

TUGAS 1
KECERDASAN BUATAN - F

SOLVING **8-PUZZLE &** **8-QUEEN**

Using Breadth-first Search & Depth-first Search

GET TO KNOW US !

ONG MING SEN AI



Arif Nugraha Santosa

5025211048



Laurivasya Gadhing

Syahafidh

5025211136



Rayhan Arvianta

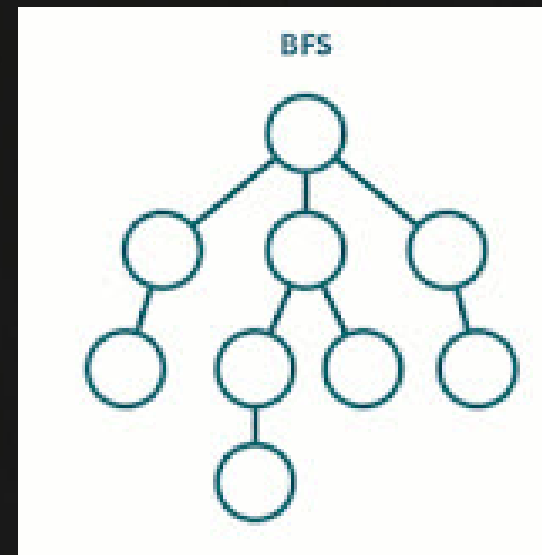
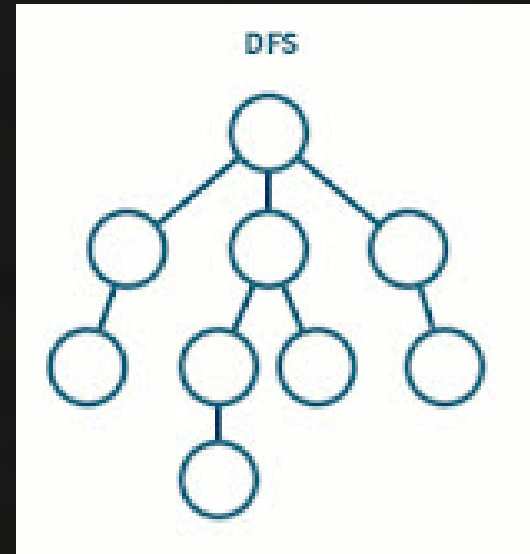
Bayuputra

5025211217

BREADTH-FIRST SEARCH METHOD

untuk menemukan atau melintasi sebuah graph maupun tree dengan arah lintasan secara melebar.

Algoritma Breadth First Search adalah algoritma pencarian melebar yang dilakukan dengan mengunjungi node pada level n terlebih dahulu sebelum mengunjungi node-node pada level $n+1$.



DEPTH-FIRST SEARCH METHOD

untuk menemukan atau melintasi sebuah graph maupun tree dengan arah lintasan secara mendalam.

Algoritma Depth First Search adalah algoritma pencarian mendalam yang dimulai dari node awal dilanjutkan dengan hanya mengunjungi node anak paling kiri pada tingkat selanjutnya.



8-PUZZLE

8-Puzzle merupakan sebuah game dengan kotak berukuran 3x3, dimana disetiap kotaknya berisi sebuah angka acak 1-8 dan menyisakan 1 kotak kosong, problem yang akan kita selesaikan adalah bagaimana cara mengurutkan angka acak di dalam kotak menjadi sebuah angka yang berurutan menggunakan metode DFS dan BFS

