CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGYDEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Computer Science & Engineering

NAME: PARTH NITESHKUMAR PATEL

ID: 19DCS098

SUBJECT: ARTIFICIAL INTELLIGENCE

CODE: CS 341

PRACTICAL-1

Write Programs to demonstrate knowledge of Prolog Basics.

PRACTICAL-1.1

AIM:

Write a program in prolog to implement simple facts and Queries.

PROGRAM CODE:

- **If anything X is animal if it is dog**

```
dog(fido).
dog(scooby).
dog(ghost).
dog(max).
cat(pablo).
cat(felix).
cat(henry).
cat(jane).

animal(X):-dog(X).
```

OUTPUT:

```
?-
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1_animals.pl compiled 0.00 sec, 9 clauses
?- dog(fido).
true.

?- dog(ghost).
true.

?- cat(pablo).
true.

?- animal(fido).
true.

?- animal(pablo).
false.
```

- **Lion, Tiger and goat are animals. Lion and tiger are carnivores but goat is not.**

PROGRAM CODE:

```
lion(simba).
tiger(rudra).
goat(ramesh).

animal(X):-
    lion(X);
    tiger(X);
    goat(X).

carnivores(X):-
    animal(X),lion(X);
    animal(X),tiger(X).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1
?- lion(X).
X = simba.

?- tiger(X).
X = rudra.

?- goat(X).
X = ramesh.

?- animal(simba).
true .

?- carnivores(simba).
true .

?- animal(rudra).
true .

?- carnivores(rudra).
true .

?- animal(ramesh).
true.

?-
|    carnivores(ramesh).
false
```

- **To show the use of unknown variable.**

PROGRAM CODE:

```
animal(mammel,lion,carnivore,king).
animal(mammel,tiger,carnivore,stripes).
animal(mammel,zebra,herbivores,stripes).
animal(mammel,elephant,herbivores,tusks).
animal(mammel,bear,omnivores,black).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1_
pl compiled 0.00 sec, 5 clauses
?-
|    animal(mammel,X,_,_).
X = lion ;
X = tiger .

?- animal(mammel,X,carnivore,_).
X = lion ;
X = tiger.

?- animal(mammel,X,_,_).
X = lion ;
X = tiger ;
X = zebra ;
X = elephant ;
X = bear.

?- animal(mammel,_,_,stripes).
true ;
true.

?- animal(mammel,X,_,stripes).
X = tiger ;
X = zebra.
```

- **COUPLE PROBLEM**

PROGRAM CODE:

```
%FACTS
person(harry,male).
person(jenna,female).
person(tom,male).
person(cortney,female).
person(sam,male).
person(michelle,female).
person(robin,male).
person(jenny,female).
person(sunny,male).
person(cherry,female).

%Rule

couple(X,Y):- person(X,male),person(Y,female).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1_
?- couple(X,Y).
X = harry,
Y = jenna ;
X = harry,
Y = cortney ;
X = harry,
Y = michelle ;
X = harry,
Y = jenny ;
X = harry,
Y = cherry ;
X = tom,
Y = jenna ;
X = tom,
Y = cortney ;
X = tom,
Y = michelle .

?- couple(harry,jenna).
true .

?- couple(harry,tom).
false.

?- couple(michelle,cortney).
false.
```

- **Person X teaches subject Y and student Z is studying the subject having subject Y.**

PROGRAM CODE:

```
% teaches(X,Y): Person X teaches subject Y

teaches(ram,ai).
teaches(rahul,os).
teaches(narendra,android).
teaches(amit,networks).
teaches(paul,hss).

% studies(Z,Y): Student Z studies subject Y

studies(parth,android).
studies(parth,ai).
studies(roshan,networks).
studies(arthur,hss).
studies(krishna,ai).
studies(het,os).
studies(jack,hss).
studies(henry,android).

% Rule : professor(X,Y,Z): X is professor of Z if X teaches Y and Z
% studies the subject that X teaches.


professor(X,Y,Z):- teaches(X,Y),studies(Z,Y).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1
?- teaches(X,android).
X = narendra.

?- studies(Z,android).
Z = parth ;
Z = henry.

?- professor(narendra,android,parth).
true .

?- professor(X,os,Z).
X = rahul,
Z = het.

?- professor(X,hss,Z).
X = paul,
Z = arthur ;
X = paul,
Z = jack.
```

CONCLUSION:

- By performing the above practicals, we came to know about the basics of Prolog.

  We also learnt about the facts and queries in Prolog.

PRACTICAL-1.2

AIM:

Write a program in prolog to implement phone list which stores name, phone number and birthdays of friends and family members. Write a query to get a list of people whose birthdays are in the current month.

PROGRAM CODE:

```
phoneList(person(parth,patel),"8980145590",birthDate(day(06),month(04),year(2001))).

phoneList(person(narendra,shah),"1290986744",birthDate(day(15),month(08),year(1988))
).

phoneList(person(nimit,modi),"9978145590",birthDate(day(15),month(11),year(1999))).

phoneList(person(renu,patel),"2500997764",birthDate(day(28),month(02),year(2001))).

phoneList(person(mafat,patel),"8182144570",birthDate(day(01),month(03),year(1955))).

phoneList(person(vaani,patil),"1432265590",birthDate(day(19),month(12),year(2020))).

phoneList(person(rasam,ganguly),"4295155521",birthDate(day(16),month(06),year(1987))
).

phoneList(person(het,jaiswal),"9420707009",birthDate(day(01),month(01),year(1991))).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Pracitcal_1_2.pl compiled 0.00 sec, 8 clauses
?- phoneList(person(X,patel),Y,birthDate(day(Z),month(W),year(A))).
X = parth,
Y = "8980145590",
Z = 6,
W = 4,
A = 2001 ;
X = renu,
Y = "2500997764",
Z = 28,
W = 2,
A = 2001 ;
X = mafat,
Y = "8182144570",
Z = 1,
W = 3,
A = 1955.

?- phoneList(person(parth,patel),"8980145590",birthDate(day(06),month(04),year(2001))).
true .

% e:/5_AI_CS341/19DCS098_Prolog/Pracitcal_1_2.pl compiled 0.00 sec, -2 clauses
?- phoneList(person(X,Y),_,birthDate(day(_),month(04),year(_))).
X = parth,
Y = patel ;
false.
```

CONCLUSION:

- By performing the above practical, we learnt about how to write facts, queries and rules in prolog.
- We also learnt about how to execute a program in prolog.

PRACTICE PROBLEM

AIM:

The Family Problem

PROGRAM CODE:

```
%male
male(jim).
male(bob).
male(tom).
male(peter).

%female
female(penny).
female(liza).
female(anna).
female(alisa).

%parent(X,Y)
parent(penny,bob).
parent(tom,liza).
parent(tom,bob).
parent(bob,anna).
parent(bob,alisa).
parent(penny,jim).
parent(bob,peter).
parent(peter,jim).

%rules
mother(X,Y):- parent(X,Y),female(X).
father(X,Y):- parent(X,Y),male(X).
hasChild(X):- parent(X,_).
sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):- parent(Z,X),parent(Z,Y),male(X),X\==Y.
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Parent.pl
?- parent(X,jim).
X = penny ;
X = peter.

?- mother(X,Y).
X = penny,
Y = bob ;
X = penny,
Y = jim ;
false.

?- hasChild(bob).
true .

?- brother(X,Y).
X = bob,
Y = jim ;
X = bob,
Y = liza ;
X = jim,
Y = bob ;
X = peter,
Y = anna ;
X = peter,
Y = alisa ;
false.
```

CONCLUSION:

- By performing the above practical, we learnt about how to write facts, queries and rules in prolog.
- We also learnt about how to execute a program in prolog.

PRACTICAL-1.3

AIM:

Write predicates one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

PROGRAM CODE:

```
convertFtoC(Celsius,Fahrenheit):-
    Celsius is  ((Fahrenheit-32)*5)/9.


convertCtoF(Celsius,Fahrenheit):-
    Fahrenheit is ((Celsius*9)/5)+32.

freezing(F):-
    F=<32.
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1_3.
00 sec, 2 clauses
.

?- convertFtoC(Celsius,90).
Celsius = 32.2222222222222.

?- convertCtoF(32.222,Fahrenheit).
Fahrenheit = 89.9996.

?- freezing(0).
true.

?- freezing(50).
false.
```

CONCLUSION:

- By performing the above practical, we learnt about how the use of neck symbol.
- We also learnt about the basic procedure to create a prolog program.

PRACTICAL-1.4

AIM:

Demonstrate Backtracking in Prolog.

