Trabalho 2 - Análise de Algoritmos

Integrantes:

André Costa de Andrade - 2012498

Laura Mendonça da Glória Luz – 2010709

Observação sobre montagem do grafo:

Utilizamos um dicionário, cuja chave é uma string da matriz que representa a configuração. Tivemos que fazer essa conversão, pois o dicionário do Python não aceita uma lista como chave. Por exemplo:

```
tmp_exemplo = np.array(cfg)
tmp_exemplo = np.array2string(tmp_exemplo, separator=", ")
```

A primeira linha converte a configuração em formato de lista para array do numpy.

A segunda linha transforma o array em uma string.

Tarefa 1: Criação do grafo de espaço de estados

Foram encontrados `362880 nós e 483840 arestas.`

Analisando o problema, pode-se ver que existem 362880 estados possíveis, esse número é obtido a partir de análise combinatória. São 9 objetos a serem permutados sem repetição (considerando o espaço em branco como um objeto em si), portanto o resultado é 9! .

A função que fizemos para encontrar o grafo de espaço de estados funciona basicamente por força bruta. Dado uma configuração inicial, exploramos todas as configurações possíveis a partir dela e adicionamos ao grafo. Dito isso, a partir da análise de configurações possíveis por permutação descobrimos que a partir de alguma configuração inicial não era possível conseguir todo o espaço de estados, então rodamos a função para dois inputs em conjuntos disjuntos:

```
cfg_inicial_1 = [['1', '2', '3'],
                 ['4', '5', '6'],
                 ['7', '8', ' ']]


cfg_inicial_2 = [['1', '2', '3'],
                 ['4', '5', '6'],
                 ['8', '7', ' ']]
```

Nós conectados por uma aresta

```
[['1', '2', '3'],       [['1', '2', '3'],
 ['4', '5', '6'],  ---   ['4', '5', ' '],
 ['7', '8', ' ']]        ['7', '8', '6']]
```

Nós NÃO conectados por uma aresta

```
[['1', '2', '3'],     [['1', '2', '3'],
 ['4', '5', '6'],     ['4', ' ', '6'],
 ['7', '8', ' ']]     ['7', '5', '8']]
```

Tarefa 2: Implementação de BFS e contagem de componentes conexos N

1. O código principal da BFS.

```python
def bfs(G, s):
    visited = {}
    queue = [s]
    while queue:
        node = queue.pop(0)
        if visited.get(node, 0) == 0:
            visited[node] = 1
            queue.extend(G.edges[node])
    return visited
```

Foram encontrados 2 componentes conexos para o grafo construído na primeira questão. Esse número foi encontrado a partir da seguinte função:

```python
def connected_components(G):
    components = []
    visited = {}
    for node in G.edges:
        if visited.get(node, 0) == 0:
            component = bfs(G, node)
            components.append(component)
            visited.update(component)
    return components
```

Tarefa 3: Caminhos mais curto

Para esta questão utilizamos uma BFS modificada (no código bfs_2) para retornar

```
def bfs_2(G, s):
    visited = {}
    queue = [s]
    layer = 0
    parents = {}
    parents[s] = None
    while queue:
        layer_nodes = []
        for node in queue:
            if visited.get(node, 0) == 0:

                visited[node] = [layer, parents[node]]
                layer_nodes.extend(G.edges[node])

                for child in G.edges[node]:
                    if visited.get(child, 0) == 0:
                        parents[child] = node
        queue = layer_nodes
        layer += 1
    return visited
```

Foram encontradas duas configurações viáveis com maior número de movimentos para chegar à configuração dada:

```
[['8', '6', '7'],
 ['2', '5', '4'],
 ['3', ' ', '1']]
```

e

```
[['6', '4', '7'],
 ['8', '5', ' '],
 ['3', '2', '1']]
```

São necessários 31 movimentos para se chegar ao caminho mais curto com maior número de movimentos.