Enumerating entries in a functional array

In this short note, we look at the problem of enumerating the entries of a functional array in order. This was the topic of [2015P1Q1 (c)], where a solution with time complexity $O(n \log n)$ was presented, where $n$ is the number of entries in the functional array, which in the worst-case is asymptotically as efficient as retrieving all indices and sorting them. Here, we will look at a solution that does this in $O(n)$ time.

1 Problem formulation and main observation

Consider the following functional array with 15 entries, where $A[i]$ indicates the value of the $i$-th entry in the functional array.

\[
\ell = [\ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots ; \ldots]@ [\ldots ; \ldots ]
\]

Why does this observation hold? Note that the indices of the children of a node with index $x$ at level $i$ is given by $x + 2^{i-1}$ (left child) and $x + 2^i$ (right child). You can justify by looking at the binary representation of $x$. For
example, if \( x = 10 = 1010_2 \), then the path to index 10 is given by 0102, so the path to its left child is 100102 (which adds \( 2^3 \)) and to its right child is 110102 (which adds \( 2^4 \)). More generally for index \( x \) its left child holds index \( x + 2^i \) and its right child holds index \( x + 2^i \) where \( i \) is the depth of the node \( x \).

## 2 Implementation

We are going to break down the implementation in the following:

1. Given the nodes at the current level sorted by index, create a list containing the nodes at the next level sorted by index (get_next_layer).
2. Recursively call the above function to create an enumeration of the elements of the functional array in order of their index (get_in_order).
3. Implement the filter function on the returned list (filter_indices). (what was originally asked in the question)

Before proceeding, try to implement these on your own.

### Implementation of get_next_layer:

(* Some simple functions defined here to avoid clutter. *)

``` Ocaml 
let get_left (Br(_, l, _)) = l;;
let get_right (Br(_, _, r)) = r;;
let get_value (Br(v, _, _)) = v;;

let get_next_layer cur =
  List.filter (fun x -> x <> Lf)
  (* The main part of the implementation is the following line. The filter is needed just in case there are leaf nodes. *)
  (List.map get_left cur) @ (List.map get_right cur));;
```

### Implementation of get_in_order:

``` Ocaml 
let rec get_in_order = function
  | [] -> []
  | rt -> (List.map get_value rt) @ get_in_order (get_next_layer rt);;

get_in_order [root];;
```

### Implementation of filter_indices:

``` Ocaml 
let rec filter_indices f n = function
  | [] -> []
  | x::xs -> if f x then n :: filter_indices f (n+1) xs
    else filter_indices f (n+1) xs;;

filter_indices (fun x -> x mod 3 = 0) 0 [2;3;6;19;21;27;10;11];;
```

## 3 Verifying correctness

We will perform a simple test for enumeration using the function array implementation used in the lecture notes and a functional array with 25 elements.

(* Functional array *)

``` Ocaml 
type 'a btree = Lf | Br of 'a * 'a btree * 'a btree;;
exception Subscript;;

let rec sub = function
  | Lf, _ -> raise Subscript
  | Br (v, t1, t2), i -> v
```
let rec update = function
| Lf, k, w ->
  if k = 1 then Br (w, Lf, Lf)
  else raise Subscript (* Gap in tree *)
| Br (v, t1, t2), k, w ->
  if k = 1 then Br (w, t1, t2)
  else if mod 2 = 0 then Br (v, update (t1, k / 2, w), t2)
  else Br (v, t1, update (t2, k / 2, w));

let root = Lf;;
let root = update (root, 1, 1);;
let root = update (root, 2, 2);;
let root = update (root, 3, 3);;
let root = update (root, 4, 4);;
let root = update (root, 5, 5);;
let root = update (root, 6, 6);;
let root = update (root, 7, 7);;
let root = update (root, 8, 8);;
let root = update (root, 9, 9);;
let root = update (root, 10, 10);;
let root = update (root, 11, 11);;
let root = update (root, 12, 12);;
let root = update (root, 13, 13);;
let root = update (root, 14, 14);;
let root = update (root, 15, 15);;
let root = update (root, 16, 16);;
let root = update (root, 17, 17);;
let root = update (root, 18, 18);;
let root = update (root, 19, 19);;
let root = update (root, 20, 20);;
let root = update (root, 21, 21);;
let root = update (root, 22, 22);;
let root = update (root, 23, 23);;
let root = update (root, 24, 24);;
let root = update (root, 25, 25);;

get_in_order [root];;

(*
- : int list =
[1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19; 20; 21;
22; 23; 24; 25]
*)