PROGRAM CODE:

```
branch(a,b).
branch(a,c).
branch(c,d).
branch(c,e).

path(X,X).
path(X,Y):-
    branch(X,Z),path(Z,Y).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Practical_1_4.pl
?- path(a,d).
true .

?- trace.
true.
```

```
[trace]  ?- path(a,e).
   Call: (10) path(a, e) ? creep
   Call: (11) branch(a, _9508) ? creep
   Exit: (11) branch(a, b) ? creep
   Call: (11) path(b, e) ? creep
   Call: (12) branch(b, _9640) ? creep
   Fail: (12) branch(b, _9684) ? creep
   Fail: (11) path(b, e) ? creep
   Redo: (11) branch(a, _9772) ? creep
   Exit: (11) branch(a, c) ? creep
   Call: (11) path(c, e) ? creep
   Call: (12) branch(c, _9904) ? creep
   Exit: (12) branch(c, d) ? creep
   Call: (12) path(d, e) ? creep
   Call: (13) branch(d, _10036) ? creep
   Fail: (13) branch(d, _10080) ? creep
   Fail: (12) path(d, e) ? creep
   Redo: (12) branch(c, _10168) ? creep
   Exit: (12) branch(c, e) ? creep
   Call: (12) path(e, e) ? creep
   Exit: (12) path(e, e) ? creep
   Exit: (11) path(c, e) ? creep
   Exit: (10) path(a, e) ? creep
true .
```

CONCLUSION:

By performing the above practical, we learnt about the concept of backtracking in prolog.

EXTRA PRACTICAL:

AIM:

Problem of checknumber.

PROGRAM CODE:

```
checknumber(X,Y):-
    A is  ((X+Y)/2),

    B is sqrt(X*Y),

    C is max(X,Y),

   write("X+Y/2 : "),
   write(A),nl,
   write("sqrt(X,Y): "),
   write(B),nl,
   write("max(X,Y) : "),
   write(C),nl.
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/CheckNumber
 sec, 0 clauses
?- checknumber(10,21).
X+Y/2 : 15.5
sqrt(X,Y): 14.491376746189438
max(X,Y) : 21
true.
```

CONCLUSION:

By performing the above practical, we learnt the use of write() in Prolog.

PRACTICAL-2.1

AIM:

Write a program to display Fibonacci series in prolog

PROGRAM CODE:

```
fibonacci(0,0).
fibonacci(1,1).

fibonacci(F,N):-

    N>1,

    N1 is N-1,

    N2 is N-2,

    fibonacci(F1,N1),
    fibonacci(F2,N2),

    F is F1+F2,

     write(F),
     write(" ").
```

OUTPUT:

```
?- fibonacci(F,3).
1 2
F = 2
```

CONCLUSION:

By performing the above practical, we learnt about the logic behind the Fibonacci series in Prolog.

PRACTICAL-2.2

AIM:

Write a program to display Factorial in prolog

PROGRAM CODE:

```
factorial(0,1).

factorial(N,F):-

    N>0,
    N1 is N-1,
    factorial(N1,F1),
    F is N*F1.
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Pracitcal_2_2.
?- factorial(5,X).
X = 120 .

?- factorial(10,X).
X = 3628800 .
```

CONCLUSION:

- By performing the above practical, we learnt the basic syntax of prolog and also about the arithmetic operators in Prolog.

PRACTICAL-3

AIM:

Write a prolog program for medical diagnosis system of childhood diseases.

PROGRAM CODE:

```
go :-
write('What is the patient''s name? '),
read(Patient),get_single_char(Code),
hypothesis(Patient,Disease),
write_list([Patient,', probably has ',Disease,'.']),nl.


go :-
write('Sorry, I don''t seem to be able to'),nl,
write('diagnose the disease.'),nl.


symptom(Patient,fever) :-
verify(Patient," have a fever (y/n) ?").
symptom(Patient,rash) :-
verify(Patient," have a rash (y/n) ?").
symptom(Patient,headache) :-
verify(Patient," have a headache (y/n) ?").
symptom(Patient,runny_nose) :-
verify(Patient," have a runny_nose (y/n) ?").
symptom(Patient,conjunctivitis) :-
```

```
verify(Patient," have a conjunctivitis (y/n) ?").

symptom(Patient,cough) :-

verify(Patient," have a cough (y/n) ?").

symptom(Patient,body_ache) :-

verify(Patient," have a body_ache (y/n) ?").

symptom(Patient,chills) :-

verify(Patient," have a chills (y/n) ?").

symptom(Patient,sore_throat) :-

verify(Patient," have a sore_throat (y/n) ?").

symptom(Patient,sneezing) :-

verify(Patient," have a sneezing (y/n) ?").

symptom(Patient,swollen_glands) :-

verify(Patient," have a swollen_glands (y/n) ?").


ask(Patient,Question) :-

        write(Patient),write(', do you'),write(Question),

        read(N),

        ( (N == yes ; N == y)

    ->

     assert(yes(Question)) ;

     assert(no(Question)), fail).


:- dynamic yes/1,no/1.
```

```
verify(P,S) :-
   (yes(S) -> true ;
    (no(S) -> fail ;
     ask(P,S))).

undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.




hypothesis(Patient,german_measles) :-
symptom(Patient,fever),
symptom(Patient,headache),
symptom(Patient,runny_nose),
symptom(Patient,rash).

hypothesis(Patient,common_cold) :-
symptom(Patient,headache),
symptom(Patient,sneezing),
symptom(Patient,sore_throat),
symptom(Patient,runny_nose),
symptom(Patient,chills).
```

```
hypothesis(Patient,measles) :-
symptom(Patient,cough),
symptom(Patient,sneezing),
symptom(Patient,runny_nose).

hypothesis(Patient,flu) :-
symptom(Patient,fever),
symptom(Patient,headache),
symptom(Patient,body_ache),
symptom(Patient,conjunctivitis),
symptom(Patient,chills),
symptom(Patient,sore_throat),
symptom(Patient,runny_nose),
symptom(Patient,cough).


hypothesis(Patient,mumps) :-
symptom(Patient,fever),
symptom(Patient,swollen_glands).

hypothesis(Patient,chicken_pox) :-
symptom(Patient,fever),
symptom(Patient,chills),
```

```
symptom(Patient,body_ache),

symptom(Patient,rash).


write_list([]).

write_list([Term| Terms]) :-

write(Term),

write_list(Terms).


response(Reply) :-

get_single_char(Code),

put_code(Code), nl,

char_code(Reply, Code).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Pracitcal_3.pl c
 sec, 27 clauses
?- go.
What is the patient's name? parth.
parth, do you have a fever (y/n) ?y.
parth, do you have a headache (y/n) ?|: n.
parth, do you have a cough (y/n) ?|: y.
parth, do you have a sneezing (y/n) ?|: n.
parth, do you have a swollen_glands (y/n) ?|: y.
parth, probably has mumps.
true .
```

CONCLUSION:

- By performing the above practical, we can conclude that prolog is most useful in the areas related to AI research, such as problem solving, planning or naural language interpretation.

PRACTICAL-4

AIM:

Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin.

PROGRAM CODE:

```
male(tom).

male(jerry).

male(harry).

male(sunny).

male(balmar).

female(anne).

female(jenna).

female(arthur).

female(jake).

female(granny).

male(barry).

male(goffy).

female(goffy).

parent(sunny,jerry).
```

```
parent(sunny,harry).

parent(sunny,anne).

parent(jenna,jerry).

parent(jenna,harry).

parent(jenna,anne).

parent(balmar,sunny).

parent(jake,sunny).

parent(arthur,jenna).

parent(arthur,tom).

parent(granny,arthur).

parent(jerry,barry).

male(milan).

male(tino).

parent(rahul,milan).

parent(rahul,tino).

indian(anne).

indian(X) :- ancestor(X,anne).

indian(X) :- ancestor(anne,X).

relation(X,Y) :- ancestor(A,X), ancestor(A,Y).

father(X,Y) :- male(X),parent(X,Y).

father(goffy, _) :- male(goffy).

mother(X,Y) :- female(X),parent(X,Y).

son(X,Y) :- male(X),parent(Y,X).

daughter(X,Y) :- female(X),parent(Y,X).
```

grandfather(X,Y) :- male(X),parent(X,Somebody),parent(Somebody,Y).

aunt(X,Y) :- female(X),sister(X,Mom),mother(Mom,Y).

aunt(X,Y) :- female(X),sister(X,Dad),father(Dad,Y).

sister(X,Y) :- female(X),parent(Par,X),parent(Par,Y), X \= Y.

uncle(X,Y) :- brother(X,Par),parent(Par,Y).

cousin(X,Y) :- uncle(Unc , X),father(Unc,Y).

ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(X,Somebody),ancestor(Somebody,Y).

brother(X,Y) :- male(X),parent(Somebody,X),parent(Somebody,Y), X \= Y.

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/Pracitcal_4.
 sec, 48 clauses
?- father(sunny,jerry).
true .

?- mother(jenna,harry).
true.
```

CONCLUSION:

- By performing the above practical, we came to know that Prolog is a declarative programming language where logic is expressed in terms of relations.

PRACTICAL-5

AIM:

Write a program to perform following operations on lists in prolog.

PROGRAM CODE:

- **Create a list in Prolog:**

```
?- X=parth,Y=tom,Z=jerry,
|    W=[alpha,beta],
|    write('List : '),
|    write([X,Y,Z,W]),
|    nl.
List : [parth,tom,jerry,[alpha,beta]]
X = parth,
Y = tom,
Z = jerry,
W = [alpha, beta].
```

- **Cons Notation:**

```
?- write([parth | [tom,jerry]]),nl.
[parth,tom,jerry]
true.
```

```
?- write([tom,jerry | [parth]]),nl.
[tom,jerry,parth]
true.
```

```
?- write([tom,jerry,parth | []]),nl.
[tom,jerry,parth]
true.
```

```
?- L=[parth,tom,jerry],
|    write([disney | L]),nl.
[disney,parth,tom,jerry]
L = [parth, tom, jerry].
```

- **Membership in List:**

```
?- member(parth,[parth,tom,jerry]),nl.

true
```

```
?- member([1,2,3],[parth,tom,[1,2,3],jerry]).
true .
```

```
?- member(X,[parth,tom,jerry]).
X = parth ;
X = tom ;
X = jerry.
```

- **Length:**

```
?- length([[1,2,3],[a,b,c],4,5,6],L).
L = 5.

