Algoritmos de Busca na Resolução de Problemas

Prof. Ademir A. Constantino

DIN/UEM

Busca na Resolução de Problemas

- Objetivo:
  - dado um ***estado inicial*** de um problema, encontrar o conjunto de operações que leve à solução (***estados finais***).

- O *espaço de estados* é representado por um grafo, onde:
  - *vértice* (ou nodo) é um estado;
  - *aresta* é uma possibilidade de transição.

Algoritmo de Busca

- Algoritmo de Busca:
  - um procedimento de exploração do espaço de estados.

- Operador de transição.
  - a transição de um estado (vértice) para outro é definida por um ***operador***.
  - ***operador*** = gerador de sucessores.

Busca em largura (breadth-first)

Algoritmo Geral:

nodos <-- CRIAR-FILA(*estado-inicial*)

loop

se *nodos* é vazio **retorna** falha
nodo <-- TIRAR-PRIMEIRO(*nodos*)
se TESTE-SUCESSO(*nodo*) tem sucesso

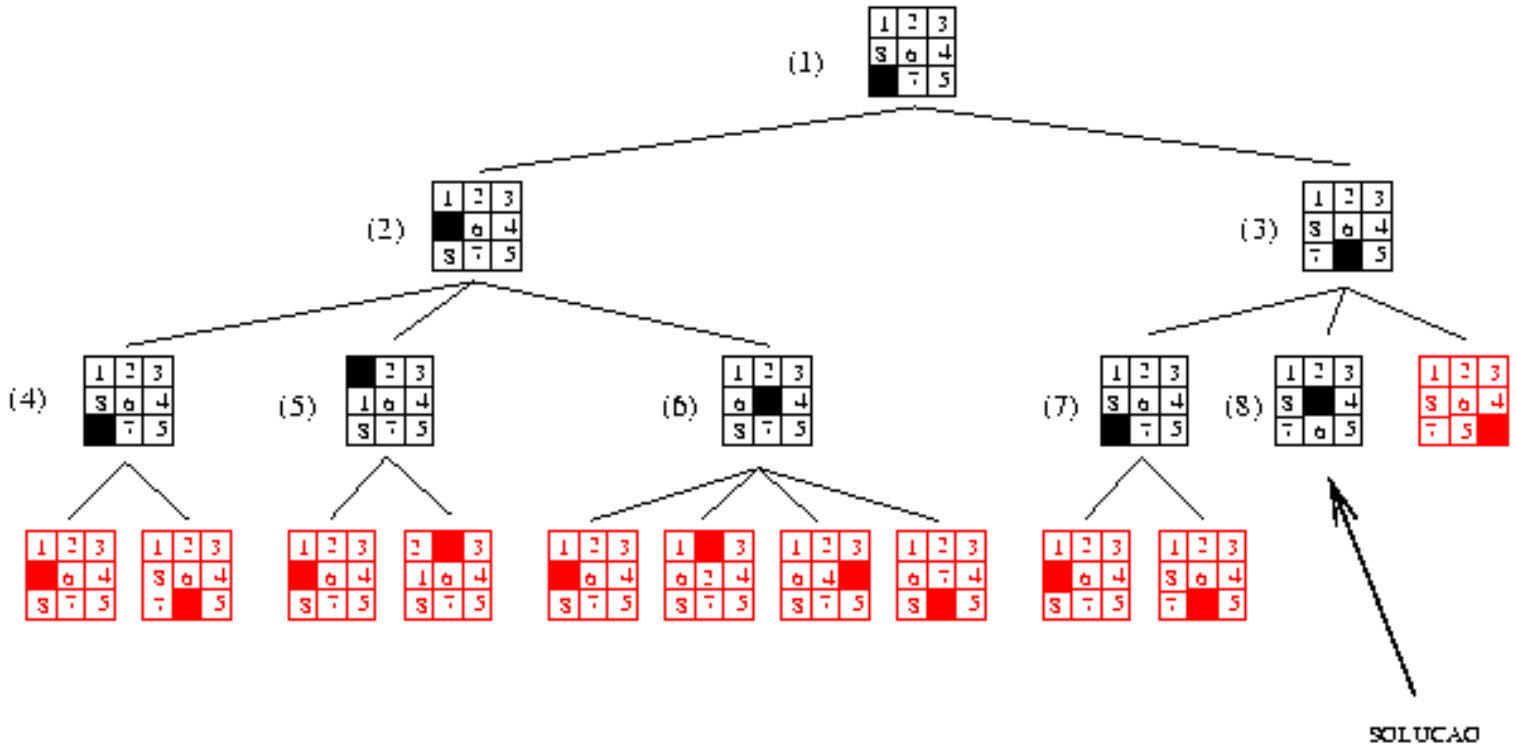
retorna *nodo*

novos-nodos <-- EXPANDIR(*nodo*)
nodos <-- ACRESCENTAR-NO-FIM(*nodos*,*novos-nodos*)

