DATE..... PAGE No.....01..... EXP. No.....01.....

Program-1

1. Write a program to implement Best First search

```
class Node:
    def __init__(self, v, weight):
        self.v = v
        self.weight = weight

class pathNode:
    def __init__(self, node, parent):
        self.node = node
        self.parent = parent

def addEdge(u,v, weight):
    adj[u].append(Node(v,weight))
adj = [ ]

def GBFS(h, v, src, dest):
    openList = [ ]
    closeList = [ ]
    openList.append(pathNode(src, None))
    while(openList):
        currentNode = openList[0]
        currentIndex = 0
        for i in range(len(openList)):
            if(h[openList[i].node]<
                h[currentNode.node]):
                currentNode = openList[i]
                currentIndex = i
```

```
                openList.pop(currentIndex)
                closeList.append(currentNode)
                if (currentNode.node == dest):
                        path=[]
                        cur = currentNode
                        while(cur1 != None):
                                path.append(cur1.node)
                                cur1 = cur1.parent
                        path.reverse()
                        return path
                for node in adj[currentNode.node]:
                        for x in openList:
                                if (x.node == node.v):
                                        continue
                        for x in closeList:
                                if (x.node == node.v):
                                        continue
                        openList.append(pathNode(node.v,
                                                currentNode))

                return []
v=10
for i in range(v):
        adj.append([])
addEdge(0,1,2)
add Edge(0,2,1)
add Edge(0,3,10)
add Edge(1,4,3)
add Edge(1,5,2)
addEdge(2,6,9)
add Edge(3,7,5)
addEdge(3,8,2)
```

output:

0 → 3 → 7 → 9

DATE ..... PAGE No. 03 EXP. NO .....

```
addEdge(7,9,5)
h= [20,22,21,10,25,24,30,5,12,1]
path = GBFS(h,V,0,9)
for i in range(len(path)-1):
        print(path[i], end = " → ")
print(path[(len(path)-1)])
```

Program : 2

2. Write a program to implement Hill climbing.

```
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]
        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]
                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]
    ]
```

output:-

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

DATE..... PAGE No.....06.....

```
print(hillClimbing(tsp))
if __name__ == "__main__":
    main()
```

DATE..... PAGE No.....07..... EXP. No.....03.....

Program - 3

3. Write a program to implement A* search algorithm

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis
    def get_neighbors(self, v):
        return self.adjac_lis[v]
    def h(self,n):
        H={
            'A':1,
            'B':1,
            'C':1,
            'D':1
          }
        return H[n]
    def a_star_algorithm(self,start,stop):
        open_lst=set([start])
        closed_lst=set([])
        poo={}
        poo[start]=0
        par={}
        par[start]=start
        while len(open_lst)>0:
            n=None
            for v in open_lst:
                if n==None or poo[v]+self.h(v)
                    <poo[n]+self.h(n):
                    n=v
```

```
if n==None:
    print("path does not exist!")
    return None
if n==stop:
    reconst_path=[]
    while par[n]!=n:
        reconst_path.append(n)
        n=par[n]
    reconst_path.append(start)
    reconst_path.reverse()
    print("path found:{}".
        format(reconst_path))
    return reconst_path
for(m,weight) in self.get_neigh
                    bors(n):
    if m not in open_lst and
        m not in closed_lst:
        open_lst.add(m)
        par[m]=n
        poo[m]=poo[n]+weight
    else:
        if poo[m]>poo[n]+weight
    else:
        if poo[m]>poo[n]+weight:
            poo[m]=poo[n]+weight
            par[m]=n
            if m in closed_lst:
                closed_lst.
                    remove(m)
                open_lst.add(m)
open_lst.remove(n)
```

output:

path found : ['A', 'B', 'D']

DATE..... PAGE No.....09..... EXP. No.....

