In [21]:
```
#Set 1:-
#a) Write a python program to remove punctuations from the given string?
import string
s = "Hello!!!, How are you??"
result = "".join(ch for ch in s if ch not in string.punctuation)
print(result)
```

```
Hello How are you
```

In [48]:
```
#b) Write a Python program to implement Depth First Search algorithm. Refer th
graph = {2:[3,4],3:[5],4:[6],5:[7],6:[7],7:[]}
visited = set()

def dfs(node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for n in graph[node]:
            dfs(n)

dfs(2)
```

```
2 3 5 7 4 6
```

In [23]:
```
#Set 2:-
#a) Write a python program implement tic-tac-toe using alpha beta pruning
import math

def print_board(b):
    for i in range(0,9,3):
        print(b[i], b[i+1], b[i+2])
    print()

def winner(b, p):
    c = [(0,1,2),(3,4,5),(6,7,8),
         (0,3,6),(1,4,7),(2,5,8),
         (0,4,8),(2,4,6)]
    return any(b[x]==b[y]==b[z]==p for x,y,z in c)

def moves(b):
    return [i for i in range(9) if b[i]=="-"]

def terminal(b):
    return winner(b,"X") or winner(b,"O") or "-" not in b

def minimax(b, depth, alpha, beta, maxp):
    if winner(b,"O"): return 1
    if winner(b,"X"): return -1
    if "-" not in b: return 0

    if maxp:
        v = -math.inf
        for m in moves(b):
            b[m] = "O"
            v = max(v, minimax(b, depth+1, alpha, beta, False))
            b[m] = "-"
            alpha = max(alpha, v)
            if beta <= alpha: break
        return v
    else:
        v = math.inf
        for m in moves(b):
            b[m] = "X"
            v = min(v, minimax(b, depth+1, alpha, beta, True))
            b[m] = "-"
            beta = min(beta, v)
            if beta <= alpha: break
        return v

def best_move(b):
    best = -math.inf
    move = None
    for m in moves(b):
        b[m] = "O"
        val = minimax(b, 0, -math.inf, math.inf, False)
        b[m] = "-"
        if val > best:
            best = val
            move = m
    return move

board = ["-"]*9

while True:
    print_board(board)
    if winner(board,"O"):
        print("AI wins!")
        break
    if winner(board,"X"):
        print("You win!")
        break
    if "-" not in board:
        print("Draw!")
        break

    x = int(input("Your move (0-8): "))
    if board[x] != "-":
        print("Invalid move")
        continue
    board[x] = "X"

    if terminal(board):
        continue

    ai = best_move(board)
    board[ai] = "O"
```

```
- - -
- - -
- - -

Your move (0-8): 1
O X -
- - -
- - -

Your move (0-8): 2
O X X
O - -
- - -

Your move (0-8): 3
Invalid move
O X X
O - -
- - -

Your move (0-8): 4
O X X
O X -
O - -

AI wins!
```

In [24]:
```
#b) Write a Python program to implement Simple Chatbot
while True:
    q = input("You: ")
    if "hi" in q.lower():
        print("Bot: Hello!")
    elif "bye" in q.lower():
        print("Bot: Goodbye!")
        break
    else:
        print("Bot: I don't understand.")
```

```
You: HELLO
Bot: I don't understand.
You: HI
Bot: Hello!
You: BYE
Bot: Goodbye!
```

In [25]:
```
#Set 3:-
#a) Write a Python program to accept a string. Find and print the number of up
s = input("Enter string: ")
upper = sum(1 for c in s if c.isupper())
lower = sum(1 for c in s if c.islower())
print("Upper:", upper, "Lower:", lower)
```

```
Enter string: Hello World
Upper: 2 Lower: 8
```

