Inteligência Artificial

Métodos de resolução de problemas

Técnicas de busca

Prof. Paulo Martins Engel

IA e Busca

- Considera-se que o "nascimento" da IA se dá na conferência de Dartmouth (meados de 1956).
- Até então, o esforço para se criar sistemas inteligentes se baseava em redes neurais primitivas.
- A partir daí, a IA seguiu uma abordagem de sistemas simbólicos.
- Na *Tenth Turing Award Lecture*, Allen Newell e Herbert A. Simon discutem a Hipótese do Sistema Simbólico Físico e a idéia de Busca Heurística como meios para realizar inteligência.

A Hipótese do Sistema Simbólico Físico

- "Um sistema simbólico físico tem os meios necessários e suficientes para a ação inteligente".
- Um sistema simbólico físico consiste de um conjunto de entidades, chamadas *símbolos*, que são padrões físicos que podem ocorrer como componentes de um outro tipo de entidade chamada expressão (ou *estrutura simbólica*).
- A estrutura simbólica é composta de um número de ocorrências (*tokens*) de símbolos relacionados entre si de alguma forma (p. ex. um *token* seguido de outro).

O Sistema Simbólico Físico

- Num instante de tempo, o sistema contém uma coleção destas *estruturas simbólicas*.
- Além destas estruturas, o sistema contém uma coleção de *processos* que operam sobre expressões produzindo outras expressões.
- Um sistema simbólico físico é uma máquina que produz uma coleção de estruturas simbólicas que evoluem no tempo.
- *Sistemas simbólicos são coleções de padrões e processos*, os últimos sendo capazes de produzir, destruir e modificar os primeiros.

O Sistema Simbólico Físico

- A propriedade mais importante dos padrões é que eles podem *designar* objetos, processos ou outros padrões e que quando eles *designam processos eles podem ser interpretados…*
- Uma segunda lei da estrutura qualitativa da IA é que *sistemas simbólicos resolvem problemas gerando soluções potenciais e testando-as – isto é, realizando busca.*
- Normalmente as soluções são procuradas criando-se expressões simbólicas e modificando-as seqüencialmente até que elas satisfaçam as condições para serem uma solução.

Resolução de problemas por busca

Durante o *Turing Award Lecture* (1976), Newell e Simon sustentam que a atividade inteligente, quer seja humana ou de uma máquina, é alcançada pelo uso de:

- *Padrões simbólicos* para representar aspectos significativos de um domínio de problema.
- *Operações* sobre estes padrões para gerar soluções potenciais dos problemas.
- *Busca* para selecionar uma solução entre estas possibilidades.

As questões de representação do conhecimento e busca são o núcleo da pesquisa moderna em IA

Resolução de problemas por busca

- Representação do estado (uma configuração do problema)
- Representação das ações: operadores
- Busca: processo de examinar as diversas opções de seqüências de ações possíveis que podem levar ao estado objetivo, escolhendo a melhor seqüência.
- Um algoritmo de busca recebe como entrada um *problema* e retorna uma *solução* na forma de uma seqüência de ações.

Métodos Fracos

- Uma Máquina de Turing, e por extensão um sistema simbólico físico, especifica o que é necessário para *se* escrever um programa que calcule uma saída desejada a partir de uma expressão de entrada.
- Até agora, alguém escreve os programas.
- Um dispositivo verdadeiramente inteligente deveria ser capaz de escrever seus próprios programas.
- A IA tenta resolver o problema de se obter inteligência "sem programação", expandindo o modelo computacional básico definido por uma MT.

Busca Cega

- A busca cega é a estratégia menos inteligente de todas.
- A idéia tem origem no que é conhecido como *Algoritmo do Museu Britânico*: se você puser um chimpanzé ou o que quer que seja na frente de um teclado, então algum dia ele será capaz de gerar todos os livros do Museu Britânico!
- A busca cega é obviamente um procedimento não sistemático e ineficiente.
- Para aumentar a eficiência deve-se acrescentar algum tipo de *estrutura de controle* à geração de alternativas.

Busca em Grafo

- Se representarmos as várias soluções candidatas como nós num grafo, nós podemos visualizar facilmente diferentes tipos de controle.
- Num grafo de espaço de estados, cada nó representa um estado legal.
- Um elo de um nó *N* para um nó *M* denota o fato que a aplicação de um certo gerador (operador) ao estado *N* mapeia este estado para o estado *M*.
- Neste caso, diz-se que *M* é *alcançável diretamente* do estado *N*.

Grafo de Estados

- Podem existir outros estados, além de M, que são diretamente alcançáveis de N.
- O número destes estados é chamado de *fator de ramificação*.
- Graficamente, estas alternativas são representadas como um conjunto de elos indo de N para o conjunto de estados diretamente alcançáveis, $\{M_1, M_2,\dots M_i, \dots M_j\}$.
- Tipicamente, muitos estados diferentes podem ser diretamente alcançados de cada um dos estados M_i.
- Por ex., L_1 pode ser diretamente alcançável de M_1.
- Diz-se que L_1 é alcançável de N (*caminho* de N a L_1)

Busca Sistemática

- Um método de busca sistemático é aquele que organiza eficientemente a geração e a busca dos diversos caminhos representados num grafo de estados.
- Os nós que são diretamente alcançáveis de um outro nó *N* são chamados de nós *filhos* de *N* e o nó *N* é o nó *pai*.
- Se dois nós têm o mesmo pai eles são nós *irmãos*.
- Nós que estão no caminho até um nó *M* são chamados de *ancestrais* de *M* (*M* é *descendente* de um destes nós).
- Um grafo *radicado* tem um único nó (a raiz) do qual se originam todos os caminhos do grafo. (não tem pai)
- Um nó *folha* é um nó terminal, que não tem filhos.

