Problema dos Jarros D'Água por meio de Algoritmos de Busca sem Informação

Fernanda Vitória Araújo Santos[1], Renan da Costa Silva[2]

[1]Análise em Desenvolvimentos de Sistemas – Instituto Federal do Piauí (IFPI)
Caixa Postal gabinete.capic@ifpi.edu.br – 64605-500 – Picos – PI – Brasil

Resumo. *Este relatório apresenta a aplicação de três algoritmos de busca não informada — Busca em Largura (BFS), Busca em Profundidade (DFS) e Aprofundamento Iterativo (IDS) — para resolver o Problema dos Jarros d'Água, cujo objetivo é obter exatamente 2 litros em um dos jarros de capacidades 4 L e 3 L. São descritas as funções utilizadas, o método de geração de sucessores, os percursos encontrados e a comparação do desempenho entre as estratégias.*

1. Introdução

Este relatório apresenta a implementação, solução e a análise para o Problema dos Jarros d'Água, que consiste em determinar uma sequência de operações (encher, esvaziar e derramar) para obter exatamente 2 litros de água em um dos vasos, utilizando um vaso de 4 litros e um vaso de 3 litros.

O trabalho tem como objetivo aplicar e comparar três estratégias de busca sem informação, sendo elas as buscas em largura (BFS), profundidade (DFS) e com aprofundamento iterativo (IDS) para resolver esse problema, destacando o comportamento e o desempenho de cada estratégia.

2. Metologia

A implementação foi realizada na linguagem Python, utilizando as bibliotecas collections e time. O código define as seguintes constantes e estruturas:

- **VASO_4, VASO_3:** capacidades dos vasos.

```python
VASO_4 = 4

VASO_3 = 3
```

- **ESTADO_INICIAL:** ambos os vasos inicialmente vazios.

```python
ESTADO_INICIAL = (0, 0)
```

- **objetivo(estado):** função que verifica se um dos vasos contém 2 litros.

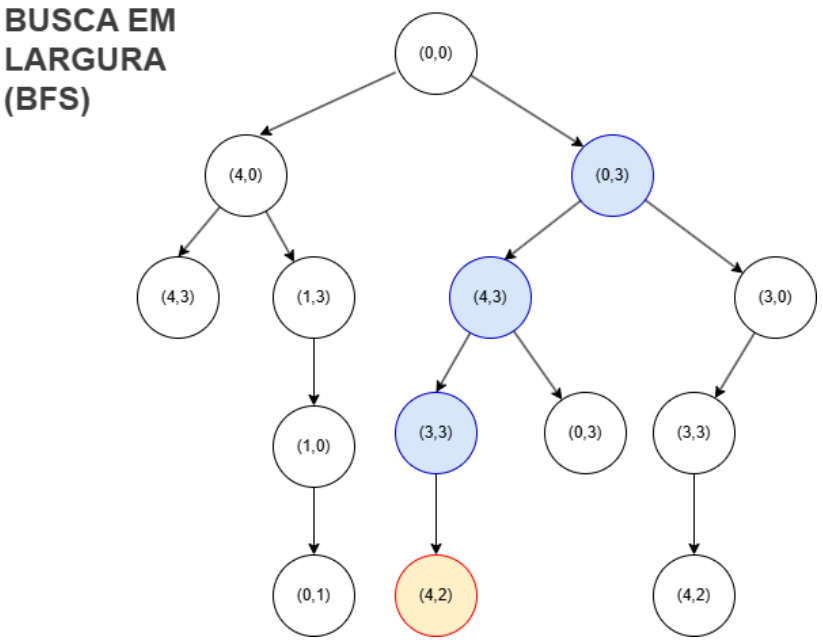
```python
def objetivo(estado):

    a, b = estado

    return a == 2 or b == 2
```

- **sucessores(estado):** gera todos os possíveis estados alcançáveis a partir do estado atual, considerando as operações permitidas (encher, esvaziar e derramar no outro).

