Inteligência Artificial
Relatório 1 - Buscas

Grupo 8

Luís Duarte Carneiro Pinto, nº 201704025

20 de Março de 2019

Conteúdo

1 Introdução

Com este trabalho, pretende-se conseguir implementar algoritmos de procura dados dois estados, inicial e final. Assim, conseguiremos encontrar uma sequência de "passos"para chegar ao estado desejado (final). Para isto, serão utilizados os diferentes tipos de busca:

- DFS - Busca em profundidade
- IDFS - Busca Iterativa Limitada em Profundidade
- DFS - Busca em largura
- Greedy
- A*

As funções relevantes para o problema em questão, estão definidos da seguinte forma:

1. **depthfirst**(*configInicial, configFinal, max_depth*)* → busca em profundidade do estado *configInicial* para o estado *configFinal* com limite de profundidade igual a *max_depth*;
2. **bfs**(*configInicial,configFinal*)→ busca em largura do estado *configInicial* para o estado *configFinal*;
3. **depth_firsti**(*configInicial, configFinal*)→ busca em profundidade iterativa do estado *configInicial* para o estado *configFinal*;
4. **greedy_h1**(*configInicial,configFinal*)→ Busca gulosa com a função heurística *Número de peças fora do lugar* do estado *configInicial* para o estado *configFinal*;
5. **greedy_h2**(*configInicial,configFinal*)→ Busca gulosa com a função heurística *Manhattan distance* do estado *configInicial* para o estado *configFinal*;
6. **Astar_h1**(*configInicial,configFinal*)→ Busca A* com a função heurística *Número de peças fora do lugar* do estado *configInicial* para o estado *configFinal*;
7. **Astar_h2**(*configInicial,configFinal*)→ Busca A* com a função heurística *Manhattan distance* do estado *configInicial* para o estado *configFinal*;

NOTA: Todas as funções retornam a resposta na seguinte forma:
(**lista com passos do estado inicial para final*, *segundos que demorou a correr o código*,*máximo número de memória usada na execução da função**)

*Em todos os casos, configInicial e configFinal estão na forma de string: '$a_1\ a_2 \ldots a_{16}$', onde $\forall i \in \{1,\ldots,16\}, a_i \in \{0,\ldots,15\}$ e $\forall i \neq j, a_i \neq a_j$

2 Estratégias de Procura

2.1 Procura não guiada

Profundidade (DFS - Depth First Search):

O algoritmo de **busca em profundidade**, irá realizar uma busca seguindo instruções pré-definidas. Quero com isto dizer que irá seguir sempre uma direção (previamente decidida) até não conseguir realizar mais movimentos e só aí é que faz um *backtracking* para testar outras direções. É utilizada, normalmente, quando existe pouco espaço para armazenar os nós visitados.
Esta estratégia **não é otima nem completa**.

- **Otimalidade:** Este algoritmo apenas será ótimo quando existir apenas um caminho para chegar a um determinado estado final porque, caso contrário, tanto pode ser ótimo como pode não o ser como será visto, mais tarde, nos testes/resultados.
- **Completude:** Este algoritmo pode gerar um loop infinito onde executa a mesma sequência de movimentos infinitamente. Apenas atinge completude caso seja verificado se um dado nó já foi visitado mas, mesmo assim, poderá demorar imenso tempo.

Complexidade Temporal	Complexidade Espacial	Completude	Otimalidade
O(b^m)	O($b \times m$)	Sim, se houver verificação de nós repetidos	Nos casos onde apenas existe um caminho do estado Inicial para Final

b - fator ramificação da árvore
m - profundidade ótima onde se encontra a solução

Busca em Largura (BFS - Breadth First Search):

O algoritmo de **busca em largura**, começará por explorar todos os filhos do nó inicial e, caso não encontre a solução, irá expandir todos os nós filhos obtendo, assim, os filhos de todos estes já explorados. O processo será repetido até encontrar a solução. Ou seja, nunca visitará um nó de profundidade n sem ter explorado todos os nós de profundidade $n-1$. É uma busca utilizada quando a solução está num nivel baixo de profundidade.
Esta estratégia **é otima e completa**.

- **Otimalidade:** Iremos supor que a solução ótima se encontra no nível de profundidade n. Tal como dito anteriormente, nenhum nó de profundidade $k > n$ será visitado sem ter sido explorada esta solução ótima, logo irá encontrar a solução ótima.
- **Completude:** Ao haver verificação de nós visitados, a cada expansão nova, a árvore irá gerar nós nunca antes visitados. Se estivermos a tratar com um grafo de profundidade finita, irá haver um número finito de nós a serem explorados e, portanto, irá haver um fim a este processo de procura.

