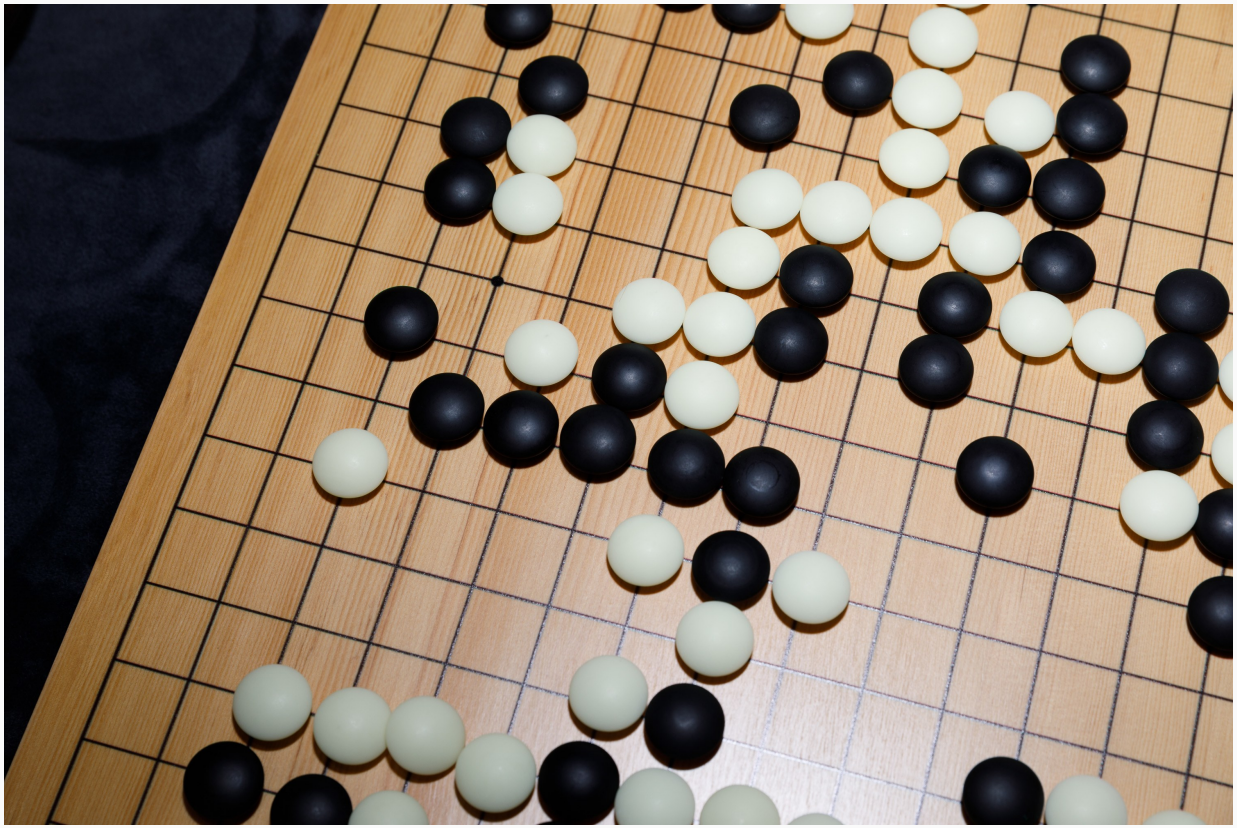
ACH2016 - Inteligência Artificial
Aula 10 - Busca

