ISMAI

Licenciatura em Informática

UC de Sistemas inteligentes

2023/2024

Elaborado por: Bernardo Fraga Nº A043363
João Lino Nº A042083

Índice

Índice de ilustrações

Introdução

Descrição do puzzle

O puzzle do 8, também conhecido como Quebra-cabeças do 8 ou puzzle 3x3, é um jogo popular que envolve mover peças numeradas numa grelha 3x3. O objetivo é reorganizar as peças numa configuração específica, geralmente começando com uma configuração inicial desordenada e terminando com uma configuração ordenada. A configuração ordenada é normalmente representada da seguinte forma:

1	2	3
4	5	6
7	8	

Figura 1- Puzzle concluído

Onde os números de 1 a 8 representam as peças numeradas e 0 (zero) representa o espaço vazio. As peças podem ser movidas horizontal ou verticalmente para o espaço vazio, com o objetivo de alcançar a configuração final.

O desafio do puzzle do 8 reside na capacidade de encontrar a sequência de movimentos correta que leva da configuração inicial à configuração final. Isso pode envolver a aplicação de diferentes algoritmos de pesquisa para encontrar a solução mais eficiente.

O Puzzle do 8 é um problema bem conhecido em inteligência artificial e tem aplicações em várias áreas, incluindo jogos, robótica e pesquisa operacional. A implementação de algoritmos eficientes para resolver este quebra-cabeça é um desafio interessante que envolve conceitos-chave em computação, como pesquisa em grafos, heurísticas e complexidade algorítmica.

Descrição dos algoritmos implementados

Neste trabalho prático, implementamos três algoritmos de busca para resolver o puzzle do 8: Breadth-First Search (BFS), Depth-First Search (DFS) e Greedy Best-First Search. Cada um desses algoritmos possui características distintas que afetam a sua eficiência e comportamento durante a resolução do puzzle.

Breadth-First Search (BFS):

O BFS é um algoritmo de busca que explora todos os nós de um determinado nível antes de avançar para os nós do próximo nível na árvore de busca. Ele começa na

raiz (configuração inicial) e explora todos os nós vizinhos (possíveis movimentos) antes de passar para os nós do próximo nível. Este algoritmo garante que a solução encontrada seja a mais curta em termos de número de movimentos, pois explora todas as possibilidades de movimento a partir da configuração inicial.

Depth-First Search (DFS):

O DFS é um algoritmo de busca que explora o máximo possível em uma ramificação antes de retroceder e tentar outra ramificação. Isso significa que ele explora uma ramificação até o nó mais profundo possível antes de retroceder e explorar outros ramos. No contexto do puzzle do 8, isso pode levar a soluções que não são necessariamente as mais curtas, pois o algoritmo pode ficar preso em uma ramificação particular antes de explorar todas as possibilidades.

Greedy Best-First Search:

O Greedy Best-First Search é uma variação do DFS que utiliza uma heurística para determinar qual nó expandir primeiro. Em vez de expandir o nó mais próximo da raiz, como no BFS, o Greedy Best-First Search expande o nó que parece ser mais promissor de acordo com a heurística escolhida. Duas heurísticas comuns utilizadas no Greedy Best-First Search para o puzzle do 8 são:

Manhattan City Block (Distância de Manhattan): Esta heurística calcula a distância total que cada peça numerada está do seu objetivo final, somando as distâncias horizontais e verticais entre as posições atual e final de cada peça. Quanto menor a distância de Manhattan, mais promissor o nó é considerado.

Hamming Distance (Distância de Hamming): Esta heurística conta o número de peças fora do lugar na configuração atual em comparação com a configuração final. Quanto menor o número de peças fora do lugar, mais promissor o nó é considerado.

Essas heurísticas são utilizadas para guiar a busca na direção de configurações que parecem estar mais próximas da solução final, sem necessariamente garantir a otimização da solução encontrada. O Greedy Best-First Search pode encontrar soluções rapidamente, mas não garante a solução mais curta. A escolha entre as heurísticas pode afetar significativamente o desempenho do algoritmo em diferentes configurações do puzzle.

Discussão das principais características de cada um dos algoritmos

Breadth-First Search (BFS):

- Optimalidade: O BFS é ótimo para encontrar a solução mais curta em termos de número de movimentos, pois explora todas as possibilidades a partir da configuração inicial antes de passar para os próximos níveis na árvore de busca.
- Completude: O BFS é completo, o que significa que sempre encontrará uma solução se ela existir. Isso ocorre porque ele explora sistematicamente todas as possibilidades até encontrar a solução.
- Complexidade: A complexidade de tempo do BFS é O(b^d), onde b é o fator de ramificação (número máximo de filhos de um nó) e d é a profundidade da solução mais rasa. Isso significa que o BFS pode ser computacionalmente caro quando o fator de ramificação é alto ou quando a solução está muito distante da raiz.

Depth-First Search (DFS):

- Optimalidade: O DFS não é ótimo, pois pode encontrar uma solução que não é necessariamente a mais curta. Ele pode ficar preso explorando uma ramificação particular e não considerar outras possibilidades.
- Completude: O DFS não é completo em todos os casos, pois pode entrar em um loop infinito se houver ciclos no grafo de busca ou se a árvore de busca for infinita.
- Complexidade: A complexidade de tempo do DFS depende da profundidade máxima da árvore de busca. Se a profundidade máxima for limitada, o DFS pode ser mais eficiente em termos de tempo do que o BFS. No entanto, se a profundidade máxima não for limitada, o DFS pode não terminar.

Greedy Best-First Search:

- Optimalidade: O Greedy Best-First Search não é ótimo, pois não garante a solução mais curta. Ele pode ser influenciado pela heurística escolhida e encontrar uma solução subótima.
- Completude: A completude do Greedy Best-First Search depende da heurística utilizada. Em alguns casos, pode não encontrar uma solução se a heurística levar a escolhas que o impeçam de alcançar a solução.
- Complexidade: A complexidade de tempo do Greedy Best-First Search depende da eficácia da heurística escolhida. Em geral, é mais eficiente do que o BFS em termos de tempo, mas pode não ser tão rápido quanto o

DFS. A complexidade também depende da estrutura específica do problema e das características das heurísticas utilizadas.

Em resumo, cada algoritmo de busca tem suas próprias características em termos de ótimo, completude e complexidade. A escolha do algoritmo mais adequado depende das características específicas do problema e dos recursos disponíveis, como tempo e memória.

Combinações testadas e respetivas respostas

Combinação:

6	4	7
8	5	
3	2	1

Figura 2 - Puzzle de exemplo

Breadth-First Search:

Com esta matriz, o BFS conseguiu resolver o problema em 31 passos, eles sendo:

R, D, R, U, L, D, R, U, U, L, D, R, D, R, U, U, L, D, R, U, L, L, D, R, U, U, L, D, R, D, R, U, L, L, D, R, U, U, L, D, R, D, R, U, L, L

Depth-First Search:

Já o DFS apenas finalizou o problema ao fim de 24250 passos.

Greedy Best-First Search:

Finalmente o GBFS acaba o algoritmo com 47 passos, os quais:

R, D, R, U, L, D, R, U, U, L, D, R, D, R, U, U, L, D, R, U, L, L, D, R, U, U, L, D, R, D, R, U, L, L, D, R, U, U, L, D, R, D, R, U, L, L, U, R, D, L, U, R, D, L, U, R, D, L, U, R, D, L

É de notar que mesmo com um maior número de passos que o BFS, o Greedy tem um tempo de execução exponencialmente mais pequeno que o BFS por causa das suas heurísticas, mais detalhe no próximo capítulo.

Custo de tempo e memória utilizando cada um dos algoritmos

Tendo por base a combinação apresentada em cima foi feito um ensaio aos fatores de tempo e memoria utilizados por cada um dos algoritmos.

