Sistemas Inteligentes
Prof. Elder Rizzon Santos
Universidade Federal de Santa Catarina
Sistemas de Informação

Alunos:
Barbara Idaerla Santos Calderon - 19202599
Edmilson Domingues - 19204766

Atividade Prática 1
Trabalho sobre Métodos de busca (2022/2)

O propósito do trabalho é implementar[1] o algoritmo de busca **A***. A implementação será testada através do jogo 8-puzzle[2], o qual também fornece o contexto para a heurística.

A *entrada* do programa é um tabuleiro desordenado (com o quadrado sem número **em qualquer lugar** do tabuleiro) e um algoritmo de busca (detalhes a seguir). A saída principal do programa **é o menor caminho** (a sequência de movimentos do quadrado sem número) para chegar-se ao tabuleiro ordenado[3]. Além do caminho, ao final, deve ser exibido:

a) O total de nodos visitados
b) O total de nodos expandidos/criados
c) O tamanho do caminho

Para a implementação do algoritmo, a equipe deve implementar 3 variações do algoritmo:

1. Custo Uniforme (sem heurística)
2. A* com uma heurística simples
3. A* com a heurística mais precisa que conseguirem

Juntamente com a implementação (.ZIP) deverá ser entregue um mini-relatório explicando brevemente:

1. **Qual a representação (estrutura de dados) do estado;**

A estrutura de dados que estamos utilizando, tanto para nodos abertos (fronteira) quanto para nodos fechados é uma lista composta pelos seguintes elementos:

- Lista com todos os estados percorridos desde a origem até o nodo de interesse (estado), representando o caminho percorrido
- Valor do custo do nodo de interesse (cada nível percorrido na árvore corresponde a um custo unitário)
- Valor da heurística (estimado para o nodo de interesse - estado, até o nodo objetivo)
- Valor total para a função de avaliação, correspondente ao percurso da origem até o nodo objetivo e estimado como sendo a soma dos dois parâmetros anteriores (custo do nodo e heurística)
- Campo de comentário correspondente ao “nível alcançado” pelo estado de interesse. Este campo foi utilizado para avaliação na consistência de resultados e poderia ter sido descartado.

Segue um exemplo real para ilustrar a descrição acima.

Estado final (para o caso do benchmarking fornecido em aula):
[[[6, 7, 5, 1, 2, 3, 9, 4, 8], [6, 7, 5, 9, 2, 3, 1, 4, 8], [6, 7, 5, 2, 9, 3, 1, 4, 8], [6, 9, 5, 2, 7, 3, 1, 4, 8], [9, 6, 5, 2, 7, 3, 1, 4, 8], [2, 6, 5, 9, 7, 3, 1, 4, 8], [2, 6, 5, 1, 7, 3, 9, 4, 8], [2, 6, 5, 1, 7, 3, 4, 9, 8], [2, 6, 5, 1, 9, 3, 4, 7, 8], [2, 9, 5, 1, 6, 3, 4, 7, 8], [2, 5, 9, 1, 6, 3, 4, 7, 8], [2, 5, 3, 1, 6, 9, 4, 7, 8], [2, 5, 3, 1, 9, 6, 4, 7, 8], [2, 9, 3, 1, 5, 6, 4, 7, 8], [9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8], [1, 2, 3, 4, 5, 6, 7, 9, 8], [1, 2, 3, 4, 5, 6, 7, 8, 9]], 18, 0, 18, 'Nivel: 18']

Neste caso a origem foi [6, 7, 5, 1, 2, 3, 9, 4, 8] e o objetivo [1, 2, 3, 4, 5, 6, 7, 8, 9].

2. Qual a estrutura de dados para a fronteira e nodos fechados;

A estrutura de dados, tanto dos nodos abertos (fronteira) quanto dos nodos fechados, é formada por uma lista de nodos, onde cada nodo apresenta o formato do item 1 acima.