In [26]:
```python
#b) Write a Python program to solve tic-tac-toe problem.
import math

def print_board(b):
    for i in range(0,9,3):
        print(b[i], b[i+1], b[i+2])
    print()

def check_win(b, p):
    wins = [(0,1,2),(3,4,5),(6,7,8),
            (0,3,6),(1,4,7),(2,5,8),
            (0,4,8),(2,4,6)]
    return any(b[a]==b[b_]==b[c]==p for a,b_,c in wins)

def empty_cells(b):
    return [i for i in range(9) if b[i] == "-"]

def minimax(board, isMax):
    if check_win(board, "O"): return 1
    if check_win(board, "X"): return -1
    if "-" not in board: return 0

    if isMax:
        best = -math.inf
        for i in empty_cells(board):
            board[i] = "O"
            best = max(best, minimax(board, False))
            board[i] = "-"
        return best
    else:
        best = math.inf
        for i in empty_cells(board):
            board[i] = "X"
            best = min(best, minimax(board, True))
            board[i] = "-"
        return best

def best_move(board):
    best_val = -math.inf
    move = None
    for i in empty_cells(board):
        board[i] = "O"
        val = minimax(board, False)
        board[i] = "-"
        if val > best_val:
            best_val = val
            move = i
    return move

board = ["-"] * 9

while True:
    print_board(board)

    if check_win(board, "X"):
        print("You Win!")
        break
    if check_win(board, "O"):
        print("Computer Wins!")
        break
    if "-" not in board:
            print("Draw!")
            break

        x = int(input("Enter your move (0-8): "))
        if board[x] != "-":
            print("Invalid move")
            continue

        board[x] = "X"

        if "-" not in board or check_win(board, "X"):
            continue

        ai = best_move(board)
        board[ai] = "O"
```

```
- - -
- - -
- - -

Enter your move (0-8): 0
X - -
- O -
- - -

Enter your move (0-8): 1
X X O
- O -
- - -

Enter your move (0-8): 6
X X O
O O -
X - -

Enter your move (0-8): 5
X X O
O O X
X O -

Enter your move (0-8): 8
X X O
O O X
X O X

Draw!
```

In [27]:
```
#Set 4:-
#a) Write Python program to implement crypt arithmetic problem TWO+TWO=FOUR
import itertools

letters = ("T","W","O","F","U","R")
digits = "0123456789"

for perm in itertools.permutations(digits, len(letters)):
    m = dict(zip(letters, perm))

    if m["T"] == "0":
        continue
    if m["F"] == "0":
        continue

    TWO  = int(m["T"] + m["W"] + m["O"])
    FOUR = int(m["F"] + m["O"] + m["U"] + m["R"])

    if TWO + TWO == FOUR:
        print("Solution:")
        print("TWO =", TWO)
        print("FOUR =", FOUR)
        print(m)
        break
```

```
Solution:
TWO = 734
FOUR = 1468
{'T': '7', 'W': '3', 'O': '4', 'F': '1', 'U': '6', 'R': '8'}
```

In [28]:
```
#b) Write a Python program to implement Simple Chatbot.
while True:
    q = input("You: ")
    if "hi" in q.lower():
        print("Bot: Hello!")
    elif "bye" in q.lower():
        print("Bot: Goodbye!")
        break
    else:
        print("Bot: I don't understand.")
```

```
You: hi
Bot: Hello!
You: good day
Bot: I don't understand.
You: bye
Bot: Goodbye!
```

In [29]:
```
#Set 5:-
#a) Write a python program to generate Calendar for the given month and year?.
import calendar
y, m = 2025, 11
print(calendar.month(y, m))
```

```
   November 2025
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

In [30]:
```
#b) Write a Python program to simulate 4-Queens problem.
from itertools import permutations
n = 4
cols = range(n)
for perm in permutations(cols):
    if n == len(set(perm[i]+i for i in cols)) == len(set(perm[i]-i for i in cc
        print(perm)
```

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

In [31]:
```
#Set 6:-
#a) Write a Program to Implement Tower of Hanoi using Python
def hanoi(n, src, dst, aux):
    if n == 1:
        print(src, "->", dst)
    else:
        hanoi(n-1, src, aux, dst)
        print(src, "->", dst)
        hanoi(n-1, aux, dst, src)

hanoi(3, 'A', 'C', 'B')
```

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

In [32]:
```
#b) Write a Python program to solve tic-tac-toe problem.
import math

def print_board(b):
    for i in range(0,9,3):
        print(b[i], b[i+1], b[i+2])
    print()

def check_win(b, p):
    wins = [(0,1,2),(3,4,5),(6,7,8),
            (0,3,6),(1,4,7),(2,5,8),
            (0,4,8),(2,4,6)]
    return any(b[a]==b[b_]==b[c]==p for a,b_,c in wins)