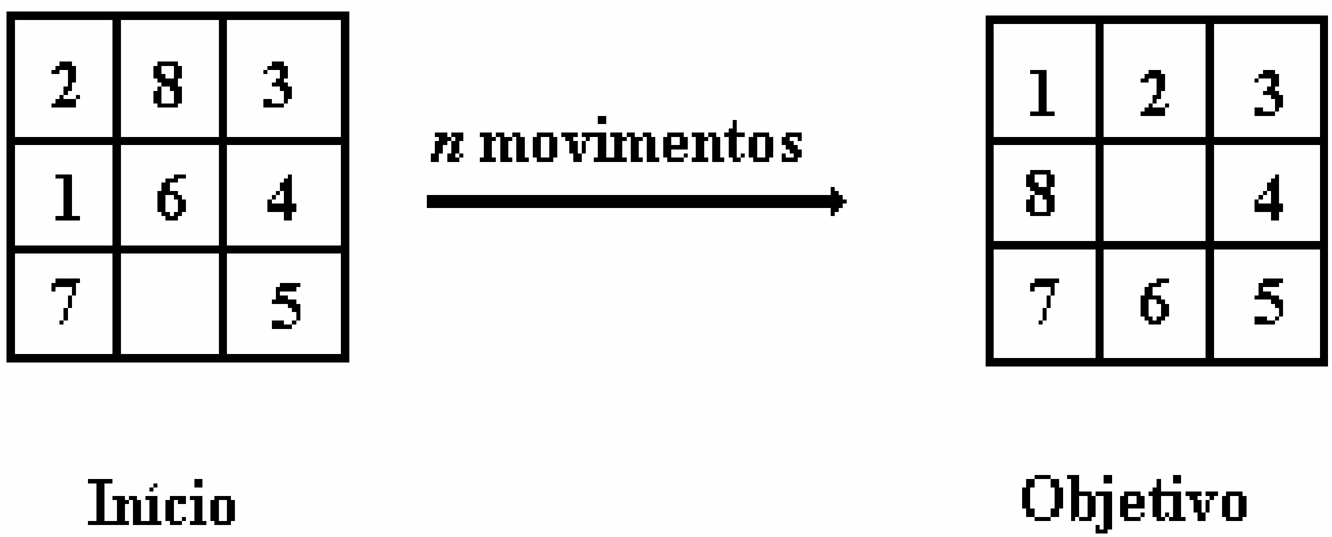
Exemplo de representação por espaço de estados

Exemplo: jogo dos 8

9!/2 = 181.440 estados

2	8	3
1	6	4
7		5

Jogo dos 8



Estados e Operadores

- **Estado**: uma configuração particular das peças
- **Operador**: transforma um estado em outro

A configuração inicial e o objetivo do jogo são os estados inicial e final.

Jogo dos 8: operadores

- mover a peça 1 para cima, baixo, direita, esquerda
- mover a peça 2 para cima, baixo, direita, esquerda
- mover a peça 3 para cima, baixo, direita, esquerda
- ....
- mover a peça 8 para cima, baixo, direita, esquerda

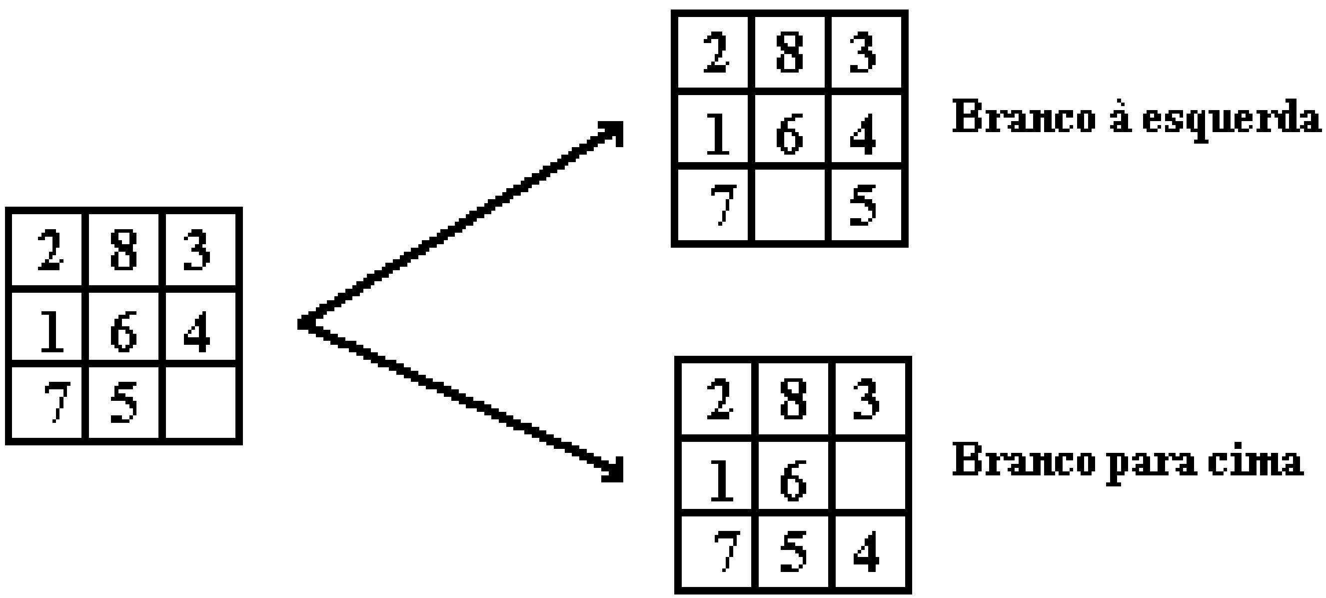
total de 32 operadores

Jogo dos 8: operadores

- branco para cima
- branco para baixo
- branco para a direita
- branco para a esquerda

total de 4 operadores

Jogo dos 8: operadores



Representação do problema

A representação de um problema deve conter:

- forma de representar os estados
- descrição dos estados inicial e objetivo
- descrição dos operadores

Exemplo de representação: listas

- Estado inicial: [2,8,3,1,6,4,7,**0**,5]
- Estado objetivo: [1,2,3,8,**0**,4,7,6,5]
- exemplos de operadores

