HW #2
CS 461-2
06/04/2016

Cagdas OZTEKIN
Fatih KARAOGLANOGLU
Baraa ORABI

Table of the successful solvings:

ω	Successful solvings out of 1000
1	84
2	968
3	1000
4	999

Sample solution tracing:

The puzzle is represented by a 9 element array. So for example, [0,1,2,3,4,5,6,7,8,-1] refers to the following puzzle setup (which is also the winning setup):

1	2	3
4	5	6
7	8	

Now the following listing, shows the listing of the program as it is solving the puzzle [4, 3, 0, -1, 2, 5, 7, 6, 1]. Each line shows the nodes that the program kept after keeping only the best $\omega = 3$ nodes. The highlighted puzzle setups are form the path from the winning position all the way back to the initial setup.

[4, 3, 0, -1, 2, 5, 7, 6, 1]
[[-1, 3, 0, 4, 2, 5, 7, 6, 1], [4, 3, 0, 7, 2, 5, -1, 6, 1], [4, 3, 0, 2, -1, 5, 7, 6, 1]]
[[4, 3, 0, 7, 2, 5, 6, -1, 1], [3, -1, 0, 4, 2, 5, 7, 6, 1], [4, -1, 0, 2, 3, 5, 7, 6, 1]]
[[4, 3, 0, 7, -1, 5, 6, 2, 1], [4, 3, 0, 7, 2, 5, 6, 1, -1], [3, 2, 0, 4, -1, 5, 7, 6, 1]]
[[4, -1, 0, 7, 3, 5, 6, 2, 1], [4, 3, 0, -1, 7, 5, 6, 2, 1], [3, 2, 0, -1, 4, 5, 7, 6, 1]]
[[-1, 2, 0, 3, 4, 5, 7, 6, 1], [-1, 4, 0, 7, 3, 5, 6, 2, 1], [4, 0, -1, 7, 3, 5, 6, 2, 1]]
[[2, -1, 0, 3, 4, 5, 7, 6, 1], [7, 4, 0, -1, 3, 5, 6, 2, 1], [4, 0, 5, 7, 3, -1, 6, 2, 1]]
[[2, 0, -1, 3, 4, 5, 7, 6, 1], [7, 4, 0, 3, -1, 5, 6, 2, 1], [2, 4, 0, 3, -1, 5, 7, 6, 1]]
[[7, -1, 0, 3, 4, 5, 6, 2, 1], [7, 4, 0, 3, 2, 5, 6, -1, 1], [2, 0, 5, 3, 4, -1, 7, 6, 1]]
[[-1, 7, 0, 3, 4, 5, 6, 2, 1], [7, 0, -1, 3, 4, 5, 6, 2, 1], [7, 4, 0, 3, 2, 5, 6, 1, -1]]
[[3, 7, 0, -1, 4, 5, 6, 2, 1], [7, 0, 5, 3, 4, -1, 6, 2, 1], [7, 4, 0, 3, 2, -1, 6, 1, 5]]
[[7, 0, 5, 3, 4, 1, 6, 2, -1], [3, 7, 0, 6, 4, 5, -1, 2, 1], [3, 7, 0, 4, -1, 5, 6, 2, 1]]
[[7, 0, 5, 3, 4, 1, 6, -1, 2], [3, 7, 0, 6, 4, 5, 2, -1, 1], [3, -1, 0, 4, 7, 5, 6, 2, 1]]
[[7, 0, 5, 3, -1, 1, 6, 4, 2], [7, 0, 5, 3, 4, 1, -1, 6, 2], [3, 7, 0, 6, 4, 5, 2, 1, -1]]
[[7, -1, 5, 3, 0, 1, 6, 4, 2], [7, 0, 5, 3, 1, -1, 6, 4, 2], [7, 0, 5, -1, 3, 1, 6, 4, 2]]
[[7, 0, -1, 3, 1, 5, 6, 4, 2], [-1, 7, 5, 3, 0, 1, 6, 4, 2], [7, 5, -1, 3, 0, 1, 6, 4, 2]]
[[7, -1, 0, 3, 1, 5, 6, 4, 2], [7, 5, 1, 3, 0, -1, 6, 4, 2], [3, 7, 5, -1, 0, 1, 6, 4, 2]]
[[7, 1, 0, 3, -1, 5, 6, 4, 2], [-1, 7, 0, 3, 1, 5, 6, 4, 2], [7, 5, 1, 3, 0, 2, 6, 4, -1]]
[[7, 1, 0, 3, 4, 5, 6, -1, 2], [7, 1, 0, -1, 3, 5, 6, 4, 2], [7, 1, 0, 3, 5, -1, 6, 4, 2]]
[[7, 1, 0, 3, 4, 5, 6, 2, -1], [7, 1, 0, 3, 4, 5, -1, 6, 2], [-1, 1, 0, 7, 3, 5, 6, 4, 2]]
[[7, 1, 0, 3, 4, -1, 6, 2, 5], [7, 1, 0, -1, 4, 5, 3, 6, 2], [1, -1, 0, 7, 3, 5, 6, 4, 2]]
[[7, 1, -1, 3, 4, 0, 6, 2, 5], [7, 1, 0, 3, -1, 4, 6, 2, 5], [-1, 1, 0, 7, 4, 5, 3, 6, 2]]
[[7, -1, 1, 3, 4, 0, 6, 2, 5], [7, 1, 0, 3, 2, 4, 6, -1, 5], [7, -1, 0, 3, 1, 4, 6, 2, 5]]
[[-1, 7, 1, 3, 4, 0, 6, 2, 5], [7, 1, 0, 3, 2, 4, 6, 5, -1], [7, 4, 1, 3, -1, 0, 6, 2, 5]]
[[7, 1, 0, 3, 2, -1, 6, 5, 4], [3, 7, 1, -1, 4, 0, 6, 2, 5], [7, 4, 1, 3, 2, 0, 6, -1, 5]]
[[7, 1, -1, 3, 2, 0, 6, 5, 4], [7, 1, 0, 3, -1, 2, 6, 5, 4], [7, 4, 1, 3, 2, 0, 6, 5, -1]]
[[7, 1, 0, 3, 5, 2, 6, -1, 4], [7, -1, 1, 3, 2, 0, 6, 5, 4], [7, -1, 0, 3, 1, 2, 6, 5, 4]]

