Program-1:

Aim: -

To Implement Tic-Tac-Toe game using approach 2.

Source Code:-

```
def print_board(board):
    print(d[board[1]],'|',d[board[2]],'|',d[board[3]])
    print("--+---+--")
    print(d[board[4]],'|',d[board[5]],'|',d[board[6]])
    print("--+---+--")
    print(d[board[7]],'|',d[board[8]],'|',d[board[9]])

def check_2(x,y,z):
    if(board[x]==2):
        return x
    if(board[y]==2):
        return y
    return z

def poss_win(a,turn):
    res=turn*turn*2
    if(a[1]*a[2]*a[3]==res):
        return check_2(1,2,3)
    if(a[4]*a[5]*a[6]==res):
        return check_2(4,5,6)
    if(a[7]*a[8]*a[9]==res):
        return check_2(7,8,9)
    if(a[1]*a[4]*a[7]==res):
        return check_2(1,4,7)
    if(a[2]*a[5]*a[8]==res):
        return check_2(2,5,8)
    if(a[3]*a[6]*a[9]==res):
        return check_2(3,6,9)
    if(a[1]*a[5]*a[9]==res):
        return check_2(1,5,9)
    if(a[3]*a[5]*a[7]==res):
        return check_2(3,5,7)
    return 0

def make2(board):
    for i in (5,2,4,6,8):
        if(board[i]==2):
            return i

def computer(board,i):
    if(i==1):
        board[1]=3
        return False
```

```
if(i==2):
    if(board[5]==2):
        board[5]=3
        return False
    else:
        board[1]=3
        return False
if(i==3):
    if(board[9]==2):
        board[9]=3
        return False
    else:
        board[3]=3
        return False
if(i==4):
    x=poss_win(board,5)
    if(x!=0):
        board[x]=3
    else:
        board[make2(board)]=3
if(i==5):
    x=poss_win(board,3)
    if(x!=0):
        board[x]=3
        return True
    else:
        y=poss_win(board,5)
        if(y!=0):
            board[y]=3
            return False
        elif(board[7]==2):
            board[7]=3
            return False
        else:
            board[3]=3
            return False
if(i==6):
    x=poss_win(board,3)
    if(x!=0):
        board[x]=3
        return True
    else:
        y=poss_win(board,5)
        if(y!=0):
            board[y]=3
            return False
        else:
            board[make2(board)]=3
if(i==7 or i==8 or i==9):
    x=poss_win(board,3)
```

```
    if(x!=0):
        board[x]=3
        return True
    else:
        y=poss_win(board,5)
        if(y!=0):
            board[y]=3
            return False
        else:
            for j in board:
                if(board[j]==2):
                    board[j]=3
                    return False

board={1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 2}
print("this is initial board")
d={2:' ',3:'X',5:'O'}
print_board(board)
pp=int(input("who wants to play first ....press 1 for computer 2 for human:"))
print()
print()
for i in range(pp,pp+9):
    if(i%2):
        x=computer(board,i-pp+1)
        print_board(board)
        print()
        print()
        if(x==True):
            print("computer won")
            break
    else:
        mm=poss_win(board,5)
        p=int(input("enter the position you want to play : "))
        while(board[p]!=2):
            print('wrong place enter again')
            p=int(input("enter the position you want to play : "))
        board[p]=5
        print_board(board)
        print()
        print()
        if(mm==p):
            print('player won')
            break
else:
    print("Tie! well played")
```

Result:-

The Program is successfully executed.

Output-1:-

```
this is initial board
  |   |
--+---+--
  |   |
--+---+--
  |   |
who wants to play first.... press 1 for computer 2 for human:1


X |   |
--+---+--
  |   |
--+---+--
  |   |


enter the position you want to play : 3
X |   | O
--+---+--
  |   |
--+---+--
  |   |


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


enter the position you want to play : 5
X |   | O
--+---+--
  | O |
--+---+--
  |   | X


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


enter the position you want to play : 4
X |   | O
--+---+--
O | O |
--+---+--
X |   | X
X |   | O

--+---+--
```

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


computer won
```

Output-2:-

```
this is initial board
  |   |
--+---+--
  |   |
--+---+--
  |   |
who wants to play first.... press 1 for computer 2 for human:2


enter the position you want to play : 1
O |   |
--+---+--
  |   |
--+---+--
  |   |


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


enter the position you want to play : 7
O |   |
--+---+--
  | X |
--+---+--
O |   |


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


enter the position you want to play : 6
O |   |
--+---+--
X | X | O
--+---+--
O |   |
O | X |
--+---+--
```

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


enter the position you want to play : 8
O | X |
--+---+--
X | X | O
--+---+--
O | O |


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


enter the position you want to play : 3
O | X | O
--+---+--
X | X | O
--+---+--
O | O | X


Tie! well played
```

Program-2:

Aim:-

To implement Tic Tac Toe game using approach 3.

Source Code:-

```
from itertools import combinations

def value(x):
    if x==1:
        return 8
    elif x==2:
        return 1
    elif x==3:
        return 6
    elif x==4:
        return 3
    elif x==5:
        return 5
    elif x==6:
        return 7
    elif x==7:
        return 4
    elif x==8:
        return 9
    else:
        return 2

def place(x):
    if x==8:
        return 1
    elif x==1:
        return 2
    elif x==6:
        return 3
    elif x==3:
        return 4
    elif x==5:
        return 5
    elif x==7:
        return 6
    elif x==4:
        return 7
    elif x==9:
        return 8
    else:
```

```
        return 9
def display_board():
    print(d[1]+' |'+d[2]+' |'+d[3])
    print('------')
    print(d[4]+' |'+d[5]+' |'+d[6])
    print('------')
    print(d[7]+' |'+d[8]+' |'+d[9])
    print('------')
    print()

def hinput():       #takes input from human
    p=int(input("Enter the place you want : "))
    if d[p]=='':
        d[p]='X'
        X.append(value(p))
    else:
        print('It is already filled. Please enter again')
        hinput()

def cinput():     #keep input of computer
    x=check_win('c') #checks chance for win of computer
    if x!=0:
        d[x]='O'
        O.append(value(x))
        return 1
    y=check_win('h') #checks chance for win of human
    if y!=0:
        d[y]='O'
        O.append(value(y))
    else:
        for i in (5,1,3,7,9,6,4,2,8):
            if d[i]=='':
                d[i]='O'
                O.append(value(i))
                break
    return 0

def check_win(st):
    lst=[]
    if st=='h': # if 'h' it checks for possibility for human win else for computer
win
        lst=X
    else:
        lst=O
    res = list(combinations(lst, 2))
    for i in res:
```

```
        tot=sum(list(i))
        if (15-tot) in range(1,10) and d[place(15-tot)]=='':
            return place(15-tot)
    return 0
d={1:'',2:'',3:'',4:'',5:'',6:'',7:'',8:'',9:''}
s=input("Who is going to play first ! human (H) or computer (C) ...")
c=1
O=[]
X=[]
while(c<=9):
    display_board()
    if s=='H'or s=='h':
        hinput()
        s=s[0:0]+'C'
    else:
        if cinput():
            display_board()
            print('\nGame over\n')
            print('****Computer won****')
            break
        else:
            s=s[0:0]+'H'
    c+=1
if c==10:
    display_board()
    print('\nGame over\n')
    print("It's a Tie")
```

Result:-

The Program is Successfully executed.