[a,b,c,d,e,f,g,h,**0**] → [a,b,c,d,e,f,g,**0**,h] p/ esquerda

[a,b,c,d,e,f,g,h,**0**] → [a,b,c,d,e,**0**,g,h,f] p/ cima

total de 24 casos possíveis

[a,b,c,d,e,f,g,h,**0**] → [a,b,c,d,e,f,g,**0**,h] E
[a,b,c,d,e,f,g,h,**0**] → [a,b,c,d,e,**0**,g,h,f] C
[a,b,c,d,e,f,g,**0**,h] → [a,b,c,d,e,f,**0**,g,h] E
[a,b,c,d,e,f,g,**0**,h] → [a,b,c,d,**0**,f,g,e,h] C
[a,b,c,d,e,f,g,**0**,h] → [a,b,c,d,e,f,g,h,**0**] D
[a,b,c,d,e,f,**0**,g,h] → [a,b,c,**0**,e,f,d,g,h] C
[a,b,c,d,e,f,**0**,g,h] → [a,b,c,d,e,f,g,**0**,h] D
[a,b,c,d,e,**0**,f,g,h] → [a,b,c,d,**0**,e,f,g,h] E
[a,b,c,d,e,**0**,f,g,h] → [a,b,**0**,d,e,c,f,g,h] C
[a,b,c,d,e,**0**,f,g,h] → [a,b,c,d,e,h,f,g,**0**] B
[a,b,c,d,**0**,e,f,g,h] → [a,b,c,**0**,d,e,f,g,h] E
[a,b,c,d,**0**,e,f,g,h] → [a,**0**,c,d,b,e,f,g,h] C
[a,b,c,d,**0**,e,f,g,h] → [a,b,c,d,e,**0**,f,g,h] D
[a,b,c,d,**0**,e,f,g,h] → [a,b,c,d,g,e,f,**0**,h] B

[a,b,c,**0**,d,e,f,g,h] → [**0**,b,c,a,d,e,f,g,h] C
[a,b,c,**0**,d,e,f,g,h] → [a,b,c,d,**0**,e,f,g,h] D
[a,b,c,**0**,d,e,f,g,h] → [a,b,c,f,d,e,**0**,g,h] B
[a,b,**0**,c,d,e,f,g,h] → [a,**0**,b,c,d,e,f,g,h] E
[a,b,**0**,c,d,e,f,g,h] → [a,b,e,c,d,**0**,f,g,h] B
[a,**0**,b,c,d,e,f,g,h] → [**0**,a,b,c,d,e,f,g,h] E
[a,**0**,b,c,d,e,f,g,h] → [a,b,**0**,c,d,e,f,g,h] D
[a,**0**,b,c,d,e,f,g,h] → [a,d,b,c,**0**,e,f,g,h] B
[**0**,a,b,c,d,e,f,g,h] → [a,**0**,b,c,d,e,f,g,h] D
[**0**,a,b,c,d,e,f,g,h] → [c,a,b,**0**,d,e,f,g,h] B

Exemplo de representação: matrizes

$\begin{vmatrix} 2 & 8 & 3 \\ 1 & \mathbf{0} & 4 \\ 7 & 6 & 5 \end{vmatrix}$		$\begin{vmatrix} 1 & 2 & 3 \\ 8 & \mathbf{0} & 4 \\ 7 & 6 & 5 \end{vmatrix}$
Estado Inicial		Estado Objetivo

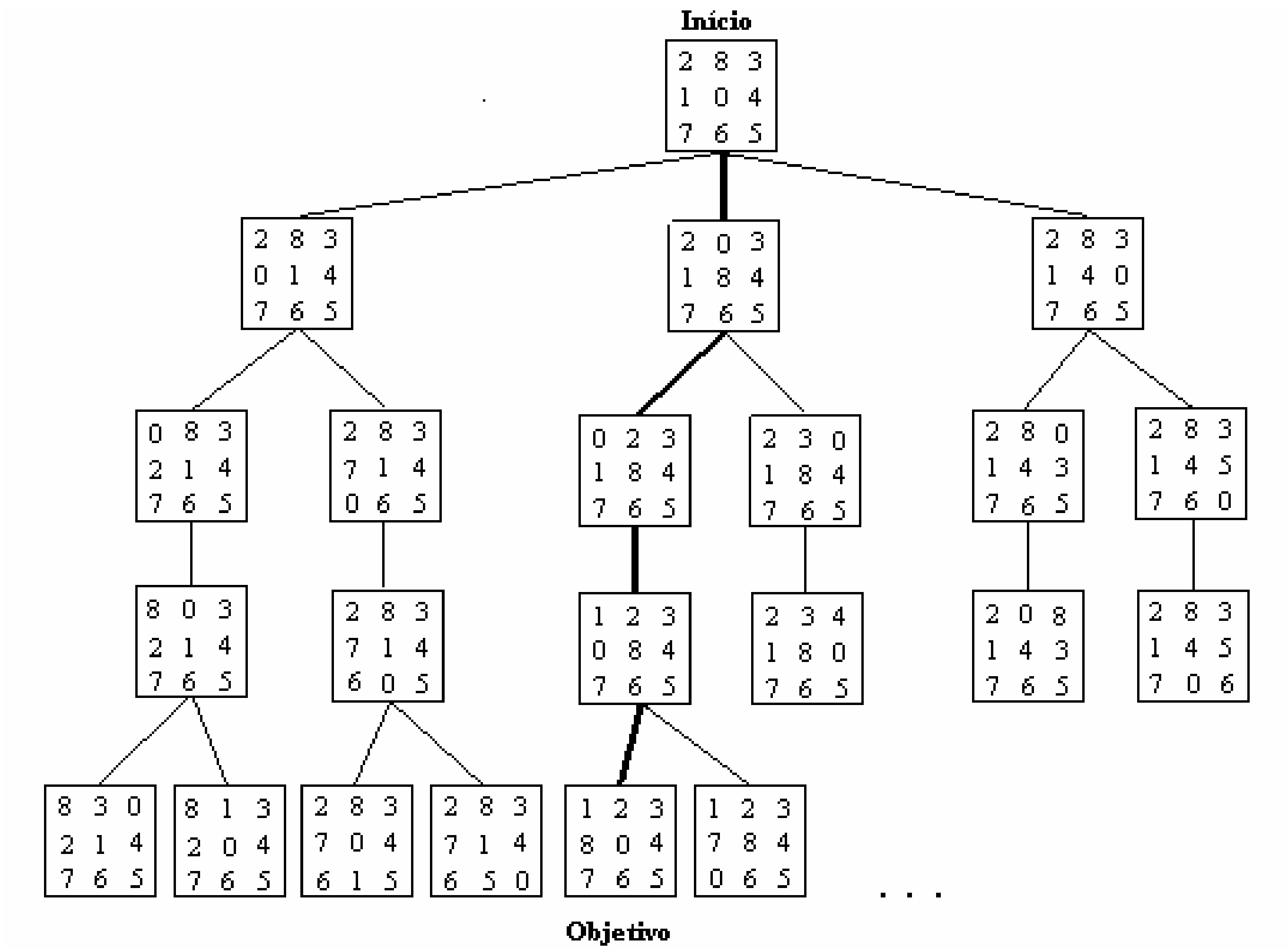
Exemplo de operador:

$$\begin{vmatrix} a & b & \mathbf{0} \\ c & d & e \\ f & g & h \end{vmatrix} \rightarrow \begin{vmatrix} a & \mathbf{0} & b \\ c & d & e \\ f & g & h \end{vmatrix} \quad \text{esquerda}$$

