Algorithms Notebook

Robin Scragg

ID : 22132481

September 2022

Logbook Exercise 1

Insert a 'code' cell below. In this do the following:

- line 1 - create a list named "shopping_list" with items: milk, eggs, bread, cheese, tea, coffee, rice, pasta, milk, tea (NOTE: the duplicate items are intentional)
- line 2 - print the list along with a message e.g. "This is my shopping list ..."
- line 3 - create a tuple named "shopping_tuple" with the same items
- line 4 - print the tuple with similar message e.g. "This is my shopping tuple ..."
- line 5 - create a set named "shopping_set" from "shopping_list" by using the set() method
- line 6 - print the set with appropriate message and check duplicate items have been removed
- line 7 - make a dictionary "shopping_dict" - copy and paste the following items and prices: "milk": "£1.20", "eggs": "£0.87", "bread": "£0.64", "cheese": "£1.75", "tea": "£1.06", "coffee": "£2.15", "rice": "£1.60", "pasta": "£1.53".
- line 8 - print the dictionary with an appropriate message

An example of fully described printed output is presented below (some clues here also) Don't worry of your text output is different - it is the contents of the compund variables that matter

```
This is my shopping list ['milk', 'eggs', 'bread', 'cheese', 'tea', 'coffee', 'ric
e', 'pasta', 'milk', 'tea']
This is my shopping tuple ('milk', 'eggs', 'bread', 'cheese', 'tea', 'coffee', 'ri
ce', 'pasta', 'milk', 'tea')
This is my Shopping_set with duplicates removes {'rice', 'milk', 'pasta', 'chees
e', 'eggs', 'tea', 'bread', 'coffee'}
This is my shopping_dict {'milk': '£1.20', 'eggs': '£0.87', 'bread': '£0.64', 'che
ese': '£1.75', 'tea': '£1.06', 'coffee': '£2.15', 'rice': '£1.60', 'pasta': '£1.5
3'}
```

In [1]:

```python
shopping_list = ["milk", "eggs", "bread", "cheese", "tea", "coffee", "rice", "pasta", '
print("This is my shopping list", shopping_list)

shopping_tuple = ("milk", "eggs", "bread", "cheese", "tea", "coffee", "rice", "pasta",
print("This is my shopping tuple", shopping_tuple)

shopping_set = set(shopping_list)
print("This is my shopping set", shopping_set)

shopping_dict = {"milk": "£1.20", "eggs": "£0.87", "bread": "£0.64", "cheese": "£1.75",
print("This is my shopping dictionary", shopping_dict)
```

```
This is my shopping list ['milk', 'eggs', 'bread', 'cheese', 'tea', 'coffe
e', 'rice', 'pasta', 'milk', 'tea']
This is my shopping tuple ('milk', 'eggs', 'bread', 'cheese', 'tea', 'coffe
e', 'rice', 'pasta', 'milk', 'tea')
This is my shopping set {'coffee', 'pasta', 'bread', 'eggs', 'milk', 'rice',
'tea', 'cheese'}
This is my shopping dictionary {'milk': '£1.20', 'eggs': '£0.87', 'bread':
'£0.64', 'cheese': '£1.75', 'tea': '£1.06', 'coffee': '£2.15', 'rice': '£1.6
0', 'pasta': '£1.53'}
```

Logbook Exercise 2

Create a 'code' cell below. In this do the following:

- line 1 - Use a comment to title your exercise - e.g. "Unit 2 Exercise"
- line 2 - create a list ... li = ["USA","Mexico","Canada"]
- line 3 - append "Greenland" to the list
- l4 - print the list to de,monstrate that Greenland is attached
- l5 - remove "Greenland"
- l6 - print the list to de,monstrate that Greenland is removed
- l7 - insert "Greenland" at the beginning of the list
- l8 - print the resul of l7
- l9 - shorthand slice the list to extract the first two items - simultaneausly print the output
- l10 - use a negative index to extract the second to last item - simultaneausly print the output
- l11 - use a splitting sequence to extract the middle two items - simultaneausly print the output

An example of fully described printed output is presented below (some clues here also) Don't worry of your text output is different - it is the contents of the list that matter

```
li.append('Greenland') gives ...  ['USA', 'Mexico', 'Canada', 'Greenland']
li.remove('Greenland') gives ...  ['USA', 'Mexico', 'Canada']
li.insert(0,'Greenland') gives ...  ['Greenland', 'USA', 'Mexico', 'Canada']
li[:2] gives ...  ['Greenland', 'USA']
li[-2] gives ...  Mexico
li[1:3] gives ...  ['USA', 'Mexico']
```

In [2]:

```python
# Unit 2 Exercise
li = ["USA", "Mexico", "Canada"]
li.append("Greenland")
print(li)
li.remove("Greenland")
print(li)
li.insert(0,"Greenland")
print(li)
print(li[:2])
print(li[-2])
print(li[1:3])
```

```
['USA', 'Mexico', 'Canada', 'Greenland']
['USA', 'Mexico', 'Canada']
['Greenland', 'USA', 'Mexico', 'Canada']
['Greenland', 'USA']
Mexico
['USA', 'Mexico']
```

Logbook Exercise 3

Create a 'code' cell below. In this do the following:

- on the first line create the following set ... a=[0,1,2,3,4,5,6,7,8,9,10]
- on the second line create the following set ... b=[0,5,10,15,20,25]
- on the third line create the following dictionary ... topscores={"Jo":999, "Sue":987, "Tara":960; "Mike":870}
- use a combination of print() and type() methods to produce the following output

```
list a is ...  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list b is ...  [0, 5, 10, 15, 20, 25]
The type of a is now ... <class 'list'>
```

- on the next 2 lines convert list a and b to sets using set()
- on the following lines use a combination of print(), type() and set notaion (e.g. 'a & b', 'a | b', 'b-a') to obtain the following output

```
set a is ...  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
set b is ...  {0, 5, 10, 15, 20, 25}
The type of a is now ... <class 'set'>
Intersect of a and b is [0, 10, 5]
Union of a and b is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25]
Items unique to set b are {25, 20, 15}
```

- on the next 2 lines use print(), '.keys()' and '.values()' methods to obtain the following output

```
topscores dictionary keys are dict_keys(['Jo', 'Sue', 'Tara', 'Mike'])
topscores dictionary values are dict_values([999, 987, 960, 870])
```

In [16]:

```python
a = [0,1,2,3,4,5,6,7,8,9,10]
b=[0,5,10,15,20,25]
topscores={"Jo":999, "Sue":987, "Tara":960, "Mike":870}
print("list a is ... ", a)
print("list b is ... ", b)
print("The type of a is now ... ",type(a))
a = set(a)
b = set(b)
print("set a is ... ", a)
print("set b is ... ", b)
print("The type of a is now ... ", type(a))
print("Insersect of a and b is ", list(a & b))
print("Union of a and b is ", list(a | b))
print("Items unique to set b are", b - a)
print("topscore dictionary keys are ", topscores.keys())
print("topscores dictionary values are ", topscores.values())
```

```
list a is ...  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list b is ...  [0, 5, 10, 15, 20, 25]
The type of a is now ...  <class 'list'>
set a is ...  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
set b is ...  {0, 5, 10, 15, 20, 25}
The type of a is now ...  <class 'set'>
Insersect of a and b is  [0, 10, 5]
Union of a and b is  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25]
Items unique to set b are {25, 20, 15}
topscore dictionary keys are  dict_keys(['Jo', 'Sue', 'Tara', 'Mike'])
topscores dictionary values are  dict_values([999, 987, 960, 870])
```

