# Trabalho Prático I - Introdução à Inteligência Artificial

Luís Felipe Ramos Ferreira

lframos.lf@gmail.com

## 1 Introdução

O Trabalho Prático I da disciplina de Introdução a Inteligência Artificial teve como objetivo a implementação de 5 algoritmos de busca diferentes em um problema de menor caminho entre dois pontos em um mapa bidimensional.

## 2 Implementação

O projeto foi implementado utilizando C++ e Python. O código principal, que contêm a implementação dos 5 algoritmos citados foi escrito em C++, versão 17, e está todo contido no arquivo `main.cpp`. Arquivos utilitários utilizados para realização de *benchmarks* comparativos entre os algoritmos, criação dos gráficos para análise ds dados foram criados utilizando Python, versão 3.12.3, e os pacotes utilizados foram manejados com o gerenciador de pacotes pip.

Instruções de como executar o programa principal e os arquivos utilitários, assim como a listagem de dependências, estão presentes no arquivo `README.md`, no diretório raiz do projeto.

Os testes apresentados na análise de resultados foram feitos em uma máquina com Ubuntu 24.04.01 e 16GB de memória RAM, na CPU 11th Gen Intel i5-11400F (12) @ 4.400GHz.

## 3 Descrição dos algoritmos

Nesta seção, é feita uma breve descrição dos algoritmos utilizados e suas principais diferenças. Para todos eles, consideramos um *branch factor* $b$ e que a solução está no nível $d$.

- *Breadth First Search (BFS)* A busca em largura é um algoritmo em que, a cada iteração, se expande o nó mais raso ainda não expandido na busca. Em termos informais, primeiro se expande a raiz, depois os sucessores da raiz, depois os sucessores dos sucessores da raiz, e assim se segue. O algoritmo pode ser implementado com uma fila, pois a ideia dele segue um

arquitetura de FIFO (*First In First Out*). No algoritmo podemos fazer algo chamado *Early Goal Test*, em que checamos se um nó adicionado na fila já é o nó final antes de processá-lo ao sair da fila. Faz-se isso pois com isso o algoritmo não irá precisar adicionar mais nós à fila e nem processar os que já estão lá se o nó final já estiver pronto para ser processado.

  - `Completo`
  - `Ótimo`, se e somente se o custo seja uma função não decrescente da profundidade do nó (Por exemplo, na busca de menor caminho em um grafo sem pesos nas arestas.).
  - `Complexidade`: tempo ($\mathcal{O}(b^d)$) e espaço ($\mathcal{O}(b^d)$)

- *Iterative Depth Search (IDS)* A busca com profundidade iterativa é uma junção dos algoritmos BFS e DFS. Nele, aplicamos uma busca em profundidade iterativa, isto é, a cada iteração, aumentamos a profundidade permitida na busca. Nesse sentido, os benefícios da BFS e da DFS são combinado.

  - `Completo`.
  - `Ótimo`, considerando custo crescente como no caso da busca em largura.
  - `Complexidade`: tempo $\mathcal{O}(b^d)$ e espaço

- *Uniform Cost Search (UCS)*

  O algoritmo de busca de custo uniforme é muito semelhante à BFS, mas o próximo nó expandido é na verdade o nó com menor custo até o momento. Para fazer isso, é necessário manter uma ordenação dos custos dos nós, e pra isso podemos utilizar uma fila de prioridades ou um *heap* na implementação.

  - `Completo`, se e somente se cada passo do algoritmo tem custo positivo ( Considerando um grafo com pesos nas arestas, se houver uma aresta com peso negativo, o algoritmo não funcionaria).
  - `Ótimo`, uma vez que seguimos o menor custo
  - `Complexidade`: se $C^*$ for a solução ótima, o pior caso de tempo e espaço é $\mathcal{O}(b^{1+\frac{C^*}{\epsilon}})$.

- *Greedy*
- A$^*$

# 4 Heurísticas utilizadas

Nesta seção são apresentadas as heurísticas utilizadas nos algoritmos *Greedy* e A$^*$.

- Distância *Manhattan*
- algm outras

# 5 Resultados

## 5.1 Número de estados expandidos

## 5.2 Tempo de execução

# 6 Discussão dos resultados

blabla