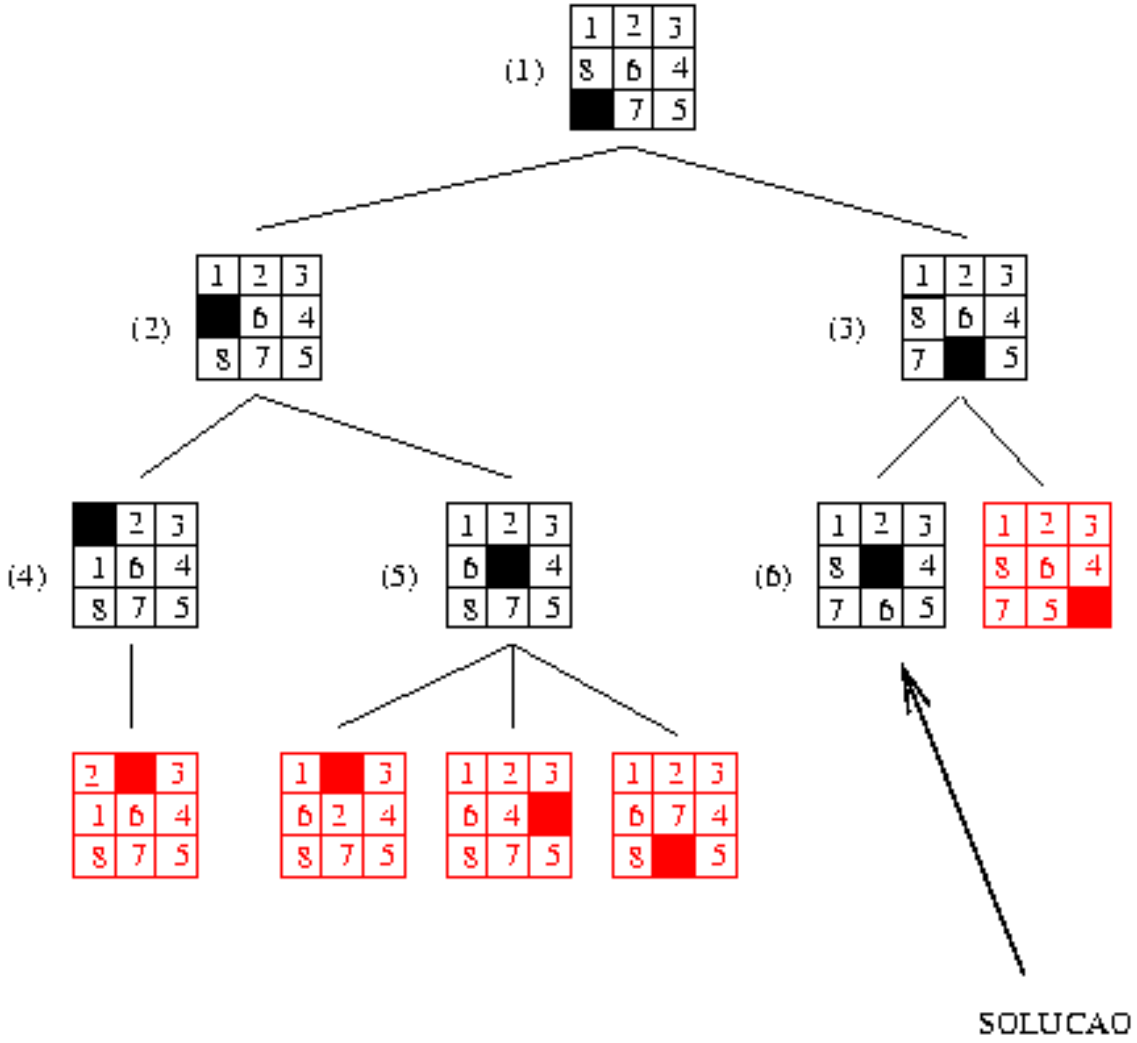
fim

Nota: a função EXPANDIR retorna uma lista de *nodos* que representam os estados sucessores (adjacentes) do estado atual *nodo*.

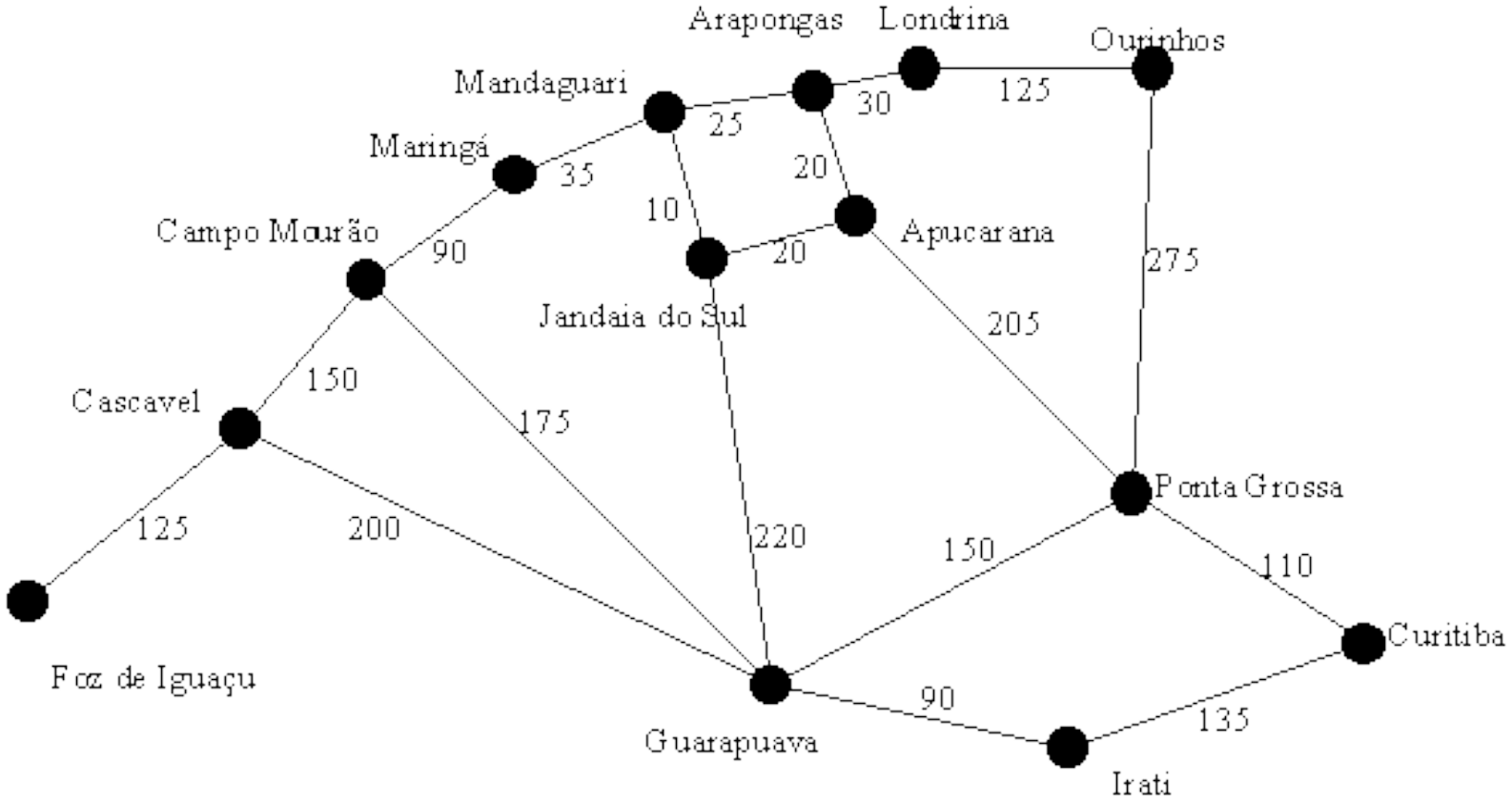
Busca em largura - Aplicação ao tabuleiro de 8 peças.