?- length([],L).
L = 0.

?- N is 4,
|    length([parth,tom,jerry,india],N).
N = 4.
```

- **Reverse:**

```
?- reverse([1,2,3,4],L).
L = [4, 3, 2, 1].
```

```
?- reverse(L,[1,2,3,4]).
L = [4, 3, 2, 1]
```

```
?- reverse([[parth,tom],[parth,jerry],[1,2,3],[a,b,c]],L).
L = [[a, b, c], [1, 2, 3], [parth, jerry], [parth, tom]].
```

- **Append:**

```
?- append([parth],[tom,jerry],L).
L = [parth, tom, jerry].
```

- **Permutation**:

```
permutation(X1,Y1):-
    msort(X1,Sorted),
    msort(Y1,Sorted).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/permutation.
 sec, 1 clauses
?- permutation([1,2],[X,Y]).
X = 1,
Y = 2.

?- permutation([parth,tom,jerry],[X,Y,Z]).
X = jerry,
Y = parth,
Z = tom.
```

CONCLUSION:

- By performing the above practical, we learnt about the basic concept of lists in PROLOG.

PRACTICAL-6

AIM:

Write a program to demonstrate cut and fail in prolog.

PROGRAM CODE:

Program-1:

```
max(X,Y,X):-   X>=Y,!.

max(X,Y,Y):- X<Y.
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/cut.pl
clauses
?- trace.
true.

[trace]  ?- max(100,40,Z).
   Call: (10) max(100, 40, _19116) ? creep
   Call: (11) 100>=40 ? creep
   Exit: (11) 100>=40 ? creep
   Exit: (10) max(100, 40, 100) ? creep
Z = 100.

[trace]  ?- max(40,100,Z).
   Call: (10) max(40, 100, _20488) ? creep
   Call: (11) 40>=100 ? creep
   Fail: (11) 40>=100 ? creep
   Redo: (10) max(40, 100, _20488) ? creep
   Call: (11) 40<100 ? creep
   Exit: (11) 40<100 ? creep
   Exit: (10) max(40, 100, 100) ? creep
Z = 100.
```

Program-2:

```
animal(cobra).

animal(python).

animal(blackMamba).


snake(cobra).

snake(python).

snake(blackMamba).


likes(raj,X):- snake(X),!,fail.

likes(raj,X):- animal(X).
```

OUTPUT:

```
% e:/5_AI_CS341/19DCS098_Prolog/cutFail.pl
, -2 clauses
?- likes(raj,tiger).
true.

?- likes(raj,cobra).
false.
```

```
[trace]  ?- likes(raj,tiger).
   Call: (10) likes(raj, tiger) ? creep
   Call: (11) snake(tiger) ? creep
   Fail: (11) snake(tiger) ? creep
   Redo: (10) likes(raj, tiger) ? creep
   Call: (11) animal(tiger) ? creep
   Exit: (11) animal(tiger) ? creep
   Exit: (10) likes(raj, tiger) ? creep
true.
```

```
[trace]  ?- likes(raj,cobra).
   Call: (10) likes(raj, cobra) ? creep
   Call: (11) snake(cobra) ? creep
   Exit: (11) snake(cobra) ? creep
   Call: (11) fail ? creep
   Fail: (11) fail ? creep
   Fail: (10) likes(raj, cobra) ? creep
false.
```

CONCLUSION:

By performing the above practical, we learned about cut and fail.

CUT:

- Represented by !.
- It always succeeds, but cannot be backtracked. It is best used to prevent unwanted backtracking,

fail is a special symbol that will immediately fail when Prolog encounters it as a goal

PRACTICAL-7.1

AIM:

Design Depth First Search Tree and Breadth First Search Tree for Water-Jug Problem in python

PROGRAM CODE:

BFS APPROACH:

```
#BFS APPROACH
print("--------------------------------------------------------------------------------")
print("SOLUTION OF WATER JUG PROBLEM WITH BFS APPROACH")
print("--------------------------------------------------------------------------------")

#INPUT FOR JUG-1
capacity_of_X = int ( input ( "ENTER THE MAXIMUM CAPACITY OF JUG-1 : " ))
print("--------------------------------------------------------------------------------")
#INPUT FOR JUG-2
capacity_of_Y = int ( input ( "ENTER THE MAXIMUM CAPACITY OF JUG-2 : " ))
print("--------------------------------------------------------------------------------")
#INPUT FOR GOAL
end = int ( input ( "ENTER THE DESIRED VOLUME (GOAL) : " ))
print("--------------------------------------------------------------------------------")

#MAIN LOGIC

def bfs(begin, end, capacity_of_X, capacity_of_Y):
```

```
traversal_path = []
front = []
front.append(begin)
visited = []

while ( not ( not front)):
    current = front.pop()
    x = current[ 0 ]
    y = current[ 1 ]
    traversal_path.append(current)
    if x == end or y == end:
        print ( "PATH FOUND" )
        print("-----------------------------------------------------------------------------------")
        return traversal_path
    # RULE 1
    if current[ 0 ] < capacity_of_X and ([capacity_of_X, current[ 1 ]] not in visited):
        front.append([capacity_of_X, current[ 1 ]])
        visited.append([capacity_of_X, current[ 1 ]])
    # RULE 2
    if current[ 1 ] < capacity_of_Y and ([current[ 0 ], capacity_of_Y] not in visited):
        front.append([current[ 0 ], capacity_of_Y])
        visited.append([current[ 0 ], capacity_of_Y])
    # RULE 3
    if current[ 0 ] > capacity_of_X and ([ 0 , current[ 1 ]] not in visited):
```

```
            front.append([ 0 , current[ 1 ]])
            visited.append([ 0 , current[ 1 ]])
        # RULE 4
        if current[ 1 ] > capacity_of_Y and ([capacity_of_X, 0 ] not in visited):
            front.append([capacity_of_X, 0 ])
            visited.append([capacity_of_X, 0 ])
        # RULE 5
        #(x, y) -> (min(x + y, capacity_of_X), max(0, x + y - capacity_of_X))
        if current[ 1 ] > 0 and ([ min (x + y, capacity_of_X), max ( 0 , x + y - capacity_of_X)]
not in visited):
            front.append([ min (x + y, capacity_of_X), max ( 0 , x + y - capacity_of_X)])
            visited.append([ min (x + y, capacity_of_X), max ( 0 , x + y - capacity_of_X)])
        if current[ 0 ] > 0 and ([ max ( 0 , x + y - capacity_of_Y), min (x + y, capacity_of_Y)]
not in visited):
            front.append([ max ( 0 , x + y - capacity_of_Y), min (x + y,capacity_of_Y)])
            visited.append([ max ( 0 , x + y - capacity_of_Y), min (x + y,capacity_of_Y)])
    return "PATH NOT FOUND"


def hcf(a,b):
    if a==0:
        return b
    return hcf(b%a,a)


begin = [ 0 , 0 ]
```

```
if end % hcf(capacity_of_X,capacity_of_Y) == 0 :

    print (bfs(begin, end, capacity_of_X, capacity_of_Y))

else :

    print("--------------------------------------------------------------------------------")

    print ( " CANNOT FIND THE SOLUTION FOR THE GIVEN PROBLEM/STATE" )

print("--------------------------------------------------------------------------------")

print("PARTH PATEL\n19DCS098")

print("--------------------------------------------------------------------------------")
```

OUTPUT:

```
-----------------------------------------------------------------------
SOLUTION OF WATER JUG PROBLEM WITH BFS APPROACH
-----------------------------------------------------------------------
ENTER THE MAXIMUM CAPACITY OF JUG-1 : 5
-----------------------------------------------------------------------
ENTER THE MAXIMUM CAPACITY OF JUG-2 : 3
-----------------------------------------------------------------------
ENTER THE DESIRED VOLUME (GOAL) : 2
-----------------------------------------------------------------------
PATH FOUND
-----------------------------------------------------------------------
[[0, 0], [0, 3], [3, 0], [3, 3], [5, 1], [5, 3], [5, 0], [2, 3]]
-----------------------------------------------------------------------
PARTH PATEL
19DCS098
-----------------------------------------------------------------------
```

DFS APPROACH:

```
#DFS APPROACH


print("--------------------------------------------------------------------------------")
print("SOLUTION OF WATER JUG PROBLEM WITH DFS APPROACH")
print("--------------------------------------------------------------------------------")
#capacity = (10, 9, 8)

#INPUT FOR JUG-1
x = int ( input ( "ENTER THE MAXIMUM CAPACITY OF JUG-1 : " ))
print("--------------------------------------------------------------------------------")
#INPUT FOR JUG-2
y = int ( input ( "ENTER THE MAXIMUM CAPACITY OF JUG-2 : " ))
print("--------------------------------------------------------------------------------")
#INPUT FOR GOAL
z = int ( input ( "ENTER THE DESIRED VOLUME (GOAL) : " ))
print("--------------------------------------------------------------------------------")