2	8	3
1	6	4
7		5

STATE AWAL

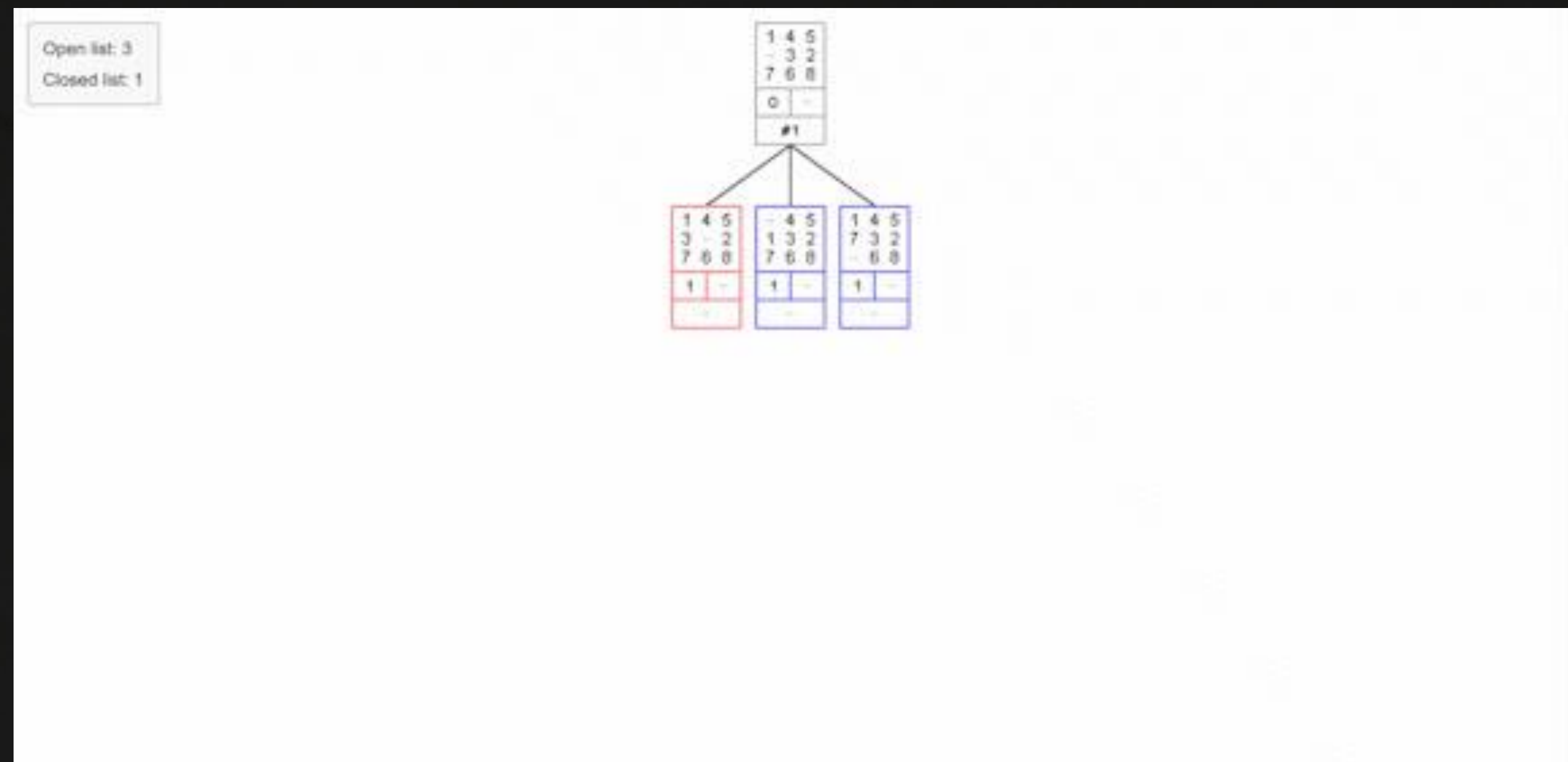
1	2	3
8		4
7	6	5

STATE AKHIR

ILUSTRASI

8-PUZZLE

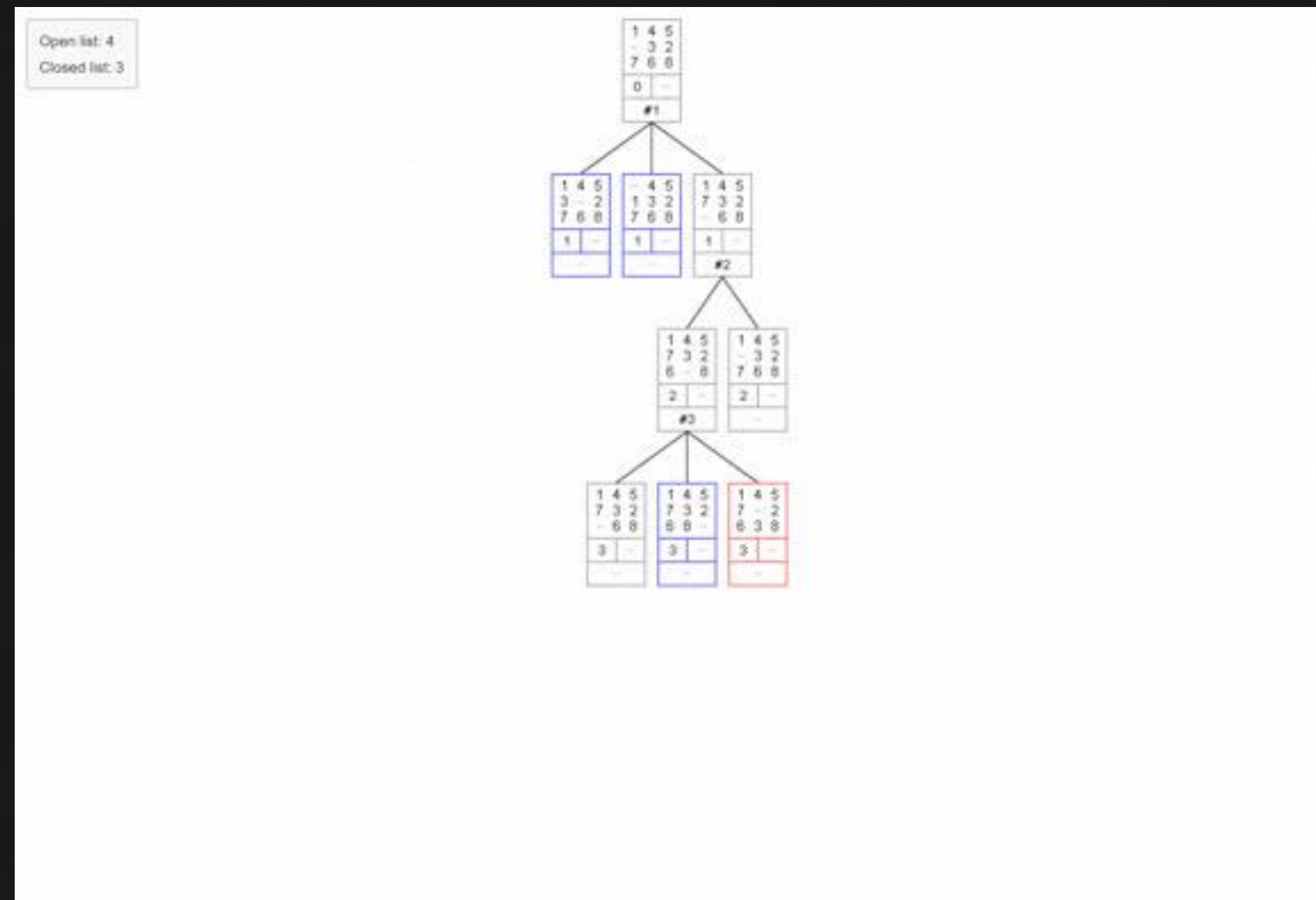
BREADTH-FIRST SEARCH



ILUSTRASI

8-PUZZLE

DEPTH-FIRST SEARCH



FUNGSI CLASS NODE : Mendeklarasikan tiap state yang disimpan

```
1  from platform import node
2  import sys
3  import numpy as np
4
5
6  class Node:
7      def __init__(self, state, parent, action):
8          self.state = state
9          self.parent = parent
10         self.action = action
```


FUNGSI CLASS STACKFRONTIER : Menyimpan fungsi dari DFS method

```
13 class StackFrontier:
14     def __init__(self):
15         self.frontier = []
16
17         self.frontier.append(node)
18
19     def contains_state(self, state):
20         return any((node.state[0] == state[0]).all() for node in self.frontier)
21
22     def empty(self):
23         return len(self.frontier) == 0
24
25     def remove(self):
26         if self.empty():
27             raise Exception("Empty Frontier")
28         else:
29             node = self.frontier[-1]
30             self.frontier = self.frontier[:-1]
31             return node
```


FUNGSI CLASS QUEUEFRONTIER : Menyimpan fungsi dari BFS method

```
34 class QueueFrontier(StackFrontier):
35     def remove(self):
36         if self.empty():
