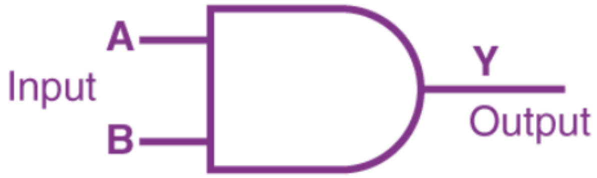
Practical 1

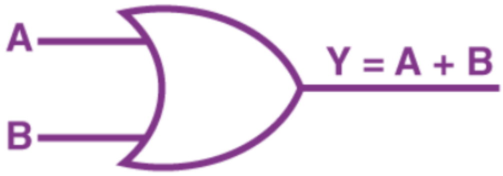
Aim: To implement logic gates like AND, OR, XOR in python and performing operations on set and list.

Theory:

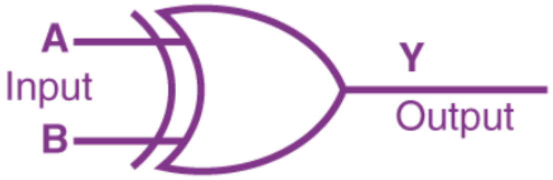
AND Gate: In the AND gate, the output of an AND gate attains state 1 if and only if all the inputs are in state 1.



OR Gate: In an OR gate, the output of an OR gate attains state 1 if one or more inputs attain state 1.



XOR Gate: In an XOR gate, the output of a two-input XOR gate attains state 1 if one adds only input attains state 1.

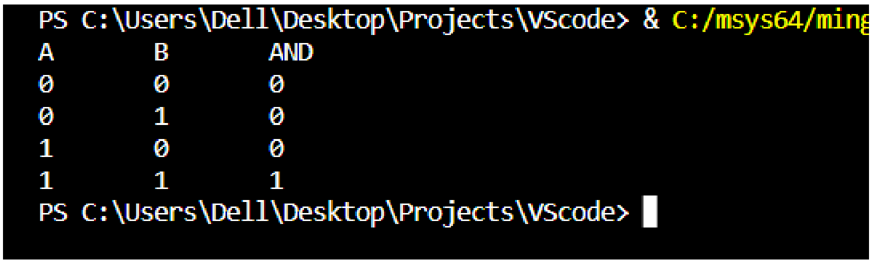


Code:

AND Gate:

```
def Andgate(a, b):
    if(a == 0 or b == 0):
        return 0
    else:
        return 1
print("A\tB\tAND")
print("0\t0\t"+str(Andgate(0,0)))
print("0\t1\t"+str(Andgate(0,1)))
print("1\t0\t"+str(Andgate(1,0)))
print("1\t1\t"+str(Andgate(1,1)))
```

Output:

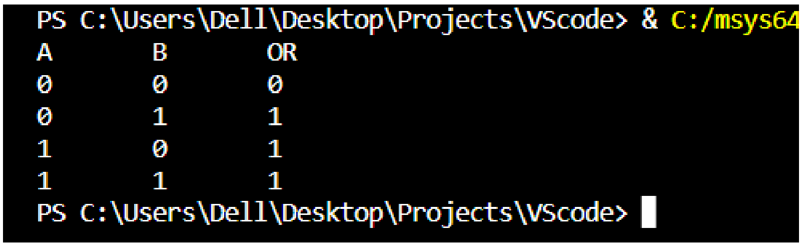


```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/ming
A       B       AND
0       0       0
0       1       0
1       0       0
1       1       1
PS C:\Users\Dell\Desktop\Projects\VScode> █
```

OR Gate:

```
def Orgate(a, b):
    if(a == 1 or b == 1):
        return 1
    else:
        return 0

print("A\tB\tOR")
print("0\t0\t"+str(Orgate(0,0)))
print("0\t1\t"+str(Orgate(0,1)))
print("1\t0\t"+str(Orgate(1,0)))
print("1\t1\t"+str(Orgate(1,1)))
```

Output:

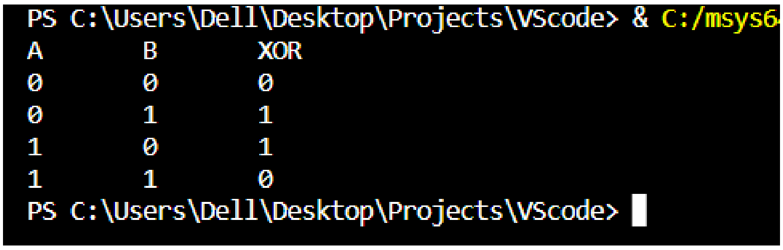


```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64
A       B       OR
0       0       0
0       1       1
1       0       1
1       1       1
PS C:\Users\Dell\Desktop\Projects\VScode> █
```

XOR Gate:

```
def Xorgate(a, b):
    if(a == b):
        return 0
    else:
        return 1


print("A\tB\tXOR")
print("0\t0\t"+str(Xorgate(0,0)))
print("0\t1\t"+str(Xorgate(0,1)))
print("1\t0\t"+str(Xorgate(1,0)))
print("1\t1\t"+str(Xorgate(1,1)))
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys6
A       B       XOR
0       0       0
0       1       1
1       0       1
1       1       0
PS C:\Users\Dell\Desktop\Projects\VScode> 
```

Operations on set:

Set: Sets are used to store multiple items in a single variable.

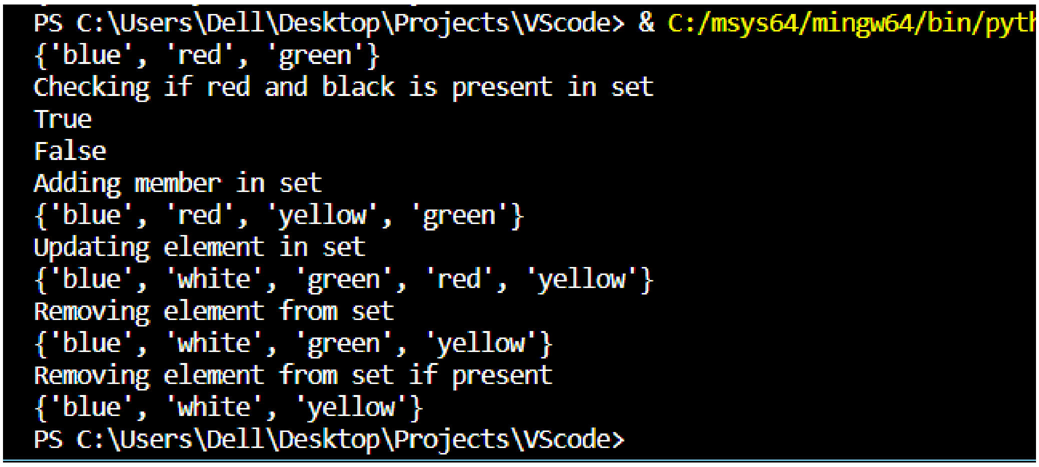
Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are list, tuple, and dictionary all with different qualities and usage. A set is a collection which is unordered, unchangeable*, and unindexed.

- Insertion in set is done through set.add() function.
- The update() method updates the current set, by adding items from another set (or any other iterable). If an item is present in both sets, only one appearance of this item will be present in the updated set.
- The remove(x) function removes the element x from a set. It returns a KeyError if x is not part of the set.
- The discard(x) method removes x from the set, but doesn't raise any error if x is not present in the set.

Code:

```
mySet = set(['red','green','blue'])
print(mySet)

print("Checking if red and black is present in set")
print('red' in mySet)
print('black' in mySet)

print("Adding member in set")
mySet.add('yellow')
print(mySet)

print("Updating element in set")
newSet = set(['white'])
mySet.update(newSet)
print(mySet)

print("Removing element from set")
mySet.remove('red')
print(mySet)

print("Removing element from set if present")
mySet.discard('red')
mySet.discard('green')
print(mySet)
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/pyth
{'blue', 'red', 'green'}
Checking if red and black is present in set
True
False
Adding member in set
{'blue', 'red', 'yellow', 'green'}
Updating element in set
{'blue', 'white', 'green', 'red', 'yellow'}
Removing element from set
{'blue', 'white', 'green', 'yellow'}
Removing element from set if present
{'blue', 'white', 'yellow'}
PS C:\Users\Dell\Desktop\Projects\VScode>
```

Operations on lists:

Lists: Lists are used to store multiple items in a single variable. List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index [0], the second item has index [1] etc.

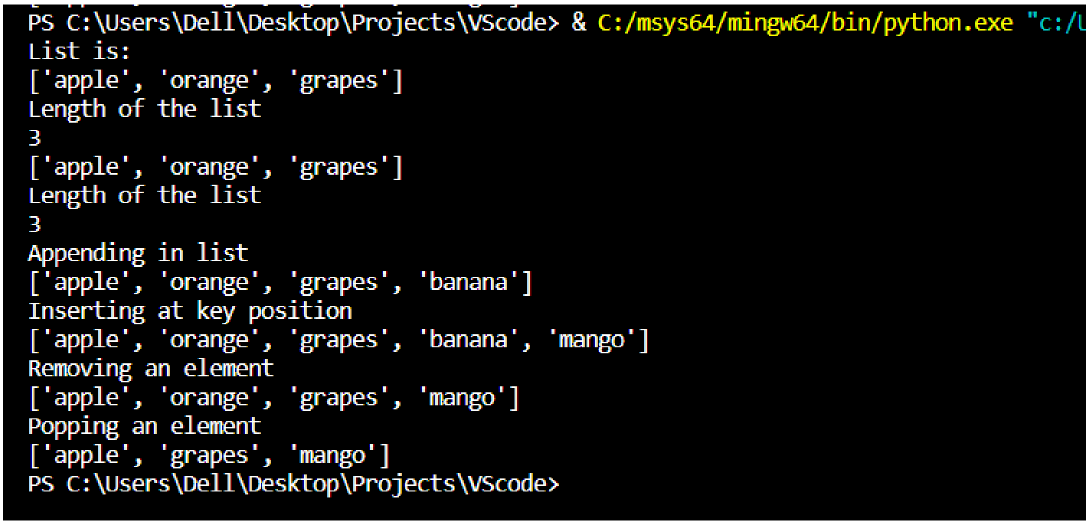
- To determine how many items a list has, use the len() function
- The append() method is used to add elements at the end of the list.
- The insert() method can add an element at a given position in the list.
- The remove() method is used to remove an element from the list.
- The method pop() can remove an element from any position in the list.

Code:

```
myList = ["apple", "orange", "grapes"]

print("List is:")

print(myList)


print("Length of the list")

print(len(myList))


print("Appending in list")

myList.append("banana")

print(myList)


print("Inserting at key position")
```

```
myList.insert(4,"mango")

print(myList)


print("Removing an element")

myList.remove("banana")

print(myList)


print("Popping an element")

myList.pop(1)

print(myList)
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/python.exe "c:/U
List is:
['apple', 'orange', 'grapes']
Length of the list
3
['apple', 'orange', 'grapes']
Length of the list
3
Appending in list
['apple', 'orange', 'grapes', 'banana']
Inserting at key position
['apple', 'orange', 'grapes', 'banana', 'mango']
Removing an element
['apple', 'orange', 'grapes', 'mango']
Popping an element
['apple', 'grapes', 'mango']
PS C:\Users\Dell\Desktop\Projects\VScode>
```

Result:

Basic logic gates like AND, OR, XOR and operations on set and list have been implemented successfully in python.

Practical 2

Aim: To implement water jug using BFS in python.

Theory:

You are given an m liter jug and a n liter jug. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d is less than n.

(X, Y) corresponds to a state where X refers to the amount of water in Jug1 and Y refers to the amount of water in Jug2
Determine the path from the initial state (xi, yi) to the final state (xf, yf), where (xi, yi) is (0, 0) which indicates both Jugs are initially empty and (xf, yf) indicates a state which could be (0, d) or (d, 0).

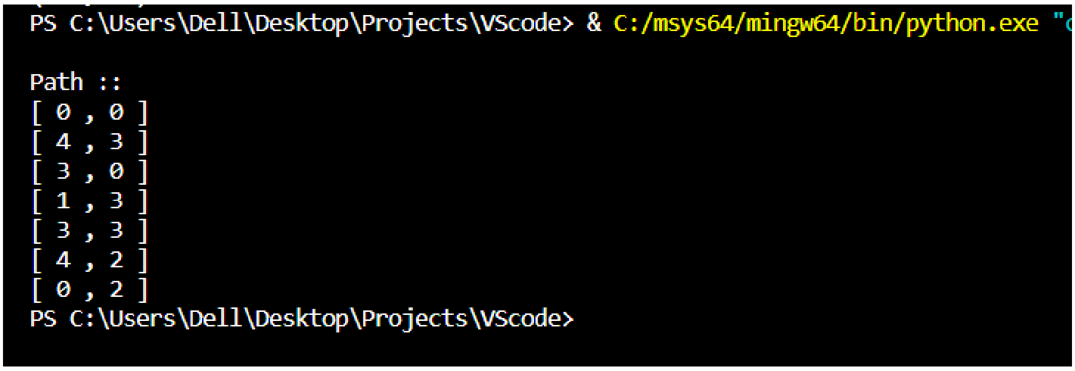
Some of the operations you can perform are:

1. Empty a Jug, (X, Y)->(0, Y) Empty Jug 1
2. Fill a Jug, (0, 0)->(X, 0) Fill Jug 1
3. Pour water from one jug to the other until one of the jugs is either empty or full, (X, Y) -> (X-d, Y+d)

Code:

```
from collections import deque

def BFS(a, b, tgt):

    s = {}

    isSolvable = False

    path = []

    w = deque()

    w.append((0, 0))

    while (len(w) > 0):

        v = w.popleft()

        if ((v[0], v[1]) in s):

            continue

        if ((v[0] > a or v[1] > b or

            v[0] < 0 or v[1] < 0)):

            continue

        path.append([v[0], v[1]])

        s[(v[0], v[1])] = 1
        if (v[0] == tgt or v[1] == tgt):
            isSolvable = True
            if (v[0] == tgt):
                if (v[1] != 0):
                    path.append([v[0], 0])
            else:
                if (v[0] != 0):
                    path.append([0, v[1]])
            n = len(path)
            for i in range(n):
                print("[", path[i][0], ",",
                    path[i][1], "]")
            break
        w.append([v[0], b])
        w.append([a, v[1]])
        for x in range(max(a, b) + 1):
            c = v[0] + x
            d = v[1] - x
            if (c == a or (d == 0 and d >= 0)):
                w.append([c, d])
            c = v[0] - x
            d = v[1] + x
            if ((c == 0 and c >= 0) or d == b):
                w.append([c, d])
        w.append([a, 0])
        w.append([0, b])
    if (not isSolvable):
        print("Not solvable")
if __name__ == '__main__':
    J1, J2, tgt = 4, 3, 2
    print("Path ::")
    BFS(J1, J2, tgt)
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/python.exe "

Path ::
[ 0 , 0 ]
[ 4 , 3 ]
[ 3 , 0 ]
[ 1 , 3 ]
[ 3 , 3 ]
[ 4 , 2 ]
[ 0 , 2 ]
PS C:\Users\Dell\Desktop\Projects\VScode>
```

Result:

Water Jug problem using BFS has been successfully implemented in python.

Practical 3

Aim: To implement depth first search in python.

Theory:

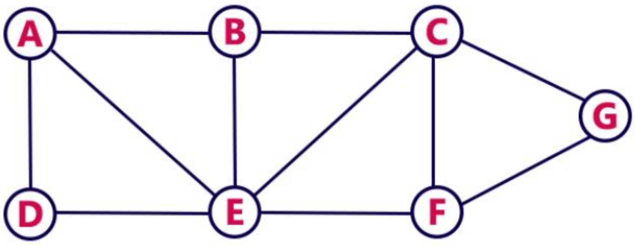
Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

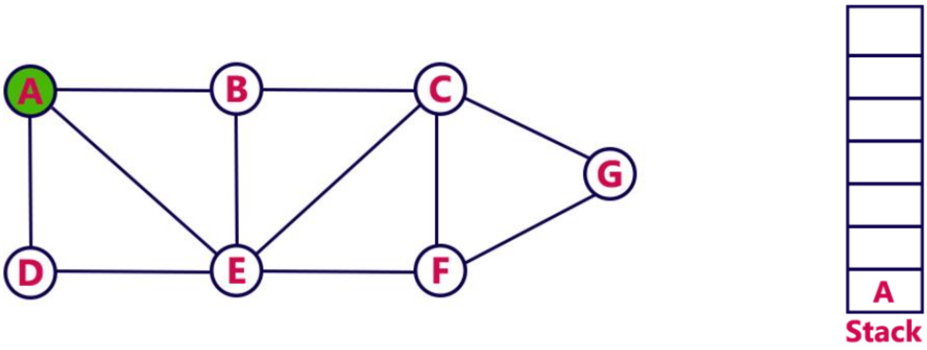
DFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops. We use Stack data structure with maximum size of total number of vertices in the graph to implement DFS traversal.

