Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Bacharelado em Ciências de Computação – 2020.2
SCC0230 - Inteligência Artificial

Trabalho Prático 1

Prof.º Alneu de Andrade Lopes

Caio Augusto Duarte Basso - 10801173
Gabriel Garcia Lorencetti - 10691891
Henrique de Souza Queiroz dos Santos - 10819029
Witor Matheus Alves de Oliveira - 10692190

São Carlos, 20 de outubro de 2020

1 Introdução

O trabalho prático consiste na implementação de 5 algoritmos, sendo dois de busca cega (busca em profundidade e busca em largura) e três de busca informada (busca Best-First Search, busca A* e Hill Climbing). O objetivo é, dado um labirinto, um ponto inicial e um final, percorrê-lo de modo a encontrar um caminho entre eles.

2 Descrição das implementações

Para a realização do trabalho foram utilizados 10 labirintos, alguns com mais de um possível caminho até se chegar no ponto final e com tamanho diversos. Estes labirintos utilizados foram gerados utilizando o site *dCode* e modificados pelos integrantes do grupo colocando os pontos iniciais e finais em cada labirinto, além de retirar algumas partes deles.

Os algoritmos foram implementados na versão 3 da linguagem Python. Os labirintos foram convertidos para um grafo do tipo lista de adjacências, considerando como nós somente os caminhos válidos (‘*’, ‘#’, ‘$’) e como vértices os passos possíveis entre cada nó dentro do labirinto.

Com a finalidade de realizar as comparações entre os algoritmos contabilizamos o tempo de execução de cada um deles para cada um dos labirintos, bem como a contabilização de extensões, ou seja, o número de nós visitados no total incluindo os que não fazem parte dos possíveis caminhos.

2.1 Busca em profundidade

A partir de um nó inicial, utilizamos uma função iterativa para adentrar na árvore de nós adjacentes ao inicial até que o nó encontrado seja o destino. Para controlar se os nós já foram visitados anteriormente, utilizamos um vetor de nós visitados. Para fins de testes, controlamos o número de nós que são visitados à medida que o algoritmo se aprofunda na árvore de nós.

A complexidade desse algoritmo no pior caso é $O = (|E| + |V|)$, onde $|V|$ é o número de vértices e $|E|$ o número de arestas.

2.2 Busca em largura

O algoritmo BFS explora todos os nós vizinhos antes de seguir para o próximo nível. Ou seja, começamos pelo vértice inicial (no nosso caso, representado

pelo símbolo ‘#’ no labirinto), e exploramos todos os nós vizinhos a ele. Então, para cada um desses vértices mais próximos, exploramos todos seus respectivos vizinhos que ainda não foram visitados. O algoritmo segue essa lógica até encontrar o nó final (representado em nosso labirinto pelo caractere ‘$’).

A complexidade desse algoritmo no pior caso é $O = (|E| + |V|)$, onde $|V|$ é o número de vértices e $|E|$ é o número de arestas.

2.3 Busca Best-First Search

Dado um nó inicial, adicionamos ele em uma fila de prioridade ordenada por custo total (*real* + *heurística*), depois disso o algoritmo entra em um loop retirando o primeiro elemento da fila de prioridade, que será chamado de nó atual. A partir desse nó retirado, é feita a verificação se ele é o nó objetivo, caso seja, o algoritmo se encerra, caso não, para cada nó adjacente do atual que não tenha sido visitado, é calculado custo real até chegar à ele (*g(n)*) e sua heurística (*h(n)*), que nesse caso foi a distância de Manhattan, então esse vizinho é adicionado na fila de prioridade e marcado como visitado. Depois de verificar todos os vizinhos do nó atual, o nó atual é marcado como visitado e reinicia-se o processo de escolher um nó retirando o primeiro da fila de prioridade até que se chegue no nó objetivo.

A complexidade desse algoritmo no pior caso é $O = (|V| * Log\,|V|)$, onde $|V|$ é o número de vértices do grafo.

