UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DO ALGORITMO A* NO JOGO DOS 8

ATIVIDADE 2

ROBERT BIASOLI DREY

CHAPECÓ
2024

SUMÁRIO

1 INTRODUÇÃO

Este relatório técnico apresenta uma análise detalhada da implementação do algoritmo A* no contexto do jogo de 8 peças. O jogo de 8 peças é um quebra-cabeça clássico que envolve a reorganização de peças numeradas em uma grade 3x3, deixando um espaço vazio, até que as peças estejam ordenadas de acordo com uma configuração alvo pré-definida.

O principal objetivo deste trabalho é explorar a eficácia do algoritmo A* na resolução do jogo de 8 peças, considerando sua capacidade de encontrar a solução mais curta e eficiente para o problema. Além disso, pretende-se analisar o desempenho do algoritmo em termos de tempo de execução e uso de recursos computacionais.

No entanto, é importante ressaltar que testar o algoritmo A* no contexto do jogo de 8 peças apresenta desafios significativos, principalmente devido ao curto tempo disponível para execução dos testes. A complexidade computacional do jogo, combinada com a necessidade de explorar várias configurações possíveis, impõe restrições temporais que podem limitar a abrangência dos testes realizados.

Neste relatório, discutiremos os métodos e técnicas utilizados para implementar o algoritmo A* no jogo de 8 peças, bem como os resultados obtidos a partir dos testes realizados. Além disso, serão abordadas possíveis melhorias e extensões para este trabalho, visando aprimorar a eficiência e a precisão da solução proposta.

2 O ALGORITMO

O presente projeto teve início com a entrega aos alunos de uma base de algoritmos em sala de aula, programados em Python, destinados a resolver o Jogo de 8 peças por meio do algoritmo de Busca em Largura (BFS). Utilizamos esse código como ponto de partida para a implementação do algoritmo desejado, o A*, popularmente conhecido como "A Estrela".

Nesse viés, o projeto foi iniciado a partir da função responsável por calcular a heurística do problema, optando por utilizar a Distância de Manhattan. Esta distância é uma métrica de distância em um espaço bidimensional, determinada pela soma das diferenças absolutas entre as coordenadas dos pontos. No contexto específico do jogo de 8 peças, a Distância de Manhattan é uma heurística amplamente empregada para estimar o custo necessário para mover uma peça de sua posição atual para sua posição correta. Essa heurística é obtida pela soma das distâncias horizontais e verticais entre a posição atual de uma peça e sua posição correta no tabuleiro.

No entanto, no contexto específico do código em questão, o jogo não estava armazenado de forma bidimensional, mas sim em um array unidimensional. Isso introduziu um desafio adicional ao desenvolver a heurística, pois não havia uma representação direta das coordenadas em um plano cartesiano. Para contornar essa limitação, foi necessário conceber uma abordagem que permitisse calcular as coordenadas desejadas e atuais das peças dentro do array unidimensional.

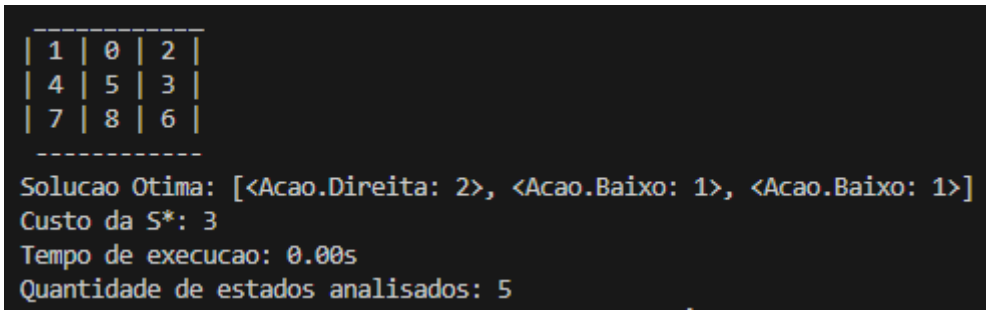
A solução envolveu o uso de operadores específicos do Python, como o operador de módulo (%) e o operador de divisão inteira (//), para mapear as posições no array unidimensional para as coordenadas correspondentes em um plano bidimensional. Esses operadores foram fundamentais para calcular as posições desejadas e atuais das peças no tabuleiro, mesmo com a representação linear do jogo. O desafio foi, portanto, não apenas implementar a heurística de Distância de Manhattan, mas também adaptá-la para funcionar eficientemente com a estrutura de dados unidimensional utilizada no código.