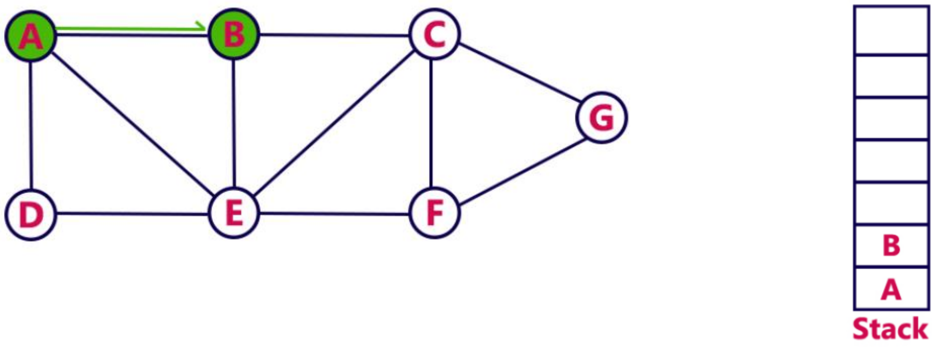
We use the following steps to implement DFS traversal-

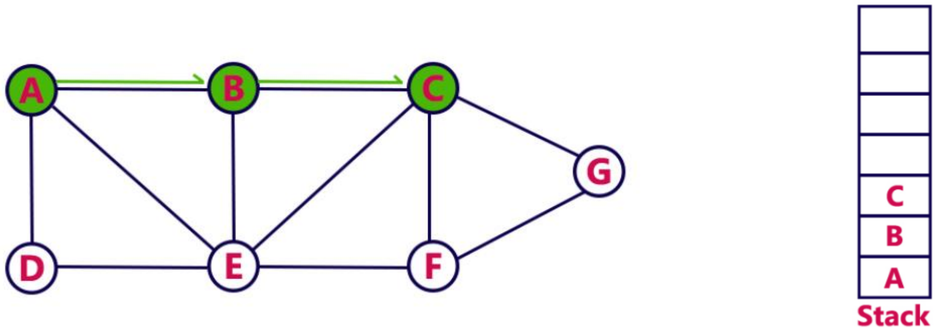
- Step 1 - Define a Stack of size total number of vertices in the graph.
- Step 2 - Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
- Step 3 - Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
- Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- Step 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
- Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
- Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

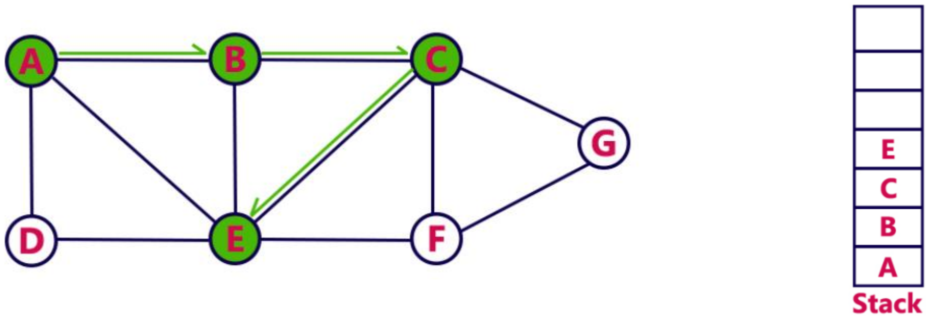
Consider the following graph for traversal:

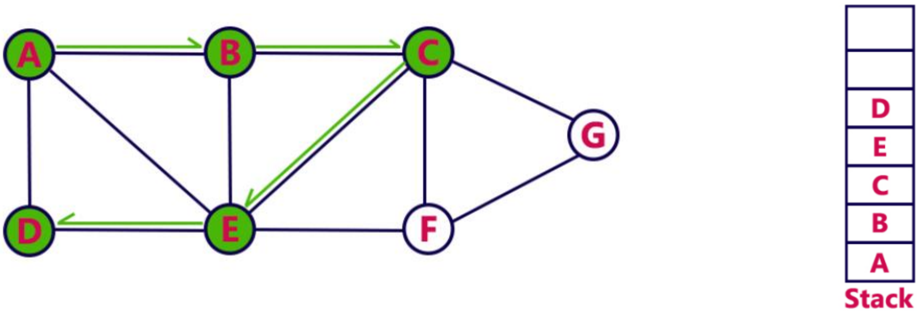


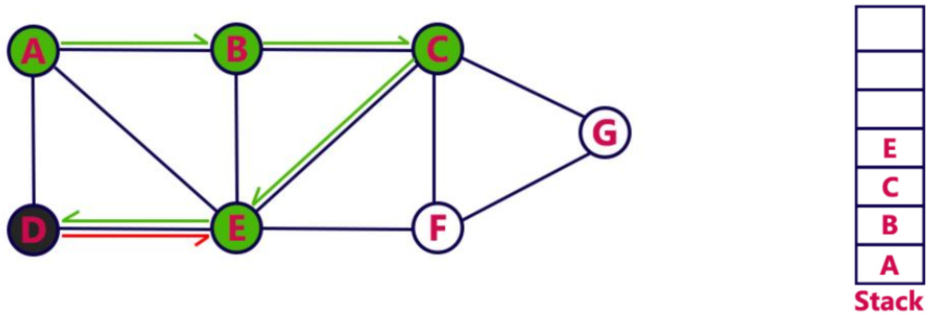


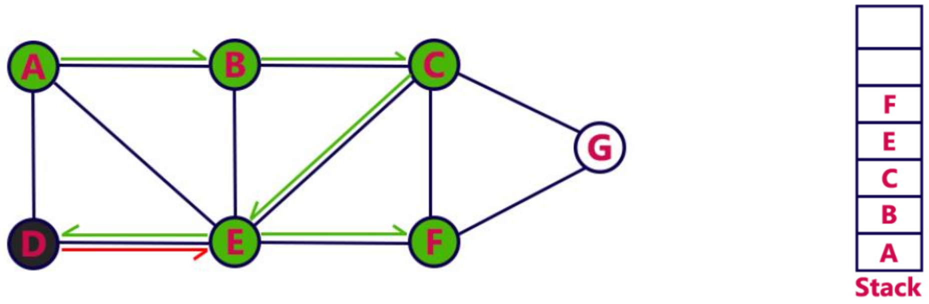

A
B
C
G
D
E
F
B
A
Stack

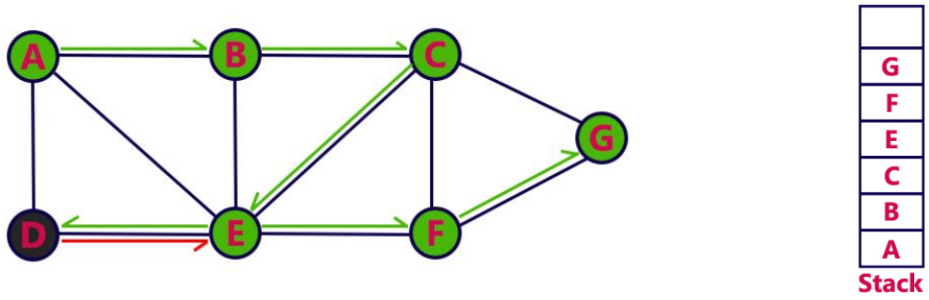

A
B
C
G
D
E
F
C
B
A
Stack

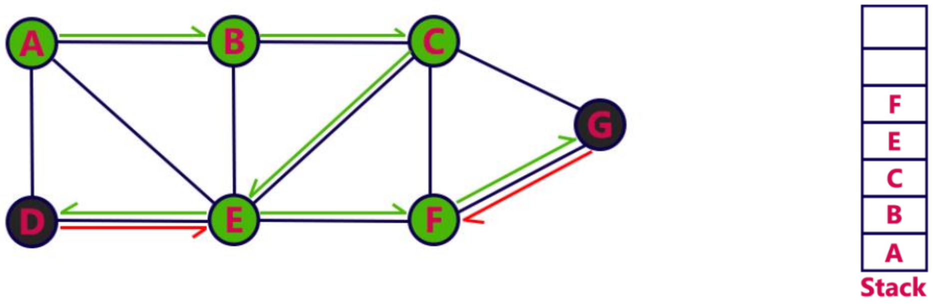

A
B
C
G
D
E
F
E
C
B
A
Stack

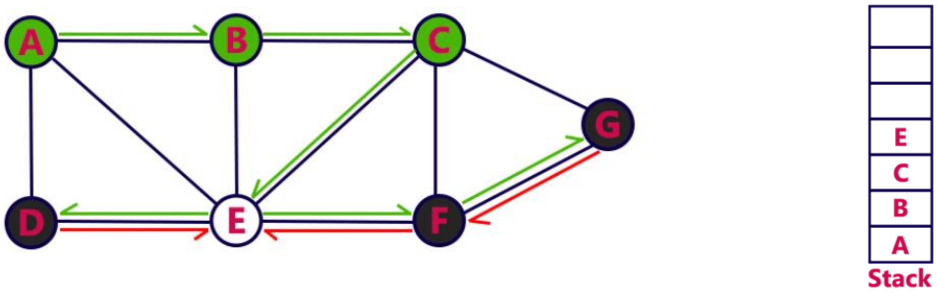

A
B
C
G
D
E
F
D
E
C
B
A
Stack

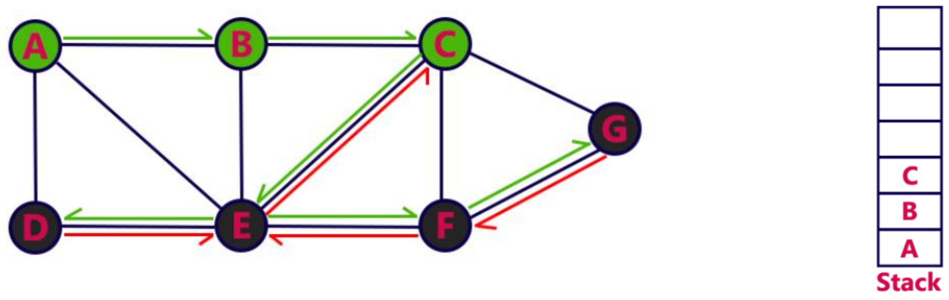

A
B
C
G
D
E
F
E
C
B
A
Stack

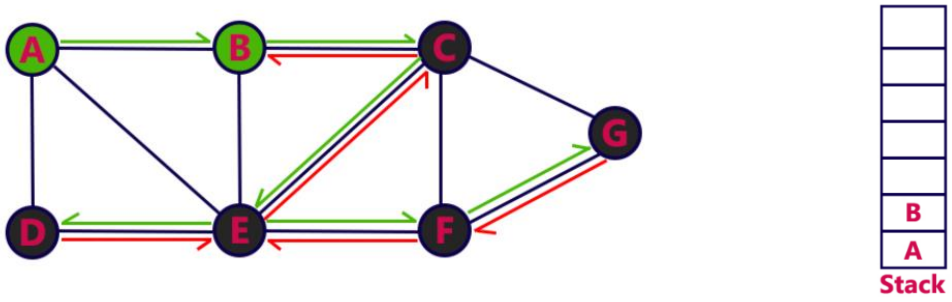

A
B
C
G
D
E
F
F
E
C
B
A
Stack

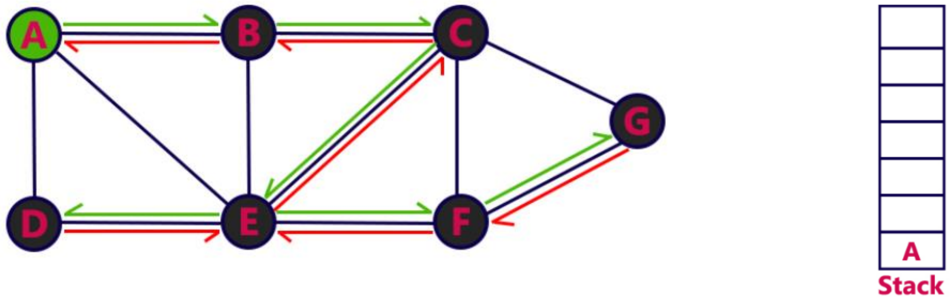

A
B
C
G
D
E
F
G
F
E
C
B
A
Stack

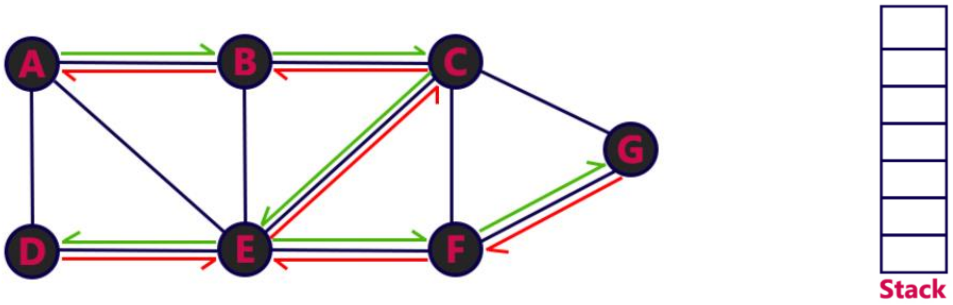

A
B
C
G
D
E
F
F
E
C
B
A
Stack

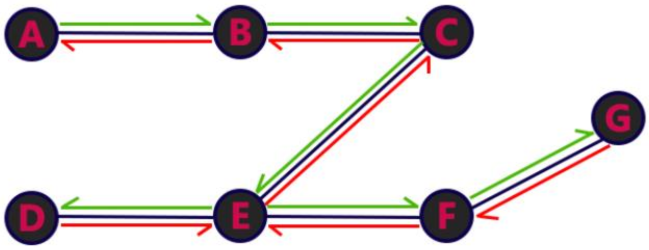

A
B
C
G
D
E
F
E
C
B
A
Stack


A
B
C
G
D
E
F
C
B
A
Stack


A
B
C
G
D
E
F
B
A
Stack


A
B
C
G
D
E
F
A
Stack

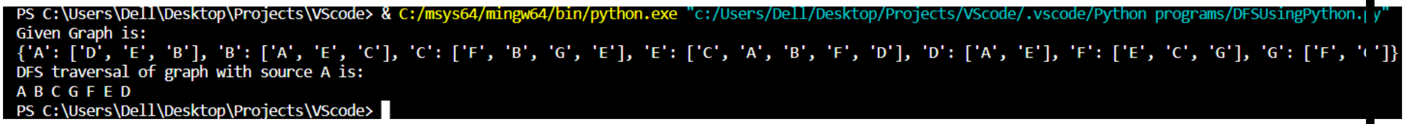

A
B
C
G
D
E
F
Stack


A
B
C
G
D
E
F

Code:

```
graph = {'A': ['D', 'E', 'B'], 'B': ['A', 'E', 'C'], 'C': ['F', 'B', 'G', 'E'], 'E': ['C', 'A', 'B', 'F', 'D'], 'D': ['A', 'E'],  'F': ['E', 'C', 'G'],   'G': ['F', 'C']}
print("Given Graph is:")
print(graph)
def dfs_traversal(input_graph, source):
    stack = list()
    visited_list = list()
    stack.append(source)
    visited_list.append(source)
    while stack:
        vertex = stack.pop()
        print(vertex, end=" ")
        for u in input_graph[vertex]:
            if u not in visited_list:
                stack.append(u)
                visited_list.append(u)
print("DFS traversal of graph with source A is:")
dfs_traversal(graph, "A")
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/python.exe "c:/Users/Dell/Desktop/Projects/VScode/.vscode/Python programs/DFSUsingPython.py"
Given Graph is:
{'A': ['D', 'E', 'B'], 'B': ['A', 'E', 'C'], 'C': ['F', 'B', 'G', 'E'], 'E': ['C', 'A', 'B', 'F', 'D'], 'D': ['A', 'E'], 'F': ['E', 'C', 'G'], 'G': ['F', 'C']}
DFS traversal of graph with source A is:
A B C G F E D
PS C:\Users\Dell\Desktop\Projects\VScode>
```

Result:

Depth first search has been implemented in python successfully.

