210CT – Programming, Algorithms and Data Structures Coursework

GitHub Link

<https://github.com/pratheep96/210CTcoursework>

Question 1

1. #This asks for the user to input
2. firstString=input("Enter the first word: ")
3. secondString=input("Enter the second word: ")
4. #This converts the user input into list
5. firstWord=list(firstString)
6. secondWord=list(secondString)
7. #This sets up a blank array
8. finalWord=[]
9. count = 0
11. #Each loop places each letter one after the other depending on which string is longer
12. **if** len(firstString)<len(secondString):
13. **while** count < len(secondString):
14. **if** count == len(firstString):
15. **while** count < len(secondString):
16. finalWord.append(secondWord[count])
17. count = count + 1
18. **elif** count < len(firstString) **and** count < len(secondString):
19. finalWord.append(firstWord[count])
20. finalWord.append(secondWord[count])
21. count = count + 1
22. **elif** len(secondString) < len(firstString):
23. **while** count < len(firstString):
24. **if** count == len(secondString):
25. **while** count < len(firstString):
26. finalWord.append(firstWord[count])
27. count = count + 1
28. **elif** count < len(firstString) **and** count < len(secondString):
29. finalWord.append(firstWord[count])
30. finalWord.append(secondWord[count])
31. count = count + 1
32. **else**:
33. **while** count < len(firstString):
34. finalWord.append(firstWord[count])
35. finalWord.append(secondWord[count])
36. count = count + 1
38. #This joins the letters together
39. finalString=''.join(finalWord)
40. **print**(finalString)

Question 2

1. #Asks for user input
2. initialNumber = int(input("Enter the number: "))
3. #Converts numbers into string
4. numberList = str(initialNumber)
5. #Converts string into list
6. numberList = list(numberList)
7. #Seperates numbers into digits
8. numberList = list(map(int, numberList))
9. count = 0
10. total = 0
11. #Peforms armstrong number calculation for each digit
12. **while** count < len(numberList):
13. total = total + ((numberList[count])\*\*3)
14. count = count + 1
16. **if** total == initialNumber:
17. **print**("Yes it is an Armstrong number")
18. **else**:
19. **print**("No it is not an Armstrong number")

Question 3a

P1 🡨 [0, …..., m-1]

P2 🡨 [0, ……, n – 1]

Sum 🡨 [] size(m x n)

Sum 🡨 P1

FOR i IN P2

Sum 🡨 Sum[i] + P2[i]

PRINT Sum

Question 3b

1. #Asks for user input
2. P1 = input("Enter the coefficients of the first polynomial(with commas): ")
3. P2 = input("Enter the coefficients of the second polynomial(with commas): ")
4. #Splits user input into elements in a list
5. P1 = [int(x) **for** x **in** P1.split(',') **if** x]
6. P2 = [int(x) **for** x **in** P2.split(',') **if** x]
7. **print**(P1)
8. **print**(P2)
9. #Sets up the counter for the while loop
10. count1 = len(P1)-1
11. count2 = len(P2)-1
12. #Sets up blank array for the results
13. List1 = []
14. List2 = []
15. #Both loops add each element from one list to another
16. **while** count1 > -1:
17. List1.append(count1)
18. count1 = count1 - 1
20. **while** count2 > -1:
21. List2.append(count2)
22. count2 = count2 - 1
24. #Both loops work out the multiplication of each element in the list based on where the elements are in the list
25. P1a = zip(P1, List1)
26. P2a = zip(P2, List2)
27. result = []
28. **for** i **in** P1a:
29. **for** j **in** P2a:
30. **if** i[1] + j[1] **in** result:
31. result[i[1] + j[1]] += i[0] \* j[0]
32. **else**:
33. result[i[1] + j[1]] = i[0] \* j[0]
35. **print**(result)

Question 3c

1. #Asks for user input
2. P1 = input("Enter the coefficients of the first polynomial(with commas): ")
3. P2 = input("Enter the coefficients of the second polynomial(with commas): ")
4. #Splits user input into elements in a list
5. P1 = [int(x) **for** x **in** P1.split(',') **if** x]
6. P2 = [int(x) **for** x **in** P2.split(',') **if** x]
7. #Sets up the counter for the while loop
8. count1 = len(P1)-1
9. count2 = len(P2)-1
10. count = 0
11. #Sets up blank array for the results
12. List1 = []
13. List2 = []
14. #Both loops work out the derivative of each element in the list based on where the elements are in the list
15. **while** count1 > -1:
16. result = count1 \* P1[count]
17. List1.append(result)
18. count1 = count1 - 1
19. count = count + 1
21. count = 0
23. **while** count2 > -1:
24. result = count2 \* P2[count]
25. List2.append(result)
26. count2 = count2 - 1
27. count = count + 1
29. **print**(List1)
30. **print**(List2)

Question 3d

Run time is O(N) for part a and part c

Run time is O(N2 + N) for part b.

