```
# Mostrando informações iniciais
print(f"Initial state: \n{start}")
print("*"*15)
print(f"Target state: \n{target}")
print("*"*15)
```

```
Initial state:
[[1 0 2]
 [8 4 3]
 [7 6 5]]
***************
Target state:
[[1 2 3]
 [8 0 4]
 [7 6 5]]
***************
```

BFS

In [ ]:

DFS

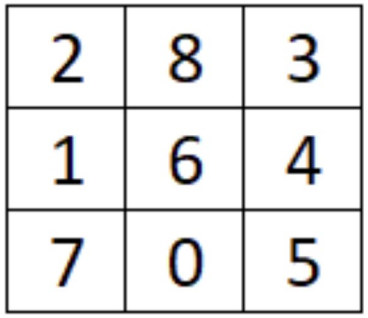
In [ ]:

Resultados

In [ ]:

Exercícios

1. Alterar a matriz inicial de posições para a apresentada na imagem abaixo e avaliar a performance das duas abordagens de busca cega



In [ ]:
```
# Pacote auxiliar para o cálculo do tempo
from time import time

# Criando objeto do problema
problema = SlidingPuzzle(3)

# Criando Matriz inicial e matriz alvo
start = np.matrix([[2,8,3],[1,6,4],[7,0,5]])
target = np.matrix([[1,2,3],[8,0,4],[7,6,5]])

# Mostrando informações iniciais
print(f"Initial state: \n{start}")
print("*"*15)
print(f"Target state: \n{target}")
print("*"*15)
```

```
Initial state:
[[2 8 3]
 [1 6 4]
 [7 0 5]]
***************
Target state:
[[1 2 3]
 [8 0 4]
 [7 6 5]]
***************
```

In [ ]:

```python
# Execução do BFS
bfs = BreadthFirstSearch(problema)

ini = time() # Tempo inicial

bfs_solucao, bfs_estados_visitados, bfs_num_visitados = bfs.busca(start, target) # chamando busca

bfs_time = time()-ini # Tempo total

if bfs_solucao:
    print(f"Solution found!!!")
else:
    print("Solution not found!!!")
```

```
Visitando #1
Visitando #2
Visitando #3
Visitando #4
Visitando #5
Visitando #6
Visitando #7
Visitando #8
Visitando #9
Visitando #10
Visitando #11
Visitando #12
Visitando #13
Visitando #14
Visitando #15
Visitando #16
Visitando #17
Visitando #18
Visitando #19
Visitando #20
Visitando #21
Visitando #22
Visitando #23
Visitando #24
Visitando #25
Visitando #26
Visitando #27
Visitando #28
Visitando #29
Visitando #30
Visitando #31
Visitando #32
Visitando #33
Visitando #34
Solution found!!!
```

In [ ]:
```python
# Execução do DFS
dfs = DepthFirstSearch(problema)
```

```
ini = time() # Tempo inicial

dfs_solucao, dfs_estados_visitados, dfs_num_visitados = dfs.busca(start, target) # chamando busca

dfs_time = time()-ini # Tempo total

if dfs_solucao:
    print(f"Solution found!!!")
else:
    print("Solution not found!!!")
```

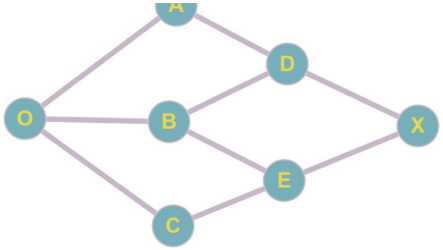
In [ ]:
```
# Apresentando resultados
print("==== BFS ====")
print(f"Solução encontrada? {bfs_solucao}")
print(f"Número de estados visitados: {bfs_num_visitados}")
print(f"Tempo de execução: {bfs_time}")

print("==== DFS ====")
print(f"Solução encontrada? {dfs_solucao}")
print(f"Número de estados visitados: {dfs_num_visitados}")
print(f"Tempo de execução: {dfs_time}")
```

```
==== BFS ====
Solução encontrada? True
Número de estados visitados: 34
Tempo de execução: 0.0029354095458984375
==== DFS ====
Solução encontrada? True
Número de estados visitados: 13747
Tempo de execução: 207.89054584503174
```

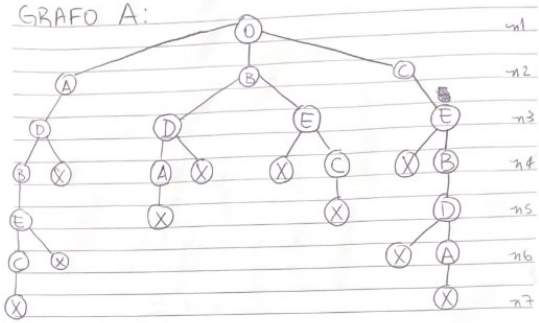
Para essa solução o DFS demorou muito mais para encontrar a resposta. Visitou 13700 estados a mais que o BFS.

2. Desenhe no papel, a árvore dos seguintes grafos e diga qual o caminho BFS e DFS saindo de O e indo para X.

A)



GRAFO A:



DFS: O → A → D → B → E → C → X

BFS: O → A → B → C → D → E → X

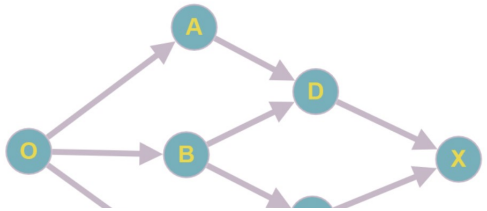
GRAFO A POSSUI CONEXÕES NÃO DIRECIONADAS, O FLUXO PODE SER NOS DOIS SENTIDOS.

veja em:

https://graphonline.ru/pt/

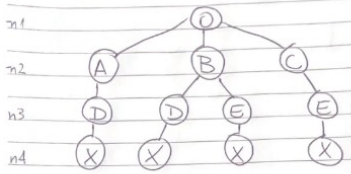
O site consegue executar graficamente os Algoritmos BFS e DFS no seu grafo.

B)

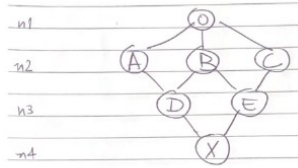


GRAFO B:

ROTACIONE O GRAFO para a vertical.



— OU —



DFS : O → A → D → X

BFS : O → A → B → C → D → E → X

O FLUXO DO GRAFO É DIRECIONAL, ESSA propriedade facilita a montagem da árvore.