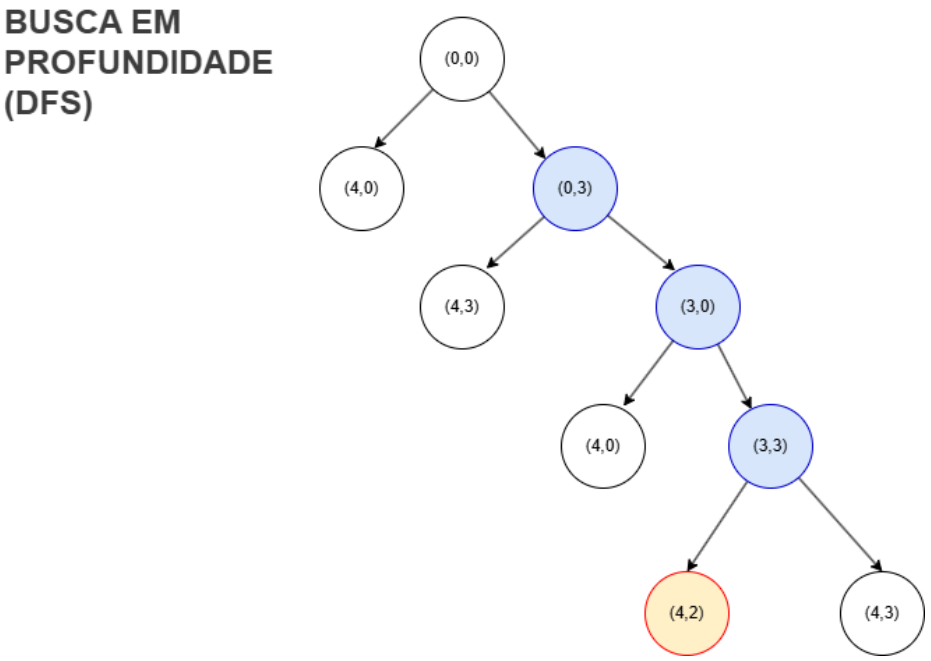
```
def sucessores(estado):
    a, b = estado
    sucessores = []

    if a < VASO_4:
        sucessores.append((VASO_4, b))
    if b < VASO_3:
        sucessores.append((a, VASO_3))
    if a > 0:
        sucessores.append((0, b))
    if b > 0:
        sucessores.append((a, 0))
    if a > 0 and b < VASO_3:
        derramar = min(a, VASO_3 - b)
        sucessores.append((a - derramar, b + derramar))
    if b > 0 and a < VASO_4:
        derramar = min(b, VASO_4 - a)
        sucessores.append((a + derramar, b - derramar))

    return sucessores
```

Buscas:

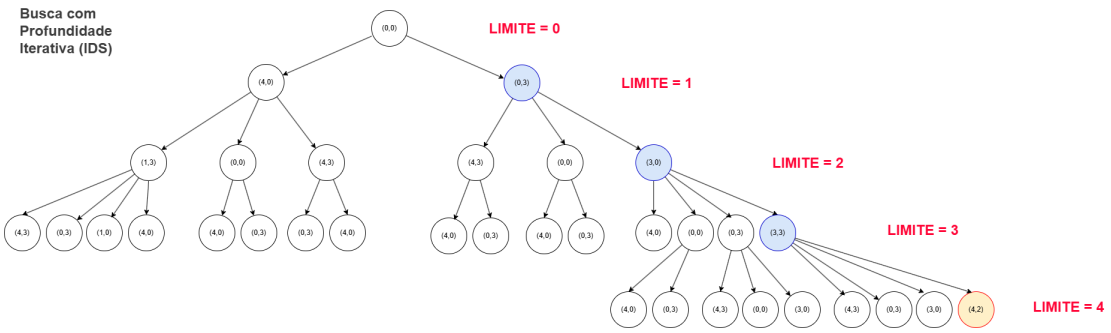
1. **BFS** - utiliza uma fila para explorar os estados em ordem de profundidade crescente. Garante a solução mais curta.



