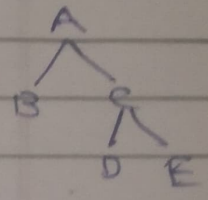
15/10/2024

DATE: PAGE:

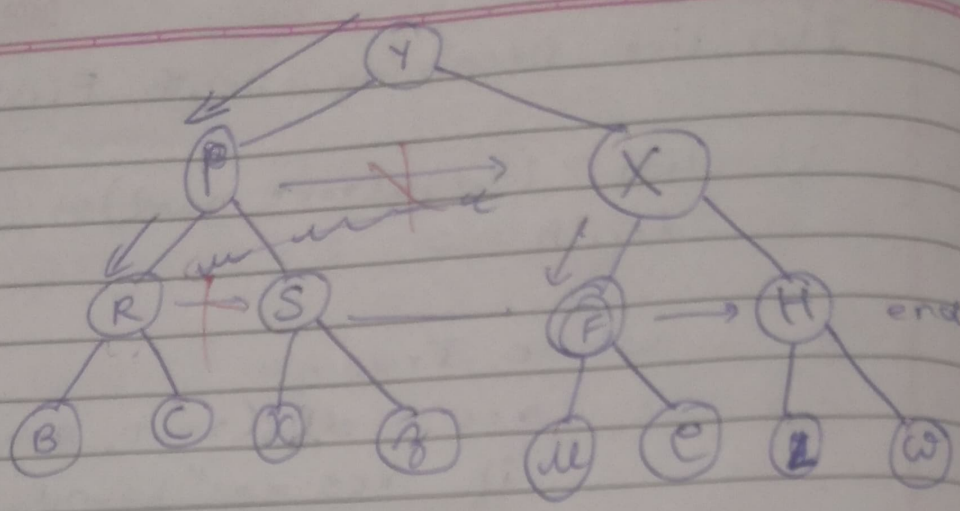
Iterative Deepening Depth First Search

```
function iddfs(root, goal):
    depth = 0

    while True:
        res = dfs(root, goal, depth)
        if res == "found":
            return "Goal State Found"

        depth += 1
```




```
function dls(node, goal, depth):
    if node == goal and depth == 0:
        return "found"
    else if depth > 0:
        for each child in children[node]:
            if child == goal:
                return "found"
            res = dls(child, goal, depth-1)
            ~~return "not found"~~
            if res == "found"
                return "found"
    return "not found"
```

DATE: PAGE:



root = Y depth = 0 , goal = F

Y : [P, X] , P : [R, S] , X : [F, H]
R : [B, C] , S : [x, z] , F : [u, e],
H : [L, w] , B : [ ] , C : [ ],
x : [ ] , z : [ ] , u : [ ] , e : [ ]
L : [ ] , w : [ ]

* if Y == F → return "not found"
* depth = 0 + 1

child in [P, X]

if P == F ~~not~~ false
X == F false
return not found

* depth = 1 + 1 = 2

=

P . [R, S] ≠= F
X . [F, H]

F == F found

ends at H.

DATE: PAGE:

* A* for 8 puzzle problem.

function A star( )

initial state

1	2	3
8		4
7	6	5

g s

2	8	1
	4	3
7	6	5

goal_state = [[2, 8, 1], [0, 4, 3], [7, 6, 5]]

```
find_Blank_tile(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
    return None

manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            tile = state[i][j]
            if tile != 0:
                goal_x, goal_y = divmod
                                 ((t-1), 3)
                distance += abs(i - goal_x)
                          + abs(j - goal_y)
    return distance.
```

DATE: PAGE:

```
make_move(state, move):
    n_s = state
    blank_x, blank_y = find_blank_tile(state)

    if move == "up" and blank_x > 0:
        newstate swap(n_s[x][y], n_s[x-1][y])

    elif move == "down" and x < 2:
        swap(n_s[x][y], n_s[x+1][y])

    elif move == "right" and y < 2:
        swap(n_s[x][y], n_s[x][y+1])

    elif move == "left" and y > 0:
        swap(n_s[x][y], n_s[x][y-1])

    return n_s

function a_star(start_goal)
    open_list = priority_heap()
    open_list.push(start, manhatten(start, goal))

    close_list = set()
    g_score = {}
    g_score[start] = manhatten(start, goal)
    parent = {}

    while ! open_list.empty():
        cur_state = open_list.pop()
        if cur_state == goal_state
```

DATE: PAGE:

return path(parent, curr)

close_list.add(curr)
for neighbour in neighbours[curr]:
if neighbour in close_list:
continue
newg = g_score[curr] + 1
if neighbour not in open_list
~~and not in close_list~~
or newg < g_score[neighbour]
parent[neighbour] = curr
g_score[neighbour] = newg

f_score = g_score[neighbour] +
manhattan(neighbour, goal)

open_list.add(neighbour, fscore)

return "No solution"

- From valid_state we get neighbour to blank.

- function path(parent, curr)
followed = [curr]
while curr in parent:
curr = parent[curr]
followed.append(curr)
print(reverse(path))

15/10/24