

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5003CEM ADVANCED ALGORITHMS** | 2020

**PORTFOLIO OF CODE | REPORT**

**NAME: Saad Iftikhar**

**SID: 9789180**

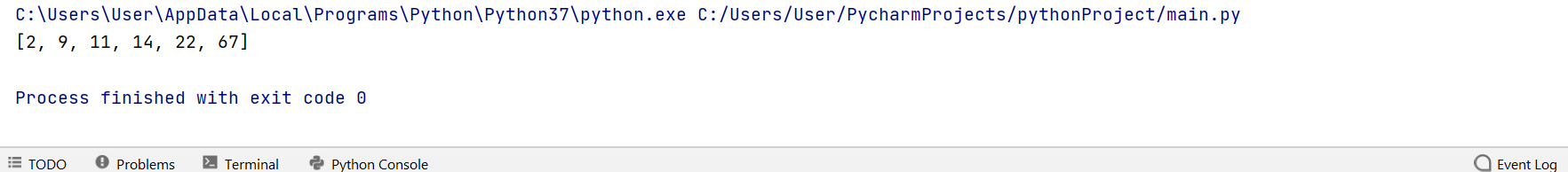
1. **STANDARD TASKS**

1.1 **Selection sort**

1. **Commented Code**

*"""SELECTION\_SORT  
 input: unsorted list  
 output: sorted list  
 Implements selection sort with integral swap  
"""*

def selection\_sort(a):  
 for i in range(len(a) - 1): *# loop - iterate through the unsorted list from start to the penultimate element.* minimum = i *# Assignment - Set minimum to i.* for j in range(i + 1, len(a)): *# inner loop - iterate through unsorted list from the current  
 # ith + 1 element to the last element.* if a[j] < a[minimum]: *# if 1st element value is less than the element to the right (2nd).* minimum = j *# Reassignment - Set minimum to j.* a[i], a[minimum] = a[minimum], a[i] *# swap - set 1st element to be the value of the 2nd and set the  
 # 2nd element to be the value of the 1st.* return a *# when the loop completes, return the sorted list.*print(selection\_sort([11, 22, 14, 67, 2, 9]))



1. **Explanation of code**

Selection sort is implemented in this code.

The sorted section and the unsorted section are included subarrays in the selection sort. It works by taking the unsorted array's minimum value, appending it to the end of the sorted array, and repeating until there is no more unsorted array.

We'll start with an array that isn't sorted. The minimum value is found and moved to the end of the unsorted array, which is initially just the first element. The sorted subarray is now the first part, and the unsorted subarray is everything else. Then, in the unsorted subarray, we find the smallest value and pass it to the end of the sorted subarray. We keep repeating this process until the whole array has been sorted. Now we return the sorted array.

1. **Critical comment**

This code is a straightforward implementation of selection sort. Still There are a few things to consider. The code implements swap without using a buffer however this method cannot be implemented by other programming languages. We achieved the swap by doing the following:

a[i], a[minimum] = a[minimum], a[i]

However, this operation, was done in another programming language it would require us to use a buffer and the code for that would be as follows.

buffer = a[i]

a[i] = a[minimum]

a[minimum] = buffer

The second issue is that this sorting algorithm checks the minimum value element by iterating through the whole list and adding it to the first element in a sorted list. Doing so may work well when working with small lists but when working with larger list this would be a time consuming and inefficient process.

Writing the same code in C++ would require us to use pointers, a buffer and the data type of variables must be specified thus we have to write more code and it would be slower to work with. I used python because it avoids all the mentioned additional steps that must be implemented in C++ increasing code readability and saving writing time. The mentioned reasons are why I’ve written the code in python then C++.

1.2 **Search method for Binary Tree class**

1. **Commented Code**

*"""   
  
 Basic BST code for inserting (i.e. building) and printing a tree  
  
 second standard viva task\*\*\* (of 5) will be to implement a find method into  
 the class BinaryTree from pseudocode. See the STD\_2 task sheet for Week 5.   
  
"""*import math *# The Maths module is imported.***""" Node class  
"""**class Node: *# A class object called "Node" is created.  
  
 # The first parameter "self" is a reference to the current instance of the class.* def \_\_init\_\_(self, data=None): *# The class is initialized with the parameter defined. The 2nd  
 # parameter is called "data" which has a default value of 'None'.* self.data = data *# Instance variable "data" is created.* self.left = None *# The default value of 'None' assigned to "self.left".* self.right = None *# The default value of 'None' assigned to "self.right".***""" BST class with insert and display methods. display pretty prints the tree  
"""**class BinaryTree: *# A class object called "BinaryTree" is created.* def \_\_init\_\_(self): *# The class is initialized with the parameter defined.* self.root = None *# The default value of 'None' assigned to the instance variable "root".* def insert(self, data): *# A function to insert data into the root,  
 # branch and leaf Nodes of the BinaryTree with the second parameter 'data'.* if self.root is None: *# It's checked that if the "self.root" is pointing towards the object 'None'.* self.root = Node(data) *# The value of the "data" instance of the "Node" class is*

*# assigned to "self.root".* else: *# Otherwise do the following.* self.\_insert(data, self.root) *# The "data" is inserted in a "Node" of BinaryTree class using the  
 # private function "\_insert".* def \_insert(self, data, cur\_node): *# A private function to insert data into the branch 'Nodes' of the  
 # BinaryTree with the parameters defined. The 2nd parameter is the "data" to be inserted and the  
 # third parameter is initially the "root Node" in our BinaryTree.* if data < cur\_node.data: *# if the data to be inserted in the Node is less than the current  
 # selected Node's data.* if cur\_node.left is None: *# It's checked that if the left branch or leaf of the current node is  
 # pointing towards the object 'None'.* cur\_node.left = Node(data) *# The left branch or leaf of the current 'Node' is assigned the  
 # 'Node(data)' value.* else: *# Otherwise do the following.* self.\_insert(data, cur\_node.left) *# The private function "\_insert" calls is called  
 # recursively. with 2 arguments, the value of the selected "data" variable and  
 # "cur\_node.left"/(the left of the current Node) until the data has been placed inside a  
 # Node with 'None' value on a left branch or leaf of the BinaryTree.* elif data > cur\_node.data: *# if the data to be inserted in the Node is greater*

*# than the current selected Node's data.* if cur\_node.right is None: *# It's checked that if the right branch or leaf of the current 'Node' is  
 # pointing towards the object 'None'.* cur\_node.right = Node(data) *# The right branch or leaf of the current node is assigned the Node*

*# data value.* else: *# Otherwise do the following.* self.\_insert(data,  
 cur\_node.right) *# The private function \_insert calls is called recursively.  
 # with 2 arguments, data and cur\_node.right until the data has been placed inside a*

*# Node with 'None'  
 # value on a right branch or leaf of the BinaryTree.* else: *# Otherwise do the following.* print(data, **"Already present in tree"**) *# print the data to be inserted is already present in the*

*# tree. The same element can't be entered more than once.* def find\_i(self, data): *# A function to search the entered data iteratively from the Binarytree.  
 # The 2nd parameter is the data to be checked.* cur\_node = self.root *# The current node is given the value of the node self.root.* while self.root: *# While self.root is True/(has a value other than '0' or 'None')* if data == cur\_node.data: *# If the value of data is equal to the value of the current selected Nodes data.* return True *# return True* elif data < cur\_node.data: *# If the value of data is lees to the value of the current selected Nodes data.* cur\_node = cur\_node.left *# Current selected Node is changed to the Node to its left.* elif cur\_node.right is not None: *# If the value of data is equal to the value of the current  
 # selected Nodes data.* cur\_node = cur\_node.right *# Current selected Node is changed to the Node to its right.* else: *# otherwise* return False *# return Flase* def find\_r(self, data): *# A function to search the entered data recursively from the Binarytree.*

