Universidade Federal de Viçosa

Campus **Rio Paranaíba**

João Victor Nascimento Silva - 7020

REROLUÇÃO DE LABIRINTOS UTILIZANDO ALGORITMOS DE BUSCA

Rio Paranaíba - MG

2024

Universidade Federal de Viçosa

Campus **Rio Paranaíba**

João Victor Nascimento Silva - 7020

REROLUÇÃO DE LABIRINTOS UTILIZANDO ALGORITMOS DE BUSCA

Trabalho apresentado para obtenção de créditos na disciplina SIN 323 - Inteligência Artificial da Universidade Federal de Viçosa - Campus de Rio Paranaíba, ministrada pela Professora. Dra. Larissa Ferreira Rodrigues Moreira.

Rio Paranaíba - MG

2024

1 RESUMO

Este projeto implementa uma solução interativa para a resolução de labirintos utilizando algoritmos clássicos de busca em grafos, como Busca em Largura (BFS), Busca em Profundidade (DFS) e A* (A-Estrela). A aplicação foi desenvolvida com o uso da biblioteca ‘pygame‘, permitindo a visualização dinâmica do comportamento de cada algoritmo, incluindo a expansão da fronteira e o caminho encontrado até o objetivo.

O algoritmo BFS explora todos os nós em cada nível antes de avançar para o próximo, enquanto o DFS segue um caminho até encontrar o objetivo ou precisar retroceder. Já o A*, considerado mais eficiente, combina o custo acumulado do caminho com uma heurística (distância Euclidiana) para priorizar a expansão dos nós mais promissores. Cada algoritmo foi implementado com estruturas de dados apropriadas, como filas, pilhas e a estrutura *heapq* para a fila de prioridade no A*.

O projeto também inclui uma interface gráfica interativa, permitindo ao usuário selecionar o algoritmo desejado e visualizar gradualmente as decisões tomadas durante a busca. Os resultados mostram diferenças significativas no comportamento e eficiência dos algoritmos em diferentes cenários, com o A* destacando-se por encontrar soluções mais rapidamente em situações complexas. Este projeto demonstra a aplicabilidade dos algoritmos de busca e oferece uma ferramenta didática para aprendizado e análise de Inteligência Artificial e resolução de problemas em grafos.

...

Sumário

2 INTRODUÇÃO

A resolução de labirintos é um problema clássico em computação e inteligência artificial (IA), amplamente utilizado para ilustrar conceitos fundamentais de algoritmos de busca e grafos. Este problema tem aplicações práticas em áreas como robótica, navegação em ambientes complexos, jogos e otimização de caminhos em redes. Através do desenvolvimento e análise de algoritmos de busca, é possível compreender como diferentes abordagens exploram e encontram soluções em cenários variados.

O objetivo deste projeto foi implementar três algoritmos de busca bem estabelecidos: Busca em Largura (BFS), Busca em Profundidade (DFS) e A* (A-Estrela), com foco na visualização de suas expansões de fronteira e no comportamento durante a busca pelo caminho em um labirinto. A BFS explora os nós do grafo camada por camada, garantindo a solução mais curta em grafos não ponderados, enquanto a DFS adota uma abordagem de exploração profunda, que pode não ser ótima em termos de custo. Por outro lado, o A* utiliza uma combinação de custo acumulado e heurística para priorizar a expansão de nós mais promissores, sendo amplamente utilizado em aplicações práticas que requerem eficiência.

A interface gráfica foi desenvolvida com a biblioteca *pygame*, permitindo a visualização interativa e dinâmica dos algoritmos em ação. O usuário pode selecionar o algoritmo desejado, observar a expansão da fronteira de busca em tempo real e visualizar o caminho final encontrado. Essa funcionalidade não apenas enriquece a experiência do usuário, mas também torna o projeto uma ferramenta didática para ensino e aprendizado de algoritmos de busca.

