Resolução de Problemas

- Já vimos o que é um problema. Vamos agora buscar mecanismos para representá-lo e resolvê-lo, utilizando as técnicas da IA, ou seja, usando e manipulando CONHECIMENTO
- O Estudo do Conhecimento
- Resolução de Problemas por Busca
- Representação de Conhecimento

Resolução de Problemas

- O Estudo do Conhecimento
  - Uma teoria em IA consiste na especificação do conhecimento necessário a uma entidade cognitiva.
  - O que é uma ENTIDADE COGNITIVA?
    - É o "mecanismo" inteligente que permite entre outras atividades: solução de problemas, uso de linguagem, tomada de decisões, percepção, etc...
    - Na abordagem da IA Simbólica, a simulação da capacidade cognitiva requer conhecimento declarativo (definição declarativa da função) e algum tipo de raciocínio. Além disso, a evolução dos estados de conhecimento de um agente pode ser descrita em forma de linguagem (lógica ou natural).

Resolução de Problemas

- O conhecimento é central para a tarefa inteligente e na IAS, para que esta tarefa ocorra são necessários:
  - Uma BASE DE CONHECIMENTOS
  - Um MOTOR DE INFERÊNCIA
- **Base de Conhecimentos:**
  - Contém a informação específica sobre o domínio e será tão complexa quanto for o domínio e a capacidade cognitiva a ser simulada.
- **Motor de Inferência:**
  - Mecanismo que manipula a Base de Conhecimentos e gera novas conhecimentos.

Métodos de Busca

- A maioria dos problemas interessantes de IA não dispõe de soluções algorítmicas. Porém:
  - São solucionáveis por seres humanos e, neste caso, sua solução está associada à "inteligência";
  - Formam classes de complexidade variável existindo desde pequenos problemas triviais (jogo da velha) até instâncias extremamente complexas (xadrez);
  - São problemas de conhecimento total, isto é, tudo o que é necessário para solucioná-los é conhecido, o que facilita sua formalização.
  - Suas soluções têm a forma de uma seqüência de situações legais e as maneiras de passar de uma situação para outra são em número finito e conhecidas.
- Diante da falta de solução algorítmica viável, o único método de solução possível é a BUSCA.

Métodos de Busca

- Espaço de estados
  - é a árvore de todos os estados que podemos produzir a partir do estado inicial. A busca vai percorrer esse espaço até achar o estado desejado
- *Exemplo: Problema das jarras de água.*
  - Um estado é representado por um par (X,Y), onde X e Y são números que indicam a quantidade de água que contém as jarras de 4 e 3 litros, respetivamente. O estado inicial é (0,0) e o sistema de produção consiste nos seguintes operadores:
    1. (X,Y) --> (4,Y) se X < 4
    2. (X,Y) --> (X,3) se Y < 3
    3. (X,Y) --> (0,Y) se X > 0
    4. (X,Y) --> (X,0) se Y > 0
    5. (X,Y) --> (X - min(X, 3-Y), min(3, X+Y)) se Y < 3
    6. (X,Y) --> (min(4, X+Y), Y - min(4-X, Y)) se X < 4

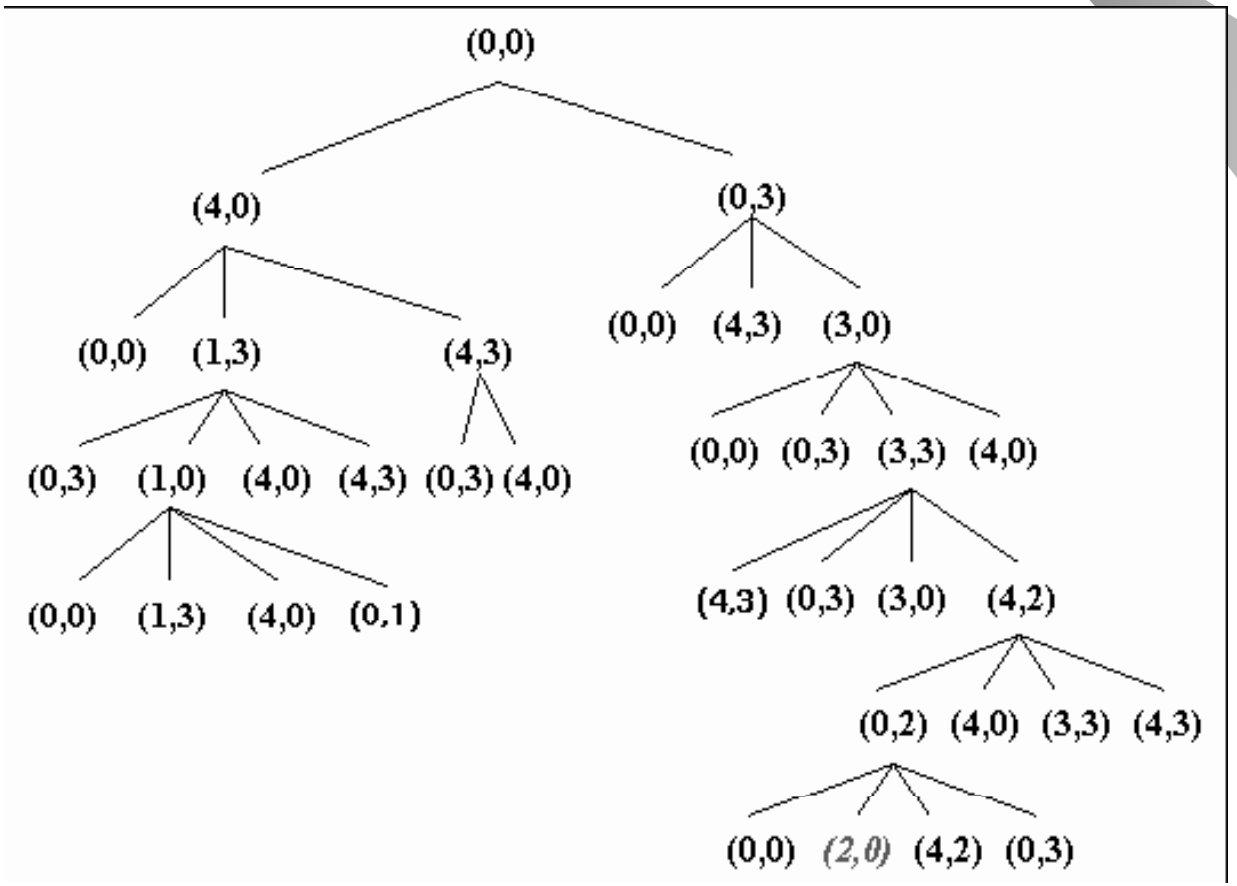
Métodos de Busca

1. (X,Y) --> (4,Y) se X < 4
2. (X,Y) --> (X,3) se Y < 3
3. (X,Y) --> (0,Y) se X > 0
4. (X,Y) --> (X,0) se Y > 0
5. (X,Y) --> (X - min(X, 3-Y), min(3, X+Y)) se Y < 3
6. (X,Y) --> (min(4, X+Y), Y - min(4-X, Y)) se X < 4

- Os dois primeiro operadores representam a ação de encher uma das jarras.
- Os operadores 3 e 4 representam a ação de esvaziar uma jarra.
- Finalmente, os dois últimos operadores representam a ação destransvasar (talvez parcialmente) o conteúdo de uma jarra na outra.
- O objetivo, nesse problema, é de obter 2 litros de água na jarra de 4 litros. Na nossa representação, isso corresponde ao estado (2,0).

Métodos de Busca

- Espaço de estados
  - Eis uma ilustração de parte do espaço de estados que contém o estado desejado (não mostramos a continuação dos estados repetidos):



Métodos de Busca

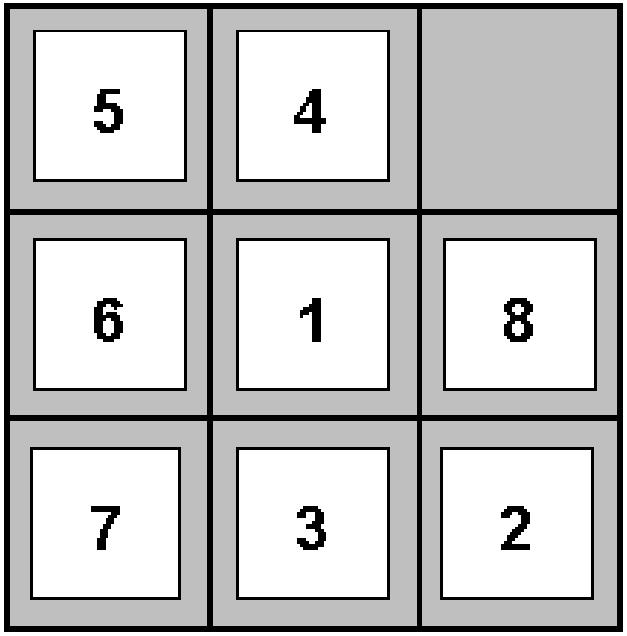
- Exemplos
  - Jogo da velha
    - Pode gerar 9! (362.880) caminhos diferentes
  - Jogo de xadrez
    - 10^{120} caminhos possíveis
  - Jogo de damas
    - 10^{40} caminhos
  - Jogo dos 15
    - 16!
  - Jogo dos 8
    - 9!
  - Problema do caixeiro viajante
    - (N-1)!

Métodos de Busca

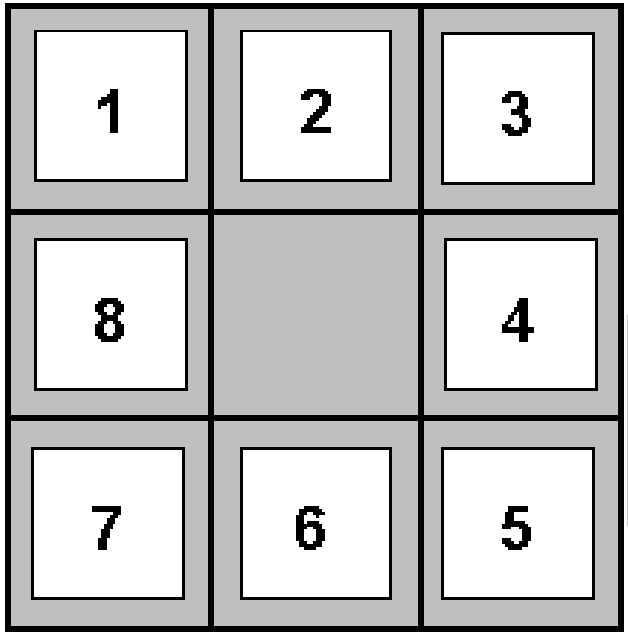
- Agentes de Resolução de Problemas
- Decidem o que fazer pela busca de ações que levem a estados desejáveis
  - estado inicial
  - operadores
  - teste de meta
  - função de custo de caminho
- Desempenho da Busca
  - Encontra uma solução?
  - É uma boa solução?
    - Custo do caminho
  - Qual o custo da busca?
    - Tempo e memória
  - Custo total=Custo da busca + custo do caminho

Métodos de Busca

- Exemplo: Jogo do Oito



Start State



Goal State

- **Estados:** local de cada uma das peças e do espaço
- **Operadores:** mover o espaço para cima, para baixo, esquerda ou direita.
- **Teste de meta:** dado na figura
- **Custo do caminho:** 1 para cada movimento

Métodos de Busca

- Método:
  - Geração dinâmica de uma árvore representando os estados alcançáveis a partir de um estado. A seleção do estado a ser expandido é determinado pela estratégia de controle. O processo pára quando o estado designado por um nodo folha corresponde ao objetivo.
- Estruturas necessárias :
  - **Nodo:**
    - Estado
    - Nodo pai
    - Operador utilizado para produzir o nodo
    - Profundidade
    - Custo do caminho até o nodo
  - **Fila:**
    - Contém os nodos produzidos que estão esperando para ser expandidos. Esses nodos constituem a **fronteira da busca.**

Métodos de Busca

- Algoritmo geral de busca :

```
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-NA-FILA(nodos,novos-nodos)
fim
```

- Nota: A função EXPANDIR retorna uma lista de nodos que representam os estados resultando da aplicação de todos os operadores possíveis ao estado representado por nodo.

Métodos de Busca

- Estratégias de Busca
  - Critérios
    - Completude
    - Complexidade de Tempo
    - Complexidade de Espaço
    - Otimização
  - Métodos
    - Busca Cega - Não existe informação
    - Busca Heurística - Faz uso de informação

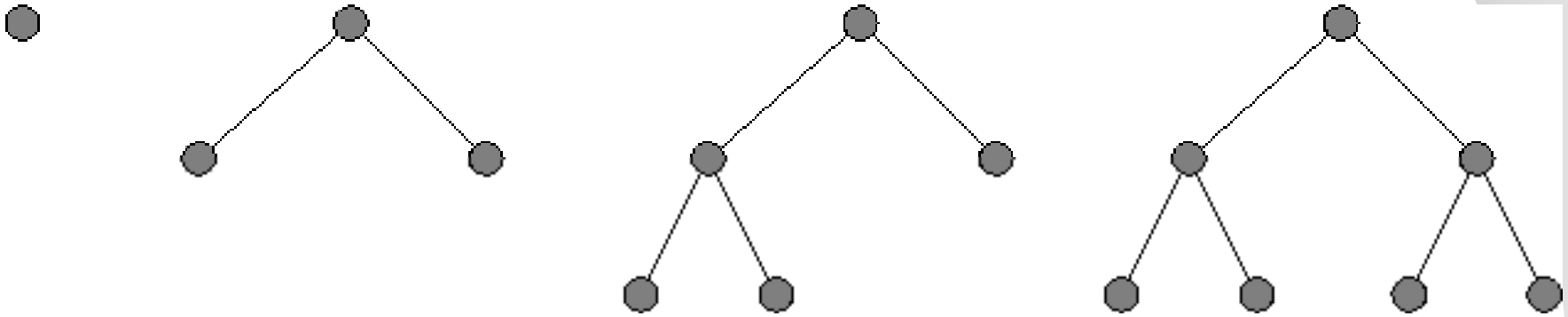
Busca Cega
(Blind Search ou Uninformed Search)

- Uma estratégia de busca é dita cega se ela não leva em conta informações específicas sobre o problema a ser resolvido.

