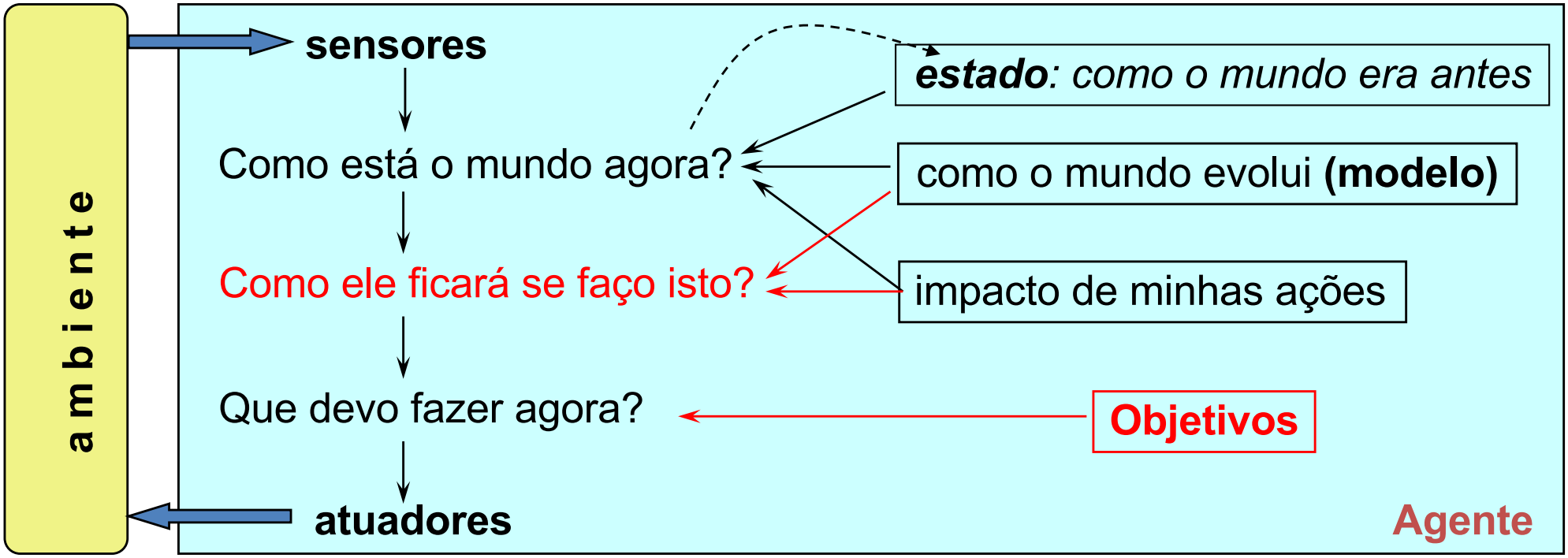
Resolvendo Problemas através de Busca

Inteligência Artificial
PCS3438

Escola Politécnica da USP
Engenharia de Computação (PCS)

Agente solucionador de problemas (guiado por objetivo – deliberativo)

- Busca uma *sequência de ações* que o leve a estados desejáveis (*objetivos*).



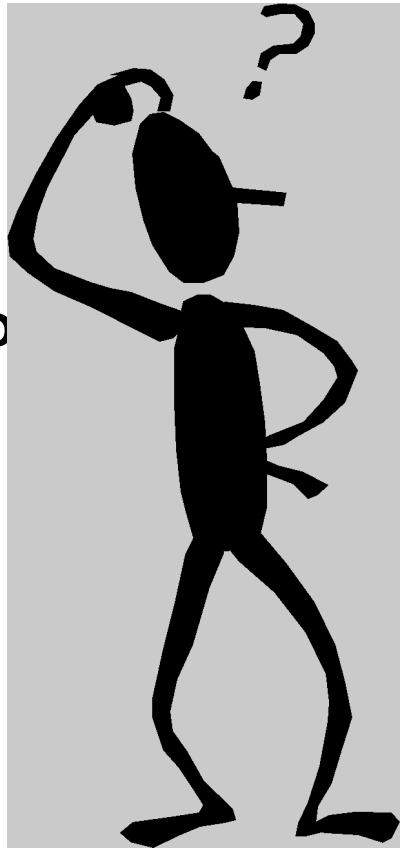
Agente solucionador de problemas

- Propriedades do ambiente para este agente:
  - **Estático**: não muda enquanto o agente delibera (“pensa”)
  - **Discreto**: enumera sequências de estados e alternativas de ações
  - **Determinístico**: solução é uma sequência definida de ações
    - não lida com eventos inesperados ou incertezas
    - executa a sequência definida sem considerar percepções ➔ sistema de controle em malha aberta
  - **Observável:** observa completamente os estados e sabe seu estado inicial

- *Algumas flexibilizações serão feitas em relação às propriedades de determinismo e observabilidade.*

Agentes solucionadores de problemas

- O que é um problema em I.A.?
- Como formulá-lo?
- Como buscar a solução do problema?
- Como avaliar a solução e o processo de encontrá-la?

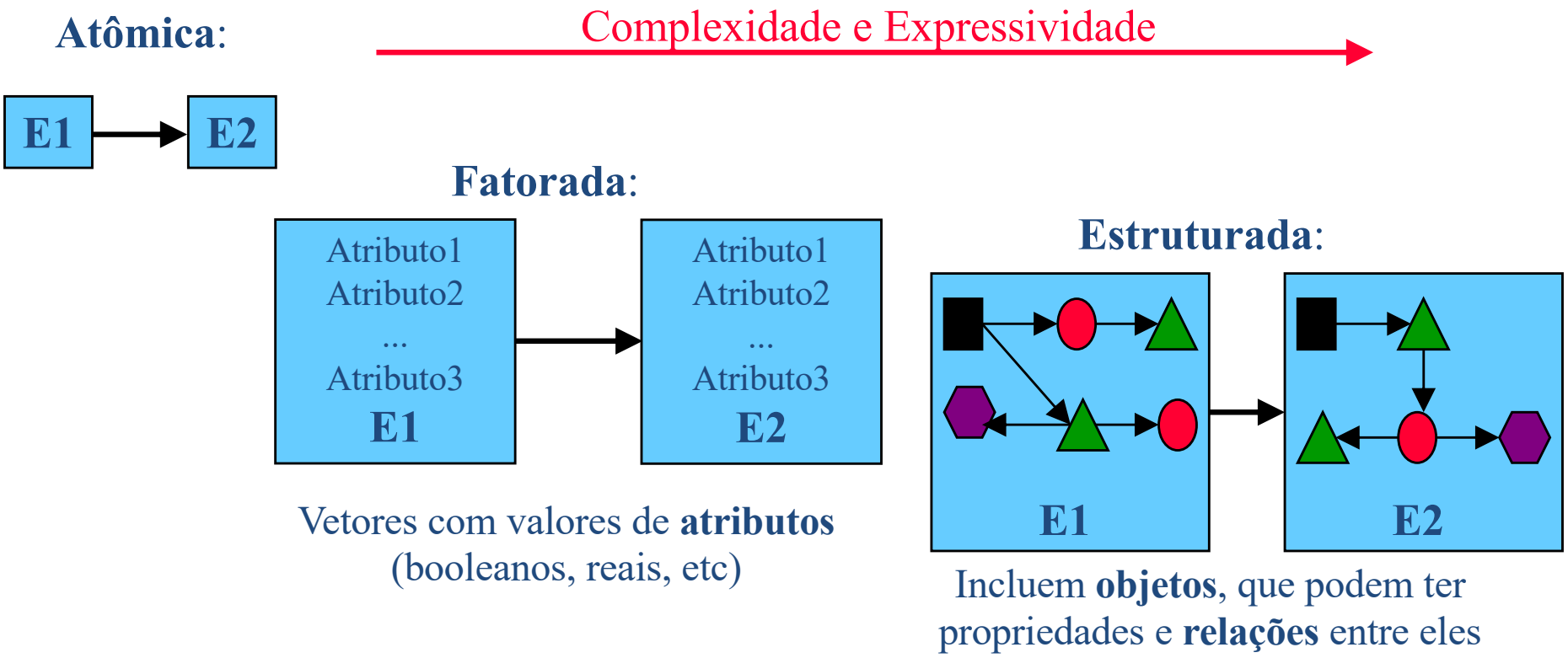


Definição de Problema: Quatro componentes

- **Estado inicial** do problema (onde o agente inicia)
- Descrição das possíveis **ações** do agente:
  - Pela função sucessor: dado um estado *x*, ***suc(x)*** retorna um conjunto de pares ordenados ***(a,y)***, onde ***a*** indica cada ação válida em ***x*** e ***y*** é o estado sucessor.
  - Pelo conjunto de operadores que podem ser aplicados em um estado para gerar os sucessores.
- Um teste de **término**:
  - Pode ser um conjunto de estados-objetivos ou
  - Propriedade mais abstrata (ex. cheque-mate em xadrez)
- Uma **função de custo** da solução
  - avalia numericamente cada solução (medida de desempenho)

