Laboratory - 3

implement A* search algorithm

8 puzzle problem

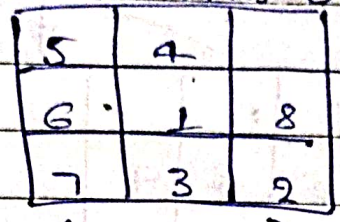
initial

5	4	
6	1	8
7	3	2

final

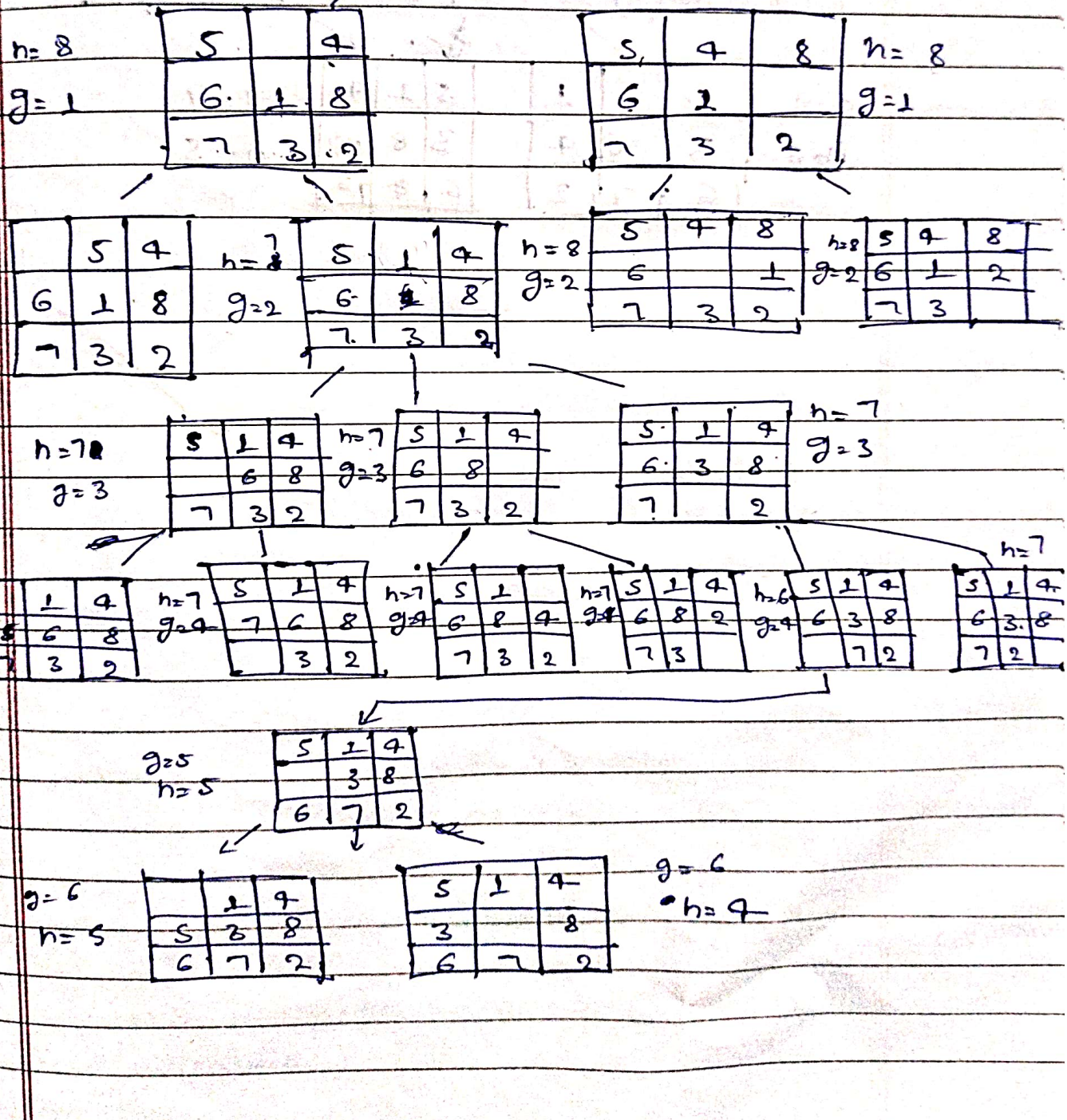
	1	2
3	4	5
6	7	8

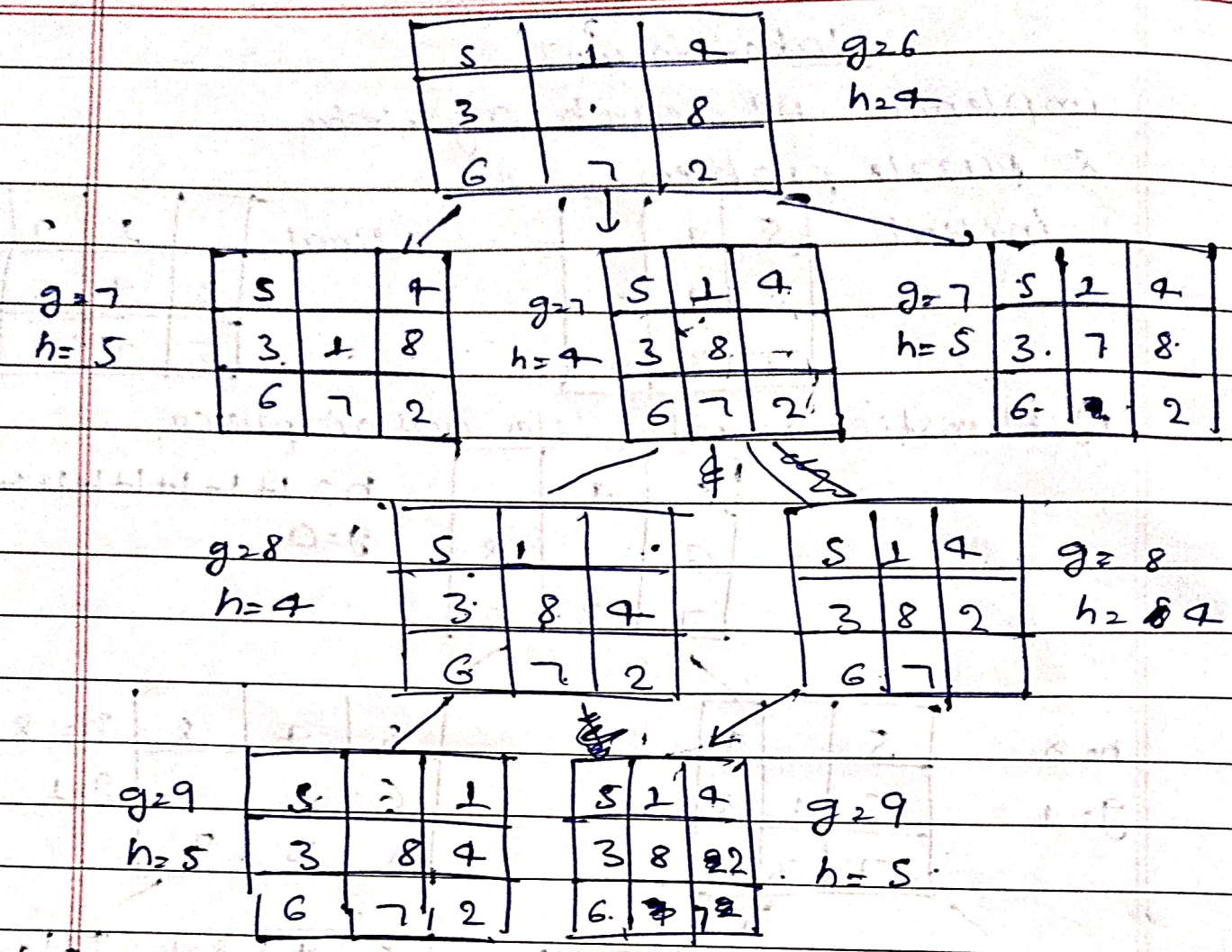
1) Heuristic: Number tiles out of place.



