

# Trabalho Prático I - 2024/2

Igor Lacerda Faria da Silva

## 1. Introdução

O Trabalho Prático I de Introdução à Inteligência Artificial consistiu na implementação (em C++) de 5 algoritmos para cálculo de distância (exploração de espaço de busca) em mapas de um jogo simples. Os 5 algoritmos foram: *Breadth First Search* (BFS), *Iterative Deepening Search* (IDS), *Uniform Cost Search* (UCS), Busca Gulosa (*Greedy*) e A\* (A-Star ou A-Estrela). A função heurística escolhida para os algoritmos que dependem dela (*Greedy* e A\*) foi a **distância de Manhattan**. O repositório deste trabalho pode ser encontrado neste [link](#).

A documentação está dividida da seguinte maneira: esta breve introdução descreve o que foi realizado, a Seção 2 descreve a modelagem do programa, entrando nos detalhes dos algoritmos e estruturas de dados; a Seção 3, por sua vez, foca na heurística utilizada e, por fim, a Seção 4 é uma análise experimental dos diversos algoritmos implementados.

## 2. Algoritmos, Estruturas de Dados e Modelagem

Sem dúvidas, a principal estrutura de dados do trabalho foi uma matriz de double, com os pesos das posições do mapa. Outras estruturas são específicas dos algoritmos: a maioria dos algoritmos usa um *heap* (*priority\_queue* do C++) para controlar qual será o próximo estado da busca, exceto pela BFS, que usa uma fila (*queue*) e a IDS, que usa uma pilha (*stack*). No mais, as estruturas restantes são estruturas padrões de C++, como vetores (*vector*) e pares ordenados (*pairs*).

### 2.1. Formulação do Problema

Os 5 componentes da formulação do problema foram:

1. **Estados:** posições na matriz, com os pesos associados
2. **Estado Inicial:** posição inicial, da entrada
3. **Sequência de Ações:** subir, descer, ir para esquerda ou para direita
4. **Teste do Objetivo:** posição final atingida
5. **Custo da Solução:** soma dos custos pelo qual a busca passou por

A função sucessora recebe uma das ações e faz o movimento da posição atual para a nova posição correspondente.

### 2.2. Algoritmos

A seguir, uma breve descrição das principais diferenças entre os algoritmos, baseada na descrição de cada um. A BFS é usada como base e os outros algoritmos são vistos como “incrementos”.

#### 2.2.1. BFS

A BFS é um algoritmo relativamente simples. Dado um nó inicial, sua vizinhança é colocada na fila (primeiro a entrar é o primeiro a sair) e os nós são visitados na ordem em que foram inseridos, recursivamente. Obviamente, nós inválidos ou já visitados são excluídos da busca.

### 2.2.2. IDS

A IDS é uma DFS aplicada repetida diversas vezes, para profundidades cada vez maiores, até que o objetivo seja alcançado, ou o limite de profundidade seja atingido. A DFS é uma busca muito semelhante à BFS, a única diferença é que é usada uma pilha para se gerenciar os nós (ou seja, o primeiro a entrar é o último a sair). Além disso, como a busca da IDS tem a profundidade limitada, é necessário checar se a profundidade máxima daquela iteração foi atingida.

### 2.2.3. UCS

A UCS é também muito similar aos algoritmos anteriores. A diferença aqui é que usado um *heap*, que prioriza o nó de menor custo até então. Para que isso possa ser feito de maneira eficiente, é importante manter em memória uma matriz com o custo de cada posição. É possível visitar o mesmo nó mais de uma vez.

### 2.2.4. Greedy

A busca gulosa é semelhante à UCS, pois também usa um *heap* (apesar de não precisar de matriz de custos). A ideia é seguir pelo “melhor” caminho, mas usando uma aproximação definida por uma heurística (mais detalhes na Seção 3). Na busca gulosa, é explorado o “melhor neste momento”, em detrimento do “melhor geral”.

### 2.2.5. A\*

Por fim, o A-Estrela é uma combinação da busca gulosa com o UCS, que considera a soma dos custos gerados nestes algoritmos. A ideia é fazer uma busca que sempre é ótima, como no UCS, mas que é rápida porque é guiada pela heurística.

