EXPERIMENT NAME
Implementation of Toy Problems
Developing Agent programs for real world problems
Implementation of constraint satisfaction problems
Implementation and Analysis of DFS and BFS for an application
Developing Best first search and A* Algorithm for real world problems
Implementation of mini max algorithm for an application
Implementation of unification and resolution for real world problems
Monty Hall problem
Machine Learning – Linear Progression
NLP and SVM

1.Implementation of Toy Problems

Tower of hanoi: https://www.geeksforgeeks.org/python-program-for-tower-of-hanoi/

```
def TowerOfHanoi(n , source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to destination",destination)
        return
    TowerOfHanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to destination",destination)
    TowerOfHanoi(n-1, auxiliary, destination, source)
n = 3
TowerOfHanoi(n,'A','B','C')
```

2.Developing Agent programs for real world problems

```
def GoForward(rooms, pos):
    while pos != len(rooms):
        if rooms[pos] == False:
            rooms[pos] = True
            print("Room {} is dirty, Clean and moving forward".format(pos+1))
            pos += 1
        else:
            print("Room {} is clean, moving forward".format(pos+1))
            pos += 1
```

```
        print("Reached the End, so turning back")
def GoBackward(rooms, pos):
    while pos >= 0:
        if rooms[pos] == False:
            rooms[pos] = True
            print("Room {} is dirty, Clean and moving back".format(pos+1))
            pos -= 1
        else:
            print("Room {} is clean, moving back".format(pos+1))
            pos -= 1
        print("Reached the start, so turning back")
n = int(input("Enter Number of Rooms: "))
rooms = []
for i in range(n):
    x = int(input("Enter 1 for Clean and 0 for dirty for room {} :".format(i+1)))
    if x == 1:
        rooms.append(True)
    else:
        rooms.append(False)
pos = int(input("Enter Initial position of the vacuum cleaner (1-{})".format(n)))
pos -= 1
while rooms != [True for i in range(n)]:
    GoForward(rooms, pos)
    GoBackward(rooms, pos)
print("All Rooms are cleaned")
```

3.Implementation of constraint satisfaction problems

```
order_of_sq = int(input("Enter order of square: "))
top_left = int(input("Enter top left number: "))
top_left_init = 0
top_left_init += top_left
count = 0
for values in range(1, order_of_sq + 1):
    while count != order_of_sq:
   print(top_left, sep=" ", end=" ")
        top_left += 1
        count += 1
    If top_left == order_of_sq + 1:
```

```
        top_left = 1
print()
count = 0
top_left_init += 1
if top_left_init == order_of_sq + 1:
    top_left_init = 1
    top_left = top_left_init
else:
    top_left = top_left_init
```

4.Implementation and Analysis of DFS and BFS for an application

https://favtutor.com/blogs/depth-first-search-python
https://favtutor.com/blogs/breadth-first-search-python

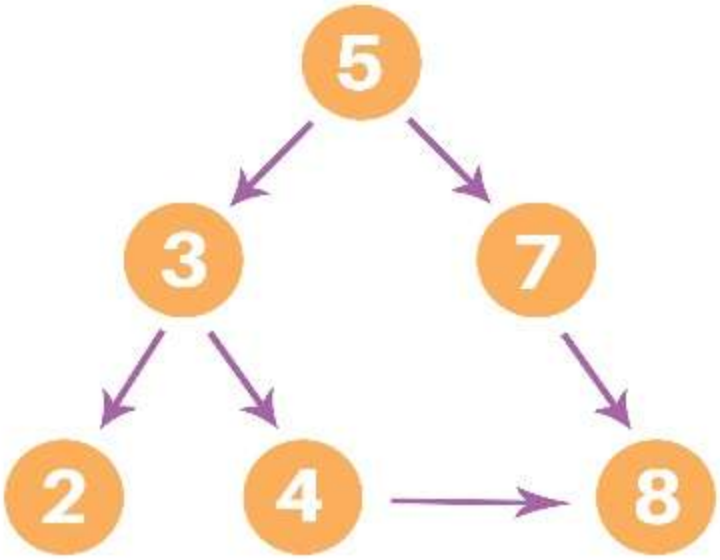

FIGURE 0

BFS:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = []
queue = []
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

DFS:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = set()
def dfs(visited, graph, node):
    if node not in visited:
```

```
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

5a. Developing Best first search
5b. A* Algorithm

6.Implementation of mini max algorithm for an application

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

```
import math
def minimax (curDepth, nodeIndex,
                    maxTurn, scores,
                    targetDepth):
        if (curDepth == targetDepth):
                return scores[nodeIndex]
        if (maxTurn):
                return max(minimax(curDepth + 1, nodeIndex * 2,
                                    False, scores, targetDepth),
                           minimax(curDepth + 1, nodeIndex * 2 + 1,
                                    False, scores, targetDepth))
        else:
                return min(minimax(curDepth + 1, nodeIndex * 2,
                                    True, scores, targetDepth),
                           minimax(curDepth + 1, nodeIndex * 2 + 1,
                                    True, scores, targetDepth))
scores = [3, 5, 2, 9, 12, 5, 23, 23]
treeDepth = math.log(len(scores), 2)
print("The optimal value is : ", end = "")
```

```
print(minimax(0, 0, True, scores, treeDepth))
```

7.Implementation of unification

8.Monty hall

```
import random
def run_trial(switch_doors, ndoors=3):
    chosen_door = random.randint(1, ndoors)
    if switch_doors:
        revealed_door = 3 if chosen_door==2 else 2
        available_doors = [dnum for dnum in range(1,ndoors+1)
                           if dnum not in (chosen_door, revealed_door)]
        chosen_door = random.choice(available_doors)
    return chosen_door == 1
def run_trials(ntrials, switch_doors, ndoors=3):
    nwins = 0
    for i in range(ntrials):
        if run_trial(switch_doors, ndoors):
            nwins += 1
    return nwins
ndoors, ntrials = 3, 10000
nwins_without_switch = run_trials(ntrials, False, ndoors)
nwins_with_switch = run_trials(ntrials, True, ndoors)
print('Monty Hall Problem with {} doors'.format(ndoors))
print('Proportion of wins without switching:
{:.4f}'.format(nwins_without_switch/ntrials))
print('Proportion of wins with switching: {:.4f}'
.format(nwins_with_switch/ntrials))
```

9.Linear Regression

https://www.geeksforgeeks.org/linear-regression-python-implementation/

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
        n = np.size(x)
```

```
        m_x = np.mean(x)
        m_y = np.mean(y)
        SS_xy = np.sum(y*x) - n*m_y*m_x
        SS_xx = np.sum(x*x) - n*m_x*m_x
        b_1 = SS_xy / SS_xx
        b_0 = m_y - b_1*m_x
        return (b_0, b_1)
def plot_regression_line(x, y, b):
        plt.scatter(x, y, color = "m",
                            marker = "o", s = 30)
        y_pred = b[0] + b[1]*x
        plt.plot(x, y_pred, color = "g")
        plt.xlabel('x')
        plt.ylabel('y')
        plt.show()
def main():
        x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
        b = estimate_coef(x, y)
        print("Estimated coefficients:\nb_0 = {} \
                \nb_1 = {}".format(b[0], b[1]))
        plot_regression_line(x, y, b)
if __name__ == "__main__":
        main()
```

10.SVM