Universidade Federal do Ceará
Campus de Russas

Curso: ciência da computação

Disciplina: Inteligência Artificial

Professor: Alexandre Matos Arruda

Relatório:

Alunos: Victor Wesley - (509906) e João Victor - (519334)
Github: https://github.com/vctrwesley/n-puzzle

1.Busca em largura (BFS);

O algoritmo de Busca em Largura (BFS) foi utilizado para resolver o problema de busca, especificamente o N-puzzle, onde o objetivo era alcançar o estado final a partir do estado inicial fornecido. A seguir, são apresentados os resultados de desempenho do algoritmo, detalhando o uso de memória, a quantidade de nós expandidos, o fator de ramificação médio e o tempo de execução.

Detalhes do Desempenho

- **Uso máximo de memoria:** 66.87890625 MB
- **Quantidade de nós expandidos:** 97278
- **Fator de ramificação media:** 2.7414626123069965
- **Tempo de execução:** 0.37451601028442383

Abaixo são apresentados os passos detalhados pelos quais o algoritmo BFS passou até alcançar a solução:

Descrição dos Passos	
Passo 0:	[7, 0, 6, 1, 5, 2, 3, 4, 8]
Passo 1:	[7, 5, 6, 1, 0, 2, 3, 4, 8]
Passo 2:	[7, 5, 6, 0, 1, 2, 3, 4, 8]
Passo 3:	[0, 5, 6, 7, 1, 2, 3, 4, 8]
Passo 4:	[5, 0, 6, 7, 1, 2, 3, 4, 8]
Passo 5:	[5, 1, 6, 7, 0, 2, 3, 4, 8]

Passo 6:	[5, 1, 6, 0, 7, 2, 3, 4, 8]
Passo 7:	[5, 1, 6, 3, 7, 2, 0, 4, 8]
Passo 8:	[5, 1, 6, 3, 7, 2, 4, 0, 8]
Passo 9:	[5, 1, 6, 3, 0, 2, 4, 7, 8]
Passo 10:	[5, 1, 6, 0, 3, 2, 4, 7, 8]
Passo 11:	[0, 1, 6, 5, 3, 2, 4, 7, 8]
Passo 12:	[1, 0, 6, 5, 3, 2, 4, 7, 8]
Passo 13:	[1, 3, 6, 5, 0, 2, 4, 7, 8]
Passo 14:	[1, 3, 6, 5, 2, 0, 4, 7, 8]
Passo 15:	[1, 3, 0, 5, 2, 6, 4, 7, 8]
Passo 16:	[1, 0, 3, 5, 2, 6, 4, 7, 8]
Passo 17:	[1, 2, 3, 5, 0, 6, 4, 7, 8]
Passo 18:	[1, 2, 3, 0, 5, 6, 4, 7, 8]
Passo 19:	[1, 2, 3, 4, 5, 6, 0, 7, 8]
Passo 20:	[1, 2, 3, 4, 5, 6, 7, 0, 8]
Passo 21:	[1, 2, 3, 4, 5, 6, 7, 8, 0]

Conclusão

A execução do algoritmo BFS demonstrou ser relativamente eficiente, encontrando a solução em 0.374 segundos. No entanto, o uso de memória foi substancial, atingindo um pico de 66.87890625 MB. A quantidade de nós expandidos foi extremamente alta, com um total de 97278 nós, indicando que o BFS explora uma grande quantidade de possibilidades antes de encontrar a solução. O fator de ramificação médio de 2.7414626123069965 mostra que, em média, cada nó gerou aproximadamente 2.74 nós filhos.

Durante a implementação e análise do algoritmo, observamos que o BFS se destaca por garantir a menor solução possível devido à sua exploração completa e sistemática de todas as possibilidades em cada nível antes de avançar. No entanto, esta abordagem pode levar a um uso intensivo da memória e maior tempo de processamento mesmo que o retorno tenha sido bem rápido em todos os testes. Essa característica torna o BFS menos eficiente em termos de recursos computacionais comparado a outros métodos.

2.Busca em profundidade iterativa (IDS);

O algoritmo de Busca em Profundidade Iterativa (IDS) foi executado para resolver o problema de busca no qual o objetivo era alcançar o estado final para solucionar o N-puzzle a partir do estado inicial fornecido. Os resultados do desempenho do algoritmo, incluindo

uso de memória, número de nós expandidos, fator de ramificação médio e tempo de execução, estão detalhados abaixo.

