IA – Trabalho Prático 1
Aluno: Guilherme Torres Matricula: 2015083744

1. Introdução

O objetivo do trabalho é fazer uma implementação das técnicas de pathfinding aprendidas em sala de aula, com a intenção de fazer um estudo comparado destas técnicas. Os algoritmos utilizados no trabalho foram a Busca de Aprofundamento Iterativo (IDS), que combina a busca limitada da busca em largura com o espaço linear da busca em profundidade, Busca de Custo Uniforme (UCS), Greedy Best-First Search(BG) e A*, que é basicamente uma combinação das últimas duas técnicas.

O programa for implementado em C++17 e posde ser compilado através da execução do script *compila.sh*. Cada algoritmo tem seu script para a execução do programa. Para executar, o comando é <shell do algoritmo> <nome do mapa> <x inicial> < y inicial> <x final> < y final> <heuristica>

2. Implementação

O programa recebe como entrada o nome do arquivo do mapa, que é um grid onde os caracteres '.' são considerados como espaços livres; o nome do algoritmo que vai executar, um par de números indicando as coordenadas da posição inicial, outro par indicando a posíção final e, caso o algoritmo tenha uma heurística (BG/A*), o nome "manhattan" ou "octile", indicando qual deve ser usada.

Para o IDS, como é uma busca em profundidade com um limite na profundidade dos nós, para fazer a busca foi usada uma pilha de posições. Caso o nó esteja na profundidade que está sendo investigada, é retornado se for um estado final. Caso contrário, são aprofundados os nós onde a busca será efetuada. Este é um algoritmo ótimo, porém o seu tempo de buca é exponencial com base no branching factor do problema (como o mapa é um grid e pode-se andar em 8 direções, esse fator é em média 8) e expoente na profundidade do nó de solução.

As buscas de custo uniforme (no caso, é o algoritmo de Dijkstra), greedy BFS e A* tem estruturas de dados semelhantes. Os nós explorados pelos algoritmos são ordenados de acordo com o seu custo aparente f(x), onde f(x) é dado como:

$$f(x) = (2-w) * g(x) + wh(x)$$

onde g(x) é o custo para chegar no ponto, h(x) é o valor da heurística no ponto (foram usados a distância de Manhattan e a distância Octile, como veremos adiante) e w vale zero para a UCS, um para o A* e 2 para a greedy best-first. Dessa forma, usa-se a mesma função com a variação de apenas um termo para representar os custos de três algoritmos diferentes.

As heurísticas usadas foram:
- distância de Manhattan, dada por dx + dy, sendo d_ a diferença entre a coordenada _ do ponto sendo explorado e do objetivo.
- distância Octile: dada por max(dx, dy) + 0.5 * min(dx.dy)