```
                closed_lst.add(n)
        print("path not found!")
        return None
adjac_lis={
        'A': [('B',1),('C',3),('D',7)],
        'B': [('D',5)],
        'C': [('D',12)]
        }
Graph1=Graph(adjac_lis)
Graph1.a_star_algorithm('A','D')
```

DATE..... PAGE No.....10..... EXP. No.....04.....

Program-4

4. Write a program to implement AO* search algorithm

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph = graph
        self.H = heuristicNodeList
        self.start = startNode
        self.parent = { }
        self.status = { }
        self.solutionGraph = { }

    def applyAOStar(self):
        self.aoStar(self, start, False)

    def getNeighbors(self, v):
        return self.graph.get(v, '')

    def getStatus(self, v):
        return self.status.get(v, 0)

    def setStatus(self, v, val):
        self.status[v] = val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n, 0)

    def setHeuristicNodeValue(self, n, value):
        self.H[n] = value
```

```
def printSolution(self):
    print("For graph solution, traverse the
          graph from the start node:",
                                  self.start)
    print("------------")
    print(self.solutionGraph)
    print("------------")

def computeMinimumCostChildNodes(self,v):
    minimumCost=0
    costToChildNodeListDict={}
    costToChildNodeListDict[minimumCost]=[]
    flag=True
    for nodeInfoTupleList in self.getNeighbors(v):
        cost=0
        nodeList=[]
        for c,weight in nodeInfoTupleList:
            cost=cost+self.getHeuristicNode
                 Value(c)+weight nodeList
                                  .append(c)
        if flag==True:
            minimumCost=cost
            costToChildNodeListDict[mini
              mumCost]=nodeList
            flag=False
        else:
            if minimumCost>cost:
            minimumCost=cost
            costToChildNodeListDict
              [minimumCost]=nodeList
return minimumCost,costToChildNodeListDict[minimumCo
                                          st]
```

```
def aoStar(self, v, backTracking):
    print("Heuristic values:", self.H)
    print("solution graph:", self.solutionGraph)
    print("------------")
    if self.getStatus(v) >= 0:
        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        print(minimumCost, childNodeList)
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v, len(childNodeList))
        solved = True
        for childNode in childNodeList:
            self.parent[childNode] = v
            if self.getStatus(childNode) != 1:
                solved = solved & False
        if solved == True:
            self.setStatus(v, -1)
            self.solutionGraph[v] = childNodeList
        if v != self.start:
            self.aoStar(self.parent[v], True)
        if backTracking == False:
            for childNode in childNodeList:
                self.setStatus(childNode, 0)
                self.aoStar(childNode, False)
print("Graph-1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
      'H': 7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
```

output:

```
Graph-1
Heuristic values : {'A' : 1, 'B' : 6, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1,
    'G' : 5, 'H' : 7, 'I' : 7, 'J' : 1}
 solution graph : { }
 processing node : A
 - - - - - - - - - -
 10 ['B', 'C']
Heuristic values : {'A' : 10, 'B' : 6, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1,
        'G' : 5, 'H' : 7, 'I' : 7, 'J' : 1}
 solution graph : { }
 Processing node : B
 - - - - - - - - - -
  6 ['G']
Heuristic values : {'A' : 10, 'B' : 6, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1, 'G' : 5,
        'H' : 7, 'I' : 7, 'J' : 1}
 Solution Graph : { }
 Processing node : A
 - - - - - - - - - -
 10 ['B', 'C']
Heuristic values : {'A' : 10; 'B' : 6, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1,
        'G' : 5, 'H' : 7, 'I' : 7, 'J' : 1}
 solution graph : { }
 processing node : G
 - - - - - - - - - -
  8 ['I']
Heuristic values : {'A' : 10, 'B' : 6, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1,
    'G' : 8, 'H' : 7, 'I' : 7, 'J' : 1}
 solution graph : { }
 processing node : B
 8 ['H']
Heuristic values : {'A' : 10, 'B' : 8, 'C' : 2, 'D' : 12, 'E' : 2, 'F' : 1,
```

DATE ..... PAGE No. 13 EXP. NO

```
        'D' : [[('E', 1), ('F', 1)]],
        'G' : [[('I', 1)]]
    }
G1 = Graph (graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

'G': 8, 'H': 7, 'I': 7, 'J': 1}
solution graph : { }
processing node : A

---

12 ['B', 'C']
Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 8, 'H': 7, 'I': 7, 'J': 1}
solution graph : { }
Processing node : I

---

0 [ ]
Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 8, 'H': 7, 'I': 0, 'J': 1}
solution graph : {'I': [ ]}
processing node : G

---

1 ['I']
Heuristic values : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 1, 'H': 7, 'I': 0, 'J': 1}
solution graph : {'I': [ ], 'G': ['I']}
processing node : B

---

2 ['G']
Heuristic values : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 1, 'H': 7, 'I': 0, 'J': 1}
solution graph : {'J': [ ], 'G': ['I'], 'B': ['G']}
processing node : A

---

6 ['B', 'C']
Heuristic values : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G':
1, 'H': 7, 'I': 0, 'J': 1}
solution graph : {'I': [ ], 'G': ['I'], 'B': ['G']}

DATE PAGE No 14

processing node: C

2 ['I']

Heuristic values: {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

solution graph: {'I': [], 'G': ['I'], 'B': ['G']}

processing node: A

6 ['B', 'C']

Heuristic values: {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

solution graph: {'I': [], 'G': ['I'], 'B': ['G']}

processing node: J

0 []

Heuristic values: {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

solution graph: {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

processing node: C

1 ['J']

Heuristic values: {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

solution graph: {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}

processing node: A

5 ['B', 'C']

For graph solution, traverse the graph from the start node: A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

DATE..... PAGE No. 15

output:-

* Enter the temperature in celsius : 37
37.0 degree celsius is equal to 98.6 degree fahrenheit

* Enter the temperature in celsius : -1
-1.0 degree celsius is equal to 30.2 degree fahrenheit
The temperature is below freezing point

DATE PAGE NO. 16 EXP NO 05

Program-5

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

