# Busca no Espaço de Estados

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

# Idéia básica

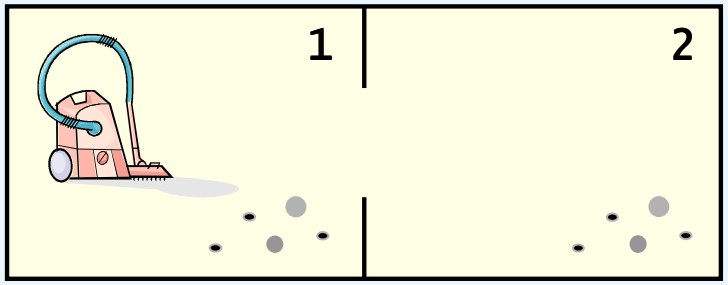
## A idéia básica da busca no espaço de estados

é considerar a existência de um **agente** cujas **ações** modificam o **estado** do mundo.

**Suposições clássicas**:

- o mundo muda apenas quando o agente executa uma ação
- não há incerteza acerca dos efeitos das ações do agente
- a meta do agente é alcançar um determinado estado final

## Exemplo 1. Mundo do aspirador simplificado [Russel & Norvig, 2004]



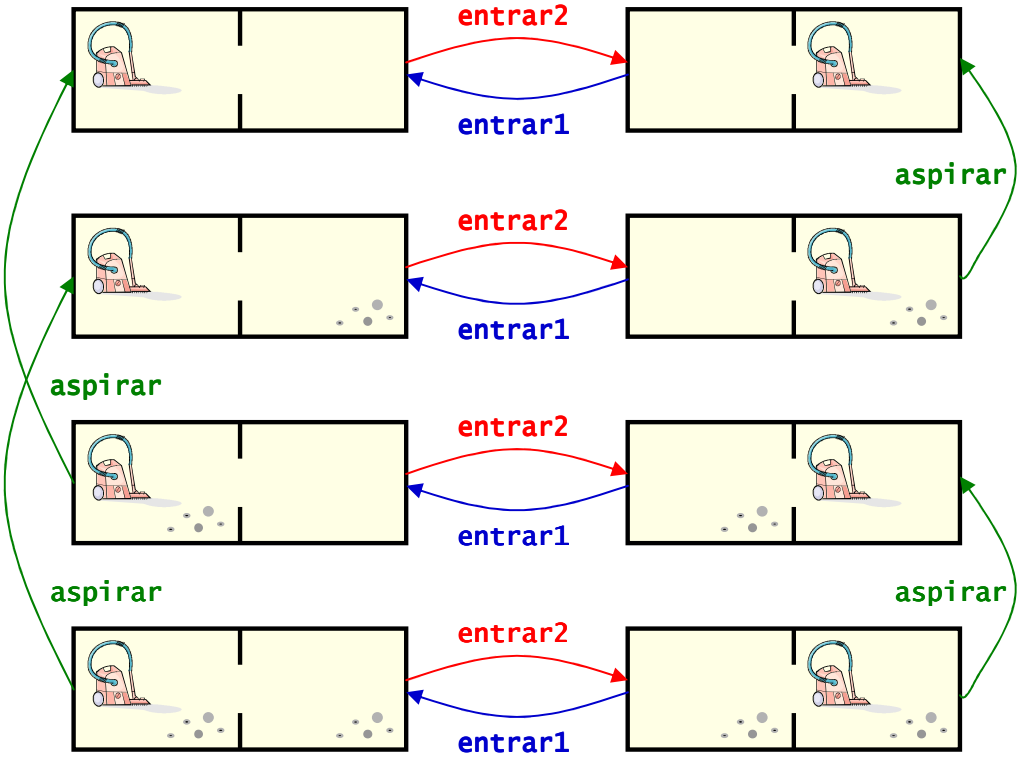

**Ações do agente**:

- `entrar1`
- `entrar2`
- `aspirar`

# Espaço de estados

**Um espaço de estados é um grafo definido por:**

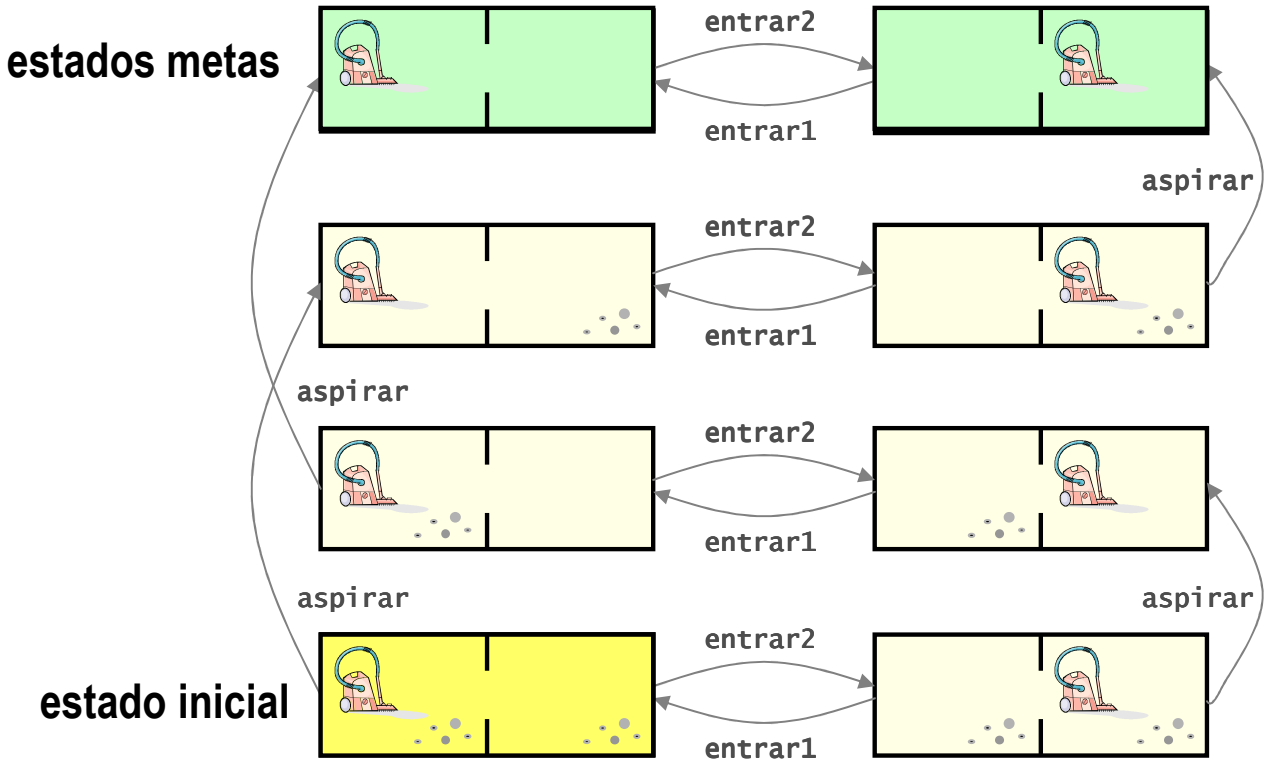
- um **conjunto de estados** (todos os possíveis estados em que o mundo pode estar)
- um **conjunto de ações** (todas as ações que o agente é capaz de executar).



