Jayati Gumber

Lab assignment 3

Roll no 101803546

Solution 1

```
import sys
import copy

q = []
visited = []

def compare(s,g):
    if s==g:
        return(1)
    else:
        return(0)

def find_pos(s):

    for i in range(3):
        for j in range(3):
            if s[i][j] == 0:
                return([i,j])


def up(s,pos):

    i = pos[0]
    j = pos[1]

    if i > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i-1][j]
        temp[i-1][j] = 0
        return (temp)
    else:
        return (s)


def down(s,pos):

    i = pos[0]
    j = pos[1]

    if i < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i+1][j]
        temp[i+1][j] = 0
        return (temp)
    else:
        return (s)


def right(s,pos):

    i = pos[0]
    j = pos[1]

    if j < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j+1]
        temp[i][j+1] = 0
        return (temp)
    else:
        return (s)


def left(s,pos):

    i = pos[0]
    j = pos[1]

    if j > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j-1]
        temp[i][j-1] = 0
        return (temp)
    else:
        return (s)

def enqueue(s,val):
    global q
    q = q + [(val,s)]

def heuristic(s,g):
    d = 0
    for i in range(3):
        for j in range(3):
            if s[i][j] != g[i][j]:
                d += 1
    return d


def dequeue(g):

    global q
    global visited

    q.sort()
    visited = visited + [q[0][1]]
    elem = q[0][1]
    del q[0]
    return (elem)

def search(s,g):

    curr_state = copy.deepcopy(s)
    if s == g:
        return

    global visited
    while(1):

        pos = find_pos(curr_state)
        new = up(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found!! The intermediate states are:")
                print (visited + [g])
                return
            else:
                if new not in visited:
                    enqueue(new,heuristic(new,g))


        new = down(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found!! The intermediate states are:")
                print (visited + [g])
                return
            else:
                if new not in visited:
                    enqueue(new,heuristic(new,g))

        new = right(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found!! The intermediate states are:")
                print (visited + [g])
                return
            else:
                if new not in visited:
                    enqueue(new,heuristic(new,g))

        new = left(curr_state,pos)

        if new != curr_state:
            if new == g:
                print ("found!! The intermediate states are:")
                print (visited + [g])
                return
            else:
                if new not in visited:
                    enqueue(new,heuristic(new,g))

        if len(q) > 0:
            curr_state = dequeue(g)
        else:
            print ("not found")
            return



def main():
    s = [[2,0,3],[1,8,4],[7,6,5]]
    g = [[1,2,3],[8,0,4],[7,6,5]]
    global q
    global visited
    q = q
    visited = visited + [s]

    search(s,g)

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

```
found!! The intermediate states are:
[[[2, 0, 3], [1, 8, 4], [7, 6, 5]], [[0, 2, 3], [1, 8, 4], [7, 6, 5]], [[1, 2, 3], [6
```

Solution 2

```
import sys
import copy
open = []
path = []

def find_pos(s):
    for i in range(3):
        for j in range(3):
            if s[i][j] == 0:
                return ([i, j])

def up(s, pos):
    i = pos[0]
    j = pos[1]

    if i > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i - 1][j]
        temp[i - 1][j] = 0
        return (temp)
    else:
        return (s)
```

```
def down(s, pos):
    i = pos[0]
    j = pos[1]

    if i < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i + 1][j]
        temp[i + 1][j] = 0
        return (temp)
    else:
        return (s)

def right(s, pos):
    i = pos[0]
    j = pos[1]

    if j < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j + 1]
        temp[i][j + 1] = 0
        return (temp)
    else:
        return (s)

def left(s, pos):
    i = pos[0]
    j = pos[1]

    if j > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j - 1]
        temp[i][j - 1] = 0
        return (temp)
    else:
        return (s)

def enqueue(s, val):
    global open
    open = open + [(val, s)]

def heuristic(s, g):
    d = 0
    for i in range(3):
        for j in range(3):
            if s[i][j] != g[i][j]:
                d += 1
    return d

def dequeue(g):
    global open
    global path
    open.sort()
    path = path + [open[0][1]]
    best_state = open[0][1]
    i = 0
    while(i < len(open)):
        del open[i]
    return (best_state)

def MoveGen(s, g):
    curr_state = copy.deepcopy(s)
    if s == g:
        return

    global path

    while (1):

        pos = find_pos(curr_state)

        new = up(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g))

        new = down(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g))

        new = right(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g))

        new = left(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g))

        if len(open) > 0:
            curr_state = dequeue(g)
        else:
            print("not found")
            return

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

path = path + [initial_state]
MoveGen(initial_state, goal_state)

print(f"Number of moves is {len(path)}")
print(len(open))
```

```
not found
Number of moves is 31
0
```

Solution 3

```
import sys
import copy
open = []
path = []

def find_pos(s):
    for i in range(3):
        for j in range(3):
            if s[i][j] == 0:
                return ([i, j])

def up(s, pos):
    i = pos[0]
    j = pos[1]

    if i > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i - 1][j]
        temp[i - 1][j] = 0
        return (temp)
    else:
        return (s)

def down(s, pos):
    i = pos[0]
    j = pos[1]

    if i < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i + 1][j]
        temp[i + 1][j] = 0
        return (temp)
    else:
        return (s)

def right(s, pos):
    i = pos[0]
    j = pos[1]

    if j < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j + 1]
        temp[i][j + 1] = 0
        return (temp)
    else:
        return (s)

def left(s, pos):
    i = pos[0]
    j = pos[1]

    if j > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j - 1]
        temp[i][j - 1] = 0
        return (temp)
    else:
        return (s)

def enqueue(s, val):
    global open
    open = open + [(val, s)]

def heuristic(s, g, d):
    h = 0
    for i in range(3):
        for j in range(3):
            if s[i][j] != g[i][j]:
                h += 1
    f = h + d
    return f

def dequeue(g):
    global open
    global path
    open.sort()
    path = path + [open[0][1]]
    best_state = open[0][1]
    del open[0]
    return (best_state)

def MoveGen(s, g):
    curr_state = copy.deepcopy(s)
    if s == g:
        return
    dist_initial = 1
    global path

    while (1):

        pos = find_pos(curr_state)

        new = up(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g))

        new = down(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g, dist_initial))

        new = right(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g, dist_initial))

        new = left(curr_state, pos)

        if new != curr_state:
            if new == g:
                print("found!! The intermediate states are:")
                print(path + [g])
                return
            else:
                if new not in path:
                    enqueue(new, heuristic(new, g, dist_initial))

        if len(open) > 0:
            curr_state = dequeue(g)
        else:
            print("not found")
            return
        dist_initial += 1

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

path = path + [initial_state]
MoveGen(initial_state, goal_state)

print(f"Number of moves is {len(path)}")
```

```
found!! The intermediate states are:
[[[2, 0, 3], [1, 8, 4], [7, 6, 5]], [[0, 2, 3], [1, 8, 4], [7, 6, 5]], [[1, 2, 3], [
Number of moves is 3
```