SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

Deemed to be **University** *u/s 3 of UGC Act, 1956*

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM DISTRICT

SCHOOL OF COMPUTING
AND DEPARTMENT OF COMPUTER SCIENCE

Course Code: 18CSC305J

Course Name: Artificial Intelligence

Faculty Incharge: Dr. B.Baranidharan

LAB REPORT

Name: Rashi Agarwal

Register Number: RA1911003010015

Section: CSE A1

SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

Deemed to be **University** *u/s 3 of UGC Act, 1956*

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM DISTRICT

BONAFIDE CERTIFICATE

Register No: RA1911003010015

This is to certify that this AI Lab Experiment Record is the bonafide work done by **Rashi Agarwal** of B.Tech CSE Department, SRM Institute of Science and Technology during the academic year 2021-2022 under my supervision.

SIGNATURE

Dr.B.Baranidharan
Designation
Department
SRM IST , SRM Nagar
Potheri , Kattankulathur
Tamil Nadu 603203

SIGNATURE

Course Coordinator
Associate Professor,
SRM IST , SRM Nagar
Potheri , Kattankulathur
Tamil Nadu 603203

ACKNOWLEDGEMENT

I express my heartfelt thanks to the honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all my endeavors.

I would like to extend my gratitude to the **Registrar Dr. S. Ponnusamy**, for his encouragement

I express my sincere thanks to the **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

I also wish to thank the C**hairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete the lab experiments and this report

I am deeply grateful to the **Course project Internal guide Dr.B.Baranidharan , Associate Professor , Department of Computer Science and Engineering**, for his assistance, timely suggestion and guidance throughout the duration of the lab sessions

I extend my gratitude to **Dr.M.Pushpalatha of the Department of Computing Technologies** and Departmental colleagues for their Support.

Finally, I thank my parents and friends near and dear ones who directly and indirectly contributed to the successful completion of the report. Above all, I thank the almighty for showering his blessings on me to complete the lab experiments and this report

INDEX

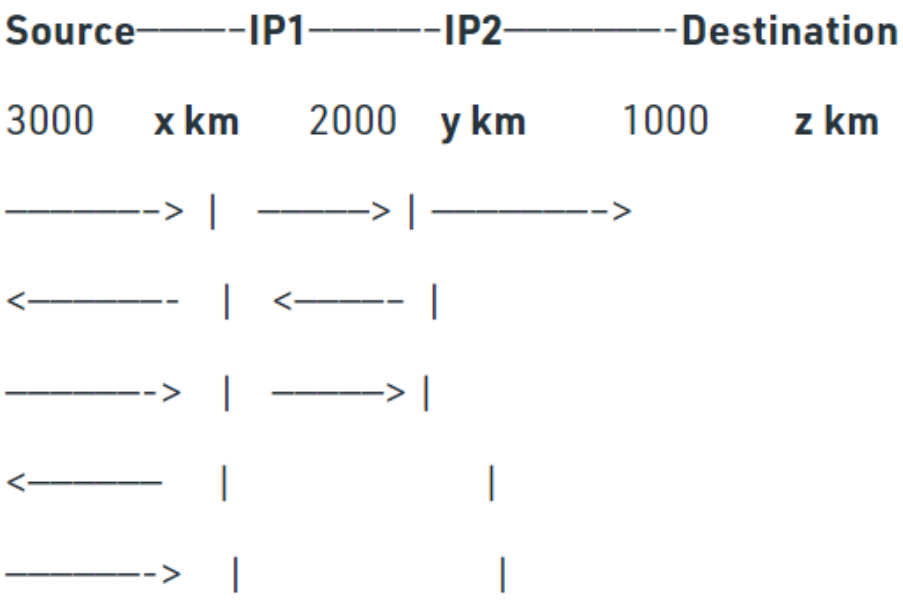
Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Ex no:1 Toy Problem Solving

Aim:
To implement Camel Banana Problem

Requirements:
Python, AWS

Introduction:
A person has 3000 bananas and a camel. The person wants to transport the maximum number of bananas to a destination which is 1000 KMs away, using only the camel as a mode of transportation. The camel cannot carry more than 1000 bananas at a time and eats a banana every km it travels. What is the maximum number of bananas that can be transferred to the destination using only camel (no other mode of transportation is allowed).
We have a total of 3000 bananas.
The destination is 1000KMs
Only 1 mode of transport.
Camel can carry a maximum of 1000 banana at a time.
Camel eats a banana every km it travels.

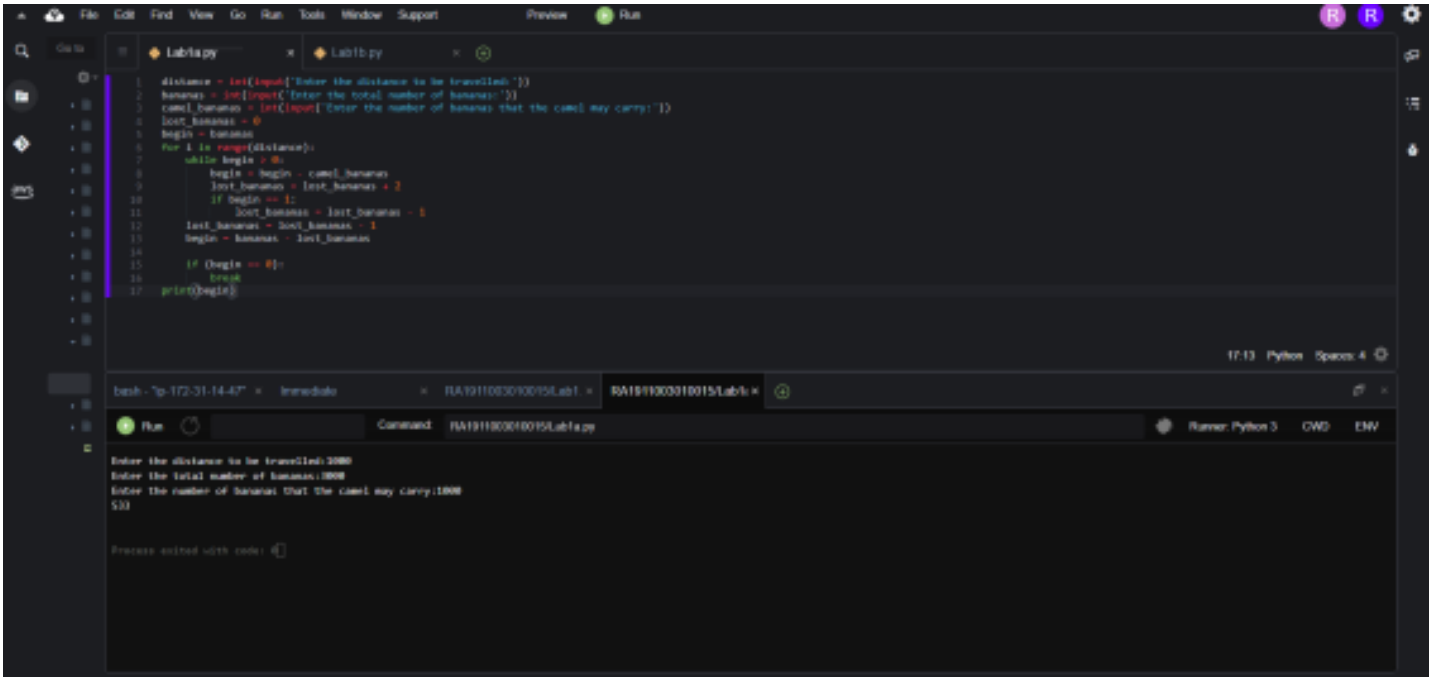


Code:

```
bs = int(input("Enter total number of available bananas at the start"))
distance = int(input("Enter the total distance to be travelled by the camel "))
maxload = int(input("Enter max number of bananas camel can transport at a time"))
lost = 0
temp = bs
for i in range(distance):
while(temp>0):
```

```
temp = temp - maxload
if temp==1:
lost = lost - 1
lost = lost + 2
lost = lost - 1
temp = bs - lost
if temp == 0:
break
print(temp)
```

Program Output:



Aim:
To implement Max Ticket Problem:

Requirements:
Python, AWS

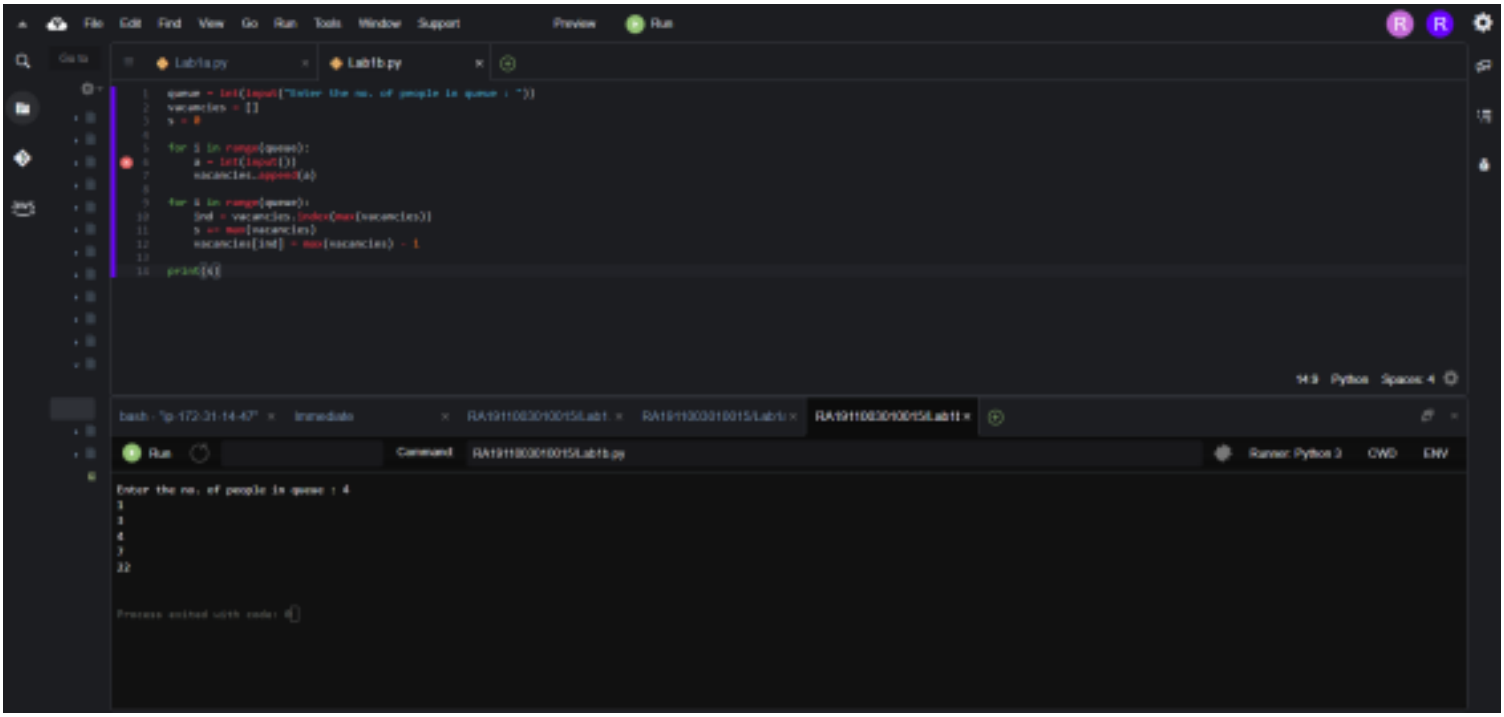
Introduction:
Given array seats[] where seat[i] is the number of vacant seats in the ith row in a stadium for a cricket match. There are N people in a queue waiting to buy the tickets. Each seat costs equal to the number of vacant seats in the row it belongs to. The task is to maximize the profit by selling the tickets to N people.

Code:

```
m = int(input("Enter number of people standing in queue"))
n = int(input("Enter number of rows which are vacant"))
totalearning = 0
list = []
for i in range(0,n):
    ele = int(input())
    list.append(ele)
for i in range(0,m):
    list.sort(reverse=True)
    totalearning = totalearning + list[0]
```

```
        list[0]=list[0]-1
        list.sort(reverse=True)
        if(list[0]<=0):
            break
print("Thus the total number of money earned by maximising ticket amount is ")
print(totalearning)
```

Program Output:



Result:
The Toy Problems were successfully executed.

