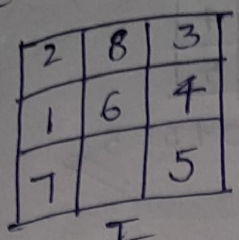
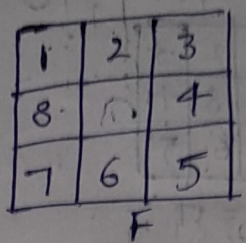
Apply A* Algorithm

Misplaced Tiles

Manhattan Distance



I

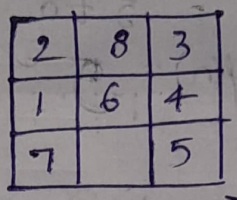


F

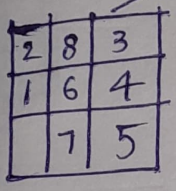
$f(n) = g(n) + h(n)$

= Misplaced Tiles

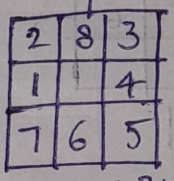
solution



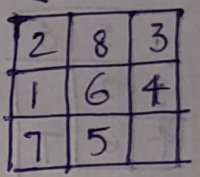
g=0



H.V=6+1=7

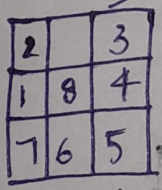


H.V=3+1=4

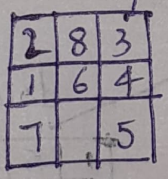


H.V=6+1=7

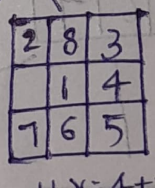
g=1



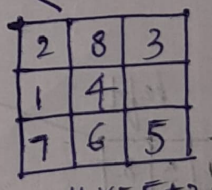
H.V=4+2=6



H.V=5+2=7

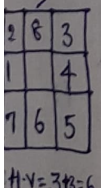


H.V=4+2=6

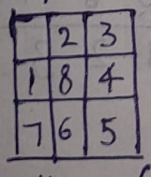


H.V=5+2=7

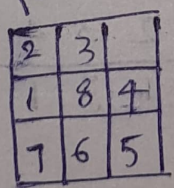
g=2



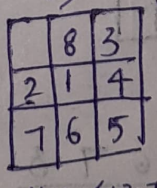
H.V=3+3=6



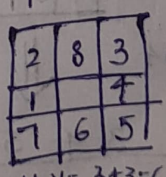
H.V=2+3=6



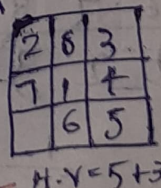
H.V=5+3=8



H.V=4+3=7

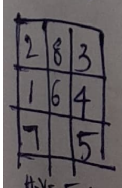


H.V=3+3=6

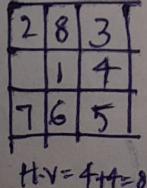


H.V=5+3=8

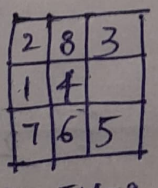
g=3



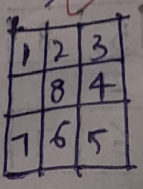
H.V=5+4=9



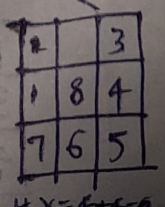
H.V=4+4=8



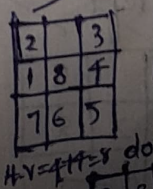
H.V=5+4=9



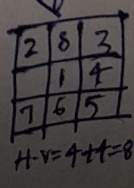
H.V=2+4=6