Complexidade Temporal	Complexidade Espacial	Completude	Otimalidade
O(b^m)	O(b^m)	Sim	Sim

b - fator ramificação da árvore
m - profundidade máxima explorada

Busca Iterativa Limitada em Profundidade (IDFS):

O algoritmo de **busca iterativa limitada em profundidade** é semelhante à busca *DFS*, mas irá realizar uma busca limitada iterativa, começando com limite 1 e aumentando a profundidade máx em cada iteração caso não encontre a solução desejada. Uma **busca limitada em profundidade** resume-se a estabelecer um limite que, no passo em que o algoritmo atinge essa profundidade, irá ter o mesmo comportamento ao de quando não consegue realizar mais movimentos. É uma busca utilizada quando se quer encontrar a solução ótima e se tem espaço limitado.
Esta estratégia **é ótima e completa**.

- **Otimalidade:** É um algoritmo ótimo pois, tal como BFS, apenas explora os nós de profundidade n após ter explorado todos os nós de profundidade inferior a n.
- **Completude:** Tal como no BFS, irá encontrar sempre a solução.

Complexidade Temporal	Complexidade Espacial	Completude	Otimalidade
O(b^m)	O($b \times m$)	Sim	Sim

b - fator ramificação da árvore
m - profundidade ótima onde se encontra a solução

2.2 Procura guiada

Função heurística: Uma função heurística permite atribuir um valor a um dado estado e, com isso, podemos avaliar qual o melhor estado para prosseguir com a nossa procura.

Busca Gulosa (Greedy):

O algoritmo de **busca gulosa** atribuirá um valor a cada nó. Desta forma, o algorítmo começará por expandir o nó inicial e irá expandido os nós cujo valor é o menor/maior (dependendo da heurística) daqueles expandidos. Caso não exista verificação de nós visitados, poderá entrar em loops infinitos e, por isso, apenas haverá completude se existir tal verificação. É uma boa busca para se utilizar caso se queira encontrar solução num curto período de tempo.
Esta estratégia **não é ótima** mas **é completa** caso haja verificação de nós visitados.

- **Otimalidade:** Certos estados podem ter valores associados mais atraentes mas podem não ser o caminho otimal, como será visto nos testes, mais à frente, mas, tal como DFS, caso só exista um caminho entre o estado Inicial e Final, este algoritmo irá obter a solução ótima.
- **Completude:** Tal como todos os outros algoritmos, ao se tratar de grafos finitos, este algoritmo acabará sempre por chegar a gerar todas as jogadas possíveis caso haja a tal verificação acima referida.

Complexidade Temporal	Complexidade Espacial	Completude	Otimalidade
O(b^m)	O($b \times m$)	Sim, se houver verificação de nós repetidos	Nos casos onde apenas existe um caminho do estado Inicial para Final

b - fator ramificação da árvore
m - profundidade máxima explorada

Busca A*:

O algorítmo **busca A*** é semelhante à busca *greedy*, mas com uma pequena diferença. Ao atribuir os valores aos nós, irá calcular o valor da heurística e adicionra o valor do nó pai. Esta pequena diferença irá afetar o comportamento do algoritmo a longo prazo pois nunca irá explorar demasiado numa "direção"sem explorar outras. Pois, mesmo que o resultado da função heurística num dado nó seja baixa, ao adicionar o valor de todos os nós que ele é descendente, este irá ficar com um valor elevado o que fará com que deixe de ser explorado num próximo passo. É uma busca que encontra sempre a solução ótima.
Esta estratégia **é ótima e completa**.

Complexidade Temporal	Complexidade Espacial	Completude	Otimalidade
$O(log(h^*(x)))$	O(b^m)	Sim	Sim

h* - custo real do estado inicial, x, para o estado final
b - fator ramificação da árvore
m - profundidade otimal da solução

3 Descrição do problema

O jogo dos 15 tem dois "tipos de estados"diferentes. Seja $< X >= \{x : \text{there is a path from } X \text{ to } x\}$, supondo que não consigo chegar de um estado, A, para um outro, B, então, $< A > \cap < B >= \emptyset$ e $< A > \cup < B >= \Omega$ (espaço de todas as soluções). A cada um destes dois diferentes estados, define-se como par ou ímpar. Existe uma solução entre dois estados se e só eles têm a mesma paridade.

Com este trabalho, pretende-se encontrar soluções entre dois estados cujas paridades coincidem, o que será explorado futuramente.