Redução de estados repetidos

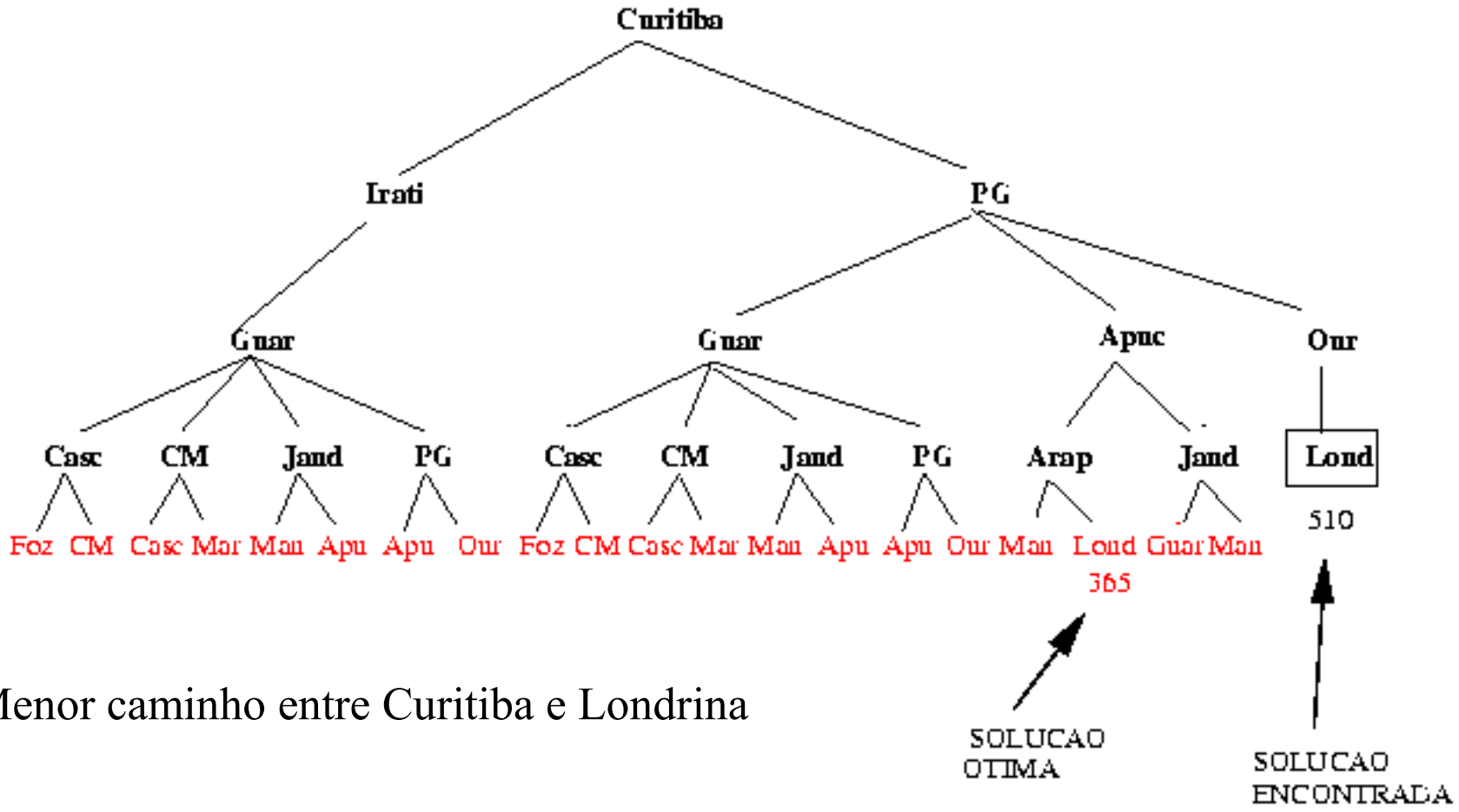


Busca em Largura - Aplicação na obtenção no menor caminho



Árvore de Busca em Largura

(solução X Solução Ótima)



Menor caminho entre Curitiba e Londrina

Busca em profundidade (depth-first)

nodos <-- CRIAR-FILA(*estado-inicial*)

loop

se *nodos* é vazio **retorna** falha
nodo <-- TIRAR-PRIMEIRO(*nodos*)
se TESTE-SUCESSO(*nodo*) tem sucesso