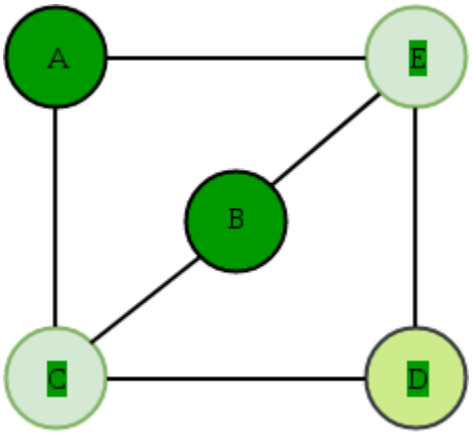
Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Ex no:2 Graph Coloring

Aim:
To implement Graph Coloring Code

Requirements:
Python, Google Colab

Introduction:
Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints. Vertex coloring is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color. The other graph coloring problems like Edge Coloring (No vertex is incident to two edges of same color) and Face Coloring (Geographical Map Coloring) can be transformed into vertex coloring.

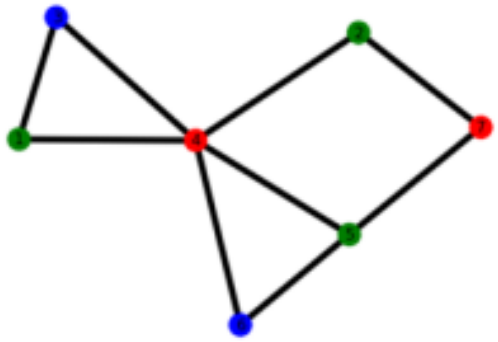


Code:

```
import matplotlib.pyplot as plt
import networkx as nx
G = nx.Graph()
colors = {0:"red", 1:"green", 2:"blue"}
G.add_nodes_from([1,2,3,4,5, 6, 7])
G.add_edges_from([(1,4), (1,3), (2,4), (2,7), (3,4), (4,5), (4,6), (5,6), (5,7)])
d = nx.coloring.greedy_color(G, strategy = "largest_first")
node_colors = []
for i in sorted (d.keys()):
node_colors.append(colors[d[i]])
nx.draw(G, node_color = node_colors, with_labels = True, width = 5)
plt.show()
```

Program Output:

```
In [5]: import matplotlib.pyplot as plt
        import networkx as nx
        G = nx.Graph()
        colors = {0:"red", 1:"green", 2:"blue"}
        G.add_nodes_from([1,2,3,4,5, 6, 7])
        G.add_edges_from([(1,4), (1,3), (2,4), (2,7), (3,4), (4,5), (4,6), (5,6), (5,7)])
        d = nx.coloring.greedy_color(G, strategy = "largest_first")
        node_colors = []
        for i in sorted (d.keys()):
            node_colors.append(colors[d[i]])
        nx.draw(G, node_color = node_colors, with_labels = True, width = 5)
        plt.show()
```



Result:
The Graph Coloring was successfully executed.

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Ex no:3 Cryptarithmetic Problem

Aim:
To implement Cryptarithmetic Problem Code

Requirements:
Python, AWS

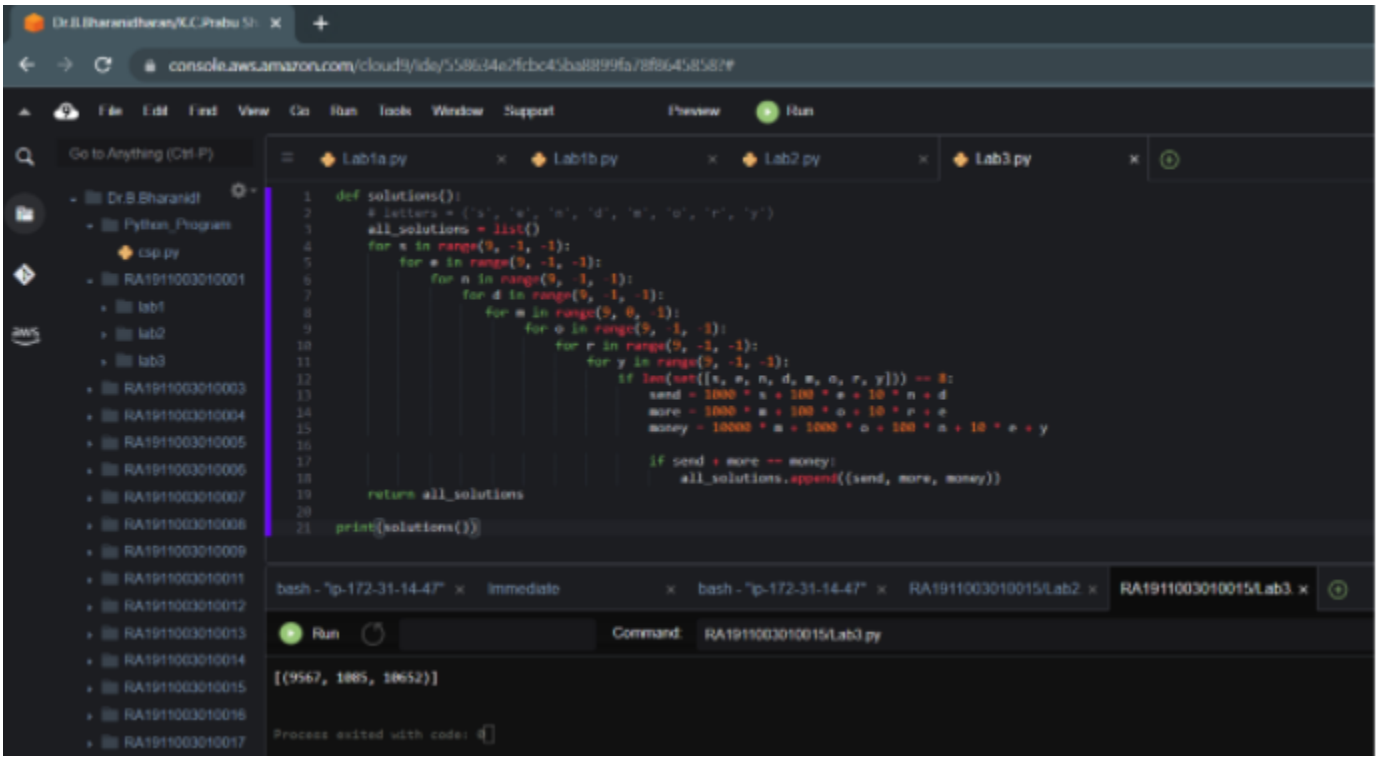
Introduction:
Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.
The rules or constraints on a cryptarithmetic problem are as follows:
There should be a unique digit to be replaced with a unique alphabet.
The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
Digits should be from 0-9 only.
There should be only one carry forward, while performing the addition operation on a problem.
The problem can be solved from both sides, i.e., left hand side (L.H.S), or right hand side (R.H.S)

```
  SEND
+MORE
------
 MONEY
------
```

Code:

```
def solutions():
# letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
    all_solutions = list()
      for s in range(9, -1, -1):
        for e in range(9, -1, -1):
          for n in range(9, -1, -1):
            for d in range(9, -1, -1):
              for m in range(9, 0, -1):
                for o in range(9, -1, -1):
                  for r in range(9, -1, -1):
                    for y in range(9, -1, -1):
                      if len(set([s, e, n, d, m, o, r, y])) == 8:
                        send = 1000 * s + 100 * e + 10 * n + d
                        more = 1000 * m + 100 * o + 10 * r + e
                        money = 10000 * m + 1000 * o + 100 * n + 10 * e + y
                        if send + more == money:
                          all_solutions.append((send, more, money))
      return all_solutions
print(solutions())
```

Program Output:



Result:

The was cryptarithmetic problem is successfully solved and executed.

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Lab 4 - Implementation and Analysis of DFS and BFS for an Application

- **Breadth First Search:**

Aim:
To implement a python program of BFS and apply any combination of arithmetic operators over it..

Requirements:
GCC Compiler

Introduction:
1. Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
2. It employs the following rules.
   Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
   Rule 2 − If no adjacent vertex is found, remove the first vertex from the queue.
   Rule 3 − Repeat Rule 1 and Rule 2 until the queue is empty.

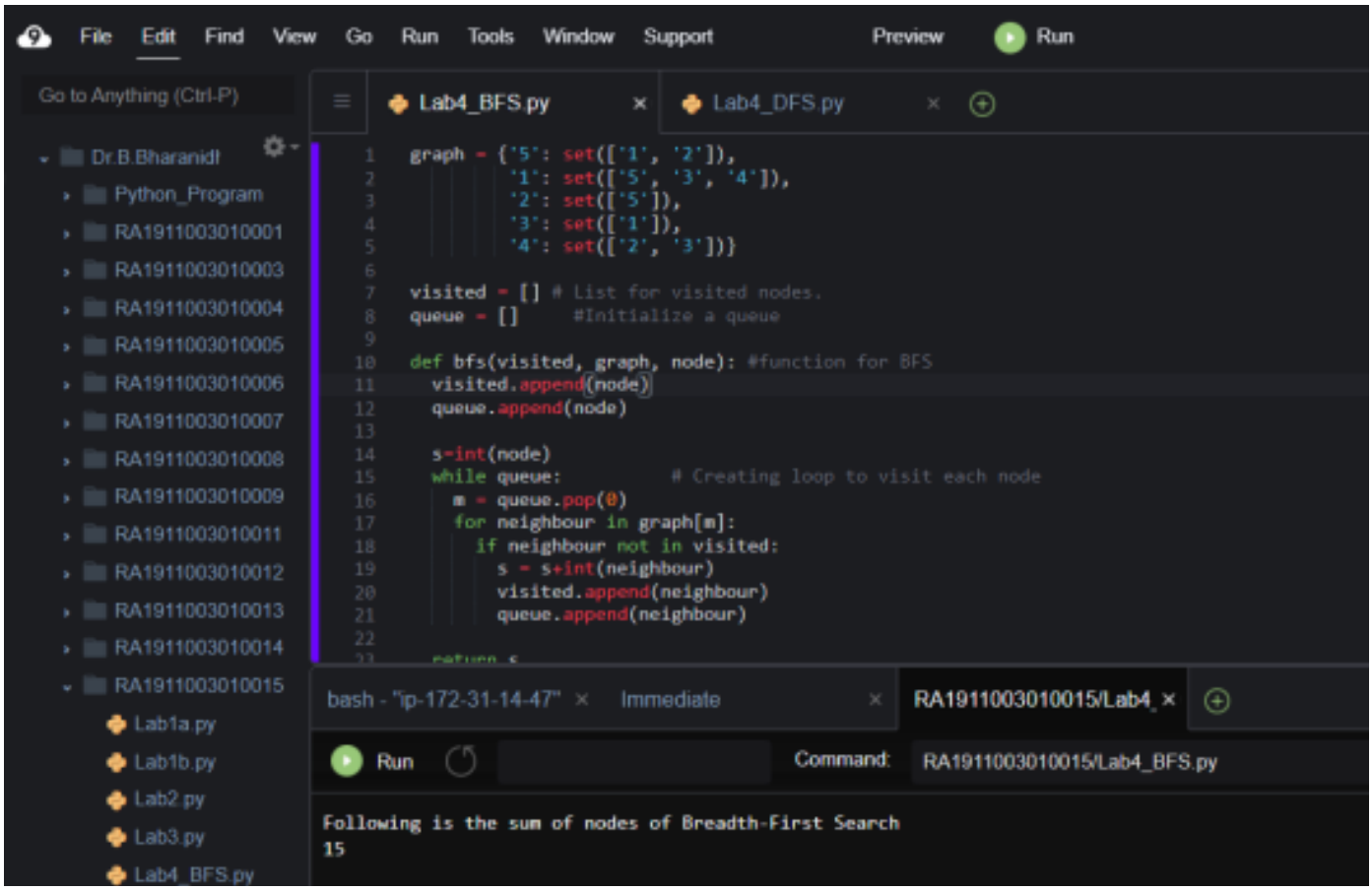
Algorithm:
1. We will create the graph for which we will use the breadth-first search.
2. After creation, we will create two lists, one to store the visited node of the graph and another one for storing the nodes in the queue.
3. After the above process, we will declare a function with the parameters as visited nodes, the graph itself and the node respectively. And inside a function, we will keep appending the visited and queue lists.
4. Then we will run the while loop for the queue for visiting the nodes and then will remove the same node.
5. At last, we will run the for loop to check the not visited nodes and then append the same from the visited and queue list and add them to the sum
6. As the driver code, we will call the user to define the bfs function with the first node we wish to visit.

Complexity:
Time Complexity: O(V+E) where V is vertices and E is edges
Space Complexity: O(w) where w is the maximum width of the tree

Code:
```
graph = {'5': set(['1', '2']),
         '1': set(['5', '3', '4']),
         '2': set(['5']),
         '3': set(['1']),
         '4': set(['2', '3'])}
visited = [] # List for visited nodes.
queue = [] #Initialize a queue

def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)
```

```
    s=int(node)
    while queue: # Creating loop to visit each node
        m = queue.pop(0)
        for neighbour in graph[m]:
            if neighbour not in visited:
                s = s+int(neighbour)
                visited.append(neighbour)
                queue.append(neighbour)
    return s
    print("Following is the sum of nodes of Breadth-First Search")
    x = bfs(visited, graph, '5')
    print(x)
```

Program Output:



- **Depth First Search:**

Aim:
To implement a python program of DFS and apply any combination of arithmetic operators over it.

