SVR ENGINEERING COLLEGE

AYYALURUMETTA (V), NANDYAL, KURNOOL DT.
ANDHRA PRADESH – 518502

2021 – 2022

LAB MANVAL

of

ARTIFICIAL INTELLIGENCE (19A05502T)

(R-19 REGULATION)

Prepared by
Mr. Oruganti.Sampath
Asst. Professor

For
B.Tech. III Year/ I Sem. (CSE & AI)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SVR ENGINEERING COLLEGE

(AFFILIATED TO JNTUA ANANTHAPURAM- AICITE-INDIA)
AYYALURUMETTA (V), NANDYAL, KURNOOL DT.
ANDHRA PRADESH – 518502

LAB MANVAL CONTENT

ARTIFICIAL INTELLIGENCE (19A05502T)

1. Institute Vision & Mission, Department Vision & Mission

Institute Vision:

To produce Competent Engineering Graduates & Managers with a strong base of Technical & Managerial Knowledge and the Complementary Skills needed to be Successful Professional Engineers & Managers.

Institute Mission:

To fulfill the vision by imparting Quality Technical & Management Education to the Aspiring Students, by creating Effective Teaching/Learning Environment and providing State – of the – Art Infrastructure and Resources.

Department Vision:

To produce Industry ready Software Engineers to meet the challenges of 21st Century.

Department Mission:

- Impart core knowledge and necessary skills in Computer Science and Engineering through innovative teaching and learning methodology.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society.
- Cultivate the students with all-round competencies, for career, higher education and self-employability.

2. PO, PEO& PSO Statements

PROGRAMME OUTCOMES (POs)

PO-1: Engineering knowledge - Apply the knowledge of mathematics, science, engineering fundamentals of Computer Science& Engineering to solve complex real-life engineering problems related to CSE.

PO-2: Problem analysis - Identify, formulate, review research literature, and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO-3: Design/development of solutions - Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, cultural, societal and environmental considerations.

PO-4: Conduct investigations of complex problems - Use research-based knowledge and research methods, including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

PO-5: Modern tool usage - Select/Create and apply appropriate techniques, resources and modern engineering and IT tools and technologies for rapidly changing computing needs, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

PO-6: The engineer and society - Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice.

PO-7: Environment and Sustainability - Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO-8: Ethics - Apply ethical principles and commit to professional ethics and responsibilities and norms of the relevant engineering practices.

PO-9: Individual and team work - Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO-10: Communication - Communicate effectively on complex engineering activities with the engineering community and with the society-at-large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.

PO-11: Project management and finance - Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO-12: Life-long learning - Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadcast context of technological changes.

Program Educational Objectives (PEOs):

PEO 1:Graduates will be prepared for analyzing, designing, developing and testing the software solutions and products with creativity and sustainability.

PEO 2: Graduates will be skilled in the use of modern tools for critical problem solvingand analyzing industrial and societal requirements.

PEO 3:Graduates will be prepared with managerial and leadership skills for career and starting up own firms.

Program Specific Outcomes (PSOs):

PSO 1: Develop creative solutions by adapting emerging technologies / tools for real time applications.

PSO 2: Apply the acquired knowledge to develop software solutions and innovative mobile apps for various automation applications

2.1 Subject Time Table

SVR ENGINEERING COLLEGE::NANDYAL DEPARTMENT OF CSE								
ORUGANTI. SAMPATH						III-I		
Day/ Time	9:30 AM	10:20 AM	11:30 AM	12:20 PM-	LUNCH BREAK	02:00 PM	02:50 PM	03:40 PM
	10:20 AM	11:10AM	12:20 PM	01:10 PM		02:50 PM	03:40 PM	04:30 PM
MON						AI		
TUE			AI			AI - LAB		
WED							AI	
THU								
FRI				AI				
SAT	AI							

3.0 JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
B.Tech (CSE) – III-I L T P C
(19A05502P) ARTIFICIAL INTELLIGENCE LABORATORY

Course Objectives:

This course is designed to:

1. Explore the methods of implementing algorithms using artificial intelligence techniques
2. Illustrate search algorithms
3. Demonstrate building of intelligent agents

List of Experiments:

1. Write a program to implement DFS
2. Write a program to implement BFS
3. Write a Program to find the solution for travelling salesman Problem
4. Write a program to implement Simulated Annealing Algorithm
5. Write a program to find the solution for wampus world problem
6. Write a program to implement 8 puzzle problem
7. Write a program to implement Towers of Hanoi problem
8. Write a program to implement A* Algorithm
9. Write a program to implement Hill Climbing Algorithm
10. Build a bot which provides all the information related to you in college.
11. Build a virtual assistant for Wikipedia using Wolfram Alpha and Python
12. The following is a function that counts the number of times a string occurs in another string:

```
# Count the number of times string s1 is found in string s2
def countsubstring(s1,s2):
count = 0
for i in range(0,len(s2)-len(s1)+1):
if s1 == s2[i:i+len(s1)]:
count += 1
return count
```

For instance, countsubstring('ab','cabalaba') returns 2.

69 Page

Write a recursive version of the above function. To get the rest of a string (i.e. everything but the first character).

13. Higher order functions. Write a higher-order function count that counts the number of elements in a list that satisfy a given test. For instance: count(lambda x: x>2, [1,2,3,4,5]) should return 3, as there are three elements in the list larger than 2. Solve this task without using any existing higher-order function.

14. Brute force solution to the Knapsack problem. Write a function that allows you to generate random problem instances for the knapsack program. This function should generate a list of items containing N items that each have a unique name, a random size in the range 1 ..... 5 and a random value in the range 1 ..... 10.

Next, you should perform performance measurements to see **how long the given knapsack solver** take to solve different problem sizes. You should peform atleast 10 runs with different randomly generated problem instances for the problem sizes 10,12,14,16,18,20 and 22. Use a backpack size of 2:5 x N for each value problem size N. Please note that the method used to generate random numbers can also affect performance, since different distributions of values can make the initial conditions of the problem slightly more or less demanding.

How much longer time does it take to run this program when we increase the number of items? Does the backpack size affect the answer?

Try running the above tests again with a backpack size of 1 x N and with 4:0 x N.

15. Assume that you are organising a party for N people and have been given a list L of people who, for social reasons, should not sit at the same table. Furthermore, assume that you have C tables (that are infinitly large).
Write a function layout(N,C,L) that can give a table placement (ie. a number from 0 : : :C -1) for each guest such that there will be no social mishaps.
For simplicity we assume that you have a unique number 0 .....N-1 for each guest and that the list of restrictions is of the form [(X,Y), ...] denoting guests X, Y that are not allowed to sit together. Answer with a dictionary mapping each guest into a table assignment, if there are no possible layouts of the guests you should answer False.

