This module contains operations for working with radix trees. A radix tree is a data structure for efficient storage and lookup of values that often share prefixes, typically used with strings.

A common question when I show this to people is: how do I add to the tree? delete? update? For these, grab the *Range* of the tree, use set logic to add/remove any elements, and construct a new tree with *RadixTree*.

For educational purposes, I've heavily commented all the operations. I recommend using the constant expression evaluator to try the building blocks to learn how things work if you're confused. That's how I learned.

```
---------------------------- MODULE RadixTrees ----------------------------
LOCAL INSTANCE FiniteSets
LOCAL INSTANCE Sequences
LOCAL INSTANCE Integers

----------------------------------------------------------------------------

\* Helpers that aren't Radix-tree specific.

\* shortestSeq returns the shortest sequence in a set.
LOCAL shortestSeq(set) ==
  CHOOSE seq \in set :
    \A other \in set :
      Len(seq) <= Len(other)

\* Filter the sequence to only return the values that start with c.
LOCAL filterPrefix(set, c) == {seq \in set : seq[1] = c}

\* Strip the prefix from each element in set. This assumes that each
\* element in set already starts with prefix. Empty values will not
\* be returned.
LOCAL stripPrefix(set, prefix) ==
  {SubSeq(seq, Len(prefix)+1, Len(seq)) : seq \in set \ {prefix}}

\* Returns the set of first characters of a set of char sequences.
LOCAL firstChars(set) == {seq[1] : seq \in set}

\* Find the longest shared prefix of a set of sequences. Sequences can
\* be different lengths, but must have comparable types.
\* i.e. longestPrefix({1, 2}, {1, 2, 3}, {1, 2, 5}) == {1, 2}
LOCAL longestPrefix(set) ==
    \* Every item must at least have a common first character otherwise
    \* the longest prefix is surely empty
  IF \A x \in set, y \in set :
     /\ Len(x) >= 1
     /\ Len(y) >= 1
     /\ x[1] = y[1]
   THEN
  LET
```

$seq \triangleq shortestSeq(set)$
$end \triangleq$ CHOOSE $end \in 1 \,..\, Len(seq)$:
  $\land \forall\, i \in 1 \,..\, end$:
    $\forall\, x,\, y \in set : x[i] = y[i]$
  $\land \lor Len(seq) \leq end$ we're at the end
    $\lor \exists\, x,\, y \in set : x[end + 1] \neq y[end + 1]$ or there is no longer prefix
IN $SubSeq(seq,\, 1,\, end)$
ELSE $\langle\rangle$

Radix tree helpers

RECURSIVE $range(\_,\, \_)$
LOCAL $range(T,\, prefix) \triangleq$
LET
  $current \triangleq$ IF $Len(T.Value) > 0$ THEN $\{T.Value\}$ ELSE $\{\}$
    current value of node (if exists)

  $children \triangleq$ UNION $\{$
    $range(T.Edges[edge],\, prefix \circ T.Prefix)$:
    $edge \in$ DOMAIN $T.Edges$
  $\}$
    child values for each edge. this creates a set of sets
    so we call union to flatten it.
IN $current \cup children$

Returns the constructed radix tree for the set of keys *Keys*.
RECURSIVE $radixTree(\_,\, \_)$
LOCAL $radixTree(Keys,\, Base) \triangleq$
IF $Keys = \{\}$ THEN $\{\}$ base case, no keys empty tree
ELSE LET
  $prefix \triangleq longestPrefix(Keys)$
    longest shared prefix
  $base \triangleq Base \circ prefix$
    our new base
  $keys \triangleq stripPrefix(Keys,\, prefix)$
    keys for children, prefix stripped
  $edgeLabels \triangleq firstChars(stripPrefix(Keys,\, prefix))$
    labels for each edge (single characters)
IN [
  $Prefix \mapsto prefix$,
  $Value \mapsto$ IF $prefix \in Keys$ THEN $base$ ELSE $\langle\rangle$,
  $Edges \mapsto [L \in edgeLabels \mapsto radixTree(filterPrefix(keys,\, L),\, base)]$
]

Returns the minimal radix tree for the set of keys $Keys$.

$RadixTree(Keys) \triangleq$

LET

$edgeLabels \triangleq firstChars(Keys)$

IN [

The root of a radix tree is always a non-value that only has outward edges to the first sets of values.

$Prefix \mapsto \langle\rangle,$

$Value \mapsto \langle\rangle,$

$Edges \mapsto [L \in edgeLabels \mapsto radixTree(filterPrefix(Keys, L), \langle\rangle)]$

]

Range returns all of the values that are in the radix tree T.

$Range(T) \triangleq range(T, \langle\rangle)$

Nodes returns all the nodes of the tree T.

RECURSIVE $Nodes(\_)$

$Nodes(T) \triangleq \{T\} \cup \text{UNION } \{Nodes(T.Edges[e]) : e \in \text{DOMAIN } T.Edges\}$

TRUE iff the radix tree T is minimal. A tree T is minimal if there are the minimum number of nodes present to represent the range of the tree.

$Minimal(T) \triangleq$

$\neg\exists\, n \in (Nodes(T) \setminus \{T\}) :$

$\land\ Cardinality(\text{DOMAIN } n.Edges) = 1$

$\land\ n.Value = \langle\rangle$

---

* Modification History

* Last modified *Fri Jul* 02 13:52:35 *PDT* 2021 by *mitchellh*

* Created *Mon Jun* 28 08:08:13 *PDT* 2021 by *mitchellh*