This module verifies the *SeekLowerBound* algorithm in the go-immutable-radix Go library (https://*github.com*/hashicorp/go-immutable-radix).

MODULE *RadixSeekLowerBound*

EXTENDS *FiniteSets*, *Integers*, *Sequences*, *TLC*

Set of characters to use for the alphabet of generated strings.

CONSTANT *Alphabet*

CmpOp is the comparison operator for ordered iteration. This should be TRUE if the first value is less than the second value. This is called on a single element of a sequence.

CONSTANT $CmpOp(\_, \_)$

Length of input strings generated

CONSTANT *MinLength*, *MaxLength*

ASSUME

$\quad \land \{MinLength, MaxLength\} \subseteq Nat$

$\quad \land MinLength \leq MaxLength$

Number of unique elements to construct the radix tree with. This is a set of numbers so you can test with inputs of multiple sizes.

CONSTANT *ElementCounts*

ASSUME $ElementCounts \subseteq Nat$

INSTANCE *RadixTrees*

INSTANCE *RadixIterator*

Inputs is the set of input strings valid for the tree.

$Inputs \triangleq$ UNION $\{[1 \mathrel{..} n \rightarrow Alphabet] : n \in MinLength \mathrel{..} MaxLength\}$

InputSets is the full set of possible inputs we can send to the radix tree.

$InputSets \triangleq \{T \in$ SUBSET $Inputs : Cardinality(T) \in ElementCounts\}$

TRUE iff the sequence s contains no duplicates. Copied from *CommunityModules*.

$isInjective(s) \triangleq \forall i, j \in$ DOMAIN $s : (s[i] = s[j]) \Rightarrow (i = j)$

Converts a set to a sequence that contains all the elements of S exactly once. Copied from *CommunityModules*.

$setToSeq(S) \triangleq$ CHOOSE $f \in [1 \mathrel{..} Cardinality(S) \rightarrow S] : isInjective(f)$

bytes.Compare in Go

RECURSIVE $GoBytesCompare(\_, \_)$

$GoBytesCompare(X, Y) \triangleq$

```
  CASE X = Y          → 0
    □  Len(X) = 0     → −1
    □  Len(Y) = 0     → 1
  □  OTHER  →
     IF X[1] = Y[1]
        THEN GoBytesCompare(Tail(X), Tail(Y))
        ELSE IF CmpOp(X[1], Y[1]) THEN −1 ELSE 1

 CmpSeq compares two full inputs whereas CmpOp compares only a single element
 of the alphabet.
CmpSeq(X, Y) ≜ GoBytesCompare(X, Y) ≤ 0

 CmpGte checks if X ≥ Y
CmpGte(X, Y) ≜ X = Y ∨ ¬CmpOp(X, Y)

 Sorted edge labels based on CmpOp.
SortedEdgeLabels(Node) ≜ SortSeq(setToSeq(DOMAIN Node.Edges), CmpOp)

 Returns the index of the first element that is greater than or equal to
 to the search label.
GetLowerBoundEdgeIndex(Node, Label) ≜
  IF ¬∃ e ∈ DOMAIN Node.Edges : e = Label ∨ ¬CmpOp(e, Label) THEN 0
     if there is no lower bound edge, return 0
   ELSE LET
    e ≜ SortedEdgeLabels(Node)
       sorted edges
  IN   CHOOSE idx ∈ 1 .. Len(e) :      find the index
      ∧ CmpGte(e[idx], Label)           ≥ to our search label
      ∧ ∨ idx = 1                       and its the first element that is gte
        ∨ CmpOp(e[idx − 1], Label)
```

```
 The expected value is the sorted set of all inputs where the element
 is greater than or equal to the given key.

 EXPLANATION:
  1. We convert the input set to a sequence