Output-1:-

```
Who is going to play first ! human (H) or computer (C) ...c
 | |
----------------
 | |
----------------
 | |
----------------

 | |
----------------
 |O |
----------------
 | |
----------------

Enter the place you want : 1
X | |
----------------
 |O |
----------------
 | |
----------------

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

Enter the place you want : 7
X | |O
----------------
 |O |
----------------
X | |
----------------

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

Enter the place you want : 6
X | |O
----------------
O |O |X
----------------
X | |
----------------

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

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

Enter the place you want : 2
X |X |O
---------------
O |O |X
---------------
X | |O
---------------

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


Game over

It's a Tie
```

Output-2:-

```
Who is going to play first ! human (H) or computer (C) ...h
 | |
---------------
 | |
---------------
 | |
---------------

Enter the place you want : 1
X | |
---------------
 | |
---------------
 | |
---------------

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

Enter the place you want : 2
X |X |
---------------
 |O |
---------------
 | |
---------------
```

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

Enter the place you want : 6
X |X |O
---------------
 |O |X
---------------
 | |
---------------

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


Game over

****Computer won****
```

Program-3:

Aim:-

To Implement Exhaustive search techniques using

a. BFS
b. Bidirectional

a) Breadth First Search (BFS)

Source Code:-

```
class GeneralAlgorithms:
        def __init_(self,graph,start,goal):
            self.graph=graph
            self.start=start
            self.goal = goal
            self.openlist=[]
            self.closelist=[]
        def dfs(self,graph):
            openlist = []
            closelist = []
            openlist.append(self.start)
            while openlist:
                node = openlist.pop()
                print(node,end=" ")
                closelist.append(node)
                if node == self.goal:
                    print("\nSuccessfully found")
                    return
                lst=[]
                for successors in graph[node]:
                    if successors not in openlist:
                        lst.append(successors)
                openlist=openlist+lst[::-1]
            print("\nNot found")
            return
n = int(input("Enter the no.of nodes:"))
graph={}
for i in range(n):
    j=input("Enter the nodel:").strip()
    lst=input("Enter the childrens:").split()
    graph[j]=lst
start=input("Enter start node:")
goal=input("Enter goal node:")
general = GeneralAlgorithms(graph,start,goal)
general.bfs(graph)
```

Result:-

The Program is successfully executed.

Output-1:-

Enter the no.of nodes:8
Enter the nodel:A
Enter the childrens:B C
Enter the nodel:B
Enter the childrens:D E
Enter the nodel:C
Enter the childrens:F G
Enter the nodel:D
Enter the childrens:
Enter the nodel:E
Enter the childrens:H
Enter the nodel:F
Enter the childrens:
Enter the nodel:G
Enter the childrens:
Enter the nodel:H
Enter the childrens:
Enter start node:A
Enter goal node:G

A B C D E F G
Successfully found

Output-2:-

Enter the no.of nodes:7
Enter the nodel:A
Enter the childrens:B C D
Enter the nodel:B
Enter the childrens:E
Enter the nodel:C
Enter the childrens:F G
Enter the nodel:D
Enter the childrens:
Enter the nodel:E
Enter the childrens:
Enter the nodel:F
Enter the childrens:
Enter the nodel:G
Enter the childrens:
Enter start node:A
Enter goal node:F

A B C D E F
Successfully found

b) Bi-Directional:

Source Code:-

```
from collections import defaultdict

class Graph:
   def __init__(self):
      self.graph=defaultdict(list)

   def add_edge(self,u,v):
      self.graph[u].append(v)
      self.graph[v].append(u)

   def path(self,parent,i):
      t=[]
      while(parent[i]!=i):
         t.append(parent[i])
         i=parent[i]
      return t

   def Bid(self,start,goal):
      parent=[-1]*999
      parent[start]=start
      parent[goal]=goal
      Path=[]
      Open_start,Open_goal,close_start,close_goal=[start],[goal],[],[]
      p1,p2=start,goal
      print("Start   Open   close","   Goal   Open   close")
      while(1):
         p1=Open_start.pop(0)
         close_start.append(p1)
         p2=Open_goal.pop(0)
         close_goal.append(p2)
         for i in self.graph[p1]:
            if((i not in close_start) and (i not in Open_start)):
               Open_start.append(i)
               if(parent[i]==-1):
                  parent[i]=p1
               else:
                  print(p1," ",Open_start," ",close_start," ",p2," ",Open_goal," ",close_goal)
                  print("\nGoal Node found from start node")
                  Path.extend(self.path(parent,i))
                  parent[i]=p1
                  Path=self.path(parent,i)[::-1]+[i]+Path
                  print(Path)
                  return
         for i in self.graph[p2]:
            if((i not in close_goal) and (i not in Open_goal)):
               Open_goal.append(i)
```

```
                if(parent[i]==-1):
                    parent[i]=p2
                else:
                    print(p2," ",Open_start," ",close_start," ",p2," ",Open_goal," ",close_goal)
                    print("\nGoal Node found from goal node")
                    Path.extend(self.path(parent,i))
                    parent[i]=p2
                    Path=Path[::-1]+[i]+self.path(parent,i)
                    print(Path)
                    return
            print(p1," ",Open_start," ",close_start," ",p2," ",Open_goal," ",close_goal)
g=Graph()
n=int(input("Enter the Number of edges:"))
for _ in range(n):
    u,v=map(int,input("Enter the Edge:").split())
    g.add_edge(u,v)
start,goal=map(int,input("Enter the start Node and Goal Node:\n").split())
g.Bid(start,goal)
```

Result:-

The Program is successfully executed.

Output-1:-

Enter the Number of edges:7
Enter the Edge:1 2
Enter the Edge:1 3
Enter the Edge:2 4
Enter the Edge:2 5
Enter the Edge:3 6
Enter the Edge:3 7
Enter the Edge:5 8
Enter the start Node and Goal Node:
1 7
Start Open close Goal Open close
7 [2, 3] [1] 7 [3] [7]

Goal Node found from goal node
[1, 3, 7]

Output-2:-

Enter the Number of edges:9
Enter the Edge:1 2
Enter the Edge:1 3
Enter the Edge:2 4
Enter the Edge:2 5
Enter the Edge:3 6
Enter the Edge:3 7
Enter the Edge:6 8
Enter the Edge:7 9
Enter the Edge:7 10
Enter the start Node and Goal Node:
1 9
Start Open close Goal Open close
1 [2, 3] [1] 9 [7] [9]
7 [3, 4, 5] [1, 2] 7 [3] [9, 7]

Goal Node found from goal node
[1, 3, 7, 9]

Program-4:

Aim:-

To Implement Exhaustive search techniques using

a. DFS
b. Depth First Iterative Deepening (DFID)

a) Depth First Search (DFS)

Source Code:-

```
class GeneralAlgorithms:
    def __init_(self,graph,start,goal):
        self.graph=graph
        self.start=start
        self.goal = goal
        self.openlist=[]
        self.closelist=[]
    def dfs(self,graph):
        openlist = []
        closelist = []
        openlist.append(self.start)
        while openlist:
            node = openlist.pop()
            print(node,end=" ")
            closelist.append(node)
            if node == self.goal:
                print("\nSuccessfully found")
                return
            lst=[]
            for successors in graph[node]:
                if successors not in openlist:
                    lst.append(successors)
            openlist=openlist+lst[::-1]
        print("\nNot found")
        return
n = int(input("Enter the no.of nodes:"))
graph={}
for i in range(n):
  j=input("Enter the nodel:").strip()
  lst=input("Enter the childrens:").split()
  graph[j]=lst
start=input("Enter start node:")
goal=input("Enter goal node:")
general = GeneralAlgorithms(graph,start,goal)
general.dfs(graph)
```

Result:-

The Program is successfully executed.

Output-1:-

Enter the no.of nodes:8
Enter the nodel:A
Enter the childrens:B C
Enter the nodel:B
Enter the childrens:D E
Enter the nodel:C
Enter the childrens:F G
Enter the nodel:D
Enter the childrens:
Enter the nodel:E
Enter the childrens:H
Enter the nodel:F
Enter the childrens:
Enter the nodel:G
Enter the childrens:
Enter the nodel:H
Enter the childrens:
Enter start node:A
Enter goal node:G

A B D E H C F G
Successfully found

Output-2:-

Enter the no.of nodes:7
Enter the nodel:A
Enter the childrens:B C D
Enter the nodel:B
Enter the childrens:E
Enter the nodel:C
Enter the childrens:F G
Enter the nodel:D
Enter the childrens:
Enter the nodel:E
Enter the childrens:
Enter the nodel:F
Enter the childrens:
Enter the nodel:G
Enter the childrens:
Enter start node:A
Enter goal node:F

A B E C F
Successfully found

b) Depth First Iterative Deepening (DFID):

Source Code:-