2. **DFS** - utiliza uma pilha para explorar os caminhos até o fundo antes de voltar, podendo gerar soluções mais longas.



3. **IDS** - combina a Busca em Profundidade com um limite de profundidade crescente para garantir completude e otimalidade do caminho.



O programa principal mede o tempo de execução de cada algoritmo e imprime o caminho percorrido desde o estado inicial até a solução, além de medir os passos necessários, os estados visitados e a quantidade de nós expandidos;

3. Busca em Largura (BFS)

A Busca em Largura (Breadth-First Search) é uma estratégia de busca não informada que explora o espaço de estados nível por nível, garantindo que todos os nós de uma determinada profundidade sejam visitados antes de prosseguir para o próximo nível.

Essa busca utiliza uma fila (FIFO), em que o primeiro estado inserido é o primeiro a ser expandido.

```
def bfs(estado_inicial=ESTADO_INICIAL):
    fila = deque([(estado_inicial, [])])
    visitado = set()

    while fila:
        estado, caminho = fila.popleft()

        if objetivo(estado):
            return caminho + [estado]

        if estado in visitado:
            continue
        visitado.add(estado)
```

```
        for successor in sucessores(estado):
            if successor not in visitado:]
                fila.append((successor, caminho + [estado]))

    return None
```

No contexto do Problema dos Jarros d'Água, a BFS inicia no estado (0, 0) ou seja ambos os jarros vazios e, a cada iteração, gera todos os estados sucessores possíveis como encher, esvaziar ou transferir água de um jarro para outro. Cada novo estado é adicionado ao final da fila. O processo continua até que um dos vasos contenha exatamente 2 litros de água, condição que caracteriza o estado objetivo.

4. Busca em Profundidade (DFS)

A Busca em Profundidade (Depth-First Search) também é uma técnica de busca não informada, mas segue uma lógica oposta à da BFS. Ela utiliza uma pilha (LIFO) para armazenar os estados, explorando o espaço de busca o mais profundamente possível antes de retroceder.

```
def dfs(estado_inicial=ESTADO_INICIAL):
    pilha = [(estado_inicial, [])]
    visitado = set()

    while pilha:
        estado, caminho = pilha.pop()

        if objetivo(estado):
            return caminho + [estado]

        if estado in visitado:
            continue
```

```
        visitado.add(estado)



        for successor in sucessores(estado):

            if successor not in visitado:]

                pilha.append((successor, caminho + [estado]))



    return None
```

No problema dos jarros, a DFS inicia igualmente no estado (0, 0) e segue uma sequência de operações até atingir um estado sem novos sucessores. Nesse ponto, o algoritmo retrocede ao último estado com caminhos não explorados, continuando a busca até que encontre o estado objetivo, ou seja aquele em que um dos vasos contém 2 litros.

5. Busca com Aprofundamento Iterativo (IDS)

A Busca com Aprofundamento Iterativo (Iterative Deepening Search - IDS) é uma estratégia de busca não informada que combina a completude e otimalidade da Busca em Largura (BFS) com a eficiência de memória da Busca em Profundidade (DFS).

O IDS funciona realizando uma série de Buscas em Profundidade Limitada (DLS), aumentando o limite de profundidade em uma unidade a cada iteração.

1. Na primeira iteração, o limite de profundidade é 0.
2. Na segunda, o limite é 1.
3. E assim sucessivamente $(L=2, L=3, \dots)$, até que a solução seja encontrada.

A função dls() é a implementação da Busca em Profundidade Limitada:

```
def dls(estado_inicial, limite_profundidade):
    pilha = [(estado_inicial, [], 0)]
    visitados = 0

    while pilha:
```

```
        estado, caminho, profundidade = pilha.pop()
        visitados += 1

        if objetivo(estado):
            return caminho + [estado], visitados

        if profundidade < limite_profundidade:
            for successor, acao in sucessores(estado):
                pilha.append((successor, caminho + [estado],
profundidade + 1))

        print()

    return None, visitados
```