Cagdas OZTEKIN
Fatih KARAOGLANOGLU
Baraa ORABI

[[7, 1, 0, 3, 5, 2, 6, 4, -1], [7, 1, 0, 3, 5, 2, -1, 6, 4], [7, 2, 1, 3, -1, 0, 6, 5, 4]]
[[7, 2, 1, 3, 5, 0, 6, -1, 4], [7, 2, 1, 3, 0, -1, 6, 5, 4], [7, 1, 0, -1, 5, 2, 3, 6, 4]]
[[7, 2, 1, 3, 5, 0, 6, 4, -1], [7, 2, -1, 3, 0, 1, 6, 5, 4], [7, 2, 1, 3, 0, 4, 6, 5, -1]]
[[7, -1, 2, 3, 0, 1, 6, 5, 4], [7, 2, 1, 3, 5, -1, 6, 4, 0], [7, 2, 1, 3, 0, 4, 6, -1, 5]]
[[7, 0, 2, 3, -1, 1, 6, 5, 4], [-1, 7, 2, 3, 0, 1, 6, 5, 4], [7, 2, 1, 3, -1, 5, 6, 4, 0]]
[[7, 2, 1, 3, 4, 5, 6, -1, 0], [7, 0, 2, 3, 5, 1, 6, -1, 4], [7, 0, 2, 3, 1, -1, 6, 5, 4]]
[[7, 2, 1, 3, 4, 5, 6, 0, -1], [7, 2, 1, 3, 4, 5, -1, 6, 0], [7, 0, 2, 3, 5, 1, 6, 4, -1]]
[[7, 2, 1, 3, 4, -1, 6, 0, 5], [7, 0, 2, 3, 5, -1, 6, 4, 1], [7, 2, 1, -1, 4, 5, 3, 6, 0]]
[[7, 0, 2, 3, -1, 5, 6, 4, 1], [7, 2, -1, 3, 4, 1, 6, 0, 5], [7, 2, 1, 3, -1, 4, 6, 0, 5]]
[[7, 0, 2, 3, 4, 5, 6, -1, 1], [7, -1, 2, 3, 0, 5, 6, 4, 1], [7, -1, 2, 3, 4, 1, 6, 0, 5]]
[[7, 0, 2, 3, 4, 5, 6, 1, -1], [7, 0, 2, 3, 4, 5, -1, 6, 1], [-1, 7, 2, 3, 0, 5, 6, 4, 1]]
[[7, 0, 2, 3, 4, -1, 6, 1, 5], [7, 0, 2, -1, 4, 5, 3, 6, 1], [3, 7, 2, -1, 0, 5, 6, 4, 1]]
[[7, 0, -1, 3, 4, 2, 6, 1, 5], [7, 0, 2, 3, -1, 4, 6, 1, 5], [-1, 0, 2, 7, 4, 5, 3, 6, 1]]
[[0, -1, 2, 7, 4, 5, 3, 6, 1], [7, -1, 0, 3, 4, 2, 6, 1, 5], [7, -1, 2, 3, 0, 4, 6, 1, 5]]
[[0, 4, 2, 7, -1, 5, 3, 6, 1], [0, 2, -1, 7, 4, 5, 3, 6, 1], [-1, 7, 0, 3, 4, 2, 6, 1, 5]]
[[0, 4, 2, 7, 6, 5, 3, -1, 1], [0, 4, 2, -1, 7, 5, 3, 6, 1], [0, 4, 2, 7, 5, -1, 3, 6, 1]]
[[0, 4, 2, 3, 7, 5, -1, 6, 1], [0, 4, 2, 7, 6, 5, -1, 3, 1], [0, 4, 2, 7, 6, 5, 3, 1, -1]]
[[0, 4, 2, 3, 7, 5, 6, -1, 1], [0, 4, 2, -1, 6, 5, 7, 3, 1], [0, 4, 2, 7, 6, -1, 3, 1, 5]]
[[0, 4, 2, 3, -1, 5, 6, 7, 1], [0, 4, 2, 3, 7, 5, 6, 1, -1], [0, 4, 2, 6, -1, 5, 7, 3, 1]]
[[0, -1, 2, 3, 4, 5, 6, 7, 1], [0, 4, 2, -1, 3, 5, 6, 7, 1], [0, 4, 2, 3, 5, -1, 6, 7, 1]]
[[-1, 0, 2, 3, 4, 5, 6, 7, 1], [0, 2, -1, 3, 4, 5, 6, 7, 1], [0, 4, 2, 3, 5, 1, 6, 7, -1]]
[[3, 0, 2, -1, 4, 5, 6, 7, 1], [0, 2, 5, 3, 4, -1, 6, 7, 1], [0, 4, 2, 3, 5, 1, 6, -1, 7]]
[[0, 2, 5, 3, 4, 1, 6, 7, -1], [3, 0, 2, 6, 4, 5, -1, 7, 1], [3, 0, 2, 4, -1, 5, 6, 7, 1]]
[[0, 2, 5, 3, 4, 1, 6, -1, 7], [3, -1, 2, 4, 0, 5, 6, 7, 1], [3, 0, 2, 6, 4, 5, 7, -1, 1]]
[[-1, 3, 2, 4, 0, 5, 6, 7, 1], [0, 2, 5, 3, -1, 1, 6, 4, 7], [0, 2, 5, 3, 4, 1, -1, 6, 7]]
[[4, 3, 2, -1, 0, 5, 6, 7, 1], [0, -1, 5, 3, 2, 1, 6, 4, 7], [0, 2, 5, 3, 1, -1, 6, 4, 7]]
[[4, 3, 2, 0, -1, 5, 6, 7, 1], [0, 2, -1, 3, 1, 5, 6, 4, 7], [4, 3, 2, 6, 0, 5, -1, 7, 1]]
[[0, -1, 2, 3, 1, 5, 6, 4, 7], [4, -1, 2, 0, 3, 5, 6, 7, 1], [4, 3, 2, 0, 7, 5, 6, -1, 1]]
[[0, 1, 2, 3, -1, 5, 6, 4, 7], [-1, 0, 2, 3, 1, 5, 6, 4, 7], [-1, 4, 2, 0, 3, 5, 6, 7, 1]]
[[0, 1, 2, 3, 4, 5, 6, -1, 7], [0, 1, 2, -1, 3, 5, 6, 4, 7], [0, 1, 2, 3, 5, -1, 6, 4, 7]]
[[0, 1, 2, 3, 4, 5, 6, 7, -1], [0, 1, 2, 3, 4, 5, -1, 6, 7], [0, 1, 2, 3, 5, 7, 6, 4, -1]]

```python
from random import shuffle

#Heuristic asked in HW2, Counts number of puzzle parts that are not in their place
def SetDif(puzzle):
        count = 0