Este relatório apresenta o contexto e a motivação do projeto, os métodos utilizados para sua implementação e os resultados obtidos. São discutidas as diferenças no comportamento dos algoritmos, os desafios enfrentados durante o desenvolvimento, e as possíveis aplicações futuras. A análise destaca as vantagens do A* em situações mais complexas e a simplicidade das abordagens clássicas, como BFS e DFS, para cenários menos exigentes. Este trabalho demonstra a relevância dos algoritmos de busca em problemas computacionais e oferece uma base prática para sua aplicação e estudo.

...

3 MÉTODOS

3.1 Visão Geral

O projeto teve como objetivo implementar e comparar três algoritmos de busca (BFS, DFS e A*) aplicados à resolução de labirintos, destacando seus comportamentos e eficiência. Para isso, foi adotada uma abordagem modular, com o código dividido em componentes independentes que facilitaram o desenvolvimento, manutenção e testes.

A linguagem **Python** foi escolhida por sua simplicidade e pela disponibilidade de bibliotecas como a **pygame**, utilizada para criar uma interface gráfica interativa que permite a visualização em tempo real do labirinto, da expansão da fronteira e do caminho encontrado pelos algoritmos.

O projeto foi estruturado em três etapas: definição e implementação dos algoritmos, desenvolvimento da interface gráfica e integração com testes. Os principais módulos são:

- `agent.py`: Implementa os algoritmos de busca.
- `maze.py`: Define a estrutura do labirinto.
- `main.py`: Controla a interação do usuário e a exibição gráfica.
- `node.py`: Estrutura os nós com informações do estado, nó pai, ação e custo acumulado, essenciais para o cálculo da heurística no A*.

Essa arquitetura modular possibilitou um desenvolvimento organizado e a validação de cada parte do projeto de forma independente.

3.2 Descrição dos Algoritmos

3.2.1 Busca em Largura (BFS)

A Busca em Largura é um algoritmo de exploração sistemática que expande todos os nós em um nível antes de avançar para o próximo. Utiliza uma fila (*FIFO*) para gerenciar a fronteira, garantindo que os nós sejam processados na ordem em que são descobertos. Este algoritmo é completo e encontra a solução mais curta em grafos não ponderados, mas pode

ser ineficiente em termos de memória em labirintos grandes devido ao armazenamento de muitos nós na fronteira.

3.2.2 Busca em Profundidade (DFS)

A Busca em Profundidade explora cada caminho até o seu final antes de retroceder e buscar outros caminhos. Ela utiliza uma pilha (*LIFO*) para gerenciar a fronteira, priorizando a profundidade em vez da largura. Apesar de consumir menos memória do que a BFS, a DFS pode não encontrar o caminho mais curto e, em certos casos, pode entrar em ciclos infinitos se não forem tratados os estados visitados.

3.2.3 Busca A-Estrela (A*)

O algoritmo A* combina o custo acumulado do caminho (função $g(n)$) com uma heurística que estima a distância até o objetivo (função $h(n)$). Ele utiliza uma fila de prioridade (*heapq*) para sempre expandir o nó com o menor valor da função de custo total $f(n) = g(n) + h(n)$. A heurística empregada neste projeto foi a distância Euclidiana, que prioriza caminhos diretos em grids. O A* é eficiente e encontra o caminho mais curto, mas sua performance depende da qualidade da heurística utilizada.

3.3 Visualização

Para a visualização do labirinto e do comportamento dos algoritmos, foi utilizada a biblioteca **pygame**, que simplifica o desenvolvimento de aplicações gráficas interativas. A escolha pela **pygame** foi motivada por sua facilidade de uso, ampla documentação e integração nativa com o Python, permitindo um desenvolvimento incremental e eficiente.

A representação do labirinto foi realizada em uma grade 2D, onde cada célula é desenhada como um retângulo. As cores utilizadas destacam os diferentes estados do labirinto:

- **Cinza**: Paredes, indicando células intransitáveis.
- **Branco**: Caminhos possíveis no labirinto.
- **Azul**: Fronteira, representando os nós sendo processados pelo algoritmo.

