ARTIFICAL INTELLIGENCE LABORATORY
18AIL57

LABORATORY MANUAL

V Semester B.E.

(Academic Year: 2022-23)

Prepared by

Dr. Navaneeth Bhaskar

Associate Professor

Department of CSE (Data Science)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

SAHYADRI

College of Engineering & Management

Adyar, Mangaluru - 575007

Vision

To be a premier institution in Technology and Management by fostering excellence in education, innovation, incubation and values to inspire and empower the young minds.

Mission

M1. Creating an academic ambience to impart holistic education focusing on individual growth, integrity, ethical values and social responsibility.

M2. Develop skill based learning through industry-institution interaction to enhance competency and promote entrepreneurship.

M3. Fostering innovation and creativity through competitive environment with state-of-the-art infrastructure.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Vision

To be a center of excellence in Data Science and Engineering through the interactive teaching-learning process, research, and innovation.

Mission

M1. Creating competitive ambience to enhance the innovative and experiential learning process through state of the art infrastructure.

M2. Grooming young minds through industry-institute interactions to solve societal issues and inculcate affinity towards research and entrepreneurship.

M3. Promoting teamwork and leadership qualities through inter-disciplinary activities in diversified areas of data science and engineering.

Program Educational Objectives (PEOs):

PEO1: Possess theoretical and practical knowledge to identify, scrutinize, formulate and solve challenging problems related to dynamically evolving data science.

PEO2: Inculcate core competency, professionalism and ethics to cater industrial needs and to solve societal problems.

PEO3: Engage in Lifelong learning and stay intact to the transformation in technologies and pursue research.

Program Outcomes:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. 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.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. 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 professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. 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.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs):

PSO1: Exhibit competency and skills in distributed computing, information security, cyber security, data analytics, and machine learning.

PSO2: Able to provide sustainable solution to implement and validate information science projects.

COURSE OUTCOMES

COs	**Description**	**Bloom's Level**
CO1	Illustrate the syntax of the Python programming language	CL3
CO2	Demonstrate python program to implement List, Set and Dictionary operations	CL3
CO3	Demonstrate AI search algorithms in python	CL3
CO4	Demonstrate the problem-solving strategies in AI	CL3
CO5	Illustrate the game-playing strategies in AI	CL3

Contents

Part A

Program 1: Basic Programs in Python

1(a): Aim: Write a python program to print the multiplication table of the given number.

Code:

```
# Multiplication table (from 1 to 10) in Python

# To take input from the user
num = int(input("Display multiplication table of ? "))

# Iterate 10 times from i = 1 to 10
for i in range(1, 11):
   print(num, 'x', i, '=', num*i)
```

Output:

```
Display multiplication table of? 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

1(b): Aim: Write a python program to check whether the given number is prime or not.

Code:

```
num = int(input('Enter a number: '))
# If the given number is greater than 1
if num > 1:
        for i in range(2,num):
# If num is divisible by any number between 2 and n / 2, it is not prime
                if (num % i) == 0:
                        print(num, "is not a prime number")
                        break
                else:
                        print(num, "is a prime number")
else:
        print(num, "is not a prime number")
```

Output:

```
>         Enter number: 5
5 is a prime number
>         Enter number: 9
9 is not a prime number
```

1(c): Aim: Write a python program to find the factorial of the given number.

Code:

```
num = int(input("Enter a number: "))
factorial = 1
          if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

Output:

```
Enter a number: 4
The factorial of 4 is 24
```

Program 2: List operations and List methods

Aim: Write a python program to implement List operations and List methods (Nested List, Length, Concatenation, Membership, Iteration, Indexing, Slicing, Append, Extend & Delete)

