LAB-(4):

~~Iterative deepening search~~

15/10/24

8 puzzle problem using A* search:

Algorithm:

Initial state

1	2	3
8	3	4
7	6	5

Goal state

2	8	1
	4	3
7	6	5

Cost to reach - g(n)
estimated cost h(n)
f(n) = h(n) + g(n)
MD → dist from curr. tile to goal tile
A* search
↳ get the lowest value uses priority queue

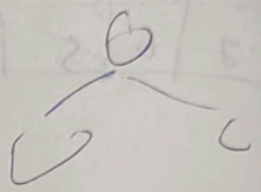
~~def func~~

```
manhattan_dist()
{
   get the dist from current tile to the goal tile
   return distance
}

get_neighbours()
{
   neighbours = []
   for each move (up, down, right, left)
        new = move
        if new is not in visited:
             dist = manhattan (next, goal)
             add (new, dist) to neighbours
   sort neighbour by dist
   for each (new, dist in neighbours:
             add to stack
}

check()
{
   if (current_state == goal_state return true
}

a_star()
{
   open_list = []
   heap.push (open_list, manhattan_dist)

   cost_so_far = { }
   Path = { }
```

while priority_queue:
    if current == goal:
        return path

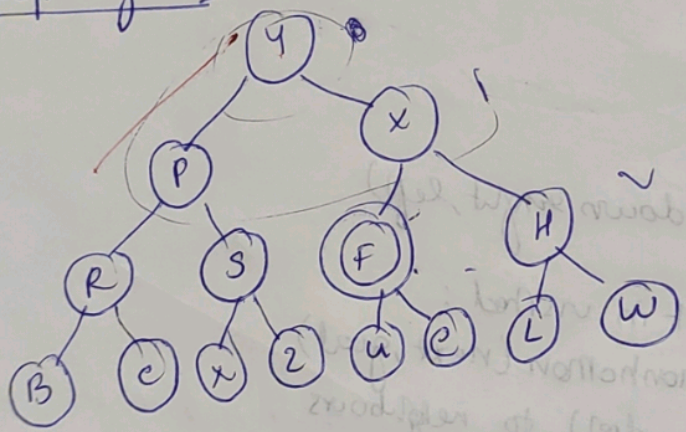
    for neighbor in neighbors list:
        get the lowest neighbour_cost
        cost_so_far.add(neighbor)
        heap.push(priority_queue)
        path [neighbor]

}



path()
{
    backtrack from the reached goal state to start state
}

Iterative Deepening search



function (source, target, level_max)
{
    for limit 0 to level_max:
        if (DLS (src, target, limit) == true)
            return true

    return false
}

```
DLS (src, target, limit)
{ if (src == target)
      return true
  if (limit <= 0)
      return false
  for each i adjacent of src
      if DLS (i, target, limit - 1)
          return true

  return false
}
```

Proceed

Path

Code:

A* search:

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

def possible_moves(state_with_lvl, visited_states):
    state, lvl = state_with_level
    b = state.index(0)
    directions = []
    pos_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')
    return pos_moves

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]
```

```
def astar(src, target):
    arr = [[src, 0]]
    visited_states = []
    n = 0
    while arr:
        n += 1
        current = min(arr, key = lambda x: fn(x, target))
        arr.remove(current)
        if current[0] == target:
            return 'Found'
        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:

currentstate:

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

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

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

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

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

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

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

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

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

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

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

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

Found with 40 iterations.

Iterative deepening search

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

    return None

    depth = 0
    while True:
        result = depth_search
        depth += 1

def get_graph():
    graph = {}
    num_edges = int(input("Enter no. of edges:"))
    print("Enter each edge in format 'node1 node2':")
    for _ in range(num_edges):
        node1, node2 = input().split()
        if node1 in graph:
            graph[node1] = [node2]
        else
            graph [node2] = [node1]

    return graph
def main():
    graph = get_graph()
    start_node = input("Enter starting node:")
    goal_node = input("Enter the goal node:")
```

```
path = iterative(graph, start_node, goal_node)
if path:
    print("Path found :", join(path))
else:
    print("No path found")

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

Output:

```
Enter the no. of edges : 14
Enter each edge in format 'node1 node2' :
Y P
Y X
P R
P S
X F
X H
R B
R C
S X
S Z
F U
F E
H L
H W
Enter the starting node : Y
Enter the goal node : F
Path found : Y → X → F
```

15/10/24