```
from collections import defaultdict

class Graph:

    def __init__(self):
        self.graph=defaultdict(list)
    def add_edge(self,u,v):
        self.graph[u].append([v,0])
        self.graph[v].append([u,0])

    def dfs(self,start,goal,i):
        Open,close=[],[]
        Open.append([start,0])
        while(Open):
            u,v=Open.pop()
            close.append(u)
            if(u==goal):
                print(u,"        ",Open,"        ",close)
                return 1
            if(v<i):
                for j in self.graph[u]:
                    j[1]=v+1
                    if(j[0] not in close):
                        Open.append(j)
            print(u,"        ",Open,"        ",close)
        return 0

    def dfid(self,start,goal):
        x=0
        i=0
        while(x!=1):
            print("DFID with depth = ",i)
            x=self.dfs(start,goal,i)
            i+=1
        print("\nGoal Found")
g=Graph()
n=int(input("Enter the number of Edges:"))
for i in range(n):
    u,v=map(int,input("Enter the Edge:").split())
    g.add_edge(u,v)
start,goal=map(int,input("Enter the Start Node And Goal Node:\n").split())
g.dfid(start,goal)
```

Result:-

The Program is successfully executed.

Output-1:-

```
Enter the number of Edges:5
Enter the Edge:1 2
Enter the Edge:1 3
Enter the Edge:2 4
Enter the Edge:2 5
Enter the Edge:5 1
Enter the Start Node And Goal Node:
1 5
DFID with depth = 0
1	[]	[1]
DFID with depth =  1
1	[[2, 1], [3, 1], [5, 1]]	[1]
5	[[2, 1], [3, 1]]	[1, 5]

Goal Found
```

Output-2:-

```
Enter the number of Edges:9
Enter the Edge:1 2
Enter the Edge:1 3
Enter the Edge:2 4
Enter the Edge:2 5
Enter the Edge:3 6
Enter the Edge:3 7
Enter the Edge:6 8
Enter the Start Node And Goal Node:
1 9
DFID with depth = 0
1	[]	[1]
DFID with depth =  1
1	[[2, 1], [3, 1]]	[1]
3	[[2, 1]]	[1, 3]
2	[]	[1, 3, 2]
DFID with depth =  2
1	[[2, 1], [3, 1]]	[1]
3	[[2, 1], [6, 2], [7, 2]]	[1, 3]
7	[[2, 1], [6, 2]]	[1, 3, 7]
6	[[2, 1]]	[1, 3, 7, 6]
2	[[4, 2], [5, 2]]	[1, 3, 7, 6, 2]
5	[[4, 2]]	[1, 3, 7, 6, 2, 5]
4	[]	[1, 3, 7, 6, 2, 5, 4]
DFID with depth =  3
1	[[2, 1], [3, 1]]	[1]
3	[[2, 1], [6, 2], [7, 2]]	[1, 3]
7	[[2, 1], [6, 2], [9, 3], [10, 3]]	[1, 3, 7]
10	[[2, 1], [6, 2], [9, 3]]	[1, 3, 7, 10]
9	[[2, 1], [6, 2]]	[1, 3, 7, 10, 9]
Goal Found
```

Program-5:

Aim:-

To Implement water jug problem with Search tree generation using BFS

Source Code:-

```
class Waterjug:

  def __init__(self,bjmax,sjmax,bj,sj,goal):
    self.bjmax=bjmax
    self.sjmax=sjmax
    self.bj=bj
    self.sj=sj
    self.goal=goal

  def bfs(self):
    open=[]
    closed=[]
    open.append((self.bj,self.sj))
    closed.append((self.bj,self.sj))

    while open:

      recordx=open.pop(0)
      print(recordx)
      bj=recordx[0]
      sj=recordx[1]

      if sj==self.goal or bj==self.goal:
        print(" successful measuring")
        return

      if bj==0 and (self.bjmax,sj) not in closed:
        open.append((self.bjmax,sj))
        closed.append((self.bjmax,sj))

      if sj==0 and (bj,self.sjmax) not in closed:
        open.append((bj,self.sjmax))
        closed.append((bj,self.sjmax))

      if bj>0 and (0,sj) not in closed:
        open.append((0,sj))
        closed.append((0,sj))

      if sj>0 and (bj,0) not in closed:
        open.append((bj,0))
        closed.append((bj,0))

      if bj>0 and sj<self.sjmax:
```

```
        if bj>=(self.sjmax-sj):
            t1=bj-(self.sjmax-sj)
            t2=self.sjmax
        else:
            t1=0
            t2=sj+bj

        if(t1,t2) not in closed:
            open.append((t1,t2))
            closed.append((t1,t2))

if __name__=='__main__':
    b = int(input("Enter big jug capacity: "))
    s = int(input("Enter small jug capacity: "))
    g = int(input("Enter the Goal capacity:"))
    waterjug=Waterjug(b,s,0,0,g)
    waterjug.bfs()
```

Result:-

The Program is successfully executed.

Output-1:-

Enter big jug capacity: 5
Enter small jug capacity: 3
Enter the Goal capacity:4
(0, 0)
(5, 0)
(0, 3)
(5, 3)
(2, 3)
(2, 0)
(0, 2)
(5, 2)
(4, 3)
successful measuring

Output-2:-

Enter big jug capacity: 7
Enter small jug capacity: 5
Enter the Goal capacity:4
(0, 0)
(7, 0)
(0, 5)
(7, 5)
(2, 5)
(2, 0)
(0, 2)
(7, 2)
(4, 5)
successful measuring

Program-6:

Aim:-

To Implement water jug problem with Search tree generation using DFS

Source Code:-

```
class Waterjug:
  def __init__(self,bjmax,sjmax,bj,sj,goal):
    self.bjmax=bjmax
    self.sjmax=sjmax
    self.bj=bj
    self.sj=sj
    self.goal=goal


  def dfs(self):
    open=[]
    closed=[]
    open.append((self.bj,self.sj))
    closed.append((self.bj,self.sj))

    while open:
      lst=[]
      recordx=open.pop()
      print(recordx)
      bj=recordx[0]
      sj=recordx[1]

      if sj==self.goal or bj==self.goal:
        print(" successful measuring")
        return


      if bj==0 and (self.bjmax,sj) not in closed:
        lst.append((self.bjmax,sj))
        closed.append((self.bjmax,sj))


      if sj==0 and (bj,self.sjmax) not in closed:
        lst.append((bj,self.sjmax))
        closed.append((bj,self.sjmax))


      if bj>0 and (0,sj) not in closed:
        lst.append((0,sj))
        closed.append((0,sj))


      if sj>0 and (bj,0) not in closed:
```

```
            lst.append((bj,0))
            closed.append((bj,0))


        if bj>0 and sj<self.sjmax:

            if bj>=(self.sjmax-sj):
                t1=bj-(self.sjmax-sj)
                t2=self.sjmax
            else:
                t1=0
                t2=sj+bj

            if(t1,t2) not in closed:
                lst.append((t1,t2))
                closed.append((t1,t2))
        open=open+lst[::-1]

if __name__=='__main__':
    b = int(input("Enter big jug capacity: "))
    s = int(input("Enter small jug capacity: "))
    g = int(input("Enter the Goal capacity:"))

    waterjug=Waterjug(b,s,0,0,g)
    waterjug.dfs()
```

Result:-

The Program is successfully executed.

Output-1:-

Enter big jug capacity: 5
Enter small jug capacity: 3
Enter the Goal capacity:4
(0, 0)
(5, 0)
(5, 3)
(2, 3)
(2, 0)
(0, 2)
(5, 2)
(4, 3)
successful measuring

Output-2:-

Enter big jug capacity: 7
Enter small jug capacity: 5
Enter the Goal capacity:4
(0, 0)
(7, 0)
(7, 5)
(2, 5)
(2, 0)
(0, 2)
(7, 2)
(4, 5)
successful measuring

Program-7:

Aim:-

To Implement Missionaries and Cannibals problem with Search tree generation using BFS.

Source Code:-