37             raise Exception("Empty Frontier")
38         else:
39             node = self.frontier[0]
40             self.frontier = self.frontier[1:]
41             return node
```

```

44 class Puzzle:
45     def __init__(self, start, startIndex, goal, goalIndex):
46         self.start = [start, startIndex]
47         self.goal = [goal, goalIndex]
48         self.solution = None
49
50     def neighbors(self, state):
51         mat, (row, col) = state
52         results = []
53
54         if row > 0:
55             mat1 = np.copy(mat)
56             mat1[row][col] = mat1[row - 1][col]
57             mat1[row - 1][col] = 0
58             results.append(('up', [mat1, (row - 1, col)]))
59         if col > 0:
60             mat1 = np.copy(mat)
61             mat1[row][col] = mat1[row][col - 1]
62             mat1[row][col - 1] = 0
63             results.append(('left', [mat1, (row, col - 1)]))
64         if row < 2:
65             mat1 = np.copy(mat)
66             mat1[row][col] = mat1[row + 1][col]
67             mat1[row + 1][col] = 0
68             results.append(('down', [mat1, (row + 1, col)]))
69         if col < 2:
70             mat1 = np.copy(mat)
71             mat1[row][col] = mat1[row][col + 1]
72             mat1[row][col + 1] = 0
73             results.append(('right', [mat1, (row, col + 1)]))
74
75         return results

