**UNIVERSIDADE FEDERAL DE MINAS GERAIS (UFMG)**
**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

NICOLAS VON DOLINGER MOREIRA ROCHA (2022035571)

# TRABALHO PRÁTICO 0

## RESOLVENDO SUDOKU COM BUSCA EM ESPAÇO DE ESTADOS

BELO HORIZONTE

2024

# TRABALHO PRÁTICO 0: RESOLVENDO SUDOKU COM BUSCA EM ESPAÇO DE ESTADOS

**Nicolas von Dolinger Moreira Rocha**[1]

## 1. Introdução

Este código aborda a resolução de um quebra-cabeça clássico: o Sudoku. O Sudoku é um jogo de lógica onde o objetivo é preencher uma grade 9x9 com dígitos de 1 a 9, de modo que cada coluna, cada linha e cada uma das nove subgrades 3x3 contenham todos os dígitos de 1 a 9 sem repetição.

O programa implementa diferentes algoritmos de busca para resolver o Sudoku, com o objetivo de encontrar a configuração correta para o tabuleiro fornecido como entrada. Os algoritmos implementados incluem Busca em Largura (BFS), Busca de Profundidade Limitada (DLS), Busca de Profundidade Iterativa (IDFS), Busca de Custo Uniforme (UCS), Busca A* e Busca de Melhor Escolha Gulosa (Greedy Best-First Search).

Cada algoritmo é aplicado para tentar preencher o tabuleiro do Sudoku e encontrar a solução. O desempenho de cada algoritmo é medido pelo número de estados explorados durante a busca e pelo tempo de execução necessário para encontrar a solução ou alcançar o limite de busca definido.

Ao comparar os diferentes algoritmos, avaliamos não apenas sua capacidade de encontrar uma solução para o Sudoku, mas também sua eficiência em termos de número de estados explorados e tempo de execução. Isso nos permitirá entender melhor as características de cada algoritmo e sua adequação para resolver o problema do Sudoku em diferentes cenários.

## 2. Metodologia

O programa foi desenvolvido em C++, compilado pelo GCC da GNU Compiler Collection. A máquina utilizada tem as configurações descritas abaixo.

---

[1] Graduando em Sistemas de Informação pela Universidade Federal de Minas Gerais (UFMG). Product Owner na iJunior, empresa júnior do Departamento de Ciência da Computação da UFMG. Monitor de Programação e Desenvolvimento de Software II.

### 2.1. Configurações da Máquina

Sistema operacional: Ubuntu 23.04
Compilador: G++/GNU Compiler Collection
Processador: AMD Ryzen 5-5500U
Memória RAM: 16 GB

### 2.2. Abordagem

Para resolver o Sudoku, foram implementados diversos algoritmos de busca, cada um com suas características e eficiência. Além disso, foi utilizado o conceito de TAD (Tipo Abstrato de Dados) para representar o tabuleiro do Sudoku.

O TAD utilizado é uma estrutura de dados que representa o tabuleiro do Sudoku como uma matriz de 9x9, onde cada célula pode conter um número de 1 a 9 ou ser vazia. Essa representação permite armazenar o estado atual do tabuleiro e realizar operações como verificação de validade de um número em uma determinada célula, verificação de células vazias, entre outras.

A importância do uso do TAD está na abstração dos detalhes de implementação do tabuleiro do Sudoku. Em vez de lidar diretamente com uma matriz e suas operações, podemos manipular o tabuleiro por meio de funções específicas do TAD, garantindo assim uma maior modularidade e legibilidade do código. Além disso, o TAD facilita a reutilização do código em diferentes algoritmos de busca, pois a manipulação do tabuleiro é encapsulada dentro do próprio TAD.

Essa abstração também permite que o foco seja direcionado para a implementação dos algoritmos de busca em si, sem se preocupar com os detalhes de manipulação do tabuleiro. Dessa forma, o TAD atua como uma interface entre os algoritmos de busca e a representação do tabuleiro, promovendo uma melhor organização e manutenção do código.

Para resolver o Sudoku, foram implementados diversos algoritmos de busca, cada um com suas características e eficiência. Abaixo estão detalhes de cada método utilizado:

#### 2.2.1. Busca em Largura (BFS):

A busca em largura é uma técnica de busca que explora todos os nós vizinhos de um nó antes de seguir para os nós vizinhos. Para aplicar o algoritmo ao Sudoku, a estratégia é preencher os espaços vazios da grade com todos os números possíveis (de 1 a 9) e expandir cada estado possível. O critério de parada é quando o tabuleiro está completamente preenchido ou não há mais movimentos possíveis. Essa abordagem é implementada utilizando uma fila (queue) para armazenar os estados.