```
celsius = int(input("enter the temperature in celsius:"))
fahrenheit = (celsius * 1.8)+32
print("%.0f degree celsius is equal to %.0f
      degree fahrenheit" %(celsius, fahrenheit))
if fahrenheit < 32:
      print("the temperature is below freezing
            point")
```

output:-

```
move disk 1 from source a to destination c
move disk 2 from source a to destination b
move disk 1 from source c to destination b
move disk 3 from source a to destination c
move disk 1 from source b to destination a
move disk 2 from source b to destination c
move disk 1 from source a to destination c
move disk 4 from source a to destination b
move disk 1 from source c to destination b
move disk 2 from source c to destination a
move disk 1 from source b to destination a
move disk 3 from source c to destination b
move disk 1 from source a to destination c
move disk 2 from source a to destination b
move disk 1 from source c to destination b
```

DATE..... PAGE No. 17 EXP. NO 06

Program-6

6. Write a program to implement the Tower of Hanoi.

```
def towerofhanoi(n,source,destination,auxiliary):
    if n==1:
        print("move disk 1 from source",source,
              "to destination",destination)
        return
    towerofhanoi(n-1,source,auxiliary,destination)
    print("move disk",n,"from source",source,
          "to destination",destination)
    towerofhanoi(n-1,auxiliary,destination,source)
n=4
towerofhanoi(n,'a','b','c')
```

Program -07

7. Write a program to solve a 4-queen problem.

```
def place(pos, cnt, a):
    for i in range(1, pos):
        if ((a[i] == a[pos]) or ((abs(a[i]-a[pos])
                == abs(i-pos)))):
            return False
    return True

def print_sol(N, cnt, a):
    cnt += 1
    print("\n\n solution ", cnt, " : \n")
    for i in range (1, N+1):
        for j in range (1, N+1):
            if (a[i] == j):
                print("q \t", end = " ")
            else:
                print ("* \t", end = " ")
        print (" ")
    return cnt

def queen(n, cnt, a):
    K = 1
    a[K] = 0
    while (K != 0):
        a[K] = a[K] + 1
        while ((a[K] <= n) and place(K, cnt, a) !=
                True):
            a[K] += 1
```

output:

solution 1:

```
*  q  *  *
*  *  *  q
q  *  *  *
*  *  q  *
```

solution 2:

```
*  *  q  *
q  *  *  *
*  *  *  q
*  q  *  *
```

total solution 2

DATE PAGE NO. 19 EXP. NO

```
if(a[k]<=n):
    if(k==n):
        cnt=print sol(n,cnt,a)
    else:
        k+=1
        a[k]=0
else:
    k-=1
return cnt
N=4
a=[0]*30
cnt=0
cnt=queen(N,cnt,a)
print("\n total solution", cnt)
```

output:

* Enter a number -1
sorry, factorial does not exist for negative numbers

* Enter a number 0
The factorial of 0 is 1

* Enter a number 4
factorial of number 4 is 24

* Enter a number 7
factorial of number 7 is 5040

DATE..... PAGE NO. 20 EXP. NO. 08

Program-08

8. Write a program to find the factorial of a number provided by the user

```
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 ("factorial of number ", num, "is", factorial)
```

output:

* How many terms? -1
Please enter a positive integer

* How many terms? 1
fibonacci sequence upto 1:
0

* How many terms? 5
fibonacci sequence:
0
1
1
2
3

DATE..... PAGE No.....21..... EXP. No.....09.....

Program - 09

9 write a program to implement fibonacci series

```
nterms = int(input("How many terms?"))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
    print("please enter positive integer")
elif nterms == 1:
    print("fibonacci sequence upto", nterms,
                                    ":")
    print(n1)
else:
    print("fibonacci sequence:")
    while count < nterms:
        print(n1)
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1
```

output:

demo.txt

get set run get moon sun moon moon

get 2
set 1
run 1
moon 3
sun 1

DATE ..... PAGE No. 22 EXP. NO. 10

Program:10

10. write a program to find the frequency distribution of words in a text file

```
import re
frequency = {}
document_text = open('demo.txt', 'r')
text_string = document_text.read().lower()
match_pattern = re.findall(r '\b [a-z] {3,15}\b',
                           text_string)
for word in match_pattern:
        count = frequency.get(word,0)
        frequency[word] = count+1
frequency_list = frequency.keys()
for words in frequency_list:
        print(words, frequency[words])
```

Program-11

11. Write a program to implement, monkey and banana problem /demonstrate

