

```
# Giovana Lacerda Machado - Inteligência Computacional (19/08/25)
```

```
# Busca em Largura no Problema das Jarras
```

```
!pip list
```

```
torchsummary          1.5.1
torchtune              0.6.1
torchvision            0.23.0+cu126
tornado                6.4.2
tqdm                   4.67.1
traitlets              5.7.1
traitlets              0.2.1
transformers           4.55.2
treelite               4.4.1
treescope              0.1.10
triton                 3.4.0
tsfresh                0.21.0
tweepy                 4.16.0
typeguard              4.4.4
typer                  0.16.0
types-pytz             2025.2.0.20250809
types-setuptools       80.9.0.20250809
typing_extensions      4.14.1
typing-inspection      0.4.1
tzdata                 2025.2
tzlocal                5.3.1
uc-micro-py            1.0.3
ucx-py-cu12            0.44.0
ucxx-cu12              0.44.0
umap-learn             0.5.9.post2
umf                    0.11.0
uritemplate            4.2.0
urllib3                2.5.0
uvicorn                0.35.0
vega-datasets          0.9.0
wadllib                1.3.6
wandb                  0.21.1
wasabi                 1.1.3
wcwidth                0.2.13
weasel                 0.4.1
webcolors              24.11.1
webencodings           0.5.1
websocket-client       1.8.0
websockets             15.0.1
Werkzeug               3.1.3
wheel                  0.45.1
widgetsnbextension     3.6.10
wordcloud              1.9.4
wrapt                  1.17.3
wurlitzer              3.1.1
xarray                 2025.8.0
xarray-einstats        0.9.1
xgboost                3.0.4
xlrd                   2.0.2
xxhash                 3.5.0
xyzservices            2025.4.0
yarl                   1.20.1
ydf                    0.13.0
yellowbrick            1.5
yfinance               0.2.65
zict                   3.0.0
zipp                   3.23.0
zstandard              0.23.0
```

```
from collections import deque
```

```
# Capacidades máximas das jarras
JARRA_1_CAPACID = 3 # jarra menor
JARRA_2_CAPACID = 7 #5 # jarra maior
```

```
# Objetivo: obter exatamente 4 litros na jarra de 5 litros
OBJETIVO = 5 #4
```

```
# Estado inicial: ambas as jarras estão vazias
estado_inicial = (0, 0)
```

```
# Função que gera todos os estados possíveis a partir de um estado atual
def obter_estados_seguintes(estado):
    x, y = estado # x: quantidade na jarra 1, y: quantidade na jarra 2
    estados = []
```

```
    # Encher completamente a jarra 1
    estados.append((JARRA_1_CAPACID, y))
    # Encher completamente a jarra 2
```

```

# Encher completamente a jarra 2
estados.append((x, JARRA_2_CAPACID))
# Esvaziar completamente a jarra 1
estados.append((0, y))
# Esvaziar completamente a jarra 2
estados.append((x, 0))
# Transferir da jarra 1 para a 2 até encher a 2 ou esvaziar a 1
transferir = min(x, JARRA_2_CAPACID - y)
estados.append((x - transferir, y + transferir))
# Transferir da jarra 2 para a 1 até encher a 1 ou esvaziar a 2
transferir = min(y, JARRA_1_CAPACID - x)
estados.append((x + transferir, y - transferir))

return estados

# Função que implementa a busca em largura (BFS) para encontrar a solução
def bfs():
    fila = deque()          # fila para armazenar os estados a serem explorados
    visitado = set()        # conjunto para armazenar os estados já visitados
    predecessor = {}        # dicionário para rastrear o caminho de cada estado

    fila.append(estado_inicial)
    visitado.add(estado_inicial)
    predecessor[estado_inicial] = None

    while fila:
        atual = fila.popleft()
        x, y = atual
        print(f"Jarra {JARRA_1_CAPACID}L: {x}L, Jarra {JARRA_2_CAPACID}L: {y}L") # imprime o estado atual
        # Verifica se atingimos o objetivo
        if y == OBJETIVO:
            print("\nSolução encontrada!")
            caminho = []
            # Reconstrói o caminho desde o estado inicial até o objetivo
            while atual:
                caminho.append(atual)
                atual = predecessor[atual]
            caminho.reverse()
            # Imprime a sequência de estados que leva à solução
            for estado in caminho:
                print(f"Jarra {JARRA_1_CAPACID}L: {estado[0]}L, Jarra {JARRA_2_CAPACID}L: {estado[1]}L")
            return

        # Gera e explora os próximos estados possíveis
        for estado_seguente in obter_estados_seguintes(atual):
            if estado_seguente not in visitado:
                visitado.add(estado_seguente)
                predecessor[estado_seguente] = atual
                fila.append(estado_seguente)

    print("Nenhuma solução encontrada.")

# Executa o algoritmo de busca Breadth First Search
bfs()

```

 Jarra 3L: 0L, Jarra 7L: 0L
 Jarra 3L: 3L, Jarra 7L: 0L
 Jarra 3L: 0L, Jarra 7L: 7L
 Jarra 3L: 3L, Jarra 7L: 7L
 Jarra 3L: 0L, Jarra 7L: 3L
 Jarra 3L: 3L, Jarra 7L: 4L
 Jarra 3L: 3L, Jarra 7L: 3L
 Jarra 3L: 0L, Jarra 7L: 4L
 Jarra 3L: 0L, Jarra 7L: 6L
 Jarra 3L: 3L, Jarra 7L: 1L
 Jarra 3L: 3L, Jarra 7L: 6L
 Jarra 3L: 0L, Jarra 7L: 1L
 Jarra 3L: 2L, Jarra 7L: 7L
 Jarra 3L: 1L, Jarra 7L: 0L
 Jarra 3L: 2L, Jarra 7L: 0L
 Jarra 3L: 1L, Jarra 7L: 7L
 Jarra 3L: 0L, Jarra 7L: 2L
 Jarra 3L: 3L, Jarra 7L: 5L

Solução encontrada!
 Jarra 3L: 0L, Jarra 7L: 0L
 Jarra 3L: 0L, Jarra 7L: 7L
 Jarra 3L: 3L, Jarra 7L: 4L
 Jarra 3L: 0L, Jarra 7L: 4L
 Jarra 3L: 3L, Jarra 7L: 1L
 Jarra 3L: 0L, Jarra 7L: 1L
 Jarra 3L: 1L, Jarra 7L: 0L
 Jarra 3L: 1L, Jarra 7L: 7L
 Jarra 3L: 3L, Jarra 7L: 5L

```
## comentários ##
```

```
A linha de código: from collections import deque  
faz parte da biblioteca padrão do Python
```

`collections`: é um módulo que fornece tipos de dados especializados, além dos tipos básicos como listas, dicionários, etc.

`deque` (pronuncia-se “deck”, abreviação de double-ended queue): é uma estrutura de dados semelhante a uma lista, mas otimizada para inserir e remover elementos em ambos os extremos.

No algoritmo de busca em largura (BFS), é necessário:

Adicionar novos estados ao final da fila.

Remover o estado atual do início da fila.

A estrutura `deque` é ideal para isso, pois:

É mais eficiente que uma lista comum (`list`) para operações de fila.

Tem métodos como `.append()` e `.popleft()` que funcionam em tempo constante.