Universidade Federal do Rio Grande do Sul **Inteligência Artificial**
Instituto de Informática **Prof. Joel Luís Carbonera**
Departamento de Informática Aplicada

Trabalho 3 – Busca em Grafos

Instruções preliminares

As implementações devem ser feitas em Python 3. Assuma que os códigos serão executados em uma máquina com interpretador Python 3.8, com Miniconda, Pip, numba, numpy, pandas). Caso precise de instalar bibliotecas adicionais, descreva-as no seu Readme.md

IMPORTANTE: Não serão aceitas entregas que utilizem bibliotecas que resolvam diretamente os problemas (oferecendo implementações do que se pede), embora seja possível utilizar bibliotecas que implementem estruturas de dados (e.g. fila, pilha, etc.) e rotinas auxiliares, (e.g. leitura de arquivos).

Introdução

Nesta lista, vamos trabalhar com o 8-puzzle, para exercitar o básico do A*.

O 8-puzzle é composto por uma moldura 3x3 contendo um conjunto de peças numeradas de 1 a 8 e um espaço vazio. Uma peça vizinha ao espaço vazio pode ser deslizada para ele. O objetivo é alcançar um estado onde as peças numeradas estão em ordem. A Fig. 1 mostra um exemplo.

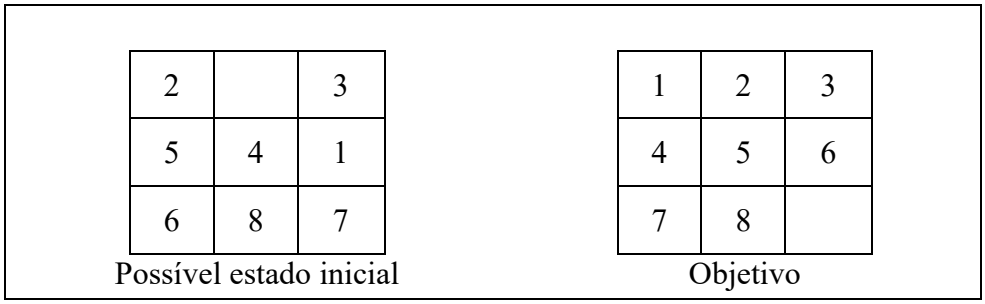

Fig. 1 – Instância do 8-puzzle

Você pode praticar o jogo online aqui. Também existem solvers disponíveis online, como este. Mas as soluções para alguns algoritmos podem diferir, em função de modificações sutis nas implementações. Assim, não é recomendável assumir as respostas destes solvers como sendo "as corretas".

Modelagem

O estado do 8-puzzle será representado como uma string contendo a sequência dos números e o espaço vazio do quebra-cabeça. Assim, o estado inicial na Fig. 1 é representado como: `"2_3541687"`. O estado objetivo possui a seguinte representação: `"12345678_"`.

Uma ação no 8-puzzle corresponde a deslizar uma peça para o espaço vazio. Reciprocamente, estamos "deslizando" o espaço vazio na direção da peça. Assim, as ações representadas serão: acima, abaixo, esquerda e direita, correspondendo à direção que estamos "movendo" o espaço vazio. No estado inicial da Fig. 1, as ações possíveis são: esquerda, abaixo e direita. Ao aplicarmos a ação "esquerda" no estado inicial (`"2_3541687"`), alcançamos o estado `"_23541687"`, representado na Fig. 2.

	2	3
5	4	1
6	8	7

Fig 2 – Estado alcançado ao executarmos a ação 'esquerda' no estado inicial da Fig. 1

1) **Exercício 1: sucessor**

A função `sucessor(estado)` recebe um estado e retorna um conjunto de tuplas (ação, estado atingido) para cada ação que pode ser realizada no estado recebido como parâmetro. O par contém a ação que pode ser realizada no estado que foi recebido como parâmetro e o estado atingido ao se realizar aquela ação. Por exemplo, se sua função é chamada com entrada `"2_3541687"`, ela deve retornar uma lista com 3 tuplas (ação, estado):

```
{(esquerda,"_23541687"), (abaixo,"2435_1687"),  (direita,"23_541687")}
```

A ordem das tuplas não é importante. Ou seja, para a mesma entrada (`"2_3541687"`), a saída abaixo também está correta:

```
{(esquerda,"_23541687"), (direita,"23_541687"), (abaixo,"2435_1687")}
```

Implemente a função `sucessor`.