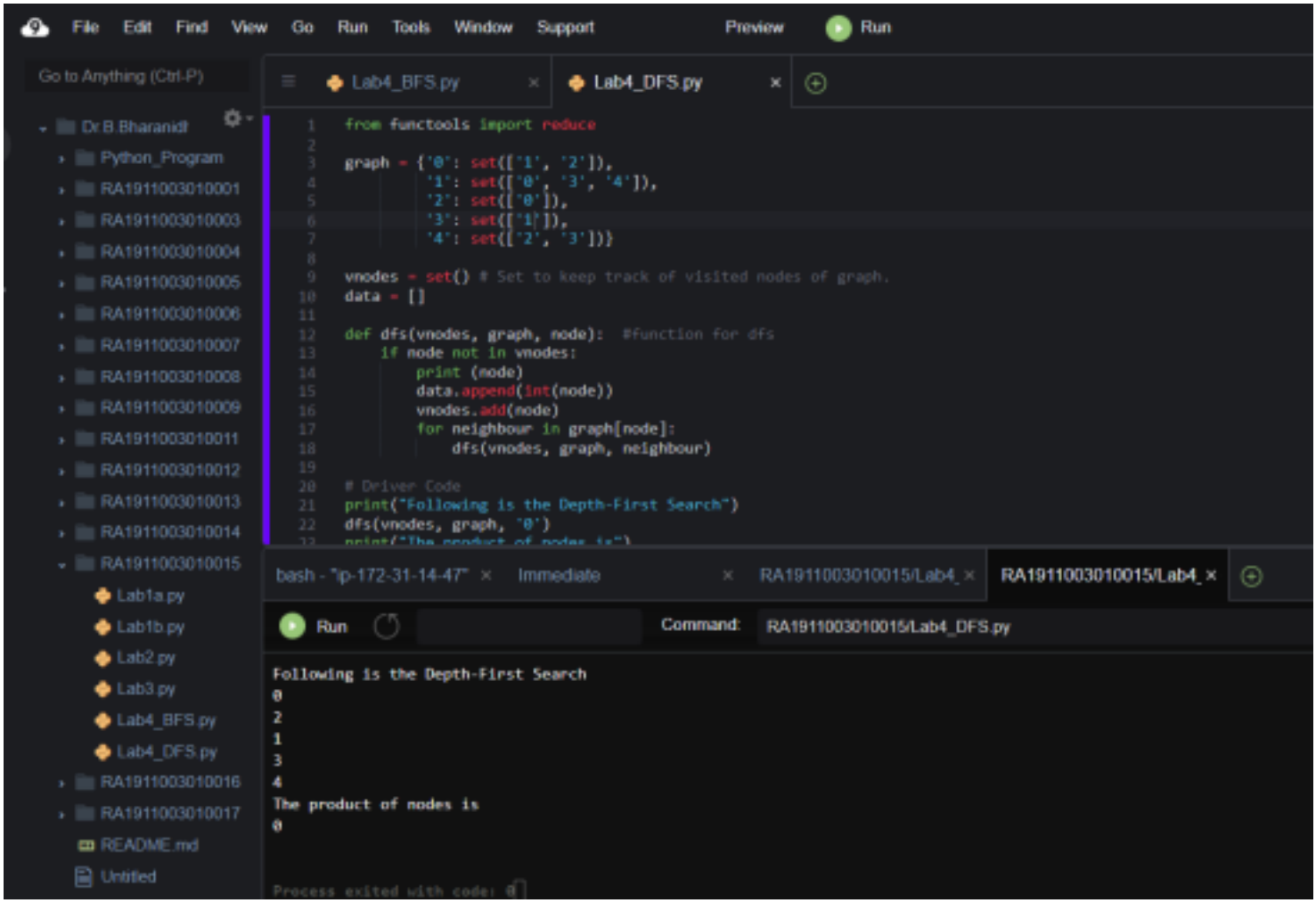
Introduction:
Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.
A standard DFS implementation puts each vertex of the graph into one of two categories:
Visited
Not Visited

Algorithm:
1. We will create the graph for which we will use the depth-first search.
2. After creation, we will create a set for storing the value of the visited nodes to keep track of the visited nodes of the graph.
3. After the above process, we will declare a function with the parameters as visited nodes, the graph itself and the node respectively.
4. And inside the function, we will check whether any node of the graph is visited or not using the “if” condition. If not, then we will print the node and add it to the visited set of nodes and add it to the other list we have created.
5. Then we will go to the neighboring node of the graph and again call the DFS function to use the neighbor parameter.
6. At last, we will run the driver code which prints the final result of DFS by calling the DFS the first time with the starting vertex of the graph and implement a lambda function.

Complexity:
Time Complexity: O(V+E) where V is vertices and E is edges
Space Complexity: O (h) where h is the maximum height of the tree

Code:
```
from functools import reduce

graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}
vnodes = set() # Set to keep track of visited nodes of graph.
data = []

def dfs(vnodes, graph, node): #function for dfs
    if node not in vnodes:
        print (node)
        data.append(int(node))
        vnodes.add(node)

        for neighbour in graph[node]:
            dfs(vnodes, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(vnodes, graph, '0')
print("The product of nodes is")
print(reduce((lambda x, y: x * y), data))
```

Program Output:



Result:

Both the algorithms were successfully executed.

Name: Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Lab 5 - Best First search and A* Algorithm

Aim:
To write a program to implement Best First Search.

Requirements:
GCC Compiler

Introduction:

1. Best first search is a traversal technique that decides which node is to be visited next by checking which node is the most promising one and then checking it. For this it uses an evaluation function to decide the traversal.
2. This best first search technique of tree traversal comes under the category of heuristic search or informed search technique.

Algorithm:

1. Take an input of the maze in binary format.
2. Taking the starting point, find all adjacent paths that can be taken.
3. Keep traversing through the array while taking the adjacent cells closest to the destination while avoiding cells with value 0.
4. If the final point is reached, save the length of the path.
5. Compare lengths of all paths that reach the destination and print the length of the shortest path.

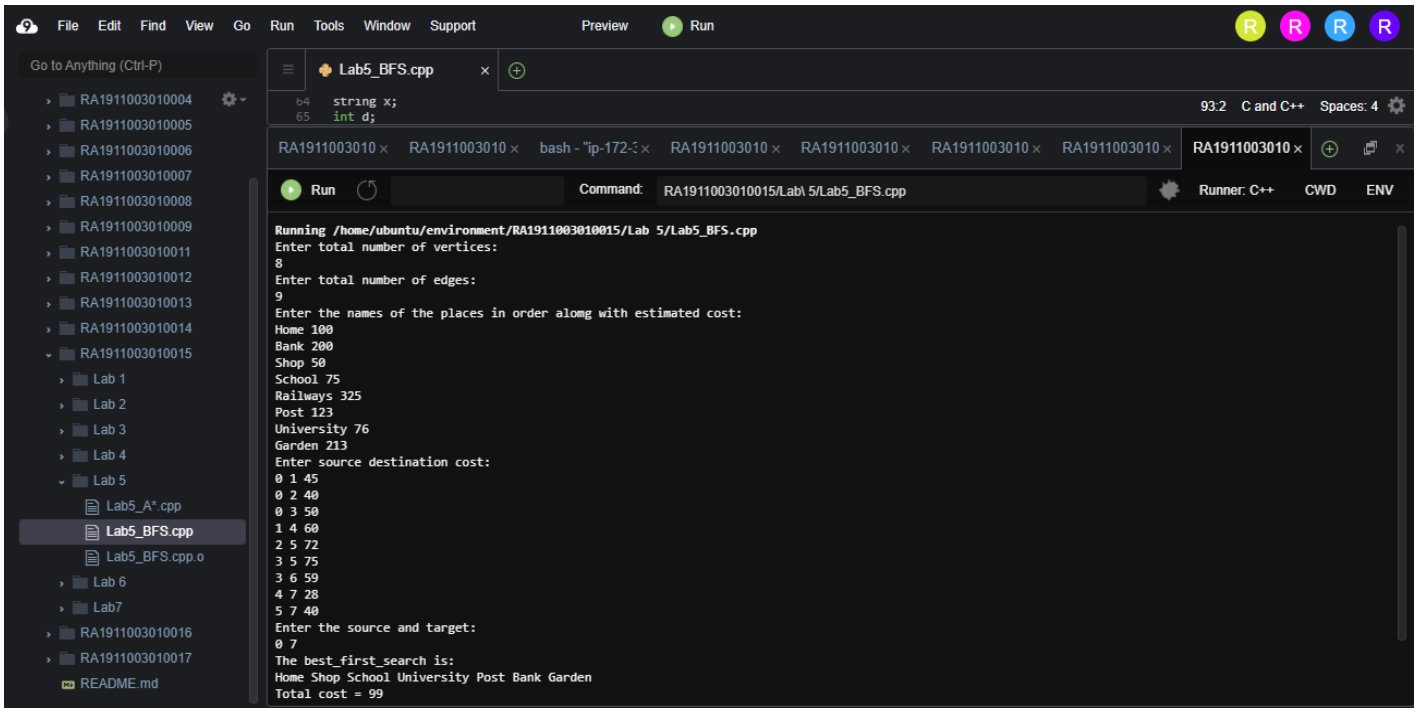
Code:

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;
vector<vector<pi>> graph;
vector<vector<pi>> g;
vector<vector<pi>> realcost;

vector<string> places;
vector<int> estimated;
vector<int> realtill;

void best_first_search(int source, int target, int n)
{
    vector<bool> visited(n, false);
    priority_queue<pi, vector<pi>, greater<pi> > pq;
    int total = 0;

    pq.push(make_pair(0, source));

    int s = source;
    visited[s] = true;
    int prev = -1;
    while (!pq.empty())
```

```
    {
        int x = pq.top().second;
        cout << places[x] << " ";

            if(prev != -1)
            {
                for (int i = 0; i < realcost[x].size(); i++)
                {
                        if(realcost[x][i].second == prev)
                        {
                            total += realcost[x][i].first;
                        }
                }
            }

            prev = x;
        pq.pop();

        if (x == target)
            break;

        for (int i = 0; i < g[x].size(); i++)
        {
            if (!visited[g[x][i].second])
            {
                visited[g[x][i].second] = true;
                pq.push(make_pair(g[x][i].first,g[x][i].second));
            }
        }
    }
    cout << "\nTotal cost = " << total << "\n";
}

int main()
{

    cout << "Enter total number of vertices: \n";
    int v;
    cin >> v;
    graph.resize(v);
    g.resize(v);
    realcost.resize(v);
    realtill.resize(v, 0);
    cout << "Enter total number of edges: \n";
    int e;
    cin >> e;
    cout << "Enter the names of the places in order alomg with estimated cost: \n";
    for(int i=0; i<v; i++)
    {
        string x;
        int d;
        cin >> x >> d;
        places.push_back(x);
        estimated.push_back(d);
    }
    realtill.push_back(0);
    cout << "Enter source destination cost:\n";
```

```
    for(int i=0; i<e; i++)
    {
        int x,y,cost;
        cin >> x >> y >> cost;
            int es = estimated[y];
            realtill[y] = realtill[x]+cost;
        graph[x].push_back(make_pair(realtill[y]+es, y));
        graph[y].push_back(make_pair(realtill[y]+es, x));
            g[x].push_back(make_pair(es, y));
        g[y].push_back(make_pair(es, x));
            realcost[x].push_back(make_pair(cost, y));
        realcost[y].push_back(make_pair(cost, x));