Grafo de estados

- nó: representa um estado
- arco: representa um operador

Grafo de estados: exemplo

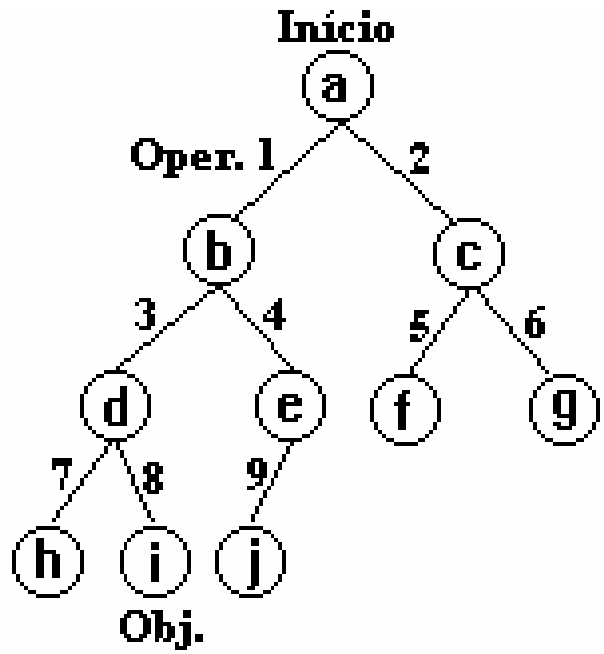


Métodos de busca em grafos de estado

- Estratégias quanto à direção de busca:
  - Percorre-se o grafo até encontrar o estado objetivo (*busca guiada por dados* ou *encadeamento progressivo*).
  - Começar pelo objetivo em direção aos fatos (*busca guiada por objetivo* ou *encadeamento regressivo*).
- Tipos de busca:
  - busca sistemática
  - busca heurística

Busca em largura ou amplitude

- Para cada estado são aplicados todos os operadores possíveis - busca por nível



- Ordem: operadores 1, 2, 3, 4, 5, 6, 7, 8, 9

Busca em amplitude

```
função busca_em_amplitude; (FIFO: fila)
início
  abertos := [Iniciar];                                         %inicialização
  fechados := [ ];
  enquanto abertos ≠ [ ] faça                                   %restam estados
    início
      remova o estado mais a esquerda em abertos, chame-o de X;
        se X for um objetivo então retorne SUCESSO       %objetivo encontrado
          senão início
            gere filhos de X;
            coloque X em fechados (na frente);          %põe na pilha
            descarte filhos de X se já estiverem em abertos ou fechados;  %laços?
            coloque os filhos que restam no final à direita de abertos  %pôr na fila
          fim
    fim
  retorne FALHA                                                 %não restam estados
fim.
```

Exemplo de Busca em Amplitude

abertos=[a] fechados=[ ]

x=a, x≠ obj, filhos(a)={b,c}, fechados=[a], abertos= [b,c]

x=b, x≠ obj, filhos(b)={d,e}, fechados=[b,a], abertos= [c,d,e]

x=c, x≠ obj, filhos(c)={f,g}, fechados=[c,b,a], abertos= [d,e,f,g]

x=d, x≠ obj, filhos(d)={h,i}, fechados=[d,c,b,a], abertos= [e,f,g,h,i]

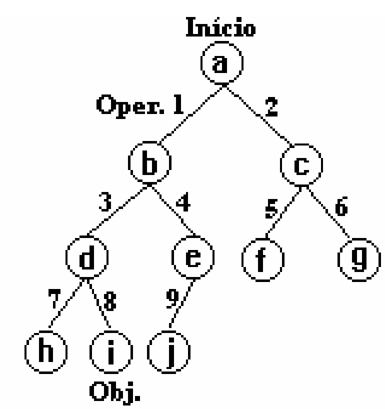
x=e, x≠ obj, filhos(e)={j}, fechados=[e,d,c,b,a], abertos= [f,g,h,i,j]

x=f, x≠ obj, filhos(f)={ }, fechados=[f,e,d,c,b,a], abertos= [g,h,i,j]

x=g, x≠ obj, filhos(g)={ }, fechados=[g,f,e,d,c,b,a], abertos= [h,i,j]

x=h, x≠ obj, filhos(h)={ }, fechados=[h,g,f,e,d,c,b,a], abertos= [i,j]

x=i, x = obj, SUCESSO, fechados=[h,g,f,e,d,c,b,a], abertos= [j]