```
class MC():

  def check(self,x):
    m,n=x
    if(m[0]>=m[1] and n[0]>=n[1]):
      return True
    elif((m[0]==0) or (n[0]==0)):
      return True
    return False

  def successors(self,t,close,x):
    if(self.check(x)):
      if(x not in close):
        t.append(x)
        close.append(x)

  def conditions(self,p1,p2,Open,close):
    t=[]

    if(p1[2]==1):
      if(p1[0]>1):
        x=((p1[0]-2,p1[1],0),(p2[0]+2,p2[1],1))
        self.successors(t,close,x)
    else:
      if(p2[0]>1):
        x=((p1[0]+2,p1[1],1),(p2[0]-2,p2[1],0))
        self.successors(t,close,x)


    if(p1[2]==1):
      if(p1[0]>0 and p1[1]>0):
        x=((p1[0]-1,p1[1]-1,0),(p2[0]+1,p2[1]+1,1))
        self.successors(t,close,x)
    else:
        if(p2[0]>0 and p2[1]>0):
          x=((p1[0]+1,p1[1]+1,1),(p2[0]-1,p2[1]-1,0))
          self.successors(t,close,x)


    if(p1[2]==1):
      if(p1[1]>1):
        x=((p1[0],p1[1]-2,0),(p2[0],p2[1]+2,1))
```

```
            self.successors(t,close,x)
        else:
          if(p2[1]>1):
            x=((p1[0],p1[1]+2,1),(p2[0],p2[1]-2,0))
            self.successors(t,close,x)


        if(p1[2]==1):
          if(p1[0]>0):
            x=((p1[0]-1,p1[1],0),(p2[0]+1,p2[1],1))
            self.successors(t,close,x)
        else:
          if(p2[0]>0):
            x=((p1[0]+1,p1[1],1),(p2[0]-1,p2[1],0))
            self.successors(t,close,x)

        if(p1[2]==1):
          if(p1[1]>0):
            x=((p1[0],p1[1]-1,0),(p2[0],p2[1]+1,1))
            self.successors(t,close,x)
        else:
          if(p2[1]>0):
            x=((p1[0],p1[1]+1,1),(p2[0],p2[1]-1,0))
            self.successors(t,close,x)
        return t

    def bfs(self,start,goal):
        Open,close=[start],[start]
        p=start
        while(p!=goal and Open):
          p=Open.pop(0)
          p1,p2=p
          t=self.conditions(p1,p2,Open,close)
          Open.extend(t)
          print(p)
        print("Missionaries and Cannibals successfully crossed the River \n")

x=int(input("Enter missionaries and cannibals:\n"))
start=((x,x,1),(0,0,0))
goal=((0,0,0),(x,x,1))
g=MC()
print("Breadth First Search")
g.bfs(start,goal)
print("Depth First Search")
g.dfs(start,goal)
```

Result:-

The Program is successfully executed.

Output:-

Enter missionaries and cannibals:
3
Breadth First Search

((3, 3, 1), (0, 0, 0))
((2, 2, 0), (1, 1, 1))
((3, 1, 0), (0, 2, 1))
((3, 2, 0), (0, 1, 1))
((3, 2, 1), (0, 1, 0))
((3, 0, 0), (0, 3, 1))
((3, 1, 1), (0, 2, 0))
((1, 1, 0), (2, 2, 1))
((2, 2, 1), (1, 1, 0))
((0, 2, 0), (3, 1, 1))
((0, 3, 1), (3, 0, 0))
((0, 1, 0), (3, 2, 1))
((1, 1, 1), (2, 2, 0))
((0, 2, 1), (3, 1, 0))
((0, 0, 0), (3, 3, 1))

Missionaries and Cannibals successfully crossed the River

Program-8:

Aim:-

To Implement Missionaries and Cannibals problem with Search tree generation using DFS.

Source Code:-

```
class MC():

  def check(self,x):
    m,n=x
    if(m[0]>=m[1] and n[0]>=n[1]):
      return True
    elif((m[0]==0) or (n[0]==0)):
      return True
    return False

  def successors(self,t,close,x):
    if(self.check(x)):
      if(x not in close):
        t.append(x)
        close.append(x)

  def conditions(self,p1,p2,Open,close):
    t=[]

    if(p1[2]==1):
      if(p1[0]>1):
        x=((p1[0]-2,p1[1],0),(p2[0]+2,p2[1],1))
        self.successors(t,close,x)
    else:
      if(p2[0]>1):
        x=((p1[0]+2,p1[1],1),(p2[0]-2,p2[1],0))
        self.successors(t,close,x)

    if(p1[2]==1):
      if(p1[0]>0 and p1[1]>0):
        x=((p1[0]-1,p1[1]-1,0),(p2[0]+1,p2[1]+1,1))
        self.successors(t,close,x)
    else:
        if(p2[0]>0 and p2[1]>0):
          x=((p1[0]+1,p1[1]+1,1),(p2[0]-1,p2[1]-1,0))
          self.successors(t,close,x)


    if(p1[2]==1):
      if(p1[1]>1):
        x=((p1[0],p1[1]-2,0),(p2[0],p2[1]+2,1))
        self.successors(t,close,x)
```

```
        else:
            if(p2[1]>1):
                x=((p1[0],p1[1]+2,1),(p2[0],p2[1]-2,0))
                self.successors(t,close,x)


        if(p1[2]==1):
            if(p1[0]>0):
                x=((p1[0]-1,p1[1],0),(p2[0]+1,p2[1],1))
                self.successors(t,close,x)
        else:
            if(p2[0]>0):
                x=((p1[0]+1,p1[1],1),(p2[0]-1,p2[1],0))
                self.successors(t,close,x)


        if(p1[2]==1):
            if(p1[1]>0):
                x=((p1[0],p1[1]-1,0),(p2[0],p2[1]+1,1))
                self.successors(t,close,x)
        else:
            if(p2[1]>0):
                x=((p1[0],p1[1]+1,1),(p2[0],p2[1]-1,0))
                self.successors(t,close,x)
        return t

    def dfs(self,start,goal):
        Open,close=[start],[start]
        p=start
        while(p!=goal and Open):
            p=Open.pop()
            p1,p2=p
            t=self.conditions(p1,p2,Open,close)
            Open.extend(t[::-1])
            print(p)
        print("Missionaries and Cannibals successfully crossed the River \n")

x=int(input("Enter missionaries and cannibals:\n"))
start=((x,x,1),(0,0,0))
goal=((0,0,0),(x,x,1))
g=MC()
print("Breadth First Search")
g.bfs(start,goal)
print("Depth First Search")
g.dfs(start,goal)
```

Result:-

The Program is successfully executed.

Output:-

Enter missionaries and cannibals:
3

Depth First Search

((3, 3, 1), (0, 0, 0))
((2, 2, 0), (1, 1, 1))
((3, 2, 1), (0, 1, 0))
((3, 0, 0), (0, 3, 1))
((3, 1, 1), (0, 2, 0))
((1, 1, 0), (2, 2, 1))
((2, 2, 1), (1, 1, 0))
((0, 2, 0), (3, 1, 1))
((0, 3, 1), (3, 0, 0))
((0, 1, 0), (3, 2, 1))
((1, 1, 1), (2, 2, 0))
((0, 0, 0), (3, 3, 1))

Missionaries and Cannibals successfully crossed the River

Problem-9:

Aim:-

To Implement the following Heuristic search techniques

a. Branch-and-Bound b. Simple Hill Climbing

a) Branch and Bound:

Source Code:-

