Date ______
Page ______

Lab-4 Iterative deepening Search and A*

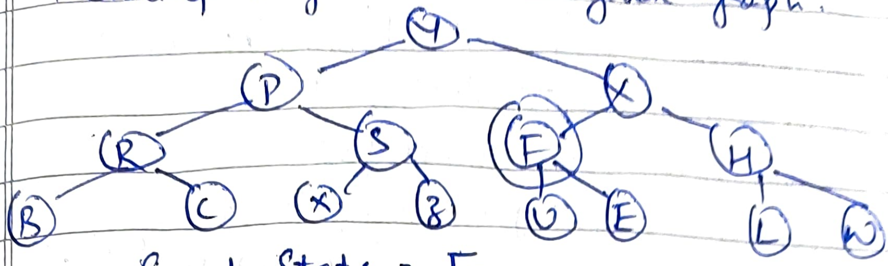
→ **Iterative deepening Search:**

```
function ids (root, ~~x~~ goal):
    Stack = []
    for limit in range (~~x~~ height (tree)):
        Stack. append (root)
        visited = []
        ~~if~~ while stack:
            ~~if~~ top = Stack [-1]
            if top.left is not None and
               top.left not in visited:
               Stack. append (top.left)
               visited. append (top.left)
               ~~if~~ if len (Stack) >= limit:
                    Stack. pop()
            ~~elif top. right~~ Continue
            elif top.right is not None and
                top.right not in visited
                Stack. append (top.right)
                visited. append (top.right)
                if len(Stack) >= limit:
                    Stack.pop()
                continue
            Stack.pop()
    return
```

Utility Code:

```
class Node:
    def __init__(~~x~~self, left= ~~right~~ None, right=None, data):
        Self.left = left
        Self.right = right
        Self.val ~~data~~ = data
```

Running algorithm on given graph:



Goal State = F

Limit = 0
Explored : Y (In order)

Limit = 1
Explored : Y P X (In order)

Limit = 2
Explored : Y P R S X F H
(In order)

Goal state found, therefore ends.

→ A* algorithm on 8-puzzle problem:
Taking ~~gf's~~ gf(x) as Manhattan Distance
g(x) as No. of misplaced tiles

– Manhattan distance:

```
def getMD(board, endgoal):
    for i in range(9):
        old = get_index(i, board)
        end = get_index(i, end)
    total_MD = 0
    for i in range(9):
        old = get_index(i, board)
        end = get_index(i, end)
        total_md += (abs(end[0] - old[0])
                     + abs(end[1] - old[1]))
    return total_MD
```

- No. of misplaced tiles:

```
def misplaced (board, ~~new~~ end):
    total = 0
    for i in range (3)
        for j in range (3):
            if board [i][j] != end [i][j]:
                total += 1
    return total
```

Using A* with the following start and end States:

Initial : 1 2 3 / 8 0 4 / 7 6 5

$8 \cdot f(x) = f(x) = 8$
$g(x) = g(x) = 6$
$f(x) =$
$2+1+1+2+1+1+0$

Goal : 2 8 1 / 0 4 3 / 7 6 5

First Move:

Possible States:

① 1 0 3 / 8 2 4 / 7 6 5

$f(x) = 10$
$g(x) = 6$
$h(x) = 16$

② 1 2 3 / 8 6 4 / 7 0 5

$f(x) = 11$
$g(x) = 7$
$h(x) = 18$

③ 1 2 3 / 0 8 4 / 7 6 5

$f(x) = 6$
$g(x) = 5$
$h(x) = 11$

15/10/24

④

1	2	3
8	4	0
7	6	5

$f(x) = 8$
$g(x) = 5$
$h(x) = 13$

Choosing ③

Possible moves:

①

0	2	3
1	8	4
7	6	5

$f(x) = 8$
$g(x) = 6$
$h(x) = 14$

②

1	2	3
7	8	4
0	6	5

$f(x) = 8$
$g(x) = 7$
$h(x) = 15$

None better, therefore choosing ④ from previous state

Possible moves:

①

1	2	0
8	4	3
7	6	5

$f(x) = 8$
$g(x) = 4$
$h(x) = 12$

Better ~~②~~ therefore choosing said state

Possible moves:

①

1	0	2
8	4	3
7	6	5

$f(x) = 8$
$g(x) = 4$
$h(x) = 12$

~~Choosing above state~~ :

Utility function:

```
~~get-i~~ def get_index(el, board):
    for i in range(3)
        for j in range(3):
            if board[i][j] == el:
                return [i, j]
```

15/10/24