Relatório A*

Algoritmos em Grafos

Filipe Abner Soares Melo
Lucas Pereira Freitas

Montes Claros - MG
12/08/2024

---

1. Introdução:

O projeto "N-Puzzle Solver" é um algoritmo desenvolvido para resolver o problema do N-Puzzle, um quebra-cabeça deslizante onde o objetivo é ordenar peças em um tabuleiro até alcançar uma configuração alvo, movendo uma peça de espaço vazio. O desenvolvimento deste projeto foi motivado pelo interesse em aplicar técnicas de busca para resolver problemas clássicos em computação.

Este projeto tem como objetivo explorar diferentes algoritmos de busca para resolver o N-Puzzle, como A* e IDA*, permitindo aos usuários observar o desempenho de cada abordagem.

O relatório apresenta soluções comparativas entre os resultados de puzzles 3x3 e 4x4, porém, o trabalho foi desenvolvido de maneira a permitir execuções para puzzles de diferentes tamanhos e objetivos, dessa forma, é possível resolver puzzles de tamanhos superiores observando os limites computacionais fornecidos.

---

2. Decisões de Projeto:

2.1 Heurística da distância de Manhattan.

A heurística utilizada é a distância de Manhattan, que calcula a soma das distâncias absolutas das peças do quebra-cabeça em relação às suas posições corretas no estado objetivo.

A distância de Manhattan é uma escolha popular para problemas de busca, como o Quebra-Cabeça 8, porque é admissível (nunca superestimar o custo real de alcançar o objetivo) e consistente (a estimativa é sempre menor ou igual ao custo do passo atual mais a estimativa do próximo passo).

2.2 Pré-computação dos pesos.

A correta utilização do algoritmo de busca exige uma estimativa dos pesos heurísticos, que são fundamentais para a determinação do caminho mais eficiente. Para otimizar o desempenho do algoritmo, foi implementada uma estratégia de pré-computação dos pesos, evitando assim a necessidade de recalcular o valor heurístico a cada nó instanciado durante a execução.

A pré-computação resulta em uma matriz onde cada linha representa o elemento que se moverá, e cada coluna corresponde à sua posição de destino. Assim, o cálculo que seria realizado N*T vezes (onde N representa o número de iterações do algoritmo e T o tamanho do puzzle) é feito apenas uma vez, permitindo que os valores fiquem armazenados em memória para acesso rápido e eficiente. Por exemplo, para um puzzle que tenha como estado objetivo:

0	1	2
3	4	5
6	7	8

Onde, 0 representa a opção vazia, temos a matriz de pré-computação dos pesos igual a:

0	0	0	0	0	0	0	0	0
1	0	1	2	1	2	3	2	3
2	1	0	3	2	1	4	3	2
1	2	3	0	1	2	1	2	3
2	1	2	1	0	1	2	1	2
3	2	1	2	1	0	3	2	1
2	3	4	1	2	3	0	1	2
3	2	3	2	1	2	1	0	1
4	3	2	3	2	1	2	1	0

Note que o movimento da posição vazia sempre possui pesos 0, pois não é possível movê-lo.

Com os valores já calculados e armazenados em memória, o algoritmo pode acessá-los de forma imediata durante sua execução. Isso não apenas reduz o tempo de processamento, mas também minimiza a sobrecarga computacional, garantindo um desempenho mais eficiente e uma execução mais rápida do algoritmo.

2.3 Número de Inversões.

O algoritmo contém uma checagem do número de inversões para *determinar se os puzzles testados são resolvíveis*. O número de inversões é calculado contando os números menores que cada número que ocorre depois dele e encontrando a soma para todos os dígitos na configuração de entrada (exceto 0 ou espaço em branco). Abaixo exemplificamos utilizando uma representação vetorial do puzzle:

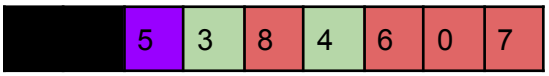
Numero de inversões inicialmente igual a 0

2	1	5	3	8	4	6	0	7

Numero de inversões igual a 1, inversões totais: 1

	1	5	3	8	4	6	0	7

Numero de inversões igual a 0, inversões totais: 1



Numero de inversões igual a 2, inversões totais: 3

Prosseguindo até o final do teste obteremos o *número total de inversões igual a 6*, onde, a cor roxa **indica o número do puzzle que estamos analisando**, a cor verde **indica que é uma inversão**, a cor vermelha **indica que não é uma inversão** e a cor preta indica **números que já foram analisados**.