Essa técnica garante encontrar uma solução, se existir, com o menor número de movimentos, mas pode ser computacionalmente custosa devido à exploração de muitos estados.

### 2.2.2. Busca de Profundidade Limitada (DLS):

A busca de profundidade limitada é uma variante da busca de profundidade que limita o número de níveis que podem ser explorados a partir de um determinado nó. Isso evita que o algoritmo entre em um loop infinito ou explore estados muito profundos. No Sudoku, o algoritmo tenta preencher os espaços vazios com números possíveis até atingir o limite de profundidade definido. Se uma solução não for encontrada dentro desse limite, o algoritmo recua e tenta outra abordagem. A busca de profundidade limitada é implementada de forma recursiva.

Essa abordagem é mais eficiente em termos de memória do que a busca em largura, mas pode levar mais tempo para encontrar uma solução, especialmente se o limite de profundidade for muito grande.

### 2.2.3. Busca de Profundidade Iterativa (IDFS):

A busca de profundidade iterativa é uma combinação da busca de profundidade e da busca de profundidade limitada. Ela realiza várias buscas de profundidade com diferentes limites de profundidade, começando com um limite pequeno e aumentando gradualmente até que uma solução seja encontrada. Isso combina a eficiência da busca de profundidade com a garantia de que a solução mais rasa seja encontrada primeiro.

No Sudoku, o IDFS é aplicado da mesma forma que a busca de profundidade limitada, mas com a diferença de que o limite de profundidade é aumentado iterativamente até encontrar uma solução.

### 2.2.4. Busca de Custo Uniforme (UCS):

A busca de custo uniforme é uma técnica de busca que expande sempre o nó com o custo mais baixo. No Sudoku, o custo é determinado pela profundidade do nó na árvore de busca. Isso significa que o algoritmo explora primeiro os nós mais rasos antes de passar para os mais profundos. Isso é útil para o Sudoku, pois garante que sejam feitas as escolhas mais seguras primeiro. Essa abordagem é implementada utilizando uma fila de prioridade (priority_queue) para ordenar os estados pela profundidade.

### 2.2.5. Busca A* (A-star):

A busca A* é uma extensão da busca de custo uniforme que utiliza uma função heurística para estimar o custo de alcançar o objetivo a partir de um determinado estado. No Sudoku, uma heurística comum é priorizar as células que possuem menos opções de preenchimento, uma vez que isso limita o crescimento da árvore de estados. Isso é feito contando o número de opções possíveis para cada célula vazia. A ideia por trás dessa heurística é que, ao priorizar as células com menos escolhas, reduzimos a complexidade da busca e focamos nos caminhos mais promissores primeiro.

Ao priorizar as células com menos opções de preenchimento, estamos direcionando a busca para os estados que têm uma probabilidade maior de levar a uma solução. Isso significa que exploramos primeiro os caminhos que têm menos ramificações, limitando assim o crescimento da árvore de busca. Como resultado, o algoritmo A* expande os nós com menor número de opções, o que leva a uma busca mais direcionada e eficiente em direção à solução do Sudoku.

Essa abordagem utiliza uma fila de prioridade para ordenar os estados com base no custo total, que é a soma do custo real do nó (profundidade) e da estimativa heurística do custo restante. Dessa forma, a busca A* consegue encontrar soluções eficientes para o Sudoku, explorando primeiro os caminhos mais promissores, conforme indicado pela heurística.

### 2.2.6. Busca de Melhor Escolha Gulosa (Greedy Best-First Search):

A busca de melhor escolha gulosa é uma abordagem heurística que seleciona o nó que parece ser o mais próximo do objetivo, sem considerar o custo total do caminho. No Sudoku,

isso significa priorizar o preenchimento das células que permitem o número que mais se repete no tabuleiro. Essa heurística é rápida, pois seleciona diretamente as células que têm mais chance de levarem a uma solução.

Ao priorizar as células que permitem o número que mais se repete no tabuleiro, estamos focando nas opções que têm maior probabilidade de nos levar a uma solução mais rapidamente. Isso pode ser visto como uma abordagem de "escolha gulosa", onde tomamos a decisão localmente ótima em cada passo, sem considerar a situação global do tabuleiro.

Cada algoritmo é implementado para tentar preencher o tabuleiro do Sudoku e encontrar a solução. O desempenho de cada algoritmo é medido pelo número de estados explorados durante a busca e pelo tempo de execução necessário para encontrar a solução ou alcançar o limite de busca definido.

Ao comparar os diferentes algoritmos, avaliamos não apenas sua capacidade de encontrar uma solução para o Sudoku, mas também sua eficiência em termos de número de estados explorados e tempo de execução. Isso nos permitirá entender melhor as características de cada algoritmo e sua adequação para resolver o problema do Sudoku em diferentes cenários.