Detalhes do Desempenho

- **Uso máximo de memoria:** 32.80859375 MB
- **Quantidade de nós expandidos:** 71289
- **Fator de ramificação media:** 1.6687146684621752
- **Tempo de execução:** 0.09310102462768555

Abaixo são apresentados os passos detalhados pelos quais o algoritmo IDS passou até alcançar a solução:

Descrição dos Passos	
Passo 0:	[7, 0, 6, 1, 5, 2, 3, 4, 8]
Passo 1:	[7, 5, 6, 1, 0, 2, 3, 4, 8]
Passo 2:	[7, 0, 6, 1, 5, 2, 3, 4, 8]
Passo 3:	[0, 7, 6, 1, 5, 2, 3, 4, 8]
Passo 4:	[1, 7, 6, 0, 5, 2, 3, 4, 8]
Passo 5:	[1, 7, 6, 5, 0, 2, 3, 4, 8]
Passo 6:	[1, 0, 6, 5, 7, 2, 3, 4, 8]
Passo 7:	[0, 1, 6, 5, 7, 2, 3, 4, 8]
Passo 8:	[5, 1, 6, 0, 7, 2, 3, 4, 8]
Passo 9:	[5, 1, 6, 3, 7, 2, 0, 4, 8]
Passo 10:	[5, 1, 6, 3, 7, 2, 4, 0, 8]
Passo 11:	[5, 1, 6, 3, 0, 2, 4, 7, 8]
Passo 12:	[5, 1, 6, 0, 3, 2, 4, 7, 8]
Passo 13:	[0, 1, 6, 5, 3, 2, 4, 7, 8]
Passo 14:	[1, 0, 6, 5, 3, 2, 4, 7, 8]
Passo 15:	[1, 3, 6, 5, 0, 2, 4, 7, 8]
Passo 16:	[1, 3, 6, 5, 2, 0, 4, 7, 8]
Passo 17:	[1, 3, 0, 5, 2, 6, 4, 7, 8]
Passo 18:	[1, 0, 3, 5, 2, 6, 4, 7, 8]
Passo 19:	[1, 2, 3, 5, 0, 6, 4, 7, 8]

Passo 20:	[1, 2, 3, 0, 5, 6, 4, 7, 8]
Passo 21:	[1, 2, 3, 4, 5, 6, 0, 7, 8]
Passo 22:	[1, 2, 3, 4, 5, 6, 7, 0, 8]
Passo 23:	[1, 2, 3, 4, 5, 6, 7, 8, 0]

Conclusão

A execução do algoritmo IDS mostrou-se eficiente em termos de tempo, levando aproximadamente 0.093 segundos para completar a busca. O uso de memória foi relativamente alto, com um pico de 32.80859375 MB. O número de nós expandidos foi significativo, totalizando 71289 nós, o que reflete a natureza exaustiva do algoritmo. O fator de ramificação médio de 1.6687146684621752 indica que, em média, cada nó gerou cerca de 1.67 nós filhos durante a busca.

Durante os estudos para implementar o algoritmo, observamos que o IDS acaba sendo o menos eficiente dos quatro métodos analisados na busca por encontrar a solução. Sem uma limitação adequada e com exemplos difíceis, ele pode demorar muito para encontrar a solução ou, em muitas situações, não conseguir resolvê-lo. Esta limitação destaca a importância de selecionar o algoritmo de busca adequado com base na natureza específica do problema e nas restrições do ambiente onde ele será aplicado.

3.Busca A* com heurística de quantidade de peças erradas;

O algoritmo de Busca A*, foi aplicado ao problema do N-puzzle utilizando uma heurística baseada na quantidade de peças fora de suas posições corretas. A seguir, estão os resultados de desempenho do algoritmo, incluindo uso máximo de memória, número de nós expandidos, fator de ramificação médio e tempo de execução.

