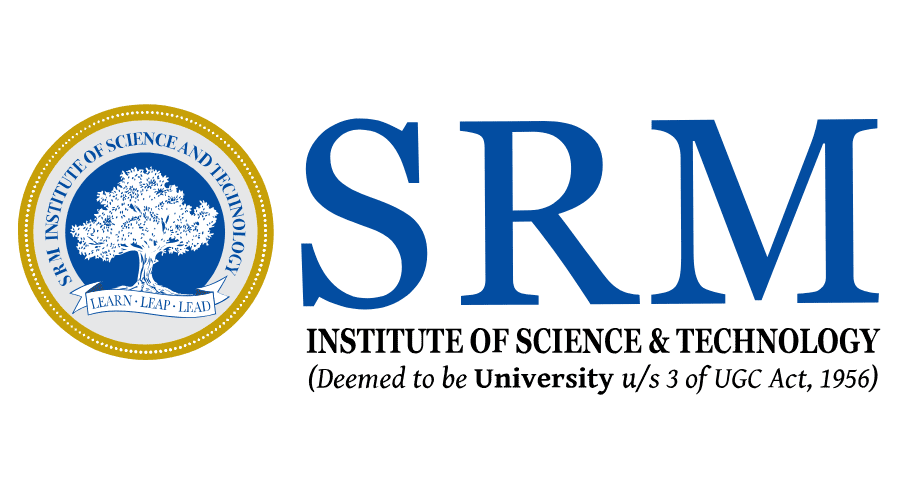
****

**18CSC305J**

**Artificial Intelligence**

Department of Computing Technologies

*Submitted by-*

**Name:-**

**Reg. No. :-**

**Branch:-**

**Section :-**

**Sem:-**

**18CSC305J - Artificial Intelligence**

**Lab Experiments**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Name of the Topic** | **Page No.** | **Signature**  **(Faculty)** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Experiment 1 06/01/2022

**Implementation of Toy Problem (Tic-Tac-Toe)**

**PROBLEM STATEMENT: - To implement a toy problem using AI**

**Algorithm:**

* Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.
* In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
* Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

If X wins on the board we give it a positive value of +10…….and so on

* Make all the functions you need to make the game.
* Make the 3x3 matrix and assign X and O to user and computer.
* Make spaces and symbols between matrix to avoid confusion and for

better look.

* Ask user for the turn if he wants to go first or second.
* Scan the input from the user and map it on the matrix.
* Show the computers chosen grid, computer selects the grid where

there is maximum possibility to win.

* Compute the result, if all the 3 adjacent grid have same symbol, out

put the result

* Else give output- ‘the match is drawn’.
* Ask the user if he wants to play again and if yes repeat the program
* Else exit the code.

**Code**

#include <stdio.h>

void dr\_board(void);

void play\_game(void);

void computer\_move(void);

void player\_move(void);

int user\_first(void);

int play\_again(void);

int find\_win(char);

int middle\_open(void);

int find\_corner(void);

int find\_side(void);

int symbol\_won(char);

int square\_valid(int);

char board[3][3];

char computer, user;

int main(void)

{

int row,col;

while(1)

{

for (row = 0; row < 3; row++)

for (col = 0; col < 3; col++)

board[row][col] = ' ';

if (user\_first())

{

computer = 'O';

user = 'X';

}

else

{

computer = 'X';

user = 'O';

}

play\_game();

if (!play\_again())

break;

}

return 0;

}

void play\_game(void)

{

int turn;

for (turn = 1; turn <= 9; turn++)

{

if (turn % 2 == 1)

{

if (computer == 'X')

computer\_move();

else

player\_move();

}

else

{

if (computer == 'O')

computer\_move();

else

player\_move();

}

dr\_board();

if (symbol\_won(computer))

{

printf("\nComputer Win...\n");

return;

}

else if (symbol\_won(user))

{

printf("Congratulation,You are win!\n");

return;

}

}

printf("The game is a draw.\n");

return;

}

void dr\_board(void)

{

int row,col;

printf("\n\n");

for(row=0;row<3;row++)

{

printf("\t\t\t| %c \t| %c \t| %c \t|\n",board[row][0],board[row][1],board[row][2]);

}

if(row !=2)

{

printf("\t\t################################\n\t\t\t\t");

}

printf("\n");

return ;

}

int user\_first(void)

{

char response;

printf("Do you want to go first? (y/n) ");

do

{

response = getchar();

} while ((response != 'y') && (response != 'Y') && (response != 'n') && (response

!= 'N'));

if ((response == 'y') || (response == 'Y'))

return 1;

else

return 0;

}

int middle\_open(void)

{

if (board[1][1] == ' ')

return 5;

else

return 0;

}

int find\_corner(void)

{

if (board[0][0] == ' ')

return 1;

if (board[0][2] == ' ')

return 3;

if (board[2][0] == ' ')

return 7;

if (board[2][2] == ' ')

return 9;

return 0;

}

int find\_side(void)

{

if (board[0][1] == ' ')

return 2;

if (board[1][0] == ' ')

return 4;

if (board[1][2] == ' ')

return 6;

if (board[2][1] == ' ')

return 8;

return 0;

}

int find\_win(char symbol)

{

int square, row, col;

int result=0 ;

for (square = 1; square <= 9; square++)

{

row = (square - 1) / 3;

col = (square - 1) % 3;

if (board[row][col] == ' ')

{

board[row][col] = symbol;

if (symbol\_won(symbol))

result = square;

board[row][col] = ' ';

}

}

return result;

}

int symbol\_won(char symbol)

{

int row, col;

for (row = 0; row < 3; row++)

{

if ((board[row][0] == symbol) && (board[row][1] == symbol) &&

(board[row][2] == symbol))

return 1;

}

for (col = 0; col < 3; col++)

{

if ((board[0][col] == symbol) && (board[1][col] == symbol) && (board[2][col]

== symbol))

return 1;

}

if ((board[0][0] == symbol) && (board[1][1] == symbol) && (board[2][2] ==

symbol))

return 1;

if ((board[0][1] == symbol) && (board[1][1] == symbol) && (board[2][1] == symbol))

return 1;

if ((board[0][2] == symbol) && (board[1][1] == symbol) && (board[2][0] ==

symbol))

return 1;

if ((board[1][0] == symbol) && (board[1][1] == symbol) && (board[1][2] == symbol))

return 1;

return 0;

}

void computer\_move(void)

{

int square;

int row, col;

square = find\_win(computer);

if (!square)

square = find\_win(user);

if (!square)

square = find\_corner();

if (!square)

square = middle\_open();

if (!square)

square = find\_side();

printf("Computer Choosing %dth square!\n", square);

row = (square - 1) / 3;

col = (square - 1) % 3;

board[row][col] = computer;

return;

}

void player\_move(void)

{

int square;

int row, col;

do

{

printf("Enter a square: ");

scanf("%d", &square);

} while (!square\_valid(square));

row = (square - 1) / 3;

col = (square - 1) % 3;

board[row][col] = user;

return;

}

int square\_valid(int square)

{

int row, col;

row = (square - 1) / 3;

col = (square - 1) % 3;

if ((square >= 1) && (square <= 9))

{

if (board[row][col] == ' ')

return 1;

}

return 0;

}

int play\_again(void)

{

char response;

printf("Do you want to play again? (y/n) ");

do

{response = getchar();

} while ((response != 'y') && (response != 'Y') && (response != 'n') && (response != 'N'));

if ((response == 'y') || (response == 'Y'))

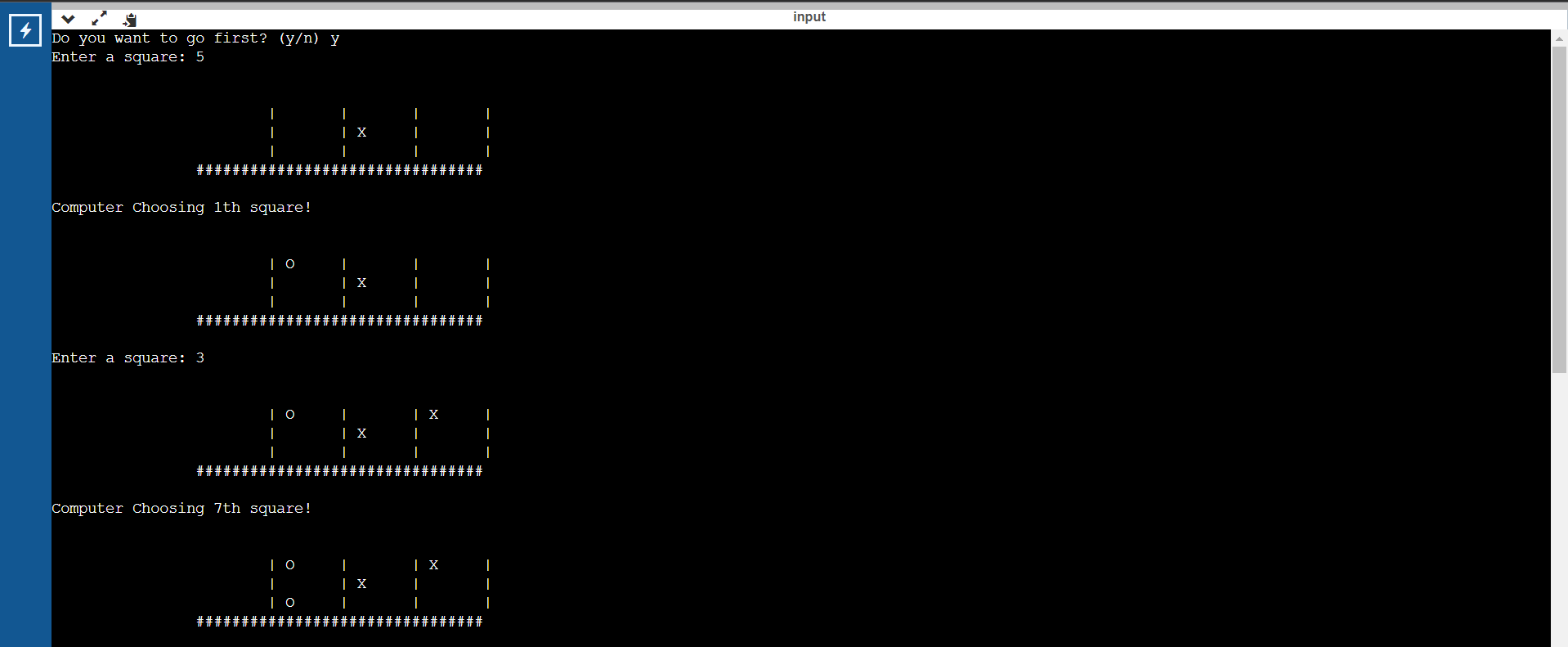
return 1;

else

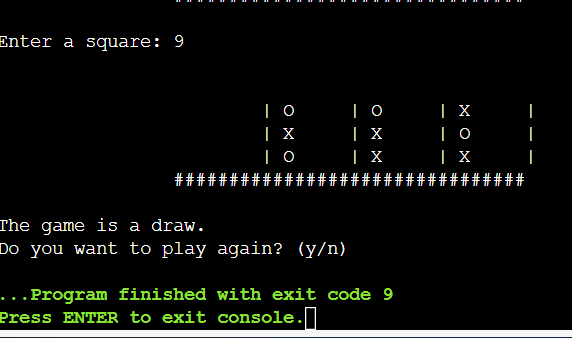
return 0;

}

**Output**

****

****

****

**Result:** The toy problem (Tic Tac Toe) was made and implemented successfully.

EX 2 *Date:25/01/22*

**Travelling Salesman Problem**

**PROBLEM STATEMENT – To implement Travelling salesman problem and hence find minimum cost and minimum path**

**Algorithm:**

* Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
* Now, we will generate all possible permutations of cities which are

(n-1)!.

* Find the cost of each permutation and keep track of the minimum cost permutation.
* Return the permutation with minimum cost**.**

**Code**

#include<iostream>

using namespace std;

int ary[10][10],completed[10],n,cost=0;

void takeInput()

{

int i,j;

cout<<"Enter the number of villages: ";

cin>>n;

cout<<"\nEnter the Cost Matrix\n";

for(i=0;i < n;i++)

{

cout<<"\nEnter Elements of Row: "<<i+1<<"\n";

for( j=0;j < n;j++)

cin>>ary[i][j];

completed[i]=0;

}

cout<<"\n\nThe cost list is:";

for( i=0;i < n;i++)

{

cout<<"\n";

for(j=0;j < n;j++)

cout<<"\t"<<ary[i][j];

}

}

int least(int c)

{

int i,nc=999;

int min=999,kmin;

for(i=0;i < n;i++)

{

if((ary[c][i]!=0)&&(completed[i]==0))

if(ary[c][i]+ary[i][c] < min)

{

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

nc=i;

}

}

if(min!=999)

cost+=kmin;

return nc;

}

void mincost(int city)

{

int i,ncity;

completed[city]=1;

cout<<city+1<<"--->";

ncity=least(city);

if(ncity==999)

{

ncity=0;

cout<<ncity+1;

cost+=ary[city][ncity];

return;

}

mincost(ncity);

}

int main()

{

takeInput();

