8-Puzzle

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Trabalho Prático - Grafos
Projeto e Análise de Algoritmos - 2018/2

Setembro de 2018

1 Resumo

Neste trabalho prático, você implementará um algoritmo capaz de resolver o jogo *8-puzzle*. Se você não conhece esse jogo, é possível jogá-lo em: `www.mypuzzle.org/sliding`. Você terá que implementar a solução, coletar resultados, e criar um relatório sobre seu trabalho.

Leia com bastante atenção este documento, que é dividido da seguinte forma:

2. **Introdução**: apresentação do problema e algumas definições teóricas
3. **Outros conceitos importantes**: conceitos relevantes para o problema e implementação
4. **Implementação**: detalhes sobre a implementação
5. **Entrada e saída**: especificação de entrada e saída e alguns casos de teste
6. **Submissão**: informações sobre a submissão
7. **Detalhes e sugestões**: questões para ficarem atentos e algumas dicas

2 Introdução

Em jogos de tipo *puzzle* o desafio geralmente é, a partir de um estado inicial, fazer uso de ações disponíveis que podem nos levar, rodada a rodada, até o estado final desejado. Assim, em jogos deste tipo, é possível definir:

- **Estado inicial:** estado inicial fornecido.
- **Ações:** todas ações possíveis de serem tomadas a partir de um estado s.
- **Estados ou espaço de estados:** todos os estados possíveis de serem alcançados a partir do estado inicial utilizando um conjunto de ações.
- **Caminho:** sequência de ações que chegam até determinado estado.
- **Custo:** custo associado a uma ação ou caminho.
- **Teste final:** determina se um estado s é ou não estado final desejado.

Uma instância do jogo *N-puzzle* consiste em um tabuleiro hxh com $N = h^2 - 1$ quadrados móveis, numerados de 1 a N, mais um espaço vazio, representado pelo número 0. O espaço vazio pode ser trocado com qualquer quadrado adjacente, e dado um **estado inicial** do tabuleiro, devemos encontrar uma sequência de movimentos ou **ações** que chegue até o **estado final**, isto é, o tabuleiro cujos quadrados sigam ordem numérica ascendente $(0, 1, \ldots, N)$.

Neste trabalho, focaremos no caso em que $h = 3$, ou seja, no 8-puzzle. A posição do espaço em branco pode ser trocada com a de quadrados adjacentes utilizando ações em quatro direções: *"Cima"*, *"Baixo"*, *"Esquerda"* e *"Direita"*, sendo permitido apenas uma troca de cada vez. A Figura 1 mostra um exemplo de transição de estado, no qual o estado $s2 = 1, 2, 5, 3, 4, 0, 6, 7, 8$ é alcançado a partir de $s1 = 1, 2, 0, 3, 4, 5, 6, 7, 8$ através de uma ação *"Cima"*. O **custo** de uma mudança é sempre 1 e portanto, o custo total de um **caminho** entre dois estados é igual a quantidade de ações tomadas.

1	2	5
3	4	
6	7	8

$\Longrightarrow$

1	2	
3	4	5
6	7	8

Figura 1: Uma transição de estado no jogo 8-puzzle: o estado $s2$ (à direita) é resultado de uma ação *"Cima"* a partir de $s1$ (esquerda).

3 Outros conceitos importantes

3.1 Árvore de busca

O espaço de estados pode ser representado por uma árvore, chamada de árvore de busca, que modela a sequência de ações de um jogo. Sua raiz é o estado inicial, suas arestas as ações, e seus nós os resultados de cada ação. Os nós possuem diversos atributos como: pai, filho, profundidade, e claro, o estado associado. Com uma árvore de busca, é possível encontrar um caminho que vai desde o estado inicial até estado final, ou seja, uma **solução** do jogo.

No espaço de estados, existem três regiões:

1. Explorados: conjunto de nós visitados (e expandidos);
2. Fronteira: próximos nós a serem visitados (filhos de nós visitados/explorados);
3. Inexplorados.

