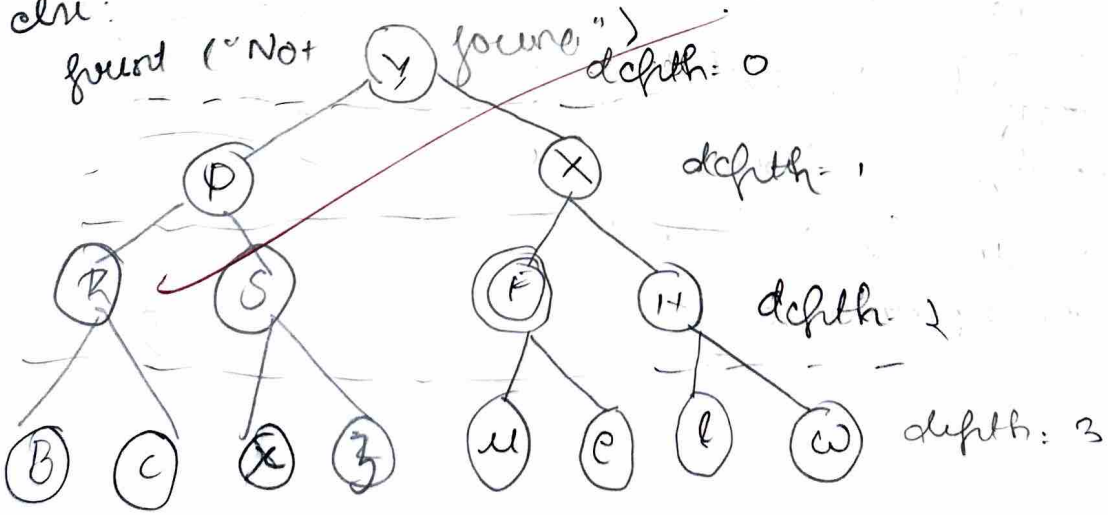
15/10/24

Iterative deepening search

```
def - ids (start, goal, graph)
if depth == 0 :
    if node == goal
      return node

elif depth > 0:
  result = ids (child, goal, depth - 1)
if result is not None
    return [node] + result

depth = 0
while True
  result = ids (start, goal, depth)
if result is not None
    return result
  depth = depth + 1
def main ( )
  noe = input ("Enter the number of edges")
start_node = input ("Enter start node")
  goal_node = input ("Enter goal node")
  if path:
    print ("Path found")
else:
    print ("Not found")
```



level 0 : Y

Level 1 : Y → P → X

level 2 : Y → P → R → S → X → F

8 Puzzle game

Initial state

1	2	3
8		4
7	6	5

goal state

2	8	1
	4	3
7	6	5

A* algorithm

$F(N) = H(N) + G(N)$

```
def H_n(state, target)
    return sum(x1 - y1)
```

The function F_n is used to decide which state to explore next. it adds the moves taken and remaining moves to determine cost

```
def possible_moves(state, lvl, visited_states):
    directions = {'d': down, 'u': up, 'l': left, 'r': right}
    if b <= 5: directions.append('d')
    if b >= 3: directions.append('u')
    if b - 1 - 3 > 0: directions.append('l')
    if b - 1 - 3 < 2: directions.append('r')
```

0 1 2
3 4 5
6 7 8

0, 3, 6

2, 5, 8

```
    if temp not in visited_states:
        pos_moves.append([temp, lvl+1])
```

It checks if the new state has not been visited yet if not it adds to list of

function to generate a new state based on a move possible
In this we create a copy of the current state to avoid modifying the original

```
    temp[b], temp[b+n] = temp[b+n], ' '   # swap empty tile with its neighbor
for i in range(3):
    for j in range(3):
        display the state in 3x3 format
```

```
def astar(src, target):
    arr = [[src, 0]] # list of states to explore
    visited_states = []
    iterations = 0
```

display the current state if the current state matches the target state return the no of iterations

```
    display_state(current[0])
    if current[0] == target
        return iterations
```

In the main function add visited states to list the user should give initial state & goal state & call a function

Proceed

Iterative deepening search

```
def ids(graph, start, goal):
    def dls(node, goal, depth):
        if depth == 0:
            if node == goal:
                return [node]
            else:
                return None
        elif depth > 0:
            for child in graph.get(node, []):
                result = dls(child, goal, depth-1)
                if result is not None:
                    return [node] + result
        return None

    depth = 0
    while True:
        result = dls(start, goal, depth)
        if result is not None:
            return result
        depth += 1

def main():
    graph = {}
    noe = int(input("Enter number of edges"))
    print("Enter each edge in the format node1 node2")
    for _ in range(noe):
        node1, node2 = input().split()
        if node1 in graph:
            graph[node1].append(node2)
        else:
            graph[node1] = [node2]
        if node2 in graph:
            graph[node2].append(node1)
```

```
else:
  graph[node2] = [node1]
return graph
def main():
 graph = main()
 s_node = input("Enter starting node:")
 g_node = input("Enter goal node:")
 path = ids(graph, s_node, g_node)
 if path:
   print("Path found")
 else
   print("No path found")

if __name__ == "__main__":
  main()
```

OIP:

=

Enter the number of edges: 14

Enter each edge

YP RC

YX SX

PR SZ

PS FM

XF FC

XH HL

RB HW

Enter starting node: Y

Enter goal node: F

Path found: Y → X → F

A star

```
def H_n(state, target):
    return sum(x!=y for x,y in zip(state, target))

def F_n(sl, target):
    state, level = sl
    return H_n(state, target) + level

def possible_moves(sl, visited_states):
    state, lvl = sl
    b = state.index(0)
    directions = []
    for_moves = []
    if b<=5:
        directions.append('d')
    if b>=3:
        directions.append('u')
    if b%3>0:
        directions.append('l')
    if b%3<2:
        directions.append('r')
for move in directions:
    temp = gen(state, move, b)
    if temp not in visited_states:
        for_moves.append([temp, level+1])

def gen(state, move, b):
    temp = state.copy()
    if move == 'l': temp[b], temp[b-1] = temp[b-1], temp[b]
    if move == 'r': temp[b], temp[b+1] = temp[b+1], temp[b]
    if move == 'u': temp[b], temp[b-3] = temp[b-3], temp[b]
    if move == 'd': temp[b], temp[b+3] = temp[b+3], temp[b]

    return temp
```

```
def display_state(state):
    print("Current state")
    for i in range(0,9,3):
        print(state[i:i+3])
    print()

def astar(src, target):
    arr = [[src,0]]
    visited = []
    i = 0
    while arr:
        i += 1
        current = min(arr, key = lambda x: f_n(x,target))
        arr.remove(current)
        display_state(current[0])
        if current[0] == target:
            return iterations
        visited_states.append(current[0])
        arr.extend(possible_moves(current, visited_states))
    return "Not found"
src = [1,2,3,8,0,4,7,6,5]
target = [2,8,1,0,4,3,7,6,5]
print(astar(src, target))
```

Output

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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