Admissibilidade

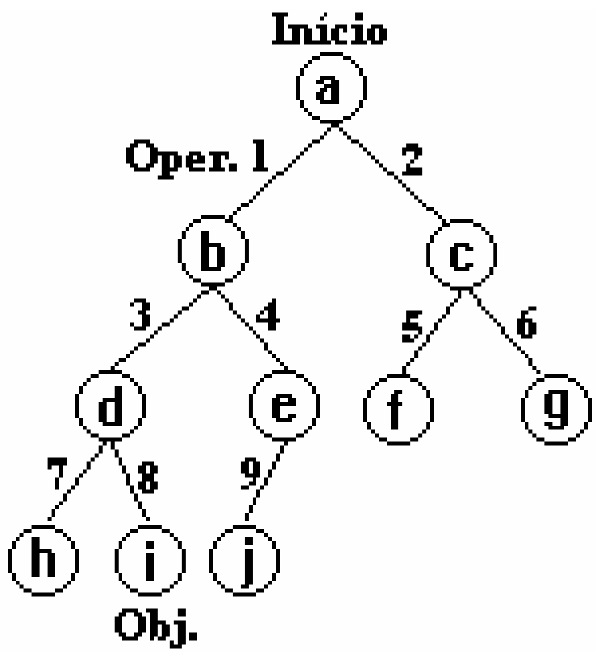
- Como a busca em amplitude examina todos os nós de um nível antes de passar para o próximo nível, ela sempre encontra o *caminho mais curto* para um nó objetivo.
- Um algoritmo de busca é *admissível* se houver a garantia de encontrar um caminho mínimo até uma solução sempre que tal solução exista.
- A busca em amplitude é um algoritmo admissível.
- Entretanto, se houver um fator de ramificação desfavorável (média alta de estados descendentes, **B**), a explosão combinatória pode impedir que o algoritmo encontre uma solução usando o espaço disponível.
- A utilização do espaço da busca em amplitude é uma função exponencial da profundidade **n**: $\mathbf{B}^{\mathbf{n}} \Rightarrow$ *problema em soluções profundas*

Busca em profundidade

- A busca em profundidade avança rapidamente num espaço de busca *profundo*.
- Se soubermos que o caminho-solução será longo, a busca em profundidade não perderá tempo examinando um grande número de estados “superficiais”.
- Por outro lado a busca em profundidade pode se “perder” nas profundezas de um grafo, **não encontrando o caminho mais curto** até um objetivo, ou mesmo ficando presa num caminho infinitamente longo que não leva a um objetivo.
- A busca em profundidade é muito mais eficiente para busca com muitos ramos, porque em cada nível ela retém apenas os filhos de um único estado.
- A utilização de espaço é linear com a profundidade: $\mathbf{B} \times \mathbf{n}$

Busca em Profundidade

Examina-se os nós sempre em direção às folhas, afastando-se da raiz.



Ordem: operadores 1, 3, 7, 8, 4, 9, 2, 5, 6

Busca em profundidade

```
função busca_em_profundidade; (LIFO: pilha)
início
    abertos := [Iniciar];                                   %inicialização
    fechados := [ ];
    enquanto abertos ≠ [ ] faça                             %restam estados
      início
        remova o estado mais a esquerda em abertos, chame-o de X;
        se X for um objetivo então retorne SUCESSO   %objetivo encontrado
        senão início
          gere filhos de X;
          coloque X em fechados;
          descarte filhos de X se já estiverem em abertos ou fechados;  %laços?
          coloque filhos restantes no início à esquerda de abertos   %pôr na pilha
        fim
      fim
    retorne FALHA                                           %não restam estados
fim.
```

Exemplo de Busca em Profundidade

abertos=[a] fechados=[ ]

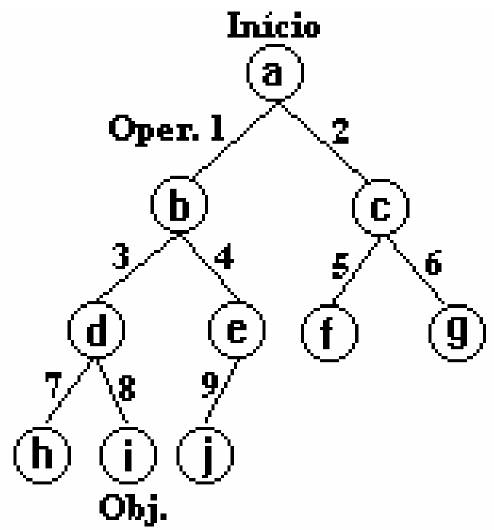
x=a, x≠ obj, filhos(a)={b,c}, fechados=[a], abertos= [b,c]

x=b, x≠ obj, filhos(b)={d,e}, fechados=[b,a], abertos= [d,e,c]

x=d, x≠ obj, filhos(d)={h,i}, fechados=[d,b,a], abertos= [h,i,e,c]

x=h, x≠ obj, filhos(h)={ }, fechados=[h,d,b,a], abertos= [i,e,c]

x=i, x = obj, SUCESSO, fechados=[h,d,b,a], abertos= [e,c]



Busca heurística

- O processo de busca é dirigido através de *informações* que auxiliam a seleção dos operadores
- Heurísticas são formalizadas como regras para escolher aqueles ramos que tem a maior probabilidade de levarem a uma solução aceitável para o problema.
- Função heurística, **h**(**n**): estima a distância entre **n** e o objetivo.
- Para favorecer soluções em caminhos mais curtos, introduz-se um termo **g**(**n**) que mede o comprimento real do caminho de um estado **n** qualquer até o estado inicial.
- Função de avaliação: **f**(**n**) = **g**(**n**) + **h**(**n**)
- Na busca pela melhor escolha, cada estado é rotulado com o seu peso heurístico **f**(**n**).

Busca heurística - exemplo

- exemplo: jogo dos 8

  Estado inicial: **[2,8,3,1,0,4,7,6,5]**

  Estado objetivo: **[1,2,3,8,0,4,7,6,5]**

  Soma das diferenças: 1+6+0+7+0+0+0+0+0 = **14**
- é escolhido o operador que gerar a menor diferença depois de aplicado
- O objetivo é alcançado quando a soma das diferenças for igual a zero.