- Tipos de Busca Cega
  - Busca em largura (breadth-first)
  - Busca pelo custo uniforme (branch-and-bound)
  - Busca em profundidade (depth-first)
  - Busca em profundidade limitada
  - Busca por aprofundamento iterativo
  - Busca bidirecional

Busca em Largura (Amplitude) - Breadth-First

- Consiste em construir uma árvore de estados a partir do estado inicial, aplicando a cada momento, todas as regras possíveis aos estados do nível mais baixo, gerando todos os estados sucessores de cada um destes estados. Assim, cada nível da árvore é completamente construído antes de qualquer nodo do próximo nível seja adicionado à árvore



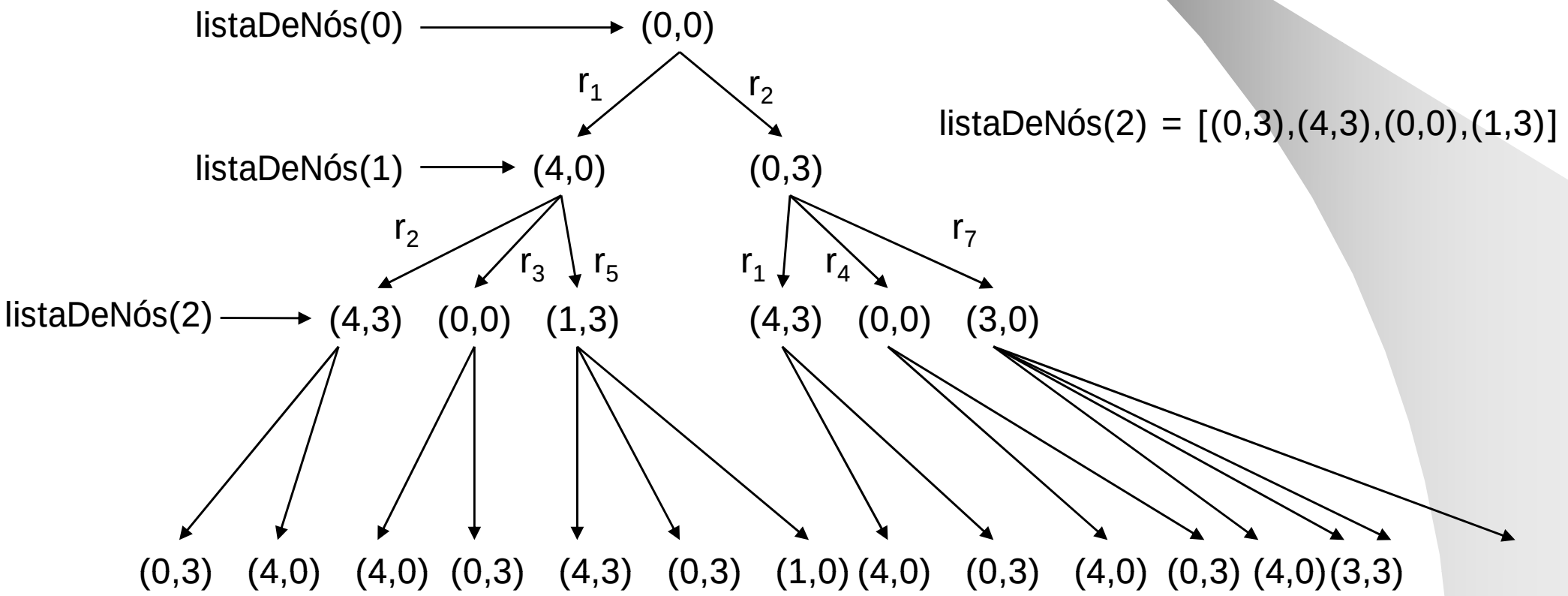
Busca em Largura (Amplitude) - Breadth-First

- Obtemos uma busca em largura se substituimos ACRESCENTAR-NA-FILA por ACRESCENTAR-NO-FIM no algoritmo geral de busca:

```
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
```

Busca em Largura (Amplitude)

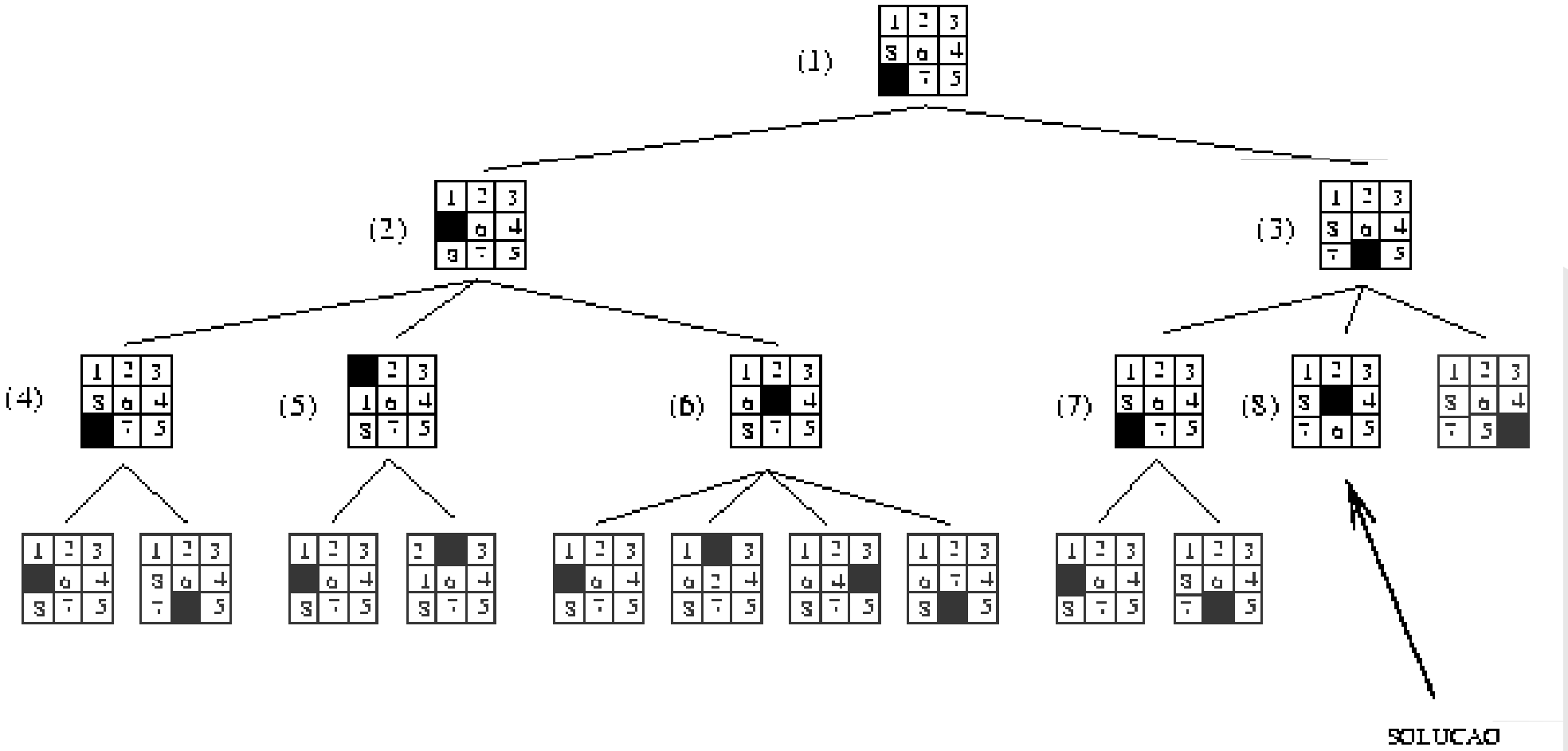
- Exemplo: Um balde de 4 litros e um balde de 3 litros. Inicialmente vazios.
  - Estado Final: um dos baldes com 2 litros de água.



Busca Cega

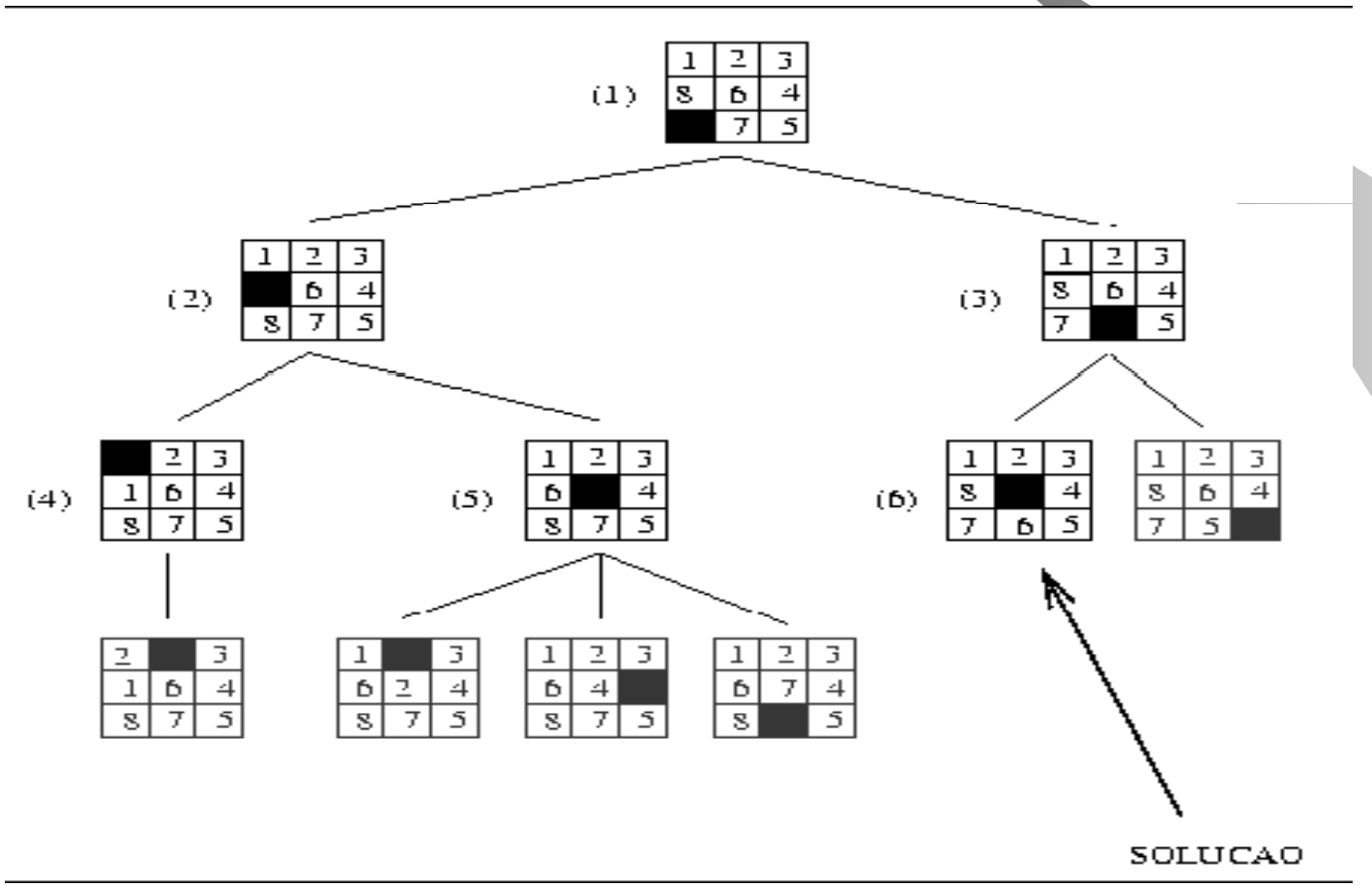
Busca em Largura (Amplitude)

- Exemplo: Jogo do Oito



Busca em Largura (Amplitude)

- Exemplo: Jogo do Oito (sem estado repetidos)



Busca em Largura (Amplitude)

- Características: Completa e Ótima (se os operadores sempre têm o mesmo custo).
  - Se existe solução, esta será encontrada;
  - A solução encontrada primeiro será a de menor profundidade.

- Análise de Complexidade – Tempo e Memória
  - Seja um fator de ramificação b.
    - Nível 0: 1 nó
    - Nível 1: b nós
    - Nível 2: b^2 nós
    - Nível 3: b^3 nós

    - Nível d (solução) b^d nós
  - Complexidade: exponencial - $O(b^d)$

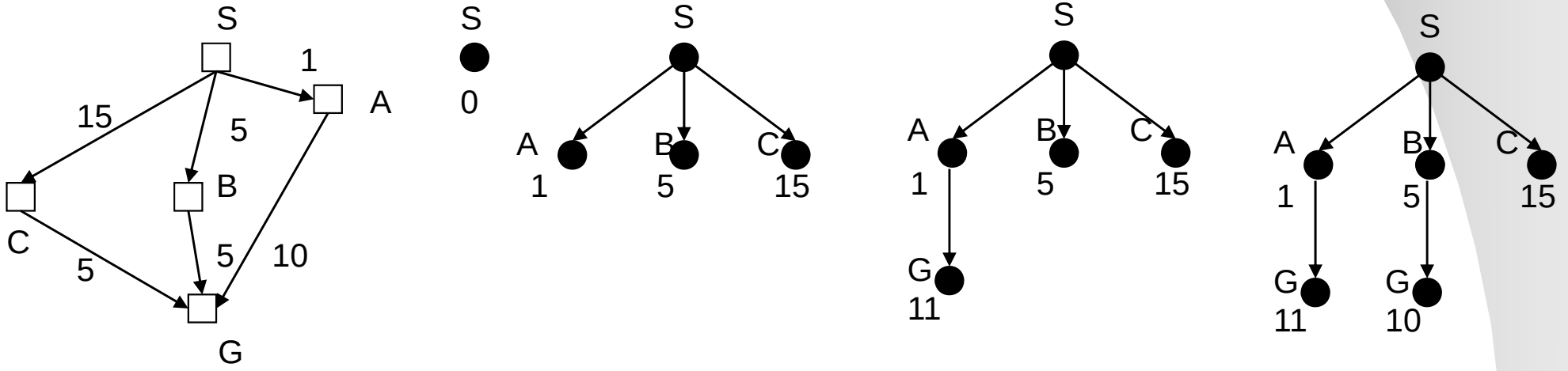
Busca em Largura (Amplitude)