4 Descrição da Implementação

4.1 Linguagem Utilizada

A linguagem que utilizei para resolver este trabalho foi Python. Parece-me que não foi/é a linguagem otimal para este tipo de problemas, mas estou limitado ao meu conhecimento e apenas sei programar em Python, por agora.

4.2 Estrutura de Dados utilizada

Escolhi usar listas como estrutura de dados por serem uma estrutura de fácil manipulação. Com listas, consigo ordená-las da forma que for mais benéfica para correr o programa e consigo escolher elementos duma lista de forma rápida e eficaz o que me permite manipular facilmente os dados do problema.

4.3 Estrutura do Código

O código começa com definições auxiliares que foram necessárias para a construção das restantes funções utilizadas no código.

5 Resultados

Nas tabelas a seguir representadas, estão ilustrados os valores obtidos experimentalmente em cada um dos diferentes tipos de busca com estados finais e iniciais com soluções de várias profundidades, para melhor comparação dos resultados teóricos com os práticos.

NOTA: O número à frente de "Greedy"e "A*"representa a função heurística utilizada. O 1 representa a heurística "Número de peças fora do lugar"e o 2 a heurística "Manhattan distance".

Complexidade Temporal (ms)

Alg\Depth	2	3	4	5	6	7	8	9	10	12	13	Sol.Ótima
BFS	1	2	4	4	20	65	162	492	1930	55614	151579	✓
DFS	1155	43635	690	10^5	\	\	\	\	\	\	\	×
IDFS	1	2	8	18	28	89	121	369	628	3588	6749	✓
Greedy1	0.5	0.5	1	1	1	3403	1	8551	5634	2160	1549	×
A*1	0.4	0.4	3	2	5	9	4	21	46	473	235	✓
Greedy2	3	2	3	3	4	5	3	9	11	7	6	×
A*2	1	2	7	8	12	17	8	38	48	199	108	✓

Complexidade Espacial (maior número de nós armazenados)

Alg\Depth	2	3	4	5	6	7	8	9	10	12	13	Sol.Ótima
BFS	9	24	61	82	207	628	576	1862	2719	19474	24950	✓
DFS	21	21	21	21	\	\	\	\	\	\	\	×
IDFS	3	4	5	6	7	8	9	10	11	13	14	✓
Greedy1	6	9	17	17	14	2095	14	3173	2955	1633	1500	×
A*1	6	12	25	28	39	63	43	130	214	595	456	✓
Greedy2	6	9	14	14	14	21	12	31	36	27	23	×
A*2	6	12	25	27	36	51	34	105	125	341	226	✓

- **Tempo:** Tal como era de prever, o tempo foi aumentando gradualmente à medida em que o nivel de profundidade da solução ótima crescia. No caso do *BFS*, podemos verificar um crescimento exponencial. Em *DFS*, o tempo aumentou drasticamente da profundidade 2 para 3. Provavelmente, pelo facto de a solução estar um pouco afasta dos ramos da "esquerda"da árvore. Em *IDFS*, o tempo de execução deveria ser semelhante ao do *BFS*, o que não aconteceu, e isto, provavelmente, deve-se ao facto de em *BFS* ser necessário armazenar um enorme número de nós e testar para cada um deles se é ou não solução e ainda é preciso testar se os descendentes de um dado nó já foram visitados ou não enquanto que, no *IDFS*, o número de nós armazenados é sempre pequeno. Com os algorismos informados, *Greedy/A**, os valores foram os esperados uma vez que as soluções foram todas encontradas num curto espaço de tempo.

- **Espaço:** Os valores obtidos para o espaço utilizazdo, da mesma forma que os dados do tempo, coincidiram com o esperado uma vez que à medida que a profundidade aumenta, o número de nós armazenado aumenta exponencialmente em *BFS*. Nos restantes tipos de busca, é realmente difícel obter um valor esperado de complexidade espacial uma vez que há demasiadas variáveis em jogo.

6 Comentários Finais e Conclusões

Com este trabalho, foi possível observar os diferentes tipos de algorismos de busca/procura em "ação"e determinar quais os melhores para cada situação. Sem sombra de dúvida que, para este jogo específico, o melhor algoritmo observado foi o *A*2*, visto que retornou sempre a solução ótima e, dos que retorna a ótima jogada, foi o mais rápido e o segundo melhor em termos de espaço ocupado.

Quanto às **funções heurísticas**, pelos dados que foram obtidos, a heurística *Manhattan distance* superou a *Número de peças fora do lugar* tanto a nível de complexidade temporal como espacial. Isto resulta do facto de a heurística *Manhattan distance* considerar mais variáveis nos seus cálculos do que a *Número de peças fora do lugar* o que leva a uma precisão maior no que toca a atribuir um valor a cada nó.