2) **Exercício 2: classe nodo**

No grafo de busca, cada nó contém informações sobre o estado a que ele se refere, além de informações que irão auxiliar a busca por uma solução.

As arestas do grafo de busca são as ações e cada nó deve ter os seguintes atributos:

- Estado: representação do estado ao qual este nó se refere (e.g.: `"_23541687"`)
- Pai: referência ao nó que precede este
- Ação: ação que foi aplicada ao nó pai para gerar este
- Custo do caminho (path cost): o custo do caminho a partir do estado inicial até este nó. No caso do 8-puzzle, cada ação (aresta do grafo) terá custo 1. Assim, o custo de um nó é o custo do pai + 1.
- (opcional) referências para os nós filhos.

A Fig. 3 ilustra alguns nós do 8-puzzle:

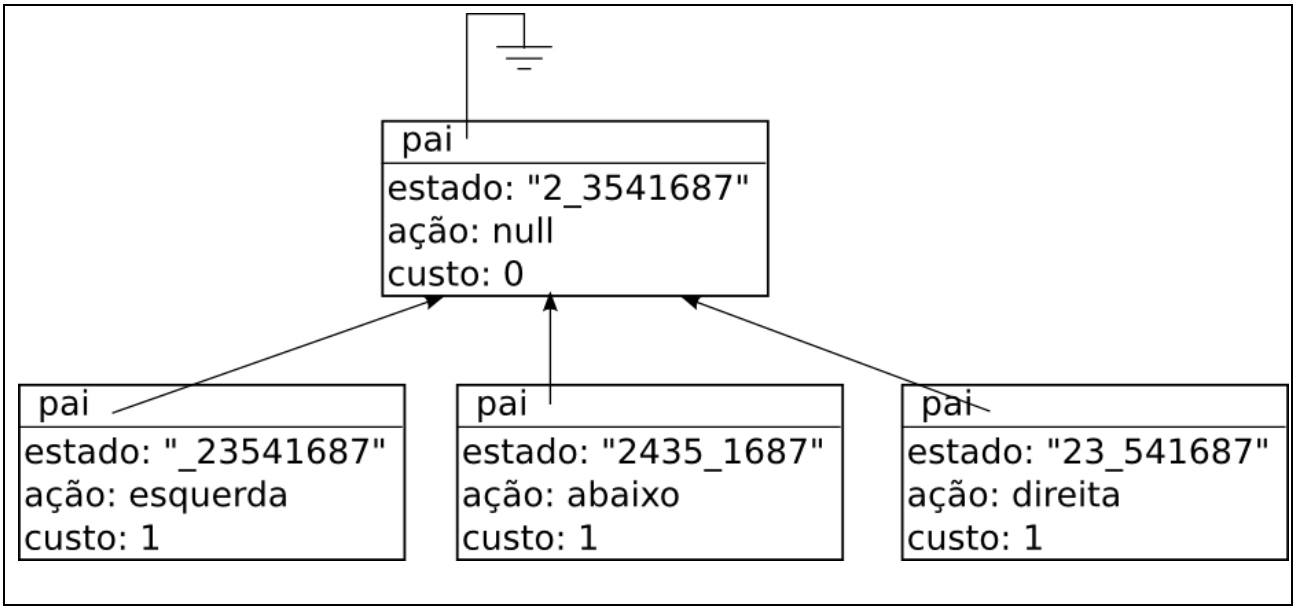

Figura 3 – Representação dos nós do 8-puzzle. Nó raiz corresponde ao estado inicial da Fig. 1.

Implemente a classe `Nodo` com os atributos `estado`, `pai`, `acao` e `custo` , cujo significado é descrito acima. Você pode implementar quantos métodos e atributos adicionais quiser, mas esses quatro atributos básicos devem ter exatamente esses nomes.

Pode ser útil implementar funções para teste de igualdade (`__eq__` chamada quando objetos são comparados com `==`) e hashing (`__hash__` chamada quando objetos são inseridos em sets ou dicts). Para mais informações, consulte: https://docs.python.org/3/reference/datamodel.html.

ATENÇÃO: Note que comparar nodos é diferente de comparar estados (o conteúdo do nodo)! Confundir ambos pode acarretar diversos problemas.

3) Exercício 3: expansão

A partir da função `sucessor` e da classe `Nodo`, você será capaz de construir a função `expande` e um grafo de busca a partir de um estado inicial.