Se o número de inversões tanto para o estado inicial quanto para o estado objetivo for ímpar ou par, o quebra-cabeça é solucionável; caso contrário, não é.

No Algoritmo A*, o número de inversões do estado atual do puzzle pode ser usado como critério de desempate na heurística, nos casos em que o valor de f é igual para os dois nós comparados. Abaixo temos uma comparação do algoritmo sendo executado com e sem a alteração.

Exemplo 1	Puzzle	Objetivo	Tempo	Memória	Lista Aberta	Lista Fechada	Iterações
Com desempate	5 6 2 7 1 8 3 4 0	0 1 2 3 4 5 6 7 8	425 ms	408 kb	473	806	807
Sem desempate			857 ms	488 kb	717	1296	1297

Exemplo 2	Puzzle	Objetivo	Tempo	Memória	Lista Aberta	Lista Fechada	Iterações
Com desempate	2 1 5 3 8 4 6 0 7	1 2 3 4 5 6 7 8 0	8299 ms	1360 kb	1900	3535	3536
Sem desempate			25349 ms	2688 kb	3713	7468	7469

Obs: Lista Aberta e Fechada consiste no número de nós que entraram na lista.

Dessa forma, podemos comprovar um desempenho substancial na melhora do algoritmo uma vez adicionado o critério de desempate, resultando em uma eficiência por vezes superior a 50%.

3. Implementação.

A representação usada em memória foi feita utilizando 2 estruturas que se complementam, os nós e o grafo. Os nós armazenam um estado qualquer do puzzle, os pesos g, h e f usados na implementação da heurística, e o índice do predecessor, importante para a recuperação do caminho mínimo. O grafo é constituído de um conjunto de nós que possuem sua relação representada através de uma de uma lista de adjacência, identificando a vizinhança dos mesmos.

3.1. A*

O funcionamento do A* envolve a exploração sistemática de nós, começando pelo estado inicial, até encontrar o objetivo. Durante essa exploração, ele mantém duas listas: uma lista aberta, que contém os nós a serem avaliados, e uma lista fechada, que armazena os nós já processados. A cada iteração, o algoritmo seleciona o nó com o menor valor de f da lista aberta, o que garante que o caminho mais curto seja explorado primeiro. Se o nó selecionado for o objetivo, a solução é encontrada; caso contrário, o nó é expandido e seus vizinhos são examinados, ajustando a lista aberta conforme necessário.

A implementação do A* segue de acordo com o seguinte pseudocódigo:

1. Adiciona o estado inicial à lista aberta
2. Enquanto a lista aberta não está vazia:
    a. Seleciona o nó com o menor f da lista aberta.
    b. Se o nó é o objetivo, retorna.
    c. Se não, coloca o nó na lista fechada e olha seus vizinhos
    d. Para cada vizinho:
        i. Se o vizinho está na lista fechada, ignora esse vizinho
        ii. Se o vizinho não estiver na lista aberta:
            1. Calcule seus pesos e coloca o nó atual como seu predecessor
            2. O adiciona na lista aberta
        iii. Se estiver e se o novo caminho para ele for menor que o caminho já encontrado:
            1. Coloca o nó atual como seu predecessor
            2. Calcula seu peso g

3.2. IDA*

O algoritmo IDA* (Iterative Deepening A*) é uma variação do A* que combina a busca em profundidade iterativa com a abordagem heurística do A*. Ele foi desenvolvido para resolver problemas de busca com grandes espaços de estados, onde o uso de memória é uma preocupação significativa, como no caso do 15-puzzle (4x4). O IDA* busca equilibrar a utilização de memória e a eficiência da busca, explorando apenas os nós necessários dentro de um limite de profundidade que é iterativamente ajustado.