## 2.3. Outras decisões de implementação

Presume-se que sempre existe um caminho de A (início) para B (objetivo). Caso algum dos nós tenha custo infinito, o programa é encerrado. Na implementação da IDS, a profundidade máxima foi definida como  $W * H$ .

## 3. Heurísticas

A heurística escolhida foi a distancia de Manhattan. Ela é bem simples, pois é a soma da diferença absoluta das coordenadas. É por causa dessa simplicidade que ela é admissível: como o custo mínimo (por nó) é 1, o melhor custo é, na situação ideal, é justamente a distancia de Manhattan: deslocar-se  $|x_i - x_f|$  unidades na horizontal e  $|y_i - y_f|$  na vertical. É claro que esse é o cenário ideal, em que não existem impedimentos no meio do caminho e existe um caminho composto exclusivamente por nós de valor 1. De qualquer modo, a heurística nunca é mais cara do que o custo real para o objetivo e, portanto, pode-se dizer que ela é admissível.

## 4. Análise Quantitativa e Discussão

Os 2 mapas (*floresta.map* e *cidade.map*) de exemplo disponibilizados via *Moodle* foram usados para se realizar um *benchmarking* dos algoritmos. De antemão, vale ressaltar que a ordem dos operadores é um fator muito importante no tempo de execução, especialmente da IDS. Com isso em mente, as comparações vão incluir os casos simétricos – ou seja, invertendo o nó de início e o nó objetivo. Além disso, na execução dos experimentos, a -03, de otimização do compilador, foi habilitada.

## 4.1. Tempo de Execução

Para avaliar o tempo de execução, foram propostas 5 pares de coordenadas em cada mapa. Como são dois mapas e a simetria é levada em conta, foram realizados 20 execuções por mapa.

### 4.1.1. Cidade

Como este mapa possui menos estados inválidos, o início do mapa foi fixado em  $(0, 0)$  e, os destinos, foram da forma  $(2^i - 1, 2^i - 1)$ , com  $i$  variando de 4 até o limite do mapa, que gera a coordenada  $(255, 255)$ . Para estes casos, os algoritmos *Greedy* e *BFS* são extremamente rápidos, sendo que, boa parte do tempo de execução está associada a algum *overhead* e não ao tempo de processamento dos estados propriamente ditos. A segunda “classe” de algoritmos é composta pelos algoritmos UCS e  $A^*$ : na teoria, o  $A^*$  deveria ser mais rápido, mas para os casos considerados nesta seção, não houve diferença significativa: ambos ficaram na faixa de dezenas de milissegundos para a maior instância.

O único algoritmo que apresentou um comportamento interessante foi o IDS. Na Figura 1, o caso base começa em  $(0, 0)$  e é até bem comportado, mas vale destacar que ele demorou muito mais do que os outros algoritmos: para a maior instância, seu tempo de execução fica próximo de 1 segundo. O caso em que a execução termina em  $(0, 0)$  é ainda mais intrigante, pois, além de demorar (ainda) mais, o caso mais demorado não é o mais distante, é o “intermediário”, com cerca de 5s. O porquê disso será melhor discutido na Seção 4.2.