Representação de Estados

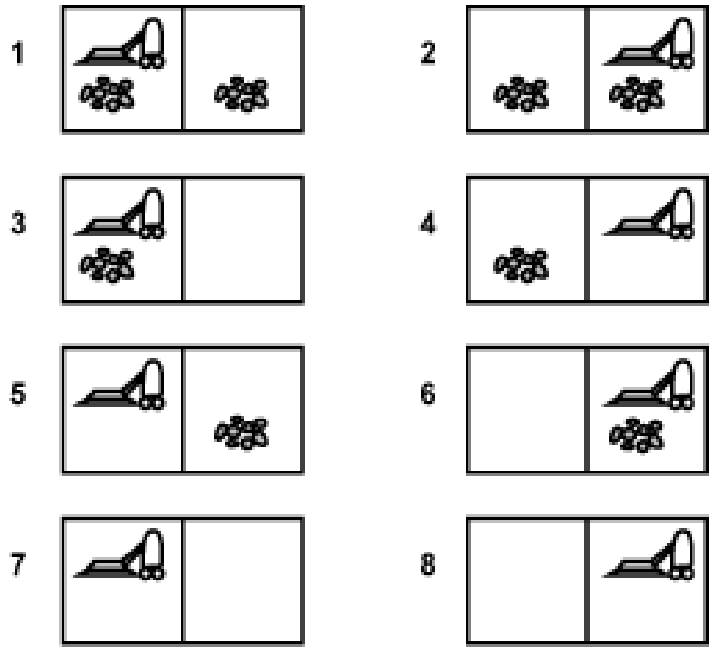
Descrição: Atômica → Fatorada → Estruturada



Definição de Solução

- O estado inicial e a função sucessor implicitamente definem o **espaço de estados** do problema
- O espaço de estados é descrito por um **grafo** onde os vértices representam estados e as arestas, ações.
- Um **caminho** no espaço de estados é uma sequência de estados conectada por uma sequência de ações.
- Uma **solução** para um problema é um caminho do estado inicial para um estado meta (objetivo).
- A **qualidade** da solução é medida pela função de custo da solução.

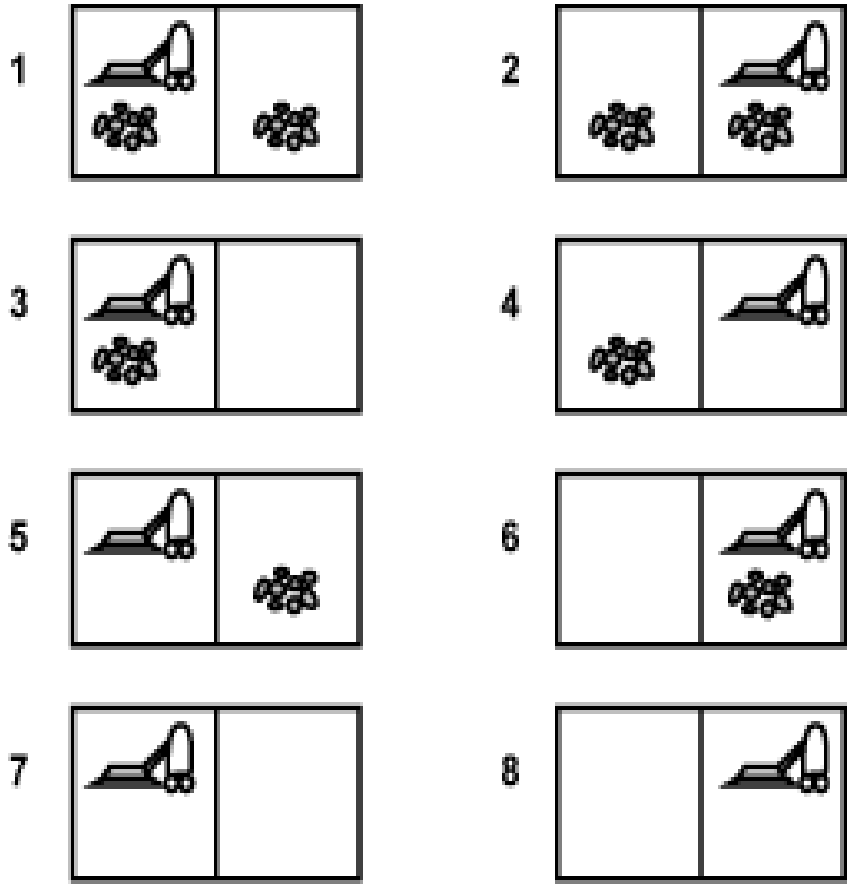
Exemplo 1: Agente Aspirador de Pó



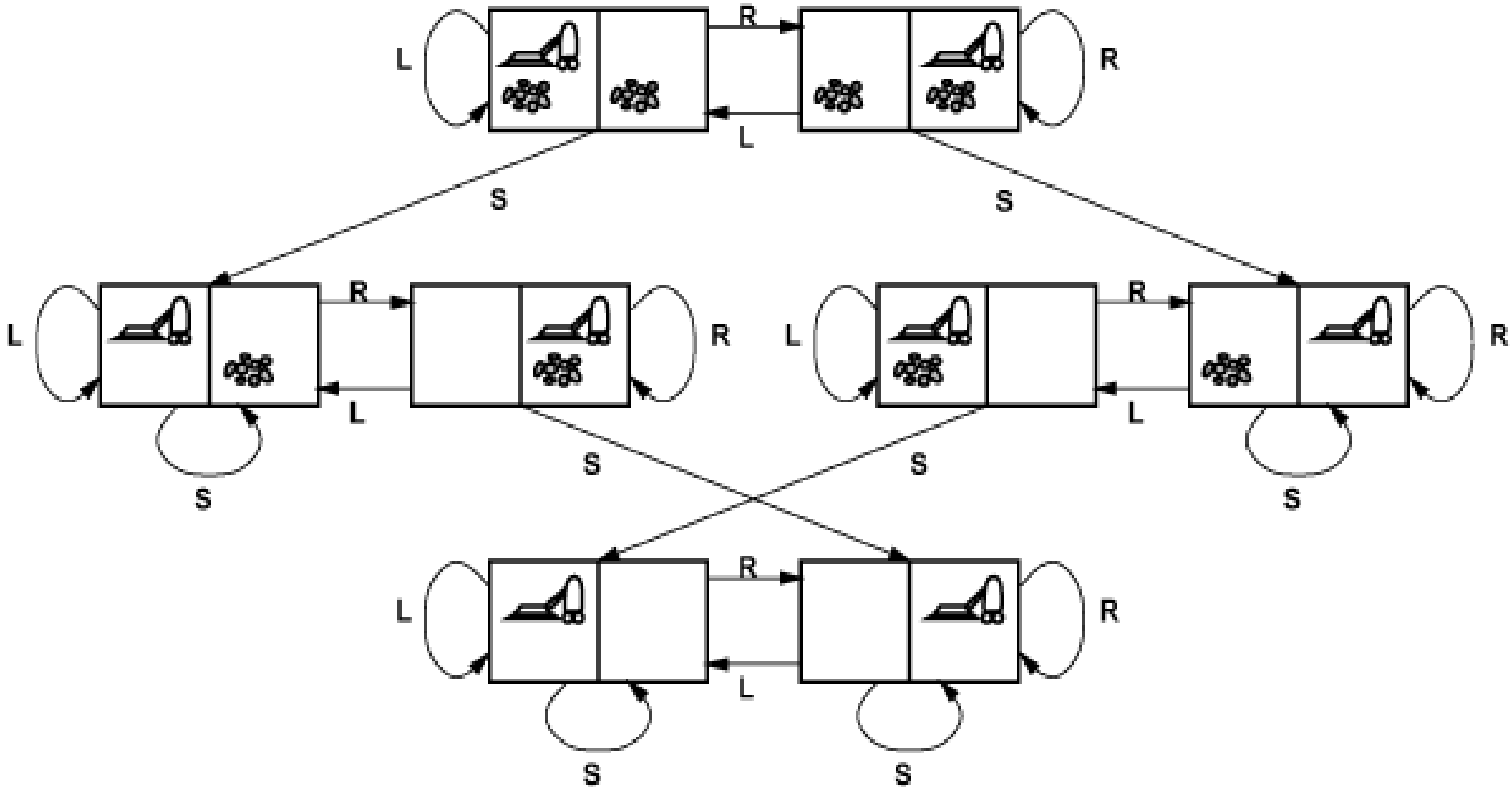
- Formulação do problema:
  - estado inicial = qualquer um dos 8 estados acima
  - função sucessor: operadores = mover direita (R), mover esquerda (L), aspirar (S); suc(1) = {(R,2), (L,1), (S,5)}, etc..
  - teste de término = os dois quartos limpos (estado 7 ou 8)
  - custo do caminho = quantidade de ações realizadas (custo 1 para cada ação)

Exercício 1:

- Considerando os oito estados possíveis (ao lado), representar o espaço de estados como um grafo, com os estados como vértices e as ações como arestas.
- Ações: R, L e S