```

FUNGSI CLASS PUZZLE :

def __init__, menginisiasi state awal - state akhir

def neighbors, untuk memprediksi kolom kosong akan digeser ke arah atas/bawah/kiri/kanan

MENAMPILKAN HASIL OUTPUT DARI DEF NEIGHBORS

```
77     def print(self):
78         solution = self.solution if self.solution is not None else None
79         print("Start State:\n", self.start[0], "\n")
80         print("Goal State:\n", self.goal[0], "\n")
81         print("\nStates Explored: ", self.num_explored, "\n")
82         print("Solution:\n ")
83         for action, cell in zip(solution[0], solution[1]):
84             print("action: ", action, "\n", cell[0], "\n")
85         print("Goal Reached!!")
```

MENCARI TAHU APAKAH PUZZLE MASIH MEMPUNYAI STATE ATAU TIDAK

```
87     def does_not_contain_state(self, state):
88         for st in self.explored:
89             if (st[0] == state[0]).all():
90                 return False
91         return True
```



```

93 def solve(self):
94     self.num_explored = 0
95
96     start = Node(state=self.start, parent=None, action=None)
97     frontier = QueueFrontier()
98     frontier.add(start)
99
100    self.explored = []
101
102    while True:
103        if frontier.empty():
104            raise Exception("No solution")
105
106        node = frontier.remove()
107        self.num_explored += 1
108
109        if (node.state[0] == self.goal[0]).all():
110            actions = []
111            cells = []
112            while node.parent is not None:
113                actions.append(node.action)
114                cells.append(node.state)
115                node = node.parent
116            actions.reverse()
117            cells.reverse()
118            self.solution = (actions, cells)
119            return
120
121        self.explored.append(node.state)
122
123        for action, state in self.neighbors(node.state):
124            if not frontier.contains_state(state) and self.does_not_contain_state(state):
125                child = Node(state=state, parent=node, action=action)
126                frontier.add(child)

```

```

129 start = np.array([[1, 2, 3], [7, 8, 4], [0, 6, 5]])
130 goal = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])
131
132
133 startIndex = (2, 0)
134 goalIndex = (1, 1)
135
136
137 p = Puzzle(start, startIndex, goal, goalIndex)
138 p.solve()
139 p.print()

```

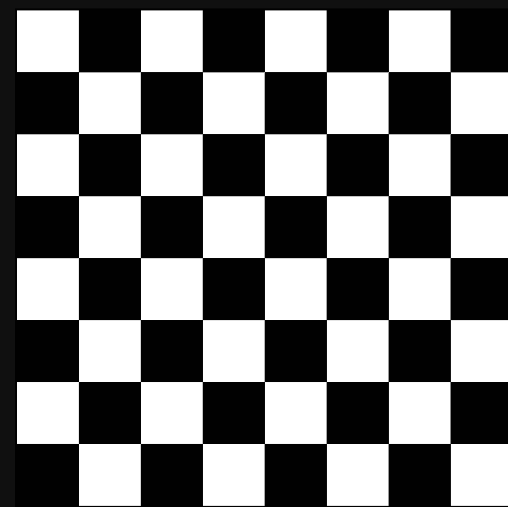


8-QUEEN

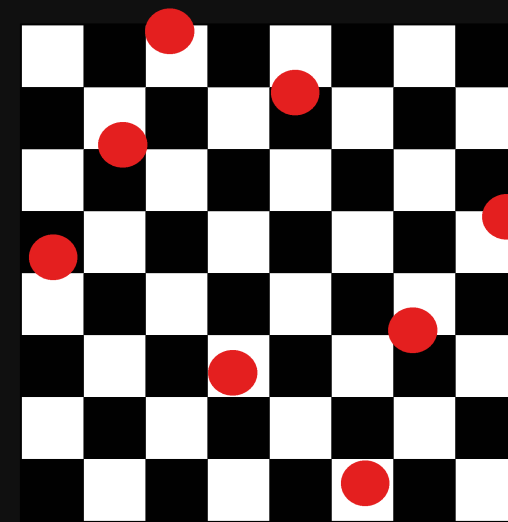
Terdapat sebuah papan catur yang masih kosong, problem yang akan kita selesaikan adalah meletakkan pion queen sebanyak 8 buah dengan posisi tertentu agar pion queen tidak bisa saling menyerang satu sama lain.

Ketentuan dalam permainan catur adalah queen bisa bergerak secara diagonal atau tegak lurus

Algoritma dimulai dengan memposisikan queen di baris pertama, lalu berlanjut ke baris berikutnya dengan menempatkan ratu di kolom yang aman pertama baris tersebut. Jika algoritma sudah mencapai baris terakhir dan semua queen sudah ditempatkan di posisi yang aman, maka ketentuan dari state akhir telah tercapai



STATE AWAL



STATE AKHIR