UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Atividade prática 1 - Busca A*

Florianópolis
2025

Jader Theisges (22215141)
Abmael Batista da Silva (22203744)

Instruções:

Estrutura do código

Principais classes e funções:

A classe Node é a abstração de um nó em uma árvore de busca, e armazena as seguintes variáveis:

```
def __init__(self, state, parent=None, action=None, g=0, h=0):
    self.state = state
    self.parent = parent
    self.action = action
    self.g = g  #metrica g: custo acumulado desde o estado inicial ate o
no corrente.
    self.h = h  #metrica h: estimativa heuristica do custo do no corrente
ao estado-meta.

def f(self):
    return self.g + self.h
```

Funções auxiliares:

find_empty_tile: localiza a posição vazia (0).

get_successors: gera os estados vizinhos (expansão).

reconstruct_path: reconstroi o caminho da solução.

visualize_path: exibe passo a passo do caminho encontrado.

Funções de heurísticas:

As funções de heurística são o coração do A*, pois define como será escolhido o próximo nó, na sequência de busca, abaixo estão as funções que utilizamos no código para cada heurística.

- **heuristica_custo_uniforme:** h(n) = 0 (Busca de Custo Uniforme). (Dijkstra.)
- **heuristica_nao_admissivel**: 2 × Manhattan (superestima o custo).
- **heuristica_simples_admissivel**: contagem de peças fora do lugar (Hamming).
- **heuristica_manhattan_admissivel**: distância Manhattan (admissível e mais precisa).

Funcão principal do A*:

a_star_search:
Essa função usa heapq para a fronteira.

Tem um dicionário visitados para armazenar o menor custo conhecido para cada estado.

Coleta métricas de desempenho durante a execução.

Gerenciamento da fronteira

Implementamos a fronteira como uma fila de prioridade com o **heapq**. É um módulo da biblioteca do Python que implementa uma fila de prioridade usando a estrutura de dados heap binário (min-heap). Ou seja, sempre expande o nó com menor valor f(n).

Na lógica, antes de adicionar um sucessor:

- É calculado o custo acumulado (variável g).
- Se o estado já existe em visitados com custo menor ou igual, nao adicionamos ele na fronteira.
- Se não, o custo é atualizado em visitados e o nó é inserido na fronteira.

```
if new_state_tuple in visitados and visitados[new_state_tuple] <= new_g:
    continue

visitados[new_state_tuple] = new_g
new_h = heuristic_func(new_state_list, goal_state)
successor_node = Node(state=new_state_list, parent=current_node,
action=action, g=new_g, h=new_h)
heapq.heappush(fronteira, successor_node)
```

Heurísticas

Heurística de Custo Uniforme:

Retorna sempre zero. Nesse caso, o algoritmo se reduz à Busca de Custo Uniforme, que é equivalente ao Dijkstra. Pode ser extremamente ineficiente, pois não usa nenhuma informação sobre o problema.

Heurística Simples Admissível (Hamming):

Conta o número de peças que não estão em sua posição correta, ignorando a peça vazia. Simples de implementar, porém é pouco informativa, pois trata todas as peças fora do lugar como “1 passo”, mesmo que algumas estejam longe da posição final.

Heurística Admissível Precisa (Manhattan)

Para cada peça, calcula a distância Manhattan (|linha atual – linha final| + |coluna atual – coluna final|) até sua posição correta. Somar as distâncias para todas as peças. Muito mais informativa do que Hamming, pois distingue peças “próximas” de peças “distantes”.

Heurística Não Admissível (2 × Manhattan)

Calcula a mesma distância Manhattan, mas multiplica por 2, propositalmente superestimando o custo real. Em alguns casos pode reduzir o número de expansões, porque força o algoritmo a explorar caminhos mais diretos

Output - Comparação em execuções:
Fácil:

```
[[1, 2, 3], [4, 5, 6], [0, 7, 8]]
```

```
=== comparacao final ===
custo uniforme                      | nodos:      6 | caminho:   2 |
tempo: 0.000229s | fronteira:     6
a* nao admissivel                   | nodos:      3 | caminho:   2 |
tempo: 0.000067s | fronteira:     3
a* admissivel simples               | nodos:      3 | caminho:   2 |
tempo: 0.000052s | fronteira:     3
a* admissivel precisa               | nodos:      3 | caminho:   2 |
tempo: 0.000057s | fronteira:     3
```

médio:

```
[[1, 0, 3], [4, 2, 5], [7, 8, 6]]
```

```
=== comparacao final ===
custo uniforme                      | nodos:     13 | caminho:   3 |
tempo: 0.000642s | fronteira:    10
a* nao admissivel                   | nodos:      4 | caminho:   3 |
tempo: 0.000155s | fronteira:     6
a* admissivel simples               | nodos:      4 | caminho:   3 |
tempo: 0.000132s | fronteira:     6
a* admissivel precisa               | nodos:      4 | caminho:   3 |
tempo: 0.000127s | fronteira:     6
```

difícil:

```
=== comparacao final ===
custo uniforme                      | nodos:  85525 | caminho:  22 |
tempo: 1.672795s | fronteira: 24969
a* nao admissivel                   | nodos:     65 | caminho:  26 |
tempo: 0.001161s | fronteira:    46
a* admissivel simples               | nodos:   6489 | caminho:  22 |
tempo: 0.101431s | fronteira:  3737
a* admissivel precisa               | nodos:    330 | caminho:  22 |
tempo: 0.004721s | fronteira:   213
```

Comparação - heurística admissível simples VS heurística admissível precisa

Fácil:
a* admissivel simples | nodos: 3 | caminho: 2 | tempo: 0.000103s | fronteira: 3
a* admissivel precisa | nodos: 3 | caminho: 2 | tempo: 0.000074s | fronteira: 3

Médio:
a* admissivel simples | nodos: 4 | caminho: 3 | tempo: 0.000125s | fronteira: 6

a* admissivel precisa | nodos: 4 | caminho: 3 | tempo: 0.000126s | fronteira: 6

Difícil:

a* admissivel simples | nodos: 6489 | caminho: 22 | tempo: 0.101282s | fronteira: 3737
a* admissivel precisa | nodos: 330 | caminho: 22 | tempo: 0.004851s | fronteira: 213

Comparação - Custo Uniforme VS heurística não admissível

Fácil:

custo uniforme | nodos: 6 | caminho: 2 | tempo: 0.000685s | fronteira: 6
a* nao admissivel | nodos: 3 | caminho: 2 | tempo: 0.000111s | fronteira: 3

médio:

custo uniforme | nodos: 13 | caminho: 3 | tempo: 0.000544s | fronteira: 10
a* nao admissivel | nodos: 4 | caminho: 3 | tempo: 0.000161s | fronteira: 6

difícil:

custo uniforme | nodos: 181439 | caminho: 31 | tempo: 3.774536s | fronteira: 25142
a* nao admissivel | nodos: 423 | caminho: 31 | tempo: 0.006260s | fronteira: 265