*# The 2nd parameter is the data to be checked.* if self.root: *# While self.root is True/(has a value other than '0' or 'None')* search = self.\_find\_r(data, self.root) *# The search variable is assigned the recursive result of*

*# the private function \_find\_r. The First argument being the value of the*

*# given data and the 2nd argument being the value of self.root.* if search: *# if search is True / (has a value other than '0' or 'None')* return True *# return True* return False *# if search is False return Flase* else: *# While self.root is False return Flase* return None *# return 'None'* def \_find\_r(self, data, cur\_node): *# A function to search the entered data recursively from the Binarytree.  
 # The 2nd parameter is the data to be checked. The third parameter is the current selected Node.*

if data == cur\_node.data: *# If the value of data is equal to the value of the current*

*# selected Nodes data.* return True *# return True* elif data < cur\_node.data and cur\_node.left: *# The value of the data variable is less than*

*# the current Selected* return self.\_find\_r(data, cur\_node.left) *# return the private function \_find\_r recursively. The  
 # 1st argument is the data to be checked. The 2nd argument is the Node left to the*

*# current selected Node.* elif data > cur\_node.data and cur\_node.right: *# The value of the data variable is greater than the  
 # current Selected* return self.\_find\_r(data, cur\_node.right) *# return the private function \_find\_r*

*# recursively. The 1st argument is the data to be checked. The 2nd argument*

*# is the Node right to the current selected Node.*

def display(self, cur\_node): *# A function to display the current selected Node, all*

*# the left and right Nodes with data present in them. It has to parameters* lines, \_, \_, \_ = self.\_display(cur\_node) *# The variable is created and assigned to*

*# recursively call the self.\_display(cur\_node)private function. The last three*

*# value's returned by the call are ignored in each call.* for line in lines: *# going through each line(value) in the variable lines.* print(line) *# print each line(value)* def \_display(self, cur\_node): *# The first parameter being self is a reference to*

*# the current instance of the class and the 2nd being the currently selected Node.* if cur\_node.right is None and cur\_node.left is None: *# if both the right and the left*

*# Node to the current selected Node is the object 'None'.* line = **'%s'** % cur\_node.data *# The line variable is created and assigned with the*

*# cur\_node.data value after it has been converted to a string.* width = len(line) *# The width variable created and assigned with the length of*

*# the value assigned to the line variable.* height = 1 *# The height variable created with the value 1 assigned to it.* middle = width // 2 *# The middle variable created and assigned with the*

*# width variable's value floor divided by 2.* return [line], width, height, middle *# The line variable’s value is converted to a list type, the  
 # width variable’s value, the height variables value and the middle variables value is returned.* if cur\_node.right is None: *# if the right Node to the current selected Node is the object 'None'.* lines, n, p, x = self.\_display(cur\_node.left) *# The variables lines, n, p, x are assigned the value returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value  
 # after it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the value*

*# assigned to the line variable.* first\_line = (x + 1) \* **' '** + (n - x - 1) \* **'\_'** + s *# basic addition, subtraction and multiplication*

*# is done. The result of this calculation is a string value assigned to the variable first\_line.* second\_line = x \* **' '** + **'/'** + (n - x - 1 + u) \* **' '** *# basic addition, subtraction, and*

*# multiplication is done. The result of this calculation is a string value assigned to the variable*

*# second\_line.* shifted\_lines = [line + u \* **' '** for line in lines] *# basic addition and multiplication is done.*

*# The result of this calculation is a string value assigned to the variable shifted\_lines.* return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, n + u // 2 *# Values of the written*

*# variables and calculated values are returned.* if cur\_node.left is None: *# if the left Node to the current selected Node is the object 'None'.* lines, n, p, x = self.\_display(cur\_node.right) *# The variables lines, n, p, x*

*# are assigned the vaule returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value*

*# after it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the value*

*# assigned to the line variable.* first\_line = s + x \* **'\_'** + (n - x) \* **' '** *# basic addition, subtraction and multiplication*

*# is done. The result of this calculation is a string value assigned to the variable first\_line.* second\_line = (u + x) \* **' '** + **'**\\**'** + (n - x - 1) \* **' '** *# basic addition, subtraction,*

*# and multiplication is done. The result of this  
 # calculation is a string value assigned to the variable second\_line.* shifted\_lines = [u \* **' '** + line for line in lines] *# basic addition and multiplication is done. The*

*# result of this calculation is a string value assigned to the variable shifted\_lines.* return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, u // 2 *# Values of the written*

*# variables and calculated values are returned.* left, n, p, x = self.\_display(cur\_node.left) *# The variables left, n, p, x is assigned the*

*# value returned by function call.* right, m, q, y = self.\_display(cur\_node.right) *# The variables right, m, q, y is assigned the*

*# value returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value after  
 # it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the*

*# value assigned to the line variable.* first\_line = (x + 1) \* **' '** + (n - x - 1) \* **'\_'** + s + y \* **'\_'** + (m - y) \* **' '** *# basic addition,*

*# subtraction and multiplication are done. The result of this calculation is a*

*# string value assigned to the variable first\_line.* second\_line = x \* **' '** + **'/'** + (n - x - 1 + u + y) \* **' '** + **'**\\**'** + (m - y - 1) \* **' '** *# basic addition,*

*# subtraction, and multiplication is done. The result of this calculation is a string value assigned to*

*# the variable second\_line.* if p < q: *# If the value of the variable p is less then he value of the variable q.* left += [n \* **' '**] \* (q - p) *# The value inside the variable left is inserted inside*

*# an array with the number of elements = (q-p) and the string space between each*

*# string = n \* ' '. where the value of the variable left is the first element of the array.* elif q < p: *# If the value of the variable q is less then he value of the variable p.* right += [m \* **' '**] \* (p - q) *# The value inside the variable right is inserted inside*

*# a array with the number of elements = (p-q) and the string space between each*

*# string = m \* ' '. where the value of the variable right is the first element of the array.* zipped\_lines = zip(left, right) *# Join two left and right together.* lines = [first\_line, second\_line] + [a + u \* **' '** + b for a, b in zipped\_lines] *# The lines variable is*

*# assigned the value using addition, multiplication and a for loop to string.* return lines, n + m + u, max(p, q) + 2, n + u // 2 *# A graphical binary tree is returned.*bst = BinaryTree() *# The variable bst created from using the BinaryTree class.*bst.insert(4) *# Data is inserted in the root of the tree*bst.insert(2) *# Data is inserted in a tree node*bst.insert(6) *# Data is inserted in a tree node*bst.insert(1) *# Data is inserted in a tree node*bst.insert(3) *# Data is inserted in a tree node*bst.insert(5) *# Data is inserted in a tree node*bst.insert(7) *# Data is inserted in a tree node*bst.insert(8) *# Data is inserted in a tree node*bst.display(bst.root) *# The binary tree bst is printed.*print(**"The result for the recursive search value 1: "**, bst.find\_r(1)) *# Checks if the arguments value entered*

*# is present in the binary tree using recursive search method.*print(**"The result for the recursive search value 10: "**, bst.find\_r(10)) *# Checks if the arguments value entered*

*# is present in the binary tree using recursive search method.*print(**"The result for the iterative search value 1: "**, bst.find\_i(1)) *# Checks if the arguments value entered*

*# is present in the binary tree using iterative search method.*print(**"The result for the iterative search value 6: "**, bst.find\_i(6)) *# Checks if the arguments value entered*

*# is present in the binary tree using iterative search method.*print(**"The result for the iterative search value 10: "**, bst.find\_i(10)) *# Checks if the arguments value entered is present in the binary tree using iterative search method.*

*Text

Description automatically generated*

1. **Explanation of code**

Iterative and recursive search methods to find elements in a binary tree are implemented in this code.