De todas as buscas **não informadas**, a que obteve melhores valores foi a *IDFS*, pois conseguiu encontrar solução otiml num curto espaço de tempo. A busca *DFS*, não obteve soluções para profundidades superiores a 6, o que foi esperado devido ao facto de que este algorítmo faz uma pesquisa muito "às cegas". Ao contrário de *DFS*, *BFS* encontrou sempre solução ótimal mas demorou imenso tempo a fazê-lo nos últimos casos. Isto por ter de verificar solução numa lista com um enorme número de nós e fazer uma verificação semelhante numa lista ainda maior de nós já visitados ao formar novos filhos.

Relativamente às buscas **informadas**, apesar da procura *Greedy* obter soluções mais rápidas e, por vezes, com menor espaço ocupado, nem sempre retornou a melhor solução, o que, num jogo como este, é o desejado. Por esta razão, considero a busca *A** superior à *Greedy*.

Em suma, ao resumir tudo a algumas palavras, fica notável que todos os algoritmos de busca têm as suas vantagens e desvantagens e devem ser usados alternadamente conforme o que se deseja duma determinada procura. Tanto pode ser mais benéfico utilizar uma *BFS* como uma *DFS*, por exemplo, dependendo do que se deseja.

Estados Iniciais\Finais utilizados nos testes:

Depth 02: Inicial - 1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 03: Inicial - 1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 04: Inicial - 1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 05: Inicial - 1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 06: Inicial - 1 2 3 4 5 0 6 8 9 11 7 12 13 10 14 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 07: Inicial - 1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 08: Inicial - 6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15; Final - 14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0
Depth 09: Inicial - 1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 10: Inicial - 1 6 2 4 5 10 3 8 13 9 7 11 14 0 15 12; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 12: Inicial - 1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15; Final - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
Depth 13: Inicial - 9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11; Final - 9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11

7 Bibliografia

- *https* : *//www.tutorialspoint.com/python/python_sorting_algorithms.htm*;
- *https* : *//www.dcc.fc.up.pt/ ines/aulas/*1819/*IA/buscas_nao_informadas.pdf*
- *https* : *//www.dcc.fc.up.pt/ ines/aulas/*1819/*IA/buscas_informadas_new.pdf*

- *https* : *//www.cs.bham.ac.uk/ mdr/teaching/modules*04*/java*2*/TilesSolvability.html*
- Artificial Intelligence Modern Approach 3rd Ed - *Stuart Russell & Peter Norvig*