  2. Sort the input sequence, this is all inputs sorted now.
  3. Select the subset of the input sequence where it satisfies our comparison.
     The sequence now only has elements greater than or equal to our key
Expected(input, key) ≜
  SelectSeq(SortSeq(setToSeq(input), CmpSeq), LAMBDA elem : CmpSeq(key, elem))

  --algorithm seek_lower_bound
variables
  iterStack = ⟨⟩,
  input ∈ InputSets,
  key ∈ Inputs,
  root  = RadixTree(input),
  $node = \{\}$,
  $search = \{\}$,
  $result\ = \{\}$,
  $prefixCmp$ = “UNSET” ;

 findMin as implemented in Go
procedure findMin()begin
FindMin:
  while $Len(node.Value) = 0$ do
    with
      $labels = SortedEdgeLabels(node)$,
      $edges = [n \in 1 \,..\, Len(labels) \mapsto node.Edges[labels[n]]]$
     do
      if $Len(edges) > 1$ then
        $iterStack := iterStack \circ SubSeq(edges, 2, Len(edges))$ ;
      end if ;

      if $Len(edges) > 0$ then
          recurse again
        $node := edges[1]$ ;
       else
          shouldn't be possible
        return ;
      end if ;
    end with ;
  end while ;

  $iterStack := iterStack \circ \langle node \rangle$ ;
  return ;
end procedure ;

 This entire algorith is almost 1:1 translated where possible from the
 actual implementation in iter.go. That's the point: we're trying to verify
 our algorithm is correct for all inputs.

 Source: https://github.com/hashicorp/go-immutable-radix/blob/f63f49c0b598a5ead21c5015fb4d08fe7e3c21ea/iter.go ≠ L77
begin
   I could've just set these variables in the initializer above but
   to better closely match the algorithm, I reset them here.
Begin:
  $iterStack := \langle\rangle$ ;
  $node := root$ ;
  $search := key$ ;

Seek:
  while TRUE do
```

```
    if $Len(node.Prefix) < Len(search)$ then
      $prefixCmp := GoBytesCompare(node.Prefix, SubSeq(search, 1, Len(node.Prefix)))$ ;
     else
      $prefixCmp := GoBytesCompare(node.Prefix, search)$ ;
    end if ;

    if $prefixCmp < 0$ then
      goto $Result$ ;
     elsif $prefixCmp > 0$ then
      call $findMin()$ ;
      goto $Result$ ;
    end if ;

  $Search$:
    if $Len(node.Value) > 0 \land node.Value = key$ then
      $iterStack := iterStack \circ \langle node \rangle$ ;
      goto $Result$ ;
    end if ;

  $Consume$:
    $search := SubSeq(search, Len(node.Prefix) + 1, Len(search))$ ;

    if $Len(search) = 0$ then
      call $findMin()$ ;
      goto $Result$ ;
    end if ;

  $NextEdge$:
    with
      $idx = GetLowerBoundEdgeIndex(node, search[1])$,
      $labels = SortedEdgeLabels(node)$,
      $edges = [n \in 1 \mathrel{..} Len(labels) \mapsto node.Edges[labels[n]]]$
     do
      if $idx = 0$ then
        goto $Result$ ;
       else
        if $idx + 1 \leq Len(edges)$ then
          $iterStack := iterStack \circ SubSeq(edges, idx + 1, Len(edges))$ ;
        end if ;

        $node := edges[idx]$ ;
      end if ;
    end with ;

  end while ;

$Result$:
  $result := Iterate(iterStack)$ ;
```

```
CheckResult:
  assert result = Expected(input, key) ;
end algorithm  ;
```