Practical 4

Aim: To implement A* algorithm for a 8 puzzle problem in python.

Theory:

8 Puzzle Problem:

A 3 by 3 board with 8 tiles (each tile has a number from 1 to 8) and a single empty space is provided. The goal is to use the vacant space to arrange the numbers on the tiles such that they match the final arrangement. Four neighbouring (left, right, above, and below) tiles can be moved into the available area.

A* Algorithm:

A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. Noted for its performance and accuracy, it enjoys widespread use.
The key feature of the A* algorithm is that it keeps a track of each visited node which helps in ignoring the nodes that are already visited, saving a huge amount of time. It also has a list that holds all the nodes that are left to be explored and it chooses the most optimal node from this list, thus saving time not exploring unnecessary or less optimal nodes.

So we use two lists namely 'open list' and 'closed list' the open list contains all the nodes that are being generated and are not existing in the closed list and each node explored after it's neighboring nodes are discovered is put in the closed list and the neighbors are put in the open list this is how the nodes expand. Each node has a pointer to its parent so that at any given point it can retrace the path to the parent. Initially, the open list holds the start(Initial) node. The next node chosen from the open list is based on its f score, the node with the least f score is picked up and explored.

f-score = h-score + g-score

A* uses a combination of heuristic value (h-score: how far the goal node is) as well as the g-score (i.e. the number of nodes traversed from the start node to current node).

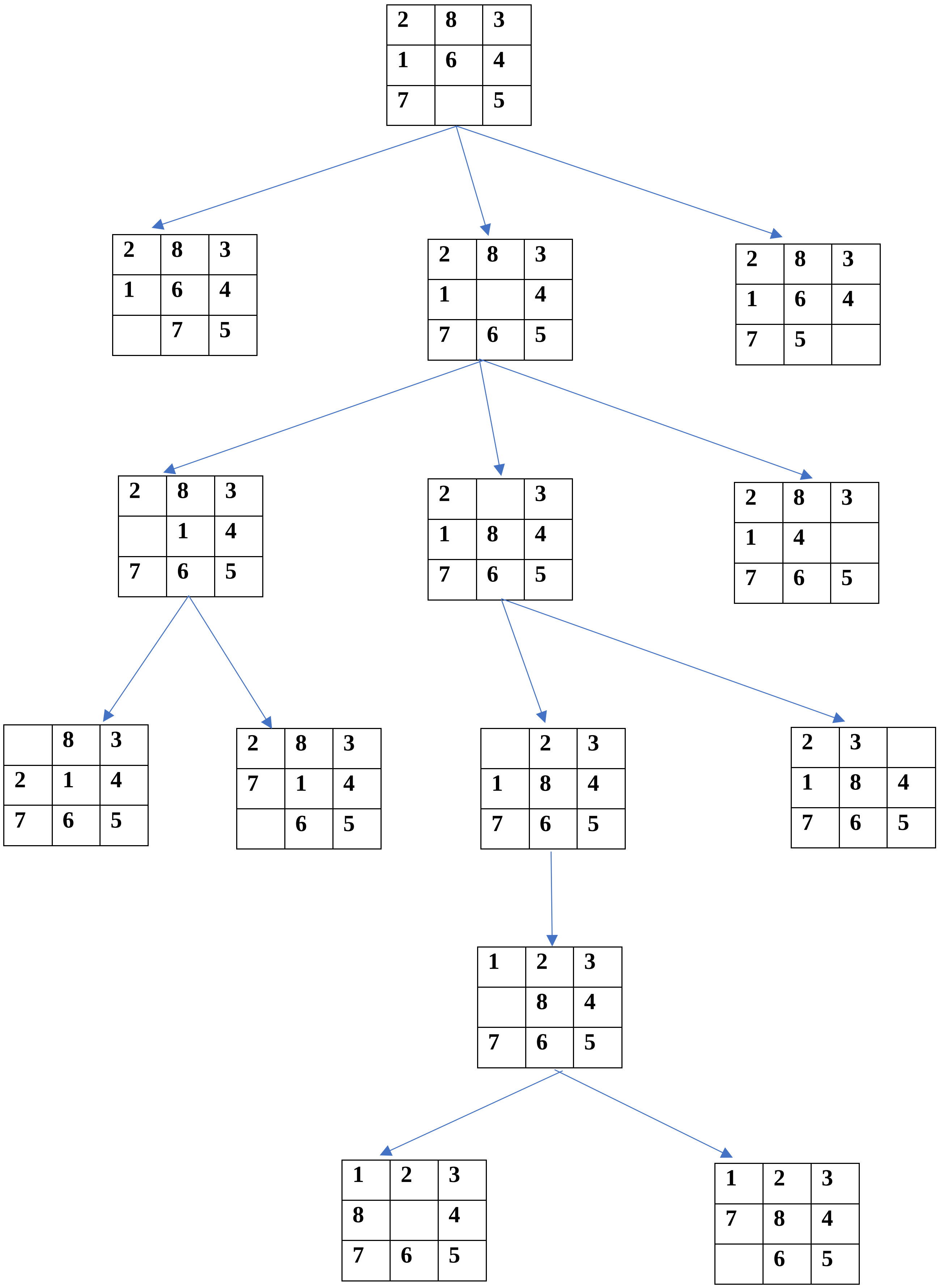
In our 8-Puzzle problem, we can define the h-score as the number of misplaced tiles by comparing the current state and the goal state or summation of the Manhattan distance between misplaced nodes.
g-score will remain as the number of nodes traversed from a start node to get to the current node.

2	8	3
1	6	4
7		5

Initial State

1	2	3
8		4
7	6	5

Final State



2	8	3
1	6	4
7		5

2	8	3
1	6	4
	7	5

2	8	3
1		4
7	6	5

2	8	3
1	6	4
7	5	

2	8	3
	1	4
7	6	5

2		3
1	8	4
7	6	5

2	8	3
1	4	
7	6	5

	8	3
2	1	4
7	6	5

2	8	3
7	1	4
	6	5

	2	3
1	8	4
7	6	5

2	3	
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

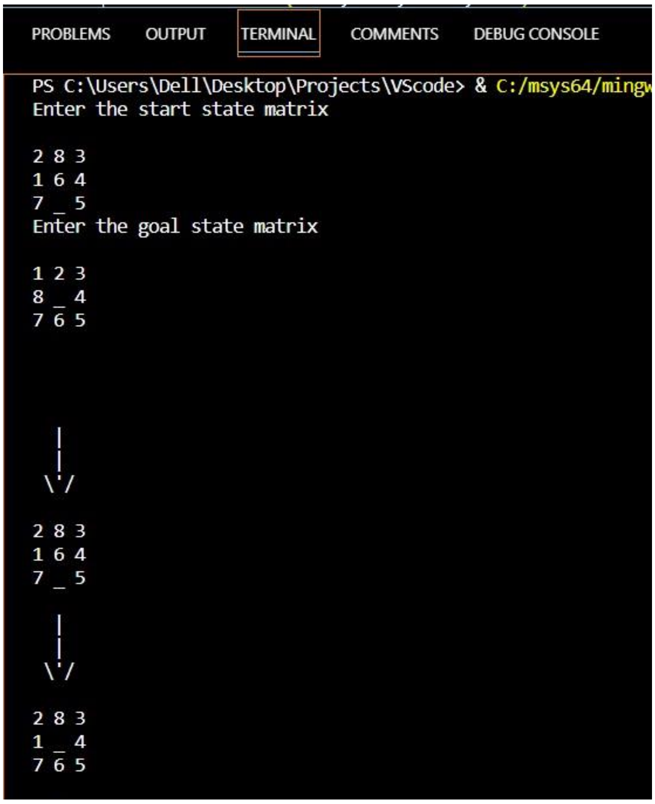
1	2	3
8		4
7	6	5

1	2	3
7	8	4
	6	5

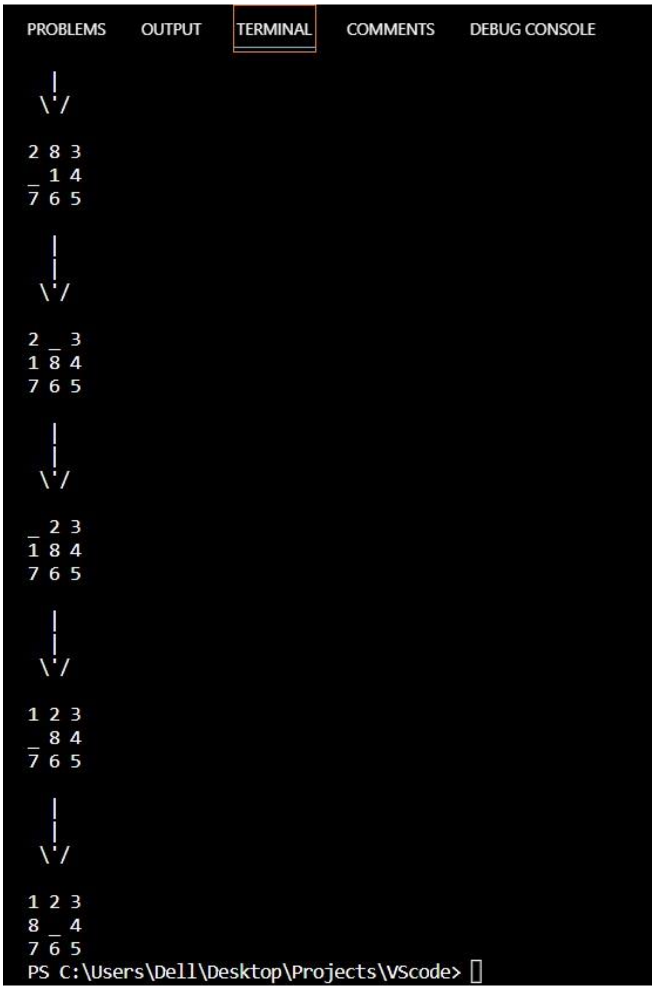
Code:

```
class Node:
    def __init__(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None

    def copy(self,root):
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp

    def find(self,puz,x):
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j

class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []
    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        return self.h(start.data,goal)+start.level
    def h(self,start,goal):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp

    def process(self):
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            self.open.sort(key = lambda x:x.fval,reverse=False)
puz = Puzzle(3)
puz.process()
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw
Enter the start state matrix

2 8 3
1 6 4
7 _ 5
Enter the goal state matrix

1 2 3
8 _ 4
7 6 5




  |
  |
 \'/

2 8 3
1 6 4
7 _ 5

  |
  |
 \'/

2 8 3
1 _ 4
7 6 5
```



```
  |
 \'/

2 8 3
_ 1 4
7 6 5

  |
  |
 \'/

2 _ 3
1 8 4
7 6 5

  |
  |
 \'/

_ 2 3
1 8 4
7 6 5

  |
  |
 \'/

1 2 3
_ 8 4
7 6 5

  |
  |
 \'/

1 2 3
8 _ 4
7 6 5
PS C:\Users\Dell\Desktop\Projects\VScode> []
```

Result:

A* algorithm has been successfully implemented for a 8 puzzle problem in python.

Practical 5

Aim: To implement single player 8 puzzle game using breadth first search.

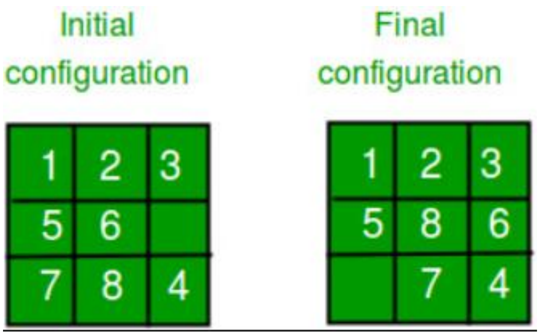
Theory:

An instance of the n-puzzle game consists of a board holding $n^{2}-1$ distinct movable tiles, plus an empty space. The tiles are numbers from the set $1,..,n^{2}-1$. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order $0,1,..,n^{2}-1$.

The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions {'Up', 'Down', 'Left', 'Right'}, one move at a time.

Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

For example:

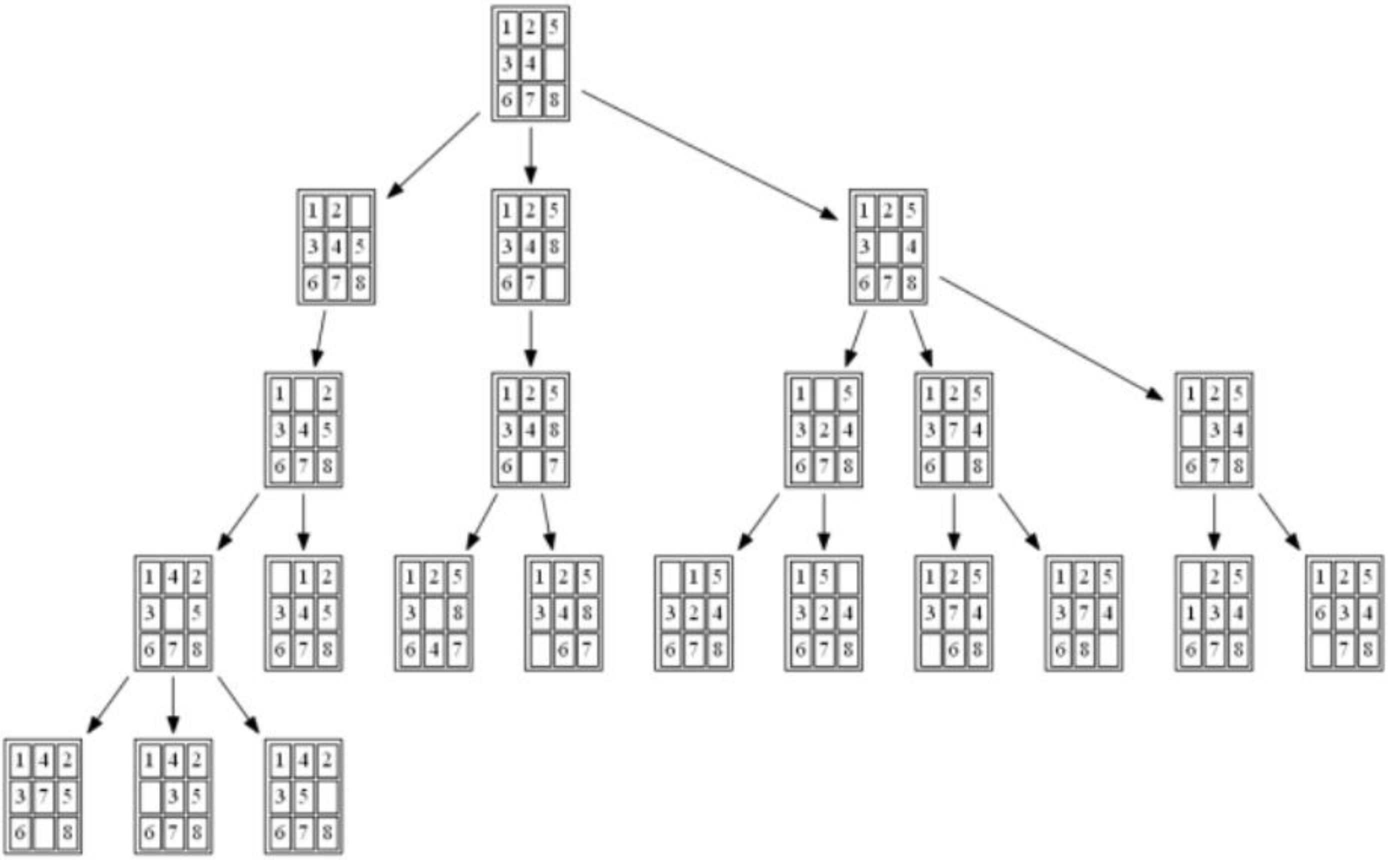


The searches begin by visiting the root node of the search tree, given by the initial state. Among other book-keeping details, three major things happen in sequence in order to visit a node:

1. First, we remove a node from the frontier set.
2. Second, we check the state against the goal state to determine if a solution has been found.
3. Finally, if the result of the check is negative, we then expand the node. To expand a given node, we generate successor nodes adjacent to the current node, and add them to the frontier set. Note that if these successor nodes are already in the frontier, or have already been visited, then they should not be added to the frontier again.