Code:

```
# Create a student nested list, each list in the nested list contains name, age, gender, and marks of the student

# Creating a list
student_list = [['john',20, 'male',70], ['josh',21, 'male',77], ['sri',19, 'female',80], ['siri',21, 'female',89]]

# Print student list
print(student_list)

# Print all the elements individually using Nested List using two for loops
for list in student_list:
    for i in list:
        print(i)

# Concatenation
student_list1 =['john',20,'male',70], ['josh',21,'male',77]
student_list2 =['sri',19,'female',80], ['siri',21,'female',89]
#Concatenate the two lists using + operator
print(student_list1+student_list2)

#Membership
#check 'a' is in the list L: Output True if the element present in the list
print(['john',20, 'male',70] in student_list)
#check 'a' not in the list L: Output False if the element is present in the list
print(['john',20, 'male',70] not in student_list)

# Iteration
for i in student_list:
    print(i)

# Accessing elements using positive and negative INDEXING
print(student_list[0])
print(student_list[1])
print(student_list[2])
print(student_list[3])
print(student_list[-1])
print(student_list[-2])
print(student_list[-3])
print(student_list[-4])
```

```
# Slicing
#Printing the elements from index 0 to index 2
print(student_list[0:3])
#Printing the elements from index 0 to end
print(student_list[0:])
#Printing the elements from index 0 to index 2
print(student_list[:3])
#Printing the elements from index 0 to the last index
print(student_list[:])
#Printing the elements from -3 to -1
print(student_list[-3:])

# Append one student's details
student_list.append(['narendra',22, 'male',89])
print(student_list)

# Adding multiple students at a time using Extend method
student_list = [['john',20, 'male',70], ['josh',21, 'male',77]]
#print the length: before extend
print(len(student_list))
#extend
student_list.extend([['sri',19,' female',80], ['siri',21, 'female',89]])
#print the details
print(student_list)
#print the length: after
print(len(student_list))

# Deleting
#print the data: before deleting
print(student_list)
#deleting one item
del student_list[1]
#print the data: after deleting
print(student_list)
```

Output:

```
>       [['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
>       john
20
male
70
josh
21
male
77
```

```
sri
19
female
80
siri
21
female
89
>        (['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89])
>        True
False
>        ['john', 20, 'male', 70]
['josh', 21, 'male', 77]
['sri', 19, 'female', 80]
['siri', 21, 'female', 89]
>        ['john', 20, 'male', 70]
['josh', 21, 'male', 77]
['sri', 19, 'female', 80]
['siri', 21, 'female', 89]
['siri', 21, 'female', 89]
['sri', 19, 'female', 80]
['josh', 21, 'male', 77]
['john', 20, 'male', 70]
>        [['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80]]
[['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
[['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80]]
[['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
[['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
>        [['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80],
['siri', 21, 'female', 89],['narendra', 22, 'male', 89]]
>        2
[['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
4
>        [['john', 20, 'male', 70], ['josh', 21, 'male', 77], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
[['john', 20, 'male', 70], ['sri', 19, 'female', 80], ['siri', 21, 'female', 89]]
```

Program 3: Chatbot

Aim: Write a python program to implement simple Chatbot with minimum 10 conversations.

Code:

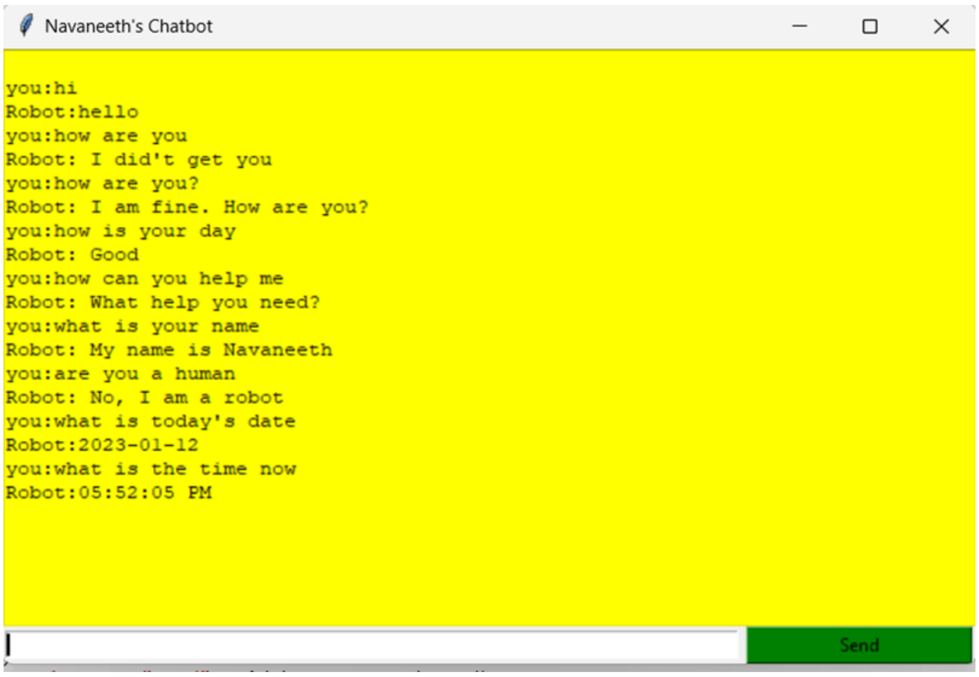
```
from tkinter import
from datetime import date
import time
root=Tk()

def send():
   send="you:"+a.get()
   text.insert('end',"\n"+send)
   if(a.get().lower()=='hi'):
      text.insert('end','\n'+"Robot:hello")
   elif(a.get().lower()=='hey'):
      text.insert('end', '\n' +"Robot:How may i help you")
   elif(a.get().lower()=='hello'):
      text.insert('end', '\n' + "Robot: hi")
   elif(a.get().lower()=='how are you?'):
      text.insert('end', '\n' + "Robot: I am fine. How are you?")
   elif(a.get().lower()=="I am fine"):
      text.insert('end', '\n' + "Robot: Nice to hear that")
   elif(a.get().lower()=='how is your day'):
      text.insert('end', '\n' + "Robot: Good")
   elif(a.get().lower()=='how can you help me'):
      text.insert('end', '\n' +"Robot: What help you need?")
   elif(a.get().lower()=='what is your name'):
      text.insert('end', '\n' + "Robot: My name is Navaneeth")
   elif(a.get().lower()=='are you a human'):
      text.insert('end', '\n' + "Robot: No, I am a robot")
   elif(a.get().lower()=="what is today's date"):
      text.insert('end', '\n' +"Robot:"+ str(date.today()))
   elif(a.get().lower()=="what is the time now"):
      text.insert('end', '\n' + "Robot:"+str(time.strftime("%I:%M:%S %p")))
   else:
      text.insert('end', '\n' + "Robot: I didn't get you")

root.title("Navaneeth's Chatbot")
text=Text(bg="yellow")
text.grid(row=0, column=0, columnspan=2)
a= Entry(root, width=80)
Send=Button(root, bg='green', text="Send", width=20,command=send)
Send.grid(row=1,column=1)
a.grid(row=1, column=0)
```

```
root.mainloop()
```