$h = 1+1+1+1+1+1+1+1 = 8$

$g = 0$




g=6
h=4
g=7
h=5
g=7
h=4
g=7
h=5
g=8
h=4
g=8
h=4
g=9
h=5
g=9
h=5

algorithm

function A_STAR_Manhattan(start_state, goalstate)
Misplaced_Tiles

open_list ← priority_queue containing start state
with f(start_state)=0

closed_list ← empty set;
g[start_state] ← 0
parents[start_state] ← None

While open_list is not empty:
current_state ← state in open_list with lowest f(n)
if current state equals goal state:
return reconstruct_path(parents, current_state)
remove current_state from open_list
add current_state to closed_list
for each neighbour of current_state
if neighbour is in closed_list:
continue
tentative_g ← g[current_state]+1
if neighbour is not in open_list or
tentative_g < g[neighbour]:
parents[neighbor] ← current state
g[neighbor] ← tentative_g
f[neighbor] ← g[neighbor] + misplaced_tiles(neighbor, goalstate)
if neighbor is not in open_list:
add neighbor to open_list with priority f[neighbor]

return failure

function misplaced_tiles(state, goalstate)
count ← 0

```
for each tile in state:
    if tile is not in its correct position
        and tile is not blank:
            count ← count + 1
    return count

function reconstruct_path (parents, current_state)
    path ← empty list
    while current_state is not None:
        add current_state to path
        current_state ← parents[current_state]
    return reverse(path)
```

Sun
15/10/22

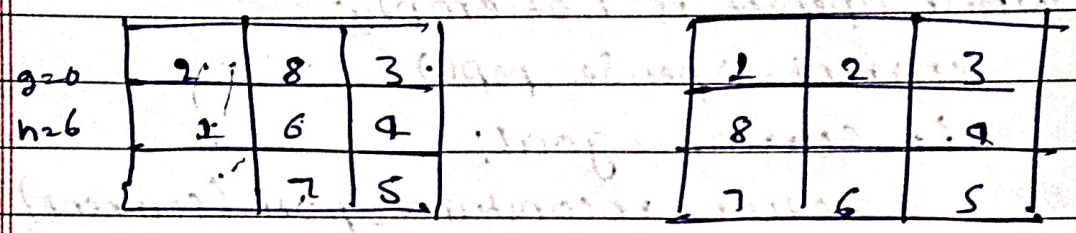
A* algorithm with Manhatten Distance

```
function A_star(start, goal, grid)
    openset = priorityQueue()
    closedset = set()
    start.g = 0
    start.h = manhatten_dist(start, goal)
    start.f = start.g + start.h

    openSet.push(start)

    while openset is not empty:
        current = openSet.pop()
        if current == goal:
            return reconstruct_path(current)
        closedSet.add(current)

        for neighbour in current neighbours:
            if neighbour in closedset:
                continue
            tentative_g = current.g + dist_bw(current, neighbour)
            if neighbour not in openset:
                openSet.push(neighbour)
            if tentative_g >= neighbour.g:
                continue
            neighbour.comeFrom = current
            neighbour.g = tentative_g
            neighbour.h = manhatten_dist(neighbour, goal)
            neighbour.f = neighbour.g + neighbour.h
    return failure
```

