**CS 2302 Data Structures**

**Lab Report #3**

**Linked Lists**

Carlos Cardenas

October 7, 2019

**Introduction**

One disadvantage of using arrays to store data is that arrays are static structures and therefore cannot be easily extended or reduced to fit the data set. Arrays are also expensive to maintain new insertions and deletions. In this chapter we consider another data structure called Linked Lists that addresses some of the limitations of arrays. A linked list is a linear data structure where each element is a separate object.

Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty, then the head is a null reference.

A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

**Implementation**

The following methods were implemented:

**Print:**

Traverses through the lists and prints each number while it is not None.

**Insert:**

Traverses through the whole array and places the new node in the correct place, then modifies the pointers accordingly.

**Delete:**

Traverses through the whole array and if the number matches the input, removes the pointer of the previous node and updates it to the next node.

**Merge:**

Reads two linked lists and adds the elements of the second one into the first one.

**IndexOf:**

Starts a position counter that returns the position when it matches the input number.

**Clear:**

Points the head to None.

**Min:**

Gets the data of the first node and compares it to all the remaining nodes, returning the minimum.

**Max:**

Gets the data of the first node and compares it to all the remaining nodes, returning the max.

**HasDuplicates:**

Checks if the next node in the list contains the same number as before, since it is a sorted list.

**Select:**

Returns the k,th element of the list by increasing a counter every time the loop executes.

**Experimental Results**

To test if my code works, I first started by creating an empty list and see what the outputs where.

Next, I created a simple list to test if all of the methods work as intended. After inspection, all methods completed worked as excepted with no errors in output. However, given that the code was specifically made for sorted lists, creating an unsorting list will cause some of the methods to fail, since it relies on the list being sorted, specifically the HasDuplicates method.

Below are the outputs of my test cases:

A screenshot of a cell phone

Description automatically generatedOutput with an empty list:

**A screenshot of a cell phone

Description automatically generatedOutput with sorted list:**

**Conclusion**

**A screenshot of a cell phone

Description automatically generatedOutput with unsorted list:**

|  |  |  |
| --- | --- | --- |
| Function | SortedList | List |
| Print() | O(n) | O(n) |
| Insert(i) | O(1) | O(1) |
| Delete(i) | O(1) | O(1) |
| Merge(M) | O(n^2) | O(n2) |
| IndexOf(i) | O(n) | O(n) |
| Clear(i) | O(1) | O(1) |
| Min() | O(1) | O(n) |
| Max() | O(1) | O(n) |
| HasDuplicates() | O(n) | O(n^2) |
| Select(k) | O(n) | O(n) |

**Conclusion**

After being able to successfully solve most of the problems, I feel confident that my skills at Linked Lists have greatly increase and that I can solve most of the problems given to me. I also found it easier to implement the methods by first creating a drawing with the pointers and what they need to do before actually writing the code for them. This technique made it so much easier to solve most of the problems. I was a bit disappointed that my Merge method could not work.

**Appendix**

**﻿**import math

class Node(object):

# Constructor

def \_\_init\_\_(self, data, next=None):

self.data = data

self.next = next

class SortedList(object):

# Constructor

def \_\_init\_\_(self):

self.head = None

self.tail = None

def BuildList(pList):

L = SortedList()

for d in pList:

Append(L,d)

return L

def Append(L,x):

if L.head is None:

L.head = Node(x)

L.tail = L.head

else:

L.tail.next = Node(x)

L.tail = L.tail.next

def Insert(self,i):

t = self.head

if t == None:

return None

if t.data > i:

n = Node(i)

n.next = t

self.head = n

while t != None:

if t.data < i:

if t.next == None:

n = Node(i)

t.next = n

if t.next.data > i:

n = Node(i)

n.next = t.next

t.next = n

t = t.next

def Delete(self,i):

t = self.head

prev = None;

while t is not None:

if t.data == i:

if prev is None:

self.head = t.next

if prev is not None:

prev.next = t.next

prev = t

t = t.next

def Merge(self,M):

t = self.head

p = M.head

if t == None and p == None:

return None

if t == None:

return p

if p == None:

return t

if t.data > p.data:

n = Node(p.data)

n.next = t

self.head = n

while t and p:

if t.data < p.data:

if t.next == None:

n = Node(p.data)

t.next = n

if t.next.data > p.data:

n = Node(p.data)

n.next = t.next

t.next = n

t = t.next

p = p.next

def Print(L):

t = L.head

while t is not None:

print(t.data,end=' ')

t = t.next

print()

def IndexOf(self,i):

count = 0

t = self.head

while t != None:

if t.data == i:

print(count)

count += 1

t = t.next

return -1

def Clear(self):

if self.head != None and self.head.next != None:

self.head = None

def Min(self):

t = self.head

if t == None:

return None

mini = t.data

while t != None:

if t.data < mini:

mini = t.data

t = t.next

return mini

def Max(self):

t = self.head

if t == None:

return None

maxi = t.data

while t != None:

if t.data > maxi:

maxi = t.data

t = t.next

return maxi

def HasDuplicates(self):

t = self.head

if t == None:

return

while t.next != None:

if t.data == t.next.data:

return True

t = t.next

return False

def Select(self,k):

count = 0

t = self.head

if t == None:

return None

while t != None:

if t.data == k:

return count

count += 1

t = t.next

return -math.inf

L = BuildList((1,2,3,4,5,6,7,8,9))

M = BuildList((0,3,7,12))

print("Print:")

Print(L)

print("Insert:")

Insert(L,0)

Insert(L,3)

Insert(L,9)

Insert(L,11)

Print(L)

print("Delete:")

Delete(L,7)

Print(L)

print("Merge:")

Merge(L,M)

Print(L)

print("Index:")

print(IndexOf(L,10))

print("Clear:")

Clear(L)

Print(L)

print("Print:")

M = BuildList((1,3,5,7,9,10,15))

Print(M)

print("Min:")

print(Min(M))

print("Max:")

print(Max(M))

print("Duplicates:")

print(HasDuplicates(M))

print("Select:")

print(Select(M,3))

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”