Output:



```
you:hi
Robot:hello
you:how are you
Robot: I did't get you
you:how are you?
Robot: I am fine. How are you?
you:how is your day
Robot: Good
you:how can you help me
Robot: What help you need?
you:what is your name
Robot: My name is Navaneeth
you:are you a human
Robot: No, I am a robot
you:what is today's date
Robot:2023-01-12
you:what is the time now
Robot:05:52:05 PM
```

Program 4: Set operations

Aim: Write a python program to illustrate different set operations in Jupyter notebook.

Code:

```
# Consider there is a supermarket and customers are purchasing different products.
# Assume that two customers purchased different products.
# Write the code to find the following:
c1_basket = {'brush','paste','bread','jam','soap'}
c2_basket = {'paste','brush','soap','washingpowder','biscuit'}

# i) All the products purchased by customer 1 and customer 2
print(c1_basket.union(c2_basket))

# ii) The products commonly purchased by both
print(c1_basket.intersection(c2_basket))

# iii) The products purchased by customer 1 and not by customer 2
print(c1_basket.difference(c2_basket))

# iv) The products purchased by customer 2 and not by customer 1
print(c2_basket.difference(c1_basket))

# v) Products purchased by either customer 1 or customer 2 but not both
print(c1_basket.symmetric_difference(c2_basket))

# Membership: in and not in
# Verify if the element is in the set
print("brush" in c1_basket) #True if the element present in the set
print("brush" not in c1_basket) #True if the element not present in the set
```

Output:

```
>       {'brush', 'jam', 'biscuit', 'bread', 'washingpowder', 'soap', 'paste'}
>       {'soap', 'paste', 'brush'}
>       {'bread', 'jam'}
>       {'biscuit', 'washingpowder'}
>       {'biscuit', 'bread', 'washingpowder', 'jam'}
>       True
        False
```

Program 5: Count the string

Aim: Write a program to implement a function that counts the number of times a string (s1) occurs in another string (s2).

Code:

```
def count_element(s1,s2):
    return s2.count(s1)

s2 = input("Enter the main string \n")
s1 = input("Enter the string to search \n")
print(count_element(s1,s2))
```

Output:

```
Enter the main string
hello world, welcome to this world
Enter the string to search
World
2
```

Program 6: Dictionary operations and methods

Aim: Write a program to illustrate the dictionary operations and methods in Jupyter notebook.