The methods search for the data from the binary tree by moving around the tree according to the value of the said data and relevant to the value of the current node. If the data is found in the current node then it returns True. If the data is not present or None then it returns False or None respectively.

The method selects the root of the binary tree as the current node to check the data being searched for so it check’s if the data present in the root of the binary tree is None if it is then it simply returns None but if the data is present in the root of the binary tree then it returns True.

If the data being searched for is less than the value of the current node being checked of the binary tree then it checks the left of the binary tree and if the data on the left from the root of the binary tree has a value of None then it simply returns False.

If the data being searched for is greater than the value of the current node being checked of the binary tree then it checks the right of the binary tree and if the data on the left from the root of the binary tree has a value of None then it simply returns False.

If there is data present on the left or right node then it is checked if the data present, there is comparatively equal to the data being searched for if so, then it returns True, if not then it select’s the respective node as the current node. Then it moves to the left or right node repeating the same process until it either find’s the value returning True otherwise it returns False.

1. **Critical comment**

This code is an implementation of basic search method used in binary search. It uses either depth first or breath first approach as required. The code is written in python to increase the code readability and saving writing time.

In this implementation of the binary tree it is the data being search can only be present in the binary tree once if there was a different implementation which allowed storing the same value two or more times then this code will still return True if the data being search for is present in any node that the value being searched for is present in. To search for a specific value of the same kind could be done by checking the memory address where the value is stored.

In this implementation of the binary tree the data in each node is sorted with values less than the data on the left and greater then data values on the right but if were to search for the same data from a unsorted tree then this methods would be ineffective. It would be better to use breath or depth first approach then according to the data being searched.

This implementation of the binary tree the shape of the binary search tree relies on the order of insertions if the incursions are liner then finding a specified value would be done in the same as using a list.

**1.3 ADJACENCY MATRIX**

1. **Commented Code**

*"""  
  
 Basic code for inserting (i.e. building) and printing a Graph as Adjacency Matrix  
  
 third standard viva task\*\*\* (of 5)   
  
"""***"""   
  
Graph class  
  
"""**class Graph(object): *# A class object called "Graph" is created.  
 # The first parameter "self" is a reference to the current instance of the class.* def \_\_init\_\_(self, size): *# The class is initialized with the parameter defined. The 2nd  
 # parameter is called "size".* self.adjMatrix = [] *# The default value of an empty list assigned to the variable "self.adjMatrix".* if size <= 0: *# if the number entered is less than or equal to 0* print(**"enter a postive number greater then or equal to 1"**) *# printing a instruction* for i in range(size): *# looping from 0 to the value of the "size" variable.* self.adjMatrix.append([0 for i in range(size)]) *# Making empty lists with the number value of the size  
 # variable and filling list's with 0's equal to the value of the size variable.* self.size = size *# Instance variable "size" is assigned.  
  
 # methods for (1) adding a vertex; (2) adding an edge; (3) removing an edge; and (4) printing the  
 # matrix should appear here* def addVertex(self, number\_to\_append): *# A function to append vertex into the matrix,  
 # with the second parameter 'number\_to\_append'.* matrix\_length = len(self.adjMatrix) *# matrix\_length variable is assigned the value of the length of the  
 # self.adjMatrix* inserted\_row = number\_to\_append - matrix\_length *# inserted\_row variable is created and assigned the value of  
 # number\_to\_append - matrix\_length variable* for i in range(matrix\_length): *# looping from 0 to the value of the matrix\_length varibale.* for j in range(inserted\_row): *# looping from 0 to the value of the inserted\_row varibale.* self.adjMatrix[i].append(0) *# appending the newly added vertex to each list in the matrix.* self.adjMatrix.append([0 for i in range(number\_to\_append)]) *# adding the new vertex to the matrix and appending 0's to it.* def addEdge(self, vertex1, vertex2): *# A function to add edges into the matrix,  
 # with the second parameter 'vertex1' and the third parameter 'vertex2'.* if vertex1 > len(self.adjMatrix) or vertex1 > len(self.adjMatrix) or vertex1 < 0 or vertex1 < 0: *# checking if  
 # the first and the secound argument added to the parameter is greater than or less then the size of the elements in the matrix.* print(**"vertex does not exist"**) *# printing a instruction* elif vertex1 == vertex2: *# checking if both vertexes are equal in value* print(**"Same vertex"**,vertex1,**" and vertex "**, vertex2) *# printing an instruction* elif self.adjMatrix[vertex1-1][vertex2-1] == 1: *# if there is an edge present between vertex 1 and vertex 2* print(**"Edge already present between vertex"**,vertex1,**" and vertex "**, vertex2) *# printing a instruction* else: *#otherwise* self.adjMatrix[vertex1-1][vertex2-1] = 1 *# assigning the value of 1 into the first selected  
 # vertex 'vertex 1'* self.adjMatrix[vertex2-1][vertex1-1] = 1 *# assigning the value of 1 into the second selected  
 # vertex 'vertex 2'* def removeEdge(self, vertex1, vertex2): *# A function to remove edges from the matrix,  
 # with the second parameter 'vertex1' and the third parameter 'vertex2'.* if vertex1 > len(self.adjMatrix) or vertex1 > len(self.adjMatrix) or vertex1 < 0 or vertex1 < 0: *# checking if  
 # the first and the second argument added to the parameter is greater than or less than the size of the elements in the matrix.* print(**"vertex does not exist"**) *# printing a instruction* elif vertex1 == vertex2: *# checking if both vertexes are equal in value* print(**"Same vertex"**,vertex1,**" and vertex "**, vertex2) *# printing a instruction* elif self.adjMatrix[vertex1-1][vertex2-1] == 0: *# if there is a edge present between vertex 1 and vertex 2* print(**"No edge between vertex"**,vertex1,**" and vertex "**, vertex2) *# printing instructions* else: *# otherwise* self.adjMatrix[vertex1-1][vertex2-1] = 0 *# assigning the value of 0 into the first selected  
 # vertex 'vertex 1'* self.adjMatrix[vertex2-1][vertex1-1] = 0 *# assigning the value of 0 into the second selected  
 # vertex 'vertex 2'* def display(self): *# A function to return the matrix* title = [] *# The default value of an empty list assigned to the variable "title".* title.append([i for i in range(1, len(self.adjMatrix)+1)]) *# appending the numbers from 1 to one item more  
 # then length of the self.adjMatrix variable to make a title for the number of edges.* a = 1 *# the variable a is set to 1* print(**" "**,title) *# the title variables value is printed* print(**"-"** \* (len(self.adjMatrix) \* 4)) *# '-' symbol is printed times the length of the list times 4.* for i in range(len(self.adjMatrix)): *# elements are looped through to the length of the  
 # self.adjMatrix and assighned to i.* print(a,**"|"**, self.adjMatrix[i]) *# the value of a then a '|' symbol and the self.adjMatrix  
 # lists single element is printed in each iteration of the loop.* a += 1 *# a is added to the value of a*def main(): *# main function is created* new\_graph = Graph(4) *# new graph variable is created using the Graph class* new\_graph.addEdge(2, 3) *# edges are inserted inside the Graph.* new\_graph.addEdge(3, 4) *# edges are inserted inside the Graph.* new\_graph.addEdge(4, 2) *# edges are inserted inside the Graph.* new\_graph.addEdge(1, 3) *# edges are inserted inside the Graph.* new\_graph.display() *# the graph is printed.* new\_graph.removeEdge(2, 3) *# edges are removed from the Graph.* print(**"**\n**"**)  
 new\_graph.display() *# the graph is printed.* new\_graph.addVertex(5) *# vertex are inserted inside the Graph.* print(**"**\n**"**)  
 new\_graph.display() *# the graph is printed.*if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main() *# main function is called.*