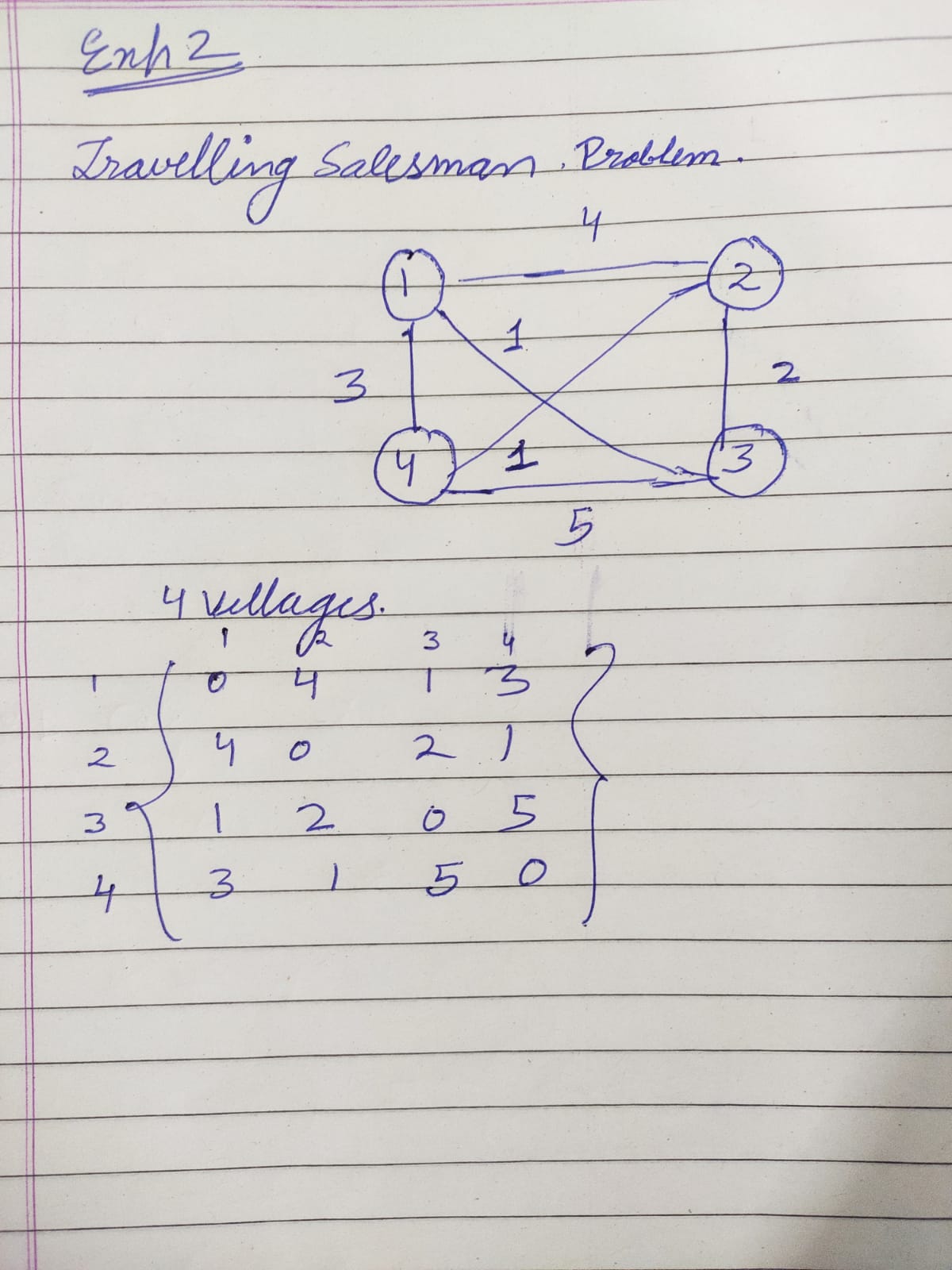
cout<<"\n\nThe Path is:\n";

mincost(0); //passing 0 because starting vertex

cout<<"\n\nMinimum cost is "<<cost;

return 0;

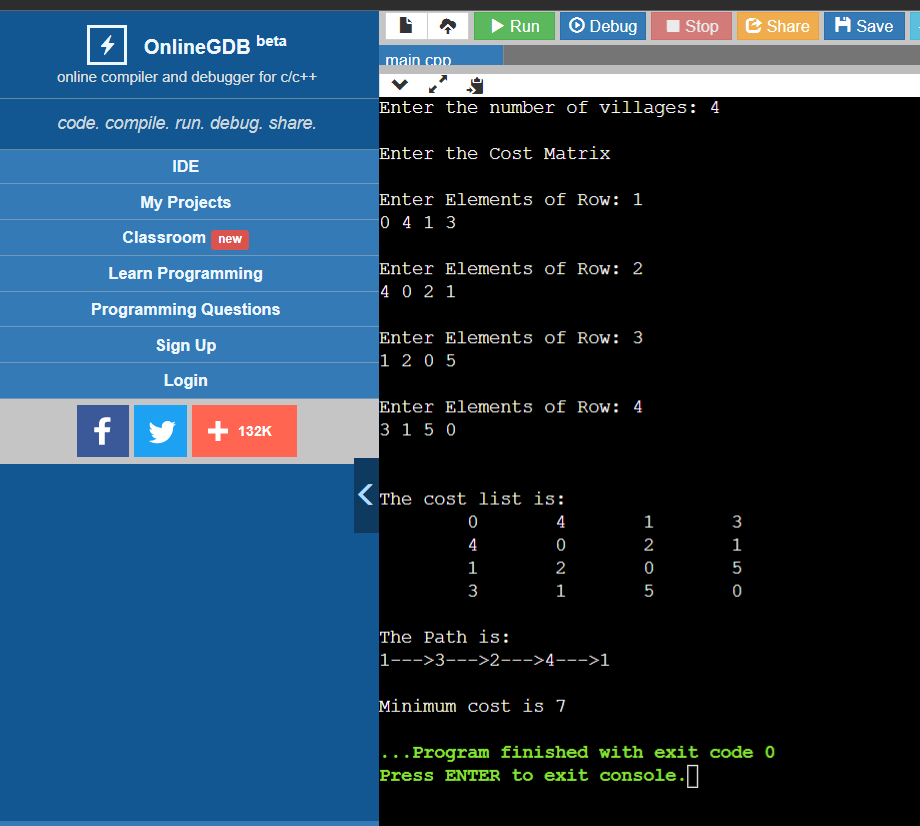
}

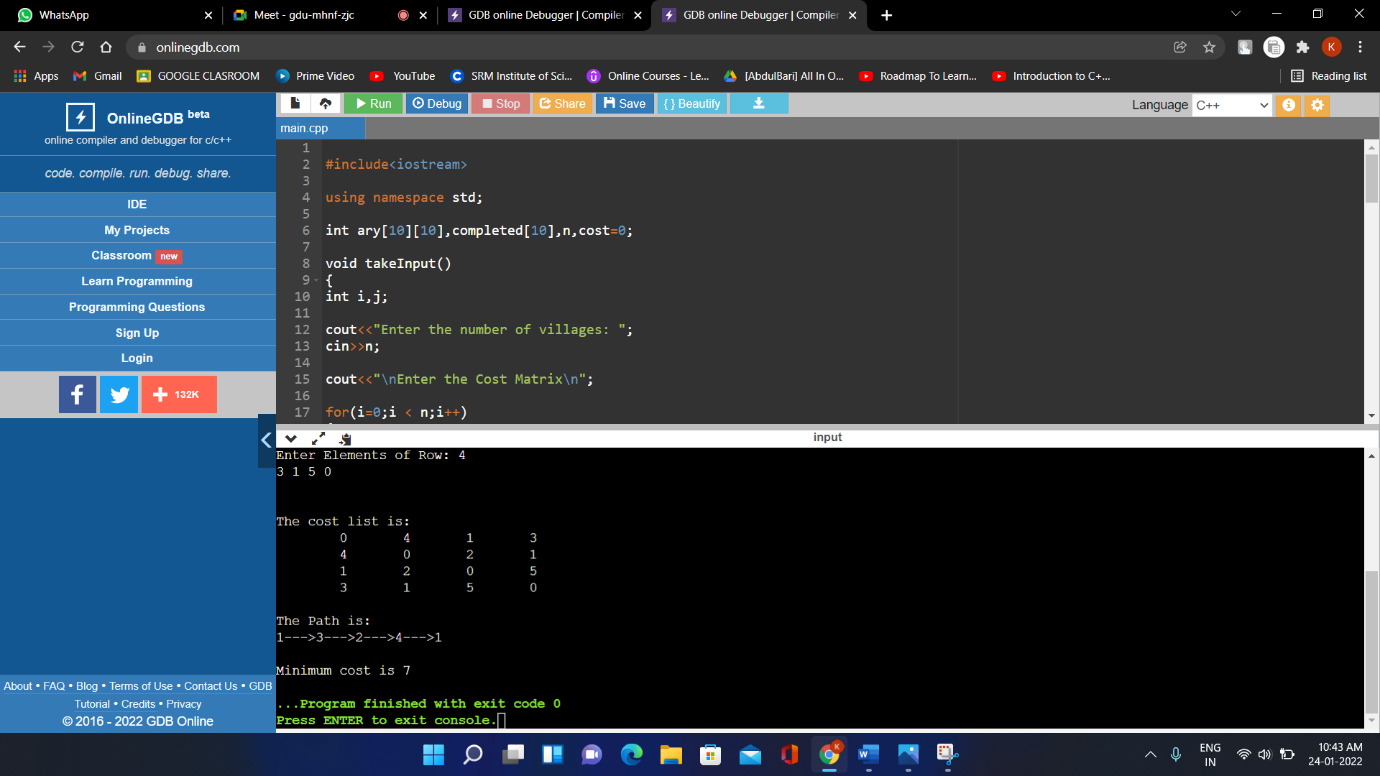


**We will solve it for above problem**

**It can take other input also**

**OUTPUT**





Minimum cost for above problem is: **7**

Minimum path **1🡪3🡪2🡪4🡪1**

**Result: -** Travelling Salesman problem was implemented and minimum path and minimum cost was found

**Ex 3** 02/02/2022

**Constrain Satisfaction Problem**

**(Cryptarithmic problem)**

**Problem statement:** To implement Cryptarithmic problem

**Algorithm**  – Assign each letter a digit from 0 to 9 so that the arithmetic works out correctly. The rules are that all occurrences of a letter must be assigned the same digit, and no digit can be assigned to more than one letter.

* First, create a list of all the characters that need assigning to pass to Solve
* If all characters are assigned, return true if puzzle is solved, false otherwise
* Otherwise, consider the first unassigned character
* for (every possible choice among the digits not in use)

make that choice and then recursively try to assign the rest of the characters  
if recursion successful, return true  
if! successful, unmake assignment and try another digit

* If all digits have been tried and nothing worked, return false to trigger backtracking

**Code**

#include <bits/stdc++.h>

using namespace std;

vector<int> use(10);

struct node

{

char c;

int v;

};

int check(node\* nodeArr, const int count, string s1,

string s2, string s3)

{

int val1 = 0, val2 = 0, val3 = 0, m = 1, j, i;

for (i = s1.length() - 1; i >= 0; i--)

{

char ch = s1[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val1 += m \* nodeArr[j].v;

m \*= 10;

}

m = 1;

for (i = s2.length() - 1; i >= 0; i--)

{

char ch = s2[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val2 += m \* nodeArr[j].v;

m \*= 10;

}

m = 1;

for (i = s3.length() - 1; i >= 0; i--)

{

char ch = s3[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val3 += m \* nodeArr[j].v;

m \*= 10;

}

if (val3 == (val1 + val2))

return 1;

// else return false

return 0;

}

bool permutation(const int count, node\* nodeArr, int n,

string s1, string s2, string s3)

{

if (n == count - 1)

{

for (int i = 0; i < 10; i++)

{

// if not used

if (use[i] == 0)

{

nodeArr[n].v = i;

// if solution found

if (check(nodeArr, count, s1, s2, s3) == 1)

{

cout << "\nSolution found: ";

for (int j = 0; j < count; j++)

cout << " " << nodeArr[j].c << " = "

<< nodeArr[j].v;

return true;

}

}

}

return false;

}

for (int i = 0; i < 10; i++)

{

if (use[i] == 0)

{

nodeArr[n].v = i;

use[i] = 1;

if (permutation(count, nodeArr, n + 1, s1, s2, s3))

return true;

use[i] = 0;

}

}

return false;

}

bool solveCryptographic(string s1, string s2,

string s3)

{

int count = 0;

int l1 = s1.length();

int l2 = s2.length();

int l3 = s3.length();

vector<int> freq(26);

for (int i = 0; i < l1; i++)

++freq[s1[i] - 'A'];

for (int i = 0; i < l2; i++)

++freq[s2[i] - 'A'];

for (int i = 0; i < l3; i++)

++freq[s3[i] - 'A'];

for (int i = 0; i < 26; i++)

if (freq[i] > 0)

count++;

if (count > 10)

{

cout << "Invalid strings";

return 0;

}

node nodeArr[count];

for (int i = 0, j = 0; i < 26; i++)

{

if (freq[i] > 0)

{

nodeArr[j].c = char(i + 'A');

j++;

}

}

return permutation(count, nodeArr, 0, s1, s2, s3);

}

int main()

{

string s1,s2,s3;

cout<<"Enter 1 string : ";

cin>>s1;

cout<<"Enter 2 string : ";

cin>>s2;

cout<<"Enter 3 string : ";

cin>>s3;

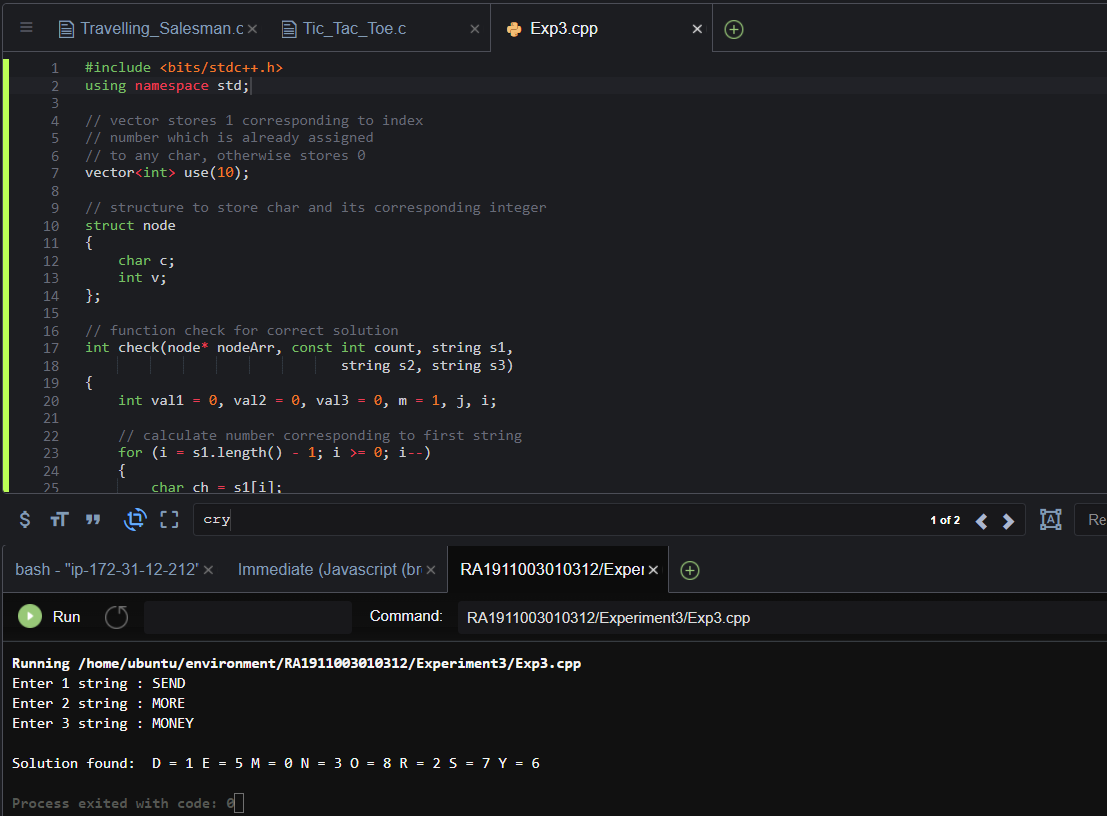
if (solveCryptographic(s1, s2, s3) == false)

cout << "No solution";

return 0;

}

**OUTPUT**









**Result:- Cryptarithmetic Problem was successfully implemented.**

**EX 4** 22/02/2022

**Implementation of BFS AND DFS**

**Problem statement: - To implement BFS and DFS on given graph.**

**BFS algorithm**

A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node.