```
from collections import defaultdict

class Branch_bound():

   def __init_(self,start,goal):
      self.start=start
      self.goal=goal
      self.graph=defaultdict(list)

   def add_edge(self,u,v,k):
      self.graph[u].append([v,k])
      self.graph[v].append([u,k])

   def branch_bound(self):
      open=[]
      closed=[]
      visited=[]
      open.append([self.start,0])
      visited.append(self.start)
      found=False
      while open and found == False:
         print("open",open)
         print("closed",closed)
         p=open.pop(0)
         closed.insert(0,p[0])
         visited.append(p[0])
         if p[0]==self.goal:
            print("open",open)
            print("closed",closed)
            print("Goal node found")
            found = True
            return
         g=int(p[-1])
         for i in self.graph[p[0]]:
            if i not in closed and i not in open and i[0] not in visited:
               i[1]=i[1]+g
               open.append(i)
```

```
            open.sort(key=lambda x:x[1])
        print("Goal node not found")

if __name_=='_main_':
    n=int(input("Number of nodes:"))
    e=int(input("Number of edges:"))
    start=input("Enter start state:").upper()
    goal=input("Enter goal state:").upper()
    algo=Branch_bound(start,goal)
    for i in range(e):
        u,v,g=input("Enter parent node,child node,g-value(three values with SPACE
seperated):").split(" ")
        algo.add_edge(u.upper(),v.upper(),int(g))
    print(algo.graph)
    algo.branch_bound()
```

Result:-

The Program is successfully executed.

Output:-

```
Number of nodes:18
Number of edges:18
Enter start state:A
Enter goal state:M
Enter parent node,child node,g-value(three values with SPACE seperated):A B 5
Enter parent node,child node,g-value(three values with SPACE seperated):A C 9
Enter parent node,child node,g-value(three values with SPACE seperated):A D 12
Enter parent node,child node,g-value(three values with SPACE seperated):B E 3
Enter parent node,child node,g-value(three values with SPACE seperated):B F 5
Enter parent node,child node,g-value(three values with SPACE seperated):C G 4
Enter parent node,child node,g-value(three values with SPACE seperated):C H 5
Enter parent node,child node,g-value(three values with SPACE seperated):D I 6
Enter parent node,child node,g-value(three values with SPACE seperated):D J 7
Enter parent node,child node,g-value(three values with SPACE seperated):E K 8
Enter parent node,child node,g-value(three values with SPACE seperated):E L 6
Enter parent node,child node,g-value(three values with SPACE seperated):F M 4
Enter parent node,child node,g-value(three values with SPACE seperated):G M 7
Enter parent node,child node,g-value(three values with SPACE seperated):G N 5
Enter parent node,child node,g-value(three values with SPACE seperated):H O 6
Enter parent node,child node,g-value(three values with SPACE seperated):I P 9
Enter parent node,child node,g-value(three values with SPACE seperated):J Q 3
Enter parent node,child node,g-value(three values with SPACE seperated):J R 2
open [['A', 0]]
closed []
open [['B', 5], ['C', 9], ['D', 12]]
closed ['A']
open [['E', 8], ['C', 9], ['F', 10], ['D', 12]]
closed ['B', 'A']
open [['C', 9], ['F', 10], ['D', 12], ['L', 14], ['K', 16]]
closed ['E', 'B', 'A']
open [['F', 10], ['D', 12], ['G', 13], ['L', 14], ['H', 14], ['K', 16]]
closed ['C', 'E', 'B', 'A']
open [['D', 12], ['G', 13], ['L', 14], ['H', 14], ['M', 14], ['K', 16]]
closed ['F', 'C', 'E', 'B', 'A']
open [['G', 13], ['L', 14], ['H', 14], ['M', 14], ['K', 16], ['I', 18], ['J', 19]]
closed ['D', 'F', 'C', 'E', 'B', 'A']
open [['L', 14], ['H', 14], ['M', 14], ['K', 16], ['I', 18], ['N', 18], ['J', 19], ['M', 20]]
closed ['G', 'D', 'F', 'C', 'E', 'B', 'A']
open [['H', 14], ['M', 14], ['K', 16], ['I', 18], ['N', 18], ['J', 19], ['M', 20]]
closed ['L', 'G', 'D', 'F', 'C', 'E', 'B', 'A']
open [['M', 14], ['K', 16], ['I', 18], ['N', 18], ['J', 19], ['M', 20], ['O', 20]]
closed ['H', 'L', 'G', 'D', 'F', 'C', 'E', 'B', 'A']
open [['K', 16], ['I', 18], ['N', 18], ['J', 19], ['M', 20], ['O', 20]]
closed ['M', 'H', 'L', 'G', 'D', 'F', 'C', 'E', 'B', 'A']
Goal node found
```

b) Simple Hill Climbing:

Source Code:-

```
class Heuristic:
  def __init__(self,graph,start,goal,values):
    self.graph=graph
    self.start=start
    self.goal=goal
    self.h=values
    self.found=False
  def hill_climbing(self):
    open_list=[]
    closed_list=[]
    open_list.append((self.start,self.h[start]))
    while(open_list!=0 and self.found==False):
      if (open_list[0][0]==goal):
        self.found=True
      else:
        node=open_list.pop(0)
        print(node[0])
        lst=[]
        for successors in graph[node[0]]:
          if (successors,self.h[successors]) not in closed_list:
            lst.append((successors,self.h[successors]))
            closed_list.append((successors,self.h[successors]))
      lst.sort(key=self.second)
      open_list=lst+open_list
    if (self.found==True):
      print(self.goal)
      print("Goal Node Found")
    else:
      print("please give node that is present in graph as goal node")
if__name__=='__main__':
  n=int(input("Enter the number of nodes in a graph:"))
  values={}
  graph={}
  for i in range(n):
    x,y=input("Enter the Node and their Estimated vales:").split()
    child=input("Enter the successors of the node:").split()
    graph[x]=child
    values[x]=int(y)
  start=input("Enter the start Node:")
  goal=input("Enter the goal Node:")
  algo=Heuristic(graph,start,goal,values)
  algo.hill_climbing()
```

Result:-

The Program is successfully executed.

Output:-

Enter the number of nodes in a graph:12
Enter the Node and their Estimated vales:A 10
Enter the successors of the node:B J F
Enter the Node and their Estimated vales:B 10
Enter the successors of the node:D C
Enter the Node and their Estimated vales:J 8
Enter the successors of the node:K
Enter the Node and their Estimated vales:F 7
Enter the successors of the node:E G
Enter the Node and their Estimated vales:D 4
Enter the successors of the node:
Enter the Node and their Estimated vales:C 2
Enter the successors of the node:H
Enter the Node and their Estimated vales:K 0
Enter the successors of the node:
Enter the Node and their Estimated vales:E 5
Enter the successors of the node:I
Enter the Node and their Estimated vales:G 3
Enter the successors of the node:
Enter the Node and their Estimated vales:I 6
Enter the successors of the node:K
Enter the Node and their Estimated vales:K 0
Enter the successors of the node:
Enter the Node and their Estimated vales:H 0
Enter the successors of the node:
Enter the start Node:A
Enter the goal Node:K
A
F
G
E
I
K
Goal Node Found

Problem-10:

Aim:-

To Implement the following Heuristic search techniques

a. Beam Search b. Best-First Search

a) Beam Search:

Source code:-

```
class Heuristic:

   def second(self,elem):
      return elem[1]

   def __init__(self,graph,start,goal,values,width):
      self.graph=graph
      self.start=start
      self.goal=goal
      self.width=width
      self.h=values
      self.found=False

   def beam_search(self):
      open_list=[]
      closed_list=[]
      open_list.append((self.start,self.h[start]))

      while(open_list!=0 and self.found==False):
         print("open_list",open_list)
         print("closed_list",closed_list)
         if (open_list[0][0]==goal):
            self.found=True
            print("open_list",open_list)
            print("closed_list",closed_list)
         else:
            node=open_list.pop(0)
            print(node[0])

            lst=[]
            for successors in graph[node[0]]:
               if (successors,self.h[successors]) not in closed_list:
                  lst.append((successors,self.h[successors]))
                  closed_list.append((successors,self.h[successors]))

         lst.sort(key=self.second)
         open_list=lst+open_list
         print(lst)
         print(open_list)
```