    }
    cout << "Enter the source and target: \n";
    int source, target;
    cin >> source >> target;
    cout << "The best_first_search is: \n";
    best_first_search(source, target, v);
    cout << "\n";
    return 0;
}
```

Program Output:



Aim:

To write a program to implement A* Search

Introduction:

It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete

Algorithm finds all the possible outcomes of a problem.

Algorithm:

1. Initialize the open list
2. Initialize the closed list, put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty
    4. Find the node with the least f on the open list, call it ‘q’
    5. Pop q off the open list
    6. generate q's 8 successors and set their parents to q
        7. for each successor
        8. if successor is the goal, stop search successor.g = q.g + distance between successor and q successor.h = distance from goal to successor successor.f = successor.g + successor.h
        9. if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor
        10. if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list
    11. Push q on the closed list

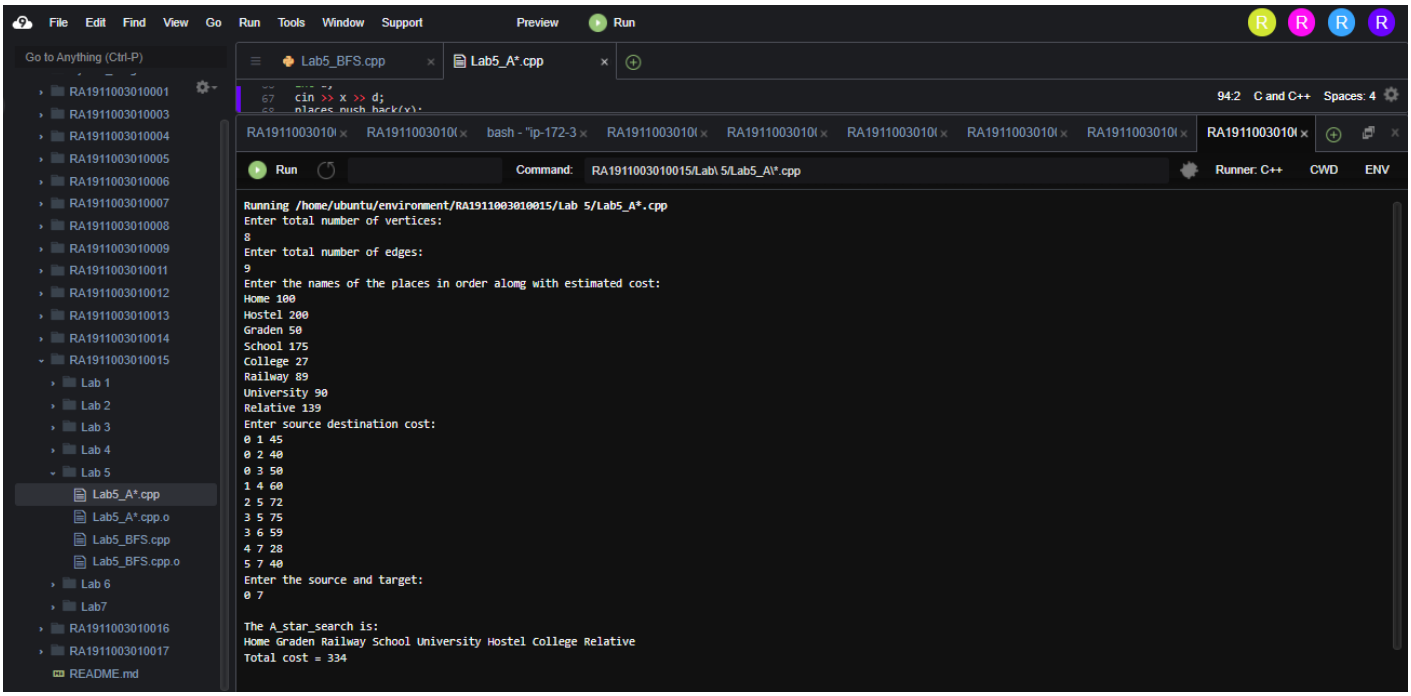
Code:

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;
vector<vector<pi>> graph;
vector<vector<pi>> g;
vector<vector<pi>> realcost;

vector<string> places;
vector<int> estimated;
vector<int> realtill;

void a_star_search(int source, int target, int n)
{
   vector<bool> visited(n, false);
   priority_queue<pi, vector<pi>, greater<pi> > pq;

   pq.push(make_pair(0, source));
   int s = source;
   visited[s] = true;

   int total=0;
   int prev = -1;

   while (!pq.empty())
   {
      int x = pq.top().second;
      cout << places[x] << " ";

         if(prev != -1)
         {
            for (int i = 0; i < realcost[x].size(); i++)
            {
                  if(realcost[x][i].second == prev)
                  {
```

```
                    total += realcost[x][i].first;
                    break;
                }
            }
        }

        prev = x;

    pq.pop();

    if (x == target)
        break;

    for (int i = 0; i < graph[x].size(); i++)
    {
        if (!visited[graph[x][i].second])
        {
            visited[graph[x][i].second] = true;
            pq.push(make_pair(graph[x][i].first,graph[x][i].second));
        }
    }
  }
  cout << "\nTotal cost = " << total << "\n";
}
int main()
{
  cout << "Enter total number of vertices: \n";
  int v;
  cin >> v;
  graph.resize(v);
  g.resize(v);
  realcost.resize(v);
  realtill.resize(v, 0);
  cout << "Enter total number of edges: \n";
  int e;
  cin >> e;
  cout << "Enter the names of the places in order alomg with estimated cost: \n";
  for(int i=0; i<v; i++)
  {
    string x;
    int d;
    cin >> x >> d;
    places.push_back(x);
    estimated.push_back(d);
  }
  realtill.push_back(0);
  cout << "Enter source destination cost:\n";
  for(int i=0; i<e; i++)
  {
    int x,y,cost;
    cin >> x >> y >> cost;
      int es = estimated[y];
      realtill[y] = realtill[x]+cost;
    graph[x].push_back(make_pair(realtill[y]+es, y));
    graph[y].push_back(make_pair(realtill[y]+es, x));
      g[x].push_back(make_pair(es, y));
    g[y].push_back(make_pair(es, x));
```

```
            realcost[x].push_back(make_pair(cost, y));
        realcost[y].push_back(make_pair(cost, x));

    }
    cout << "Enter the source and target: \n";
    int source, target;
    cin >> source >> target;

    cout << "\nThe A_star_search is: \n";
    a_star_search(source, target, v);

    cout << "\n";
    return 0;
}
```

Program Output:



Result:

Both the algorithms were successfully executed.

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Lab 6 : Implementation of Uncertain Methods for an Application (Fuzzy logic)

Aim:
To implement uncertain methods for an application (to calculate membership of certain values based on user's input and display the same) using Fuzzy logic

Requirements:
GCC Compiler

Introduction:

1. Fuzzy logic is an approach to variable processing that allows for multiple possible truth values to be processed through the same variable. Fuzzy logic attempts to solve problems with an open, imprecise spectrum of data and heuristics that makes it possible to obtain an array of accurate conclusions.

2. The term fuzzy refers to things that are not clear or are vague. In the real world many times we encounter a situation when we can't determine whether the state is true or false, their fuzzy logic provides very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation.

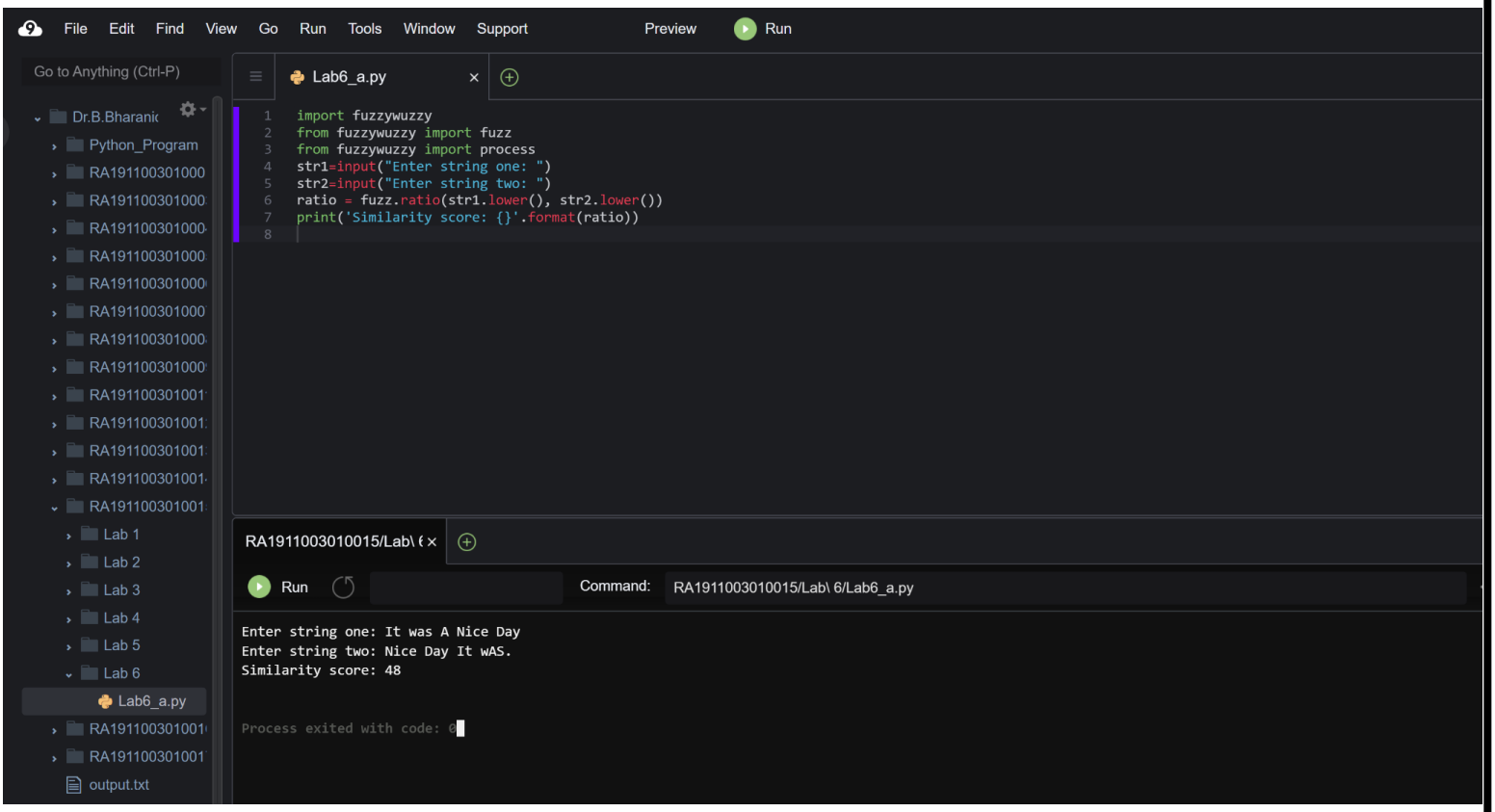
Algorithm:

1. Define Non Fuzzy Inputs with Fuzzy Sets. The non-fuzzy inputs are numbers from a certain range, and find how to represent those non-fuzzy values with fuzzy sets.
2. Locate the input, output, and state variables of the plane under consideration.
3. Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.
4. Obtain the membership function for each fuzzy subset.
5. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and output of fuzzy subsets on the other side, thereby forming the rule base.
6. Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0,1] and [-1, I] intervals.
7. Carry out the fuzzification process.
8. Identify the output contributed from each rule using fuzzy approximate reasoning.
9. Combine the fuzzy outputs obtained from each rule.
10. Finally, apply defuzzification to form a crisp output.

Code:

```
import fuzzywuzzy
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
str1=input("Enter string one: ")
str2=input("Enter string two: ")
ratio = fuzz.ratio(str1.lower(), str2.lower())
print('Similarity score: {}'.format(ratio))
```

Program Output:



Optimization Technique:

Various numerical optimization techniques can be used such as dynamic programming, Lagrangian relaxation method , mixed integer programming ,and branch-and-bound method . The dynamic programming method is simple but the calculation time required to converge to the optimal solution is quite long. Regarding the branch-and-bound method, it adopts a linear function to represent the fuel and start-up costs during a time horizon. The mixed integer programming uses linear programming to attain optimal solutions. Nevertheless, this method was applied to small problems of unit commitment and they required major assumptions that limit the margin of solutions. For the Lagrangian relaxation method, we note that the convergence time is an advantage, but the obtained solution is not ideal because of the complexity of the problem especially when the optimization problem contains a great number of production units.

Result:

Implementation of uncertain methods for an application (to calculate membership of certain values) is successfully implemented

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Lab 7 - Implementation of Unification and Resolution for Real World Problems

Aim:
Develop a program to unify expressions and direct the output of resolution to output.txt after taking input from input.txt file in same directory

Requirements:
Python 3

Introduction:

1. Unification:
   - Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
   - It takes two literals as input and makes them identical using substitution.
   - Let Ψ1 and Ψ2 be two atomic sentences and □ be a unifier such that, Ψ1□ = Ψ2□, then it can be expressed as UNIFY(Ψ1, Ψ2).
2. Resolution:
   - Resolution is used, if various statements are given, and we need to prove a conclusion of those statements.
   - Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

Algorithm of Unification:

1. If Ψ1 or Ψ2 is a variable or constant, then:
   a) If Ψ1 or Ψ2 are identical, then return NIL.
   b) Else if Ψ1is a variable,
      a. then if Ψ1 occurs in Ψ2, then return FAILURE
      b. Else return { (Ψ2/ Ψ1)}.
   c) Else if Ψ2 is a variable,
      a. If Ψ2 occurs in Ψ1 then return FAILURE,
      b. Else return {(Ψ1/ Ψ2)}.
   d) Else return FAILURE.
2. If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then return FAILURE.
3. IF Ψ1 and Ψ2 have a different number of arguments, then return FAILURE.
4. Set Substitution set(SUBST) to NIL.
5. For i=1 to the number of elements in Ψ1.
   a) Call Unify function with the ith element of Ψ1 and ith element of Ψ2, and put the result into S.
   b) If S = failure then returns Failure
   c) If S ≠ NIL then do,
      a. Apply S to the remainder of both L1 and L2.
      b. SUBST= APPEND(S, SUBST).
6. Return SUBST.