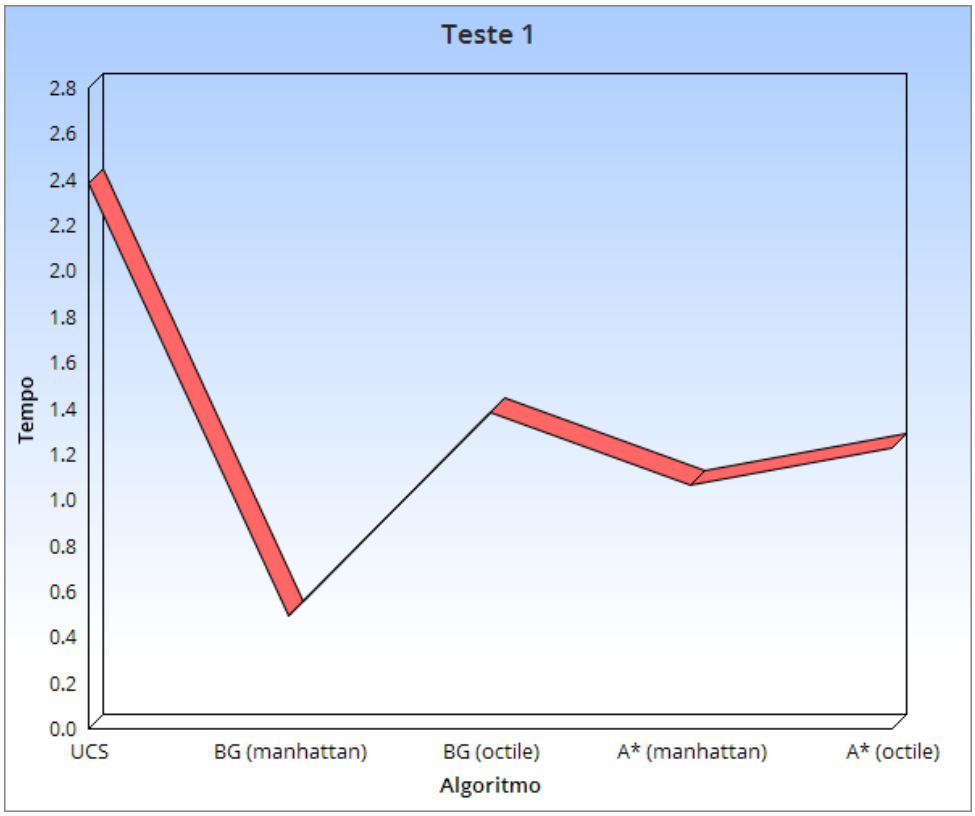
A heurística de Manhattan não é admissível, porque o seu valor pode ser maior do que o custo ótimo em alguns casos. Suponhamos que os locais de início e meta sejam (0, 0) e (10, 10). A distância de Manhattan entre eles é 20, mas o custo pelas diagonais (se não houver um obstáculo no caminho) é 15. A distância octile, neste caso, pode ser igual à distância real, portanto também não é admissível. Uma heurística que seria admissível seria por exemplo a distância euclidiana.