A função `expande` recebe um nó (objeto da classe `Nodo`) e retorna um conjunto de nós que são sucessores do nó recebido, usando a função `sucessor`. Por exemplo, ao passar o nó raiz da Fig. 3, os três nós inferiores são retornados.

Implemente a função `expande(nodo)`, a qual recebe um nodo (objeto da classe `Nodo`) e retorna um conjunto (ou lista, ou qualquer `iterable`) de nodos. Por exemplo, se a função for chamada com um node cujo estado é `2_3541687` e o custo é zero, ela deve retornar um conjunto contendo nodos com os seguintes atributos (novamente, a ordem não é importante):

- Nodo 1
  - acao: esquerda
  - estado: _23541687
  - pai: (referencia para o nodo recebido na funcao expande)
  - custo: 1
- Nodo 2
  - acao: abaixo
  - estado: 2435_1687

    - pai: (referencia para o nodo recebido na funcao expande)
    - custo:1

- Nodo 3
    - acao: direita
    - estado: 23_541687
    - pai: (referencia para o nodo recebido na funcao expande)
    - custo: 1

4) A*

Com a função `expande`, será possível implementar a busca em grafos. Neste procedimento, os nós expandidos pertencerão ao conjunto `explorados` (ou fechado/expandido) e os candidatos pertencerão à `fronteira` (ou aberto). A maneira como a `fronteira` é implementada dá origem aos diferentes algoritmos de busca (BFS, DFS, A*, etc.). Observe o pseudocódigo genérico de busca em grafos (adaptado dos slides), onde `s` é o estado inicial, `X` é o conjunto de explorados e `F` é a fronteira:

```
busca_grafo(s, objetivo):
    X ← {}
    F ← {new Node(s)}
loop:
    se F = ∅: retornar FALHA
    v ← retira(F)
    se v é o objetivo: retornar caminho s-v
    se v ∉ X:
      Insere v em X
      Cria Node pra cada u vizinho de v e insere em F (se u ∉ X)
```

O caminho s-v é a sequência de ações que levam do estado inicial (`s`) até o objetivo encontrado (`v`).

O que você deve fazer:

Implemente duas versões do algoritmo A*. Ambas vão instanciar o pseudocódigo genérico usando uma fila de prioridades como `fronteira`. Ou seja, a função de retirar da fila irá retornar o nó `v` com menor custo `f(v) = g(v) + h(v)`. Nessa função de custo, `g(v)` é o custo do caminho do estado inicial até `v` e `h(v)` é a distância heurística (estimativa) de v até o objetivo. Cada versão do A* usará uma heurística:

A. Implemente a função `astar_hamming(estado)`, que recebe o estado inicial (string) e executa o A* com `h(v)` sendo o número de peças fora do lugar (distância de Hamming);
B. Implemente a função `astar_manhattan(estado)`, que recebe o estado inicial (string) e executa o A* com `h(v)` sendo a soma da distância Manhattan de cada a peça para seus respectivos lugares corretos. A distância Manhattan conta quantos movimentos horizontais e verticais são necessários para chegar à posição correta.

IMPORTANTE: todas as funções deste exercício recebem o estado inicial e retornam uma lista (ou outro `iterable`, mas a ordem é importante!) com as ações que levam do estado inicial recebido para a solução. Caso o estado recebido já seja a solução, as funções devem retornar uma lista vazia. Caso não haja solução a partir do estado inicial recebido, as funções devem retornar `None`, como indicativo

de falha

Por exemplo, para a entrada `“123456_78”`, as funções devem retornar: `[“direita”, “direita”]`. Para a entrada `“185423_67”`, as funções devem retornar `None`.

IMPORTANTE: Espera-se que as funções retornem a resposta para qualquer entrada dentro do tempo-limite de 1 minuto.