Logbook Exercise 4

Create a 'code' cell below. In this do the following:

- Given the following 4 lists of names, house number and street addresses, towns and postcodes ...

  ```
  ["T Cruise","D Francis","C White"]
  ["2 West St","65 Deadend Cls","15 Magdalen Rd"]
  ["Canterbury", "Reading", "Oxford"]
  ["CT8 23RD", "RG4 1FG", "OX4 3AS"]
  ```

- write a Custom 'address_machine' function that formats 'name', 'hs_number_street', 'town', 'postcode' with commas and spaces between items
- create a 'newlist' that repeatedly calls 'address_machine' and 'zips' items from the 4 lists
- write a 'for loop' that iterates over 'new list' and prints each name and address on a separate line
- the output should appear as follows

  ```
  T Cruise, 2 West St, Canterbury, CT8 23RD
  D Francis, 65 Deadend Cls, Reading, RG4 1FG
  C White, 15 Magdalen Rd, Oxford, OX4 3AS
  ```

- HINT: look at "# CUSTOM FUNCTION WORKED EXAMPLES 3 & 4" above

In [18]:

```
name = ["T Cruise","D Francis","C White"]
hs_number_street = ["2 West St","65 Deadend Cls","15 Magdalen Rd"]
town = ["Canterbury", "Reading", "Oxford"]
postcode = ["CT8 23RD", "RG4 1FG", "OX4 3AS"]

def address_machine (name, hs_number_street, town, postcode):
    newlist = "{0} , {1} , {2} , {3}".format(name, hs_number_street, town, postcode)
    return newlist

newlist = [address_machine(name, hs_number_street, town, postcode) for name, hs_number_

for item in newlist:
    print(item)
```

```
T Cruise , 2 West St , Canterbury , CT8 23RD
D Francis , 65 Deadend Cls , Reading , RG4 1FG
C White , 15 Magdalen Rd , Oxford , OX4 3AS
```

Logbook Exercise 5

Create a 'code' cell below. In this do the following:

- Create a super class "Person" that takes three string and one integer parameters for first and second name, UK Postcode and age in years.
- Give "Person" a method "greeting" that prints a statement along the lines "Hello, my name is Freddy Jones. I am 22 years old and my postcode is HP6 7AJ"
- Create a "Student" class that extends/inherits "Person" and takes additional parameters for degree_subject and student_ID.
- give "Student" a "studentGreeting" method that prints a statement along the lines "My student ID is SN123456 and I am reading Computer Science"
- Use either Python {} format or C-type %s/%d notation to format output strings
- Create 3 student objects and persist these in a list
- Iterate over the three objects and call their "greeting" and "studentGreeting" methods
- Output should be along the lines of the following

```
Hello, my name is Dick Turpin. I am 32 years old and my postcode is HP11 2JZ
My student ID is DT123456 and I am reading Highway Robbery

Hello, my name is Dorothy Turpin. I am 32 years old and my postcode is SO14 7AA
My student ID is DT123457 and I am reading Law

Hello, my name is Oliver Cromwell. I am 32 years old and my postcode is OX35 14RE
My student ID is OC123456 and I am reading History
```

In [14]:

```python
class Person:
    def __init__(self, first_name, second_name, postcode, age):
        self.fn = first_name
        self.sn = second_name
        self.pc = postcode
        self.yr = age

    def greeting(self):
        print("Hello my name is " + self.fn + " " + self.sn + " I am " + str(self.yr) +

class Student(Person):
    def __init__(self, first_name, second_name, postcode, age, degree_subject, student_
        super().__init__(first_name, second_name, postcode, age)
        self.ds = degree_subject
        self.id = student_ID
    def studentGreeting(self):
        print("My student ID is " + self.id + " and I am studying " + self.ds)

student1 = Student("Dick", "Turpin", "HP11 2JZ", 32, "Highway Robbery", "DT123456")
student2 = Student("Dorothy", "Turpin", "SO14 7AA", 32, "Law", "DT123457")
student3 = Student("Oliver", "Cromwell", "OX35 14RE", 32, "History", "OC123456")

students = [student1, student2, student3]

for student in students:
    student.greeting()
    student.studentGreeting()
```

```
Hello my name is Dick Turpin I am 32 years old  and my postcode is HP11 2JZ
My student ID is DT123456 and I am studying Highway Robbery
Hello my name is Dorothy Turpin I am 32 years old  and my postcode is SO14 7
AA
My student ID is DT123457 and I am studying Law
Hello my name is Oliver Cromwell I am 32 years old  and my postcode is OX35
14RE
My student ID is OC123456 and I am studying History
```

Logbook Exercise 6

Insert a 'code' cell below. In this do the following:

- 1 - Define a Node class and a (singly) LinkedList class.
- 2 - Now create five instances (objects) of the Node class and 'add' them to an instance of the LinkedList. Print this data to the screen to check that the node objects have been added correctly from.
- 3 - Test the search method. Check that a node can be found (print an appropriate message to the screen). Also check that a message is displayed when you search for a node that is not in the list.
- 4 - Test that you can remove both the start of the list and a node in the middle of the list. Print before and after to check that this works.
- 5 - Now add another attribute to the Node class. Remove all the older nodes still remaining in the list, and add new node objects with values for this new attribute.

- 6 - Now amend (or overload if necessary) the exisiting search method so that you can search for nodes via either of the two attributes. Test that this works as expected.
- 7 - Notice that you have an 'insert_beginning' function in the LinkedList class, which 'prepends' a new node to the start of the list. Now write an 'insert_end' function that will 'append' a new node to the end of the list.
- 8 - now add additional node instances to the list and check that adds them to the end of the list, rather than at the start of the list.
- 9 - Implement the updates suggested to modify the singly linked list to be a doubly linked list. Don't forget to update the insert_end method from the previous point! Check that you can navigate back and forth in the doubly linked list.

