**Implement A star Algorithm for any game search problem.**

The A\* algorithm works by expanding the node with the lowest f-score. The f-score is calculated as the sum of the node's cost and the heuristic value for the node. The heuristic value is an estimate of the distance from the node to the goal. The A\* algorithm always expands the node with the lowest f-score, which ensures that it will eventually find the shortest path to the goal.

In [23]:

g=0

def print\_board(elements):

for i in range(9):

if i%3 == 0:

print()

if elements[i]==-1:

print("\_", end = " ")

else:

print(elements[i], end = " ")

print()

def solvable(start):

inv=0

for i in range(9):

if start[i] <= 1:

continue

for j in range(i+1,9):

if start[j]==-1:

continue

if start[i]>start[j]:

inv+=1

if inv%2==0:

return True

return False

def heuristic(start,goal):

global g

h = 0

for i in range(9):

for j in range(9):

if start[i] == goal[j] and start[i] != -1:

h += (abs(j-i))//3 + (abs(j-i))%3

return h + g

def moveleft(start,position):

start[position],start[position-1]= start[position-1],start[position]

def moveright(start,position):

start[position],start[position+1]= start[position+1],start[position]

def moveup(start,position):

start[position],start[position-3]= start[position-3],start[position]

def movedown(start,position):

start[position],start[position+3]= start[position+3],start[position]

def movetile(start,goal):

emptyat= start.index(-1)

row = emptyat//3

col = emptyat%3

t1,t2,t3,t4 = start[:],start[:],start[:],start[:]

f1,f2,f3,f4 = 100,100,100,100

if col -1 >=0:

moveleft(t1, emptyat)

f1 = heuristic(t1, goal)

if col+1<3:

moveright(t2, emptyat)

f2 = heuristic(t2, goal)

if row + 1 <3:

movedown(t3, emptyat)

f3 = heuristic(t3, goal)

if row-1>=0:

moveup(t4, emptyat)

f4 = heuristic(t4, goal)

min\_heuristic = min(f1, f2,f3,f4)

if f1==min\_heuristic:

moveleft(start, emptyat)

elif f2==min\_heuristic:

moveright(start, emptyat)

elif f3==min\_heuristic:

movedown(start, emptyat)

elif f4 == min\_heuristic:

moveup(start, emptyat)

def solveEight(start,goal):

global g

g+=1

movetile(start,goal)

print\_board(start)

f = heuristic(start,goal)

if f == g:

print("Solved in {} moves".format(f))

return

solveEight(start,goal)

def main():

global g

start = list()

goal = list()

print("Enter the start state:(Enter -1 for empty):")

for i in range(9):

start.append(int(input()))

print("Enter the goal state:(Enter -1 for empty):")

for i in range(9):

goal.append(int(input()))

print\_board(start)

*# To check if solvable*

if solvable(start):

solveEight(start,goal)

print("Solved in {} moves".format(g))

else:

print("Not possible to solve")

if \_\_name\_\_ == '\_\_main\_\_':

main()-

Enter the start state:(Enter -1 for empty):

1

2

3

-1

4

6

7

5

8

Enter the goal state:(Enter -1 for empty):

1

2

3

4

5

6

7

8

-1

1 2 3

\_ 4 6

7 5 8

1 2 3

4 \_ 6

7 5 8

1 2 3

4 5 6

7 \_ 8

1 2 3

4 5 6

7 8 \_

Solved in 3 moves

Solved in 3 moves

The first function, print\_board(), prints the current state of the board. It takes a list of elements as input, and it prints each element followed by a space. If the element is -1, it prints a space instead.

The second function, solvable(), checks if the current state of the board is solvable. It does this by counting the number of inversions in the board. An inversion is a pair of elements where the first element is greater than the second element. If the number of inversions is even, then the board is solvable. Otherwise, the board is not solvable.

The third function, heuristic(), calculates the heuristic value for the current state of the board. The heuristic value is a measure of how close the current state is to the goal state. The heuristic function works by counting the number of misplaced tiles in the board. The lower the heuristic value, the closer the current state is to the goal state.

The fourth function, moveleft(), moves the empty tile to the left. It takes the current state of the board and the index of the empty tile as input. It then swaps the empty tile with the tile to its left.

The fifth function, moveright(), moves the empty tile to the right. It takes the current state of the board and the index of the empty tile as input. It then swaps the empty tile with the tile to its right.

The sixth function, moveup(), moves the empty tile up. It takes the current state of the board and the index of the empty tile as input. It then swaps the empty tile with the tile above it.

The seventh function, movedown(), moves the empty tile down. It takes the current state of the board and the index of the empty tile as input. It then swaps the empty tile with the tile below it.

The eighth function, movetile(), moves the empty tile to the nearest empty space. It takes the current state of the board and the goal state as input. It then calculates the heuristic values for all four possible moves, and it moves the empty tile to the space with the lowest heuristic value.

The ninth function, solveEight(), solves the 8-puzzle. It takes the current state of the board and the goal state as input. It then repeatedly calls the movetile() function until the current state of the board is equal to the goal state.

The tenth function, main(), is the main function of the program. It prompts the user to enter the start state and the goal state. It then calls the solveEight() function to solve the 8-puzzle. If the 8-puzzle is solvable, the program prints the number of moves it took to solve the puzzle. Otherwise, the program prints a message saying that the puzzle is not solvable.