def empty_cells(b):
    return [i for i in range(9) if b[i] == "-"]

def minimax(board, isMax):
    if check_win(board, "O"): return 1
    if check_win(board, "X"): return -1
    if "-" not in board: return 0

    if isMax:
        best = -math.inf
        for i in empty_cells(board):
            board[i] = "O"
            best = max(best, minimax(board, False))
            board[i] = "-"
        return best
    else:
        best = math.inf
        for i in empty_cells(board):
            board[i] = "X"
            best = min(best, minimax(board, True))
            board[i] = "-"
        return best

def best_move(board):
    best_val = -math.inf
    move = None
    for i in empty_cells(board):
        board[i] = "O"
        val = minimax(board, False)
        board[i] = "-"
        if val > best_val:
            best_val = val
            move = i
    return move

board = ["-"] * 9

while True:
    print_board(board)

    if check_win(board, "X"):
        print("You Win!")
        break
    if check_win(board, "O"):
        print("Computer Wins!")
        break
    if "-" not in board:
            print("Draw!")
            break

        x = int(input("Enter your move (0-8): "))
        if board[x] != "-":
            print("Invalid move")
            continue

        board[x] = "X"

        if "-" not in board or check_win(board, "X"):
            continue

        ai = best_move(board)
        board[ai] = "O"
```

```
- - -
- - -
- - -

Enter your move (0-8): 8
- - -
- O -
- - X

Enter your move (0-8): 7
- - -
- O -
O X X

Enter your move (0-8): 2
- - X
- O O
O X X

Enter your move (0-8): 3
O - X
X O O
O X X

Enter your move (0-8): 1
O X X
X O O
O X X

Draw!
```

In [33]:

```
#Set 7:-
#a) Write a python program to remove punctuations from the given string
import string
s = "Hello!!!, How are you??"
result = "".join(ch for ch in s if ch not in string.punctuation)
print(result)
```

```
Hello How are you
```

In [34]:
```
#b) Write a Python program for the following Cryptarithmetic problems. GO + TC
import itertools

letters = ("G", "O", "T", "U")
digits = "0123456789"

for perm in itertools.permutations(digits, len(letters)):
    m = dict(zip(letters, perm))

    # Leading letters cannot be zero
    if m["G"] == "0" or m["T"] == "0" or m["O"] == "0":
        continue

    GO = int(m["G"] + m["O"])
    TO = int(m["T"] + m["O"])
    OUT = int(m["O"] + m["U"] + m["T"])

    if GO + TO == OUT:
        print("Solution found:")
        print(f"GO = {GO}")
        print(f"TO = {TO}")
        print(f"OUT = {OUT}")
        print(m)
        break
```

```
Solution found:
GO = 81
TO = 21
OUT = 102
{'G': '8', 'O': '1', 'T': '2', 'U': '0'}
```

In [35]:
```
#Set 8:-
#a) Write a Program to Implement Tower of Hanoi using Python.
def hanoi(n, src, dst, aux):
    if n == 1:
        print(src, "->", dst)
    else:
        hanoi(n-1, src, aux, dst)
        print(src, "->", dst)
        hanoi(n-1, aux, dst, src)

hanoi(3, 'A', 'C', 'B')
```

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

In [37]:
```
#b) Write a Python program for the following Cryptarithmetic problemsSEND + MC
import itertools

letters = ("S","E","N","D","M","O","R","Y")
digits = "0123456789"

for perm in itertools.permutations(digits, len(letters)):
    m = dict(zip(letters, perm))

    # Leading letters cannot be zero
    if m["S"] == "0" or m["M"] == "0":
        continue

    SEND  = int(m["S"] + m["E"] + m["N"] + m["D"])
    MORE  = int(m["M"] + m["O"] + m["R"] + m["E"])
    MONEY = int(m["M"] + m["O"] + m["N"] + m["E"] + m["Y"])

    if SEND + MORE == MONEY:
        print("Solution found:")
        print(f"SEND  = {SEND}")
        print(f"MORE  = {MORE}")
        print(f"MONEY = {MONEY}")
        print("Mapping:", m)
        break
```