A implementação do IDA* segue de acordo com o seguinte pseudocódigo:

1. Calcula o f do nó inicial e define esse f como o limite
2. Repita:
    a. Inicializa nova lista
    b. Adicione nó inicial na lista
    c. Inicia Busca:
        i. Pega o último nó da lista
        ii. Calcule seu peso f
        iii. Se o f é maior que o limite, retorna o f
        iv. Se o nó é a solução, retorna
        v. Define o mínimo como INT_MAX
        vi. Para sucessor do nó atual:
            1. Se o sucessor está na lista, salta o sucessor
            2. Coloca o nó como predecessor do sucessor
            3. Adiciona sucessor na lista
            4. Inicia Busca recursivamente
            5. Se a Busca encontrou a solução, retorna
            6. Se a Busca retornou um valor menor que o mínimo:
                a. define valor como novo mínimo
            7. Remove sucessor da lista
        vii. Retorna o mínimo
    d. Se a Busca encontrou a solução, retorna
    e. Se a Busca retornou INT_MAX, então retorna -1
    f. Caso contrário, define o limite como o retorno da Busca

---

4. Resultados.

O projeto foi bem-sucedido na implementação de uma solução eficiente para puzzles de tamanho 3x3 utilizando o algoritmo A*, demonstrando alta eficácia tanto em termos de tempo quanto de uso de memória. O A* mostrou-se especialmente eficaz ao lidar com o quebra-cabeça 8-puzzle, resolvendo-o de maneira consistente e dentro de limites aceitáveis de recursos computacionais.

Para o algoritmo IDA*, a solução também se mostrou eficaz, especialmente ao escalar o problema para puzzles de tamanho 4x4 (15-puzzle). O IDA* foi capaz de explorar o espaço de estados de forma mais profunda, mantendo a eficiência em termos de memória, que é uma das suas principais vantagens em comparação com o A*.

Abaixo, são apresentados os resultados de algumas execuções que permitem uma análise comparativa detalhada entre os algoritmos A* e IDA*. Esses resultados incluem métricas de desempenho como tempo de execução, consumo de memória, e a profundidade máxima explorada em diferentes cenários de puzzles, destacando as situações em que cada algoritmo se sobressai.

Exemplo 3	Puzzle	Objetivo	Tempo	Memória	Nós explorados	Profundidade
A*	8 6 7 2 5 4 3 0 1	1 2 3 4 5 6 7 8 9	29821 ms	2696 Kb	18807	31
IDA*			378 ms	28 Kb	42472	

Exemplo 4	Puzzle	Objetivo	Tempo	Memória	Nós explorados	Profundidade
A*	1 4 8 0 7 5 2 6 3	0 1 2 3 4 5 6 7 8	326 ms	256 Kb	1802	23
IDA*			26 ms	28 Kb	3341	

Exemplo 5 IDA*	Puzzle	Objetivo	Tempo	Memória	Nós explorados	Profundidade
14 13 15 7 11 12 9 5 6 0 2 1 4 8 10 3		0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	7399916 ms	68 Kb	443408963	57

Exemplo 4 IDA*	Puzzle	Objetivo	Tempo	Memória	Nós explorados	Profundidade
5 9 2 1 6 10 4 0 13 14 3 11 7 15 12 8		1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0	28744 ms	52 Kb	2810025	40

Abaixo será apresentado o resultado para 100 instâncias de puzzles 3x3 para ambos algoritmos A* e IDA* bem como o resultado médio de cada.

IDA*				
Nº	**Total de Nós Gerados**	**Total Iterações**	**Tempo (ms)**	**Profundidade**
1	2997	1901	25	22
2	11439	7192	49	23
3	11747	7393	1	17
4	18735	11762	42	25
5	19377	12175	3	16
6	23753	14935	23	22
7	26033	16375	12	22
8	27650	17401	8	21
9	29582	18628	9	17
10	31277	19696	8	21
11	33778	21264	13	23

12	38903	24480	28	23
13	40628	25571	9	19
14	43764	27543	19	22
15	45187	28451	7	19
16	53215	33481	44	22
17	55455	34903	12	20
18	55598	34997	1	21
19	56596	35628	7	22
20	57689	36319	5	22
21	57793	36391	0	16
22	57992	36525	1	16
23	64490	40587	36	25
24	68533	43134	23	22

25	72015	45323	19	23
26	79307	49891	43	25
27	82556	51933	17	21
28	82863	52136	1	13
29	87741	55191	26	23
30	88504	55673	4	24
31	93433	58766	29	24
32	93779	58991	1	16
33	93867	59052	0	18
34	97208	61153	18	23
35	107535	67594	61	26
36	118463	74385	66	27
37	120081	75407	8	22

38	134861	84634	90	26
39	141822	89007	43	22
40	153666	96435	66	25
41	153959	96628	1	14
42	161826	101521	45	25
43	166928	104723	30	21
44	191779	120280	158	26
45	205020	128571	78	24
46	205907	129126	4	24
47	206093	129250	1	18
48	209052	131104	16	23
49	211258	132499	11	21
50	230156	144260	120	28