8 Execução dos testes

```
#Busca em largura
>>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12
    ','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3
     4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2
     5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9
     7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10
     15 3 11']:
  if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
    print(bfs(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
  if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
    print(bfs(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
  elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15
    3 11'):
    print(bfs(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))

('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.00099945068359375, '
    Space used:', 9)
('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:',
    0.0020008087158203125, 'Space used:', 24)
('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:',
    0.003998994827270508, 'Space used:', 61)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:',
    0.0039899349212646484, 'Space used:', 82)
('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:',
    0.020986080169677734, 'Space used:', 207)
('Movements from Inicial to Final:', ['u', 'r', 'r', 'd', 'l', 'd', 'r'], 'Seconds it took:'
    , 0.06495904922485352, 'Space used:', 628)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it
    took:', 0.16189789772033691, 'Space used:', 576)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'r'], 'Seconds
     it took:', 0.4917006492614746, 'Space used:', 1862)
('Movements from Inicial to Final:', ['l', 'u', 'r', 'u', 'u', 'r', 'd', 'd', 'r', 'd'], '
    Seconds it took:', 1.9309442043304443, 'Space used:', 2719)
('Movements from Inicial to Final:', ['r', 'u', 'l', 'd', 'd', 'l', 'l', 'u', 'r', 'r', 'd',
     'r'], 'Seconds it took:', 55.61377549171448, 'Space used:', 19474)
('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'd', 'l', 'l', 'u', 'r',
     'r', 'u'], 'Seconds it took:', 151.5786578655243, 'Space used:', 24950)


#Busca em profundidade (Com limite de profundidade 20)
>>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12
    ','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3
     4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2
     5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9
     7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10
     15 3 11']:
  if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
    print(depthfirst(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0',20))
  if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
```

```
    print(depthfirst(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11',20))
  elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11'):
    print(depthfirst(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0',20))

('Movements from Inicial to Final:', ['l', 'l', 'd', 'r', 'r', 'u', 'r', 'u', 'l', 'd', 'd', 'l', 'l', 'u', 'r', 'r', 'u', 'r', 'd', 'd'], 'Seconds it took:', 11.549108982086182, 'Maximum space used:', 21)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'r', 'u', 'r', 'r', 'd', 'l', 'u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'r', 'd'], 'Seconds it took:', 42.63512134552002, 'Maximum space used:', 21)
('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'r', 'd', 'l', 'l', 'u', 'l', 'u', 'r', 'd', 'd', 'r', 'r', 'u', 'l', 'd', 'r'], 'Seconds it took:', 0.689577579498291, 'Maximum space used:', 21)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'r', 'd', 'd', 'r', 'u', 'l', 'u', 'l', 'u', 'r', 'r', 'd', 'l', 'd', 'r', 'd'], 'Seconds it took:', 109.79216718673706, 'Maximum space used:', 21)


#Busca Iterativa Limitada em Profundidade
>>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3 4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9 7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11']:
  if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
    print(depth_firsti(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
  if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
    print(depth_firsti(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
  elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11'):
    print(depth_firsti(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))

('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.0010001659393310547, 'Maximum space used:', 3)
('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:', 0.0019986629486083984, 'Maximum space used:', 4)
('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:', 0.007996320724487305, 'Maximum space used:', 5)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:', 0.0179898738861084, 'Maximum space used:', 6)
('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:', 0.027985095977783203, 'Maximum space used:', 7)
('Movements from Inicial to Final:', ['u', 'r', 'r', 'd', 'l', 'd', 'r'], 'Seconds it took:', 0.08896303176879883, 'Maximum space used:', 8)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it took:', 0.12092328071594238, 'Maximum space used:', 9)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'r'], 'Seconds it took:', 0.3687739372253418, 'Maximum space used:', 10)
('Movements from Inicial to Final:', ['l', 'u', 'r', 'u', 'u', 'r', 'd', 'd', 'r', 'd'], 'Seconds it took:', 0.6276133060455322, 'Maximum space used:', 11)
('Movements from Inicial to Final:', ['r', 'u', 'l', 'd', 'd', 'l', 'l', 'u', 'r', 'r', 'd', 'r'], 'Seconds it took:', 3.587789297103882, 'Maximum space used:', 13)
('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'd', 'l', 'l', 'u', 'r', 'r', 'u'], 'Seconds it took:', 6.748943328857422, 'Maximum space used:', 14)