Critérios de correção

Item	**Pontos**
Implementação da função sucessor	5
Implementação da função expande	10
A* - heurística Hamming - função `astar_hamming(estado)`	40
A* - heurística Manhattan - função `astar_manhattan(estado)`	40
Misc (e.g. estrutura de diretórios correta? Informações sobre o grupo no Readme.md completas?)	5
Extras (satura em 10 pontos): busca em largura (função `bfs`, 5 pontos), busca em profundidade (função `dfs`, 5 pontos), outra heurística pro A* (função `astar_new_heuristic`, 5 pontos).	10
Total	**110**

Entrega

Você deve entregar um arquivo `.zip` contendo:

- O arquivo `solucao.py` com as funções preenchidas para os exercícios deste trabalho
- Todos os arquivos auxiliares do seu código-fonte.
- Um arquivo `Readme.md` de texto sem formatação (ou formatado em Markdown) contendo os nomes, cartões de matrícula e turma dos integrantes do grupo. Descreva também as bibliotecas que precisem ser instaladas para executar sua implementação. Ao final, descreva também, para cada algoritmo:
    - Quantos nós são expandidos, o tempo decorrido e qual o custo da solução encontrada para o estado inicial `“2_3541687”`.
- (opcional) Um script `prepara.sh`, que instale as bibliotecas adicionais que você porventura use. Pode ser via `pip`, `conda` ou `apt`.

ATENÇÃO: o `solucao.py`, o `Readme.md` e o `prepara.sh` devem se localizar na raiz do seu arquivo `.zip`! Isto é, o conteúdo do seu arquivo `.zip` deverá ser o seguinte:

```
solucao.py
Readme.md
prepara.sh [opcional]
[demais arquivos de código fonte] <<pode criar subdiretórios se necessário
```

Conteúdo do arquivo `.zip` a ser enviado.

Observações

- Você pode adotar internamente outra representação para o estado do 8-puzzle, desde que seu programa aceite como entrada e escreva na saída a representação convencionada neste enunciado.
- A estrutura de dados da fronteira tem impacto direto no tempo de execução do A*. Uma implementação com uma lista linear pode demorar muito. Portanto, é aconselhável usar uma estrutura de dados mais inteligente.
- Garanta que sua implementação respeite a especificação descrita aqui. Haverá um corretor automatizado para avaliar as implementações. Caso a especificação não seja respeitada, seu trabalho não será avaliado adequadamente. Para verificar que sua implementação está aderente à especificação, execute o `testa_solucao.py` do kit e corrija eventuais erros até passar em todos os testes (o que ainda não garante que sua implementação está correta, mas pelo menos ela respeita a interface especificada). Você pode criar testes adicionais para checar outras coisas no seu código.

Observações gerais

- O trabalho deve ser feito em grupos.
- Fiquem atentos à política de plágio!
- É a corretude da sua implementação - e não o julgamento do corretor automatizado - que dará o veredito final da sua nota. Os envios também são revisados manualmente para garantir que seu grupo receba a pontuação apropriada para o trabalho. Porém, notem que é importante não haver erros provocados por não aderência à especificação do trabalho (e.g. coisas que "quebram" o corretor, como trocar o tipo dos argumentos ou do retorno das funções). Esse tipo de erro é considerado como implementação incorreta.

Política de Plágio

Grupos poderão apenas discutir questões de alto nível relativas a resolução do problema em questão. Poderão discutir, por exemplo, questões sobre as estruturas de dados na implementação da fronteira, as heurísticas do A*, vantagens e desvantagens, etc. Não é permitido que os grupos utilizem quaisquer códigos fonte provido por outros grupos, ou encontrados na internet ou gerados automaticamente por ferramentas baseadas em IA (com ChatGPT, copilot, e semelhantes).

Pode-se fazer consultas na internet ou em livros apenas para estudar o modo de funcionamento das técnicas de IA, e para analisar o pseudo-código que as implementa. O objetivo deste trabalho é justamente implementar as técnicas do zero e descobrir as dificuldades envolvidas na sua utilização para resolução do problema em questão.

Usamos rotineiramente um sistema anti-plágio que compara o código-fonte desenvolvido pelos grupos com soluções enviadas em edições passadas da disciplina, e também com implementações disponíveis online.

Qualquer nível de plágio (ou seja, utilização de implementações que não tenham sido 100% desenvolvidas pelo grupo) poderá resultar em nota zero no trabalho. Caso a cópia tenha sido feita de outro grupo da disciplina, todos os envolvidos (não apenas os que copiaram) serão penalizados. Se você tiver quaisquer dúvidas se uma determinada prática pode ou não, ser considerada plágio, pergunte ao professor.

Note que, considerando-se os pesos das avaliações desta disciplina (especificados e descritos no Plano de Ensino) nota zero em qualquer um dos trabalhos de implementação abaixa muito a média final dos projetos práticos, e se ela for menor que 6, não é permitida prova de recuperação. Ou seja: caso seja detectado plágio, há o risco direto de reprovação. Os grupos deverão desenvolver o trabalho sozinhos.