Valdinei Freire da Silva

valdinei.freire@usp.br - Bloco A1 100-O

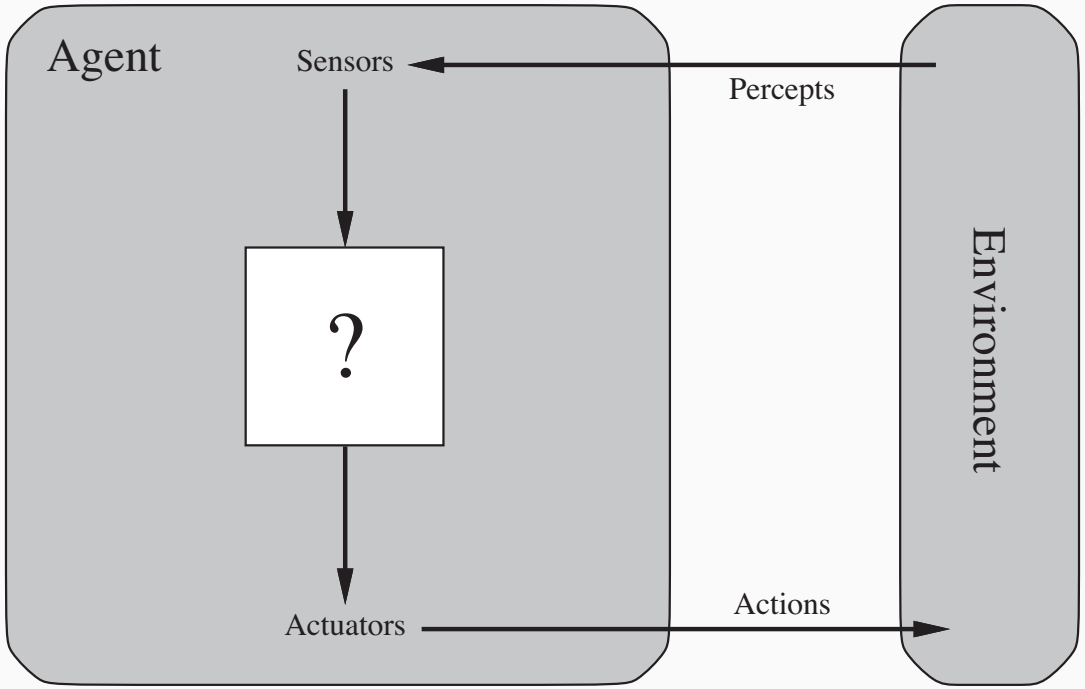
Russell e Norvig, Capítulo 3

AlphaGo



- Solução: busca aleatória heurística + aprendizado por reforço + redes neurais
- Documentário: AlphaGo

Agentes Inteligentes



- Agente: percebe o ambiente por meio de sensores e age no ambiente por meio de atuadores para atingir o melhor resultado.
- PEAS: Performance, Environment, Agent, Sensors

Resolução de Problemas

Ambientes: completamente observável, único agente, conhecido, determinístico, discreto, sequencial, estático.

Formulação de Problemas

- estado inicial: é o estado no tempo $t = 0$.
- ações possíveis: a função $\mathrm{ACTIONS}(s)$ retorna o conjunto de ações que podem ser executadas no estado s.
- modelo de transição: a função $\mathrm{RESULT}(s, a)$ retorna o estado resultante de aplicar a ação a no estado s.
- teste de meta: a função $\mathrm{GOAL}(s)$ determina se o estado s é um estado meta.
- custo de caminho: a função $\mathrm{PATHCOST}(h)$ retorna o custo associado a uma sequência $h = s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T$.

Resolução de Problemas

O modelo de transição resulta em grafo orientado.

O custo de uma sequência pode ser modelado por uma função de custo local $c : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, onde

$$\text{PATHCOST}(h) = \sum_{t=0}^{T-1} c(s_t, a_t, s_{t+1}).$$

A função $\text{GOAL}(s)$ pode ser implementada simplesmente por um conjunto de estados metas $\mathcal{S}_G$.

Uma vez que o ambiente é determinista, a solução para o problema é uma sequência de ações $a_0, \ldots, a_{T-2}, a_{T-1}$ que gera um caminho $s_0, \ldots, s_{T-1}, s_T$ no grafo de transição, onde s_0 é o estado inicial e $\text{GOAL}(s_T) = \texttt{TRUE}$.

