Search process

i) A

ii) A→B→C (Depth: 0)

iii) A→B→D→E→C (Depth 1)

iv) A→B→D→E→C→F→G (Depth 2)

path: A→C→G

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~~ set row, col = find-blank()
        create possible_actions
        if row > 0 then add 'up' to possible_actions
        if row < 2 then add 'down' 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-1][col]
        else if action == 'down' then swap new-state[row][col] with new-state[row+1][col]
        else if action == '~~down~~ 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 newNode(new-state, self, action,
        self.path-cost+1) to children
        return children
function find-blank():
        for 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 point-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(initi-
                        al-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
        return none
    for each child in expand (node);
        set result = depth-limited_search
                ( child-state, goal-state, limi
            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".
```