UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CIÊNCIA DA COMPUTAÇÃO DISCIPLINA DE INTELIGÊNCIA ARTIFICIAL

Projeto 2 - Algoritmos de Busca

Relatório Técnico

Leticia Zanellatto de Oliveira

Raquel Silvério Lopes

Foz do Iguaçu

2025

1 Introdução

O problema abordado neste trabalho é inspirado no mito do labirinto do Minotauro. Teseu deve atravessar um labirinto até o centro, onde enfrentará a criatura, e conta com a ajuda de Ariadne, que lhe fornece um fio para garantir o retorno. Como o comprimento desse fio é limitado, qualquer rota ineficiente compromete o sucesso da missão: é essencial percorrer o caminho de menor custo até o objetivo. Essa narrativa serve aqui como metáfora clara para o problema computacional de busca de caminhos em grafos com restrição de recursos, no qual a eficiência do método de busca impacta diretamente a viabilidade da solução.

Para transformar essa história em um problema resolvível por computador, modelamos o labirinto como um *grafo dirigido* em que cada vértice representa um ponto do labirinto e cada aresta representa um deslocamento permitido, associado a um custo (por exemplo, distância ou tempo). Este relatório técnico descreve a implementação de dois algoritmos de busca de caminho em grafos com o objetivo de encontrar o percurso de menor custo entre um ponto de origem e um ponto de destino. O trabalho abrange a definição das estruturas de dados do grafo, a implementação de ambos os algoritmos em C++ e a definição de métricas de desempenho que permitam uma análise comparativa objetiva.

Como extensão opcional (bônus), incorporamos explicitamente a limitação do fio: o usuário informa o comprimento máximo e o algoritmo não explora caminhos cujo custo acumulado ultrapasse esse valor, reportando em cada iteração o "fio restante". Essa variação evidencia o efeito da restrição de recursos sobre a estratégia de busca, bem como a importância de heurísticas que conduzam a soluções viáveis dentro do limite imposto.

1.1 Objetivo do Projeto

O projeto consiste em implementar um sistema que:

1. Lê um arquivo texto com a especificação do grafo, contendo o ponto inicial, o ponto final, a indicação se o grafo é orientado ou não, as arestas com seus custos e a heurística.
2. Permite escolher qual algoritmo executar entre as opções disponíveis no programa.
3. Executa o algoritmo selecionado mostrando a busca de forma iterativa. Em cada iteração, exibe o estado atual da estrutura de controle e uma medida de desempenho definida para o trabalho.
4. Ao final, apresenta um resumo com a distância total do caminho encontrado, o caminho reconstruído e o valor total da medida de desempenho.

2 Algoritmos Escolhidos

2.1 Melhor Solução

O algoritmo A*, também conhecido como *A-Star*, é um método de busca informada que avalia os nós combinando dois custos: $g(n)$, o custo já incorrido para alcançar o nó n a partir do nó inicial, e $h(n)$, uma estimativa do custo restante para alcançar o objetivo a partir de n. A avaliação de cada nó é feita pela função

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

na qual $f(n)$ representa o custo estimado do melhor caminho que passa por n. Quando a heurística $h(n)$ é admissível (isto é, nunca superestima o custo real até o objetivo) e os pesos de aresta são não negativos, o A* é completo e encontra um caminho de custo ótimo. Em cenários comuns, adotar heurísticas consistentes (monotônicas) evita reexpansões desnecessárias e melhora ainda mais a eficiência da busca. Dessa forma, o A* combina orientação a objetivos com economia de esforço de exploração, reduzindo o número de nós expandidos sem abrir mão da qualidade da solução (Russell e Norvig 2016).

2.1.1 Justificativa da Escolha do A*

O A* foi selecionado como abordagem para a melhor solução de busca por conciliar eficiência e garantia de otimalidade nas condições do problema. Ao incorporar a heurística $h(n)$ à avaliação, o algoritmo realiza uma exploração direcionada do espaço de estados, priorizando nós mais promissores conforme o equilíbrio entre custo acumulado $g(n)$ e custo estimado $h(n)$. Na prática, isso se traduz em menos expansões irrelevantes, menor consumo de memória e tempo de execução mais favorável quando comparado a buscas cegas. No contexto deste trabalho, em que é importante alcançar o destino pelo menor custo possível (e, na variação com restrição de recurso, respeitando o comprimento máximo do "fio"), o A* é particularmente adequado: ele tende a convergir rapidamente para rotas de baixo custo, preservando as propriedades de completude e otimalidade sempre que a heurística adotada for admissível.