Diante dessa necessidade, foi imperativo ajustar algumas das estruturas fundamentais que já haviam sido estabelecidas no código anterior, relativo ao algoritmo BFS (Busca em Largura). No algoritmo A*, é crucial armazenar não apenas o estado atual, mas também a heurística de cada estado, juntamente com o custo adicional associado a ele. Essas informações são essenciais para a avaliação e seleção dos estados durante o processo de busca, garantindo que o algoritmo A* possa encontrar a solução mais eficiente e ótima possível. Por isso, foi criada a função que expande cada estado com custo. Basicamente, o algoritmo busca expandir o conjunto de estados possíveis, considerando as ações disponíveis para cada estado atual. Durante a expansão, são avaliados diversos critérios, como a validade das ações, a existência de ciclos e o custo associado a cada movimento. Essa função também realiza uma importante otimização, evitando a repetição de ações inversas à última utilizada, o que contribui para um processo de busca mais eficiente. Ao final, a função retorna uma lista de estados expandidos, cada um

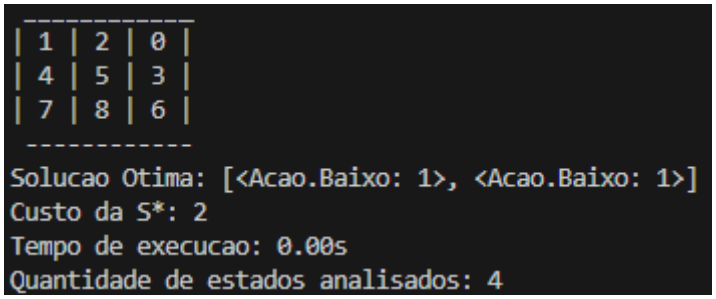
acompanhado de um custo total estimado, que guiará a busca pelo estado objetivo de forma mais inteligente e econômica.

Durante o desenvolvimento do projeto, o estudante experimentou uma montanha-russa de resultados, que variam desde momentos de êxito até momentos de total frustração. Houve ocasiões em que o código rodava com uma eficiência surpreendente, fornecendo soluções instantâneas para casos em que a solução estava a apenas uma ação de distância. Esses momentos proporcionam uma sensação de progresso e validação do trabalho realizado. No entanto, esses momentos de sucesso foram intercalados com longos períodos de espera, nos quais o algoritmo rodava por horas sem apresentar qualquer resultado, mesmo dentro de complexidades consideradas triviais. Essa inconsistência nos resultados gerou uma sensação de perplexidade e dúvida em relação à robustez do código implementado. A alternância entre momentos de sucesso e de estagnação deixou o pesquisador em um estado de incerteza quanto à eficiência do algoritmo, levando-o a questionar se o problema estava na falta de tempo para testes mais abrangentes ou se houve, de fato, alguma falha na implementação do código, visto que não houve erros no console durante as execuções.

As imagens abaixo explicitam o processo descrito anteriormente, durante os testes do algoritmo, algumas soluções foram encontradas instantaneamente, com tempos de execução na ordem de milissegundos, enquanto outras, mesmo com baixa complexidade, demandam horas de processamento sem gerar resultados. Essa discrepância ilustra a imprevisibilidade da resolução de problemas por algoritmos de busca, mesmo em situações consideradas simples.



(Anexo 1 - Teste jogo de 8, com complexidade fácil)



(Anexo 2 - Teste jogo de 8, com complexidade fácil)

3 Considerações finais

O estudo do algoritmo A* nesta pesquisa ressalta a importância de adquirir habilidades em inteligência artificial durante a graduação. Aprender o A* em disciplinas de IA proporciona compreensão sólida sobre busca heurística e resolução de problemas complexos. A transição do BFS para o A* foi desafiadora, mas enriquecedora, permitindo aplicar conceitos teóricos na prática. No entanto, a falta de resultados eficazes nos testes trouxe frustração e incerteza sobre a validade da implementação. Apesar dos obstáculos, essa experiência ressalta a importância da persistência e exploração de diferentes abordagens para resolver problemas desafiadores, fortalecendo o aprendizado e crescimento profissional.

REFERÊNCIAS

PINTO, Ricardo. **Entendendo porque é que a distância certa faz toda a diferença**. 2019. Dissertação. Disponível em: https://medium.com/data-hackers/entendendo-porque-é-que-a-distância-certa-faz-toda-a-diferença-648030c9bae2. Acesso em: 29 mar. 2024.

DicionárioTec. **Entenda o que é o algoritmo A-estrela (A*) em menos de 3 minutos.** 2023. Vídeo. Disponível em: https://www.youtube.com/watch?v=goHblSxcEwA. Acesso em: 29 mar. 2024.