ECE374 Fall2020

Lab9: Self-Organizing List

Name: Zhang Yichi 3180111309

Dec. 9th 2020

1 Introduction

In this lab, we will implement and test some functions of the self-organizing list with move to front (MTF).

2 Python Code for Self-Organizing List

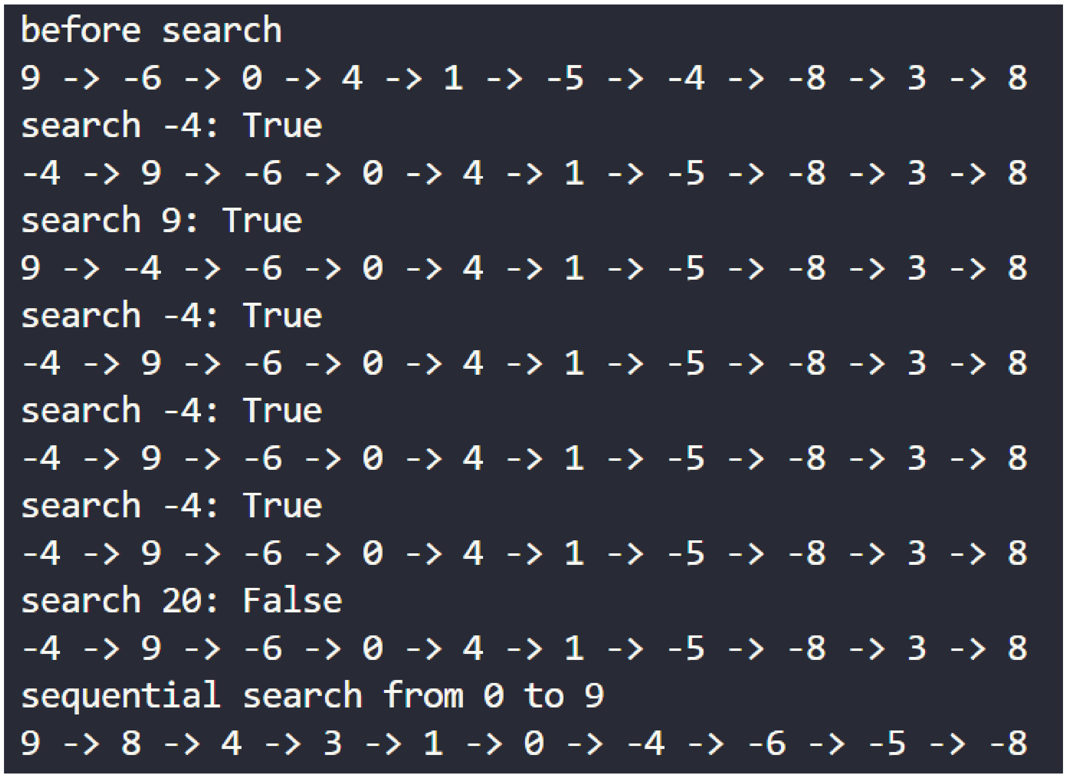
```
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None

class SelfOrganizingList:
    def __init__(self,elements=[]):
        self.first = None
        self.last = None
        for i in elements:
            self.insert(i)

    def insert(self,x):
        temp = Node(x)
        if self.first == None:
            self.first = self.last = temp
        else:
            self.last.next = temp
            self.last = temp

    def delete(self,x):
        prev = None
        curr = self.first
        while curr != None:
            if curr.val == x:
                if prev != None:
                    prev.next = curr.next
                else:
                    self.first = None
                return True
            prev = curr
            curr = curr.next
        return False

    def search(self,x):
        prev = None
        curr = self.first
        while curr != None:
            if curr.val == x:
                if prev != None:
                    prev.next = curr.next
                    curr.next = self.first
                    self.first = curr
                return True
            prev = curr
            curr = curr.next
        return False

    def printSOL(self):
        if self.first == None:
            print("Empty List")
            return
        temp = self.first
        while temp != None:
            print(temp.val,end='')
            if temp.next != None:
                print(" -> ",end='')
            temp = temp.next
        print()
```

Self-organizing list is a modified linked list. Each node in the list stores a value and its next node. Every time when a search is successfully applied, the element searched for will be move to the front and be the new header.

3 Test Example for Self-Organizing List

```
l = random.sample(range(-10,10),10)
SOL = SelfOrganizingList(l)
print("before search")
SOL.printSOL()
a = random.choice(l)
print(f"search {a}: {SOL.search(a)}")
SOL.printSOL()
a = random.choice(l)
print(f"search {a}: {SOL.search(a)}")
SOL.printSOL()
a = random.choice(l)
print(f"search {a}: {SOL.search(a)}")
SOL.printSOL()
print(f"search {a}: {SOL.search(a)}")
SOL.printSOL()
```

```
print(f"search {a}: {SOL.search(a)}")
SOL.printSOL()
print(f"search 20: {SOL.search(20)}")
SOL.printSOL()
for i in range(10):
    SOL.search(i)
print("sequential search from 0 to 9")
SOL.printSOL()
```



```
before search
9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -4 -> -8 -> 3 -> 8
search -4: True
-4 -> 9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
search 9: True
9 -> -4 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
search -4: True
-4 -> 9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
search -4: True
-4 -> 9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
search -4: True
-4 -> 9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
search 20: False
-4 -> 9 -> -6 -> 0 -> 4 -> 1 -> -5 -> -8 -> 3 -> 8
sequential search from 0 to 9
9 -> 8 -> 4 -> 3 -> 1 -> 0 -> -4 -> -6 -> -5 -> -8
```

Figure 1: Self-organizing list test result.

In the first test, we randomly searched for 3 items one by one and found that they were moved to the front. In the second test, we searched the same item for several times and found that the list stays the same. In the third test, we searched for a nonexisting item and found the list was unchanged. In the last test, we sequentially searched 0 to 9 and found that the first several elements are in [0,9] and are in decreasing order. All the results are reasonable and expected.

4 Competitive Analysis

MTF is 4-Competitive for self-organizing lists.