Breadth-First Search:

```
Total steps: 31
Tempo de execução: 11.29501223564148 segundos
Memory Usage: 88.859375 MB
```

Figura 3 - Dados sobre BFS

Greedy Best-First Search:

```
Total steps: 47
Tempo de execução: 0.06637787818908691 segundos
Memory Usage: 21.85546875 MB
```

Figura 4 - Dados sobre Greedy Best-First Search

Depth-First Search:

```
Total steps: 24249
Tempo de execução: 49.57043528556824 segundos
Memory Usage: 69.265625 MB
```

Figura 5 - Dados sobre DFS

Discussão de resultados:

O algoritmo BFS explora todos os nós num nível antes de passar para o próximo nível. Isso garante que a solução encontrada seja a mais curta em termos de passos, resultando em um total de passos relativamente baixo. No entanto, como o BFS explora todas as opções em cada nível antes de avançar para o próximo, ele pode exigir uma quantidade significativa de memória, como evidenciado pelo alto uso de memória neste caso. Além disso, o tempo de execução é relativamente alto devido à natureza abrangente da busca em largura.

O algoritmo Greedy prioriza a escolha da próxima etapa com base em uma heurística local, ou seja, ele escolhe a próxima etapa que parece mais promissora no momento atual, sem considerar futuras consequências. Isso pode levar a soluções que não são necessariamente as mais curtas em termos de passos, mas são encontradas muito rapidamente. Como resultado, o algoritmo Greedy pode usar mais passos, mas em menos tempo, como observado nos resultados. Além disso, como o Greedy não precisa manter informações sobre todos os nós visitados, seu uso de memória é significativamente menor em comparação com o BFS.

O algoritmo DFS explora tão profundamente quanto possível ao longo de cada ramificação antes de fazer um retrocesso. Isso pode levar a soluções que não são necessariamente as mais curtas, como evidenciado pelo grande número de passos neste caso. No entanto, como o DFS não mantém informações sobre todos os nós visitados, seu uso de memória é mais eficiente do que o BFS, embora ainda seja relativamente alto. O tempo de execução também é mais longo devido à natureza exploratória do algoritmo, onde ele pode precisar retroceder profundamente em várias ramificações antes de encontrar a solução.

Conclusões principais do trabalho

Em grosso modo podemos dividir este trabalho em três tópicos de discussão:

Desempenho e Eficiência:

O Breadth-First Search é eficaz para encontrar a solução mais curta em termos de passos, mas pode ser lento e exigir uma quantidade significativa de memória, especialmente para problemas complexos.

Já o greedy prioriza a velocidade em detrimento da solução mais curta, resultando em um menor tempo de execução, mas pode levar a soluções mais longas em termos de passos.

A Busca em Profundidade (DFS) é rápida e eficiente em termos de memória, mas pode levar a soluções que não são necessariamente as mais curtas, e seu tempo de execução pode ser alto, especialmente para problemas com muitas soluções possíveis.

Trade-offs entre Tempo, Memória e Solução Ótima:

A escolha do algoritmo de busca depende das necessidades específicas do problema em questão. Se o objetivo é encontrar a solução mais curta em termos de passos, independentemente do tempo de execução ou uso de memória, o BFS é a escolha ideal. No entanto, se o tempo de execução for uma preocupação primária e uma solução ótima não for necessária, a Busca Gulosa pode ser preferível.

Por outro lado, se o espaço de busca for grande e a memória for uma preocupação significativa, o DFS pode ser uma opção viável devido ao seu uso eficiente de memória, embora possa levar a soluções subótimas em termos de passos.

Complexidade do Problema:

A complexidade do problema, incluindo o número de estados possíveis e a estrutura do espaço de busca, influencia diretamente o desempenho e a eficiência dos algoritmos de busca.

Problemas mais simples podem ser resolvidos eficientemente por qualquer algoritmo de busca, enquanto problemas mais complexos podem exigir um balanceamento cuidadoso entre tempo, memória e qualidade da solução.

Em resumo, a escolha do algoritmo de busca depende de uma análise cuidadosa das necessidades do problema específico, considerando fatores como tempo de execução, uso de memória e a qualidade da solução desejada. Cada algoritmo possui suas próprias vantagens e limitações, e a escolha adequada depende do contexto e das restrições do problema em questão.