A biblioteca oferece funções como `draw.rect()` e `display.flip()` para manipulação de gráficos em 2D e controle de eventos. Essa abordagem possibilitou a visualização dinâmica e em tempo real da execução dos algoritmos, transformando o projeto em uma ferramenta didática e interativa.

3.4 Heursística A*

No algoritmo A*, a heurística desempenha um papel fundamental na priorização dos nós a serem explorados, guiando a busca em direção ao objetivo de forma eficiente. A heurística escolhida neste projeto foi a **distância Euclidiana**, que estima o custo restante para atingir o objetivo assumindo um caminho direto e sem obstáculos.

3.4.1 Fórmula da Distância Euclidiana

A distância Euclidiana entre dois pontos (x_1, y_1) e (x_2, y_2) é dada por:

$$h(n) = \sqrt{(x_{\text{objetivo}} - x_{\text{atual}})^2 + (y_{\text{objetivo}} - y_{\text{atual}})^2}$$

Essa fórmula calcula a distância em linha reta entre o nó atual n e o nó objetivo, considerando um grid 2D.

3.4.2 Por Que Escolher a Distância Euclidiana?

- **Aproximação Realista**: Em grids, a distância Euclidiana fornece uma boa estimativa do custo restante quando o movimento é permitido em todas as direções (diagonal, horizontal e vertical).
- **Eficiência**: A heurística é rápida de calcular, exigindo apenas operações básicas de subtração, elevação ao quadrado e raiz quadrada.
- **Admissibilidade**: A distância Euclidiana nunca superestima o custo real do caminho, o que garante que o A* encontre o caminho ótimo.
- **Consistência**: A heurística é consistente (ou monotônica), pois satisfaz a propriedade $h(n) \leq d(n, n') + h(n')$, onde $d(n, n')$ é o custo do movimento entre os nós n e n'.

3.4.3 Integração no Algoritmo

No A*, a heurística é combinada com o custo acumulado $g(n)$ para calcular a função de custo total $f(n)$, definida como:

$$f(n) = g(n) + h(n)$$

Essa abordagem prioriza os nós que têm menor custo estimado para atingir o objetivo, resultando em um balanceamento entre exploração (analisar novos nós) e eficiência (seguir o caminho mais curto).

3.5 Estrutura do Código

O projeto foi desenvolvido de forma modular, com o objetivo de organizar o código em componentes independentes que facilitam a manutenção, compreensão e expansão. A seguir, é apresentada uma visão geral dos principais módulos:

3.5.1 Módulo `main.py`

O arquivo principal do projeto é responsável pela integração dos componentes e pela interface gráfica utilizando a biblioteca **pygame**. Ele desempenha as seguintes funções:

- Gerenciar a escolha do algoritmo (BFS, DFS ou A*).
- Iniciar a interface gráfica e desenhar o labirinto na tela.
- Atualizar a exibição do labirinto e a expansão da fronteira em tempo real.
- Permitir a interação do usuário e finalizar o programa de forma controlada.

3.5.2 Módulo `agent.py`

Este módulo implementa os algoritmos de busca:

- **Busca em Largura (BFS)**: Explora os nós por camada, utilizando uma fila.
- **Busca em Profundidade (DFS)**: Explora caminhos profundos antes de retroceder, utilizando uma pilha.

- **A* (A-Estrela)**: Utiliza a heurística Euclidiana para priorizar nós promissores, utilizando uma fila de prioridade (`heapq`).

Cada algoritmo inclui:

- Expansão dos nós, respeitando a ordem predefinida (cima, esquerda, direita, baixo).
- Função para exibir a fronteira em tempo real.
- Reconstrução do caminho até o objetivo.

3.5.3 Módulo `maze.py`

Este módulo define a estrutura do labirinto. Ele utiliza uma matriz 2D onde:

- `1` representa caminhos transitáveis.
- `0` representa paredes ou obstáculos.