Question 4

FUNCTION palindromeCheck(arrayList)

backwardArray 🡨 arrayList reversed

IF backwardArray is equal to the arrayList

PRINT (It is a palindrome)  
 ELSE

PRINT (Not a palindrome)

arrayInput 🡨 INPUT(STRING)

arrayList 🡨 SPLIT arrayInput (“,”)

PRINT (arrayList)

Palindrome 🡨 palindromeCheck (arrayList)

1. #Function to check for palindrome
2. **def** palindromeCheck(arrayList):
3. backwardArray = arrayList[::-1]
4. **if** backwardArray == arrayList:
5. **print**("It is a palindrome")
6. **else**:
7. **print**("Not a palindrome")
9. #Asks for user input
10. arrayInput=input(str("Enter array(use commas and no spaces): "))
11. arrayList = arrayInput.split(",")
12. #Prints result
13. **print**(arrayList)
14. palindrome = palindromeCheck(arrayList)

Run time is O(N) because the function runs n number of times.

Question 5

1. #Function to mirror each word in the sentence
2. **def** mirrorString(string):
3. **return** ' '.join(word[::-1] **for** word **in** string.split())
5. #Asks user for input
6. string = str(input("Enter a sentence: "))
7. mirrorSentence = mirrorString(string)
8. #Prints the result
9. **print**(mirrorSentence)

Time complexity is O(N) because the function runs n number of times.

Question 6

1. #Function searches for number in numberlist
2. **def** linearSearch(numberInput, numberList):
3. **if** numberList:
4. #Conditional statement that checks if number is in numberList
5. **if** numberInput == numberList[0]:
6. **print**("Found the number " + str(numberInput))
7. #Reduces numberList for function to repeat its search
8. **else**:
9. numberList = numberList[1:]
10. linearSearch(numberInput, numberList)
11. **else**:
12. **print**("Number not found")
14. #Asks user for input
15. numberInput = int(input("Enter number: "))
16. numberList = [4, 7, 20, 13, 17, 24, 32, 38, 43, 23, 12, 57, 14, 19, 70, 56, 2, 54, 34]
17. linearSearch(numberInput, numberList)

In the best case scenario the time complexity is O(1) if the number being searched for is the first number in the number list.

In the worst case scenario the time complexity O(N) if the number being searched for is the last number in the number list or if the number doesn’t exist in the list.

Question 7

Insertion sort

1. #This is the given number list
2. initialnumberList = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]
3. #For each element in the list the for loop is implemented n number of times
4. **for** i **in** range( 1, len(initialnumberList) ):
5. temp = initialnumberList[i]
6. x = i
7. #This while loops sorts the numbers out in ascending order
8. **while** x > 0 **and** temp < initialnumberList[x - 1]:
9. initialnumberList[x] = initialnumberList[x - 1]
10. x = x - 1
11. initialnumberList[x] = temp
12. #This prints the numbers in ascending order
13. **print**(initialnumberList)

Bubble sort

1. #This is the given number list
2. initialnumberList = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]
3. #This is a loop that sorts the numbers into ascending order based where they are in the list
4. count = len(initialnumberList)-1
5. **while** count > 0:
6. **for** i **in** range(0,count):
7. **if** initialnumberList[i]>=initialnumberList[i+1]:
8. initialnumberList[i], initialnumberList[i+1] = initialnumberList[i+1], initialnumberList[i]
9. count = count - 1
10. #This prints the numbers in ascending order
11. **print**(initialnumberList)

Selection sort

1. #This is the given number list
2. initialnumberList = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]
4. #This for loops sorts the numbers in ascending order
5. **for** i **in** range(len(initialnumberList)):
6. minimum = min(initialnumberList[i:])
7. minimumIndex = initialnumberList[i:].index(minimum)
8. initialnumberList[i + minimumIndex] = initialnumberList[i]
9. initialnumberList[i] = minimum
10. #This prints the numbers in ascending order
11. **print** (initialnumberList)

Insertion sort used 8 comparisons and 10 elements were moved.

Bubble sort used 31 comparisons and 10 elements were moved.

Selection sort used 10 comparisons and 10 elements were moved.

Therefore insertion sort is the best performing method.

The worst arrangement of this sequence would be the numbers in descending order as that’ll take the longest to arrange back into ascending order.

