1. null。
2. 非 null，且该项是活动的(isActive 为 true)。
3. 非 null，且该项标记被删除(isActive 为 false)。

该表(图 5-15)的构造由分配空间然后设置每个 HashEntry 引用为 null 组成。

```
1       /**
2        * Construct the hash table.
3        */
4       public QuadraticProbingHashTable( )
5       {
6           this( DEFAULT_TABLE_SIZE );
7       }
8
9       /**
10       * Construct the hash table.
11       * @param size the approximate initial size.
12       */
13      public QuadraticProbingHashTable( int size )
14      {
15          allocateArray( size );
16          makeEmpty( );
17      }
18
19      /**
20       * Make the hash table logically empty.
21       */
22      public void makeEmpty( )
23      {
24          currentSize = 0;
25          for( int i = 0; i < array.length; i++ )
26              array[ i ] = null;
27      }
28
29      /**
30       * Internal method to allocate array.
31       * @param arraySize the size of the array.
32       */
33      private void allocateArray( int arraySize )
34      {
35          array = new HashEntry[ arraySize ];
36      }
```

图 5-15 初始化散列表的例程

在图 5-16 中所示的 contains(x)调用私有方法 isActive 和 findPos。这里的 private 方法 findPos 实施对冲突的解决。我们肯定在 insert 例程中散列表至少为该表中元素个数的两倍大，这样平方探测解决方案总可以实现。在图 5-16 的实现中，标记为删除的那些元素被认为还在表内。这可能引起一些问题，因为该表可能提前过满。我们现在就来讨论它。