# Primeiro Trabalho de IA/SI: Buscas

## Entrega: 03/03/2019 (2 semanas)

18 de Fevereiro de 2019

Este trabalho é para ser submetido via Moodle. Será desenvolvido principalmente durante as aulas práticas, mas espera-se que o estudante complemente com trabalho extra-classe.

**Os testes e exames terão perguntas relacionadas com este trabalho.** Para saber o método e critério de avaliação, por favor consulte a ficha da unidade curricular na página do sigarra.

## 1 Descrição do Problema

O jogo dos 15 é representado por uma matriz 4x4 onde há 15 células numeradas e uma célula em branco. Variações deste jogo podem conter parte de uma imagem em cada célula. O problema consiste em partir de uma configuração inicial embaralhada das células e chegar a uma configuração final com uma ordenação determinada de algarismos (no caso da matriz de números) ou de imagens (no caso da matriz onde as células representam partes de uma imagem). Os movimentos/operadores possíveis para se chegar de uma configuração a outra são: 1) mover a célula em branco para cima, 2) mover a célula em branco para baixo, 3) mover a célula em branco para a direita e 4) mover a célula em branco para a esquerda.

**Nota:** Para alguns conjuntos de configurações iniciais e finais, este problema, assim como a sua versão reduzida – jogo dos oito, não tem solução. Investigue sobre este assunto e implemente no seu programa, um código que verifique se, para uma dada configuração inicial do jogo dos 15, há um caminho que leve à solução final, **SEM** fazer nenhuma busca. (Na página da disciplina `http://www.dcc.fc.up.pt/~ines/aulas/1819/IA/IA.html`, poderá encontrar alguns *links* relevantes sobre a dualidade de problemas deste tipo e dicas de como economizar memória).

## 2 Implementação

Implemente as estratégias de busca:

- profundidade
- largura
- iterativa em profundidade
- A*
- gulosa (*greedy*) com heurística

Aplique estas estratégias ao problema do jogo dos 15 descrito na seção 1. No caso da implementação das estratégias A* e gulosa, utilize duas heurísticas: (a) somatório do número de peças fora do lugar e (b) manhattan distance (somatório das distâncias de cada peça ao seu lugar na configuração final).

Dadas as configurações inicial e final (que são argumentos de entrada para o programa), deve imprimir os operadores utilizados no caminho que leva à solução, número de passos, tempo de execução e quantidade de memória utilizada (contabilizada como número máximo de nós armazenados simultaneamente em memória durante a execução).

Compare a sua implementação com os resultados produzidos pelos seguintes métodos de procura:

- **busca em profundidade** (com verificação de ciclos em cada caminho!)
- **busca em largura**

A implementação pode ser em qualquer linguagem.

Para cada estratégia, analise:

- tempo para chegar a uma solução,
- quantidade de espaço gasto (número de nós gerados e armazenados),
- completude (o algoritmo consegue encontrar uma solução?) e
- otimalidade (Qual é a profundidade da solução encontrada? A solução encontrada corresponde ao menor número de passos para chegar da configuração inicial à final ou com menor custo utilizando pouco tempo e pouca memória?)

Utilize gráficos ou tabelas para comparar as várias estratégias.

Utilize as seguintes configurações iniciais de teste:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 8 | 12 |
| 13 | 9 | | 7 |
| 14 | 11 | 10 | 15 |

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 13 | 6 | 8 | 12 |
| 5 | 9 | | 7 |
| 14 | 11 | 10 | 15 |

Utilize a seguinte configuração final para testar as duas configurações iniciais sugeridas:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | |

A primeira configuração tem solução ótima em profundidade 12 (menor número de movimentos possível para chegar da configuração inicial à configuração final).

A segunda configuração inicial não leva à configuração final proposta. Explique o porque.

**O seu programa deve estar preparado para receber qualquer configuração inicial ou final num formato em linha**. Por exemplo,

```
1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
```

onde a primeira configuração corresponde à primeira configuração inicial sugerida anteriormente e a segunda configuração corresponde à configuração de teste sugerida também anteriormente.

O seu código deve verificar se há solução para chegar do estado inicial ao estado final **antes** de iniciar a busca, e deve emitir uma mensagem de erro se não houver caminho entre a solução inicial e a final.

Utilize o algoritmo 1 como base para implementar **todas** as buscas (este é o algoritmo geral de busca, mas com um teste inicial específico para jogos do tipo jogo dos 15 - teste de resolubilidade):

**Algorithm 1** Algoritmo Geral de Busca

```
1: GeneralSearchAlgorithm(QueueingFunction,configInicial,configFinal)
2: if thereIsNoSolution(configInicial,configFinal) then
3:     return "It is impossible to reach a solution"
4: end if
5: queue = configInicial
6: while queue not empty do
7:     node = removeFrontNodeFrom(queue)
8:     if node is solution then
9:         return Path to solution
10:    end if
11:    descendantList = MakeDescendants(node)
12:    insert(descendantList,queue,QueueingFunction)
13: end while
14: return "solution not found"
```

Este algoritmo, além de receber as configurações inicial e final, recebe também como parâmetro uma função de enfileiramento que será utilizada para ordenar a lista de nós (configurações) abertos (ainda não explorados).