2.4 Busca A*

O algoritmo de busca A* parte de um nó inicial e salva os adjacentes à ele em uma fila de prioridades. Essa fila controla qual o próximo nó a ser visitado, sendo que a prioridade é dada para os nós cujo o cálculo heurístico do custo da distância dele até o nó final seja o menor. O cálculo é dado pela distância de Manhattan entre as coordenadas do próximo nó (vizinho do atual) e do nó final somada com o custo do caminho feito até o nó atual incrementado de 1 (o custo incrementa 1 a cada nó visitado, já que fora especificado que o custo de se visitar os nós é um). O algoritmo é executado iterativamente até que o nó visitado seja o nó de destino.

A complexidade desse algoritmo no pior caso é $O = (|E|)$, onde $|E|$ é o número de arestas.

2.5 Hill Climbing

O algoritmo de Hill Climbing é um algoritmo de busca local iterativo. A cada iteração do algoritmo, ele se move sempre em uma direção que fique mais perto do objetivo. É um algoritmo guloso, que não garante que uma solução seja encontrada, porém quando consegue encontrar uma solução, é sempre uma solução boa. O algoritmo foi aplicado, de forma que para se movimentar em alguma direção, a posição nova deve ser mais próxima ao alvo, quando comparada com a posição atual.

É um algoritmo com uma complexidade no pior caso de $O = (n + m)$ iterações, onde n é o número de linhas e m é o número de colunas na matriz labirinto.

3 Resultados

Os resultados mostraram que, apesar de rápido, o algoritmo *Hill Climbing* se mostrou ineficaz nos testes feitos, sendo o algoritmo com pior desempenho entre todos, apenas encontrando o caminho correto quando era impossível não encontrá-lo (Labirinto 3). Os outros algoritmos que conseguiram encontrar o destino em todos os testes feitos foram o *Depth-First Search (DFS)*, *Breadth-First Search (BFS)*, *Best-First Search* e A* (A *star*).

Em relação ao tempo, calculamos a média de tempo de execução entre os 10 labirintos. Sobre essa métrica, o algoritmo que melhor performou foi o algoritmo *BFS*, obtendo uma média de tempo de execução de 4,45E-04 segundos. Em segundo lugar, o algoritmo *Best-First Search* com 1,34E-03 segundos. Em terceiro, o algoritmo DFS, com uma média de 1,73E-03 segundos. Por último, o algoritmo A*, com uma média de 3,17E-03 segundos.

3.1 Tempo despendido pelos algoritmos

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	**1**	5,07E-04	424	Sim
BFS		1,57E-04	670	Sim
Best-First-Search		4,17E-04	127	Sim

A Star	13 x 15	1,10E-03	711	Sim
Hill Climbing		4,13E-05	44	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	2	1,01E-04	32	Sim
BFS		4,11E-05	177	Sim
Best-First-Search		1,34E-04	57	Sim
A Star	12x12	2,60E-04	82	Sim
Hill Climbing		3,03E-05	39	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	3	2,74E-04	296	Sim
BFS		1,25E-04	670	Sim
Best-First-Search		3,92E-04	144	Sim
A Star	12x12	9,59E-04	635	Sim
Hill Climbing		6,74E-05	98	Sim

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	4	3,46E-05	38	Sim
BFS		2,79E-05	123	Sim
Best-First-Search		6,14E-05	32	Sim
A Star	12x12	2,11E-04	66	Sim
Hill Climbing		3,03E-05	41	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	5	5,05E-05	88	Sim
BFS		4,24E-05	179	Sim
Best-First-Search		1,03E-04	55	Sim
A Star	12x12	3,40E-04	135	Sim
Hill Climbing		3,56E-05	45	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	6	1,34E-03	667	Sim
BFS		4,52E-04	1913	Sim
Best-First-Search		1,47E-03	614	Sim
A Star	37x37	3,05E-03	1463	Sim
Hill Climbing		4,61E-06	2	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	7	1,46E-03	491	Sim
BFS		1,88E-03	7848	Sim
Best-First-Search		4,86E-03	1175	Sim
A Star	50x50	2,56E-02	9545	Sim
Hill Climbing		1,00E-04	113	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	8	8,27E-04	363	Sim
BFS		2,90E-04	1215	Sim
Best-First-Search		8,67E-04	328	Sim
A Star	34x60	4,58E-03	909	Sim
Hill Climbing		4,00E-06	2	Não