**Code for BFS**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int main(){

int V,e;

cout<<"Enter no of vertices: "<<endl;

cin>>V;

cout<<"Enter no of edges: "<<endl;

cin>>e;

vector<int>adj[V];

cout<<"Create graph: "<<endl;

for(int i=0;i<e;i++){

int start,end;

cin>>start>>end;

adj[start].push\_back(end);

}

vector<int>ans;

bool visited[V]={false};

queue<int>q1;

q1.push(0);

while(!q1.empty()){

int node=q1.front();

q1.pop();

visited[node]=true;

ans.push\_back(node);

for(auto it:adj[node]){

if(visited[it]==false){

q1.push(it);

visited[it]=true;

}

}

}

cout<<"Bfs: "<<endl;

for(auto it:ans)

cout<<it<<" ";

}

**Depth First Search Algorithm**

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

**Code for DFS**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

void dfs(int node,vector<bool>&visited,vector<int>adj[],vector<int>&ans){

if(visited[node]==true)

return;

ans.push\_back(node);

visited[node]=true;

for(int i=0;i<adj[node].size();i++){

if(visited[adj[node][i]]==false)

dfs(adj[node][i],visited,adj,ans);

}

}

int main(){

int V,e;

cout<<"Enter no of vertices: "<<endl;

cin>>V;

cout<<"Enter no of edges: "<<endl;

cin>>e;

vector<int>adj[V];

cout<<"Create graph: "<<endl;

for(int i=0;i<e;i++){

int start,end;

cin>>start>>end;

adj[start].push\_back(end);

}

vector<bool>visited(V,false);

vector<int>ans;

dfs(0,visited,adj,ans);

cout<<"dfs: "<<endl;

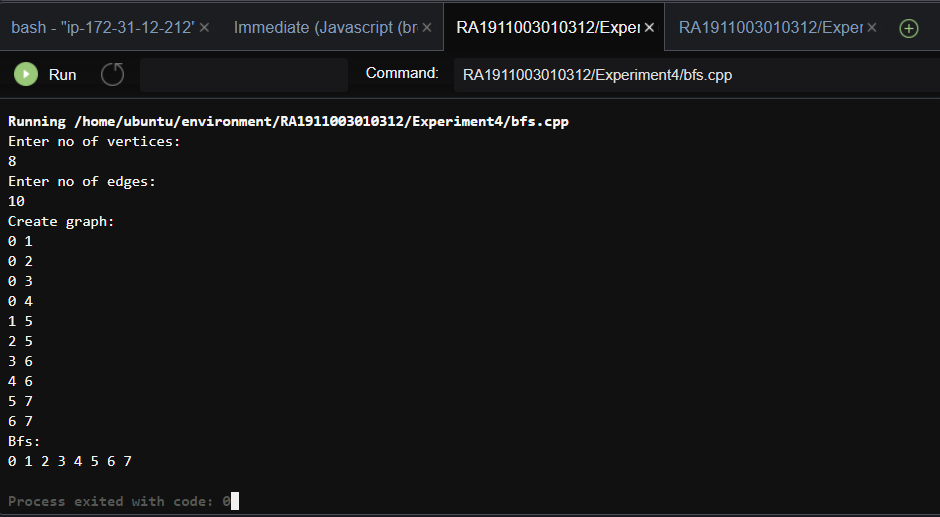
for(auto it:ans)

cout<<it<<" ";

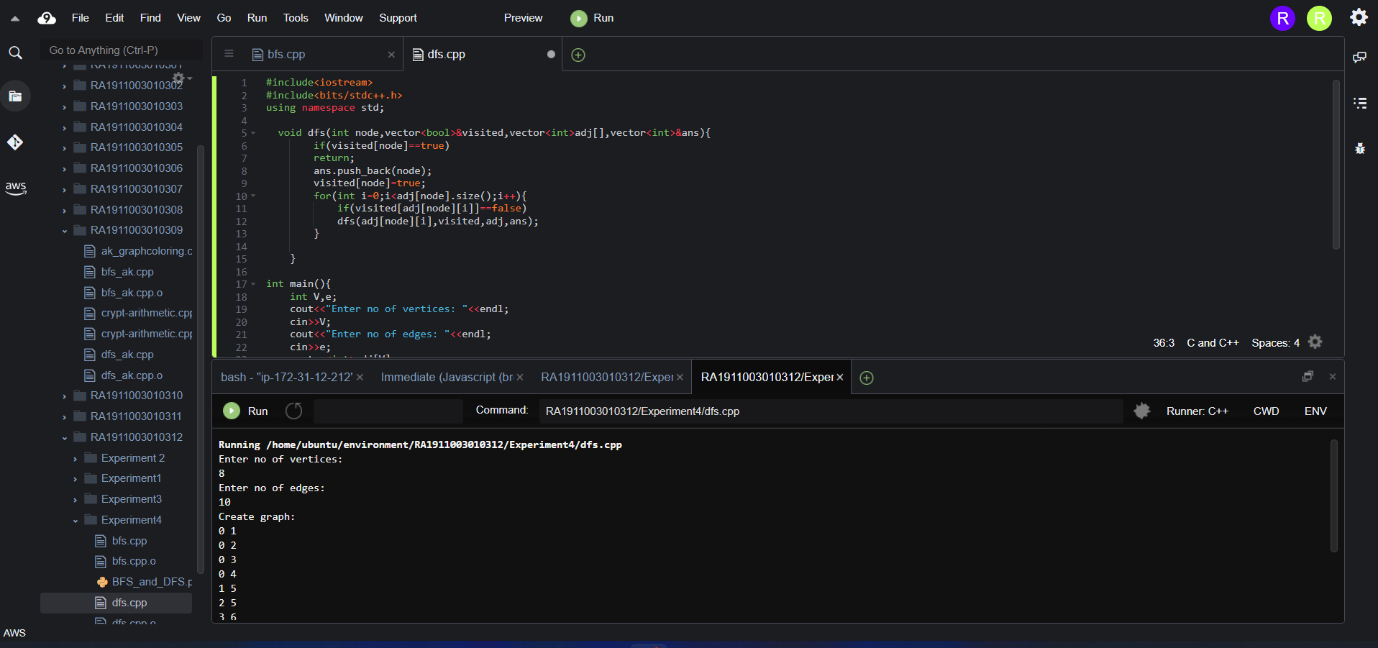
}

*OUTPUT for BFS:*





**Output for DFS:**





**Result: - Code for BFS and DFS in graph were successfully implemented.**

Ex 5 Date:8/02/2022

Implementation of BFS (Best First Search)

& A\* Algorithm

**Problem statement: - Implementation of BFS (Best First Search) and A\* Algorithm and finding out the path**

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 check it**

**Algorithm for implementing Best First Search**

**Step 1**: Create a priority Queue pqueue.

**Step 2**: insert ‘start’ in pqueue : pqueue.insert(start)

**Step 3:** delete all elements of pqueue one by one.

   Step 3.1: if, the element is goal. Exit.

   Step 3.2: else, traverse neighbours and mark the node examined.

**Step 4:** End.

**Algorithm of A\* search:**

**Step1:** Place the starting node in the OPEN list.

**Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

**Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

**Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

**Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

**Step 6:** Return to **Step 2**.

**Code For BFS**

#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pi;

vector<vector<pi> > graph;

void addedge(int x, int y, int cost)

{

graph[x].push\_back(make\_pair(cost, y));

graph[y].push\_back(make\_pair(cost, x));

}

void best\_first\_search(int source, int target, int n)

{

vector<bool> visited(n, false);

priority\_queue<pi, vector<pi>, greater<pi> > pq;

// sorting in pq gets done by first value of pair

pq.push(make\_pair(0, source));

int s = source;

visited[s] = true;

while (!pq.empty()) {

int x = pq.top().second;

// Displaying the path having lowest cost

cout << 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));

}

}

}

}

int main()

{

int v,e,i,a,b,c;

printf("Enter the number of nodes: ");

scanf("%d",&v);

graph.resize(v);

printf("Enter the number of edges: ");

scanf("%d",&e);

for(i=0;i<e;i++){

scanf("%d %d %d",&a,&b,&c);

addedge(a,b,c);

}

int source;

int target;

printf("Enter the source node: ");

scanf("%d",&source);

printf("Enter the target node: ");

scanf("%d",&target);

best\_first\_search(source, target, v);

return 0;

}

**Code for A\* Algorithm**

def aStarAlgo(start\_node, stop\_node):

print("\nA\* Algorithm \n")

open\_set = set(start\_node)

closed\_set = set()

g = {} #store distance from starting node

parents = {}# parents contains an adjacency map of all nodes

#ditance of starting node from itself is zero

g[start\_node] = 0

#start\_node is root node i.e it has no parent nodes

#so start\_node is set to its own parent node

parents[start\_node] = start\_node

while len(open\_set) > 0:

n = None

#node with lowest f() is found

for v in open\_set:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n == stop\_node or Graph\_nodes[n] == None:

pass

else:

for (m, weight) in get\_neighbors(n):

#nodes 'm' not in first and last set are added to first

#n is set its parent

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

#for each node m,compare its distance from start i.e g(m) to the

#from start through n node

else:

if g[m] > g[n] + weight:

#update g(m)

g[m] = g[n] + weight

#change parent of m to n

parents[m] = n

#if m in closed set,remove and add to open

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop\_node

# then we begin reconstructin the path from it to the start\_node

if n == stop\_node:

path = []

while parents[n] != n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

print('Path found: {}'.format(path))

return path

# remove n from the open\_list, and add it to closed\_list

# because all of his neighbors were inspected

open\_set.remove(n)

closed\_set.add(n)

print('Path does not exist!')

return None

#define fuction to return neighbor and its distance

#from the passed node

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

def heuristic(n):

H\_dist = {

'A': 11,

'B': 6,

'C': 9,

'D': 1,

'E': 7,

'G': 0,

}

return H\_dist[n]

#Describe your graph here

Graph\_nodes = {

'A': [('B', 2), ('E', 3)],

'B': [('C', 1),('G', 9)],

'C': None,

'E': [('D', 5)],

'D': [('G', 1)],

}

print("Enter start Node: ")

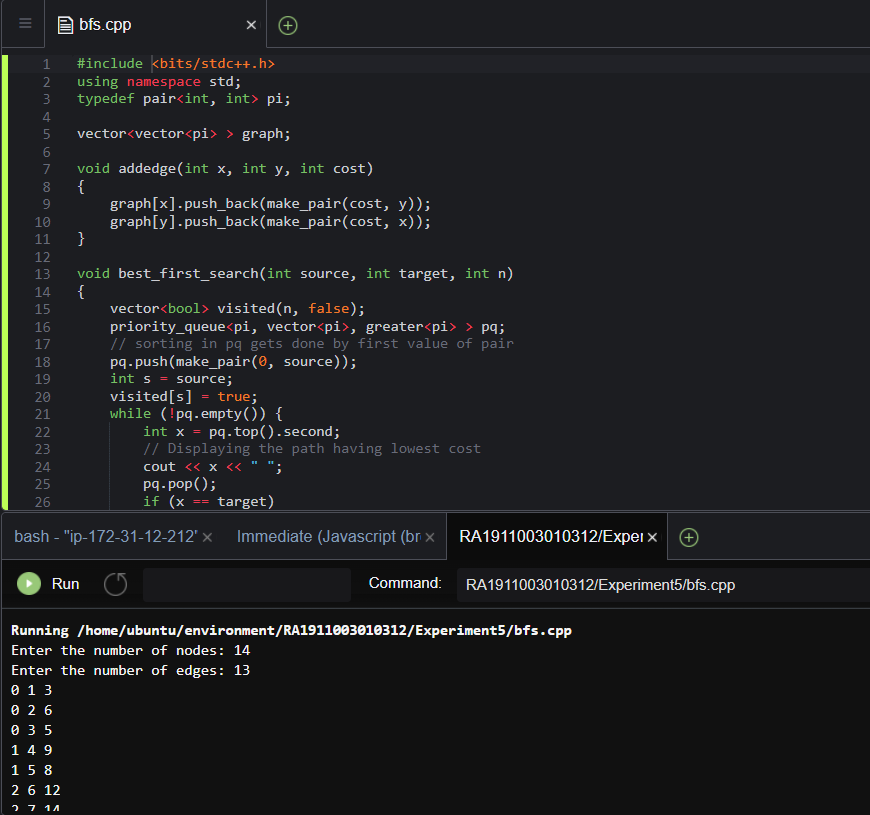
s=input()