## 3. Testes

Para verificar a corretude e eficiência dos algoritmos de resolução do Sudoku, foram realizados os seguintes testes:

### 3.1. Corretude dos Resultados

Foram realizados testes simples para garantir que os algoritmos retornam as soluções corretas para tabuleiros de Sudoku simples e bem definidos. Os resultados obtidos foram comparados com soluções conhecidas para confirmar a precisão dos algoritmos. Por exemplo, foram utilizados tabuleiros de Sudoku resolvidos manualmente ou disponíveis em fontes confiáveis para validar a corretude dos algoritmos. Os tabuleiros foram fornecidos como entrada para os algoritmos, que foram executados para preencher os espaços vazios. As soluções resultantes foram então comparadas com as soluções conhecidas.

### 3.2. Análise de Desempenho

Para avaliar a eficiência dos algoritmos, foram executados testes de desempenho com diferentes tabuleiros de Sudoku. Os tempos de execução de cada algoritmo foram medidos utilizando tabuleiros com diferentes níveis de dificuldade, desde tabuleiros simples até tabuleiros mais complexos. Os tempos de execução foram registrados e comparados entre os diferentes algoritmos. Além disso, foi observado o número de estados explorados durante a busca em cada algoritmo, fornecendo uma medida da complexidade de busca de cada método.

Esses testes permitiram entender como o desempenho dos algoritmos varia em diferentes cenários de entrada e identificar padrões de desempenho em relação à complexidade dos tabuleiros de Sudoku.

### 3.3. Comparação de Eficiência

Os tempos de execução e o número de estados explorados dos algoritmos foram comparados para determinar qual abordagem é mais eficiente em termos de tempo e espaço para resolver o Sudoku. Além disso, foram examinados os padrões de desempenho em relação ao aumento da complexidade dos tabuleiros de Sudoku. Isso permitiu identificar em qual ponto cada algoritmo começa a mostrar uma degradação significativa no desempenho, fornecendo insights sobre os limites de aplicabilidade de cada abordagem.

```
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe G "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
2054000 5632
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe A "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
51 0
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe I "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
151050 116
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe U "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
4631 43
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe D "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
4208 2
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
PS D:\Code\introduction_to_AI\tp1> bin/TP1.exe B "530070000 600195000 098000060 800060003 400803001 700020006 060000280 000419005 000080079"
4631 32
534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179
```

1. **Greedy (G)**:
    - Estados explorados: 2054000
    - Tempo de execução: 5632 ms
    - Comportamento: O algoritmo guloso explora um grande número de estados (2054000), pois prioriza preencher as células com menos opções disponíveis. Isso resulta em um tempo de execução mais longo (5632 ms) em comparação

com outros métodos. Ele desempenha bem em testes fáceis, mas testes difíceis/intermediários, expande muitos espaços.

2. **A* (A)**:
    - Estados explorados: 51
    - Tempo de execução: 0 ms
    - Comportamento: O algoritmo A* explora apenas 51 estados para encontrar uma solução, o que indica uma busca muito eficiente. No entanto, o tempo de execução é negligenciável (0 ms), o que sugere que a solução foi encontrada quase que instantaneamente.
3. **Iterative Deepening (I)**:
    - Estados explorados: 151050
    - Tempo de execução: 116 ms
    - Comportamento: O método de busca em profundidade iterativa explora um número considerável de estados (151050), o que indica uma busca menos eficiente do que o A*. No entanto, o tempo de execução ainda é razoavelmente rápido (116 ms).
4. **Uniform-Cost (U)**:
    - Estados explorados: 4631
    - Tempo de execução: 43 ms
    - Comportamento: O algoritmo de busca de custo uniforme explora 4631 estados para encontrar uma solução. Embora tenha uma quantidade moderada de estados explorados, o tempo de execução é relativamente rápido (43 ms).
5. **Depth-First (D)**:
    - Estados explorados: 4208
    - Tempo de execução: 2 ms
    - Comportamento: O método de busca em profundidade explora 4208 estados, o que é um número moderado. No entanto, o tempo de execução é muito curto (2 ms), indicando uma busca bastante eficiente.
6. **Breadth-First (B)**:
    - Estados explorados: 4631
    - Tempo de execução: 32 ms

- Comportamento: O algoritmo de busca em largura explora 4631 estados, semelhante ao Uniform-Cost. O tempo de execução é razoável (32 ms), indicando uma busca eficiente, embora não tão rápida quanto o Depth-First.

Através desses testes, foi possível validar tanto a corretude quanto a eficiência dos algoritmos em diferentes cenários de entrada. Essas análises permitiram compreender melhor as características de desempenho de cada abordagem e tomar decisões informadas sobre a escolha do método mais adequado para resolver o Sudoku em diferentes situações.

