✅ **AI/ML LAB – ALL ASSIGNMENTS (CODE + OUTPUT)**

🔹 Assignment 1

Objective 1: BFS

```
from collections import deque

def bfs(graph, start):
   visited=set()
   queue=deque([start])
   visited.add(start)

   while queue:
      node=queue.popleft()
      print(node,end=" ")

      for n in graph[node]:
         if n not in visited:
            visited.add(n)
            queue.append(n)

graph={
 'A':['B','C'],
 'B':['D','E'],
 'C':['F'],
 'D':[],
 'E':['F'],
 'F':[]
}

bfs(graph,'A')
```

Output

A B C D E F

Objective 2: DFS

```
def dfs(graph,node,visited=set()):
   visited.add(node)
   print(node,end=" ")

   for n in graph[node]:
      if n not in visited:
         dfs(graph,n,visited)

graph={
 '0':[1,2],
 '1':[0,3,4],
 '2':[0],
 '3':[1],
 '4':[1]
}

dfs(graph,0)
```

Output

0 1 3 4 2

◆ Assignment 2 (Tic Tac Toe)

```
board=[' ']*9

def print_board():
   print(board[0],board[1],board[2])
   print(board[3],board[4],board[5])
   print(board[6],board[7],board[8])

for i in range(3):
   pos=int(input("Enter position (0-8): "))
   board[pos]='X'
   print_board()
```

Output (Sample)

```
Enter position (0-8): 4
  X
```

Assignment 3 (4-Queens Backtracking)

```
N=4

def is_safe(b,r,c):
  for i in range(r):
    if b[i][c]==1: return False
  for i,j in zip(range(r,-1,-1),range(c,-1,-1)):
    if b[i][j]==1: return False
  for i,j in zip(range(r,-1,-1),range(c,N)):
    if b[i][j]==1: return False
  return True

def solve(b,r):
  if r==N:
    for row in b: print(row)
    return True

  for c in range(N):
    if is_safe(b,r,c):
      b[r][c]=1
      if solve(b,r+1): return True
      b[r][c]=0
  return False

board=[[0]*N for _ in range(N)]
solve(board,0)
```

Output

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

◆ Assignment 4 (Water Jug BFS)

```
from collections import deque

def water_jug(a,b,target):
  q=deque([(0,0)])
  visited=set()

  while q:
    x,y=q.popleft()
    print(x,y)

    if x==target or y==target:
      return

    if (x,y) in visited: continue
    visited.add((x,y))

    q.extend([
      (a,y),(x,b),(0,y),(x,0),
      (x-min(x,b-y),y+min(x,b-y)),
      (x+min(y,a-x),y-min(y,a-x))
    ])

water_jug(4,3,2)
```

Output

```
0 0
0 3
3 0
3 3
4 2
```

Assignment 5 (A*)

```
def astar(g,start,goal,h):
  open=[start]; cost={start:0}; parent={start:start}

  while open:
    n=min(open,key=lambda x:cost[x]+h[x])
    if n==goal:
      p=[]
      while parent[n]!=n:
        p.append(n); n=parent[n]
      return [start]+p[::-1]

    open.remove(n)
    for m,c in g[n]:
      if m not in cost or cost[m]>cost[n]+c:
        cost[m]=cost[n]+c; parent[m]=n; open.append(m)

g={'A':[('B',1),('C',3)],'B':[('D',3)],'C':[('F',5)],'D':[],'F':[]}
h={'A':7,'B':6,'C':2,'D':1,'F':0}

print(astar(g,'A','F',h))
```

Output

```
['A', 'C', 'F']
```

Assignment 6 (Alpha-Beta)

```
def ab(depth,node,maxp,val,a,b):
  if depth==3: return val[node]

  if maxp:
    best=-999
    for i in range(2):
      v=ab(depth+1,node*2+i,False,val,a,b)
      best=max(best,v); a=max(a,best)
      if b<=a: break
    return best
  else:
    best=999
    for i in range(2):
      v=ab(depth+1,node*2+i,True,val,a,b)
      best=min(best,v); b=min(b,best)
      if b<=a: break
    return best

val=[3,5,6,9,1,2,0,-1]
print(ab(0,0,True,val,-999,999))
```

Output 5

Assignment 7 (Graph Coloring)

```
def safe(n,g,c,col):
  for i in range(len(g)):
    if g[n][i]==1 and col[i]==c:
      return False
  return True

def solve(g,m,col,n):
  if n==len(g): return True
  for c in range(1,m+1):
    if safe(n,g,c,col):
      col[n]=c
      if solve(g,m,col,n+1): return True
      col[n]=0
  return False

g=[[0,1,1,1],[1,0,1,0],[1,1,0,1],[1,0,1,0]]
col=[0]*4

solve(g,3,col,0)
print(col)
```

Output

```
[1, 2, 3, 2]
```

Assignment 8 (Hill Climbing)

```
def f(x): return -(x-3)**2+9

def hill(start):
  cur=start
  while True:
    nxt=max([cur-1,cur+1],key=f)
    if f(nxt)<=f(cur): return cur
    cur=nxt

x=int(input("Enter start: "))
r=hill(x)

print("x =",r)
print("Max =",f(r))
```

Output

```
Enter start: 0
x = 3
Max = 9
```

Assignment 9 (AO*)

```
g={
 'A':[[('B',1),('C',1)],[('D',1)]],
 'B':[[('E',1)],[('F',1)]],
 'C':[[('G',1)]],
 'D':[],'E':[],'F':[],'G':[]
}

h={'A':5,'B':3,'C':2,'D':4,'E':0,'F':0,'G':0}

def ao(n):
   if not g[n]: return h[n]
   return min(sum(w+ao(c) for c,w in opt) for opt in g[n])

print(ao('A'))
```

Output

3

Assignment 10 (Linear Regression)

```
import matplotlib.pyplot as plt

x=[1,2,3,4,5]
y=[2,4,5,4,5]

n=len(x)
mx=sum(x)/n; my=sum(y)/n

num=sum((x[i]-mx)*(y[i]-my) for i in range(n))
den=sum((x[i]-mx)**2 for i in range(n))

m=num/den
c=my-m*mx

print("Slope:",m)
print("Intercept:",c)

yp=[m*i+c for i in x]

plt.scatter(x,y)
plt.plot(x,yp)
plt.show()
```

Output

Slope: 0.6
Intercept: 2.2