References:

1 Tensorflow:
https://www.tensorflow.org/
2 Pytorch:
https://pytorch.org/
https://github.com/pytorch
3 Keras:
https://keras.io/
https://github.com/keras-team
4 Theano:
http://deeplearning.net/software/theano/
https://github.com/Theano/Theano

Course Outcomes:

Upon completion of the course, the students should be able to:
1. Implement search algorithms (L3)
2. Solve Artificial intelligence problems (L3)
3. Design chatbot and virtual assistant (L6)

4. CO-PO ATTAINMENT:

CO- ATTAINMENT:

SVR ENGINEERING COLLEGE				
Department:	**CSE**			
Course Outcome Attainment – Internal Assessments				
Name of the faculty :	O.SAMPATH		Academic Year:	2021-2022
Branch & Section:	CSE		Exam:	EXTERNAL LAB
Course:	AI-LAB		Semester:	III-1

Course Outcomes	Internal Lab		Internal Lab	University Exam
19A05502.1	3		3	3
19A05502.2	3		3	3
19A05502.3	3		3	3
19A05502.4	3		3	3
19A05502.5	3		3	3

Course Outcomes		Attainment Level
19A05502.1	Implement search algorithms	**3**
19A05502.2	Solve Artificial intelligence problems	**3**
19A05502.3	Design chatbot and virtual assistant	**3**
19A05502.4	Identify and apply Machine Learning algorithms to solve real-world problems	**3**
19A05502.5	Understand the implementation procedures for the machine learning algorithms	**3**
	Average Attainment	**3**
Overall Course Attainment		**3**

PO- ATTAINMENT:

SVR ENGINEERING COLLEGE

DEPARTMENT	CSE		
PROGRAM OUTCOME ATTAINMENT			
Name of Faculty:	O. SAMPATH	Academic Year	2021-2022
Branch & Section:	CSE	SUB CODE:	19A05502P
Course:	AI-LAB	Semester:	III-1

COURSE OUTCOME ATTAINMENT

Course outcome attainment	Internal		Internal	External
19A05502.1	3		3	3
19A05502.2	3		3	3
19A05502.3	3		3	3
19A05502.4	3		3	3
19A05502.5	3		3	3

COURSE OUTCOMES AND PROGRAM OUTCOMES MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO
19A05502.1	3	3	2	2	1	2			1		1	2	3	1
19A05502.2	3	2	2	1		1	1	1			2	1	2	2
19A05502.3	3	2	2	2	2	2	1		1	1			2	1
19A05502.4	3	3	1	2		2	2		2	2	2	2	2	2
19A05502.5	3	2	2	2	2	2		2		1		2	3	1

PO-ATTAINMENT

		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
INTERNAL	19A05502.1	9	9	6	6	3	6			3		3	6	9	3
	19A05502.	9	6	6	3		3	3	3			6	3	6	6
	19A05502.	9	6	6	6	6	6	3		3	3			6	3
	19A05502.	9	9	3	6		6	6		6	6	6	6	6	6
	19A05502.	9	6	6	6	6	6		6		3		6	9	3
UNIVERSITY	19A05502.1	9	9	6	6	3	6			3		3	6	9	3
	19A05502.	9	6	6	3		3	3	3			6	3	6	6
	19A05502.	9	6	6	6	6	6	3		3	3			6	3
	19A05502.	9	9	3	6		6	6		6	6	6	6	6	6
	19A05502.	9	6	6	6	6	6		6		3		6	9	3
OVERALL	19A05502.1	3	3	3	3	3	3			3		3	3	3	3
	19A05502.	3	3	3	3		3	3	3			3	3	3	3
	19A05502.	3	3	3	3	3	3	3		3	3			3	3
	19A05502.	3	3	3	3		3	3		3	3	3	3	3	3
	19A05502.	3	3	3	3	3	3		3		3		3	3	3
Attainment		3	3	3	3	3	3	3	3	3	3	3	3	3	3

Faculty Head of the Department

5.EXPERIMENT SOURCE CODE AND OUTPUTS

EXPERIMENT NO: 1

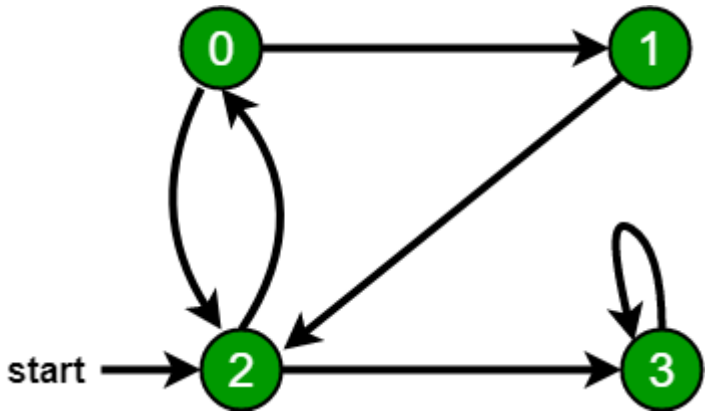
AIM: Write a Program to Implement Breadth First Search using Python.

```
# Python3 Program to print BFS traversal
# from a given source vertex. BFS(int s)
# traverses vertices reachable from s.
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then mark it
            # visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True

# Driver code

# Create a graph given in
# the above diagram
```

```
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print ("Following is Breadth First Traversal"
                  " (starting from vertex 2)")
g.BFS(2)
```

Output:

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```



EXPERIMENT NO: 2

AIM: Write a Program to Implement Depth First Search using Python.

```
# Python3 program to print DFS traversal
# from a given given graph
from collections import defaultdict

# This class represents a directed graph using
# adjacency list representation


class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # A function used by DFS
    def DFSUtil(self, v, visited):

        # Mark the current node as visited
        # and print it
        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self, v):

        # Create a set to store visited vertices
        visited = set()

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

# Driver code


# Create a graph given
# in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
```