#x = capacity[0]
#y = capacity[1]
#z = capacity[2]
```

```
data = {}


path = []


def find_states(state):
    a = state[0]
    b = state[1]
    c = state[2]
    if a == 6 and b == 6:
        path.append(state)
        return True
    if (a, b, c) in data:
        return False


    data[(a, b, c)] = 1


    if a > 0:
        if a + b <= y:
            if find_states((0, a + b, c)):
                path.append(state)
                return True
        else:
            if find_states((a - (y - b), y, c)):
                path.append(state)
```

```
            return True


        if a + c <= z:
            if find_states((0, b, a + c)):
                path.append(state)
                return True
        else:
            if find_states((a - (z - c), b, z)):
                path.append(state)
                return True


    if b > 0:
        if a + b <= x:
            if find_states((a + b, 0, c)):
                path.append(state)
                return True
    else:
        if find_states((x, b - (x - a), c)):
            path.append(state)
            return True
        if b + c <= z:
            if find_states((a, 0, b + c)):
                path.append(state)
                return True
```

```
        else:
            if find_states((a, b - (z - c), z)):
                path.append(state)
                return True

    if c > 0:
        if a + c <= x:
            if find_states((a + c, b, 0)):
                path.append(state)
                return True
        else:
            if find_states((x, b, c - (x - a))):
                path.append(state)
                return True

    if b + c <= y:
        if find_states((a, b + c, 0)):
            path.append(state)
            return True
    else:
        if find_states((a, y, c - (y - b))):
            path.append(state)
            return True
```

```
    return False



initial_state = (12, 0, 0)

find_states(initial_state)

path.reverse()

for i in path:

    print(i)


print("-------------------------------------------------------------------------------------")

print("PARTH PATEL\n19DCS098")

print("-------------------------------------------------------------------------------------")
```

OUTPUT:

```
SOLUTION OF WATER JUG PROBLEM WITH DFS APPROACH
-----------------------------------------------------------------
ENTER THE MAXIMUM CAPACITY OF JUG-1 : 10
-----------------------------------------------------------------
ENTER THE MAXIMUM CAPACITY OF JUG-2 : 9
-----------------------------------------------------------------
ENTER THE DESIRED VOLUME (GOAL) : 8
-----------------------------------------------------------------
(12, 0, 0)
(3, 9, 0)
(0, 9, 3)
(9, 0, 3)
(4, 0, 8)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(10, -2, 4)
(6, -2, 8)
(6, 0, 6)
(0, 6, 6)
(6, 6, 0)
-----------------------------------------------------------------
PARTH PATEL
19DCS098
-----------------------------------------------------------------
```

CONCLUSION:

- By performing the above practical, we learned how to solve water jug problem.

PRACTICAL-7.2

AIM:

Write a program to solve The N queen Problem using Hill Climbing with random Neighbor

PROGRAM CODE:

```
#include <iostream>
#include <math.h>
#include<ctime>

#define N 4
using namespace std;
void configureRandomly(int board[][N], int* state)
{
srand(time(0));
for (int i = 0; i < N; i++) {
state[i] = rand() % N;
board[state[i]][i] = 1;
}
}
```

```
void printBoard(int board[][N])

{

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

cout << " ";

for (int j = 0; j < N; j++) {

cout << board[i][j] << " ";

}

cout << "\n";

}

}


void printState(int* state)

{

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

cout << " " << state[i] << " ";

}

cout << endl;

}

bool compareStates(int* state1, int* state2)

{

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

if (state1[i] != state2[i]) {

return false;

}
```

```
}
return true;
}
void fill(int board[][N], int value)
{
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++) {
board[i][j] = value;
}
}
}
int calculateObjective(int board[][N], int* state)
{
int attacking = 0;

int row, col;
for (int i = 0; i < N; i++) {
row = state[i], col = i - 1;
while (col >= 0&& board[row][col] != 1) {
col--;
}
if (col >= 0&& board[row][col] == 1) {
attacking++;
}
```

```
row = state[i], col = i + 1;
while (col < N&& board[row][col] != 1) {
col++;
}
if (col < N&& board[row][col] == 1) {
attacking++;
}
row = state[i] - 1, col = i - 1;
while (col >= 0 && row >= 0&& board[row][col] != 1) {
col--;
row--;
}
if (col >= 0 && row >= 0&& board[row][col] == 1) {
attacking++;
}
row = state[i] + 1, col = i + 1;
while (col < N && row < N&& board[row][col] != 1) {
col++;
row++;
}
if (col < N && row < N&& board[row][col] == 1) {
attacking++;
}
row = state[i] + 1, col = i - 1;
```

```
while (col >= 0 && row < N&& board[row][col] != 1) {

col--;

row++;

}

if (col >= 0 && row < N&& board[row][col] == 1) {

attacking++;

}

row = state[i] - 1, col = i + 1;

while (col < N && row >= 0  && board[row][col] != 1) {

col++;

row--;

}

if (col < N && row >= 0&& board[row][col] == 1) {

attacking++;

}

}

return (int)(attacking / 2);

}

void generateBoard(int board[][N], int* state)

{

fill(board, 0);

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

board[state[i]][i] = 1;

}
```

```
}
void copyState(int* state1, int* state2)
{
for (int i = 0; i < N; i++) {
state1[i] = state2[i];
}
}
void getNeighbour(int board[][N], int* state)
{

int opBoard[N][N];
int opState[N];
copyState(opState, state);
generateBoard(opBoard, opState);
int opObjective = calculateObjective(opBoard, opState);
int NeighbourBoard[N][N];
int NeighbourState[N];
copyState(NeighbourState, state);
generateBoard(NeighbourBoard, NeighbourState);
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++) {
if (j != state[i]) {
NeighbourState[i] = j;
NeighbourBoard[NeighbourState[i]][i]= 1;
```

```
NeighbourBoard[state[i]][i]= 0;

int temp= calculateObjective(NeighbourBoard,NeighbourState);

if (temp <= opObjective) {

opObjective = temp;

copyState(opState,NeighbourState);

generateBoard(opBoard,opState);

}

NeighbourBoard[NeighbourState[i]][i]= 0;

NeighbourState[i] = state[i];

NeighbourBoard[state[i]][i] = 1;

}

}

}

copyState(state, opState);

fill(board, 0);

generateBoard(board, state);

}

void hillClimbing(int board[][N], int* state)

{

int neighbourBoard[N][N] = {};

int neighbourState[N];

copyState(neighbourState, state);

generateBoard(neighbourBoard,neighbourState);
```

```
do {
copyState(state, neighbourState);
generateBoard(board, state);
getNeighbour(neighbourBoard, neighbourState);
if (compareStates(state, neighbourState)) {
printBoard(board);
break;
}
else if (calculateObjective(board, state) ==
calculateObjective(neighbourBoard,neighbourState)) {
neighbourState[rand() % N]= rand() % N;
generateBoard(neighbourBoard,neighbourState);
}
}
 while (true);
}
int main()
{
int state[N] = {};
int board[N][N] = {};
configureRandomly(board, state);
hillClimbing(board, state);
 cout<<"PARTH PATEL\n19DCS098"<<endl;
return 0;
}
```

OUTPUT:

```
 0 0 1 0
 1 0 0 0
 0 0 0 1
 0 1 0 0
PARTH PATEL
19DCS098
```

CONCLUSION:

- By performing the above practical, we learnt the concept of hill climbing by solving N-Queen problem.

PRACTICAL-7.3

AIM:

Implement Travelling Salesman Problem using Simulated Annealing Algorithm in Python.

PROGRAM CODE:

```
#TSP USING SIMULATED ANNEALING

# Import libraries
import sys
import random
import copy
import numpy as np
# This class represent a state
class State:
    # Create a new state
    def __init__(self, route:[], distance:int=0):
        self.route = route
        self.distance = distance
    # Compare states
    def __eq__(self, other):
        for i in range(len(self.route)):
```

```
            if(self.route[i] != other.route[i]):
                return False
        return True
    # Sort states
    def __lt__(self, other):
         return self.distance < other.distance
    # Print a state
    def __repr__(self):
        return ('({0},{1})\n'.format(self.route, self.distance))
    # Create a shallow copy
    def copy(self):
        return State(self.route, self.distance)
    # Create a deep copy
    def deepcopy(self):
        return State(copy.deepcopy(self.route), copy.deepcopy(self.distance))
    # Update distance
    def update_distance(self, matrix, home):

        # Reset distance
        self.distance = 0
        # Keep track of departing city
        from_index = home
        # Loop all cities in the current route
        for i in range(len(self.route)):
```

```
            self.distance += matrix[from_index][self.route[i]]
            from_index = self.route[i]
        # Add the distance back to home
        self.distance += matrix[from_index][home]
# This class represent a city (used when we need to delete cities)
class City:
    # Create a new city
    def __init__(self, index:int, distance:int):
        self.index = index
        self.distance = distance
    # Sort cities
    def __lt__(self, other):
         return self.distance < other.distance
# Return true with probability p
def probability(p):
    return p > random.uniform(0.0, 1.0)
# Schedule function for simulated annealing
def exp_schedule(k=20, lam=0.005, limit=1000):
    return lambda t: (k * np.exp(-lam * t) if t < limit else 0)