Algorithm of Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Code of Unification:

```
def get_index_comma(string):
    index_list = list()
    par_count = 0

    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
            index_list.append(i)
        elif string[i] == '(':
            par_count += 1
        elif string[i] == ')':
            par_count -= 1

    return index_list


def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False

    return True


def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')
    expr = expr[1:len(expr) - 1]
    arg_list = list()
    indices = get_index_comma(expr)

    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
        for i, j in zip(indices, indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

    return predicate_symbol, arg_list


def get_arg_list(expr):
    _, arg_list = process_expression(expr)
```

```
    flag = True
    while flag:
        flag = False

        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
                        arg_list.append(j)
                arg_list.remove(i)

    return arg_list


def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False


def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
            return False
        else:
            tmp = str(expr2) + '/' + str(expr1)
            return tmp
    elif not is_variable(expr1) and is_variable(expr2):
        if check_occurs(expr2, expr1):
            return False
        else:
            tmp = str(expr1) + '/' + str(expr2)
            return tmp
    else:
        predicate_symbol_1, arg_list_1 = process_expression(expr1)
        predicate_symbol_2, arg_list_2 = process_expression(expr2)

        # Step 2
        if predicate_symbol_1 != predicate_symbol_2:
            return False
        # Step 3
        elif len(arg_list_1) != len(arg_list_2):
            return False
        else:
            # Step 4: Create substitution list
            sub_list = list()

            # Step 5:
```

```
            for i in range(len(arg_list_1)):
                tmp = unify(arg_list_1[i], arg_list_2[i])

                if not tmp:
                    return False
                elif tmp == 'Null':
                    pass
                else:
                    if type(tmp) == list:
                        for j in tmp:
                            sub_list.append(j)
                    else:
                        sub_list.append(tmp)

            # Step 6
            return sub_list


if __name__ == '__main__':

    f1 = 'Q(a, g(x, a), f(y))'
    f2 = 'Q(a, g(f(b), a), x)'
    # f1 = input('f1 : ')
    # f2 = input('f2 : ')

    result = unify(f1, f2)
    if not result:
        print('The process of Unification failed!')
    else:
        print('The process of Unification successful!')
        print(result)
```

Code of Resolution:

```
import copy
import time


class Parameter:
    variable_count = 1

    def __init__(self, name=None):
        if name:
            self.type = "Constant"
            self.name = name
        else:
            self.type = "Variable"
            self.name = "v" + str(Parameter.variable_count)
            Parameter.variable_count += 1

    def isConstant(self):
        return self.type == "Constant"

    def unify(self, type_, name):
        self.type = type_
        self.name = name

    def __eq__(self, other):
        return self.name == other.name
```

```
    def __str__(self):
        return self.name


class Predicate:
    def __init__(self, name, params):
        self.name = name
        self.params = params

    def __eq__(self, other):
        return self.name == other.name and all(a == b for a, b in zip(self.params, other.params))

    def __str__(self):
        return self.name + "(" + ",".join(str(x) for x in self.params) + ")"

    def getNegatedPredicate(self):
        return Predicate(negatePredicate(self.name), self.params)


class Sentence:
    sentence_count = 0

    def __init__(self, string):
        self.sentence_index = Sentence.sentence_count
        Sentence.sentence_count += 1
        self.predicates = []
        self.variable_map = {}
        local = {}

        for predicate in string.split("|"):
            name = predicate[:predicate.find("(")]
            params = []

            for param in predicate[predicate.find("(") + 1: predicate.find(")")].split(","):
                if param[0].islower():
                    if param not in local:  # Variable
                        local[param] = Parameter()
                        self.variable_map[local[param].name] = local[param]
                    new_param = local[param]
                else:
                    new_param = Parameter(param)
                    self.variable_map[param] = new_param

                params.append(new_param)

            self.predicates.append(Predicate(name, params))

    def getPredicates(self):
        return [predicate.name for predicate in self.predicates]

    def findPredicates(self, name):
        return [predicate for predicate in self.predicates if predicate.name == name]

    def removePredicate(self, predicate):
        self.predicates.remove(predicate)
        for key, val in self.variable_map.items():
            if not val:
                self.variable_map.pop(key)
```

```
    def containsVariable(self):
        return any(not param.isConstant() for param in self.variable_map.values())

    def __eq__(self, other):
        if len(self.predicates) == 1 and self.predicates[0] == other:
            return True
        return False

    def __str__(self):
        return "".join([str(predicate) for predicate in self.predicates])


class KB:
    def __init__(self, inputSentences):
        self.inputSentences = [x.replace(" ", "") for x in inputSentences]
        self.sentences = []
        self.sentence_map = {}

    def prepareKB(self):
        self.convertSentencesToCNF()
        for sentence_string in self.inputSentences:
            sentence = Sentence(sentence_string)
            for predicate in sentence.getPredicates():
                self.sentence_map[predicate] = self.sentence_map.get(
                    predicate, []) + [sentence]

    def convertSentencesToCNF(self):
        for sentenceIdx in range(len(self.inputSentences)):
            # Do negation of the Premise and add them as literal
            if "=>" in self.inputSentences[sentenceIdx]:
                self.inputSentences[sentenceIdx] = negateAntecedent(
                    self.inputSentences[sentenceIdx])

    def askQueries(self, queryList):
        results = []

        for query in queryList:
            negatedQuery = Sentence(negatePredicate(query.replace(" ", "")))
            negatedPredicate = negatedQuery.predicates[0]
            prev_sentence_map = copy.deepcopy(self.sentence_map)
            self.sentence_map[negatedPredicate.name] = self.sentence_map.get(
                negatedPredicate.name, []) + [negatedQuery]
            self.timeLimit = time.time() + 40

            try:
                result = self.resolve([negatedPredicate], [
                                      False]*(len(self.inputSentences) + 1))
            except:
                result = False

            self.sentence_map = prev_sentence_map

            if result:
                results.append("TRUE")
            else:
                results.append("FALSE")

        return results
```

```
    def resolve(self, queryStack, visited, depth=0):
        if time.time() > self.timeLimit:
            raise Exception
        if queryStack:
            query = queryStack.pop(-1)
            negatedQuery = query.getNegatedPredicate()
            queryPredicateName = negatedQuery.name
            if queryPredicateName not in self.sentence_map:
                return False
            else:
                queryPredicate = negatedQuery
                for kb_sentence in self.sentence_map[queryPredicateName]:
                    if not visited[kb_sentence.sentence_index]:
                        for kbPredicate in kb_sentence.findPredicates(queryPredicateName):

                            canUnify, substitution = performUnification(
                                copy.deepcopy(queryPredicate), copy.deepcopy(kbPredicate))

                            if canUnify:
                                newSentence = copy.deepcopy(kb_sentence)
                                newSentence.removePredicate(kbPredicate)
                                newQueryStack = copy.deepcopy(queryStack)

                                if substitution:
                                    for old, new in substitution.items():
                                        if old in newSentence.variable_map:
                                            parameter = newSentence.variable_map[old]
                                            newSentence.variable_map.pop(old)
                                            parameter.unify(
                                                "Variable" if new[0].islower() else "Constant", new)
                                            newSentence.variable_map[new] = parameter

                                    for predicate in newQueryStack:
                                        for index, param in enumerate(predicate.params):
                                            if param.name in substitution:
                                                new = substitution[param.name]
                                                predicate.params[index].unify(
                                                    "Variable" if new[0].islower() else "Constant", new)

                                for predicate in newSentence.predicates:
                                    newQueryStack.append(predicate)

                                new_visited = copy.deepcopy(visited)
                                if kb_sentence.containsVariable() and len(kb_sentence.predicates) > 1:
                                    new_visited[kb_sentence.sentence_index] = True

                                if self.resolve(newQueryStack, new_visited, depth + 1):
                                    return True
                    return False
        return True


def performUnification(queryPredicate, kbPredicate):
    substitution = {}
    if queryPredicate == kbPredicate:
        return True, {}
    else:
        for query, kb in zip(queryPredicate.params, kbPredicate.params):
```

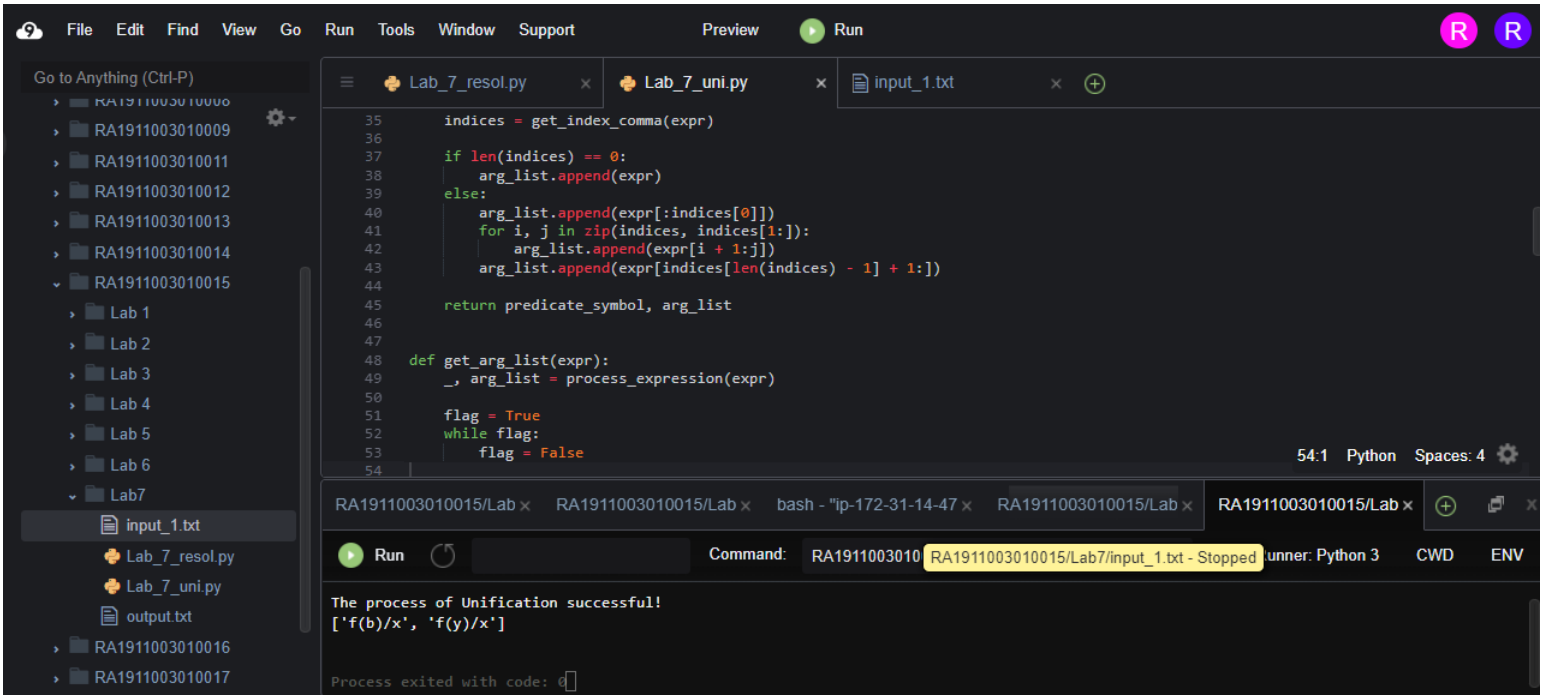
```
            if query == kb:
                continue
            if kb.isConstant():
                if not query.isConstant():
                    if query.name not in substitution:
                        substitution[query.name] = kb.name
                    elif substitution[query.name] != kb.name:
                        return False, {}
                    query.unify("Constant", kb.name)
                else:
                    return False, {}
            else:
                if not query.isConstant():
                    if kb.name not in substitution:
                        substitution[kb.name] = query.name
                    elif substitution[kb.name] != query.name:
                        return False, {}
                    kb.unify("Variable", query.name)
                else:
                    if kb.name not in substitution:
                        substitution[kb.name] = query.name
                    elif substitution[kb.name] != query.name:
                        return False, {}
    return True, substitution


def negatePredicate(predicate):
    return predicate[1:] if predicate[0] == "~" else "~" + predicate


def negateAntecedent(sentence):
    antecedent = sentence[:sentence.find("=>")]
    premise = []

    for predicate in antecedent.split("&"):
        premise.append(negatePredicate(predicate))

    premise.append(sentence[sentence.find("=>") + 2:])
    return "|".join(premise)


def getInput(filename):
    with open(filename, "r") as file:
        noOfQueries = int(file.readline().strip())
        inputQueries = [file.readline().strip() for _ in range(noOfQueries)]
        noOfSentences = int(file.readline().strip())
        inputSentences = [file.readline().strip()
                          for _ in range(noOfSentences)]
        return inputQueries, inputSentences


def printOutput(filename, results):
    print(results)
    with open(filename, "w") as file:
        for line in results:
            file.write(line)
            file.write("\n")
    file.close()
```

```
if __name__ == '__main__':
    inputQueries_, inputSentences_ = getInput('input_1.txt')
    knowledgeBase = KB(inputSentences_)
    knowledgeBase.prepareKB()
    results_ = knowledgeBase.askQueries(inputQueries_)
    printOutput("output.txt", results_)
```