Detalhes do Desempenho

- **Uso máximo de memoria:** 34.2421875 MB
- **Quantidade de nós expandidos:** 6600
- **Fator de ramificação media:** 2.7103030303030304
- **Tempo de execução:** 0.6328020095825195

A seguir estão os passos detalhados que o algoritmo A* executou para alcançar a solução:

Descrição dos Passos	
Passo 0:	[7, 0, 6, 1, 5, 2, 3, 4, 8]
Passo 1:	[0, 7, 6, 1, 5, 2, 3, 4, 8]
Passo 2:	[1, 7, 6, 0, 5, 2, 3, 4, 8]
Passo 3:	[1, 7, 6, 5, 0, 2, 3, 4, 8]

Passo 4:	[1, 0, 6, 5, 7, 2, 3, 4, 8]
Passo 5:	[0, 1, 6, 5, 7, 2, 3, 4, 8]
Passo 6:	[5, 1, 6, 0, 7, 2, 3, 4, 8]
Passo 7:	[5, 1, 6, 3, 7, 2, 0, 4, 8]
Passo 8:	[5, 1, 6, 3, 7, 2, 4, 0, 8]
Passo 9:	[5, 1, 6, 3, 0, 2, 4, 7, 8]
Passo 10:	[5, 1, 6, 0, 3, 2, 4, 7, 8]
Passo 11:	[0, 1, 6, 5, 3, 2, 4, 7, 8]
Passo 12:	[1, 0, 6, 5, 3, 2, 4, 7, 8]
Passo 13:	[1, 3, 6, 5, 0, 2, 4, 7, 8]
Passo 14:	[1, 3, 6, 5, 2, 0, 4, 7, 8]
Passo 15:	[1, 3, 0, 5, 2, 6, 4, 7, 8]
Passo 16:	[1, 2, 3, 5, 0, 6, 4, 7, 8]
Passo 17:	[1, 2, 3, 5, 0, 6, 4, 7, 8]
Passo 18:	[1, 2, 3, 0, 5, 6, 4, 7, 8]
Passo 19:	[1, 2, 3, 4, 5, 6, 0, 7, 8]
Passo 20:	[1, 2, 3, 4, 5, 6, 7, 0, 8]
Passo 21:	[1, 2, 3, 4, 5, 6, 7, 8, 0]

Conclusão

A aplicação da heurística de quantidade de peças erradas no algoritmo A* mostrou-se eficiente em termos de uso de memória e tempo de execução, comparando-se favoravelmente a outros métodos, especialmente ao considerar o número relativamente baixo de nós expandidos. No entanto, o tempo de execução de 0.632 segundos foi maior do que em outras abordagens devido à complexidade computacional envolvida no cálculo da heurística.

Durante a implementação do algoritmo A*, constatamos que a escolha de uma heurística adequada é crucial para seu desempenho. A heurística de quantidade de peças erradas proporcionou uma boa estimativa dos custos, embora, para casos mais complexos, heurísticas mais sofisticadas, como a distância de Manhattan, possam oferecer melhor desempenho.

4.Busca A* com heurística de distância de Manhattan;

O algoritmo de Busca A*, foi executado utilizando a heurística de distância de Manhattan para resolver o problema do N-puzzle. Os resultados do desempenho do algoritmo, incluindo uso máximo de memória, número de nós expandidos, fator de ramificação médio e tempo de execução, estão apresentados a seguir.

- **Uso máximo de memoria:** 30.66015625 MB
- **Quantidade de nós expandidos:** 590
- **Fator de ramificação media:** 2.676271186440678
- **Tempo de execução:** 0.009999752044677734

A seguir, são apresentados os passos que o algoritmo executa para alcançar a solução:

Descrição dos Passos	
Passo 0:	[7, 0, 6, 1, 5, 2, 3, 4, 8]
Passo 1:	[0, 7, 6, 1, 5, 2, 3, 4, 8]
Passo 2:	[1, 7, 6, 0, 5, 2, 3, 4, 8]
Passo 3:	[1, 7, 6, 5, 0, 2, 3, 4, 8]
Passo 4:	[1, 0, 6, 5, 7, 2, 3, 4, 8]
Passo 5:	[0, 1, 6, 5, 7, 2, 3, 4, 8]
Passo 6:	[5, 1, 6, 0, 7, 2, 3, 4, 8]
Passo 7:	[5, 1, 6, 3, 7, 2, 0, 4, 8]
Passo 8:	[5, 1, 6, 3, 7, 2, 4, 0, 8]
Passo 9:	[5, 1, 6, 3, 0, 2, 4, 7, 8]
Passo 10:	[5, 1, 6, 0, 3, 2, 4, 7, 8]
Passo 11:	[0, 1, 6, 5, 3, 2, 4, 7, 8]
Passo 12:	[1, 0, 6, 5, 3, 2, 4, 7, 8]
Passo 13:	[1, 3, 6, 5, 0, 2, 4, 7, 8]
Passo 14:	[1, 3, 6, 5, 2, 0, 4, 7, 8]
Passo 15:	[1, 3, 0, 5, 2, 6, 4, 7, 8]
Passo 16:	[1, 0, 3, 5, 2, 6, 4, 7, 8]
Passo 17:	[1, 2, 3, 5, 0, 6, 4, 7, 8]
Passo 18:	[1, 2, 3, 0, 5, 6, 4, 7, 8]

Passo 19:	[1, 2, 3, 4, 5, 6, 0, 7, 8]
Passo 20:	[1, 2, 3, 4, 5, 6, 7, 0, 8]
Passo 21:	[1, 2, 3, 4, 5, 6, 7, 8, 0]

Conclusão

A heurística de distância de Manhattan utilizada no algoritmo A*, demonstrou um desempenho notável, expandindo apenas 590 nós e consumindo 30.660 MB de memória. O tempo de execução de aproximadamente 0,01 segundos reflete a eficiência dessa abordagem.

Durante a implementação, observamos que a heurística de distância de Manhattan, por levar em conta a distância real das peças de suas posições corretas, oferece uma estimativa mais precisa dos custos, resultando em um número significativamente menor de nós expandidos comparado a outras heurísticas, como a de peças erradas.

A vantagem desta heurística é sua capacidade de encontrar soluções rapidamente e de forma eficiente em termos de memória, mesmo para problemas de maior complexidade. Em comparação com a Busca em Largura (BFS) e a Busca em Profundidade Iterativa (IDS), o A* com distância de Manhattan apresentou-se como a opção mais eficaz para resolver o N-puzzle, equilibrando bem o uso de recursos computacionais e a velocidade de execução.

Pontos Fortes e Fracos

	Pontos Fortes	Pontos Fracos
BFS	Garante encontrar a solução mais curta, pois explora todos os nós em cada nível antes de avançar.	Elevado uso de memória e número de nós expandidos. Não é eficiente em termos de recursos computacionais.
IDS	Menor uso de memória comparado ao BFS e tempo de execução relativamente rápido.	A quantidade significativa de nós expandidos pode torná-lo ineficiente para problemas complexos. Pode ser lento e não garantir a solução ótima.
Peças Erradas A*	Menor número de nós expandidos comparado a BFS e IDS. Heurística simples e fácil de implementar.	Tempo de execução relativamente alto devido à complexidade computacional da heurística. A heurística de peças erradas pode não ser a mais eficiente para casos complexos.
Manhattan A*	Melhor desempenho geral, com menor número de nós expandidos e tempo de execução mais rápido. A	Nenhum destacado, sendo a heurística mais eficiente e eficaz para resolver o N-puzzle entre as abordagens comparadas.

	heurística de distância de Manhattan fornece uma estimativa mais precisa dos custos.	

Considerações Finais

Melhor Desempenho:

- A com Heurística de Distância de Manhattan:* Este algoritmo apresentou o melhor equilíbrio entre uso de memória, tempo de execução e quantidade de nós expandidos. É altamente eficiente para resolver o N-puzzle, especialmente para casos de maior complexidade.

Eficiência em Memória:

- Busca em Profundidade Iterativa (IDS): Embora não seja a mais rápida, usa menos memória que o BFS, tornando-se uma alternativa viável quando a memória é uma preocupação maior.

Facilidade de Implementação:

- Busca em Largura (BFS) e A com Heurística de Peças Erradas:* Ambos são relativamente simples de implementar, mas apresentam maior uso de recursos comparados ao A* com heurística de distância de Manhattan.

logo, para resolver o N-puzzle de forma eficiente e rápida, percebemos que a Busca A* com heurística de distância de Manhattan é a melhor escolha, destacando-se tanto em termos de desempenho quanto na otimização do uso de recursos computacionais.