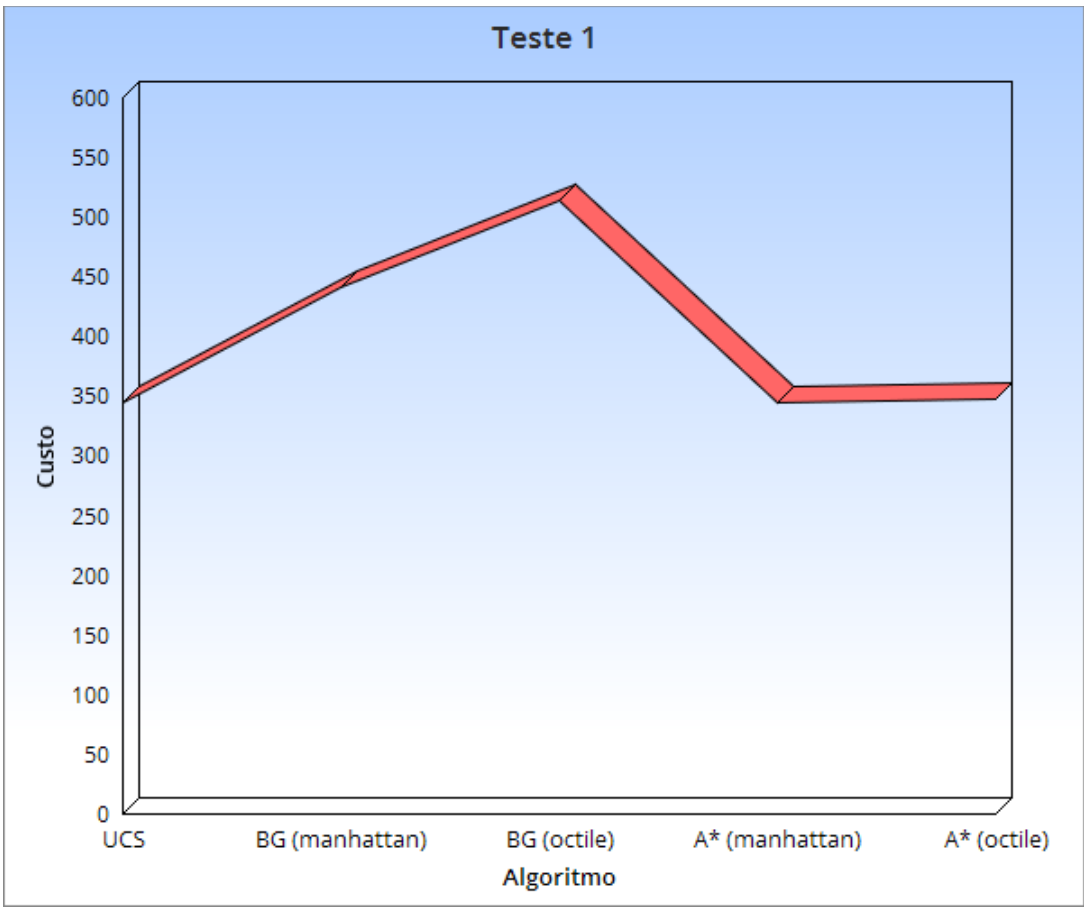
3. Experimentos

Os testes foram realizados em uma máquina com processador Pentium G4400 dual-core, 3.3Ghz, 8GB de RAM e sistema operacional Debian 9.

Como o IDS gasta tempo demasiado para completar as suas buscas e a sua diferença perante aos outros é de uma ordem de grandeza, além de termos certeza que os seus resultados são ótimos, foi desconsiderado nos testes. Abaixo temos a comparação dos tempos de execução e resultados encontrados por cada um dos algoritmos em questão para 3 mapas.

3.1. map1.map, posição (10, 10) a (200, 200), custo ótimo = 334

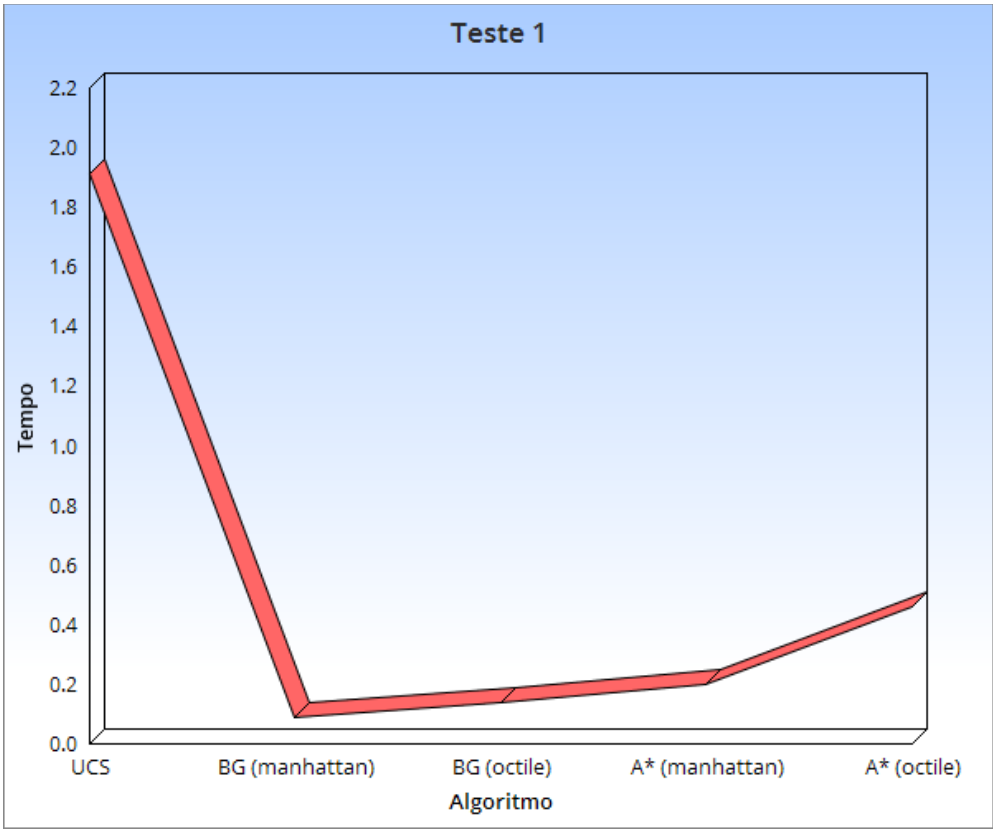


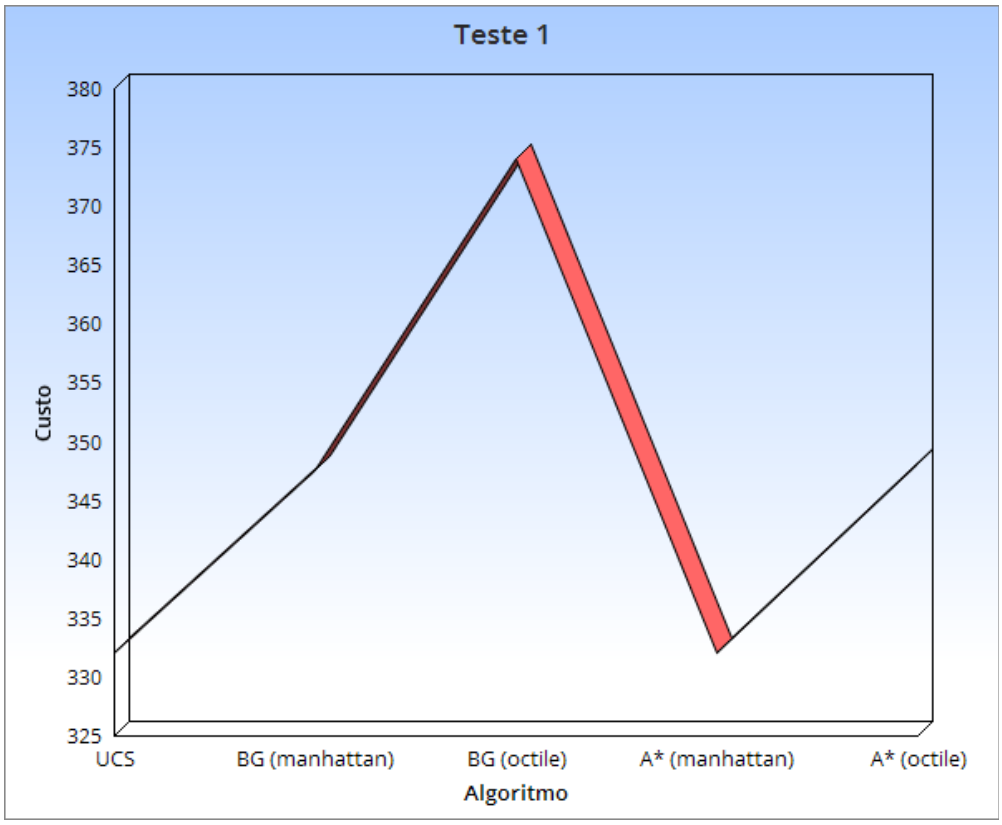


Segundo dados das figuras acima, vemos que há uma tendência clara da busca de custo uniforme em gastar mais tempo para completar a sua busca. Isso se deve à maior quantidade de nós explorados pelo algoritmo.

Além disso, vemos que a busca mais rápida em média é a BG , particularmente com a heurística de Manhattan. Isso também é compreensível, já que se sabe que a sua quantidade de nós explorados é menor. Porém, as soluções encontradas por esse algoritmo, nessa instância, foram as mais distantes das ótimas.Uma outra série de testes confirma essas tendências, como vemos abaixo.

3.2. map2.map, posiçao inicial (10, 10), meta (200, 200), ótimo = 332





Novamente, vemos que a UCS é a que leva mais tempo, porém leva fatalmente a um resultado ótimo, a BG é a mais rápida, porém com os resultados mais caros e a A* é um ponto intermediário entre as duas. Vale lembrar também a vantagem que a distância de Manhattan teve sobre a distância Octile. Parte do que pode ter causado isso no fator tempo é o fato de ser uma heurística que gera valores maiores, ou seja, uma posição que está mais distante da final tem o valor da heurística maior e, portanto, é isolada da busca por mais tempo. Isso ajuda a acelerar o processo de busca através dos bons locais.

Sobre a aparente vantagem dos custos na distância de Manhattan, a tendência que se percebeu nesse sentido foi muito leve, e não é provável que se possa estabelecer uma relação de cuasa-efeito apenas com esses dados.