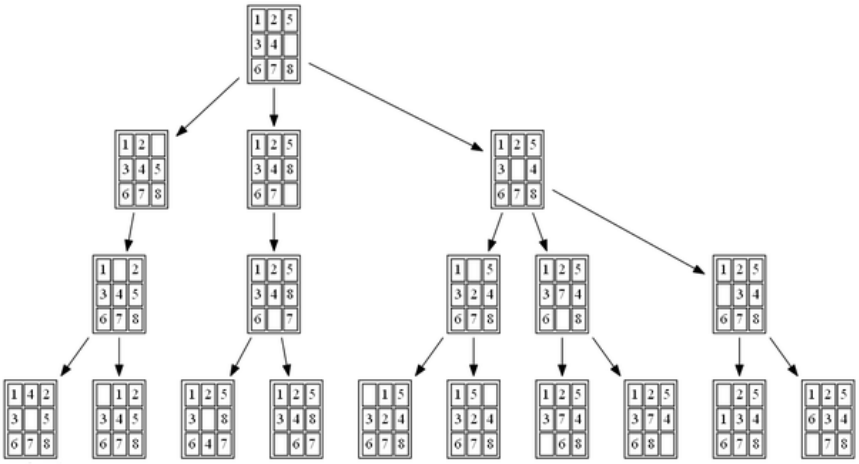

Figura 2: Exemplo de árvore de busca

A Figura 2 mostra um exemplo de árvore de busca, expandida até o terceiro nível. A raiz representa o estado inicial $1, 2, 5, 3, 4, 0, 6, 7, 8$. As ações *"Cima"*, *"Baixo"*, *"Esquerda"* e *"Direita"* são omitidas na figura. Todos os nós com filhos são nós expandidos (no conjunto de explorados), os nós ainda não expandidos (terceiro e último nível) estão na fronteira, e quaisquer outros nós que possam entrar na árvore são inexplorados. Note que o segundo nó da esquerda do terceiro nível representa o estado final.

3.2 Algoritmo de busca

Um algortimo de busca implementa uma estratégia para mover nós da região 3 para 2 e posteriormente 1, aonde o **teste final** é executado. No exemplo da Figura 2, o um algoritmo de busca a ser implementado deve ser capaz de encontrar um caminho como: "Cima", "Esquerda", "Esquerda", que leva a uma solução, mostrada na Figura 3.

Neste trabalho, você deve implementar um algoritmo de busca que encontre a **solução ótima** (de menor custo) de uma instância do jogo 8-puzzle caso ela exista. Lembre-se que o custo de cada ação é 1 e que uma sequência ótima nunca passa duas vezes pelo mesmo estado.

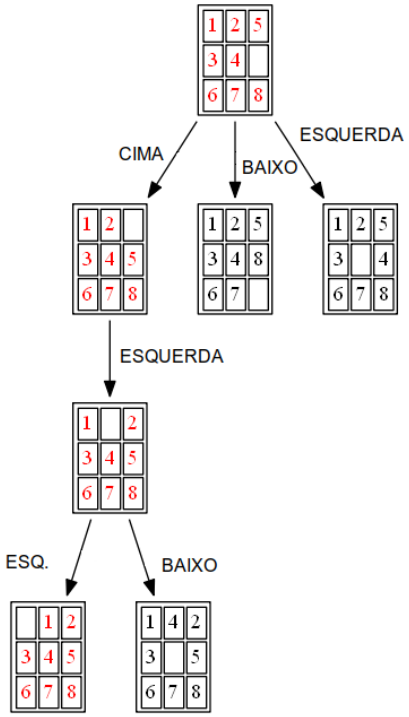

Figura 3: Exemplo de solução da instância $1, 2, 5, 3, 4, 0, 6, 7, 8$ do 8-puzzle.

4 Implementação

As implementações devem ser testadas em uma máquina Linux do Departamento de Ciência da Computação de livre acesso aos alunos da pós-graduação via acesso remoto [1]. Essa é a garantia de que a implementação será compilada e executada em um ambiente conhecido pelo aluno.

Você pode implementar na linguagem de programação de sua preferência.

Não é permitido o compartilhamento de código entre os estudantes.