Example: Breadth-First Search

Initial State: 1,2,5,3,4,0,6,7,8



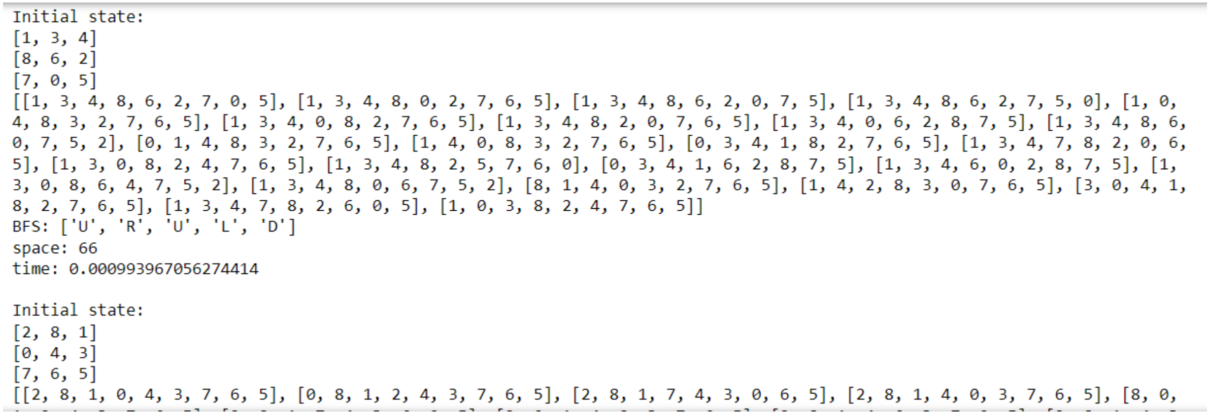
Code:

```
#Import the necessary libraries
from time import time
from queue import Queue
#Creating a class Puzzle
class Puzzle:
    #Setting the goal state of 8-puzzle
    goal_state=[1,2,3,8,0,4,7,6,5]
    num_of_instances=0
    #constructor to initialize the class members
    def __init__(self,state,parent,action):
        self.parent=parent
        self.state=state
        self.action=action

        #TODO: incrementing the number of instance by 1
        # num_of_instances = num_of_instances + 1
        Puzzle.num_of_instances+= 1
```

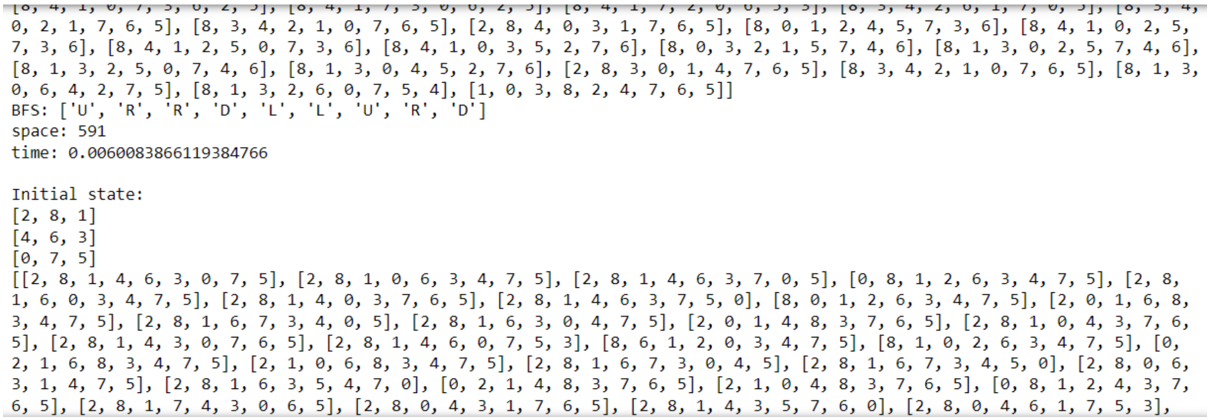
```
#function used to display a state of 8-puzzle
def __str__(self):
    return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6:9])

#method to compare the current state with the goal state
def goal_test(self):
    #TODO: include a condition to compare the current state with the goal state
    if self.state == self.goal_state:
        return True
    return False


#static method to find the legal action based on the current board position
@staticmethod
def find_legal_actions(i,j):
    legal_action = ['U', 'D', 'L', 'R']
    if i == 0:
        # if row is 0 in board then up is disable
        legal_action.remove('U')
    elif i == 2:
        #TODO: down is disable
        legal_action.remove('D')
    if j == 0:
        #TODO: Left is disable
        legal_action.remove('L')
    elif j == 2:
        #TODO: Right is disable
        legal_action.remove('R')
    return legal_action


#method to generate the child of the current state of the board
def generate_child(self):
    #TODO: create an empty list
    children=[]
    x = self.state.index(0)
    i = int(x / 3)
    j = int(x % 3)
    #TODO: call the method to find the legal actions based on i and j values
    legal_actions=self.find_legal_actions(i,j)


    #TODO:Iterate over all legal actions
    for action in legal_actions:
        new_state = self.state.copy()
```

```
#if the legal action is UP
            if action == 'U':
                #Swapping between current index of 0 with its up element on the board
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                #TODO: Swapping between current index of 0 with its down element on the board
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
              #TODO: Swapping between the current index of 0 with its left element on the board
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
             #TODO: Swapping between the current index of 0 with its right element on the board
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            children.append(Puzzle(new_state,self,action))
        #TODO: return the children
        return children
    #method to find the solution
    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
        solution = solution[:-1]
        solution.reverse()
        return solution
#method for breadth first search
#TODO: pass the initial_state as parameter to the breadth_first_search method
def breadth_first_search(initial_state):
    start_node = Puzzle(initial_state, None, None)
    print("Initial state:")
    print(start_node)
    if start_node.goal_test():
        return start_node.find_solution()
    q = Queue()
    #TODO: put start_node into the Queue
    q.put(start_node)

    #TODO: create an empty list of explored nodes
    explored=[]

    #TODO: Iterate the queue until empty. Use the empty() method of Queue
    while not(q.empty()):
        #TODO: get the current node of a queue. Use the get() method of Queue
        node=q.get()
        #TODO: Append the state of node in the explored list as node.state
        explored.append(node.state)

        #TODO: call the generate_child method to generate the child nodes of current node
        children= node.generate_child()
        #TODO: Iterate over each child node in children
        for child in children:
            if child.state not in explored:
                if child.goal_test():
                    print(explored)
                    return child.find_solution()
                q.put(child)
    return
#Start executing the 8-puzzle with setting up the initial state
#Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

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

       [2, 8, 1,
        4, 6, 3,
        0, 7, 5]]
#Iterate over number of initial_state
for i in range(0,3):
    #TODO: Initialize the num_of_instances to zero
    Puzzle.num_of_instances=0
    #Set t0 to current time
    t0=time()
    bfs=breadth_first_search(state[i])
    #Get the time t1 after executing the breadth_first_search method
    t1=time()-t0
    print('BFS:', bfs)
    print('space:',Puzzle.num_of_instances)
    print('time:',t1)
    print()
print('------------------------------------------')
```

Output:



```
Initial state:
[1, 3, 4]
[8, 6, 2]
[7, 0, 5]
[[1, 3, 4, 8, 6, 2, 7, 0, 5], [1, 3, 4, 8, 0, 2, 7, 6, 5], [1, 3, 4, 8, 6, 2, 0, 7, 5], [1, 3, 4, 8, 6, 2, 7, 5, 0], [1, 0,
4, 8, 3, 2, 7, 6, 5], [1, 3, 4, 0, 8, 2, 7, 6, 5], [1, 3, 4, 8, 2, 0, 7, 6, 5], [1, 3, 4, 0, 6, 2, 8, 7, 5], [1, 3, 4, 8, 6,
0, 7, 5, 2], [0, 1, 4, 8, 3, 2, 7, 6, 5], [1, 4, 0, 8, 3, 2, 7, 6, 5], [0, 3, 4, 1, 8, 2, 7, 6, 5], [1, 3, 4, 7, 8, 2, 0, 6,
5], [1, 3, 0, 8, 2, 4, 7, 6, 5], [1, 3, 4, 8, 2, 5, 7, 6, 0], [0, 3, 4, 1, 6, 2, 8, 7, 5], [1, 3, 4, 6, 0, 2, 8, 7, 5], [1,
3, 0, 8, 6, 4, 7, 5, 2], [1, 3, 4, 8, 0, 6, 7, 5, 2], [8, 1, 4, 0, 3, 2, 7, 6, 5], [1, 4, 2, 8, 3, 0, 7, 6, 5], [3, 0, 4, 1,
8, 2, 7, 6, 5], [1, 3, 4, 7, 8, 2, 6, 0, 5], [1, 0, 3, 8, 2, 4, 7, 6, 5]]
BFS: ['U', 'R', 'U', 'L', 'D']
space: 66
time: 0.000993967056274414

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



```
[8, 4, 1, 0, 7, 3, 6, 2, 5], [8, 4, 1, 7, 3, 0, 6, 2, 5], [8, 4, 1, 7, 2, 0, 6, 3, 5], [8, 3, 4, 2, 6, 1, 7, 0, 5], [8, 3, 4,
0, 2, 1, 7, 6, 5], [8, 3, 4, 2, 1, 0, 7, 6, 5], [2, 8, 4, 0, 3, 1, 7, 6, 5], [8, 0, 1, 2, 4, 5, 7, 3, 6], [8, 4, 1, 0, 2, 5,
7, 3, 6], [8, 4, 1, 2, 5, 0, 7, 3, 6], [8, 4, 1, 0, 3, 5, 2, 7, 6], [8, 0, 3, 2, 1, 5, 7, 4, 6], [8, 1, 3, 0, 2, 5, 7, 4, 6],
[8, 1, 3, 2, 5, 0, 7, 4, 6], [8, 1, 3, 0, 4, 5, 2, 7, 6], [2, 8, 3, 0, 1, 4, 7, 6, 5], [8, 3, 4, 2, 1, 0, 7, 6, 5], [8, 1, 3,
0, 6, 4, 2, 7, 5], [8, 1, 3, 2, 6, 0, 7, 5, 4], [1, 0, 3, 8, 2, 4, 7, 6, 5]]
BFS: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 591
time: 0.0060083866119384766

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

Result:

Single player 8 puzzle game using breadth first search has been successfully implemented in python.

Practical 6-(A)

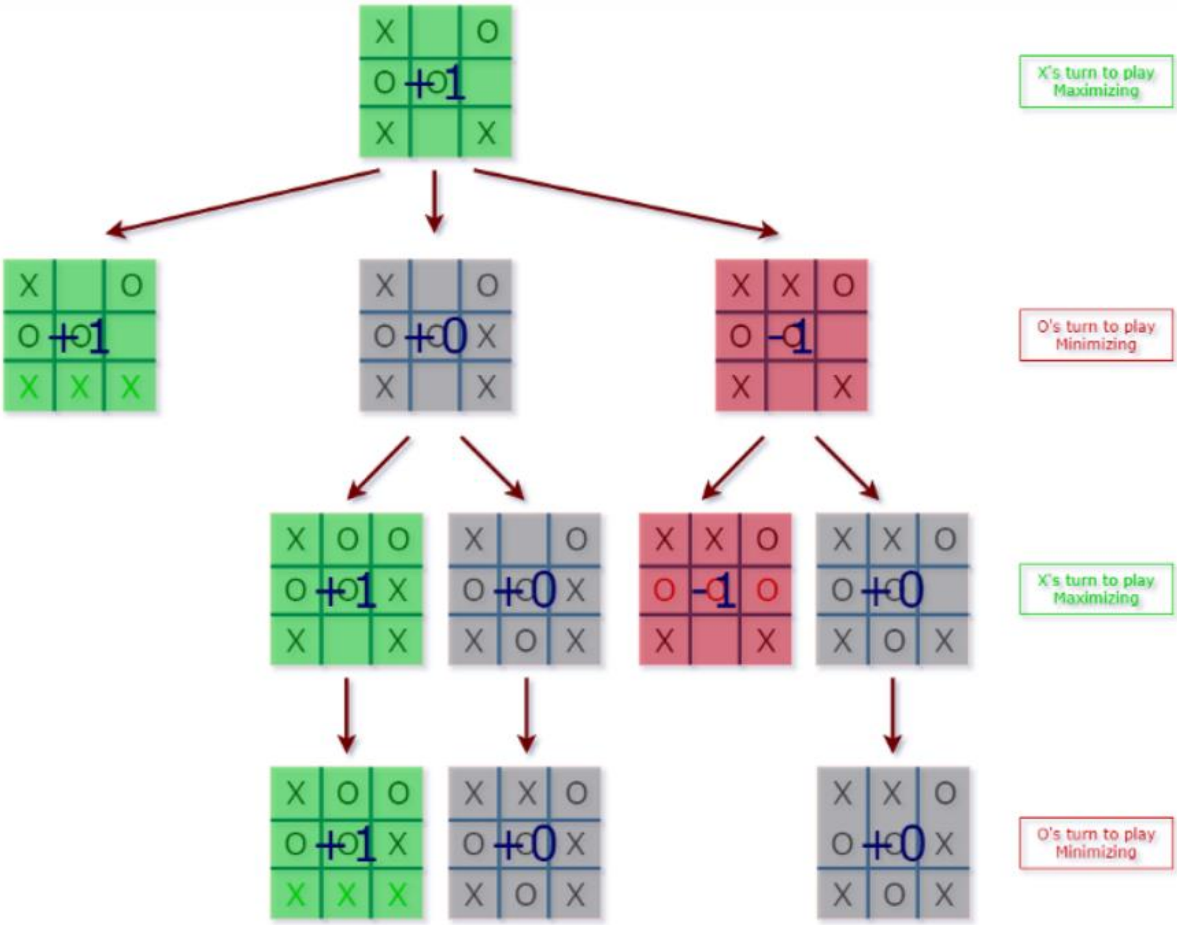
Aim: To implement Tic-Tac-Toe Game using MiniMax Algorithm.

Theory:

What is Minimax?

Minimax is a artifical intelligence applied in two player games, such as tic-tac-toe, checkers, chess and go. This games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of anyone not to win (0). Minimax is a type of adversarial search algorithm for generating and exploring game trees. It is mostly used to solve zero-sum games where one side's gain is equivalent to other side's loss, so adding all gains and subtracting all losses end up being zero.

Adversarial search differs from conventional searching algorithms by adding opponents into the mix. Minimax algorithm keeps playing the turns of both player and the opponent optimally to figure out the best possible move.


X's turn to play Maximizing
O's turn to play Minimizing
X's turn to play Maximizing
O's turn to play Minimizing
+1
+1
+0
-1
+1
+0
-1
+0
+1
+0
+0

How does it work?

The algorithm search, recursively, the best move that leads the Max player to win or not lose (draw). It considers the current state of the game and the available moves at that state, then for each valid move it plays (alternating min and max) until it finds a terminal state (win, draw or lose).