In [7]:

```
class Node:
  def __init__(self, value, value2, next_node=None, prev_node=None):
    self.value = value
    self.next_node = next_node
    self.prev_node = prev_node
    self.value2 = value2

  def get_value(self):
    return self.value

  def get_value2(self):
    return self.value2

  def get_next_node(self):
    return self.next_node

  def set_next_node(self, next_node):
    self.next_node = next_node

  def get_prev_node(self):
    return self.prev_node

  def set_prev_node(self, prev_node):
    self.prev_node = prev_node


class LinkedList:
  def __init__(self, value=None, value2=None):
    self.head_node = Node(value, value2)
    self.tail_node = Node(value, value2)

  def get_head_node(self):
     return self.head_node

  def get_tail_node(self):
     return self.tail_node

  def insert_beginning(self, new_value, new_value2):
    new_head = Node(new_value, new_value2)
    current_head = self.head_node

    if current_head != None:
      current_head.set_prev_node(new_head)
      new_head.set_next_node(current_head)

    self.head_node = new_head

    if self.tail_node == None:
      self.tail_node = new_head

  def insert_end(self, new_value, new_value2):
    new_tail = Node(new_value, new_value2)
    current_tail = self.tail_node

    if current_tail != None:
      current_tail.set_next_node(new_tail)
      new_tail.set_prev_node(current_tail)

    self.tail_node = new_tail
```

```

    if self.head_node == None:
      self.head_node = new_tail

  def stringify_list(self):
    string_list = ""
    current_node = self.get_head_node()
    while current_node:
      if current_node.get_value() != None:
        string_list += str(current_node.get_value()) + " " + str(current_node.get_valu
      current_node = current_node.get_next_node()
    return string_list

  def find_node(self, value_to_find):
    current_node = self.get_head_node()
    found = False
    while current_node:
        if current_node.get_value() == value_to_find or current_node.get_value2() == v
            found = True
            current_node = None
        else:
          current_node = current_node.get_next_node()
    if found == True:
        print("Found the value: "+ str(value_to_find))
    else:
        print("Cannot find the value: " + str(value_to_find))

  def remove_head(self):
    removed_head = self.head_node

    if removed_head == None:
      return None

    self.head_node = removed_head.get_next_node()

    if self.head_node != None:
      self.head_node.set_prev_node(None)

    if removed_head == self.tail_node:
      self.remove_tail()

    return removed_head.get_value()


  def remove_tail(self):
    removed_tail = self.tail_node

    if removed_tail == None:
      return None

    self.tail_node = removed_tail.get_prev_node()

    if self.tail_node != None:
      self.tail_node.set_next_node(None)

    if removed_tail == self.head_node:
      self.remove_head()

    return removed_tail.get_value()

```

```python
  def remove_by_value(self, value_to_remove):
    if type(value_to_remove) == int:
        value_to_remove = str(value_to_remove)
    print("Removing: " + value_to_remove)
    node_to_remove = None
    current_node = self.head_node

    while current_node != None:
      if current_node.get_value() == value_to_remove:
        node_to_remove = current_node
        break

      current_node = current_node.get_next_node()

    if node_to_remove == None:
      return None

    if node_to_remove == self.head_node:
      self.remove_head()
    elif node_to_remove == self.tail_node:
      self.remove_tail()
    else:
      next_node = node_to_remove.get_next_node()
      prev_node = node_to_remove.get_prev_node()
      next_node.set_prev_node(prev_node)
      prev_node.set_next_node(next_node)

    return node_to_remove


linkedList = LinkedList("apple", 5)
linkedList.insert_beginning("banana", 10)
linkedList.insert_beginning("pear", 8)
linkedList.insert_beginning("orange", 7)
linkedList.insert_beginning("mango", 4)

print(linkedList.stringify_list())

linkedList.find_node("mango")
linkedList.find_node("kiwi")

print(linkedList.stringify_list())
linkedList.remove_by_value("mango")
print(linkedList.stringify_list())

print(linkedList.stringify_list())
linkedList.remove_by_value("pear")
print(linkedList.stringify_list())

linkedList.find_node(5)

linkedList.insert_end("guava", 1)
linkedList.insert_end("lemon", 3)
print(linkedList.stringify_list())
```

```
mango 4
orange 7
pear 8
banana 10
apple 5
```

```
Found the value: mango
Cannot find the value: kiwi
mango 4
orange 7
pear 8
banana 10
apple 5

Removing: mango
orange 7
pear 8
banana 10
apple 5

orange 7
pear 8
banana 10
apple 5

Removing: pear
orange 7
banana 10
apple 5

Found the value: 5
orange 7
banana 10
apple 5
```

Logbook Exercise 7

Insert a 'code' cell below. In this do the following:

- 1 - Write a function that uses a random number generator to populate a list with values between 0 and 9.

  Return the populated array and print to screen. e.g: [7, 5, 1, 4, 9, 2, 6, 8, 0, 9]
- 2 - Now pass this array to the Bubble Sort method and

  print out the list to check it has been sorted.
- 3 - Now call the linear search function to check that you can find a number that has been randomly chosen within your list.
- 4 - Apply the binary search function to check that it can find the same value.
- 5 - Now reuse the function you wrote in step 1 and scale the size to 10,000 integers.

  DO NOT attempt to print the individual numbers to screen, but do check that the size of the array is 10,000 elements (hint: use the len() method)
- 6 - Now time how long it takes to run the bubble sort algorithm.

  Start the timer in the statement before calling the OPTIMISED bubble_sort method.

  Then stop the timer in the statement after the bubble_sort() method call.

  Output how long it took.
- 7 - Now generate a new unsorted list of 10,000 integers with values ranging from 0-9,

  and time how long it takes to sort using the OPTIMISED version of the Bubble sort algorithm.

Output the time to the screen and compare against the UNOPTIMISED version.

- 8 - Now generate a new unsorted list of 10,000 integers with values ranging from 0-9,

  and time how long it takes to sort using the MERGE SORT algorithm.

  Output the time to the screen and compare against the optimised and unoptimised bubble sort algorithms.
- 9 - Now that you have sorted the array of 10,000 integers. Change the value of element 9999 to the value 999 in the array. e.g. list[9999] = 999.

  Time how long it takes for the Binary Search method to find this 999 value.

  Also time how long it takes the Linear Search algorithm to find this 999 value.

  Output these times to the screen and observe if or how they differ.