```
        if (self.found==True):
            print(self.goal)
            print("Goal Node Found")
        else:
            print("please give node that is present in graph as goal node")


if __name__=='__main__':

    n=int(input("Enter the number of nodes in a graph:"))
    values={}
    graph={}
    for i in range(n):
        x,y=input("Enter the Node and their Estimated vales:").split()
        child=input("Enter the successors of the node:").split()
        graph[x]=child
        values[x]=int(y)
    start=input("Enter the start Node:")
    goal=input("Enter the goal Node:")
    width=input("Enter the width of the list:")
    algo=Heuristic(graph,start,goal,values,width)
    print(algo.graph)
    algo.beam_search()
```

Result:-

The Program is successfully executed.

Output:-

```
Enter the number of nodes in a graph:18
Enter the Node and their Estimated vales:A 0
Enter the successors of the node:B C D
Enter the Node and their Estimated vales:B 10
Enter the successors of the node:E F
Enter the Node and their Estimated vales:C 13
Enter the successors of the node:G
Enter the Node and their Estimated vales:D 9
Enter the successors of the node:H I J
Enter the Node and their Estimated vales:E 7
Enter the successors of the node:K L
Enter the Node and their Estimated vales:F 11
Enter the successors of the node:M
Enter the Node and their Estimated vales:G 8
Enter the successors of the node:N
Enter the Node and their Estimated vales:H 9
Enter the successors of the node:O
Enter the Node and their Estimated vales:I 4
Enter the successors of the node:P Q
Enter the Node and their Estimated vales:J 12
Enter the successors of the node:R
Enter the Node and their Estimated vales:K 9
Enter the successors of the node:
Enter the Node and their Estimated vales:L 5
Enter the successors of the node:
Enter the Node and their Estimated vales:M 7
Enter the successors of the node:
Enter the Node and their Estimated vales:N 10
Enter the successors of the node:
Enter the Node and their Estimated vales:O 12
Enter the successors of the node:
Enter the Node and their Estimated vales:P 4
Enter the successors of the node:
Enter the Node and their Estimated vales:Q 3
Enter the successors of the node:
Enter the Node and their Estimated vales:R 9
Enter the successors of the node:
Enter the start Node:A
Enter the goal Node:Q
Enter the width of the list:2
open_list [('A', 0)]
closed_list []
A
[('D', 9), ('B', 10), ('C', 13)]
[('D', 9), ('B', 10), ('C', 13)]
open_list [('D', 9), ('B', 10), ('C', 13)]
closed_list [('B', 10), ('C', 13), ('D', 9)]
D
```

```
[('I', 4), ('H', 9), ('J', 12)]
[('I', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
open_list [('I', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
closed_list [('B', 10), ('C', 13), ('D', 9), ('H', 9), ('I', 4), ('J', 12)]
I
[('Q', 3), ('P', 4)]
[('Q', 3), ('P', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
open_list [('Q', 3), ('P', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
closed_list [('B', 10), ('C', 13), ('D', 9), ('H', 9), ('I', 4), ('J', 12), ('P', 4), ('Q', 3)]
open_list [('Q', 3), ('P', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
closed_list [('B', 10), ('C', 13), ('D', 9), ('H', 9), ('I', 4), ('J', 12), ('P', 4), ('Q', 3)]
[('Q', 3), ('P', 4)]
[('Q', 3), ('P', 4), ('Q', 3), ('P', 4), ('H', 9), ('J', 12), ('B', 10), ('C', 13)]
Q
Goal Node Found
```

b) Best First Search:

Source code:-

```
class Heuristic:
    def __init__(self,graph,start,goal,values):
        self.graph=graph
        self.start=start
        self.goal=goal
        self.h=values
        self.found=False
    def bfs(self):
        open_list=[]
        closed_list=[]
        open_list.append((self.start,self.h[start]))

        while(open_list!=0 and self.found==False):
            if (open_list[0][0]==goal):
                self.found=True
            else:
                node=open_list.pop(0)
                print(node[0])
                for successors in graph[node[0]]:
                    if (successors,self.h[successors]) not in closed_list:
                        open_list.append((successors,self.h[successors]))
                        closed_list.append((successors,self.h[successors]))
            open_list.sort(key=self.second)
        if (self.found==True):
            print(self.goal)
            print("Goal Node Found")
        else:
            print("please give node that is present in graph as goal node")
if__name__=='_main__':
    n=int(input("Enter the number of nodes in a graph:"))
    values={}
    graph={}
    for i in range(n):
        x,y=input("Enter the Node and their heuristic vales:").split()
        child=input("Enter the successors of the node:").split()
        graph[x]=child
        values[x]=int(y)
    start=input("Enter the start Node:")
    goal=input("Enter the goal Node:")
    algo=Heuristic(graph,start,goal,values)
    algo.bfs()
```

Result:-

The Program is successfully executed.

Output:-

Enter the number of nodes in a graph:13
Enter the Node and their heuristic vales:A 10
Enter the successors of the node:B C D
Enter the Node and their heuristic vales:B 4
Enter the successors of the node:E F G
Enter the Node and their heuristic vales:C 5
Enter the successors of the node:H
Enter the Node and their heuristic vales:D 6
Enter the successors of the node:I J
Enter the Node and their heuristic vales:E 8
Enter the successors of the node:K
Enter the Node and their heuristic vales:F 7
Enter the successors of the node:L M
Enter the Node and their heuristic vales:G 9
Enter the successors of the node:
Enter the Node and their heuristic vales:H 5
Enter the successors of the node:
Enter the Node and their heuristic vales:I 10
Enter the successors of the node:
Enter the Node and their heuristic vales:J 11
Enter the successors of the node:
Enter the Node and their heuristic vales:K 7
Enter the successors of the node:
Enter the Node and their heuristic vales:L 4
Enter the successors of the node:
Enter the Node and their heuristic vales:M 0
Enter the successors of the node:
Enter the start Node:A
Enter the goal Node:M
A
B
C
H
D
F
M
Goal Node Found

Problem-11:

Aim:-

To Implement the following

a. A* algorithm
b. 8-puzzle problem using A* algorithm

a) A* Algorithm:

Source Code:-

```
from collections import defaultdict
class Graph:
  def __init_(self):
    self.graph=defaultdict(dict)
    self.h=defaultdict(lambda : 0)

  def add_edge(self,u,v,w):
    self.graph[u][v]=w
    self.graph[v][u]=w

  def add_h(self,u,h):
    self.h[u]=h

  def in_Open(self,Open,i,p,d):
    if(Open[i][0]>d+self.graph[p][i]+self.h[i]):
      Open[i][2]=p
      Open[i][0]=d+self.graph[p][i]+self.h[i]
      Open[i][3]=d+self.graph[p][i]

  def in_close(self,close,Open,i,p,d):
    if(close[i][1]>d+self.graph[p][i]+self.h[i]):
      x=close[i][1]-(d+self.graph[p][i]+self.h[i])
      close[i][0]=p
      close[i][1]=d+self.graph[p][i]+self.h[i]
      d_open,d_close=[[i,d+self.graph[p][i]]],[p,i]
      while(d_open):
        element,d=d_open.pop()
        for s in self.graph[element]:
          if s not in d_close:
            if s in Open:
              if(Open[s][0]>d+self.graph[element][s]+self.h[s]):
                Open[s][0]=d+self.graph[element][s]+self.h[s]
                Open[s][3]=d+self.graph[element][s]
                d_open.append([s,d+self.graph[element][s]])
```