A função ids() realiza as chamadas à dls(), incrementando o limite até encontrar a solução:

```
# busca com profundidade iterativa
def ids(estado_inicial=ESTADO_INICIAL):
    limite = 0
    total_visitados_geral = 0

    while True:

        solucao, visitados = dls(estado_inicial, limite)
        total_visitados_geral += visitados

        if solucao:
            return solucao, total_visitados_geral
```

```
        limite += 1


        if limite > 50:
            return None
```

No Problema dos Jarros d'Água, o IDS garante encontrar o caminho mais curto porque ele visita todos os nós de profundidade x antes de expandir para a profundidade x+1. A principal desvantagem é o re-exame repetitivo dos nós nas profundidades mais rasas, que são visitados a cada nova iteração de limite.

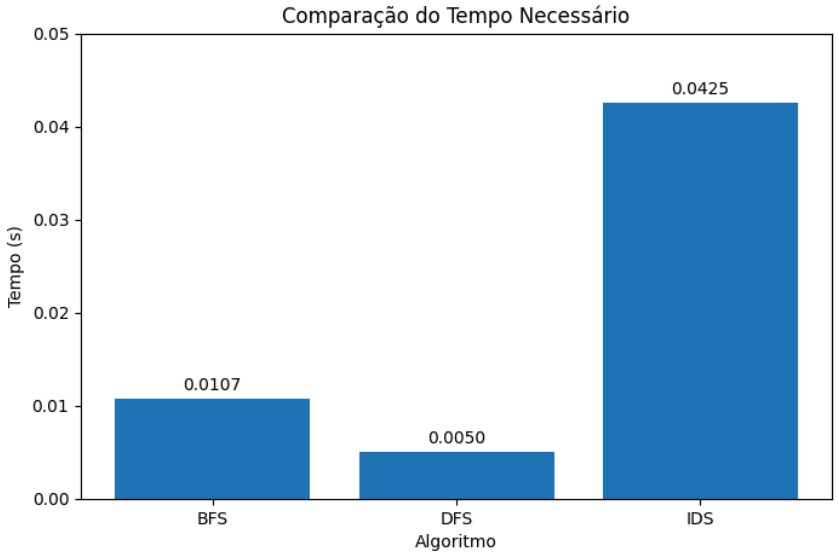
6. Análise Comparativa

Com base em execuções do código , todos alcançam a solução ótima devido ao espaço pequeno. Diferenças surgem em exploração e eficiência.

1. Número de Passos

- BFS: 4 ações (ótimo). Explora niveladamente, garantindo o caminho mais curto.
- DFS: 4 ações (ótimo). Prioriza profundidade, mas encontrou o mínimo devido à ordem de sucessores.
- IDS: 4 ações (ótimo). Simula BFS com limites, assegurando otimalidade.
- Comparação: Empate em otimalidade. Em espaços maiores, DFS poderia ter mais passos; BFS/IDS garantem mínimo.