# Get best solution by distance
def get_best_solution_by_distance(matrix:[], home:int):


    # Variables
```

```
route = []
from_index = home
length = len(matrix) - 1
# Loop until route is complete
while len(route) < length:
     # Get a matrix row
    row = matrix[from_index]
    # Create a list with cities
    cities = {}
    for i in range(len(row)):
        cities[i] = City(i, row[i])
    # Remove cities that already is assigned to the route
    del cities[home]
    for i in route:
        del cities[i]
    # Sort cities
    sorted = list(cities.values())
    sorted.sort()
    # Add the city with the shortest distance
    from_index = sorted[0].index
    route.append(from_index)
# Create a new state and update the distance
state = State(route)
state.update_distance(matrix, home)
```

```
    # Return a state
    return state


# Mutate a solution
def mutate(matrix:[], home:int, state:State, mutation_rate:float=0.01):


    # Create a copy of the state
    mutated_state = state.deepcopy()
    # Loop all the states in a route
    for i in range(len(mutated_state.route)):
        # Check if we should do a mutation
        if(random.random() < mutation_rate):
            # Swap two cities
            j = int(random.random() * len(state.route))
            city_1 = mutated_state.route[i]
            city_2 = mutated_state.route[j]
            mutated_state.route[i] = city_2
            mutated_state.route[j] = city_1
    # Update the distance
    mutated_state.update_distance(matrix, home)
    # Return a mutated state
    return mutated_state
# Simulated annealing
def simulated_annealing(matrix:[], home:int, initial_state:State, mutation_rate:float=0.01,
schedule=exp_schedule()):
```

```
    # Keep track of the best state
    best_state = initial_state
    # Loop a large number of times (int.max)
    for t in range(sys.maxsize):
        # Get a temperature
        T = schedule(t)
        # Return if temperature is 0
        if T == 0:
            return best_state
        # Mutate the best state
        neighbor = mutate(matrix, home, best_state, mutation_rate)
        # Calculate the change in e
        delta_e = best_state.distance - neighbor.distance
        # Check if we should update the best state
        if delta_e > 0 or probability(np.exp(delta_e / T)):
            best_state = neighbor
# The main entry point for this module
def main():
    # Cities to travel
    cities = ['AHMEDABAD', 'BHOPAL', 'CHANDIGARH', 'DELHI', 'BENGALURU',
'CHENNAI', 'SURAT', 'VADODARA', 'MUMBAI', 'LUCKNOW', 'PUNE', 'HYDERABAD',
'NOIDA']
    city_indexes = [0,1,2,3,4,5,6,7,8,9,10,11,12]
    # Index of start location
    home = 2 # Chicago
```

```
# Distances in kiometres between cities, same indexes (i, j) as in the cities array
matrix = [[0, 1451, 700, 1000, 1600, 1300, 2000, 200, 2500, 800, 100, 2100, 1900],
        [2400, 0, 1750, 1500, 850, 1250, 950, 2600, 400, 1600, 1350, 300, 600],
        [700, 1700, 0, 350, 900, 800, 1700, 850, 1800, 200, 950, 1400, 1200],
        [1100, 1500, 350, 0, 700, 800, 1400, 1100, 1500, 400, 1050, 1000, 900],
        [1600, 800, 900, 700, 0, 600, 1000, 1700, 950, 800, 800, 500, 350],
        [1300, 1200, 800, 800, 600, 0, 1600, 1500, 1700, 500, 200, 800, 1000],
        [2400, 900, 1700, 1300, 1000, 1600, 0, 2400, 600, 1700, 1890, 1110, 700],
        [213, 2596, 851, 1123, 1769, 1551, 2493, 0, 2699, 1038, 1605, 2300, 2099],
        [2571, 403, 1858, 1584, 949, 1765, 678, 2699, 0, 1744, 1645, 653, 600],
        [875, 1589, 262, 466, 796, 547, 1724, 1038, 1744, 0, 679, 1272, 1162],
        [1420, 1374, 940, 1056, 879, 225, 1891, 1605, 1645, 679, 0, 1017, 1200],
        [2145, 357, 1453, 1280, 586, 887, 1114, 2300, 653, 1272, 1017, 0, 504],
        [1972, 579, 1260, 987, 371, 999, 701, 2099, 600, 1162, 1200, 504, 0]]
# Get the best route by distance
state = get_best_solution_by_distance(matrix, home)
print('-- Best solution by distance --')
print(cities[home], end='')
for i in range(0, len(state.route)):
  print(' -> ' + cities[state.route[i]], end='')
print(' -> ' + cities[home], end='')
print('\n\nTotal distance: {0} KM'.format(state.distance))
print()
```

```
    # Run simulated annealing to find a better solution
    state = get_best_solution_by_distance(matrix, home)
    state = simulated_annealing(matrix, home, state, 0.1)
    print('-- Simulated annealing solution --')
    print(cities[home], end='')
    for i in range(0, len(state.route)):
        print(' -> ' + cities[state.route[i]], end='')
    print(' -> ' + cities[home], end='')
    print('\n\nTotal distance: {0} KM'.format(state.distance))
    print()
# Tell python to run main method
if __name__ == "__main__": main()
```

OUTPUT:

```
-- Best solution by distance --
CHANDIGARH -> LUCKNOW -> DELHI -> BENGALURU -> NOIDA -> HYDERABAD -> BHOPAL -> MUMBAI -> SURAT -> CHENNAI -> PUNE -> AHMEDAB
AD -> VADODARA -> CHANDIGARH

Total distance: 7926 KM

-- Simulated annealing solution --
CHANDIGARH -> LUCKNOW -> DELHI -> BENGALURU -> NOIDA -> SURAT -> MUMBAI -> BHOPAL -> HYDERABAD -> CHENNAI -> PUNE -> AHMEDAB
AD -> VADODARA -> CHANDIGARH

Total distance: 7278 KM

PARTH PATEL
19DCS098
```

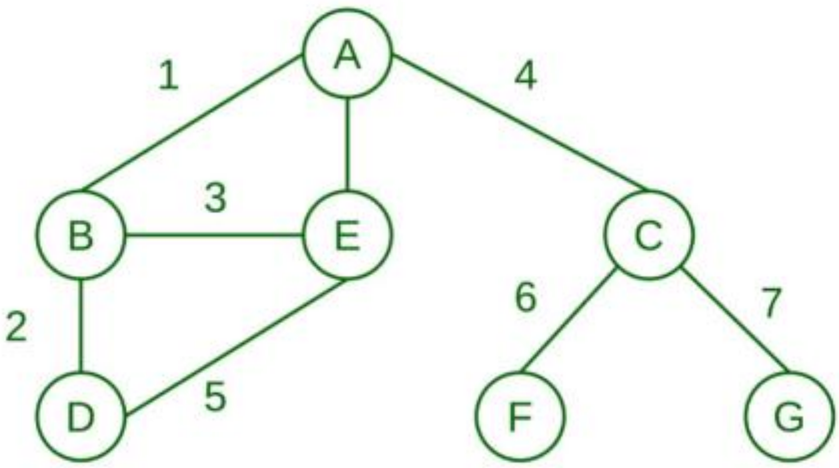
CONCLUSION:

By performing the above practical, we learnt the logic and concept of solving the problem of **TRAVELING SALESMAN PROBLEM** using **SIMULATED ANNEALING**.

PRACTICAL-7.4

AIM:

Write a Program to find Shortest Path using Best First Search Algorithm.



PROGRAM CODE:

```
#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;
        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; //NUMBER OF NODES
  cout<<"ENTER THE SIZE OF THE GRAPH: ";
   cin>>v;
   cout<<"----------------------------------"<<endl;
   graph.resize(v);

   int start,end,cost;

   for(int i=1;i<=v/2;i++){
  cout<<"ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : ";
       cin>>start>>end>>cost;
       addedge(start,end,cost);
       cout<<"----------------------------------------------------------------"<<endl;
   }
   int source;
   int destination;
   cout<<"ENTER THE SOURCE : ";
   cin>>source;
   cout<<"-------------------"<<endl;
   cout<<"ENTER THE DESTINATION : ";
   cin>>destination;
   cout<<"--------------------------"<<endl;    best_first_search(source, destination, v);
```

```
    cout<<endl;

    cout<<"PARTH PATEL\n19DCS098"<<endl;



    return 0;

}
```

OUTPUT:

```
Practicals\CPP PROGRAMS\ BEST_FIRST_SEARCH
ENTER THE SIZE OF THE GRAPH: 14
----------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 1 2 1
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 1 3 4
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 2 4 2
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 2 5 3
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 3 6 6
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 3 7 7
```

```
-------------------------------------------------------------------
ENTER THE STARTING POSITION,ENDING POSITION, AND COST TO REACH : 5 4 5
-------------------------------------------------------------------
ENTER THE SOURCE : 1
-------------------
ENTER THE DESTINATION : 5
--------------------------
1 2 4 5
PARTH PATEL
19DCS098
```

CONCLUSION:

By performing the above practical, we learnt the concept of **BEST FIRST SEARCH.**

PRACTICAL-7.5

AIM:

Write a program to solve 8 puzzle problem using A*Algorithm in python

PROGRAM CODE:

```
class Node:
    def __init__(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        """ Generate child nodes from the given node by moving the blank space
            either in the four directions {up,down,left,right} """
        x,y = self.find(self.data,'_')
        """ val_list contains position values for moving the blank space in either of
            the 4 directions [up,down,left,right] respectively. """
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
```

```
            child_node = Node(child,self.level+1,0)
            children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):
        """ Move the blank space in the given direction and if the position value are out
            of limits the return None """
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None

    def copy(self,root):
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
```

```
        return temp


    def find(self,puz,x):
        """ Specifically used to find the position of the blank space """
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j



class Puzzle:
    def __init__(self,size):
        """ Initialize the puzzle size by the specified size,open and closed lists to empty """
        self.n = size
        self.open = []
        self.closed = []


    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz
```

```
    def f(self,start,goal):
        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
        return self.h(start.data,goal)+start.level


    def h(self,start,goal):
        """ Calculates the different between the given puzzles """
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp
    def process(self):
        """ Accept Start and Goal Puzzle state"""
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        """ Put the start node in the open list"""
        self.open.append(start)
        print("\n\n")
        while True:
```

```
        cur = self.open[0]
        print("")
        print("  | ")
        print("  | ")
        print(" --- ")
       # print(" \\\'/ \n")
        for i in cur.data:
            for j in i:
                print(j,end=" ")
            print("")
        """ If the difference between current and goal node is 0 we have reached the goal node"""
        if(self.h(cur.data,goal) == 0):
            break
        for i in cur.generate_child():
            i.fval = self.f(i,goal)
            self.open.append(i)
        self.closed.append(cur)
        del self.open[0]
        """ sort the opne list based on f value """
        self.open.sort(key = lambda x:x.fval,reverse=False)
puz = Puzzle(3)
puz.process()
```

OUTPUT:

```
Enter the start state matrix

1 2 3
4 _ 6
5 7 8
Enter the goal state matrix

1 2 3
4 7 6
5 8 _


         1 2 3
         4 _ 6
         5 7 8

           |
           |
          ---
         1 2 3
         4 7 6
         5 _ 8

           |
           |
          ---
         1 2 3
         4 7 6
         5 8 _
```

PRACTICAL-8

AIM:

Write a program for game Tic-Tac-Toe using MINIMAX Algorithm in python.

PROGRAM CODE:

```
from math import inf as infinity

from random import choice

import platform

import time

from os import system


HUMAN = -1

AI = +1

board = [

    [0, 0, 0],

    [0, 0, 0],

    [0, 0, 0],

]




def evaluate(state):
```

```
#     Function to heuristic evaluation of state.
#     :param state: the state of the current board
#     :return: +1 if the AIuter wins; -1 if the human wins; 0 draw

    if wins(state, AI):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0

    return score

def wins(state, player):
#     This function tests if a specific player wins. Possibilities:
#     * Three rows    [X X X] or [O O O]
#     * Three cols    [X X X] or [O O O]
#     * Two diagonals [X X X] or [O O O]
#     :param state: the state of the current board
#     :param player: a human or a AIuter
#     :return: True if the player wins
```

```
    win_state = [
        [state[0][0], state[0][1], state[0][2]],
        [state[1][0], state[1][1], state[1][2]],
        [state[2][0], state[2][1], state[2][2]],
        [state[0][0], state[1][0], state[2][0]],
        [state[0][1], state[1][1], state[2][1]],
        [state[0][2], state[1][2], state[2][2]],
        [state[0][0], state[1][1], state[2][2]],
        [state[2][0], state[1][1], state[0][2]],
    ]
    if [player, player, player] in win_state:
        return True
    else:
        return False



def game_over(state):

#     This function test if the human or AIuter wins
#     :param state: the state of the current board
#     :return: True if the human or AIuter wins

    return wins(state, HUMAN) or wins(state, AI)
```

```
def empty_cells(state):

#     Each empty cell will be added into cells' list
#     :param state: the state of the current board
#     :return: a list of empty cells

    cells = []

    for x, row in enumerate(state):
        for y, cell in enumerate(row):
            if cell == 0:
                cells.append([x, y])

    return cells


def valid_move(x, y):

#     A move is valid if the chosen cell is empty
#     :param x: X coordinate
#     :param y: Y coordinate
#     :return: True if the board[x][y] is empty
```

```
    if [x, y] in empty_cells(board):
        return True
    else:
        return False



def set_move(x, y, player):


#     Set the move on board, if the coordinates are valid
#     :param x: X coordinate
#     :param y: Y coordinate
#     :param player: the current player

    if valid_move(x, y):
        board[x][y] = player
        return True
    else:
        return False



def minimax(state, depth, player):


#     AI function that choice the best move
#     :param state: current state of the board
```

```
#     :param depth: node index in the tree (0 <= depth <= 9),
#     but never nine in this case (see iaturn() function)
#     :param player: an human or a AIuter
#     :return: a list with [the best row, best col, best score]

    if player == AI:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]

    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]

    for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y

        if player == AI:
            if score[2] > best[2]:
                best = score  # max value
```

```
        else:
            if score[2] < best[2]:
                best = score  # min value

    return best



def clean():

#     Clears the console

    os_name = platform.system().lower()
    if 'windows' in os_name:
        system('cls')
    else:
        system('clear')



def render(state, c_choice, h_choice):

#     Print the board on console
#     :param state: current state of the board

    chars = {
```

```
        -1: h_choice,
        +1: c_choice,
        0: ' '
    }
    str_line = '---------------'

    print('\n' + str_line)
    for row in state:
        for cell in row:
            symbol = chars[cell]
            print(f'| {symbol} |', end='')
        print('\n' + str_line)



def ai_turn(c_choice, h_choice):

#     It calls the minimax function if the depth < 9,
#     else it choices a random coordinate.
#     :param c_choice: AIuter's choice X or O
#     :param h_choice: human's choice X or O

    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return
```

```
    clean()
    print(f'AIuter turn [{c_choice}]')
    render(board, c_choice, h_choice)

    if depth == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
    else:
        move = minimax(board, depth, AI)
        x, y = move[0], move[1]

    set_move(x, y, AI)
    time.sleep(1)


def human_turn(c_choice, h_choice):

#     The Human plays choosing a valid move.
#     :param c_choice: AIuter's choice X or O
#     :param h_choice: human's choice X or O
#     :return:

    depth = len(empty_cells(board))
```

```
    if depth == 0 or game_over(board):
        return

    # Dictionary of valid moves
    move = -1
    moves = {
        1: [0, 0], 2: [0, 1], 3: [0, 2],
        4: [1, 0], 5: [1, 1], 6: [1, 2],
        7: [2, 0], 8: [2, 1], 9: [2, 2],
    }

    clean()
    print(f'Human turn [{h_choice}]')
    render(board, c_choice, h_choice)

    while move < 1 or move > 9:
        try:
            move = int(input('Use numpad (1..9): '))
            coord = moves[move]
            can_move = set_move(coord[0], coord[1], HUMAN)

            if not can_move:
                print('Bad move')
                move = -1
```

```
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')


def play():

#     Main function that calls all functions

    clean()
    h_choice = ''  # X or O
    c_choice = ''  # X or O
    first = ''  # if human is the first

    # Human chooses X or O to play
    while h_choice != 'O' and h_choice != 'X':
        try:
            print('')
            h_choice = input('Choose X or O\nChosen: ').upper()
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
```

```
        except (KeyError, ValueError):
            print('Bad choice')

    # Setting AIuter's choice
    if h_choice == 'X':
        c_choice = 'O'
    else:
        c_choice = 'X'

    # Human may starts first
    clean()
    while first != 'Y' and first != 'N':
        try:
            first = input('First to start?[y/n]: ').upper()
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')

    # Main loop of this game
    while len(empty_cells(board)) > 0 and not game_over(board):
        if first == 'N':
            ai_turn(c_choice, h_choice)
```

```
            first = ''

        human_turn(c_choice, h_choice)
        ai_turn(c_choice, h_choice)

    # Game over message
    if wins(board, HUMAN):
        clean()
        print(f'Human turn [{h_choice}]')
        render(board, c_choice, h_choice)
        print('YOU WIN!')

    elif wins(board, AI):
        clean()
        print(f'AIuter turn [{c_choice}]')
        render(board, c_choice, h_choice)
        print('YOU LOSE!')

    else:
        clean()
        render(board, c_choice, h_choice)
        print('DRAW!')
```

```
    exit()

play()

print()

print("PARTH PATEL\n19DCS098")
```

OUTPUT:

```
Choose X or O
Chosen: X
First to start?[y/n]: n
AIuter turn [O]

---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------

Human turn [X]

---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
| O ||   ||   |
---------------
Use numpad (1..9): 1
AIuter turn [O]

---------------
| X ||   ||   |
---------------
|   ||   ||   |
---------------
| O ||   ||   |
---------------
```

```
Human turn [X]

---------------
| X ||   || O |
---------------
|   ||   ||   |
---------------
| O ||   ||   |
---------------
Use numpad (1..9): 5
AIuter turn [O]

---------------
| X ||   || O |
---------------
|   || X ||   |
---------------
| O ||   ||   |
---------------
```

```
Human turn [X]

---------------
| X ||   || O |
---------------
|   || X ||   |
---------------
| O ||   || O |
---------------
Use numpad (1..9): 8
AIuter turn [O]

---------------
| X ||   || O |
---------------
|   || X ||   |
---------------
| O || X || O |
---------------
AIuter turn [O]

---------------
| X ||   || O |
---------------
|   || X || O |
---------------
| O || X || O |
---------------

YOU LOSE!