Como exemplo, no caso testado, em que a configuração inicial foi [9, 2, 3, 1, 5, 6, 4, 7, 8] e o objetivo foi [1, 2, 3, 4, 5, 6, 7, 8, 9], os nodos abertos e fechados foram representados pelas seguintes estruturas finais:

- Nodos abertos:

[
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8], [1, 2, 3, 4, 5, 6, 7, 9, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8]], 4, 2, 6, 'Nivel: 4'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8], [1, 2, 3, 4, 5, 6, 7, 9, 8], [1, 2, 3, 4, 9, 6, 7, 5, 8]], 4, 2, 6, 'Nivel: 4'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8]], 3, 3, 6, 'Nivel: 3'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 5, 9, 6, 4, 7, 8]], 2, 4, 6, 'Nivel: 2'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [9, 2, 3, 1, 5, 6, 4, 7, 8]], 2, 4, 6, 'Nivel: 2'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [2, 9, 3, 1, 5, 6, 4, 7, 8]], 1, 5, 6, 'Nivel: 1']
]

- Nodos fechados:

[
[[[9, 2, 3, 1, 5, 6, 4, 7, 8]], 0, 4, 4],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8]], 1, 3, 4, 'Nivel: 1'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8]], 2, 2, 4, 'Nivel: 2'],
[[[9, 2, 3, 1, 5, 6, 4, 7, 8], [1, 2, 3, 9, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 9, 7, 8], [1, 2, 3, 4, 5, 6, 7, 9, 8]], 3, 1, 4, 'Nivel: 3']
]

Em ambas as estruturas de dados, nodos abertos e nodos fechados, a inserção de novo dado ocorre de modo ordenado, ou seja, o nodo sempre ocupará posição ordenada na lista já ordenada.

3. **Como foi gerenciada a fronteira, verificações, quais etapas foram feitas ao adicionar um estado na fronteira (explicação das estratégias, respectivos métodos e possibilidades além do que foi implementado);**

   A fronteira foi gerenciada da seguinte forma:

   - Retirada do nodo da vez (nodo de menor custo total):

A retirada do nodo se deu sempre a partir da posição ZERO desta lista, independentemente se o método foi o de Custo Uniforme ou A* com heurística. Cada nodo retirado desta lista passou a compor uma variável isolada, representativa do nodo da vez. Uma vez realizado o tratamento, o nodo da vez passou a compor a lista de nodos fechados.

- Método: puzzle.extrai_nodo_da_vez()

- Colocação de novos elementos na lista de nodos abertos (fronteira):
  Avalia-se cada nodo da vez para verificar se o mesmo corresponde ao objetivo do jogo (configuração final pretendida). Em não sendo, é verificada a existência de nodos filhos. Para cada nodo filho é avaliado se o mesmo não está presente nem na lista de abertos e nem na lista de fechados. Caso não esteja em nenhum dos dois, ele é inserido na lista de abertos, de modo ordenado, e finaliza-se a avaliação. Já na condição de este já estar na lista de abertos, então mantém-se o nodo com menor custo total na lista (nodo da vez versus nodo na lista). Caso o valor do nodo da vez venha a ser menor, este assume a lista de abertos e o outro vai para a lista de nodos fechados e encerra-se a avaliação. Por fim, na condição de este já existir na lista de fechados, então coloca-se o nodo com menor custo na lista de abertos (por meio do método coloca_em_abertos, que garante sua inserção na lista ordenada), eliminando-o da lista de fechados.
  - Métodos:
    - puzzle.coloca_em_abertos(caminho);
    - puzzle.avalia_substituicao_em_abertos(filhos[i])
    - puzzle.avalia_substituicao_fechados(filhos[i])

4. **Descrição das heurísticas e algumas simulações dos seus valores (pior caso, melhor caso, caso médio); breve descrição sobre suas implementações;**

Heurística simples: A heurística adotada corresponde à contagem simples de quantos (dos 8 estados) não são os mesmos entre a origem e o objetivo.

Heurística mais precisa: Adotamos a heurística apresentada no livro do Luger, onde a mesma é estimada como o somatório de casas a serem percorridas entre a casa de origem até a casa final, para cada peça (1 a 8).