In [4]:

```python
import random
import time

def randomArray(array_size):
    array = [random.randint(0,9) for i in range(array_size)]
    return array

def swap(arr, index_1, index_2):
  temp = arr[index_1]
  arr[index_1] = arr[index_2]
  arr[index_2] = temp

def bubble_sort_unoptimized(arr):
  iteration_count = 0
  for el in arr:
    for index in range(len(arr) - 1):
      iteration_count += 1
      if arr[index] > arr[index + 1]:
        swap(arr, index, index + 1)
  return arr

def bubble_sort_optimized(arr):
  iteration_count = 0
  for i in range(len(arr)):
    # iterate through unplaced elements
    for idx in range(len(arr) - i - 1):
      iteration_count += 1
      if arr[idx] > arr[idx + 1]:
        # replacement for swap function
        arr[idx], arr[idx + 1] = arr[idx + 1], arr[idx]
  return arr

def linear_search(search_list, target_value):
  for idx in range(len(search_list)):
    if search_list[idx] == target_value:
      return idx
  raise ValueError("{0} not in list".format(target_value))

def binary_search(sorted_list, target):
  left_pointer = 0
  right_pointer = len(sorted_list)

  # fill in the condition for the while loop
  while left_pointer < right_pointer:
    # calculate the middle index using the two pointers
    mid_idx = (left_pointer + right_pointer) // 2  # floor division
    mid_val = sorted_list[mid_idx]
    if mid_val == target:
      return target
    if target < mid_val:
      # set the right_pointer to the appropriate value
      right_pointer = mid_idx
    if target > mid_val:
      # set the left_pointer to the appropriate value
      left_pointer = mid_idx + 1

  return "Value not in list"

def merge_sort(items):
```

```
  if len(items) <= 1:
    return items

  middle_index = len(items) // 2
  left_split = items[:middle_index]
  right_split = items[middle_index:]

  left_sorted = merge_sort(left_split)
  right_sorted = merge_sort(right_split)

  return merge(left_sorted, right_sorted)

def merge(left, right):
  result = []

  while (left and right):
    if left[0] < right[0]:
      result.append(left[0])
      left.pop(0)
    else:
      result.append(right[0])
      right.pop(0)

  if left:
    result += left
  if right:
    result += right

  return result

array = randomArray(10)
print(array)
print()
sorted_array = bubble_sort_optimized(array)
print(sorted_array)
print()
random_number = array[random.randint(0,len(array) - 1)]
print(str(random_number) + " was found at index value " + str(linear_search(array, ran
print()
print(str(binary_search(sorted_array, random_number)) + " was found")
print()
array2 = randomArray(10000)
print(len(array2))
print()
time_start = time.time()
bubble_sort_optimized(array2)
time_end = time.time()
total_time = time_end - time_start
print("Optimized bubble sort took " + str(total_time) + " seconds")
print()
array3 = randomArray(10000)
time_start2 = time.time()
bubble_sort_unoptimized(array3)
time_end2 = time.time()
total_time2 = time_end2 - time_start2
print("Unoptimized bubble sort took " + str(total_time2) + " seconds")
time_start3 = time.time()
bubble_sort_optimized(array3)
time_end3 = time.time()
total_time3 = time_end3 - time_start3
print("Optimized bubble sort took " + str(total_time3) + " seconds")
```

```python
print()
array4 = randomArray(10000)
time_start4 = time.time()
sorted_array2 = merge_sort(array4)
time_end4 = time.time()
total_time4 = time_end4 - time_start4
print("Merge sort took " + str(total_time4) + " seconds")
time_start5 = time.time()
bubble_sort_unoptimized(array4)
time_end5 = time.time()
total_time5 = time_end5 - time_start5
print("Unoptimized bubble sort took " + str(total_time5) + " seconds")
time_start6 = time.time()
bubble_sort_optimized(array4)
time_end6 = time.time()
total_time6 = time_end6 - time_start6
print("Optimized bubble sort took " + str(total_time6) + " seconds")
print()
sorted_array2[9999]=999
time_start7 = time.time()
print(str(binary_search(sorted_array2, 999)) + " was found")
time_end7 = time.time()
total_time7 = time_end7 - time_start7
print("Binary search took " + str(total_time7) + " seconds")
time_start8 = time.time()
print("999 was found at index value " + str(linear_search(sorted_array2, 999)))
time_end8 = time.time()
total_time8 = time_end8 - time_start8
print("Linear search took " + str(total_time8) + " seconds")
```

```
[1, 2, 8, 4, 0, 3, 6, 4, 6, 8]

[0, 1, 2, 3, 4, 4, 6, 6, 8, 8]

4 was found at index value 4

4 was found

10000

Optimized bubble sort took 7.2681334018707275 seconds

Unoptimized bubble sort took 12.677436351776123 seconds
Optimized bubble sort took 4.75063157081604 seconds

Merge sort took 0.13599395751953125 seconds
Unoptimized bubble sort took 13.374462604522705 seconds
Optimized bubble sort took 4.698065519332886 seconds

999 was found
Binary search took 0.0 seconds
999 was found at index value 9999
Linear search took 0.0 seconds
```

Logbook Exercise 8

Insert a 'code' cell below. In this do the following:

- 1 - Instantiate the Stack class (copy from above).
- Push five integer values onto the stack object.

- Check that you can also 'Pop' these items off the stack and display their values to screen. What do you notice about the order of the items?

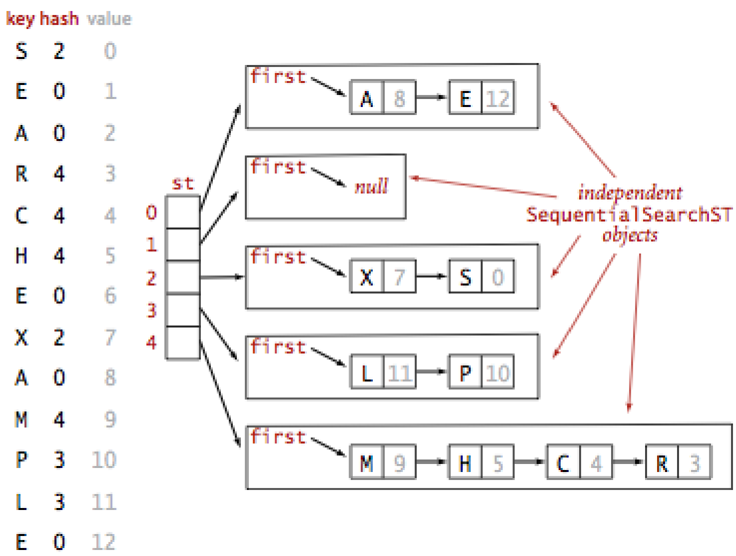
- 2 - Instantiate the Queue class -
- check that you can successfully insert (enqueue),
- and remove (dequeue) items.
- Also check the order of the removal (dequeue).
- Is the First In First Out principle implemented correctly.

- 3 - Now instantiate the HashMap class above.
- Create a Person class that has attributes for first_name, second_name and phone_number,
- and appropriate methods that get, set and print values for these attributes.
- Now instantiate

- 4 - Now create a new class called LinkedStack.
- This class should be able to merge the functionality of the original LinkedList class and Stack class.
- Each object added to the LinkedStack, should be a separate linkedlist. So that when one object is 'popped' from the stack, you can also print the entire list from this object returned from the stack.

- 5 - Now do the same for the Queue - create a LinkedQueue class that enables items stored in the queue to link the remainder of other linkedlists.

- 6 - Extend the hash_map class to create a linked list in each element position where there is a collision.
- A colliding item should be added to the end of a linked list for that index position.