```
                elif(s in close):
                    if(close[s][1]>d+self.graph[element][s]+self.h[s]):
                        close[s][1]-=x
                        d_open.append([s,d+self.graph[element][s]])
                else:
                    pass
                d_close.append(s)

    def A_star(self,start,goal):
        Open,close={start:[self.h[start],start,-1,0]},{}
        p=start
        while(Open):
            Open=list(Open.items())
            p,arr=Open.pop(0)
            f,p,parent,d=arr
            Open=dict(Open)
            close[p]=[parent,f]
            if(p==goal):
                print(p,'      ',Open,'        ',close)
                print('Goal Node found')
                return
            for i in self.graph[p]:
                if parent!=i:
                    if i in Open:
                        self.in_Open(Open,i,p,d)
                    if i in close:
                        self.in_close(close,Open,i,p,d)
                    if i not in Open and i not in close:
                        Open[i]=[d+self.graph[p][i]+self.h[i] , i , p , d+self.graph[p][i]]
            Open=dict(sorted(Open.items(), key =lambda kv:(kv[1], kv[0])))
            print(p,'      ',Open,'        ',close)

g=Graph()
n=input('enter edges and their weights separated by coma : ').split(',')
for i in n:
    u,v,w=map(int,i.split())
    g.add_edge(u,v,w)
print(list(g.graph.items()))
start,goal=map(int,input('enter start,goal').split())
for i in g.graph:
    g.add_h(i,int(input('enter heuristic of node '+str(i)+' : ')))
g.add_h(goal,0)
g.A_star(start,goal)
```

Result:-

The Program is Successfully executed.

Output:-

enter edges and their weights separated by coma : 1 2 6,2 3 4,3 4 3,1 5 2,1 6 7,6 7 1

[(1, {2: 6, 5: 2, 6: 7}), (2, {1: 6, 3: 4}), (3, {2: 4, 4: 3}), (4, {3: 3}), (5, {1: 2}), (6, {1: 7, 7: 1}), (7, {6: 1})]

enter start,goal1 7

enter heuristic of node 1 : 10

enter heuristic of node 2 : 6

enter heuristic of node 3 : 12

enter heuristic of node 4 : 15

enter heuristic of node 5 : 4

enter heuristic of node 6 : 3

enter heuristic of node 7 : 0

1 {5: [6, 5, 1, 2], 6: [10, 6, 1, 7], 2: [12, 2, 1, 6]} {1: [-1, 10]}

5 {6: [10, 6, 1, 7], 2: [12, 2, 1, 6]} {1: [-1, 10], 5: [1, 6]}

6 {7: [8, 7, 6, 8], 2: [12, 2, 1, 6]} {1: [-1, 10], 5: [1, 6], 6: [1, 10]}

7 {2: [12, 2, 1, 6]} {1: [-1, 10], 5: [1, 6], 6: [1, 10], 7: [6, 8]}

Goal Node found

b) 8-Puzzle Program Using A* Algorithm:

source code:-

```
from collections import defaultdict

class A_star8:

    def __init_(self,start,goal):
        self.goal=goal
        self.start=start
        self.d={
            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]
        }

    def gen_succ(self,x,index,pos):
        x=list(x)
        x[index],x[pos]=x[pos],x[index]
        return "".join(x)

    def h_value(self,x):
        count=0
        for i in range(9):
            if x[i]!=self.goal[i]:
                count+=1
        count=count-1 if count!=0 else 0
        return count

    def a_star8(self):
        open=[[self.start,0,self.h_value(self.start)]]
        closed=[]
        print("___"*40)
        while open:
            print("open::",open)
            print("closed::",closed)
            print("___"*40)
            x,g,h=open.pop(0)
            closed.append(x)
            if x==self.goal:
```

```
            print("goal node found")
            print("open::",open)
            print("closed::",closed)
            return
        index = x.index("0")
        le=len(self.d[index])
        for i in range(le):
            x2=self.gen_succ(x,index,self.d[index][i])
            if x2 not in closed:
                x2=[x2,g+1,self.h_value(x2)]
                if x2 not in open:
                    open.append(x2)
        open.sort(key=lambda x:x[1]+x[2])


s=input("enter the start state (like-123456780)::")
g=input("enter the goal state (like-123456780)::")
algo=A_star8(s,g)
algo.a_star8()
```

Result:-

The Program is Successfully executed.

Output:-

enter the start state (like-123456780)::123046758
enter the goal state (like-123456780)::123456780

---

open:: [['123046758', 0, 3]]
closed:: []

---

open:: [['123406758', 1, 2], ['023146758', 1, 4], ['123746058', 1, 4]]
closed:: ['123046758']

---

open:: [['123456708', 2, 1], ['023146758', 1, 4], ['123746058', 1, 4], ['103426758', 2, 3], ['123460758', 2, 3]]
closed:: ['123046758', '123406758']

---

open:: [['123456780', 3, 0], ['023146758', 1, 4], ['123746058', 1, 4], ['103426758', 2, 3], ['123460758', 2, 3], ['123456078', 3, 2]]
closed:: ['123046758', '123406758', '123456708']

---

goal node found
open:: [['023146758', 1, 4], ['123746058', 1, 4], ['103426758', 2, 3], ['123460758', 2, 3], ['123456078', 3, 2]]
closed:: ['123046758', '123406758', '123456708', '123456780']

Program-12:

Aim:-

To Implement 8-puzzle problem with the following techniques

a. Branch-and-Bound b. Simple Hill Climbing
c. Beam Search d. Best-First Search

a) 8-Puzzle problem using Branch and Bound:

Source Code:-

```
from collections import defaultdict

class Branch_bound8:

  def __init__(self,start,goal):
    self.goal=goal
    self.start=start
    self.d={
      0: [3,1],
      1: [4,0,2],
      2: [5,1],
      3: [0,6,4],
      4: [1,7,3,5],
      5: [2,8,4],
      6: [3,7],
      7: [4,6,8],
      8: [5,7]
    }

  def gen_succ(self,x,index,pos):
    x=list(x)
    x[index],x[pos]=x[pos],x[index]
    return "".join(x)

  def branch_bound8(self):
    open=[[self.start,0]]
    closed=[]
    print("___"*40)
    while open:
      print("open::",open)
      print("closed::",closed)
      print("___"*40)
      x,h=open.pop(0)
      print(x,h)
      closed.append(x)
      if x==self.goal:
        print("goal node found")
```

```
            print("open::",open)
            print("closed::",closed)
            return
        index = x.index("0")
        le=len(self.d[index])
        for i in range(le):
            x2=[self.gen_succ(x,index,self.d[index][i]),h+1]
            if x2 not in open and x2[0] not in closed:
                open.append(x2)
        open.sort(key=lambda x:x[1])


s=input("enter the start state (like-123456780)::")
g=input("enter the goal state (like-123456780)::")
algo=Branch_bound8(s,g)
algo.branch_bound8()
```

Result:-

The Programming is successfully executed.

Output:-

```
enter the start state (like-123456780)::123864705
enter the goal state (like-123456780)::123804765
________________________________________________
open:: [['123864705', 0]]
closed:: []
________________________________________________
123864705 0
open:: [['123804765', 1], ['123864075', 1], ['123864750', 1]]
closed:: ['123864705']
________________________________________________
123804765 1
goal node found
open:: [['123864075', 1], ['123864750', 1]]
closed:: ['123864705', '123804765']
```

Output-2:-

```
enter the start state (like-123456780)::123840765
enter the goal state (like-123456780)::123804765
________________________________________________
open:: [['123840765', 0]]
closed:: []
________________________________________________
123840765 0
open:: [['120843765', 1], ['123845760', 1], ['123804765', 1]]
closed:: ['123840765']
________________________________________________
120843765 1
open:: [['123845760', 1], ['123804765', 1], ['102843765', 2]]
closed:: ['123840765', '120843765']
________________________________________________
123845760 1
open:: [['123804765', 1], ['102843765', 2], ['123845706', 2]]
closed:: ['123840765', '120843765', '123845760']
________________________________________________
123804765 1
goal node found
open:: [['102843765', 2], ['123845706', 2]]
closed:: ['123840765', '120843765', '123845760', '123804765']
```

b) 8-Puzzle Problem using Simple Hill Climbing:

Source Code:-