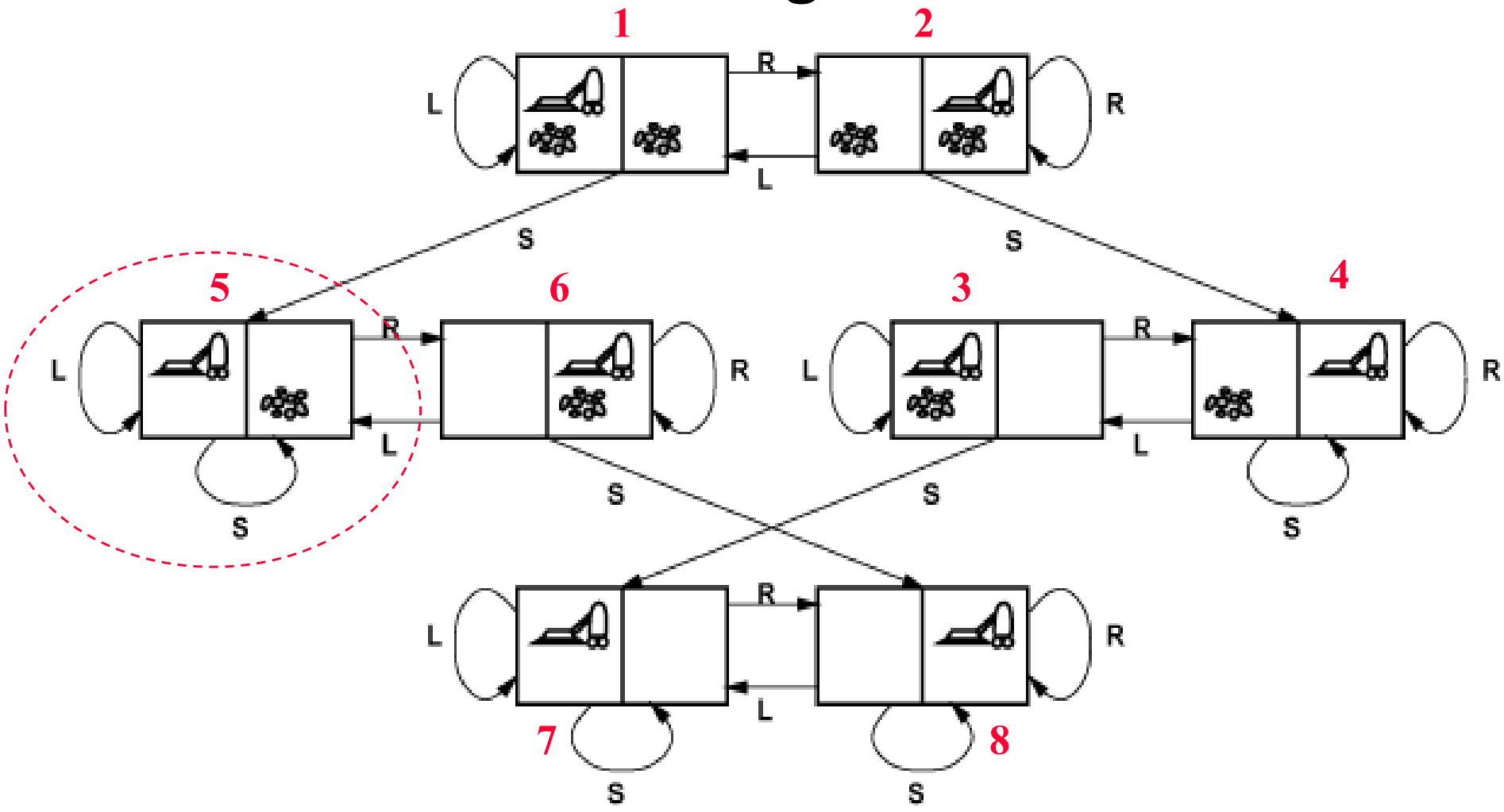
Espaço de estados do agente aspirador



Exercício 2

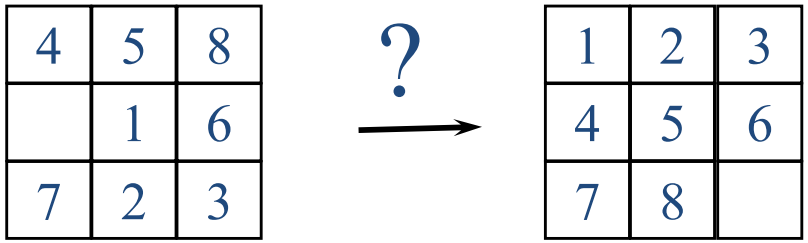
- Considere que o agente inicie no estado 5.
- Utilizando o espaço de estados do problema, encontre pelo menos três soluções para o problema.

Problema: se agente em 5?



Solução 1: LLSRRS ➔ CUSTO=6
Solução 2: SRRS ➔ CUSTO=4
Solução 3: RS ➔ CUSTO=2 .....

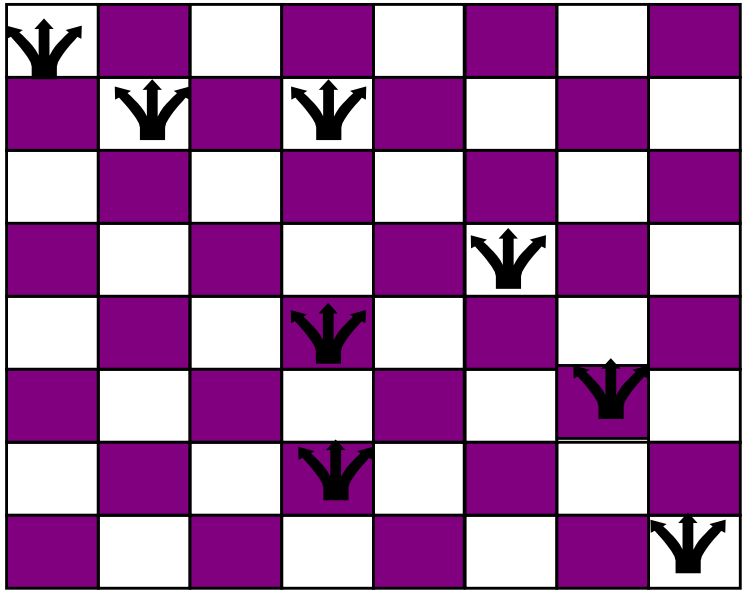
Exemplo 2: Jogo dos 8 Números



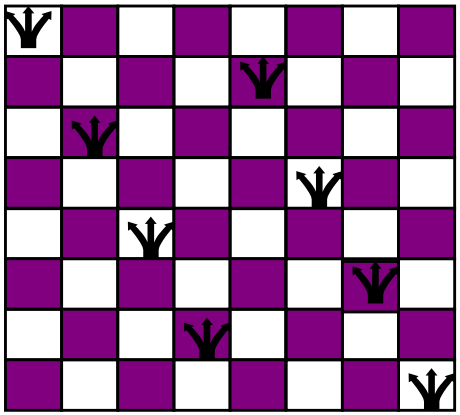
- Estado inicial:
  - cada estado especifica a posição de cada uma das 8 peças e do branco no tabuleiro de 9 posições
  - O estado inicial pode ser qualquer estado
- Função sucessor:
  - Gera os estados possíveis que resultam ao aplicar cada uma de 4 ações: branco para esquerda (L), para a direita (R), para cima (U), para baixo (D)
- Teste de término
  - Números ordenados, branco em [3,3].
- Custo do caminho
  - quantidade de ações realizadas (custo 1 para cada ação)

Importância da formulação (1)

- Jogo das 8 Rainhas
  - Teste de término: dispor 8 rainhas de forma que não possam se "atacar"
  - Custo da solução: ignorado aqui.



Importância da formulação (2)



- **Formulações:**
  - Incremental: envolve operadores que "crescem" a descrição de estado, iniciando com um tabuleiro vazio
  - Estado completo:
    - ***Estado inicial****: qualquer estado com disposição das 8 rainhas, uma em cada coluna*
    - ***Função sucessor****: retorna todos os possíveis estados ao mover uma rainha para outra casa na mesma coluna (cada estado tem 8x7=56 sucessores).*

Importância da formulação (3)

- Formulação incremental (1)
  - **Estado inicial**: tabuleiro sem rainhas; estado: qualquer disposição de 0 a 8 rainhas no tabuleiro
  - **Função sucessor**: adicionar uma rainha a qualquer célula vazia
  - **Teste de término**: 8 rainhas sem ataque mútuo

  Nesta formulação o espaço de estados é:

  $= 64 \times 63 \times \ldots \times 57 \approx 3 \times 10^{14}$

Importância da formulação (4)

- Formulação incremental (2)
  - **Estado inicial**: tabuleiro sem rainhas; estado: tabuleiro com n ($0 \leq n \leq 8$) rainhas dispostas na n-ézima coluna mais à esquerda sem ataque mútuo
  - **Função sucessor**: adicionar uma rainha em qualquer casa na coluna vazia mais à esquerda de forma que não possa ser atacada (**teste gradual**)

Formulação melhor (espaço de estados bem menor!)

Como encontrar a solução?

