Exercise 4.2 Enumerate Elements in Interval

Write a function to in-order enumerate all elements of a BST in a given interval. Let $i, r$ be a range and $n$ shall enumerate all elements $x$ with $i \leq x \leq r$. Write a recursive function that does not descend into nodes that definitely contain no elements in the given range.

```plaintext
fun in_range :: "a tree => 'a => 'a => 'list" where
  "in_range t u v = [\ x | in_range (t1 t2) u v]" | "in_range (t1 t2) u v = undefined"
```

Show that you enumerate the right set of elements

lemma "\ t u v. in_range t u v = set (\ x. if in_range (t1 t2) u v then [x] else undefined)"

Show that your enumeration is actually in-order

lemma "\ t u v. in_range t u v = filter (\ x. i \leq x \&\ x \leq r) (inorder t)"

Lemma 4.1 Height of Join Tree

Theorem height [join t1 t2] \leq max (height t1) (height t2) + 1
fun in_range :: "a\tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\list" where
  "in_range [] u v" \<
  "in_range (t # l) u v" \<
  \equiv (in_range l u v) \wedge \wedge (u \leq x \leq v)"

lemma "\exists t. set (in_range t u v) = \{ x | \set_tree t, u \leq x \leq v \}" \<
proof
  case (In l)
  thus \\qed

lemma "\exists t. set (in_range t u v) = \{ x | \set_tree t, u \leq x \leq v \}" \<
proof (prove)
  goal (1 subgoal):
  \1. \exists t. set (in_range t u v) = \{ x | \set_tree t, u \leq x \leq v \}"
proof (prove):
goal (1 subgoal):
1. \((a \not\in b \land b \not\in l) = (a \not\in b \land l = [] \land l = []\)
apply (auto simp: filter_empty_conv) |
apply (auto simp: filter_empty_conv) |
apply (auto simp: filter_empty_conv) |
apply (auto simp: filter_empty_conv) |
the filter_empty |
find_theorems "\_ = \_\_\_" |

proof (prove): |

goal (0 subgoals): |

1. \( (t1 \& t2) \): |

- \[ \text{in_range } t1 \& u = [\text{x-inorder } t1 \& u \leq x \leq v]; \]
- \[ \text{in_range } t2 \& u = [\text{x-inorder } t2 \& u \leq x \leq v]; \]
- \[ \text{bst } t1; \]
- \[ \text{bst } t2; \]
- \[ \text{inset_tree } t1 \& v \times x \times x; \]
- \[ \text{inset_tree } t2 \& v \times x \times x; \]
- \[ \text{inset } v \times x \times x; \]
- \[ u < x2; \]
- \[ u \neq x2; \]
- \[ \Rightarrow \ltrue \] |
- \[ \ltrue \]
- \[ \Rightarrow \ltrue \] |
- \[ \ltrue \]

2. \( \text{bst } t2; \)

Inner syntax error: unexpected end of input |
Failed to parse prop

proof (prove): |

goal (1 subgoal): |

1. \( \ltrue \Rightarrow \ltrue \)
proof (prove)
goal: No subgoals!

proof (prove)
goal: (1 subgoal):

1. Ati x2 t2.
   \[ \text{bst t2} \Rightarrow \text{in-range t2 u v} = [\text{-inorder t2}, x \leq x \leq y] \]
fun tree2 :: "a tree ⇒ b tree ⇒ bool" where
  "tree2 tree1 tree2" ≡ True
| "tree2 (Leaf x) (Leaf y)" ⇒ tree2 (Leaf x) (Leaf y)
| "tree2 (Leaf x) (Node y)" ⇒ tree2 (Leaf x) (Node y)
| "tree2 (Node x) (Leaf y)" ⇒ tree2 (Node x) (Leaf y)
| "tree2 (Node x) (Node y)" ⇒ tree2 (Node x) (Node y)

print_statement tree2.induct
end

theorem induct:
  fixes P :: "a tree ⇒ b tree ⇒ bool"
  and a :: "a tree"
  and al :: "b tree"
  assumes "P a al"
  and "∀ x r l l x r r. P l l x r l l x r r" ⇒ P l l x r l l x r r
  and "∀ x va vb. P (x, va vb) ()"

proof (induction a arbitrary: al)
  case tree2

print_statement tree2.induct
end
fun pretty :: "a tree ⇒ 'a tchar list" where
  "pretty () = [l]"
| "pretty (l,x,r) = N x pretty l pretty r"

lemma "pretty t1 = pretty t2 ⇒ t1 = t2" apply (induction t1 t2 rule: tree2.induct) apply auto

proof (prove)
goal (1 subgoal):
1. pretty t1 = pretty t2 ⇒ t1 = t2
print_statement fun_tree2.induct

Binary trees can be uniquely pretty-printed by emitting a symbol L for a leaf, and a
symbol N for a node. Each N is followed by the pretty-print of the left and right tree.
No additional brackets are required!

datatype 'a tree = L | N 'a

fun pretty :: "'a tree => 'a tree list"

Show that pretty-printing is actually unique, i.e., no two different trees are pretty-printed
the same way. Hint: Auxiliary lemmas. Simultaneous induction over both trees.

lemma pretty_unique: "pretty t = pretty t' => t = t'"

Homework 4 Delete Minimum

Submission until Friday, May 26, 11:59pm.
Define a function to return and delete the minimum element from a non-empty BST.
You may omit the equation for the empty tree.

lemma "[(x#Leaf); lst t] == let (snd (del_min t))

Show that your function returns the first element of the inorder traversal, and a tree
whose inorder traversal corresponds to the tail of the original inorder traversal. Hint:
You may need auxiliary lemmas. Some subgoals may require more forceful methods
than auto.

lemma "x#Leaf = del_min t = (x,t')

Define a function del_min that uses so called continuations. The second argument is
a function which is applied to the result. (i.e. the minimum element and the tree of
remaining elements)
Do not use del_min for the definition, but define del_min2 recursively, passing down
(modified) continuations.

lemma "x#Leaf = del_min t = (x,t')

Define a function del_min2 that uses so called continuations. The second argument is
a function which is applied to the result. (i.e. the minimum element and the tree of
remaining elements)
Do not use del_min for the definition, but define del_min2 recursively, passing down
(modified) continuations.

lemma "x#Leaf = del_min t = (x,t')

Define a function del_min that uses so called continuations. The second argument is
a function which is applied to the result. (i.e. the minimum element and the tree of
remaining elements)
Do not use del_min for the definition, but define del_min2 recursively, passing down
(modified) continuations.