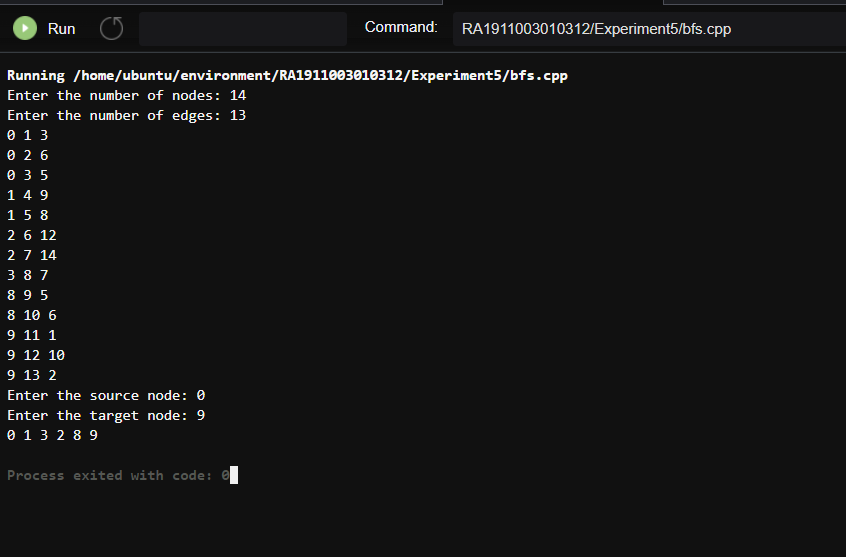
print("Enter target Node: ")

t=input()

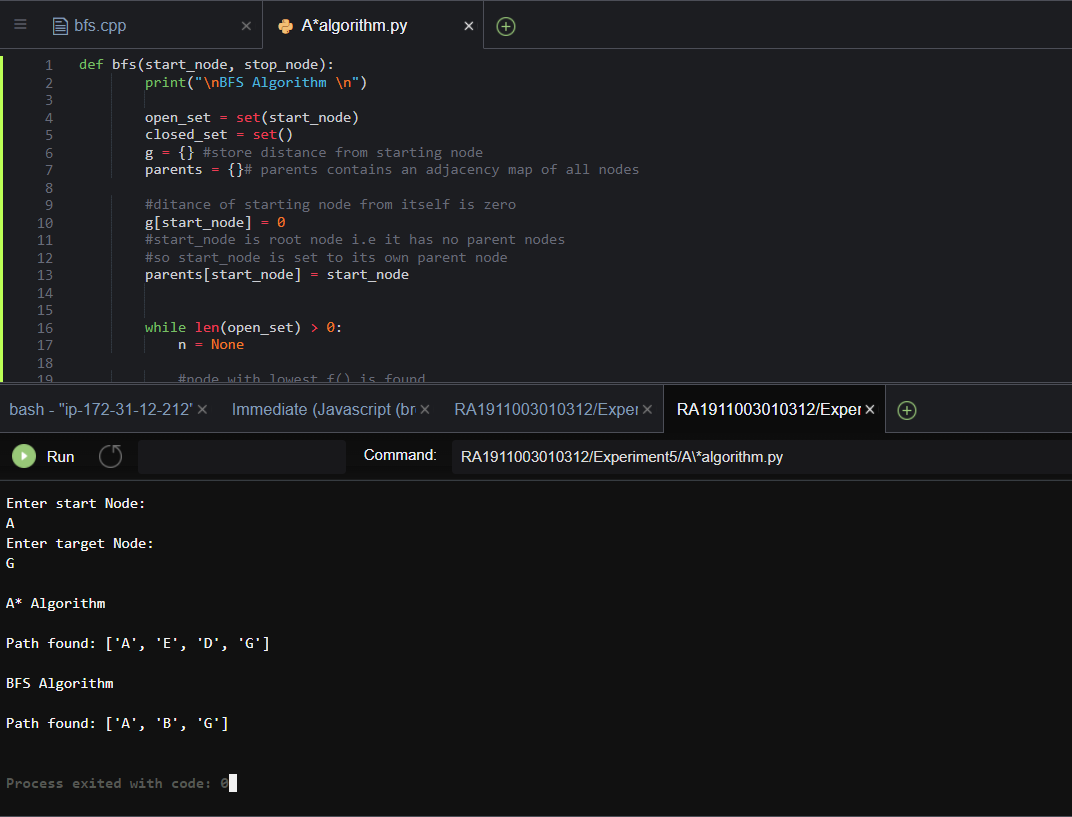
aStarAlgo(s, t)

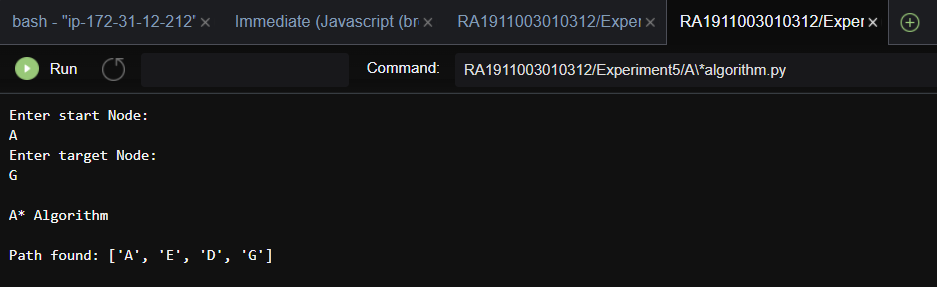
**Output for BFS**





**FOR A\* ALGORITHM**





**Result:-** BFS and A\* algorithm were successfully implemented and their respective paths were found.

Ex 6 22/03/2022

Implementation of uncertain methods for an application (Fuzzy logic/ Dempster Shafer Theory)

**Problem statement:- Implementation of uncertain methods for an application using Fuzzy logic.**

**Algorithm:**

1. Locate the input, output, and state variables of the plane under consideration. I

2. 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.

3. Obtain the membership function for each fuzzy subset.

4. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side

and the output of fuzzy subsets on the other side, thereby forming the rule base.

5. Choose appropriate scaling factors for the input and output variables for normalizing

the variables between [0, 1] and [-1, I] interval.

6. Carry out the fuzzification process.

7. Identify the output contributed from each rule using fuzzy approximate reasoning.

8. Combine the fuzzy outputs obtained from each rule.

9. Finally, apply defuzzification to form a crisp output

**Code:**

A = dict()

B = dict()

Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

for A\_key, B\_key in zip(A, B):

A\_value = A[A\_key]

B\_value = B[B\_key]

if A\_value > B\_value:

Y[A\_key] = A\_value

else:

Y[B\_key] = B\_value

print('Fuzzy Set Union is :', Y)

for A\_key, B\_key in zip(A, B):

A\_value = A[A\_key]

B\_value = B[B\_key]

if A\_value < B\_value:

Y[A\_key] = A\_value

else:

Y[B\_key] = B\_value

print('Fuzzy Set Intersection is :', Y)

for A\_key in A:

Y[A\_key]= 1-A[A\_key]

print('Fuzzy Set Complement of A is :', Y)

for B\_key in B:

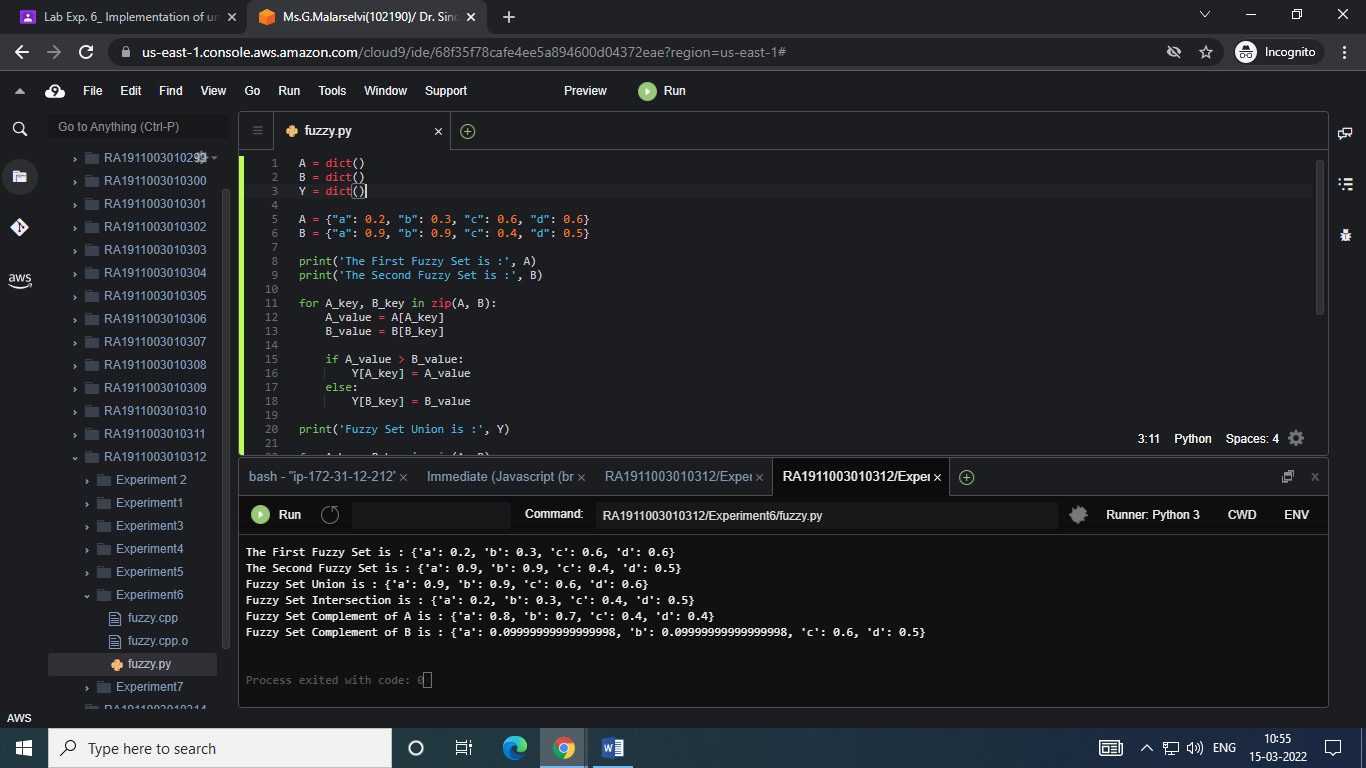
Y[B\_key]= 1-B[B\_key]

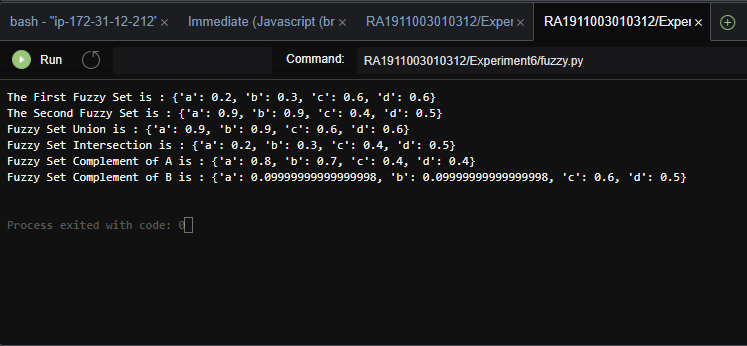
print('Fuzzy Set Complement of B is :', Y)

**Input:**

****

**Output:**

****

****

**Result: Fuzzy logic for an application was successfully implemented.**

Ex.7 Date:22/03/2022

Implementation of unification and resolution for real world problems.

**Problem statement: To implement unification and resolution for real world problem.**

Algorithm:-

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,

**CODE: unification**

def get\_index\_comma(string):

index\_list = list()

# Count open parentheses

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 == '(':

return False

return True

def process\_expression(expr):

# Remove space in expression

expr = expr.replace(' ', '')

# Find the first index == '('

index = None

for i in range(len(expr)):

if expr[i] == '(':

index = i

break

# Return predicate symbol and remove predicate symbol in expression

predicate\_symbol = expr[:index]

expr = expr.replace(predicate\_symbol, '')

# Remove '(' in the first index and ')' in the last index

expr = expr[1:len(expr) - 1]

# List of arguments

arg\_list = list()

# Split string with commas, return list of arguments

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):

# Step 1:

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\_\_':

# Data 1

#f1 = 'p(b(A), X, f(g(Z)))'

#f2 = 'p(Z, f(Y), f(Y))'

# Data 2

f1 = 'Q(a, g(x, a), f(y))'

f2 = 'Q(a, g(f(b), a), x)'

# Data 3

#f1 = 'Q(a, g(x, a, d), f(y))'

#f2 = 'Q(a, g(f(b), a), x)'

result = unify(f1, f2)

if not result:

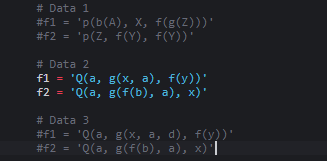
print('Unification failed!')

else:

