BinTree

El tipo de dato ***'a bintree*** definido a continuación puede servir para representar árboles binarios en los que los nodos estén etiquetados con valores de tipo ***'a***

```
type 'a bintree = Empty | BT of 'a bintree * 'a * 'a bintree
```

Así, ***Empty*** podría representar el árbol vacío (sin nodos), y ***BT (l, x, r)*** representaría el árbol que tiene a ***l*** como rama izquierda, ***r*** como rama derecha y la raíz etiquetada con el valor ***x***.

Se trata de implementar un módulo OCaml, ***BinTree***, que reúna esta definición con una colección de valores (casi todos funciones) que permitan realizar las operaciones más habituales con estos árboles. En concreto, se pretende definir todos los valores descritos en la interfaz proporcionada (***binTree.mli***).

Para construir la implementación del módulo, copie la definición anterior en un archivo ***binTree.ml*** (este sería el archivo de implementación del módulo ***BinTree***) y añada todas las definiciones necesarias para satisfacer la interfaz proporcionada.

Es una práctica habitual entre los programadores de OCaml, cuando se implementa un módulo que gira en torno a la definición de un tipo de dato, llamarle a ese tipo simplemente ***t***, ya que el nombre del módulo debería ser ya bastante significativo. Para ello añada, después de la definición de ***bintree***, la siguiente definición, que establece que ***t*** es un alias para ***bintree***.

```
type 'a  t = 'a  bintree
```

Así, desde fuera del módulo, el constructor de tipos ***bintree*** se verá simplemente como ***BinTree.t*** en vez de ***BinTree.bintree***

Finalmente complete la implementación con la definición de todos los valores que se mencionan en la interfaz del módulo.

Para el desarrollo de esta implementación, seguramente sea cómodo utilizar el compilador interactivo ocaml (para hacer pruebas); pero, una vez terminado, debe compilarse con la orden

```
ocamlc -c binTree.mli binTree.ml
```

Esto generará dos archivos (***binTree.cmi*** y ***binTree.cmo***) que son, respectivamente la interfaz y la implementación compiladas del módulo ***BinTree***.

Si quiere utilizar este módulo desde el compilador interactivo ***ocaml*** debe cargarlo con el comando

```
#load "binTree.cmo"
```

Una de las funciones más difíciles de definir de esta implementación es ***breadth*** (la que proporciona el recorrido en anchura de un árbol). Una posible definición sería la siguiente

```
let breadth a =
    let rec aux = function
        [] -> []
      | Empty::t -> aux t
      | BT (l,x,r) :: t -> x :: aux (t @ [l;r])  (* ineficiente *)
    in aux [a]
```

Pero con esta definición, la función puede resultar muy ineficiente con algunos árboles (recuerde cuál es el coste de la concatenación de listas).

Para comprobarlo puede utilizar la función complete_tree definida a continuación para general un árbol binario “completo” con sus nodos numerados “en anchura”

```
let complete_tree n =
    let rec aux i =
        if i > n then Empty
        else BT (aux (2*i), i, aux (2*i+1))
   in aux 1
```

Pruebe la función ***breadth*** definida más arriba con árboles completos de, por ejemplo, 10.000 y 20.000 nodos. Seguramente podrá apreciar que ya tarda bastante. Cambie, por tanto, la definición de ***breadth*** para que no tenga este problema. Con una buena definición, ***breadth***, no debía de tener mayor problema para recorrer árboles mucho mayores (incluso de varios millones de elementos)