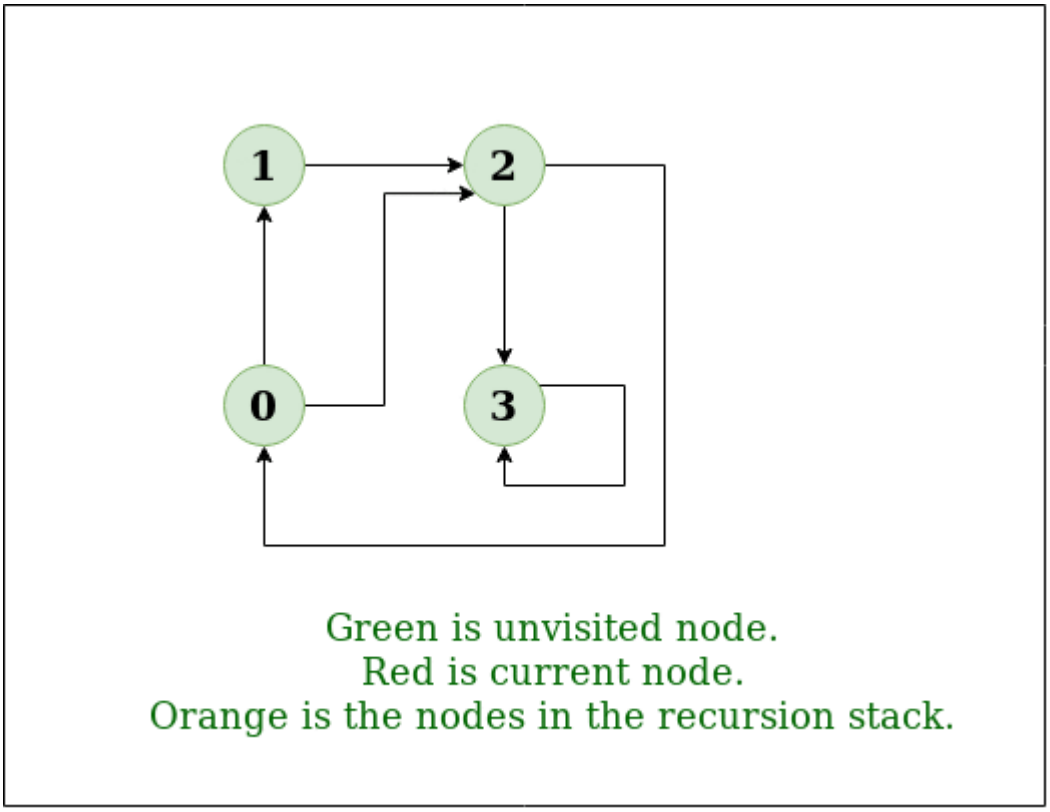
```
    g.addEdge(2, 0)
    g.addEdge(2, 3)
    g.addEdge(3, 3)

    print("Following is DFS from (starting from vertex 2)")
    g.DFS(2)
```

Output:

Following is Depth First Traversal (starting from

vertex 2)

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

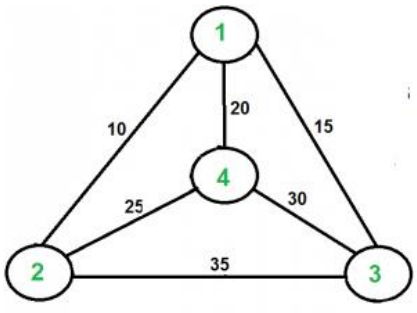


EXPERIMENT NO: 3

AIM: **Write a Program to find the solution for travelling salesman Problem**

```
# Python3 program to implement traveling salesman
from sys import maxsize
from itertools import permutations
V = 4
def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
    return min_path
if __name__ == "__main__":
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

Output 80



EXPERIMENT NO: 4

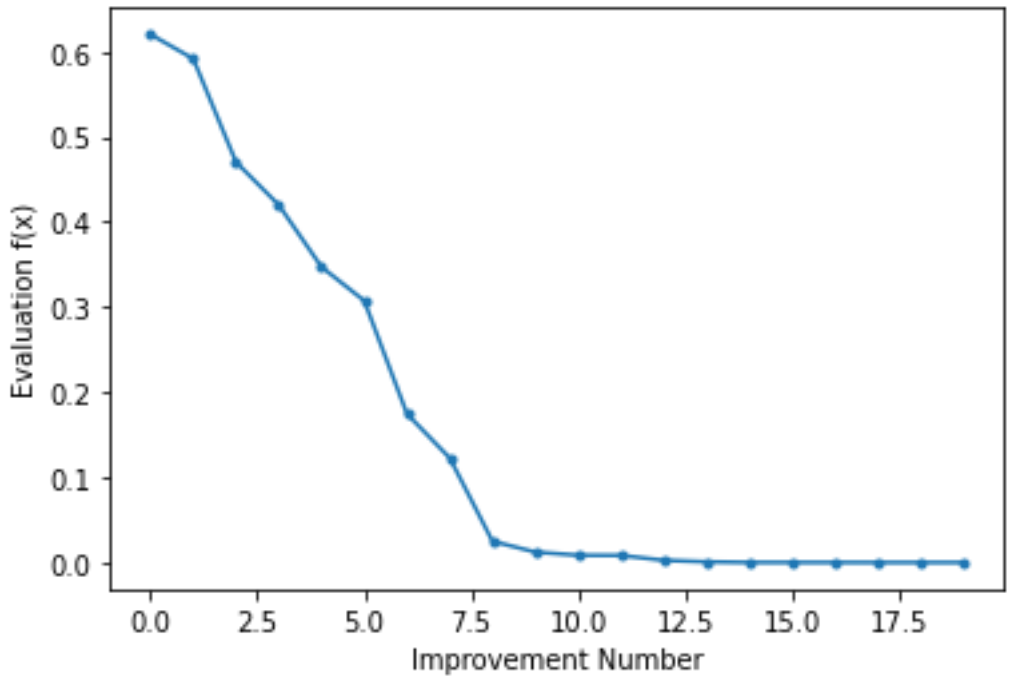
AIM: **Write a program to implement Simulated Annealing Algorithm**

```
# simulated annealing search of a one-dimensional objective function
from numpy import asarray
from numpy import exp
from numpy.random import randn
from numpy.random import rand
from numpy.random import seed
from matplotlib import pyplot

# objective function
def objective(x):
	return x[0]**2.0

# simulated annealing algorithm
def simulated_annealing(objective, bounds, n_iterations, step_size, temp):
	# generate an initial point
	best = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
	# evaluate the initial point
	best_eval = objective(best)
	# current working solution
	curr, curr_eval = best, best_eval
	scores = list()
	# run the algorithm
	for i in range(n_iterations):
		# take a step
		candidate = curr + randn(len(bounds)) * step_size
		# evaluate candidate point
		candidate_eval = objective(candidate)
		# check for new best solution
		if candidate_eval < best_eval:
			# store new best point
			best, best_eval = candidate, candidate_eval
			# keep track of scores
			scores.append(best_eval)
			# report progress
			print('>%d f(%s) = %.5f' % (i, best, best_eval))
		# difference between candidate and current point evaluation
		diff = candidate_eval - curr_eval
		# calculate temperature for current epoch
		t = temp / float(i + 1)
		# calculate metropolis acceptance criterion
		metropolis = exp(-diff / t)
		# check if we should keep the new point
		if diff < 0 or rand() < metropolis:
			# store the new current point
			curr, curr_eval = candidate, candidate_eval
	return [best, best_eval, scores]

# seed the pseudorandom number generator
seed(1)
# define range for input
bounds = asarray([[-5.0, 5.0]])
# define the total iterations
n_iterations = 1000
# define the maximum step size
step_size = 0.1
# initial temperature
temp = 10
# perform the simulated annealing search
best, score, scores = simulated_annealing(objective, bounds, n_iterations, step_size, temp)
print('Done!')
print('f(%s) = %f' % (best, score))
# line plot of best scores
pyplot.plot(scores, '.-')
pyplot.xlabel('Improvement Number')
```

```
pyplot.ylabel('Evaluation f(x)')
pyplot.show()
```

output:

```
>34 f([-0.78753544]) = 0.62021
>35 f([-0.76914239]) = 0.59158
>37 f([-0.68574854]) = 0.47025
>39 f([-0.64797564]) = 0.41987
>40 f([-0.58914623]) = 0.34709
>41 f([-0.55446029]) = 0.30743
>42 f([-0.41775702]) = 0.17452
>43 f([-0.35038542]) = 0.12277
>50 f([-0.15799045]) = 0.02496
>66 f([-0.11089772]) = 0.01230
>67 f([-0.09238208]) = 0.00853
>72 f([-0.09145261]) = 0.00836
>75 f([-0.05129162]) = 0.00263
>93 f([-0.02854417]) = 0.00081
>144 f([0.00864136]) = 0.00007
>149 f([0.00753953]) = 0.00006
>167 f([-0.00640394]) = 0.00004
>225 f([-0.00044965]) = 0.00000
>503 f([-0.00036261]) = 0.00000
>512 f([0.00013605]) = 0.00000
Done!
f([0.00013605]) = 0.000000
```

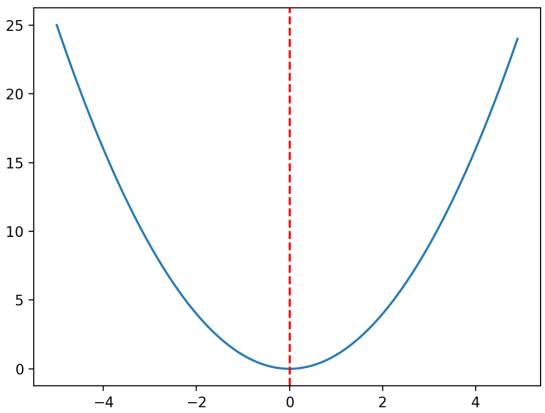


EXPERIMENT NO: 4(B)

AIM: **Write a program to implement Simulated Annealing Algorithm**

```
# convex unimodal optimization function
from numpy import arange
from matplotlib import pyplot

# objective function
def objective(x):
        return x[0]**2.0

# define range for input
r_min, r_max = -5.0, 5.0
# sample input range uniformly at 0.1 increments
inputs = arange(r_min, r_max, 0.1)
# compute targets
results = [objective([x]) for x in inputs]
# create a line plot of input vs result
pyplot.plot(inputs, results)
# define optimal input value
x_optima = 0.0
# draw a vertical line at the optimal input
pyplot.axvline(x=x_optima, ls='--', color='red')
# show the plot
pyplot.show()
```

output:



EXPERIMENT NO: 5

AIM: **Write a program to find the solution for wampus world problem**

AGENT.PY

```
class Agent:
    def __init__(self):
        self.__wumpusWorld = [
                ["","'P'",""], # Rooms [1,1] to [4,1]
                ["","",""], # Rooms [1,2] to [4,2]
                ['W',"",""], # Rooms [1,3] to [4,3]
                ["","",""],  # Rooms [1,4] to [4,4]
               ] # This is the wumpus world shown in the assignment question.
                 # A different instance of the wumpus world will be used for evaluation.
        self.__curLoc = [1,1]
        self.__isAlive = True
        self.__hasExited = False

    def __FindIndicesForLocation(self,loc):
        x,y = loc
        i,j = y-1, x-1
        return i,j

    def __CheckForPitWumpus(self):
        ww = self.__wumpusWorld
        i,j = self.__FindIndicesForLocation(self.__curLoc)
        if 'P' in ww[i][j] or 'W' in ww[i][j]:
            print(ww[i][j])
            self.__isAlive = False
            print('Agent is DEAD.')
        return self.__isAlive

    def TakeAction(self,action): # The function takes an action and returns whether the Agent is
alive
                                 # after taking the action.
        validActions = ['Up','Down','Left','Right']
        assert action in validActions, 'Invalid Action.'
        if self.__isAlive == False:
            print('Action cannot be performed. Agent is DEAD. Location:{0}'.format(self.__curLoc))
            return False
        if self.__hasExited == True:
            print('Action cannot be performed. Agent has exited the Wumpus
world.'.format(self.__curLoc))
            return False

        index = validActions.index(action)
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        move = validMoves[index]
        newLoc = []
        for v, inc in zip(self.__curLoc,move):
            z = v + inc #increment location index
            z = 4 if z>4 else 1 if z<1 else z #Ensure that index is between 1 and 4
```

```
            newLoc.append(z)
        self.__curLoc = newLoc
        print('Action Taken: {0}, Current Location {1}'.format(action,self.__curLoc))
        if self.__curLoc[0]==4 and self.__curLoc[1]==4:
            self.__hasExited=True
        return self.__CheckForPitWumpus()

    def __FindAdjacentRooms(self):
        cLoc = self.__curLoc
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        adjRooms = []
        for vM in validMoves:
            room = []
            valid = True
            for v, inc in zip(cLoc,vM):
                z = v + inc
                if z<1 or z>4:
                    valid = False
                    break
                else:
                    room.append(z)
            if valid==True:
                adjRooms.append(room)
        return adjRooms


    def PerceiveCurrentLocation(self): #This function perceives the current location.
                                #It tells whether breeze and stench are present in the current location.
        breeze, stench = False, False
        ww = self.__wumpusWorld
        if self.__isAlive == False:
            print('Agent cannot perceive. Agent is DEAD. Location:{0}'.format(self.__curLoc))
            return [None,None]
        if self.__hasExited == True:
            print('Agent cannot perceive. Agent has exited the Wumpus
World.'.format(self.__curLoc))
            return [None,None]

        adjRooms = self.__FindAdjacentRooms()
        for room in adjRooms:
            i,j = self.__FindIndicesForLocation(room)
            if 'P' in ww[i][j]:
                breeze = True
            if 'W' in ww[i][j]:
                stench = True
        return [breeze,stench]

    def FindCurrentLocation(self):
        return self.__curLoc

def main():
    ag = Agent()
```

```
    print('curLoc',ag.FindCurrentLocation())
    print('Percept [breeze, stench] :',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())


if __name__=='__main__':
    main()
```

WAMPUS.PY

```
from Agent import * # See the Agent.py file
import copy
numberOfCalls=0

class KnowledgeBase:
    #clause[i]=-1 represents the presence of negative literal represented by i
    #clause[i]=1 represents the presence of positive literal represented by i
    #values 0 to 15 represents W(1,1) to W(4,4)
    #values 16 to 31 represents S(1,1) to S(4,4)
    #values 33 to 47 represents P(1,1) to P(4,4)
    #values 48 to 63 represents B(1,1) to B(4,4)
    def __init__(self):
        self.clauses= []

        #clauses for atleast 1 Wumpus and 1 Pit
        atleast1Wumpus= {}
        atleast1Pit = {}
        for i in range (16):
            atleast1Wumpus[i]=1
            atleast1Pit[i+32]=1
        self.clauses.append(atleast1Wumpus)
        self.clauses.append(atleast1Pit)