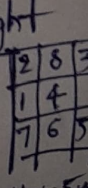
H.V=4+4=8

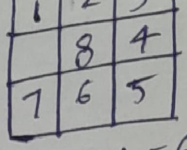


H.V=4+4=8

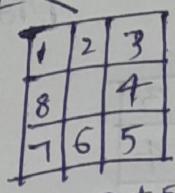


H.V=4+4=8

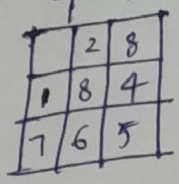




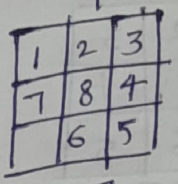
H.V = 2+4 = 6



right



up



down

3+5 = 7

=0+5 = 5

~~H.V = 2+5 = 7~~

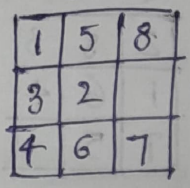
7+5 = 9

$f(n) = g(n) + h(n)$

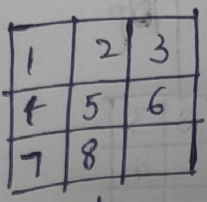
$= 5 + 0$

$f(n) = 5$

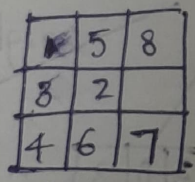
Manhattan distance



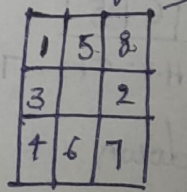
Initial



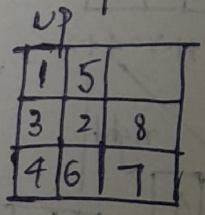
goal



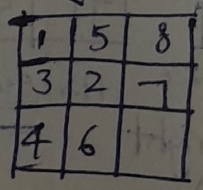
= 0



left



up



down

= 1

1 2 3 4 5 6 7 8

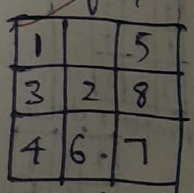
0 2 3 1 1 2 2 3 = 14

1 2 3 4 5 6 7 8

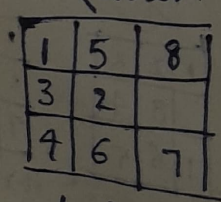
0 1 3 1 1 2 3 3 = 14

1 2 3 4 5 6 7 8

0 1 3 1 1 2 2 2 = 12



left



down

= 2

1 2 3 4 5 6 7 8

0 1 2 1 2 2 2 2 = 13

1 2 3 4 5 6 7 8

Algorithm for (

1. start → put initial state in OPEN ($f = g+h$)
2. pick state with smallest f
3. if goal → stop
4. Expand neighbours (move blank)
5. For each neighbour
   - $g := parent.g + 1$
   - $h :=$ misplaced files
   - $f = g + h$
   - add / update in open
6. Repeat until goal found ~~(or)~~ open empty

output:

2	8	3
1	6	4
7	0	5

⇒ initial

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

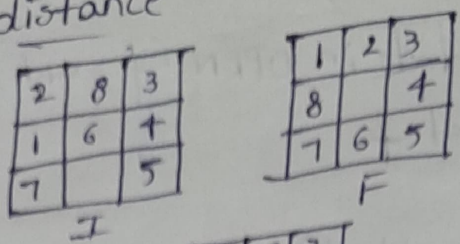
0 2 3
1 8 4
7 6 5

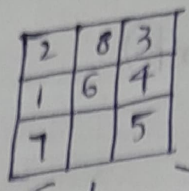
1 2 3
0 8 4
7 6 5

1	2	3
8	0	4
7	6	5

= goal

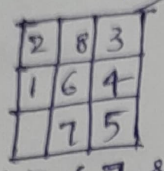
Manhattan distance



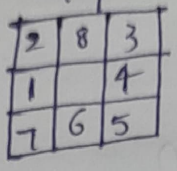


= 0

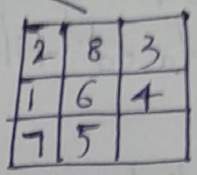
left



up



right



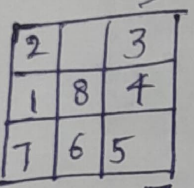
= 1

1 2 3 4 5 6 7 8
1 1 0 0 0 1 1 2
= 6

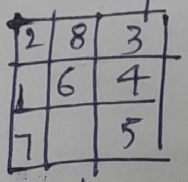
1 2 3 4 5 6 7 8
1 1 0 0 0 0 0 2
= 4

1 2 3 4 5 6 7 8
1 1 0 0 1 1 0 2
= 6

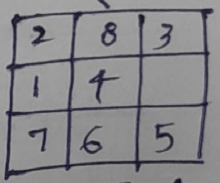
up



down



right



left

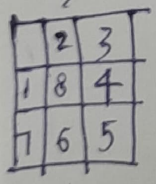
1 2 3 4 5 6 7 8
1 1 0 0 0 0 0 1
= 3

1 2 3 4 5 6 7 8
1 1 0 0 0 1 0 2
= 5

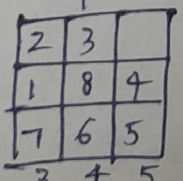
1 2 3 4 5 6 7 8
1 1 0 1 0 0 0 2
= 5

1 2 3 4 5 6 7
2 1 0 0 0 0 0
= 5

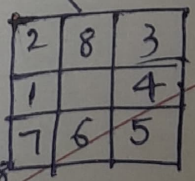
left



right



down



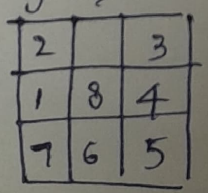
= 3

1 2 3 4 5 6 7 8
1 1 1 0 0 0 0 1
= 4

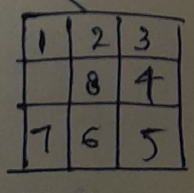
1 2 3 4 5 6 7 8
1 1 0 0 0 0 0 2
= 4

1 2 3 4 5 6 7 8
1 0 0 0 0 0 0 1
= 2

right



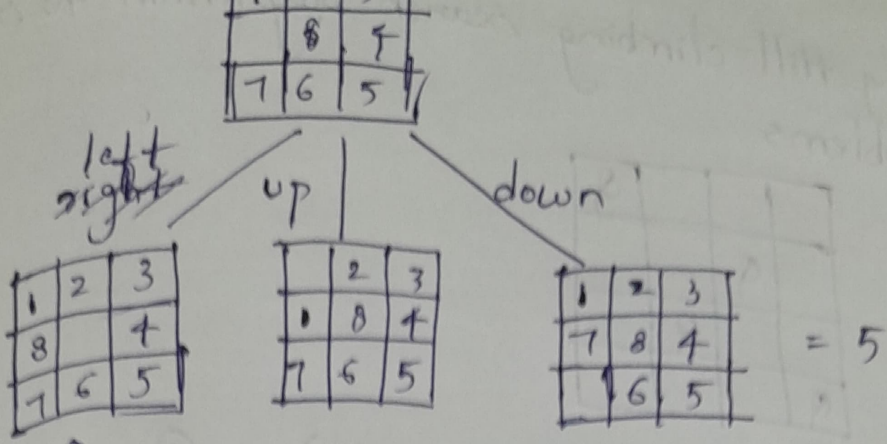
down



= 4

1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 1 = 1

1 2 3 4 5 6 7 8
1 0 0 0 0 0 0 1 = 3



Algorithm

1. Start with the current puzzle state
2. Create an empty list to keep track of tile matches
3. For each tile from 1 to 8:
   - → if the tile is in correct position, add 0 to the list
   - → otherwise, add 1 to the list
4. check the list:
   - → if the list contains all 0's the puzzle is solved - stop
   - → otherwise, make a move to get closer to the goal
5. Repeat steps 2 to 4 until the puzzle is solved

output

2	8	3
1	6	4
7	0	5

= initial

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1	2	3
8	0	4
7	6	5

= goal