Hashing with separate chaining for standard indexing client

In [1]:

```
class Node:
  def __init__(self, value, next_node=None):
    self.value = value
    self.next_node = next_node

  def get_value(self):
    return self.value

  def get_next_node(self):
    return self.next_node

  def set_next_node(self, next_node):
    self.next_node = next_node


class Stack:
  def __init__(self, limit=1000):
    self.top_item = None
    self.size = 0
    self.limit = limit

  def push(self, value):
    if self.has_space():
      item = Node(value)
      item.set_next_node(self.top_item)
      self.top_item = item
      self.size += 1
      print("Adding {} to the stack!".format(value))
    else:
      print("No room for {}!".format(value))

  def pop(self):
    if not self.is_empty():
      item_to_remove = self.top_item
      self.top_item = item_to_remove.get_next_node()
      self.size -= 1
      print("Delivering " + str(item_to_remove.get_value()))
      return item_to_remove.get_value()
    print("All out of items.")

  def peek(self):
    if not self.is_empty():
      return self.top_item.get_value()
    print("Nothing to see here!")

  def has_space(self):
    return self.limit > self.size

  def is_empty(self):
    return self.size == 0

class Queue:
  def __init__(self, max_size=None):
    self.head = None
    self.tail = None
    self.max_size = max_size
    self.size = 0

  def enqueue(self, value):
```

```
  if self.has_space():
   item_to_add = Node(value)
   print("Adding " + str(item_to_add.get_value()) + " to the queue!")
   if self.is_empty():
    self.head = item_to_add
    self.tail = item_to_add
   else:
    self.tail.set_next_node(item_to_add)
    self.tail = item_to_add
   self.size += 1
  else:
   print("Sorry, no more room!")

 def dequeue(self):
  if self.get_size() > 0:
   item_to_remove = self.head
   print(str(item_to_remove.get_value()) + " is removed from the queue!")
   if self.get_size() == 1:
    self.head = None
    self.tail = None
   else:
    self.head = self.head.get_next_node()
   self.size -= 1
   return item_to_remove.get_value()
  else:
   print("The queue is totally empty!")

 def peek(self):
  if self.is_empty():
   print("Nothing to see here!")
  else:
   return self.head.get_value()

 def get_size(self):
  return self.size

 def has_space(self):
  if self.max_size == None:
   return True
  else:
   return self.max_size > self.get_size()

 def is_empty(self):
  return self.size == 0

class LinkedList:
 def __init__(self, value=None, next_list = None):
  self.head_node = Node(value)
  self.next_list = None

 def get_head_node(self):
    return self.head_node

 def get_next_list(self):
    return self.next_list

 def set_next_list(self, next_list):
   self.next_list = next_list

 def insert_beginning(self, new_value):
  new_node = Node(new_value)
```

```
    new_node.set_next_node(self.head_node)
    self.head_node = new_node

  def stringify_list(self):
    string_list = ""
    current_node = self.get_head_node()
    while current_node:
      if current_node.get_value() != None:
        if current_node.get_next_node().get_value() != None:
           string_list += str(current_node.get_value()) + " , "
        else:
            string_list += str(current_node.get_value())
      current_node = current_node.get_next_node()
    return string_list

  def remove_node(self, value_to_remove):
    current_node = self.get_head_node()
    if current_node.get_value() == value_to_remove:
      self.head_node = current_node.get_next_node()
    else:
      while current_node:
        next_node = current_node.get_next_node()
        if next_node.get_value() == value_to_remove:
          current_node.set_next_node(next_node.get_next_node())
          current_node = None
        else:
          current_node = next_node

class HashMap:
  def __init__(self, array_size):
    self.array_size = array_size
    self.array = [None for item in range(array_size)]

  def hash(self, key, count_collisions=0):
    key_bytes = key.encode()
    hash_code = sum(key_bytes)
    return hash_code + count_collisions

  def compressor(self, hash_code):
    return hash_code % self.array_size

  def assign(self, key, value):
    array_index = self.compressor(self.hash(key))
    current_array_value = self.array[array_index]

    if current_array_value is None:
      self.array[array_index] = [key, value]
      return

    if current_array_value[0] == key:
      self.array[array_index] = [key, value]
      return

    number_collisions = 1

    while(current_array_value[0] != key):
      new_hash_code = self.hash(key, number_collisions)
      new_array_index = self.compressor(new_hash_code)
      current_array_value = self.array[new_array_index]

      if current_array_value is None:
```

```
        self.array[new_array_index] = [key, value]
        return

      if current_array_value[0] == key:
        self.array[new_array_index] = [key, value]
        return

      number_collisions += 1

    return

  def retrieve(self, key):
    array_index = self.compressor(self.hash(key))
    possible_return_object = self.array[array_index]

    if possible_return_object is None:
       return None

    if possible_return_object[0] == key:
      possible_return_object[1].print_all_attributes()
      return

    retrieval_collisions = 1

    while (possible_return_object != key):
      new_hash_code = self.hash(key, retrieval_collisions)
      retrieving_array_index = self.compressor(new_hash_code)
      possible_return_object = self.array[retrieving_array_index]

      if possible_return_object is None:
        return None

      if possible_return_object[0] == key:
        possible_return_object[1].print_all_attributes()
        return

      number_collisions += 1

    return

class Person:
  def __init__(self, first_name, second_name, phone_number):
    self.first_name = first_name
    self.second_name = second_name
    self.phone_number = phone_number

  def get_first_name(self):
    return self.first_name

  def get_second_name(self):
    return self.second_name

  def get_phone_number(self, next_node):
    return self.phone_number

  def set_first_name(self):
    self.first_name = first_name

  def set_second_name(self):
    self.second_name = second_name