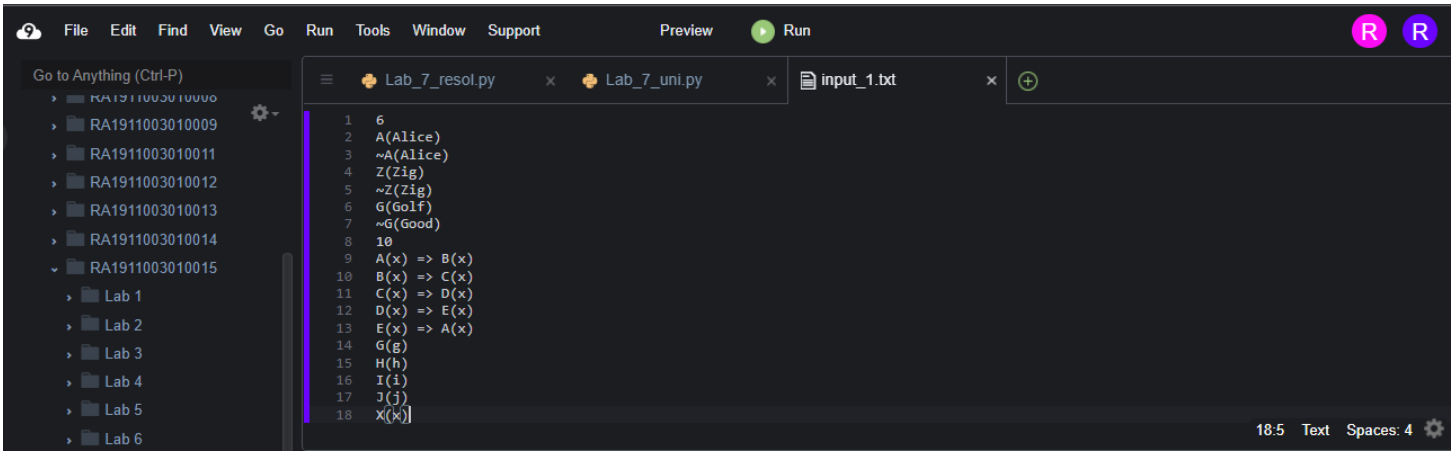
Input file of Resolution:

```
6
A(Alice)
~A(Alice)
Z(Zig)
~Z(Zig)
G(Golf)
~G(Good)
10
A(x) => B(x)
B(x) => C(x)
C(x) => D(x)
D(x) => E(x)
E(x) => A(x)
G(g)
H(h)
I(i)
J(j)
X(x)
```

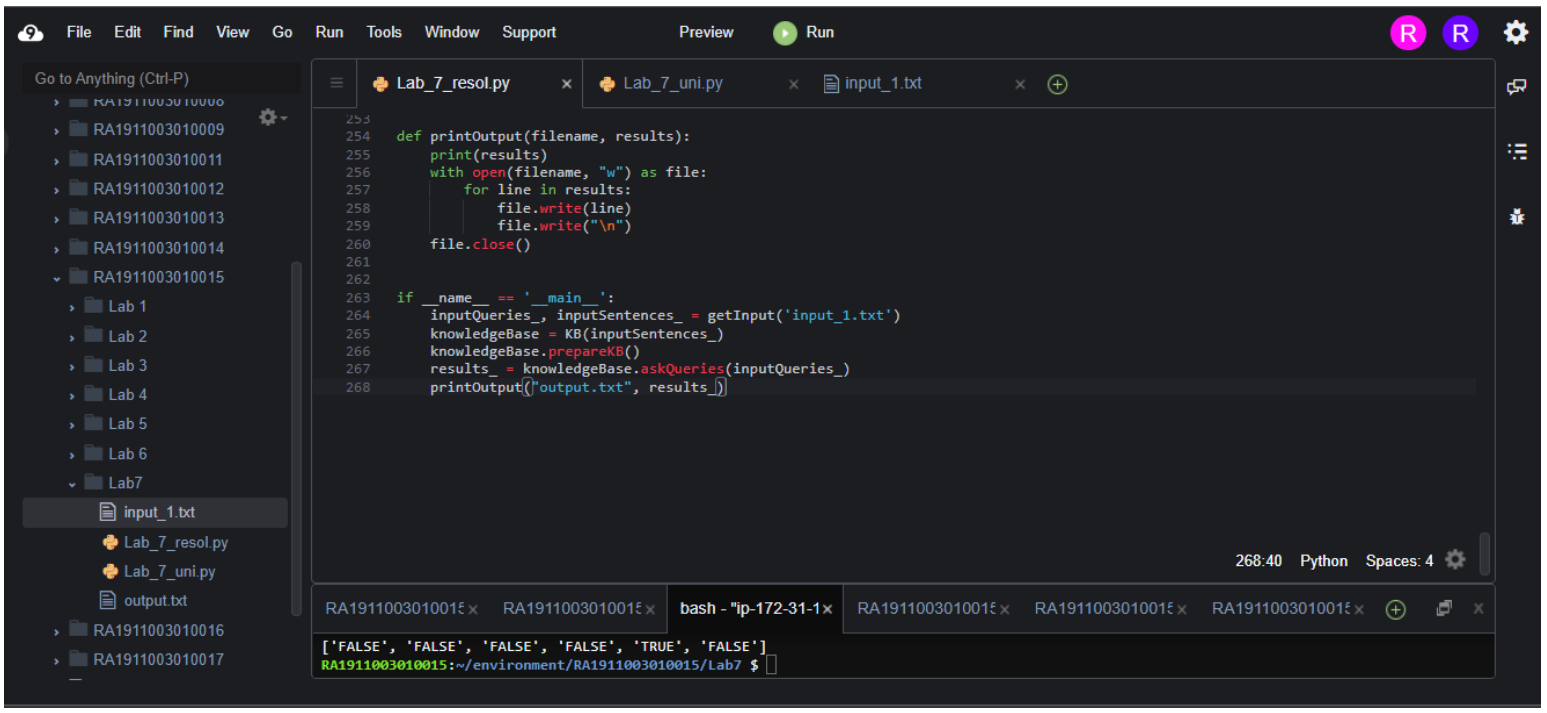
Program Output of Unification:



Input of Resolution:



Output of Resolution:



Result:

Unification of expression was done and the conversion set was printed and the result of all queries in input file were printed and written to output.txt.

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

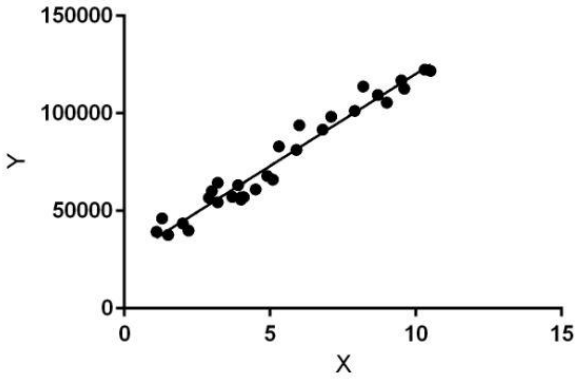
Ex no:8 Implementation of Learning Algorithms for an Application

Aim:
a) Implementation of Linear Regression algorithm to predict students' score using the given dataset.
b) Implementation of Support Vector Classification algorithm to classify the cases of breast cancer using the given dataset.
c) Implementation of K-means clustering algorithm to group the customers based on their demographic detail using the given dataset.

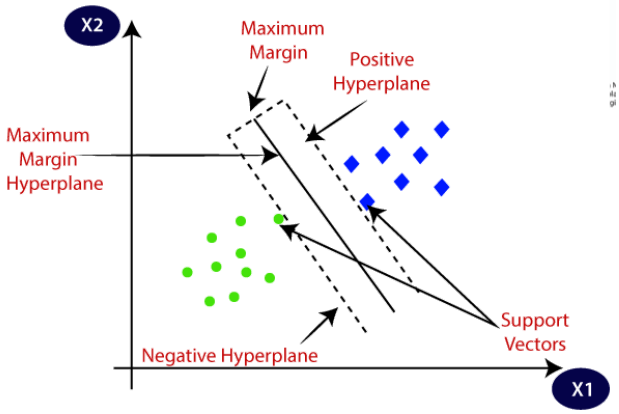
Requirements:
Google Colab

Introduction:

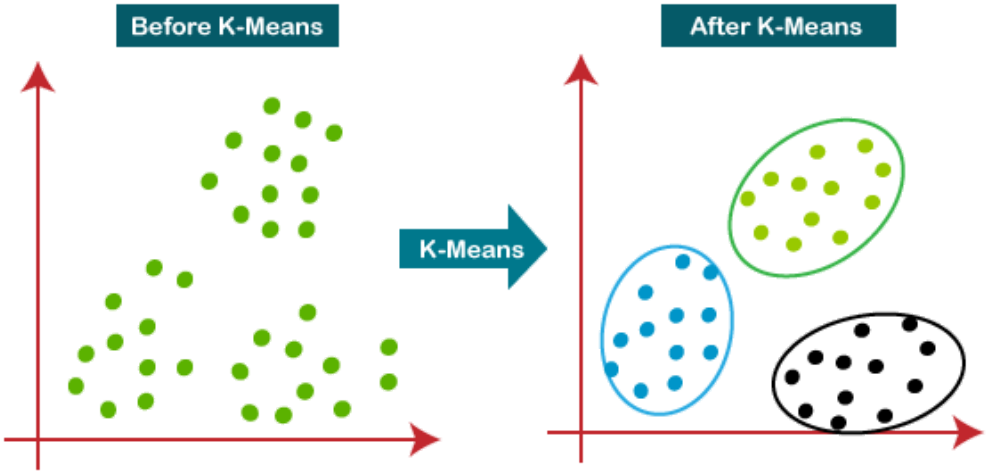
1. Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



2. Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
   The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.



3. K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.



Code: Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
Import required modules and packages
dataset = pd.read_csv('….\student_scores.csv')
dataset.head()
Import data set
dataset.describe()
dataset.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
Visualize the data.
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
Identify the independent (X) and dependent variables (y) in the data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
print('X train shape: ', X_train.shape)
print('Y train shape: ', Y_train.shape)
print('X test shape: ', X_test.shape)
Splitting the given data in to training set (80%) and testing set (20%)
print('Y test shape: ', Y_test.shape)
regressor = LinearRegression()
Model instantiation
regressor.fit(X_train, y_train) Model Training
print(regressor.intercept_)
print(regressor.coef_)
Finding out the coefficient (a) and intercept (b) value of linear model
(y=aX+b)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

```
print(df)
Testing the model
print('Mean Absolute Error:',
metrics.mean_absolute_error (y_test, y_pred))
print('Mean Squared Error:',
metrics.mean_squared_error (y_test, y_pred))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error (y_test, y_pred)))
MAE, MSE, RMSE – Evaluation metrics of Model
```

Program Input and Output:

```
[ ] dataset = pd.read_csv('C:\\Users\DELL\Desktop\student_scores.csv')
    dataset.shape()
```

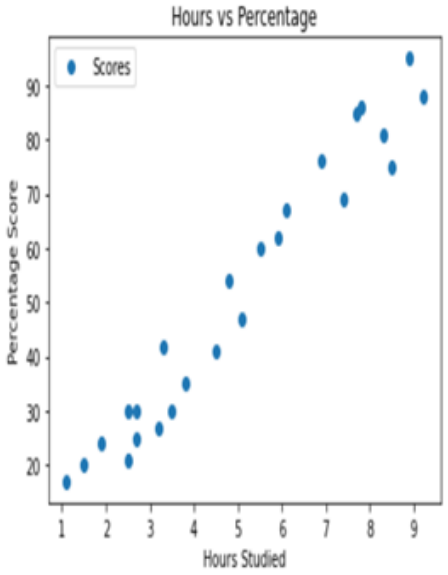
	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
dataset.shape
```

```
(25, 2)
```

```
dataset.describe()
```

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000



```
[ ] print(regressor.coef_)

    [9.91065648]

[ ] y_pred = regressor.predict(X_test)
    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    df
```

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