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

When code tested with the edge already present and checking if there is no edge to remove

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. **Explanation of code**

Methods of adding a vertex, an edge, a function to remove an edge inside an adjacency matrix and Printing the graph as a matrix are implemented in this code.

In the method of adding a vertex 0’s are added to the end each already present nested list depending upon the number of nested lists making the number of elements in these nested lists equal to the number of total nested lists after the new nested lists have to be added to matrix. Then the new nested lists are added to the end of the matrix and are filled with the number of zeros equal to the number of total nested lists in the matrix.

In the method of adding an edge to vertexes inside a matrix. By using two vertexes as arguments firstly it is checked that if the edge being added is a part of a valid vertex if not so then secondly it is checked if the edges that need to be added are not both the same number if they are not the same number then thirdly it is cheeked that the edge we are trying to add is already present in the vertexes or not if it is not present then finally the first vertex argument is assigned to the value of the second argument and its value is replaced by one and the second vertex argument is assigned to the value of the first argument and its value is replaced by one. So, both the edges are assigned the value of one for each other respectively.

In the above method if the edge being added is not a valid vertex, if both the vertexes are not the same number or if the edges are already present in the vertex then instructions are generated to informing the user of the mentioned.

In the method of removing an edge from vertexes inside a matrix. By using two vertexes as arguments firstly it is checked that if the edge being added is a part of a valid vertex if not so then secondly it is checked if the edges that need to be added are not both the same number if they are not the same number then thirdly it is cheeked that the edge we are trying to add is already present in the vertexes or not if it is not present then finally the first vertex argument is assigned to the value of the second argument and its value is replaced by zero and the second vertex argument is assigned to the value of the first argument and its value is replaced by zero. So, both the edges are assigned the value of zero for each other respectively.

In the above method if the edge being added is not a valid vertex, if both the vertexes are not the same number or if the edges are already present in the vertex then instructions are generated to informing the user of the mentioned.

In the method of Printing the graph as a matrix the number of edges are printed followed by a number of hyphens printed in the next line to separate the edges from the next printed statements.Then the number of the vertex followed by a vertical bar and the vertexes are printed each on a new line.

1. **Critical comment**

Methods of adding a vertex, an edge, a function to remove an edge inside an adjacency matrix and Printing the graph as a matrix are implemented The code is written in python to increase the code readability, compactness, conciseness and saving writing time.

**1.4 Prim’s Algorithm**

1. **Commented Code**

import sys *#needed for maxsize*class Graph():   
   
 def \_\_init\_\_(self, vertices): *#implements graph as adjacency matrix* self.V = vertices *#number of vertices* self.graph = [[0 for column in range(vertices)] *#adjacency matrix with no edges (all connections set to zero)* for row in range(vertices)]   
   
 def printMST(self, parent):   
 print (**"Edge** \t **Weight"**)  
 for i in range(1, self.V):   
 print (parent[i], **"-"**, i, **"**\t**"**, self.graph[i][ parent[i] ])  
   
 *#from reached nodes find the unreached node with the minimum cost* def minKey(self, key, mstSet):   
 min = sys.maxsize *#set min to infinity (use maxsize which is next best thing!)* for v in range(self.V): *#count through number of vertices* if key[v] < min and mstSet[v] == False: *#if vertex is less than min and unreached* min = key[v] *#assign to min* min\_index = v *#min\_index is position of min in key* return min\_index *#return min\_index  
   
 #find MST* def primMST(self):   
 key = [sys.maxsize] \* self.V *#initialise key to list of values all set to infinity; same length as self.V* parent = [None] \* self.V *#list for constructed MST* key[0] = 0 *#set first element of key to zero (this is where we start)* mstSet = [False] \* self.V *#mstSet is list of booleans set to False* parent[0] = -1 *#first element of parent list set to -1* for vertex in range(self.V): *#go through all vertices* u = self.minKey(key, mstSet) *#call minKey; minKey returns u (index of unreached node)* mstSet[u] = True *#mstSet at index of node is set to True* for v in range(self.V): *#go through all vertices  
   
 #if edge from u to connected node v is > 0 (if there is an edge)  
 ### COMPLETE CODE HERE; FULL COMMENTS TO #and mstSet[v] is unreached   
 ### THE RIGHT #and key[v] is greater than the edge cost* if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:   
 *#if edge from u to connected node v is > 0 (if there is an edge)* key[v] = self.graph[u][v] *# key[v] is assigned the value of self.graph[u][v].* parent[v] = u *#parent[v] is index of node; so list of parents (nodes) is the MST* self.printMST(parent) *#print the list of parents, i.e. the MST*g = Graph(5)   
g.graph = [ [0, 2, 0, 6, 0],   
 [2, 0, 3, 8, 5],   
 [0, 3, 0, 0, 7],   
 [6, 8, 0, 0, 9],   
 [0, 5, 7, 9, 0]]   
   
g.primMST();

**Graphical user interface, text, application, email

Description automatically generated**

* 1. **Linked List**

1. **Commented Code**

class Node:  
  
 def \_\_init\_\_(self, dataval=None):  
 self.dataval = dataval *# Instance variable "dataval" is created.* self.nextval = None *# The default value of 'None' assigned to "self.nextval".*class SLinkedList:  
 def \_\_init\_\_(self):  
 self.headval = None *# The default value of 'None' assigned to "self.headval".* def listprint(self):  
 printval = self.headval *# assigning value of the self.headval to printval.* while printval is not None: *# while the printval variable is not None.* print(printval.dataval) *# printing the value of the variable printval.dataval* printval = printval.nextval *# Assigning value of the self.printval.nextval to printval.* def AtBeginning(self, newdata):  
 NewNode = Node(newdata) *# creating a NewNode variable with the value of Node(newdata) //creating a New Node* def AtEnd(self, newdata):  
 NewNode = Node(newdata) *# creating a NewNode variable with the value of Node(newdata) //creating a New Node* if self.headval is None: *# if the value of self.headval variable is 'None'* self.headval = NewNode *# The value of self.headval is assigned to the new node* return  
 last = self.headval *# The variable last is assigned the self.headval varable's value* while (last.nextval): *# while last.nextval varible's value is not 0.* last = last.nextval *# The last varible is assigned the value of last.nextval* last.nextval = NewNode *# The last.nextval variable is assighned the value of NewNode.* def Insert(self, val\_before, newdata):  
 if val\_before is None: *# if the value of val\_before is None* print(**"No node to insert after"**) *# printing an instruction* return  
 else:  
 NewNode = Node(newdata) *# creating a NewNode variable with the value of Node(newdata) //creating a New Node* NewNode.nextval = val\_before.nextval *# assign the value of val\_before.nextval to the variable NewNode.nextval.* val\_before.nextval = NewNode *# assign the value of NewNode to the varible val\_before.nextval.*list = SLinkedList() *# new list variable is created using the SLinkedList class*list.headval = Node(**"Mon"**) *# the value "Mon" is assigned to the head of the list.headval  
# creating variables for the linked list*e2 = Node(**"Tue"**)  
e3 = Node(**"Thur"**)  
e4 = Node(**"Fri"**)  
e5 = Node(**"Sat"**)  
*# linking the list*list.headval.nextval = e2  
e2.nextval = e3  
e3.nextval = e4  
e4.nextval = e5  
  
list.AtEnd(**"Sun"**) *# defining the last variable of the linked list*

list.listprint()  
print(**"**\n**"**)

list.Insert(list.headval.nextval, **"Wed"**) *# inserting the "Wed" value inside our linked list*list.listprint()