Toy Problems - 8-puzzle

7	2	4
5		6
8	3	1

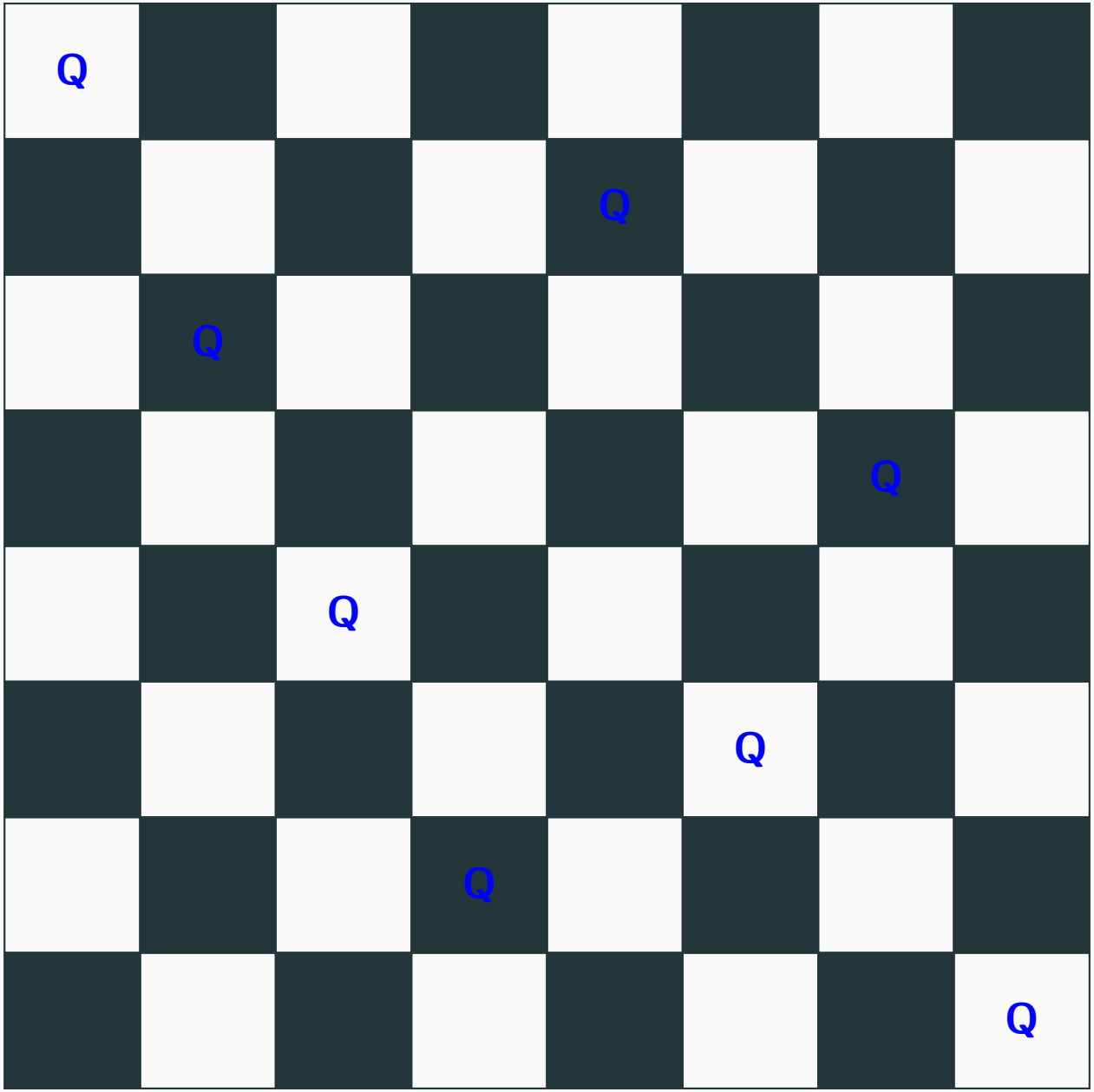
estado inicial

	1	2
3	4	5
6	7	8

estado meta

- estados: descrevem a posição de cada peça, onde as peças não podem ocupar o mesmo espaço.
- estado inicial: metade dos estados podem ser designado como estado inicial.
- ações: movimentação do espaço vazio; *Left*, *Right*, *Up*, ou *Down*.
- modelo de transição: troca de lugar o vazio com uma peça de acordo com ação escolhida.
- teste de meta: verifica se chegou no único estado meta.
- custo de caminho: custo local igual à 1.

Toy Problems - 8-rainhas



Toy Problems - 8-puzzle

- estados: qualquer arranjo de 0 à 8 rainhas no tabuleiro.
- estado inicial: tabuleiro sem nenhuma rainha.
- ações: adição de uma rainha a um espaço vazio.
- modelo de transição: retorna o tabuleiro com a rainha colocada no local escolhido.
- teste de meta: 8 rainhas no tabuleiro e nenhuma delas ataca às outras.
- custo de caminho: desnecessário.

Toy Problems - Gerar qualquer inteiro positivo

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor$$

- estados: números positivos.
- estado inicial: 4.
- ações: aplicar fatorial (apenas a inteiros), raiz quadrada, ou piso.
- modelo de transição: segunda as definições matemáticas.
- teste de meta: estado é o inteiro positivo escolhido.
- custo de caminho: desnecessário.

Problema Real - Problema do Caixeiro-Viajante

A partir de uma cidade origem, visitar um conjunto de cidades exatamente uma vez e retornar para a cidade origem.