O código implementado deve ser compilado da seguinte maneira:

```
$ ./compilar.sh
```

O shell script "compilar.sh"deverá conter:

```
#!/bin/bash
< código para compilar seu programa >
```

O código implementado deverá ser executado com a seguinte linha de comando:

```
$ ./executar.sh entrada saída
```

O shell script "executar.sh"deverá conter:

```
#!/bin/bash
in = $1
out = $2
< código para executar o seu programa de modo que a entrada seja lida em $in e a saída em $out >
```

5 Entrada e saída

Tanto entrada quanto saída são bem simples. A entrada é composta apenas pelo estado inicial, indicado pelos números e separados por vírgula. A saída, por sua vez, deve ser da seguinte forma:

[1] https://www.crc.dcc.ufmg.br/bc/howto/remote/ssh

```
custo_total: [o número de ações tomadas para chegar ao estado final]
```

5.1 Exemplos

```
Entrada:
1,2,5,3,4,0,6,7,8
Saída:
custo_total: 3

Entrada:
6,1,8,4,0,2,7,3,5
Saída:
custo_total: 20

Entrada:
8,6,4,2,1,3,5,7,0
Saída:
custo_total: 26
```

Mais exemplos serão disponibilizados.

6 Submissão

Deve-se submeter um arquivo **.zip** contendo os códigos-fonte, *shell scripts* e o relatório, até as 23h55 do dia 17 de outubro de 2018 na págino do curso no moodle.

6.1 Relatório

O relatório não deve exceder 10 páginas e deverá contemplar os seguintes itens:

- Introdução com explicação clara e objetiva de como o problema foi resolvido e justificativas de quaisquer decisões feitas por você no decorrer do desenvolvimento, como em relação a estruturas de dados usadas, por exemplo. Não é necessário incluir trechos do seu código, mas pode fazer se assim preferir. Utilize o que achar conveniente, seja diagramas, pseudo-códigos, figuras, etc.
- Análise de complexidade de tempo e espaço do seu algoritmo utilizando o formalismo da notação assintótica. Aqui, gere suas próprias entradas, meça o tempo de execução e consumo de memória e depois apresente os resultados da maneira que achar melhor (tabelas, gráficos, etc).
- Análise teórica de complexidade de tempo e espaço do seu algoritmo para o caso genérico do *N-puzzle* ($N = h^2 - 1,\ h = 3, 4, 5, ...$). Nesse caso não é necessário gerar entradas, apenas resolver analiticamente.
- Questão extra (desafio): Utilize seu algoritmo para determinar o pior caso para *8-puzzle*.
- Questão extra (desafio++): Estenda seu algoritmo para solucionar de maneira ótima o *15-puzzle* e determine seu pior caso.

Para tópicos sobre complexidade assintótica, você pode fazer o número de nós em função de parâmetros como: d (profundidade da solução), b (fator de ramificação máxima dos nós ou número de ações) e m (profundidade máxima do espaço de estados).

Os dois itens extras são opcionais.

7 Detalhes e sugestões

1. Seu trabalho será avaliado a partir dos seguintes critérios:
   - A qualidade do código (código bem organizado, com comentários explicativos, variáveis com nomes intuitivos, modularidade, etc).
   - Execução correta do código em entradas de testes. Além dos casos de testes disponibilizados, serão utilizados outros casos de testes a fim de verificar a corretude da solução proposta.
   - Conteúdo do relatório, que deve conter os itens mencionados anteriormente, além do esforço em escrever um texto coeso, organizado e de fácil compreensão.
2. Diversas configurações iniciais do *8-puzzle* não são solucionáveis. Isso não afetará os exemplos deste trabalho, mas esteja ciente de que você pode encontrar um desses casos em algum momento.
3. Incluia uma maneira de lidar com estados repetidos no seu algoritmo de busca.
4. Mantenha variáveis que possam te ajudar no entendimento do algoritmo como o número de nós expandidos, a profundidade da busca e da solução. Elas podem também ajudar a montar o relatório.
5. Se você tiver que fazer uma escolha aleatória em relação a ordem de visita dos nós filhos, use CBED (Cima, Baixo, Esquerda, Direita).
6. Implemente operações com complexidade menor que linear. Você não quer operações $O(n)$, sendo o n o número de nós da árvore.
7. Mesmo quem é bastante proficiente em C, C++ ou Java pode achar mais fácil e rápido implementar em Python.
8. Após submeter seu arquivo .zip no moodle, faça o download da submissão e verifique que o mesmo não está corrompido. Não será dada segunda chance de submissão para arquivos corrompidos.
9. **Comece o trabalho o quanto antes.**