Graphical user interface, text, application, email

Description automatically generated

**2 ADVANCED TASKS**

1.1 **Remove method for Binary Tree class**

1. **Commented Code**

*"""   
  
 Basic BST code for inserting (i.e. building) and printing a tree  
  
 second standard viva task\*\*\* (of 5) will be to implement a find method into  
 the class BinaryTree from pseudocode. See the STD\_2 task sheet for Week 5.   
  
"""*import math *# The Maths module is imported.***""" Node class  
"""**class Node: *# A class object called "Node" is created.  
  
 # The first parameter "self" is a reference to the current instance of the class.* def \_\_init\_\_(self, data=None): *# The class is initialized with the parameter defined. The 2nd  
 # parameter is called "data" which has a default value of 'None'.* self.data = data *# Instance variable "data" is created.* self.left = None *# The default value of 'None' assigned to "self.left".* self.right = None *# The default value of 'None' assigned to "self.right".***""" BST class with insert and display methods. display pretty prints the tree  
"""**class BinaryTree: *# A class object called "BinaryTree" is created.* def \_\_init\_\_(self): *# The class is initialized with the parameter defined.* self.root = None *# The default value of 'None' assigned to the instance variable "root".* def insert(self, data): *# A function to insert data into the root,  
 # branch and leaf Nodes of the BinaryTree with the second parameter 'data'.* if self.root is None: *# It's checked that if the "self.root" is pointing towards the object 'None'.* self.root = Node(data) *# The value of the "data" instance of the "Node" class is*

*# assigned to "self.root".* else: *# Otherwise do the following.* self.\_insert(data, self.root) *# The "data" is inserted in a "Node" of BinaryTree class using the  
 # private function "\_insert".* def \_insert(self, data, cur\_node): *# A private function to insert data into the branch 'Nodes' of the  
 # BinaryTree with the parameters defined. The 2nd parameter is the "data" to be inserted and the  
 # third parameter is initially the "root Node" in our BinaryTree.* if data < cur\_node.data: *# if the data to be inserted in the Node is less than the current  
 # selected Node's data.* if cur\_node.left is None: *# It's checked that if the left branch or leaf of the current node is  
 # pointing towards the object 'None'.* cur\_node.left = Node(data) *# The left branch or leaf of the current 'Node' is assigned the  
 # 'Node(data)' value.* else: *# Otherwise do the following.* self.\_insert(data, cur\_node.left) *# The private function "\_insert" calls is called  
 # recursively. with 2 arguments, the value of the selected "data" variable and  
 # "cur\_node.left"/(the left of the current Node) until the data has been placed inside a  
 # Node with 'None' value on a left branch or leaf of the BinaryTree.* elif data > cur\_node.data: *# if the data to be inserted in the Node is greater*

*# than the current selected Node's data.* if cur\_node.right is None: *# It's checked that if the right branch or leaf of the current 'Node' is  
 # pointing towards the object 'None'.* cur\_node.right = Node(data) *# The right branch or leaf of the current node is assigned the Node*

*# data value.* else: *# Otherwise do the following.* self.\_insert(data,  
 cur\_node.right) *# The private function \_insert calls is called recursively.  
 # with 2 arguments, data and cur\_node.right until the data has been placed inside a*

*# Node with 'None'  
 # value on a right branch or leaf of the BinaryTree.* else: *# Otherwise do the following.* print(data, **"Already present in tree"**) *# print the data to be inserted is already present in the*

*# tree. The same element can't be entered more than once.* def remove(tree, target): *# The function to remove data from the Binarytree. The second parameter is the*

*# target the user enters to be removed.* if tree.root is None : *# If tree.root node is None* return False *# Then return Flase* elif tree.root.data == target: *# if tree.root node's value is the same as the target variable's value.* if tree.root.left is None and tree.root.right is None: *#if the left node of the tree.root value is None*

*# and the right node of the tree.root is None.* tree.root = None *# then the tree.root's is assigned the value None.* elif tree.root.left and tree.root.right is None: *# if the left node of the tree.root's node has a value*

*# other then None and right nodes's with respect to tree.root's node has a value of None.* tree.root = tree.root.left *# then tree.root's node is assigned the left node's value with*

*# respect to tree.root.* elif tree.root.left is None and tree.root.right: *# if the left node of the tree.root node's value is*

*# None and the right node of the tree.root's node has a value other than None.* tree.root = tree.root.right # *Then tree.root is assigned the value of the node on the right in*

*# respect to it.* elif tree.root.left and tree.root.right: *# if the left node and the right node with*

*# respect to the tree.root node has a value other than None.* tree.if\_left\_and\_right(tree.root) *# the function if\_left\_and\_right is called with*

*# tree.root as it's given argument.* parent = None *# The parent variable is created and assigned the value of None.* node = tree.root *# The node variable is created and assigned the tree.root's value.  
 # case 1: Target not found* while node and node.data != target: *# if the node and 'node.data' hold values equal to the target* parent = node *# The parent variable stores the node value* if target < node.data: *# if the target is less than the 'node.data' data* node = node.left *# then the node variable is assigned* elif target > node.data: *# if target is more than the value in node.data* node = node.right *# node variable is given 'node.right' value* if node is None or node.data != target: *# if the node is empty, or node.data is not equal to target* return False  
  
 *# case 2: Target has no children* elif node.left is None and node.right is None: *# if node.left has a value of None and node.right has a*

*# value of None.* if target < parent.data: *# if the value of the target variable is less than the parent.data*

*# variable's value.* parent.left = None *# then assigns parent.left value to None* else: *# otherwise* parent.right = None *# assighn parent.right variable's value to None* return True  
  
 *# case 3: Target has left child only* elif node.left and node.right is None: *# if node.left has a value other than None and node.right's*

*# value is None* if target < parent.data: *# if the target varbiles value is less than the parent varibles value.* parent.left = node.left *# the node.left data is added to the parent.left node* else: *# otherwise* parent.right = node.left *# parent.right varible is Assigned the node.left varible's value.* return True *# return True  
  
 # case 4: Target has right child only* elif node.left is None and node.right: *# if node.left has a value of None and node.right has a value other*

*# then None.* if target < parent.data: *# if target variables value is less than parent.data varibles value.* parent.left = node.right *# parent.left variable is Assigned the node.right varible's value.* else: *# otherwise* parent.right = node.right *# parent.right variable is Assigned the node.right variable’s value.* return True *# return True  
 # case 5:* Target has left and right childrenelse: *# otherwise* tree.if\_left\_and\_right(node) *# The function if\_left\_and\_right is called with node as the argument*def if\_left\_and\_right(tree,node): *# This is a subfunction to the remove function. This function helps in the*

*# deletion of a node. The second parameter is obtained from the remove function.* delNodeParent = node *# delNodeParent variable is created and assigned the node variables value* delNode = node.right *# delNode variable created and assigned the value of node.right* while delNode.left: *# while delNode.left varibles has a value other than None.* delNodeParent = delNode *# delNodeParent is assigned the value of the variable delNode.* delNode = delNode.left *# delNode variable is assigned the value of delNode.left variable.* node.data = delNode.data *# the node.data variable is assigned the delNode.data variables value* if delNode.right: *# if delNode.right varible has a value other than None.* if delNodeParent.data > delNode.data: *# if the delNodeParent.data variables value greater than the*

*# delNode.data varibles value.* delNodeParent.left = delNode.right *# delNodeParent.left variable is assigned the delNode.right*

*# varibles value* else: *# otherwise* delNodeParent.right = delNode.right *# delNodeParent.right variable is assigned delNode.right*

*# variables value.* else: *# otherwise* if delNode.data < delNodeParent.data: *# if the delNode.data variable’s value is lesser than the*

*# delNodeParent.data varible's value.* delNodeParent.left = None *# delNodeParent.left variable is assigned the value of None.* else: *# otherwise* delNodeParent.right = None *# delNodeParent.right variables is assigned the Value None.*

def display(self, cur\_node): *# A function to display the current selected Node, all*

*# the left and right Nodes with data present in them. It has to parameters* lines, \_, \_, \_ = self.\_display(cur\_node) *# The variable is created and assigned to*

*# recursively call the self.\_display(cur\_node)private function. The last three*

*# value's returned by the call are ignored in each call.* for line in lines: *# going through each line(value) in the variable lines.* print(line) *# print each line(value)* def \_display(self, cur\_node): *# The first parameter being self is a reference to*

*# the current instance of the class and the 2nd being the currently selected Node.* if cur\_node.right is None and cur\_node.left is None: *# if both the right and the left*

*# Node to the current selected Node is the object 'None'.* line = **'%s'** % cur\_node.data *# The line variable is created and assigned with the*

*# cur\_node.data value after it has been converted to a string.* width = len(line) *# The width variable created and assigned with the length of*

*# the value assigned to the line variable.* height = 1 *# The height variable created with the value 1 assigned to it.* middle = width // 2 *# The middle variable created and assigned with the*

*# width variable's value floor divided by 2.* return [line], width, height, middle *# The line variable’s value is converted to a list type, the  
 # width variable’s value, the height variables value and the middle variables value is returned.* if cur\_node.right is None: *# if the right Node to the current selected Node is the object 'None'.* lines, n, p, x = self.\_display(cur\_node.left) *# The variables lines, n, p, x are assigned the value returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value  
 # after it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the value*

*# assigned to the line variable.* first\_line = (x + 1) \* **' '** + (n - x - 1) \* **'\_'** + s *# basic addition, subtraction and multiplication*

*# is done. The result of this calculation is a string value assigned to the variable first\_line.* second\_line = x \* **' '** + **'/'** + (n - x - 1 + u) \* **' '** *# basic addition, subtraction, and*

*# multiplication is done. The result of this calculation is a string value assigned to the variable*

*# second\_line.* shifted\_lines = [line + u \* **' '** for line in lines] *# basic addition and multiplication is done.*

*# The result of this calculation is a string value assigned to the variable shifted\_lines.* return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, n + u // 2 *# Values of the written*

*# variables and calculated values are returned.* if cur\_node.left is None: *# if the left Node to the current selected Node is the object 'None'.* lines, n, p, x = self.\_display(cur\_node.right) *# The variables lines, n, p, x*

*# are assigned the vaule returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value*

*# after it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the value*

*# assigned to the line variable.* first\_line = s + x \* **'\_'** + (n - x) \* **' '** *# basic addition, subtraction and multiplication*

*# is done. The result of this calculation is a string value assigned to the variable first\_line.* second\_line = (u + x) \* **' '** + **'**\\**'** + (n - x - 1) \* **' '** *# basic addition, subtraction,*

*# and multiplication is done. The result of this  
 # calculation is a string value assigned to the variable second\_line.* shifted\_lines = [u \* **' '** + line for line in lines] *# basic addition and multiplication is done. The*

*# result of this calculation is a string value assigned to the variable shifted\_lines.* return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, u // 2 *# Values of the written*

*# variables and calculated values are returned.* left, n, p, x = self.\_display(cur\_node.left) *# The variables left, n, p, x is assigned the*

*# value returned by function call.* right, m, q, y = self.\_display(cur\_node.right) *# The variables right, m, q, y is assigned the*

*# value returned by function call.* s = **'%s'** % cur\_node.data *# The s variable is created and assigned with the cur\_node.data value after  
 # it has been converted to a string.* u = len(s) *# The u variable created and assigned with the length of the*

*# value assigned to the line variable.* first\_line = (x + 1) \* **' '** + (n - x - 1) \* **'\_'** + s + y \* **'\_'** + (m - y) \* **' '** *# basic addition,*

*# subtraction and multiplication are done. The result of this calculation is a*

*# string value assigned to the variable first\_line.* second\_line = x \* **' '** + **'/'** + (n - x - 1 + u + y) \* **' '** + **'**\\**'** + (m - y - 1) \* **' '** *# basic addition,*

*# subtraction, and multiplication is done. The result of this calculation is a string value assigned to*

*# the variable second\_line.* if p < q: *# If the value of the variable p is less then he value of the variable q.* left += [n \* **' '**] \* (q - p) *# The value inside the variable left is inserted inside*

*# an array with the number of elements = (q-p) and the string space between each*

*# string = n \* ' '. where the value of the variable left is the first element of the array.* elif q < p: *# If the value of the variable q is less then he value of the variable p.* right += [m \* **' '**] \* (p - q) *# The value inside the variable right is inserted inside*

*# a array with the number of elements = (p-q) and the string space between each*

*# string = m \* ' '. where the value of the variable right is the first element of the array.* zipped\_lines = zip(left, right) *# Join two left and right together.* lines = [first\_line, second\_line] + [a + u \* **' '** + b for a, b in zipped\_lines] *# The lines variable is*

*# assigned the value using addition, multiplication and a for loop to string.* return lines, n + m + u, max(p, q) + 2, n + u // 2 *# A graphical binary tree is returned.*bst = BinaryTree() *# The variable bst created from using the BinaryTree class.*bst.insert(4) *# Data is inserted in the root of the tree*bst.insert(2) *# Data is inserted in a tree node*bst.insert(6) *# Data is inserted in a tree node*bst.insert(1) *# Data is inserted in a tree node*bst.insert(3) *# Data is inserted in a tree node*bst.insert(5) *# Data is inserted in a tree node*bst.insert(7) *# Data is inserted in a tree node*bst.insert(8) *# Data is inserted in a tree node*bst.display(bst.root) *# The binary tree bst is printed.*bst.remove(6) # Data is removed from a node of the tree  
bst.display(bst.root) *# The binary tree bst is printed.*

Graphical user interface, text, application

Description automatically generated

The above is the test given in the code. It removes the root.right or the right branch node.

The results from other tests are given below.

Removing 4 or the root node.

Graphical user interface, text, application, email

Description automatically generated

Removing 2 or the root.left branch node.

Graphical user interface, text, application, email

Description automatically generated

1. **Explanation of code**

The methods search for the given target from the binary tree by moving around the tree according to the value of the said target and relevant to the value of the current node. If the data is found in the current node then it is removed and replaced. If the data is not present or None then it simply returns False.

First the remove function takes a value called target the method checks if the Binary tree has the value None. if the root node of the tree has the value of None the function simply returns False.

Then the value is present in the root node’s left and right node is checked if both have a value of None then then simply the root node is assigned the value of None.

If the left node has a value other than None and the right node with respect to tree.root has the value of None then then the left nodes value is assigned to the root node.

If the right node has a value other than None and the left node with respect to tree.root has the value of None then then the right nodes value is assigned to the root node.

If the both the left and right node with respect to the tree.root has a value other than None then the value of tree.root is passed in the function if\_left\_and\_right.

In the if\_left\_and\_right function. If the node required to be removed has left child and a right child in this case a variable delnode and a delnode parent has to be found by descending the tree which is the lowest value in the right sub tree this is done using a while loop. Then that value of delnode is assighned to the node.

Then finally the delnode needs to be removed. If the delnode has a right child only by setting its value to None.

while the tree is not empty, and the data is not in the root node then two variables node and parent are created. Then using the node variable, the tree is searched in a descending order to find the required target node. If the value is found, then the node is set equal to that node where the value was found, the parent variable is set to that node’s parent node and the node is deleted. Then if the data is not found then False is returned.

If the removed node has no children, then either the left or right child of the node is set to None depending on which side of the node is on.

If the removed node has a left child only then it is checked that is node a left child or a right child of its parent. If it is the left child, then if the node is the parents left child the nodes left child is assigned to the parents left child otherwise the node is the parents right child and so the nodes left child is assigned to the parents right child.

If the removed node has a right child only then it is checked that is node a left child or a right child of its parent. If it is the left child, then if the node is the parents left child the nodes right child is assigned to the parents left child if not then the node is the parents right child and so the node’s right child is assigned to the parents right child otherwise the if\_left\_and\_right function is called with the node variables value as the argument.

In the if\_left\_and\_right function. If the node required to be removed has left child and a right child in this case a variable delnode and a delnode parent has to be found by descending the tree which is the lowest value in the right sub tree this is done using a while loop. Then that value of delnode is assighned to the node.

Then finally the delnode needs to be removed. If the delnode has a right child only by setting its value to None.

1. **Critical comment**

This code is an implementation of basic remove method used to remove a node in a binary tree. The code is written in python to increase the code readability, compactness, conciseness and saving writing time.

In this implementation of the binary tree it is the target being search can only be present in the binary tree once if there was a different implementation which allowed storing the same value two or more times than this code will still return True if the data being search for is present in any node that the value being searched for is present in. To search for a specific value of the same kind could be done by checking the memory address where the value is stored.

In this implementation of the binary tree the data in each node is sorted with values less than the data on the left and greater then data values on the right but if were to search for the same data from a unsorted tree then this methods would be ineffective.

This implementation of the binary tree the shape of the binary search tree relies on the order of insertions if the incursions are liner then finding a specified value to remove would be done in the same as using a list.

Finally, as the code is written in python \_ symbol has been used to create private functions but if it was written in c++ this could have been handled better and the method wouldn’t be accessible outside the class.

1.2 **Dijkstra’s algorithm**

1. **Commented Code**

from collections import defaultdict  
import sys  
  
  
class Graph():  
 def \_\_init\_\_(self, size):  
 self.edges = defaultdict(list) *# dictionary of all connected nodes e.g. {'X': ['A', 'B', 'C', 'E'], ...}* self.weights = {} *# dictionary of edges and weights e.g. {('X', 'A'): 7, ('X', 'B'): 2, ...}* self.size = size *# Instance variable "size" is created.* self.dist = [] *# The varible "self.dist"is created and the default value of an empty list assigned to the  
 # variable "self.dist".* for i in range(size): *# looping from 0 to the value of the "size" variable.* self.dist.append(sys.maxsize) *# appending the value of"sys.maxsize" varible to the self.dist varible.* self.previous = [] *# The varible "self.previous" is created and the default value of an empty list assigned  
 # to the variable "self.previous"* for i in range(size): *# looping from 0 to the value of the "size" variable.* self.previous.append(None) *# appending "None" value to the self.dist varible.* def add\_edge(self, from\_node, to\_node, weight): *# bidirectional* self.edges[from\_node].append(to\_node) *# connecting the edges between nodes.* self.edges[to\_node].append(from\_node) *# connecting the edges between nodes.* self.weights[(from\_node, to\_node)] = weight *# inserting weights in between two nodes.* self.weights[(to\_node, from\_node)] = weight *# inserting weights in between two nodes.* def findSmallestNode(self):  
 smallest = self.dist[self.getIndex(self.Q[0])] *# assighning the value of smallest weight's of Node's starting  
 # from our initial node to our end Node* result = self.getIndex(self.Q[0]) *# assighning indexes as we move through the Node* node = self.dist *# assighning the value of smallest weight's of Node's starting from our initial node to our  
 # end Node inside lists.* for i in range(len(self.dist)): *# looping from 0 to the length of the "self.dist" variable* if self.dist[i] < smallest: *# if the value of "self.dist[i]" is less than  
 # the value of the "smallest" varible.* if node in self.Q: *# if the node varible's value is present in the Q varible.* smallest = self.dist[i] *# assighn the current value of the  
 # "self.dist[i]" varible to the smallest varible.* result = self.getIndex(node) *# assighning the value of "self.getIndex(node)"  
 # varible to the result varible.* return result *# retruning th result varible's value* def getIndex(self, neighbour):  
 for i in range(len(self.unpoppedQ)): *# looping from 0 to the length of the "self.unpoppedQ" variable* if neighbour == self.unpoppedQ[i]: *# if the value of both the varibles "neighbour" and  
 # "self.unpoppedQ[i]" is equal.* return i *# returning the i varibles value.* def getPopPosition(self, uNode):  
 result = 0 *# assighning the value of 0 to the result varible.* for i in range(len(self.Q)): *# looping from 0 to the length of the "self.Q" variable* if self.Q[i] == uNode: *# if the value of both the varibles "self.Q[i]" and "uNode" is equal.* return i *# returning the i varibles value.* return result *# returning the result varibles value.* def getUnvisitedNodes(self, uNode):  
 resultList = [] *# The varible "resultList"is created and is assighned the value of an empty list.* allNeighbours = self.edges[uNode] *# assigning the edges of the nodes to the "allNeighbours" varible.* for neighbour in allNeighbours: *# looping through each element in the "allNeighbours" varible using the  
 # vraible "neighbour".* if neighbour in self.Q: *# if the neighbour varible's value is present in the self.Q varible.* resultList.append(neighbour) *# appending the neighbour variable's value to the resultList varible.* return resultList *# returning the resultList varibles value.* def dijsktra(self, start, end):  
 self.Q = [] *# The varible "self.Q "is created and is assighned the value of an empty list.* for key in self.edges: *# looping through each element in the "self.edges" varible using the vraible "key".* self.Q.append(key) *# appending the key variable's value to the self.Q varible.* for i in range(len(self.Q)): *# looping from 0 to the length of the "self.Q" variable* if self.Q[i] == start: *# if Q[i] varible's value is equal to the start varible's value  
 # (in our case it's O).* self.dist[i] = 0 *# self.dist[i] varible is assighned the value 0.* self.unpoppedQ = self.Q[0:] *# assighning the value of "self.Q[0:]" varible to the "self.unpoppedQ" varible.* while self.Q: *# while self.Q not equal to 0 or False. (can also be broken by using a break statement)* u = self.findSmallestNode() *# using the findSmallestNode() function to assign the value of the smallest  
 # Node to u.* if self.dist[u] == sys.maxsize: *# if the value of the varibles "self.dist[u]" and "sys.maxsize" are equal.* break *# break from the loop.* if self.unpoppedQ[u] == end: *# if the "self.unpoppedQ[u]" varible's value is equal to the  
 # "end" varibles value.* break *# break from the loop.* uNode = self.unpoppedQ[u] *# assighning the value of "self.unpoppedQ[u]" varible to the "uNode" varible.* for i in self.edges[uNode]: *# looping through each element in the "self.edges[uNode]"  
 # varible using the vraible "i".* if i in self.Q: *# if the "i" varible's value is present in the self.Q varible.* if i in self.getUnvisitedNodes(uNode): *# if the "i" varible's value is present in  
 # the self.getUnvisitedNodes(uNode).* if self.dist[self.unpoppedQ.index(uNode)] + self.weights[(uNode, i)] < self.dist[self.unpoppedQ.index(i)]:  
 *# if the combined value of both the "(self.dist[self.unpoppedQ.index(uNode)])" and  
 # "self.weights[(uNode,i)]" vaibles  
 # is less then the value of the "self.dist[self.unpoppedQ.index(i)" varible  
 # (distance + weight < unpoped).* self.dist[self.unpoppedQ.index(i)] = self.dist[self.unpoppedQ.index(uNode)] + self.weights[(uNode,i)]  
 *# the varible "self.dist[self.unpoppedQ.index(i)]" is assighned the combined  
 # value of the "self.dist[self.unpoppedQ.index(uNode)]" and  
 # "self.weights[(uNode,i)]" varibles.* self.previous[self.unpoppedQ.index(i)] = uNode *# The  
 # "self.previous[self.unpoppedQ.index(i)]" varible is assighned the value of  
 # the variable "uNode".* self.Q.pop(self.Q.index(uNode)) *# The value of "self.Q.index(uNode)" is poped from the self.Q list.* weight = self.dist[self.unpoppedQ.index(uNode) + 1] *# assighing the value of the  
 # "self.dist[self.unpoppedQ.index(uNode)+1]" varible to the "weight" varible.* shortest\_path = [] *# The varible "shortest\_path"is created and is assighned the value of an empty list.* shortest\_path.insert(0, end) *# inserting the 0 as the start and the end varible's value as to the shortest  
 # path varible.* u = self.getIndex(end) *# The varible u is assighned the value of "self.getIndex(end)".* while self.previous[u] != None: *# as long as the value of "self.previous[u]" varible is not None* shortest\_path.insert(0, self.previous[u]) *# insert the "self.previous[u]" varible's value to  
 # the "shortest\_path" varible.* u = self.getIndex(self.previous[u]) *# assigning the value of "self.previous[u]" varible to the "u" varible.* return shortest\_path, weight *# returning the values of the "shortest\_path" and the "weight" varibles*graph = Graph(8) *# new graph variable is created using the Graph class  
  
# Assighning the weights of the edges.*edges = [  
 (**'O'**, **'A'**, 2),  
 (**'O'**, **'B'**, 5),  
 (**'O'**, **'C'**, 4),  
 (**'A'**, **'B'**, 2),  
 (**'A'**, **'D'**, 7),  
 (**'A'**, **'F'**, 12),  
 (**'B'**, **'C'**, 1),  
 (**'B'**, **'D'**, 4),  
 (**'B'**, **'E'**, 3),  
 (**'C'**, **'E'**, 4),  
 (**'D'**, **'E'**, 1),  
 (**'D'**, **'T'**, 5),  
 (**'E'**, **'T'**, 7),  
 (**'F'**, **'T'**, 3),  
]  
  
for edge in edges: *# looping through each element in the "edges" varible using the vraible "edge".* graph.add\_edge(\*edge) *# adding the required edges to the graph*print(graph.dijsktra(**'O'**, **'T'**)) *# printing the path and the total number of weight for the path.*

1.3 **concurrent headline scraper**

1. **Commented Code**

*# importing required labraries*import concurrent.futures  
import urllib.request  
import timeit  
import newspaper  
from newspaper import Article  
  
URLs = [**'http://www.foxnews.com/'**,  
 **'http://www.cnn.com/'**,  
 **'http://www.derspiegel.de/'**,  
 **'http://www.bbc.co.uk/'**,  
 **'https://theguardian.com'**, ] *# list of urls to scrape*def get\_headlines():  
 for url in URLs: *# looping through each URLs* result = newspaper.build(url, memoize\_articles=False) *# The results are built* print(**'**\n**''The headlines from %s are'** % url, **'**\n**'**) *# The headlines are printed* for i in range(1,6): *# looping from 1 to 5* art = result.articles[i] *# The article results are assigned to the art variable.* art.download() *# downloading the results* art.parse() *# parsing the results* print(art.title) *# printing the title*def load\_url(url, timeout):  
 with urllib.request.urlopen(url, timeout=timeout) as conn:  
 return conn.read()  
  
  
def concurrent\_URLs\_example():  
 with concurrent.futures.ThreadPoolExecutor(max\_workers=4) as executor: *# assigning the concurrent  
 # thread workers as the executor* future\_to\_url = {executor.submit(load\_url, url, 60): url for url in URLs} *# loading the URL and assigning  
 # it to a variable* for future in concurrent.futures.as\_completed(future\_to\_url): *# looping through the future\_to\_url varible.* url = future\_to\_url[future] *# assigning the future\_to\_url[future] variables value to the url variable* result = newspaper.build(url, memoize\_articles=False) *# The results are built.* print(**'**\n**''The headlines from %s are'** % url, **'**\n**'**) *# The headlines are printed* for i in range(1, 6): *# looping from 1 to 5* art = result.articles[i] *# The article results are assigned to the art varible* art.download() *# downloading the results* art.parse() *# parsing the results* print(art.title) *# printing the title*if \_\_name\_\_ == **'\_\_main\_\_'**:  
 import timeit  
  
 *# the elapsed time for concurrent urls* elapsed\_time = timeit.timeit(**"concurrent\_URLs\_example()"**, setup=**"from \_\_main\_\_ import concurrent\_URLs\_example"**,number=2) / 2  
 *# the elapsed time for non-concurrent urls  
 # elapsed\_time = timeit.timeit("get\_headlines()", setup="from \_\_main\_\_ import get\_headlines", number=2)/2* print(elapsed\_time) *# printing the elapsed time*

**A screenshot of a computer

Description automatically generated**

1. **Explanation of code**

Rather than the traditional non-concurrent processing the newspaper articles are scrapped and processed using a concurrent method the articles are processed parallelly meaning they are processed at the same time using more than 1 processors (in our case 4 processors). This way the processing saves time and the whole process just happens faster.

1. **Critical comment**

This code is an Implementing a concurrent headline scraper concurrently. The code is written in python to increase the code readability, compactness, conciseness and saving writing time.

In this instance the concurrent processing saves time but if the size of the data is quite small then the time taken to set up the processors is greater then the time saved by them so in that instance we should use non-concurrent methods of processing.