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

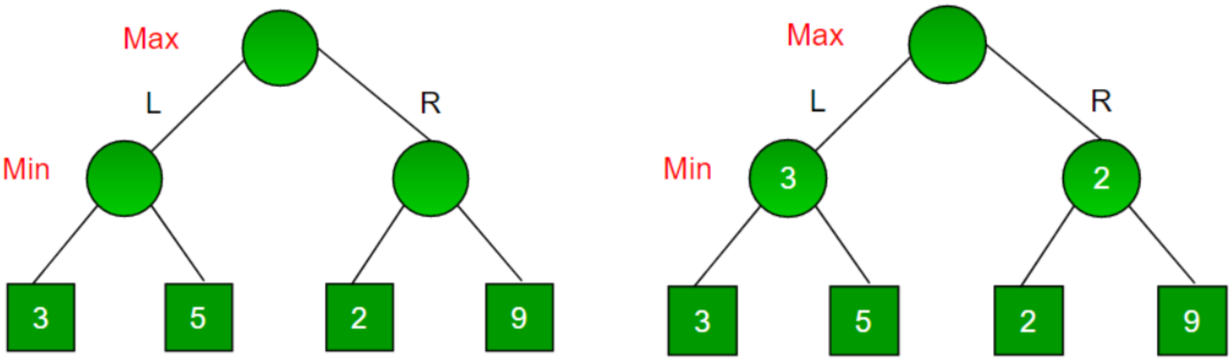
Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Example: Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level. Which move you would make as a maximizing player considering that your opponent also plays optimally?

Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values. Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below:

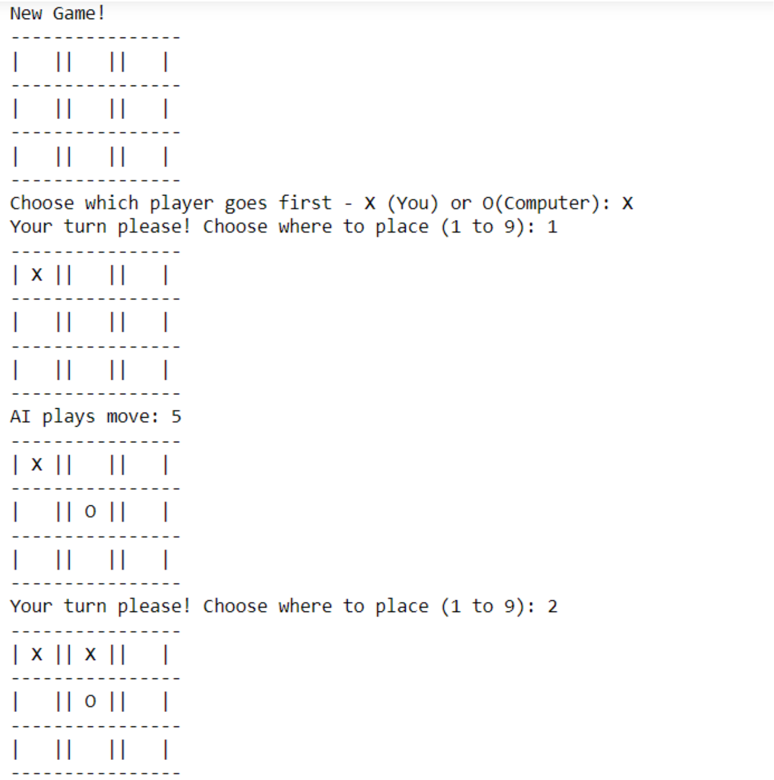


The above tree shows two possible scores when maximizer makes left and right moves.

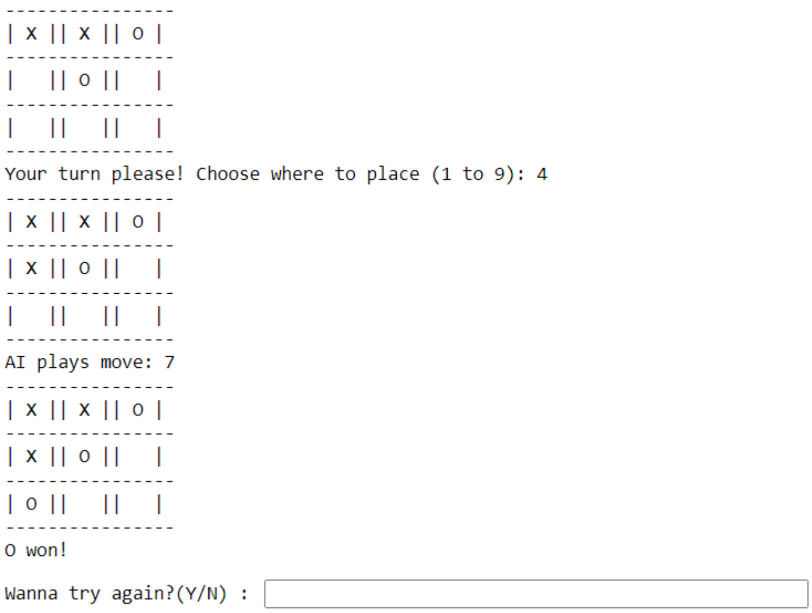
Code:

```
#Import the necessary libraries
import numpy as np
from math import inf as infinity

#Set the Empty Board
game_state = [[' ',' ',' '],
              [' ',' ',' '],
              [' ',' ',' ']]
#Create the Two Players as 'X'/'O'
players = ['X','O']

#Method for checking the correct move on Tic-Tac-Toe
def play_move(state, player, block_num):
    if state[int((block_num-1)/3)][(block_num-1)%3] == ' ':
        #TODO: Assign the player move on the current position of Tic-Tac-Toe if condition is True
        state[int((block_num-1)/3)][(block_num-1)%3] = player
    else:
        block_num = int(input("Block is not empty, ya blockhead! Choose again: "))
        play_move(state, player, block_num)
        #TODO: Recursively call the play_move

#Method to copy the current game state to new_state of Tic-Tac-Toe
def copy_game_state(state):
    new_state = [[' ',' ',' '],[' ',' ',' '],[' ',' ',' ']]
    for i in range(3):
        for j in range(3):
            #TODO: Copy the Tic-Tac-Toe state to new_state
            new_state[i][j] = state[i][j]
    #TODO: Return the new_state
    return new_state

#Method to check the current state of the Tic-Tac-Toe
def check_current_state(game_state):
    #TODO: Set the draw_flag to 0
    draw_flag = 0
    for i in range(3):
        for j in range(3):
            if game_state[i][j] == ' ':
                draw_flag = 1

    if draw_flag == 0:
        return None, "Draw"

    # Check horizontals in first row
    if (game_state[0][0] == game_state[0][1] and game_state[0][1] == game_state[0][2] and game_state[0][0] != ' '):
        return game_state[0][0], "Done"
    #TODO: Check horizontals in second row
    if (game_state[1][0] == game_state[1][1] and game_state[1][1] == game_state[1][2] and
game_state[1][0] != ' '):
        return game_state[1][0], "Done"
    #TODO: Check horizontals in third row
    if (game_state[2][0] == game_state[2][1] and game_state[2][1] == game_state[2][2] and
game_state[2][0] != ' '):
        return game_state[2][0], "Done"

    # Check verticals in first column
    if (game_state[0][0] == game_state[1][0] and game_state[1][0] == game_state[2][0] and
game_state[0][0] != ' '):
        return game_state[0][0], "Done"
    # Check verticals in second column
    if (game_state[0][1] == game_state[1][1] and game_state[1][1] == game_state[2][1] and
game_state[0][1] != ' '):
        return game_state[0][1], "Done"
    # Check verticals in third column
    if (game_state[0][2] == game_state[1][2] and game_state[1][2] == game_state[2][2] and
game_state[0][2] != ' '):
        return game_state[0][2], "Done"

    # Check left diagonal
    if (game_state[0][0] == game_state[1][1] and game_state[1][1] == game_state[2][2] and
game_state[0][0] != ' '):
        return game_state[1][1], "Done"
    # Check right diagonal
    if (game_state[2][0] == game_state[1][1] and game_state[1][1] == game_state[0][2] and
game_state[2][0] != ' '):
        return game_state[1][1], "Done"

    return None, "Not Done"

#Method to print the Tic-Tac-Toe Board
def print_board(game_state):
    print('----------------')
    print('| ' + str(game_state[0][0]) + ' || ' + str(game_state[0][1]) + ' || ' + str(game_state[0][2])
+ ' |')
    print('----------------')
    print('| ' + str(game_state[1][0]) + ' || ' + str(game_state[1][1]) + ' || ' + str(game_state[1][2])
+ ' |')
    print('----------------')
    print('| ' + str(game_state[2][0]) + ' || ' + str(game_state[2][1]) + ' || ' + str(game_state[2][2])
+ ' |')
    print('----------------')

#Method for implement the Minimax Algorithm
def getBestMove(state, player):
    #TODO: call the check_current_state method using state parameter
    winner_loser , done = check_current_state(state)
    #TODO:Check condition for winner, if winner_loser is 'O' then Computer won
    #else if winner_loser is 'X' then You won else game is draw
    if done == "Done" and winner_loser == 'O': # If AI won
        return (1,0)
    elif done == "Done" and winner_loser == 'X': # If Human won
        return (-1,0)
    elif done == "Draw":    # Draw condition
        return (0,0)

    #TODO: set moves to empty list
    moves = []
    #TODO: set empty_cells to empty list
    empty_cells = []

    #Append the block_num to the empty_cells list
    for i in range(3):
        for j in range(3):
            if state[i][j] == ' ':
                empty_cells.append(i*3 + (j+1))

    #TODO:Iterate over all the empty_cells
    for empty_cell in empty_cells:
        #TODO: create the empty dictionary
        move = {}

        #TODO: Assign the empty_cell to move['index']
        move['index'] = empty_cell

        #Call the copy_game_state method
        new_state = copy_game_state(state)

        #TODO: Call the play_move method with new_state,player,empty_cell
        play_move(new_state, player, empty_cell)

        #if player is computer
        if player == 'O':
            #TODO: Call getBestMove method with new_state and human player ('X') to make
more depth tree for human
            result,_ = getBestMove(new_state, 'X')
            move['score'] = result
        else:
            #TODO: Call getBestMove method with new_state and computer player('O') to make
more depth tree for computer
            result,_ = getBestMove(new_state, 'O')
            move['score'] = result

        moves.append(move)

    # Find best move
    best_move = None
    #Check if player is computer('O')
    if player == "O":
        #TODO: Set best as -infinity for computer
        best = -infinity
        for move in moves:
            #TODO: Check if move['score'] is greater than best
            if move['score'] > best:
                best = move['score']
                best_move = move['index']

    else:
        #TODO: Set best as infinity for human
        best = infinity
        for move in moves:
            #TODO: Check if move['score'] is less than best
            if move['score'] < best:
                best = move['score']
                best_move = move['index']
    return (best, best_move)

# Now PLaying the Tic-Tac-Toe Game
play_again = 'Y'
while play_again == 'Y' or play_again == 'y':
    #Set the empty board for Tic-Tac-Toe
    game_state = [[' ',' ',' '],
              [' ',' ',' '],
              [' ',' ',' ']]
    #Set current_state as "Not Done"
    current_state = "Not Done"
    print("\nNew Game!")


    #print the game_state
    print_board(game_state)

    #Select the player_choice to start the game
    player_choice = input("Choose which player goes first - X (You) or O(Computer): ")

    #Set winner as None
    winner = None

    #if player_choice is ('X' or 'x') for humans else for computer
    if player_choice == 'X' or player_choice == 'x':
        #TODO: Set current_player_idx is 0
        current_player_idx = 0
    else:
        #TODO: Set current_player_idx is 1
        current_player_idx = 1

    while current_state == "Not Done":
        #For Human Turn
        if current_player_idx == 0:
            block_choice = int(input("Your turn please! Choose where to place (1 to 9): "))
            #TODO: Call the play_move with parameters as game_state
,players[current_player_idx], block_choice
            play_move(game_state ,players[current_player_idx], block_choice)
        else:   # Computer turn
            _,block_choice = getBestMove(game_state, players[current_player_idx])
            #TODO: Call the play_move with parameters as game_state
,players[current_player_idx], block_choice
            play_move(game_state ,players[current_player_idx], block_choice)
            print("AI plays move: " + str(block_choice))
        print_board(game_state)
        #TODO: Call the check_current_state function for game_state
        winner, current_state = check_current_state(game_state)
        if winner is not None:
            print(str(winner) + " won!")
        else:
            current_player_idx = (current_player_idx + 1)%2

        if current_state == "Draw":
            print("Draw!")

    play_again = input('Wanna try again?(Y/N) : ')
    if play_again == 'N':
        print('Thank you for playing Tic-Tac-Toe Game!!!!!!!')
```

Output:



```
New Game!
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Choose which player goes first - X (You) or O(Computer): X
Your turn please! Choose where to place (1 to 9): 1
----------------
| X ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
AI plays move: 5
----------------
| X ||   ||   |
----------------
|   || O ||   |
----------------
|   ||   ||   |
----------------
Your turn please! Choose where to place (1 to 9): 2
----------------
| X || X ||   |
----------------
|   || O ||   |
----------------
|   ||   ||   |
----------------
```



```
----------------
| x || x || o |
----------------
|   || o ||   |
----------------
|   ||   ||   |
----------------
Your turn please! Choose where to place (1 to 9): 4
----------------
| x || x || o |
----------------
| x || o ||   |
----------------
|   ||   ||   |
----------------
AI plays move: 7
----------------
| x || x || o |
----------------
| x || o ||   |
----------------
| o ||   ||   |
----------------
O won!
Wanna try again?(Y/N) :
```

Result:

Tic-Tac-Toe Game using MiniMax Algorithm has been successfully implemented in python.

Practical 6-(B)

Aim: To implement Tic-Tac-Toe Game using Alpha-Beta Pruning Algorithm.

Theory:

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:

  1. Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.

  2. Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

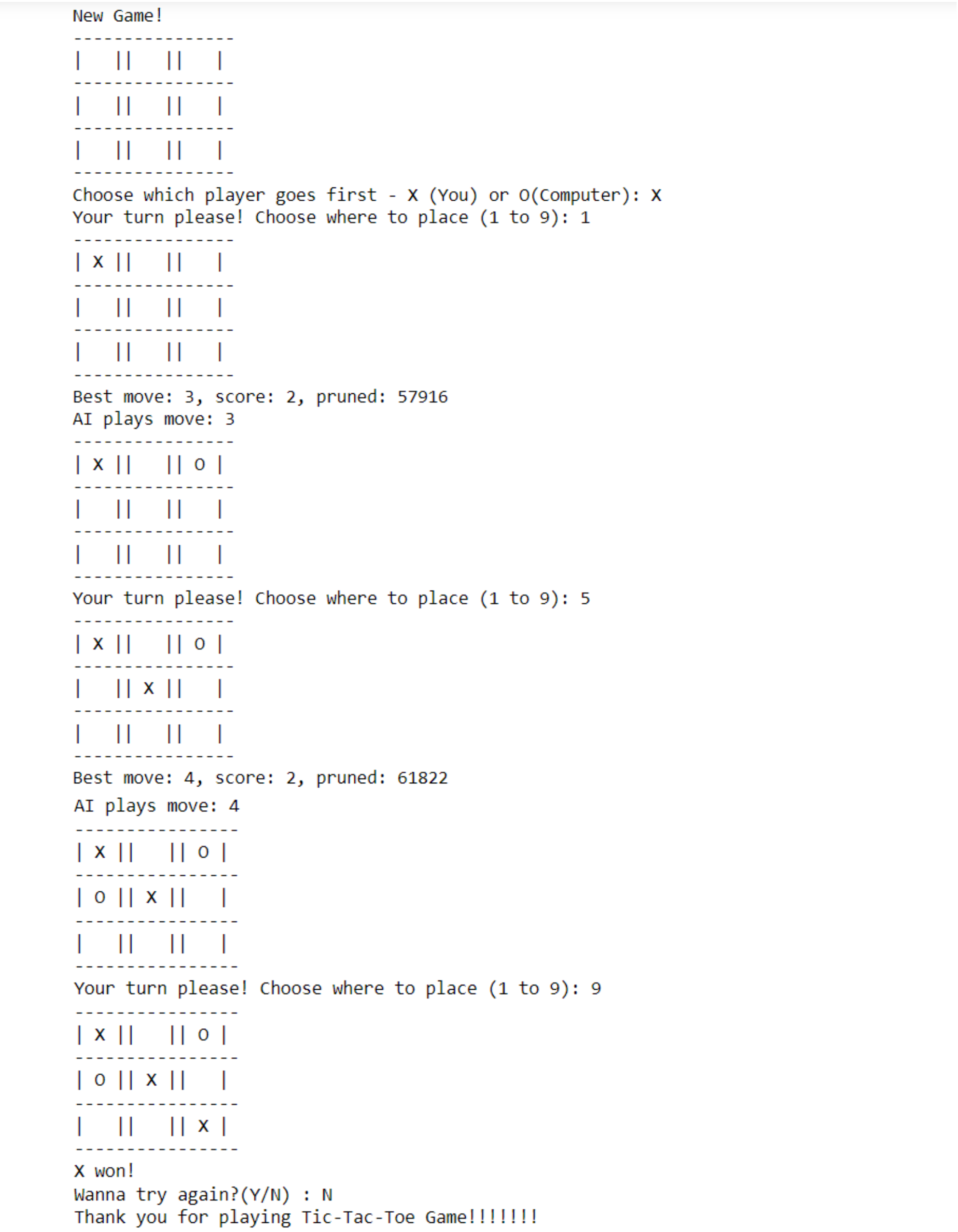
Condition for Alpha-beta pruning

- Alpha: At any point along the Maximizer path, Alpha is the best option or the highest value we’ve discovered. The initial value for alpha is – ∞.
- Beta: At any point along the Minimizer path, Beta is the best option or the lowest value we’ve discovered.. The initial value for alpha is + ∞.
- The condition for Alpha-beta Pruning is that α >= β.
- The alpha and beta values of each node must be kept track of. Alpha can only be updated when it’s MAX’s time, and beta can only be updated when it’s MIN’s turn.
- MAX will update only alpha values and the MIN player will update only beta values.
- The node values will be passed to upper nodes instead of alpha and beta values during going into the tree’s reverse.
- Alpha and Beta values only are passed to child nodes.