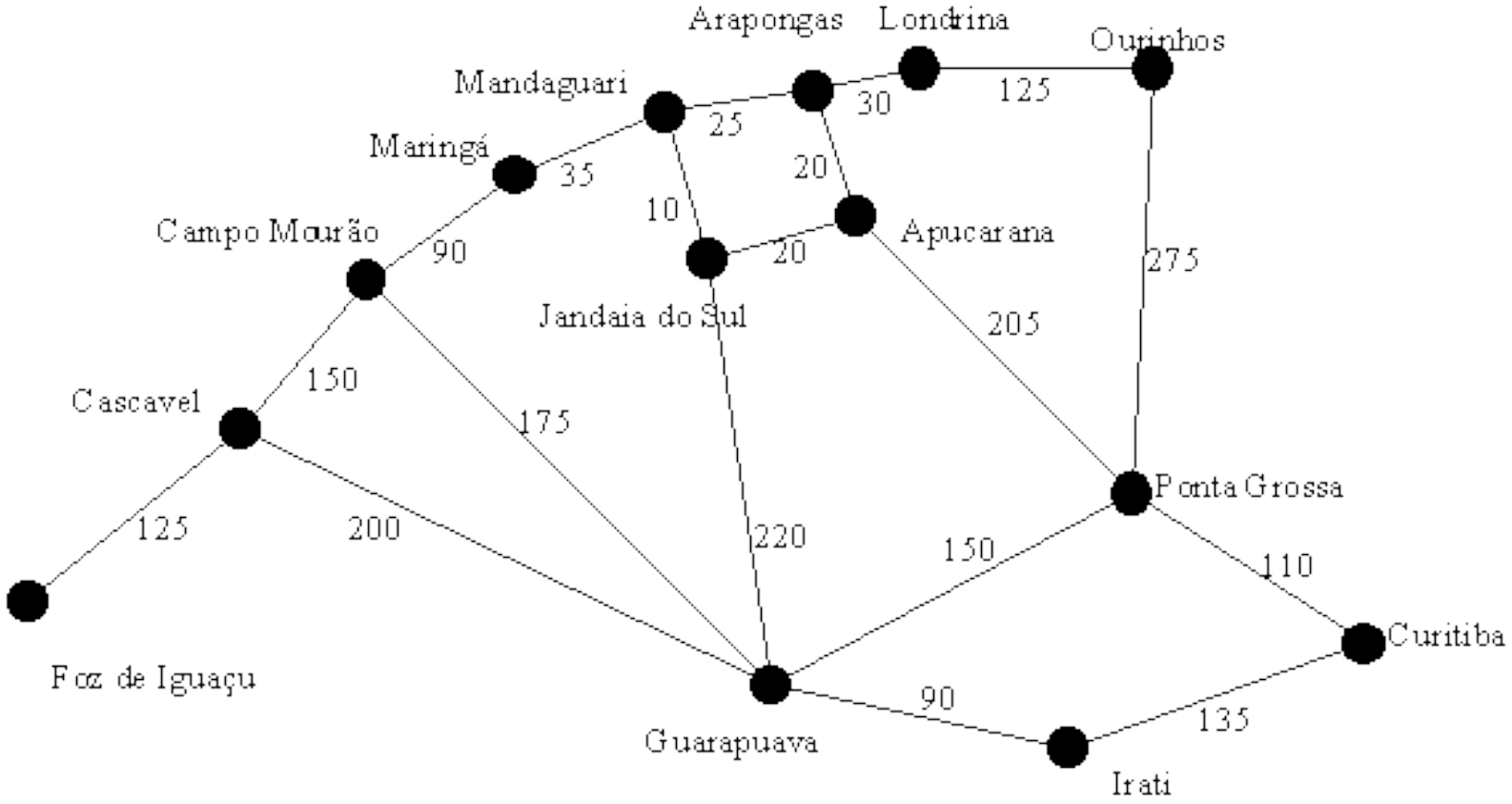
retorna *nodo*

novos-nodos <-- EXPANDIR(*nodo*)
nodos <-- ACRESCENTAR-NO-INICIO(*nodos*,*novos-nodos*)

fim

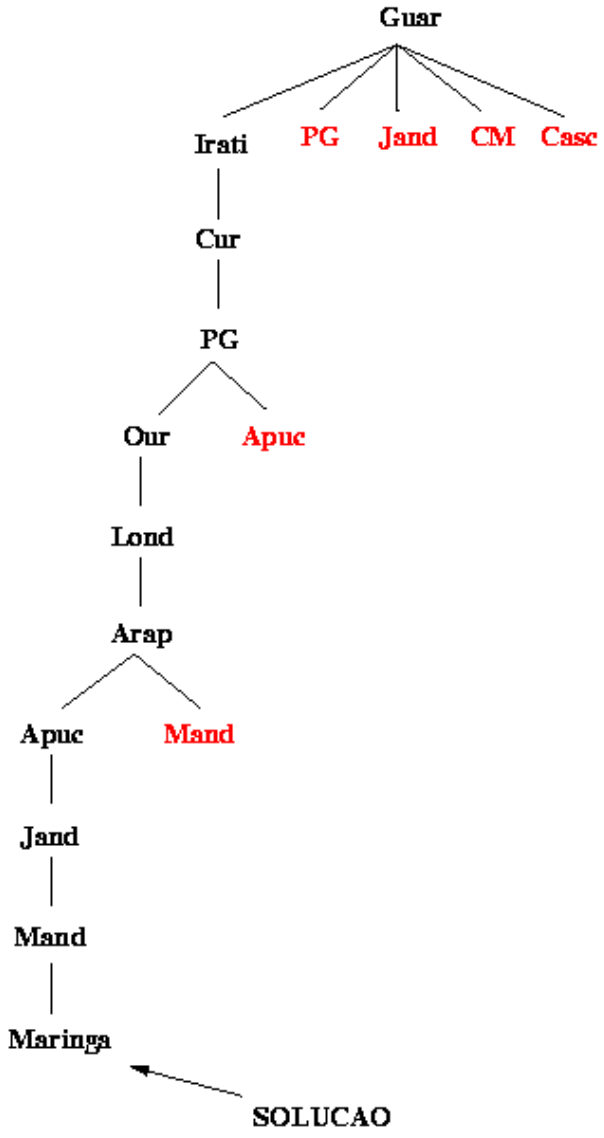
Busca em Profundidade

- Aplicação na obtenção do menor caminho



Árvore de Busca em Profundidade

Menor caminho entre
Guarapuava e Maringá



Busca em largura a custo uniforme (Dijkstra)

nodos <-- CRIAR-FILA(*estado-inicial*)
loop

se *nodos* é vazio **retorna** falha
nodo <-- TIRAR-PRIMEIRO(*nodos*)
se TESTE-SUCESSO(*nodo*) tem sucesso