2.2 Pior Solução

Neste trabalho, analisamos algoritmos clássicos de busca em grafos para estabelecer uma referência de desempenho desfavorável ao problema de caminho mínimo. Entre as abordagens de *busca não informada* discutidas na literatura (por exemplo, busca em largura e busca em

profundidade), adotamos a Busca em Profundidade (DFS) como linha de base "pior" para comparação, no sentido de que ela não utiliza informação de custo nem heurística e, portanto, não tem garantia de retornar o caminho de *menor custo* em grafos ponderados (Cormen et al. 2012, Russell e Norvig 2016, Lee, Baranauskas e Souto 2024). Essa escolha evidencia, por contraste, os ganhos obtidos por uma busca informada como o A*.

2.3 Busca em Profundidade

A DFS explora o grafo avançando o mais fundo possível a partir do nó inicial e, quando não há como prosseguir, retrocede para tentar outros caminhos. Pode ser implementada de forma recursiva ou com uma pilha explícita. Em termos assintóticos, seu custo por visita é baixo; o tempo de execução é proporcional ao número de vértices e arestas efetivamente explorados, e o uso de memória é proporcional à profundidade da busca (pilha de chamadas/estrutura auxiliar) (Cormen et al. 2012).

Para o problema de caminho mínimo com pesos, entretanto, a DFS não leva em conta os custos das arestas nem possui função de avaliação que priorize estados promissores. Como consequência, o caminho retornado pode ser substancialmente mais caro do que o ótimo, além de a ordem de expansão dos vizinhos influenciar fortemente o resultado. Em grafos dirigidos e finitos com marcação de visitados, a DFS é completa no sentido de encontrar *algum* caminho, se existir, mas não garante *otimalidade do custo* (Russell e Norvig 2016, Lee, Baranauskas e Souto 2024).

2.3.1 Justificativa da Escolha da DFS

A adoção da DFS como pior solução fundamenta-se em aspectos conhecidos na literatura para problemas de caminho mínimo ponderado:

1. **Ausência de informação de custo**: por ser uma busca cega, a estratégia não utiliza heurística nem custo acumulado para orientar a expansão, o que aumenta a chance de seguir rotas longas e pouco eficientes (Russell e Norvig 2016).

2. **Dependência da ordem de vizinhos**: o caminho encontrado varia conforme a ordem em que os sucessores são visitados; escolhas iniciais desfavoráveis podem atrasar a descoberta de rotas mais curtas (Cormen et al. 2012).

3. **Não otimalidade em grafos ponderados**: mesmo quando encontra um caminho, a DFS não oferece garantias sobre o custo mínimo; frequentemente retorna soluções subótimas (Lee, Baranauskas e Souto 2024).

2.3.2 Decisão de projeto: por que não usar a variante sem backtracking

Não implementamos a variação de DFS *sem backtracking*. Tal variante não cumpriria o propósito fundamental da busca em profundidade, cujo objetivo é encontrar uma solução (quando ela existe) ou, no limite, explorar sistematicamente o espaço de estados — o que requer a capacidade de *retroceder* para tentar outras vias quando a primeira escolha leva a um beco sem saída. Sem retrocesso, a busca segue apenas uma trilha determinada pela ordem dos vizinhos e pode declarar falha mesmo havendo caminho no grafo, além de não oferecer qualquer cobertura sobre o restante do espaço.

Adicionalmente, como o arquivo de entrada descreve custos nas arestas, adaptamos a DFS para *acumular o custo do caminho* durante a exploração: a cada aresta percorrida, o valor do custo é somado, e a distância reportada ao final corresponde à soma dos pesos do caminho encontrado. Essa adaptação não transforma a DFS em um método ótimo para caminho mínimo (a ordem de expansão continua sem considerar custos), mas garante que os resultados apresentados sejam coerentes com a modelagem ponderada do problema e *comparáveis* aos do A* no que diz respeito ao valor de custo efetivamente percorrido.

3 Definições e Escolhas de Implementação

3.1 Organização do Código e Bibliotecas

O projeto está organizado em dois diretórios. Em `header` ficam as interfaces das classes e os protótipos de função. Em `src` ficam as implementações. Essa separação facilita a leitura, a compilação incremental e os testes.

As bibliotecas centrais para o carregamento dos dados e a representação do problema são `arquivo.h` e `grafo.h`. Os algoritmos ficam em bibliotecas próprias, `a_estrela.h` e `alg_dfs.h`.

