AI PRACTICALS

1.Write a program to implement depth first search algorithm.

Code:

```
t={
1:[2,3],
2:[4,5],
3:[6],
4:[],
5:[],
6:[]
}
def dfs(node):
    print (node, end=' ')
    for child in t[node]:
        dfs(child)

print("DFS Traversal: ")
dfs(1)
```

2. Write a program to implement breadth first search algorithm.

Code:

```
from collections import deque
t={
  1:[2,3],
  2:[4,5],
  3:[6],
  4:[],
  5:[],
```

```
 6:[]
}

def bfs(root):
 queue=deque([root])
 while queue:
  node=queue.popleft()
  print(node,end=' ')
  for child in t[node]:
   queue.append(child)

print("BFS Traversal:")
bfs(1)
```

3.State the water jug problem. Write a simple program to solve the water jug problem in AI.

Code:

```
def water_jug():
# initial amounts
   x, y = 0, 0

   # capacities and target
   X, Y, Z = 4, 3, 2

   print(f"Initial State: ({x},{y})")

   while True:        #
Goal check        if x
== Z or y == Z:
          print(f"Goal Reached: ({x},{y})")
break

       # If jug1 is empty, fill
jug1        elif x == 0:
x = X
          print(f"Fill Jug 1: ({x},{y})")
```

```
        # If jug2 is not full, pour from jug1 →
jug2        elif y != Y:            pour = min(x,
Y - y)
x = x - pour            y = y +
  pour
            print(f"Pour Jug 1 -> Jug 2:
({x},{y})")


        # If jug2 becomes full, empty jug2
elif y == Y:
y = 0
            print(f"Empty Jug 2: ({x},{y})")

# Run the function
water_jug()
```

5. Solve travelling salesman problems using artificial intelligence techniques.

Code:

```
from itertools import permutations
 d =
[
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]  cities = [0, 1,
2, 3]
 min_cost =
999999 best_path
= None
 for p in
permutations(cities):
    cost = 0      for i in range(len(p) - 1):
cost += d[p[i]][p[i+1]]      cost += d[p[-
1]][p[0]]   # return to start
     if cost <
min_cost:
min_cost = cost
best_path = p
 print("Best Path:", best_path)
print("Minimum Cost:",
min_cost)
```

6. Write a program to solve the Tower of Hanoi problem.

Code:

```
def hanoi(n, source, helper, destination):
if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
return

    hanoi(n-1, source, destination, helper)
    print(f"Move disk {n} from {source} to {destination}")
hanoi(n-1, helper, source, destination)


n = 3

hanoi(n, "A", "B", "C")
```

ML PRACTICALS

11. Implementing a K-Nearest Neighbor (KNN) algorithm.

Code:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

X = np.array([
 [2, 200],
[5, 550],
[1, 150],
 [6, 700],
[3, 300],
 [4, 450]
])

y = ['Low', 'High', 'Low', 'High', 'Low', 'High']

 model = KNeighborsClassifier(n_neighbors=3)

model.fit(X, y)

new_customer = np.array([[3, 400]])
 prediction = model.predict(new_customer)

print("Prediction for (Items=3, Bill=400):", prediction[0])
```