```
[ ] print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
    print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
    print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

    Mean Absolute Error: 4.183859899002975
    Mean Squared Error: 21.5987693072174
    Root Mean Squared Error: 4.6474476121003665
```

Code: Support Vector Classification

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,
classification_report
Import required modules and packages
dataset = pd.read_csv('….\diabetes data.csv')
print(dataset.head())
Import data set
Choose the right path for the dataset
def diagnosis(x):
if x=='M' :
return 1
if x=='B' :
return 0
dataset['diagnosis'] = dataset['diagnosis'].apply(diagnosis)
print(dataset)
Data cleaning process. Converting categorical value in to numerical value.
M = malignant, B = benign
print("Any missing sample in data set:",
dataset.isnull().values.any(), "\n")
Check for any missing values in the data set
dataset = dataset.replace([np.inf, -np.inf], np.nan)
dataset= dataset.fillna(dataset.mean())
dataset
Replace the missing value with its mean value of the respective attribute
dataset= dataset.drop(columns=["Unnamed: 32"]) drop this column because it's not
necessary (null)
Y = dataset['diagnosis']
X = dataset.drop(columns=['diagnosis'])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=9)
print('X train shape: ', X_train.shape)
print('Y train shape: ', Y_train.shape)
print('X test shape: ', X_test.shape)
```

```
print('Y test shape: ', Y_test.shape)
Splitting the given data in to training set
(80%) and testing set (20%)
svc_classifier= SVC(kernel='poly')
Model instantiation. Apply SVM with different kernels 'linear', 'poly', 'rbf', 'sigmoid' and verify the accuracy of the
model
svc_classifier.fit(X_train,Y_train)
Model Training
y_pred=svc_classifier.predict(X_test)
Testing the model
print(confusion_matrix(Y_test,y_pred))
print(classification_report(Y_test,y_pred))
Evaluation metrics to measure the performance of the model
```

Program Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
%matplotlib inline
```

```
dataset = pd.read_csv('C:\\Users\DELL\Desktop\data.csv')
dataset.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	

5 rows × 33 columns

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimete
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17.33	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23.41	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25.53	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26.50	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.40	
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.25	
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.12	
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.42	
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37	

569 rows × 33 columns

```
svc_classifier= SVC(kernel='rbf')
svc_classifier
```

```
SVC()
```

```
svc_classifier=svc_classifier.fit(X_train,Y_train)
```

```
y_pred=svc_classifier.predict(X_test)
```

```
print(confusion_matrix(Y_test,y_pred))
```

```
[[74  0]
 [40  0]]
```

```
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.65      1.00      0.79        74
           1       0.00      0.00      0.00        40

    accuracy                           0.65       114
   macro avg       0.32      0.50      0.39       114
weighted avg       0.42      0.65      0.51       114
```

Code: K-means clustering

Problem: Client is owning a supermarket mall and through membership cards, client have some basic data about your customers like Customer ID, age, gender, annual income and spending score. Help the client to understand the customers like who are the target customers so that the sense can be given to marketing

Program Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
%matplotlib inline
```

```
data=pd.read_csv('C:\\Users\DELL\Desktop\mall_customers.csv')
print(data.head())
```

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

```
inVsout=data.iloc[:,[3,4]]
inVsout
```

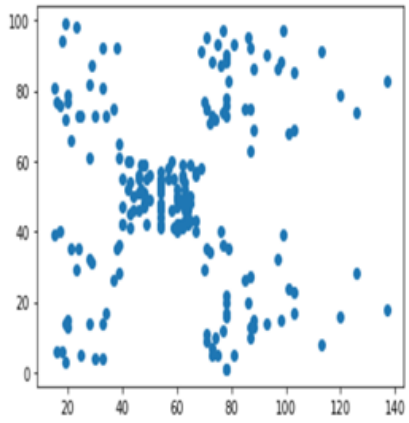
```
inVsout=data.iloc[:,[3,4]]
inVsout
```

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1])
```

```
<matplotlib.collections.PathCollection at 0x1fa26e7ca90>
```

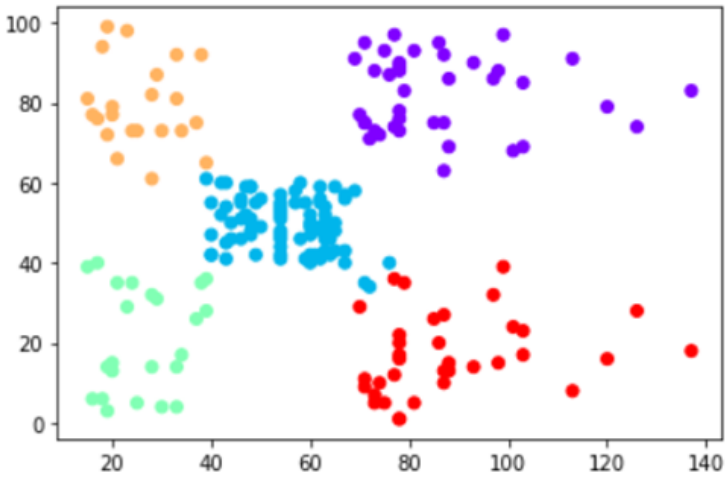



```
kmeans=KMeans(n_clusters=5)
kmeans.fit(inVsout)
```

```
KMeans(n_clusters=5)
```

```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1], c=kmeans.labels_, cmap='rainbow')
plt.show()
```

```
plt.scatter(inVsout.iloc[:,0],inVsout.iloc[:,1], c=kmeans.labels_, cmap='rainbow')
plt.show()
```



	Annual Income (k$)	Spending Score (1-100)
0	15	39
2	16	6
4	17	40
6	18	6
8	19	3
10	19	14
12	20	15
14	20	13
16	21	35
18	23	29
20	24	35
22	25	5
24	28	14
26	28	32
28	29	31
30	30	4
32	33	4
34	33	14

```
silhouette_score(inVsout,kmeans.labels_)
```

```
0.553931997444648
```

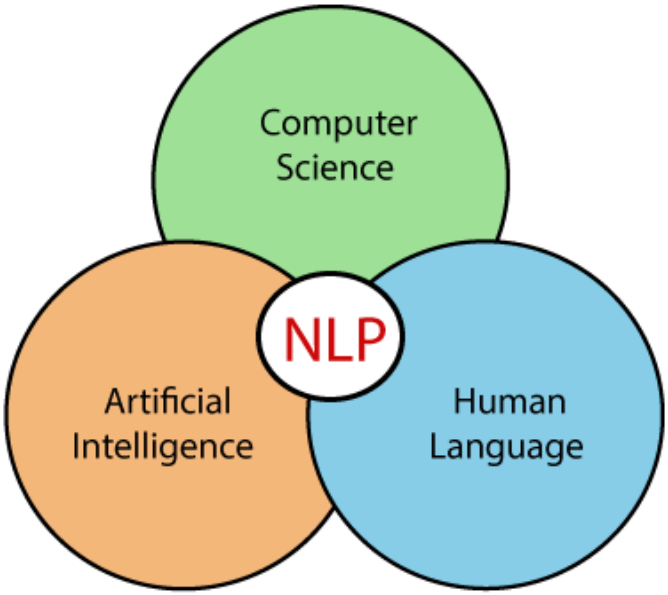
Result:
All three were successfully executed.

Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Ex no:9 Implementation of Natural Language Processing

Aim:
To Implement natural language processing programs
Requirements:
Google Colab

Introduction:
Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models.
Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models.



Code:

```
!pip install -q wordcloud
import wordcloud

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

import pandas as pd
import matplotlib.pyplot as plt
```

```
import io
import unicodedata
import numpy as np
import re
import string
# Constants
# POS (Parts Of Speech) for: nouns, adjectives, verbs and adverbs
DI_POS_TYPES = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}
POS_TYPES = list(DI_POS_TYPES.keys())

# Constraints on tokens
MIN_STR_LEN = 3
RE_VALID = '[a-zA-Z]'
# Upload from google drive
from google.colab import files
uploaded = files.upload()
print("len(uploaded.keys():", len(uploaded.keys()))

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(name=fn, length=len(uploaded[fn])))

# Get list of quotes
df_quotes = pd.read_csv(io.StringIO(uploaded['quotes.txt'].decode('utf-8')), sep='\t')

# Display
print("df_quotes:")
print(df_quotes.head().to_string())
print(df_quotes.describe())

# Convert quotes to list
li_quotes = df_quotes['Quote'].tolist()
print()
print("len(li_quotes):", len(li_quotes)
# Get stopwords, stemmer and lemmatizer
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.PorterStemmer()
lemmatizer = nltk.stem.WordNetLemmatizer()

# Remove accents function
def remove_accents(data):
    return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in string.ascii_letters or x == " ")

# Process all quotes
li_tokens = []
li_token_lists = []
li_lem_strings = []

for i,text in enumerate(li_quotes):
    # Tokenize by sentence, then by lowercase word
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]

    # Process all tokens per quote
    li_tokens_quote = []
    li_tokens_quote_lem = []
    for token in tokens:
        # Remove accents
        t = remove_accents(token)

        # Remove punctuation
```

```
        t = str(t).translate(string.punctuation)
        li_tokens_quote.append(t)

        # Add token that represents "no lemmatization match"
        li_tokens_quote_lem.append("-") # this token will be removed if a lemmatization match is found below

        # Process each token
        if t not in stopwords:
            if re.search(RE_VALID, t):
                if len(t) >= MIN_STR_LEN:
                    # Note that the POS (Part Of Speech) is necessary as input to the lemmatizer
                    # (otherwise it assumes the word is a noun)
                    pos = nltk.pos_tag([t])[0][1][:2]
                    pos2 = 'n'  # set default to noun
                    if pos in DI_POS_TYPES:
                      pos2 = DI_POS_TYPES[pos]

                    stem = stemmer.stem(t)
                    lem = lemmatizer.lemmatize(t, pos=pos2)  # lemmatize with the correct POS

                    if pos in POS_TYPES:
                        li_tokens.append((t, stem, lem, pos))

                        # Remove the "-" token and append the lemmatization match
                        li_tokens_quote_lem = li_tokens_quote_lem[:-1]
                        li_tokens_quote_lem.append(lem)

    # Build list of token lists from lemmatized tokens
    li_token_lists.append(li_tokens_quote)

    # Build list of strings from lemmatized tokens
    str_li_tokens_quote_lem = ' '.join(li_tokens_quote_lem)
    li_lem_strings.append(str_li_tokens_quote_lem)

# Build resulting dataframes from lists
df_token_lists = pd.DataFrame(li_token_lists)

print("df_token_lists.head(5):")
print(df_token_lists.head(5).to_string())

# Replace None with empty string
for c in df_token_lists:
    if str(df_token_lists[c].dtype) in ('object', 'string_', 'unicode_'):
        df_token_lists[c].fillna(value='', inplace=True)

df_lem_strings = pd.DataFrame(li_lem_strings, columns=['lem quote'])

print()
print("")
print("df_lem_strings.head():")
print(df_lem_strings.head().to_string())
# Add counts
print("Group by lemmatized words, add count and sort:")
df_all_words = pd.DataFrame(li_tokens, columns=['token', 'stem', 'lem', 'pos'])
df_all_words['counts'] = df_all_words.groupby(['lem'])['lem'].transform('count')
df_all_words = df_all_words.sort_values(by=['counts', 'lem'], ascending=[False, True]).reset_index()

print("Get just the first row in each lemmatized group")
df_words = df_all_words.groupby('lem').first().sort_values(by='counts', ascending=False).reset_index()
```

```
print("df_words.head(10):")
print(df_words.head(10))
df_words = df_words[['lem', 'pos', 'counts']].head(200)
for v in POS_TYPES:
    df_pos = df_words[df_words['pos'] == v]
    print()
    print("POS_TYPE:", v)
    print(df_pos.head(10).to_string())
li_token_lists_flat = [y for x in li_token_lists for y in x]  # flatten the list of token lists to a single list
print("li_token_lists_flat[:10]:", li_token_lists_flat[:10])

di_freq = nltk.FreqDist(li_token_lists_flat)
del di_freq['']
li_freq_sorted = sorted(di_freq.items(), key=lambda x: x[1], reverse=True)  # sorted list
print(li_freq_sorted)

di_freq.plot(30, cumulative=False)
li_lem_words = df_all_words['lem'].tolist()
di_freq2 = nltk.FreqDist(li_lem_words)
li_freq_sorted2 = sorted(di_freq2.items(), key=lambda x: x[1], reverse=True)  # sorted list
print(li_freq_sorted2)

di_freq2.plot(30, cumulative=False)
```