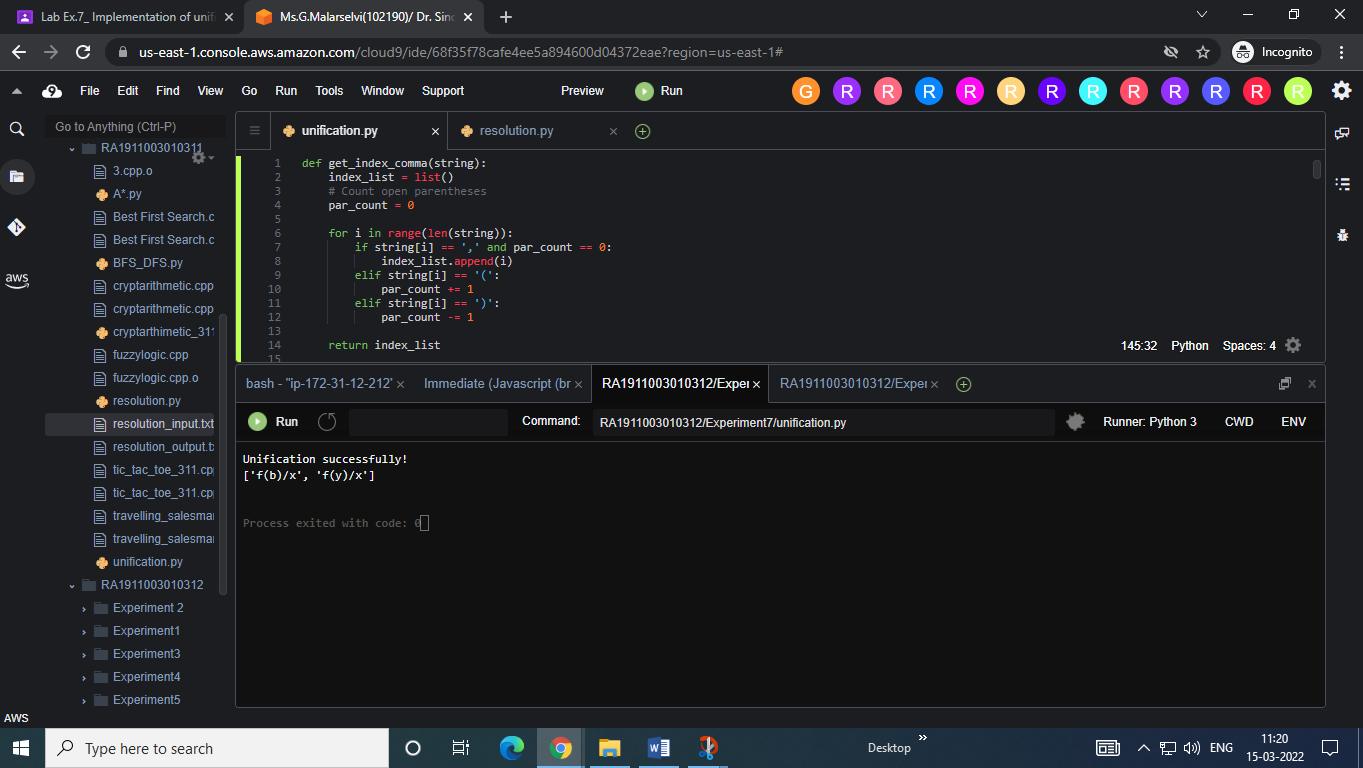
print('Unification successfully!')

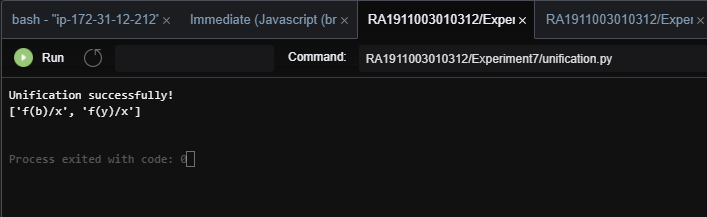
print(result)

**Input:**



**Output:**





**Code 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)):

if "=>" in self.inputSentences[sentenceIdx]: # Do negation of the Premise and add them as literal

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("RA1911003010312/Experiment7/resolution\_input.txt")

knowledgeBase = KB(inputSentences\_)

knowledgeBase.prepareKB()

results\_ = knowledgeBase.askQueries(inputQueries\_)

printOutput("RA1911003010311/resolution\_output.txt", results\_)

**resolution\_input.txt**

2

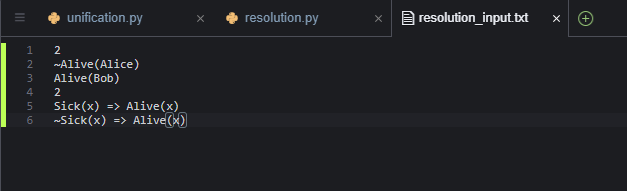
~Alive(Alice)

Alive(Bob)

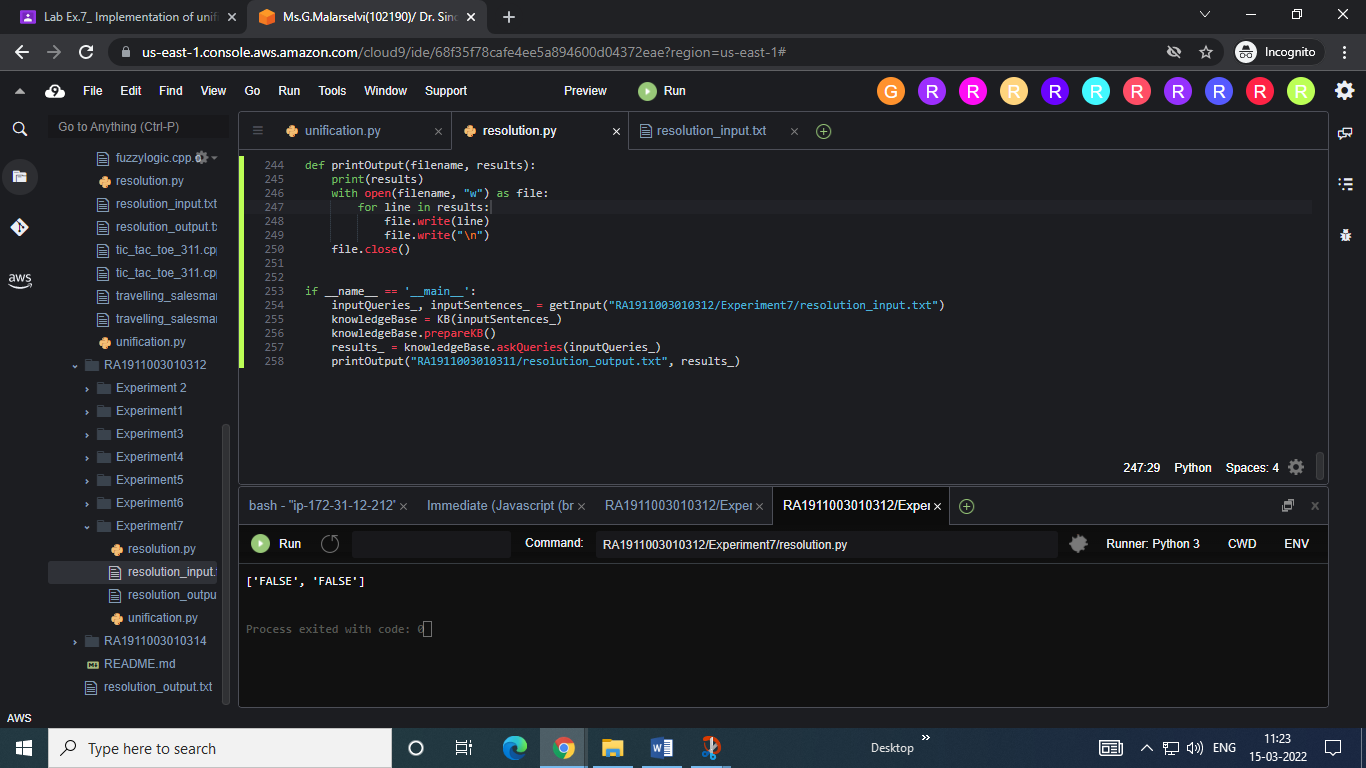
2

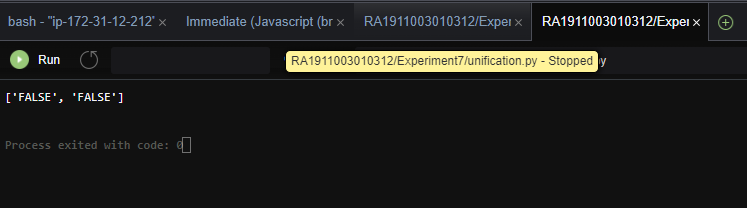
Sick(x) => Alive(x)

~Sick(x) => Alive(x)



**OUTPUT**

****

****

**Result : Unification and resolution problems were successfully implemented.**

**Experiment – 8 Date – 04/04/22**

**Implementation of Learning Algorithm**

**Problem statement: To implement Random Forest Algorithm (regression and classification) for real world problem**

**Regression:**

**Source Code**

import pandas as pd

import numpy as np

dataset = pd.read\_csv('D:\Datasets\petrol\_consumption.csv')

dataset.head()

X = dataset.iloc[:, 0:4].values

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n\_estimators=20, random\_state=0)

regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test)

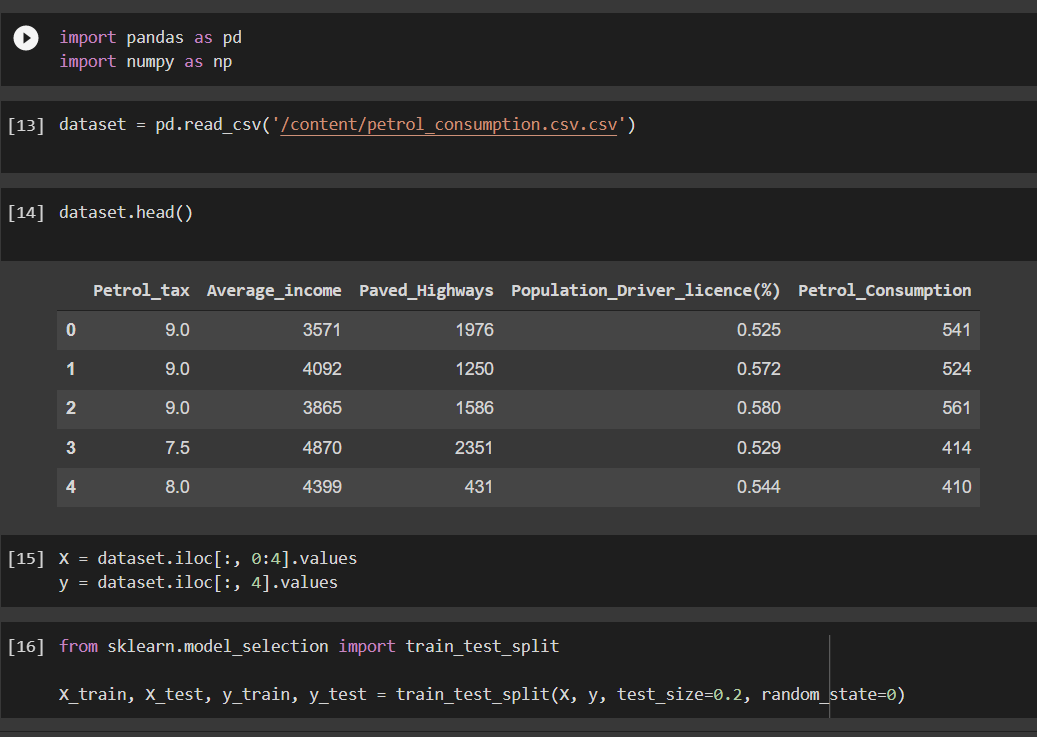
from sklearn import metrics

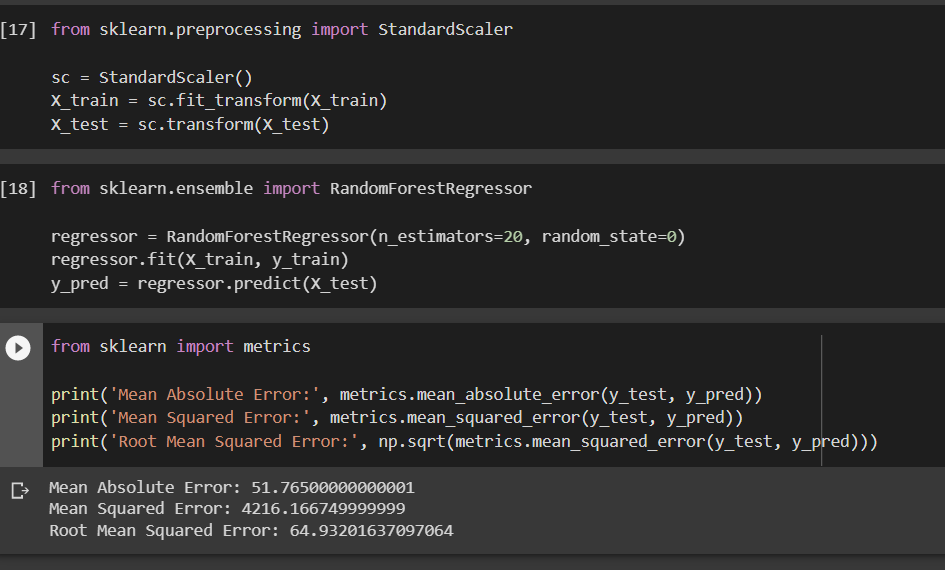
print('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))

print('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))

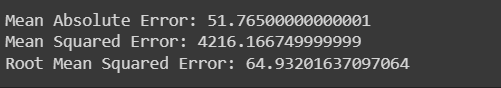
print('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

**Screenshot Input and output:**





Output:



**Classification**

**Source Code:**

#Importing libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

#Importing Dataset

col = [ 'Class Name','Left weight','Left distance','Right weight','Right distance']

df = pd.read\_csv('balance-scale.data',names=col,sep=',')

df.head()

#Importing Dataset

col = [ 'Class Name','Left weight','Left distance','Right weight','Right distance']

df = pd.read\_csv('balance-scale.data',names=col,sep=',')

df.head()

#Importing Dataset

col = [ 'Class Name','Left weight','Left distance','Right weight','Right distance']

df = pd.read\_csv('balance-scale.data',names=col,sep=',')

df.head()