- Análise de Complexidade – Tempo e Memória

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Método do Custo Uniforme - Branch-and-Bound)

- Supondo que exista um "custo do caminho" associado a cada nó percorrido e que se deseje achar o caminho de custo mínimo.
- Neste caso, o algoritmo anterior é modificado para expandir primeiro o nó de menor custo.
- Exemplo: Problema de Rota entre S e G



Busca Cega

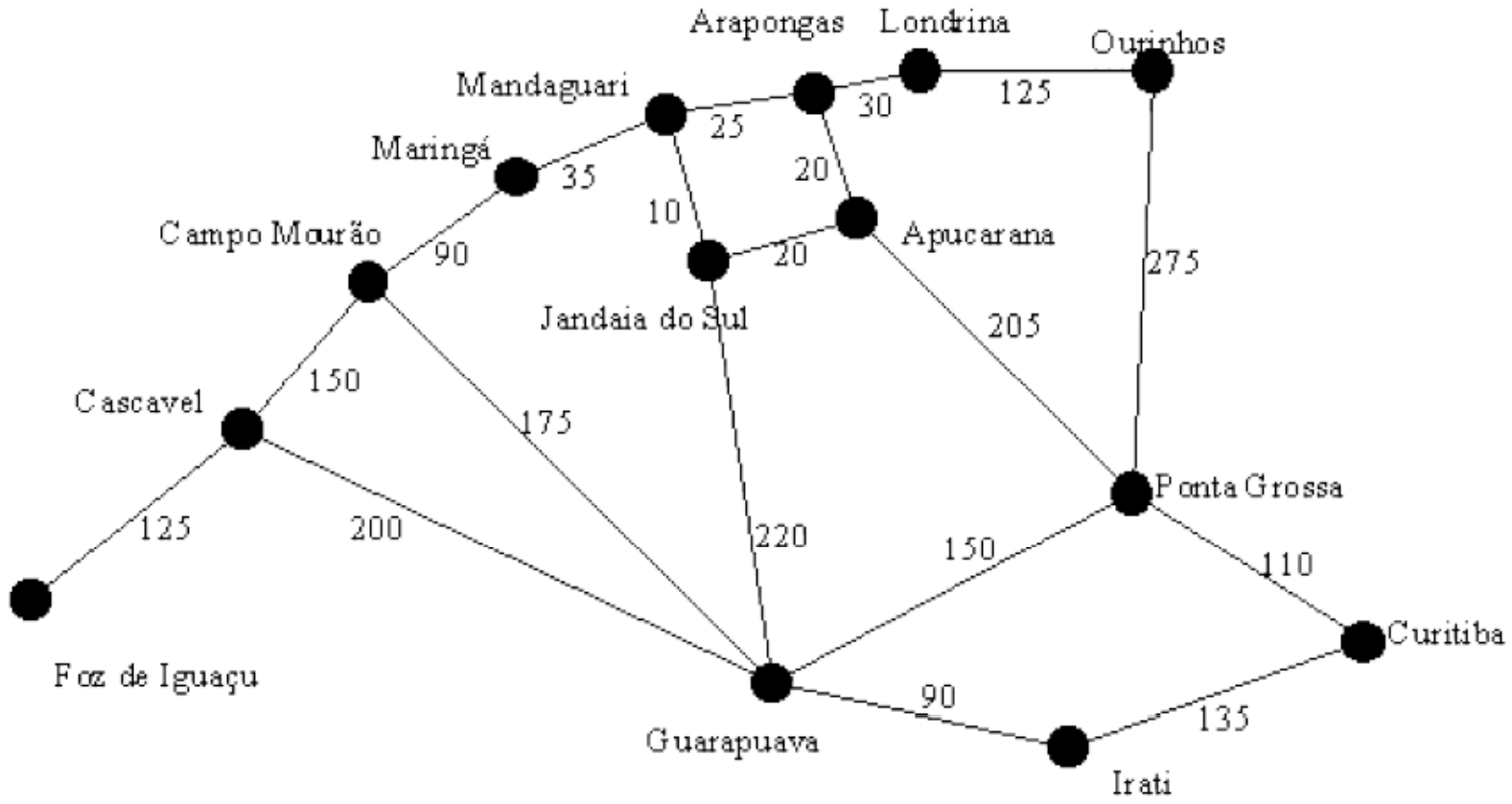
Método do Custo Uniforme - Branch-and-Bound)

```
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
```

Busca Cega

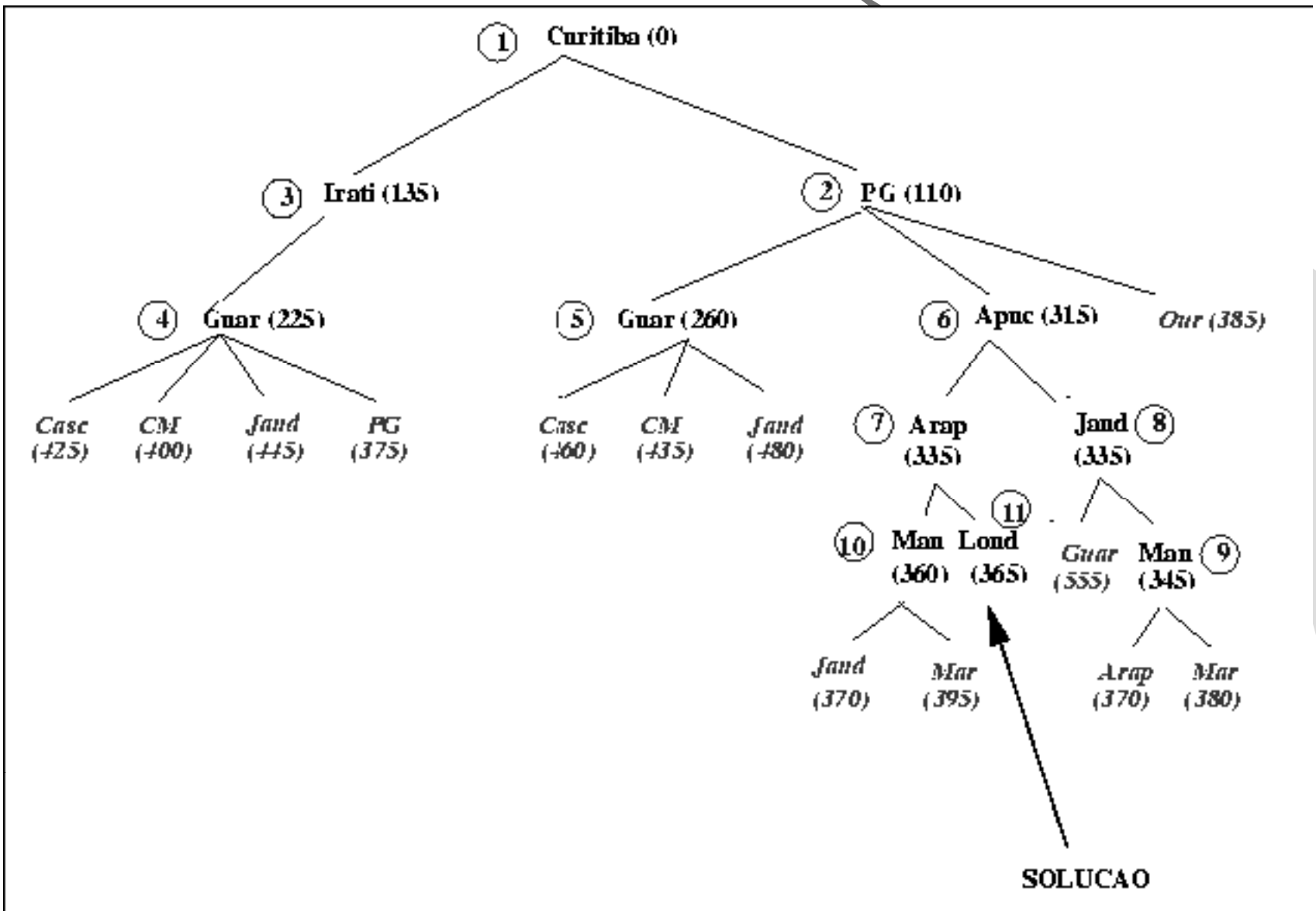
Método do Custo Uniforme - Branch-and-Bound)

- *Exemplo: Achar o caminho mais curto entre Curitiba e Londrina*



Busca Cega

Método do Custo Uniforme - Branch-and-Bound)



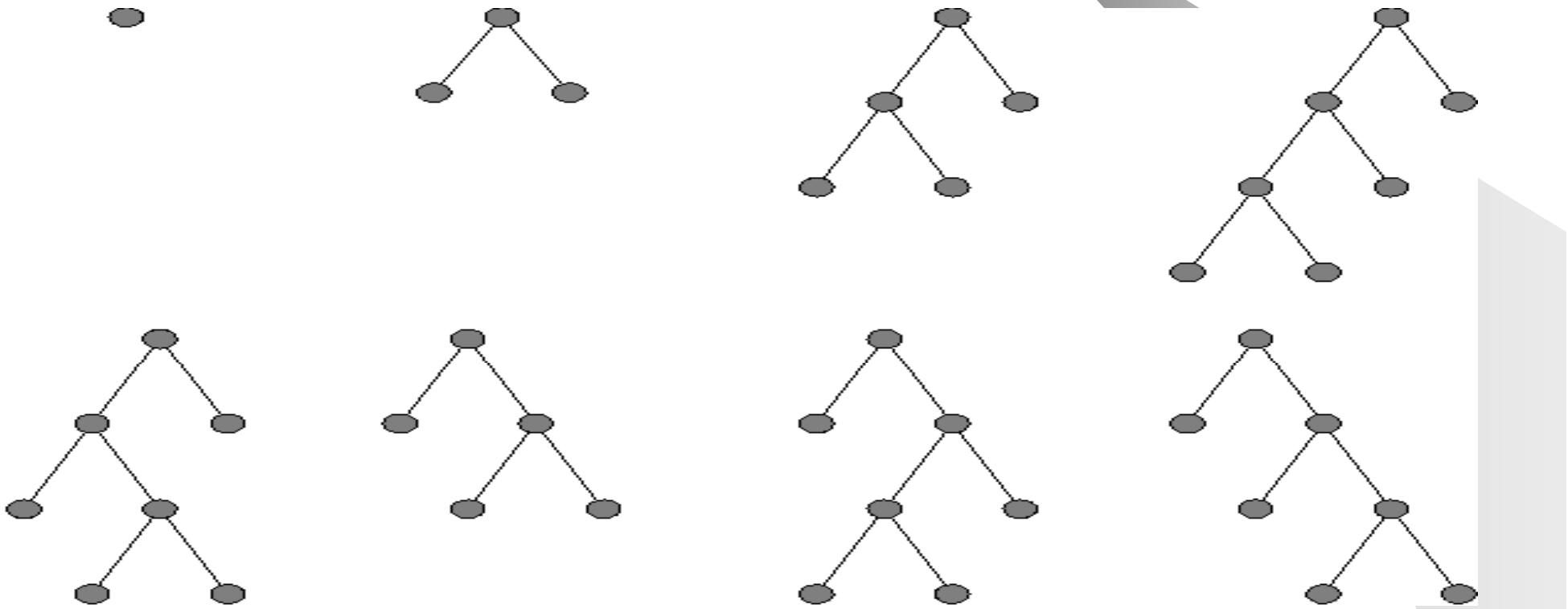
Busca Cega

Método do Custo Uniforme - Branch-and-Bound)

- Vantagens desse algoritmo:
  - Completo
  - Ótimo, se o custo até o próximo nodo nunca é negativo.
- Desvantagens :
  - Complexidade em memória e tempo igual à da busca em largura: $O(b^d)$ onde b é o fator de ramificação e d a profundidade.

Busca em Profundidade – Depth-First

- Procurar explorar completamente cada ramo da árvore antes de tentar o ramo vizinho.



Busca em Profundidade

- Obtemos uma busca em profundidade se substituirmos ACRESCENTAR-NO-FIM por ACRESCENTAR-NO-INICIO no algoritmo geral de busca:

```
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

- Obtemos uma busca em profundidade se substituirmos ACRESCENTAR-NO-FIM por ACRESCENTAR-NO-INICIO no algoritmo geral de busca:

```
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
```

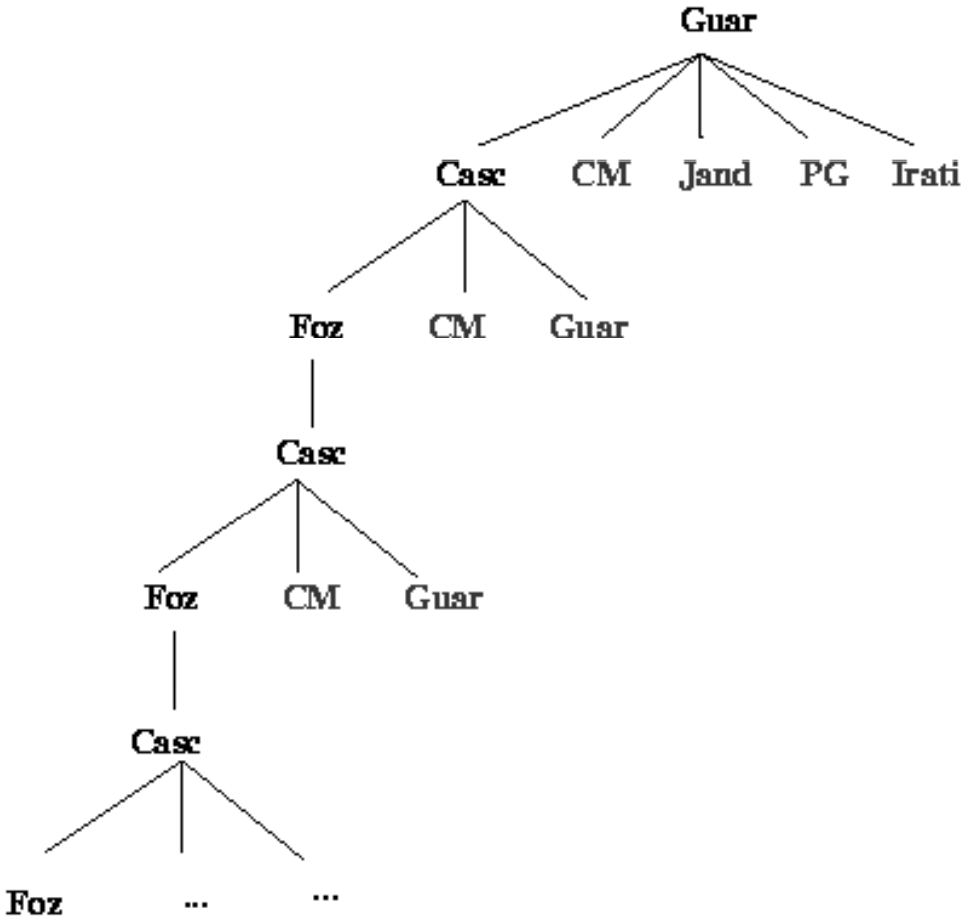
Busca em Profundidade

- Características: Não é Completa e Não é Ótima
  - Se admitir estados repetidos ou um nível máximo de profundidade, pode nunca encontrar a solução.
  - A solução encontrada primeiro poderá não ser a de menor profundidade.
  - O algoritmo não encontra necessariamente a solução mais próxima, mas pode ser MAIS EFICIENTE se o problema possui um grande número de soluções ou se a maioria dos caminhos pode levar a uma solução.

- Análise de Complexidade – Tempo e Memória
  - Seja m a profundidade máxima e um fator de ramificação b.
  - Tempo: $O(b^m)$
  - Memória: $O(b.m)$

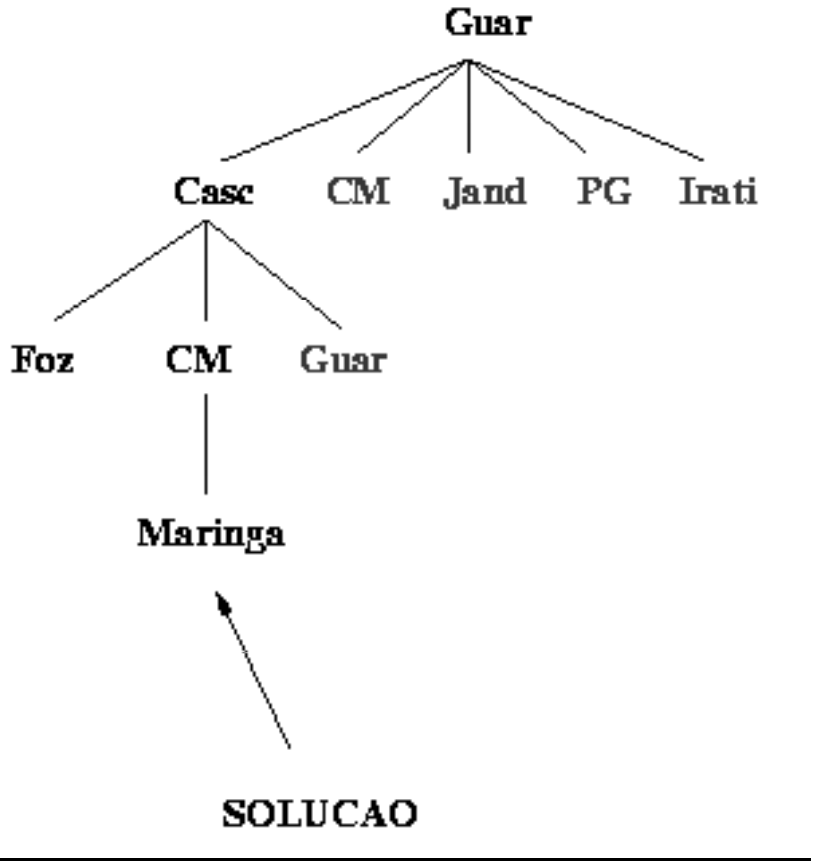
Busca em Profundidade

- Exemplo: Achar um caminho entre Guarapuava e Maringá (admitindo estados repetidos)



Busca em Profundidade

- Exemplo: Achar um caminho entre Guarapuava e Maringá (SEM estados repetidos)



Busca em Profundidade

- O que acontece quando nenhuma regra pode ser aplicada, ou a árvore atinge uma profundidade muito grande sem que tenha encontrado uma solução?
  - Neste caso ocorre o BACKTRACKING, ou seja, o algoritmo volta atrás e tenta outro caminho.
  - Considere o seguinte sistema de produção:

E={0,1,2,3,4,5}

e0=0

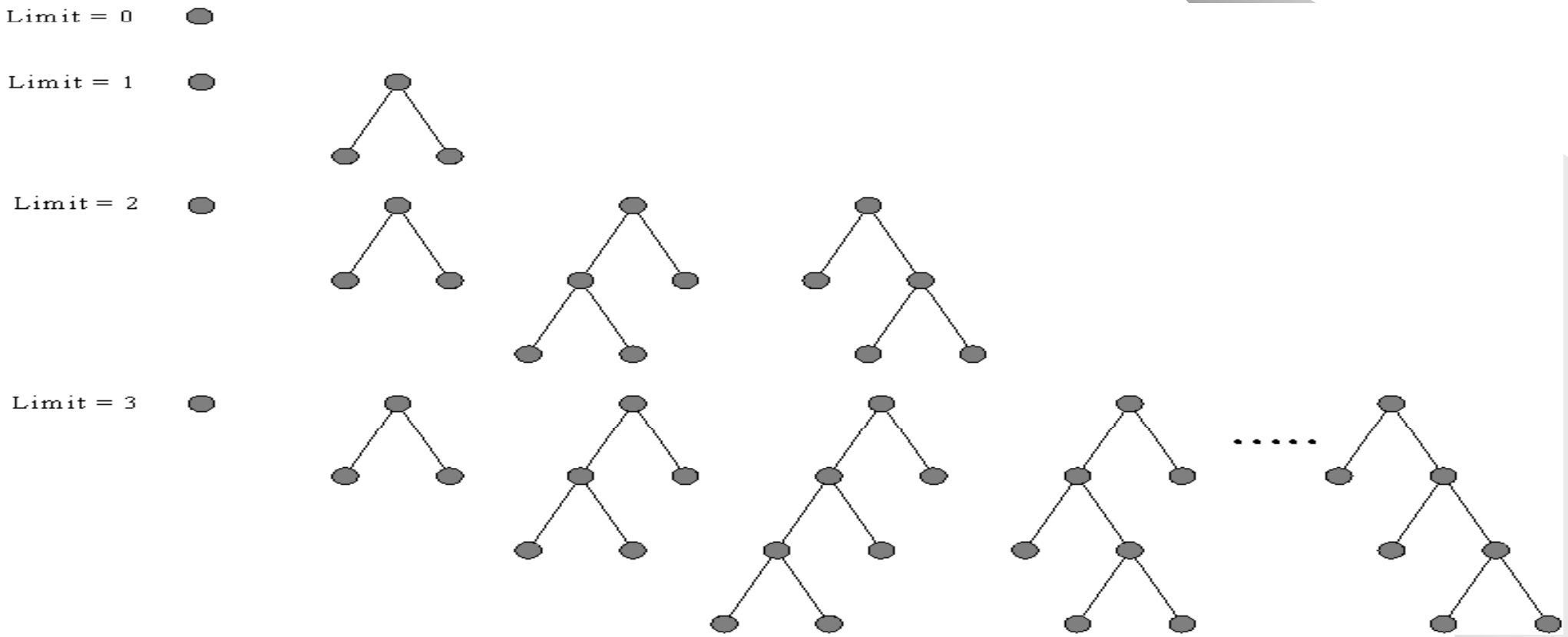
F={3}

R={ r1=(x|x >= 1 e x <= 2)->(2*x)

r2=(x|é Par(x))->(x+1)}

Busca por Aprofundamento Iterativo

- Teste de todos os possíveis limites com busca por profundidade limitada.
- Em geral é o melhor método quando o espaço de busca é grande e a profundidade é desconhecida.



Busca por Aprofundamento Iterativo

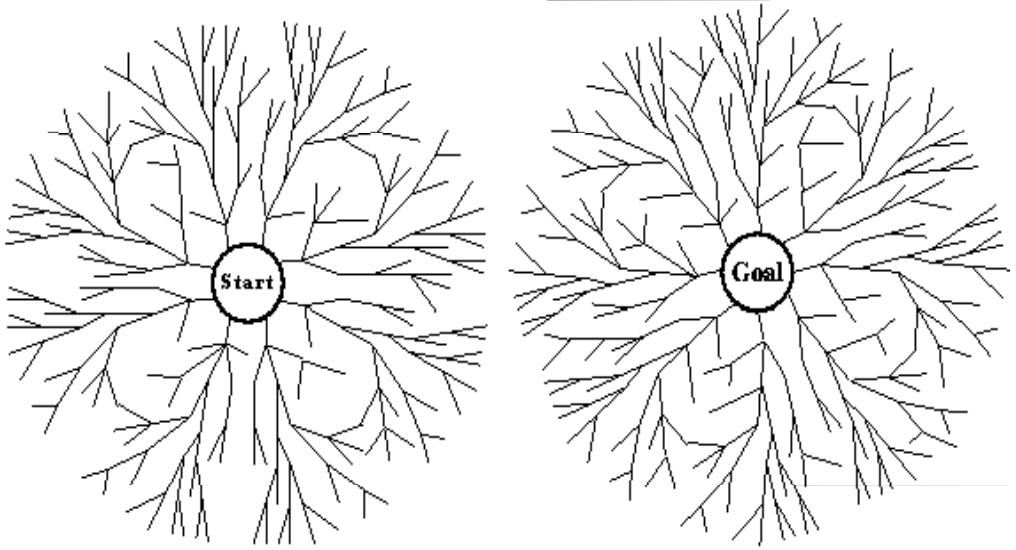
- A busca em profundidade iterativa parece muito ineficiente pois o mesmo nodo pode ser expandido muitas vezes.
- Quando fazemos mais uma iteração para o nível n+1, todos os nodos que foram criados no nível n vão ser criados de novo.
- Surpreendentemente, o algoritmo não é muito mais ineficiente que a busca em largura. Por exemplo, com um fator de ramificação de 10, o número de nodos expandidos, em relação à busca em largura, é somente 11% maior.
- Com a busca em profundidade iterativa, obtemos um comportamento idêntico à busca em largura no que concerne a completude e a otimalidade, mas com um uso de memória mais viável.
- Se a função de custo não é uniforme, ele tem também o mesmo defeito que a busca em largura: é possível que a resposta retornada não seja ótima.

Busca Cega

Busca Bidirecional

- A idéia deste método de busca é procurar simultaneamente "para a frente" a partir do estado inicial e "para trás" a partir do estado final, e parar quando as duas buscas se encontrarem no meio.
- Nem sempre isto é possível, para alguns problemas os operadores não são reversíveis, isto é, não existe a função predecessora e portanto não é possível fazer a busca "para trás".

- Análise de Complexidade
  - Comparando com a busca em largura, o tempo e o espaço para a busca é $O(b^{d/2})$, onde d é o nível onde está a solução e b é o fator de ramificação da árvore.
  - Exemplo: Para b=10 e d=6, na busca em largura seriam gerados 1.111.111 nós, enquanto que na busca bidirecional seriam gerados 2.222 nós.



Comparação entre Métodos de Busca

	Prof.	Ampl.	Custo unif.	Prof. limit.	Prof. iterat.
Tempo	b^m	b^d	b^d	b^l	b^d
Memória	bm	b^d	b^d	bl	bd
Sol. ótima	Não	Sim*	Sim**	Não	Sim*
Completude	Não	Sim	Sim	Sim, se l >= d	Sim

b = fator de ramificação
d = profundidade da solução
l = limite de profundidade especificado
m = profundidade máxima atingida na busca
* Somente se o custo para de um estado ao próximo é sempre o mesmo (função de custo uniforme).
** Somente se o custo não diminua quando o caminho aumenta.

Busca Heurística (Informed Search)

- Os métodos de busca vistos anteriormente fornecem uma solução para o problema de achar um caminho até um nó meta. Entretanto, em muitos casos, a utilização destes métodos é impraticável devido ao número muito elevado de nós a expandir antes de achar uma solução.
- Para muitos problemas, é possível estabelecer princípios ou regras práticas para ajudar a reduzir a busca.
- A técnica usada para melhorar a busca depende de informações especiais acerca do problema em questão.
- Chamamos a este tipo de informação de INFORMAÇÃO HEURISTICA e os procedimentos de busca que a utilizam de MÉTODOS DE BUSCA HEURISTICA.

Busca Heurística (Informed Search)

- A informação que pode compor uma informação heurística é o Custo do Caminho (f(n)).
