Pesquisa Primeiro em Largura (Breadth-first search)

- Estratégia: todos os nós de menor profundidade são expandidos primeiro
- Bom: pesquisa muito sistemática
- Mau: Normalmente demora muito tempo e sobretudo ocupa muito espaço
- Propriedades:
    - Completa: sim, se b(fator de ramificação) for finito
    - Tempo: supondo fator de ramificação b então $n=1+b+b2+b3+ \dots + b^n = O(b^d)$ é exponencial em d
    - Espaço: guarda cada nó em memória $O(b^d)$
    - Otimal: sim, se o custo de cada passo for 1
- Em geral só pequenos problemas podem ser resolvidos assim!

Pesquisa Primeiro em Profundidade (Depth-first search)

- Estratégia: expandir sempre um do nós mais profundos da árvore
- Bom: muito pouca memória necessária, bom para problemas com muitas soluções
- Mau: não pode ser usada para árvores com profundidade infinita, pode ficar presa em ramos errados
- Propriedades:
    - Completa: não, falha em espaços de profundidade infinita, com repetições(loops)
        - Modifique para evitar estados repetidos ao longo do caminho
    - Tempo: $O(b^m)$, mau se m > d
    - Espaço: O(bm), espaço linear
    - Otimal: não (em princípio devolve a 1º solução que encontra)
- Por vezes é definida uma profundidade limite (I) e transforma-se em Pesquisa com Profundidade Limitada.

Como dito em cima, um dos problemas da pesquisa Primeiro em Profundidade prende-se com a incapacidade desta lidar com caminhos infinitos. A Pesquisa em Profundidade com limite de profundidade procura evitar este problema fixando o nível máximo de procura.

Pesquisa em profundidade com limite de profundidade I, ie, nós em profundidade I não têm sucessores.

Pesquisa Primeiro em Profundidade (Depth-First Search)

- Escolha o primeiro elemento da lista de estados não expandidos
- Adicione extensões de caminho à frente da lista de estados não expandidos

Pesquisa Primeiro em Largura (Breadth-first search)

- Escolha o primeiro elemento de lista de estados não expandidos
- Adicione extensões de caminho ao final da lista de estados não expandidos

Pesquisa de Custo Uniforme

- Estratégia:
    - Para cada nó da lista de estados não expandidos, salve o custo total do caminho do estado inicial para esse nó
    - Expandir sempre o nó com menor custo da lista de estados não expandidos (medido pela função de custo da solução)
- Pesquisa Primeiro em Largura é igual a Pesquisa de Custo Uniforme se g(N)=Depth(N)
- Equivalente a Pesquisa Primeiro em Largura, se os custos todos iguais
- Implementação: da lista de estados não expandidos é uma fila prioritária ordenada pelo custo do caminho
- Temos de garantir que g(sucessor)>=g(N)
- Propriedades:
    - Completa: Sim, se o custo da etapa for maior que alguma constante positiva ε (não queremos sequências infinitas de etapas com um custo total finito)
    - Tempo:
        - Número de nós com custo de caminho =< custo da solução ideal (C*), $O(b^{C^*/\varepsilon})$
        - Isso pode ser maior que O(bd): a pesquisa pode explorar caminhos longos que consistem em pequenos passos entes de explorar caminhos mais curtos que consistem em passos maiores
    - Espaço: $O(b^{C^*/\varepsilon})$
    - Otimal: sim
- Em geral só pequenos problemas podem ser resolvidos assim!

Pesquisa Iterativa Aprofundamento Progressivo

Se não conhecermos o valor limite máximo, estaremos condenados a uma estratégia de procura em profundidade primeiro e temos que lidar com o problema de eventuais caminhos infinitos. A resposta passa pela alteração do principio da procura limitada fazendo variar esse limite entre zero e infinito.

Use Pesquisa primeiro em Profundidade como uma sub-rotina

- Verifique a raiz
- Faça um DFS procurando um caminho de comprimento 1
- Se não houver um caminho de comprimento 1, faça um DFS procurando um caminho de comprimento 2
- Se não houver um caminho de comprimento 2, faça um DFS procurando um caminho de comprimento 3…

- Estratégia: Executar pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade
- Propriedades:
    - Completa: Yes
    - Tempo: $(d+1)b^0 + d\,b^1 + (d-1)\,b^2 + b^d = O(b^d)$

    - Espaço: O(b^d)
    - Otimal: sim, se o custo for 1
- Em geral é a melhor estratégia (não informada ou cega) para problemas com um grande espaço de pesquisa e em que a profundidade da solução não é conhecida

Pesquisa Bidirecional

- Estratégia: Executar pesquisa para a frente desde o estado inicial e para trás desde o objetivo, simultaneamente
    - Bom: pode reduzir enormemente a complexidade no tempo
- Eg., Para encontrar uma rota na Romênia, existe apenas um estado objetivo, portanto, a pesquisa para trás é muito parecida com a pesquisa para frente; Mas se o objetivo é uma descrição abstrata, como o de que "nenhuma rainha ataca outra rainha" no problema das rainhas n, a pesquisa bidirecional é difícil de usar.

Comparação entre Estratégias de Pesquisa

- Avaliação das estratégias de pesquisa:
    - B é o fator de ramificação
    - d é a profundidade da solução
    - m é a máxima profundidade da árvore
    - l é a profundidade limite de pesquisa

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Pesquisa informada (Heurística)

- Utiliza informação do problema para evitar que o algoritmo de pesquisa fique “perdido vagueando no escuro”

Estratégia de Pesquisa:

- Definida escolhendo a ordem de expansão dos nós

Pesquisa do Melhor Primeiro (Best-First Search)

- Utiliza uma função de avaliação que retorna um número indicando o interesse de expandir um nó
- Pesquisa Gulosa (Greedy-Search) – f(n) = h(n) (função que estima distância à solução)
- Algoritmo A* - f(n) = g(n) + h(n) (estima o custo da melhor solução que passa por n)

Heurísticas

- Como técnica de procura para a obtenção de metas em problemas não algorítmicos que geram “explosões” combinatórias;
- Como um método aproximado de resolução de problemas utilizando funções de avaliação de tipo heurística (dica);
- Como um método de poda (corte) para estratégias de programas de jogos.
- Numa procura, podemos aplicar dois tipos genéricos de heurísticas, sobre a decisão sobre qual nó será feita a expansão e sobre a decisão sobre quais os nós que devem ser descartados.
    - Se o universo é totalmente conhecido, a heurística será realizada através da atribuição de números;
    - Se o universo não é totalmente conhecido, no qual a heurística será realizada através da aplicação de regras.
- As heurísticas são específicas para cada problema. Estas funções podem ser pouco certas ou podem não encontrar a melhor resposta, no entanto a sua utilização permite a libertação do uso das análises combinatórias.
- A implementação de uma heurística é difícil! É difícil medir precisamente o valor de uma determinada solução e é difícil medir determinados conhecimentos de forma a permitir que seja efetuada uma análise matemática do seu efeito no processo de busca

Pesquisa informada (Pesquisa gulosa – Greedy-Search)

- Estratégia:
    - Expandir o nó que parece estar mais perto da solução
- h(n) = custo do caminho mais curto do estado n para o objetivo (função heurística)
- Propriedades:
    - Completa? Não, pode entrar em ciclos
        - Suscetível a falsos começos
    - Complexidade no tempo?
        - Mas com uma boa função heurística pode diminuir consideravelmente
    - Complexidade no espaço?
        - Mantém todos os nós na memória
    - Ótima? Não, não encontra sempre a solução ótima
    - Necessário detetar estados repetidos.

Pesquisa informada (Pesquisa A*)

- Estratégia:
    - Evite expandir caminhos que são caros
    - O algoritmo A* combina a pesquisa gulosa com a uniforma, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. Usa a função:

f(n) = g(n) + h(n)
    - g(n) = custo total, até agora, para chegar ao estado n (custo do percurso)
    - h(n) = custo estimado para chegar ao objetivo (não deve sobrestimar o custo para chegar à solução (heurística))
    - f(n)= custo estimado da solução mais barata que passa pelo nó n
- Algoritmo A* é ótimo e completo.
- Complexidade no tempo exponencial em (erro relativo de h*comprimento da solução)
- Complexidade no espaço: mantém todos os nós em memória

Pesquisa com Memória Limitada – IDA*/SMA*

- IDA* - Pesquisa com Profundidade Iterativa
    - Estratégia: utilização de um custo limite em cada iteração e realização de pesquisa em profundidade iterativa
    - Problemas em alguns problemas reais com funções de custo com muitos valores
- SMA* - Pesquisa Simplificada com Memória Limitada
    - IDA* de uma iteração para a seguinte só se lembra de um valor (o custo limite)
    - SMA* utiliza toda a memória disponível, evitando estados repetidos
    - Estratégia: quando necessita de gerar um sucessor e não tem memória, esquece um nó da fila que pareça pouco prometedor (com custo alto)

Resumo

Algoritmo	Completo	Otimal	Tempo complexidade	Espaço complexidade
Primeiro em Largura (BFS)	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(b^d)$
Primeiro em Profundidade (DFS)	No	Não	$O(b^m)$	O(bm)
Pesquisa Iterativa	Yes	Se todos os custos escalonados forem iguais	$O(b^d)$	O(bd)
Custo Uniforme	Yes	Sim	Number of nodes with $g(n) \leq C^*$	
Greedy	No	Não	Worst case: $O(b^m)$ Best case: O(bd)	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	