16/11/26

8 puzzle BFS

Aim - To implement 8 puzzle game using BFS search algorithm.

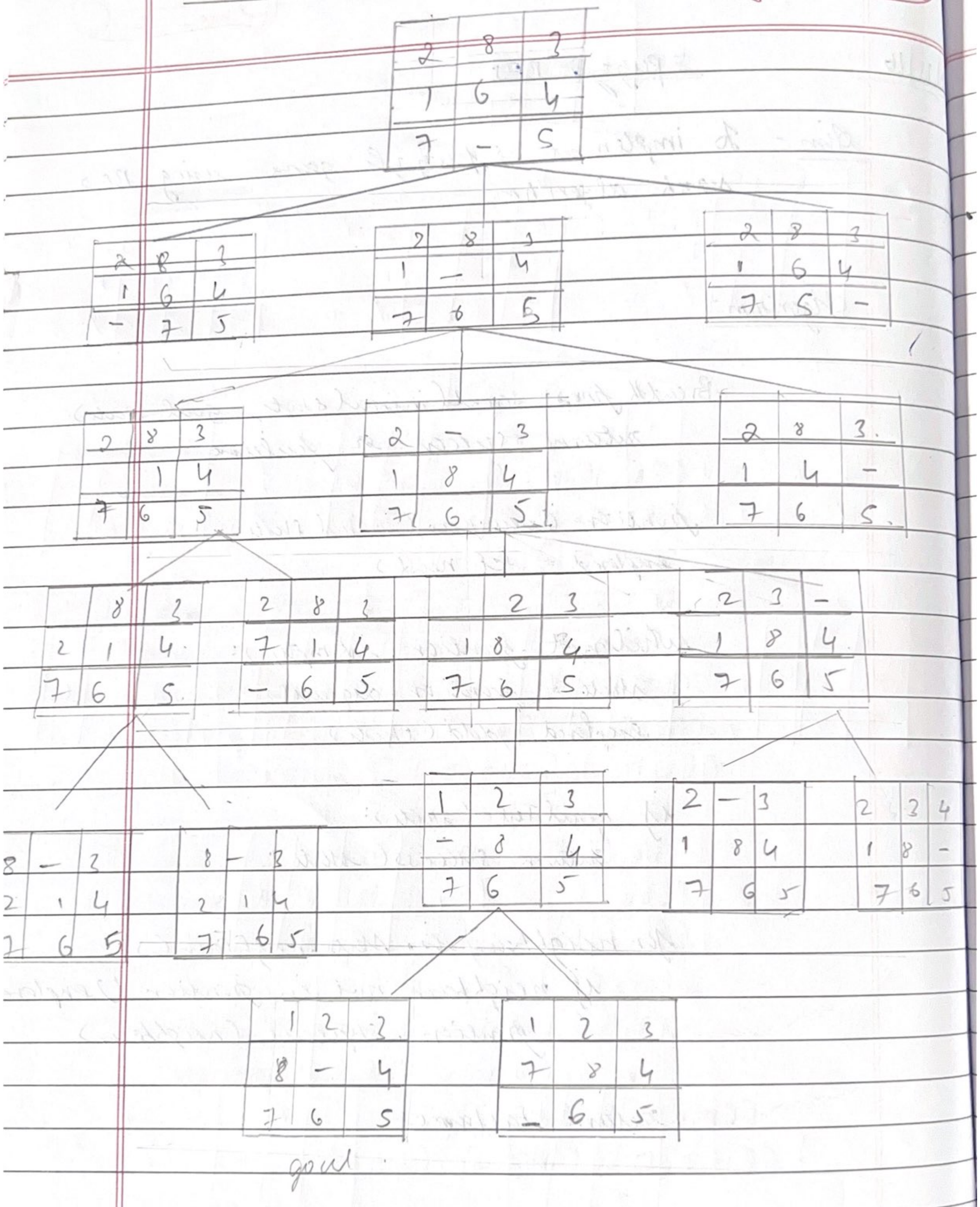
Algorithm

```
Breadth first search(initial state, goal state).
    returns success or failure

frontier = Queue.new(initial state)
  explored = Set.new()

while not frontier.isEmpty():
    state = frontier.dequeue()
    explored.add(state)

    if goalTest(state):
        return success(state)

    for neighbour in state.neighbours():
        if neighbour not in frontier U explored:
            frontier.enqueue(neighbour)

    return Failure.
```