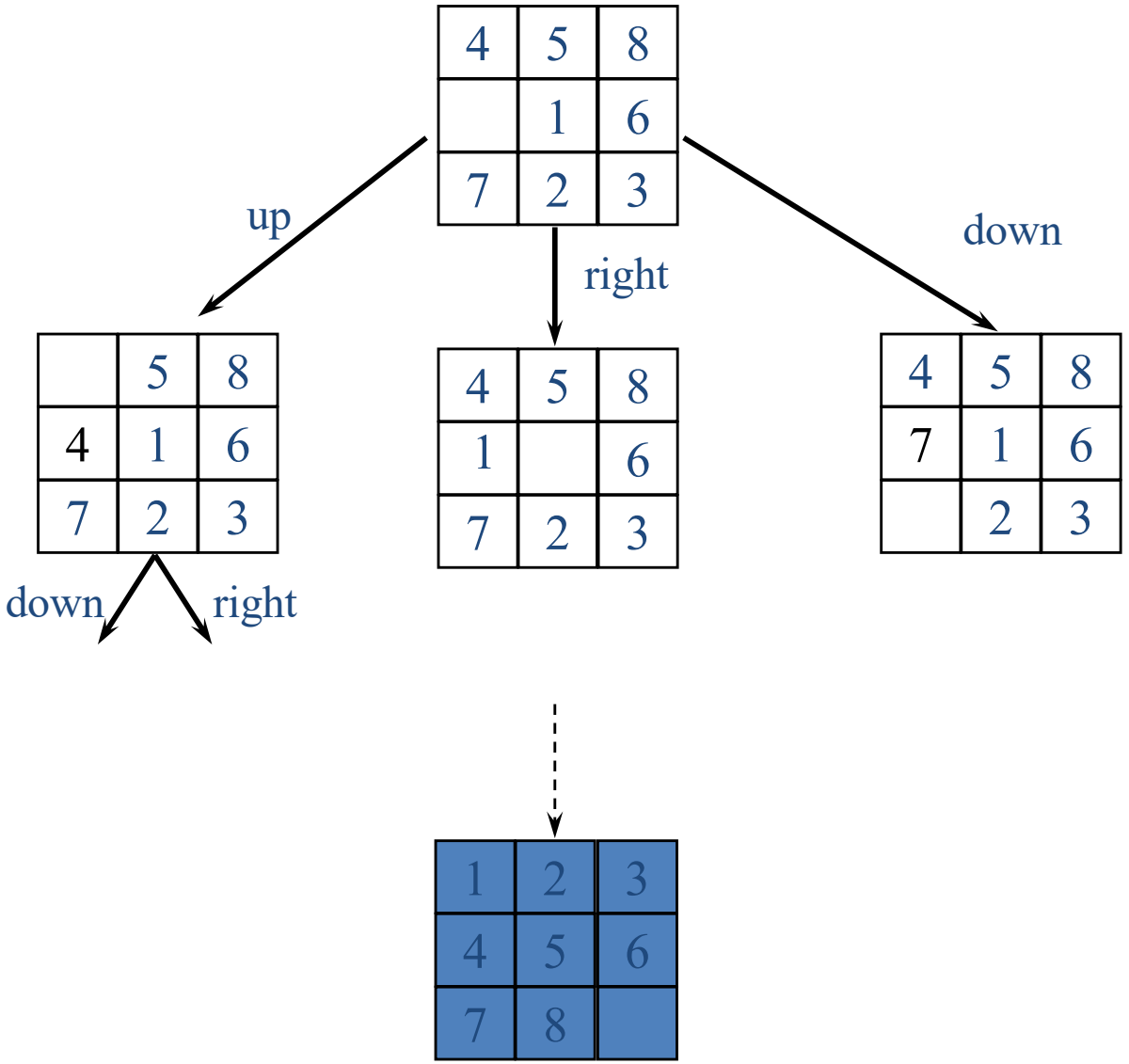
- Uma vez o problema bem formulado, o estado meta deve ser **buscado** no espaço de estados
- A busca é representada em uma **árvore de busca**:
  1. **Raiz**: corresponde ao estado inicial
  2. **Expande-se o estado corrente**: aplica-se a função sucessor ao estado corrente, gerando um novo conjunto de sucessores
  3. **Escolhe-se o próximo estado** a expandir seguindo uma **estratégia de busca**
  4. **Prossegue-se** até sucesso (atingir estado meta – retorna solução) ou falha

Como encontrar a solução?

- Espaço de estados ≠ árvore de busca

- Ex.: TSP com 20 cidades
  - espaço de estados = 20 vértices (=cidades)
  - árvore de busca com infinitos vértices (na prática, a busca evita repetir estados)

Árvore de busca para o "jogo dos 8 números"



Medida de Desempenho na Busca (1)

- Desempenho de um algoritmo de busca:
  - **Completo**: se existir uma solução, ela certamente é encontrada
  - **Ótimo**: a busca encontra a solução de menor custo
  - **Complexidade temporal**: quanto tempo demora para encontrar a solução
  - **Complexidade espacial**: quanta memória é usada para realizar a busca

Medida de Desempenho na Busca (2)

- Em IA a árvore de busca é tipicamente infinita
  ➔ complexidade é expressa por:
  - **b** – fator de ramificação (*branching*) ou número máximo de sucessores de um nó;
  - **d** – profundidade (*depth*) do nó-meta mais próximo da raiz;
  - **m** – comprimento máximo de um caminho no espaço de estados.

Medida de Desempenho na Busca (3)

- Custo total = custo da solução + custo da busca
  - custo da solução (ex. TSP: caminho a percorrer, em km)
  - custo da busca (tipicamente depende da complexidade em tempo)

  Problema: relacionar custo da solução (km) com o da busca (seg)
- Espaço de estados grande:
  - compromisso (conflito) entre a melhor solução (menor custo da solução) e a solução mais barata (menor custo da busca)

Métodos de Busca

- **Busca cega** (ou busca não informada)
  - Não tem informação sobre qual sucessor é mais promissor para atingir a meta.
  - Estratégias de Busca (ordem de expansão dos nós):
    - busca em largura
    - busca de custo uniforme
    - busca em profundidade
    - busca em profundidade limitada
    - busca em profundidade com aprofundamento iterativo
    - busca bidirecional
- **Busca heurística** (ou busca informada)
  - Possui informação (estimativa) de qual sucessor é mais promissor para atingir a meta.

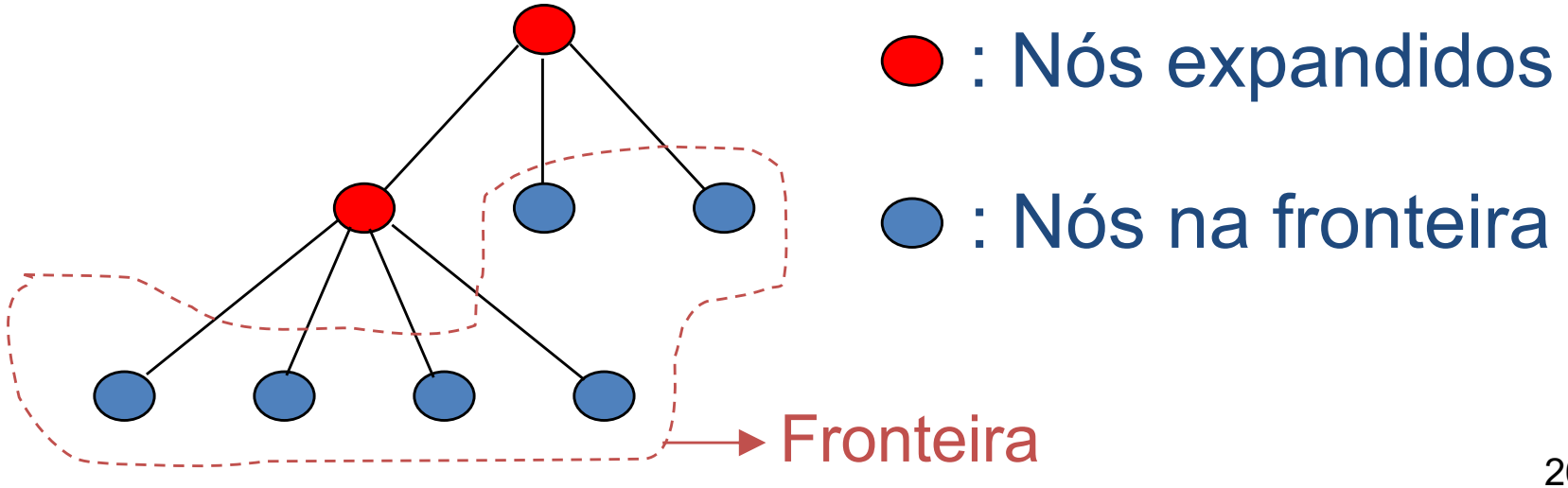
Busca não Informada

Inteligência Artificial
PCS3438

Escola Politécnica da USP
Engenharia de Computação (PCS)

Busca não informada

- A busca não informada (ou busca cega) não possui estimativas sobre qual sucessor é mais promissor para atingir a meta.
- Fronteira (ou borda): todos os nós **gerados** e ainda não **expandidos** (ou não **visitados**) da árvore de busca.



