1. Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

In [1]:
```
#Bfs
graph = {
    'A': ['B', 'C', 'D'],
    'B': ['E'],
    'C': ['D', 'E'],
    'D': [],
    'E': []
}
visited = []
queue = []
def bfs(graph, visited, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print(s)
        for n in graph[s]:
            if n not in visited:
                visited.append(n)
                queue.append(n)

bfs(graph, visited, 'A')
```

```
A
B
C
D
E
```

In [2]:
```
#DFS
graph = {
    'A': ['B', 'C', 'D'],
    'B': ['E'],
    'C': ['D', 'E'],
    'D': [],
    'E': []
}
visited = set()
def dfs(graph, visited, node):
    if node not in visited:
        print(node)
        visited.add(node)
        for n in graph[node]:
            dfs(graph, visited, n)

dfs(graph, visited, 'A')
```

```
A
B
E
C
D
```

2. Implement A star Algorithm for any game search problem.

In [3]:
```
def board(elements):
    for i in range(9):
        if i % 3 == 0:
            print()
        if elements[i] == -1:
            print('_', end = ' ')
        else:
            print(elements[i], end = ' ')
    print()
```

In [4]:
```
def heuristic(start, goal):
    h = 0
    for i in range(9):
        if start[i] != -1:
            row, col = i // 3, i % 3
            g_i = goal.index(start[i])
            g_row, g_col = g_i // 3, g_i % 3
            h += abs(row - g_row) + abs(col - g_col)
    return h
```

In [5]:
```
def moveleft(s, p):
    s[p], s[p - 1] = s[p - 1], s[p]

def moveright(s, p):
    s[p], s[p + 1] = s[p + 1], s[p]

def moveup(s, p):
    s[p], s[p - 3] = s[p - 3], s[p]

def movedown(s, p):
    s[p], s[p + 3] = s[p + 3], s[p]
```

In [6]:
```
def movetile(start, goal, g):
    empty = start.index(-1)
    row, col = empty // 3, empty % 3
    t1, t2, t3, t4 = start[:], start[:], start[:], start[:]
    f1, f2, f3, f4 = 100, 100, 100, 100

    if col - 1 >= 0:
        moveleft(t1, empty)
        f1 = heuristic(t1, goal) + g
    if col + 1 < 3:
        moveright(t2, empty)
        f2 = heuristic(t2, goal) + g
    if row + 1 < 3:
        movedown(t3, empty)
        f3 = heuristic(t3, goal) + g
    if row - 1 >= 0:
        moveup(t4, empty)
        f4 = heuristic(t4, goal) + g

    min_h = min(f1, f2, f3, f4)
    if f1 == min_h:
        moveleft(start, empty)
    elif f2 == min_h:
        moveright(start, empty)
    elif f3 == min_h:
        movedown(start, empty)
    elif f4 == min_h:
        moveup(start, empty)

    return min_h - g
```

In [7]:
```
def solveE(start, goal):
    g = 0
    while True:
        board(start)
        h = heuristic(start, goal)
        print(f"h(n): {h}")
        if h == 0:
            print("Solved!")
            break
        g += movetile(start, goal, g)
```

In [8]:
```
start = [1, 2, 3, -1, 4, 6, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, -1]
'''print("Enter the start state: ")
for i in range (9):
    start.append(int(input()))
print("Enter the goal state: ")
for i in range (9):
    goal.append(int(input()))
print("---------------------------------------------")'''
```

Out[8]:
```
'print("Enter the start state: ")\nfor i in range (9):\n    start.append(int(input()))\nprint("Enter the goal state: ")\nfor i in range (9):\n    goal.append(int(input()))\nprint("---------------------------------------------")'
```

In [9]:
```
print("Start state: ")
board(start)
print("---------------------------------------------")
solveE(start, goal)
```

```
Start state:

1 2 3
_ 4 6
7 5 8
---------------------------------------------
h(n): 3

1 2 3
4 _ 6
7 5 8
h(n): 2

1 2 3
4 5 6
7 _ 8
h(n): 1

1 2 3
4 5 6
7 8 _
h(n): 0
Solved!
```

3. Implement Greedy search algorithm for any of the following application: Selection Sort

In [10]:
```
x = []
n = int(input("How many elements do you want to sort?"))
```

In [11]:
```
for i in range(n):
    x.append(int(input('Enter: ')))
print('Unsorted: ', x)
```

```
Unsorted:  [1, 6, 2]
```

In [12]:
```
for i in range(n-1):
    for j in range(i+1, n):
        if (x[i] > x[j]):
            x[i], x[j] = x[j], x[i]

print('Sorted: ', x)
```

```
Sorted:  [1, 2, 6]
```

4. Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.