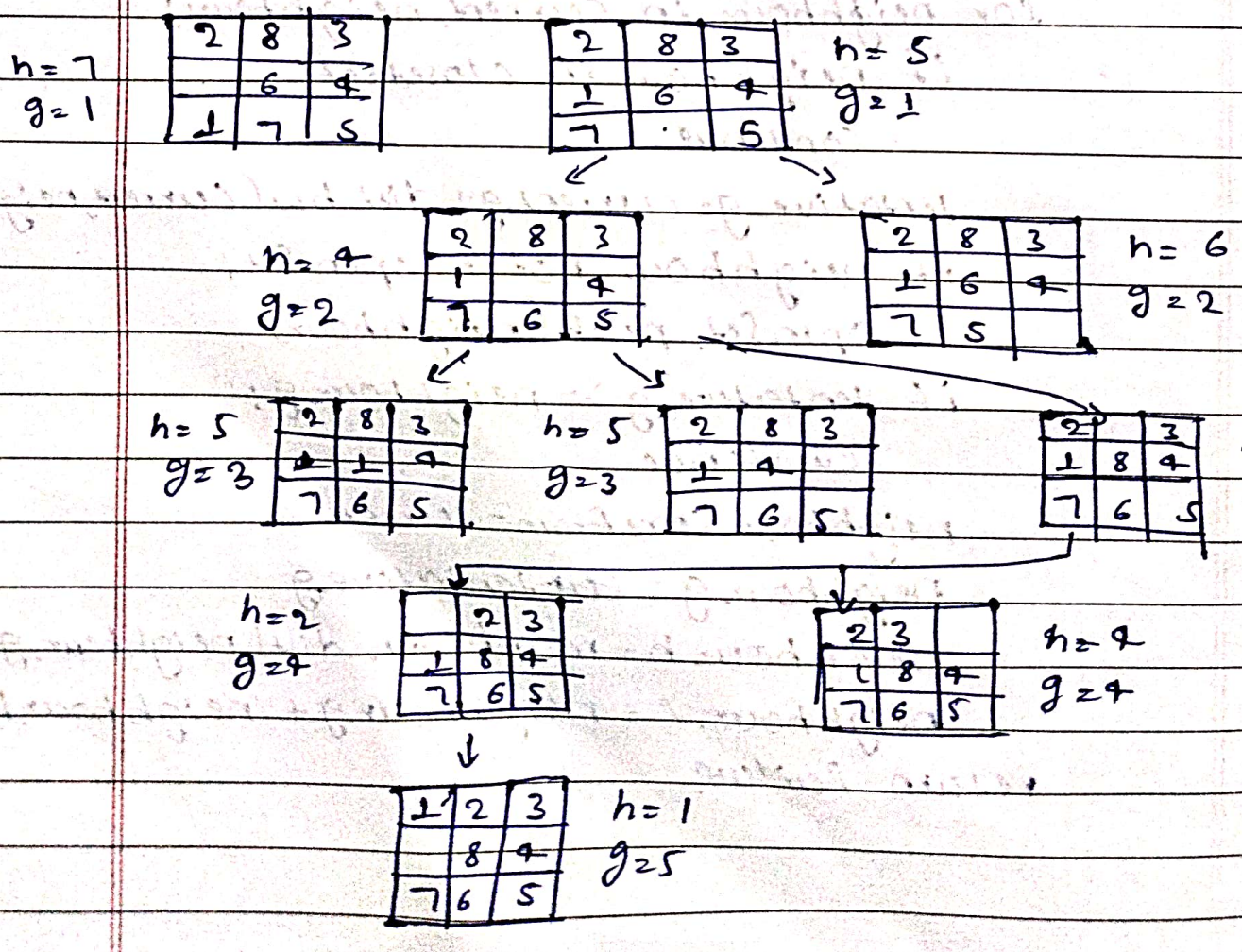
function manhatten_dist (node, goal)
return abs(node.x - goal.x) + abs(node.y - goal.y)

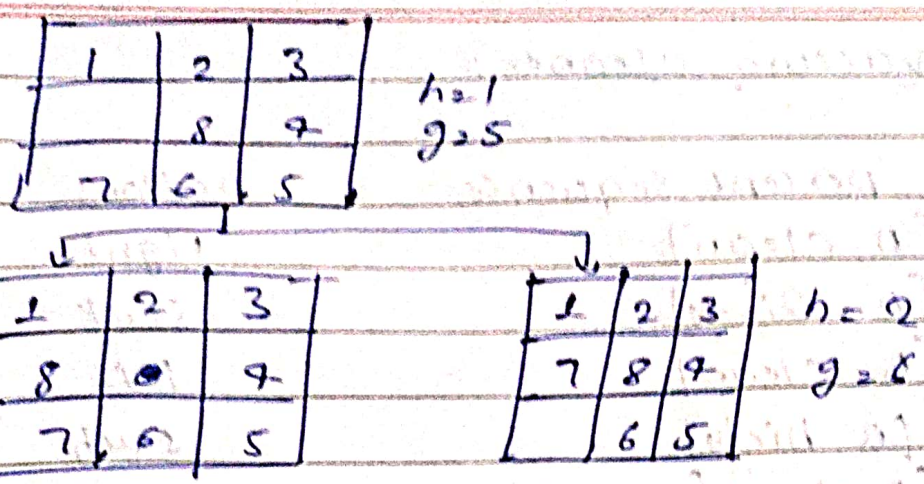
function reconstruct_path(node)
total_path = [node]
while node.comefrom is not None:
node = node.comefrom
total_path.prepend(node)
return total_path.

State space tree



h(n) = 6





Goal