Estratégias de Busca Cega

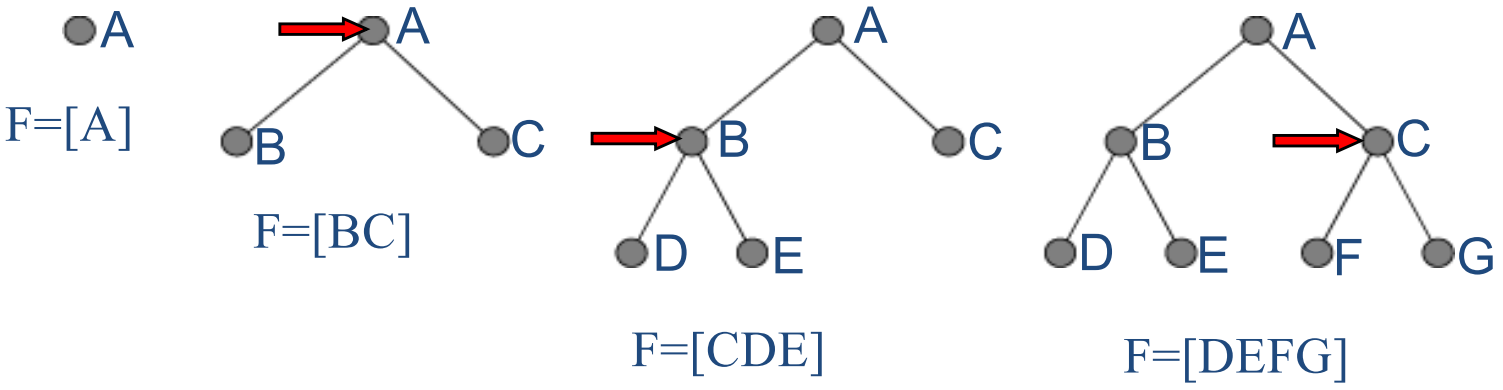
- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca em Largura

- Ordem de expansão dos nós:
  1. Nó raiz
  2. Todos os nós de profundidade 1
  3. Todos os nós de profundidade 2, etc…

Fronteira = FIFO (first-in-first-out)

➔ insere no fim da fila



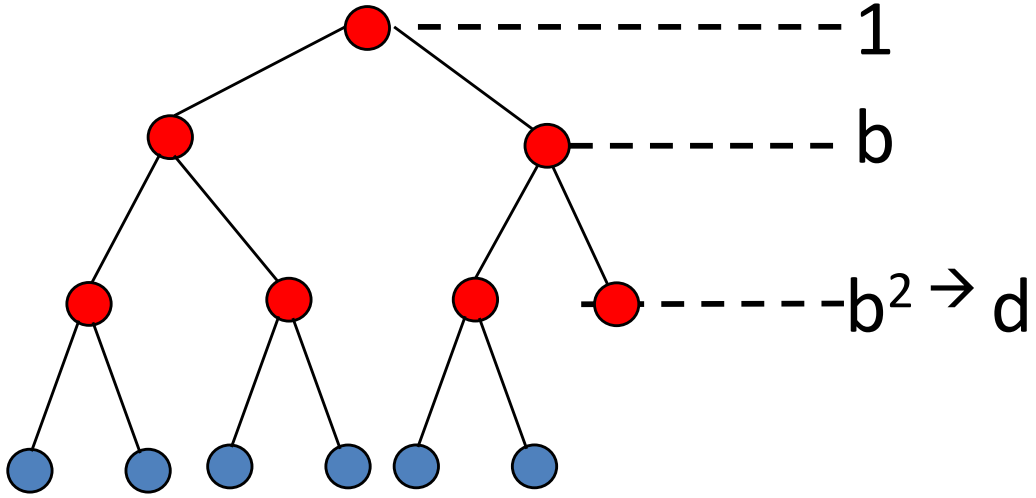
Desempenho da busca em largura

- Completa?
  - Se b finito, é completa: se um nó-meta estiver a uma profundidade d, a busca em largura sempre irá encontrá-lo.
- Ótima?
  - Nem sempre – caminho mais curto (nó-meta mais próximo da raiz) $\neq$ melhor caminho
  - É ótima se o custo do caminho for uma função não-decrescente da profundidade do nó (ex: todas ações têm mesmo custo) e custo dos passos é igual

Desempenho da busca em largura

- Complexidade de tempo
  - Meta em d, cada nó tem b filhos. No pior caso, vem (teste ao **expandir/visitar** cada nó):

$$1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$

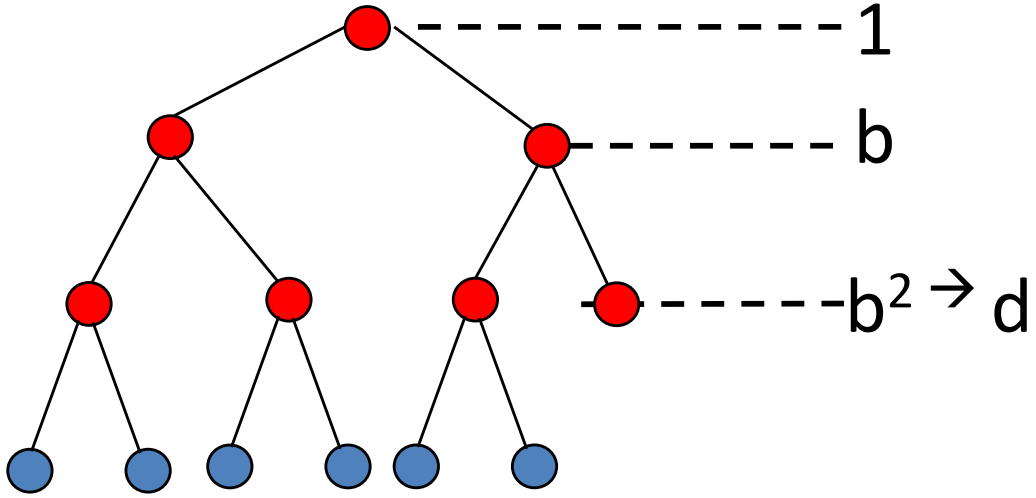


Se testar ao gerar: $1 + b + b^2 + \ldots + b^d = \mathcal{O}(b^d)$

Desempenho da busca em largura

- Complexidade de espaço
  - Mantém todos nós gerados (ou está na fronteira ou está na lista de visitados)

$$1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$



Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca de Custo Uniforme

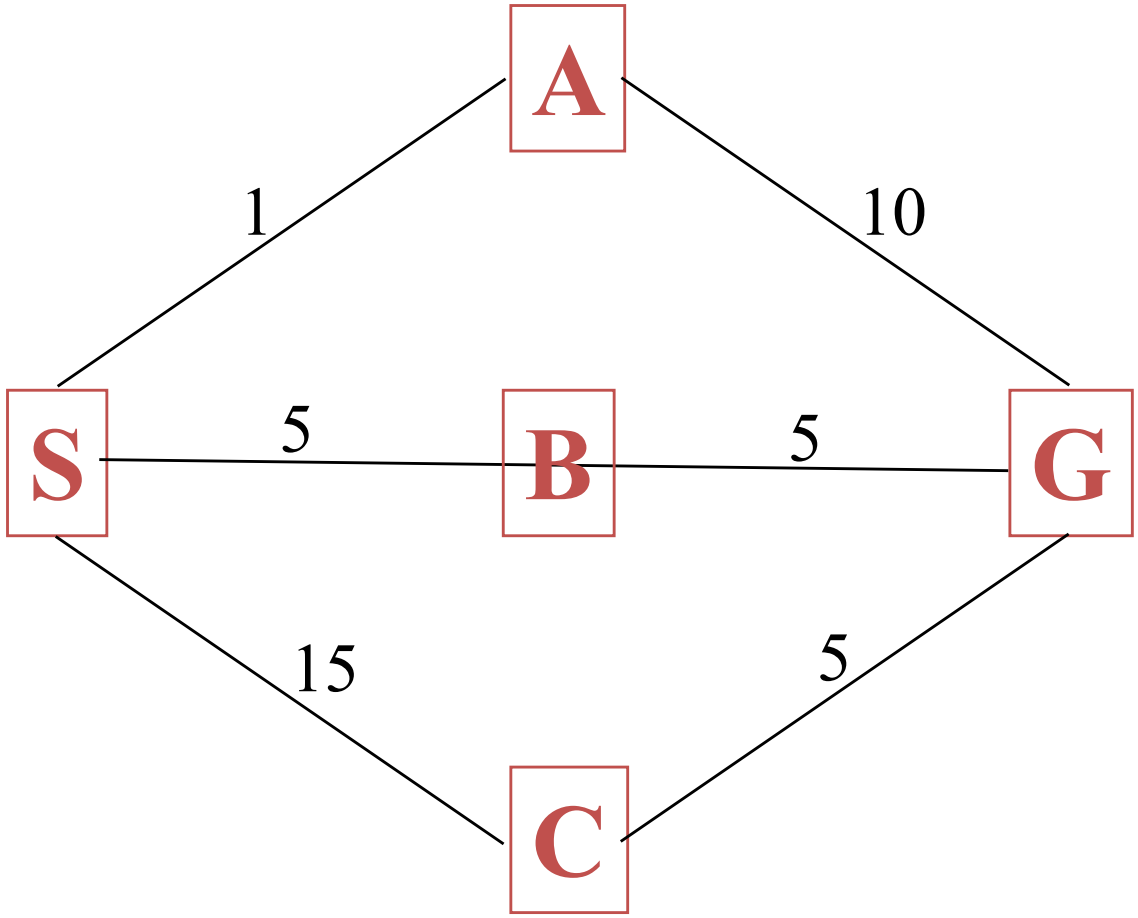
- Modifica a busca em largura:
  - **Em vez de expandir o nó gerado primeiro, expande o nó da fronteira com menor custo de caminho (da raiz ao nó)**
  - Sempre testa objetivo ao visitar (não ao gerar)

Fronteira ➔ insere em ordem crescente de custo

- Não se importa com o número de passos, mas com o custo total
- g(n) dá o custo do caminho da raiz ao nó n
  - Na busca em largura: ***g(n) = profundidade (n)***

Busca de Custo Uniforme

- **Exercício**: gerar a árvore de busca usando busca de custo uniforme, partindo de S e chegando a G. Mostre todos os estados da fronteira.