- O CUSTO DO CAMINHO pode ser composto pelo somatório de dois outros custos:
  1. O custo do caminho do estado inicial até o estado atual que está sendo expandido (função g(n)); e
  2. Uma estimativa do custo do caminho do estado atual até o estado meta (função heurística h(n)).

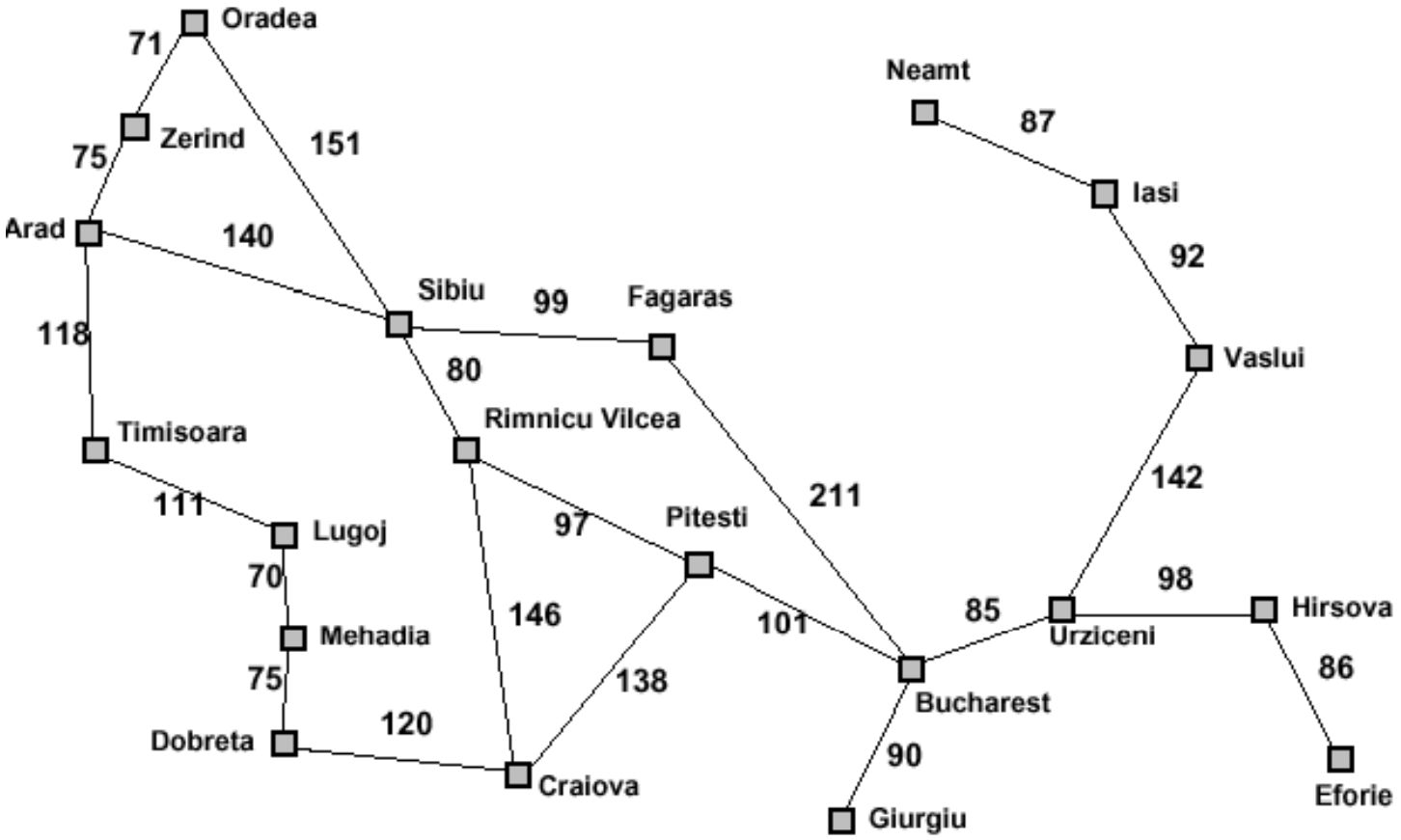
- A filosofia geral que move a busca heurística é: O MELHOR PRIMEIRO (best-first). Isto é, no processo de busca deve-se primeiro expandir o nó "mais desejável" segundo uma função de avaliação.

Busca Heurística (Informed Search)

nodos <-- CRIAR-FILA(estado-inicial)
loop
  se *nodos é vazio retorna falha*
  nodo <-- TIRAR-MELHOR-NODO(nodos)
  se TESTE-SUCESSO(*nodo*) *tem sucesso*
    retorna *nodo*
  novos-nodos <-- EXPANDIR(nodo)
  nodos <-- ACRESCENTAR(*nodos,novos-nodos)*
fim

Busca Heurística (Informed Search)

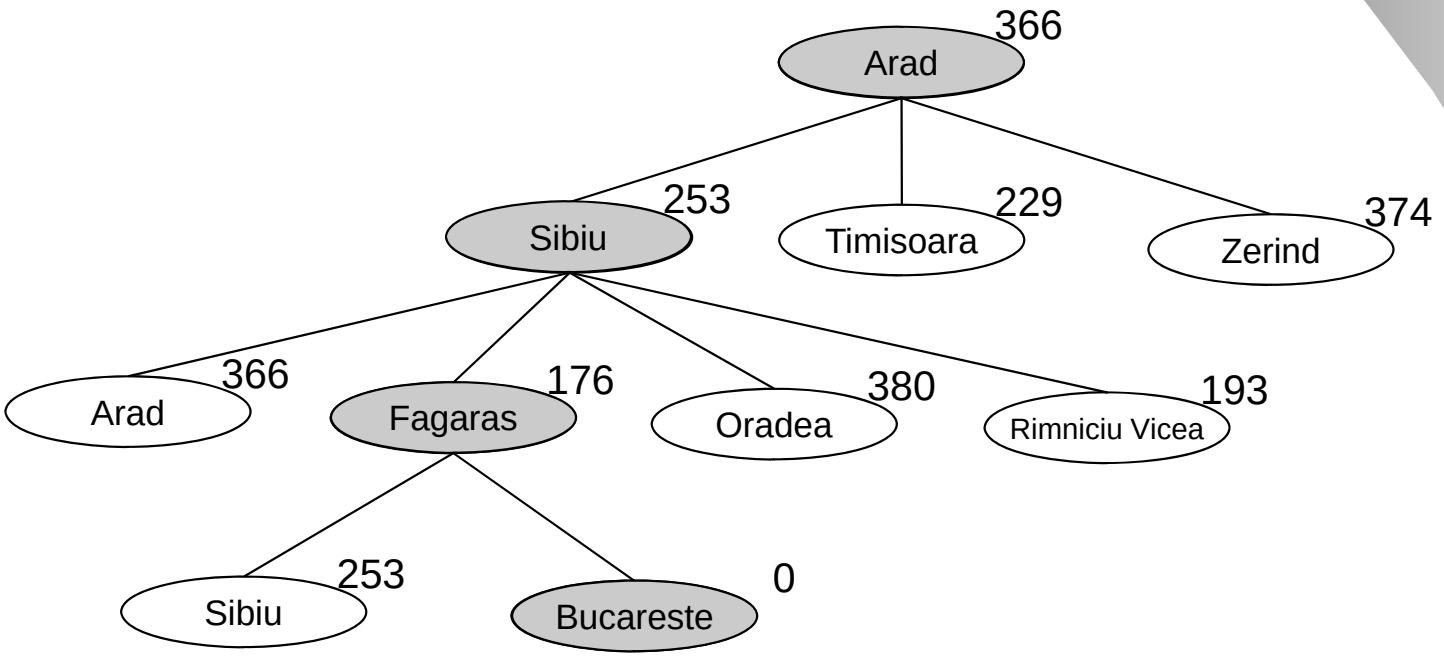
Romania with step costs in km



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca Gulosa (Greedy Search)

- Semelhante à busca em profundidade com backtracking.
- Tenta expandir o nó que parece mais próximo ao nó meta com base na estimativa feita pela função heurística h.
- No caso do mapa da Romênia, h(n) é a distância em linha reta de n até Bucareste.



Busca Gulosa (Greedy Search)

- Análise de Complexidade
  - É completa se não admitir estados repetidos;
  - Tempo: $O(b^m)$, mas uma boa heurística pode reduzir drasticamente o tempo;
  - Espaço: $O(b^m)$, todos os nós são mantidos na memória;
  - Não garante a solução ótima.

Busca Heurística

Busca A* (A estrela)

- Filosofia: procurar evitar expandir nós que já são "custosos".
- É um método de busca que procura otimizar a solução, considerando todas as informações disponíveis até aquele instante, não apenas as da última expansão.
- Todos os estados abertos até determinado instante são candidatos à expansão.
- Combina, de certa forma, as vantagens tanto da busca em largura como em profundidade
- Busca onde o nó de menor custo "aparente" na fronteira do espaço de estados é expandido primeiro.

- f(n) = g(n) + h(n) onde
  - g(n) = custo do caminho do nó inicial até o nó n.
  - h(n) = custo do caminho estimado do nó n até o nó final.
  - f(n) = custo do caminho total estimado.

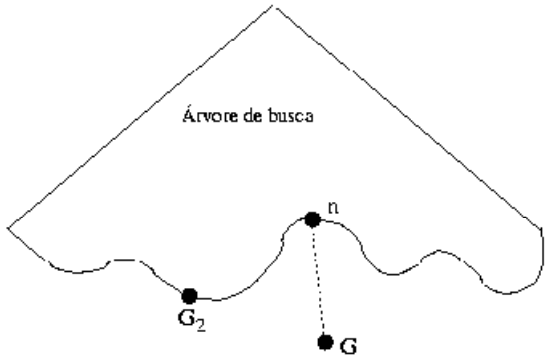
Busca A* (A estrela)

- A* expande o nó de menor valor de f a cada instante.
- A* deve usar uma heurística admissível, isto é, $h(n) \leq h^*(n)$ onde $h^*(n)$ é o custo real para ir de n até o nó final.
- **Admissibilidade de A***
  - Diz-se que um método de busca é ADMISSÍVEL se ele sempre encontra uma solução e se esta solução é a de menor custo.
  - A busca em largura é admissível. O mesmo não ocorre com a busca em profundidade.

- **Teorema da Admissibilidade de A***
  - A busca A* é ótima, isto é, sempre encontra o caminho de menor custo até a meta.

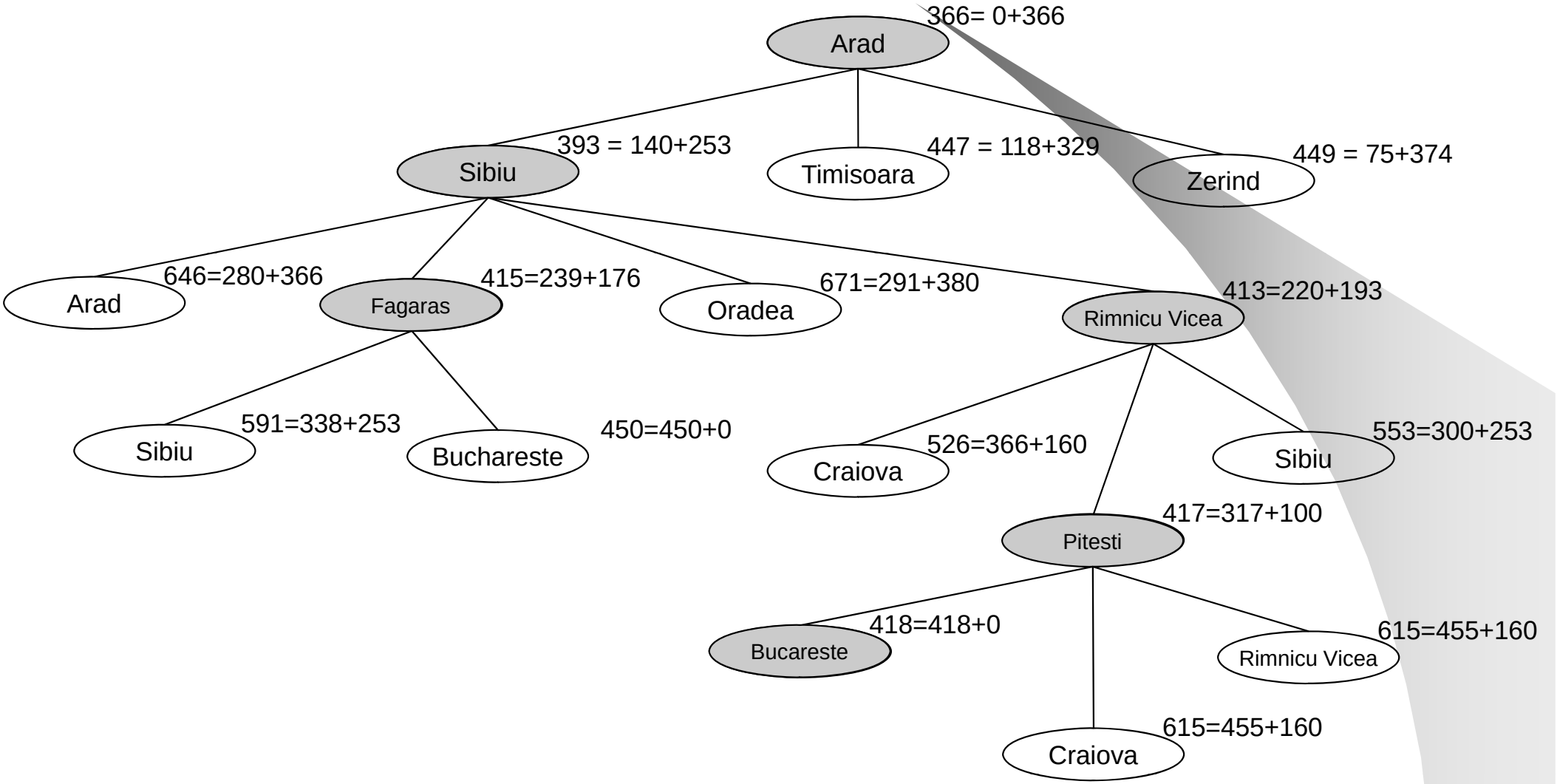
Busca A* (A estrela)

- *Prova da otimalidade do algoritmo A*:*
  - Seja *G o estado final de uma solução ótima, f* o custo dessa solução ótima, e G2 o estado final de uma* solução não ótima com custo *g(G2) > f*.*
  - Suponhamos que o algoritmo A* visita *G2 primeiro. Então existe, nas lista dos nodos abertos, um nodo n* que faz parte do caminho até *G. Como a função h(n) é admissível, temos f* >= f(n).*
  - Visto que *G2 foi selecionado antes de n, temos também f(n) >= f(G2). Como f* >= f(n), podemos deduzir f* >= f(G2).*
  - *G2 sendo o estado final, f(G2) = g(G2) + 0.*
  - Então, *f* >= g(G2), o que contradiz a hipótese.*



Busca Heurística

Busca A* (A estrela)



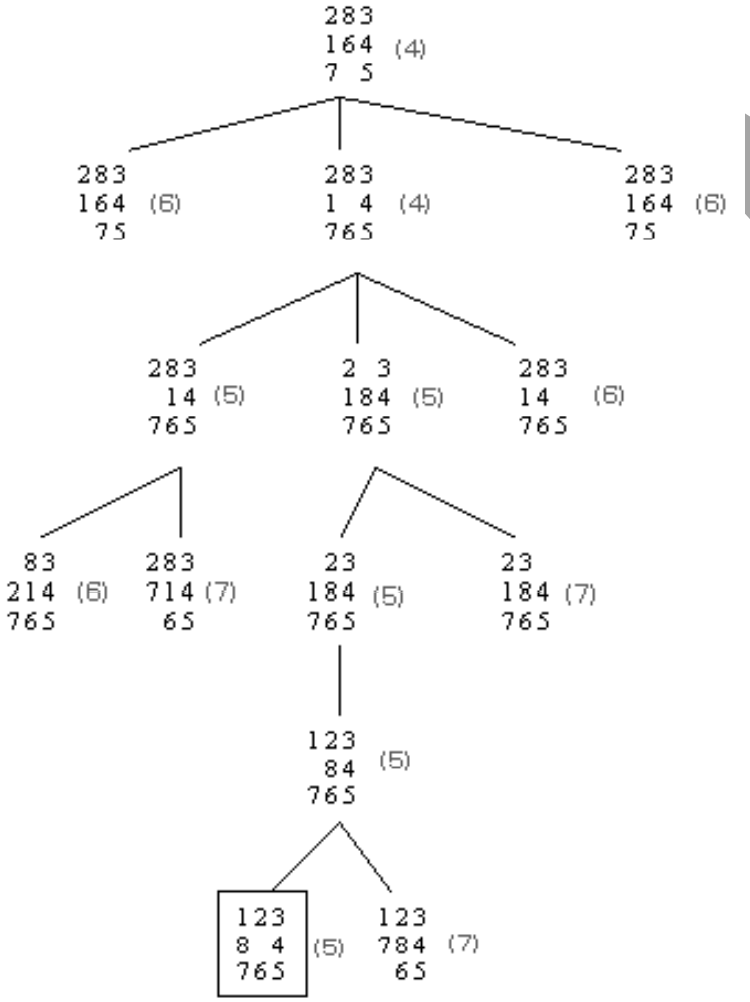
Busca A* (A estrela)

- Quanto mais admissível a heurística, menor o custo da busca.
- Exemplo: Para o jogo do oito
  - h1(n): número de peças fora do lugar
  - h2(n): distância Manhattan (número de casas longe da posição final em cada direção)

Busca Heurística

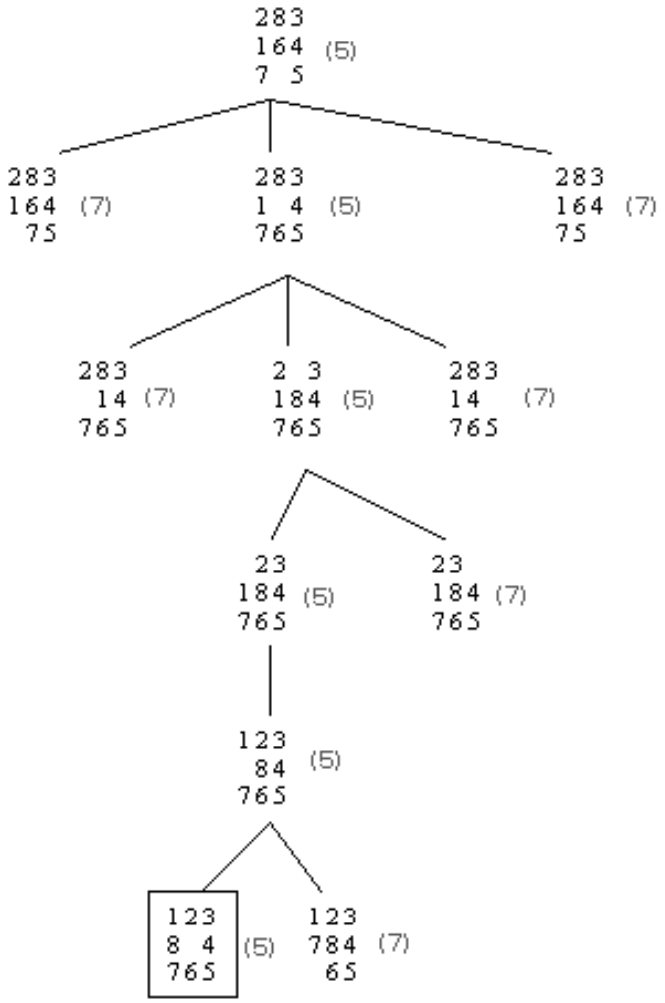
Busca A* (A estrela)

- Exemplo: Para o jogo do oito
  - h1(n): número de peças fora do lugar



Busca A* (A estrela)

- Exemplo: Para o jogo do oito
  - h2(n): distância Manhattan (número de casas longe da posição final em cada direção)

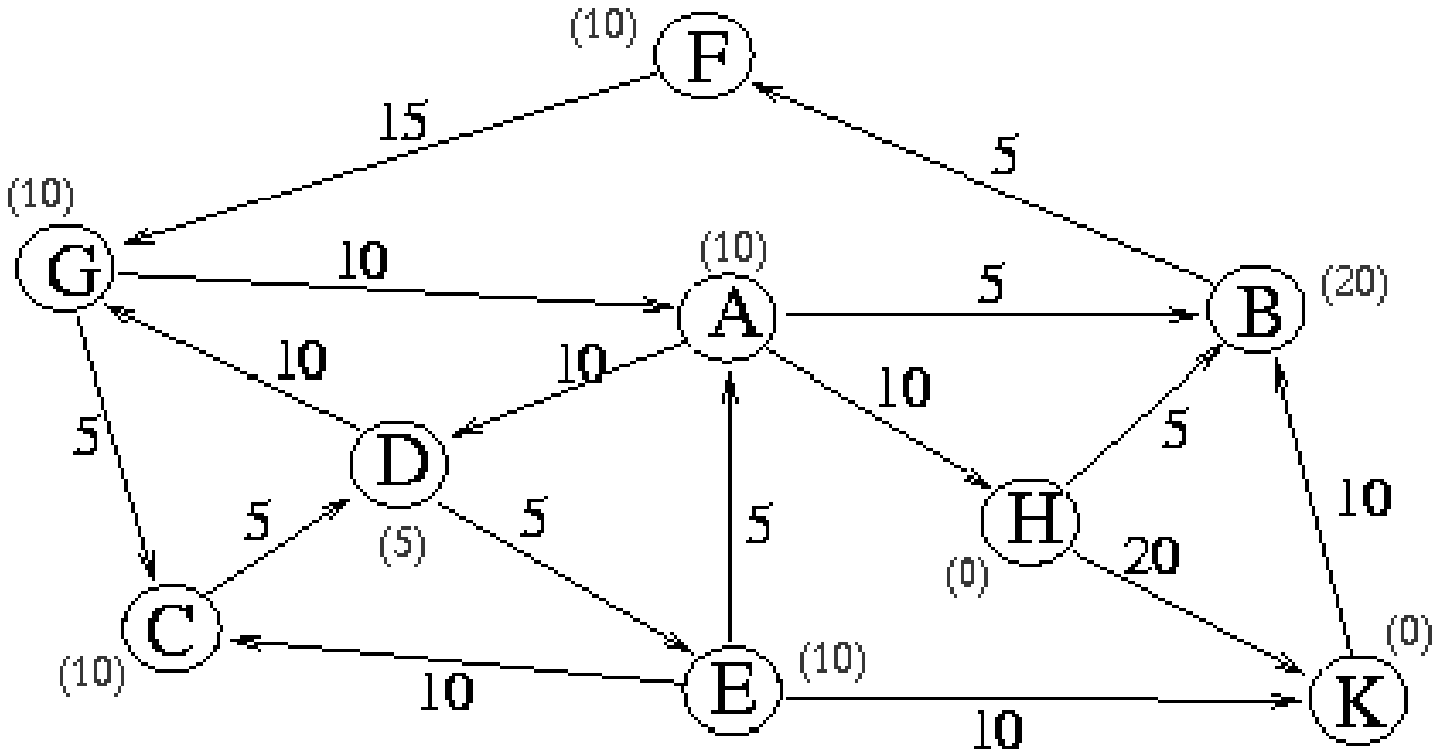


Busca IDA* (A estrela iterativo)

- O maior problema com A* é o uso da memória.
- Para contornar esse problema, podemos utilizar a mesma técnica que foi utilizada para a busca cega: uma versão iterativa do algoritmo.
- A idéia é o seguinte:
  - Calculamos o valor f(n) do estado inicial. Como a solução ótima não pode ter um custo menor, realizaremos uma busca em profundidade na qual serão expandidos somente os nodos que nãoultrapassam esse limite.
  - Se, de todos os nodos visitados, nenhum é a solução, recomeçamos uma nova busca, aumentando o limite.
  - Qual será o melhor valor para o novo limite? Considere os nodos que não foram expandidos. Um deles minimiza o valor f(n). Como a solução ótima não pode ter um valor menor, recomeçaremos com esse novo valor. Assim por diante até que o estado final seja visitado.

Busca IDA* (A estrela iterativo)

- Exemplo: G é o estado inicial e K é o estado final. O número entre parênteses representa h(n) para cada estado.



Busca Subida da Encosta (Hill climbing)

- É a estratégia mais simples e popular. Baseada na Busca em Profundidade.
- É um método de busca local que usa a idéia de que o objetivo deve ser atingido com o menor número de passos.
- A idéia heurística que lhe dá suporte é a de que o número de passos para atingir um objetivo é inversamente proporcional ao tamanho destes passos.
- Empregando uma ordenação total ou parcial do conjunto de estados, é possível dizer se um estado sucessor leva para mais perto ou para mais longe da solução. Assim o algoritmo de busca pode preferir explorar em primeiro lugar os estados que levam para mais perto da solução.

Busca Subida da Encosta (Hill climbing)

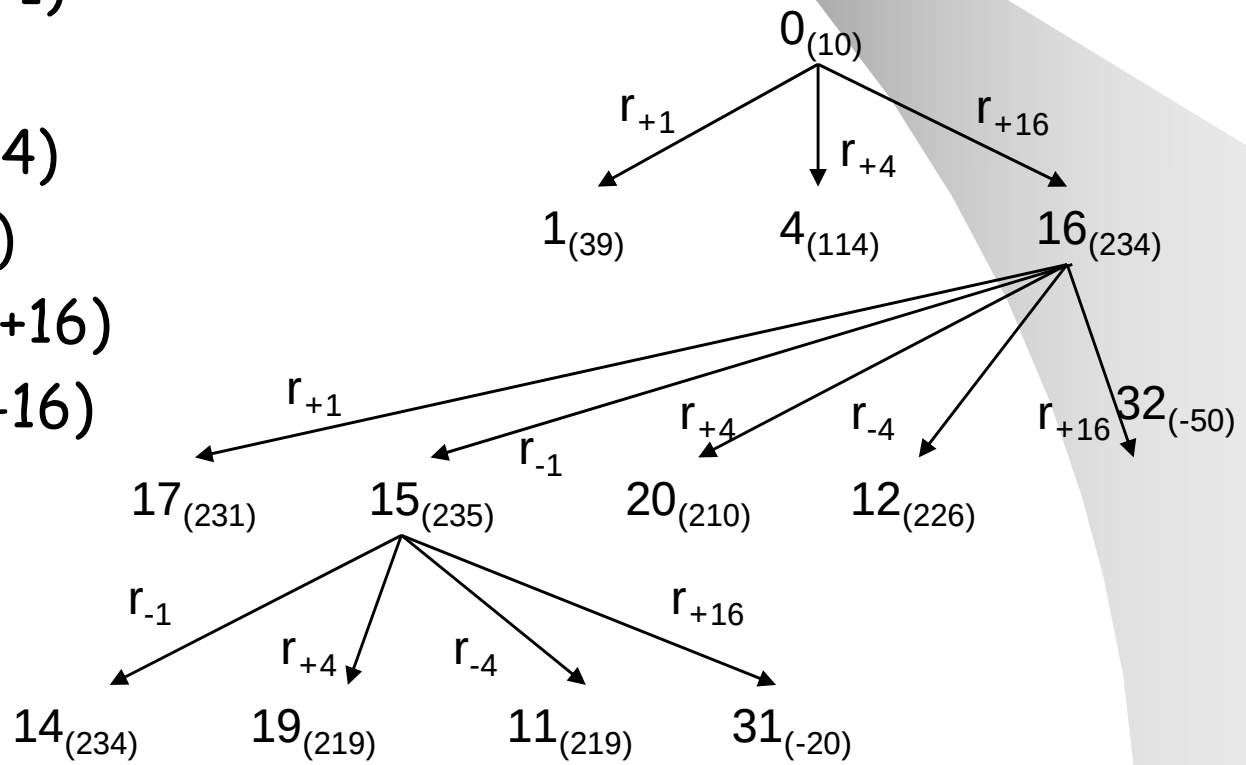
- Há duas variações do método:
  - **SUBIDA DE ENCOSTA SIMPLES:** Vai examinando os sucessores do estado atual e segue para o primeiro estado que for maior que o atual.
  - **SUBIDA DE ENCOSTA PELA TRILHA MAIS ÍNGREME:** Examina TODOS os sucessores do estado atual e escolhe entre estes sucessores qual é o que está mais próximo da solução.
- Este método não assegura que se atinja o ponto mais alto da montanha.
- Ele assegura somente que atingido um ponto mais alto do que seus vizinhos, então encontramos uma boa solução local.

Busca Subida da Encosta (Hill climbing)

- Algoritmo (supondo que temos uma função f(n) que queremos maximizar):
  1. Identificar o estado atual com o estado inicial: n_{atual} = *estado inicial. Em certas aplicações, o* estado inicial pode ser gerado aleatoriamente.
  2. Identificar todos os estados sucessores possíveis de n_{atual} *e calcular, para cada estado sucessor n, o* valor *f(n). Seja* n_i *o estado sucessor que tem o maior valor.*
  3. Se $f(ni) < n_{atual}$, *retornar* n_{atual}. *Isso significa que o algoritmo encontrou um estado que maximiza* a função *f(n).*
  4. Senão, $n_{atual} = n_i$ *e voltar à etapa 2.*

Busca Subida da Encosta (Hill climbing)

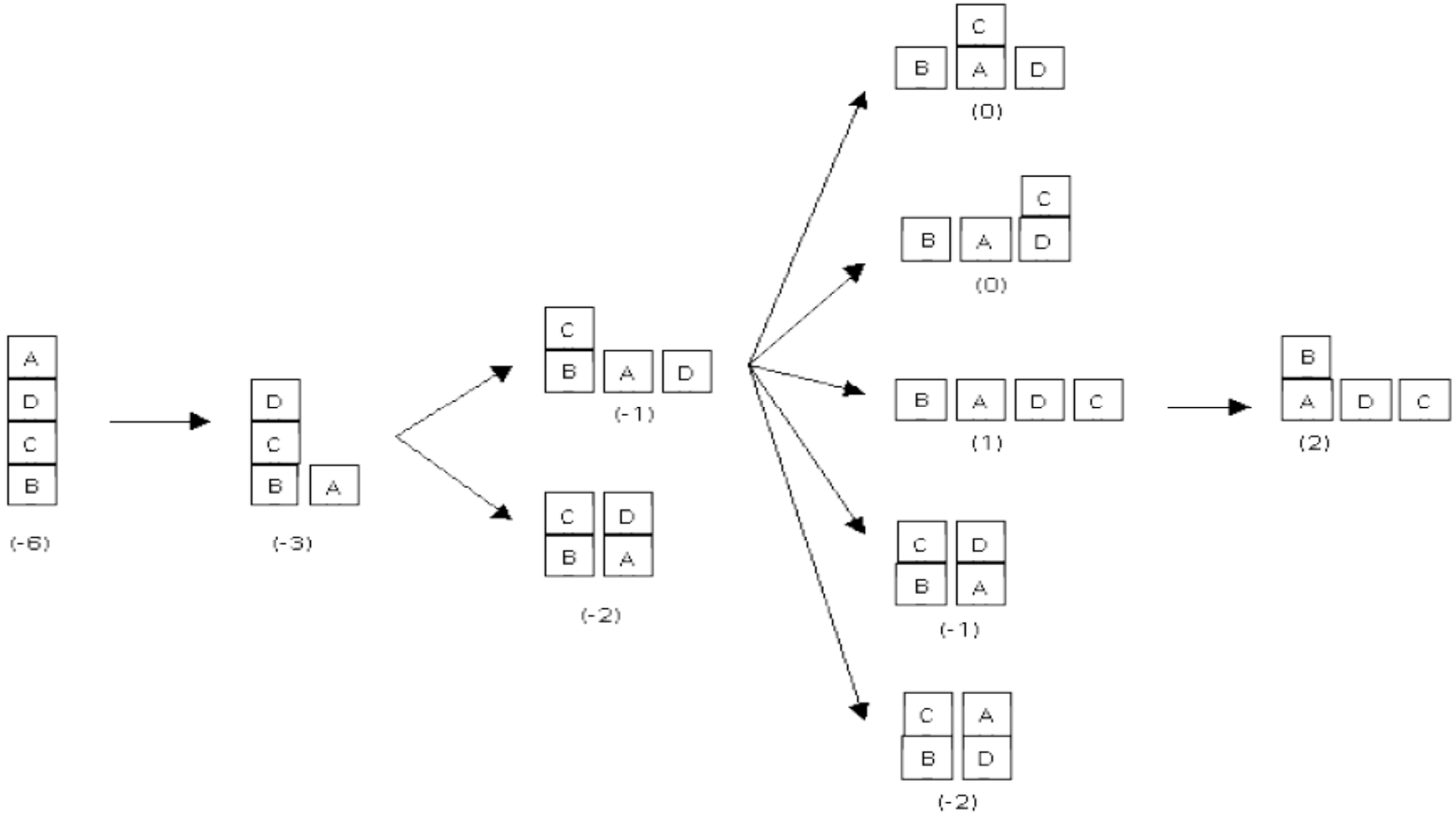
- Exemplo
  - Achar o ponto máximo da função $f(x)=-x^2+30x+10$ no intervalo [0,100].
    - r+1=(x|x<100)->(x+1)
    - r-1=(x|x>0)->(x-1)
    - r+4=(x|x<97)->(x+4)
    - r-4=(x|x>3)->(x-4)
    - r+16=(x|x<85)->(x+16)
    - r-16=(x|x>15)->(x-16)



Busca Subida da Encosta (Hill climbing)

- Exemplo: Mundo dos blocos
  - O objetivo é de colocar o bloco D sobre C, C sobre B, B sobre A e A sobre a mesa.
  - Heurística:
    - +n para cada cubo cuja estrutura de n cubo que o suporta é correta
    - -n para cada cubo cuja estrutura de n cubo que o suporta não é correta

Busca Subida da Encosta
(Hill climbing)

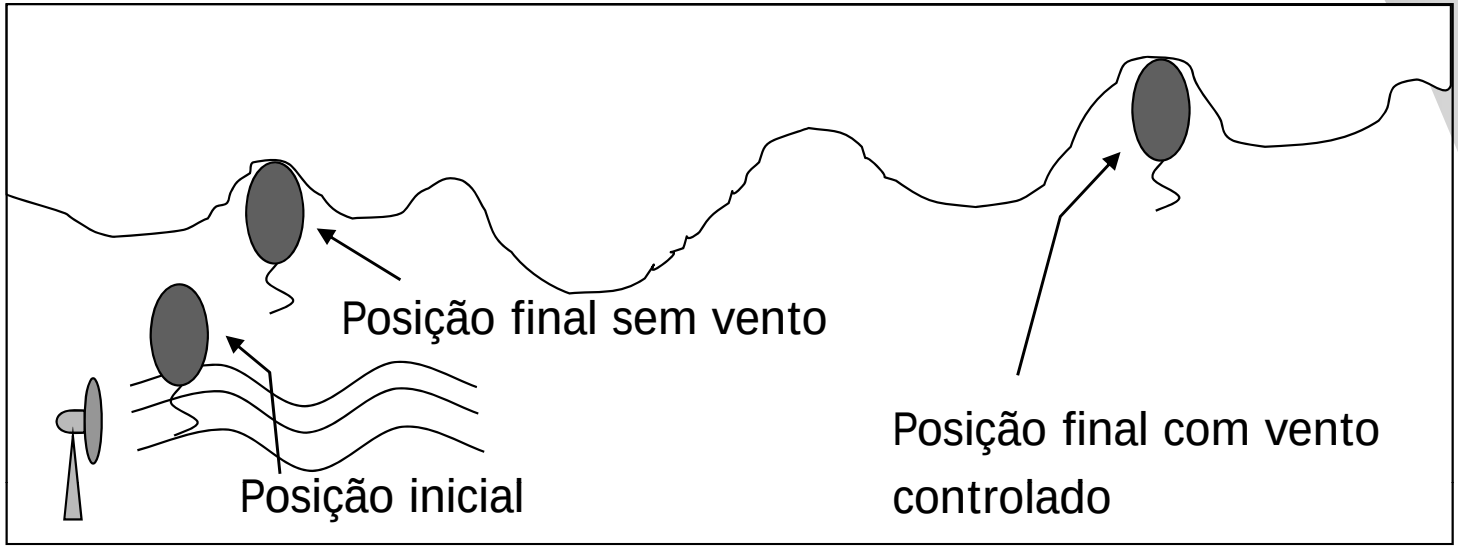


Busca por Têmpera Simulada (Simulated Annealing)

- É adequado a problemas nos quais a subida de encosta encontra muitos platôs e máximos locais.
- Não utiliza backtracking e Não garante que a solução encontrada seja a melhor possível.
- Pode ser utilizado em problemas NP-completos.
- É inspirado no processo de têmpera do aço. Temperaturas são gradativamente abaixadas, até que a estrutura molecular se torne suficientemente uniforme.

Busca por Têmpera Simulada (Simulated Annealing)

- A idéia é permitir "maus movimentos" que com o tempo vão diminuindo de freqüência e intensidade para poder escapar de máximos locais.
- O que o algoritmo de têmpera simulada faz é atribuir uma certa "energia" inicial ao processo de busca, permitindo que, além de subir encostas, o algoritmo seja capaz de descer encostas e percorrer platôs se a energia for suficiente.

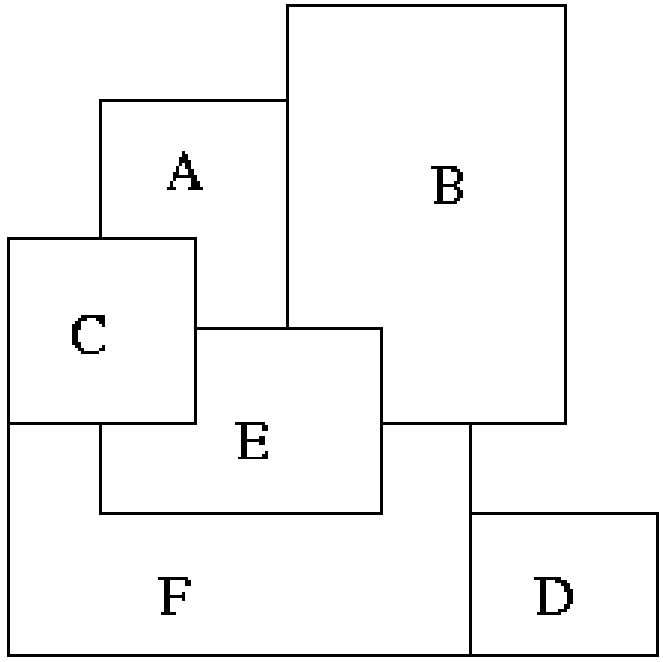


Problemas com Satisfação de Restrições

- Nesse tipo de problema, a sequência de passos para atingir o estado final é na maioria dos casos irrelevante. O que queremos é identificar um valor para cada variável de tal maneira que todas as restrições sejam respeitadas.