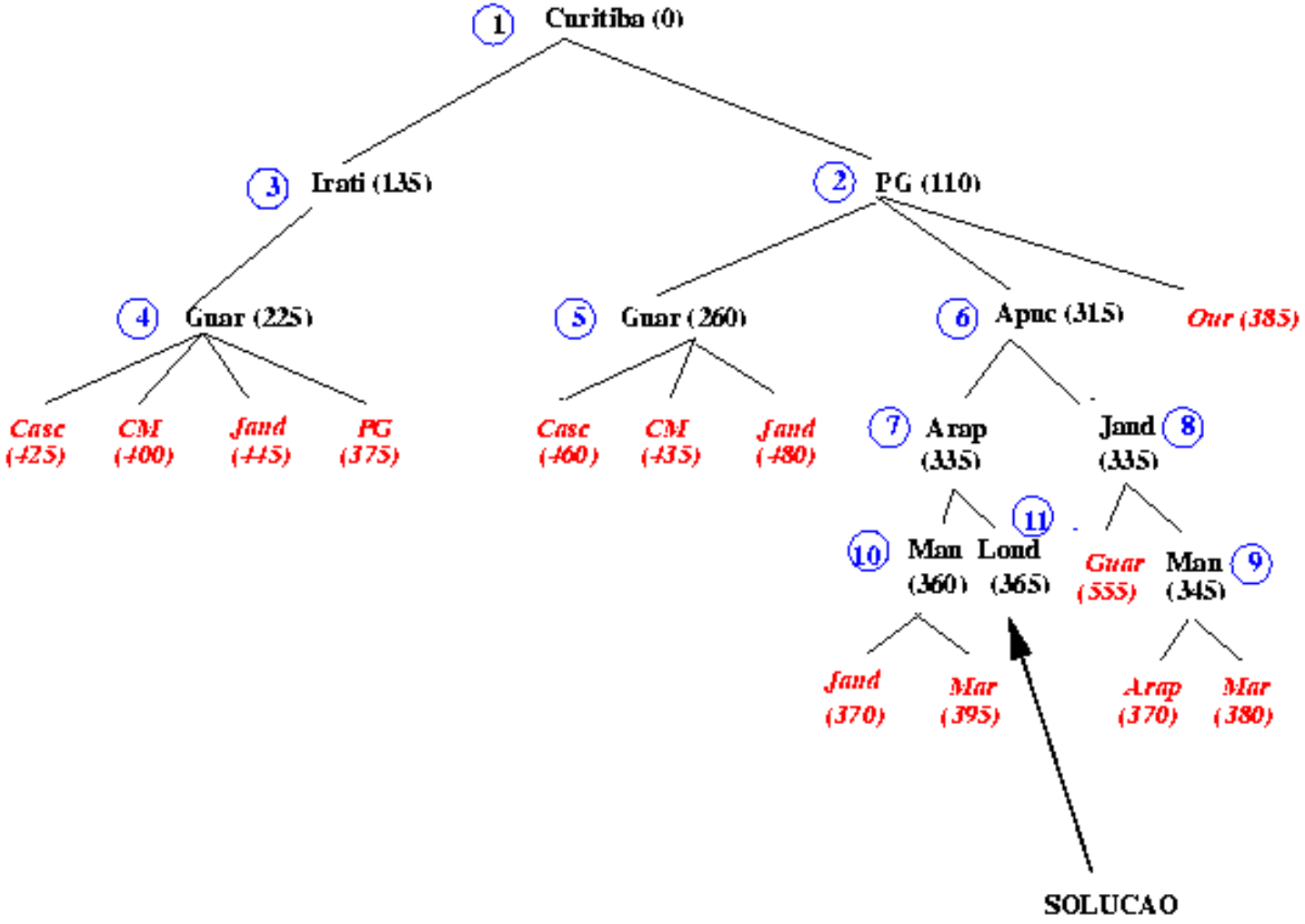
retorna *nodo*

novos-nodos <-- EXPANDIR(*nodo*)
nodos <-- ORDENAR(ACRESCENTAR-NA-FILA(*nodos*, *novos-nodos*))

fim

Nota: a cada iteração o algoritmo escolhe o nodo de menor custo

Exemplo1:


1 Curitiba (0)
3 Irati (135)
2 PG (110)
4 Guar (225)
5 Guar (260)
6 Apuc (315)
Our (385)
Casc (425)
CM (400)
Jand (445)
PG (375)
Casc (460)
CM (435)
Jand (480)
7 Arap (335)
Jand 8 (335)
11
10 Man (360)
Lond (365)
Guar (555)
Man 9 (345)
Jand (370)
Mar (395)
Arap (370)
Mar (380)
SOLUCAO

Comparação

- Os dois métodos, a busca em profundidade e a busca em largura, encontram a solução se ela existir, mas não garante o menor caminho.

Busca Heurística

Definições:

$g(n)$ = custo no menor caminho entre o estado inicial e o estado n qualquer.

$h(n)$ = custo do menor caminho entre o estado n e o estado final.

$f(n)$ = o custo do caminho do estado inicial até o estado final passando pelo estado n.

$$f(n) = g(n) + h(n)$$

$h'(n)$ - estimativa do custo do menor caminho entre o estado n e o estado final - ***função heurística***.

Algoritmo A*

Algoritmo heurístico que a cada iteração escolhe um nó n com menor valor de $f(n) = g(n) + h'(n)$.

Se $h'(n) \leq h(n)$ ==> denominamos de **Algoritmo A***

Se $h'(n) > h(n)$ ==> denominados de Algoritmo A^

Se $h'(n) \leq h(n)$, então temos uma função heurística admissível.

Se $h'(n) \leq h(n)$, existe prova que o algoritmo **A*** sempre encontra o caminho ótimo.

Algoritmo A*:

- Entrada:
  - *s*: estado inicial;
  - *t*: estado final (mas pode ser substituído por um conjunto);
- Saída:
  - Se sucesso, retorna-se o caminho da solução.

Algoritmo A*:

- Notações:
  - A : conjunto dos estados abertos;
  - F: conjunto dos estados fechados;
  - S: conjunto dos estados iniciais;
  - T: conjunto dos estados finais;
  - $P(s)$: ponteiro para o pai do nó s;
  - $\Gamma(v)$: conjunto dos estados sucessores de v.

Algoritmo A*:

a) $A \leftarrow S$, $F \leftarrow \varnothing$. Para $\forall s \in S$ faça:

- Calcule $h'(s)$ e $g(s) \leftarrow 0$; $P(s) \leftarrow 0$;

b) Se $A = \varnothing$, pare com fracasso. Senão, tome $v \in$ A tal que $f(v) = \min\{f(n), \forall n \in A,\}$ desempatando de qualquer forma, mas sempre favorecendo a $v \in T$ e faça $A \leftarrow A - \{v\}$ e $F \leftarrow F \cup \{v\}$.

c)Se $v \in T$, para com sucesso. Senão, gere $\Gamma(v)$. Se $\Gamma(v) = \varnothing$, volte a b).

d)Para cada $m \in \Gamma(v)$ faça:

calcule $g(m)$;

Se $m \notin A \cup F$ ou ($m \in A \cup F$ e $g(m) < g(m') | m' = m$))

então $A \leftarrow A \cup \{m\}$ e $F \leftarrow F - \{m\}$ (se $m \in F$)

$P(m) \leftarrow v$; e calcule $h'(m)$;

volte a b).

Algoritmo A* (versão II)

$A \leftarrow S$, $F \leftarrow \emptyset$.

Para $\forall s \in S$ faça:

Calcule $h'(s)$ e $g(s) \leftarrow 0$; $P(s) \leftarrow 0$;

Enquanto $\exists\ v \in \{v \in A \mid f(v) = \min\{f(n), \forall n \in A\}$ e $v \notin T\}$ faça:

Seja $v \in \{v \in A \mid f(v) = \min\{f(n)\}$

$A \leftarrow A - \{v\}$ e $F \leftarrow F \cup \{v\}$.

Para cada $m \in \Gamma(v)$ faça:

Calcule $g(m)$;

Se $\exists\ m' \in A \mid m' = m$ e $g(m) < g(m')$ então

$A \leftarrow A - \{m'\}$;// remova de A porque encontramos um caminho melhor

Se $\exists\ m' \in F \mid m' = m$ e $g(m) < g(m')$ então *

$F \leftarrow F - \{m'\}$; // remova de F porque encontramos um caminho melhor

Se $m \notin A \cup F$ então

$A \leftarrow A \cup \{m\}$;

$P(m) \leftarrow v$;

Calcule $h'(m)$;

Se $\exists\ v \in \{v \in A \mid f(v) = \min\{f(n), \forall n \in A\}$ e $v \in T\}$ então Sucesso, senão Fracasso.