Funções principais:

- `is_valid_position`: Verifica se uma posição no labirinto é válida para expansão.

3.5.4 Módulo `node.py`

Define a estrutura de um nó utilizado pelos algoritmos de busca. Cada nó armazena:

- Estado atual (posição no labirinto).
- Referência ao nó pai (para reconstrução do caminho).
- Ação que levou ao estado atual.
- Custo acumulado desde o estado inicial.

A classe também implementa comparações para uso na fila de prioridade do A*.

Essa organização modular permite que cada componente seja desenvolvido e testado de forma independente, facilitando a expansão do projeto, como a inclusão de novos algoritmos ou melhorias na interface gráfica.

3.6 Testes

Os testes foram realizados utilizando o labirinto padrão fornecido no trabalho, que possui um tamanho limitado e um caminho definido entre o ponto inicial e o objetivo. O objetivo foi avaliar a eficiência e o comportamento dos algoritmos de busca implementados (BFS, DFS e A*) na resolução do labirinto, com base no número de passos tomados para alcançar a solução.

Todos os algoritmos conseguiram encontrar o caminho até o objetivo, no entanto, apresentaram diferenças no número de passos necessários para a solução. A seguir, estão os resultados e observações sobre o desempenho:

- **Busca em Largura (BFS)**: Foi o algoritmo menos eficiente em termos de passos tomados. Apesar de garantir a solução mais curta, a BFS expande muitos nós devido à sua estratégia de explorar todos os caminhos possíveis por camada antes de avançar para níveis mais profundos. Isso resultou em um maior número de passos no cenário testado.

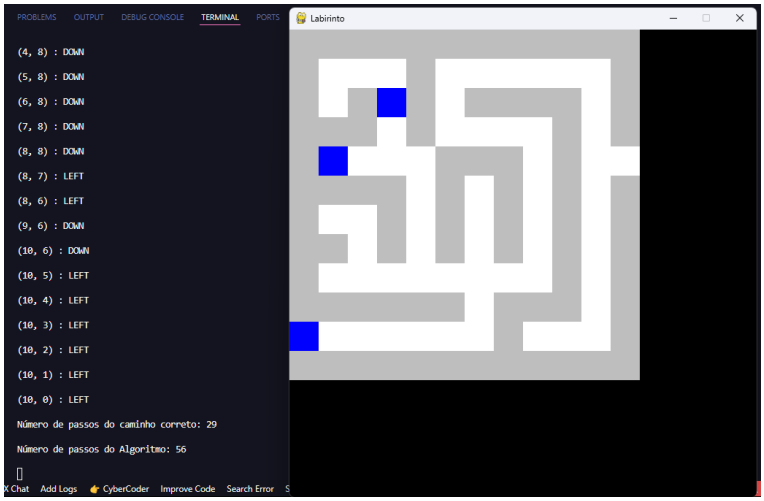

Figura 1: Imagem algoritmo BFS em ação com número de passos

- **Busca A* (A-Estrela)**: Apresentou desempenho intermediário. Ao utilizar a heurística Euclidiana, o A* conseguiu priorizar caminhos mais promissores, reduzindo a quantidade de nós expandidos em relação à BFS. No entanto, como o labirinto testado era pequeno e possuía um caminho relativamente direto, o impacto da heurística foi limitado.