## 4. Discussão de Resultados

A análise comparativa dos resultados dos testes revela insights significativos sobre o desempenho dos algoritmos de resolução do Sudoku. Aqui estão algumas observações-chave:

### 4.1. Corretude dos Resultados

Ambos os algoritmos foram validados com sucesso em termos de corretude dos resultados para tabuleiros de Sudoku simples e bem definidos. Os resultados obtidos para os tabuleiros de teste foram consistentes com as soluções conhecidas, confirmando a precisão dos algoritmos.

### 4.2. Análise de Desempenho

Os testes de desempenho revelaram diferenças significativas entre os tempos de execução dos algoritmos. Abordagens como BFS e UCS mostraram-se eficientes em termos de tempo para tabuleiros mais simples, onde a busca é menos complexa. Por outro lado, algoritmos como IDFS e A* mostraram ser mais eficazes em tabuleiros de Sudoku mais complexos, onde a busca é mais profunda e precisa de uma heurística mais sofisticada.

A BFS, embora seja eficiente, pode se tornar impraticável para tabuleiros de Sudoku mais complexos devido à sua necessidade de espaço de memória. Já a busca de profundidade limitada (DLS) pode encontrar soluções em tempo razoável para esses tabuleiros, mas tem o problema de encontrar um limite de profundidade adequado. O IDFS resolve o problema do DLS aumentando gradualmente o limite de profundidade, permitindo encontrar soluções eficientes mesmo em tabuleiros complexos.

A busca A* utiliza uma heurística para guiar a busca de forma mais direcionada, resultando em desempenho superior em tabuleiros mais complexos, mas requer uma função heurística bem ajustada. Por fim, a busca de melhor escolha gulosa (Greedy Best-First Search) tende a ser rápida, mas pode não encontrar a solução mais eficiente.

### 4.3. Impactos da Utilização

A escolha entre os algoritmos têm um impacto significativo na eficiência e na escalabilidade da solução do Sudoku. Para tabuleiros simples, algoritmos como BFS e UCS podem ser adequados devido à sua simplicidade e eficiência de tempo. No entanto, para tabuleiros mais complexos, o IDFS, A* ou até mesmo o Greedy Best-First Search podem ser mais adequados, dependendo das características específicas do problema. A escolha do algoritmo certo é crucial para garantir um desempenho eficiente da solução do Sudoku.

Além disso, considerações sobre espaço de memória também devem ser levadas em conta, especialmente ao lidar com tabuleiros maiores ou mais complexos. Em resumo, os resultados dos testes destacam a importância de selecionar o algoritmo adequado com base nos requisitos específicos de desempenho e complexidade do tabuleiro de Sudoku. Cada algoritmo tem suas vantagens e desvantagens, e a escolha depende das características do problema em questão.

## 5. Conclusão

Este exercício proporcionou uma análise abrangente dos algoritmos de resolução do Sudoku, comparando várias abordagens de busca. As principais conclusões são as seguintes:

i. Os testes revelaram que diferentes algoritmos têm desempenhos variados dependendo da complexidade do tabuleiro de Sudoku. Enquanto abordagens como BFS e UCS são mais eficientes em tabuleiros mais simples, algoritmos como IDFS, A* e Greedy Best-First Search mostraram-se mais adequados para tabuleiros mais complexos.

ii. A BFS e UCS podem ser eficientes em termos de tempo de execução para tabuleiros mais simples, mas podem se tornar impraticáveis para tabuleiros mais complexos devido à sua necessidade de espaço de memória.

iii. O IDFS resolve o problema de profundidade limitada aumentando gradualmente o limite de profundidade, tornando-se eficaz mesmo em tabuleiros complexos.

iv. O A* utiliza uma heurística para guiar a busca, resultando em desempenho superior em tabuleiros mais complexos, mas requer uma função heurística bem ajustada.

v. A busca de melhor escolha gulosa (Greedy Best-First Search) tende a ser rápida, mas pode não encontrar a solução mais eficiente, especialmente em tabuleiros complexos.

Em resumo, este exercício proporciona insights valiosos sobre os diferentes algoritmos de resolução do Sudoku e sua aplicabilidade em diferentes cenários. A escolha do algoritmo certo depende da complexidade do tabuleiro e dos requisitos de desempenho. Este estudo destaca a importância de considerar o tempo de execução, a escalabilidade e a complexidade do problema ao escolher um algoritmo para resolver o Sudoku. Essas conclusões podem ser úteis para desenvolvedores ao implementar sistemas de resolução de Sudoku e otimizar seu desempenho.