Busca heurística - exemplo

Estado inicial: **[2,8,3,1,0,4,7,6,5]**

Estado objetivo: **[1,2,3,8,0,4,7,6,5]**

sucessores possíveis soma das diferenças

a) [2,0,3,1,8,4,7,6,5] 1+2+0+7+8+0+0+0+0=18

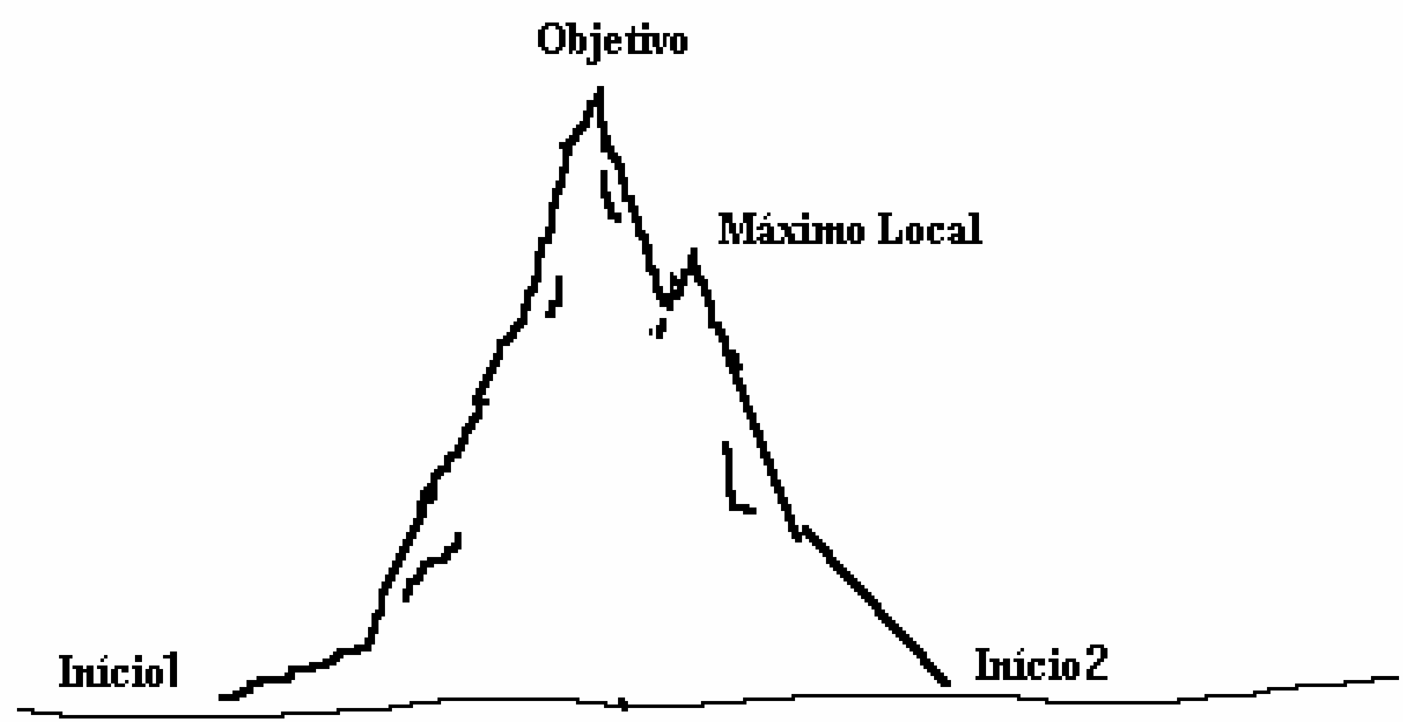
b) [2,8,3,0,1,4,7,6,5] 1+6+0+8+1+0+0+0+0=16

c) [2,8,3,1,4,0,7,6,5] 1+6+0+7+4+4+0+0+0=22

d) [2,8,3,1,6,4,7,0,5] 1+6+0+7+6+0+0+6+0=26

Seria escolhida a jogada b)

Busca heurística



Busca pela melhor escolha

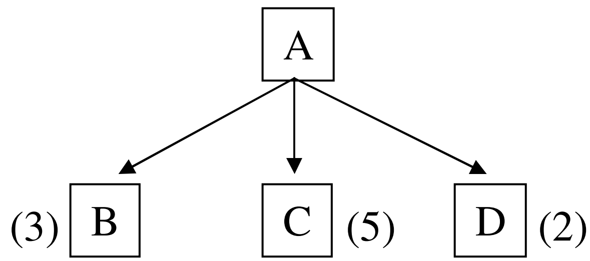
- Busca heurística
- em cada etapa escolhemos o nó mais promissor gerado até o momento
- utiliza uma função de avaliação que retorna o **custo** de se chegar a uma solução (quanto menor melhor)
- o método A* é derivado deste

Buscas Admissíveis

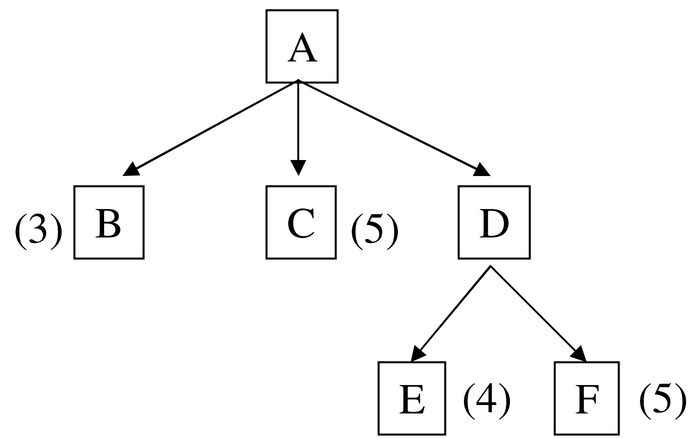
- Definindo a função de avaliação **f*(n) = g*(n) + h*(n)**
  g*(n) custo do caminho mais curto do nó inicial até **n**
  h*(n) custo real do menor caminho até o objetivo.
- Pode-se provar que a busca pela melhor escolha utilizando **f*** é admissível.
- Além disso, se utilizarmos estimativas **g(n)** e **h(n)** para **g*(n)** e **h*(n)**, desde que **g(n)** ≥ **g*(n)** e **h(n)** ≤ **h*(n)**, então a estratégia de busca resultante também é admissível (algoritmo A*).