```

```
59
60 #Busca gulosa usando heuristica numero pecas fora do lugar
61 >>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12
       ','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3
        4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2
        5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9
        7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10
        15 3 11']:
62   if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
63     print(greedy_h1(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
64   if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
65     print(greedy_h1(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
66   elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15
       3 11'):
67     print(greedy_h1(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))
68
69 ('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.0004963874816894531,
       'Space used:', 6)
70 ('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:',
       0.00049591064453125, 'Space used:', 9)
71 ('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:',
       0.0009920597076416016, 'Space used:', 17)
72 ('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:',
       0.0014872550964355469, 'Space used:', 17)
73 ('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:',
       0.0009899139404296875, 'Space used:', 14)
74 ('Movements from Inicial to Final:', ['r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'd', 'r',
        'u', 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r
       ', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'd', 'r', 'u',
       'u', 'l', 'd', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'r'
       , 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'd', '
       r', 'u', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'l',
        'u', 'r', 'r', 'd', 'l', 'u', 'l', 'u', 'r', 'd', 'd', 'l', 'u', 'r', 'u', 'l', 'l', 'd
       ', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'd', 'l',
       'u', 'r', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'd', 'l', 'u', 'u', 'r', 'd'
       , 'l', 'd', 'r', 'u', 'l', 'u', 'r', 'd', 'l', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'r', '
       u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'r',
        'd', 'l', 'u', 'r', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'l', 'u', 'r
       ', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'u', 'l', 'd', 'd', 'r', 'u', 'u', 'l', 'd',
       'r', 'd', 'l', 'u', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'l'
       , 'u', 'r', 'd', 'l', 'l', 'u', 'r', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'u', 'l', 'd', '
       r', 'r'], 'Seconds it took:', 3.403381109237671, 'Space used:', 2095)
75 ('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it
       took:', 0.0009911060333251953, 'Space used:', 14)
76 ('Movements from Inicial to Final:', ['l', 'd', 'r', 'd', 'l', 'd', 'r', 'u', 'r', 'd', 'l',
        'u', 'u', 'l', 'u', 'r', 'd', 'd', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'd', 'r', 'r', 'u
       ', 'l', 'd', 'r', 'u', 'l', 'u', 'r', 'd', 'd', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l',
       'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'l', 'u', 'u', 'r', 'd', 'l', 'd'
       , 'r', 'u', 'u', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r', '
       u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'r', 'd',
        'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'r
       ', 'd', 'l', 'l', 'u', 'r', 'r', 'd', 'l', 'd', 'r', 'u', 'u', 'l', 'd', 'd', 'r', 'u',
       'l', 'd', 'r', 'u', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'r', 'd', 'l', 'u', 'u', 'r', 'd'
       , 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', '
       u', 'u', 'r', 'd', 'd', 'r', 'u', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'l',
        'd', 'r', 'u', 'u', 'l', 'l', 'd', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'l', 'd
       ', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'l', 'u',
```

```
    'r', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'u', 'l', 'l', 'u', 'r', 'd'
    , 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'l', 'd', 'r', 'u', '
    r', 'd', 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd',
     'r', 'r', 'u', 'l', 'd', 'r', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'l', 'l', 'd', 'r
    ', 'r', 'r', 'u', 'l', 'l', 'd', 'l', 'u', 'r', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u',
    'l', 'd', 'l', 'l', 'u', 'r', 'r', 'r', 'd', 'l', 'l', 'l', 'u', 'r', 'r', 'r', 'd', 'l'
    , 'l', 'l', 'u', 'r', 'r', 'd', 'r', 'u', 'l', 'd', 'l', 'l', 'u', 'r', 'd', 'l', 'u', '
    r', 'd', 'r', 'r', 'u', 'l', 'l', 'l', 'd', 'r', 'u', 'r', 'r', 'd', 'l', 'l', 'l', 'u',
     'r', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'l', 'u', 'r', 'r
    ', 'r', 'd', 'l', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'd', 'r',
    'u', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'l', 'u', 'r', 'r', 'd', 'l', 'u', 'l', 'd', 'r'
    , 'u', 'l', 'd', 'r', 'r'], 'Seconds it took:', 8.550615549087524, 'Space used:', 3173)
77 ('Movements from Inicial to Final:', ['l', 'u', 'r', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r',
     'u', 'l', 'l', 'd', 'r', 'r', 'd', 'l', 'u', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'u', 'l
    ', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'd', 'd', 'r', 'u', 'l', 'u', 'r',