```

```
  def set_phone_number(self):
    self.phone_number = phone_number

  def print_first_name(self):
    print(self.first_name)

  def print_second_name(self):
    print(self.second_name)

  def print_phone_number(self):
    print(self.phone_number)

  def print_all_attributes(self):
    print(self.first_name, " ", self.second_name, " ", self.phone_number)

class LinkedStack:
    def __init__(self, limit =1000):
        self.top_list = None
        self.size = 0
        self.limit = limit

    def createLinkedList(self):
        linkedList = LinkedList()
        linkedListSize = int(input("How many items are there in the linked list? "))
        for x in range(linkedListSize):
            value = input("Enter the value to be added to the linked list ")
            linkedList.insert_beginning(value)
        return linkedList


    def push(self):
        if self.has_space():
            linkedList = self.createLinkedList()
            linkedList.set_next_list(self.top_list)
            self.top_list = linkedList
            self.size +=1
            print("Adding to the stack ", linkedList.stringify_list())
        else:
            print("No room for another list")

    def pop(self):
        if not self.is_empty():
            list_to_remove = self.top_list
            self.top_list = list_to_remove.get_next_list()
            self.size -= 1
            print("Removing from the stack" , list_to_remove.stringify_list())
            return
        print ("The stack is empty")

    def has_space(self):
        return self.limit > self.size

    def is_empty(self):
        return self.size == 0

class LinkedQueue:
  def __init__(self, max_size=None):
    self.head = None
    self.tail = None
    self.max_size = max_size
    self.size = 0
```

```

  def createLinkedList(self):
        linkedList = LinkedList()
        linkedListSize = int(input("How many items are there in the linked list? "))
        for x in range(linkedListSize):
            value = input("Enter the value to be added to the linked list ")
            linkedList.insert_beginning(value)
        return linkedList

  def enqueue(self):
    if self.has_space():
      list_to_add = self.createLinkedList()
      print("Adding to the queue ", list_to_add.stringify_list())
      if self.is_empty():
        self.head = list_to_add
        self.tail = list_to_add
      else:
        self.tail.set_next_list(list_to_add)
        self.tail = list_to_add
      self.size += 1
    else:
      print("Sorry, no more room!")

  def dequeue(self):
    if self.get_size() > 0:
      list_to_remove = self.head
      print(list_to_remove.stringify_list() + " is removed from the queue!")
      if self.get_size() == 1:
        self.head = None
        self.tail = None
      else:
        self.head = self.head.get_next_list()
      self.size -= 1
      return
    else:
      print("The queue is totally empty!")

  def peek(self):
    if self.is_empty():
      print("Nothing to see here!")
    else:
      print(self.head.stringify_list())
      return

  def get_size(self):
    return self.size

  def has_space(self):
    if self.max_size == None:
      return True
    else:
      return self.max_size > self.get_size()

  def is_empty(self):
    return self.size == 0

stack = Stack(5)
stack.push(1)
stack.push(2)
stack.push(3)
stack.push(4)
```

```python
stack.push(5)
print()
stack.pop()
stack.pop()
stack.pop()
stack.pop()
stack.pop()
stack.pop()
print()
queue = Queue(3)
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print()
print("The first item in the queue is " + str(queue.peek()))
queue.dequeue()
print()
print("The first item in the queue is " + str(queue.peek()))
queue.dequeue()
print()
print("The first item in the queue is " + str(queue.peek()))
queue.dequeue()
print()
queue.dequeue()
print()
person1 = Person("Bob", "Smith", 812345678910)
person2 = Person("Steve", "Smith", 12434239427)
person3 = Person("Frank", "Williams", 73842912839)
hash_map = HashMap(3)
hash_map.assign("1", person1)
hash_map.assign("2", person2)
hash_map.assign("3", person3)
hash_map.retrieve("1")
hash_map.retrieve("2")
hash_map.retrieve("3")
print()
linkedStack = LinkedStack()
linkedStack.push()
linkedStack.push()
linkedStack.push()
print()
linkedStack.pop()
linkedStack.pop()
linkedStack.pop()
print()
linkedQueue = LinkedQueue()
linkedQueue.enqueue()
linkedQueue.enqueue()
linkedQueue.enqueue()
print()
linkedQueue.dequeue()
linkedQueue.dequeue()
linkedQueue.dequeue()
```

```
Adding 1 to the stack!
Adding 2 to the stack!
Adding 3 to the stack!
Adding 4 to the stack!
Adding 5 to the stack!
```

```
Delivering 5
Delivering 4
Delivering 3
Delivering 2
Delivering 1
All out of items.

Adding 1 to the queue!
Adding 2 to the queue!
Adding 3 to the queue!

The first item in the queue is 1
1 is removed from the queue!

The first item in the queue is 2
2 is removed from the queue!

The first item in the queue is 3
3 is removed from the queue!

The queue is totally empty!

Bob   Smith   812345678910
Steve   Smith   12434239427
Frank   Williams   73842912839

How many items are there in the linked list? 3
Enter the value to be added to the linked list 2
Enter the value to be added to the linked list 1
Enter the value to be added to the linked list 3
Adding to the stack  3 , 1 , 2
How many items are there in the linked list? 2
Enter the value to be added to the linked list 1
Enter the value to be added to the linked list 3
Adding to the stack  3 , 1
How many items are there in the linked list? 2
Enter the value to be added to the linked list 1
Enter the value to be added to the linked list 5
Adding to the stack  5 , 1

Removing from the stack 5 , 1
Removing from the stack 3 , 1
Removing from the stack 3 , 1 , 2

How many items are there in the linked list? 2
Enter the value to be added to the linked list 1
Enter the value to be added to the linked list 8
Adding to the queue  8 , 1
How many items are there in the linked list? 2
Enter the value to be added to the linked list 9
Enter the value to be added to the linked list 4
Adding to the queue  4 , 9
How many items are there in the linked list? 2
Enter the value to be added to the linked list 5
Enter the value to be added to the linked list 9
Adding to the queue  9 , 5

8 , 1 is removed from the queue!
4 , 9 is removed from the queue!
9 , 5 is removed from the queue!
```

Logbook Exercise 9

Insert a 'code' cell below. In this do the following:

- 1 - Implement the Binary Search Tree class as described above.
- 2 - Create a small dataset of integers
- and insert these nodes into an object of the BST class.
- For simplicity, generate a dataset which size is 'odd' so there is a natural mid point.
- Also avoid duplicate values. (Is there a Python structure which ignores duplicate values? Think back to an earlier lecture...)
- 3 - Check that you can traverse the tree in order.
- 4 - Check that you can successfully retreive (search for) a node within the Tree.
- Check positive
- and negative cases (what happens when the item does not exist).
- 5 - Now generate a small dataset of single letters.
- Generate an odd size, and avoid duplicates again.
- 6 - Think about how you will insert these nodes into the tree... can they be added in the order they were generated?
- 7 - Once you have decided how to insert these into a BST, check that these were added in alphabetical order, by printing out the tree in order.
- 8 - Study the remove_child method given above for a non-binary search tree. Are you able to use some of that code for a method that will remove a node from a BST?
- Your task is to code a remove_child method for the BST so it can remove any node from any position in the tree (leaf, parent, or root).
- You'll need to thoroughly test this to check that node references are maintained, and the correct node is made the parent.
- 9 - Challenge: Rather than printing out a vertical list of the tree nodes, can you print in the BST nodes in the arrangement shown in diagrams above?
- Are you able to format the print out of tree elements to look something like the below which shows the nodes and their edges:

```
      D
    /   \
   C     F
        / \
       E   G
```

In [13]:

```
class BinarySearchTree:
  def __init__(self, value, depth=1):
    self.value = value
    self.depth = depth
    self.left = None
    self.right = None
    self.nodes = []

  def insert(self, value):
    if (value < self.value):
      if (self.left is None):
        self.left = BinarySearchTree(value, self.depth + 1)
        print(f'Tree node {value} added to the left of {self.value} at depth {self.dept
        self.nodes.append(value)
      else:
        self.left.insert(value)
    else:
      if (self.right is None):
        self.right = BinarySearchTree(value, self.depth + 1)
        print(f'Tree node {value} added to the right of {self.value} at depth {self.dep
        self.nodes.append(value)
      else:
        self.right.insert(value)

  def get_node_by_value(self, value):
    if (self.value == value):
      return self
    elif ((self.left is not None) and (value < self.value)):
      return self.left.get_node_by_value(value)
    elif ((self.right is not None) and (value >= self.value)):
      return self.right.get_node_by_value(value)
    else:
      return None

  def depth_first_traversal(self):
    if (self.left is not None):
      self.left.depth_first_traversal()
    print(f'Depth={self.depth}, Value={self.value}')
    if (self.right is not None):
      self.right.depth_first_traversal()

  def remove_node(self, remove_node):
    print("Removing " , remove_node)
    self.nodes = [node for node in self.nodes if node is not remove_node]
    self = BinarySearchTree(self.nodes[0])
    for x in range(1,len(self.nodes)):
        self.insert(self.nodes[x])


integers = [30,38,48,11,27]
tree = BinarySearchTree(integers[0])
for x in range(1,5):
    tree.insert(integers[x])
print()

tree.depth_first_traversal()
print()
print(tree.get_node_by_value(38) , "is found")
print(tree.get_node_by_value(9))
```

```python
print()

letters= ["m", "f", "a", "x", "t"]
tree2 = BinarySearchTree(letters[0])
for x in range(1,5):
    tree2.insert(letters[x])
print()
tree2.depth_first_traversal()
tree.remove_node(27)
tree.depth_first_traversal()
```

```
Tree node 38 added to the right of 30 at depth 2
Tree node 48 added to the right of 38 at depth 3
Tree node 11 added to the left of 30 at depth 2
Tree node 27 added to the right of 11 at depth 3

Depth=2, Value=11
Depth=3, Value=27
Depth=1, Value=30
Depth=2, Value=38
Depth=3, Value=48

<__main__.BinarySearchTree object at 0x000001BA5E3D2610> is found
None

Tree node f added to the left of m at depth 2
Tree node a added to the left of f at depth 3
Tree node x added to the right of m at depth 2
Tree node t added to the left of x at depth 3

Depth=3, Value=a
Depth=2, Value=f
Depth=1, Value=m
Depth=3, Value=t
Depth=2, Value=x
Removing  27
Depth=2, Value=11
Depth=3, Value=27
Depth=1, Value=30
Depth=2, Value=38
Depth=3, Value=48
```