        #clauses for atmost 1 Wumpus and 1 Pit
        for i in range(16):
            for j in range(i+1, 16):
                atmost1Wumpus={}
                atmost1Pit={}
                atmost1Wumpus[i]=-1
```

```
                atmost1Wumpus[j]=-1
                atmost1Pit[i+32]=-1
                atmost1Pit[j+32]=-1
                self.clauses.append(atmost1Wumpus)
                self.clauses.append(atmost1Pit)

        #Stench-Wumpus bijection clauses
        for i in range(16):
            stenchWumpusClause={}
            stenchWumpusClause[i+16]=-1
            if (i+4)//4 < 4:
                stenchWumpusClause[i+4]=1
                stenchClause={}
                stenchClause[i+16]=1
                stenchClause[i+4]=-1
                self.clauses.append(stenchClause)
            if(i-4)//4 >= 0:
                stenchWumpusClause[i-4]=1
                stenchClause={}
                stenchClause[i+16]=1
                stenchClause[i-4]=-1
                self.clauses.append(stenchClause)
            if i//4 == (i+1)//4:
                stenchWumpusClause[i+1]=1
                stenchClause={}
                stenchClause[i+16]=1
                stenchClause[i+1]=-1
                self.clauses.append(stenchClause)
            if i//4 == (i-1)//4:
                stenchWumpusClause[i-1]=1
                stenchClause={}
                stenchClause[i+16]=1
                stenchClause[i-1]=-1
                self.clauses.append(stenchClause)
            self.clauses.append(stenchWumpusClause)

        #Breeze-Pit Bijection Clauses
        for i in range(16):
            breezePitClause={}
            breezePitClause[i+48]=-1
            if(i+4)//4 < 4:
                breezePitClause[i+4+32]=1
                pitClause={}
                pitClause[i+48]=1
                pitClause[i+4+32]=-1
                self.clauses.append(pitClause)
            if(i-4)//4 >= 0:
                breezePitClause[i-4+32]=1
```

```
                pitClause={}
                pitClause[i+48]=1
                pitClause[i-4+32]=-1
                self.clauses.append(pitClause)
            if i//4 == (i+1)//4:
                breezePitClause[i+1+32]=1
                pitClause={}
                pitClause[i+48]=1
                pitClause[i+1+32]=-1
                self.clauses.append(pitClause)
            if i//4 == (i-1)//4:
                breezePitClause[i-1+32]=1
                pitClause={}
                pitClause[i+48]=1
                pitClause[i-1+32]=-1
                self.clauses.append(pitClause)
            self.clauses.append(breezePitClause)

        #No wumpus and pit at [1, 1]
        noWumpusStart={0:-1}
        noPitStart={32:-1}
        self.clauses.append(noWumpusStart)
        self.clauses.append(noPitStart)

    def AddClause(self, clause): #adding a clause to knowledge base
        self.clauses.append(clause)

    def getclauses(self): #return Wumpus clauses
        return copy.deepcopy(self.clauses)


def FindPureSymbol(clauses, symbols):
    for symbol in symbols:
        positive=0
        negative=0
        for clause in clauses:
            if symbol in clause:
                if clause[symbol]==1:
                    positive= positive+1
                else:
                    negative= negative+1
        if negative==0:
            return symbol, 1
        elif positive==0:
            return symbol, -1
    return -1, 0

def FindUnitClause(clauses):
```

```
    for clause in clauses:
        if len(clause)==1:
            for symbol in clause:
                return symbol, clause[symbol]
    return -1, 0

def selectSymbol(clauses, symbols):
    count={}
    positive={}
    negative={}
    for clause in clauses:
        for literal in clause:
            if literal not in count:
                count[literal]=0
                positive[literal]=0
                negative[literal]=0

            count[literal]= count[literal]+1
            if clause[literal]==1:
                positive[literal]=positive[literal]+1
            else:
                negative[literal]=negative[literal]+1

    maxLiteral= list(symbols.keys())[0]
    maxCount=0
    for literal in count:
        if count[literal]>maxCount:
            maxLiteral= literal
            maxCount= count[literal]

    if positive[maxLiteral]>negative[maxLiteral]:
        return maxLiteral, 1
    return maxLiteral, -1

def DPLL(clauses, symbols, model):
    global numberOfCalls
    numberOfCalls= numberOfCalls+1
    removeClauses=[]
    for clause in clauses:
        valueUnknown=True
        deleteLiterals=[]
        for literal in clause.keys():
            if literal in model.keys():
                if model[literal]==clause[literal]: #clause is true
                    removeClauses.append(clause)
                    valueUnknown=False
                    break
                else:
```

```
                    deleteLiterals.append(literal)

        for literal in deleteLiterals:
            del clause[literal]
        if valueUnknown==True and not bool(clause): #clause is false
            return False

    clauses= [ x for x in clauses if x not in removeClauses]

    if len(clauses)==0: #all clauses are true
        return True

    pureSymbol, value = FindPureSymbol(clauses, symbols)
    if value!=0:
        del symbols[pureSymbol]
        model[pureSymbol]=value
        return DPLL(clauses, symbols, model)

    unitSymbol, value = FindUnitClause(clauses)
    if value!=0:
        del symbols[unitSymbol]
        model[unitSymbol]=value
        return DPLL(clauses, symbols, model)

    symbol, value= selectSymbol(clauses, symbols)
    del symbols[symbol]
    model[symbol]= value

    if DPLL(copy.deepcopy(clauses), copy.deepcopy(symbols),
copy.deepcopy(model)):
        return True

    model[symbol]= -value
    return DPLL(clauses, symbols, model)


def DPLLSatisfiable(clauses):
    symbols={}
    for clause in clauses:
        for literal in clause:
            symbols[literal]=True

    model={}
    return DPLL(clauses, symbols, model)

def MoveToUnvisited(ag, visited, goalLoc, dfsVisited): #dfs to new safe room
    curPos=ag.FindCurrentLocation()
    curLoc= 4*(curPos[0]-1)+curPos[1]-1
    if(curLoc==goalLoc):
```