    'u', 'l', 'd', 'd', 'r', 'u', 'l', 'd', 'r', 'd', 'l', 'u', 'r', 'u', 'l', 'd', 'r', 'd'
    , 'l', 'u', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'l', 'u', 'r', 'd', '
    l', 'u', 'r', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'd', 'r', 'd', 'l', 'u', 'u', 'l', 'd',
     'r', 'u', 'l', 'u', 'r', 'r', 'd', 'l', 'd', 'l', 'u', 'r', 'u', 'l', 'd', 'r', 'r', 'u
    ', 'l', 'l', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'l', 'd', 'r',
    'u', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'd', 'r', 'u', 'l', 'u', 'r', 'd', 'd', 'l'
    , 'u', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'u', 'l', 'd', 'd', 'r', '
    u', 'l', 'u', 'r', 'd', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'd', 'r', 'u', 'r', 'u', 'l',
     'l', 'd', 'd', 'r', 'u', 'r', 'u', 'l', 'l', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'r', 'd
    ', 'd', 'l', 'u', 'r', 'u', 'l', 'd', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'u', 'r',
    'd', 'l', 'd', 'r', 'u', 'u', 'l', 'l', 'd', 'r', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'd'
    , 'r', 'u', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'r', 'd', 'd', 'l', 'u', 'r', 'u', 'l', '
    d', 'd', 'r', 'u', 'l', 'l', 'u', 'r', 'r', 'd', 'd', 'l', 'l', 'u', 'u', 'r', 'd', 'l',
     'd', 'r', 'r', 'u', 'l', 'u', 'l', 'd', 'd', 'r', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd
    ', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l',
    'd', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'r', 'u', 'l'
    , 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'r', 'u', '
    l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l',
     'u', 'l', 'd', 'r', 'r'], 'Seconds it took:', 5.63426661491394, 'Space used:', 2955)
78 ('Movements from Inicial to Final:', ['l', 'l', 'd', 'r', 'r', 'r', 'u', 'l', 'l', 'd', 'l',
     'u', 'r', 'r', 'r', 'u', 'l', 'd', 'l', 'd', 'l', 'u', 'u', 'r', 'd', 'l', 'u', 'r', 'd
    ', 'l', 'u', 'r', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'd', 'r', 'u', 'l',
    'd', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'u', 'r', 'd'
    , 'l', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'u', 'r', '
    r', 'd', 'l', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r',
     'u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'd', 'l', 'u', 'r', 'u', 'l', 'd', 'd', 'r', 'u
    ', 'l', 'l', 'd', 'l', 'u', 'r', 'r', 'r', 'd', 'l', 'l', 'u', 'l', 'd', 'r', 'u', 'r',
    'd', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'u'
    , 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'u', 'l', 'l', 'd', 'r', 'u', 'l', 'd', '
    r', 'r', 'r', 'u', 'l', 'l', 'l', 'd', 'r', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd',
     'l', 'u', 'r', 'r', 'd', 'l', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'l', 'u', 'r
    ', 'r', 'r', 'r', 'd', 'l', 'l', 'u', 'l', 'd', 'r', 'r', 'r', 'u', 'l', 'l', 'd', 'r',
    'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'r', 'r'
    , 'u', 'l', 'l', 'd', 'r', 'u', 'r', 'd', 'r'], 'Seconds it took:', 2.1601641178131104, '
    Space used:', 1633)
79 ('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'l', 'l', 'd', 'r', 'u',
     'l', 'd', 'r', 'u', 'r', 'u', 'l', 'l', 'd', 'r', 'u', 'l', 'd', 'r', 'u', 'r', 'd', 'l
    ', 'u', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l', 'd',
    'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'd', 'r', 'u', 'u', 'l', 'd', 'r'
    , 'u', 'l', 'd', 'd', 'r', 'u', 'u', 'l', 'd', 'r', 'd', 'l', 'u', 'r', 'u', 'l', 'd', '
    r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u',
     'l', 'd', 'l', 'u', 'r', 'r', 'd', 'l', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l', 'd', 'r
```

```
    ', 'r', 'u', 'l', 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'r', 'u', 'l', 'd', 'l', 'u', 'r',
    'r', 'd', 'l', 'l', 'u', 'r', 'd', 'r', 'u', 'l', 'l', 'd', 'r', 'u', 'r', 'd', 'l', 'u'
    , 'l', 'd', 'r', 'r', 'u', 'l', 'd', 'l', 'u', 'r', 'r'], 'Seconds it took:',
    1.5494897365570068, 'Space used:', 1500)
80
81
82 #Busca gulosa usando heuristica de manhattan distance
83 >>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12
    ','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3
     4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2
     5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9
     7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10
     15 3 11']:
84   if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
85     print(greedy_h2(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
86   if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
87     print(greedy_h2(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
88   elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15
     3 11'):
89     print(greedy_h2(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))
90
91 ('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.0029954910278320312,
    'Space used:', 6)
92 ('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:',
    0.0020046234130859375, 'Space used:', 9)
93 ('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:',
    0.00299835205078125, 'Space used:', 14)
94 ('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:',
    0.002997159957885742, 'Space used:', 14)
95 ('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:',
    0.003996849060058594, 'Space used:', 14)
96 ('Movements from Inicial to Final:', ['u', 'r', 'r', 'd', 'l', 'd', 'r'], 'Seconds it took:'
    , 0.004996776580810547, 'Space used:', 21)