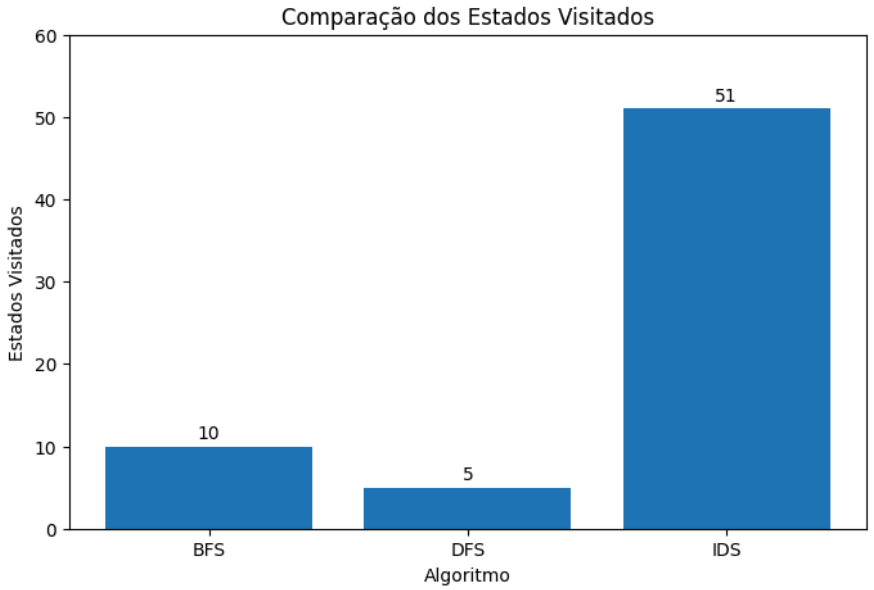
2. Tempo Necessário



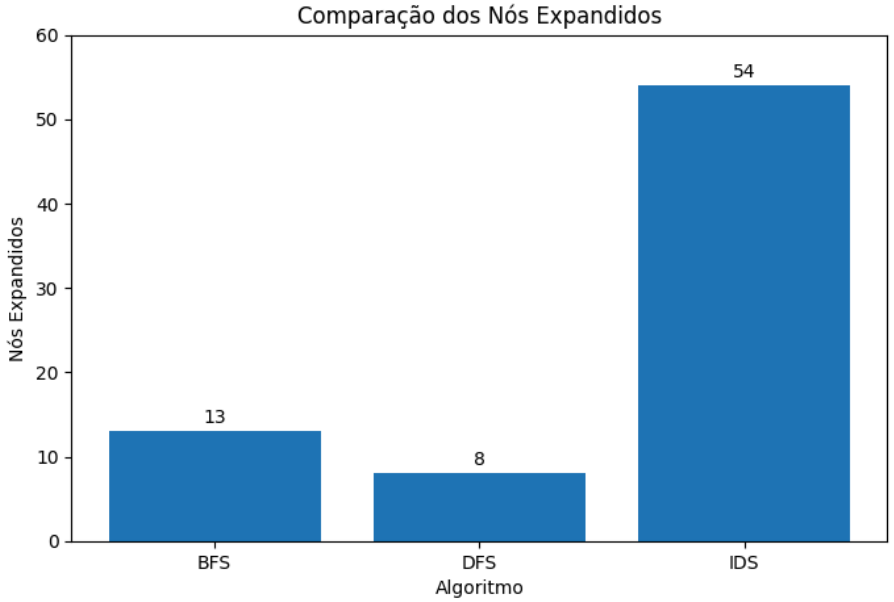
- DFS: Mais rápido. Para cedo, explorando menos.
- BFS: Médio. Sistemático, mas visita mais antes da solução.
- IDS: Mais lento. Reexecução - 4 iterações.
- Comparação: DFS é cerca de 2x mais rápido que BFS e 8x que IDS. Em problemas maiores o BFS pode ser mais lento.

3. Outras Métricas

- Estados Visitados:



  - DFS = baixo;
  - BFS = médio;
  - IDS = alto devido a repetições.
- Nós Expandidos:



- DFS = baixo;
- BFS = médio;
- IDS = alto.

Tabela de Comparação:

Critério	**BFS**	**DFS**	**IDS**
Solução ótima	Sim	Não	Sim
Tempo de execução	Médio	Menor	Maior
Estados visitados	Médio	Baixo a médio	Alto
Nós expandidos	Médio	Baixo	Alto
Uso de memória	Alto	Baixo	Baixo

7. Conclusão

O Problema dos Jarros d'Água permite observar as diferenças entre algoritmos de busca sem informação:

- **BFS** é o melhor algoritmo para este problema, garantindo a solução mais curta de forma direta e organizada.

- **DFS** embora rápida e leve, pode gerar caminhos longos e ineficientes, sendo menos indicada quando a optimalidade é necessária.
- **IDS** oferece uma solução ótima com baixo consumo de memória, mas a um custo maior de tempo devido às reexplorações necessárias.

Em resumo:

- BFS é eficiente e ótima,
- DFS é rápida mas inconsistente,
- IDS é confiável, porém mais lento.