# Problema de busca

**Um problema de busca em um determinado espaço de estados é definido por:**

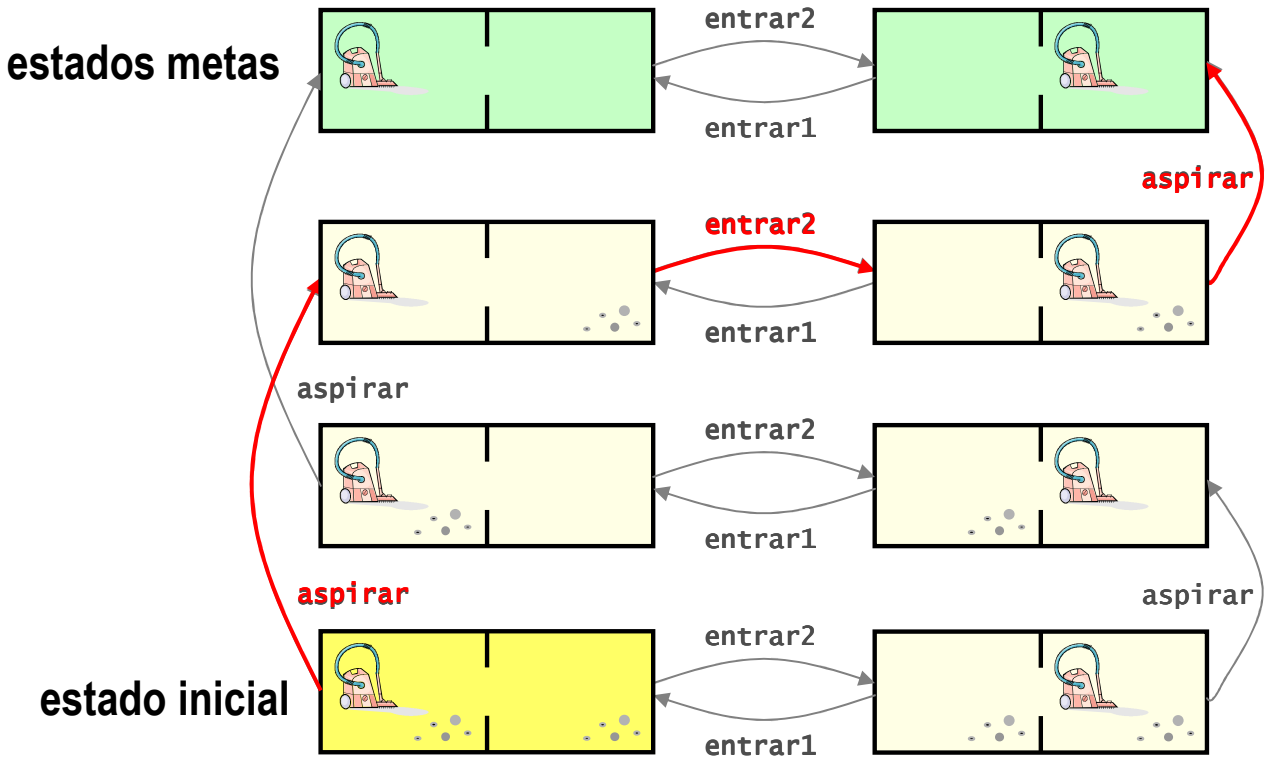
- um **estado inicial** (indicando o estado corrente do mundo do agente)
- um conjunto de **estados finais** ou **estados metas** (indicando a meta do agente)



# Solução

**A solução para um problema de busca é**

uma sequência de ações (ou **plano**) que, quando executada pelo agente, transforma o estado inicial em um estado meta



# Representação de estados

## Estados são representados por estruturas

cujos componentes denotam suas propriedades ou características particulares

No mundo do aspirador, há três características que distinguem os possíveis estados do mundo. Assim, podemos representar estes estados por estruturas da forma `[X,Y,Z]`, onde:

- `X` ∈ {1,2} indica se o agente está na sala 1 ou 2
- `Y` ∈ {l,s} indica se a primeira sala está limpa ou suja
- `Z` ∈ {l,s} indica se a primeira sala está limpa ou suja

## Exemplo 2. Representação do estado inicial e dos estados metas

```
inicial([1,s,s]).
meta([_,l,l]).
```

# Representação de ações

## Ações são representadas por regras da forma

ação($\alpha$,$e_1$,$e_2$) :- $\beta$.

onde:

- $\alpha$ é o identificador da ação
- $e_1$ é o estado em que o mundo se encontra antes da execução da ação $\alpha$
- $e_2$ é o estado em que o mundo fica após a execução da ação $\alpha$
- $\beta$ é uma especificação das precondições e/ou efeitos da ação $\alpha$

## Exemplo 3. Representação da ação `aspirar` no mundo do aspirador

```
ação(aspirar,[X1,Y1,Z1],[X2,Y2,Z2]) :-
   X1=1, Y1=s,                          % precondições
   X2=1, Y2=l, Z2=Z1.                   % efeitos
ação(aspirar,[X1,Y1,Z1],[X2,Y2,Z2]) :-
   X1=2, Z1=s,                          % precondições
   X2=2, Y2=Y1, Z2=l.                   % efeitos
```

# Representação de ações

## Representação implícita de precondições e efeitos

Precondições e efeitos especificados por meio de condições que envolvem apenas teste de **igualdade** e **unificação** podem ser representados de maneira implícita

## Exemplo 4. Representação completa das ações para o mundo do aspirador

```
ação(entrar1,[2,Y,Z],[1,Y,Z]).
ação(entrar2,[1,Y,Z],[2,Y,Z]).
ação(aspirar,[1,s,Z],[1,l,Z]).
ação(aspirar,[2,Y,s],[2,Y,l]).
```

## Uma ação A é aplicável a um estado E

se as suas precondições são satisfeitas neste estado

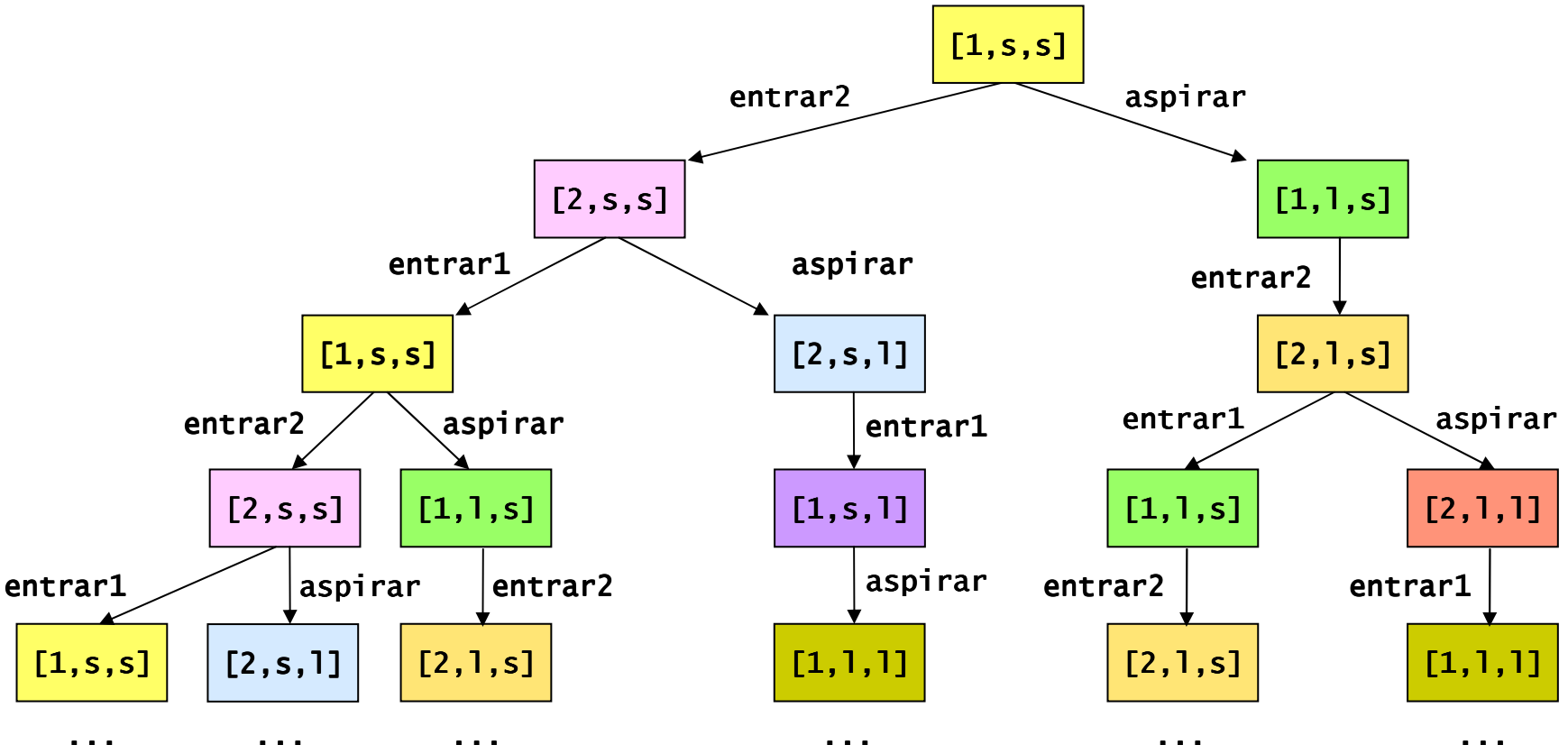
## Os estados sucessores de um estado E

são aqueles que podem ser diretamente alcançados por uma ação aplicável ao estado E

# Árvore de busca

## Uma árvore de busca

é o desdobramento (infinito) de um espaço de estados, a partir de um estado inicial particular



# Estratégias de busca

## Uma estratégia de busca

especifica em que ordem devemos visitar os nós da árvore de busca para achar uma solução

As principais estratégias de busca podem ser divididas em dois grupos:

- **busca não-informada**: as estratégias neste grupo não dispõem de informação que possibilite a escolha do nó mais promissor para ser visitado em cada instante.
  - **busca aleatória**
  - **busca em largura**
  - **busca em profundidade**
- **busca informada**: as estratégias neste grupo dispõem de informação que possibilitam a escolha do nó mais promissor para ser visitado em cada instante.
  - **busca pelo menor custo**
  - **busca pela melhor estimativa**
  - **busca ótima (A*)**