51	232557	145773	12	20
52	263735	165275	201	27
53	265183	166196	7	19
54	275569	172659	62	26
55	276073	172984	2	18
56	281232	176221	30	24
57	282117	176791	4	18
58	287919	180425	32	24
59	288639	180880	3	23
60	297080	186164	53	24
61	319227	199948	192	27
62	335004	209849	149	25
63	336524	210812	12	21

64	336769	210971	1	22
65	367885	230392	259	28
66	368799	230973	8	20
67	391680	245274	216	27
68	410842	257236	191	27
69	413779	259078	29	25
70	422334	264438	89	24
71	424616	265869	23	24
72	425734	266581	13	22
73	456996	286105	248	27
74	465012	291103	55	26
75	478559	299594	85	25
76	483131	302443	29	26

A*					
Nº	**Total de Nós gerados**	**Tempo(ms)**	**Open List**	**Closed List**	**Profundidade**
1	2178	423	473	806	22
2	2583	684	551	966	23
3	200	5	51	73	17
4	2633	694	580	991	25
5	205	5	58	72	16
6	2199	480	484	820	22
7	1045	100	242	389	22
8	1050	101	244	388	21
9	599	33	147	219	17
10	493	24	117	184	21
11	1513	267	335	567	23

12	2996	811	626	1117	23
13	607	33	146	223	19
14	1469	193	335	546	22
15	653	46	148	243	19
16	3412	1159	736	1276	22
17	904	106	220	333	20
18	81	1	21	30	21
19	1043	108	223	392	22
20	1338	202	313	499	22
21	89	1	25	32	16
22	304	12	75	111	16
23	3937	1760	869	1473	25
24	1796	291	384	669	22

25	2109	421	466	785	23
26	3111	882	662	1164	25
27	840	64	194	311	21
28	153	2	41	55	13
29	1535	204	342	573	23
30	732	51	173	271	24
31	2853	691	632	1054	24
32	295	8	74	108	16
33	76	0	22	27	18
34	1802	296	401	672	23
35	4617	1908	958	1726	26
36	5520	2625	1158	2079	27
37	646	40	142	243	22

38	5464	2771	1153	2039	26
39	1961	362	434	730	22
40	5129	2882	1104	1907	25
41	231	5	59	84	14
42	2511	701	558	944	25
43	1260	187	291	470	21
44	6474	3667	1349	2429	26
45	4214	1761	879	1572	24
46	860	75	197	321	24
47	124	2	34	45	18
48	1501	244	330	564	23
49	1226	163	285	454	21
50	9260	8779	1870	3489	28

51	705	53	178	258	20
52	7822	5432	1559	2939	27
53	531	28	126	197	19
54	3223	913	684	1213	26
55	554	28	129	204	18
56	2091	379	457	776	24
57	556	30	128	205	18
58	3302	1029	711	1239	24
59	435	21	105	162	23
60	1791	353	384	670	24
61	3988	1777	816	1506	27
62	6291	3412	1283	2351	25
63	487	42	114	180	21

64	125	3	29	48	22
65	11308	11355	2184	4247	28
66	257	9	63	95	20
67	8882	10604	1769	3329	27
68	6411	5701	1329	2408	27
69	1922	562	424	717	25
70	5390	3726	1129	2006	24
71	2155	635	471	804	24
72	695	58	171	258	22
73	10417	11401	2052	3901	27
74	4140	1699	877	1557	26
75	6997	4406	1422	2160	25
76	1968	343	414	745	26

77	899	77	198	337	20
78	5630	2747	1159	2115	26
79	259	15	72	92	16
80	7680	5061	1590	2882	27
81	885	77	194	330	19
82	1962	306	400	748	25
83	170	3	48	61	15
84	105	1	29	38	17
85	362	14	94	132	15
86	2506	588	524	940	25
87	456	23	109	168	17
88	6518	4154	1351	2458	27
89	1052	118	250	390	20

90	785	70	182	291	20
91	3803	1316	804	1430	27
92	2255	444	475	847	26
93	1826	295	394	683	21
94	725	48	162	271	22
95	1051	106	231	392	22
96	537	30	130	199	19
97	1434	200	325	536	20
98	2045	381	460	753	21
99	2852	755	629	1066	24
100	3446	1142	713	1293	25
Média	2395.17	1183	507.42	891.62	22.15