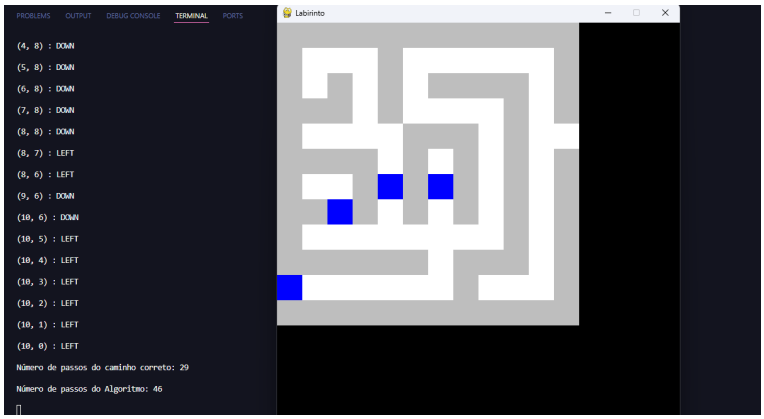

Figura 2: Imagem algoritmo A*(heuristica euclidiana) em ação com número de passos

- **Busca em Profundidade (DFS)**: Foi o algoritmo mais eficiente no contexto do labirinto fornecido, tomando o menor número de passos. Sua estratégia de explorar profundamente cada caminho antes de retroceder favoreceu a solução direta, dado o formato e a simplicidade do labirinto.

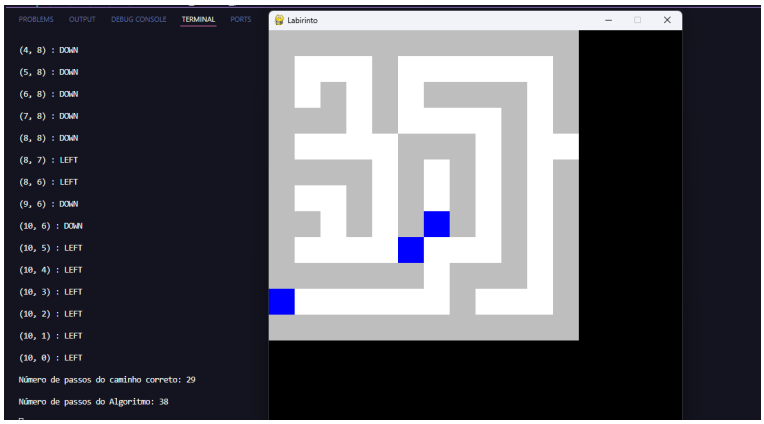

Figura 3: Imagem algoritmo DFS em ação com número de passos

É importante destacar que, embora a DFS tenha sido mais eficiente neste teste específico, ela não garante o caminho mais curto e pode não ser tão eficaz em labirintos maiores ou mais complexos. Nesses casos, o A* tende a se destacar, pois combina o custo acumulado com a estimativa heurística, priorizando caminhos mais promissores e otimizando a busca.

Esses resultados demonstram como o tamanho e a complexidade do labirinto influenciam o desempenho dos algoritmos, ressaltando a importância de selecionar a abordagem mais adequada ao problema em questão.

4 CONCLUSÃO

O projeto alcançou com sucesso o objetivo de implementar e comparar diferentes algoritmos de busca aplicados à resolução de labirintos, destacando seus comportamentos e eficiência. A utilização de uma abordagem modular permitiu uma estrutura organizada, facilitando o desenvolvimento e a validação independente de cada componente. A integração com a biblioteca **pygame** possibilitou a criação de uma interface gráfica interativa, que foi essencial para a visualização em tempo real da execução dos algoritmos, tornando o projeto uma ferramenta didática.

Os testes realizados demonstraram que todos os algoritmos foram capazes de encontrar a solução no labirinto fornecido, mas com diferenças no número de passos tomados. A **Busca em Largura (BFS)** apresentou maior número de expansões, enquanto a **Busca em Profundidade (DFS)** foi mais eficiente no cenário testado. Já o **A***, ao utilizar a heurística Euclidiana, mostrou-se promissor para labirintos maiores e mais complexos, onde sua capacidade de priorizar caminhos mais curtos se destaca.

Este trabalho reforça a importância da escolha do algoritmo de busca de acordo com o problema em questão, evidenciando as vantagens e limitações de cada abordagem. Além disso, a implementação gráfica proporcionou uma experiência prática para o aprendizado de conceitos fundamentais de Inteligência Artificial e busca em grafos.