PARTH PATEL
19DCS098
```

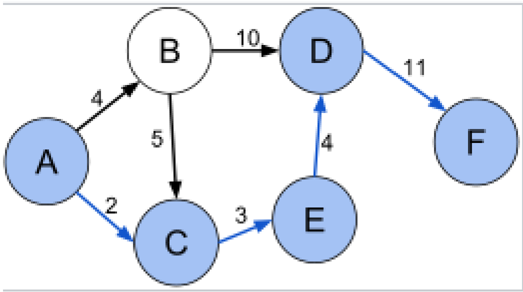
CONCLUSION:

By performing the above practical, we learnt the concept of minimax algorithm.

PRACTICAL-9

AIM:

Find out shortest path using Ant Colony Optimization for given Graph.



PROGRAM CODE:

```
import random as rn
import numpy as np
from numpy.random import choice as np_choice

class AntColony(object):

    def __init__(self, distances, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):

        self.distances  = distances
        self.pheromone = np.ones(self.distances.shape) / len(distances)
        self.all_inds = range(len(distances))
        self.n_ants = n_ants
        self.n_best = n_best
```

```
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta

    def run(self):
        shortest_path = None
        all_time_shortest_path = ("placeholder", np.inf)
        for i in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, self.n_best, shortest_path=shortest_path)
            shortest_path = min(all_paths, key=lambda x: x[1])
            #print (shortest_path)
            if shortest_path[1] < all_time_shortest_path[1]:
                all_time_shortest_path = shortest_path
            self.pheromone = self.pheromone * self.decay
        return all_time_shortest_path

    def spread_pheronome(self, all_paths, n_best, shortest_path):
        sorted_paths = sorted(all_paths, key=lambda x: x[1])
        for path, dist in sorted_paths[:n_best]:
            for move in path:
                self.pheromone[move] += 1.0 / self.distances[move]
```

```
    def gen_path_dist(self, path):
        total_dist = 0
        for ele in path:
            total_dist += self.distances[ele]
        return total_dist

    def gen_all_paths(self):
        all_paths = []
        for i in range(self.n_ants):
            path = self.gen_path(0)
            all_paths.append((path, self.gen_path_dist(path)))
        return all_paths

    def gen_path(self, start):
        path = []
        visited = set()
        visited.add(start)
        prev = start
        for i in range(len(self.distances) - 1):
            move = self.pick_move(self.pheromone[prev], self.distances[prev], visited)
            path.append((prev, move))
            prev = move
            visited.add(move)
        path.append((prev, start)) # going back to where we started
```

```
        return path


    def pick_move(self, pheromone, dist, visited):
        pheromone = np.copy(pheromone)
        pheromone[list(visited)] = 0

        row = pheromone ** self.alpha * (( 1.0 / dist) ** self.beta)

        norm_row = row / row.sum()
        move = np_choice(self.all_inds, 1, p=norm_row)[0]
        return move

distances = np.array([[np.inf,4,2,14,5,20],
                      [4,np.inf,5,10,8,21],
                      [2,5,np.inf,7,3,18],
                      [9,10,7,np.inf,4,11],
                      [5,8,3,4,np.inf,15],
                      [20,21,18,11,15,np.inf]])
ant_colony = AntColony(distances, 1, 1, 100, 0.95, alpha=1, beta=1)
shortest_path = ant_colony.run()
print ("shorted_path: {}".format(shortest_path))
```

OUTPUT:

```
shortest_path: ([(0, 2), (2, 4), (4, 3), (3, 5), (5, 1), (1, 0)], 45.0)
```

CONCLUSION:

By performing the above practical, we learnt about solving to find Shortest Path using Best First Search Algorithm

PRACTICAL-10

AIM:

Write a program to solve Multi Objective Optimization problem Using Particle Swarm Optimization

PROGRAM CODE:

```
import numpy as np

from numpy import matlib

import matplotlib.pyplot as plt

import random as random

import math


def deleteOneRepositoryMember(rep , gamma):

    gridindices = [item.gridIndex for item in rep]

    OCells = np.unique(gridindices)

    N = np.zeros(len(OCells))

    for k in range(len(OCells)):

        N[k] = gridindices.count(OCells[k])

    # selection probablity

    p = [math.exp(gamma*item) for item in N]

    p = np.array(p)/sum(p)


    # select cell index
```

```
    sci = roulettewheelSelection(p)
    SelectedCell = OCells[sci]

    #selected Cell members
    selectedCellmembers = [item for item in gridindices if item == SelectedCell]

    selectedmemberindex = np.random.randint(0,len(selectedCellmembers))
    #selectedmember = selectedCellmembers[selectedmemberindex]

    # delete memeber
    #rep[selectedmemberindex] = []
    rep = np.delete(rep, selectedmemberindex)

    return rep.tolist()



def SelectLeader(rep , beta):
    gridindices = [item.gridIndex for item in rep]
    OCells = np.unique(gridindices) # ocupied cells
    N = np.zeros(len(OCells))
    for k in range(len(OCells)):
        N[k] = gridindices.count(OCells[k])
    # selection probablity
    p = [math.exp(-beta*item) for item in N]
```

```
    p = np.array(p)/sum(p)


    # select cell index
    sci = roulettewheelSelection(p)
    SelectedCell = OCells[sci]


    #selected Cell members
    selectedCellmembers = [item for item in gridindices if item == SelectedCell]


    selectedmemberindex = np.random.randint(0,len(selectedCellmembers))
    # selectedmember = selectedCellmembers[selectedmemberindex]


    return rep[selectedmemberindex]




def roulettewheelSelection(p):
    r = random.random()
    cumsum = np.cumsum(p)
    y = (cumsum<r)
    x= [i for i in y if i==True]
    return len(x)


def FindGridIndex(particle, grid):
```

```
    nObj = len(particle.cost)
    NGrid = len(grid[0].LowerBounds)

    particle.gridSubIndex = np.zeros((1,nObj))[0]
    for j in range(nObj):
        index_in_Dim = len( [item for item in grid[j].UpperBounds if particle.cost[j]>item])
        particle.gridSubIndex[j] = index_in_Dim

    particle.gridIndex = particle.gridSubIndex[0]

    for j in range(1,nObj):
        particle.gridIndex = particle.gridIndex
        particle.gridIndex = NGrid*particle.gridIndex
        particle.gridIndex = particle.gridIndex + particle.gridSubIndex[j]

    return particle



def CreateGrid(pop,nGrid,alpha,nobj):
    costs = [item.cost for item in pop]
    Cmin = np.min(costs,axis=0)
    Cmax = np.max(costs,axis=0)
    deltaC = Cmax - Cmin
```

```
    Cmin =  Cmin - alpha*deltaC
    Cmax = Cmax + alpha*deltaC

    grid = [GridDim() for p in range(nobj)]
    for i in range(nobj):
      dimValues = np.linspace(Cmin[i],Cmax[i],nGrid+1).tolist()
      grid[i].LowerBounds = [-float('inf')] + dimValues
      grid[i].UpperBounds = dimValues  + [float('inf')]
    return grid




def Dominates(x,y):
    x=np.array(x)
    y=np.array(y)
    x_dominate_y = all(x<=y) and any(x<y)
    return x_dominate_y

def DetermineDomination(pop):
    pop_len= len(pop)
    for i in range(pop_len):
        pop[i].IsDominated = False

    for i in range(pop_len-1):
```

```
        for j in range(i+1,pop_len):
            if Dominates(pop[i].cost,pop[j].cost):
                pop[j].IsDominated = True
            if Dominates(pop[j].cost,pop[i].cost):
                pop[i].IsDominated = True

    return pop



# problem definition
def MOP2(x):
    x = np.array(x)
    n= len(x)
    z1 = 1 - math.exp(-sum((x-1/math.sqrt(n))**2))
    z2 = 1 - math.exp(-sum((x+1/math.sqrt(n))**2))
    return [z1,z2]

costfunction = lambda x: MOP2(x)

nVar = 5 # number of decision vars
varMin = -4
varMax = 4
maxIt = 100
nPop = 200    # population size
```