# Busca não-informada

busca aleatória

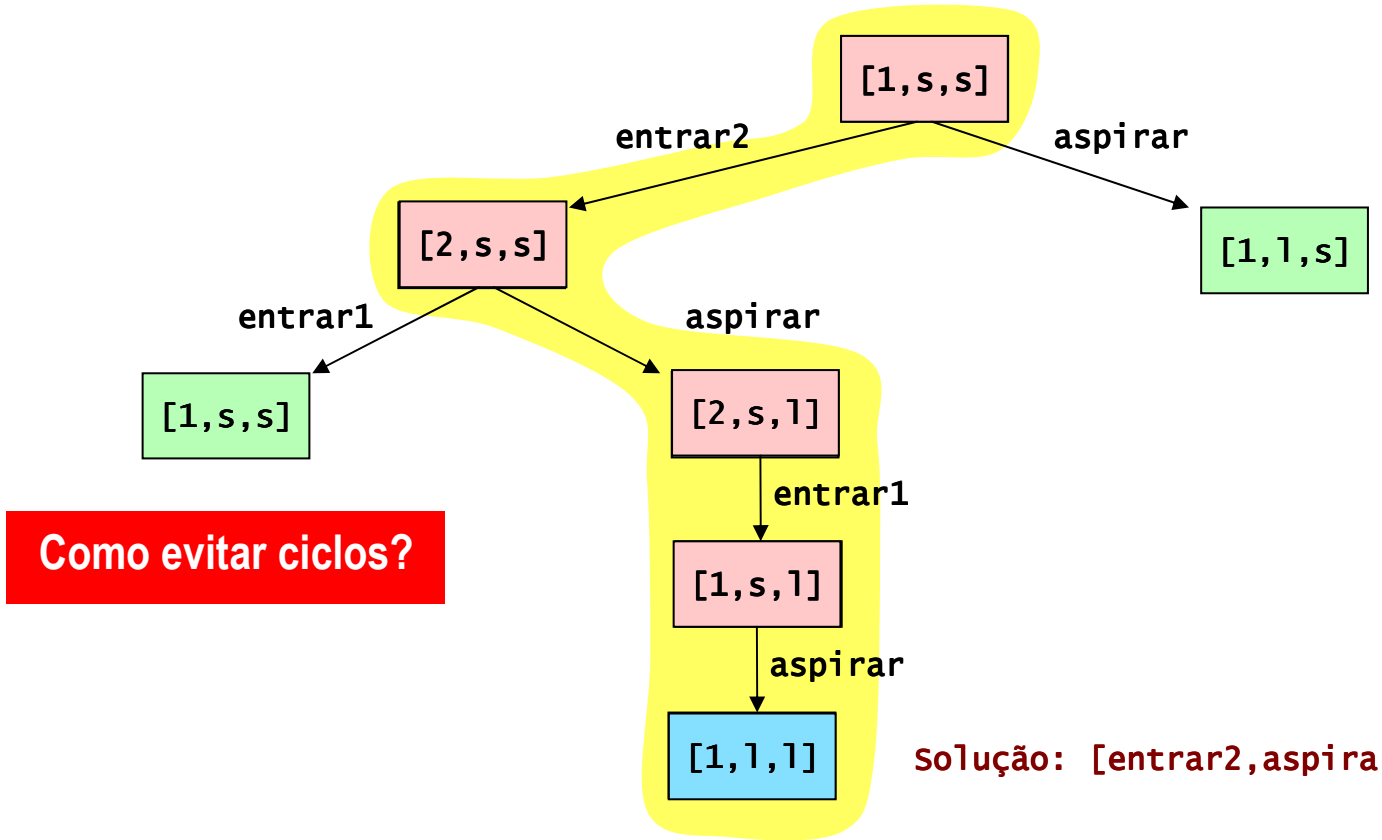
busca em largura

busca em profundidade

# Busca aleatória

## Na busca aleatória

a escolha das folhas a serem expandidas é feita aleatoriamente



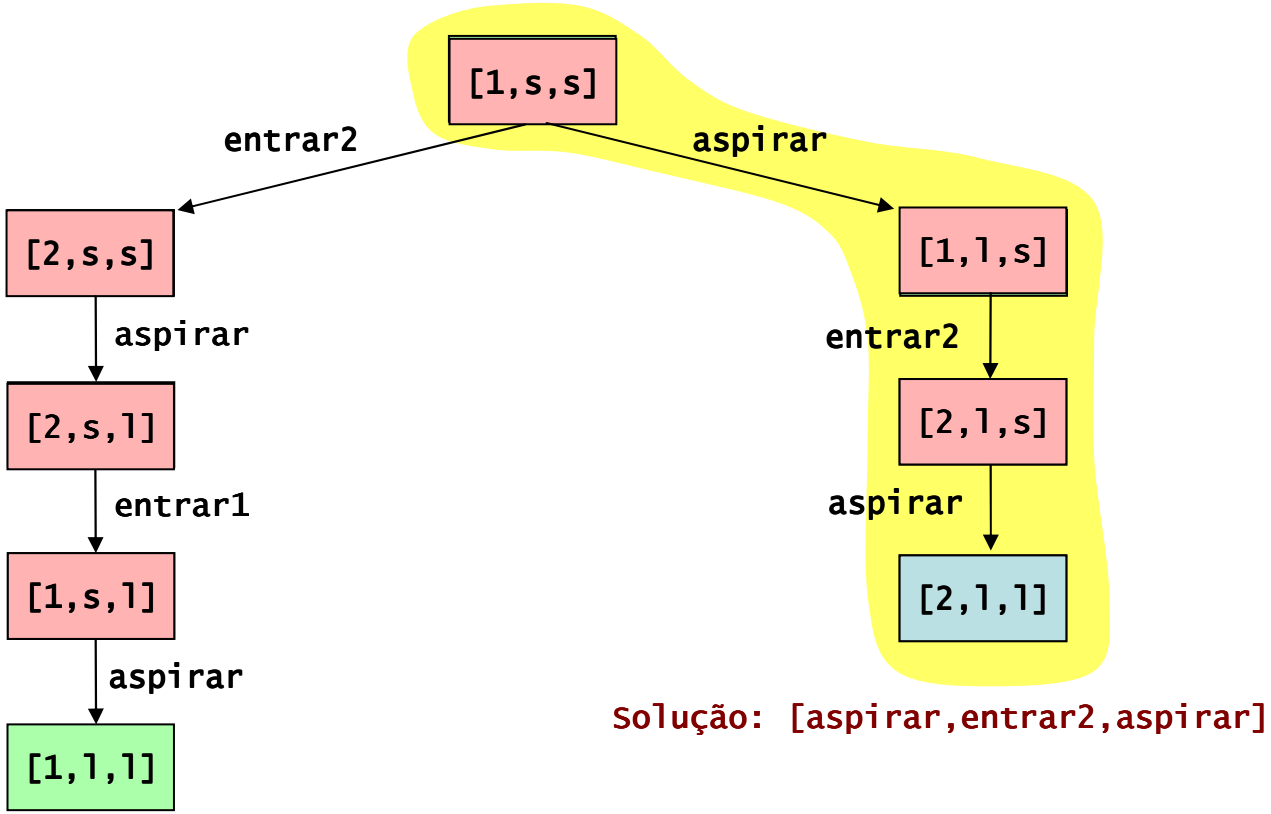

**Como evitar ciclos?**

Solução: [entrar2,aspirar,entrar1,aspirar]