	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	9	1,91E-03	565	Sim
BFS		6,91E-04	2927	Sim
Best-First-Search		2,48E-03	896	Sim
A Star	21x101	8,61E-03	2581	Sim
Hill Climbing		2,96E-05	33	Não

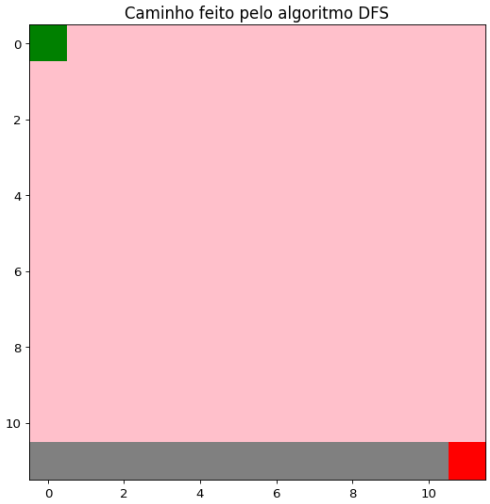
	Labirinto	Média do tempo de execução (s)	Média de extensões	Encontrou
DFS	10	1,08E-02	1565	Sim
BFS		7,40E-04	3091	Sim
Best-First-Search		2,63E-03	839	Sim
A Star	50x49	4,30E-03	1177	Sim
Hill Climbing		5,15E-06	2	Não

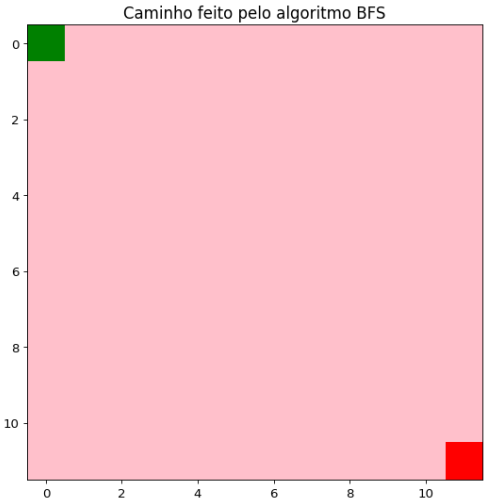
3.2 Comparação de caminho entre algoritmos

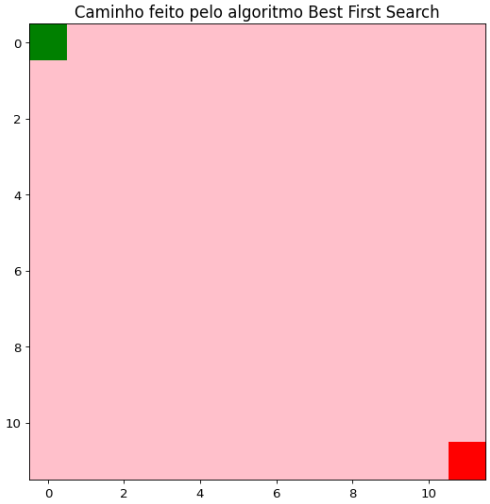
Nos labirintos a seguir representados, o quadrado verde representa o início do caminho, o vermelho o fim, o cinza os caminhos disponíveis, o preto as paredes e o rosa o caminho percorrido pelo algoritmo em questão.