Program Output:

```
Group by lemmatized words, add count and sort:
Get just the first row in each lemmatized group
df_words.head(10):
         lem  index      token    stem pos  counts
0     always     50     always   alway  RB      10
1    nothing    116    nothing    noth  NN       6
2       life     54       life    life  NN       6
3        man     74        man     man  NN       5
4       give     39       gave    gave  VB       5
5       fact    106       fact    fact  NN       5
6      world    121      world   world  NN       5
7  happiness    119  happiness   happi  NN       4
8       work    297       work    work  NN       4
9     theory    101     theory  theori  NN       4
```

POS_TYPE: NN

	lem	pos	counts
1	nothing	NN	6
2	life	NN	6
3	man	NN	5
5	fact	NN	5
6	world	NN	5
7	happiness	NN	4
8	work	NN	4
9	theory	NN	4
10	woman	NN	4
17	holmes	NN	3

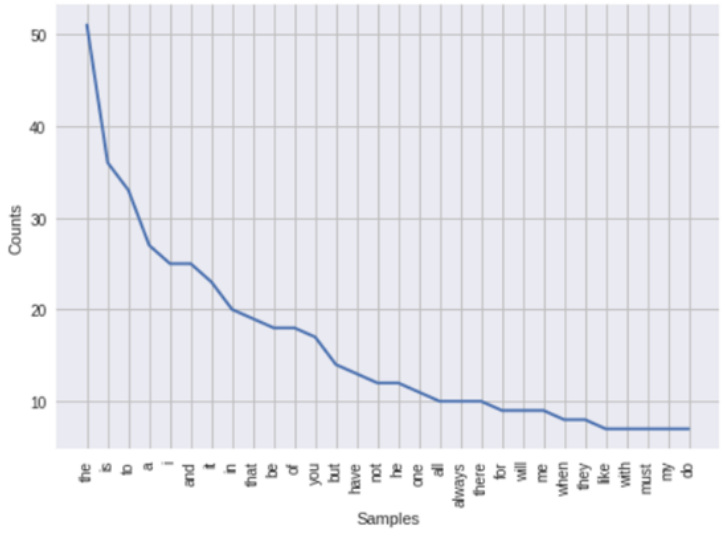
POS_TYPE: JJ

	lem	pos	counts
11	impossible	JJ	4
15	certain	JJ	3
18	curious	JJ	3
34	nice	JJ	2
43	little	JJ	2
48	good	JJ	2
61	improbable	JJ	2
62	best	JJ	2
72	philosophical	JJ	1
81	possible	JJ	1

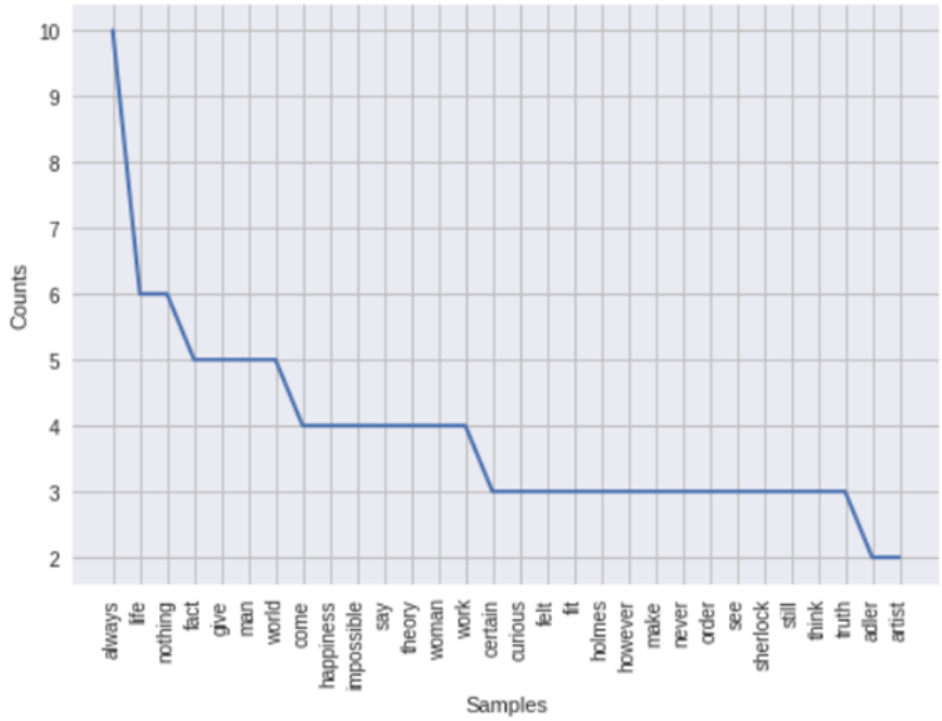
POS_TYPE: VB

	lem	pos	counts
4	give	VB	5
12	say	VB	4
13	come	VB	4
22	see	VB	3
23	make	VB	3
26	think	VB	3

```
li_token_lists_flat[:10]: ['i', 'like', 'living', '', 'i', 'have', 'sometimes', 'been', 'wildly', '']
[('the', 51), ('is', 36), ('to', 33), ('a', 27), ('i', 25), ('and', 25), ('it', 23), ('in', 20), ('that
```




```
[('always', 10), ('life', 6), ('nothing', 6), ('fact', 5), ('give', 5), ('man', 5),
```



Program Output:
The NLP program was successfully executed

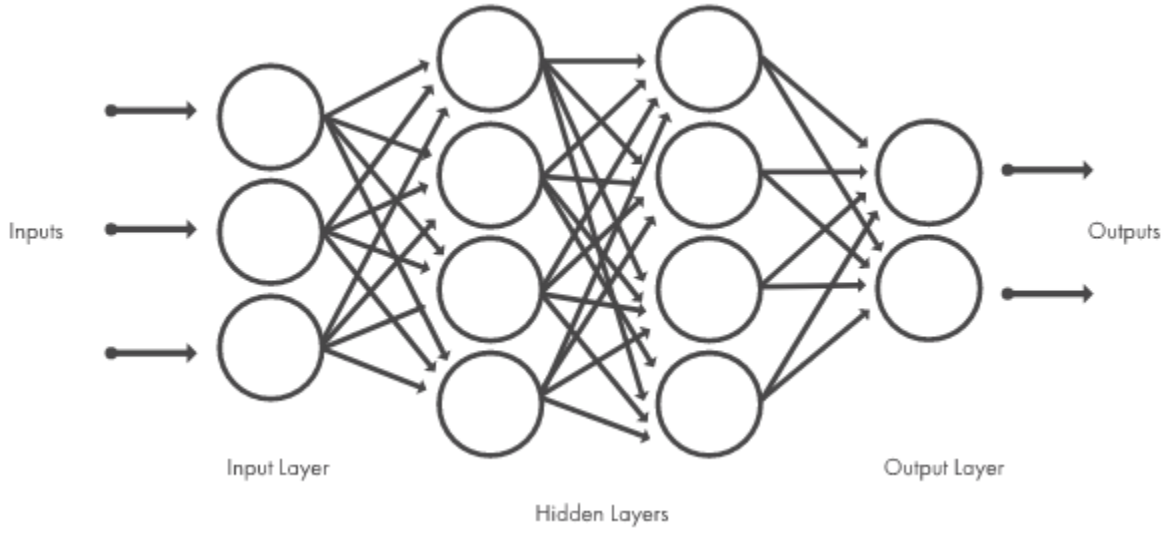
Name : Rashi Agarwal
Register Number: RA1911003010015
Subject: Artificial Intelligence
Section: CSE A1

Ex no:10 Implementation of Deep Learning based Solutions for Real World Problem

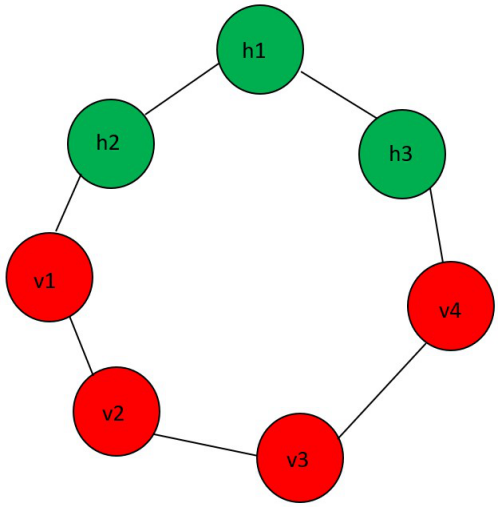
Aim:
To implement Deep Learning based solutions for real world problem

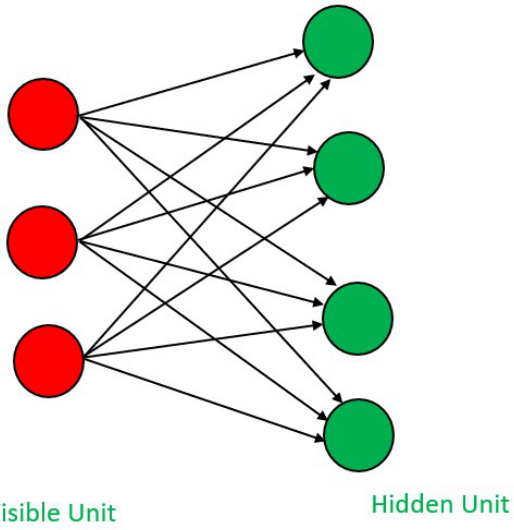
Requirements:
Google Colab

Introduction:
Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to “learn” from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.
Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).



Restricted Boltzmann Machine:
It is a network of neurons in which all the neurons are connected to each other. In this machine, there are two layers named visible layer or input layer and hidden layer. The visible layer is denoted as v and the hidden layer is denoted as the h. In the Boltzmann machine, there is no output layer. Boltzmann machines are random and generative neural networks capable of learning internal representations and are able to represent and (given enough time) solve tough combinatorial problems**.**





Code:

```
from __future__ import print_function
import numpy as np

class RbmImpl:
    '''
    This class implements Restricted Boltzman Machines
    '''

    def __init__(self, num_visible, num_hidden):
        self.num_hidden = num_hidden
        self.num_visible = num_visible
        self.verbose = True
        np_rng = np.random.RandomState(3412)

        self.weights = np.asarray(np_rng.uniform(
                low=-4 * np.sqrt(6. / (num_hidden + num_visible)),
                high=4 * np.sqrt(6. / (num_hidden + num_visible)),
                size=(num_visible, num_hidden)))

        self.weights = np.insert(self.weights, 0, 0, axis = 0)
        self.weights = np.insert(self.weights, 0, 0, axis = 1)


    def train_rbm(self, data, max_epochs = 2000, learning_rate = 0.08):

        num_examples = data.shape[0]
        data = np.insert(data, 0, 1, axis = 1)

        for epoch in range(max_epochs):
            pos_hid_activations = np.dot(data, self.weights)
            pos_hid_probs = self.sigmoid(pos_hid_activations)
            pos_hid_probs[:,0] = 1
            pos_hid_states = pos_hid_probs > np.random.rand(num_examples,
                        self.num_hidden + 1)
            pos_associations = np.dot(data.T, pos_hid_probs)


            neg_vis_activations = np.dot(pos_hid_states, self.weights.T)
            neg_vis_probs = self.sigmoid(neg_vis_activations)
```

```
            neg_vis_probs[:,0] = 1

            neg_hid_activations = np.dot(neg_vis_probs, self.weights)
            neg_hid_probs = self.sigmoid(neg_hid_activations)
            neg_associations = np.dot(neg_vis_probs.T, neg_hid_probs)

            self.weights += learning_rate * ((pos_associations -
                        neg_associations) / num_examples)
            error = np.sum((data - neg_vis_probs) ** 2)
            if self.verbose:
                print('Epoch %s: Error is: %s', (epoch, error))



    def sigmoid(self, val):
        return 1.0 / (1 + np.exp(-val))

if __name__ == '__main__':
    rbmInstance = RbmImpl(num_visible = 6, num_hidden = 2)
    training_data = np.array([[1,1,1,0,0,0], [1,0,1,0,0,0], [1,1,1,0,0,0],
                [0,0,1,1,1,0], [0,0,1,1,0,0], [0,0,1,1,1,0]])
    rbmInstance.train_rbm(data = training_data, max_epochs = 5000)
    print('The weights obtained after training are:')
    print(rbmInstance.weights)
```

Program Output:

```
Epoch %s: Error is: %s (4997, 1.6241900538769207)
Epoch %s: Error is: %s (4998, 0.6703837562577838)
Epoch %s: Error is: %s (4999, 0.6703820186860315)
The weights obtained after training are:
[[ 2.71587159  1.16286077 -0.2108514 ]
 [ 2.3674933   2.19390329 -8.33444308]
 [-1.07796556  1.75176879 -5.12903493]
 [ 5.02838737  4.96332032  0.85604174]
 [-1.53257765 -2.80730465  8.06970939]
 [ 0.34173408 -7.50109917  3.40701386]
 [-4.56405903 -3.11934196 -1.03588066]]
```

Result:
The Deep Learning solution was successfully executed.