Logbook Exercise 10

Insert a 'code' cell below. In this do the following:

- 1 - First implement a standard (non-binary) tree class. Remember that levels in a non-binary tree can have more than two children.

- 2 - Now insert nodes with values of single letters (A-Z) in your tree at a variety of levels. For this exercise, it would be best to avoid duplicates again. At least four levels are recommended.

- 3 - Now, copy the BFS code from above,
- and modify it so that it works with your Queue class that you have written previously.

- 4 - Implement BFS on your sample tree, and direct it towards one of the nodes within your tree.
- Run this several times to check that it correctly finds the path to the target node.
- Feel free to print the tree so you can check whether it returns the correct path.

- 5 - Now, copy the DFS code from above,
- and modify it to work with your Stack class written previously.

- 6 - Implement DFS on the same dataset.
- Check this correctly finds the path to the target node specified.

In [8]:

```
class TreeNode:
  def __init__(self, value):
    self.value = value
    self.children = []

  def __repr__(self, level=0):
    ret = "--->" * level + repr(self.value) + "\n"
    for child in self.children:
      ret += child.__repr__(level+1)
    return ret

  def add_child(self, child_node):
    self.children.append(child_node)

root_node = TreeNode("B")
s = TreeNode("S")
k = TreeNode("K")
root_node.children = [s, k]
d = TreeNode("D")
r = TreeNode("R")
s.children = [d, r]
p = TreeNode("P")
l = TreeNode("L")
k.children = [p, l]
w = TreeNode("W")
f = TreeNode("F")
d.children = [w, f]
m = TreeNode("M")
v = TreeNode("V")
r.children = [m, v]
e = TreeNode("E")
c = TreeNode("C")
p.children = [e, c]
i = TreeNode("I")
o = TreeNode("O")
l.children = [i, o]
print(root_node)


class Node:
  def __init__(self, value, next_node=None):
    self.value = value
    self.next_node = next_node

  def get_value(self):
    return self.value

  def get_next_node(self):
    return self.next_node

  def set_next_node(self, next_node):
    self.next_node = next_node

class Queue:
  def __init__(self, max_size=None):
    self.head = None
    self.tail = None
    self.max_size = max_size
    self.size = 0
```

```
  def enqueue(self, value):
    if self.has_space():
      item_to_add = Node(value)
      if self.is_empty():
        self.head = item_to_add
        self.tail = item_to_add
      else:
        self.tail.set_next_node(item_to_add)
        self.tail = item_to_add
      self.size += 1
    else:
      print("Sorry, no more room!")

  def dequeue(self):
    if self.get_size() > 0:
      item_to_remove = self.head
      if self.get_size() == 1:
        self.head = None
        self.tail = None
      else:
        self.head = self.head.get_next_node()
      self.size -= 1
      return item_to_remove.get_value()
    else:
      print("The queue is totally empty!")

  def peek(self):
    if self.is_empty():
      print("Nothing to see here!")
    else:
      return self.head.get_value()

  def get_size(self):
    return self.size

  def has_space(self):
    if self.max_size == None:
      return True
    else:
      return self.max_size > self.get_size()

  def is_empty(self):
    return self.size == 0

def bfs(root_node, goal_value):

  # initialize frontier queue
  path_queue = Queue()

  # add root path to the frontier
  initial_path = [root_node]
  path_queue.enqueue(initial_path)

  # search loop that continues as long as
  # there are paths in the frontier
  while path_queue.size > 0:
    # get the next path and node
    # then output node value
    current_path = path_queue.peek()
    path_queue.dequeue()
```

```
    current_node = current_path[-1]

    # check if the goal node is found
    if current_node.value == goal_value:
      print("Path found for", goal_value)
      for node in current_path:
            print(node.value)
      return

    # add paths to children to the  frontier
    for child in current_node.children:
      new_path = current_path[:]
      new_path.append(child)
      path_queue.enqueue(new_path)

  # return an empty path if goal not found
  print ("Path not found for", goal_value)
  return None

bfs(root_node, "S")
print()
bfs(root_node, "C")
print()
bfs(root_node, "A")
print()

class Stack:
  def __init__(self, limit=1000):
    self.top_item = None
    self.size = 0
    self.limit = limit

  def push(self, value):
    if self.has_space():
      item = Node(value)
      item.set_next_node(self.top_item)
      self.top_item = item
      self.size += 1
    else:
      print("No room for {}!".format(value))

  def pop(self):
    if not self.is_empty():
      item_to_remove = self.top_item
      self.top_item = item_to_remove.get_next_node()
      self.size -= 1
      return item_to_remove.get_value()
    print("All out of items.")

  def peek(self):
    if not self.is_empty():
      return self.top_item.get_value()
    print("Nothing to see here!")

  def has_space(self):
    return self.limit > self.size

  def is_empty(self):
    return self.size == 0

def dfs(root, target, path=(), current_path = []):
```

```
  path = Stack()
  path.push(root.value)
  current_node = path.peek()
  current_path.append(current_node)

  if root.value == target:
    path.pop()
    print("Path found for", target)
    return current_path

  path.pop()
  for child in root.children:
    path.push(child)
    path_found = dfs(child, target, path)

    if path_found is not None:
      return current_path

  return None

print(dfs(root_node, "B"))
print()
print(dfs(root_node, "K"))
print()
print(dfs(root_node, "E"))
print()
print(dfs(root_node, "A"))
```

```
'B'
--->'S'
--->--->'D'
--->--->--->'W'
--->--->--->'F'
--->--->'R'
--->--->--->'M'
--->--->--->'V'
--->'K'
--->--->'P'
--->--->--->'E'
--->--->--->'C'
--->--->'L'
--->--->--->'I'
--->--->--->'O'

Path found for S
B
S

Path found for C
B
K
P
C

Path not found for A

Path found for B
['B']

Path found for K
['B', 'B', 'S', 'D', 'W', 'F', 'R', 'M', 'V', 'K']
```

```
Path found for E
['B', 'B', 'S', 'D', 'W', 'F', 'R', 'M', 'V', 'K', 'B', 'S', 'D', 'W',
'F', 'R', 'M', 'V', 'K', 'P', 'E']

None
```

Logbook Exercise 11

Your last logbook is a challenge!

