Date: 08/10/24

① Solve- 8 puzzle problem.

Pseudocode:

```
class Node:
    function _init_(state, parent, action, path-cost=0):
        set self.state = state
        set self.parent= parent
        set self.action= action
        set self.path-cost= path-cost
    function expand():
        create children
        set row, col= find-blank()
        create possible-actions
        if row>0 then add 'up' to possible-actions
        if row<2 then add 'dooon' to possible-actions
        if col>0 then add 'left' to possible-actions
        if col<2 then add 'right' to possible-actions
        for action in possible-actions:
            create new-state as a copy of self.state
            if action=='up' then swap new-state [row]
            [col] with new-state[row][col]
            else if action=='down' then swap
            new-state[row][col] with new-state[row+1][col]
            else if action=='left' then swap new-state
            [row][col] with new-state[row][col-1]
            else if action=='right' then swap
            new-state[row][col] with new-state[row]
            [col+1]

        append new Node(new-state, self, action, self.path-cost+1)
        to children
        return children.
```

```
function find- blank();
  from row from 0 to 2:
    for col from 0 to 2:
      if self. state [row] [col]==0 then
    return row, col

function depth-first_search( initial- state, goal-state):
set frontier = [Node (initial- state)]
  set explored= empty set
  while frontier is not empty:
    set node= frontier. pop()
    if node. state== goal- state then
    return node
  add tuple of node state to explored
  for child in node expand()%
    if tuple of child state not in
    explored then append child to frontier
  return none
  function print- solution(node):
    create path
    while node is not none:
      append (node action, node state)
      to path
      set node= node parent
      reverse path
      for (action, state) in path:
      if action is not none then print
      "action:", action
        print state
        print" "
```

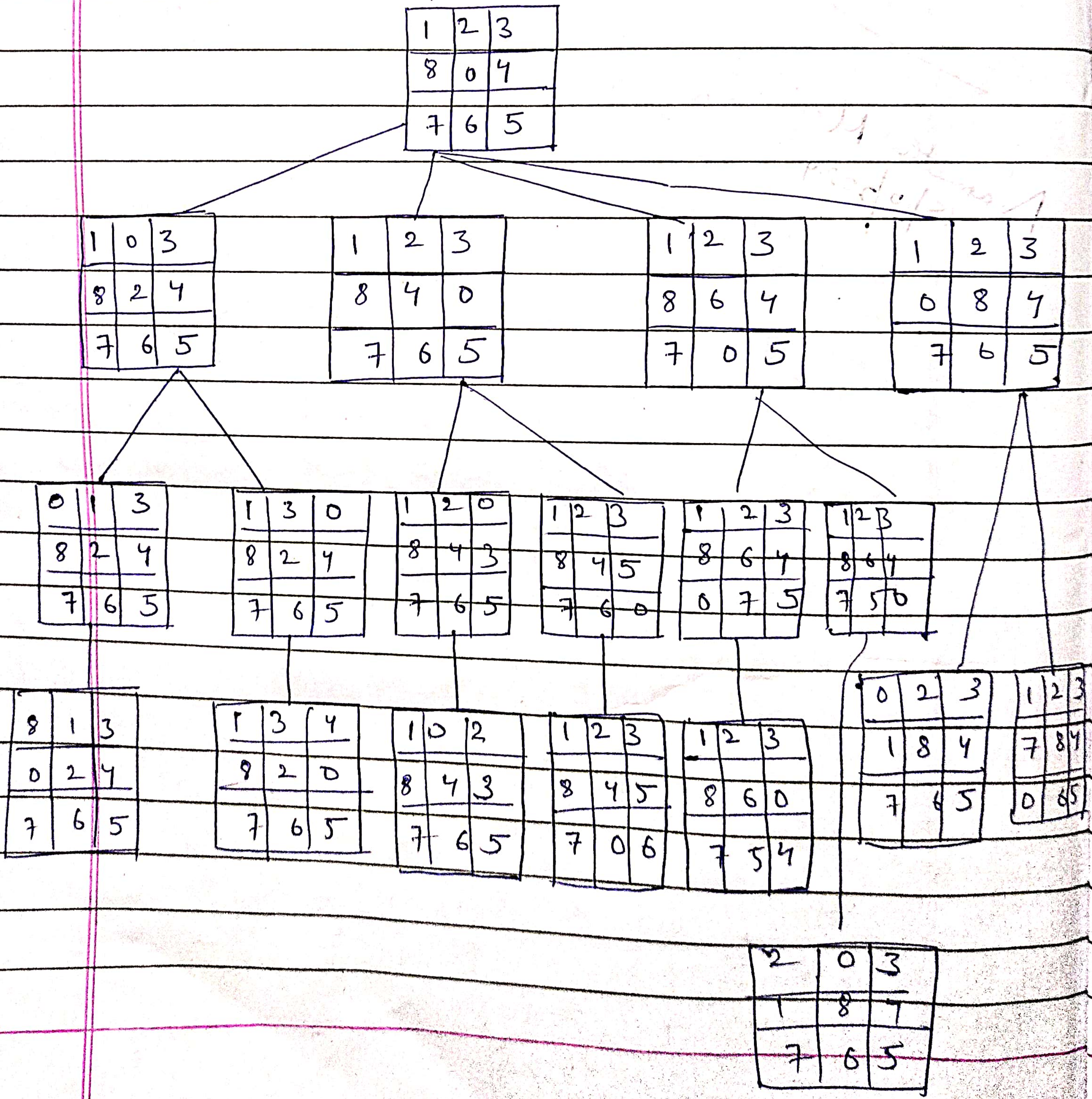
```
set initial- state =[[1,2,3],[0,4,6],[7,5,8]]
set goal- state=[[1,2,3],[4,5,6],[7,8,0]]
set solution = depth-first-search (initial- state, goal- state).
 if solution is not none then
     print" solution found:"
     call print -solution( solution)
 else
     print " solution not found".
```

② Implement iterative deepening search algorithm.

→

```
function iterative-deepening- search(initial- state, goal-
state , max- depth):
    for depth from 0 to max- depth:
        set result= depth- limited- search(initial-state, goal-
                                              state, depth)
        if result is not none then
          return result
      return none
function depth -limited- search (node, goal -state, limit):
    if node. state = = goal- state then
        return node
    if node. depth >= limit then
        retur none
    for each child in expand (node):
        set result = depth- limited- search( child, goal-state
                                                limit)
        if result is not none then
            return result
        return none
 set initial -state, goal-state, max-depth
 set solution= iterative- deepening -search(initial- state, goal-
                state, max -depth)
 if solution is not none then print solution
 else print " No solution found".
```

(#). State space tree (8-puzzle problem):


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