In [13]:
```
n = int(input("Enter the value for n\n"))
if n <= 0:
    print("Please enter a positive integer for 'n'.")
    exit(0)
board = [[0 for _ in range(n)] for _ in range(n)]
for row in board:
    print(row)
```

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

In [14]:
```
def check_column(board, row, column):
    for i in range(row, -1, -1):
        if board[i][column] == 1:
            return False
    return True
```

In [15]:
```
def check_diagonal(board, row, column):
    for i, j, in zip(range(row, -1, -1), range(column, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j, in zip(range(row, -1, -1), range(column, n)):
        if board[i][j] == 1:
            return False
    return True
```

In [16]:
```
def nqn(board, row):
    if row == n:
        return True
    for i in range(n):
        if (check_column(board, row, i) == True and check_diagonal(board, row, i) == True):
            board[row][i] = 1
            if nqn(board, row + 1):
                return True
            board[row][i] = 0
    return False
```

In [17]:
```
if nqn(board, 0):
    for row in board:
        print(row)
else:
    print(f"No solution exists for {n}-queens problem.")
```

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

5. Develop an elementary chatbot for any suitable customer interaction application.

In [18]:
```
def greet(name, year):
    print(f'Hello. My name is {name}. I was born in {year}.')
```

In [19]:
```
def name():
    n = input('Please tell me your name: ')
    print(f"That's a nice name {n}.")
```

In [20]:
```
def age():
    print('Let me guess your age: ')
    rem3 = int(input('Enter the remainder of your age when divided by 3: '))
    rem5 = int(input('Enter the remainder of your age when divided by 5: '))
    rem7 = int(input('Enter the remainder of your age when divided by 7: '))
    age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105
    print(f'Your age is {age}.')
```

In [21]:
```
def count():
    n = int(input('Enter a number: '))
    for i in range(n+1):
        print(i)
```

In [22]:
```
def test():
    print("Let's test your programming knowledge: \nWhy are methods used?\n1. To repeat a set of statements.\n2. To interrupt a program.")
    print('3. To divide the program into sub parts. \n4. To check a condition\n')
    print('Enter your choice: ')
    guess = int(input())
    answer = 3
    while guess != answer:
        print('Wrong answer. Try again.')
        guess = int(input())
    print('Right answer.')
```

In [23]:
```
def end():
    print('Have a good day.')
    input()
```

In [24]:
```
greet('TE-A', 2023)
```

```
Hello. My name is TE-A. I was born in 2023.
```

In [25]:
```
name()
```

```
That's a nice name Fari.
```

In [26]:
```
age()
```

```
Let me guess your age:
Your age is 20.
```

In [27]:
```
count()
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

In [28]:
```
test()
```

```
Let's test your programming knowledge:
Why are methods used?
1. To repeat a set of statements.
2. To interrupt a program.
3. To divide the program into sub parts.
4. To check a condition

Enter your choice:
Wrong answer. Try again.
Right answer.
```

In [29]:
```
end()
```

```
Have a good day.
```

6. Implement any one of the following Expert System

In [30]:
```
problem_dict = {
 "printer not working": "Check that it's turned on and connected to the network",
 "can't log in": "Make sure you're using the correct username and password",
 "software not installing": "Check that your computer meets the system requirements",
 "internet connection not working": "Restart your modem or router",
 "email not sending": "Check that you're using the correct email server settings"
}
```

In [31]:
```
def handle_request(user_input):
    if user_input == "exit":
        return "Goodbye!"
    elif user_input in problem_dict:
        return problem_dict[user_input]
    else:
        return "I'm sorry, I don't know how to help with that problem."
```

In [32]:
```
user_input = input("What's the problem? Type 'exit' to quit. ")
response = handle_request(user_input.lower())
print(response)
```

```
Check that it's turned on and connected to the network
```