```
        return True
    dfsVisited[curLoc]=True

    if curPos[1]+1 <=4 and (visited[curLoc+1]==True or (curLoc+1)==goalLoc)
and dfsVisited[curLoc+1]==False:
        ag.TakeAction('Up')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Down')

    if curPos[0]+1 <=4 and (visited[curLoc+4]==True or (curLoc+4)==goalLoc)
and dfsVisited[curLoc+4]==False:
        ag.TakeAction('Right')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Left')

    if curPos[0]-1 >0 and (visited[curLoc-4]==True or (curLoc-4)==goalLoc) and
dfsVisited[curLoc-4]==False:
        ag.TakeAction('Left')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Right')

    if curPos[1]-1 >0 and (visited[curLoc-1]==True or (curLoc-1)==goalLoc) and
dfsVisited[curLoc-1]==False:
        ag.TakeAction('Down')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Up')

    return False

def ExitWumpusWorld(ag, kb):
    visited = [False for i in range(16)] #Rooms Visited till now
    while(ag.FindCurrentLocation()!=[4, 4]):
        percept= ag.PerceiveCurrentLocation()
        curPos = ag.FindCurrentLocation()
        curLocIndex= 4*(curPos[0]-1)+ curPos[1]-1
        visited[curLocIndex]=True

        breezeClause={}
        stenchClause={}
```

```
        if percept[0]==True: #breeze
            breezeClause[curLocIndex+48]=1
        else:
            breezeClause[curLocIndex+48]=-1
        kb.AddClause(breezeClause) #presence/absence of breeze

        if percept[1]==True: #stench
            stenchClause[curLocIndex+16]=1
        else:
            stenchClause[curLocIndex+16]=-1
        kb.AddClause(stenchClause) #presence/absence of stench

        for newLoc in range(16):
            if visited[newLoc]==False:
                tempclauses= kb.getclauses()
                checkClause={newLoc:1, newLoc+32:1}
                tempclauses.append(checkClause)
                if DPLLSatisfiable(tempclauses)==False:
                    #Room is safe
                    noWumpus={newLoc:-1}
                    noPit={newLoc+32:-1}
                    kb.AddClause(noWumpus)
                    kb.AddClause(noPit)
                    dfsVisited = [False for i in range(16)]
                    roomReachable=MoveToUnvisited(ag, visited, newLoc,
dfsVisited) #dfs to new safe Room
                    if roomReachable:
                        break

def main():
    ag = Agent()
    kb= KnowledgeBase()
    print('Start Location: {0}'.format(ag.FindCurrentLocation()))
    ExitWumpusWorld(ag, kb)
    print('{0} reached. Exiting the Wumpus
World.'.format(ag.FindCurrentLocation()))
    print('Total number of times DPLL function is called:
{0}'.format(numberOfCalls))


if __name__=='__main__':
    main()
```

EXPERIMENT NO: 6

AIM: **Write a program to implement 8 puzzle problem**

```
class Solution:
   def solve(self, board):
      dict = {}
      flatten = []
      for i in range(len(board)):
         flatten += board[i]
      flatten = tuple(flatten)

      dict[flatten] = 0

      if flatten == (0, 1, 2, 3, 4, 5, 6, 7, 8):
         return 0

      return self.get_paths(dict)

   def get_paths(self, dict):
      cnt = 0
      while True:
         current_nodes = [x for x in dict if dict[x] == cnt]
         if len(current_nodes) == 0:
            return -1

         for node in current_nodes:
            next_moves = self.find_next(node)
            for move in next_moves:
               if move not in dict:
                  dict[move] = cnt + 1
               if move == (0, 1, 2, 3, 4, 5, 6, 7, 8):
                  return cnt + 1
         cnt += 1

   def find_next(self, node):
      moves = {
         0: [1, 3],
         1: [0, 2, 4],
         2: [1, 5],
         3: [0, 4, 6],
         4: [1, 3, 5, 7],
         5: [2, 4, 8],
         6: [3, 7],
         7: [4, 6, 8],
         8: [5, 7],
      }

      results = []
      pos_0 = node.index(0)
      for move in moves[pos_0]:
         new_node = list(node)
         new_node[move], new_node[pos_0] = new_node[pos_0],
new_node[move]
         results.append(tuple(new_node))
```

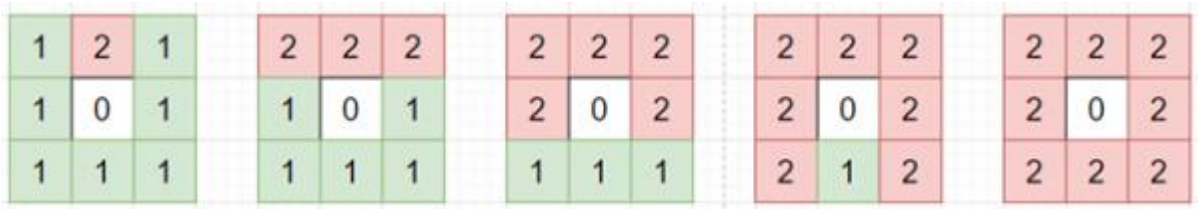
```
        return results
ob = Solution()
matrix = [
    [3, 1, 2],
    [4, 7, 5],
    [6, 8, 0]
]
print(ob.solve(matrix))
```

Input

```
matrix = [
[3, 1, 2],
[4, 7, 5],
[6, 8, 0] ]
```

Output

then the output will be 4

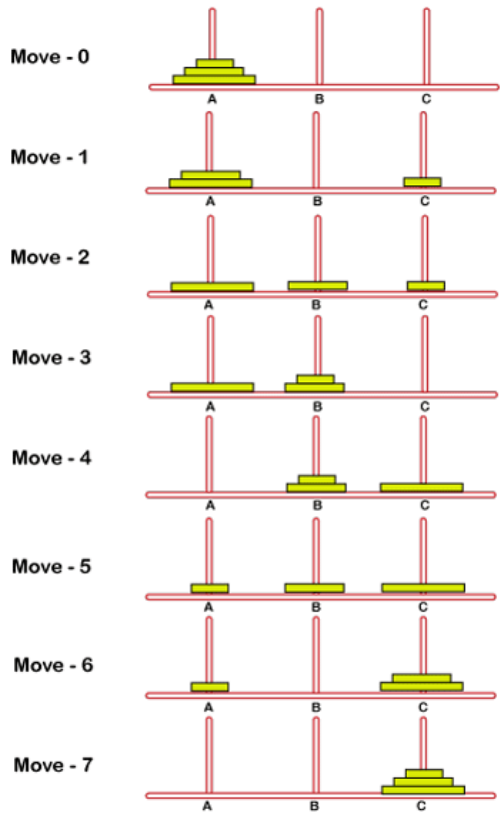


.

EXPERIMENT NO: 7

AIM: **Write a program to implement Towers of Hanoi problem**

```
# Creating a recursive function
def tower_of_hanoi(disks, source, auxiliary, target):
    if(disks == 1):
        print('Move disk 1 from rod {} to rod {}.'.format(source, target))
        return
    # function call itself
    tower_of_hanoi(disks - 1, source, target, auxiliary)
    print('Move disk {} from rod {} to rod {}.'.format(disks, source, target))
    tower_of_hanoi(disks - 1, auxiliary, source, target)

  disks = int(input('Enter the number of disks: '))
# We are referring source as A, auxiliary as B, and target as C
tower_of_hanoi(disks, 'A', 'B', 'C')  # Calling the function
```



Tower of Hanoi

Output

```
Enter the number of disks: 3
Move disk 1 from rod A to rod C.
Move disk 2 from rod A to rod B.
Move disk 1 from rod C to rod B.
Move disk 3 from rod A to rod C.
Move disk 1 from rod B to rod A.
Move disk 2 from rod B to rod C.
Move disk 1 from rod A to rod C.
```

EXPERIMENT NO: 8

AIM: **Write a program to implement A* Algorithm**