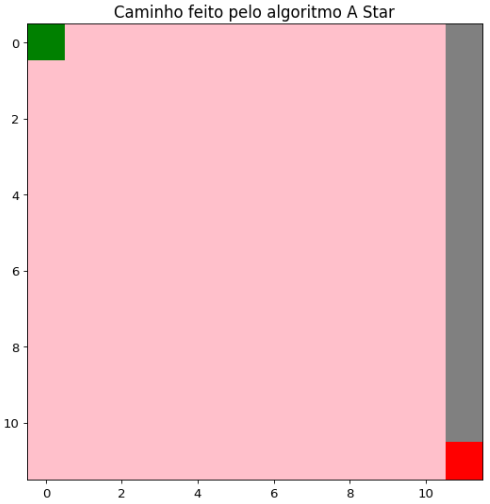
3.2.1 Labirinto 3

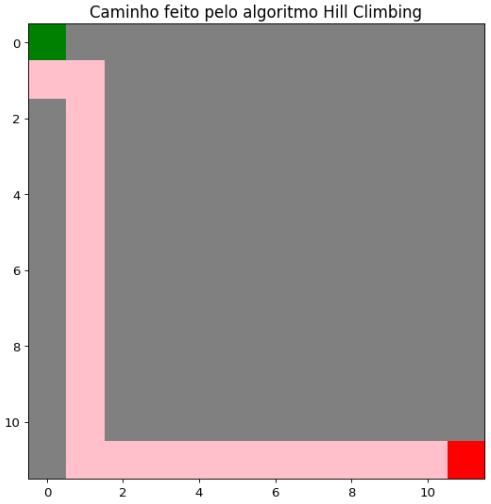
Esse foi o único labirinto em que o Hill Climbing conseguiu completar o caminho, se saindo melhor que todos os outros algoritmos, tanto em questão de menor caminho percorrido, quanto de tempo.