Dados de Entrada	Expectativa (Nro. passos até objetivo)	Custo Uniforme			Heurística Simples			Heurística Mais Precisa		
		Resultados (passos)	Nro Abertos	Nro Fechados	Resultado (passos)	Nro Abertos	Nro Fechados	Resultado (passos)	Nro Abertos	Nro Fechados
[1, 2, 3, 4, 5, 6, 7, 9, 8]	1	1	2	1	1	2	1	1	2	1
[1, 2, 3, 4, 5, 6, 9, 7, 8]	2	2	10	6	2	3	2	2	3	2
[1, 2, 3, 9, 5, 6, 4, 7, 8]	3	3	28	15	3	5	3	3	5	3
[9, 2, 3, 1, 5, 6, 4, 7, 8]	4	4	92	52	4	6	4	4	6	4
[2, 9, 3, 1, 5, 6, 4, 7, 8]	5	5	476	271	5	8	5	5	8	5
[2, 3, 9, 1, 5, 6, 4, 7, 8]	6	6	668	372	6	9	6	6	9	6
[2, 3, 6, 1, 5, 9, 4, 7, 8]	7	7	3.740	2.127	7	11	7	7	11	7
[9, 1, 2, 5, 6, 3, 4, 7, 8]	8	8	4.796	2.644	8	14	8	8	14	8
Benchmarking: [6, 7, 5, 1, 2, 3, 9, 4, 8]	18	[1]	[1]	[1]	[2]	[2]	[2]	18	411	255

Observações:
[1] Não alcançou o resultado com 2.000 iterações
[2] Não alcançou o resultado com 15.000 iterações

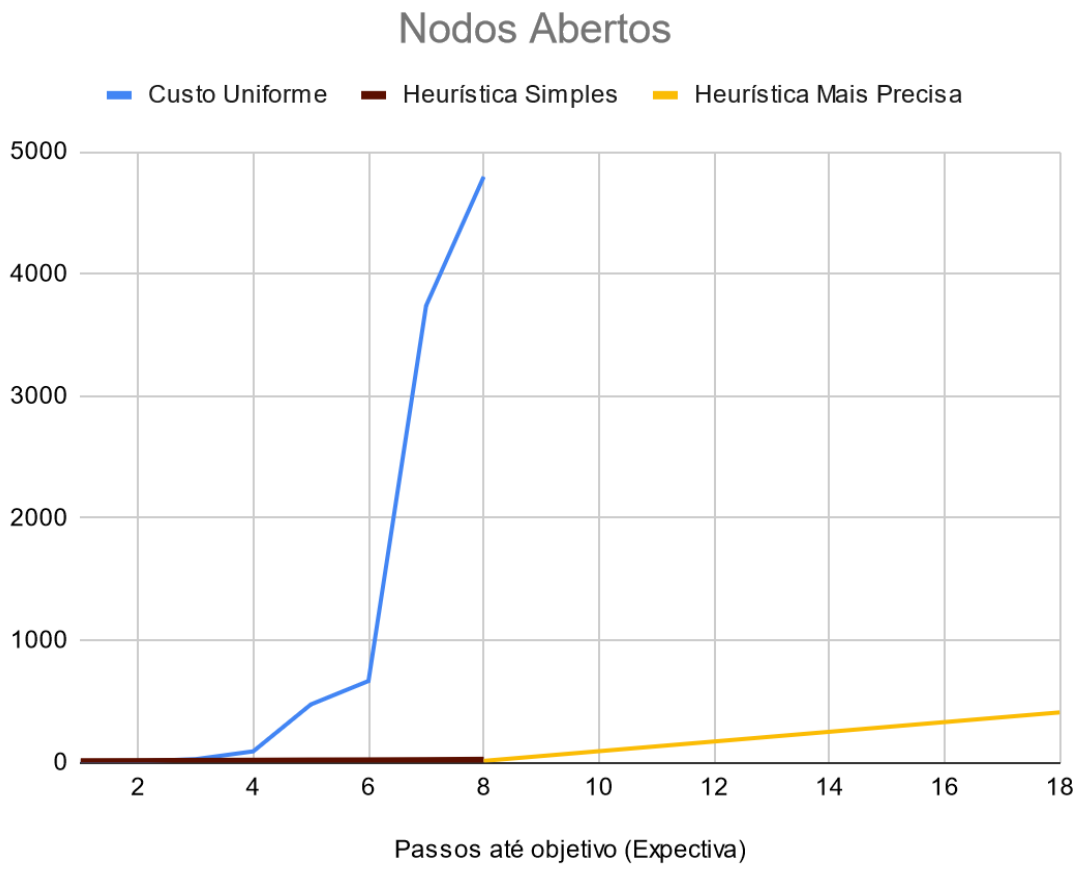
A planilha acima nos permite fazermos alguns comparativos acerca dos resultados obtidos:

- Na avaliação vertical, ou seja, dentro do mesmo método, podemos verificar que à medida que a quantidade de passos esperados aumenta, aumenta a quantidade de nodos abertos e fechados, sinalizando uma maior dificuldade em alcançar-se o objetivo
- Na avaliação horizontal, ou seja, permeando-se os métodos, podemos verificar que para uma dada expectativa de passos esperados, e também analisando-se os nodos abertos e fechados (ao final), o método com heurística mais precisa tem mais facilidade de alcançar o objetivo, em relação ao método com heurística simples. Também este último apresenta mais facilidade em relação ao método do Custo Uniforme.

A planilha permite-nos ainda observar alguns fatos:

- Alguns resultados não foram alcançados após muitas iterações. Isto fica evidente quando observamos o crescimento dos nodos abertos em relação à

expectativa do resultado. Isto nos remete ao fato de que muitos empates (de custo final) devam estar ocorrendo entre os nodos, o que leva à expansão da fronteira.



O gráfico permite-nos observar o quanto cada incremento de passo implicou em aumento na quantidade de nodos, para cada um dos métodos utilizados. Com 18 passos, para os algoritmos Custo Uniforme e Heurística Simples, os resultados não foram alcançados (até as iterações apresentadas na tabela). No entanto, para o método “Heurística Mais Precisa” o resultado foi alcançado e com um número de Nodos Abertos relativamente “baixos”.

5. **Quais os métodos principais e breve descrição do fluxo do algoritmo;**

Complementando o que foi exposto na questão 3, onde apresentamos o fluxo principal de avaliação dos estados, e suas respectivas chamadas aos métodos, aqui vamos apresentar estes métodos propriamente.

a) puzzle.calcula_heuristica_simples(nodo_fillho)
   - Entrada: O nodo_filho corresponde ao caminho desde a origem até o nodo filho de interesse.
   - Processamento: A heurística simples avalia o quão distante o estado atual (último estado do caminho nodo filho) está do objetivo, utilizando a definição dada para a heurística mais simples (neste caso a contagem de quantas casas estão fora das casas do objetivo, range de 0-8)
   - Saída: heurística (um número inteiro)

b) puzzle.calcula_heuristica_precisa(nodo_filho)
   - Entrada: O nodo_filho corresponde ao caminho desde a origem até o nodo filho de interesse.
   - Processamento: A heurística precisa avalia o quão distante o nodo filho está do objetivo, utilizando a definição dada para a heurística precisa (neste caso toma-se como referência a disposição atual de peças e a disposição final de peças, e conta-se, quantas casas cada peça precisa se deslocar até sua posição final, fazendo-se então, um somatório destas contagens individuais - range de 0-31)
   - Saída: heurística (um número inteiro)

c) puzzle.extrai_nodo_da_vez()
   - Entrada: vazia
   - Processamento: Retira-se o primeiro nodo da lista ordenada de nodos abertos que, por definição, apresenta o menor custo total (nodo mais à esquerda nesta lista). Isto acontece pelo fato da lista já estar ordenada. Este nodo retirado passa a compor o atributo self.nodo_da_vez da classe.
   - Saída: sem retorno

d) puzzle.get_nodo_da_vez()
   - Entrada: vazia
   - Processamento: retorna o valor do atributo self.nodo_da_vez da classe.
   - Saída: nodo da vez (que é o caminho da origem até o estado atual)

e) puzzle.eh_nodo_objetivo(estado)
   - Entrada: estado representado pelo último nodo do caminho de interesse (normalmente o nodo da vez)
   - Processamento: verifica se corresponde ao estado final (objetivo)
   - Saída: valor booleano, onde *True* confirma que é estado final

f) puzzle.resultado()