```
nRep = 50  # size of repository

w = 0.5 # inertia wieght

c1 = 2 # personal learning coefficient

c2 = 2 # global learning coefficient

wdamping = 0.99



# ################ constriction coefficients

# phi1 = 2.05

# phi2 = 2.05

# phi = phi1+phi2

# chi = 2/(phi - 2 + np.sqrt(phi**2 - 4*phi))

# w = chi # inertia wieght

# c1 = chi*phi1 # personal learning coefficient

# c2 = chi*phi2 # global learning coefficient

# wdamping = 1

# #################


beta = 1 # leader selection pressure

gamma = 1 # deletion selection pressure

NoGrid = 5

alpha=0.1 # nerkhe tavarrom grid


# initialization
```

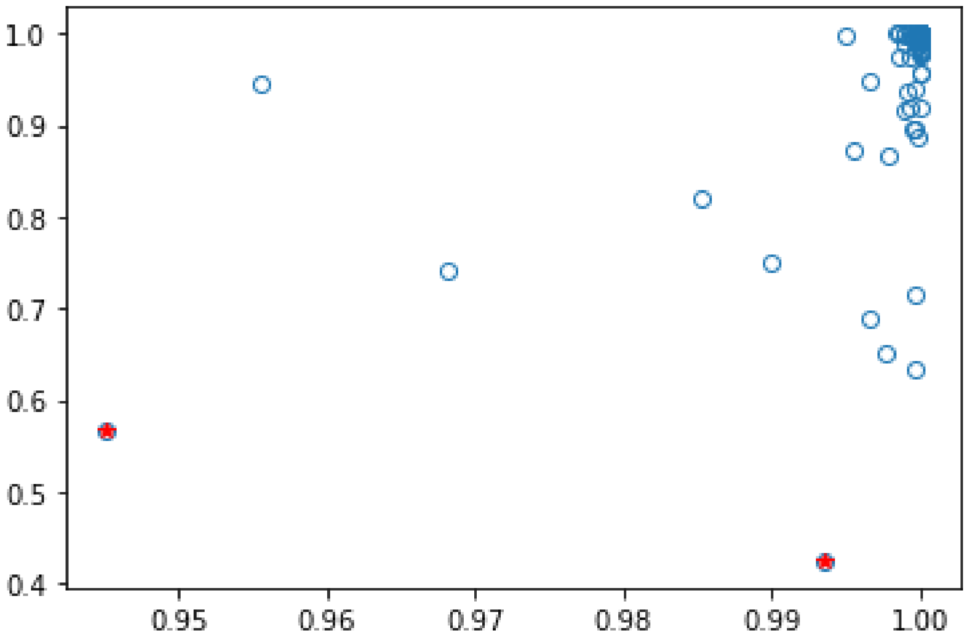
```
class Particle:
    position = []
    cost = []
    velocity = []
    best_position = []
    best_cost = []
    IsDominated = []
    gridIndex = []
    gridSubIndex = []

# for each objective a grid items is division of values of objective cost
class GridDim:
    LowerBounds = []
    UpperBounds = []

#Particles = np.matlib.repmat(Particle,nPop,1)
Particles = [Particle() for p in range(nPop)]
for i in range(nPop):
    Particles[i].position = np.random.uniform(varMin,varMax,nVar)
    Particles[i].velocity = np.zeros(nVar)
    Particles[i].cost = costfunction(Particles[i].position)
    # update best personal Best
    Particles[i].best_position = Particles[i].position
    Particles[i].best_cost = Particles[i].cost
```

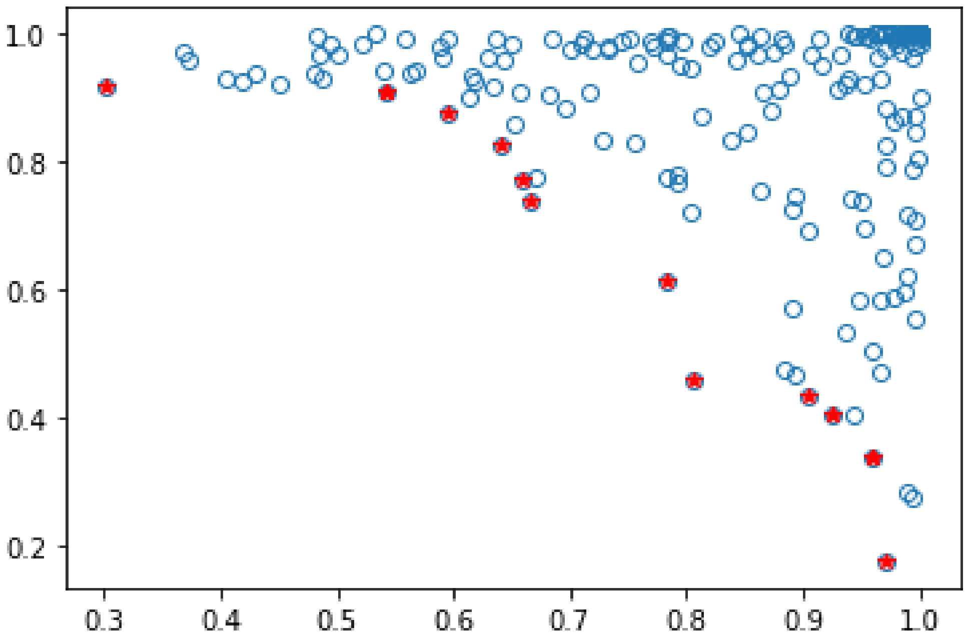
```
    Particles[i].IsDominated = False


Particles = DetermineDomination(Particles)


Repos = [item for item in Particles if item.IsDominated == False ]

nObj =len( Repos[0].cost)

grid = CreateGrid(Repos,NoGrid,alpha=0.1,nobj=nObj)


for r in range(len(Repos)):

    Repos[r] = FindGridIndex(Repos[0],grid)


# MOPSO main loop

for it in range(maxIt):

    for i in range(nPop):

        leader = SelectLeader(Repos,beta)

        # update velocity

        Particles[i].velocity = w*Particles[i].velocity  \

            + c1*np.random.rand(1,nVar)[0]*(Particles[i].best_position - Particles[i].position) \

            + c2*np.random.rand(1,nVar)[0]*(leader.position - Particles[i].position)


        # update position

        Particles[i].position = Particles[i].position + Particles[i].velocity


        # evaluation
```

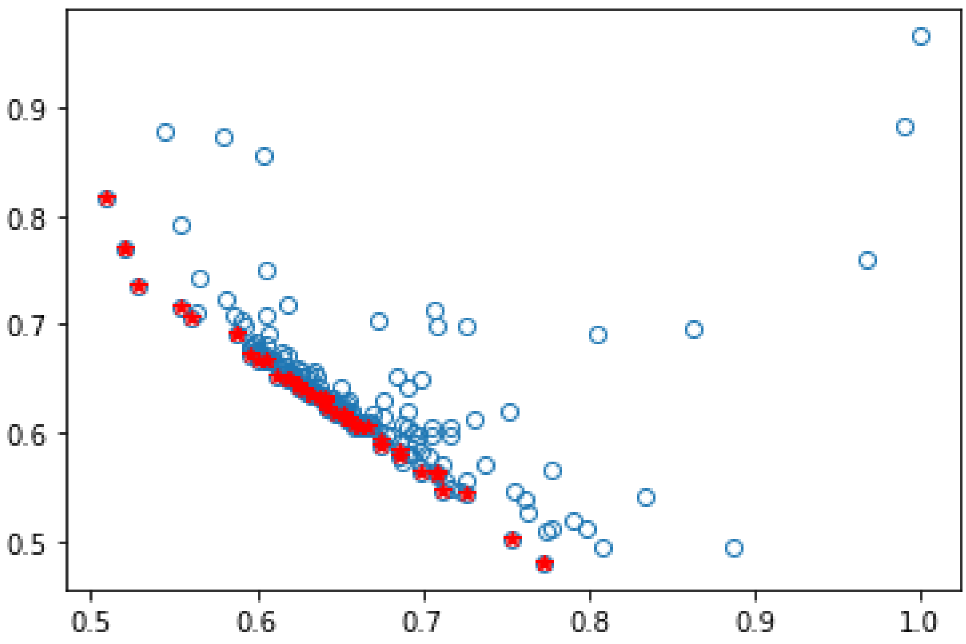
```
        Particles[i].cost = costfunction(Particles[i].position)


        if Dominates(Particles[i].cost,Particles[i].best_cost):
            Particles[i].best_position = Particles[i].position
            Particles[i].best_cost = Particles[i].cost
        else:
            if np.random.rand() > 0.5:
                Particles[i].best_position = Particles[i].position
                Particles[i].best_cost = Particles[i].cost



    Repos = Repos + Particles
    Repos = DetermineDomination(Repos)
    Repos = [item for item in Repos if item.IsDominated == False ]

    grid = CreateGrid(Repos,NoGrid,alpha=0.1,nobj=nObj)
    for r in range(len(Repos)):
        Repos[r] = FindGridIndex(Repos[r],grid)

    # check if repository is full
    if len(Repos) > nRep :
        extra = len(Repos) - nRep
        for e in range(extra):
            Repos = deleteOneRepositoryMember(Repos,gamma)
```

```
    ########## show figure ##########
    plt.clf()
    particlesCost = np.reshape( [item.cost for item in Particles ],newshape=(nPop,2))
    repositoryCost = [item.cost for item in Repos]
    repositoryCost = np.reshape( repositoryCost, newshape=(len(repositoryCost),2))
    plt.plot(particlesCost[:,0], particlesCost[:,1], 'o' ,mfc='none')
    plt.plot(repositoryCost[:,0], repositoryCost[:,1], 'r*')

    plt.draw()
    plt.pause(0.00000000001)

    w=w*wdamping

plt.show()
```

OUTPUT:


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.95
0.96
0.97
0.98
0.99
1.00


1.0
0.8
0.6
0.4
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0



CONCLUSION:

By performing the above practical, we learnt about how to solve Multi Objective Optimization problem Using Particle Swarm Optimization