```
Solution found:
SEND  = 9567
MORE  = 1085
MONEY = 10652
Mapping: {'S': '9', 'E': '5', 'N': '6', 'D': '7', 'M': '1', 'O': '0', 'R':
'8', 'Y': '2'}
```

In [38]:
```
#Set 9:-
#a) Write a python program to sort the sentence in alphabetical order?
s = "hello world this is python"
words = s.split()
words.sort()
print(" ".join(words))
```

```
hello is python this world
```

In [39]:
```
#b) Write a Python program for the following Cryptorithmetic problemsCROSS+ROA
import itertools

letters = ("C","R","O","S","A","D","N","G","E")
digits = "0123456789"

for perm in itertools.permutations(digits, len(letters)):
    m = dict(zip(letters, perm))

    # Leading letters cannot be zero
    if m["C"] == "0" or m["R"] == "0" or m["D"] == "0":
        continue

    CROSS  = int(m["C"] + m["R"] + m["O"] + m["S"] + m["S"])
    ROADS  = int(m["R"] + m["O"] + m["A"] + m["D"] + m["S"])
    DANGER = int(m["D"] + m["A"] + m["N"] + m["G"] + m["E"] + m["R"])

    if CROSS + ROADS == DANGER:
        print("Solution found:")
        print(f"CROSS  = {CROSS}")
        print(f"ROADS  = {ROADS}")
        print(f"DANGER = {DANGER}")
        print("Mapping:", m)
        break
```

```
Solution found:
CROSS  = 96233
ROADS  = 62513
DANGER = 158746
Mapping: {'C': '9', 'R': '6', 'O': '2', 'S': '3', 'A': '5', 'D': '1', 'N':
'8', 'G': '7', 'E': '4'}
```

In [40]:
```
#Set 10:-
#a) Build a bot which provides all the information related to you in college
while True:
    q = input("You: ")
    if "name" in q.lower():
        print("Bot: I am your College Bot.")
    elif "college" in q.lower():
        print("Bot: Vidya Pratishthan College.")
    elif "bye" in q.lower():
        print("Bot: Goodbye!")
        break
    else:
        print("Bot: Ask about me or college.")
```

```
You: hi
Bot: Ask about me or college.
You: name
Bot: I am your College Bot.
You: ok
Bot: Ask about me or college.
You: vidya pratishthan college
Bot: Vidya Pratishthan College.
You: ok
Bot: Ask about me or college.
You: bye
Bot: Goodbye!
```

In [41]:
```
#b)  Write a Python program to solve 8-puzzle problem.
import heapq

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

moves = [(0,1),(0,-1),(1,0),(-1,0)]

def heuristic(state):
    h = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                x,y = divmod(state[i][j]-1,3)
                h += abs(x-i)+abs(y-j)
    return h

def serialize(state):
    return tuple(tuple(row) for row in state)

def find_zero(state):
    for i in range(3):
        for j in range(3):
            if state[i][j]==0:
                return i,j

def astar(start):
    pq = []
    heapq.heappush(pq, (heuristic(start), 0, start, []))
    visited = set()
    while pq:
        f,g,state,path = heapq.heappop(pq)
        if state == goal:
            return path
        visited.add(serialize(state))
        i,j = find_zero(state)
        for di,dj in moves:
            ni,nj = i+di,j+dj
            if 0<=ni<3 and 0<=nj<3:
                new_state = [row[:] for row in state]
                new_state[i][j], new_state[ni][nj] = new_state[ni][nj], new_st
                if serialize(new_state) not in visited:
                    heapq.heappush(pq, (g+1+heuristic(new_state), g+1, new_sta

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

solution = astar(start)
for step in solution:
    for row in step:
        print(row)
    print()
```

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

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

In [42]:
```
#SET -11
#a) Write a Python program to implement Mini-Max Algorithm.
def minimax(depth, node, maximizing, scores):
    if depth == 0:
        return scores[node]
    if maximizing:
        return max(minimax(depth-1, node*2, False, scores),
                   minimax(depth-1, node*2+1, False, scores))
    else:
        return min(minimax(depth-1, node*2, True, scores),
                   minimax(depth-1, node*2+1, True, scores))

scores = [3,5,6,9,1,2,0,-1]
print("Mini-Max Result:", minimax(3, 0, True, scores))
```