        for i in xrange(len(puzzle)):
                if(puzzle[i] is not -1):
                        if(puzzle[i] is not i):
                                count = count +1
        return count
#A better heuristic which sums up total manhattan distance of all parts to their actual place
def Manhattan(puzzle):
        distance = 0
        for i in range(len(puzzle)):
                if(puzzle[i] is not -1):
                        #print puzzle[i] , "------"
                        distance = distance + abs(puzzle[i] % 3 - i % 3) + abs(puzzle[i] / 3 - i /3)

        return distance

#Generates Puzzles first creates alist -1 to 7 then shuffles it if it is solveable return else try
again
def PuzzleGenerator():
        lis = [-1,0,1,2,3,4,5,6,7]#Repsresents value+1
        shuffle(lis)
        #print lis

        okashina = 0
        for i in range(len(lis)):
                if lis[i] is not -1:
                        for j in range(i,len(lis)):
                                if lis[j] is not -1:
                                        if(lis[j] < lis[i]):
                                                okashina = okashina + 1
        if( okashina % 2 is 1):#Not Solvable
                return PuzzleGenerator()
        else:#Solvable so return
                return lis

#A function that converts comparator to Key object
def cmp_to_key(mycmp):
    'Convert a cmp= function into a key= function'
    class K(object):
        def __init__(self, obj, *args):
            self.obj = obj
        def __lt__(self, other):
            return mycmp(self.obj, other.obj) < 0
        def __gt__(self, other):
            return mycmp(self.obj, other.obj) > 0
        def __eq__(self, other):
            return mycmp(self.obj, other.obj) == 0
        def __le__(self, other):
            return mycmp(self.obj, other.obj) <= 0
        def __ge__(self, other):
            return mycmp(self.obj, other.obj) >= 0
        def __ne__(self, other):
            return mycmp(self.obj, other.obj) != 0
    return K

#A comparator that uses Manhattan Heuristic
def CmpPuzzle2(p1,p2):#My Hurustic works better
        return Manhattan(p1) - Manhattan(p2)

#A comparator that uses SetDif Heuristic
def CmpPuzzle(p1,p2):#Bad Hurustic of HW
        return SetDif(p1) - SetDif(p2)

#Generates A list of Possible Moves From a game state
def PossibleMoves(puzzle):
        al = []
        empty = puzzle.index(-1);
```

```
        row = empty /3
        col = empty %3
        #print (str(row)+       "-"+str(col))
        tmp = list(puzzle)
        if row  > 0:
                tmp[empty] = tmp[(row -1)*3+col]
                tmp[(row -1)*3+col] = -1
                al.append(list(tmp))
        tmp = list(puzzle)
        if row  < 2:
                tmp[empty] = tmp[(row +1)*3+col]
                tmp[(row +1)*3+col] = -1
                al.append(list(tmp))
        tmp = list(puzzle)
        if col > 0:
                tmp[empty] = tmp[row*3+col-1]
                tmp[row*3+col-1] = -1
                al.append(list(tmp))
        tmp = list(puzzle)