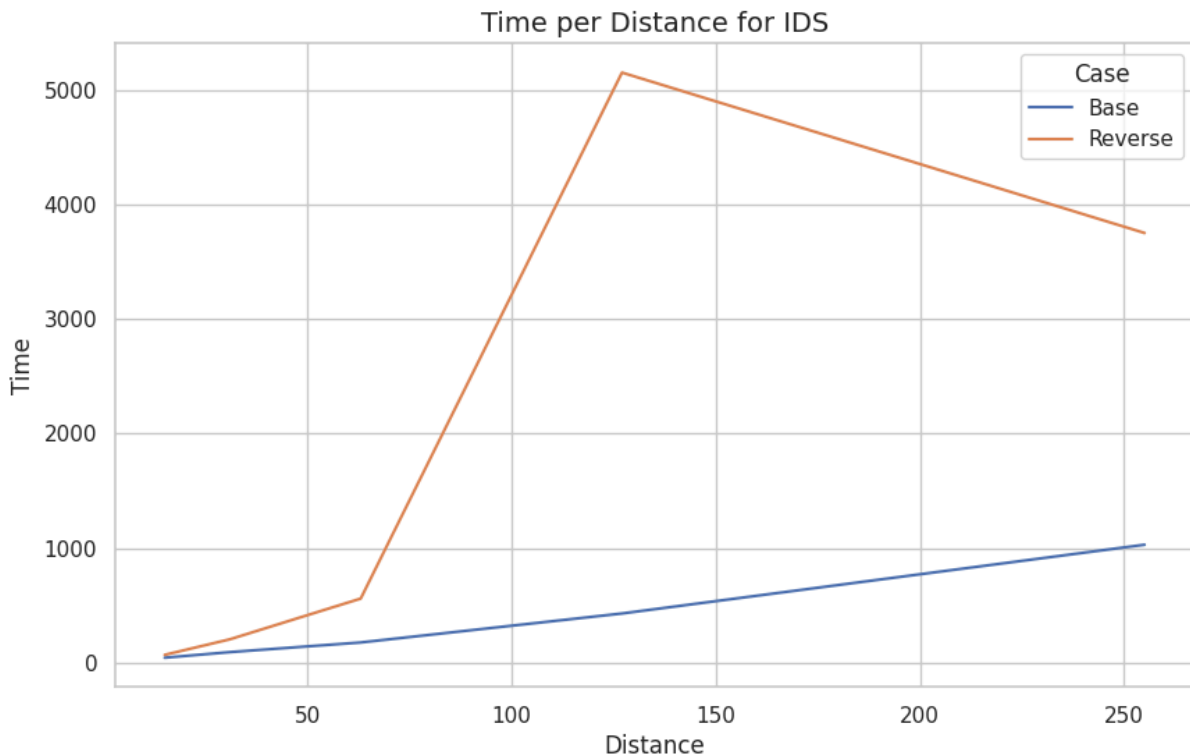


Figura 1: Tempo de execução do IDS no mapa Cidade.

De forma geral, não se pode dizer que o tempo escala proporcionalmente à distância de Manhattan. Esse efeito foi observado, em menor escala, com os algoritmos rápidos também.

#### 4.1.2. Floresta

A floresta é um mapa mais denso, então não é possível selecionar os pontos usando-se a “estratégia de base 2” da seção anterior. No entanto, ela foi usada de base para se definir o ponto de início (17, 0), que é o canto superior mais à esquerda. Os pontos de destino escolhidos foram: (17, 16), (35, 31), (102, 63), (132, 127) e (208, 255). A escolha se deu a partir do deslocamento dos pontos da estratégia anterior para posições próximas que não estivesse bloqueadas. Como é possível observar nas figuras a seguir, não parece haver um padrão para o tempo de execução.

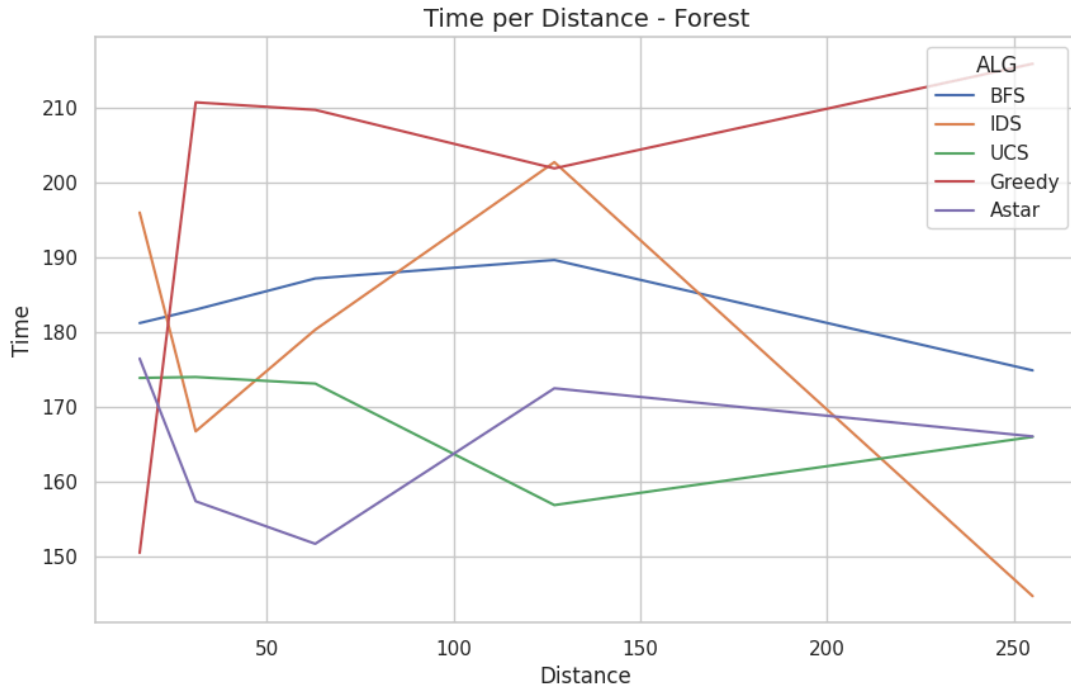


Figura 2: Tempo de execução no mapa Floresta.

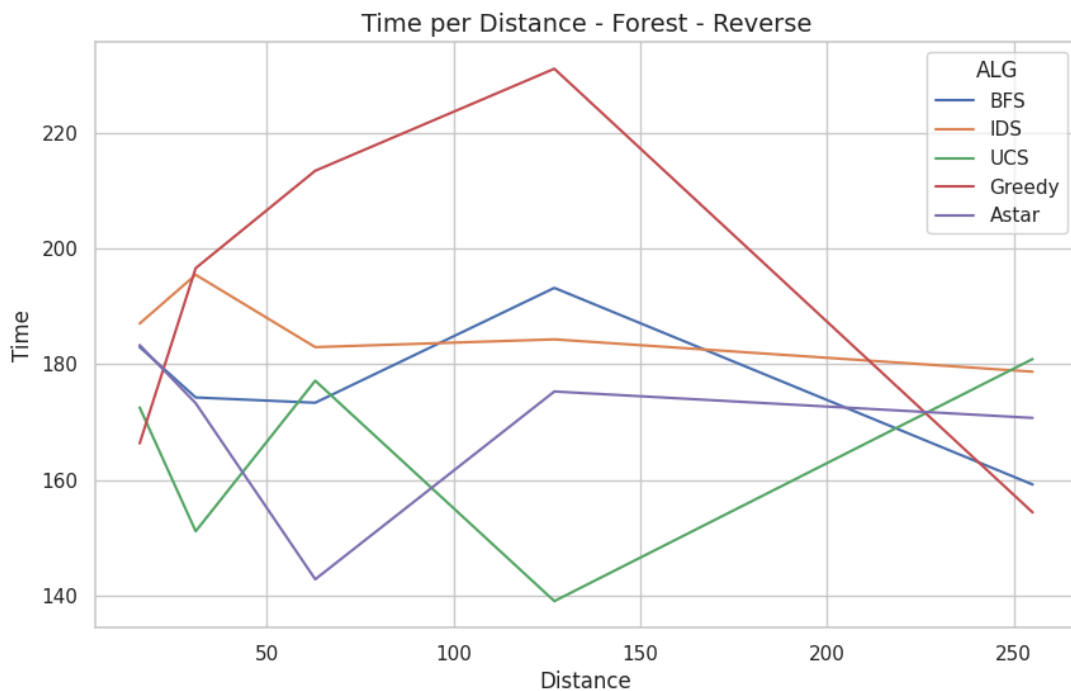


Figura 3: Tempo de execução no mapa Floresta, fazendo o caminho reverso.

Como as coordenadas  $x$  e  $y$  dos pontos são diferentes, foi adotado o  $y$  como base para a distância (para manter a consistência com o gráfico anterior).

Essa falta de correlação pode estar associada ao tamanho das instâncias ser muito pequeno, de modo que questões de *overhead* não são insignificantes. É interessante observar que apesar disso, os tempos de execução são bem consistentes entre os algoritmos, de forma geral. Esse é um padrão claramente distinto do comportamento observado no outro mapa, em que o IDS foi consideravelmente mais longo. Essa distinção provavelmente aconteceu pelo fato do IDS não precisar “perder tempo” explorando caminhos infrutíferos, pois muitos deles estariam bloqueados.

## 4.2. Número de Estados Expandidos