

28-11-23

## Program-2 8-Puzzle problem using breadth first search

8 puzzle problem (N puzzle problem / sliding puzzle problem) - Uninformed approach

~~N puzzle~~

In the problem, the square will have  $N+1$  tiles where  
 $N = 8, 15, 24$  and so on.

$N=8$  means the square will have 9 tiles (3 rows & 3 columns)

In this problem, ~~initial~~ initial state / initial configuration (start state) will given and we have to reach the goal state / goal configuration.

Suppose:

Initial state:

1	2	3
	4	6
7	5	8

Goal state:

1	2	3
4	5	6
7	8	

The puzzle can be solved by moving the tiles one by one in the single empty space & thus achieve the goal state.

Rules

- The empty can only move in 4 directions:
  - 1) up
  - 2) down
  - 3) right
  - 4) left
- It cannot move diagonally
- Can take only one step at a time.

0	x	0
x	#	x
0	x	0

Tiles at 0 — no. of possible moves = 2.

Tiles at x — no. of possible moves = 3

Tiles at # — no. of possible moves = 4.

This problem can be solved using breadth first search approach:

↳ Uninformed / non-heuristic search

- This approach explores all nodes (does not use intelligence).

Complexity:  $O(b^d)$  where

$b$  - branching factor

$d$  - depth ~~of~~ factor

Worst case —  $3^{20}$ .

For 8 puzzle problem.

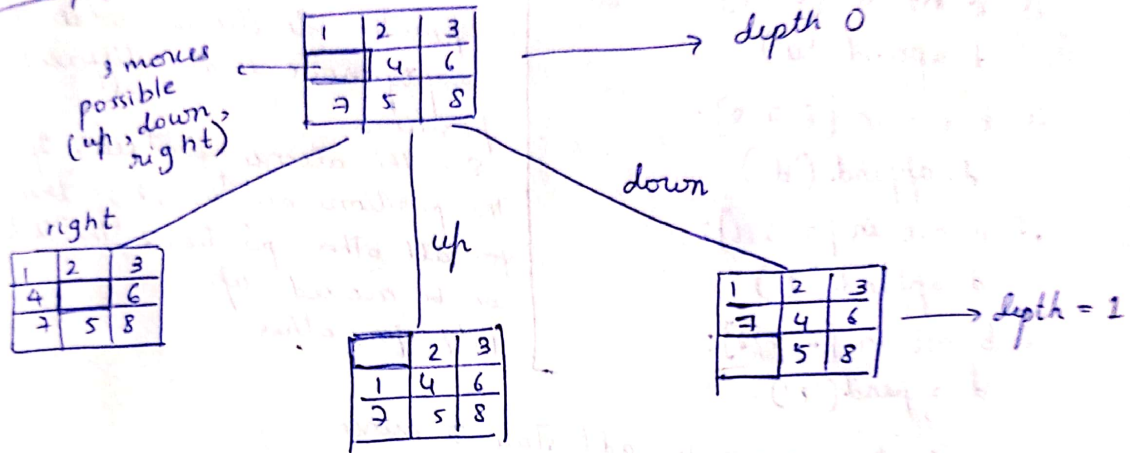
Branching factor  $b = \frac{\text{all possible moves of empty tile at each position}}{\text{no. of tiles.}}$

$$= \frac{24}{9} = 2.67 \sim 3$$

depth factor - initially will be 0.  
As we explore the nodes, the depth factor increases.

Every time, we check the possible moves of the empty tile,  
branch at, ~~poss~~ consider all possible cases and finally  
to the goal state.

For example



||| continue branching these nodes.

Since this is breadth first search, branch for possibilities of 'right' node, then 'up' node, then 'down' node and then only move to branch in the next level.

Code

```
import numpy as np
import pandas as pd
import os
```

```
def bfs(src, target):
```

```
    queue = []
```

```
    queue.append(src)
```

```
    exp = []
```

```
    while len(queue) > 0:
```

```
        source = queue.pop(0)
```

```
        exp.append(source)
```

```
        print(source)
```

```
        if source == target:
```

```
            print("success")
```

```
            return
```

```
        poss_moves_to_do = []
```

```
        poss_moves_to_do = possible_moves(source, exp)
```

```
        for move in poss_moves_to_do:
```

```
            if move not in exp and move not in queue:
```

```
                queue.append(move)
```

Same BFS code



def possible\_moves(state, visited\_states):

# index of empty spot

b = state.index(0)

# directions array

d = []

# add all the possible directions

if b not in [0, 1, 2]:

d.append('u')

if b not in [6, 7, 8]:

d.append('d')

if b not in [0, 3, 6]:

d.append('l')

if b not in [2, 5, 8]:

d.append('r')

# if direction is possible, add state to move

pos\_moves\_it\_can = []

# for all possible directions, find the state if that move is played by calling 'gen' function

for i in d:

# Now for each direction, I want to generate different states of the matrix which I append in pos\_moves\_it\_can

pos\_moves\_it\_can.append(gen(state, i, b))

return [move\_it\_can for move\_it\_can in pos\_moves\_it\_can if move\_it\_can not in visited\_states]

— here, once I get a state which is not in visited\_states, I return it.

def gen(state, m, b):

temp = state.copy()

if m == 'd':

temp[b+3], temp[b] = temp[b], temp[b+3]

elif m == 'u':

temp[b-3], temp[b] = temp[b], temp[b-3]

elif m == 'l':

temp[b-1], temp[b] = temp[b], temp[b-1]

elif m == 'r':

temp[b+1], temp[b] = temp[b], temp[b+1]

return temp

return temp

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

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

bfs(src, target).

0	1	2
3	4	5
6	7	8

This is ~~based on~~ to find the possible directions when the empty node is at different positions.

See the above matrix. If the positions are 0, 1, 2, then for all other positions, the tile can be moved up. Similarly for other.

To move down, I shift 3 indices ~~to~~ front.

If I am at position 4, and I want to move down, I do  $5+3=8$ .

14-11-21

Output:

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

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

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

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

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

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

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

success

01Prun

8	7	6
5	4	3
2	1	0

state 1000

8	7	6
5	4	3
2	1	0

state 1000

0 = 6

p = 4

4 + 0 = 4



1 = 6  
2 = 4  
3 = 2 + 1 = 3

Δ = 6 + 1 = 7  
(total)

1 = 6  
2 = 4  
3 = 2 + 1 = 3



PS C:\Users\neha2\OneDrive\Documents\NehaKamath\_1BM21CS113\_AILab> python

1	2	3
4	5	6
0	7	8

1	2	3
0	5	6
4	7	8

1	2	3
4	5	6
7	0	8

0	2	3
1	5	6
4	7	8

1	2	3
5	0	6
4	7	8

1	2	3
4	0	6
7	5	8

1	2	3
4	5	6
7	8	0