Question 8

1. #Sets upper bound and lower bound of the search
2. highNumber = 2000
3. lowNumber = 1
4. feedback = "No"
5. #This while loop repeats until the user confirms the computer guess is correct
6. **while** feedback == "No":
7. computerGuess = int(round(highNumber + lowNumber)/2)
8. **print**("Computer guesses " + str(computerGuess))
9. feedback = str(input("Is this correct(Yes/No)? "))
10. **if** feedback == "Yes":
11. **print**("Computer has guessed your number correctly")
12. **elif** feedback == "No":
13. highOrLow = str(input("Higher or Lower(Higher/Lower)? "))
14. **if** highOrLow == "Higher":
15. lowNumber = computerGuess
16. **elif** highOrLow == "Lower":
17. highNumber = computerGuess

Question 9

1. **class** Node(object):
2. **def** \_\_init\_\_(self, value):
3. self.value=value
4. self.next=None
5. self.previous=None
7. **class** List(object):
8. **def** \_\_init\_\_(self):
9. self.head=None
10. self.tail=None
11. **def** insert(self,n,x):
12. #Not actually perfect: how do we prepend to an existing list?
13. **if** n!=None:
14. x.next=n.next
15. n.next=x
16. x.previous=n
17. **if** x.next!=None:
18. x.next.previous=x
19. **if** self.head==None:
20. self.head=self.tail=x
21. x.previous=x.next=None
22. **elif** self.tail==n:
23. self.tail=x
25. #Function to remove duplicate value nodes
26. **def** deleteDuplicate(self):
27. x = self.head
28. duplicate = {}
29. **while** x **is** **not** None:
30. **if** x.value **in** duplicate:
31. **if** x.previous **is** **not** None:
32. x.previous.next = x.next
33. **if** x.next **is** **not** None:
34. x.next.previous = x.previous
35. temporary = x
36. x=x.next
37. **del** temporary
39. **else**:
40. duplicate [x.value] = True
41. x = x.next
43. **def** display(self):
44. values=[]
45. n=self.head
46. **while** n!=None:
47. values.append(str(n.value))
48. n=n.next
49. **print** ("List: ",",".join(values))
51. **if** \_\_name\_\_ == '\_\_main\_\_':
52. l=List()
53. l.insert(None, Node(4))
54. l.insert(l.head,Node(6))
55. l.insert(l.head,Node(8))
56. l.insert(l.head,Node(5))
57. l.insert(l.head,Node(3))
58. l.insert(l.head,Node(8))
59. #Removes duplicate value from list
60. l.deleteDuplicate()
62. l.display()

65. #Title: Python Double Linked List Source Code
66. #Author: Hintea, D.
67. #Date: 2017
68. #Availability: http://cumoodle.coventry.ac.uk

Question 10

1. **class** BinTreeNode(object):
3. **def** \_\_init\_\_(self, value):
4. self.value=value
5. self.left=None
6. self.right=None
8. **def** tree\_insert( tree, item):
9. **if** tree==None:
10. tree=BinTreeNode(item)
11. **else**:
12. **if**(item < tree.value):
13. **if**(tree.left==None):
14. tree.left=BinTreeNode(item)
15. **else**:
16. tree\_insert(tree.left,item)
17. **else**:
18. **if**(tree.right==None):
19. tree.right=BinTreeNode(item)
20. **else**:
21. tree\_insert(tree.right,item)
22. **return** tree
24. **def** postorder(tree):
25. **if**(tree.left!=None):
26. postorder(tree.left)
27. **if**(tree.right!=None):
28. postorder(tree.right)
29. **print** (tree.value)

32. **def** in\_order(tree):
33. **if**(tree.left!=None):
34. in\_order(tree.left)
35. **print** (tree.value)
36. **if**(tree.right!=None):
37. in\_order(tree.right)
39. **if** \_\_name\_\_ == '\_\_main\_\_':
41. t=tree\_insert(None,6);
42. tree\_insert(t,10)
43. tree\_insert(t,5)
44. tree\_insert(t,2)
45. tree\_insert(t,3)
46. tree\_insert(t,4)
47. tree\_insert(t,11)
48. in\_order(t)
50. #Function to delete node
51. **def** delete\_node(self, node):
52. **if** node==None:
53. **print** ("Node to be deleted is not found in BST")
54. **return** None
55. **if** node.parent==None:
56. self.tree=None
57. **return** None
59. tree\_parent=tree.parent
61. #CASE 1 - There are no children in the node
62. **if** tree\_children==0:
63. **if** tree\_parent.left\_child==node:
64. tree\_parent.left\_child=None
65. **else**:
66. tree\_parent.right\_child=None

69. #CASE 2 - Node has 1 child
70. **if** tree\_children==1:
71. **if** tree.left\_child!=None:
72. child=tree.left\_child
73. **else**:
74. child=tree.right\_child
75. **if** tree\_parent.left\_child==tree:
76. tree\_parent.left\_child=child
77. **else**:
78. tree\_parent.right\_child=child
79. child.parent=tree\_parent
81. #CASE 3 - Node has 2 children
82. **if** tree\_children==2:
83. **if** tree.left\_child>tree.right\_child
84. tree\_parent=tree.left\_child
85. **else**:
86. tree.right\_>tree.left\_child
87. tree\_parent=tree.right\_child
89. #Title: Binary Tree Search in Python
90. #Author: Hintea, D.
91. #Date: 2017
92. #Availability: http://cumoodle.coventry.ac.uk