**INT404(Artificial intelligence)**

**NAME :**  G.Yaswanth Kumar

Registration number : 11806541

Section : k18FG (33)

//TESTING OF THE IMPLEMENTATION OF THE **CODE :**

**PYTHON PROGRAMMING CODE :**

import random

class IDAstar:

def \_\_init\_\_(self, h, neighbours):

self.h = h

self.neighbours = neighbours

self.FOUND = object()

def solve(self, root, is\_goal, max\_cost=None):

self.is\_goal = is\_goal

self.path = [root]

self.is\_in\_path = {root}

self.path\_descrs = []

self.nodes\_evaluated = 0

bound = self.h(root)

while True:

t = self.\_search(0, bound)

if t is self.FOUND: return self.path, self.path\_descrs, bound, self.nodes\_evaluated

if t is None: return None

bound = t

def \_search(self, g, bound):

self.nodes\_evaluated += 1

node = self.path[-1]

f = g + self.h(node)

if f > bound: return f

if self.is\_goal(node): return self.FOUND

m = None # Lower bound on cost.

for cost, n, descr in self.neighbours(node):

if n in self.is\_in\_path: continue

self.path.append(n)

self.is\_in\_path.add(n)

self.path\_descrs.append(descr)

t = self.\_search(g + cost, bound)

if t == self.FOUND: return self.FOUND

if m is None or (t is not None and t < m): m = t

self.path.pop()

self.path\_descrs.pop()

self.is\_in\_path.remove(n)

return m

def slide\_solved\_state(n):

return tuple(i % (n\*n) for i in range(1, n\*n+1))

def slide\_randomize(p, neighbours):

for \_ in range(len(p) \*\* 2):

\_, p, \_ = random.choice(list(neighbours(p)))

return p

def slide\_neighbours(n):

movelist = []

for gap in range(n\*n):

x, y = gap % n, gap // n

moves = []

if x > 0: moves.append(-1) # Move the gap left.

if x < n-1: moves.append(+1) # Move the gap right.

if y > 0: moves.append(-n) # Move the gap up.

if y < n-1: moves.append(+n) # Move the gap down.

movelist.append(moves)

def neighbours(p):

gap = p.index(0)

l = list(p)

for m in movelist[gap]:

l[gap] = l[gap + m]

l[gap + m] = 0

yield (1, tuple(l), (l[gap], m))

l[gap + m] = l[gap]

l[gap] = 0

return neighbours

def slide\_print(p):

n = int(round(len(p) \*\* 0.5))

l = len(str(n\*n))

for i in range(0, len(p), n):

print(" ".join("{:>{}}".format(x, l) for x in p[i:i+n]))

def encode\_cfg(cfg, n):

r = 0

b = n.bit\_length()

for i in range(len(cfg)):

r |= cfg[i] << (b\*i)

return r

def gen\_wd\_table(n):

goal = [[0] \* i + [n] + [0] \* (n - 1 - i) for i in range(n)]

goal[-1][-1] = n - 1

goal = tuple(sum(goal, []))

table = {}

to\_visit = [(goal, 0, n-1)]

while to\_visit:

cfg, cost, e = to\_visit.pop(0)

enccfg = encode\_cfg(cfg, n)

if enccfg in table: continue

table[enccfg] = cost

for d in [-1, 1]:

if 0 <= e + d < n:

for c in range(n):

if cfg[n\*(e+d) + c] > 0:

ncfg = list(cfg)

ncfg[n\*(e+d) + c] -= 1

ncfg[n\*e + c] += 1

to\_visit.append((tuple(ncfg), cost + 1, e+d))

return table

def slide\_wd(n, goal):

wd = gen\_wd\_table(n)

goals = {i : goal.index(i) for i in goal}

b = n.bit\_length()

def h(p):

ht = 0 # Walking distance between rows.

vt = 0 # Walking distance between columns.

d = 0

for i, c in enumerate(p):

if c == 0: continue

g = goals[c]

xi, yi = i % n, i // n

xg, yg = g % n, g // n

ht += 1 << (b\*(n\*yi+yg))

vt += 1 << (b\*(n\*xi+xg))

if yg == yi:

for k in range(i + 1, i - i%n + n): # Until end of row.

if p[k] and goals[p[k]] // n == yi and goals[p[k]] < g:

d += 2

if xg == xi:

for k in range(i + n, n \* n, n): # Until end of column.

if p[k] and goals[p[k]] % n == xi and goals[p[k]] < g:

d += 2

d += wd[ht] + wd[vt]

return d

return h

if \_\_name\_\_ == "\_\_main\_\_":

solved\_state = slide\_solved\_state(4)

neighbours = slide\_neighbours(4)

is\_goal = lambda p: p == solved\_state

tests = [

(5,1,7,3,9,2,11,4,13,6,15,8,0,10,14,12),

(2,5,13,12,1,0,3,15,9,7,14,6,10,11,8,4),

(5,2,4,8,10,0,3,14,13,6,11,12,1,15,9,7),

(11,4,12,2,5,10,3,15,14,1,6,7,0,9,8,13),

(5,8,7,11,1,6,12,2,9,0,13,10,14,3,4,15),

]

slide\_solver =IDAstar (slide\_wd(4, solved\_state), neighbours)

for p in tests:

path, moves, cost, num\_eval = slide\_solver.solve(p, is\_goal, 80)

slide\_print(p)

print(", ".join({-1: "Left", 1: "Right", -4: "Up", 4: "Down"}[move[1]] for move in moves))

print(cost, num\_eval)

**OUTPUT :** we have compiled and run the code in python programming using anaconda navigator , the output is :

