = ↑nodo;
    nodo = registro
              dato: ítem;
              primogénito: arbEquipamiento;
              sigHermano: arbEquipamiento
           freg
```

© CC BY-NC-SA 4.0 – R. J. Rodríguez (DIIS, Universidad de Zaragoza)



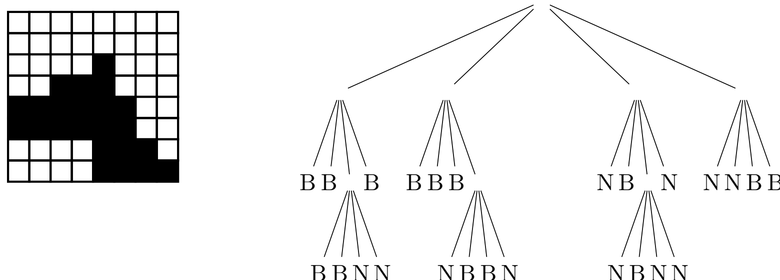
Nodos y líneas punteadas representan equipamiento posible pero no disponible hasta que sea adquirido.
Según el árbol de equipamiento de la figura:

- No es posible adquirir el ítem "espada Azar", ya que no aparece en el árbol.
- No es posible adquirir el ítem "Violín" porque ya está disponible.
- Con 250€ es posible adquirir los "explosivos morantianos" sin que sobre dinero de la compra.
- Con 250€ no es posible adquirir la "munición matademonios" porque además es necesario comprar la "Gran ballesta" y ambas cosas suman un precio total de 300€.
- Con 250€ es posible adquirir "pico y pala", comprando además el "cinturón de herramientas", y sobrarían 5€.

4. Quad-trees

La estructura de datos *quad-tree* se usa en informática gráfica para representar figuras planas en blanco y negro. Se trata de un árbol en el cuál cada nodo, o bien tiene exactamente cuatro hijos, o bien es una hoja. En este último caso, puede ser una hoja blanca o una hoja negra.

El árbol asociado a una figura dibujada dentro de un plano (que, para simplificar, podemos suponer un cuadrado de lado 2^k) se construye de la forma siguiente: se subdivide el plano en cuatro cuadrantes; los cuadrantes que estén completamente dentro de la figura corresponderán a hojas negras, los que estén completamente fuera de la región, a hojas blancas y los que estén parcialmente dentro y parcialmente fuera, a nodos internos; para estos últimos se aplica recursivamente el mismo algoritmo. Como ejemplo, véase la siguiente figura en blanco y negro y su árbol asociado (considerando los cuadrantes en el sentido de las agujas del reloj, a partir del cuadrante superior izquierdo).



Se debe **proponer una representación en memoria para los *quad-trees* e implementar las siguientes operaciones**:

© CC BY-NC-SA 4.0 – R. J. Rodríguez (DIIS, Universidad de Zaragoza)

```
procedimiento creaQuadTree(sal qt: quadtree; ent f: figura)
{ Construye en qt el quad-tree asociado a la figura f. }

procedimiento creaFigura(sal f: figura; ent qt: quadtree)
{ Reconstruye en f la figura representada en el quad-tree qt. }
```

Donde el tipo figura es el siguiente:

```
constante N = ... {N = 2k, para alguna constante k > 0}
tipo figura = vector[1..N, 1..N] de booleano {verdad representa negro}
```

© CC BY-NC-SA 4.0 – R. J. Rodríguez (DIIS, Universidad de Zaragoza)