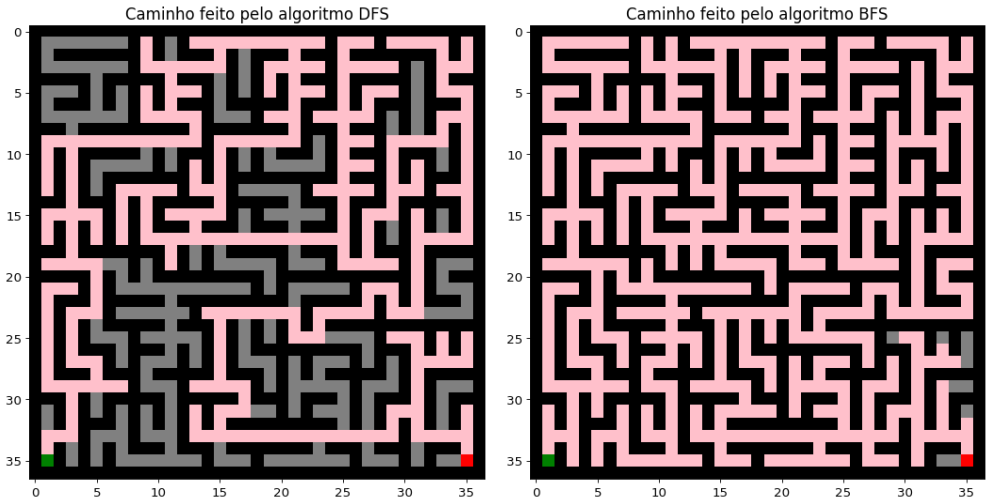


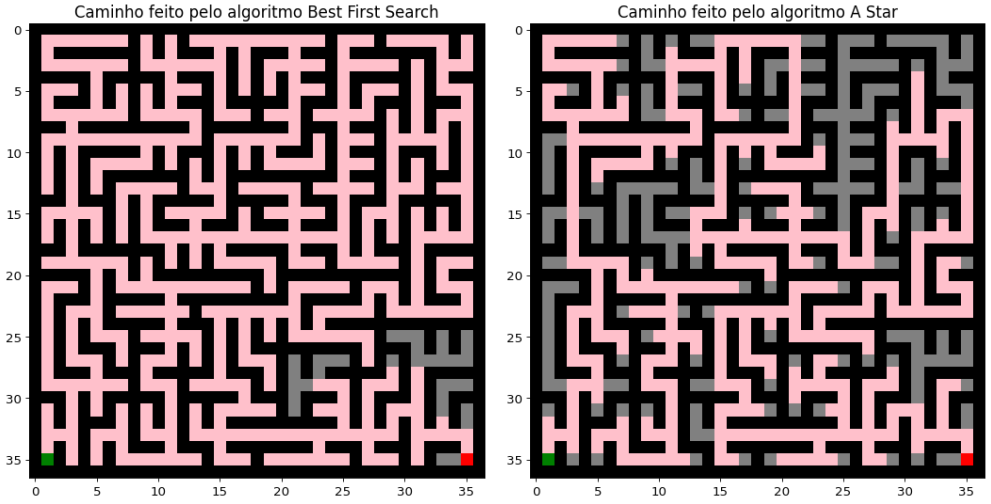


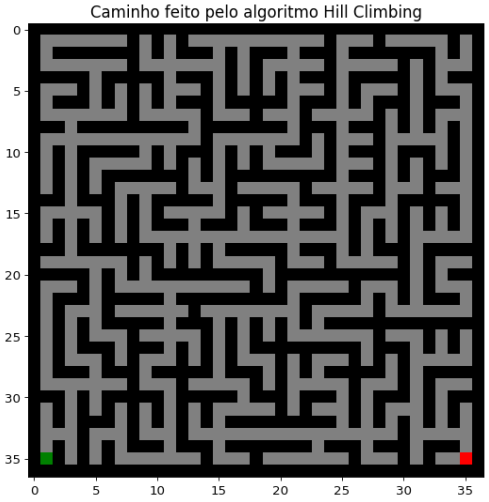


3.2.2 Labirinto 6

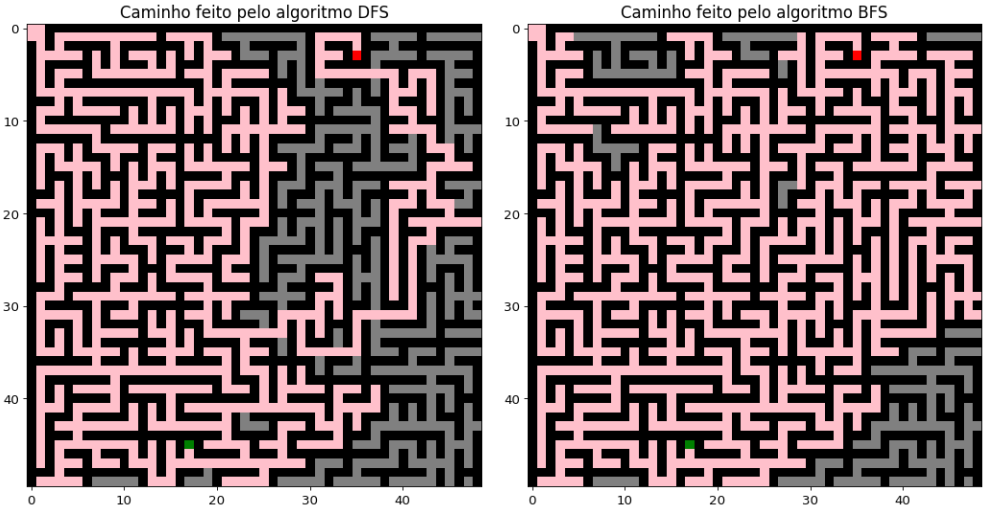
Nesse labirinto em questão, somente o Hill Climbing não conseguiu encontrar o caminho.