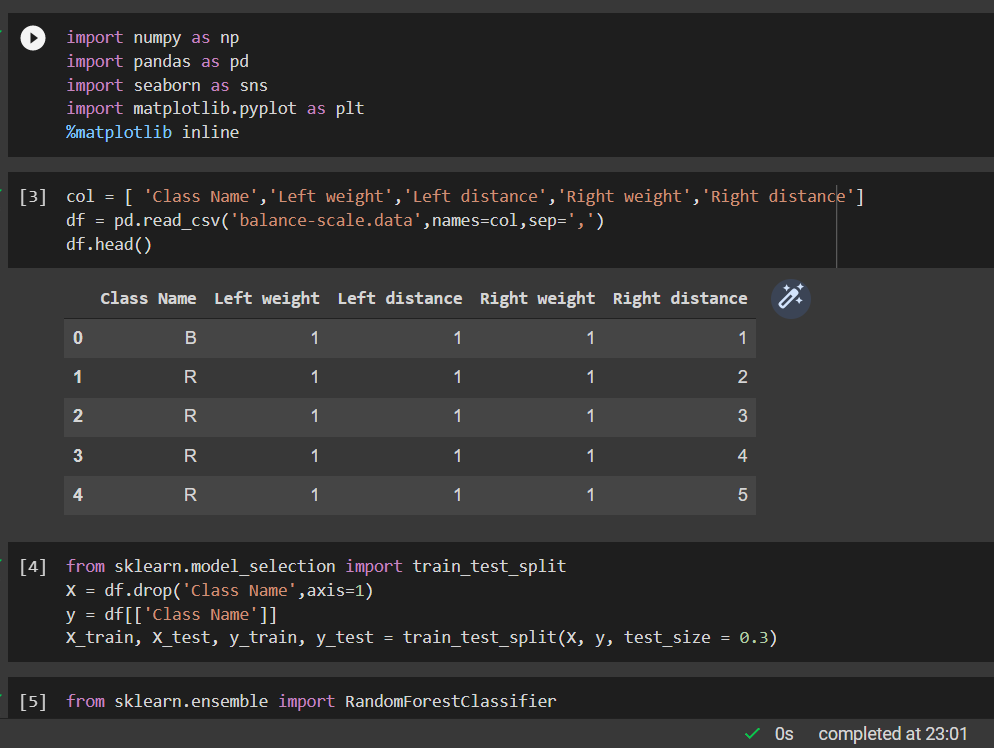
#Testing Accuracy

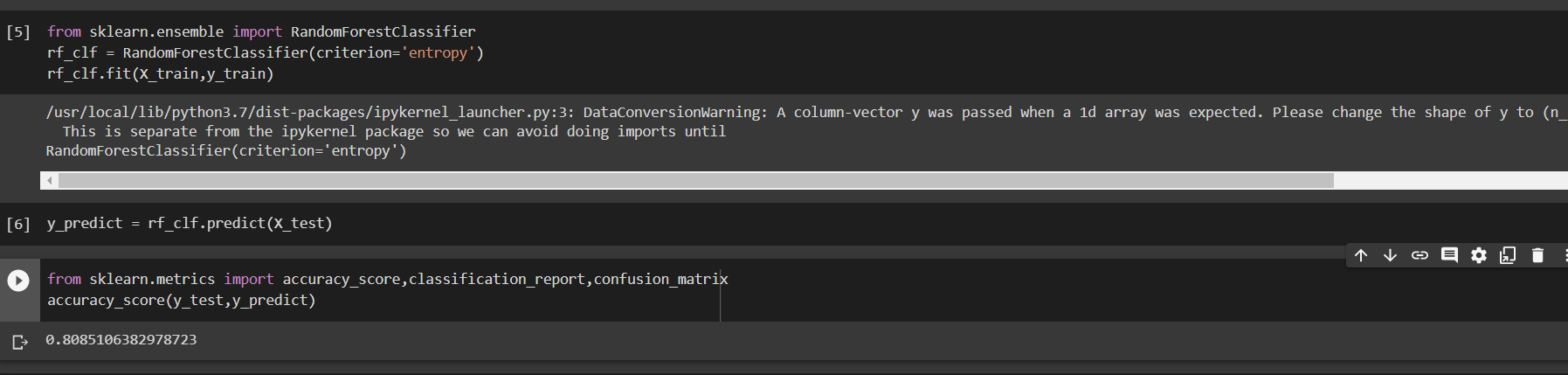
y\_predict = rf\_clf.predict(X\_test)

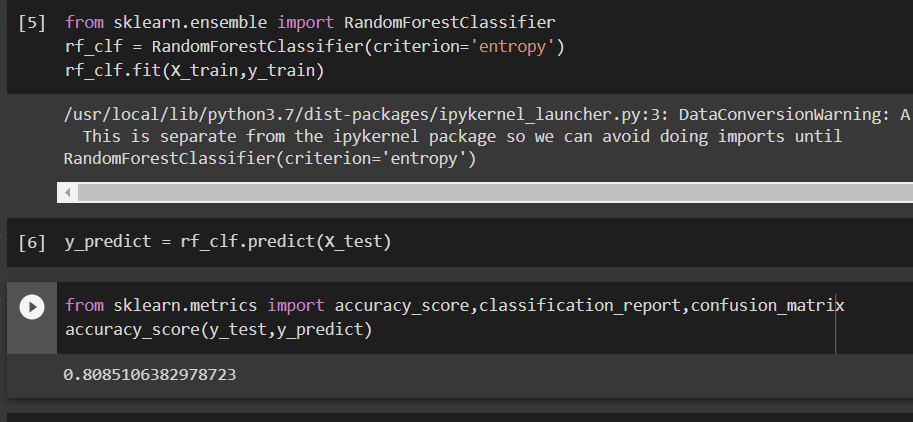
from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

accuracy\_score(y\_test,y\_predict)

**Input And Output Screenshot:**

****

****

****

**Result:- Random Forest Algorithm (regression and classification) for real world problem is implemented**

**Ex – 9 18/04/2022**

**Implementation of NLP Programs**

**Problem statement: To Implement Sentiment Analysis (NLP Technique)**

* **Algorithm:** Find and extract the opinionated data (aka sentiment data) on a specific platform (customer support, reviews, etc.)
* Determine its polarity (positive or negative)
* Define the subject matter (what is being talked about in general and specifically)
* Identify the opinion holder (on its own and in correlation with the existing audience segments)

**Program Code:**

import pandas as pd import numpy as np import nltk

nltk.download('vader\_lexicon')

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from google.colab import drive drive.mount('/content/gdrive')

df\_avatar = pd.read\_csv('/content/gdrive/MyDrive/Colab Notebooks/avatar.csv', encoding = 'unicode\_escape', engine='python')

df\_avatar\_lines = df\_avatar.groupby('character').count()

df\_avatar\_lines = df\_avatar\_lines.sort\_values(by=['character\_words'], ascending

=False)[:10]

top\_character\_names = df\_avatar\_lines.index.values

# filtering out non-top characters

df\_character\_sentiment = df\_avatar[df\_avatar['character'].isin(top\_character\_na mes)]

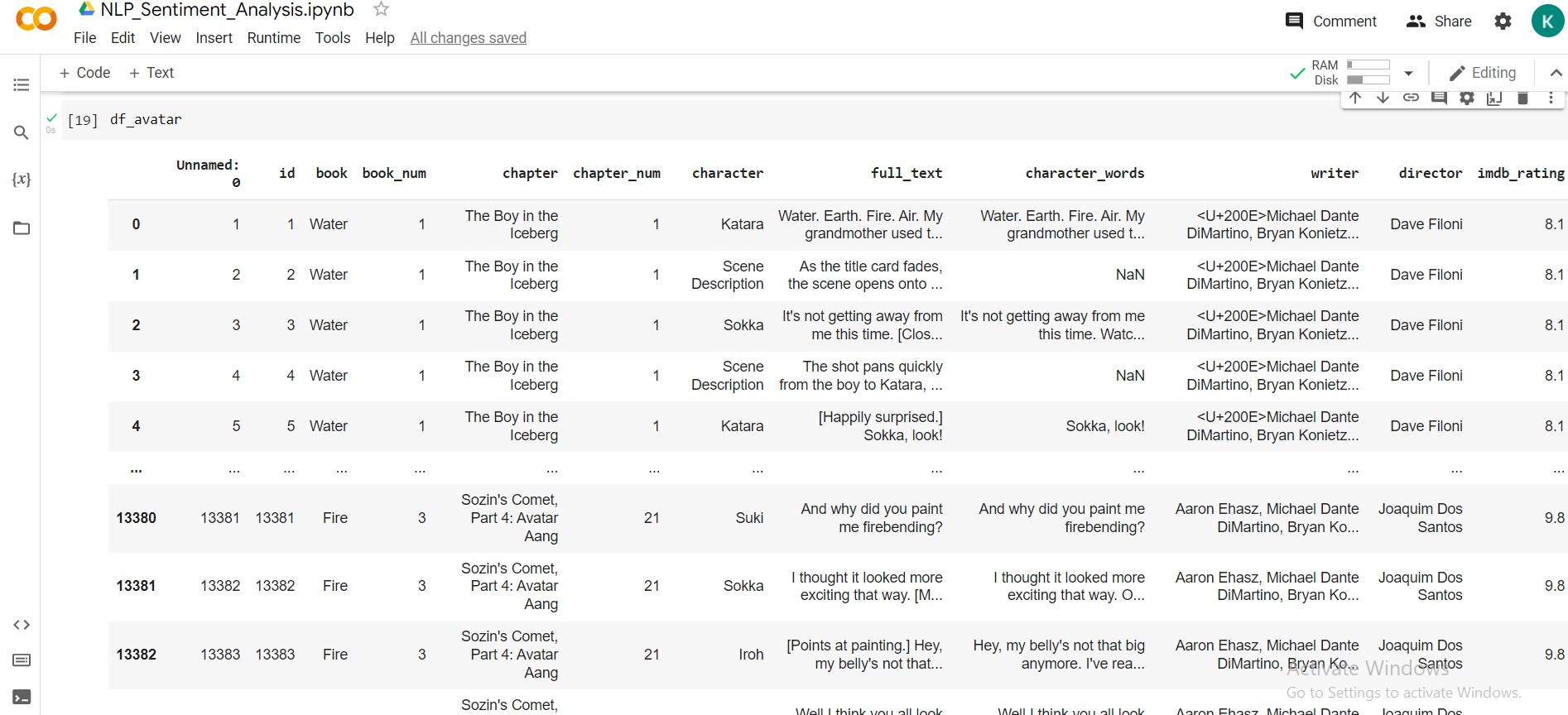
df\_character\_sentiment = df\_character\_sentiment[['character', 'character\_words'

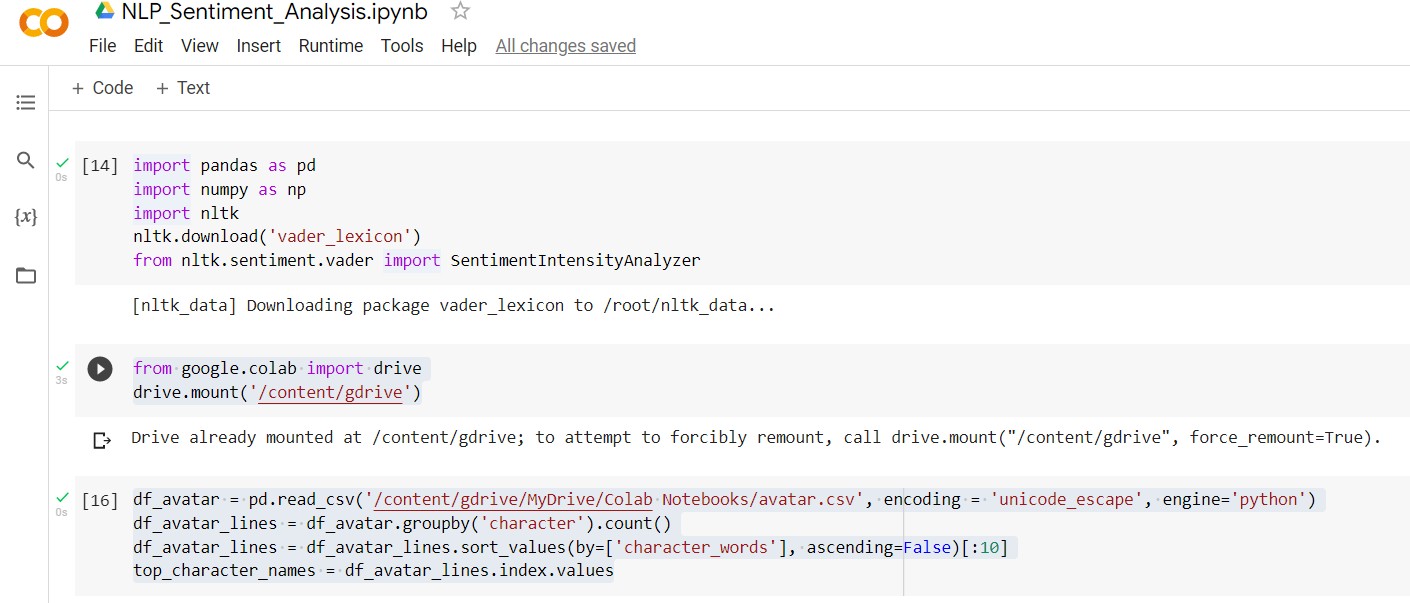
]]