3.2 Leitura e configuração (`arquivo.h`)

O arquivo texto descreve a instância do problema por meio de predicados simples:

- `ponto_inicial(a0).` e `ponto_final(f0).`
- `orientado(s).` ou `orientado(n).`
- `pode_ir(u,v,c).` indicando aresta dirigida de `u` para `v` com custo `c`.
- `h(n,goal,v).` fornecendo a heurística usada pelo A*, associada ao par `(n, goal)`.

Nesse cenário, a biblioteca `arquivo.h` expõe a classe `Arquivo`, responsável por ler o arquivo de entrada, armazenar os pontos inicial e final e popular um objeto `Grafo` com orientação, arestas e heurísticas. O método `ler_arquivo(...)` recebe o caminho do arquivo e referências para `Grafo`, `Aresta` e `Heuristica`; ao término, o grafo está configurado para execução dos algoritmos. A classe mantém os campos `ponto_inical` e `ponto_final` com seus respectivos acessores.

3.3 Modelo de dados do grafo (`grafo.h`)

A biblioteca `grafo.h` define três classes.

- `Aresta`: guarda `lig_inicio`, `lig_fim` e `lig_custo` e oferece getters e setters.
- `Heuristica`: guarda `h_inicio`, `h_fim` e `h_heuristica` para representar $h(n, \text{goal})$.
- `Grafo`: mantém um vetor de `Aresta`, um vetor de `Heuristica` e um `bool` indicando se é orientado. Expõe `add_aresta(...)` e `add_heuristica(...)` para inserção, além de `get_aresta()`, `get_heuristicas()` e `get_orientado()`.

3.4 Linguagem e Ferramentas

A implementação é em C++ utilizando apenas bibliotecas padrão (STL): `vector`, `string`, `priority_queue`, `unordered_map` e `iomanip`/`iostream`. O código-fonte está organizado em diretórios `header/` (arquivos de interface) e `src/` (implementações), com um `main.cpp` que apresenta a interface de linha de comando e um arquivo de entrada `teste.txt` com a especificação dos grafos; o repositório também inclui um resumo em PDF e um arquivo de comandos para execução.

4 Medidas de Avaliação de Desempenho

Os algoritmos candidatos foram comparados em relação às seguintes medidas de desempenho: **Tempo de Execução** e **Nós Gerados**.

- **Tempo de Execução**: escolhido porque, no contexto do labirinto, a rapidez para encontrar a solução é crucial, especialmente quando há restrições (por exemplo, o comprimento máximo do "fio" no bônus). O tempo captura o custo real de manipular as estruturas de dados e o impacto de detalhes de implementação (ordenação da fronteira, verificação

de consistência etc.). Contudo, por ser uma medida *relativa* — dependente de hardware, sistema operacional, compilador e carga de máquina — ela pode variar entre ambientes distintos.

- **Nós Gerados**: adotada como segunda métrica por ser *independente do ambiente* e quantificar diretamente o trabalho potencial da busca. Define-se como a quantidade de nós *colocados* na estrutura de controle (inserções na *priority_queue* do A* ou na pilha/fila da DFS). No modo bônus, sucessores cujo custo acumulado ultrapassa o comprimento do fio *não* são gerados e, portanto, não contam. Essa medida evidencia a qualidade da heurística (heurísticas mais informativas tendem a gerar menos nós), correlaciona-se com o consumo de memória da fronteira e permite comparação justa entre A* e DFS. Neste trabalho, contabilizamos *cada* inserção na estrutura (inclusive re-inserções com custo melhor), pois isso captura o custo real de manutenção da fronteira.

4.1 Análise

Tabela 1: A* — notebook Dell

Cenário	Custo de Tempo (s)	Nós Gerados	Distância
Arq 1, grafo orientado	0.021	7	132
Arq 1, grafo não orientado	0.029	10	110
Arq 2, grafo orientado	0.013	6	9
Arq 2, grafo não orientado	0.014	6	9

Tabela 2: A* — notebook Lenovo

Cenário	Custo de Tempo (s)	Nós Gerados	Distância
Arq 1, grafo orientado	0.028	7	132
Arq 1, grafo não orientado	0.040	10	110
Arq 2, grafo orientado	0.016	6	9
Arq 2, grafo não orientado	0.019	6	9

Tabela 3: DFS — notebook Dell

Cenário	Custo de Tempo (s)	Nós Gerados	Distância
Arq 1, grafo orientado	0.012	5	139
Arq 1, grafo não orientado	0.012	6	221
Arq 2, grafo orientado	0.009	4	13
Arq 2, grafo não orientado	0.009	6	17

Foi feito a comparação entre notebooks (Dell G15 — CPU Intel Core i7-12700H 2,30 GHz, 16 GB RAM, RTX 3060 (mais potente) — vs. Lenovo — CPU AMD Ryzen 7 5700U, 16 GB

