Relatório TP1

Aluno: Victor Augusto Hon Fonseca

-Parte 1 Estruturas usadas:

As estruturas de dados utilizadas se resumem à “list”, utilizada principalmente no cálculo de quais são os nós que foram gerados, “deque” que funciona como uma queue do c++ e “set” que corresponde à implementação do python de uma tabela hash, usada para tentar otimizar o tempo, além disso utilizou-se a fila de prioridades (heapq).

Além delas, foram criadas as classes “Node” contendo os atributos estado (uma string de números de 0 a 8, cuja ordenação representa o estado do 8-puzzle), ‘parent’ (o nó a partir do qual gerou-se o nó atual), ‘costG’ (custo da ação), ‘costH’ ou ‘value’(o valor da heurística usada), e a classe “Problema” com o estado inicial e o estado objetivo, representado por ‘1 2 3 4 5 6 7 8 0’.

-Parte 2 Diferenças entre os algoritmos:

-Busca sem informação:

-Breadth-first Search (BFS): Algoritmo que opera iniciando pelo nó raiz e depois para os nós mais rasos, analisando camada por camada da “árvore de estados”.

-Iterative deepening Search(IDS): Algoritmo que busca combinar os benefícios da Bfs e da Dfs, fazendo repetidas buscas em profundidade aumentando o limite a cada iteração.

-Uniform Cost Search(UCS): Algoritmo semelhante à Bfs, mas expande os nós de menor custo até o momento

-Busca com informação:

-A* Search: Algoritmo de certa forma semelhante ao UCS, mas se baseia nos nós que tenham o menor custo “f”, sendo f a soma do custo até o momento do nó mais o valor da heurística associada ao estado.

-Greedy Best-First Search: Algoritmo semelhante ao UCS, mas se baseia puramente no valor da heurística para expandir os nós, expandindo os que tiverem o menor valor da heurística.

-Busca Local:

-Hill Climbing: Move-se na direção de estados cujo valor da função objetivo (essencialmente uma heurística) seja maior. Porém com o intuito de diminuir as chances de se ficar preso em máximos locais, permitiu-se movimentos laterais, ou seja, mover para estados cujo valor da função seja igual ao atual, sendo

aleatório se moverá para estados com valor igual ou para estados com valor maior ao atual.

-Parte 3 Heurísticas Utilizadas

Utilizou-se para a 'Greedy Best-First Search' o número de peças erradas, para o 'Hill-Climbing', a mesma heurística, só que com seu valor se iniciando como 9 e sendo reduzido em 1 unidade a cada peça que estiver na posição errada e, por fim, para o 'A* search', a soma das distâncias de cada peça às suas posições corretas.

O 'número de peças erradas' é uma heurística admissível, pois cada peça deve ser movimentada pelo menos uma vez.

A 'soma das distâncias de cada peça às suas posições corretas' é uma heurística admissível, pois cada ação movimenta cada peça apenas um passo mais próximo do objetivo.

-Parte 4 Exemplos de soluções encontradas pelos algoritmos

-Para uma situação cuja número mínimo de passos para se checar ao estado objetivo é 10

Bfs: 10 passos

Ids: 10 passos

Ucs: 10 passos

A*: 10 passos

Greedy Best-First Search: 54 passos

Hill Climbing: 11 passos e 4 tentativas

-Parte 5 Análise quantitativa comparando os algoritmos com relação ao número de estados expandidos e tempo de execução à medida que o número de passos para a solução aumenta

