Week - 04 >

15/10/2024

Date __/__/__

Page ______

Solving 8 puzzle using IDSA & A*

Step 1 :- initialize the problem
- Define the initial state of puzzle (3x3 grid)
- Define Goal state.
- Randomly arranged list of range [1 - 8]
- Define 1 empty block / space as '0' called tiles.

initial state.

1	2	3
8		4
7	6	5

final state.

2	8	1
	4	3
7	6	5

Step 2 :- Defining the method A*

to solve the problem we use the manhattan method to find distance b/w initial final states.

distance += abs(i - goal - i) + abs(j - goal - j)
return distance.

get neighbour states
using moves [(0 1) (1 0) (-1 0) (0 -1)
find neighbour state to present state.

priority Queue
implementing priority Queue. to select (or) chose next move.
chose the ~~lowest distance and. move the current state to lowest state.~~
~~if (current state == final state):~~
else ~~return~~ path
find lowest (distance): move to lowest state

Date __/__/__
Page ____

using back tracking to return path
path cost
backtrack the move to print the path
?

initial state

1	2	3
8	0	4
7	6	5

Final state.

2	8	1
	4	3
7	6	5

priority queu

priority	State	H	V	Dist
2	1	2	0	2
1	2	1	0	1
1	3	0	1	1
1	4	1	0	1
0	5	0	0	0
0	6	0	0	0
0	7	0	0	0
2	8	1	1	2

highest Distance state has highest priority.
lowest priority perform first

$$\begin{matrix}1&2&3\\8&0&4\\7&6&5\end{matrix} \rightarrow \begin{matrix}1&0&3\\8&2&4\\7&6&5\end{matrix} \rightarrow \begin{matrix}0&1&3\\8&2&4\\7&6&5\end{matrix} \rightarrow \begin{matrix}8&1&3\\0&2&4\\7&6&5\end{matrix} \rightarrow \begin{matrix}8&1&3\\2&0&4\\7&6&5\end{matrix}$$

$$\begin{matrix}8&1&3\\2&4&0\\7&6&5\end{matrix} \rightarrow \begin{matrix}8&1&0\\2&4&3\\7&6&5\end{matrix} \rightarrow \begin{matrix}8&0&1\\2&4&3\\7&6&5\end{matrix} \rightarrow \begin{matrix}0&8&1\\2&4&3\\7&6&5\end{matrix} \rightarrow \begin{matrix}2&8&1\\0&4&3\\7&6&5\end{matrix}$$

Date__/__/__

Page_____

finding the minimal path ~~[illegible]~~ [illegible] b/w 2 Notes in graph

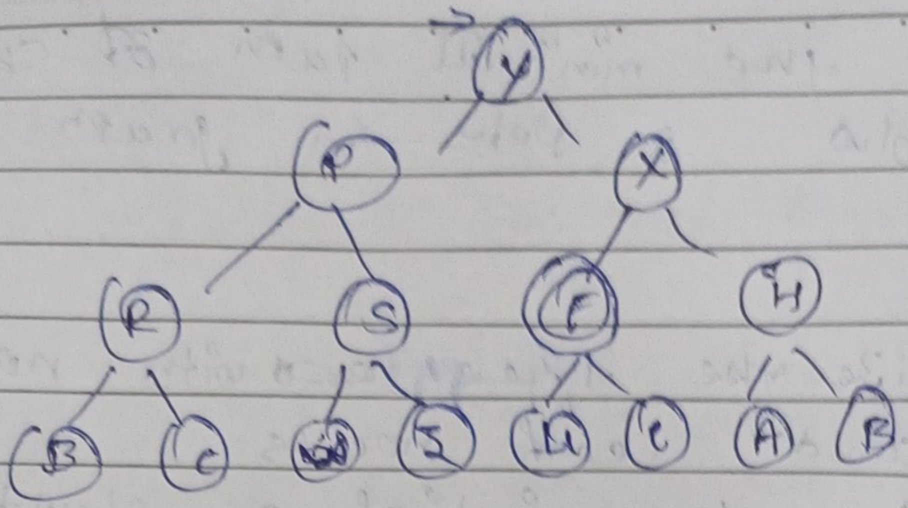
Step 1 :-

initialize the tree ~~and~~ with node and leaf nodes
mention the intial or start node and destination node.

```
# find destination node first.
find() {
    using BFS method () :
        find level by level for
            destination node.
        if. present .
            return Level
        else
            go to next level

# find the parent node. until reach
    start node.
find_Parent () {
    Back track the path of current
        node to get parent
    and store it in the list
    if it is Parent node
        return false /distance 0
```

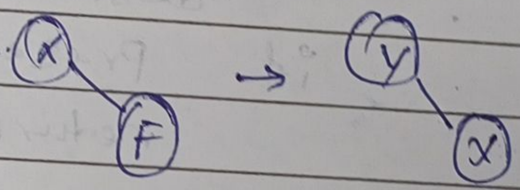
# Back track and print path
Back track destination to start node to print path.



initial / start node : Y
destination node : F

BFS :- level 1 : Y   False next level
level 2 : P, X   False next level
level 3 : R S F   Find F Break

Find-Parent :-



Y → start found.

Back track to print path.

Path :-

$$\begin{tikzcd} Y \arrow[r] & X \arrow[r] & F \end{tikzcd}$$

[illegible]
16/10/24