```
from queue import PriorityQueue

#Creating Base Class
class State(object):
    def __init__(self, value, parent, start = 0, goal = 0):
        self.children = []
        self.parent = parent
        self.value = value
        self.dist = 0
        if parent:
            self.start = parent.start
            self.goal = parent.goal
            self.path = parent.path[:]
            self.path.append(value)

        else:
            self.path = [value]
            self.start = start
            self.goal = goal

    def GetDistance(self):
        pass
    def CreateChildren(self):
        pass


# Creating subclass
class State_String(State):
    def __init__(self, value, parent, start = 0, goal = 0 ):
        super(State_String, self).__init__(value, parent, start, goal)
        self.dist = self.GetDistance()

    def GetDistance(self):
            if self.value == self.goal:
                return 0
            dist = 0
            for i in range(len(self.goal)):
                letter = self.goal[i]
                dist += abs(i - self.value.index(letter))
            return dist

    def CreateChildren(self):
            if not self.children:
                for i in range(len(self.goal)-1):
                    val = self.value
                    val = val[:i] + val[i+1] + val[i] + val[i+2:]
                    child = State_String(val, self)
                    self.children.append(child)
```

```
# Creating a class that hold the final magic
class A_Star_Solver:
    def __init__(self, start, goal):
        self.path = []
        self.vistedQueue =[]
        self.priorityQueue = PriorityQueue()
        self.start = start
        self.goal = goal

    def Solve(self):
        startState = State_String(self.start,0,self.start,self.goal)

        count = 0
        self.priorityQueue.put((0,count, startState))
        while(not self.path and self.priorityQueue.qsize()):
            closesetChild = self.priorityQueue.get()[2]
            closesetChild.CreateChildren()
            self.vistedQueue.append(closesetChild.value)
            for child in closesetChild.children:
                if child.value not in self.vistedQueue:
                 count += 1
                 if not child.dist:
                    self.path = child.path
                    break
                 self.priorityQueue.put((child.dist,count,child))
        if not self.path:
            print("Goal Of  is not possible !" + self.goal )
        return self.path

# Calling all the existing stuffs
if __name__ == "__main__":
    start1 = "path"
    goal1 = "hpta"
    print("Starting....")
    a = A_Star_Solver(start1,goal1)
    a.Solve()
    for i in range(len(a.path)):
        print("{0}){1}".format(i,a.path[i]))
```

output:

```
Starting....
0)path
1)ptah
2)ptha
3)phta
4)hpta
```

EXPERIMENT NO: 9

AIM:. Write a program to implement Hill Climbing Algorithm

```
import random
def randomSolution(tsp):
    cities = list(range(len(tsp)))
    solution = []

    for i in range(len(tsp)):
        randomCity = cities[random.randint(0, len(cities) - 1)]
        solution.append(randomCity)
        cities.remove(randomCity)
    return solution

def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength

def getNeighbours(solution):
    neighbours = []
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours

def getBestNeighbour(tsp, neighbours):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
        currentRouteLength = routeLength(tsp, neighbour)
        if currentRouteLength < bestRouteLength:
            bestRouteLength = currentRouteLength
            bestNeighbour = neighbour
    return bestNeighbour, bestRouteLength

def hillClimbing(tsp):
    currentSolution = randomSolution(tsp)
    currentRouteLength = routeLength(tsp, currentSolution)
    neighbours = getNeighbours(currentSolution)
    bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)

    while bestNeighbourRouteLength < currentRouteLength:
        currentSolution = bestNeighbour
        currentRouteLength = bestNeighbourRouteLength
        neighbours = getNeighbours(currentSolution)
        bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)
    return currentSolution, currentRouteLength
def main():
    tsp = [
        [0, 400, 500, 300],
        [400, 0, 300, 500],
        [500, 300, 0, 400],
        [300, 500, 400, 0]
    ]
    print(hillClimbing(tsp))
if __name__ == "__main__":
    main()
```

output:

```
([0, 1, 2, 3], 1400)
```

EXPERIMENT NO: 10

AIM: **Build a bot which provides all the information related to you in the college.**

```
def greet(bot_name, birth_year):
    print("Hello! My name is {0}.".format(bot_name))
    print("I was created in {0}.".format(birth_year))


def remind_name():
    print('Please, remind me your name.')
    name = input()
    print("What a great name you have, {0}!".format(name))


def guess_age():
    print('Let me guess your age.')
    print('Enter remainders of dividing your age by 3, 5 and 7.')

    rem3 = int(input())
    rem5 = int(input())
    rem7 = int(input())
    age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105

    print("Your age is {0}; that's a good time to start programming!".format(age))


def count():
    print('Now I will prove to you that I can count to any number you want.')
    num = int(input())

    counter = 0
    while counter <= num:
        print("{0} !".format(counter))
        counter += 1


def test():
    print("Let's test your programming knowledge.")
    print("Why do we use methods?")
    print("1. To repeat a statement multiple times.")
    print("2. To decompose a program into several small subroutines.")
    print("3. To determine the execution time of a program.")
    print("4. To interrupt the execution of a program.")

    answer = 2
    guess = int(input())
    while guess != answer:
        print("Please, try again.")
        guess = int(input())

    print('Completed, have a nice day!')
```

```
        print('.....................')
        print('.....................')
        print('.....................')


    def end():
        print('Congratulations, have a nice day!')
        print('.....................')
        print('.....................')
        print('.....................')
        input()