Code:

```
release_year_dict = {"Thriller": 1982, "Back in Black": 1980, \
                "The Dark Side of the Moon": 1973, "The Bodyguard": 1992, \
                "Bat Out of Hell": 1977, "Their Greatest Hits": 1976, \
                "Saturday Night Fever": 1977, "Rumours": 1977}

# Print the created dictionary
p=release_year_dict
print(p)

# Get value by keys
q= release_year_dict['Thriller']
print(q)

# Verify the key is in the dictionary
'The Bodyguard' in release_year_dict

# Iterate through all keys in a dictionary: Traversal
# Iterating over keys
for movie in release_year_dict:
    print(movie)

# Keys () method
keys = release_year_dict.keys()
print(keys)

# iterate through all values in a dictionary
for year in release_year_dict.values():
    print(year)

# iterate through all key, value : Traversal
# Pairs in a dictionary
print('List of given movies and their release year:')
for movie, year in release_year_dict.items():
    print(movie,":", year)
```

Output:

```
>       {'Thriller': 1982, 'Back in Black': 1980, 'The Dark Side of the Moon': 1973,
        'The Bodyguard': 1992, 'Bat Out of Hell': 1977, 'Their Greatest Hits': 1976,
         'Saturday Night Fever': 1977, 'Rumours': 1977}
>       1982
>       True
>         Thriller
        Back in Black
        The Dark Side of the Moon
The Bodyguard
        Bat Out of Hell
        Their Greatest Hits
        Saturday Night Fever
        Rumours
>         dict_keys(['Thriller', 'Back in Black', 'The Dark Side of the Moon', 'The Bodyguard',
         'Bat Out of Hell', 'Their Greatest Hits', 'Saturday Night Fever', 'Rumours'])
>         1982
        1980
        1973
        1992
        1977
        1976
        1977
        1977
>         List of given movies and their release year:
Thriller : 1982
        Back in Black : 1980
        The Dark Side of the Moon : 1973
        The Bodyguard : 1992
        Bat Out of Hell : 1977
        Their Greatest Hits : 1976
        Saturday Night Fever : 1977
        Rumours : 1977
```

Part B

Program 1: Water jug problem in AI

Aim: Write a python program to implement Water Jug Problem in Jupyter notebook.

Code:

```
# jug1 and jug2 contain the value for max capacity in respective jugs and aim is the amount of water to
be measured.
jug1, jug2, aim = 5, 3, 4

# Initialize dictionary with default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the intermediate steps to reach the final. solution and return Boolean
value
# (True if solution is possible, otherwise False).
# amt1 and amt2 are the amount of water present in both jugs at
a certain point of time.

def waterJugSolver(amt1, amt2):
    # Checks for our goal and returns true if achieved.
    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True

# Checks if we have already visited the combination or not. If not, then it proceeds further.
    if visited[(amt1, amt2)] == False:
        print(amt1, amt2)

# Changes the boolean value of the combination as it is visited.
visited[(amt1, amt2)] = True

 # Check for all the 6 possibilities and see if a solution is found
 in any one of them.

        return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)), amt2 - min(amt2, (jug1-amt1))) or
                waterJugSolver(amt1 - min(amt1, (jug2-amt2)), amt2 + min(amt1, (jug2-amt2))))

# Return False if the combination is already visited to avoid repetition otherwise, recursion will enter an
infinite loop.
```

```
    else:
        return False
print("Steps: ")

# Call the function and pass the initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

Output:

```
Steps:
0 0
5 0
5 3
0 3
3 0
3 3
5 1
0 1
1 0
1 3
4 0
True
```

Program 2: Best first search algorithm

Aim: Write a python program to demonstrate Best First Search algorithm in Jupyter notebook.

Code:

```
From queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function for implementing best first search gives output path having lowest cost
def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
# Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# Function for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

# The nodes are implemented using integers addedge(x,y,cost);
# S->0, A->1, B->2, C->3, D->4, E->5, F->6, G->7, H->8, I->9, J->10, K->11, L->12, M->13

addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

Output:

```
0 1 3 2 8 9
```

Program 3: AO* Search algorithm

Aim: Write a python program to demonstrate AO* Search algorithm in Jupyter notebook.