```
from collections import defaultdict

class Hill_climbing8:

    def __init__(self,start,goal):
        self.goal=goal
        self.start=start
        self.d={
            0: [3,1],
            1: [4,0,2],
            2: [5,1],
            3: [0,6,4],
            4: [1,7,3,5],
            5: [2,8,4],
            6: [3,7],
            7: [4,6,8],
            8: [5,7]
        }

    def gen_succ(self,x,index,pos):
        x=list(x)
        x[index],x[pos]=x[pos],x[index]
        return "".join(x)

    def h_value(self,x):
        count=0
        for i in range(9):
            if x[i]!=self.goal[i]:
                count+=1
        count=count-1 if count!=0 else 0
        return count

    def hill_climbing8(self):
        open=[[self.start,self.h_value(self.start)]]
        closed=[]
        print("___"*40)
        while open:
            print("open::",open)
            print("closed::",closed)
            print("___"*40)
            x,h=open.pop(0)
            closed.insert(0,x)
            if x==self.goal:
                print("goal node found")
                print("open::",open)
```

```
                print("closed::",closed)
                return
            lst=[]
            index = x.index("0")
            le=len(self.d[index])
            for i in range(le):
                x2=self.gen_succ(x,index,self.d[index][i])
                if x2 not in open and x2[0] not in closed:
                    lst.append([x2,self.h_value(x2)])
            lst.sort(key=lambda x:x[1])
            open=lst+open


s=input("enter the start state (like-123456780)::")
g=input("enter the goal state (like-123456780)::")
algo=Hill_climbing8(s,g)
algo.hill_climbing8()
```

Result:-

The Program is successfully executed.

Output:-

```
enter the start state (like-123456780)::123046758
enter the goal state (like-123456780)::123456780
________________________________________________________________________________
____________________________________________________
open:: [['123046758', 3]]
closed:: []
________________________________________________________________________________
____________________________________________________
open:: [['123406758', 2], ['023146758', 4], ['123746058', 4]]
closed:: ['123046758']
________________________________________________________________________________
____________________________________________________
open:: [['123456708', 1], ['103426758', 3], ['123046758', 3], ['123460758', 3], ['023146758',
4], ['123746058', 4]]
closed:: ['123406758', '123046758']
________________________________________________________________________________
____________________________________________________
open:: [['123456780', 0], ['123406758', 2], ['123456078', 2], ['103426758', 3], ['123046758',
3], ['123460758', 3], ['023146758', 4], ['123746058', 4]]
closed:: ['123456708', '123406758', '123046758']
________________________________________________________________________________
____________________________________________________
goal node found
open:: [['123406758', 2], ['123456078', 2], ['103426758', 3], ['123046758', 3], ['123460758',
3], ['023146758', 4], ['123746058', 4]]
closed:: ['123456780', '123456708', '123406758', '123046758']
```

c) 8-Puzzle Problem Using Beam Search:

Source Code:-

```
from collections import defaultdict

class Beam_search8:

    def __init__(self,start,goal,beam):
        self.goal=goal
        self.start=start
        self.beam=beam
        self.d={
            0: [3,1],
            1: [4,0,2],
            2: [5,1],
            3: [0,6,4],
            4: [1,7,3,5],
            5: [2,8,4],
            6: [3,7],
            7: [4,6,8],
            8: [5,7]
        }

    def gen_succ(self,x,index,pos):
        x=list(x)
        x[index],x[pos]=x[pos],x[index]
        return "".join(x)

    def h_value(self,x):
        count=0
        for i in range(9):
            if x[i]!=self.goal[i]:
                count+=1
        count=count-1 if count!=0 else 0
        return count

    def beam_search8(self):
        open=[[self.start,self.h_value(self.start)]]
        closed=[]
        w_open=[]
        print("___"*40)
        print("open::",open)
        print("w_open::",w_open)
        print("closed::",closed)
        print("___"*40)
        while open:
            w_open=open[:self.beam]
            open.clear()
```

```
        print("open::",open)
        print("w_open::",w_open)
        print("closed::",closed)
        print("___"*40)
        while w_open:
            x,h=w_open.pop(0)
            closed.append(x)
            if x==self.goal:
                print("goal node found")
                print("open::",open)
                print("closed::",closed)
                return
            index = x.index("0")
            le=len(self.d[index])
            for i in range(le):
                x2=self.gen_succ(x,index,self.d[index][i])
                if x2 not in open and x2[0] not in closed:
                    open.append([x2,self.h_value(x2)])
        open.sort(key=lambda x:x[1])
        print("open::",open)
        print("w_open::",w_open)
        print("closed::",closed)
        print("___"*40)


s=input("enter the start state (like-123456780)::")
g=input("enter the goal state (like-123456780)::")
beam=int(input("enter beam value"))
algo=Beam_search8(s,g,beam)
algo.beam_search8()
```

Result:-

The Program is Successfully Executed.

Output:-

```
enter the start state (like-123456780)::123046758
enter the goal state (like-123456780)::023146758
enter beam value2
________________________________________________________________________
____________________________________________
open:: [['123046758', 1]]
w_open:: []
closed:: []
________________________________________________________________________
____________________________________________
open:: []
w_open:: [['123046758', 1]]
closed:: []
________________________________________________________________________
____________________________________________
open:: [['023146758', 0], ['123746058', 2], ['123406758', 2]]
w_open:: []
closed:: ['123046758']
________________________________________________________________________
____________________________________________
open:: []
w_open:: [['023146758', 0], ['123746058', 2]]
closed:: ['123046758']
________________________________________________________________________
____________________________________________
goal node found
open:: []
closed:: ['123046758', '023146758']
```

d) 8-Puzzle Problem Using Best First Search:

Source Code:-

```
from collections import defaultdict

class Best_first8:

    def __init__(self,start,goal):
        self.goal=goal
        self.start=start
        self.d={
            0: [3,1],
            1: [4,0,2],
            2: [5,1],
            3: [0,6,4],
            4: [1,7,3,5],
            5: [2,8,4],
            6: [3,7],
            7: [4,6,8],
            8: [5,7]
        }

    def gen_succ(self,x,index,pos):
        x=list(x)
        x[index],x[pos]=x[pos],x[index]
        return "".join(x)

    def h_value(self,x):
        count=0
        for i in range(9):
            if x[i]!=self.goal[i]:
                count+=1
        count=count-1 if count!=0 else 0
        return count

    def best_first8(self):
        open=[[self.start,self.h_value(self.start)]]
        closed=[]
        print("___"*40)
        while open:
            print("open::",open)
            print("closed::",closed)
            print("___"*40)
            x,h=open.pop(0)
            closed.append(x)
            if x==self.goal:
                print("goal node found")
                print("open::",open)
```

```
                print("closed::",closed)
                return
            index = x.index("0")
            le=len(self.d[index])
            for i in range(le):
                x2=self.gen_succ(x,index,self.d[index][i])
                if x2 not in open and x2[0] not in closed:
                    open.append([x2,self.h_value(x2)])
            open.sort(key=lambda x:x[1])


s=input("enter the start state (like-123456780)::")
g=input("enter the goal state (like-123456780)::")
algo=Best_first8(s,g)
algo.best_first8()
```

Result:-

The Program is Successfully executed.

Output:-

enter the start state (like-123456780)::123046758
enter the goal state (like-123456780)::123456780

---

open:: [['123046758', 3]]
closed:: []

---

open:: [['123406758', 2], ['023146758', 4], ['123746058', 4]]s
closed:: ['123046758']

---

open:: [['123456708', 1], ['103426758', 3], ['123046758', 3], ['123460758', 3], ['023146758', 4], ['123746058', 4]]
closed:: ['123046758', '123406758']

---

open:: [['123456780', 0], ['123406758', 2], ['123456078', 2], ['103426758', 3], ['123046758', 3], ['123460758', 3], ['023146758', 4], ['123746058', 4]]
closed:: ['123046758', '123406758', '123456708']

---

goal node found
open:: [['123406758', 2], ['123456078', 2], ['103426758', 3], ['123046758', 3], ['123460758', 3], ['023146758', 4], ['123746058', 4]]
closed:: ['123046758', '123406758', '123456708', '123456780']