## 3 Relatório para entrega: pontos a abordar

1. Introdução
   - Definição e descrição de um problema de busca/procura
   - Principais métodos utilizados para resolver problemas de busca e breve discussão sobre suas complexidades de tempo e espaço
2. Estratégias de Procura
   (a) Procura não guiada (blind - "cega")
      - Profundidade (DFS - Depth-First Search: como funciona, quando se aplica, qual é a complexidade temporal e espacial?)
      - Largura (BFS - Breadth-First Search: como funciona, quando se aplica, qual é a complexidade temporal e espacial?)

- Busca Iterativa Limitada em Profundidade (como funciona, quando se aplica, qual é a complexidade temporal e espacial?)

(b) Procura guiada (que usa alguma **heurística** para orientar a procura)

- O que é uma heurística?
- Busca Gulosa (*greedy*)
  - Como funciona e quando se aplica?
  - Qual foi a heurística utilizada para o problema a ser resolvido e por que esta heurística foi escolhida?
- Busca A*
  - Como funciona e quando se aplica?
  - Heurísiticas comumente utilizadas pela estratégia A*

3. Descrição do problema de procura estudado (jogo dos 15) e dos dois espaços de estados

4. Descrição da Implementação

   - Linguagem utilizada? Por que escolheu esta linguagem? Há alguma vantagem em utilizar esta linguagem para resolver este tipo de problema?
   - Estruturas de dados utilizadas? Como foi que escolheu as estruturas de dados? São eficientes para manipular os dados do problema?
   - Estrutura do código?

5. Resultados

   Fazer tabela (ou curvas comparativas) com tempos de execução, utilização de memória e se encontrou a solução, para cada configuração, para cada estratégia, além da profundidade da solução encontrada. Se preferir utilizar uma tabela, esta poderá ter um sumário dos resultados organizados da seguinte forma:

| Estratégia | Tempo (segundos) | Espaço | Encontrou a solução? | Profundidade/Custo |
|---|---|---|---|---|
| DFS | ... | ... | ... | ... |
| BFS | ... | ... | ... | ... |
| IDFS | ... | ... | ... | ... |
| Gulosa | ... | ... | ... | ... |
| A* | ... | ... | ... | ... |

6. Comentários Finais e Conclusões

   Comentar sobre as estratégias fazendo uma comparação entre o seu desempenho e eficácia para encontrar as soluções. Concluir dizendo qual foi a melhor estratégia para este problema.

7. Referências Bibliográficas (precisam ser citadas no texto para saberem de onde o texto foi retirado/adaptado! Copiar é crime e poderá transformar-se em processo disciplinar, portanto evitem copiar textos e códigos. O ideal é ler textos de vários autores, reescrever com suas próprias palavras e dar a sua própria interpretação, mas sempre citando as fontes de onde retiraram as ideias.)

   Se utilizarem figuras retiradas da web ou de livros ou de artigos etc, é necessário colocar uma referência.

   Por favor, mantenham os erros ortográficos num nível mínimo.

## 4 Entrega e Avaliação

Enviar através do Moodle um arquivo zip ou similar contendo o código fonte dos programas, e instruções de como compilar e executar cada problema, isto é, um pequeno manual de como correr os programas (pode ser um 'help' ou um 'readme'). Incluam também bibliotecas especiais e versões de compiladores utilizados. Além disso, devem incluir uma pequena documentação explicando em que ambiente seu programa foi compilado (tipo e versão do SO e da linguagem). Seu programa deve correr na minha máquina (com ubuntu 18.04 LTS). Não assuma que eu tenho um IDE (Integrated Development Environment) de qualquer tipo. **O programa deve compilar e correr na linha de comando**. Prestem especial atenção à utilização de acentuação em textos incluídos nos programas (Java e python, em particular, podem não correr na minha máquina dependendo da codificação de acentuação gráfica que utilizarem: utf8, iso* etc).

O trabalho pode ser feito em grupo de **no máximo duas pessoas**.

Todos os trabalhos deverão ser apresentados em data a combinar. **Todos os componentes do grupo deverão estar presentes durante a demonstração. Um dos componentes do grupo será aleatoriamente escolhido para responder às perguntas formuladas**. Quem não estiver presente vai ter nota zero! Cada componente do grupo deverá comentar sobre sua contribuição no trabalho.

Nos trabalhos em grupo, assegurem-se de que cada um sabe exatamente o que o outro está a fazer para evitar constrangimentos e penalizações durante a sessão de apresentação.