- estados: localização atual, mais o conjunto de cidades visitadas.
- estado inicial: cidade origem, e nenhuma cidade visitada.
- ações: viajar para uma cidade vizinha.
- modelo de transição: viaja para a cidade vizinha escolhida.
- teste de meta: todas as cidades foram visitadas e se encontra na cidade origem.
- custo de caminho: custo local é a distância percorrida em cada viajem.

Problema Real - Encontrar uma Rota

A partir de um local origem, encontrar uma rota até um local destino.

- estados: localização atual.
- estado inicial: local origem.
- ações: dirigir até um novo local.
- modelo de transição: o estado é o novo local.
- teste de meta: está no local destino.
- custo de caminho: distância.

Problemas Reais

- layout de circuitos e componentes eletrônicos para caber no menor espaço possível
- navegação robótica
- distribuição de carga horária
- sistemas de planejamento de viagens

Busca por Solução

função Busca(problema) **retorna** uma solução, ou falha
    inicializar a fronteira usando o estado inicial do problema
    inicializar o conjunto explorado como vazio
    repita
        se fronteira vazia
            então retorna falha
        escolher um nó folha e remover da fronteira
        se o estado do nó for um estado objetivo
            então retorna solução correspondente
        adicionar o nó ao conjunto explorado
        expandir o nó escolhido, encontrando os próximos nós
        para cada próximo nó
            se não estiver na fronteira nem no conjunto explorado
                então adiciona à fronteira

Busca por Solução

Os algoritmos se diferenciam pela escolha do nó a ser retirado da fronteira.

Fila FIFO: busca em largura

Fila LIFO: busca em profundidade

Fila de Prioridade: custo parcial + conhecimento *a priori*

Métodos de Busca

Busca sem informação ou cega ou exaustiva ou força bruta ou sistemática: não sabe qual o melhor nó da fronteira a ser expandido. Apenas distingue o estado objetivo dos não objetivos.

Busca informada ou heurística: estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas. Sabem se um estado não objetivo é "mais promissor".

Busca local: operam em um único estado e movem-se para a vizinhança deste estado.

Busca por Solução

Avaliação de Algoritmos:

- Completude: a estratégia garante encontrar uma solução, se existir uma?
- Otimalidade: a estratégia encontra a solução ótima?
- Complexidade de tempo: quanto tempo gasta para encontrar uma solução?
- Complexidade de memória: quanto de memória é preciso?

Busca por Solução

Grafo dirigido é sempre obtido implicitamente.

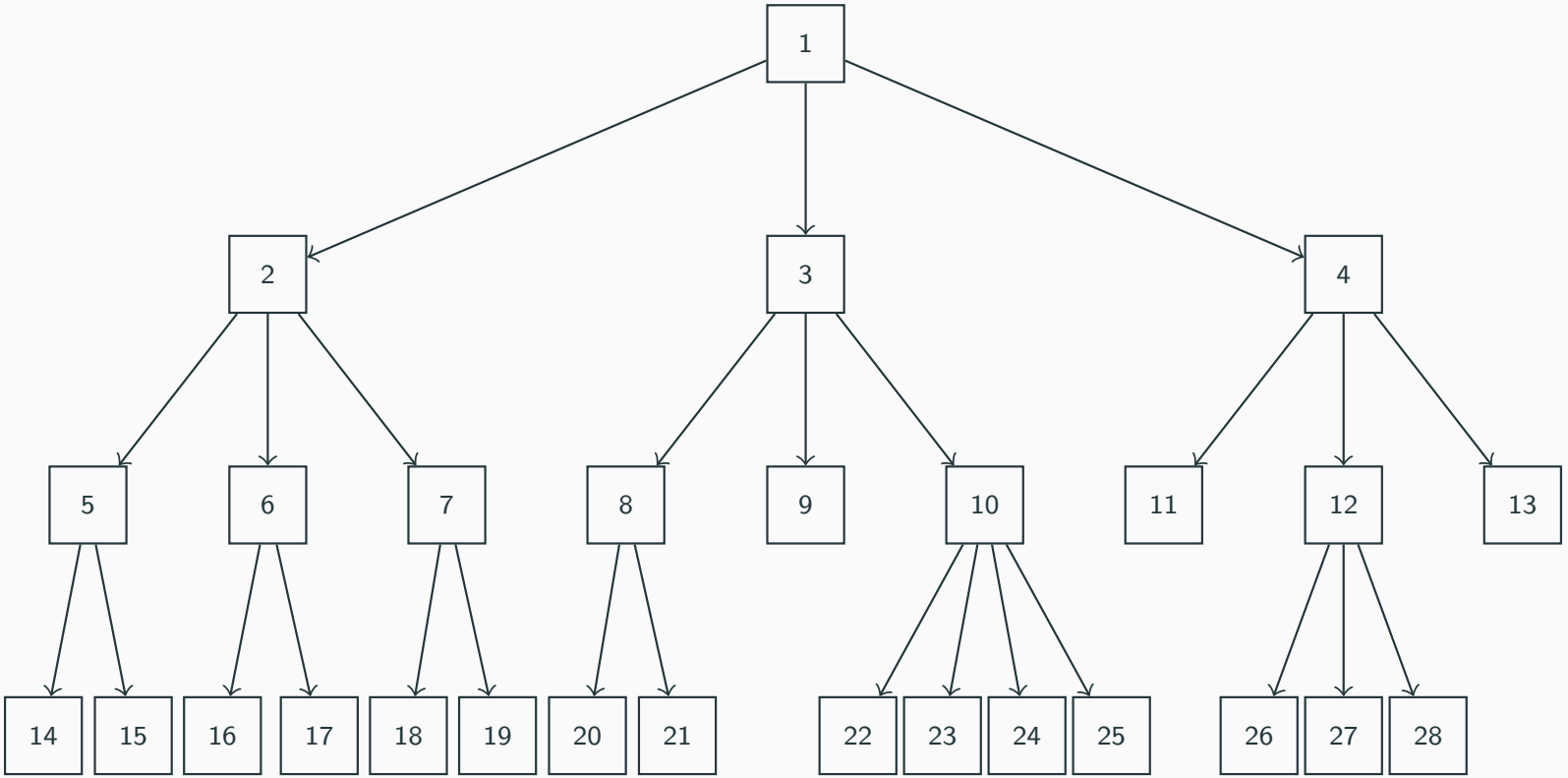
Complexidade em termos de:

- Fator de Ramificação b: número máximo de sucessores
- Profundidade d: profundidade do nó objetivo menos profundo
- comprimento máximo m: comprimento máximo de qualquer caminho no espaço de estados

Busca Cega

- Busca em Largura
- Busca em Profundidade
- Busca Iterativa
- Busca de Custo Uniforme
- Busca Bidirecional

Busca em Largura



Fila First-In First-Out: todos os nós do nível mais baixo são expandidos primeiro.

Busca em Largura

Completude: como o algoritmo percorre todos os nós de um nível antes de passar para o próximo, ele percorrerá todos os estados alcançáveis.

Otimalidade: garantida apenas se o custo é dado em função da profundidade e é não-decrescente.

Complexidade de tempo: $O(b^d)$ ou $O(b^{d+1})$, dependendo da implementação.

Complexidade de memória: $O(b^d)$.

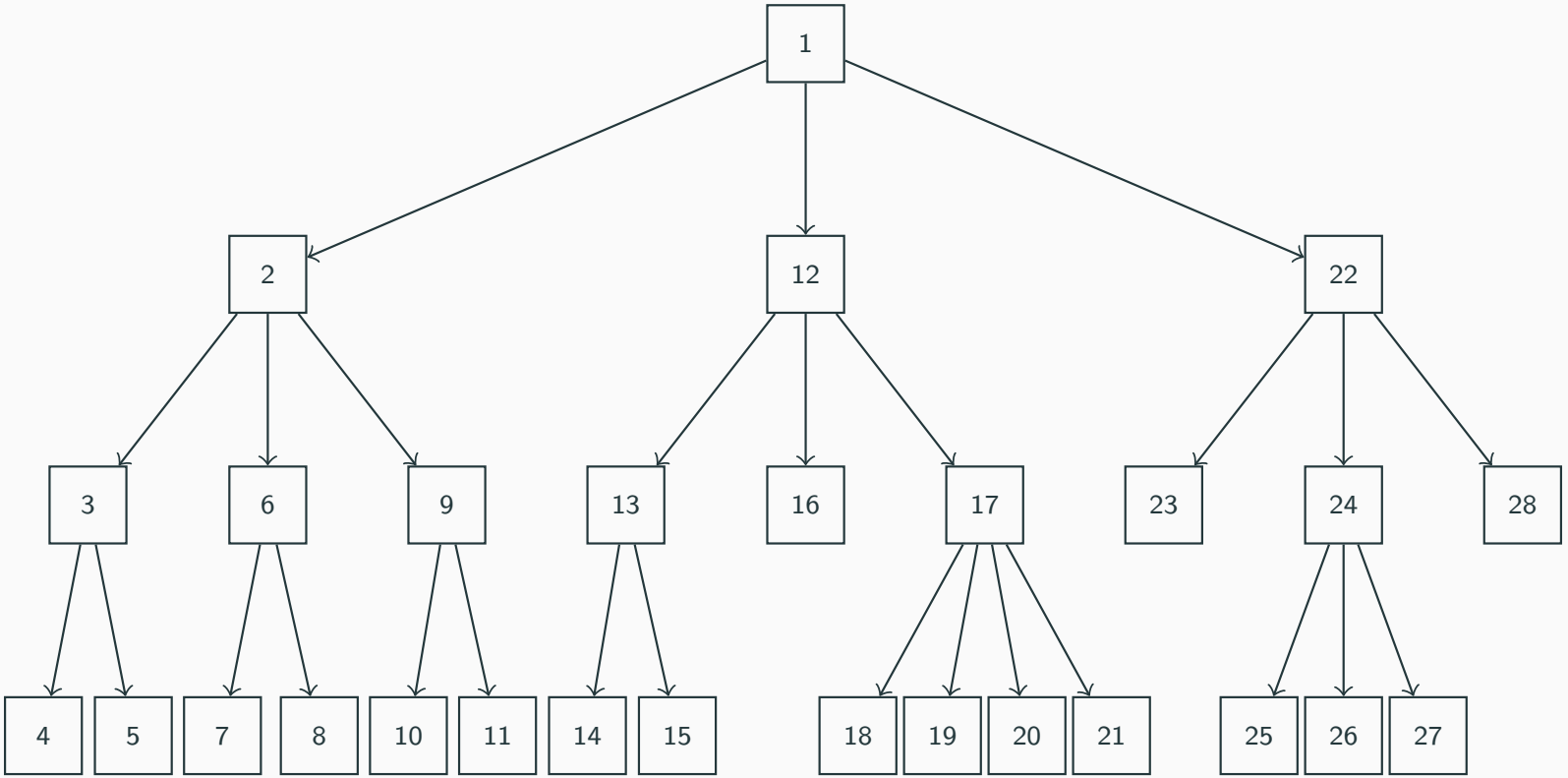
Busca em Largura

Considerando $b = 10$, geração de um milhão de nós por segundo e cada nó ocupando 1.000 byte de memória.

Profundidade (d)	Nós	Tempo	Memória
2	110	0.11 ms	107 KB
4	11.110	11 ms	10.6 MB
6	10^6	1.1 s	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 horas	10 TB
12	10^{12}	13 dias	1 PB

Uso de memória impraticável para pequenas profundidades.

Busca em Profundidade



Fila Last-In First-Out: os nós do nível mais profundo são expandidos primeiro.

Busca em Profundidade

Completude: se a quantidade de estados é finita e nenhum nó repetido é expandido, então a busca percorrerá todos os estados alcançáveis.

Otimalidade: não é garantida.

Complexidade de tempo: no pior dos casos pode resultar em $O(b^m)$.

Complexidade de memória: em uma implementação que permite nós repetidos, $O(bm)$.

Busca em Profundidade

Busca em Profundidade Limitada: considera um limite de profundidade ℓ.

Completude: se $d \leq \ell$.

Otimalidade: não é garantida.

Complexidade de tempo: no pior dos casos pode resultar em $O(b^{\ell})$.

Complexidade de memória: em uma implementação que permite nós repetidos, $O(b\ell)$.

Busca em Profundidade

Busca em Profundidade Iterativa: executa instâncias da busca em profundidade limita com ℓ crescente, isto é, $\ell = 1, 2, 3, \ldots$.

