Algorithms and Data Structures for Data Science

Lists

CS 277
Brad Solomon

September 9, 2021
lab_parsing Feedback

Average completion time looks to be ~3 hours

Not everyone found the lecture helpful

All students agreed lab matched learning objectives

Avg: 13.89 / 15
Planning out a problem

Write down the input and output (make sure you know it!)

Work out by hand the steps you need to get the output

Figure out how to do the necessary steps

Write the code — and then test it!
Reflecting on lab_parsing

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>bradsol,10,8,5,7,8</td>
<td></td>
</tr>
<tr>
<td>hdeep,10,8,5,7,8,9</td>
<td></td>
</tr>
</tbody>
</table>
Open Dialogue about CS 277
lab_cipher adjustments

Vigenere Decode now extra credit

CaesarEncode / Decode significantly up-weighted
mp_racing adjustments

MPs designed as mini-data science projects

(Most) content is reasonable — clarity not so much

Future assignments will be simpler and shorter

PuzzleMovement extra credit — for now
mp_racing: loadMap()

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

_map.txt
mp_racing: loadCargo()

Input:  

Output: 

3  
5  
3  
7  
_cargo.txt
mp_racing: checkSquare()

Input:                      Output:
def checkAvail(self):
    if self.curr == None:
        return []

    # racingBot's current position
    row = self.curr[0]
    col = self.curr[1]

    # The return list
    avail = []

    north = (0,0)
    if self.checkSquare(north):
        avail.append(north)

    return avail
Takeaway thoughts

There will be a proficiency exam in the near future

Course curriculum will slow down

Pacing and Content survey on Moodle
Learning Objectives

Define the list *abstract data type*

Discuss implementation strategies for lists

Debate which strategy is better (and how to judge ‘better’)

List Abstract Data Type

Values:

Functions:
List Implementations

1.

2.
Linked List

Head

C → S → 2 → 7 → 7 → None
Creating a linked list

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

class linkedList:
    def __init__(self, head=None):
        self.head = head

if __name__ == '__main__':
    n1 = Node(3)
    n2 = Node(5)
    n3 = Node(7)
    n1.next = n2
    n2.next = n3
    n3.next = n1
    ll = linkedList(n1)
Linked List

Add (B)
class Node:
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

class linkedList:
    def __init__(self, head=None):
        self.head = head

    def add(self, data):
        temp = self.head
        self.head = Node(data, temp)

if __name__ == '__main__':
    ll = linkedList()
    for i in range(3):
        ll.add(i)
Linked List

Index (3)

Head

C ➔ S ➔ 2 ➔ 7 ➔ 7 ➔ None
class linkedList:

def index(self, pos):
    curr = self.head
Linked List

Insert(X,2)

Head

C -> S -> 2 -> 7 -> 7 -> None
```python
class linkedList:
    def insert(self, data, pos=0):
        if (pos == 0):
            self.add(data)
        else:
            prev = self.index(pos-1)
            temp = prev.next
            prev.next = Node(data, temp)
```
Linked List

Remove (7)

Head

C → S → 2 → 7 → 7 → None
```python
class linkedList:
    def remove(self, data):
        prev = None
        curr = self.head
        while curr:
            if curr.data == data:
                if prev == None:
                    self.head = self.head.next
                    break
                else:
                    prev.next = curr.next
                    break
            else:
                prev = curr
                curr = curr.next
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```
Linked List

Head

Pop (4)
class linkedList:
    def pop(self, pos):
        if(pos == 0):
            temp = self.head.data
            self.head = self.head.next
            return temp
        else:
            prev = self.index(pos-1)
            temp = prev.next
            prev.next = temp.next
            return temp.data
Exercise 1

```python
ll = linkedList()
for i in range(3):
    ll.add(i)

ll.insert(6, 1)
ll.insert(3, 1)
ll.insert(10, 3)

ll.remove(6)
ll.remove(2)
ll.remove(12)

ll.pop(2)
ll.pop(0)
```
List Implementations

1. Linked List

2. Arrays
Array
Creating an array list

```python
class arrayList:
    def __init__(self, array=[]):
        self.array = array
```
def add(self, data):
    self.array.insert(0, data)
def index(self, pos):
return self.array[pos]
def insert(self, data, pos=0):

def remove(self, data):
    self.array.remove(data)
def pop(self, pos):
return self.array.pop(pos)
Which implementation is better?
What do we care about when we write code?