```
Mini-Max Result: 5
```

In [43]:
```
#b)  Write a Python program to simulate 8-Queens problem.
sol = []
row = [0]*8
def safe(r, c):
    for i in range(r):
        if row[i] == c or abs(row[i]-c) == abs(i-r):
            return False
    return True
def solve(r):
    if r == 8:
        sol.append(row.copy())
        return
    for c in range(8):
        if safe(r, c):
            row[r] = c
            solve(r+1)
solve(0)
sol[0]
```

Out[43]:
```
[0, 4, 7, 5, 2, 6, 1, 3]
```

In [44]:
```
#SET -12
#a) Write a Program to Implement Monkey Banana Problem using Python
from collections import deque
def nxt(s):
    m,b,on,ban = s
    pos=["door","window","middle"]
    r=[]
    if not on:
        for p in pos:
            if p!=m: r.append(((p,b,on,ban),"walk "+p))
    if m==b and not on:
        for p in pos:
            if p!=b: r.append(((p,p,False,ban),"push "+p))
        r.append(((m,b,True,ban),"climb"))
    if on:
        r.append(((m,b,False,ban),"down"))
    if on and m=="middle":
        r.append(((m,b,True,True),"get banana"))
    return r
def bfs(start,goal):
    q=deque([(start,[])])
    v={start}
    while q:
        s,act=q.popleft()
        if s==goal: return act
        for ns,a in nxt(s):
            if ns not in v:
                v.add(ns)
                q.append((ns,act+[a]))

start=("door","window",False,False)
goal=("middle","middle",True,True)
bfs(start,goal)
```

Out[44]:
```
['walk window', 'push middle', 'climb', 'get banana']
```

In [45]:
```
#b) Write a program to implement Iterative Deepening DFS algorithm. [Goal Node
graph = {
    "A": ["B", "C"],
    "B": ["D", "E"],
    "D": ["H", "I"],
    "E": [],
    "C": ["F", "G"],
    "F": ["K"],
    "G": [],
    "H": [],
    "I": [],
    "K": []
}

def dls(node, depth):
    if depth == 0 and node == "G":
        return [node]
    if depth > 0:
        for child in graph[node]:
            r = dls(child, depth-1)
            if r:
                return [node] + r

def iddfs(start):
    depth = 0
    while True:
        r = dls(start, depth)
        if r:
            return r
        depth += 1

iddfs("A")
```

Out[45]:
```
['A', 'C', 'G']
```

In [46]:
```
#SET- 13
#a) Write a Program to Implement Alpha-Beta Pruning using Python
def alphabeta(depth, node, alpha, beta, maximizing, scores):
    if depth == 0:
        return scores[node]
    if maximizing:
        v = -999
        v = max(v, alphabeta(depth-1, node*2, alpha, beta, False, scores))
        alpha = max(alpha, v)
        if beta <= alpha: return v
        v = max(v, alphabeta(depth-1, node*2+1, alpha, beta, False, scores))
        return v
    else:
        v = 999
        v = min(v, alphabeta(depth-1, node*2, alpha, beta, True, scores))
        beta = min(beta, v)
        if beta <= alpha: return v
        v = min(v, alphabeta(depth-1, node*2+1, alpha, beta, True, scores))
        return v
scores = [3,5,6,9,1,2,0,-1]
alphabeta(3, 0, -999, 999, True, scores)
```

Out[46]:
```
5
```

In [47]:
```
#b)  Write a Python program to implement Simple Chatbot
def chatbot():
    while True:
        x = input("You: ").lower()
        if "hello" in x:
            print("Bot: Hello!")
        elif "how are you" in x:
            print("Bot: I am fine.")
        elif "name" in x:
            print("Bot: I am a simple chatbot.")
        elif "bye" in x:
            print("Bot: Bye!")
            break
        else:
            print("Bot: I don't understand.")
chatbot()
```

```
You: hi
Bot: I don't understand.
You: hello
Bot: Hello!
You: how are you
Bot: I am fine.
You: ok
Bot: I don't understand.
You: bye
Bot: Bye!
```

In [ ]: