U0LPMPL2U : If you have a `fold` defined it becomes pretty easy. Just return the current node if it isn't the one you want to update, otherwise update the node
U4F64AKQV : <@U1G51S63S> What are you using the tree for? This seems like an XY problem to me.
U1G51S63S : I thought to have a fold for reducing a list of values into another list and separated recursive function for updating values in accumulator
U1G51S63S : <@U4F64AKQV>
U1G51S63S : I want to display thing like this
U0LPMPL2U : `List.foldl` is recursion over a list. You could also implement a `Tree.fold`
U1G51S63S : but from API I got plain list of records with LTREE keys (postgresql thing) like `Mahnattan.Bedford-St.Ocean_hill`
U0LPMPL2U : In your case, what your probably want a tree map function. Something like:```
Tree.map (\node -> if node.key == "some.key" then Tree.addChild child node else node)
```

U0LPMPL2U : none of these functions exist though :stuck_out_tongue:
U4F64AKQV : So the issue is parsing the strings to create a tree structure?
U1G51S63S : <@U4F64AKQV> creating a tree structure yeah. this is Elixir code which doing what I need but with dicts and not lists ```      |&gt; Enum.reduce(%{}, fn (area, acc) -&gt;      {name, path} =
      area.path
      |&gt; String.split(".")
      |&gt; List.pop_at(-1)

    node = %{children: %{},
           id: area.id,
           name: area.name}

    case path do
      [] -&gt;
        Map.put_new(acc, name, node)
      path -&gt;
        full_path =
          Enum.intersperse(path, :children) ++ [:children, name]
        put_in(acc, full_path, node)
    end
```

U1G51S63S : <@U0LPMPL2U> I thought reducing is a `n - 1` op, where recursion is here?
U1G51S63S : so basically it's iterating over all items while keeping acc, no?
U1G51S63S : or you mean under hood with head/tail?
U4F64AKQV : So why not use an Elm `Dict` then?
U1G51S63S : btw I can't get you idea with Tree.map :disappointed:
U0LPMPL2U : `List` is a recursive structure, therefore "iterating" is done recursively (although the compiler may turn that back into iteration)
U1G51S63S : <@U4F64AKQV> a I am figured out how to do it with Dict already, just want to try with List :slightly_smiling_face:
U4F64AKQV : The current impl of `foldr` happens to use a for loop. It converts the list to a JS array and iterates over it.
U0LPMPL2U : So if you wanted to update a value in a list, you could do something like this right?
```
updateList : a -&gt; a -&gt; List a -&gt; List a
updateList oldValue newValue list =
  List.map (\val -&gt; if val ==  oldValue then newValue else n) list
```

U1G51S63S : yep, correct. but problem is with nested update
U1G51S63S : so as I thought it should be recursion like a -> b -> insert c to b.children -> b -> a
U0LPMPL2U : Imagine that `Tree.map` visits all children recursively, just like `List.map` can do for a list
U1G51S63S : ok, done :smile:
U0LPMPL2U : This discusses `Tree.map` and an implementation for a binary tree <https://evancz.gitbooks.io/functional-programming-in-elm/recursion/binary-trees.html>
U1G51S63S : > A binary tree is either empty or it is a node with a value and *two subtrees*

U23SA861Y : yes, either one of which could be empty
U0LPMPL2U : Notice that node 9 has two children: 7 and "empty"
U1G51S63S : hmm, yeah. just read again Insert section and got idea. will try to implement it now for my case :thumbsup: