**NAME: NASHRA GHAFFAR**

**ROLL NO: CT-20032**

**AI LAB MANUAL**

**Lab 1**

**1. What is an expression?**

An expression is a combination of values, variables, operators, and function calls that can be evaluated to produce a result.

**2. What is a syntax error?**

A syntax error occurs when the code violates the rules of the programming language, making it unable to be parsed or executed.

**3. What is the result of this expression: “\*” \* 10.**

The result of the expression "*"10 is a string containing ten consecutive asterisks: "*\*\*\*\*\*\*\*\*".

**4. What is a variable?**

A variable is a named storage location in a program that holds a value or reference to a value.

**5. What are the primitive built-in types in Python?**

The primitive built-in types in Python are:

- Integer (`int`): Whole numbers without decimal points.

- Float (`float`): Real numbers with decimal points.

- String (`str`): Sequence of characters.

- Boolean (`bool`): Represents the values `True` or `False`.

- NoneType (`None`): Represents the absence of a value.

**6. When should we use “”” (triple quotes) to define strings?**

We use triple quotes ("""...""") to define multi-line strings or strings that span across multiple lines in Python.

**7. Assuming (name = “John Smith”). What does name[1] return?**

1. **name[1]** returns the character at the index 1 of the string "John Smith", which is "o".
2. **name[-2]** returns the character at the second-to-last index of the string "John Smith", which is "t".
3. **name[1:-1]** returns a substring of "John Smith" starting from index 1 (inclusive) to the second-to-last index (exclusive), which is "ohn Smit".
4. To get the length of **name**, you can use the **len()** function like this: **len(name)**.

**8. Given (name = “john smith”), what will name.title() return?**

`**name.title()** will return "John Smith" by capitalizing the first letter of each word in the string.

**9. What does name.strip() do?**

**name.strip()** removes any leading and trailing whitespaces from the string "John Smith".

**10. What will name.find(“Smith”) return?**

**name.find("Smith")** will return the index of the substring "Smith" in the string "John Smith". In this case, it will return 5

.**11. What will be the value of name after we call name.replace(“j”, “k”)?**

After calling **name.replace("j", "k")**, the value of **name** will still be "John Smith" as Python's string methods return a new string and do not modify the original string in-place.

**12. How can we check to see if name contains “John”?**

You can check if "John" is present in the string `name` using the `in` keyword like this: `if "John" in name:`.

**13. What are the 3 types of numbers in Python?**

The three types of numbers in Python are:

- Integer (`int`): Whole numbers without decimal points.

- Float (`float`): Real numbers with decimal points.

- Complex (`complex`): Numbers in the form `a + bj`, where `a` and `b` are real numbers, and `j` represents the imaginary unit.

**14. What is the difference between 10 / 3 and 10 // 3?**

`10 / 3` will give the floating-point division result, which is `3.3333333333333335`.

`10 // 3` will give the integer division result, which is `3`. It truncates the fractional part and returns the integer result.

**15. What is the result of 10 \*\* 3?**

The expression `10 \*\* 3` will give `1000`, which is 10 raised to the power of 3.

**16. Given (x = 1), what will be the value of x after we run (x += 2)?**

After running `(x += 2)`, the value of `x` will be `3`. It's a shorthand for `x = x + 2`.

**17. How can we round a number?**

You can round a number using the built-in `round()` function. For example, `rounded\_num = round(3.14159)` will give `3`.

**18. What is the result of float(1)?**

The expression `float(1)` will give `1.0`. It converts the integer `1` into a floating-point number.

**19. What is the result of 10 == “10”?**

The expression `10 == "10"` will return `False`. It checks if the value of `10` (an integer) is equal to the value of `"10"` (a string). Since the types are different, the result is `False`.

**20. What is the result of “bag” > “apple”?**

The expression `"bag" > "apple"` will return `True`. Python compares strings based on their lexicographic order, and in this case, "bag" comes after "apple" in the dictionary.

**21. What is the result of not(True or False)?**

The expression `not(True or False)` will return `False`. The `not` operator negates the value of the expression, so if the expression is `True`, `not` makes it `False`, and vice versa.

**22. What does range(1, 10, 2) return?**

`range(1, 10, 2)` returns an iterable object that represents the sequence of numbers starting from 1 up to (but not including) 10, with a step of 2. So, it will return the numbers 1, 3, 5, 7, and 9.

**23. Write a function that returns the maximum of two numbers.**

def max\_of\_two(a, b):

return max(a, b)

**24. Write a function called check\_num that takes a number.**

def check\_num(number):

if number % 3 == 0 and number % 5 == 0:

return "Divisible by both"

elif number % 3 == 0:

return "Divisible by 3"

elif number % 5 == 0:

return "Divisible by 5"

**25. Write a function called showNumbers that takes a parameter called limit. It should print all the numbers between 0 and limit.**

def showNumbers(limit):

for i in range(limit + 1):

print(i)

**26. Write a Python program which accepts the radius of a circle from the user and computes the area of a circle, triangle, square & rectangle.**

import math

def calculate\_area(shape, radius):

if shape == "circle":

return math.pi \* radius \*\* 2

elif shape == "triangle":

return (math.sqrt(3) / 4) \* radius \*\* 2

elif shape == "square":

return radius \*\* 2

elif shape == "rectangle":

return radius \* 2

shape = input("Enter the shape (circle, triangle, square, rectangle): ")

radius = float(input("Enter the radius: "))

area = calculate\_area(shape, radius)

print(f"The area of {shape} with radius {radius} is: {area:.2f}")

**27. Write a function that returns the sum of multiples of 3 and 5 between 0 and limit (parameter).**

def sum\_multiples(limit):

result = 0

for num in range(limit + 1):

if num % 3 == 0 or num % 5 == 0:

result += num

return result

**28. Write a function called show\_stars(rows). If rows is 5, it should print the following:**

def show\_stars(rows):

for i in range(1, rows + 1):

print("\*" \* i)

**29. Write a function that prints all the prime numbers between 0 and limit where limit is a parameter.**

def is\_prime(number):

if number < 2:

return False

for i in range(2, int(number \*\* 0.5) + 1):

if number % i == 0:

return False

return True

def print\_primes(limit):

for num in range(limit + 1):

if is\_prime(num):

print(num)

**30. Write a Python program to display the first and last colors from the following list.**

**color\_list = ["Red", "Green", "White", "Black"]**

color\_list = ["Red", "Green", "White", "Black"]

first\_color = color\_list[0]

last\_color = color\_list[-1]

print(f"First color: {first\_color}, Last color: {last\_color}")

**31. Write a Python program which accepts the user's first and last name and prints them in reverse order with a space between them.**

first\_name = input("Enter your first name: ")

last\_name = input("Enter your last name: ")

reversed\_name = last\_name + " " + first\_name

print("Reversed name:", reversed\_name)

**32. Write a Python program to find whether a given number (accept from the user) is even or odd, and print out an appropriate message to the user.**

def is\_even\_or\_odd(number):

if number % 2 == 0:

return "Even"

else:

return "Odd"

num = int(input("Enter a number: "))

result = is\_even\_or\_odd(num)

print(f"{num} is {result}.")

**33. Write a Python program that accepts an integer (n) and computes the value of n + nn + nnn.**

def compute\_value(n):

nn = int(str(n) \* 2)

nnn = int(str(n) \* 3)

return n + nn + nnn

num = int(input("Enter an integer: "))

result = compute\_value(num)

print(f"The result is: {result}")

**34. Write a Python program to test whether a passed letter is a vowel or not.**

def is\_vowel(letter):

vowels = "AEIOUaeiou"

return letter in vowels

letter = input("Enter a letter: ")

if is\_vowel(letter):

print(f"{letter} is a vowel.")

else:

print(f"{letter} is not a vowel.")

**35. Write a Python program to sum three given integers. However, if two values are equal, the sum will be zero.**

def sum\_with\_condition(a, b, c):

if a == b or b == c or a == c:

return 0

return a + b + c

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

num3 = int(input("Enter the third number: "))

result = sum\_with\_condition(num1, num2, num3)

print(f"The sum is: {result}")

**36. Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.**

def check\_values(a, b):

return a == b or abs(a - b) == 5 or a + b == 5

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = check\_values(num1, num2)

print(result)

**37. Write a Python program to solve (x + y) ^ (z).**

x = 4

y = 3

z = 2

result = (x + y) \*\* z

print(f"({x} + {y}) ^ {z} = {result}")

**LAB 2**

Q1: Define the following terms:

**Regular Graph:** A regular graph is a type of graph where all its vertices have the same degree, meaning each vertex has an equal number of edges incident to it.

**Null Graph**: The null graph is the simplest type of graph, containing no vertices and no edges. It is often denoted as "N" or "Φ."

**Trivial Graph:** The trivial graph is another simple graph that consists of a single vertex without any edges. It is sometimes referred to as a singleton graph.

**Simple Graph**: A simple graph is an undirected graph that does not contain any self-loops (edges from a vertex to itself) or parallel edges (multiple edges between the same pair of vertices).

**Connected Graph**: A connected graph is a graph in which there is a path between every pair of vertices. In other words, for any two vertices in the graph, there exists a sequence of edges that connect them.

**Disconnected Graph:** A disconnected graph is a graph that is not connected. It consists of two or more connected components, where there is no path between vertices in different components.

**Complete Graph:** A complete graph is a simple graph in which each vertex is directly connected to every other vertex by a unique edge. It is denoted as "Kn," where "n" represents the number of vertices.

**Cyclic Graph:** A cyclic graph is a graph that contains at least one cycle, which is a closed path that starts and ends at the same vertex, passing through a series of distinct vertices and edges.

**Degree of vertex**: The degree of a vertex in a graph refers to the number of edges incident to that vertex. For undirected graphs, it counts both incoming and outgoing edges.

**Loop**: A loop is an edge in a graph that connects a vertex to itself. In other words, it is an edge with both endpoints being the same vertex

**Parallel Edges:** Parallel edges refer to multiple edges between the same pair of vertices in a graph. In simple graphs, parallel edges are not allowed, but they can exist in multigraphs (graphs that permit multiple edges between vertices).

**Code Snippits:**

**TASK 1  
Code:**

#Task 1-1

graph = { "a" : ["c"],

"b":["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" :["c"],

"e" : ["c", "b"],

"f" : []

}

print(graph["a"])

print("The nodes/vertices of graph:",graph.keys())

#Returns List of dictionary keys

print("The edges of the graph for keys{0} are {1}".format(graph.keys(),graph.values()))

#Returns List of dictionary values

for i,j in graph.items():

  print(i,j)

#Task 1-2

graph = { "a" : ["b"],

"b":["c"],

"c" : ["d", "e"],

"d" :["c","e","f","g"],

"e" : ["c", "d","f"],

"f" : ["e","d"],

"g":["d"],

}

print(graph["a"])

print("The nodes/vertices of graph:",graph.keys())

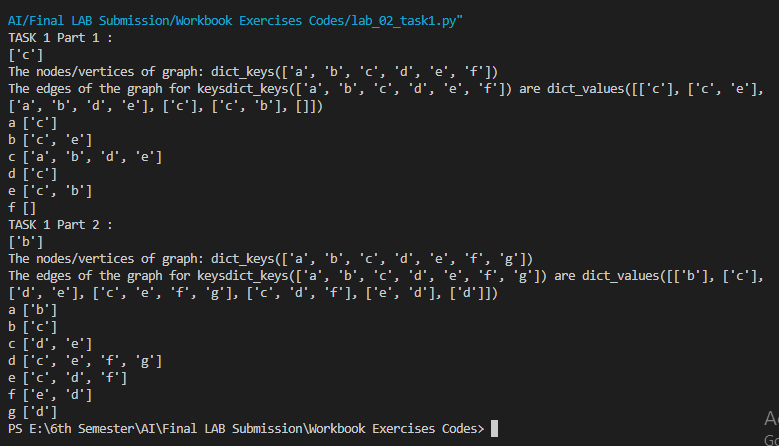
#Returns List of dictionary keys

print("The edges of the graph for keys{0} are {1}".format(graph.keys(),graph.values()))

#Returns List of dictionary values

for i,j in graph.items():

  print(i,j)

**Output: **

**TASK 2  
Code:**

#Task 2-1

graph = {

"a": ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

def generate\_edges (graph):

  edges = [] # empty list

  for node in graph: #traversing through graph

    for neighbour in graph [node]: # takes particular node

       edges.append((node, neighbour))

  return edges

print("Generating Edges:", generate\_edges (graph)) #using function

#Task 2-2

graph = {

"1": ["2","3"],

"2" : ["1", "3"],

"3" : ["1", "2", "4"],

"4" : ["3"],

}

def generate\_edges (graph):

  edges = [] # empty list

  for node in graph: #traversing through graph

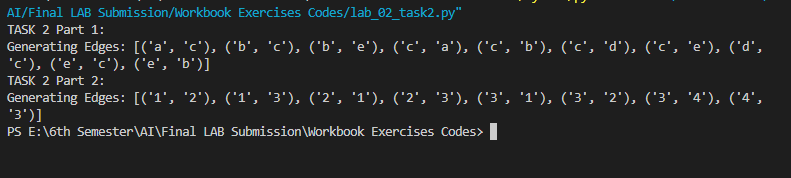
    for neighbour in graph [node]: # takes particular node

       edges.append((node, neighbour))

  return edges

print("Generating Edges:", generate\_edges (graph)) #using function

**Output:**

****

**TASK 3  
Code:**

#Task 3-1

graph = {

"a": ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

""" returns a list of isolated nodes. """

#Finding Isolated nodes

def find\_isolated\_nodes (graph):

  isolated = []

  for node in graph: #traverse each node

    if not graph[node]: #value for this node is nothing

      isolated += node # add the node to list

  return isolated

print("Isolated nodes", find\_isolated\_nodes (graph))

#Task 3-2

graph = {

"a": ["b","d"],

"b" : ["a","c", "d"],

"c" : ["b"],

"d" : ["a","b"],

"e" : [],

}

""" returns a list of isolated nodes. """

#Finding Isolated nodes

def find\_isolated\_nodes (graph):

  isolated = []

  for node in graph: #traverse each node

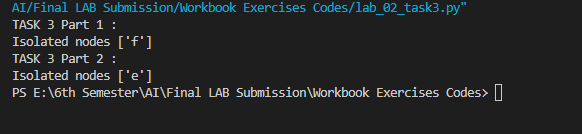
    if not graph[node]: #value for this node is nothing

      isolated += node # add the node to list

  return isolated

print("Isolated nodes", find\_isolated\_nodes (graph))

**Output:**

****

**TASK 4  
Code:**

#Task 4

graph = {

"a" : ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"F": []

}

#Finding path between two nodes

def find\_path(graph, start, end, path=[]): #empty list to save values of path

  path = path + [start] # adding first node

  if start == end: # check if both are same

    return path

  if start not in graph:

#if start is not present in graph; returns true if key is in dictionary, false otherwise

    return None

#if start and end are not same find the path

  for node in graph[start]: #for particular node 'a' traverse the neighbours

    if node not in path:

      newpath = find\_path(graph, node, end, path)

#recursive function; backtrack finds the vertices which are not present in path yet

      if newpath:

        return newpath

  return None

print("Path between nodes", find\_path(graph, 'e', 'd'))

#Task 4

graph = {

"a" : ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"F": []

}

#Finding path between two nodes

def find\_path(graph, start, end, path=[]): #empty list to save values of path

  path = path + [start] # adding first node

  if start == end: # check if both are same

    return path

  if start not in graph:

#if start is not present in graph; returns true if key is in dictionary, false otherwise

    return None

#if start and end are not same find the path

  for node in graph[start]: #for particular node 'a' traverse the neighbours

    if node not in path:

      newpath = find\_path(graph, node, end, path)

#recursive function; backtrack finds the vertices which are not present in path yet

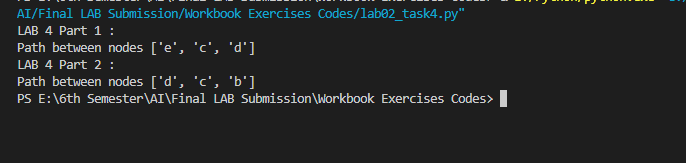
      if newpath:

        return newpath

  return None

print("Path between nodes", find\_path(graph, 'd', 'b'))

**Output:**

****

**TASK 5  
Code:**

#Task 5

graph = {

"a": ["c"],

"b": ["c", "e"],

"c": ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

#Finding all paths between two nodes

def find\_all\_paths (graph, start, end, path=[]):

  path = path + [start]

  if start == end:

    return [path]

  if start not in graph:

    return []

  paths = []

  for node in graph[start]:

    if node not in path:

      newpaths = find\_all\_paths (graph, node, end, path)

      for newpath in newpaths:

        paths.append(newpath)

  return paths

print("All Paths between nodes", find\_all\_paths (graph, 'a', 'b'))

#Task 5

graph = {

"a": ["c"],

"b": ["c", "e"],

"c": ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

#Finding all paths between two nodes

def find\_all\_paths (graph, start, end, path=[]):

  path = path + [start]

  if start == end:

    return [path]

  if start not in graph:

    return []

  paths = []

  for node in graph[start]:

    if node not in path:

      newpaths = find\_all\_paths (graph, node, end, path)

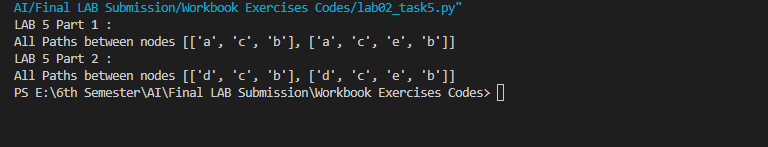
      for newpath in newpaths:

        paths.append(newpath)

  return paths

print("All Paths between nodes", find\_all\_paths (graph, 'd', 'b'))

**Output:**

****

**TASK 6  
Code:**

# Finding shortest path between two nodes

def find\_shortest\_path(graph, start, end, path=[]):

    path = path + [start]

    if start == end:

        return path

    if start not in graph:

        return None

    shortest = None

    for node in graph[start]:  # traverse through graph and find neighbors

        if node not in path:  # if neighbor node is not in path

            newpath = find\_shortest\_path(graph, node, end, path)

            # find shortest path between adjacent nodes and return the result

            if newpath:  # if you find a new path

                if not shortest or len(newpath) < len(shortest):

                    # compare the length of the acquired path with the shortest;

                    # if shortest path is found, assign the acquired path to the shortest variable

                    shortest = newpath

    return shortest

# Define the graph as a dictionary for Part 1

graph\_part1 = {

    'a': ['b', 'c'],

    'b': ['a', 'd'],

    'c': ['a'],

    'd': ['b']

}

# Task 6 Part 1

print("Shortest Path between nodes:", find\_shortest\_path(graph\_part1, 'a', 'b'))

# Define the graph as a dictionary for Part 2

graph\_part2 = {

    'd': ['b', 'e'],

    'b': ['a', 'd'],

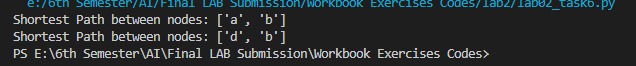
    'e': ['d']

}

# Task 6 Part 2

print("Shortest Path between nodes:", find\_shortest\_path(graph\_part2, 'd', 'b'))

**Output:**

****

**TASK 7  
Code:**

#Task 7 - 1

graph = {

"a" : ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

print("Actual graph", graph) #Adding an edge

def add\_edge(graph, edge):

  edge=set (edge) # if you don't want duplicates in list than you will use set

  (n1,n2)=tuple(edge) # same as list; can't be changed

  if n1 in graph:

    graph[n1]=n2

#graph[n1].append(n2)

  return graph

print("add an edge:", add\_edge(graph,{"a", "g"}))

#Task 7 - 2:

graph = {

"a" : ["c"],

"b" : ["c", "e"],

"c" : ["a", "b", "d", "e"],

"d" : ["c"],

"e" : ["c", "b"],

"f" : []

}

print("Actual graph", graph) #Adding an edge

def add\_edge(graph, edge):

  edge=set (edge) # if you don't want duplicates in list than you will use set

  (n1,n2)=tuple(edge) # same as list; can't be changed

  if n1 in graph:

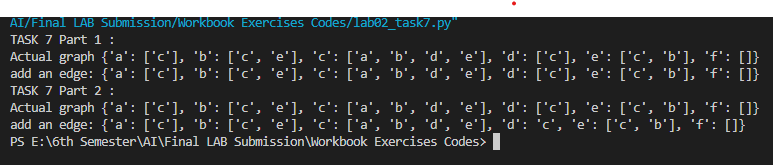
    graph[n1]=n2

#graph[n1].append(n2)

  return graph

print("add an edge:", add\_edge(graph,{"d", "c"}))

**Output:**

****

**TASK 8  
Code:**

#Task 8

graph = {

"a": ["a","c"],

"b" : ["c", "e"],

"c": ["a", "b", "d", "e"],

"d" : ["c"],

"e": ["c", "b"],

"f" : []

}

graph1 = { "a":["c"],

"c":[]

}

def find\_cycle\_single\_node (graph, start):

  for node in graph[start]:

    if node==start:

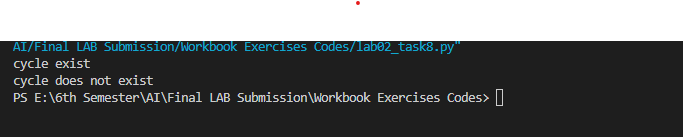
      return "cycle exist"

  return "cycle does not exist"

print(find\_cycle\_single\_node (graph, "a"))

print(find\_cycle\_single\_node (graph1, "a"))

**Output:**

****

**TASK 9  
Code:**

#Task 9

graph = { "a" : ["b"],

"b":["c"],

"c" : ["d", "e"],

"d" :["c","e","f","g"],

"e" : ["c", "d","f"],

"f" : ["e","d"],

"g":["d"],

}

def find\_degree (graph, node):

  degree=0

  t=[]

  for neighbour in graph [node ]:

    t.append(neighbour)

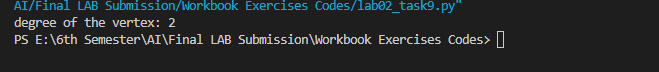
    degree=degree+1

  return degree

Degree=find\_degree(graph, "c")

print("degree of the vertex:", Degree)

**Output:**

****

**TASK 10  
Code:**

graph = { "a" : ["b"],

"b":["c"],

"c" : ["d", "e"],

"d" :["c","e","f","g"],

"e" : ["c", "d","f"],

"f" : ["e","d"],

"g":["d"],

}

def graph\_connected (graph, seen\_node=None, start=None):

  if seen\_node==None:

    seen\_node=set()

    nodes=list(graph.keys()) #list of all the graph keys

    if not start: #start is not given

      start=nodes[0] #vertex at the 0th index will be start

      seen\_node.add(start)

      if len(seen\_node) <len(nodes):

        for othernodes in graph [start]: # for the adjacent nodes of start

          if othernodes not in seen\_node:

            #if adjacent nodes are not present in seen\_node

            #it will recursively call itself

            if graph\_connected (graph, seen\_node, othernodes):

              return True

      else:

        return False

        return True

conn=graph\_connected (graph, seen\_node=None, start=None)

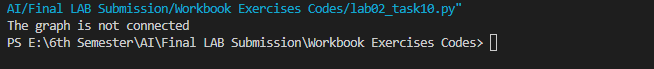
if conn:

  print("The graph is connected")

else:

  print("The graph is not connected")

**Output:**

****

**Question 2 (a): Consider the following graph:**

1. Find isolated nodes.
2. Find path between two vertex node 1 and 7.
3. Find all paths in graphs.
4. Find shortest path between nodes 1 and 7.
5. Determine cycles in graphs.
6. Add an edge named 9.
7. Find degree of vertex 4.
8. Find if the graph is connected.

**Answers:**

1. No isolated node
2. Path between node 1 and 7 is ['1', '4', '5', '6', '7']
3. All paths between 1 and 7:
   1. ['1', '4', '5', '6', '7']
   2. ['1', '4', '5', '6', '8', '7']
   3. ['1', '2', '3', '4', '5', '7']
   4. ['1', '2', '3', '4', '5', '8', '6', '7']
   5. ['1', '2', '3', '4', '5', '8', '7']
   6. ['1', '2', '4', '5', '6', '7']
   7. ['1', '2', '4', '5', '6', '8', '7']
   8. ['1', '2', '4', '5', '7']
   9. ['1', '2', '3', '4', '5', '7']
   10. ['1', '2', '3', '4', '5', '8', '6', '7']
   11. ['1', '2', '3', '4', '5', '8', '7']
   12. ['1', '2', '4', '5', '6', '7']
   13. ['1', '2', '4', '5', '6', '8', '7']
   14. ['1', '2', '4', '8', '7']
   15. ['1', '2', '3', '4', '5', '7']
   16. ['1', '2', '3', '4', '5', '8', '6', '7']
   17. ['1', '2', '3', '4', '5', '8', '7']
   18. ['1', '2', '4', '5', '6', '7']
   19. ['1', '2', '4', '5', '6', '8', '7']
   20. ['1', '2', '4', '5', '7']
   21. ['1', '2', '4', '5', '8', '6', '7']
   22. ['1', '2', '4', '5', '8', '7']
   23. ['1', '3', '2', '4', '5', '6', '8', '7']
   24. ['1', '3', '2', '4', '5', '7']
   25. ['1', '3', '2', '4', '5', '8', '6', '7']
   26. ['1', '3', '2', '4', '5', '8', '7']
   27. ['1', '3', '4', '5', '6', '8', '7']
   28. ['1', '3', '4', '5', '7']
   29. ['1', '3', '4', '5', '8', '6', '7']
   30. ['1', '3', '4', '5', '8', '7']
4. Shortest Path between node 1 and node 7 is ['1', '4', '5', '7']
5. cycle does not exist
6. add an edge:

{

'1': '9',

'2': ['1', '3', '4'],

'3': ['1', '2', '4'],

'4': ['1', '2', '3', '5'],

'5': ['4', '6', '7', '8'],

'6': ['5', '7', '8'],

'7': ['5', '6', '8'],

'8': ['5', '6', '7']

}

1. degree of the vertex 4 is : 4
2. The graph is not connected

**Question 2 (b): Consider the following graph:**

1. Find isolated nodes.
2. Find path between two vertex node B and A.
3. Find all paths in graphs.
4. Find shortest path between nodes B and A.
5. Determine cycles in graphs.
6. Add an edge named K.
7. Find degree of vertex G.
8. Find if the graph is connected.

**Answers:**

1. No isolated node
2. No Path between node Band A
3. No paths between B and A
4. No Shortest Path between node B and node A
5. cycle does not exist
6. add an edge:

{

'a': 'k',

'b': ['f', 'g'],

'c': ['a', 'g'],

'd': ['i', 'g'],

'e': ['g', 'i', 'j'],

'f': ['b', 'g'],

'g': ['b', 'f', 'h', 'j', 'e'],

'h': ['g', 'i', 'j'],

'i': ['d', 'e', 'h'],

'j': ['e', 'g', 'h']

}

1. degree of the vertex G is : 5
2. The graph is not connected

**Question 3: Consider the following graph:**

1. Find isolated nodes.
2. Find path between two vertex node Thomas’ Farm and Library.
3. Find all paths in graph.
4. Finding shortest path between nodes Thomas’ Farm and Library.
5. Determine cycles in graphs.
6. Add an edge named John’s House.
7. Find degree of vertex Bakery.
8. Find if the graph is connected.

**Answers:**

1. No isolated node
2. Path between node Thomas’ Farm and Library:

["thomas's farm", "mcfane's farm", 'bakery', "mayor's house", 'brewery', 'inn', 'library']

1. All paths between Thomas’ Farm and Library:
   1. ["thomas's farm", "mcfane's farm", 'bakery', "mayor's house", 'brewery', 'inn', 'library']
   2. ["thomas's farm", "mcfane's farm", 'bakery', "mayor's house", 'brewery', 'inn', 'dry cleaner', 'city hall', 'library']
   3. ["thomas's farm", "mcfane's farm", 'brewery', 'inn', 'library']
   4. ["thomas's farm", "mcfane's farm", 'brewery', 'inn', 'dry cleaner', 'city hall', 'library']
2. Shortest Path between nodes ["thomas's farm", "mcfane's farm", 'brewery', 'inn', 'library']
3. Shortest Path between Thomas’ Farm and Library:

["thomas's farm", "mcfane's farm", 'brewery', 'inn', 'library']

1. cycle does not exist
2. add an edge:

{

"mayor's house": "John's House",

'bakery': ["mayor's house", "mcfane's farm"],

'brewery': ["mayor's house", "mcfane's farm", 'inn'],

'inn': ['brewery', 'library', 'dry cleaner'],

"mcfane's farm": ['bakery', 'brewery', "thomas's farm"],

"thomas's farm": ["mcfane's farm"],

'library': ['inn', 'city hall'],

'dry cleaner': ['inn', 'city hall'],

'city hall': ['library', 'dry cleaner']

}

1. degree of the vertex Bakery is : 2
2. The graph is not connected

**LAB 3**

**Code Snippets:**

**TASK 1  
Code:**

graph\_A = {

'A': ['B', 'E'],

'B': ['F'],

'C': ['G'],

'D': ['E', 'H'],

'E': ['A', 'D', 'H'], 'F': ['B','G', 'I', 'J'], 'G': ['C', 'F', 'J'], 'H': ['D', 'E','I'], 'I': ['F', 'H'], 'J': ['F','G']

}

graph\_B={

  'A':['B','C','E'],

  'B':['A','D','E'],

  'C':['A','F','G'],

  'D':['B','E'],

  'E':['A','B'],

  'F':['C'],

  'G':['C']

}

# TASK 1

#visits all the nodes of a graph (connected component) using BFS

def bfs\_connected\_component (graph, start):

# keep track of all visited nodes

  explored = []

# keep track of nodes to be checked

  queue=[start]

#keep Looping until there are nodes still to be checked

  while queue:

#pop shallowest node (first node) from queue

    node= queue.pop(0)

    if node not in explored:

# add node to list of checked nodes

      explored.append(node)

      neighbours=graph[node]

# add neighbours of node to queue \

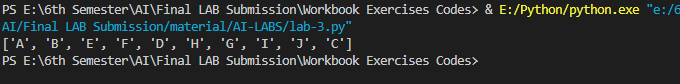
      for neighbour in neighbours:

        queue.append(neighbour)

  return explored

print(bfs\_connected\_component (graph\_A, 'A'))

**Output:**

****

**TASK 2  
Code:**

#TASK 2

# finds shortest path between 2 nodes of a graph using BFS

def bfs\_shortest\_path(graph, start, goal): #keep track of explored nodes explored = []

# keep track of all the paths to be checked

  explored=[]

  queue=[[start]]

# return path if start is goal

  if start == goal:

    return "That was easy! Start = goal"

  #keeps looping until all possible paths have been checked

  while queue:

#pop the first path from the queue

    path = queue.pop(0)

# get the last node from the path

    node = path[-1]

    if node not in explored:

      neighbours = graph[node]

# go through all neighbour nodes, construct a new path and #push it into the queue

      for neighbour in neighbours:

        new\_path = list(path)

        new\_path.append(neighbour)

        queue.append(new\_path)

# return path if neighbour is goal

      if neighbour == goal:

        return new\_path

#mark node as explored

    explored.append(node)

# in case there's no path between the 2 nodes

  return "So sorry, but a connecting path doesn't exist :("

print("Shortest path for graph A for A TO H :",bfs\_shortest\_path(graph\_A, 'A', 'H'))

print("Shortest path for graph B for A TO C :",bfs\_shortest\_path(graph\_B, 'A', 'C'))

print("Shortest path for graph A for A TO C :",bfs\_shortest\_path(graph\_A, 'A', 'C'))

graph = {

'A': set(['B', 'E']),

'B': set(['F']),

'C': set(['G']),

'D': set(['E', 'H']),

'E': set(['A', 'D', 'H']),

'F': set(['B','G', 'I', 'J']),

'G': set(['C', 'F', 'J']),

'H': set(['D', 'E','I']),

'I': set(['F', 'H']),

'J': set(['F','G'])

}

def all\_paths (graph, start, goal):

  queue=[ (start, [start])]

  while queue:

    (vertex, path)=queue.pop(0)

    for next in graph[vertex] - set(path):

      if next==goal:

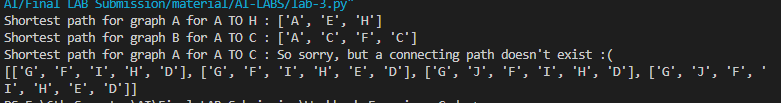
        yield path + [next]

      else:

        queue.append((next, path+[next]))

print(list(all\_paths(graph,'G','D')))

**Output:**

****

**Exercises:**

**QUESTION 1  
Code:**

# Question 1

graph\_1={

'1':set(['2','3','4']),

'2':set(['1','3','4']),

'3':set(['1','2','3']),

'4':set(['1','2','3','5']),

'5':set(['4','6','7','8']),

'6':set(['5','7','8']),

'7':set(['5','6','8']),

'8':set(['5','6','7']),

}

#part a

print("All nodes ",bfs\_connected\_component (graph\_1, '1'))

print('')

#part b

print("All paths b/w 1 & 6 ",list(all\_paths(graph\_1, '1','6')))

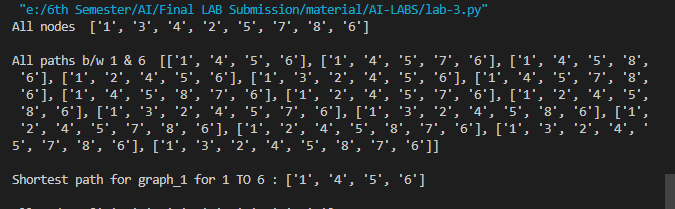
print('')

#part c

print("Shortest path for graph\_1 for 1 TO 6 :",bfs\_shortest\_path(graph\_1, '1', '6'))

print('')

**Output:**

****

**QUESTION 2  
Code:**

# Question 2

graph\_2 = {

'A': set(['B', 'C','D']),

'B': set(['A','E']),

'C': set(['A','F']),

'D': set(['A', 'E','G']),

'E': set(['B', 'D', 'G']),

'F': set(['C','G']),

'G': set(['D', 'E', 'F'])

}

#part a

print("All nodes ",bfs\_connected\_component (graph\_2, 'A'))

print('')

#part b

print("All paths b/w A&G ",list(all\_paths(graph\_2, 'A','G')))

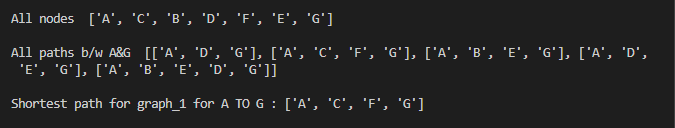
print('')

#part c

print("Shortest path for graph\_1 for A TO G :",bfs\_shortest\_path(graph\_2, 'A', 'G'))

print('')

**Output:**

****

**Lab 4**

**Code Snippets:**

**TASK 1**

**Code:**

# TASK-1

graph1={

'A':['B','E','C'],

'B':['D','E'],

'C':[],

'D':[],

'E':[]

}

def dfs(graph, node, visited):

  if node not in visited:

    visited. append (node)

    for n in graph[node]:

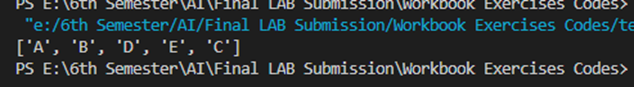
        dfs(graph,n, visited)

  return visited

visited = dfs(graph1, 'A', [])

print(visited)

**Output:**

****

**TASK 2**

**Code:**

# TASK-2

graph = {

    'A': set(['B', 'E', 'C']),

    'B': set(['D', 'E']),

    'C': set([]),

    'D': set([]),

    'E': set([])

}

def find\_path(graph, start, end, path=None):

    if path is None:

        path = []

    path = path + [start]

    if start == end:

        return path

    if start not in graph:

        return None

    for node in graph[start]:

        if node not in path:

            newpath = find\_path(graph, node, end, path)

            if newpath:

                return newpath

    return None

print(find\_path(graph, 'A', 'E'))

**Output:**

**C:\Users\wajiz.pk\AppData\Local\Microsoft\Windows\Clipboard\HistoryData\{23730705-7202-4559-B26D-D12BD55667CE}\{420CC3CF-AF89-412C-AA49-8FFF62EBF457}\ResourceMap\{96494850-BC56-48DD-903A-151CB2543BA4}**

**TASK 3**

**Code:**

# TASK-3

graph = {

    'A': set(['B', 'E', 'C']),

    'B': set(['D', 'E']),

    'C': set([]),

    'D': set([]),

    'E': set([])

}

def find\_shortest\_path(graph, start, end, path=None):

    if path is None:

        path = []

    path = path + [start]

    if start == end:

        return path

    if start not in graph:

        return None

    shortest = None

    for node in graph[start]:

        if node not in path:

            newpath = find\_shortest\_path(graph, node, end, path)

            if newpath:

                if not shortest or len(newpath) < len(shortest):

                    shortest = newpath

    return shortest

print(find\_shortest\_path(graph, 'A', 'E'))

def all\_paths\_dfs(graph, start, goal, visited=None, path=None):

    if visited is None:

        visited = set()

    if path is None:

        path = []

    visited.add(start)

    path = path + [start]

    if start == goal:

        yield path

    else:

        for next\_vertex in graph[start] - visited:

            yield from all\_paths\_dfs(graph, next\_vertex, goal, visited, path)

    visited.remove(start)

graph = {

    'A': set(['B', 'C']),

    'B': set(['A', 'C', 'D']),

    'C': set(['A', 'B', 'D']),

    'D': set(['B', 'C', 'E']),

    'E': set(['D', 'F']),

    'F': set(['E', 'G']),

    'G': set(['F', 'H']),

    'H': set(['G', 'I']),

    'I': set(['H', 'J']),

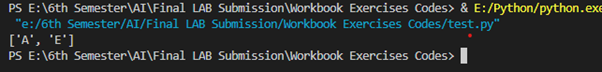
    'J': set(['I', 'K']),

    'K': set(['J'])

}

print(list(all\_paths\_dfs(graph, 'A', 'D')))

**Output:**

****

**Exercises:**

**QUESTION 1**

**Code:**

# Question 1

graph = {

'1': set(['2', '4','3']),

'2': set(['1','3','4']),

'3': set(['1','2','4']),

'4': set(['1', '2','3','5']),

'5': set(['4', '6', '7','8']),

'6': set(['5','7','8']),

'7': set(['5', '6', '8']),

'8': set(['5', '6', '7']),

}

# Part a

visited\_task1 = dfs(graph, '1', [])

print("Traversal path using DFS:", visited\_task1)

print('')

# Part b

single\_path\_task2 = find\_path(graph, '1', '6')

print("Single path between 1 & 6:", single\_path\_task2)

print('')

# Part c

all\_paths\_task3 = list(all\_paths\_dfs(graph, '1', '6'))  # Convert generator to list

print("All paths between 1 & 6:", all\_paths\_task3)

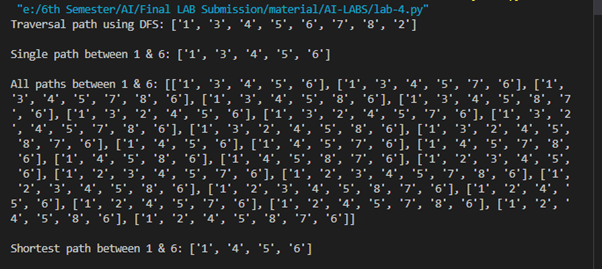
print('')

# Part d

shortest\_path\_task4 = find\_shortest\_path(graph, '1', '6')

print("Shortest path between 1 & 6:", shortest\_path\_task4)

print('')

**Output: **

**QUESTION 2**

**Code:**

# Question 2

graph\_2 = {

'A': set(['B', 'C','D']),

'B': set(['A','E']),

'C': set(['A','F']),

'D': set(['A', 'E','G']),

'E': set(['B', 'D', 'G']),

'F': set(['C','G']),

'G': set(['D', 'E', 'F'])

}

# Part a

visited\_task1 = dfs(graph\_2, 'A', [])

print("Traversal path using DFS:", visited\_task1)

print('')

# Part b

single\_path\_task2 = find\_path(graph\_2, 'A', 'G')

print("Single path between A & G:", single\_path\_task2)

print('')

# Part c

all\_paths\_task3 = list(all\_paths\_dfs(graph\_2, 'A', 'G'))  # Convert generator to list

print("All paths between A & G:", all\_paths\_task3)

print('')

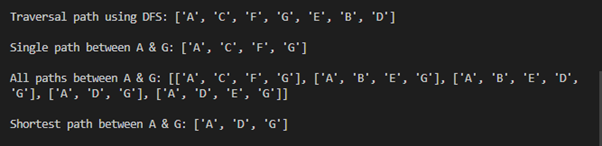
# Part d

shortest\_path\_task4 = find\_shortest\_path(graph\_2, 'A', 'G')

print("Shortest path between A & G:", shortest\_path\_task4)

print('')

**Output:**

****

**Lab 5**

**Code Snippets:**

**LAB TASK**

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

import queue as Q

def getPriorityQueue(lst):

    q = Q.PriorityQueue()

    for node in lst:

        q.put(Ordered\_Node(heuristics[node], node))

    return q, len(lst)

from google.colab import drive

drive.mount('/content/drive')

def greedyBFSUtil(G, v, visited, final\_path, dest, goal):

    if goal == 1:

        return goal

    visited[v] = True

    final\_path.append(v)

    if v == dest:

        goal = 1

    else:

        pq\_list = []

        pq, size = getPriorityQueue(G[v])

        for i in range(size):

            pq\_list.append(pq.get().description)

        for i in pq\_list:

            if goal != 1 :

              if visited[i] == False:

                  goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal)

    return goal

def greedyBFS(G, source, dest, heuristics, pos):

    visited = {}  # Initialize the visited dictionary

    for node in G.nodes():

        visited[node] = False

    final\_path = []

    goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0)  # Assuming this is a valid function call

    prev = -1

    for var in final\_path:

        if prev != -1:

            curr = var

            nx.draw\_networkx\_edges(G, pos, edgelist=[(prev, curr)], width=2.5, alpha=0.8, edge\_color='black')  # Fixed the syntax error

            prev = curr

        else:

            prev = var

    return

class Ordered\_Node(object):

    def \_\_init\_\_(self, priority, description):

        self.priority = priority

        self.description = description

    def \_\_lt\_\_(self, other):

        return self.priority < other.priority

import networkx as nx

def getHeuristics(G):

    heuristics = {}

    f = open('/content/drive/MyDrive/AI LABS/LAB\_4\_heuristics.txt')

    for i in G.nodes():

        node\_heuristic\_val = f.readline().split()

        heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

    G = nx.Graph()

    f = open('/content/drive/MyDrive/AI LABS/LAB\_4\_input.txt')

    n = int(f.readline())

    for i in range(n):

        graph\_edge\_list = f.readline().split()

        G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length=int(graph\_edge\_list[2]))

    source, dest = f.read().splitlines()

    return G, source, dest

import networkx as nx

import matplotlib.pyplot as plt

def DrawPath(G, source, dest):

    pos = nx.spring\_layout(G)

    val\_map = {}

    val\_map[source] = 'green'

    val\_map[dest] = 'red'

    values = [val\_map.get(node, 'blue') for node in G.nodes()]

    nx.draw(G, pos, with\_labels=True, node\_color=values, edge\_color='b', width=1, alpha=0.7)

    edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data=True)])

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, label\_pos=0.5, font\_size=5)

    return pos

if \_\_name\_\_ == "\_\_main\_\_":

    G, source, dest = CreateGraph()

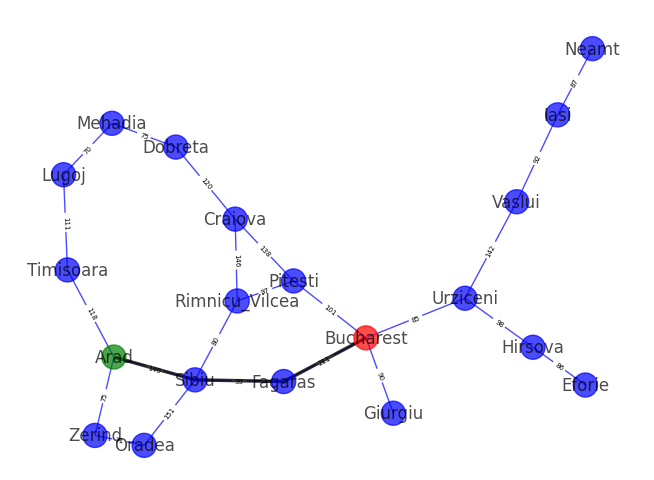
    heuristics = getHeuristics(G)

    pos = DrawPath(G, source, dest)

    greedyBFS(G, source, dest, heuristics, pos)

    plt.show()

**Output:**

****

**Exercises:**

**QUESTION 1**

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

import queue as Q

def getPriorityQueue(lst):

    q = Q.PriorityQueue()

    for node in lst:

        q.put(Ordered\_Node(heuristics[node], node))

    return q, len(lst)

def greedyBFSUtil(G, v, visited, final\_path, dest, goal):

    if goal == 1:

        return goal

    visited[v] = True

    final\_path.append(v)

    if v == dest:

        goal = 1

    else:

        pq\_list = []

        pq, size = getPriorityQueue(G[v])

        for i in range(size):

            pq\_list.append(pq.get().description)

        for i in pq\_list:

            if goal != 1 :

              if visited[i] == False:

                  goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal)

    return goal

def greedyBFS(G, source, dest, heuristics, pos):

    visited = {}  # Initialize the visited dictionary

    for node in G.nodes():

        visited[node] = False

    final\_path = []

    goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0)  # Assuming this is a valid function call

    prev = -1

    for var in final\_path:

        if prev != -1:

            curr = var

            nx.draw\_networkx\_edges(G, pos, edgelist=[(prev, curr)], width=2.5, alpha=0.8, edge\_color='black')  # Fixed the syntax error

            prev = curr

        else:

            prev = var

    return

class Ordered\_Node(object):

    def \_\_init\_\_(self, priority, description):

        self.priority = priority

        self.description = description

    def \_\_lt\_\_(self, other):

        return self.priority < other.priority

import networkx as nx

def getHeuristics(G):

    heuristics = {}

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_1\_heuristics.txt')

    for i in G.nodes():

        node\_heuristic\_val = f.readline().split()

        heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

    G = nx.Graph()

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_1\_input.txt')

    n = int(f.readline())

    for i in range(n):

        graph\_edge\_list = f.readline().split()

        G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length=int(graph\_edge\_list[2]))

    source, dest = f.read().splitlines()

    return G, source, dest

def DrawPath(G, source, dest):

    pos = nx.spring\_layout(G)

    val\_map = {}

    val\_map[source] = 'green'

    val\_map[dest] = 'red'

    values = [val\_map.get(node, 'blue') for node in G.nodes()]

    nx.draw(G, pos, with\_labels=True, node\_color=values, edge\_color='b', width=1, alpha=0.7)

    edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data=True)])

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, label\_pos=0.5, font\_size=5)

    return pos

if \_\_name\_\_ == "\_\_main\_\_":

    G, source, dest = CreateGraph()

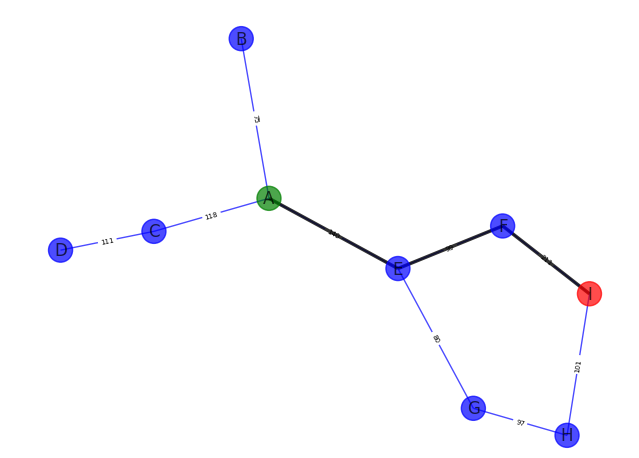
    heuristics = getHeuristics(G)

    pos = DrawPath(G, source, dest)

    greedyBFS(G, source, dest, heuristics, pos)

    plt.show()

**Output:**

****

**QUESTION 2**

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

import queue as Q

def getPriorityQueue(lst):

    q = Q.PriorityQueue()

    for node in lst:

        q.put(Ordered\_Node(heuristics[node], node))

    return q, len(lst)

def greedyBFSUtil(G, v, visited, final\_path, dest, goal):

    if goal == 1:

        return goal

    visited[v] = True

    final\_path.append(v)

    if v == dest:

        goal = 1

    else:

        pq\_list = []

        pq, size = getPriorityQueue(G[v])

        for i in range(size):

            pq\_list.append(pq.get().description)

        for i in pq\_list:

            if goal != 1 :

              if visited[i] == False:

                  goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal)

    return goal

def greedyBFS(G, source, dest, heuristics, pos):

    visited = {}  # Initialize the visited dictionary

    for node in G.nodes():

        visited[node] = False

    final\_path = []

    goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0)  # Assuming this is a valid function call

    prev = -1

    for var in final\_path:

        if prev != -1:

            curr = var

            nx.draw\_networkx\_edges(G, pos, edgelist=[(prev, curr)], width=2.5, alpha=0.8, edge\_color='black')  # Fixed the syntax error

            prev = curr

        else:

            prev = var

    return

class Ordered\_Node(object):

    def \_\_init\_\_(self, priority, description):

        self.priority = priority

        self.description = description

    def \_\_lt\_\_(self, other):

        return self.priority < other.priority

import networkx as nx

def getHeuristics(G):

    heuristics = {}

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_2\_heuristics.txt')

    for i in G.nodes():

        node\_heuristic\_val = f.readline().split()

        heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

    G = nx.Graph()

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_2\_input.txt')

    n = int(f.readline())

    for i in range(n):

        graph\_edge\_list = f.readline().split()

        G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length=int(graph\_edge\_list[2]))

    source, dest = f.read().splitlines()

    return G, source, dest

def DrawPath(G, source, dest):

    pos = nx.spring\_layout(G)

    val\_map = {}

    val\_map[source] = 'green'

    val\_map[dest] = 'red'

    values = [val\_map.get(node, 'blue') for node in G.nodes()]

    nx.draw(G, pos, with\_labels=True, node\_color=values, edge\_color='b', width=1, alpha=0.7)

    edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data=True)])

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, label\_pos=0.5, font\_size=5)

    return pos

if \_\_name\_\_ == "\_\_main\_\_":

    G, source, dest = CreateGraph()

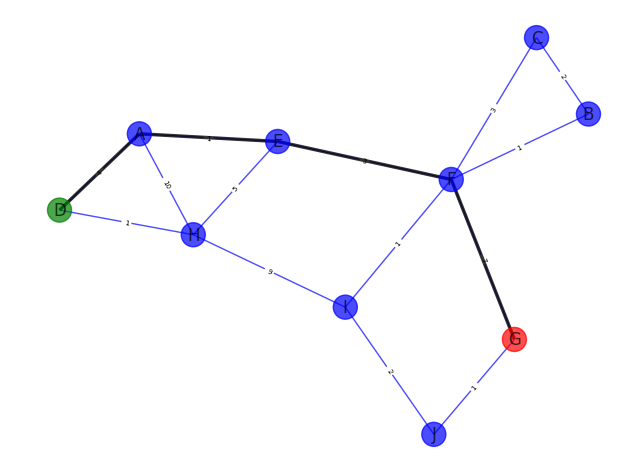
    heuristics = getHeuristics(G)

    pos = DrawPath(G, source, dest)

    greedyBFS(G, source, dest, heuristics, pos)

    plt.show()

**Output:**

****

**QUESTION 3**

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

import queue as Q

def getPriorityQueue(lst):

    q = Q.PriorityQueue()

    for node in lst:

        q.put(Ordered\_Node(heuristics[node], node))

    return q, len(lst)

def greedyBFSUtil(G, v, visited, final\_path, dest, goal):

    if goal == 1:

        return goal

    visited[v] = True

    final\_path.append(v)

    if v == dest:

        goal = 1

    else:

        pq\_list = []

        pq, size = getPriorityQueue(G[v])

        for i in range(size):

            pq\_list.append(pq.get().description)

        for i in pq\_list:

            if goal != 1 :

              if visited[i] == False:

                  goal = greedyBFSUtil(G, i, visited, final\_path, dest, goal)

    return goal

def greedyBFS(G, source, dest, heuristics, pos):

    visited = {}  # Initialize the visited dictionary

    for node in G.nodes():

        visited[node] = False

    final\_path = []

    goal = greedyBFSUtil(G, source, visited, final\_path, dest, 0)  # Assuming this is a valid function call

    prev = -1

    for var in final\_path:

        if prev != -1:

            curr = var

            nx.draw\_networkx\_edges(G, pos, edgelist=[(prev, curr)], width=2.5, alpha=0.8, edge\_color='black')  # Fixed the syntax error

            prev = curr

        else:

            prev = var

    return

class Ordered\_Node(object):

    def \_\_init\_\_(self, priority, description):

        self.priority = priority

        self.description = description

    def \_\_lt\_\_(self, other):

        return self.priority < other.priority

import networkx as nx

def getHeuristics(G):

    heuristics = {}

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_3\_heuristics.txt')

    for i in G.nodes():

        node\_heuristic\_val = f.readline().split()

        heuristics[node\_heuristic\_val[0]] = node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

    G = nx.Graph()

    f = open('/content/drive/MyDrive/AI LABS/Exercise\_Ques\_3\_input.txt')

    n = int(f.readline())

    for i in range(n):

        graph\_edge\_list = f.readline().split()

        G.add\_edge(graph\_edge\_list[0], graph\_edge\_list[1], length=int(graph\_edge\_list[2]))

    source, dest = f.read().splitlines()

    return G, source, dest

def DrawPath(G, source, dest):

    pos = nx.spring\_layout(G)

    val\_map = {}

    val\_map[source] = 'green'

    val\_map[dest] = 'red'

    values = [val\_map.get(node, 'blue') for node in G.nodes()]

    nx.draw(G, pos, with\_labels=True, node\_color=values, edge\_color='b', width=1, alpha=0.7)

    edge\_labels = dict([((u, v,), d['length']) for u, v, d in G.edges(data=True)])

    nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, label\_pos=0.5, font\_size=5)

    return pos

if \_\_name\_\_ == "\_\_main\_\_":

    G, source, dest = CreateGraph()

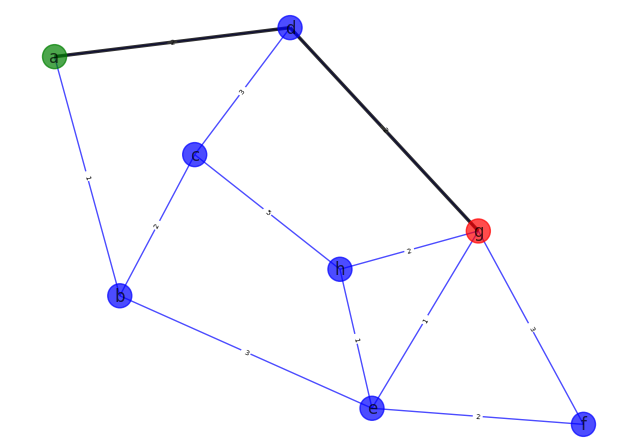
    heuristics = getHeuristics(G)

    pos = DrawPath(G, source, dest)

    greedyBFS(G, source, dest, heuristics, pos)

    plt.show()

**Output:**

****

**Lab 6**

**Code Snippets:**

**Exercises:**

**LAB TASK**

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

import queue as Q

def getPriorityQueue(G,v):

  q=Q.PriorityQueue()

  for node in G[v]:

    q.put(Ordered\_Node(float(heuristics[node])+float(G[node][v]['length']),node))

    return q,len(G[v])

def aStarSearchUtil(G,v,visited,final\_path,dest,goal):

  if goal==1:

    return goal

    visited[v]=True

    final\_path.append(v)

    if v==dest:

      goal=1

    else:

      pq\_list=[]

      pq,size=getPriorityQueue(G,v)

      for i in range(size):

        pq\_list.append(pq.get().description)

      for i in pq\_list:

        if goal!=1:

          if visited[i]==False:

            goal=aStarSearchUtil(G,i,visited,final\_path,dest,goal)

    return goal

def aStarSearch(G,source,dest,heuristic,pos):

  visited={}

  for node in G.nodes():

    visited[node]=False

  final\_path=[]

  goal=aStarSearchUtil(G,source,visited,final\_path,dest,0)

  prev=-1

  for var in final\_path:

    if prev!=-1:

      curr=var

      nx.draw\_networkx\_edges(G,pos,edgelist=[(prev,curr)],width=2.5,alpha=0.8,edge\_color='black')

      prev=curr

    else:

      prev=var

    return

from networkx.classes import graph

class ordered\_node(object):

  def \_init\_(self,priority,description):

    self.priority=priority

    self.description=description

    return

  def \_cap\_(self,other):

    return cmp(self.priority,other.priority)

def getHeuristics(G):

  heuristics={}

  f=open('lab6\heuristics.txt')

  for i in G.nodes():

    node\_heuristic\_val=f.readline().split()

    heuristics[node\_heuristic\_val[0]]=node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

  G=nx.Graph()

  f=open('lab6\input.txt')

  n=int(f.readline())

  for i in range(n):

    graph\_edge\_list=f.readline().split()

    G.add\_edge(graph\_edge\_list[0],graph\_edge\_list[1],length=graph\_edge\_list[2])

  source,dest=f.read().splitlines()

  return G,source,dest

class ordered\_node(object):

  def \_init\_(self,priority,description):

    self.priority=priority

    self.description=description

    return

  def \_cap\_(self,other):

    return cmp(self.priority,other.priority)

def getHeuristics(G):

  heuristics={}

  f=open('lab6\heuristics.txt')

  for i in G.nodes():

    node\_heuristic\_val=f.readline().split()

    heuristics[node\_heuristic\_val[0]]=node\_heuristic\_val[1]

    return heuristics

def CreateGraph():

  G=nx.Graph()

  f=open('lab6\input.txt')

  n=int(f.readline())

  for i in range(n):

    graph\_edge\_list=f.readline().split()

    G.add\_edge(graph\_edge\_list[0],graph\_edge\_list[1],length=graph\_edge\_list[2])

  source,dest=f.read().splitlines()

  return G,source,dest

def DrawPath(G,source,dest):

  pos=nx.spring\_layout(G)

  val\_map={}

  val\_map[source]='green'

  val\_map[dest]='red'

  values=[val\_map.get(node,'blue') for node in G.nodes()]

  nx.draw(G,pos,with\_labels=True,node\_color=values,edge\_color='b',width=1,alpha=0.7)

  edge\_labels=dict([((u,v),d['length']) for u,v,d in G.edges(data=True)])

  nx.draw\_networkx\_edge\_labels(G,pos,edge\_labels=edge\_labels,label\_pos=0.5,font\_size=11)

  return pos

if \_\_name\_\_=="\_\_main\_\_":

  G,source,dest=CreateGraph()

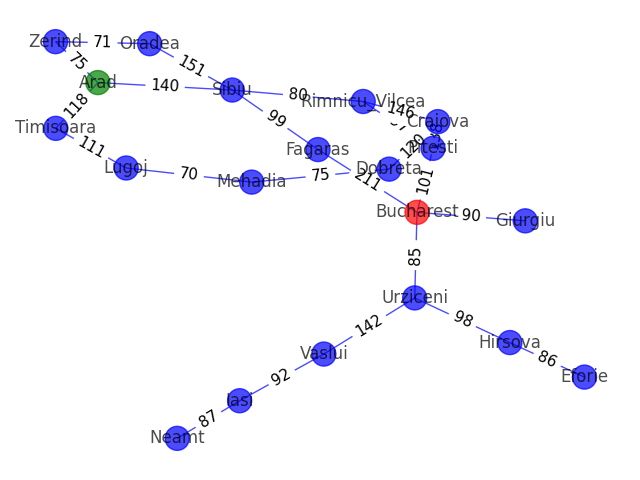
  heuristics=getHeuristics(G)

  pos=DrawPath(G,source,dest)

  aStarSearch(G,source,dest,heuristics,pos)

  plt.show()

**Output:**

****

**Lab 7**

**Q1) Describe the term object Detection. How it can be used to create artificially intelligent applications?**

Object detection is a computer vision task that involves identifying and locating multiple objects within an image or video. It's a process of recognizing specific objects in visual data and drawing bounding boxes around them.

In creating artificially intelligent applications, object detection can be used for various purposes, such as:

* **Autonomous Vehicles:** Detecting pedestrians, vehicles, and other objects on the road to enable safe navigation.
* **Surveillance Systems:** Identifying people, objects, or suspicious activities in real-time video feeds.
* **Retail:** Tracking and recognizing products on store shelves for inventory management.
* **Healthcare:** Detecting and localizing anomalies in medical images, like tumors or abnormalities.
* **Augmented Reality:** Placing virtual objects accurately in the real world based on detected surroundings.

Overall, object detection empowers AI applications to interact with and understand the visual world, enabling a wide range of valuable and practical use cases.

**Q2) What is the difference between classification and object detection and what are the key differences while implementation of classification and object detection problem domain?**

Classification and object detection are both tasks in computer vision, but they serve different purposes and have distinct characteristics:

**Classification:**

* In classification, the goal is to assign a single label or class to an entire input image.
* The output is a categorical prediction representing what the image contains.
* The model does not provide information about the location of the object(s) in the image.
* Implementation involves using image-based datasets with labeled categories.

**Object Detection:**

* Object detection aims to identify and locate multiple objects within an image or video.
* The output includes bounding boxes around detected objects along with their respective class labels.
* The model provides both class labels and spatial information about the objects.
* Implementation requires datasets with annotated bounding boxes and corresponding object labels.

**Key Differences in Implementation:**

* In classification, the output is a single class label, while object detection provides both class labels and bounding box coordinates.
* Object detection models are more complex and computationally intensive compared to classification models due to the additional task of localizing objects.
* While classification can be solved with simpler architectures like CNNs, object detection typically involves advanced models like Faster R-CNN, YOLO (You Only Look Once), or SSD (Single Shot Multibox Detector).
* Object detection datasets require more annotations, making them more time-consuming and expensive to create compared to classification datasets.

In summary, the main difference lies in the nature of their output and the complexity of the models used, with object detection providing both class labels and spatial information about multiple objects in an image or video.

**Q3) Enlist the algorithm steps of Viola-Jones Object Detector.**

The Viola-Jones object detection algorithm consists of the following key steps:

**Haar Feature Selection:** Choose Haar-like features to represent different patterns in the image, such as edges, lines, and corners.

**Integral Image Calculation:** Compute the integral image to efficiently calculate the sum of pixel values within any rectangular region of the image.

**Adaboost Training:** Train a sequence of weak classifiers using Adaboost to select the most relevant Haar features for object detection.

**Feature Thresholding:** Set thresholds for the selected Haar features to determine if a particular region contains an object or not.

**Cascade Classifiers:** Organize the weak classifiers into cascades of stages to efficiently reject non-object regions and focus on potential object areas.

**Sliding Window Detection:** Slide a window of varying sizes over the image and apply the cascade classifiers to detect objects within the window.

**Non-Maximum Suppression:** Overlapping detections are eliminated by keeping only the most confident one, reducing duplicate detections.

**Object Localization:** The final step involves localizing the detected object(s) within the image using bounding boxes.

The Viola-Jones algorithm is known for its efficiency and effectiveness in real-time object detection tasks, particularly in face detection.

**Q4) what is the Integral image?**

The integral image, also known as the summed area table, is a pre-computed representation of an image that allows for efficient computation of the sum of pixel values within any rectangular region in the image.

**Q5) what is the Integral image?**

"Strong" and "Weak" classifiers are terms related to ensemble learning techniques.

**Weak Classifier:**

* A weak classifier is a simple and relatively low-performing model that performs slightly better than random guessing.
* It focuses on a single feature (e.g., Haar-like features in the Viola-Jones algorithm) and sets a threshold to classify the data into positive and negative examples based on that feature.
* Weak classifiers have limited predictive power and are used to complement other weak classifiers in the ensemble to form a strong classifier.

**Strong Classifier:**

* A strong classifier is an ensemble of multiple weak classifiers that are combined to create a more powerful and accurate model.
* The combination is typically achieved using boosting algorithms like Adaboost, which iteratively trains weak classifiers and assigns higher weights to misclassified examples to improve their performance.
* The final strong classifier is the weighted sum of the weak classifiers, where the weights are determined based on their performance during the boosting process.

**Q6) Describe the functionality of different python libraries used for the implementation of object detection.**

Different Python libraries commonly used for implementing object detection are:

1. **TensorFlow Object Detection API:** Pre-trained models and tools for custom detection using TensorFlow.
2. **PyTorch + torchvision:** Object detection with PyTorch, customizable and fine-tunable.
3. **Detectron2:** Powerful object detection library based on PyTorch.
4. **OpenCV:** Supports YOLO, SSD, and real-time object detection.
5. **YOLO (You Only Look Once):** Real-time object detection, popular and efficient.
6. **MXNet/GluonCV**: MXNet's computer vision toolkit with pre-trained models.

**Q7) Execute the object detection code on 10 images of your choice and find out accuracy and also show the tested results.**

from \_\_future\_\_ import division

import cv2

import os

import shutil

def haarImagesHandler(folder, face\_cascade, eye\_cascade):

    global lock

    folder = os.path.dirname(folder)

    os.chdir(folder)

    # If the folder already exists

    # if lock == False and "HaarResults" in os.listdir(folder):

    #     # Deleting the folder

    #     shutil.rmtree("HaarResults")

    if lock == False:

        os.makedirs("HaarResults")

        HaarOutput = os.path.join(folder, "HaarResults")

        lock = True

        Total = 0

        Found = 0

        notFound = 0

        for filename in os.listdir(folder):

            # Check if the file has an image extension

            if filename.endswith((".jpg", ".png", ".bmp",".jpeg")):

                image = cv2.imread(os.path.join(folder, filename))

                if haarImages(image, face\_cascade, eye\_cascade, filename, HaarOutput) == False:

                    notFound += 1

                    Total += 1

                if haarImages(image, face\_cascade, eye\_cascade, filename, HaarOutput) == True:

                    Found += 1

                    Total += 1

        percentage(Found, Total)

        percentage(notFound, Total, found=False)

def percentage(obtain, total, found=True):

    try:

        if found == True:

            print('Found', (obtain / total) \* 100, '%')

        if found == False:

            print('Not Found', (obtain / total) \* 100, '%')

    except ZeroDivisionError:

        print('Please Check Your Path, No File Found')

    # Deleting the HaarResults folder

    # shutil.rmtree('HaarResults')

    exit()

def haarImages(frame, face\_cascade, eye\_cascade, filename, HaarOutput):

    faces = face\_cascade.detectMultiScale(frame, 1.1, 2)

    for (x, y, w, h) in faces:

        # Draw rectangle around the detected face

        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

        eyes = eye\_cascade.detectMultiScale(frame)

        # If no eyes detected

        if len(eyes) == 0:

            cv2.imwrite(os.path.join(HaarOutput, filename), frame)

            return False

        # If eyes detected

        if len(eyes) != 0:

            for (ex, ey, ew, eh) in eyes:

                # Draw rectangle around the detected eyes

                cv2.rectangle(frame, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

                cv2.imwrite(os.path.join(HaarOutput, filename), frame)

                print(os.path.join(HaarOutput, filename))

                return True

if \_\_name\_\_ == '\_\_main\_\_':

    global lock

    root = os.getcwd()

    face\_cascade = cv2.CascadeClassifier(os.path.join(root, 'haarcascade\_frontalface\_default.xml'))

    eye\_cascade = cv2.CascadeClassifier(os.path.join(root, 'haarcascade\_eye.xml'))

    lock = False

    haarImagesHandler(r"lab7\male.jpg", face\_cascade, eye\_cascade)

**Lab 8**

**TEST 1  
Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

freq\_samp, aud = wavfile.read('Audio\_car.wav')

print('Signal Shape', aud.shape)

print("Signal Dtype",aud.dtype)

print("Signal Duration",round(aud.shape[0]/float(freq\_samp),2),'seconds')

aud = aud / np.power(2,15)

aud = aud[:100]

time\_axis = 1000\*np.arange(0, len(aud),1) / float(freq\_samp)

plt.plot(time\_axis, aud, color = 'b')

plt.xlabel("milliseconds")

plt.ylabel('Amplitude')

plt.title("Input audio signal")

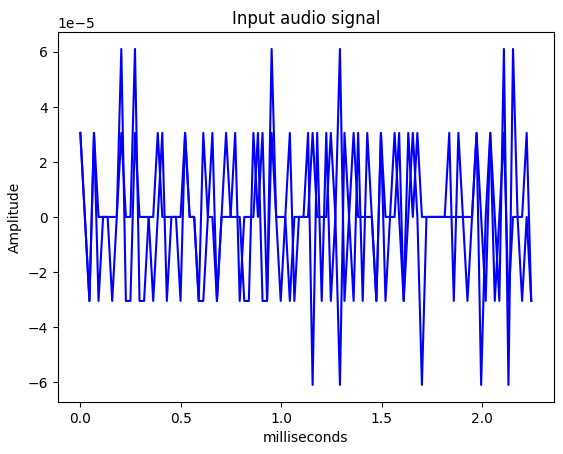
plt.show()

**Output:**

Signal Shape (1398528, 2)

Signal Dtype int16

Signal Duration 31.71 seconds

****

**TEST 2  
Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

freq\_samp, aud = wavfile.read('Audio\_dog.wav')

print('Signal Shape', aud.shape)

print("Signal Dtype",aud.dtype)

print("Signal Duration",round(aud.shape[0]/float(freq\_samp),2),'seconds')

aud = aud / np.power(2,15)

aud = aud[:100]

time\_axis = 1000\*np.arange(0, len(aud),1) / float(freq\_samp)

plt.plot(time\_axis, aud, color = 'b')

plt.xlabel("milliseconds")

plt.ylabel('Amplitude')

plt.title("Input audio signal")

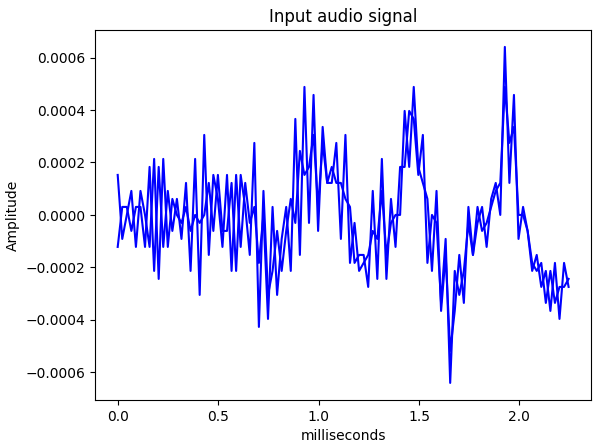
plt.show()

**Output:**

Signal Shape (946500, 2)

Signal Dtype int16

Signal Duration 21.46 seconds

****

**TEST 3  
Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

freq\_samp, aud = wavfile.read('Audio\_carengine.wav')

print('Signal Shape', aud.shape)

print("Signal Dtype",aud.dtype)

print("Signal Duration",round(aud.shape[0]/float(freq\_samp),2),'seconds')

aud = aud / np.power(2,15)

aud = aud[:100]

time\_axis = 1000\*np.arange(0, len(aud),1) / float(freq\_samp)

plt.plot(time\_axis, aud, color = 'b')

plt.xlabel("milliseconds")

plt.ylabel('Amplitude')

plt.title("Input audio signal")

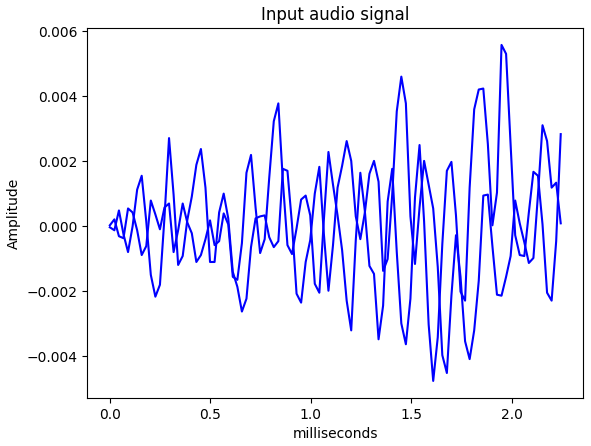
plt.show()

**Output:**

Signal Shape (900515, 2)

Signal Dtype int16

Signal Duration 20.42 seconds



**TEST 4  
Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

freq\_samp, aud = wavfile.read('Audio\_waterfall.wav')

print('Signal Shape', aud.shape)

print("Signal Dtype",aud.dtype)

print("Signal Duration",round(aud.shape[0]/float(freq\_samp),2),'seconds')

aud = aud / np.power(2,15)

aud = aud[:100]

time\_axis = 1000\*np.arange(0, len(aud),1) / float(freq\_samp)

plt.plot(time\_axis, aud, color = 'b')

plt.xlabel("milliseconds")

plt.ylabel('Amplitude')

plt.title("Input audio signal")

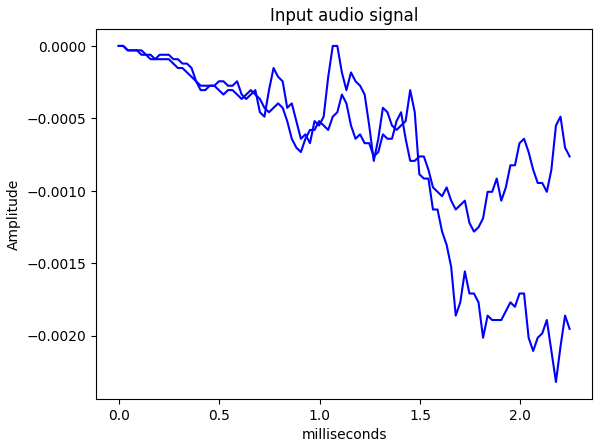
plt.show()

**Output:**

Signal Shape (1310709, 2)

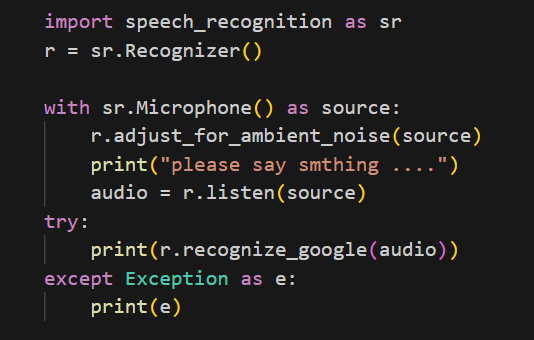
Signal Dtype int16

Signal Duration 29.72 seconds

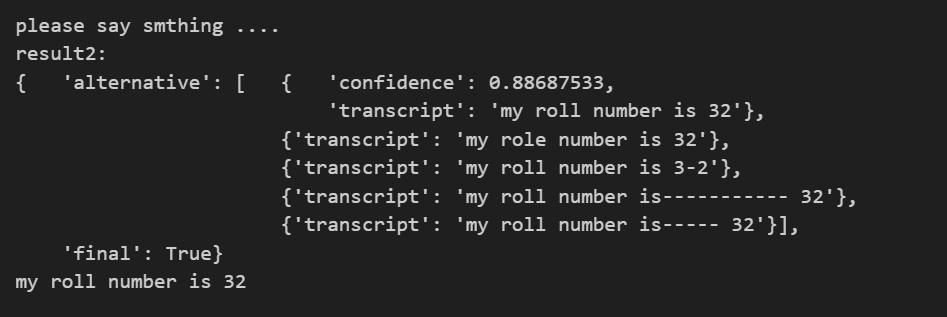


**TASK 5 :**

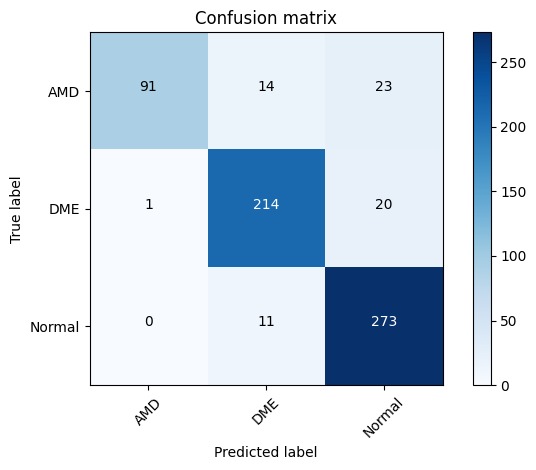
**Code:**

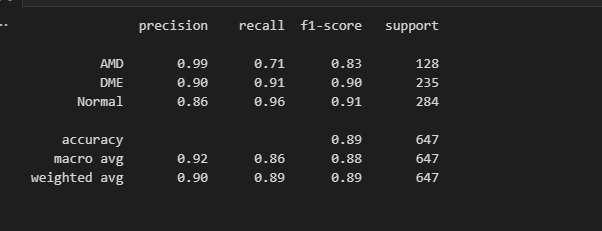
****

**Output:**



**Lab 9**

****

****

**Lab 10**

**Code:**

!pip install tensorflow tensorflow-gpu opencv-python matplotlib

!pip list

import tensorflow as tf

import os

# Avoid OOM errors by setting GPU Memory Consumption Growth

gpus = tf.config.experimental.list\_physical\_devices('GPU')

for gpu in gpus:

    tf.config.experimental.set\_memory\_growth(gpu, True)

tf.config.list\_physical\_devices('GPU')

import cv2

import imghdr

data\_dir = 'data'

image\_exts = ['jpeg','jpg', 'bmp', 'png']

for image\_class in os.listdir(data\_dir):

    for image in os.listdir(os.path.join(data\_dir, image\_class)):

        image\_path = os.path.join(data\_dir, image\_class, image)

        try:

            img = cv2.imread(image\_path)

            tip = imghdr.what(image\_path)

            if tip not in image\_exts:

                print('Image not in ext list {}'.format(image\_path))

                os.remove(image\_path)

        except Exception as e:

            print('Issue with image {}'.format(image\_path))

            # os.remove(image\_path)

import numpy as np

from matplotlib import pyplot as plt

data = tf.keras.utils.image\_dataset\_from\_directory('data')

data\_iterator = data.as\_numpy\_iterator()

batch = data\_iterator.next()

fig, ax = plt.subplots(ncols=4, figsize=(20,20))

for idx, img in enumerate(batch[0][:4]):

    ax[idx].imshow(img.astype(int))

    ax[idx].title.set\_text(batch[1][idx])

data = data.map(lambda x,y: (x/255, y))

data.as\_numpy\_iterator().next()

train\_size = int(len(data)\*.7)

val\_size = int(len(data)\*.2)

test\_size = int(len(data)\*.1)

train\_size

train = data.take(train\_size)

val = data.skip(train\_size).take(val\_size)

test = data.skip(train\_size+val\_size).take(test\_size)

train

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

model = Sequential()

model.add(Conv2D(16, (3,3), 1, activation='relu', input\_shape=(256,256,3)))

model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))

model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3), 1, activation='relu'))

model.add(MaxPooling2D())

model.add(Flatten())

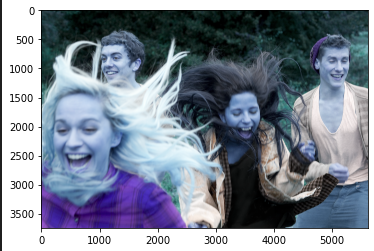
model.add(Dense(256, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])

model.summary()

**Input Image:**

****

**Output:**

****

**Lab 11**

**Code:**

import nltk

nltk.download('punkt')

from nltk.tokenize import word\_tokenize,sent\_tokenize

text="joe waited for the train. The train was late sam and marry took the bus. I looked for them"

print("\nOriginal String")

print(text)

print("\nTokenize word sentence wise")

result=[word\_tokenize(t) for t in sent\_tokenize(text)]

print("Read the list")

for s in result:

  print(s)

text="joe waited for the train. The train was late sam and marry took the bus. I looked for them"

print("\nOriginal String")

print(text)

print("\nlist of words")

print(word\_tokenize(text))

**Output:**

Original String

joe waited for the train. The train was late sam and marry took the bus. I looked for them

list of words

['joe', 'waited', 'for', 'the', 'train', '.', 'The', 'train', 'was', 'late', 'sam', 'and', 'marry', 'took', 'the', 'bus', '.', 'I', 'looked', 'for', 'them']

**Lab 12**

**Code:**

# Import all the libraries

import os

import itertools

import pandas as pd

import string

import nltk

import numpy as np

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from pandas import DataFrame

from functools import reduce

#Pre-processing of the data

#It returns a list of all words in an email including numbers

def prepare\_the\_data(fname):

    flist = open(fname,errors="surrogateescape").readlines()

    ls= [s.rstrip('\n') for s in flist]

    s=[]

    for i in ls:

        s.append(i.split())

    s= list(itertools.chain(\*s))

    #text converted to lower case

    lst = [x.lower() for x in s]

    # Removes the puctuation: !"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~

    lst = [''.join(c for c in s if c not in string.punctuation) for s in lst]

    # Removes the empty strings

    lst = [s for s in lst if s]

    return lst

# It creates a nested list for all the emails in a folder

# e.x. Train\_spam(a directory) contains 123 emails, it creates a nested list of size 123.

def extract\_data(dir\_name,l):

    for entry in os.listdir(dir\_name):

        path = os.path.join(dir\_name,entry)

        l.append(prepare\_the\_data(path))

# Learning Model

# Generate the dictionary for Training Data

# It will give all the unique words and count the number of occurences of each word

# It will take the nested list as input, for train\_spam\_list and train\_ham\_list

def generate\_dictionary(lst):

    # chnage the nested list to one single list

    single\_list = reduce(lambda x,y: x+y, lst)

    #count the no of unique words

    uniqueWordCount = 0

    count = {}

    for word in single\_list:

        if word in count :

            count[word] += 1

        else:

            count[word] = 1

    return(count)

# Using add-one smoothing.

# Calculate the conditional probability of a word given the class. (P(w|c)).

# For each class returns a dictionary of words as keys and prob of word given class as value.

def compute\_conditional\_probability(dictionary,vocubalary):

    log\_prob\_word\_class = {}

    v\_wc = len(vocubalary)

    #print("No of words in the vocubalray: " + str(v\_wc))

    class\_wc = sum(dictionary.values())

    for word in vocubalary:

        count\_word\_class = dictionary.get(word) if dictionary.get(word)!=None else 0

        log\_prob\_word\_class[word] = np.log(count\_word\_class+1/(class\_wc+v\_wc))

    return log\_prob\_word\_class

#Train the naive Bayes

def train\_naive\_bayes(spam\_dict,ham\_dict,spam\_list,ham\_list):

    # Extract vocublary of the training dataset

    # v\_wc - no of words in the vocabulary

    vocubalary = set(([\*spam\_dict,\*ham\_dict]))

# Remove stop words

# Takes input a nested list

# returns a nested list after removing the stop words

def remove\_stop\_words(dic):

    stopwords = open("stopwords.txt",errors="surrogateescape").readlines()

    stopwords= [s.rstrip('\n') for s in stopwords]

    for word in stopwords:

        if(word in dic.keys()):

            #print(word)

            del dic[word]

    return dic

    # N\_c = No of emails in the training dataset classified as c where c is either ham or spam

    N\_spam = len(spam\_list)

    N\_ham = len(ham\_list)

    # N\_doc = NO of emails in the training dataset

    N\_doc = len(spam\_list)+len(ham\_list)

    # Log probabiloty of classes ham and spam

    prior\_ham = np.log(N\_ham/N\_doc)

    prior\_spam = np.log(N\_spam/N\_doc)

#     print("Prior probability of ham: "+str(np.exp(prior\_ham)))

#     print("Prior probability of spam: "+str(np.exp(prior\_spam)))

    # Conditional Probability of each word in vocubalary given class(ham or spam)

    log\_prob\_word\_spam = {}

    log\_prob\_word\_ham = {}

    log\_prob\_word\_spam = compute\_conditional\_probability(spam\_dict,vocubalary)

#     print(log\_prob\_word\_spam.get('are'))

    log\_prob\_word\_ham = compute\_conditional\_probability(ham\_dict,vocubalary)

#     print(log\_prob\_word\_ham.get('are'))

    return prior\_ham,prior\_spam,log\_prob\_word\_ham,log\_prob\_word\_spam

#Naive Bayes Classifier for Test Data

def naive\_bayes\_classifier(org\_class\_name,ls,spam\_dict,ham\_dict,spam,ham):

    for email in org\_class\_name:

        prob\_spam = spam

        prob\_ham = ham

        for word in email:

            try:

                prob\_spam += spam\_dict.get(word) if spam\_dict.get(word)!=None else 0

            except:

                prob\_spam +=np.log(1)

            try:

                prob\_ham += ham\_dict.get(word) if ham\_dict.get(word)!=None else 0

            except KeyError:

                prob\_ham += np.log(1)

        if (prob\_ham)>(prob\_spam):

            ls.append('ham')

        else:

            ls.append('spam')

#Visualize the data

# input - a nested list

def visulaize\_data(lst):

    words = reduce(lambda x,y: x+y, lst)

    #words\_spam=list(filter(('subject').\_\_ne\_\_, words\_spam))

    df = DataFrame (words,columns=['words'])

    text = " ".join(review for review in df.words)

    wc = WordCloud(width=512,height=512).generate(text)

    plt.figure(figsize=(10,8),facecolor='k')

    plt.imshow(wc)

    plt.axis('off')

    plt.tight\_layout(pad=0)

    plt.show()

################################################# !!!!!SPAM!!!!! ##################################################

# Train spam List

train\_spam = 'corpus/train/spam' ####### Enter where your spam folder is located

train\_spam\_list = []

#pre process the data

extract\_data(train\_spam,train\_spam\_list)

print("No of emails tagged as spam:" + str(len(train\_spam\_list)))

visulaize\_data(train\_spam\_list)

#Genearte a dictionary of unique words and their count for spam with and without stop words

train\_spam\_dict = generate\_dictionary(train\_spam\_list)

print("No of unique words in train/spam :" + str(len(train\_spam\_dict)))

train\_spam\_dict\_without\_stopwords = train\_spam\_dict.copy()

train\_spam\_dict\_without\_stopwords = remove\_stop\_words(train\_spam\_dict\_without\_stopwords)

print("No of unique words in train/spam without stop words :" + str(len(train\_spam\_dict\_without\_stopwords)))

################################################# !!!!!HAM!!!!! ##################################################

# Train spam List

train\_ham = 'corpus/train/ham' ####### Enter where your ham folder is located

train\_ham\_list = []

#pre process the data

extract\_data(train\_ham,train\_ham\_list)

print("No of emails tagged as ham:" + str(len(train\_ham\_list)))

visulaize\_data(train\_ham\_list)

#Genearte a dictionary of unique words and their count for spam with and without stop words

train\_ham\_dict = generate\_dictionary(train\_ham\_list)

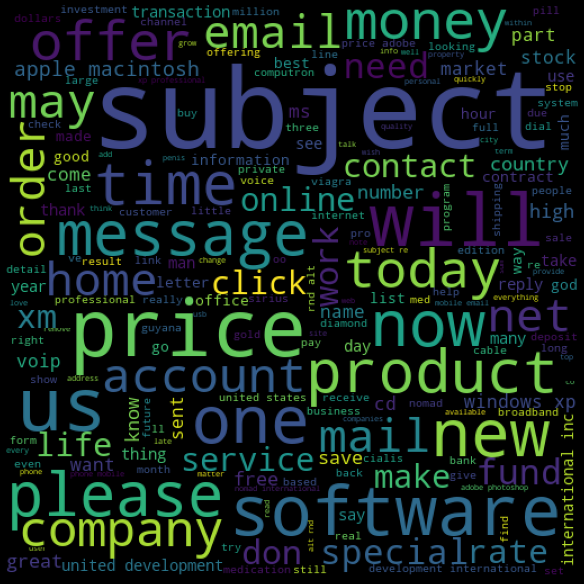
print("No of unique words in train/ham :" + str(len(train\_ham\_dict)))

train\_ham\_dict\_without\_stopwords = train\_ham\_dict.copy()

train\_ham\_dict\_without\_stopwords = remove\_stop\_words(train\_ham\_dict\_without\_stopwords)

print("No of unique words in train/ham without stop words :" + str(len(train\_ham\_dict\_without\_stopwords)))

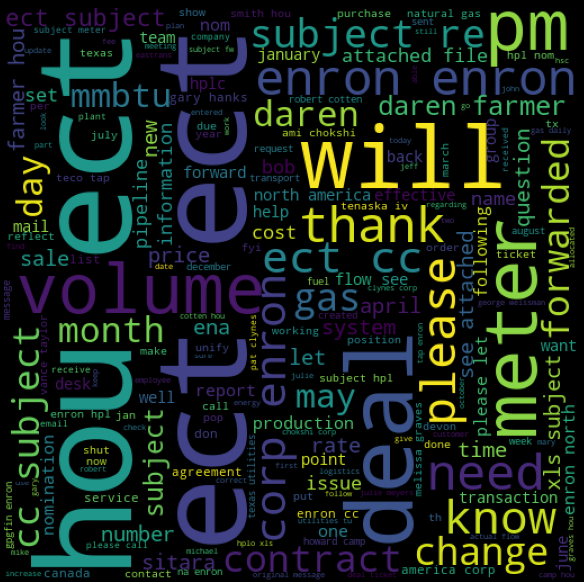
No of emails tagged as spam:123

****

No of unique words in train/spam :6256

No of unique words in train/spam without stop words :6139

No of emails tagged as ham:340

****

No of unique words in train/ham :5874

No of unique words in train/ham without stop words :5753

# Lets train our niave bayes model using trainind data for both with and withut stopwords

# With stop words

prior\_ham,prior\_spam,log\_prob\_word\_ham,log\_prob\_word\_spam = train\_naive\_bayes(train\_spam\_dict,train\_ham\_dict,train\_spam\_list,train\_ham\_list)

# print(len(log\_prob\_word\_ham))

# print(len(log\_prob\_word\_spam))

# Without stop words

prior\_ham\_sp,prior\_spam\_sp,log\_prob\_word\_ham\_sp,log\_prob\_word\_spam\_sp = train\_naive\_bayes(train\_spam\_dict\_without\_stopwords,train\_ham\_dict\_without\_stopwords,train\_spam\_list,train\_ham\_list)

# print(len(log\_prob\_word\_ham\_sp))

# print(len(log\_prob\_word\_spam\_sp))

def remove\_stop\_words\_from\_nested\_list(ls):

    stopwords = open("stopwords.txt",errors="surrogateescape").readlines()

    stopwords= [s.rstrip('\n') for s in stopwords]

    for stop\_word in stopwords:

        for email in ls:

            for word in email:

                if(word==stop\_word):

                    email.remove(word)

    return ls

# Test spam Emails

test\_spam = 'corpus/test/spam'

test\_spam\_list = []

#pre process the data

extract\_data(test\_spam,test\_spam\_list)

# print(len(test\_spam\_list))

# Without stop words

test\_spam\_list\_withput\_stop\_words = test\_spam\_list.copy()

test\_spam\_list\_withput\_stop\_words = remove\_stop\_words\_from\_nested\_list(test\_spam\_list\_withput\_stop\_words)

# Test Ham emails

test\_ham = 'corpus/test/ham'

test\_ham\_list = []

#pre process the data

extract\_data(test\_ham,test\_ham\_list)

# print(len(test\_ham\_list))

# Without stop words

test\_ham\_list\_withput\_stop\_words = test\_ham\_list.copy()

test\_ham\_list\_withput\_stop\_words = remove\_stop\_words\_from\_nested\_list(test\_ham\_list\_withput\_stop\_words)

#contains classified output of naive\_bayes\_classifier for testing spam email

classifier\_spam=[]

#Calling naive\_bayes\_classifier for testing spam emails

naive\_bayes\_classifier(test\_spam\_list,classifier\_spam,log\_prob\_word\_spam,log\_prob\_word\_ham,prior\_spam,prior\_ham)

# Without stop words

classifier\_spam\_without\_sw=[]

#Calling naive\_bayes\_classifier for testing spam emails

naive\_bayes\_classifier(test\_spam\_list\_withput\_stop\_words,classifier\_spam\_without\_sw,log\_prob\_word\_spam\_sp,log\_prob\_word\_ham\_sp,prior\_spam\_sp,prior\_ham\_sp)

# print(len(classifier\_spam\_without\_sw))

#contains classified output of naive\_bayes\_classifier for testing ham email

classifier\_ham=[]

#Calling naive\_bayes\_classifier for testing ham emails

naive\_bayes\_classifier(test\_ham\_list,classifier\_ham,log\_prob\_word\_spam,log\_prob\_word\_ham,prior\_spam,prior\_ham)

# Without stop words

classifier\_ham\_without\_sw=[]

#Calling naive\_bayes\_classifier for testing spam emails

naive\_bayes\_classifier(test\_ham\_list\_withput\_stop\_words,classifier\_ham\_without\_sw,log\_prob\_word\_spam\_sp,log\_prob\_word\_ham\_sp,prior\_spam\_sp,prior\_ham\_sp)

# print(len(classifier\_ham\_without\_sw))

# We are going to select various top % of features (words) in training dataset seperatley for spam and ham.

# percentage = 0.8, 80% of highest frequency words from the dictionary of spam and ham.

def freq\_based\_feature\_selection(percentage,dic):

    dic = {k: v for k, v in sorted(dic.items(), key=lambda item: item[1], reverse=True)}

#     print(len(dic))

    for i in range(0,len(dic)-int(np.ceil(percentage\*len(dic)))):

        dic.popitem()

#     print(len(dic))

    return dic

accuracy = {}

accuracy\_ham = {}

accuracy\_spam = {}

for i in [1,0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1]:

    sorted\_train\_spam\_dict = train\_spam\_dict.copy()

    sorted\_train\_spam\_dict = freq\_based\_feature\_selection(i,sorted\_train\_spam\_dict)

    sorted\_train\_ham\_dict = train\_ham\_dict.copy()

    sorted\_train\_ham\_dict = freq\_based\_feature\_selection(i,sorted\_train\_ham\_dict)

    # Train the Model

    prior\_ham\_fr,prior\_spam\_fr,log\_prob\_word\_ham\_fr,log\_prob\_word\_spam\_fr = train\_naive\_bayes(sorted\_train\_spam\_dict,sorted\_train\_ham\_dict,train\_spam\_list,train\_ham\_list)

    #contains classified output of naive\_bayes\_classifier for testing spam email

    classifier\_spam\_fr=[]

    #Calling naive\_bayes\_classifier for testing spam emails

    naive\_bayes\_classifier(test\_spam\_list,classifier\_spam\_fr,log\_prob\_word\_spam\_fr,log\_prob\_word\_ham\_fr,prior\_spam\_fr,prior\_ham\_fr)

    #contains classified output of naive\_bayes\_classifier for testing ham email

    classifier\_ham\_fr=[]

    #Calling naive\_bayes\_classifier for testing ham emails

    naive\_bayes\_classifier(test\_ham\_list,classifier\_ham\_fr,log\_prob\_word\_spam\_fr,log\_prob\_word\_ham\_fr,prior\_spam\_fr,prior\_ham\_fr)

    #Calculate the accuracy

    Accuracy\_for\_ham = classifier\_ham\_fr.count('ham')/len(classifier\_ham\_fr)

#     print('Accuracy for ham:'+ str(Accuracy\_for\_ham))

    accuracy\_ham[i] = Accuracy\_for\_ham

    Accuracy\_for\_spam = classifier\_spam\_fr.count('spam')/len(classifier\_spam\_fr)

    accuracy\_spam[i] = Accuracy\_for\_spam

#     print('Accuracy for spam:'+str(Accuracy\_for\_spam))

    Overall\_Accuracy = (classifier\_ham\_fr.count('ham')+classifier\_spam\_fr.count('spam'))/(len(classifier\_ham\_fr)+len(classifier\_spam\_fr))

#     print('Overall Accuracy'+str(Overall\_Accuracy))

    accuracy[i]=Overall\_Accuracy

def print\_accuracy\_of\_model():

    # Accuracy of the Naive Bayes model for test data

    #Calculate the accuracy

    print("Accuracy for ham:")

    print(classifier\_ham.count('ham')/len(classifier\_ham))

    print("Accuracy for spam:")

    print(classifier\_spam.count('spam')/len(classifier\_spam))

    print('Overall Accuracy:')

    print((classifier\_ham.count('ham')+classifier\_spam.count('spam'))/(len(classifier\_ham)+len(classifier\_spam)))

def print\_accuracy\_for\_stop\_words():

    # Accuracy of the Naive Bayes model for test data without stop words

    # Calculate the accuracy

    print("\nAccuracy for ham without stop words:")

    print(classifier\_ham\_without\_sw.count('ham')/len(classifier\_ham\_without\_sw))

    print("Accuracy for spam without stop words:")

    print(classifier\_spam\_without\_sw.count('spam')/len(classifier\_spam\_without\_sw))

    print('Overall Accuracy without stop words:')

    print((classifier\_ham\_without\_sw.count('ham')+classifier\_spam\_without\_sw.count('spam'))/(len(classifier\_ham\_without\_sw)+len(classifier\_spam\_without\_sw)))

    print("\nConclusion: Removing stop words from the dataset didn't help in improving the accuracy as the no of token were "

      +" still very large, they were changed from "+str(len(log\_prob\_word\_ham))+" to " + str(len(log\_prob\_word\_ham\_sp))+".")

def print\_and\_plt():

    print("Key Value means percentage of tokens selected from both ham and spam with highest frequency.")

    print("1 means when no features selection is done, we consider all the tokens for classification.\n")

    print("Overall Accuracy:")

    print(accuracy)

    list\_1 = sorted(accuracy.items()) # sorted by key, return a list of tuples

    x, y = zip(\*list\_1) # unpack a list of pairs into two tuples

    plt.plot(x, y, label='Overall Accuracy')

    print("\nHam Accuracy")

    print(accuracy\_ham)

    list\_2 = sorted(accuracy\_ham.items()) # sorted by key, return a list of tuples

    x, y = zip(\*list\_2) # unpack a list of pairs into two tuples

    plt.plot(x, y,label='Ham Accuracy')

    print("\nSpam Accuracy")

    print(accuracy\_spam)

    list\_3 = sorted(accuracy\_spam.items()) # sorted by key, return a list of tuples

    x, y = zip(\*list\_3) # unpack a list of pairs into two tuples

    plt.plot(x, y, label='Spam Accuracy')

    plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

    plt.show()

    print("Conclusion: I would go with selecting 30 to 35 % of tokens with highest frequency (feature selection),"

      +" for optimizing the accuracy of"+

      " both sets of email(spam and ham) and overall accuracy as observable from the above graph.\n")

def what\_to\_print(sp,feature\_selection):

    if(sp==0 and feature\_selection ==1):

        print\_and\_plt()

    elif(sp==1 and feature\_selection ==0):

        print\_accuracy\_of\_model()

        print\_accuracy\_for\_stop\_words()

    elif(sp==1 and feature\_selection ==1):

        print\_and\_plt()

        print\_accuracy\_for\_stop\_words()

    else:

        print\_accuracy\_of\_model()

what\_to\_print(1,1)

**Output:**

Key Value means percentage of tokens selected from both ham and spam with highest frequency.

1 means when no features selection is done, we consider all the tokens for classification.

Overall Accuracy:

{1: 0.9142259414225942, 0.9: 0.9016736401673641, 0.8: 0.9100418410041841, 0.7: 0.9163179916317992, 0.6: 0.9163179916317992, 0.5: 0.9100418410041841, 0.4: 0.9309623430962343, 0.3: 0.9351464435146444, 0.2: 0.9288702928870293, 0.1: 0.9142259414225942}

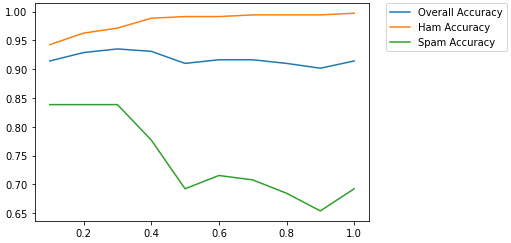
Ham Accuracy

{1: 0.9971264367816092, 0.9: 0.9942528735632183, 0.8: 0.9942528735632183, 0.7: 0.9942528735632183, 0.6: 0.9913793103448276, 0.5: 0.9913793103448276, 0.4: 0.9885057471264368, 0.3: 0.9712643678160919, 0.2: 0.9626436781609196, 0.1: 0.9425287356321839}

Spam Accuracy

{1: 0.6923076923076923, 0.9: 0.6538461538461539, 0.8: 0.6846153846153846, 0.7: 0.7076923076923077, 0.6: 0.7153846153846154, 0.5: 0.6923076923076923, 0.4: 0.7769230769230769, 0.3: 0.8384615384615385, 0.2: 0.8384615384615385, 0.1: 0.8384615384615385}

**Evaluation Graph:**



Conclusion: I would go with selecting 30 to 35 % of tokens with highest frequency (feature selection), for optimizing the accuracy of both sets of email(spam and ham) and overall accuracy as observable from the above graph.

Accuracy for ham without stop words:

0.9971264367816092

Accuracy for spam without stop words:

0.6923076923076923

Overall Accuracy without stop words:

0.9142259414225942

Conclusion: Removing stop words from the dataset didn't help in improving the accuracy as the no of token were still very large, they were changed from 10416 to 10294.

s