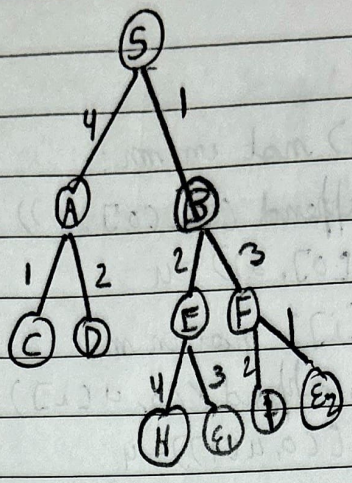
INDEX

NAME: Pulkit Shringi STD.: _____ SEC.: I I ROLL NO.: 596 SUB.: Artificial Intelligence

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	22/1/24	Implementation of toy problem.	10	
2	31/1/24	Implementation of real world problem	10	31/01/24
3	~~1/1~~ 07/2/24	Implementation of constraint satisfaction problem	10	14/01/24
4	14/2/24	Breadth first search	10	
5	~~to~~ 14/2/24	Depth first search	10	21/2/24
6	6/3/24	Implementation of BFS and A* algorithm		

- BFS :



output :

shortest path length : 4

Path : [(0,0), (0,1), (0,2), (1,2), (2,2)]

6/3/24

Ex-6

Implementation of Best First Search and A* algorithm

- **Aim:**

To implement and write the code of Best first search and A* algorithm

- **BFS →**

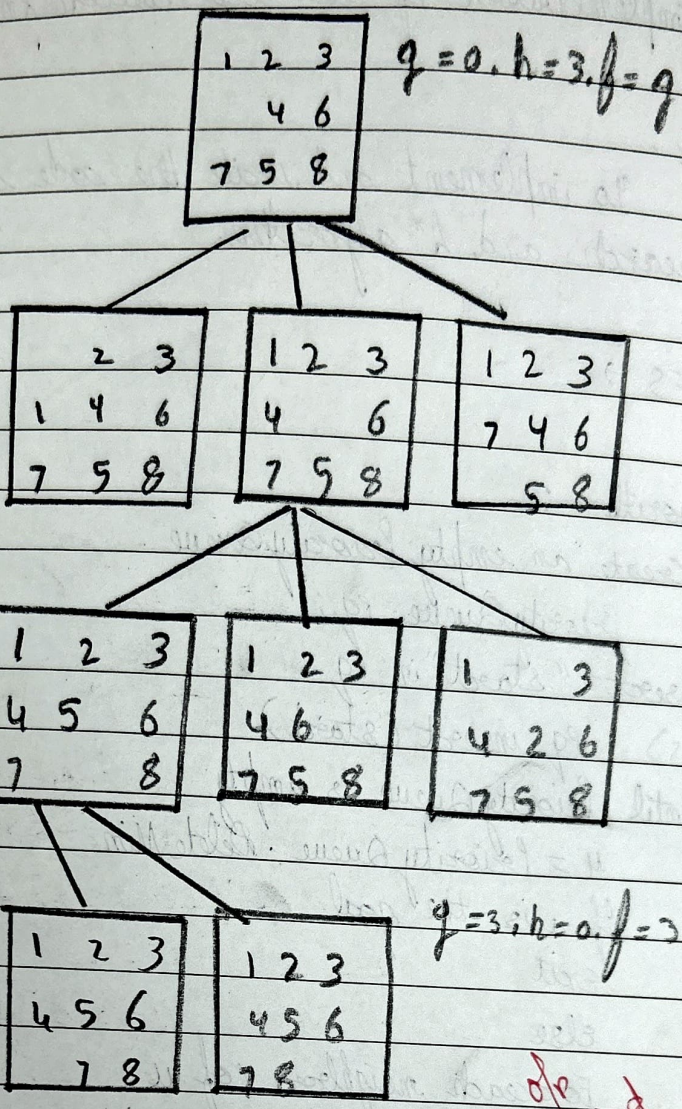
- Algorithm:

1) Create an empty PriorityQueue
   PriorityQueue pq;
2) Insert "start" in pq.
   pq.insert(start)
3) Until PriorityQueue is empty
   u = PriorityQueue.DeleteMin
   If u is the goal
   Exit
   Else
   Foreach neighbor v of u
   If v "unvisited"
   Mark v "Visited"
   pq.insert(v)
   Mark u "Examined"
   End procedure.

- **A* Algorithm →**

1) Make an open list containing starting node.
2) If it reaches the destination node:
- Make a closed empty list.
- If it does not reach the destination node then consider a node with the lowest f-score in the

A*



o/p verified

output:

shortest path length: 6

Path: [(0,0), (1,0), (2,0), (3,0), (3,1), (3,2), (3,3)]

open list.
. We are finished.
3) Else: Put the current node in the list and check its neighbors
4) For each neighbor of the current node:
. If the neighbor has a lower g value than the current node and is in the closed list: Replace neighbor with this new node as the neighbor's parent.
5) Else if (current g is lower and neighbor is in the open list):
6) Replace neighbor with the lower g value and change the neighbor's parent to the current node.
Else If the neighbor is not in both lists:
7) Add it to the open list and set its g.

. Code:

```
from queue import PriorityQueue

class PuzzleNode:
    def __init__(self, state, parent=None, move=0, g=0, h=0):
        self.state, self.parent, self.move, self.g, self.h = state, parent, move, g, h

    def __lt__(self, other):
        return self.g + self.h < other.g + other.h
```

```
def heuristic (state, goal state)
    return sum (s1 != g1 for s,g in zip(state, goal state))

def get_neighbours (state):
    z, r, c = state.index(0), divmod (state.index(0), 3).
    return [state[:z] + state[n] + state[z+1:]
        for n in (r*3+c for d in ((-1,0),(1,0),(0,-1),
            (0,1))) if 0<=n<9]

def reconstruct_path (state):
    return [] if node is none else reconstruct_path (parent).

def a_star (start, goal):
    start_node = PuzzleNode (string, g=0, h=
    heuristic (s, g)).
    open_set, closed_set = priorityQue().set()
    open_set.put (start_node)
    while not open_set.empty():
        node = open_set.get()
if node.state == goal: return reconstruct_path()
    return none
state_goal = [[1,2,3,0,4,6,7,5,8],[1,2,3,4,5,6,7,8,0]]
puts = a_star (start, goal) print (calm).
```

Result:

Best first search and A* algorithm was successfully implemented.