# Busca em largura

## Na busca em largura

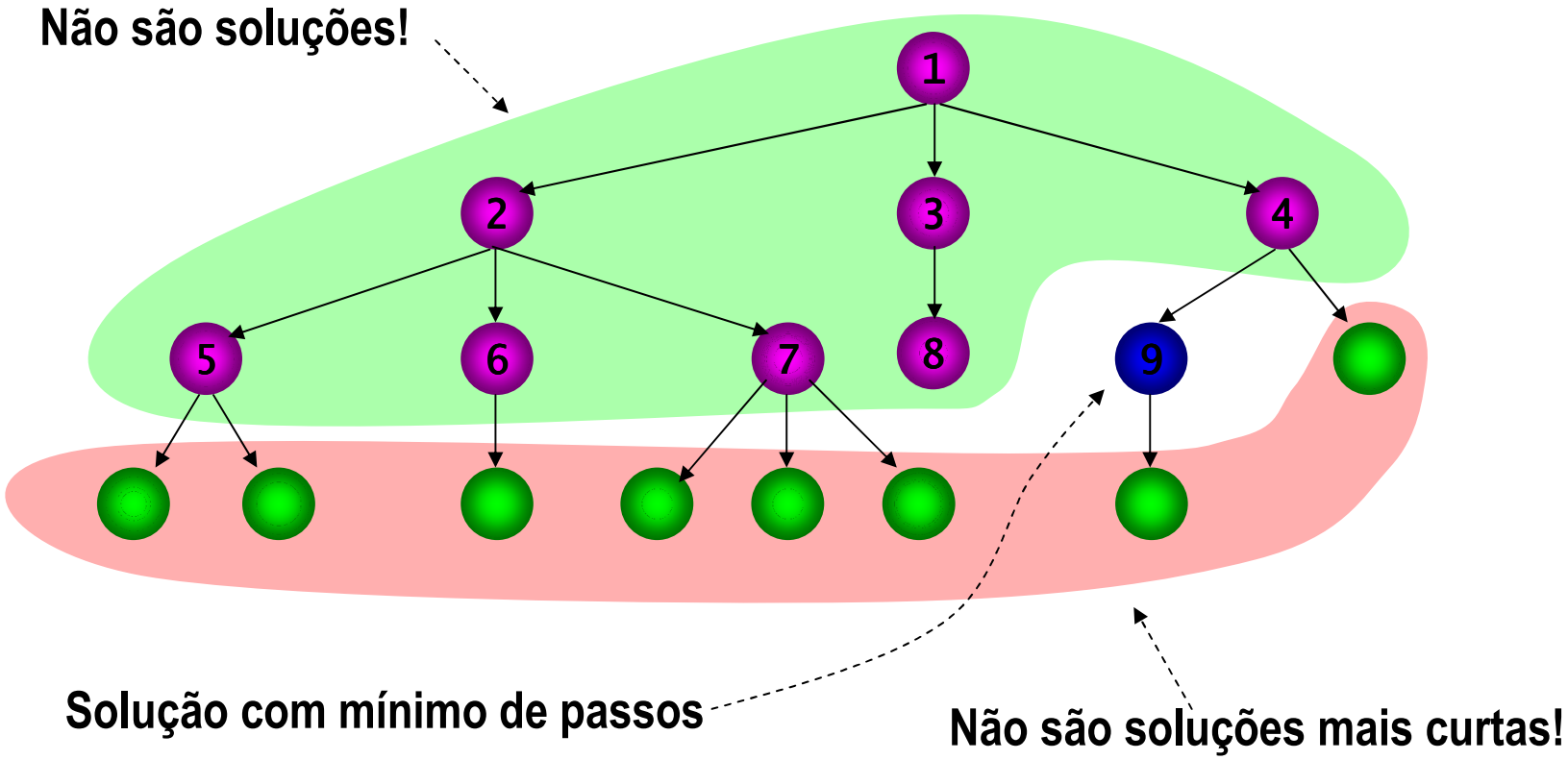
a escolha das folhas a serem expandidas é feita de acordo a política FIFO (*First-In/First-Out*)



# Busca em largura

## A busca em largura

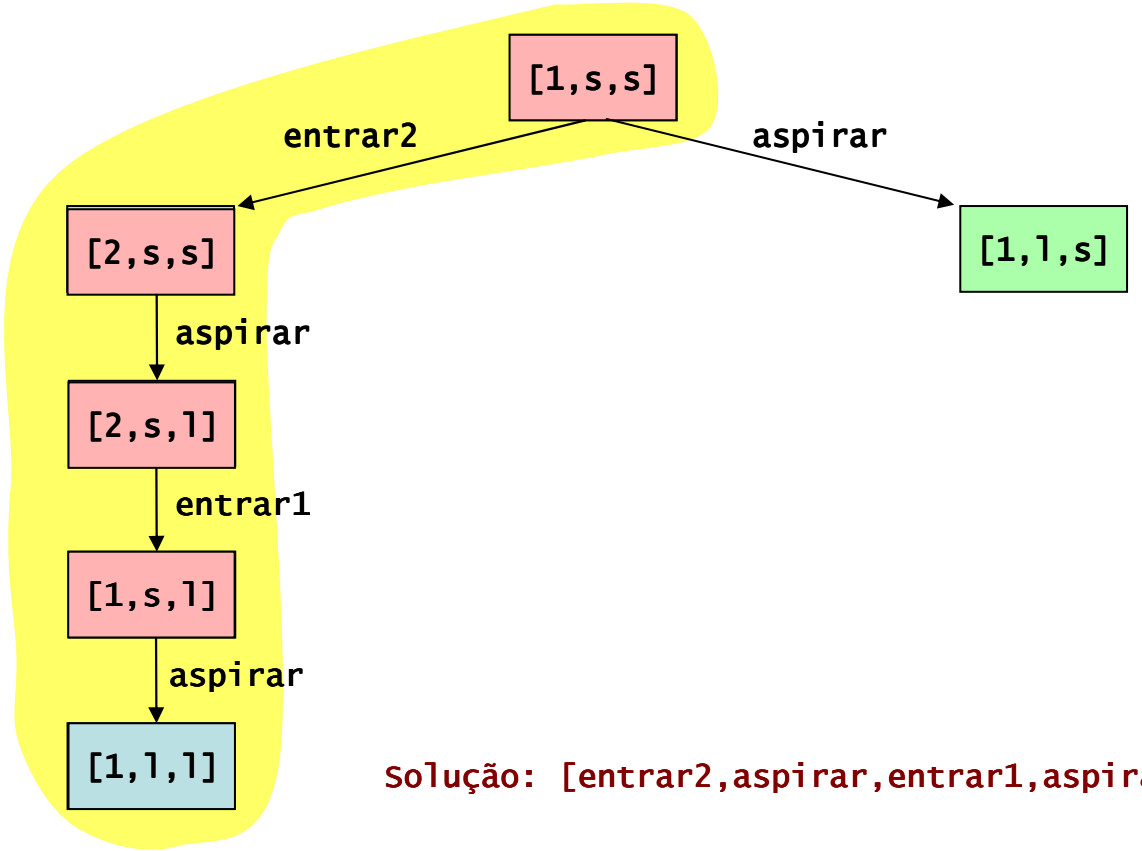
garante encontrar uma solução com número de passos mínimo (i.e., uma solução mais curta)



# Busca em profundidade

## Na busca em profundidade

a escolha das folhas a serem expandidas é feita de acordo a política LIFO (*Last-In/First-Out*)

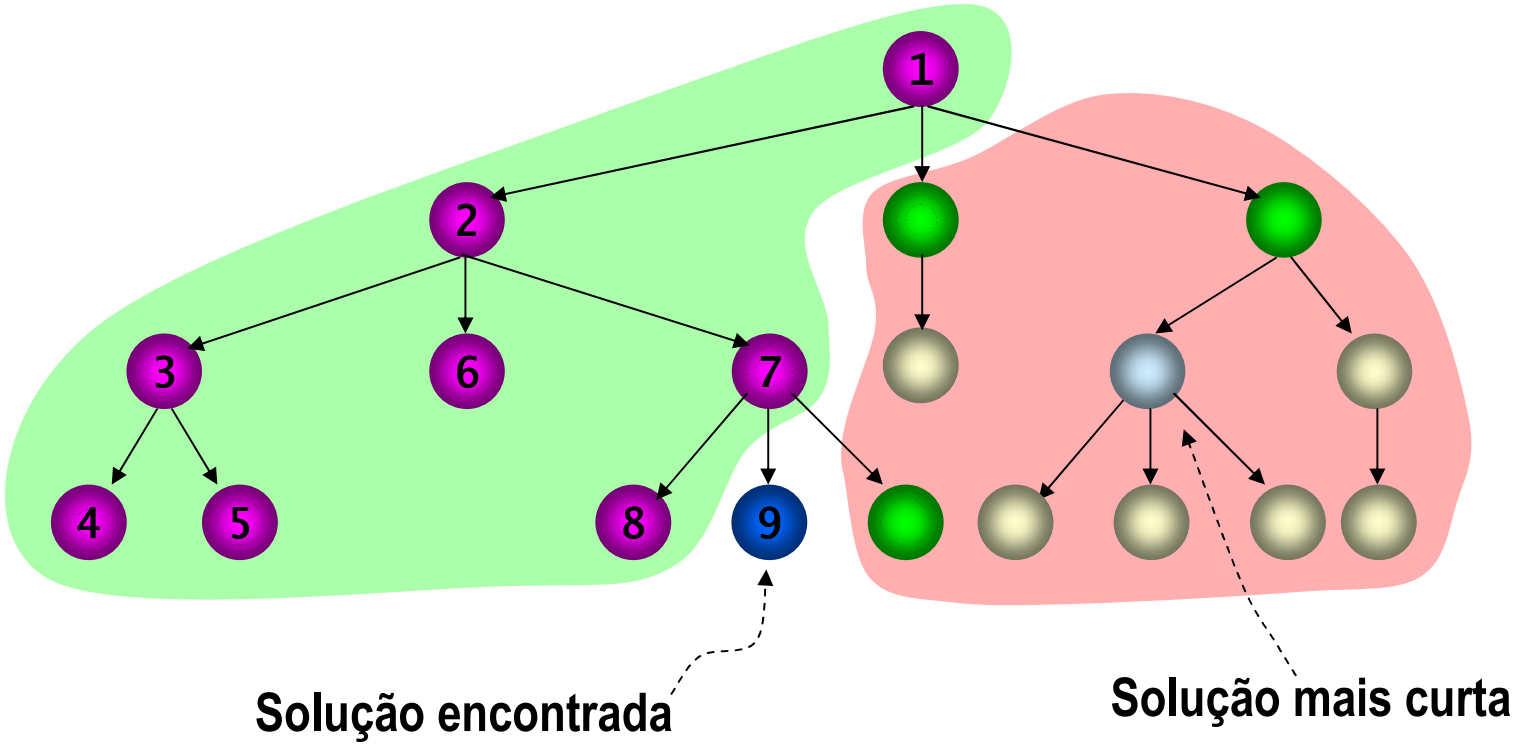



Solução: [entrar2,aspirar,entrar1,aspirar]

# Busca em profundidade

## A busca em profundidade

pode ser mais eficiente em alguns casos, mas não garante encontrar uma solução mais curta



# Busca informada

busca pelo menor custo

busca pela melhor estimativa

busca ótima (A*)

# Custo de ação

- Em muitos problemas de busca, as ações podem ter custos que refletem a dificuldade que o agente tem em executá-las.
- Este custo pode representar uma quantia de dinheiro, tempo, distância, etc.
- Para que o custo de uma ação possa ser levado em conta durante a busca, precisamos adicioná-lo como um novo argumento na especificação das ações.

