LAB-4 8-puzzle

Algorithm for A*

Creat a 3x3 grid, leaving 1 space empty.

→ Initialize:

- set the initial stete of the puzzle
- set the goal state of the puzzle

→ use priority queue:

- use the queue to store the different states of puzzle, like

$f(n) = g(n) + h(n)$

g(n) → no.of moves

h(n) → Counts the no.of ~~times~~ tiles not in their goal state.

→ States:

generat all possible new states, take the smallest f(n), if it is the solution then return it, else try other new states (move the blank tile left, right, up, down).

→ Calculate:

for each new state calculate g(n) & h(n) and add it to with f(n) to the queue

→ Repeat until the goal state is reached.

Initial state.

1 2 3
8 0 4
7 6 5

goal state

2 8 1
0 4 3
7 6 5

↓

1 0 3
8 2 4
7 6 5

$f(n) = 0+6$
$= 6$

↓

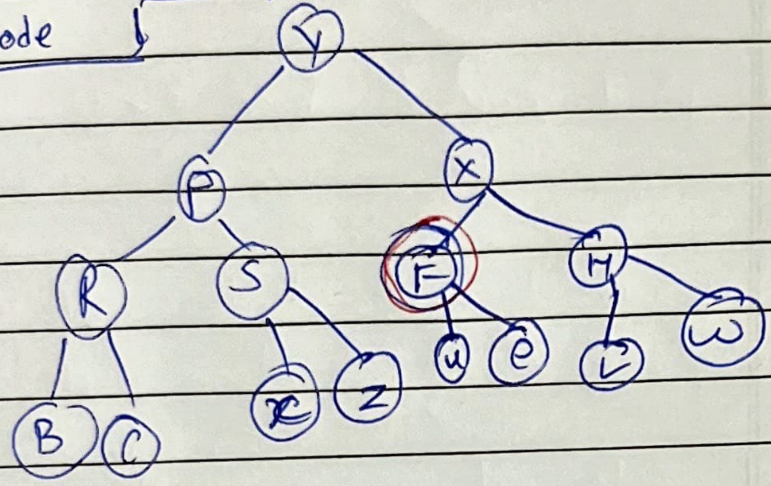
1 2 3
8 4 0
7 6 5

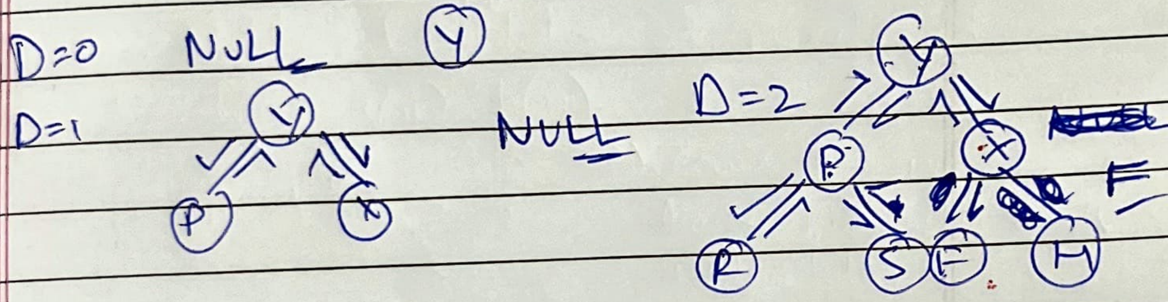
$f(n) = 0+5$
$= 5.$

Pseudocode

```
1. Initialize priority queue:
      - set g(n)=0
      - set h(n)= no. of misplaced tiles.
      - set f(n) = g(n) + h(n)
2. Remove smallest f(n)
      - See if its the goal state
        return solution.
        else
        generate all possible state.
        calculate g(n), h(n) & f(n)
3. If goal reached, return solution.
```

~~Algorithm~~ for IDF
Pseudocode





```
function IDS(root, g):
    for D = 0 to ∞:
        r = DLS(root, g, D)
        if r ≠ NULL:
            return r
    return NULL
function DLS(node, g, D):
    if D==0 and node == g:
        return node
    if D>0:
        for child in node.children:
            r=DLS(child, g, D-1)
            if result ≠ NULL:
                return r
    return NULL.
```

15/10/24