```
i=0
def Monkey_go_box(x,y):
        global i
        i=i+1
        print ("step:", i, "monkey slave", x, "go to" +y)
def Monkey_move_box(x,y):
        global i
        i=i+1
        print("step:", i, "monkey take the box from", x, "deliver below the" +y)
def Monkey_on_box():
        global i
        i=i+1
        print("step:", i, "monkey climbs up the box")
def Monkey_get_banana():
        global i
        i=i+1
        print("step:", i, "monkey picked a banana")
import sys
codeIn = sys.stdin.read()
codeInList = codeIn.split()
monkey = codeInList[0]
banana = codeInList[1]
box = codeInList[2]
print("the steps are as follows:")
```

output:

ground
ceil
window
1z

The steps are as follows:

step:1 Monkey slave ground go to window
step:2 Monkey take the box from window deliver below the ceil
Step:3 Monkey climbs up the box
step:4 Monkey picked a banana

DATE..... PAGE NO... 24

```
Monkey_go_box (monkey, box)
Monkey_move_box (box, banana)
Monkey_on_box ()
Monkey_get_banana ()
```

output:-

original list is : ['correction', 'transaction', 'station',
'location', 'computers', 'algorithms']
all strings count with given substring are : 4

DATE ..... PAGE No. 25 EXP. NO. 12

Program : 12

12 Write a program to count the number of words that contain "tion" in a given list

```
test_list = ['correction', 'transaction', 'station',
             'location', 'computers', 'algorithms']
print("original list is :" + str(test_list))
subs = "tion"
res = 0
for i in test_list:
    if i.find(subs) != -1:
        res += 1
print("all strings count with given substring are:"
      + str(res))
```

DATE..... PAGE No. 26 EXP. No. 13

Program : 13

13 Write a program to implement tree data structure

```
class BinaryTreeNode:
        def __init__(self,data):
                self.data = data
                self.leftChild = None
                self.rightChild = None
node1 = BinaryTreeNode(50)
node2 = BinaryTreeNode(20)
node3 = BinaryTreeNode(45)
node4 = BinaryTreeNode(11)
node5 = BinaryTreeNode(15)
node6 = BinaryTreeNode(30)
node7 = BinaryTreeNode(78)
node1.leftChild = node2
node1.rightChild = node3
node2.leftChild = node4
node2.rightChild = node5
node3.leftChild = node6
node3.rightChild = node7
print("Root node is :")
print(node1.data)
print("left child of node is:")
print(node1.leftChild.data)
print("right child of node is:")
print(node1.rightChild.data)

print("root node is:")
print(node2.data)
```

```
print ("left child of node is:")
print (node2.leftChild.data)
print ("right child of node is:")
print (node2.rightChild.data)

print ("root node is:")
print (node3.data)
print ("left child of node is:")
print (node3.leftChild.data)
print ("right child of node is:")
print (node3.rightChild.data)

print ("root node is:")
print (node4.data)
print ("left child of node is:")
print (node4.leftChild)
print ("right child of node is:")
print (node4.rightChild)

print ("root node is:")
print (node5.data)
print ("left child of node is:")
print (node5.leftChild)
print ("right child of node is:")
print (node5.rightChild)

print ("root node is:")
print (node6.data)
print ("left child of node is:")
print (node6.leftChild)
print ("right child of node is:")
```

output:

```
Root node is:
50
Left child of the node is:
20
Right child of node is:
45

Root node is:
20
Left child of node is:
11
Right child of node is:
15

Root node is:
45
Left child of node is:
30
Right child of node is:
78

Root node is:
11
left child of node is:
None
right child of node is:
None

root node is:
15
```

DATE..... PAGE No....28..... EXP.

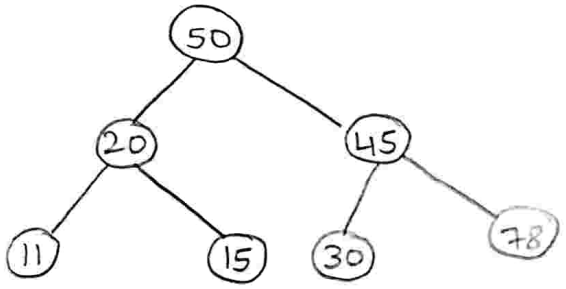
```
print (node 6.right child)

print ("root node is:")
print (node 7.data)
print ("left child of node is:")
print (node 7. left child)
print ("right child of node is:")
print (node 7. right child)
```

left child of node is:
None
right child of node is:
None

root node is:
30
left child of node is:
None
right child of node is:
None

root node is:
78
left child of node is:
None
right child of node is:
None



DATE.....

PAGE No. 29