Your task:

- Write a function called is_symmetrical
- that accepts a tree as an argument.
- This function should establish whether any tree (BST or non-binary) has the same number of nodes stored on each branch of the tree.
- The function should return True if the tree passed in is symmetrical, and should return False if not.

Insert a 'code' cell below and write an algorithm to solve this task. You are advised to test this function on several trees of different sizes and contents.

In [14]:

```
class TreeNode:
  def __init__(self, value):
    self.value = value
    self.children = []
    self.number_of_children = 0

  def __repr__(self, level=0):
    ret = "--->" * level + repr(self.value) + "\n"
    for child in self.children:
      ret += child.__repr__(level+1)
    return ret

  def add_child(self, child_node):
    self.children.append(child_node)
    self.number_of_children += 1

class BinarySearchTree:
  def __init__(self, value, depth=1):
    self.value = value
    self.depth = depth
    self.left = None
    self.right = None
    self.number_of_children = 0
    self.children= []

  def insert(self, value):
    if (value < self.value):
      if (self.left is None):
        self.left = BinarySearchTree(value, self.depth + 1)
        self.number_of_children +=1
        self.children.append(self.left)
        print(f'Tree node {value} added to the left of {self.value} at depth {self.dep
      else:
        self.left.insert(value)
    else:
      if (self.right is None):
        self.right = BinarySearchTree(value, self.depth + 1)
        self.number_of_children +=1
        self.children.append(self.right)
        print(f'Tree node {value} added to the right of {self.value} at depth {self.de
      else:
        self.right.insert(value)

  def get_node_by_value(self, value):
    if (self.value == value):
      return self
    elif ((self.left is not None) and (value < self.value)):
      return self.left.get_node_by_value(value)
    elif ((self.right is not None) and (value >= self.value)):
      return self.right.get_node_by_value(value)
    else:
      return None

  def depth_first_traversal(self):
    if (self.left is not None):
      self.left.depth_first_traversal()
    print(f'Depth={self.depth}, Value={self.value}')
    if (self.right is not None):
      self.right.depth_first_traversal()
```

```python

root_node = TreeNode("B")
s = TreeNode("S")
k = TreeNode("K")
root_node.add_child(s)
root_node.add_child(k)
d = TreeNode("D")
r = TreeNode("R")
s.add_child(d)
s.add_child(r)
p = TreeNode("P")
l = TreeNode("L")
k.add_child(p)
k.add_child(l)
w = TreeNode("W")
f = TreeNode("F")
d.add_child(w)
d.add_child(f)
m = TreeNode("M")
v = TreeNode("V")
r.add_child(m)
r.add_child(v)
e = TreeNode("E")
c = TreeNode("C")
p.add_child(e)
p.add_child(c)
i = TreeNode("I")
o = TreeNode("O")
l.add_child(i)
l.add_child(o)
print(root_node)

root_node2 = TreeNode("1")
node2 = TreeNode("2")
node3 = TreeNode("3")
root_node2.add_child(node2)
root_node2.add_child(node3)
node4 = TreeNode("4")
node5 = TreeNode("5")
node6 = TreeNode("6")
node2.add_child(node4)
node2.add_child(node5)
node2.add_child(node6)
node7 = TreeNode("7")
node3.add_child(node7)
print(root_node2)

tree = BinarySearchTree(48)
tree.insert(26)
tree.insert(56)
tree.insert(24)
tree.insert(12)
tree.insert(38)
tree.insert(55)
tree.insert(74)
tree.insert(88)
tree.depth_first_traversal()
print()

tree2 = BinarySearchTree(48)
tree2.insert(26)
```

```
tree2.insert(56)
tree2.insert(24)
tree2.insert(38)
tree2.insert(55)
tree2.insert(74)
tree2.depth_first_traversal()
print()

def is_symmetrical(node):
    if isinstance(node, TreeNode):
      node_children_number = node.number_of_children

      for child in node.children:
           number = child.number_of_children
           if number > 0 and number != node_children_number:
                return False

      return True

    else:
        def check_children(child):
           number = 0
           if child.left is not None :
              number += 1
           if child.right is not None:
               number += 1

           if node.number_of_children == number:
               for child in child.children:
                    return check_children(child)

           if node.number_of_children != number and number !=0:
               return False

        if (check_children(node.left) == None) and (check_children(node.right) == None
            return True
        return False





print("Tree 1 is symmetrical :", is_symmetrical(root_node))

print("Tree 2 is symmetrical :", is_symmetrical(root_node2))

print("Tree 3 is symmetrical :", is_symmetrical(tree))

print("Tree 4 is symmetrical :", is_symmetrical(tree2))
```

```
'B'
--->'S'
--->--->'D'
--->--->--->'W'
--->--->--->'F'
--->--->'R'
--->--->--->'M'
--->--->--->'V'
--->'K'
--->--->'P'
--->--->--->'E'
```

```
--->--->--->'C'
--->--->'L'
--->--->--->'I'
--->--->--->'O'

'1'
--->'2'
--->--->'4'
--->--->'5'
--->--->'6'
--->'3'
--->--->'7'

Tree node 26 added to the left of 48 at depth 2
Tree node 56 added to the right of 48 at depth 2
Tree node 24 added to the left of 26 at depth 3
Tree node 12 added to the left of 24 at depth 4
Tree node 38 added to the right of 26 at depth 3
Tree node 55 added to the left of 56 at depth 3
Tree node 74 added to the right of 56 at depth 3
Tree node 88 added to the right of 74 at depth 4
Depth=4, Value=12
Depth=3, Value=24
Depth=2, Value=26
Depth=3, Value=38
Depth=1, Value=48
Depth=3, Value=55
Depth=2, Value=56
Depth=3, Value=74
Depth=4, Value=88

Tree node 26 added to the left of 48 at depth 2
Tree node 56 added to the right of 48 at depth 2
Tree node 24 added to the left of 26 at depth 3
Tree node 38 added to the right of 26 at depth 3
Tree node 55 added to the left of 56 at depth 3
Tree node 74 added to the right of 56 at depth 3
Depth=3, Value=24
Depth=2, Value=26
Depth=3, Value=38
Depth=1, Value=48
Depth=3, Value=55
Depth=2, Value=56
Depth=3, Value=74

Tree 1 is symmetrical : True
Tree 2 is symmetrical : False
Tree 3 is symmetrical : False
Tree 4 is symmetrical : True
```

In [ ]:

```
 
```