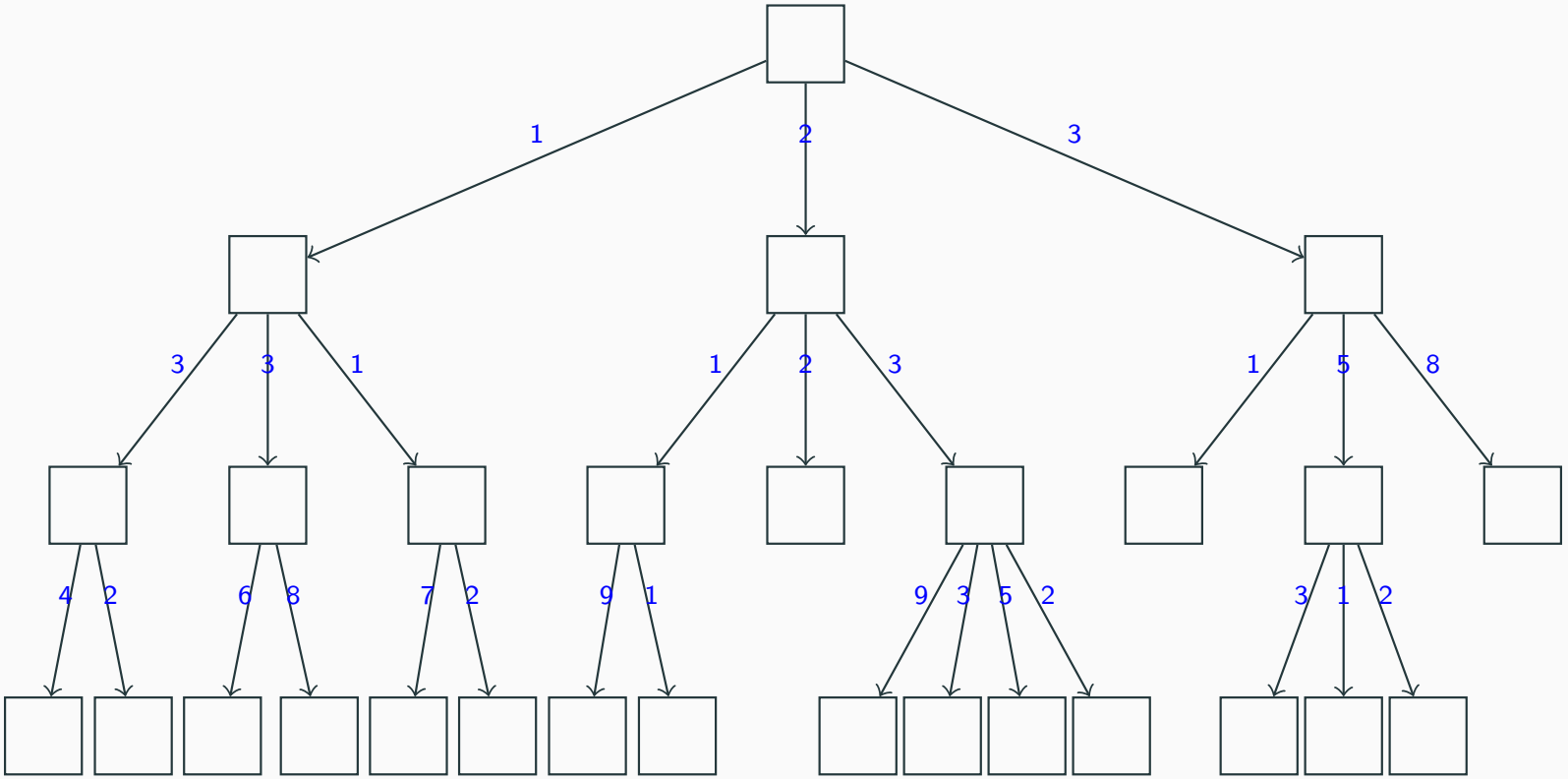
Completude: garantida.

Otimalidade: garantida apenas se o custo é dado em função da profundidade e é não-decrescente.

Complexidade de tempo: $O(b^d)$.

Complexidade de memória: em uma implementação que permite nós repetidos, $O(bd)$.