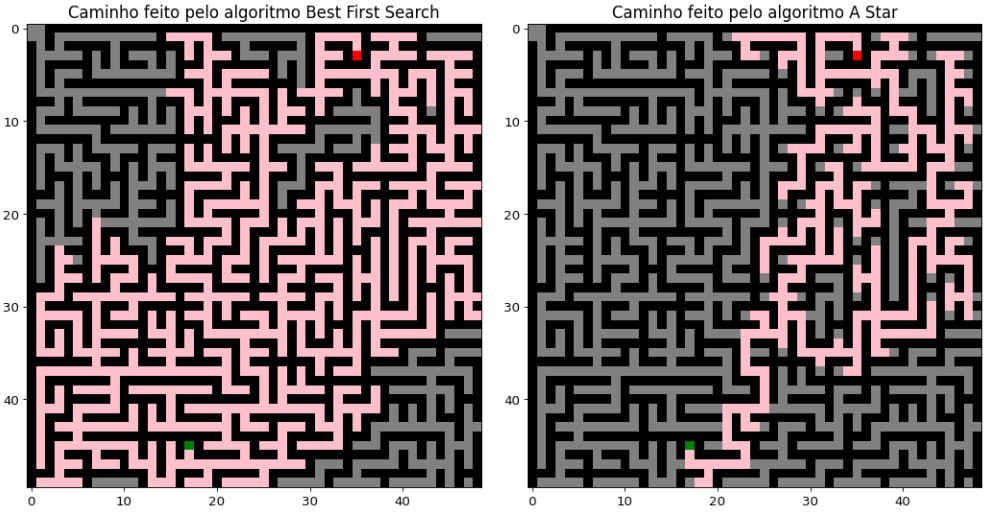


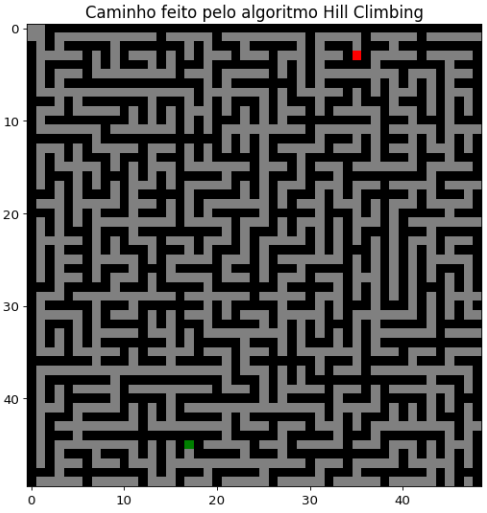




3.2.3 Labirinto 10







4 Discussão sobre as heurísticas usadas nos algoritmos de busca informada

No algoritmo *Best-First Search* e *A** utilizamos a distância de Manhattan como heurística, por ser boa para se calcular distâncias considerando que só se pode mover em direções com 90° entre elas, ou seja, só se pode mover para cima, baixo, esquerda e direita, além de também ter um custo leve de computação o que acaba por não impactar tanto na contagem dos tempos de execução dos algoritmos.

Utilizamos a mesma heurística nos algoritmos *A** e *Best-First Search* para podermos fazer uma comparação entre eles, ainda que não muito realista, já que são algoritmos relativamente parecidos. E, como foi observado no tópico de resultados, o Best-First Search foi o segundo algoritmo mais rápido numa média de tempo das médias de todos os labirintos enquanto o A* foi o último, diferença de tempo essa que foi causada pela verificação a mais (se o nó a ser inserido já foi visitado) feita na hora de fazer a inserção na fila de prioridades usada no algoritmo *A** que não é feita no *Best-First-Search*.

No algoritmo de *Hill Climbing* foi utilizada a distância euclidiana como heurística, por ser um método simples de se calcular a distância entre dois pontos em uma matriz. Como no labirinto só é possível se mover horizontalmente ou verticalmente, a distância foi calculada pela soma dos módulos das diferenças entre as coordenadas dos pontos atual e alvo.

5 Execução do programa

Os algoritmos foram implementados utilizando Linux, e faz-se necessário a utilização de tal sistema para plotagem dos labirintos. Para a execução do programa, estando na pasta dos arquivos, executar o seguinte comando:

python3 buscas_ia.py

É preciso possuir previamente instalado as bibliotecas NumPy e Matplotlib para a execução dos algoritmos e impressão dos labirintos. O código executará todos os labirintos no formato *Labirinto*.txt*, com * de 0, …, *numLabirintos* (que pode ser definido na *main*) presentes na pasta raíz do *buscas_ia.py*.