    greet('Sbot', '2021')  # change it as you need
    remind_name()
    guess_age()
    count()
    test()
    end()
```

OUTPUT:

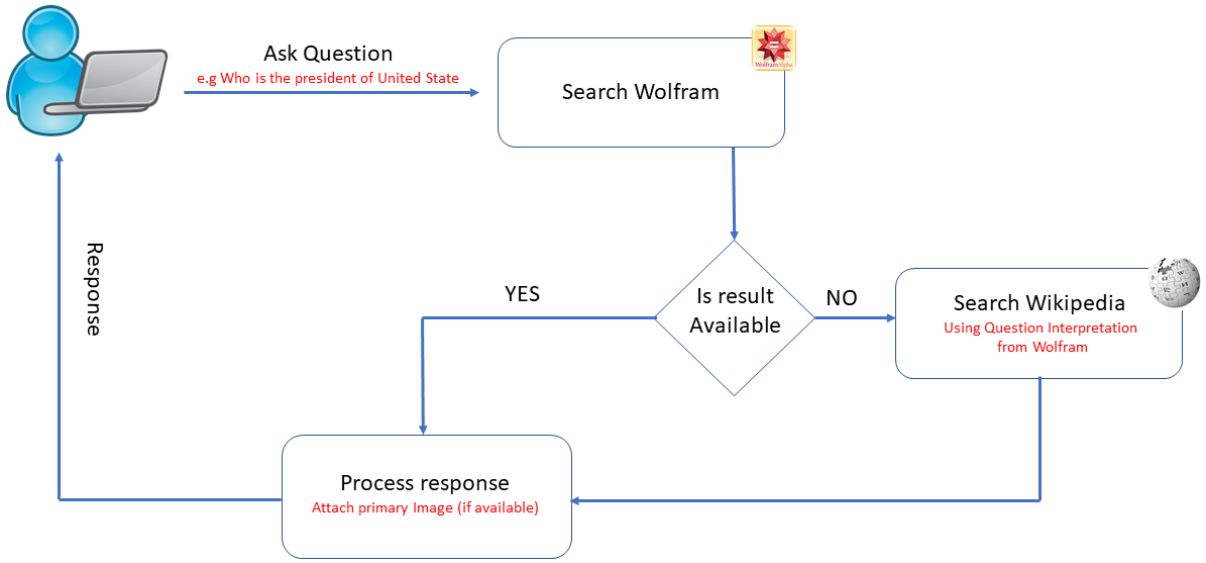
```
Hello! My name is Sbot.
I was created in 2021.
Please, remind me your name.
sampath
What a great name you have, sampath!
Let me guess your age.
Enter remainders of dividing your age by 3, 5 and 7.
5
3
Your age is 28; that's a good time to start programming!
Now I will prove to you that I can count to any number you want.
6
0 !
1 !
2 !
3 !
4 !
5 !
6 !
Let's test your programming knowledge.
Why do we use methods?
1. To repeat a statement multiple times.
2. To decompose a program into several small subroutines.
3. To determine the execution time of a program.
4. To interrupt the execution of a program.
2
Completed, have a nice day!
.....................
.....................
.....................
Congratulations, have a nice day!
.....................
.....................
.....................
```

EXPERIMENT NO: 11

AIM: **Build a virtual assistant for Wikipedia using Wolfram Alpha and Python**

```
# Python program to
# demonstrate creation of an
# assistant using wolf ram API

import wolframalpha

# Taking input from user
question = input('Question: ')

# App id obtained by the above steps
app_id = 'Your app_id'

# Instance of wolf ram alpha
# client class
client = wolframalpha.Client(app_id)

# Stores the response from
# wolf ram alpha
res = client.query(question)

# Includes only text from the response
answer = next(res.results).text

print(answer)
```

output:



EXPERIMENT NO: 12

AIM:**. WRITE A PROGRAM USING function that counts the number of times a string occurs in another string:**

```
# A Naive recursive Python program
# to find the number of times the
# second string occurs in the first
# string, whether continuous or
# discontinuous

# Recursive function to find the
# number of times the second string
# occurs in the first string,
# whether continuous or discontinuous
def count(a, b, m, n):

    # If both first and second string
    # is empty, or if second string
    # is empty, return 1
    if ((m == 0 and n == 0) or n == 0):
        return 1

    # If only first string is empty
    # and second string is not empty,
    # return 0
    if (m == 0):
        return 0

    # If last characters are same
    # Recur for remaining strings by
    # 1. considering last characters
    #    of both strings
    # 2. ignoring last character
    #    of first string
    if (a[m - 1] == b[n - 1]):
        return (count(a, b, m - 1, n - 1) +
                count(a, b, m - 1, n))
    else:

        # If last characters are different,
        # ignore last char of first string
        # and recur for remaining string
        return count(a, b, m - 1, n)

# Driver code
a = "GeeksforGeeks"
b = "Gks"

print(count(a, b, len(a),len(b)))
```

Output:

4

EXPERIMENT NO: 13

AIM: **Write a higher-order function count that counts the number of elements in a list that satisfy a given test IN PYTHON**

```
from functools import reduce

def getCount(listOfElems, cond = None):
    'Returns the count of elements in list that satisfies the given condition'
    if cond:
        count = sum(cond(elem) for elem in listOfElems)
    else:
        count = len(listOfElems)
    return count
def main():
    # List of numbers
    listOfElems = [11, 22, 33, 45, 66, 77, 88, 99, 101]
    print('**** Use map() & sum() to count elements in a list that satisfy certain conditions
****')
    print('** Example 1 **')
    # Count odd numbers in the list
    count = sum(map(lambda x : x%2 == 1, listOfElems))
    print('Count of odd numbers in a list : ', count)
    print('** Example 1 : Explanation  **')
    # Get a map object by applying given lambda to each element in list
    mapObj = map(lambda x : x%2 == 1, listOfElems)
    print('Contents of map object : ', list(mapObj))
    print('** Example 2**')
    # Count even numbers in the list
    count = sum(map(lambda x : x%2 == 0, listOfElems))
    print('Count of even numbers in a list : ', count)
    print('** Example 3**')
    # count numbers in the list which are greater than 5
    count = sum(map(lambda x : x>5, listOfElems))
    print('Count of numbers in a list which are greater than 5: ', count)
    print('**** Using sum() & Generator expression to count elements in list based on
conditions ****')
    # count numbers in the list which are greater than 5
    count = getCount(listOfElems, lambda x : x>5)
    print('Count of numbers in a list which are greater than 5: ', count)
    # count numbers in the list which are greater than 5 but less than 20
    count = getCount(listOfElems, lambda x : x>5 and x < 20)
    print('Count of numbers in a list which are greater than 5 but less than 20 : ', count)
    # Get total number of elements in the list
    count = getCount(listOfElems)
    print('Total Number of elements in List: ', count)
    print('**** Use List comprehension to count elements in list based on conditions ****')
    # count numbers in the list which are greater than 5
    count = len([elem for elem in listOfElems if elem > 5])
    print('Count of numbers in a list which are greater than 5: ', count)

    print('**** Use reduce() function to count elements in list based on conditions ****')
    # count numbers in the list which are greater than 5
    count = reduce(lambda default, elem: default + (elem > 5), listOfElems, 0)
```

```
    print('Count of numbers in a list which are greater than 5: ', count)
if __name__ == '__main__':
    main()
```

output:

```
**** Use map() & sum() to count elements in a list that satisfy certain
conditions ****
** Example 1 **
Count of odd numbers in a list :  6
** Example 1 : Explanation  **
Contents of map object :  [True, False, True, True, False, True, False,
True, True]
** Example 2**
Count of even numbers in a list :  3
** Example 3**
Count of numbers in a list which are greater than 5:  9
**** Using sum() & Generator expression to count elements in list based
on conditions ****
Count of numbers in a list which are greater than 5:  9
Count of numbers in a list which are greater than 5 but less than 20 :
1
Total Number of elements in List:  9
**** Use List comprehension to count elements in list based on
conditions ****
Count of numbers in a list which are greater than 5:  9
**** Use reduce() function to count elements in list based on
conditions ****
Count of numbers in a list which are greater than 5:  9
```

14.Write a function that allows you to generate random problem instances for the knapsack program USING PYTHON

```
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    # Build table K[][] in bottom up manner
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]

# Driver program to test above function
val = [10,12,14,16,18,20,22]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))


output:
220
```