        if col < 2:
                tmp[empty] = tmp[row*3+col+1]
                tmp[row*3+col+1] = -1
                al.append(list(tmp))
        return al
#A toString method, which can be changed later on to represent Puzzles
def ToString(puzzle):
        return str(puzzle)

#Beam Search Method
def BeamItUp(puzzle, w):

        l = []#Temp List to keep new puzzles
        ll = []#A list to keep puzzles
        ll.append(puzzle)#begin
        count = 0#counter, Probably not necessary anymore
        dic = {}#A dictionary to keep track of visited states to prevent looping
        dic[ToString(puzzle)] = 1
        try:#A try catch mechanism to detect index errors, This happens when ll is empty, so search failed
                while(Manhattan(ll[0]) is not 0 and count < 1000):#Loop until finish puzzle
                        l = []#empty temp list
                        for i in range(w):
                                if i >= len(ll):#This looks stupid atm, I dont know what I thought, I could change for loop :P
                                        break
                                moves = PossibleMoves(ll[i])#Get Possible moves
                                dic[ToString(ll[i])] = 1#Add current state to dictionary
                                for pz in moves:#Check if new moves are in the dictionary, else add to l
                                        if ToString(pz) not in dic:
                                                l.append(pz)

                                #l.extend()
                        ll = []#Reset the ll, since we expanded all w
                        ll.extend(l)# Append l to ll

                        ll = sorted(ll, key= cmp_to_key(CmpPuzzle))#Sort ll according to chosen Heuristic

                        #print ll
                        ll = ll[0:w]#Remove states except first w
                        print ll
                        count = count + 1#increment counter

        except IndexError:
                #print "Failed: ",puzzle
                return False
        return True

#Main Function
def main():
```

```
        count = 0#Counter to keep track of number of puzzles to solve
        puzzles = {}#Dictionary to be sure that all puzzles are distinct
        while count < 500:#1000 puzzles asked in hw
                l = PuzzleGenerator()#Generate a puzzle
                if ToString(l) not in puzzles:#if not in dictionary
                        puzzles[ToString(l)] = l#Added it to dictionary
                        count = count + 1#increment
        #puzzles = puzzles.items()
        puzzles =[v for k,v in puzzles.items()]#convert dictionary to a list to iterate
        l = [0,0,0,0]
        for p in puzzles:#for each puzzle
                for w in xrange(1,5):#for each omega
                        #print p
                        if BeamItUp(p,w):#BeamSearch
                                l[w-1] = l[w-1] + 1


        print l
def main2():
        l = PuzzleGenerator()
        print l
        BeamItUp(l,3)
main2()#Call Main
```