!!! NOTE !!! The rest of the file is auto-generated based on the *PlusCal* above. For those who are reading this to learn TLA+/*PlusCal*, you can stop reading here.

```
 BEGIN TRANSLATION (chksum(pcal) = "7f5569db" ∧ chksum(tla) = "177f60c")
VARIABLES iterStack, input, key, root, node, search, result, prefixCmp, pc,
          stack

vars ≜ ⟨iterStack, input, key, root, node, search, result, prefixCmp, pc,
        stack⟩

Init ≜  Global variables
        ∧ iterStack = ⟨⟩
        ∧ input ∈ InputSets
        ∧ key ∈ Inputs
        ∧ root  = RadixTree(input)
        ∧ node = {}
        ∧ search = {}
        ∧ result  = {}
        ∧ prefixCmp = "UNSET"
        ∧ stack = ⟨⟩
        ∧ pc = "Begin"

FindMin ≜  ∧ pc = "FindMin"
           ∧ IF Len(node.Value) = 0
                THEN ∧ LET labels ≜ SortedEdgeLabels(node)IN
                         LET edges ≜ [n ∈ 1 .. Len(labels) ↦ node.Edges[labels[n]]]IN
                           ∧ IF Len(edges) > 1
                                THEN ∧ iterStack′ = iterStack ∘ SubSeq(edges, 2, Len(edges))
                                ELSE ∧ TRUE
                                     ∧ UNCHANGED iterStack
                           ∧ IF Len(edges) > 0
                                THEN ∧ node′ = edges[1]
                                     ∧ pc′ = "FindMin"
                                     ∧ stack′ = stack
                                ELSE ∧ pc′ = Head(stack).pc
                                     ∧ stack′ = Tail(stack)
                                     ∧ node′ = node
                ELSE ∧ iterStack′ = iterStack ∘ ⟨node⟩
                     ∧ pc′ = Head(stack).pc
```

```
                  $\land stack' = Tail(stack)$
                  $\land node' = node$
          $\land$ UNCHANGED $\langle input, key, root, search, result, prefixCmp \rangle$

$findMin \triangleq FindMin$

$Begin \triangleq$ $\land pc =$ "Begin"
          $\land iterStack' = \langle \rangle$
          $\land node' = root$
          $\land search' = key$
          $\land pc' =$ "Seek"
          $\land$ UNCHANGED $\langle input, key, root, result, prefixCmp, stack \rangle$

$Seek \triangleq$ $\land pc =$ "Seek"
        $\land$ IF $Len(node.Prefix) < Len(search)$
             THEN $\land prefixCmp' = GoBytesCompare(node.Prefix, SubSeq(search, 1, Len(node.Prefix)))$
             ELSE $\land prefixCmp' = GoBytesCompare(node.Prefix, search)$
        $\land$ IF $prefixCmp' < 0$
             THEN $\land pc' =$ "Result"
                  $\land stack' = stack$
             ELSE $\land$ IF $prefixCmp' > 0$
                          THEN $\land stack' = \langle [procedure \mapsto$ "findMin",
                                                    $pc \mapsto$ "Result"$] \rangle$
                                                    $\circ stack$
                               $\land pc' =$ "FindMin"
                          ELSE $\land pc' =$ "Search"
                               $\land stack' = stack$
        $\land$ UNCHANGED $\langle iterStack, input, key, root, node, search, result \rangle$

$Search \triangleq$ $\land pc =$ "Search"
          $\land$ IF $Len(node.Value) > 0 \land node.Value = key$
               THEN $\land iterStack' = iterStack \circ \langle node \rangle$
                    $\land pc' =$ "Result"
               ELSE $\land pc' =$ "Consume"
                    $\land$ UNCHANGED $iterStack$
          $\land$ UNCHANGED $\langle input, key, root, node, search, result, prefixCmp,$
                          $stack \rangle$

$Consume \triangleq$ $\land pc =$ "Consume"
           $\land search' = SubSeq(search, Len(node.Prefix) + 1, Len(search))$
           $\land$ IF $Len(search') = 0$
                THEN $\land stack' = \langle [procedure \mapsto$ "findMin",
                                          $pc \mapsto$ "Result"$] \rangle$
                                          $\circ stack$
                     $\land pc' =$ "FindMin"
                ELSE $\land pc' =$ "NextEdge"
```

```
                    $\land\ stack' = stack$
              $\land$ UNCHANGED $\langle iterStack, input, key, root, node, result,$
                                $prefixCmp \rangle$

$NextEdge \triangleq\ \land\ pc = $ "NextEdge"
                     $\land$ LET $idx \triangleq GetLowerBoundEdgeIndex(node, search[1])$ IN
                         LET $labels \triangleq SortedEdgeLabels(node)$ IN
                           LET $edges \triangleq [n \in 1 \dots Len(labels) \mapsto node.Edges[labels[n]]]$ IN
                             IF $idx = 0$
                                 THEN $\land\ pc' = $ "Result"
                                      $\land$ UNCHANGED $\langle iterStack, node \rangle$
                                 ELSE $\land$ IF $idx + 1 \leq Len(edges)$
                                                THEN $\land\ iterStack' = iterStack \circ SubSeq(edges, idx + 1, Len(edges))$
                                                ELSE $\land$ TRUE
                                                     $\land$ UNCHANGED $iterStack$
                                      $\land\ node' = edges[idx]$
                                      $\land\ pc' = $ "Seek"
                     $\land$ UNCHANGED $\langle input, key, root, search, result, prefixCmp, stack \rangle$

$Result \triangleq\ \land\ pc = $ "Result"
                 $\land\ result' = Iterate(iterStack)$
                 $\land\ pc' = $ "CheckResult"
                 $\land$ UNCHANGED $\langle iterStack, input, key, root, node, search, prefixCmp,$
                                   $stack \rangle$

$CheckResult \triangleq\ \land\ pc = $ "CheckResult"
                      $\land\ Assert(result = Expected(input, key),$
                                 "Failure of assertion at line 194, column 3.")
                      $\land\ pc' = $ "Done"
                      $\land$ UNCHANGED $\langle iterStack, input, key, root, node, search,$
                                        $result, prefixCmp, stack \rangle$

 Allow infinite stuttering to prevent deadlock on termination.
$Terminating \triangleq pc = $ "Done" $\land$ UNCHANGED $vars$

$Next \triangleq findMin \lor Begin \lor Seek \lor Search \lor Consume \lor NextEdge \lor Result$
             $\lor\ CheckResult$
             $\lor\ Terminating$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$Termination \triangleq \Diamond(pc = $ "Done"$)$

 END TRANSLATION
```

* Modification History
* Last modified *Fri Jul* 02 08:15:31 *PDT* 2021 by *mitchellh*

\ * Created *Thu Jul* 01 10:43:00 *PDT* 2021 by *mitchellh*