OCaml Lab Exercise #3

Torben Amtoft

September 2024

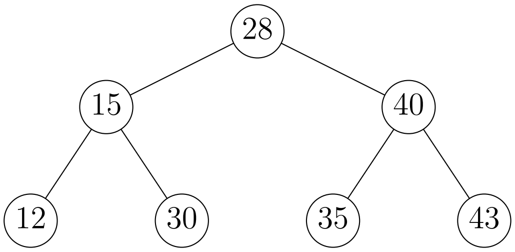
Problem Description

Goal to get some familiarity with basic techniques for tree manipulation in OCaml.

Preliminaries We shall consider **binary trees**, and define the polymorphic data type

```
type 'a bin_tree =
    Empty
  | Node of 'a              (* content   *)
         * 'a bin_tree      (* left tree *)
         * 'a bin_tree      (* right tree *)
```

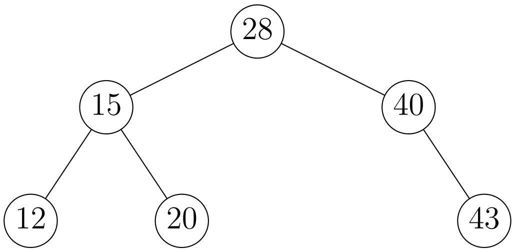
For example, the tree



can be defined by

```
let treeA =
  Node (28,
     Node (15,
        Node (12,Empty,Empty),
        Node (30,Empty,Empty)),
     Node (40,
        Node (35,Empty,Empty),
        Node (43,Empty,Empty)))
```

and the tree



can be defined by

```
let treeB =
  Node (28,
     Node (15,
        Node (12,Empty,Empty),
        Node (20,Empty,Empty)),
     Node (40,
        Empty,
        Node (43,Empty,Empty)))
```

We say that a binary tree is a **search tree** if for all non-empty subtrees:

- all nodes occurring in its left child are less than the root, and
- all nodes occurring in its right child are greater than the root.

For example:

- `treeA` is *not* a search tree since it has a subtree (itself) whose root (28) is less than a node in the left child (30)
- `treeB` *is* a search tree since:
  - all of 15, 12 and 20 are less than 28
  - all of 40, 43 are greater than 28
  - 12 is less than 15
  - 20 is greater than 15
  - 43 is greater than 40.

We say that a binary tree is a **perfect**, with height n, if

- all nodes have either two children or no children, and

- all leaves (that is nodes without children) have distance *n* from the root.

For example:

- `treeA` *is* perfect, with height 2
- `treeB` is *not* perfect, since the node labeled 40 has only one child.

Task 1

Write a function `is_search` that decides if the binary tree gives as input is a search tree. Thus we expect the dialogue

```
# is_search treeA ;;
- : bool = false
# is_search treeB ;;
- : bool = true
```

Observe that a non-empty tree `Node(n,t1,t2)` is a search tree if

- `t1` and `t2` are both search trees, and
- all nodes in `t1` are less than `n`, and all nodes in `t2` are greater than `n`.

This suggests that you will need to express `is_search` in terms of a more general function

```
let rec is_search_with_bound p t =
  match t with
  | Empty -> ...
  | Node(n,t1,t2) ->
         ...
```

where `p` is predicate that all nodes in `t` must satisfy. (At top-level, `p` is always true, but when called on a child, `p` must make appropriate comparisons to the root.)

Task 2

Write a function `is_perfect` which if the binary tree given as input is a perfect tree of height `n` returns `Some n`, and otherwise returns `None`. Thus we expect the dialogue

```
# is_perfect treeA ;;
- : int option = Some 2
# is_perfect treeB ;;
- : int option = None
```

The function `is_perfect` should be recursively defined:

```
let rec is_perfect t =
  match t with
  | Empty -> ...
  | Node(_,t1,t2) ->
     (match (is_perfect t1, is_perfect t2) with
        | (Some h1, Some h2) ->
              ...
```

Task 3

Consider a version of the *fold* template adapted for binary trees:

```
let rec foldt f e t =
   match t with
  | Empty -> e
  | Node (n,left,right) ->
      f n (foldt f e left) (foldt f e right)
```

Write a function `is_perfect'` that behaves as `is_perfect`, but is defined without recursion but using `foldt`:

```
let is_perfect' =
   foldt
      ...
      ...
```

Deliverables

Submit to Canvas a file containing an OCaml program that extends the given file `lab3_starting.ml` with the required function definitions.

You must verify, for example by from the command line typing

```
- #use "lab3.ml";;
```

that your program **is accepted by the OCaml type system**.

You should aim at verifying that you can reproduce the dialogues listed in the question text.

You may also try your functions on the trees given in `lab3_trees.ml`.