Code:

```
#Import the necessary libraries
import numpy as np
from math import inf as infinity

#Set the Empty Board
game_state = [[' ',' ',' '],
              [' ',' ',' '],
              [' ',' ',' ']]
#Create the Two Players as 'X'/'O'
players = ['X','O']
pruned=0

#Method for checking the correct move on Tic-Tac-Toe
def play_move(state, player, block_num):
    if state[int((block_num-1)/3)][(block_num-1)%3] == ' ':
        #TODO: Assign the player move on the current position of Tic-Tac-Toe if condition is True
        state[int((block_num-1)/3)][(block_num-1)%3] = player
    else:
        block_num = int(input("Block is not empty, ya blockhead! Choose again: "))
        play_move(state, player, block_num)
        #TODO: Recursively call the play_move

#Method to copy the current game state to new_state of Tic-Tac-Toe
def copy_game_state(state):
    new_state = [[' ',' ',' '],[' ',' ',' '],[' ',' ',' ']]
    for i in range(3):
        for j in range(3):
            #TODO: Copy the Tic-Tac-Toe state to new_state
            new_state[i][j] = state[i][j]
    #TODO: Return the new_state
    return new_state

#Method to check the current state of the Tic-Tac-Toe
def check_current_state(game_state):
    #TODO: Set the draw_flag to 0
    draw_flag = 0
    for i in range(3):
        for j in range(3):
            if game_state[i][j] == ' ':
                draw_flag = 1

    if draw_flag == 0:
        return None, "Draw"
    # Check horizontals in first row
    if (game_state[0][0] == game_state[0][1] and game_state[0][1] == game_state[0][2] and
game_state[0][0] != ' '):
        return game_state[0][0], "Done"
    #TODO: Check horizontals in second row
    if (game_state[1][0] == game_state[1][1] and game_state[1][1] == game_state[1][2] and
game_state[1][0] != ' '):
        return game_state[1][0], "Done"
    #TODO: Check horizontals in third row
    if (game_state[2][0] == game_state[2][1] and game_state[2][1] == game_state[2][2] and
game_state[2][0] != ' '):
        return game_state[2][0], "Done"

    # Check verticals in first column
    if (game_state[0][0] == game_state[1][0] and game_state[1][0] == game_state[2][0] and
game_state[0][0] != ' '):
        return game_state[0][0], "Done"
    # Check verticals in second column
    if (game_state[0][1] == game_state[1][1] and game_state[1][1] == game_state[2][1] and
game_state[0][1] != ' '):
        return game_state[0][1], "Done"
    # Check verticals in third column
    if (game_state[0][2] == game_state[1][2] and game_state[1][2] == game_state[2][2] and
game_state[0][2] != ' '):
        return game_state[0][2], "Done"

    # Check left diagonal
    if (game_state[0][0] == game_state[1][1] and game_state[1][1] == game_state[2][2] and
game_state[0][0] != ' '):
        return game_state[1][1], "Done"
    # Check right diagonal
    if (game_state[2][0] == game_state[1][1] and game_state[1][1] == game_state[0][2] and
game_state[2][0] != ' '):
        return game_state[1][1], "Done"

    return None, "Not Done"

#Method to print the Tic-Tac-Toe Board
def print_board(game_state):
    print('----------------')
    print('| ' + str(game_state[0][0]) + ' || ' + str(game_state[0][1]) + ' || ' + str(game_state[0][2])
+ ' |')
    print('----------------')
    print('| ' + str(game_state[1][0]) + ' || ' + str(game_state[1][1]) + ' || ' + str(game_state[1][2])
+ ' |')
    print('----------------')
    print('| ' + str(game_state[2][0]) + ' || ' + str(game_state[2][1]) + ' || ' + str(game_state[2][2])
+ ' |')
    print('----------------')

#Method for implement the alpha beta pruning function
def alphabeta(state,depth,alpha,beta,player):
    global pruned
    #TODO: call the check_current_state method using state parameter
    winner_loser , done = check_current_state(state)
    #TODO:Check condition for winner, if winner_loser is 'O' then Computer won
    #else if winner_loser is 'X' then You won else game is draw
    if done == "Done" and winner_loser == 'O': # If AI won
        return (1,0)
    elif done == "Done" and winner_loser == 'X': # If Human won
        return (-1,0)
    elif done == "Draw":    # Draw condition
        return (0,0)
    #TODO: set moves to empty list
    moves = []
    #TODO: set empty_cells to empty list
    empty_cells = []
    #Append the block_num to the empty_cells list
    for i in range(3):
        for j in range(3):
            if state[i][j] == ' ':
                empty_cells.append(i*3 + (j+1))

    #TODO:Iterate over all the empty_cells
    for empty_cell in empty_cells:
        #TODO: create the empty dictionary
        move = {}

        #TODO: Assign the empty_cell to move['index']
        move['index'] = empty_cell

        #Call the copy_game_state method
        new_state = copy_game_state(state)

        #TODO: Call the play_move method with new_state,player,empty_cell
        play_move(new_state, player, empty_cell)

        #if player is computer
        if player == 'O':
            #TODO: Call getBestMove method with new_state and human player ('X') to make
more depth tree for human
            result = alphabeta(new_state, depth-1, alpha, beta,False)[1]
            move['score'] = result
        else:
            #TODO: Call getBestMove method with new_state and computer player('O') to make
more depth tree for computer
            result = alphabeta(new_state, depth-1, alpha, beta,True)[1]
            move['score'] = result

        moves.append(move)

    # Find best move
    best_move = None
    #Check if player is computer('O')
    if player == "O":
        #TODO: Set best as -infinity for computer
        best = - infinity
        for move in moves:
            #TODO: Check if move['score'] is greater than best
            if move['score'] > best:
                best = move['score']
                best_move = move['index']
            alpha=max(alpha, best)
            if alpha >= beta:
                pruned+=(3-depth)**2
                # Increment pruned counter
                break
    else:
        #TODO: Set best as infinity for human
        best = infinity
        for move in moves:
        #TODO: Check if move['score'] is less than best
            if move['score'] < best:
                best = move['score']
                best_move = move['index']
            beta=min(alpha, best)
            if alpha >= beta:
                pruned+=(3-depth)**2
                break

    return (best, best_move,pruned)

# Now PLaying the Tic-Tac-Toe Game
play_again = 'Y'
while play_again == 'Y' or play_again == 'y':
    depth = 9
    #Set the empty board for Tic-Tac-Toe
    game_state = [[' ',' ',' '],
                ['  ','  ','  '],
                ['  ','  ','  ']]
    pruned=0
    #Set current_state as "Not Done"
    current_state = "Not Done"
    print("\nNew Game!")
    #print the game_state
    print_board(game_state)
    #Select the player_choice to start the game
    player_choice = input("Choose which player goes first - X (You) or O(Computer): ")
    #Set winner as None
    winner = None
    #if player_choice is ('X' or 'x') for humans else for computer
    if player_choice == 'X' or player_choice == 'x':
        #TODO: Set current_player_idx is 0
        current_player_idx = 0
    else:
        #TODO: Set current_player_idx is 1
        current_player_idx = 1
    while current_state == "Not Done":
        #For Human Turn
        if current_player_idx == 0:
            block_choice = int(input("Your turn please! Choose where to place (1 to 9): "))
            #TODO: Call the play_move with parameters as game_state
,players[current_player_idx], block_choice
            play_move(game_state ,players[current_player_idx], block_choice)
        else:
            best_move, best_score, pruned = alphabeta(game_state, depth, float('-inf'),
float('inf'),True)
            play_move(game_state ,players[current_player_idx], best_move)
            print(f"Best move: {best_move}, score: {best_score}, pruned: {pruned}")
            print("AI plays move: " + str(best_move))
            # Computer turn
#              _,block_choice = getBestMove(game_state,float('-
inf'),float('inf'),players[current_player_idx],pruned_states)
            #TODO: Call the play_move with parameters as game_state
,players[current_player_idx], block_choice

        print_board(game_state)
        #TODO: Call the check_current_state function for game_state
        winner, current_state = check_current_state(game_state)
        if winner is not None:
            print(str(winner) + " won!")
        else:
            current_player_idx = (current_player_idx + 1)%2
        if current_state == "Draw":
            print("Draw!")

    play_again = input('Wanna try again?(Y/N) : ')
    if play_again == 'N':
        print('Thank you for playing Tic-Tac-Toe Game!!!!!!!')
```

Output:



```
New Game!
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Choose which player goes first - X (You) or O(Computer): X
Your turn please! Choose where to place (1 to 9): 1
----------------
| X ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Best move: 3, score: 2, pruned: 57916
AI plays move: 3
----------------
| X ||   || O |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Your turn please! Choose where to place (1 to 9): 5
----------------
| X ||   || O |
----------------
|   || X ||   |
----------------
|   ||   ||   |
----------------
Best move: 4, score: 2, pruned: 61822
AI plays move: 4
----------------
| X ||   || O |
----------------
| O || X ||   |
----------------
|   ||   ||   |
----------------
Your turn please! Choose where to place (1 to 9): 9
----------------
| X ||   || O |
----------------
| O || X ||   |
----------------
|   ||   || X |
----------------
X won!
Wanna try again?(Y/N) : N
Thank you for playing Tic-Tac-Toe Game!!!!!!!
```

Result:

Tic-Tac-Toe Game using Alpha-Beta Pruning Algorithm has been successfully implemented in python.

Practical 7

Aim: To solve Crypt-Arithmetic problem of APPLE+LEMON=BANANA

Theory:

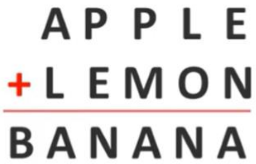
Cryptarithmetic Problem

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

We can perform all the arithmetic operations on a given cryptarithmetic problem.

The rules or constraints on a cryptarithmetic problem are as follows:

- There should be a unique digit to be replaced with a unique alphabet.
- The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
- Digits should be from 0-9 only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., lefthand side (L.H.S), or righthand side (R.H.S)



```
  APPLE
+LEMON
 BANANA
```

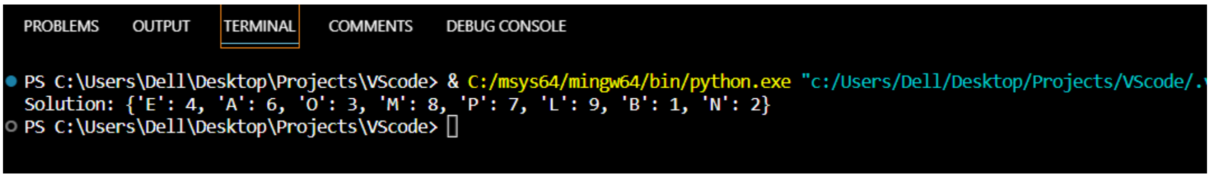
Solution to this is cryptarithmetic problem is –

L – 9, E – 4, M –8, N – 2, A – 6, B – 1, P – 7, O – 3

Code:

```
import itertools
def solve_cryptarithmetic(puzzle):
    # Extract the words and the result from the puzzle
    words = puzzle.split()
    word1, word2, result = words[0], words[2], words[4]
    # Generate all possible combinations of digits
    digits = set(range(10))
    for c in word1 + word2 + result:
        if c in digits:
            digits.remove(c)
    digit_combinations = itertools.permutations(digits, len(set(word1 + word2 + result)))

    # Check each combination of digits for a valid solution
    for combination in digit_combinations:
        mapping = dict(zip(set(word1 + word2 + result), combination))
        if (mapping[word1[0]] == 0) or (mapping[word2[0]] == 0) or (mapping[result[0]] == 0):
            continue
        if int(".join(str(mapping[c]) for c in word1)) + int(".join(str(mapping[c]) for c in word2))
== int(".join(str(mapping[c]) for c in result)):
            return mapping
    # If no solution is found, return None
    return None

# Example usage:
puzzle = "APPLE + LEMON = BANANA"
solution = solve_cryptarithmetic(puzzle)
if solution:
    print("Solution:", solution)
else:
    print("No solution ")
```

Output:



```
● PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/python.exe "c:/Users/Dell/Desktop/Projects/VScode/.
  Solution: {'E': 4, 'A': 6, 'O': 3, 'M': 8, 'P': 7, 'L': 9, 'B': 1, 'N': 2}
○ PS C:\Users\Dell\Desktop\Projects\VScode> ▯
```

Result:

Given Crypt Arithmetic problem has been successfully solved and implemented in python.

Practical 8

Aim: To implement Graph Colouring problem in python.

Theory:

Graph coloring is the procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color. The objective is to minimize the number of colors while coloring a graph. The smallest number of colors required to color a graph G is called its chromatic number of that graph. Graph coloring problem is a NP Complete problem.

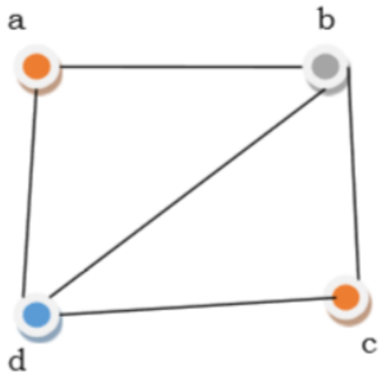
Method to Color a Graph

The steps required to color a graph G with n number of vertices are as follows −

Step 1 − Arrange the vertices of the graph in some order.

Step 2 − Choose the first vertex and color it with the first color.

Step 3 − Choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices adjacent to it. If all the adjacent vertices are colored with this color, assign a new color to it. Repeat this step until all the vertices are colored.



In the above figure, at first vertex a is colored red. As the adjacent vertices of vertex a are again adjacent, vertex b and vertex d are colored with different color, green and blue respectively. Then vertex c is colored as red as no adjacent vertex of c is colored red. Hence, we could color the graph by 3 colors. Hence, the chromatic number of the graph is 3.

Applications of Graph Coloring

Some applications of graph coloring include −

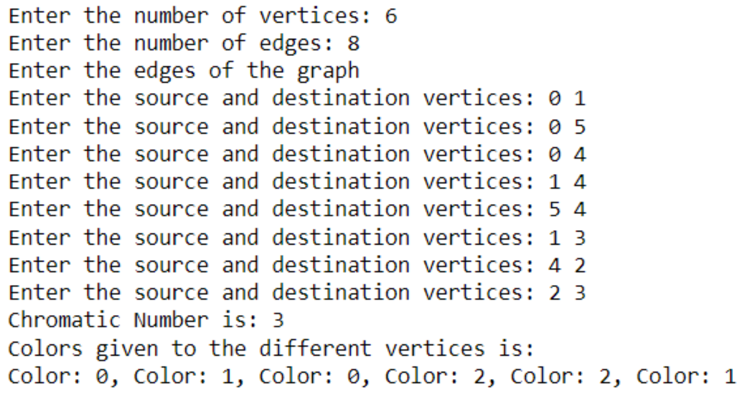
- Register Allocation
- Map Coloring
- Bipartite Graph Checking
- Mobile Radio Frequency Assignment
- Making time table, etc.