Exemplo

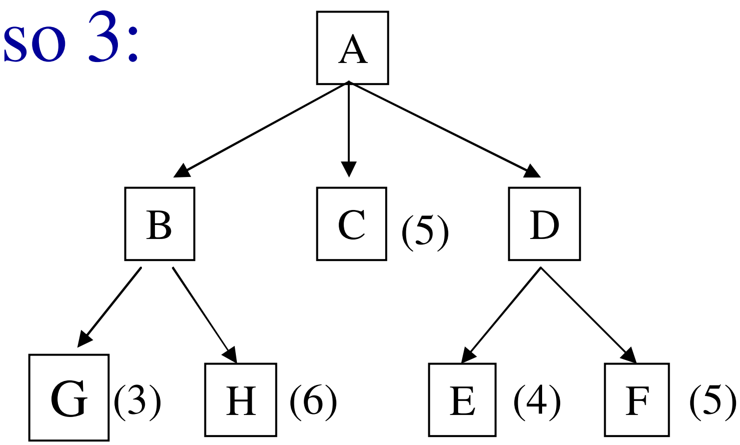
- Passo 1:



- Passo 2:



- Passo 3:



```
função busca_melhor_escolha;
início
    abertos := [Iniciar];                                  %inicialização
    fechados := [ ];
    enquanto abertos ≠ [ ] faça                            %restam estados
      início
        remova o estado mais a esquerda em abertos, chame-o de X;
        se X for um objetivo então retorne caminho de Início até X
        senão início
          gere filhos de X;
          para cada filho de X faça
          caso
            o filho não está em abertos nem em fechados:
              início
                atribua um valor heurístico ao filho;
                acrescente o filho a abertos
              fim
            o filho já está em abertos:
              se o filho foi alcançado por um caminho mais curto
              então atribua ao estado em abertos o caminho mais curto
            o filho já está em fechados
              se o filho foi alcançado por um caminho mais curto então
                início
                  remova o estado de fechados;
                  acrescente o filho em abertos
                fim
            fim                                            % fim do caso
            coloque X em fechados;
            reordene estados em abertos pelo mérito heurístico (melhor mais à esquerda)
        fim                                                %fim do senão
    retorne FALHA                                          %não restam estados
fim.
```

Redução de problema

- O método de resolução por redução de problemas raciocina a partir do problema a ser resolvido, dividindo-o em subproblemas e estes em sub-subproblemas até que o problema original seja reduzido a um conjunto de problemas primitivos de solução imediata.

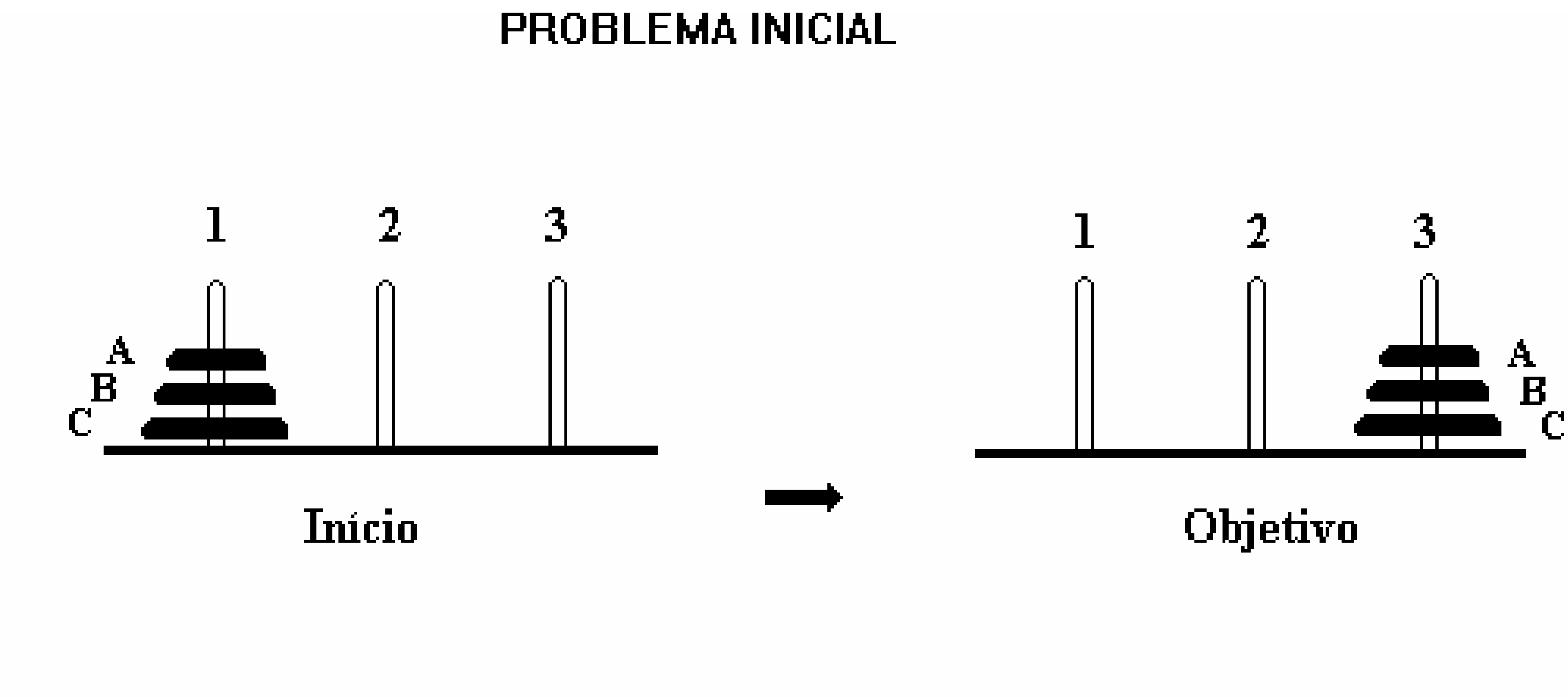
Exemplo: torres de Hanói

Um número **n** qualquer de argolas de tamanhos diferentes são colocadas em três pinos.