Busca de Custo Uniforme

Fronteira do exemplo anterior

- F = {S}
  - testa se S é o estado objetivo, expande-o e guarda seus filhos A, B e C ordenadamente (segundo custo) na fronteira
- F = {A, B, C}
  - testa A, expande-o e guarda seu filho G_A ordenadamente
  - **obs.:** o algoritmo de geração guarda na fronteira todos os nós gerados, testando se um nó é o objetivo **apenas** quando ele é **retirado** da lista (i.e., visitado ou expandido)!

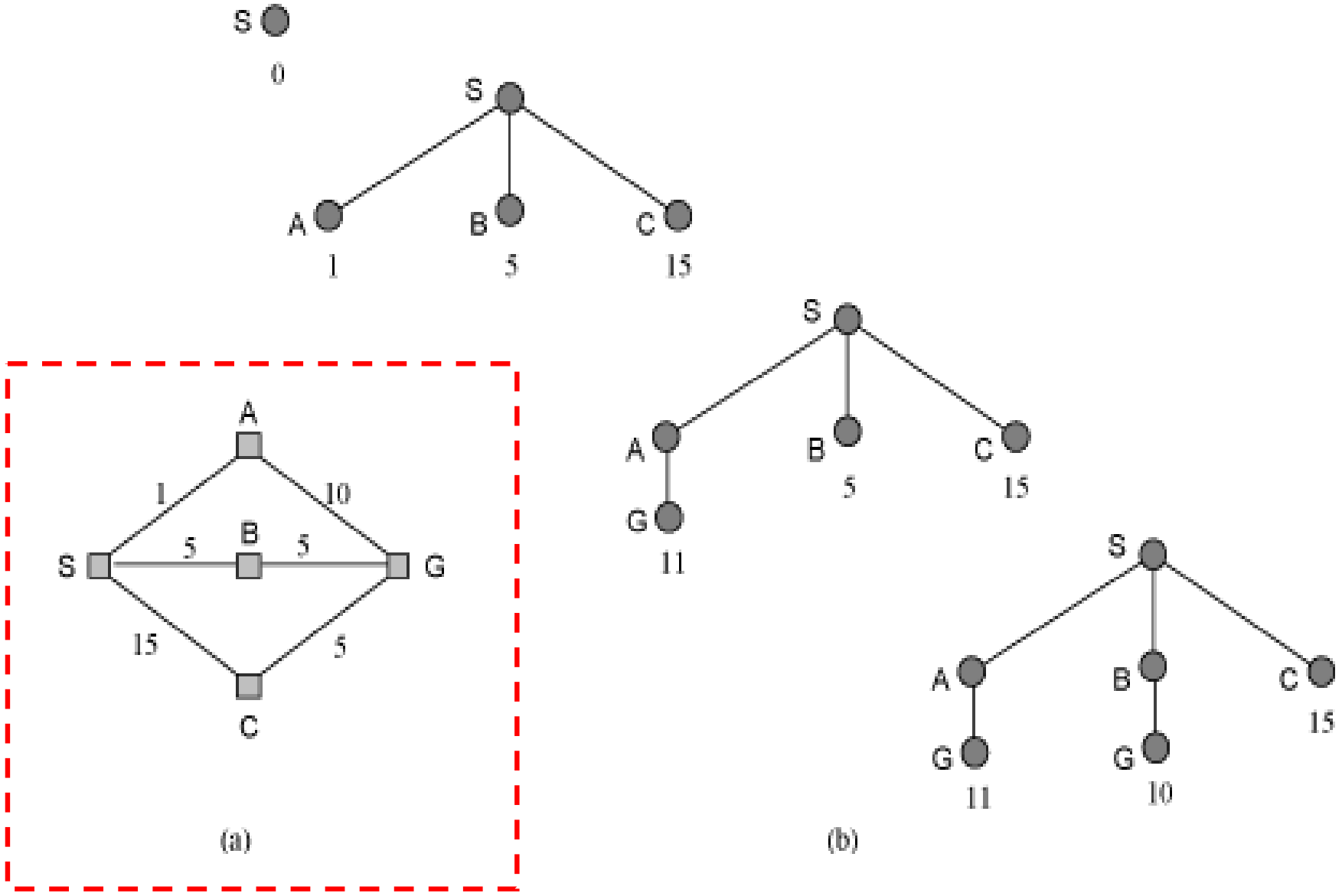
Busca de Custo Uniforme

Fronteira do exemplo anterior

- F= {B, G_A, C}
  - testa B, expande-o e guarda seu filho G_B ordenadamente
- F= {G_B, G_A, C}
  - testa G_B e para!

Observe que o menor custo (portanto, o melhor caminho) é armazenado para cada vértice (G pela caminho por A foi trocado por G pelo caminho por B).

Busca de Custo Uniforme



Desempenho da Busca de Custo Uniforme

- Completa? Só *se custo de cada ação* $\geq \varepsilon$, $\forall n$
  - ε *é uma constante pequena positiva*
    - *Loop* infinito: se existir um caminho com uma sequência infinita de ações de custo=0
- Ótima? *Só se g(sucessor(n)) > g(n)*
  - *custo* ***no mesmo caminho*** *sempre cresce (i.e., não tem ação com* ***custo negativo ou 0****)*
- Complexidade de tempo
  - *C*=custo da solução ótima (custo de cada ação*$\geq\varepsilon$*)*

➔ Pior caso: $O\ (b^{1+\lfloor C^*/\ \varepsilon\ \rfloor})$, o que pode ser bem maior que b^d

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca em Profundidade

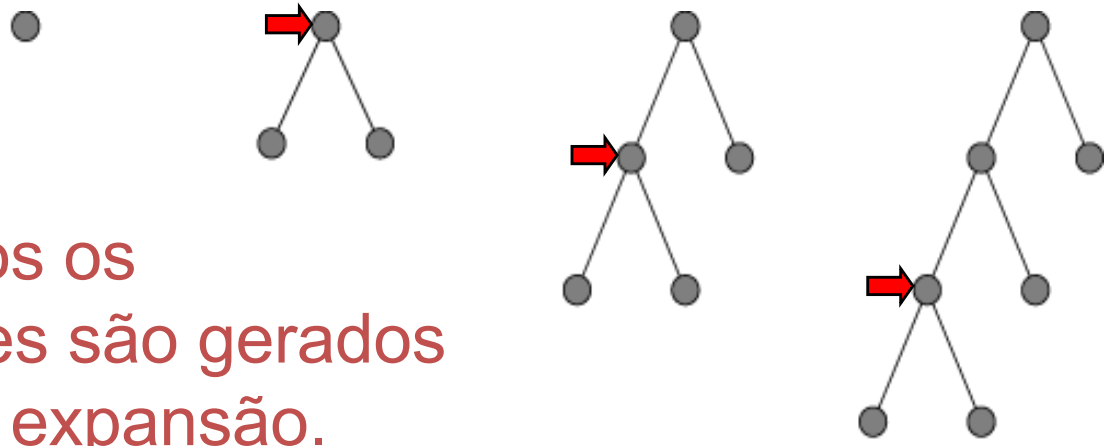
- Ordem de expansão dos nós:
  1. Nó raiz
  2. Primeiro nó de profundidade 1
  3. Primeiro nó de profundidade 2, etc…

Fronteira = LIFO (last-in-first-out)

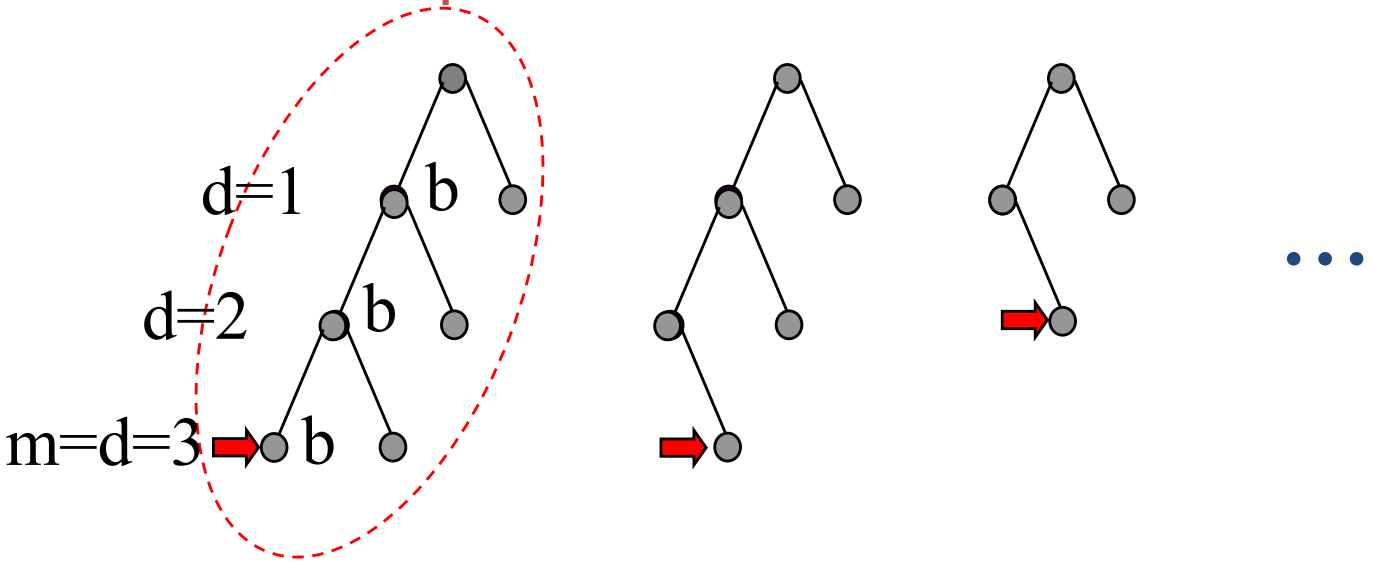
➔ insere na pilha

– Em cada visita a um nó, gera todos seus sucessores

Busca em Profundidade



Aqui, todos os sucessores são gerados durante a expansão.