## Representação de ações com custos

ação($\alpha$,$e_1$,$e_2$,g) :- $\beta$.

onde:

- $\alpha$ é o identificador da ação
- $e_1$ é o estado em que o mundo se encontra antes da execução da ação $\alpha$
- $e_2$ é o estado em que o mundo fica após a execução da ação $\alpha$
- g é o custo de executar a ação $\alpha$
- $\beta$ é uma especificação das precondições e/ou efeitos da ação $\alpha$

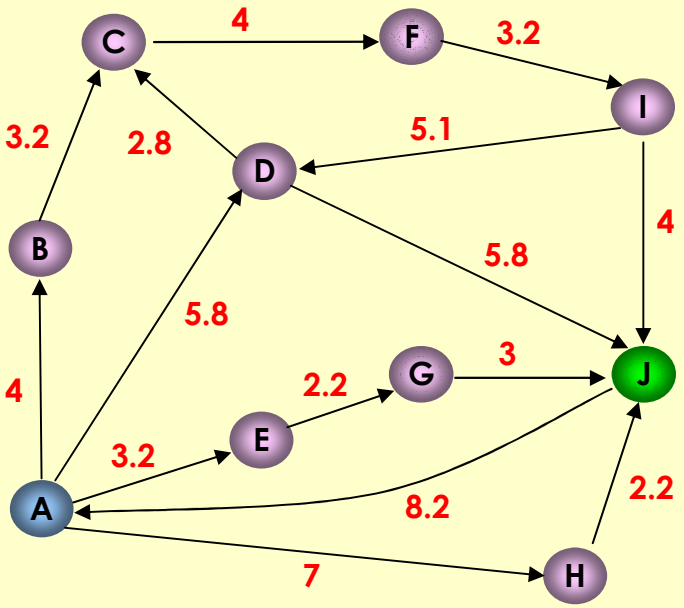
# Custo de ação

No **problema das rotas**, as ações do agente permitem seu deslocamento de um ponto ao outro do mapa e seus custos representam as distâncias percorridas.

## Exemplo 1. Problema das rotas (versão 1)

```
inicial(a).
meta(j).
ação(vai(P,Q),P,Q,D) :- via(P,Q,D).
via(a,b,4.0). via(a,d,5.8).
via(a,e,3.2). via(a,h,7.0).
via(b,c,3.2). via(c,f,4.0).
via(d,c,2.8). via(d,j,5.8).
via(e,g,2.2). via(f,i,3.2).
via(g,j,3.0). via(h,j,2.2).
via(i,d,5.1). via(i,j,4.0).
via(j,a,8.2).
```

## Mapa de vias



# Custo de caminho

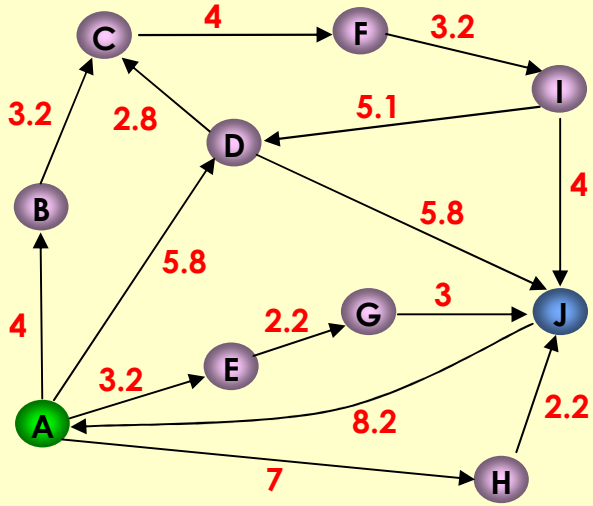
## Custo de caminho

quando as ações têm custo, o custo de um caminho $[a_1, a_2, ..., a_n]$ é

$$\sum_{i=1}^{n} g(a_i)$$

onde $g(a_i)$ é o custo da ação $a_i$.

## Mapa de vias




## Exemplo 2. Custo de caminho

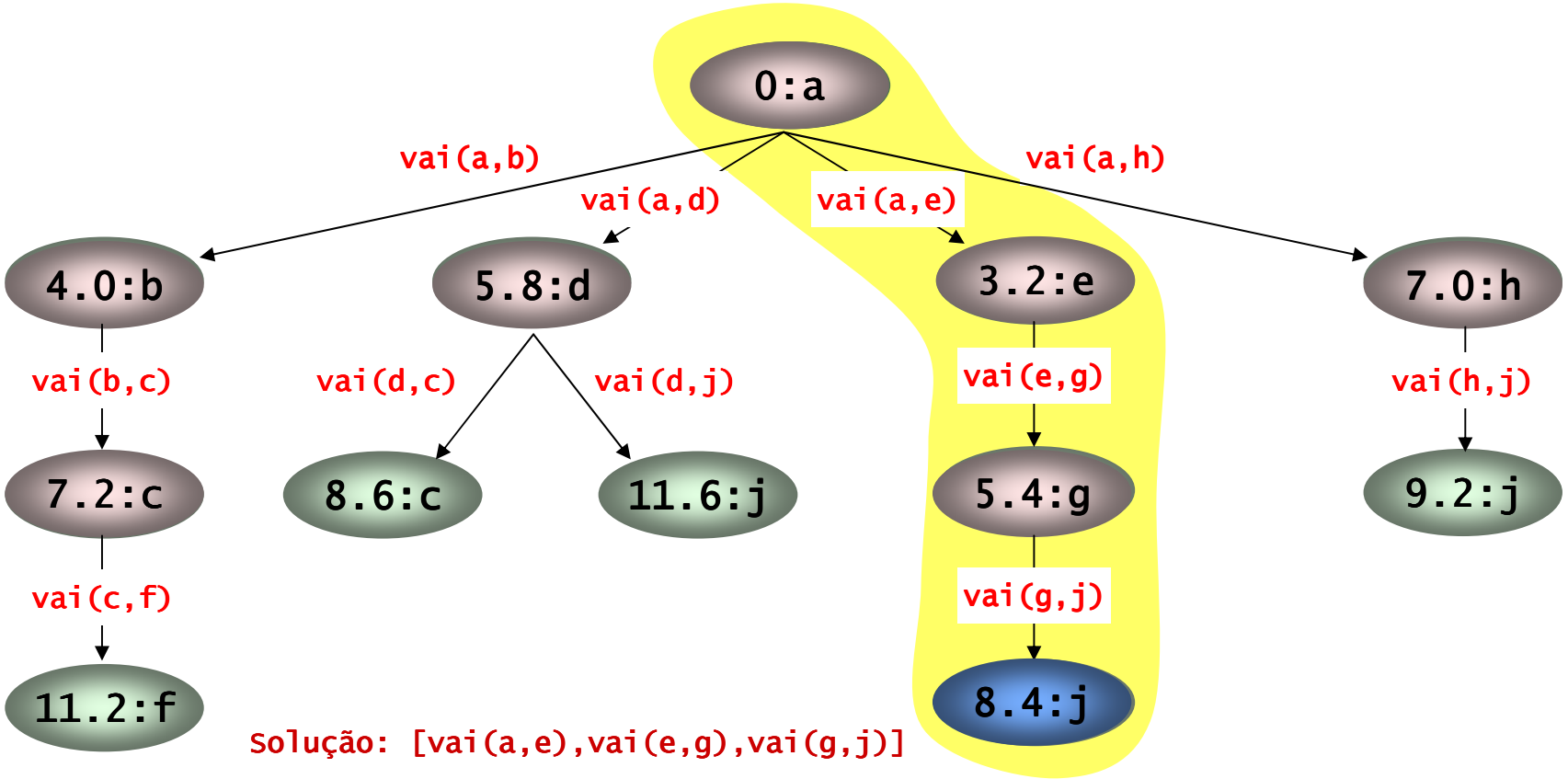
O custo do caminho `[vai(a,h),vai(h,j)]` é `7 + 2.2 = 9.2`

**Numa árvore de busca, o custo de uma folha é o custo do caminho da raiz até ela.**

# Busca pelo menor custo

## Na busca pelo menor custo

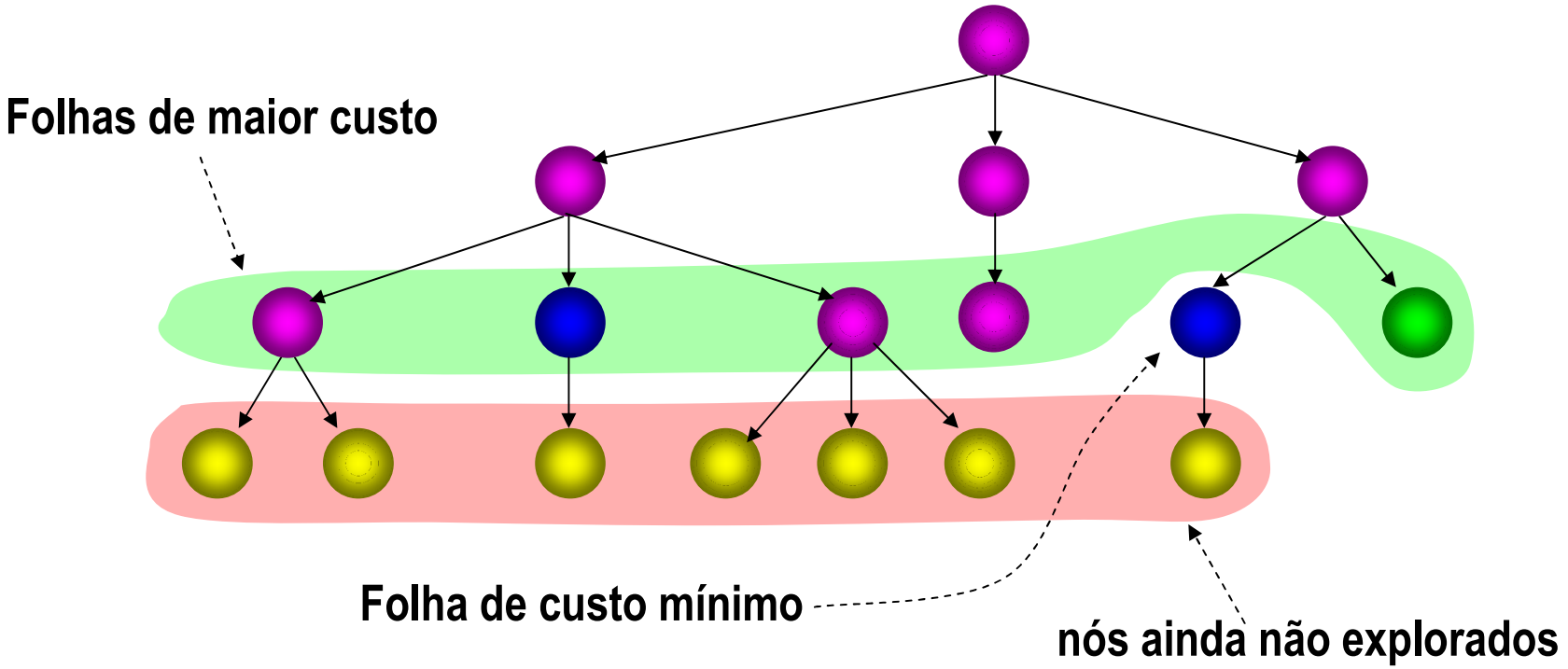
em cada instante, expande-se uma das folhas que tenha custo mínimo



# Busca pelo menor custo

## A busca pelo menor custo

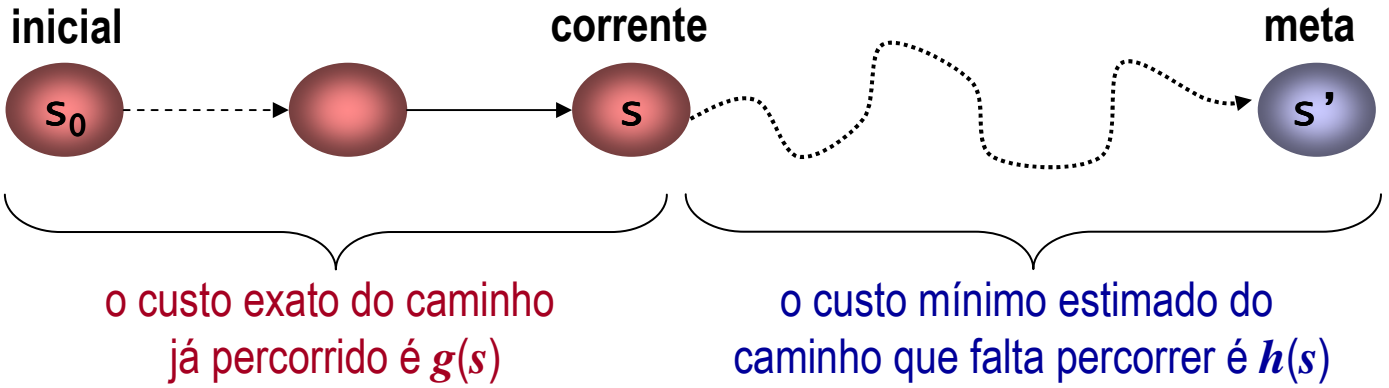
- garante encontrar uma solução de custo mínimo (não necessariamente com menos passos)
- se as ações têm custo uniforme (todos iguais), esta busca funciona como a busca em largura



# Função heurística

## Função heurística

é uma função $h$ que estima o custo mínimo de um caminho (desconhecido) que leva de um determinado estado corrente $s$ a um estado meta $s'$.



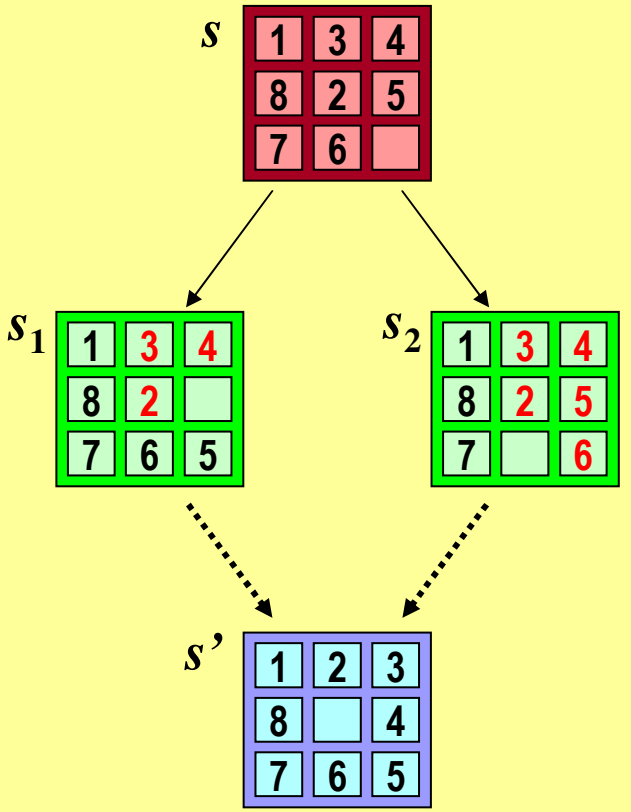

- Seja $h^*$ uma função que calcula o custo mínimo exato de um caminho que leva de um estado corrente $s$ a um estado meta $s'$.
- Uma função heurística $h$ é **admissível** se, para todo estado $s$, temos $h(s) \leq h^*(s)$.
- Particularmente, se $s$ é um estado meta, devemos ter $h(s) = 0$.

# Função heurística (distância de Manhattan)

## Exemplo 4. Função heurística admissível

- O problema do **quebra-cabeça de oito peças** consiste em encontrar uma seqüência de movimentos que transforme um estado inicial $s$ num estado meta $s'$.
- Uma heurística admissível bastante conhecida para este problema é a chamada **distância de Manhattan**.
- Esta heurística estima o custo mínimo do caminho a ser percorrido de um estado corrente $s$ a um estado meta $s'$, calculando a soma das distâncias entre as posições que as peças ocupam em $s$ e em $s'$.
  - $h(s_1) = d_1(2) + d_1(3) + d_1(4) = 1+1+1 = 3$
  - $h(s_2) = d_2(2) + d_2(3) + d_2(4) + d_2(5) = 1+1+1+1+1 = 5$
- Esta heurística é admissível porque as peças não podem se mover na diagonal.

## Sucessor mais promissor?



# Função heurística (distância em linha reta)

## Exemplo 5. Problema das rotas (versão 2)

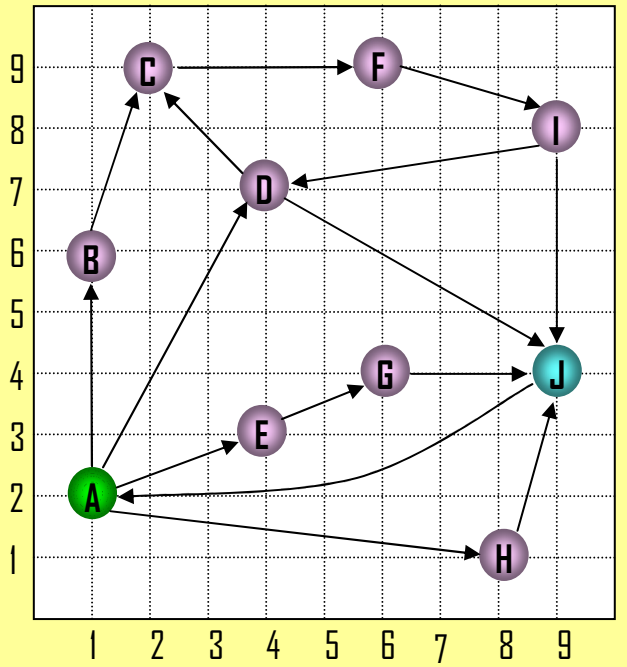
```
inicial(a).
meta(j).
ação(vai(P,Q),P,Q,D) :-
   via(P,Q), dist(P,Q,D).
h(S,H) :- meta(M), dist(S,M,H).

via(a,b). via(a,d). via(a,e). via(a,h).
via(b,c). via(c,f). via(d,c). via(d,j).
via(e,g). via(f,i). via(g,j). via(h,j).
via(i,d). via(i,j). via(j,a).

dist(P,Q,D) :-
   pos(P,Xp,Yp), pos(Q,Xq,Yq),
   D is sqrt((Xp-Xq)^2 + (Yp-Yq)^2).

pos(a,1,2). pos(b,1,6). pos(c,2,9).
pos(d,4,7). pos(e,4,3). pos(f,6,9).
pos(g,6,4). pos(h,8,2). pos(i,9,8).
pos(j,9,4).
```

## Mapa com coordenadas



# Busca heurística

## Um método de busca heurístico

é um método que usa a estimativa fornecida por uma função heurística ***h*** para escolher a folha mais promissora (isto é, mais próxima a um estado meta) a ser expandida em cada instante da busca.

De acordo com esta definição, embora o método de **busca pelo menor custo** seja um método de busca informada (pois utiliza informação sobre o custo das ações para escolher a folha mais promissora a ser expandida), ele **não é um método heurístico** (pois a informação que ele tem não serve para orientar a busca em direção à meta).

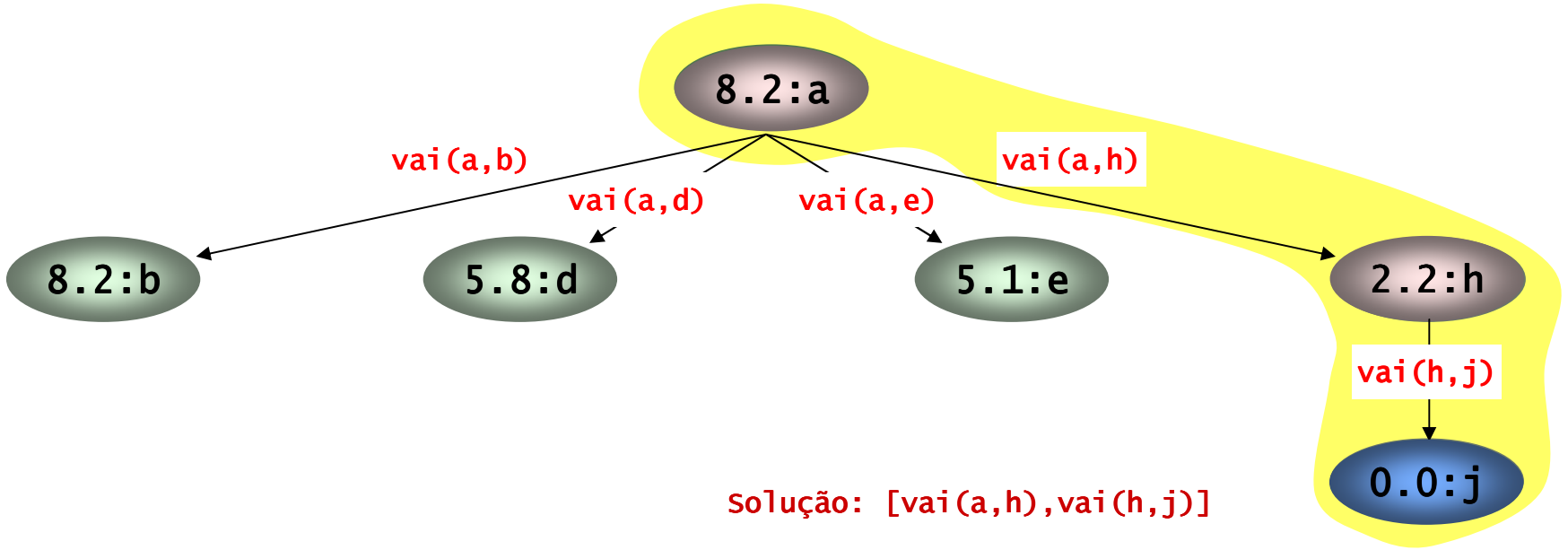
A seguir, apresentamos dois métodos de **busca heurística**:

- busca pela melhor estimativa
- busca ótima (A*)

# Busca pela melhor estimativa

## Na busca pela melhor estimativa

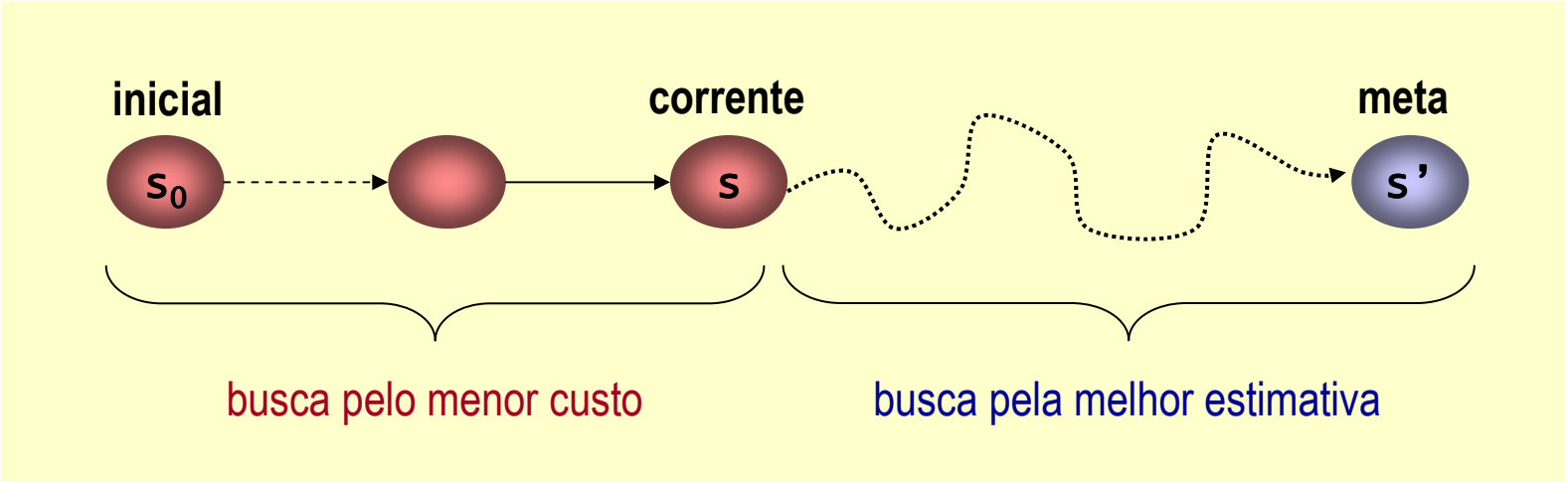
em cada instante, expande-se uma das folhas que tenha a melhor estimativa (heurística)



# Busca pela melhor estimativa

- A **busca pelo menor custo** minimiza o comprimento do caminho já percorrido, a partir do estado inicial, até um estado corrente.

  **Conseqüência**: encontra uma solução de custo mínimo.



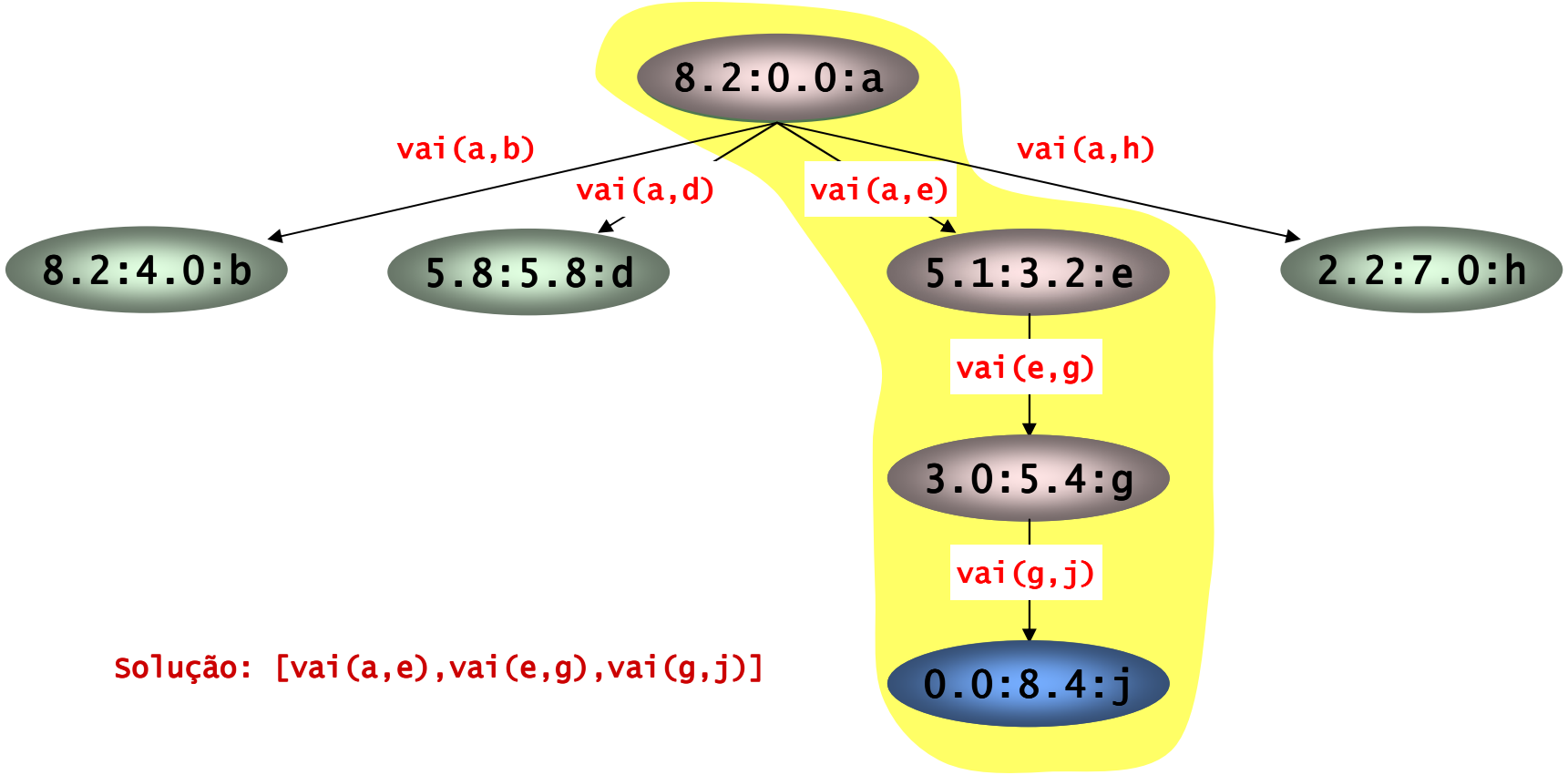

- A **busca pela melhor estimativa** minimiza o comprimento estimado do caminho que ainda falta percorrer, do estado corrente, até um estado meta.

  **Conseqüência**: encontra uma solução rapidamente.

# Busca ótima (ou A*)

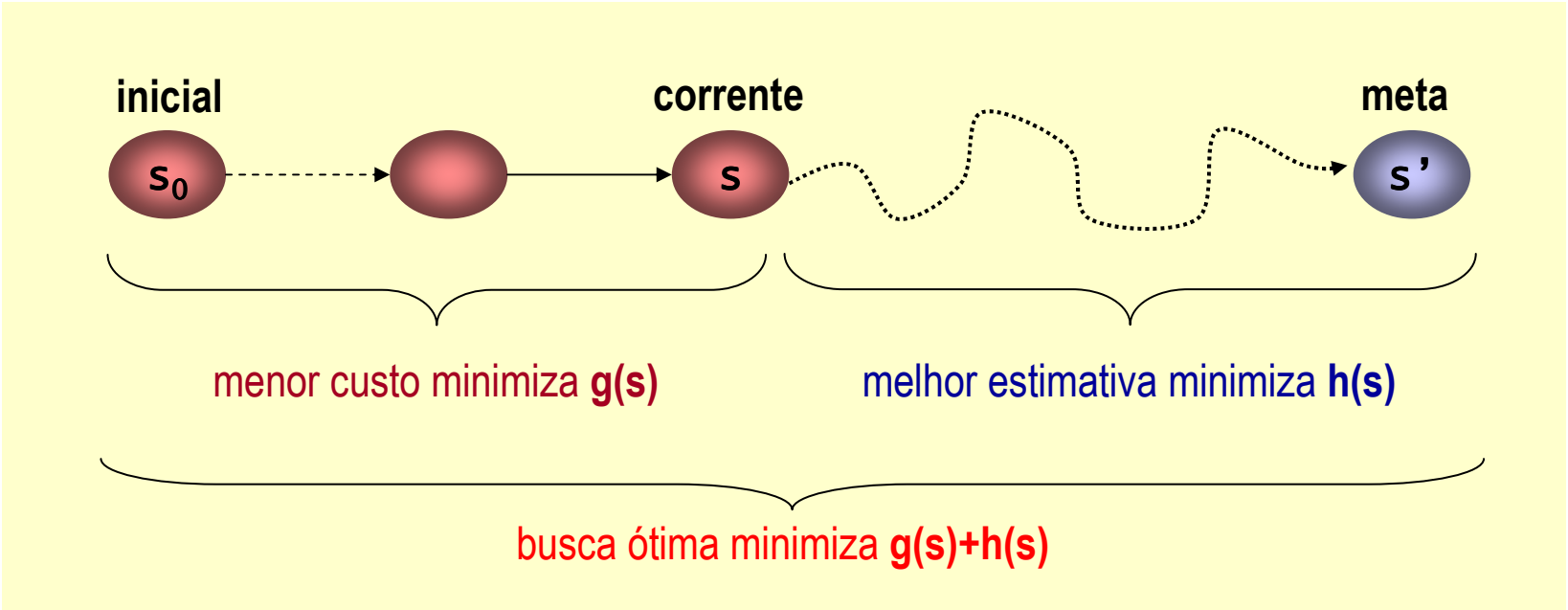
## Na busca ótima (ou A*)

em cada instante, expande-se uma das folhas que tenha a menor soma de custo e heurística.



# Busca ótima (ou A*)

- A **busca pelo menor custo** encontra uma solução de custo mínimo.
- A **busca pela melhor estimativa** encontra uma solução rapidamente.




- A **busca ótima** (ou **A***) encontra uma solução de custo mínimo rapidamente.

# Implementação em Prolog

## Exemplo 8. Algoritmo de busca generalizado

```
busca(T) :-
   inicial(E),
   busca(T,[_:_:0:E:[]],[],P:G),
   tipo(T,N),
   format('~nTipo.: ~w',[N]),
   format('~nPlano: ~w',[P]),
   format('~nCusto: ~w~n~n',[G]).

busca(_,[_:_:G:E:C|_],_,P:G) :-
   meta(E), !,
   reverse(C,P).

busca(T,[_:_:G:E:C|F],V,P) :-
   sucessores(T,G:E:C,V,S),
   insere(T,S,F,NF),
   union([E],V,NV),
   busca(T,NF,NV,P).
```

# Implementação em Prolog

## Exemplo 8. Algoritmo de busca generalizado (continuação)

```
sucessores(T,G1:E:C,V,R) :-
   findall(F:H:G:S:[A|C],
          (ação(A,E,S,G2),
           not(member(S,V)),
           h(S,H), G is G1+G2,
           (T=4 -> F is G
           ;T=5 -> F is H
           ;T=6 -> F is G+H
           ;       F is 0)),R).

insere(1,S,F,NF) :- append(S,F,R),
                    length(R,L), embaralha(L,R,NF), !.
insere(2,S,F,NF) :- append(F,S,NF), !.
insere(3,S,F,NF) :- append(S,F,NF), !.
insere(_,S,F,NF) :- append(S,F,R), sort(R,NF), !.
```

# Implementação em Prolog

## Exemplo 8. Algoritmo de busca generalizado (continuação)

```
embaralha(0,F,F) :- !.
embaralha(L,F,[X|NF]) :-
    N is random(L),
    nth0(N,F,X), delete(F,X,R),
    M is L-1,
    embaralha(M,R,NF), !.

tipo(1,aleatória).
tipo(2,largura).
tipo(3,profundidade).
tipo(4,menor_custo).
tipo(5,melhor_estimativa).
tipo(6,ótima).
```

# Fim