- Entrada: vazia
- Processamento: leitura de atributo
- Saída: nodo da vez (que é o caminho da origem até o estado final), significando que o resultado (caminho) foi alcançado

g) puzzle.gera_nodos_filhos(estado)
   - Entrada: estado
   - Processamento: utiliza estrutura de matrizes para montar os filhos a partir de dado estado (1 nodo)
   - Saída: apresenta todos os nodos filhos para o estado da entrada

h) puzzle.esta_em_nodos_abertos(filhos[i])
   - Entrada: nodo filho (último estado)
   - Processamento: Verifica se o nodo filho (estado atual), encontra-se na lista de nodos abertos, na posição de último estado de pelo menos um elemento desta lista
   - Saída: *True* se nodo de entrada (estado) está na lista de nodos abertos

i) puzzle.esta_em_nodos_fechados(filhos[i])
   - Entrada: nodo filho (último estado)
   - Processamento: Verifica se o nodo filho (estado atual), encontra-se na lista de nodos fechados, na posição de último estado de pelo menos um elemento desta lista
   - Saída: *True* se nodo de entrada (estado) está na lista de nodos fechados

j) puzzle.atribui_custos_ao_nodo(filhos[i])
   - Entrada: nodo_filho, que corresponde ao caminho desde a origem até o nodo filho de interesse
   - Processamento: atribui valores aos campos “custo”, “heurística” e “custo total” (custo + heurística) e utiliza o método heurístico de acordo com o método solicitado para a análise em questão. Além disso, também mostra o nível, que representa o número de jogadas desde a origem.
   - Saída: caminho (nodo filho com as informações processadas no método)

k) puzzle.coloca_em_abertos(caminho)
   - Entrada: caminho, que é o nodo de interesse
   - Processamento: coloca, de modo ordenado, o nodo de interesse (caminho) na lista de nodos abertos, permanecendo a lista ordenada.
   - Saída: sem retorno

l) puzzle.retira_de_nodo_da_vez_coloca_em_fechados(nodo_da_vez)
   - Entrada: nodo da vez
   - Processamento: Coloca o nodo da vez na lista de nodos fechados
   - Saída: sem retorno

6. **Caso algum dos objetivos não tenha sido alcançado explique o que você faria VS o que foi feito e exatamente qual o(s) problema(s) encontrado(s), bem como limitações da implementação.**

Para ilustrar uma situação encontrada por nós e que consideramos importante para o entendimento dos métodos de busca, vamos descrevê-la a seguir.

No caso do *benchmarking* colocado em aula, onde a expectativa era ter-se um caminho total de 18 passos (ações), quando utilizamos a heurística mais precisa, conseguimos encontrar o resultado esperado — os nodos abertos terminaram com 1.167 elementos e os nodos fechados com 693 elementos. A partir disto, realizamos uma pequena modificação na inserção de novos elementos na lista de abertos: **antes**, consideramos apenas a avaliação do custo do nodo atual versus custo dos nodos abertos, selecionando o menor entre eles como critério de comparação; **agora**, alteramos o critério para menor ou igual.

Esta alteração foi crucial para finalizarmos o teste com 411(*) elementos em nodos abertos e 255(*) elementos em nodos fechados. Isto posto, mostra-nos que a quantidade de “empates” (mesmo custo total) pode modificar bastante o resultado. É importante citar que também foram alcançados os 18 passos (níveis) esperados. No caso em questão, a heurística estava confinada ao range de 0-31.

O caso acima nos remete aos casos de empate, onde o algoritmo passa a cobrir uma maior amplitude e muitas vezes pode ser bastante demorado se chegar ao resultado.

Extrapolando-se este exemplo para o caso em que a heurística apresenta um valor em um range bastante limitado (0-8, como no caso da heurística simples), certamente os “empates” serão bem maiores, e com isto a amplitude a ser coberta pelo algoritmo será também bem maior. Naquele caso do *benchmarking*, a aplicação já havia iterado 15.000 vezes, sem alcançar o objetivo.

(*) Estes valores encontram-se na última linha da tabela da questão de número 4, referente à “Heurística Mais Precisa”.