# calculating sentiment score

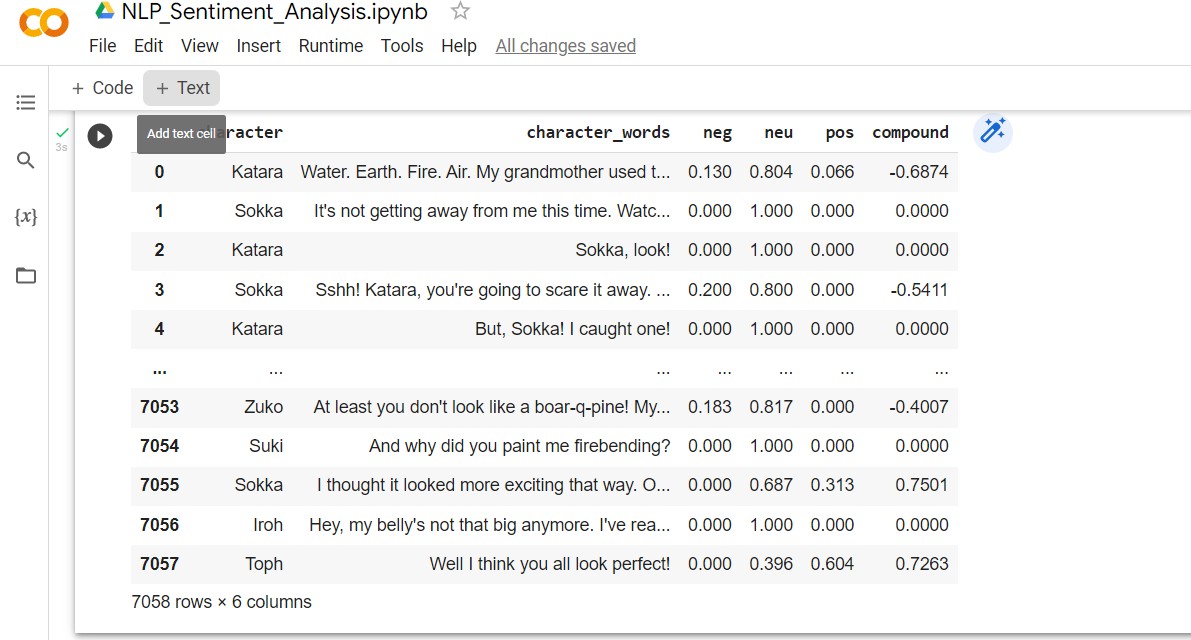
sid = SentimentIntensityAnalyzer() df\_character\_sentiment.reset\_index(inplace=True, drop=True) df\_character\_sentiment[['neg', 'neu', 'pos', 'compound']] = df\_character\_sentime nt['character\_words'].apply(sid.polarity\_scores).apply(pd.Series) df\_character\_sentiment

# Sample Input and Output:









**Result: Sentiment Analysis (NLP Technique) was successfully**

**Implemented.**

**Ex 10 20/4/2022**

**Application of Deep Learning Model on an Application**

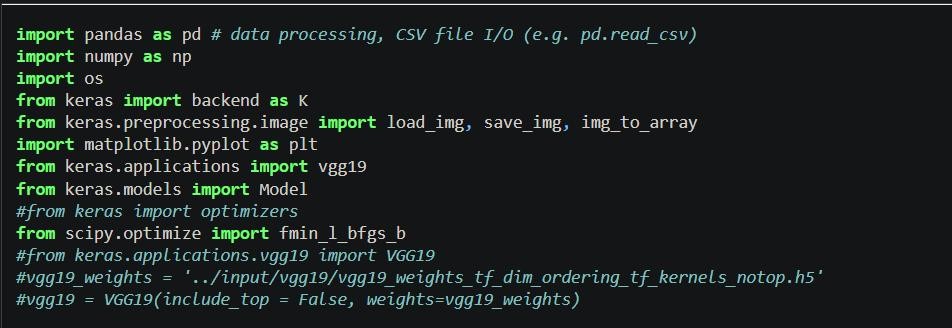
**PROBLEM STATEMENT: Transfer Image Style from one picture to other using GAN (General Adversarial Network) Target:**



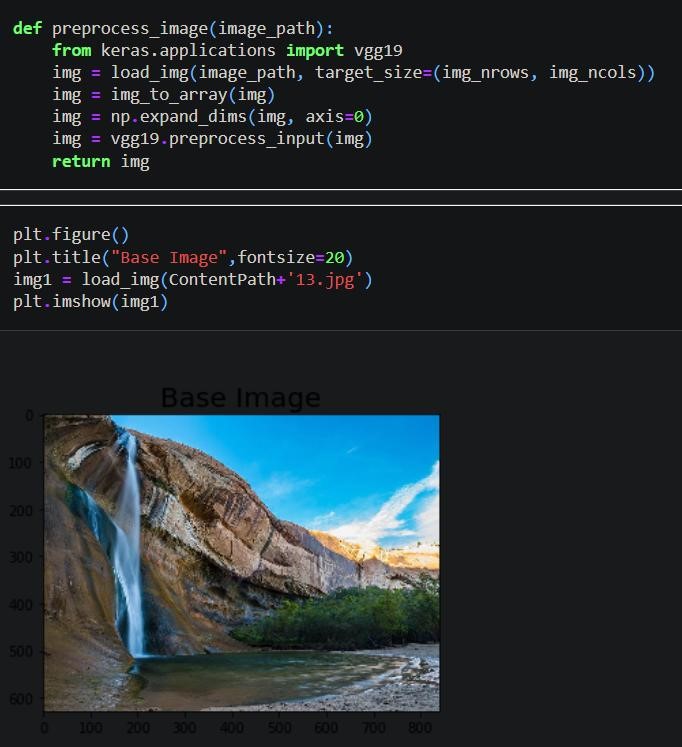
**Algorithm:**

1. Obtain the actual or base image.
2. Obtain the style image.
3. Read the pixels of the base image.
4. Generate a statistical model of the pixels and their colour, depth and intensities.
5. Remove each pixel of the actual image and regenerate the same with the pixels of the style image.
6. The image matrix and pixel statistics helps the newer pixels of the style image to adjust in the exact places and do the needful.
7. Thus the final image will be obtained with the imposed style.

**Importing the necessary packages:**



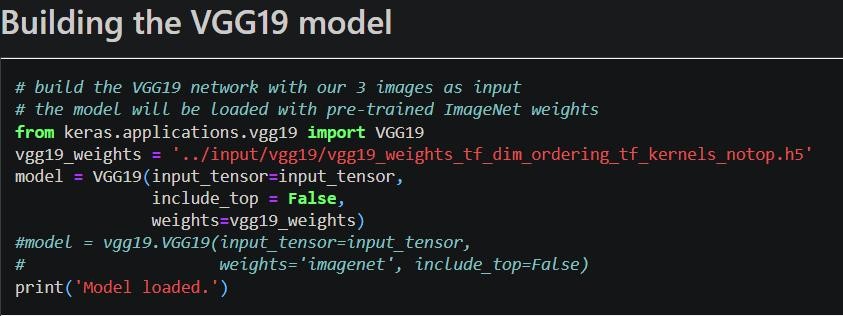
**BASE IMAGE:**

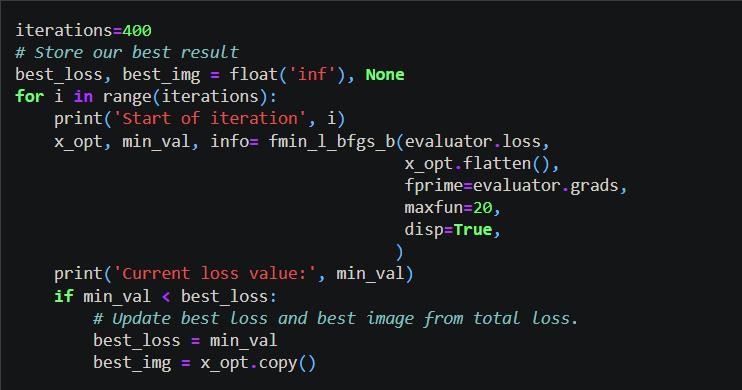
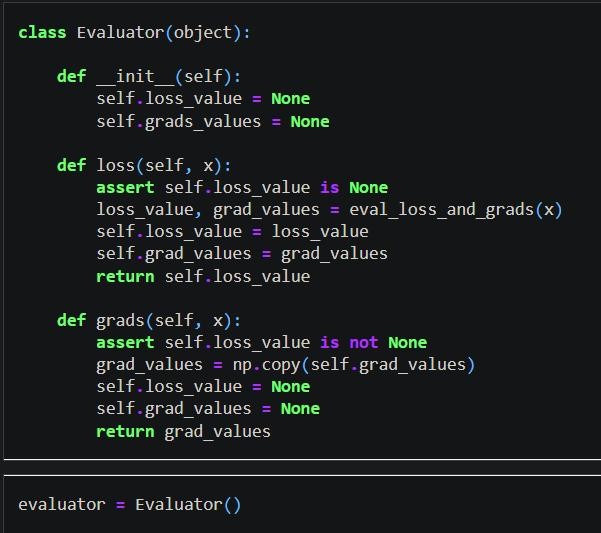
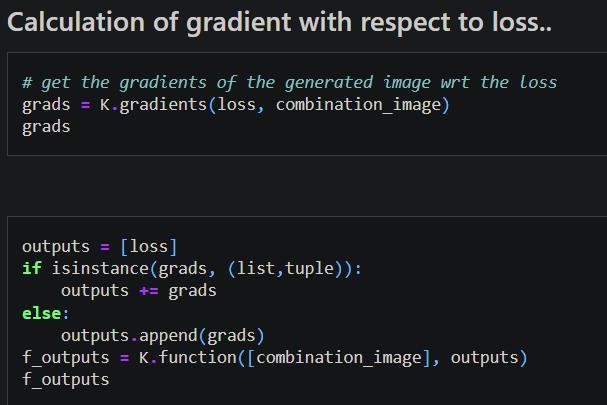
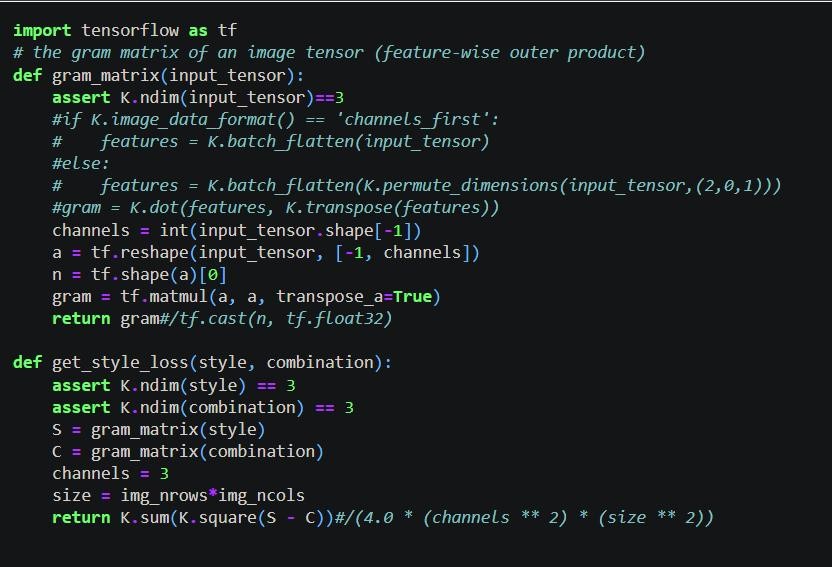
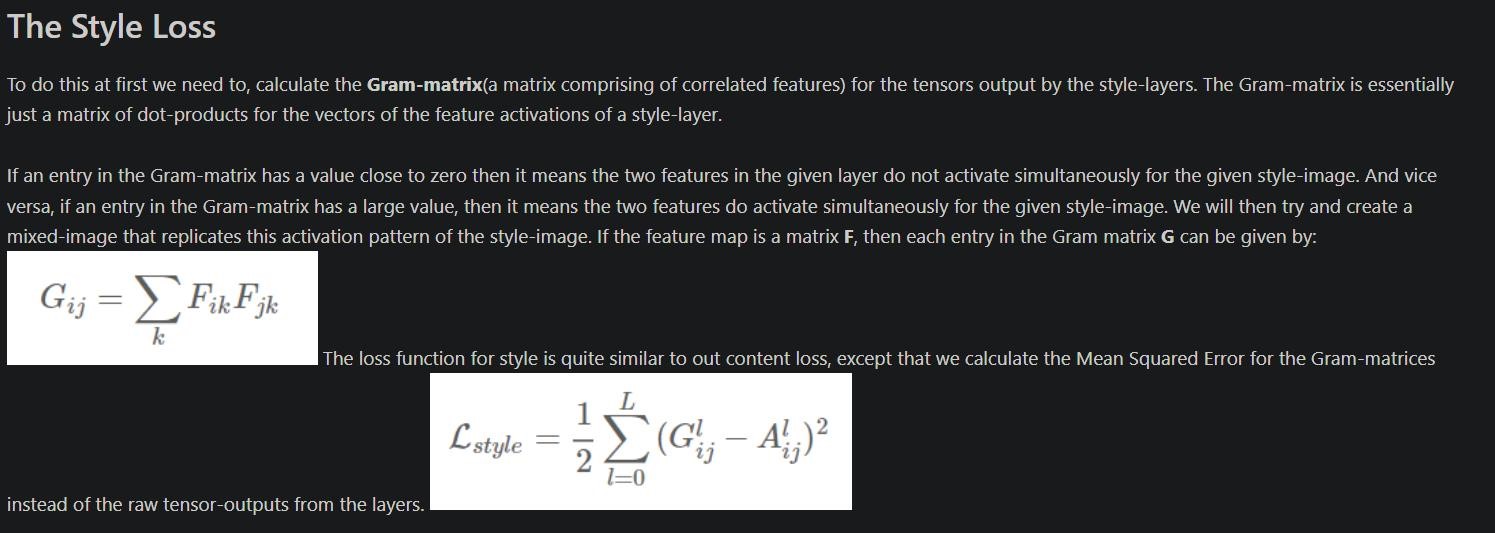
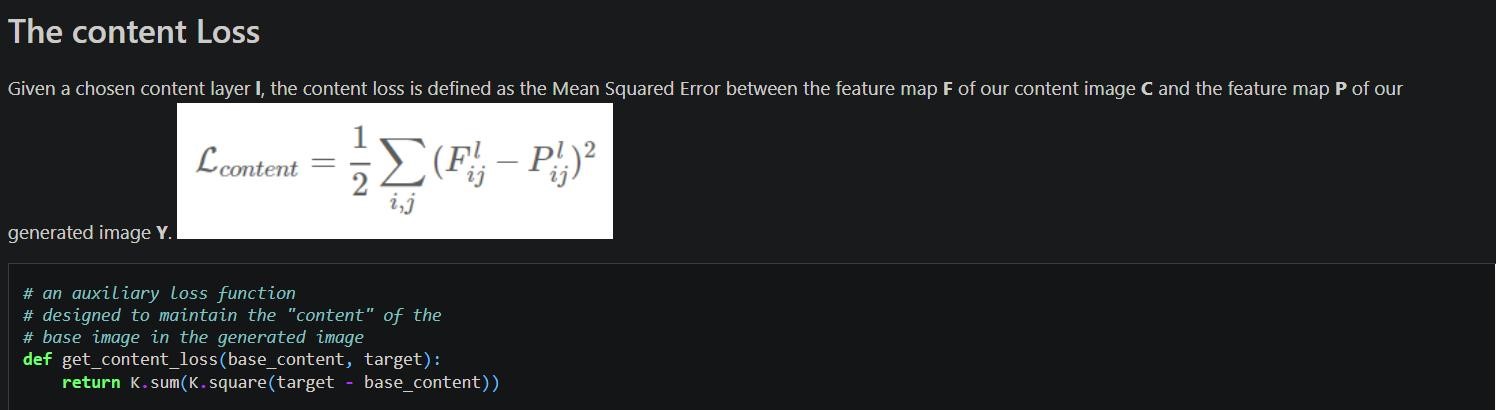
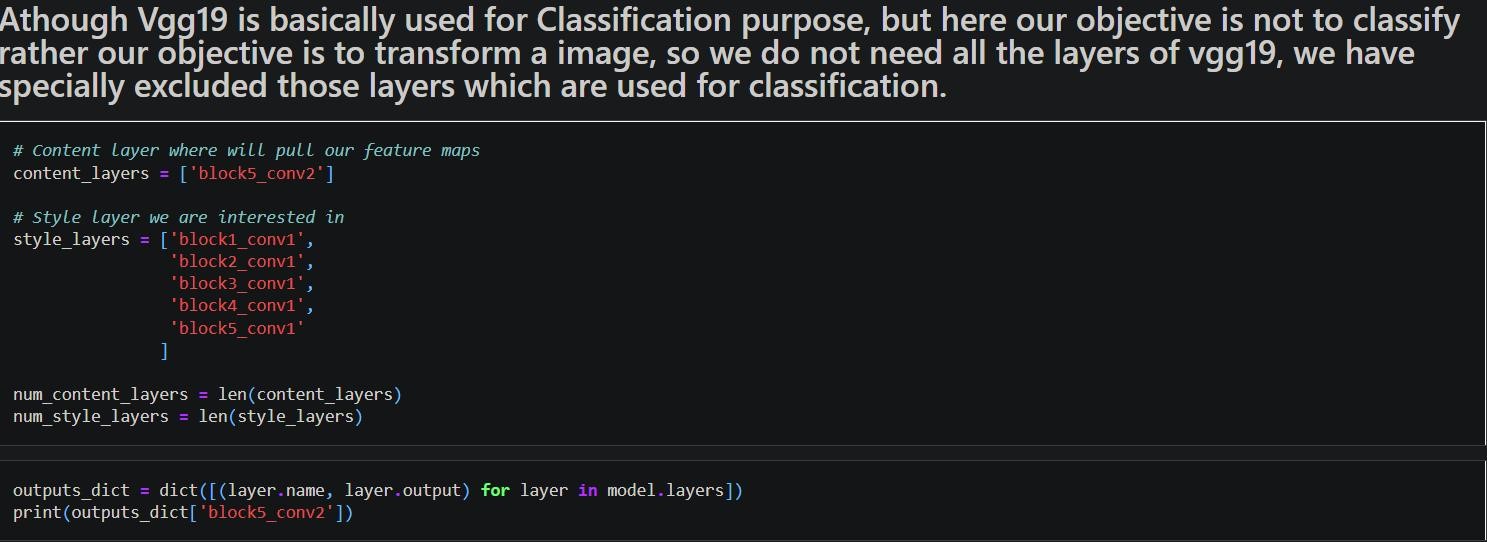