Busca de Custo Uniforme



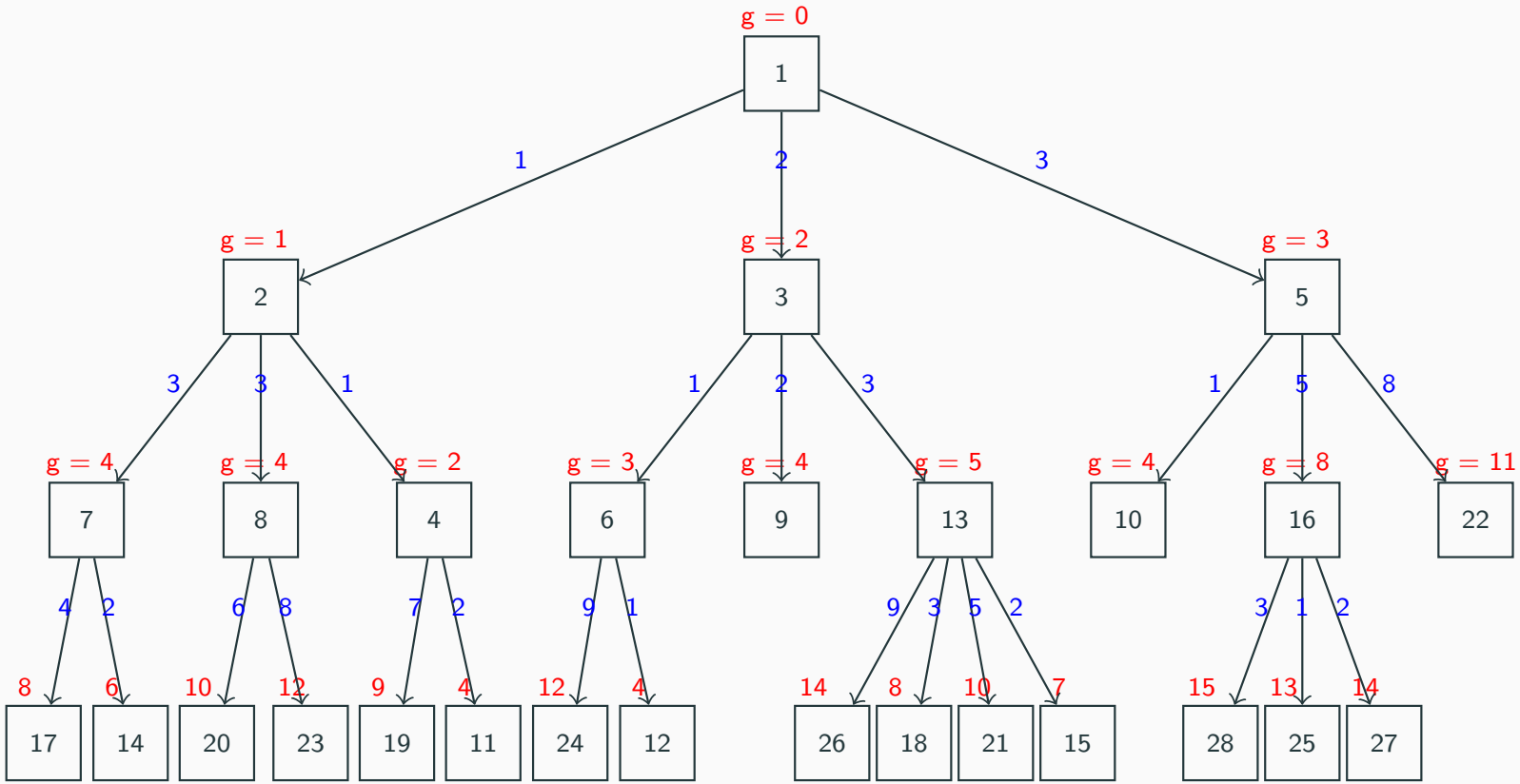
Busca de Custo Uniforme

Fila de Prioridade: os nós com custo parcial menor é expandido primeiro.

O nó é testado para a meta apenas antes de expandi-lo.

Quando um novo nó é gerado, se ele já existir na fonteira e o novo caminho possui menor custo, então o nó antigo é substituído pelo novo.

Busca de Custo Uniforme



Busca de Custo Uniforme

Completude: garantida se o custo de um caminho sempre aumenta ao longo do caminho.

Otimalidade: garantida se o custo de um caminho não diminui ao longo do caminho.

Complexidade de tempo: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$.

Complexidade de memória: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$.

C^* é o custo do caminho ótimo e ϵ é o aumento mínimo no custo do caminho quando um novo estado é adicionado.

Busca Bidirecional

Realiza duas busca em conjunto: uma a partir do estado inicial e outra a partir da meta, encontra um caminho quando as duas fronteiras se encontram.

Dificuldade: como gerar elementos predecessor? como obter um estado meta?

Completude: garantida se ambas buscas são realizadas em largura.

Otimalidade: garantida se as buscas são realizadas em largura e o custo é uma função não-decrescente da profundidade.

Complexidade de tempo: $O(b^{d/2})$.

Complexidade de memória: $O(b^{d/2})$.

Comparação entre Buscas Não-Informadas

Critério	Largura	Uniforme	Profundidade	Prof. Limitada	Iterativa	Bidirecional
Completude	sim[a]	sim[a,b]	não	não	sim[a]	sim[a,d]
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Memória	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Otimalidade	sim[c]	sim	não	não	sim[c]	sim[c,d]

[a] completo se b é finito

[b] completo se o custo local é $\geq \epsilon$ para ϵ positivo

[c] ótimo se o custo é dado por uma função não-decrescente da profundidade

[d] se ambas direções usam busca em largura