Tabela 4: DFS — notebook Lenovo

Cenário	Custo de Tempo (s)	Nós Gerados	Distância
Arq 1, grafo orientado	0.009	5	139
Arq 1, grafo não orientado	0.011	6	221
Arq 2, grafo orientado	0.012	4	13
Arq 2, grafo não orientado	0.011	6	17

RAM). Nos experimentos com A* (Tabelas 1 e 2), o notebook da Dell apresentou tempos menores em todos os cenários. Isso é coerente com as características do A*: cada iteração envolve operações de maior custo assintótico e constante (remoções/inserções em *priority_queue* com complexidade $O(\log n)$, atualizações em mapas, cálculo de $f = g + h$ e reconstrução de caminho). Esses componentes se beneficiam de CPU mais rápida, caches maiores e melhor largura de banda de memória. Já na DFS (Tabelas 3 e 4), as diferenças entre notebooks são pequenas: a DFS executa um fluxo simples com empilhamento/recursão e marcação de visitados; o custo por passo é baixo e a pressão de memória é reduzida. Como os tempos absolutos estão na casa de milissegundos, variações do ambiente (agendador do SO, processos de fundo, turbo/temperatura) podem ser do mesmo tamanho da diferença entre máquinas, diminuindo o ganho visível do notebook mais potente.

Porém nessa comparação de tempo, percebe-se que o tempo de da DFS é menor, isso occorre pois a DFS percorre o grafo aprofundando-se e retrocedendo quando necessário; o trabalho cresce aproximadamente de forma linear com a quantidade de vértices/arestas efetivamente explorados nessa busca. Em contraste, o A* mantém uma fronteira ordenada por $f(n)$ e considera alternativas para preservar otimalidade. Quando a heurística não é muito informativa (ou é enganosa em parte do grafo), o A* pode gerar mais estados e gastar tempo reordenando a fila de prioridade.¨

Nossa métrica "Nós Gerados" conta cada inserção na estrutura de controle. O A* insere *todos* os sucessores elegíveis do nó expandido e pode *re-inserir* um mesmo vértice quando encontra custo melhor, refletindo o custo real de manter alternativas. A DFS gera no máximo um sucessor por passo e marca visitados, evitando regerações. Assim, é esperado observar menos nós gerados na DFS. Importante: gerar menos, nesse caso, não significa melhor qualidade — significa apenas que a DFS não constrói nem mantém alternativas informadas por custo; por isso, pode produzir rotas mais caras do que o ótimo.

Comparando as quatro tabelas, observa-se que, em tempo e em número de nós gerados, a DFS apresenta resultados melhores do que o A*. Contudo, frente ao objetivo do trabalho — encontrar o menor caminho — o A* foi superior em todos os cenários. Portanto, mesmo quando a DFS é mais rápida e gera menos nós, o A* permanece a melhor escolha por atender ao requisito central de *otimalidade do caminho*.

5 Conclusão

Este trabalho modelou o problema do labirinto como busca de caminho em grafo e implementou duas abordagens vistas em sala: o A* como solução informada e a DFS como linha de base não informada. O sistema em C++ lê a instância a partir de arquivo, respeita a orientação do grafo, executa os algoritmos e relata a execução por iteração, incluindo as métricas de desempenho escolhidas.

Em síntese, o embasamento teórico, estudado e discutido em sala, fundamenta a escolha do A* como abordagem recomendada para o problema, por combinar eficiência com garantia de otimalidade nas condições estabelecidas. A DFS cumpre o papel de referência negativa: apresenta tempos menores nas instâncias pequenas avaliadas, mas, por não considerar custos nem heurística, não garante o menor caminho e frequentemente produz soluções subótimas. Assim, os resultados experimentais corroboram a adoção do A* quando a qualidade da solução — isto é, a minimização do custo do caminho — é o critério determinante.

7 Referências Bibliográficas

Cormen et al. 2012 CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 3. ed. Rio de Janeiro: Elsevier Editora Ltda, 2012. 944 p. Disponível em: <https://computerscience360.wordpress.com/wp-content/uploads/2018/02/algoritmos-teoria-eprc3a1tica-3ed-thomas-cormen.pdf>.

Lee, Baranauskas e Souto 2024 LEE, H. D.; BARANAUSKAS, J. A.; SOUTO, M. *Estratégias de Busca Não Informada (Exaustiva ou Cega)*. Foz do Iguaçu, 2024. 59 slides, color.

Russell e Norvig 2016 RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. Pearson, 2016. Disponível em: <https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf>.