Code :-

```
import copy

inp = [[1, 2, 3 ],[4, -1, 5] , [6, 7 , 8]]
out = [[1, 2, 3],[4 , 5, 6], [7, 8, -1]]

print ("Enter the input puzzle")
for i in range(3) :
        for j in range(3):
          inp[i][j] = inp (input("Enter number at "+
                                str(i) + "," + str(j) + " -> "))

def move(temp, movement):
    if movement = = "up" :
     for i in range (3):
      for j in range(3):
           if (temp[i][j] = = -1) :
             if i!=0.
                temp[i][j]= temp[i-1] [j]
                  temp [i-1][j] = -1
             return temp

      if movement = = "down"
        for i in range(3) :
           for j in range ( 3):
              if (temp[i][j] = =-1) :
               if i!=2:
                  temp[i][j] = temp [i+1][j]
                   temp [i+1][j] = -1
               return temp .

      if movement == "left"
        for i in rang (3):
          for j in range(3):
            if(temp[i][J] = = -1):

              if J! = 0
                temp[i][J] = temp[i] [J-1].
                temp[i][J-1] = -1
          return temp.

      if movement = = "right"
         for i in range(3):
            for j in range(3):
               if(temp [i][j] = = -1):
                  if j! = 2:
                     temp[i][J] = temp[i][j+1].
                       temp[i][j+1] = -1
               return ltmp.

def bfs():
   global inp
   global out
   path cost = 0
   queue = []
  inpx = [inp, "none"].
  queue. append (inpx)

  while(True):
    puzzle = queue.pop
    pathcost = path cost + 1
    print (str(puzzle[1] + " --> " + str(puzzle[0]))
    if (puzzle[0] = = out):
       print ("Found")
       print ("path cost -> " + str (pathcost -1))
       break
  else:
    if (puzzle[1]! = "down"):
        temp = copy.deepcopy (puzzle[0])
        up = move (temp, "up")
        upx = [up, "upx"].
        queue.insert (0, upx)

    if (puzzle[1] ! = "right"):
        temp = copy.deepcopy (puzzle[0])
        left = move(temp, "left")
        leftx = [left, "left"]
        queue.insert(0, leftx)

    if (puzzle[1]! = "up"):
        temp = copy.deepcopy (puzzle[0])
        down = move (temp, "down")
        downx = [down, "down"].
        queue.insert(0, down x)

    if (puzzle[1]! = "left"):
        temp = copy.deepcopy (puzzle[0])
        right = move (temp, "right")
        rightx = [right, "right")
        queue.insert(0, right).

bfs()
```

State space tree:



BFS O/P:

```
36  Enter number at 0,0
    Enter number at 0,1
    Enter number at 0,2
    Enter number at 1,0
    Enter number at 1,1
    Enter number at 1,2
    Enter number at 2,0
    Enter number at 2,1
    Enter number at 2,2
              — BFS —
    none → [(1,2,3), [4,5,6], [-1,7,8]]
    up → [(1,2,3), [-1,5,6], [4,7,8]].
    left → [(1,2,3), [4,5,6], [-1,7,8]].
    down → [(1,2,3), (4,5,6), [-1,7,8]].
    right → ((1,2,3), (4,5,6), [7,-1,8])
    up → [(-1,2,3], (1,5,6), [4,7,8]].
    left → [(1,2,3), (-1,5,6), (4,7,8]].
    right → [(1,2,3), (5,-1,6), (4,7,8]).
    up → [(1,2,3), [-1,5,6], [4,7,8]].
    left → [(1,2,3), [4,5,6], [-1,7,8])
    down → [(1,2,3), (4,5,6), [-1,7,8]].
    left → [(1,2,3), [4,5,6], [-1,7,8]].
    down → ([1,2,3), (4,5,6), [-1,7,8]].
    right → ([1,2,3), [4,5,6], (7,-1,8]).
    up → [(1,2,3), [4,-1,6],
    down → [(1,2,3), (4,5,6), [7,-1,8]].
    right → ([1,2,3), (4,8,6], [7,8,-1]).
```