**STYLE IMAGE:**



**ALGORITHMS IN BETWEEN:**





**FINAL IMAGE:**



**Result:**

The base image is thus style transferred using the style image and hence the resultant image is obtained.

Artificial Intelligence Lab

Experiment 6

*Aim:-* Implementation of uncertain methods for an application (Fuzzy logic/ Dempster Shafer Theory)

*Code:-* C++

#include <iostream>

#include <cmath>

#include <cstring>

const double cdMinimumPrice =0;

const double cdMaximumPrice =70;

using namespace std;

class CFuzzyFunction

{

protected :

double dLeft, dRight;

char cType;

char\* sName;

public:

CFuzzyFunction(){};

virtual ~CFuzzyFunction(){ delete [] sName; sName=NULL;}

virtual void

setInterval(double l,

double r)

{dLeft=l; dRight=r;}

virtual void

setMiddle( double dL=0,

double dR=0)=0;

virtual void

setType(char c)

{ cType=c;}

virtual void

setName(const char\* s)

{

sName = new char[strlen(s)+1];

strcpy(sName,s);

}

bool

isDotInInterval(double t)

{

if((t>=dLeft)&&(t<=dRight)) return true; else return false;

}

char getType(void)const{ return cType;}

void

getName() const

{

cout<<sName<<endl;

}

virtual double getValue(double t)=0;

};

class CTriangle : public CFuzzyFunction

{

private:

double dMiddle;

public:

void

setMiddle(double dL, double dR)

{

dMiddle=dL;

}

double

getValue(double t)

{

if(t<=dLeft)

return 0;

else if(t<dMiddle)

return (t-dLeft)/(dMiddle-dLeft);

else if(t==dMiddle)

return 1.0;

else if(t<dRight)

return (dRight-t)/(dRight-dMiddle);

else

return 0;

}

};

class CTrapezoid : public CFuzzyFunction

{

private:

double dLeftMiddle, dRightMiddle;

public:

void

setMiddle(double dL, double dR)

{

dLeftMiddle=dL; dRightMiddle=dR;

}

double

getValue(double t)

{

if(t<=dLeft)

return 0;

else if(t<dLeftMiddle)

return (t-dLeft)/(dLeftMiddle-dLeft);

else if(t<=dRightMiddle)

return 1.0;

else if(t<dRight)

return (dRight-t)/(dRight-dRightMiddle);

else

return 0;

}

};

int

main(void)

{

CFuzzyFunction \*FuzzySet[3];

FuzzySet[0] = new CTrapezoid;

FuzzySet[1] = new CTriangle;

FuzzySet[2] = new CTrapezoid;

FuzzySet[0]->setInterval(-5,30);

FuzzySet[0]->setMiddle(0,20);

FuzzySet[0]->setType('r');

FuzzySet[0]->setName("low\_price");

FuzzySet[1]->setInterval(25,45);

FuzzySet[1]->setMiddle(35,35);

FuzzySet[1]->setType('t');

FuzzySet[1]->setName("good\_price");

FuzzySet[2]->setInterval(40,75);

FuzzySet[2]->setMiddle(50,70);

FuzzySet[2]->setType('r');

FuzzySet[2]->setName("to\_expensive");

double dValue;

do

{

cout<<"\nImput the value->"; cin>>dValue;

if(dValue<cdMinimumPrice) continue;

if(dValue>cdMaximumPrice) continue;

for(int i=0; i<3; i++)

{

cout<<"\nThe dot="<<dValue<<endl;

if(FuzzySet[i]->isDotInInterval(dValue))

cout<<"In the interval";

else

cout<<"Not in the interval";

cout<<endl;

cout<<"The name of function is"<<endl;

FuzzySet[i]->getName();

cout<<"and the membership is=";

cout<<FuzzySet[i]->getValue(dValue);

}

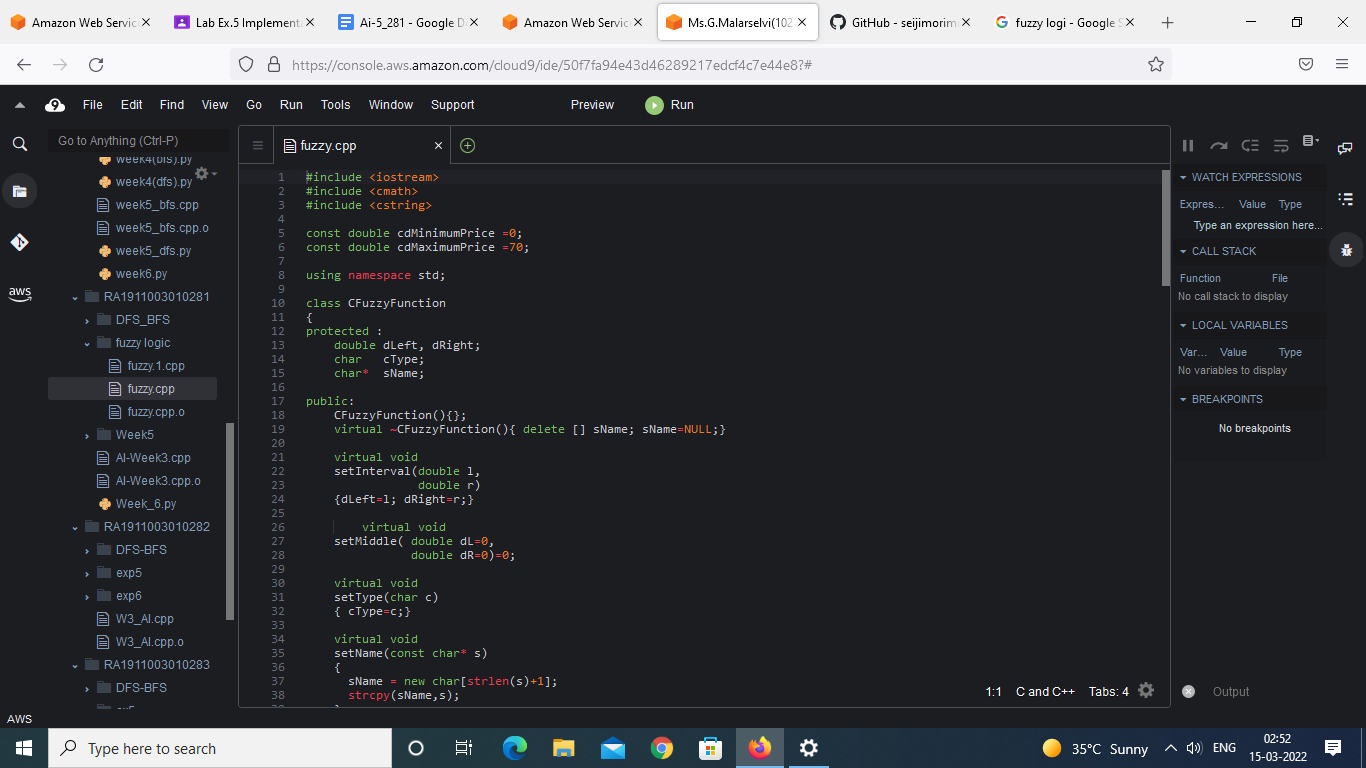
}

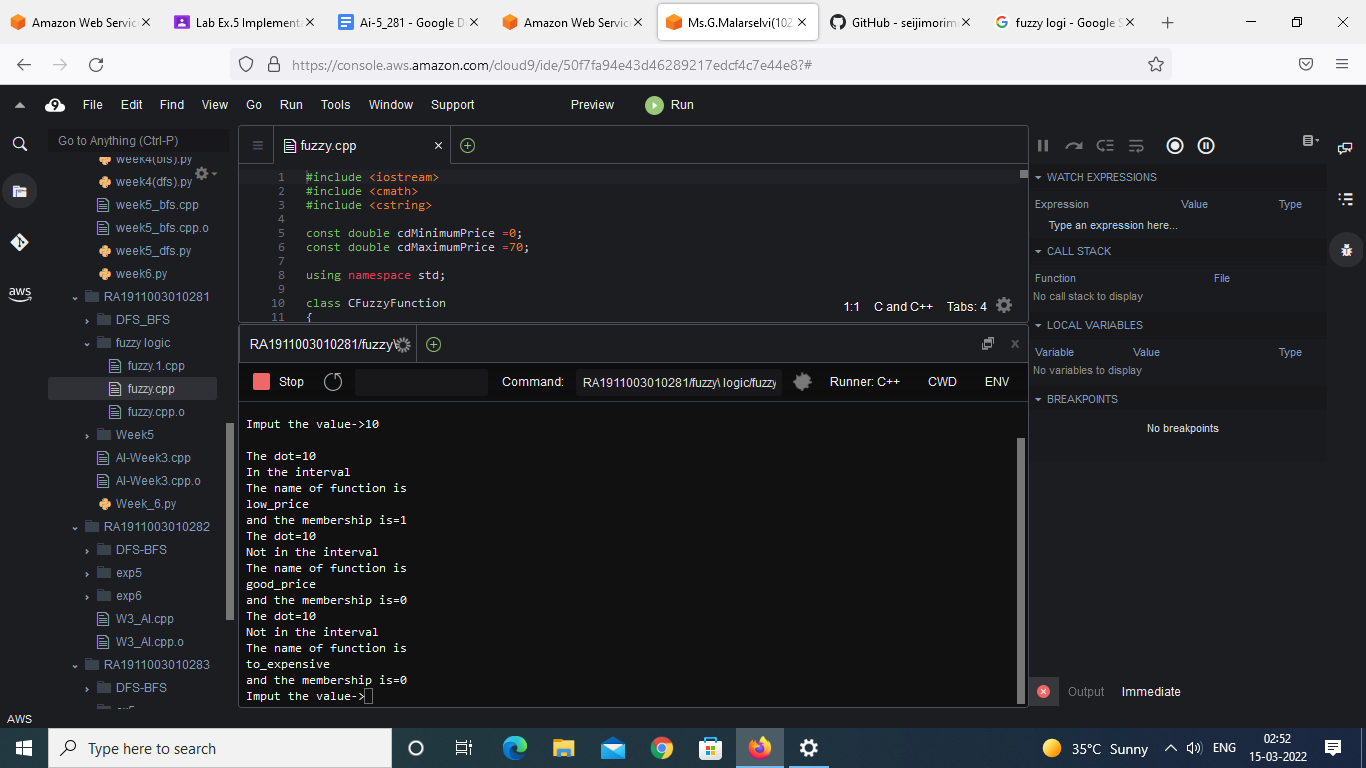
while(true);

return EXIT\_SUCCESS;

}

*Sample Output:-*

**

**