- Uma maneira simples de resolver esse tipo de problema é utilizar a técnica Gerar e Testar:
  1. Escolher um valor para cada variável do problema.
  2. Verificar se as restrições são respeitadas.
  3. Se elas são, terminar. Senão voltar à etapa 1.
- A escolhia de cada nova proposta de solução.
  - O método mais naturalconsiste em usar o retrocesso cronológico (backtrack).

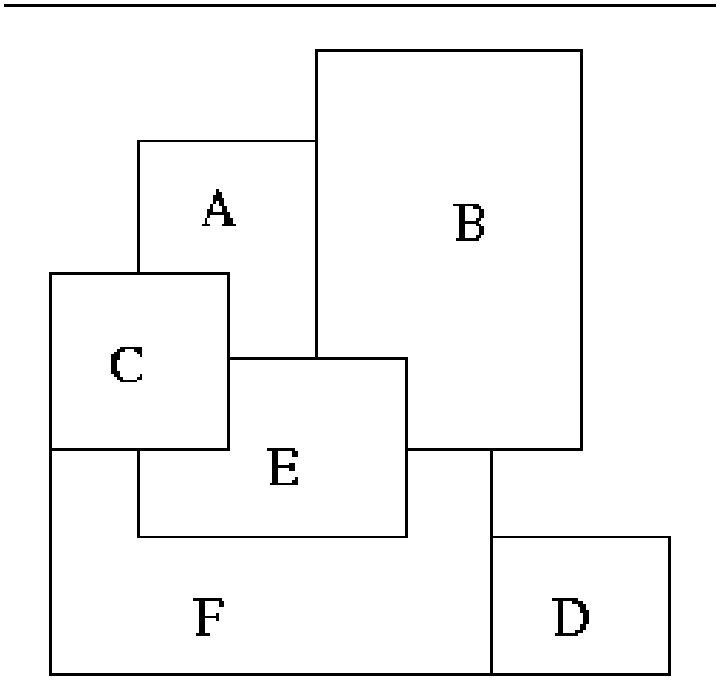
Problemas com Satisfação de Restrições

- Exemplo:
  - temos um mapa a ser colorido minimizando o número de cores utilizadas e de tal maneira que não tenha duas áreas contíguas coloridas com a mesma cor.



Problemas com Satisfação de Restrições

- Exemplo:
  - Esse problema pode ser representado pelo conjunto de variáveis {A, B, C, D, E, F} e as seguintes restrições:
  - A B
  - A C
  - A E
  - B E
  - B F
  - C E
  - C F
  - E F
  - D F



Problemas com Satisfação de Restrições

- Exemplo:
  - Supondo que toda variável tem por domínio o conjunto de cores {azul, vermelho, roxo} e que as instanciações são feitas na sequência A, B, C, D, E e F, a primeira resposta retornada será a seguinte:

    A B C D E F

    1 azul azul azul azul azul azul
  - Essa proposta é rejeitada pelas restrições, voltamos à última variável que tem mais valores a propor, que é a variável F. Obtemos assim uma nova proposta onde F tem o valor vermelho, que é também rejeitada. Mesma coisa com o último valor possível para F:

    A B C D E F

    2 azul azul azul azul azul vermelho

    3 azul azul azul azul azul roxo

Problemas com Satisfação de Restrições

- Exemplo:
  - No próximo backtrack, voltaremos à variável E, pois todas as possibilidades foram esgotadas com a variável F. O próximo valor selecionada para E é a cor vermelho.
  - Agora voltamos a selecionar um valor para F, recomeçando no início do domínio. Obteremos assim mais três possibilidades:

    A B C D E F

    4 azul azul azul azul vermelho azul

    5 azul azul azul azul vermelho vermelho

    6 azul azul azul azul vermelho roxo
  - Continuando, obteremos uma solução depois de 124 tentativas:

Problemas com Satisfação de Restrições

- Exemplo:
  - A B C D E F

  7 azul azul azul azul roxo azul

  8 azul azul azul azul roxo vermelho

  9 azul azul azul azul roxo roxo

  10 azul azul azul vermelho azul azul

  ... ...

  123 azul vermelho vermelho vermelho vermelho roxo

  124 azul vermelho vermelho vermelho roxo azul

Problemas com Satisfação de Restrições

- Exemplo:
  - Na maioria dos casos, podemos detectar conflitos antes mesmo de terminar as instanciações. Na linha 1, por exemplo, não é preciso continuar depois de ter atribuído a cor azul a A e B, pois já temos uma restrição violada.

	A	B	C	D	E	F
1	azul					
2	azul	azul	**Backtrack**			
3	azul	vermelho				
4	azul	vermelho	azul	**Backtrack**		
5	azul	vermelho	vermelho			
6	azul	vermelho	vermelho	azul		
7	azul	vermelho	vermelho	azul	azul	**Backtrack**
8	azul	vermelho	vermelho	azul	vermelho	**Backtrack**

Problemas com Satisfação de Restrições

	A	B	C	D	E	F	
9	azul	vermelho	vermelho	azul	roxo		
10	azul	vermelho	vermelho	azul	roxo	azul	**Backtrack**
11	azul	vermelho	vermelho	azul	roxo	vermelho	**Backtrack**
12	azul	vermelho	vermelho	azul	roxo	roxo	**Backtrack**
13	azul	vermelho	vermelho	vermelho			
14	azul	vermelho	vermelho	vermelho	azul		**Backtrack**
15	azul	vermelho	vermelho	vermelho	vermelho		**Backtrack**
16	azul	vermelho	vermelho	vermelho	roxo		
17	azul	vermelho	vermelho	vermelho	roxo	azul	**Solução**

Problemas com Satisfação de Restrições

- Uma outra maneira de melhorar ainda mais a busca é usar o **forward-checking**.
- A idéia é de olhar, considerando os valores já atribuídos e as restrições, se é possível reduzir o domínio das outras variáveis não instanciadas.
- Por exemplo, assim que a cor azul for escolhida para A, podemos excluir essa cor dos conjuntos de B, C e E. Além de limitar as possibilidades na busca, essa técnica tem o efeito desejável de disparar mais rapidamente o backtrack

Problemas com Satisfação de Restrições

	A	B	C	D	E	F	
0	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	
1	a	{v, r}	{v, r}	{a, v, r}	{v, r}	{a, v, r}	
2	a	v	{v, r}	{a, v, r}	{r}	{a, r}	
3	a	v	v	{a, v, r}	{r}	{a, r}	
4	a	v	v	a	{r}	{r}	
5	a	v	v	a	r	{}	Backtrack
6	a	v	v	v	{r}	{a, r}	
7	a	v	v	v	r	{a, r}	
8	a	v	v	v	r	a	

Problemas com Satisfação de Restrições

- No nosso exemplo, há um aspecto que não foi considerado e que tem uma importância considerável na obtenção da soluçao: a ordem de instanciação das variáveis.