97 ('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it
    took:', 0.0029973983764648438, 'Space used:', 12)
98 ('Movements from Inicial to Final:', ['d', 'l', 'd', 'r', 'r', 'd', 'r', 'u', 'l', 'l', 'l',
     'd', 'r', 'u', 'r', 'r', 'd'], 'Seconds it took:', 0.00899505615234375, 'Space used:',
    31)
99 ('Movements from Inicial to Final:', ['l', 'u', 'r', 'r', 'r', 'd', 'l', 'u', 'l', 'u', 'u',
     'r', 'd', 'd', 'd', 'r'], 'Seconds it took:', 0.010993003845214844, 'Space used:', 36)
100 ('Movements from Inicial to Final:', ['r', 'u', 'l', 'd', 'd', 'l', 'l', 'u', 'r', 'r', 'd',
     'r'], 'Seconds it took:', 0.006995201110839844, 'Space used:', 27)
101 ('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'd', 'l', 'l', 'u', 'r',
     'r', 'u'], 'Seconds it took:', 0.005995988845825195, 'Space used:', 23)
102
103
104 #Busca A* com a heuristica numero pecas fora do lugar
105 >>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12
    ','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3
     4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2
     5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9
     7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10
     15 3 11']:
106   if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
107     print(Astar_h1(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
108   if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
109     print(Astar_h1(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
```

```
  elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11'):
    print(Astar_h1(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))

('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.0004944801330566406, 'Space used:', 6)
('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:', 0.0004954338073730469, 'Space used:', 12)
('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:', 0.002529621124267578, 'Space used:', 25)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:', 0.001981973648071289, 'Space used:', 28)
('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:', 0.005024909973144531, 'Space used:', 39)
('Movements from Inicial to Final:', ['u', 'r', 'r', 'd', 'l', 'd', 'r'], 'Seconds it took:', 0.00942373275756836, 'Space used:', 63)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it took:', 0.00396728515625, 'Space used:', 43)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'r'], 'Seconds it took:', 0.020829439163208008, 'Space used:', 130)
('Movements from Inicial to Final:', ['l', 'u', 'r', 'u', 'u', 'r', 'd', 'd', 'r', 'd'], 'Seconds it took:', 0.046123504638671875, 'Space used:', 214)
('Movements from Inicial to Final:', ['d', 'l', 'l', 'u', 'r', 'r', 'r', 'u', 'l', 'd', 'd', 'r'], 'Seconds it took:', 0.4733090400695801, 'Space used:', 595)
('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'd', 'l', 'l', 'u', 'r', 'r', 'u'], 'Seconds it took:', 0.23460650444030762, 'Space used:', 456)


#Busca A* com a heuristica manhattan distance
>>> for i in ['1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12','1 2 3 4 5 6 0 7 9 10 11 8 13 14 15 12','1 2 3 4 5 0 7 8 9 6 10 12 13 14 11 15','1 2 3 0 5 6 8 4 9 10 7 12 13 14 11 15','1 2 3 4 5 0 6 8 9 11 7 12 13 10 14 15','1 2 3 4 5 10 6 7 9 0 12 8 13 14 11 15','6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15','1 0 3 4 6 2 7 8 5 14 10 12 9 13 11 15','1 6 2 4 5 10 3 8 13 9 7 11 14 0 15 12','1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15','9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11']:
  if i=='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15':
    print(Astar_h2(i,'14 6 12 9 7 2 5 11 8 4 13 15 3 10 1 0'))
  if i=='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11':
    print(Astar_h2(i,'9 5 12 7 14 13 0 8 1 3 2 4 6 10 15 11'))
  elif (i!='6 12 0 9 14 2 5 11 7 8 4 13 3 10 1 15' and i!='9 12 0 7 14 5 13 2 6 1 4 8 10 15 3 11'):
    print(Astar_h2(i,'1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0'))

('Movements from Inicial to Final:', ['r', 'd'], 'Seconds it took:', 0.0009984970092773438, 'Space used:', 6)
('Movements from Inicial to Final:', ['r', 'd', 'd'], 'Seconds it took:', 0.0019989013671875, 'Space used:', 12)
('Movements from Inicial to Final:', ['d', 'r', 'd', 'r'], 'Seconds it took:', 0.0069959163665771484, 'Space used:', 25)
('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r'], 'Seconds it took:', 0.007994651794433594, 'Space used:', 27)
('Movements from Inicial to Final:', ['r', 'd', 'l', 'd', 'r', 'r'], 'Seconds it took:', 0.011992692947387695, 'Space used:', 36)
('Movements from Inicial to Final:', ['u', 'r', 'r', 'd', 'l', 'd', 'r'], 'Seconds it took:', 0.016989946365356445, 'Space used:', 51)
('Movements from Inicial to Final:', ['l', 'l', 'd', 'd', 'r', 'r', 'r', 'd'], 'Seconds it took:', 0.007994413375854492, 'Space used:', 34)
```

```
142 ('Movements from Inicial to Final:', ['d', 'l', 'd', 'd', 'r', 'u', 'r', 'd', 'r'], 'Seconds
        it took:', 0.037975311279296875, 'Space used:', 105)
143 ('Movements from Inicial to Final:', ['l', 'u', 'r', 'u', 'u', 'r', 'd', 'd', 'r', 'd'], '
       Seconds it took:', 0.04797053337097168, 'Space used:', 125)
144 ('Movements from Inicial to Final:', ['d', 'l', 'l', 'u', 'r', 'r', 'r', 'u', 'l', 'd', 'd',
        'r'], 'Seconds it took:', 0.19887733459472656, 'Space used:', 341)
145 ('Movements from Inicial to Final:', ['l', 'd', 'r', 'r', 'd', 'l', 'd', 'l', 'l', 'u', 'r',
        'r', 'u'], 'Seconds it took:', 0.10793328285217285, 'Space used:', 226)
```