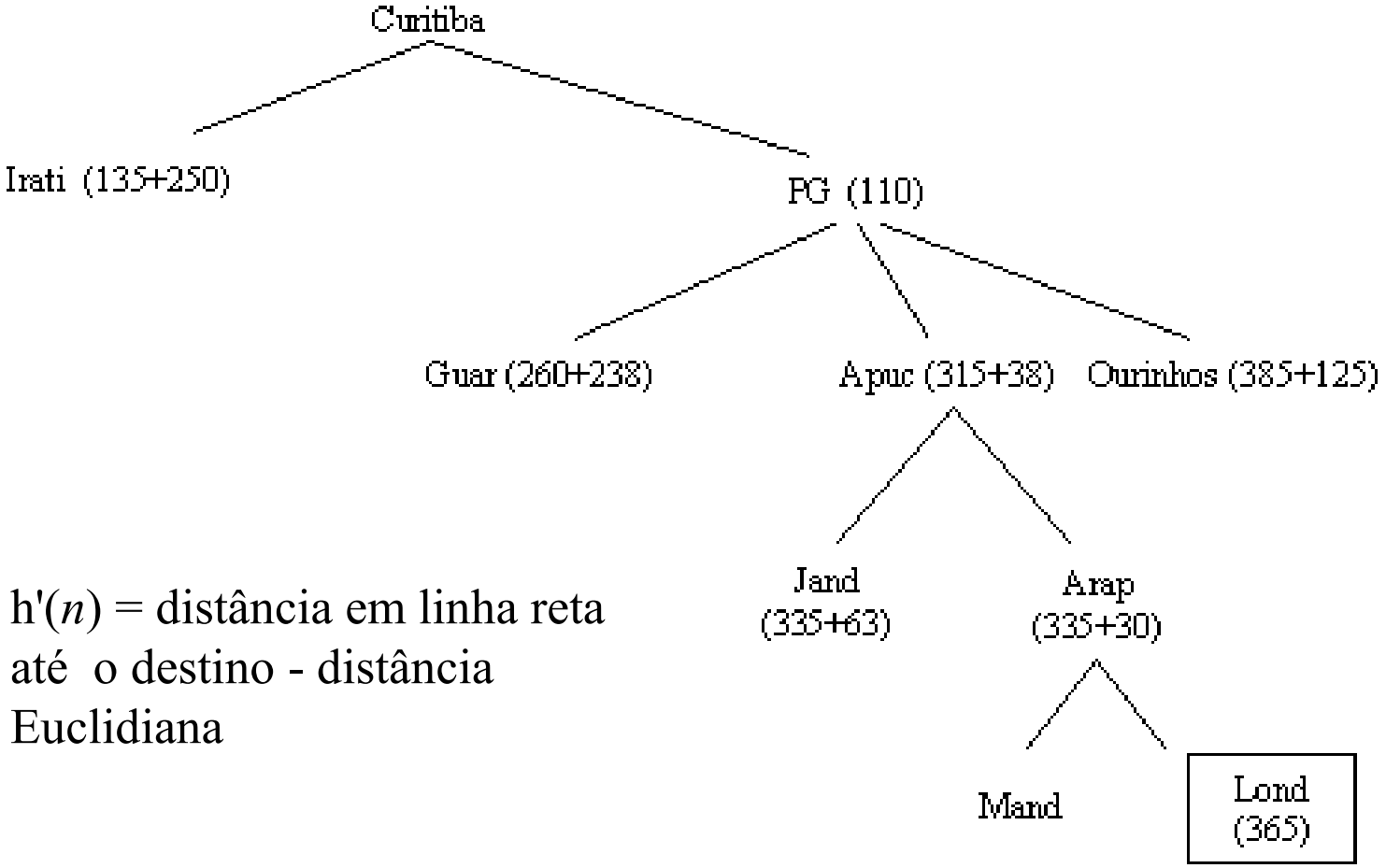
*: isto nunca irá acontecer se tivermos uma heurística admissível, mas em games é comum termos heurística não admissivel.

Heurísticas

Se $h1(n) < h2(n) \leq h(n)$, então dizemos que a função heurística h2 é **mais informada**.

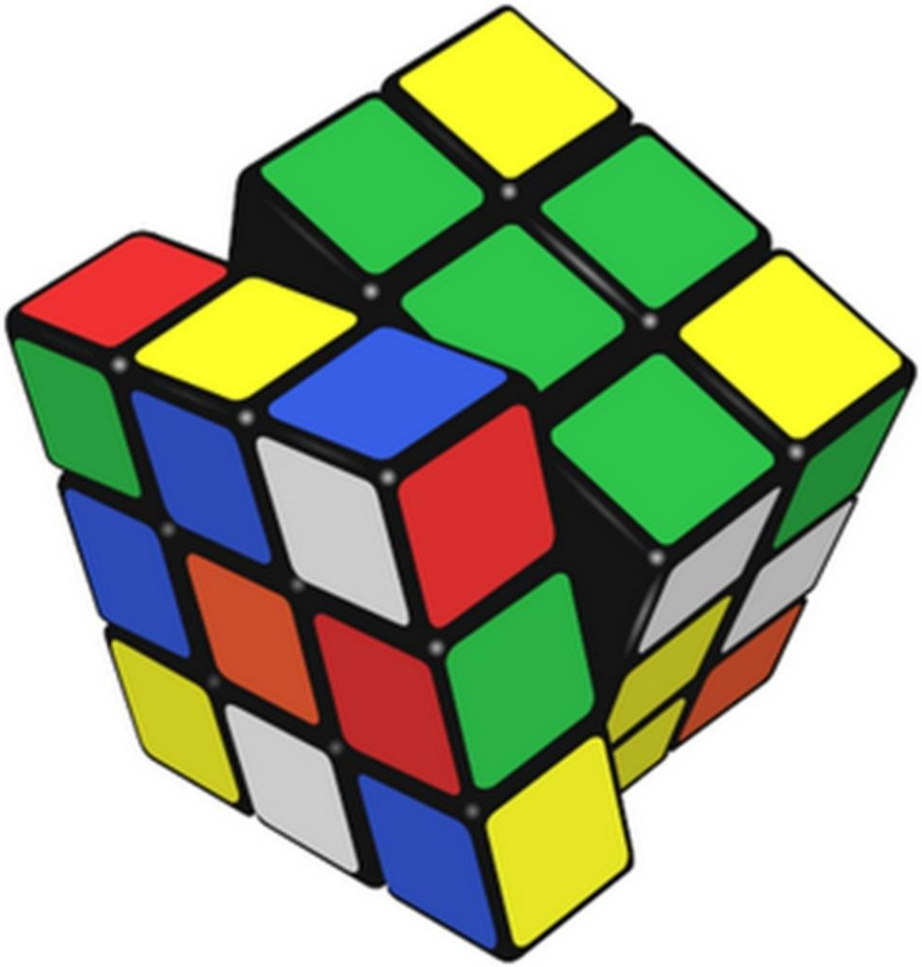
- Como encontrar uma função heurística?
- Dicas:
  - Imaginar o mesmo problema com menos restrições;
  - Se não tiver jeito de escolher entre duas heurística $h1(n)$ e $h2(n)$, usar $h(n)=\max(h1(n), h2(n))$;
  - usar informações estatísticas;
  - identificar as características de um estado que são úteis para definir a heurística.