- Considere por exemplo a ordem D, A, B, C,E, F. Nesse caso, usando o forward-checking, a busca fica mais demorada e só é encontrada depois de 15 passos.
- Existe uma maneira de determinar a ordem de instanciação das variáveis para otimizar a busca?
- A resposta é sim. Eis as heurísticas para escolha da ordem de instanciação:
  - Escolher a variável mais restringida (a que contém o menor domínio).
  - Escolher a variável implicada em mais restrições.

Problemas com Satisfação de Restrições

Variável mais restringida

	A	B	C	D	E	F	
0	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	{a, v, r}	
1	a	{v, r}	{v, r}	{a, v, r}	{v, r}	{a, v, r}	
2	a	v	{v, r}	{a, v, r}	{r}	{a, r}	
3	a	v	{v}	{a, v, r}	r	{a}	
4	a	v	v	{a, v, r}	r	{a}	
5	a	v	v	{v, r}	r	a	
6	a	v	v	v	r	a	Solução

Problemas com Satisfação de Restrições

Variável envolvida em maior número de restrições

	A	B	C	D	E	F	
0	{a, v,r}	{a, v,r}	{a, v,r}	{a, v,r}	{a, v,r}	{a, v,r}	
1	{v, r}	{v, r}	{v, r}	{a, v,r}	a	{v, r}	
2	{v, r}	{r}	{r}	{a, r}	a	v	
3	v	{r}	{r}	{a, r}	a	v	
4	v	r	{r}	{a, r}	a	v	
5	v	r	r	{a, r}	a	v	
6	v	r	r	a	a	v	Solução

Jogos

- Os jogos tem atraído a atenção da humanidade, às vezes de modo alarmante, desde a antiguidade.
- O que o torna atraente para a IA é que é uma abstração da competição (guerra), onde se idealizam mundos em que agentes agem para diminuir o ganho de outros agentes. Além disso, os estados de um jogo são facilmente representáveis (acessíveis) e a quantidade de ações dos agentes é normalmente pequena e bem definida.
- A presença de um oponente torna o problema de decisão mais complicado do que os problemas de busca, pois introduz incertezas, já que não sabemos como o oponente irá agir.
- Geralmente o oponente tentará, na medida do possível, fazer o movimento menos benéfico para o adversário.

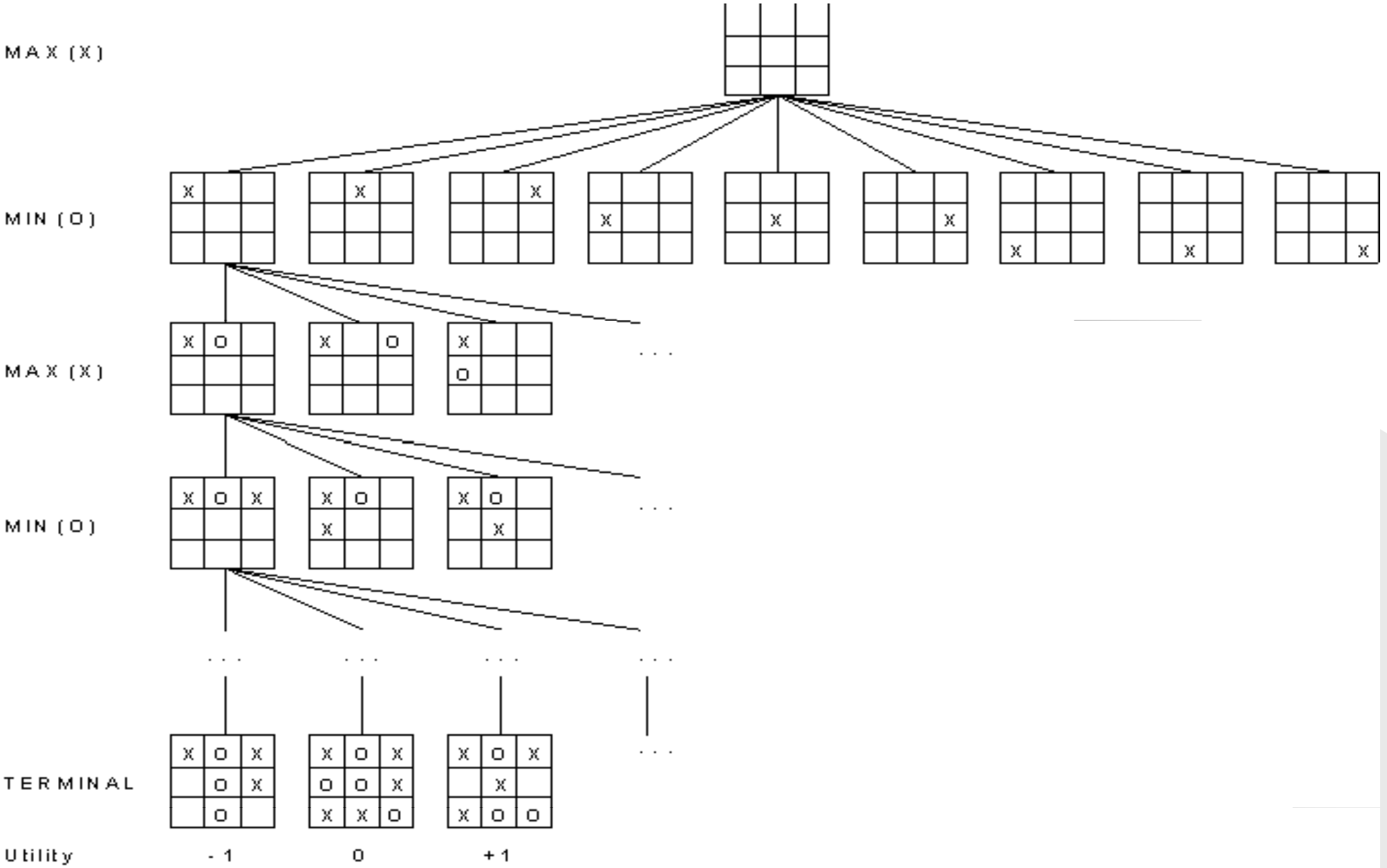
Jogos

- Jogos são, geralmente, problemas muito difíceis de resolver.
  - Xadrez difícil porque muito estados
  - Fator de ramificação 35
  - Geralmente 50 movimentos para cada jogador
  - 35^{100} estados ou nós
- Limites de tempo penalizam a ineficiência;
- Não é possível fazer a busca até o fim, de modo que devemos fazer o melhor possível baseados na experiência passada.
- Deste modo, jogos são muito mais parecidos com problemas do Mundo Real do que os problemas "Clássicos" vistos até agora.

Jogos

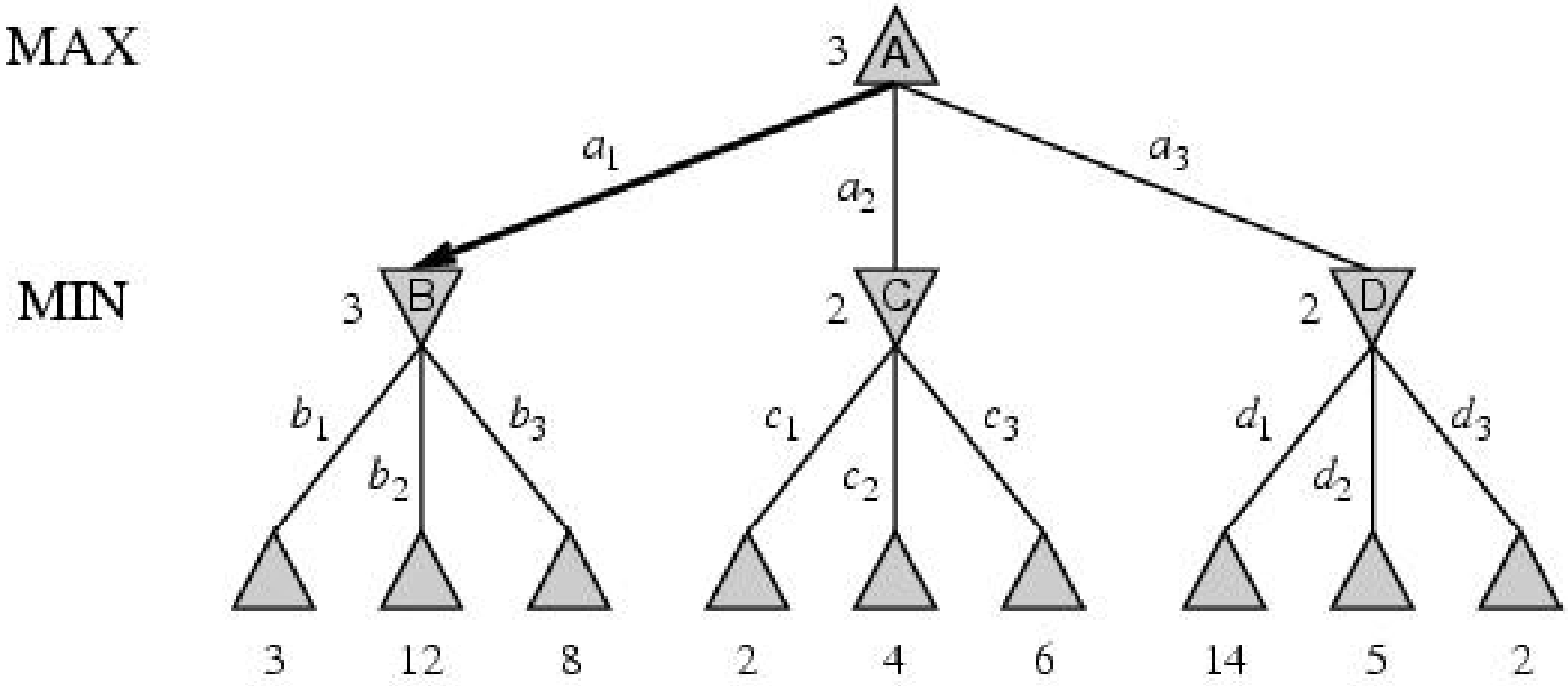
- Jogos de Duas Pessoas
  - Existem dois jogadores: MAX e MIN (MAX começa jogando).
- Jogos como um tipo de problema de busca:
  - O estado inicial;
  - Um conjunto de operadores;
  - Teste de Fim de Jogo (estados finais);
  - Função de Utilidade (payoff) – dá um resultado numérico para o resultado ou conseqüência de um jogo.
- Estratégia
  - Problemas de busca:
    - seqüência de movimentos que levam a um estado meta
  - MIN NÃO DESEJA QUE MAX ganhe;
  - MAX achar estratégia que leve a vitória independentemente dos movimentos de MIN.

Jogos



Jogos

- 1º Exemplo: Algoritmo MINMAX



Jogos

- **1º. Exemplo: Algoritmo MINMAX**
  1. Gerar toda a árvore do jogo;
  2. Aplicar a função utilidade a cada nó terminal;
  3. Usar a utilidade dos nós terminais para determinar a utilidade dos nós um nível acima:
     a) Quando acima é a vez de MIN fazer um movimento, escolher o que levaria para o retorno mínimo
     b) Quando acima é a vez de MAX fazer um movimento, escolher o que levaria para o retorno máximo
  4. Continuar calculando os valores das folhas em direção ao nó raiz;
  5. Eventualmente é alcançado o nó raiz nesse ponto MAX escolhe o movimento que leva ao maior valor.

Jogos

- Decisões Imperfeitas
  - Algoritmo MINIMAX deve procurar até alcançar um nó terminal.
  - Usualmente isso não é prático (complexidade $O(b^m)$
  - Sugestão:
    - Parar a busca num tempo aceitável;
    - A função utilidade é substituída por uma função de avaliação heurística (EVAL), que retorna uma estimativa da utilidade esperada do jogo em uma dada posição;
    - Teste terminal por um teste de corte.

Jogos

- Função de Avaliação
  - Retorna uma estimativa da utilidade do jogo a partir de uma dada posição;
  - EX: xadrez: http://caissa.onenet.net/chess/texts/Shortcut
  - Deve coincidir com a função de utilidade nos nós terminais;
  - Não deve ser difícil de calcular;
  - Deve refletir as chances reais de ganhar.

Jogos

- Poda Alfa-Beta
  - Processo de eliminar ramos da árvore sem examiná-los.
  - MINIMAX é em profundidade
- Eficiência depende da ordem em que os sucessores são examinados.