```
──────────────────── MODULE SplitOrder ────────────────────
This module implements a hashmap using Shalev et al.'s split-ordered list structure

EXTENDS Integers

CONSTANTS NULL, PossibleKeys, PossibleValues, LoadFactor, MaxSize

VARIABLES keys, list, buckets, size, count

ASSUME
    ∧ PossibleKeys ⊆ 0 . . 15
    ∧ NULL ∉ PossibleKeys
    ∧ NULL ∉ PossibleValues

The Init for split-order keys is initially empty the map maps every possible key to NULL The list
initially contains only the 0 dummy node
SOInit ≜    ∧ keys = {}
            ∧ list    = [n ∈ 0 . . 255 ↦ IF n = 0 THEN 0 ELSE NULL]
            ∧ buckets = [m ∈ PossibleKeys ↦ IF m = 0 THEN 0 ELSE NULL]
            ∧ size = 1
            ∧ count = 0

 Lookup table for bit-reversed keys with MSB set
SORegularKey(k) ≜  CASE k = 0 → 1
                    □k = 1 → 9
                    □k = 2 → 5
                    □k = 3 → 13
                    □k = 4 → 3
                    □k = 5 → 11
                    □k = 6 → 7
                    □k = 7 → 15
                    □k = 8 → 1
                    □k = 9 → 9
                    □k = 10 → 5
                    □k = 11 → 13
                    □k = 12 → 3
                    □k = 13 → 11
                    □k = 14 → 7
                    □k = 15 → 15

 Lookup table for bit-reversed keys
SODummyKey(k) ≜    CASE k = 0 → 0
                    □k = 1 → 8
                    □k = 2 → 4
                    □k = 3 → 12
                    □k = 4 → 2
                    □k = 5 → 10
```

```
                      $\Box k = 6 \rightarrow 6$
                      $\Box k = 7 \rightarrow 14$
                      $\Box k = 8 \rightarrow 1$
                      $\Box k = 9 \rightarrow 9$
                      $\Box k = 10 \rightarrow 5$
                      $\Box k = 11 \rightarrow 13$
                      $\Box k = 12 \rightarrow 3$
                      $\Box k = 13 \rightarrow 11$
                      $\Box k = 14 \rightarrow 7$
                      $\Box k = 15 \rightarrow 15$
```

Lookup table for parent buckets

```
$Parent(b) \triangleq$   CASE $b = 0 \rightarrow 0$
                    $\Box b = 1 \rightarrow 0$
                    $\Box b = 2 \rightarrow 0$
                    $\Box b = 3 \rightarrow 1$
                    $\Box b = 4 \rightarrow 0$
                    $\Box b = 5 \rightarrow 1$
                    $\Box b = 6 \rightarrow 2$
                    $\Box b = 7 \rightarrow 3$
                    $\Box b = 8 \rightarrow 0$
                    $\Box b = 9 \rightarrow 1$
                    $\Box b = 10 \rightarrow 2$
                    $\Box b = 11 \rightarrow 3$
                    $\Box b = 12 \rightarrow 8$
                    $\Box b = 13 \rightarrow 5$
                    $\Box b = 14 \rightarrow 6$
                    $\Box b = 15 \rightarrow 7$
```

Inserting into the “linked list”

```
$ListInsert(k, v) \triangleq$ IF $list[k] = NULL$
                         THEN $list' = [list$ EXCEPT $![k] = v] \land count' = count + 1$
                         ELSE  UNCHANGED $\langle list, count \rangle$
```

Removing from the “linked list”

```
$ListRemove(k) \triangleq$ IF $list[k] = NULL$
                      THEN UNCHANGED $\langle list, count \rangle$
                      ELSE  $list' = [list$ EXCEPT $![k] = NULL] \land count' = count - 1$
```

Recursively initializes buckets

```
RECURSIVE $BucketInit(\_)$
$BucketInit(b) \triangleq$ IF $buckets[Parent(b)] = NULL \land Parent(b) \neq 0$
                    THEN $BucketInit(Parent(b))$
                    ELSE  $buckets' = [buckets$ EXCEPT $![b] = SODummyKey(b)]$
```

Find the value of the key k in the bucket b Results in the value if b is initialized and k is in b

$$ListFind(b, k) \triangleq \text{IF } k > b \land list[b] \neq NULL \text{ THEN } list[k] \text{ ELSE } NULL$$

SOFind finds a key in the map

$$\begin{aligned} SOFind(k) \triangleq\ & \text{IF } buckets[k\%size] = NULL \\ & \quad \text{THEN } NULL \quad \text{should initialize bucket, but also needs a ``return value''} \\ & \quad \text{ELSE } ListFind(buckets[k\%size], k) \end{aligned}$$

$$Min(a, b) \triangleq \text{IF } a > b \text{ THEN } b \text{ ELSE } a$$

$$\begin{aligned} BucketGrow \triangleq\ & \text{IF } count \neq 0 \land size \div count > LoadFactor \\ & \quad \text{THEN } size' = Min(size * 2, MaxSize) \land \text{UNCHANGED } \langle keys, list, buckets, count \rangle \\ & \quad \text{ELSE } \text{UNCHANGED } \langle keys, list, buckets, count, size \rangle \end{aligned}$$

Inserting into the buckets

$$\begin{aligned} BucketInsert(k, v) \triangleq\ & \quad \text{Either a bucket needs to be initialized} \\ & \lor \land buckets[k\%size] = NULL \\ & \quad \land BucketInit(k\%size) \\ & \quad \land ListInsert(SORegularKey(k), v) \\ & \quad \land keys' = keys \cup \{k\} \\ & \quad \text{Or the bucket is already initialized} \\ & \lor \land buckets[k\%size] \neq NULL \\ & \quad \land ListInsert(SORegularKey(k), v) \\ & \quad \land keys' = keys \cup \{k\} \\ & \quad \land \text{UNCHANGED } \langle buckets \rangle \end{aligned}$$

Removing from the buckets

$$\begin{aligned} BucketRemove(k) \triangleq\ & \land ListRemove(SORegularKey(k)) \\ & \land keys' = keys \setminus \{k\} \\ & \land \text{UNCHANGED } \langle buckets \rangle \end{aligned}$$

$$\begin{aligned} SOInsert \triangleq\ & \land \exists k \in PossibleKeys : \\ & \qquad \exists v \in PossibleValues : \\ & \qquad \quad BucketInsert(k, v) \\ & \land \text{UNCHANGED } size \end{aligned}$$

$$\begin{aligned} SORemove \triangleq\ & \land \exists k \in PossibleKeys : \\ & \qquad BucketRemove(k) \\ & \land \text{UNCHANGED } size \end{aligned}$$

The Next for split order

$$\begin{aligned} SONext \triangleq\ & \lor SOInsert \\ & \lor SORemove \\ & \lor \exists k \in keys : SOFind(k) \in PossibleValues \land \text{UNCHANGED } \langle keys, buckets, list, count, size \rangle \\ & \lor \exists k \in (PossibleKeys \setminus keys) : SOFind(k) = NULL \land \text{UNCHANGED } \langle keys, buckets, list, count, s \\ & \lor BucketGrow \end{aligned}$$

Split-order spec

$SOSpec \triangleq SOInit \land \Box[SONext]_{\langle keys, list, buckets, size, count \rangle}$

If I can get map to work as intended . . .

INSTANCE *hashmap*

Split-order implements *hashmap*

THEOREM $SOSpec \Rightarrow HashmapSpec$