Exemplo1 - mapa do Paraná.



h'(n) = distância em linha reta até o destino - distância Euclidiana

Exemplo 2: Cubo Mágico



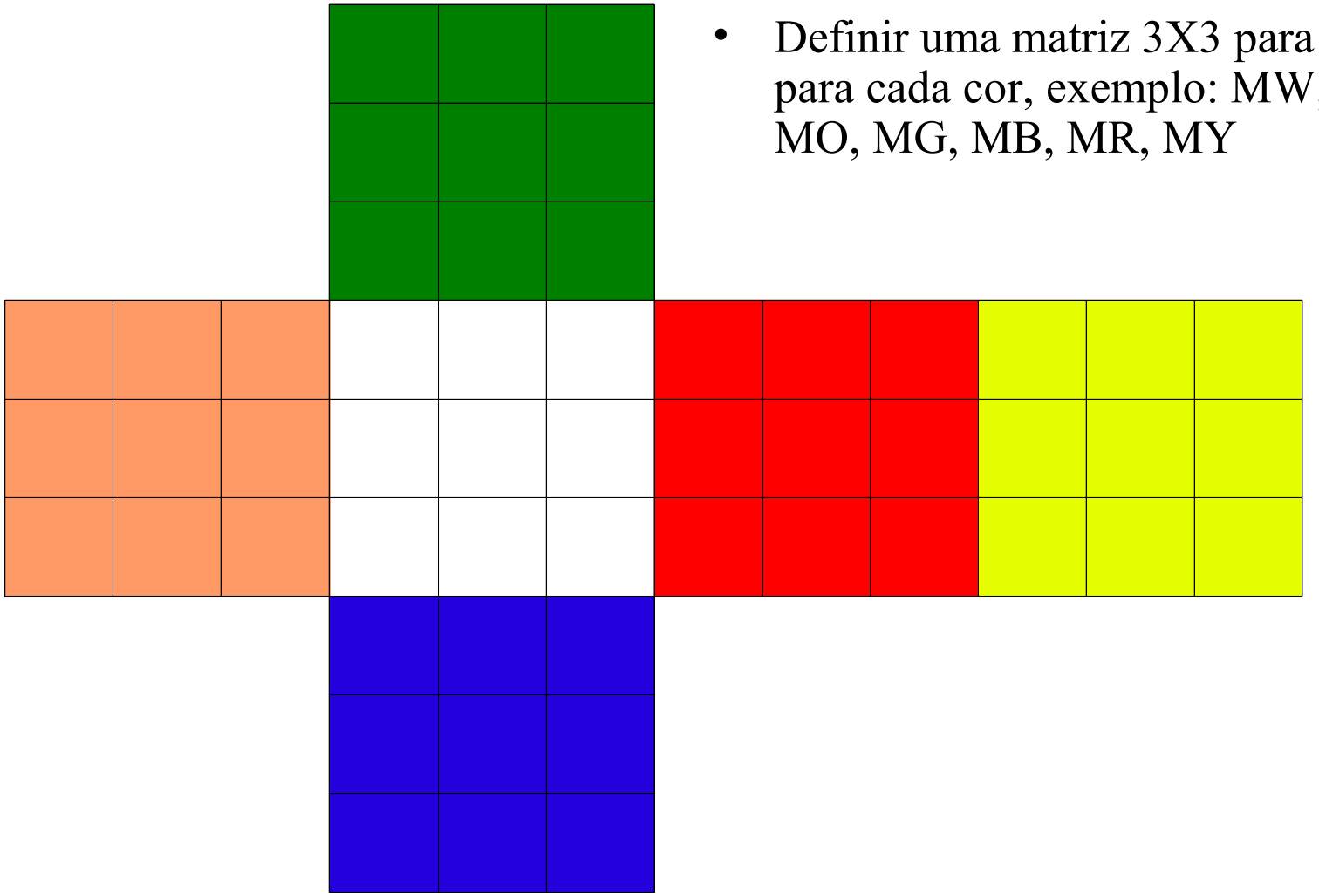
Ideias para o Cubo Mágico

- Heurísticas para h (sugestões):
  - h'1: a soma das cores fora de sua posição final*;
  - h'2: a soma das distâncias retangular de cada cor para levá-la para sua posição final*;
  - h'3: outra ideia?

Heurística final:

- h'= p1*h'1 +p2* h'2 +p3* h'3; sendo p1, p2 e p3 pesos entre 0 e 1 de cada heurística.
- * exceto para o caso em que as cores apareçam em sequência que com uma ou duas rotações todas elas sejam posicionadas.

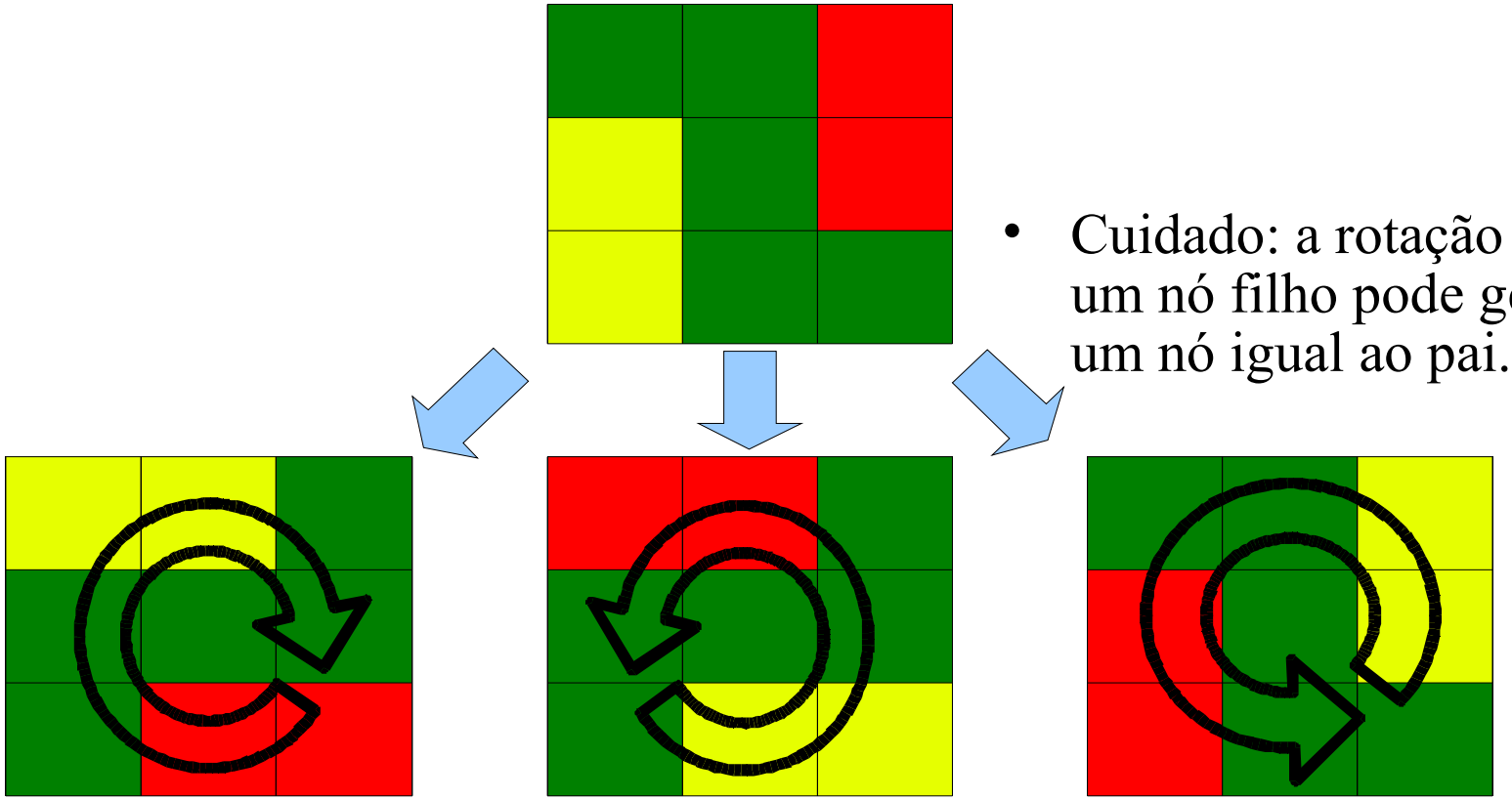
Ideias para o Cubo Mágico



- Definir uma matriz 3X3 para para cada cor, exemplo: MW, MO, MG, MB, MR, MY

Ideias para o Cubo Mágico

- Operador de rotação



- Cuidado: a rotação de um nó filho pode gerar um nó igual ao pai.

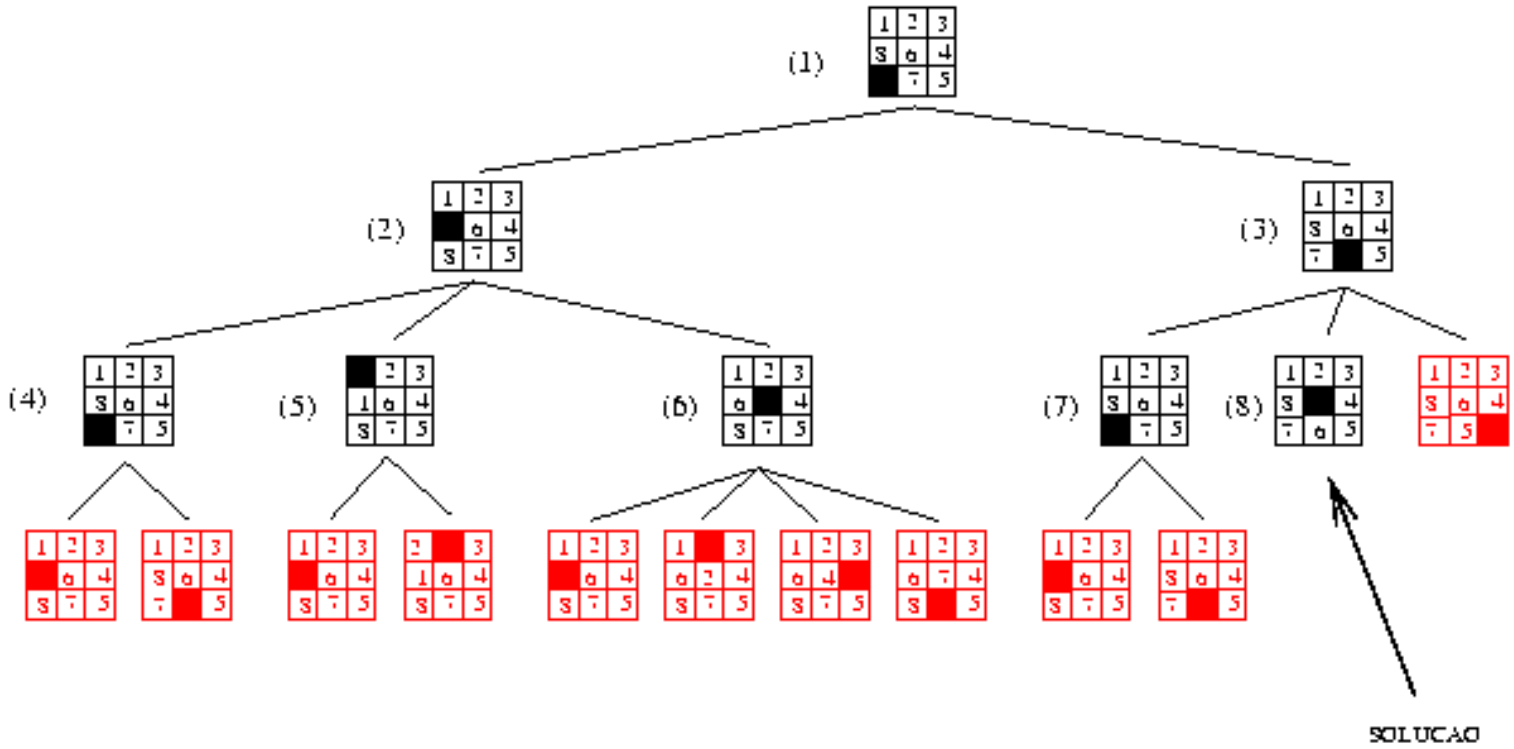
Exemplo 3 – Tabuleiro de 15 peças

- Versão de 8 peças tem 181,440 estados possíveis.
- Versão de 15 peças tem 10,461,394,944,000 estados possíveis.

Exemplo 3 - tabuleiro de 15 peças

- Funções heurísticas possíveis:
  - h'1(n) = número de peças foras de seu lugar na configuração final;
  - h'2(n) = número de peças fora de ordem seguindo a sequência de posições definida pela sequência numérica do estado final;
  - h'3(n) = para cada peça fora de seu lugar somar a distância retangular (quantidade de deslocamentos) para colocar em seu devido lugar. Neste caso considera-se que o caminho esteja livre para fazer o menor número de movimentos.

- Para este problema podemos:
  - usar apenas uma das funções;
  - fazer uma combinação do tipo: $h'(n) = p_1\ h'1(n) + p_2\ h'2(n) + p_3\ h'3(n)$, tal que $p_1 + p_2 + p_3 = 1$;
  - ou $h'(n) = \max(h'1(n), h'2(n), h'3(n))$


(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
SOLUCAO