Code:

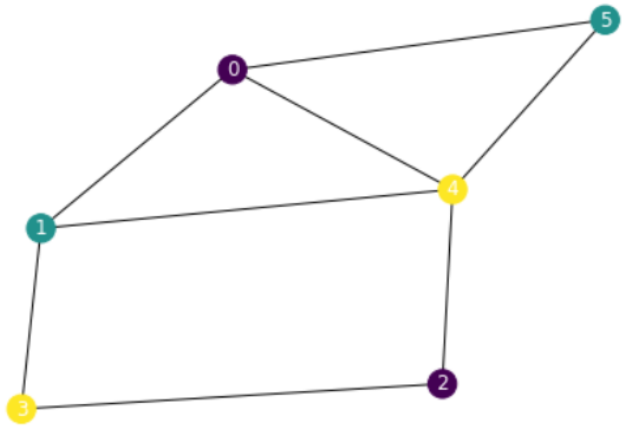
```
from typing import List
import networkx as nx
import matplotlib.pyplot as plt

def solve(vertices: int, edges: int, g: List[List[int]]) -> None:
    result = [-1] * vertices
    color_available = [True] * vertices
    chromatic_number = 0
    color = 0

    result[0] = 0
    chromatic_number += 1

    for i in range(1, vertices):
        for x in g[i]:
            if result[x] != -1:
                color_available[result[x]] = False

        for color in range(vertices):
            if color_available[color]:
                break

        result[i] = color
        chromatic_number = max(chromatic_number, color+1)

        for x in g[i]:
            if result[x] != -1:
                color_available[result[x]] = True

    print("Chromatic Number is:", chromatic_number)
    print("Colors given to the different vertices is:")
    for i in range(vertices):
        print("Color:", result[i], end=")
        if i == vertices-1:
            break
        print(", ", end=")

    # Draw the graph
    G = nx.Graph()
    for i in range(vertices):
        G.add_node(i)
    for u in range(vertices):
        for v in g[u]:
            G.add_edge(u, v)
    pos = nx.spring_layout(G)
    colors = [result[i] for i in range(vertices)]
    nx.draw(G, pos, with_labels=True, node_color=colors, font_color='white')
    plt.show()
```

```
vertices = int(input("Enter the number of vertices: "))
edges = int(input("Enter the number of edges: "))
g = [[] for _ in range(vertices)]
print("Enter the edges of the graph")
for i in range(edges):
    u, v = map(int, input("Enter the source and destination vertices: ").split())
    g[u].append(v)
    g[v].append(u)
solve(vertices, edges, g)
```

Output:



```
Enter the number of vertices: 6
Enter the number of edges: 8
Enter the edges of the graph
Enter the source and destination vertices: 0 1
Enter the source and destination vertices: 0 5
Enter the source and destination vertices: 0 4
Enter the source and destination vertices: 1 4
Enter the source and destination vertices: 5 4
Enter the source and destination vertices: 1 3
Enter the source and destination vertices: 4 2
Enter the source and destination vertices: 2 3
Chromatic Number is: 3
Colors given to the different vertices is:
Color: 0, Color: 1, Color: 0, Color: 2, Color: 2, Color: 1
```



Result:

Graph Colouring problem has been successfully solved and implemented in python.

Practical 9

Aim: To implement program to tokenize word and sentences with the help of NLTK package in python.

Theory:

What is Tokenization?

Tokenization is the process by which a large quantity of text is divided into smaller parts called tokens. These tokens are very useful for finding patterns and are considered as a base step for stemming and lemmatization. Tokenization also helps to substitute sensitive data elements with non-sensitive data elements.

Natural language processing is used for building applications such as Text classification, intelligent chatbot, sentimental analysis, language translation, etc. It becomes vital to understand the pattern in the text to achieve the above-stated purpose.

For the time being, don’t worry about stemming and lemmatization but treat them as steps for textual data cleaning using NLP (Natural language processing). We will discuss stemming and lemmatization later in the tutorial. Tasks such as Text classification or spam filtering makes use of NLP along with deep learning libraries such as Keras and tensorflow.

Natural Language toolkit has very important module NLTK tokenize sentences which further comprises of sub-modules

1. word tokenize
2. sentence tokenize

Tokenization of words

We use the method word_tokenize() to split a sentence into words. The output of word tokenization can be converted to Data Frame for better text understanding in machine learning applications. It can also be provided as input for further text cleaning steps such as punctuation removal, numeric character removal or stemming. Machine learning models need numeric data to be trained and make a prediction. Word tokenization becomes a crucial part of the text (string) to numeric data conversion.

Tokenization of Sentences

Sub-module available for the above is sent_tokenize. An obvious question in your mind would be why sentence tokenization is needed when we have the option of word tokenization. Imagine you need to count average words per sentence, how you will calculate? For accomplishing such a task, you need both NLTK sentence tokenizer as well as NLTK word tokenizer to calculate the ratio. Such output serves as an important feature for machine training as the answer would be numeric.

Stop Words

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.
We would not want these words to take up space in our database, or taking up valuable

processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory.

Stemming

Stemming is a technique used to reduce an inflected word down to its word stem. For example, the words “programming,” “programmer,” and “programs” can all be reduced down to the common word stem “program.” In other words, “program” can be used as a synonym for the prior three inflection words.

Performing this text-processing technique is often useful for dealing with sparsity and/or standardizing vocabulary. Not only does it help with reducing redundancy, as most of the time the word stem and their inflected words have the same meaning, it also allows NLP models to learn links between inflected words and their word stem, which helps the model understand their usage in similar contexts.

Stemming algorithms function by taking a list of frequent prefixes and suffixes found in inflected words and chopping off the end or beginning of the word. This can occasionally result in word stems that are not real words; thus, we can affirm this approach certainly has its pros, but it’s not without its limitations.

Lemmatization

Lemmatization is another technique used to reduce inflected words to their root word. It describes the algorithmic process of identifying an inflected word’s “lemma” (dictionary form) based on its intended meaning.

As opposed to stemming, lemmatization relies on accurately determining the intended part of speech and the meaning of a word based on its context. This means it takes into consideration where the inflected word falls within a sentence, as well as within the larger context surrounding that sentence, such as neighbouring sentences or even an entire document.

In other words, to lemmatize a document typically means to “doing things correctly” since it involves using a vocabulary and performing morphological analysis of words to remove only the inflectional ends and return the base or dictionary form of a word, which is known as the “lemma.” For example, you can expect a lemmatization algorithm to map “runs,” “running,” and “ran” to the lemma, “run.”

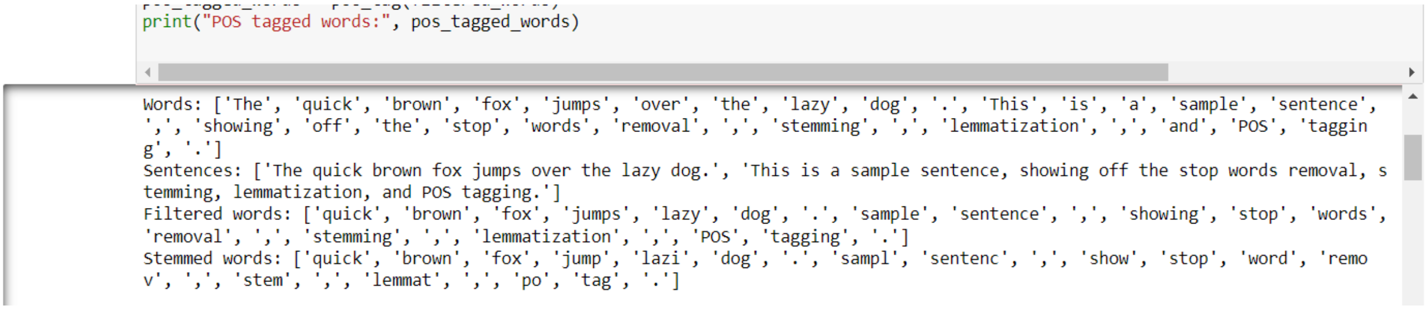
POS tagging

NLTK POS tag is the practice of marking up the words in text format for a specific segment of a speech context is known as POS Tagging (Parts of Speech Tagging). It is in charge of interpreting a language’s text and associating each word with a specific token. Grammar tagging is another term for it. A part-of-speech tagger, also known as a POS-tagger, analyses a string of words.The words will assign to part-of-speech tag to one. To begin, we must employ the concept of tokenization.The built-in voice tagger is the most useful feature of NLTK for Python. Nltk pos tagger is not flawless, but it is quite good. So we can buy some or alter the existing NLTK code if we want something better.The goal of part of speech tagging is to help us grasp sentence structure and start using our computer to follow the sentence meaning of words, part of speech, and the string it forms.

Code:

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
text = "The quick brown fox jumps over the lazy dog."
words = word_tokenize(text)
print("Words:", words)
sentences = sent_tokenize(text)
print("Sentences:", sentences)

stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.casefold() not in stop_words]
print("Filtered words:", filtered_words)

porter = PorterStemmer()
stemmed_words = [porter.stem(word) for word in filtered_words]
print("Stemmed words:", stemmed_words)

lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]
print("Lemmatized words:", lemmatized_words)

pos_tagged_words = pos_tag(filtered_words)
print("POS tagged words:", pos_tagged_words)
```

Output:



```
pos_tagged_words = pos_tag(filtered_words)
print("POS tagged words:", pos_tagged_words)
```

```
Words: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.', 'This', 'is', 'a', 'sample', 'sentence',
',', 'showing', 'off', 'the', 'stop', 'words', 'removal', ',', 'stemming', ',', 'lemmatization', ',', 'and', 'POS', 'taggin
g', '.']
Sentences: ['The quick brown fox jumps over the lazy dog.', 'This is a sample sentence, showing off the stop words removal, s
temming, lemmatization, and POS tagging.']
Filtered words: ['quick', 'brown', 'fox', 'jumps', 'lazy', 'dog', '.', 'sample', 'sentence', ',', 'showing', 'stop', 'words',
'removal', ',', 'stemming', ',', 'lemmatization', ',', 'POS', 'tagging', '.']
Stemmed words: ['quick', 'brown', 'fox', 'jump', 'lazi', 'dog', '.', 'sampl', 'sentenc', ',', 'show', 'stop', 'word', 'remo
v', ',', 'stem', ',', 'lemmat', ',', 'po', 'tag', '.']
```

Result:

Tokenization of word and Sentences with the help of NLTK package has been successfully solved and implemented in python.

Practical 10

Aim: Implement Brute force solution to the Knapsack problem in Python.

Theory:

The Knapsack problem is a classic optimization problem in computer science, where the objective is to fill a knapsack with items such that the total value of the items is maximized, subject to a constraint on the maximum weight that the knapsack can carry.

More formally, the problem can be stated as follows: Given a set of items, each with a weight and a value, and a knapsack with a capacity, determine the maximum value subset of the items that can be put into the knapsack without exceeding its capacity.

There are various algorithms that can be used to solve the Knapsack problem, including dynamic programming, greedy algorithms, and branch and bound. However, the Brute force solution involves considering all possible combinations of items and selecting the one that maximizes the value while staying within the weight capacity of the knapsack.

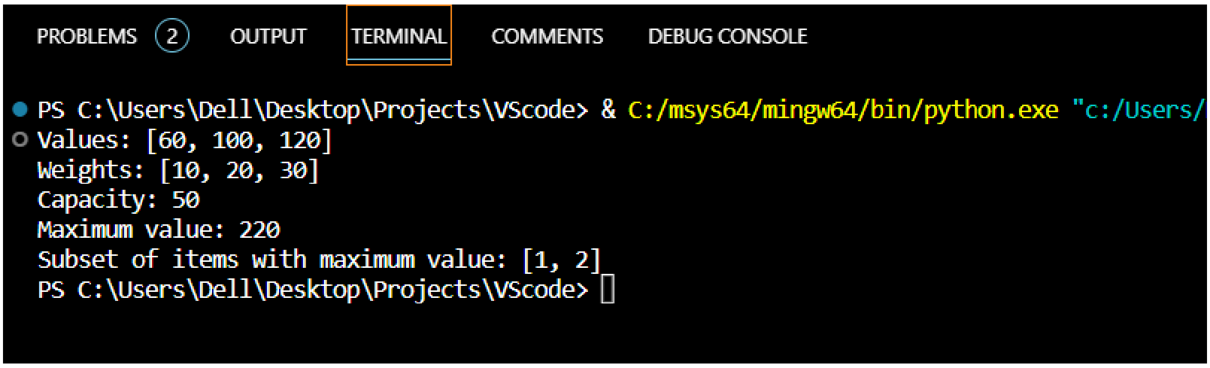
This approach involves generating all possible subsets of items and calculating the value of each subset. This can be done using binary numbers, where each digit represents an item in the set. For example, a binary number 101 represents the subset of items that includes the first and third items, but not the second.

While the Brute force solution is conceptually simple, it can be computationally expensive, especially for larger sets of items. Nonetheless, it serves as a useful baseline for evaluating other optimization algorithms for the Knapsack problem.

Code:

```
def knapsack_brute_force(values, weights, capacity):
    n = len(values)
    max_value = 0
    max_subset = []
    for i in range(2**n):
        subset_values = []
        subset_weights = []
        for j in range(n):
            if i & (1 << j):
                subset_values.append(values[j])
                subset_weights.append(weights[j])

        subset_weight = sum(subset_weights)
        subset_value = sum(subset_values)
        if subset_weight <= capacity and subset_value > max_value:
            max_value = subset_value
            max_subset = [j for j in range(n) if i & (1 << j)]
    return max_value, max_subset


values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
max_value, max_subset = knapsack_brute_force(values, weights, capacity)
print("Values:", values)
print("Weights:", weights)
print("Capacity:", capacity)
print("Maximum value:", max_value)
print("Subset of items with maximum value:", max_subset
```

Output:



```
PS C:\Users\Dell\Desktop\Projects\VScode> & C:/msys64/mingw64/bin/python.exe "c:/Users/
Values: [60, 100, 120]
Weights: [10, 20, 30]
Capacity: 50
Maximum value: 220
Subset of items with maximum value: [1, 2]
PS C:\Users\Dell\Desktop\Projects\VScode> 
```

Result:

Brute force solution to the Knapsack problem has been successfully solved and implemented in python.

Practical 11

Aim: Study of SCIKIT fuzzy.

Theory:

SCIKIT-fuzzy is a Python library for working with fuzzy logic, which is a branch of artificial intelligence that deals with reasoning under uncertainty. It provides tools for creating and manipulating fuzzy sets, fuzzy variables, fuzzy rules, and fuzzy inference systems.

The library is built on top of the popular scientific computing library, NumPy, and provides a high-level interface for working with fuzzy logic that is both intuitive and powerful. It is designed to be easy to use, flexible, and efficient, and is well-suited for both research and industrial applications.

SCIKIT-fuzzy provides a wide range of membership functions, defuzzification methods, and inference engines that can be used to model complex systems that involve imprecise or uncertain inputs. It also supports visualization tools for plotting fuzzy sets and membership functions, making it easier to understand and analyze fuzzy systems.

The library provides a wide range of membership functions, which can be used to define fuzzy sets, as well as defuzzification methods, which can be used to convert fuzzy sets into crisp outputs. In addition, it provides a range of inference engines, which can be used to make decisions based on fuzzy inputs, and it also supports visualization tools for plotting fuzzy sets and membership functions.

One of the key features of SCIKIT-fuzzy is its support for modular and extensible fuzzy logic systems. This means that users can easily create their own custom membership functions, inference engines, and defuzzification methods, and use them in their fuzzy logic systems. This makes the library highly flexible and adaptable to a wide range of applications.

SCIKIT-fuzzy is used in a wide range of applications, including control systems, data analysis, image processing, and decision-making. It is widely used in both academia and industry, and is a valuable tool for anyone working with fuzzy logic.

Overall, SCIKIT-fuzzy is a valuable tool for anyone working with fuzzy logic, whether in academia or industry, and provides a powerful framework for modeling and analyzing complex systems that involve uncertainty.

1. Fuzzy Sets
   Fuzzy sets are a fundamental concept in fuzzy logic. They represent a set of values that are not precise but have degrees of membership. SCIKIT-fuzzy provides a range of tools for working with fuzzy sets.
   For example, you can create a triangular fuzzy set with the following parameters: x = [0, 1, 2, 3, 4], and m = [0, 0, 1, 0, 0]. This defines a triangular fuzzy set with maximum membership at x=2.

2. Fuzzy Variables

   Fuzzy variables represent variables that have degrees of membership in fuzzy sets. They are used to represent imprecise or uncertain inputs or outputs of a system. SCIKIT-fuzzy provides a range of tools for working with fuzzy variables.

For example, you can create a fuzzy variable representing temperature with the following fuzzy sets: cool = [0, 0, 20], moderate = [10, 20, 30], and hot = [20, 40, 40]. This defines a fuzzy variable for temperature with three fuzzy sets: "cool", "moderate", and "hot".

3. Fuzzy Rules

   Fuzzy rules are used to map fuzzy inputs to fuzzy outputs. They are a way of formalizing expert knowledge or heuristics. SCIKIT-fuzzy provides a range of tools for working with fuzzy rules.

   For example, you can define fuzzy rules for a tipping system based on two inputs: service and food quality. The rules might look like this:

   - If the service is poor OR the food quality is poor, then the tip will be low.
   - If the service is average, then the tip will be medium.
   - If the service is good OR the food quality is good, then the tip will be high.

4. Fuzzy Inference

   Fuzzy inference is the process of using fuzzy rules to make decisions based on fuzzy inputs. SCIKIT-fuzzy provides a range of tools for performing fuzzy inference.

   For example, you can use the fuzzy rules from the previous example to determine the tip amount for a given service quality and food quality. If the service quality is 6 out of 10 and the food quality is 8 out of 10, then the fuzzy inference engine might produce the following output: the tip amount should be around 20%.

5. Membership Functions

   Membership functions are used to define fuzzy sets and fuzzy variables. They describe how each element of a universe of discourse belongs to a given fuzzy set. SCIKIT-fuzzy provides a range of membership functions, including triangular, trapezoidal, Gaussian, and more.

   For example, you can use a Gaussian membership function to represent the fuzzy set "medium" for a fuzzy variable representing speed. This would create a fuzzy set with a peak at a certain speed value and a spread that determines how quickly the membership decreases as you move away from the peak.

Overall, SCIKIT-fuzzy provides a powerful set of tools for working with fuzzy logic in Python. These tools can be used to model complex systems that involve imprecise or uncertain inputs and make decisions based on fuzzy rules.

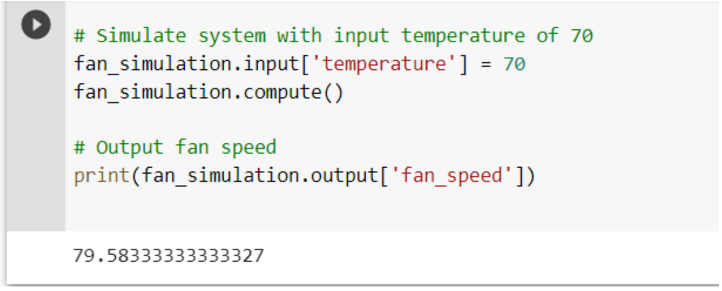
In the next example, we have defined an input variable 'temperature' and an output variable 'fan_speed', each with two membership functions: 'cold' and 'hot' for temperature, and 'low' and 'high' for fan speed. We have then defined two rules based on the temperature membership functions, one for when the temperature is 'cold' and one for when it's 'hot'.

We have created a control system using these rules, and then simulated the system with an input temperature of 70. The output fan speed is then printed to the console.

Code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
# Create input and output variables
temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

# Define membership functions
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature['hot'] = fuzz.trimf(temperature.universe, [50, 100, 100])
fan_speed['low'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Define rules
rule1 = ctrl.Rule(temperature['cold'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['hot'], fan_speed['high'])

# Create control system
fan_ctrl = ctrl.ControlSystem([rule1, rule2])

# Create simulation
fan_simulation = ctrl.ControlSystemSimulation(fan_ctrl)

# Simulate system with input temperature of 70
fan_simulation.input['temperature'] = 70
fan_simulation.compute()

# Output fan speed
print(fan_simulation.output['fan_speed'])
```

Output:



```
# Simulate system with input temperature of 70
fan_simulation.input['temperature'] = 70
fan_simulation.compute()

# Output fan speed
print(fan_simulation.output['fan_speed'])
```

```
79.58333333333327
```

Result:

SCIKIT fuzzy has been studied successfully.

Open Ended Experiment

Aim: To implement Spam SMS Detection using natural language toolkit.

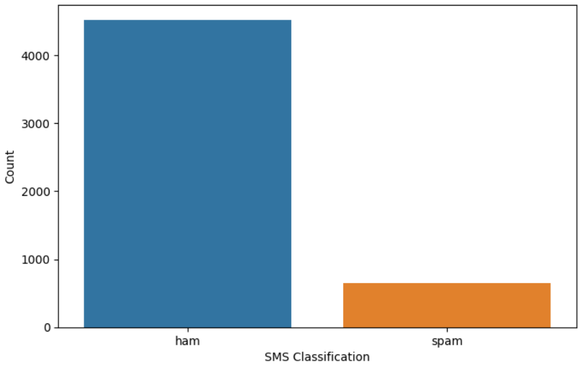
Theory:

Modules used:

1) Numpy: NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices.
2) Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.
3) Seaborn: Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.
4) Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.
5) Nltk: NLTK (Natural Language Toolkit) is the go-to API for NLP (Natural Language Processing) with Python. It is a really powerful tool to preprocess text data for further analysis like with ML models for instance. It helps convert text into numbers, which the model can then easily work with.
6) Sklearn: Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Steps to perform to make the model:

1) Import all the required modules that are to be used to perform the data analysis and apply functions on the dataset.
2) Import the dataset. In this experiment "Spam SMS Collection.txt" file has been used. For this program the dataset used is a labelled dataset.
3) The first step for data analysis is to clean the data. This include:
    a. Removing duplicate values.
    b. Deal with missing values.
    c. Fixing structural values.
    d. Filter unwanted outliers.
4) After cleaning the dataset we have perform nlp task on the dataset to get the desired input set for sms spam detection model.



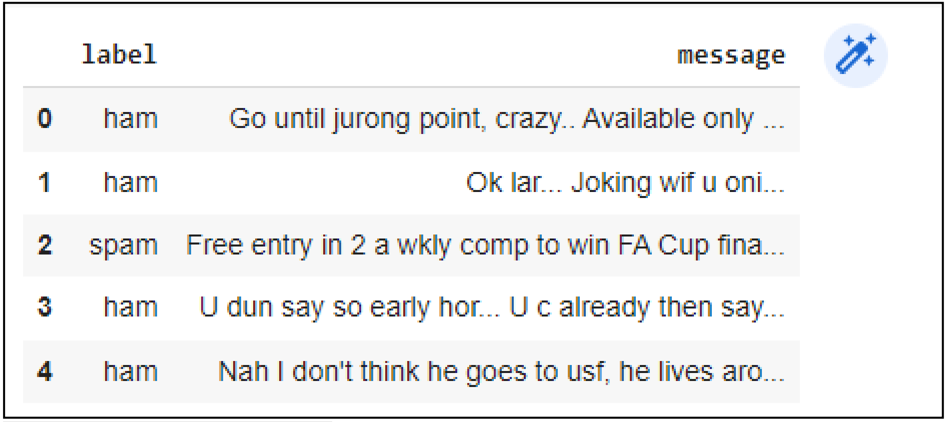
5) Steps for cleaning the message are:
    a. Cleaning special character from the message
    b. Converting the entire message into lower case
    c. Tokenizing the review by words
    d. #Removing the stop words
    e. Stemming the words
    f. Joining the stemmed words
    g. Building a corpus of messages
6) Creating the Bag of Words model
7) Extracting dependent variable from the dataset.
8) Training the model to detect spams.
9) Using Extracting dependent variable from the dataset. Checking the accuracy and precision received from the alpha Naive Bayes Classifier.
10) Fitting Naive Bayes to the Training set.
11) Predicting the Test set results and getting actual accuracy score from the model.

Code:

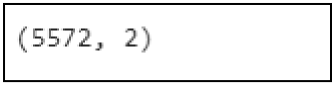
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
nltk.download('stopwords')
import re
import sklearn
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
import pickle

sms = pd.read_csv('Spam SMS Collection', sep='\t', names=['label','message'])
sms.drop_duplicates(inplace=True)
sms.reset_index(drop=True, inplace=True)

corpus = []
ps = PorterStemmer()

for i in range(0,sms.shape[0]):
    message = re.sub(pattern='[^a-zA-Z]', repl=' ', string=sms.message[i]) #Cleaning special
character from the message
    message = message.lower() #Converting the entire message into lower case
    words = message.split() # Tokenizing the review by words
    words = [word for word in words if word not in set(stopwords.words('english'))]
#Removing the stop words
    words = [ps.stem(word) for word in words] #Stemming the words
    message = ' '.join(words) #Joining the stemmed words
    corpus.append(message) #Building a corpus of messages

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2500)
X = cv.fit_transform(corpus).toarray()

y = pd.get_dummies(sms['label'])
y = y.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

acc_s = accuracy_score(y_test, y_pred)*100
print("Accuracy Score {} %".format(round(acc_s,2)))

def predict_spam(sample_message):
    sample_message = re.sub(pattern='[^a-zA-Z]',repl=' ', string = sample_message)
    sample_message = sample_message.lower()
    sample_message_words = sample_message.split()
    sample_message_words = [word for word in sample_message_words if not word in
set(stopwords.words('english'))]
    ps = PorterStemmer()
    final_message = [ps.stem(word) for word in sample_message_words]
    final_message = ' '.join(final_message)
    temp = cv.transform([final_message]).toarray()
    return classifier.predict(temp)

result = ['Wait a minute, this is a SPAM!','Ohhh, this is a normal message.']

msg = "Hi! You are pre-qulified for Premium SBI Credit Card. Also get Rs.500 worth
Amazon Gift Card*, 10X Rewards Point* & more. Click "

if predict_spam(msg):
    print(result[0])
else:
    print(result[1])

msg = "[Update] Congratulations Nile Yogesh, You account is activated for investment in
Stocks. Click to invest now: "

if predict_spam(msg):
    print(result[0])
else:
    print(result[1])

msg = "Your Stock broker FALANA BROKING LIMITED reported your fund balance
Rs.1500.5 & securities balance 0.0 as on end of MAY-20 . Balances do not cover your bank,
DP & PMS balance with broking entity. Check details at
YOGESHNILE.WORK4U@GMAIL.COM. If email Id not correct, kindly update with your
broker."

if predict_spam(msg):
    print(result[0])
else:
    print(result[1])
```

Output:

1) Dataset values before cleaning and dimensions of the dataset:

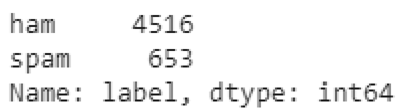


	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

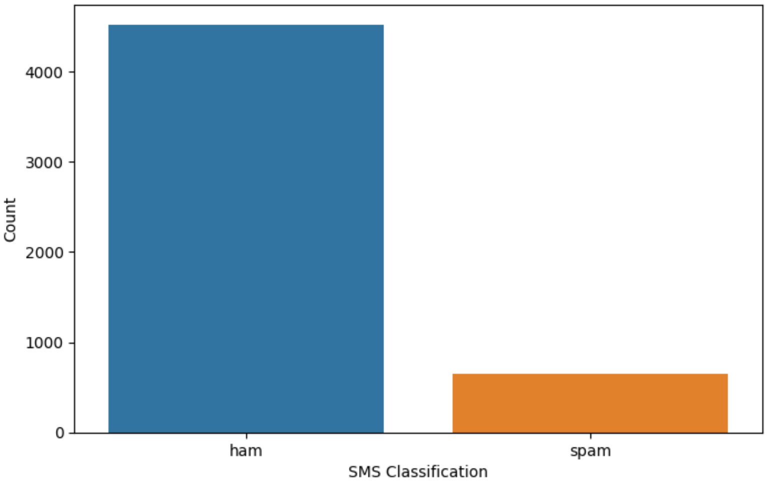


```
(5572, 2)
```

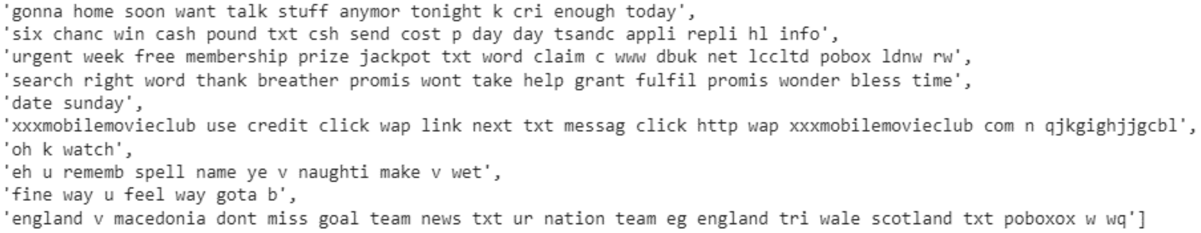
2) Dimensions of the dataset after cleaning the data and no of spam and non-spam:



```
ham     4516
spam     653
Name: label, dtype: int64
```

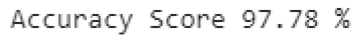


3) Creating Bag of words:



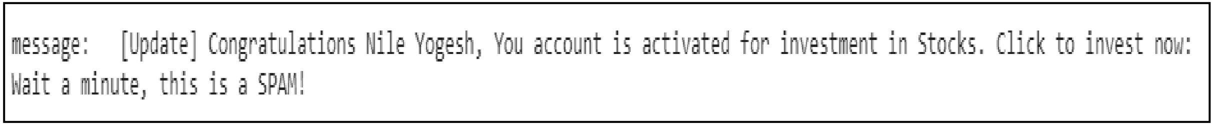
```
'gonna home soon want talk stuff anymor tonight k cri enough today',
'six chanc win cash pound txt csh send cost p day day tsandc appli repli hl info',
'urgent week free membership prize jackpot txt word claim c www dbuk net lccltd pobox ldnw rw',
'search right word thank breather promis wont take help grant fulfil promis wonder bless time',
'date sunday',
'xxxmobilemovieclub use credit click wap link next txt messag click http wap xxxmobilemovieclub com n qjkgighjjgcbl',
'oh k watch',
'eh u rememb spell name ye v naughti make v wet',
'fine way u feel way gota b',
'england v macedonia dont miss goal team news txt ur nation team eg england tri wale scotland txt poboxox w wq']
```

4) Accuracy of model:

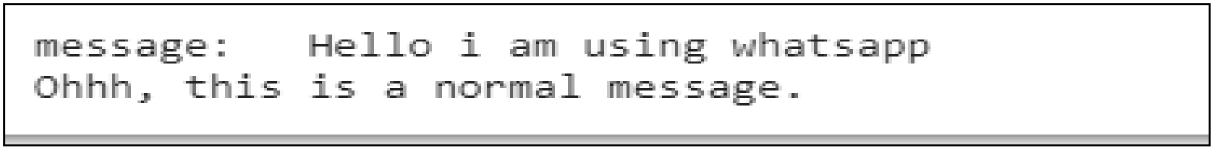


```
Accuracy Score 97.78 %
```

5) Predictive output:



```
message:   [Update] Congratulations Nile Yogesh, You account is activated for investment in Stocks. Click to invest now:
Wait a minute, this is a SPAM!
```



```
message:   Hello i am using whatsapp
Ohhh, this is a normal message.
```

Result:

Implemented and described the working of SMS Spam SMS Detection model.