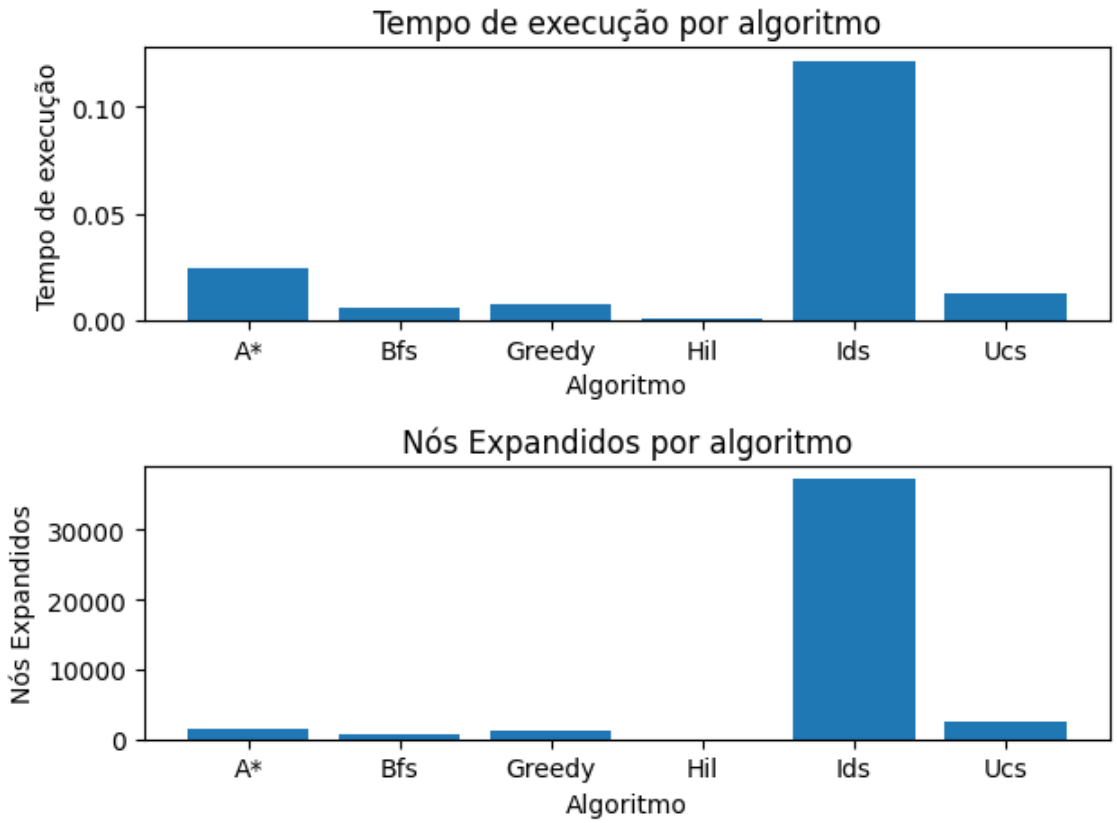
	Algoritmo	Tempo de execução	Nós Expandidos
0	A*	0.000101	0
1	A*	0.000132	6
2	A*	0.000178	19
3	A*	0.000490	45

4	A*	0.000648	85
5	A*	0.001056	130
6	A*	0.000888	140
7	A*	0.003217	408
8	A*	0.008348	641
9	A*	0.010364	916
10	A*	0.024087	1525
11	Bfs	0.000101	0
12	Bfs	0.000100	4
13	Bfs	0.000109	11
14	Bfs	0.000136	17
15	Bfs	0.000231	29
16	Bfs	0.000332	45
17	Bfs	0.000454	67
18	Bfs	0.001107	181
19	Bfs	0.001552	256
20	Bfs	0.002269	359
21	Bfs	0.006045	676
22	Greedy	0.000101	0
23	Greedy	0.000102	4
24	Greedy	0.000141	8
25	Greedy	0.000123	11
26	Greedy	0.000136	13
27	Greedy	0.000108	16
28	Greedy	0.000116	18
29	Greedy	0.002977	543
30	Greedy	0.000145	23
31	Greedy	0.001869	351
32	Greedy	0.007440	1241
33	Hil	0.000093	0

34	Hil	0.000152	2
35	Hil	0.000216	4
36	Hil	0.000292	6
37	Hil	0.000460	16
38	Hil	0.001004	34
39	Hil	0.000116	1
40	Hil	0.000289	19
41	Hil	0.000482	9
42	Hil	0.000306	7
43	Hil	0.000853	13
44	lds	0.000096	0
45	lds	0.000055	4
46	lds	0.000114	16
47	lds	0.000172	32
48	lds	0.000561	124
49	lds	0.001152	293
50	lds	0.002467	677
51	lds	0.004077	1118
52	lds	0.009612	2504
53	lds	0.042653	11923
54	lds	0.121456	37221
55	Ucs	0.000117	0
56	Ucs	0.000111	10
57	Ucs	0.000176	29
58	Ucs	0.000193	33
59	Ucs	0.000425	89
60	Ucs	0.000505	105
61	Ucs	0.000822	173
62	Ucs	0.001484	305
63	Ucs	0.002175	523

64	Ucs	0.006318	1260
65	Ucs	0.012327	2478

Gráfico:



-Parte 6 Discussão dos resultados

Como podemos observar percebemos que o Ids possuiu nos casos testados mais nós expandidos do que os demais, e também um tempo de execução maior, o que pode indicar a hipótese de que quanto maior o número de nós expandidos, maior o tempo de busca. Isso é tolerável de se pensar uma vez que, por exemplo, ele explora os mesmos nós mais de uma vez.

Em relação ao HillClimbing, considero que os resultados obtidos foram esperados, uma vez que ele nem sempre acha a solução correta na primeira tentativa, o que resulta em um menor número de nós expandidos e em um menor tempo de execução.

Quanto aos demais, o resultado também é esperado, com A*, bfs e Ucs tendendo a ser algoritmos ótimos e o Greedy, embora tendo um maior número de nós expandidos, em comparação por exemplo, à Bfs, devido à heurística adequada e condizente escolhida, ainda sim tendo uma boa performance.