Code:

```
def recAOStar(n):
global finalPath
print("Expanding Node:",n)
and_nodes = []
or_nodes =[]
   if(n in allNodes):
      if 'AND' in allNodes[n]:
         and_nodes = allNodes[n]['AND']
      if 'OR' in allNodes[n]:
         or_nodes = allNodes[n]['OR']
   if len(and_nodes)==0 and len(or_nodes)==0:
      return

   solvable = False
   marked ={}

   while not solvable:
      if len(marked)==len(and_nodes)+len(or_nodes):
         min_cost_least,min_cost_group_least = least_cost_group(and_nodes,or_nodes,{})
         solvable = True
         change_heuristic(n,min_cost_least)
         optimal_child_group[n] = min_cost_group_least
         continue
      min_cost,min_cost_group = least_cost_group(and_nodes,or_nodes,marked)
      is_expanded = False
      if len(min_cost_group)>1:
         if(min_cost_group[0] in allNodes):
            is_expanded = True
            recAOStar(min_cost_group[0])
         if(min_cost_group[1] in allNodes):
            is_expanded = True
            recAOStar(min_cost_group[1])
      else:
         if(min_cost_group in allNodes):
            is_expanded = True
            recAOStar(min_cost_group)
      if is_expanded:
         min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes, or_nodes, {})
         if min_cost_group == min_cost_group_verify:
            solvable = True
```

```
                change_heuristic(n, min_cost_verify)
                optimal_child_group[n] = min_cost_group
        else:
            solvable = True
            change_heuristic(n, min_cost)
            optimal_child_group[n] = min_cost_group
        marked[min_cost_group]=1
    return heuristic(n)

def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = {}
    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
    for node in or_nodes:
        if not node in marked:
            cost = 0
            cost = cost + heuristic(node) + 1
            node_wise_cost[node] = cost
    min_cost = 999999
    min_cost_group = None
    for costKey in node_wise_cost:
        if node_wise_cost[costKey] < min_cost:
            min_cost = node_wise_cost[costKey]
            min_cost_group = costKey
    return [min_cost, min_cost_group]

def heuristic(n):
    return H_dist[n]

def change_heuristic(n, cost):
    H_dist[n] = cost
    return

def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]
    if len(node) > 1:
        if node[0] in optimal_child_group:
            print("->", end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->", end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->", end="")
            print_path(node)
H_dist = {
 'A': -1,
 'B': 4,
 'C': 2,
 'D': 3,
 'E': 6,
 'F': 8,
 'G': 2,
 'H': 0,
 'I': 0,
 'J': 0
}
allNodes = {
 'A': {'AND': [('C', 'D')], 'OR': ['B']},
 'B': {'OR': ['E', 'F']},
 'C': {'OR': ['G'], 'AND': [('H', 'I')]},
 'D': {'OR': ['J']}
}
optimal_child_group = {}
optimal_cost = recAOStar('A')
print('Nodes which gives optimal cost are')
print_path('A')
print('\nOptimal Cost is: ', optimal_cost)
```

Output:

```
Expanding Node: A
Expanding Node: B
Expanding Node: C
Expanding Node: D
Nodes which gives optimal cost are
CD->HI->J
Optimal Cost is:  5
```

Program 4: 8-Queens Problem

Aim: Write a python program to demonstrate 8-Queens Problem in Jupyter notebook.

Code:

```
# Taking number of queens as input from user
print ("Enter the number of queens")
N = int(input())

# Here we create a chessboard NxN matrix with all elements set to 0
board = [[0]*N for _ in range(N)]

def attack(i, j):
    #checking vertically and horizontally
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonally
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False

def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0
    return False

N_queens(N)
for i in board:
    print (i)
```

Output:

```
Enter the number of queens
8
```

```
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

Program 5: Travelling salesman problem

Aim: Write a python program to implement Travelling salesman problem (TSP) using heuristic approach.

Code:

```
from sys import maxsize
from itertools import permutations
V = 4

# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):

# store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

# store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
# store current Path weight(cost)
        current_pathweight = 0

# compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]

# update minimum
        min_path = min(min_path, current_pathweight)
    return min_path
# matrix representation of graph
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
```

Program 6: Nim game problem

Aim: Write a python program to implement Nim Game in Jupyter notebook.

Code:

```
print("Nim game!!\nWe are having 12 tokens")
def getTokens(curTokens):
    global tokens
    print("How many tokens would you like to take? ", end='')
    take = int(input())
    if (take < 1 or take > 3):
        print("Number must be between 1 and 3.\n")
        getTokens(curTokens)
        return
    tokens = curTokens - take
    print('You take',take ,'tokens.')
    print(tokens ,'tokens remaining.\n')
def compTurn(curTokens):
    global tokens
    take = curTokens % 4
    tokens = curTokens - take
    print ('Computer takes ',take, ' tokens.')
    print (tokens,' tokens remaining.\n')
tokens = 12
while (tokens > 0):
    getTokens(tokens)
    compTurn(tokens)
print("Computer wins!")
```

Output:

```
Nim game!!
We are having 12 tokens
How many tokens would you like to take? 2
You take 2 tokens.
10 tokens remaining.

Computer takes 2 tokens.
8 tokens remaining.

How many tokens would you like to take? 1
You take 1 token.
7 tokens remaining.

Computer takes  3  tokens.
```

```
4  tokens remaining.

How many tokens would you like to take? 2
You take 2 tokens.
2 tokens remaining.

Computer takes  2  tokens.
0  tokens remaining.

Computer wins!
```