Neste exemplo, nós com profundidade 3 não têm sucessores.
Nós sem sucessores e já expandidos são “apagados”.

Desempenho da Busca em Profundidade

- Esta estratégia **não é *completa*** (caminho pode ser infinito) **nem é *ótima*.**
  - Se usar uma estratégia que não permite estados repetidos nem caminhos redundantes, ela é completa (mas isso piora a complexidade espacial por ter que armazenar o histórico de estados).
- Complexidade espacial:
  - mantém na memória o caminho que está sendo expandido no momento, e os nós irmãos dos nós no caminho para possibilitar o retrocesso (*backtracking*)
  - Apaga subárvores já visitadas
  - Para espaço de estados com fator de ramificação b e **profundidade máxima m** (m pode ser >> d), requer apenas **bm+1** de memória ➔ O (bm)

Desempenho da Busca em Profundidade

- Complexidade temporal: $O(b^m)$, no pior caso.
  - Para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que a busca em largura.
  - Esta estratégia deve ser evitada quando as árvores geradas são muito *profundas* (**m** muito grande) ou geram *caminhos infinitos*.

Variante: **Busca com Retrocesso (backtracking search)**

- Parecida com BP, mas **somente UM sucessor é gerado em cada iteração**
  - na BP, todos os sucessores são gerados na expansão do nó pai
- Portanto, requer só $\mathcal{O}$(m) de memória
  - BP requer $\mathcal{O}$(bm) de memória
- Restrição: deve ser capaz de retornar ao pai e criar o novo sucessor

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca em Profundidade Limitada

- Evita o problema de árvores não limitadas ao impor um limite máximo (ℓ) de profundidade para os caminhos gerados.
  - O domínio do problema estabelece a profundidade limite.
  - Problema: definir limite ℓ adequado!
- Completa? Somente se $\ell \geq$ d.
- Ótima? Não, exceto se ℓ = d e custo dos passos é igual.
- Complexidade espacial: $\mathcal{O}$ (bℓ)
- Complexidade temporal: $\mathcal{O}$ (b$^{\ell}$) no pior caso.

BP é caso particular, com $\ell = \infty$

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca com Aprofundamento Iterativo (BAI)

- Tenta limites com valores crescentes, partindo de zero, até encontrar a primeira solução (em d).
  - Combina vantagens da busca em largura (BL) com as da busca em profundidade (BP).
  - Em geral, é a estratégia preferida de busca cega para quando o espaço de estados é muito grande e a profundidade da solução d é desconhecida.
- Possui uma variante, a **Busca com Comprimento Iterativo (BCI)** – analogia entre BAI e BL, e BCI e Custo Uniforme: usa incremento iterativo do **custo** do caminho em vez de incremento na profundidade (mas BCI não é eficiente!)

Desempenho da BAI

- Completa? Sim se b for finito (idem BL).
- Ótima? Sim se o custo do caminho for uma função crescente com a profundidade do nó e custo dos passos é igual (idem BL).
- Complexidade espacial:

  $\mathcal{O}(bd)$ (idem BP).

Desempenho da BAI

- Complexidade temporal: nós na profundidade da menor solução (d) são gerados 1 vez, em d-1 são gerados 2 vezes, ... na profundidade 1 são gerados d vezes:

$$(d)b + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d = \mathcal{O}(b^d)$$

OBS: BL gera alguns nós em d+1 e BAI não gera.

BL: $\mathcal{O}(b^{d+1})$

Na realidade, BAI é mais rápida que BL.

Estratégias de Busca Cega

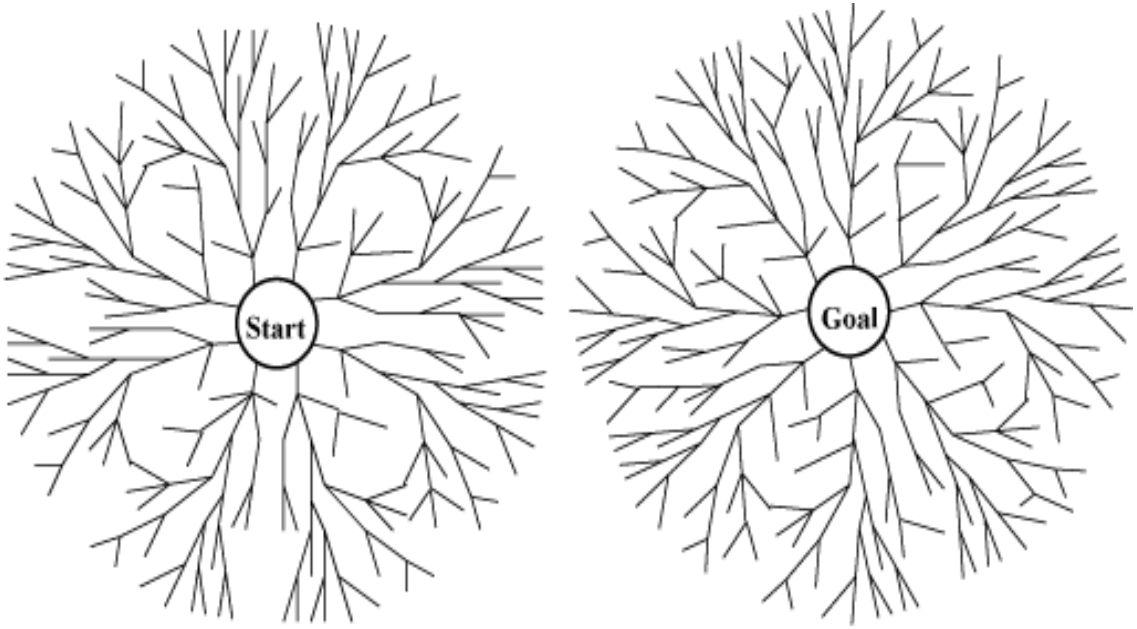
- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca Bidirecional (1)

- Busca em duas direções (duas buscas simultâneas):
  - para frente, a partir do nó inicial, e
  - para trás, a partir do nó final (objetivo)
- A busca para quando o nó a ser expandido por uma busca se encontra na fronteira da outra busca.
- **Motivação:**

$b^{d/2} + b^{d/2} < b^{d}$.

Ou: a área de dois círculos de raio R é menor que a área de um círculo de raio 2R



Busca Bidirecional (2)

- Para BL nas duas direções: $O(b^{d/2})$ no tempo e no espaço. Completeza e otimalidade: idem BL.
  - É possível utilizar *estratégias* diferentes em cada direção da busca (podendo sacrificar desempenho)
- Porém, encadeamento reverso (da meta para o início) só é possível se todas as ações no espaço de estado forem reversíveis.
- Outro problema: quando há vários estados-meta ou quando é muito difícil computar os estados-meta pelo teste de término (ex. estados para cheque-mate).

Estratégias de Busca Cega

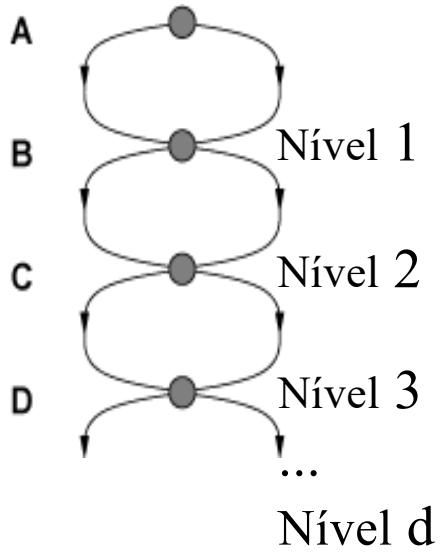
- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Evitando Estados Repetidos (1)

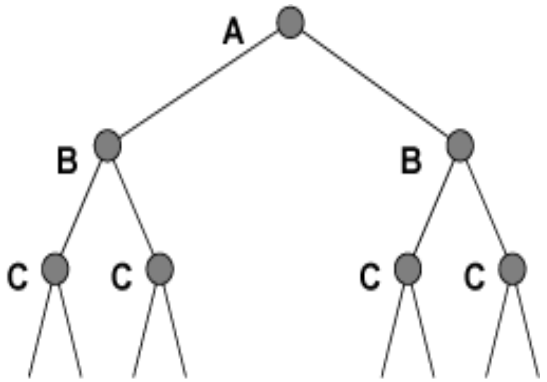
- Problema geral em Busca
  - expandir estados já previamente encontrados e expandidos
- É inevitável quando há operadores reversíveis
  - ex. encontrar rotas, canibais e missionários, 8-números, etc.
  - a árvore de busca é potencialmente infinita

Evitando Estados Repetidos (2)

Espaço de estados



Árvore de busca



Exemplo:

Número de estados= **d+1;**

2^d caminhos na árvore de busca (combinações possíveis de caminhos de A ao último estado, no nível d).

- Idéia:
  - **podar** (*prune*) estados repetidos, para gerar apenas a parte da árvore que corresponde ao grafo do espaço de estados (que é finito!)

Evitando Estados Repetidos (3)

- **Problema**: deve armazenar todos nós gerados!
  - Além da lista de fronteira (também chamada de ***open list***), os algoritmos precisam da lista de nós visitados / expandidos (***closed list***)
  - Cada nó gerado é comparado com aqueles da *closed list*: se for repetido, descarta aquele de caminho com custo pior.
  - Pode ser implementado mais eficientemente com *hash tables*
  - BP e BAI: perdem propriedade de complexidade linear no espaço.

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Conhecimento Incompleto

Busca com Conhecimento Incompleto

- E se o ambiente não for totalmente observável, determinístico e o agente não souber as consequências de suas ações?

1. Problemas **conformantes** (sem sensores): não sabe seu estado inicial e, assim, cada ação poderia levar a muitos estados sucessores.
2. Problemas **contingenciais**: quando o ambiente é parcialmente observável ou suas ações possuem incertezas. Se a incerteza é causada por ações de outro agente, é um problema com adversário.
3. Problemas de **exploração**: quando os estados e a dinâmica do ambiente são desconhecidos, o agente precisa atuar para descobrí-los (caso extremo dos problemas contingenciais).

Problemas Conformantes (sem sensores)

- Estado de crença (***belief state*** – *bs*): **conjunto de estados em que o agente acredita estar.**
- **Busca ocorre no espaço de *bs:***
  - ações são aplicadas nos bs, gerando bs sucessores (∪ sucessores da ação aplicada a cada estado ∈ bs)
  - *Solução*: caminho que leva a um bs que só contenha estados-meta
  - Mesmo procedimento para ações não determinísticas

Exemplo com o agente aspirador

- Bs inicial =

{1, 2, 3, 4, 5, 6, 7, 8}

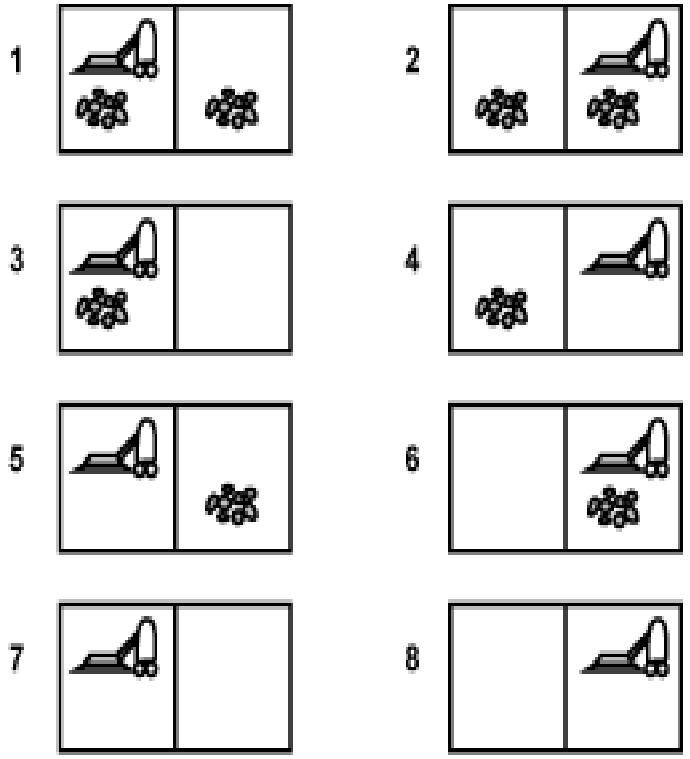
- Se aplicar a=Right em bs, vem:

  bs’ ={2, 4, 6, 8} ...

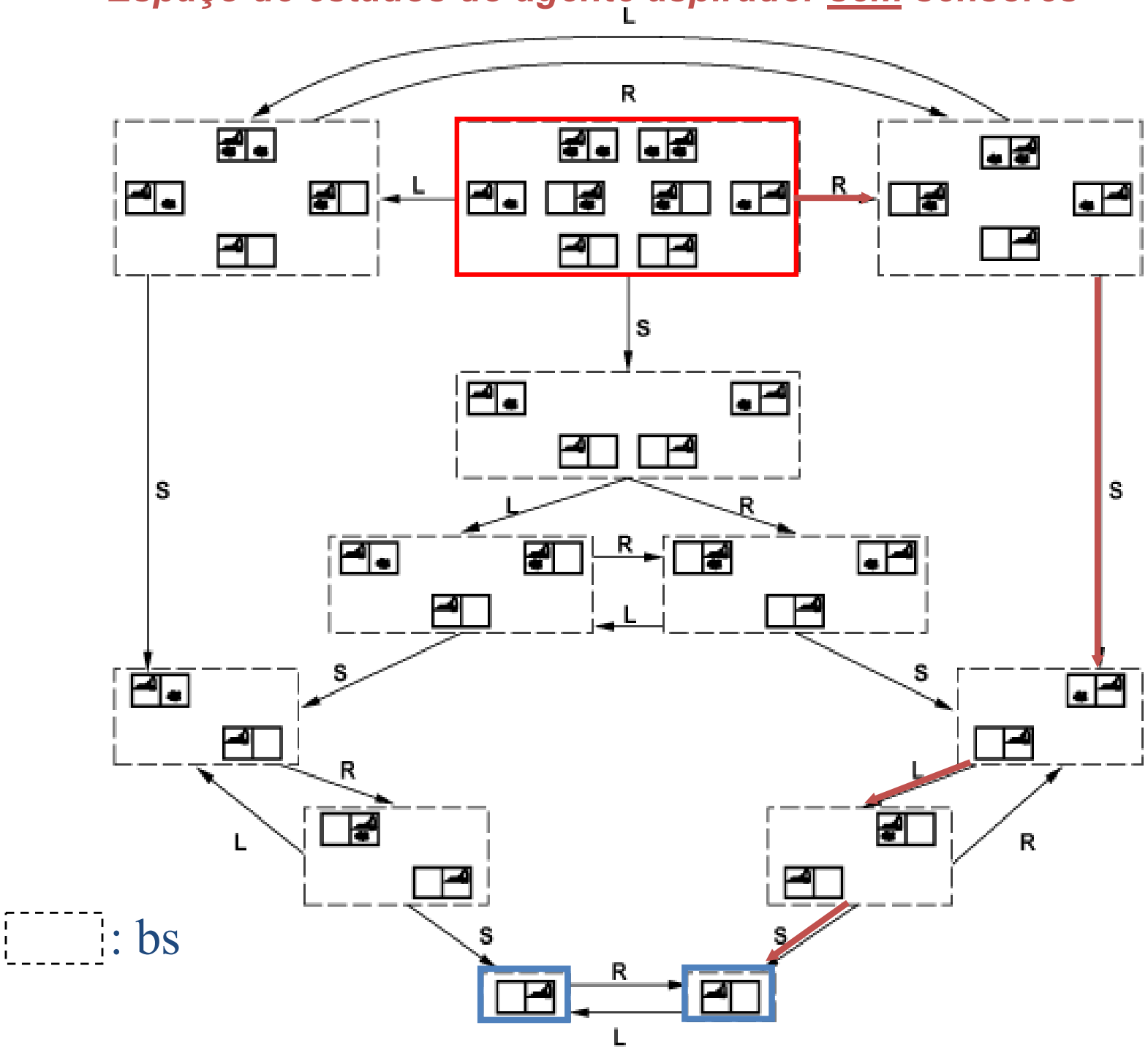
- Assim,

seq=[Right, Suck, Left, Suck]

é uma solução!



Espaço de estados do agente aspirador sem sensores



: bs

Problemas Contingenciais (1)

incertezas e observabilidade parcial

- Nenhuma sequência fixa de ações garante a solução.
- Muitos problemas reais são contingenciais pois a predição exata é impossível.
- Planejamento condicional:
  - no meio da solução são inseridas ações de sensoriamento que direcionam a execução.
  - A solução normalmente é uma árvore, onde ações são selecionadas em função das contingências sensoriadas.
- Uso de abordagem probabilística

Problemas Contingenciais (2)

- Planejamento contínuo:
  - monitora e atualiza seu modelo do mundo continuamente, mesmo quando em deliberação;
  - assim que tiver um plano parcial, executa; revê metas, inclui novas metas, descarta metas, etc.
  - Projetado para interagir indefinidamente com o ambiente. Também usado em problemas de exploração.

Problemas Contingenciais (3)

- Monitoramento da execução e replanejamento:
  - agente planeja uma solução e a executa, monitorando a execução;
  - se ocorrer contingências e o plano precisar ser revisado, o agente replaneja a partir do estado que estiver.