author: Michael Lee ml3406@RIT.EDU

Activity 1: Problem Solving

1.

"lad" → 14 → location = 14%12 = 2
"but" → 40 → location = 40%12 = 4
"is" → 26 → location = 26%12 = 2 (collision)
"chin" → 30 → location = 30%12 = 6
"be" → 5 → location = 5%12 = 5
"fun" → 38 → location = 38%12 = 2 (collision)
"blab" → 13 → location = 13%12 = 1
"zoo" → 53 → location = 53%12 = 5 (collision)

Location	Key
0	
1	"blab"
2	"lad" → "is" → "fun"
3	
4	"but"
5	"be" → "zoo"
6	"chin"
7	
8	
9	
10	
11	

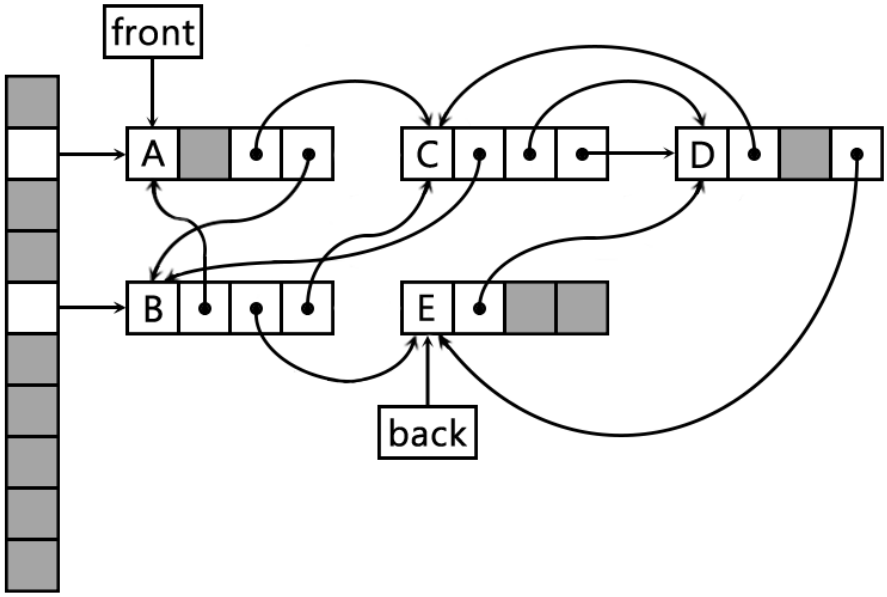
2.

"blab" → "lad" → "is" → "fun" → "but" → "be" → "zoo" → "chin"

3.

```
def hash_function(aString):
    result = 0
    for index in range(len(aString)):
        result += ord(aString [index]) * pow(31, index)
    return result
```

4. (a)



(b)

```
class ChainNode:
    __slots__ = "key", "chain", "prevInsertion", "nextInsertion"
    def __init__(self, obj=None, prev=None, link=None, chain=None):
        self.key = obj
        self.chain = chain
        self.prevInsertion = prev
        self.nextInsertion = link
class LinkedHashTable(SetType, Iterable):
    __slots__ = "table", "buckets", "LOAD_LIMIT", "front", "back"
    def remove(self, key):
        if key is in the table (use .contains(key)):
            index = self.hash_function(key) % self.buckets
            if self.size == 1:
                self.front = None
                self.back = None
                self.table[index] = None
            else:
                target_node = self.table[index]
                before_target_node = None
                while target_node.key != obj:
                    before_target_node = target_node
                    target_node = target_node.chain
                if self.front == target_node:
                    self.front = self.front.nextInsertion
                    self.front.prevInsertion = None
                elif self.back == target_node:
                    self.back = self.back.prevInsertion
```

```
                self.back.nextInsertion = None
            else:
                target_node.prevInsertion.nextInsertion = target_node.nextInsertion
                target_node.nextInsertion.prevInsertion = target_node.prevInsertion
            if before_target_node is None:
                self.table[index] = self.table[index].chain
            else:
                before_target_node.chain = target_node.chain
        self.size -= 1
```