O objetivo é transferir todas as argolas do primeiro para o último pino, respeitando as restrições:

- o único movimento possível é movimentar uma única argola de um pino para outro
- apenas a argola de cima pode ser movimentada
- uma argola de maior tamanho não pode ser colocada sobre uma argola menor

Exemplo: torres de Hanói



Exemplo: torres de Hanói

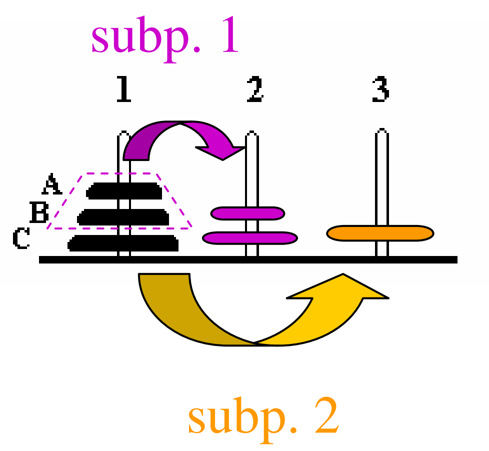
- pode ser resolvido por busca em espaço de estados
- considera-se cada configuração de pinos e argolas como um **estado**
- mas a solução só será aplicável a um número fixo de argolas

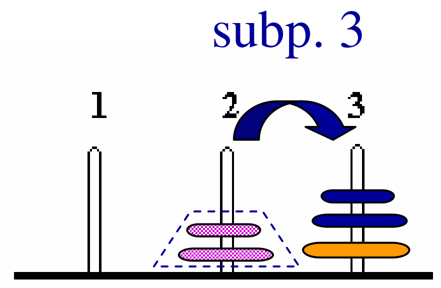
Torres de Hanói

- Através de redução de problemas, têm-se uma solução válida para qualquer número de argolas, com 3 sub-problemas:
  1- mover **n-1** argolas do pino onde estão para o pino não-objetivo
  2- mover uma argola do pino inicial para o pino objetivo
  3- mover n-1 argolas do pino onde estão para o pino objetivo

Torres de Hanói

- Sub-problema 1: passar A e B p/ pino 2
- Sub-problema 2: passar C p/ pino 3
- Sub-problema 3: passar A e B p/ pino 3



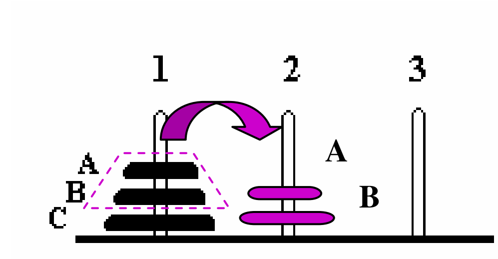


Torres de Hanói

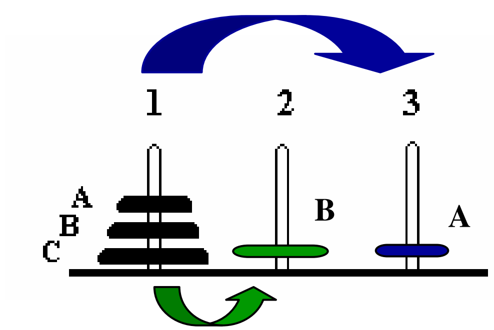
- Subp.1: passar A e B p/ pino 2
  - subp.1: passar A p/ pino 3
  - subp.2: passar B p/ pino 2
  - subp.3: passar A p/ pino 2
- Subp.2: passar C p/ pino 3
- Subp.3: passar A e B p/ pino 3
  - subp.1: passar A p/ pino 1
  - subp.2: passar B p/ pino 3
  - subp.3: passar A p/ pino 3

Torres de Hanói

- Sub-problema 1: passar A e B p/ pino 2
  - subp.1: passar A p/ pino 3
  - subp.2: passar B p/ pino 2
  - subp.3: passar A p/ pino 2

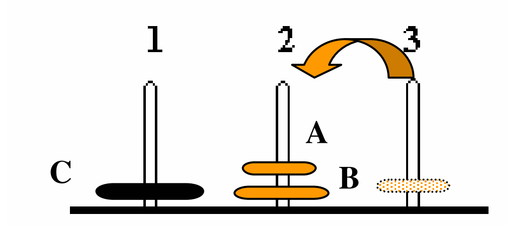


subp. 1

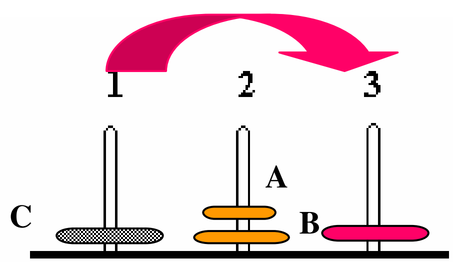


subp. 2

subp. 3

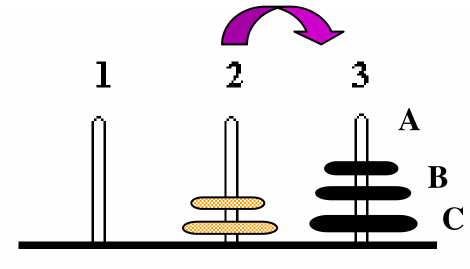


- Sub-problema 2: passar C p/ pino 3

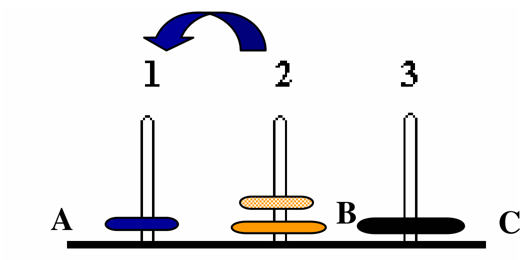


Torres de Hanói

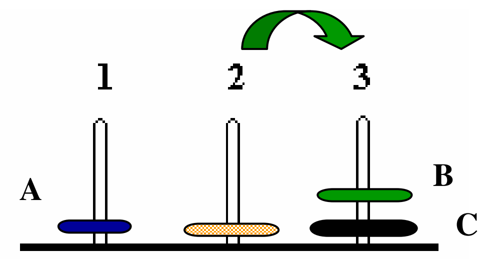
- Sub-problema 3: passar A e B p/ pino 3
  - subp.1: passar A p/ pino 1
  - subp.2: passar B p/ pino 3
  - subp.3: passar A p/ pino 3



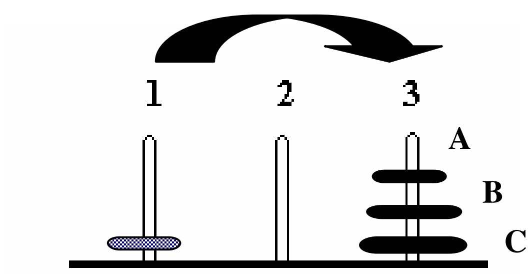
subp. 1



subp. 2



subp. 3



Representação para solução por redução de problemas

- descrição do problema inicial
- operadores de transformação - reduzem um problema a outro(s) mais simples
- descrição dos problemas primitivos - de solução imediata

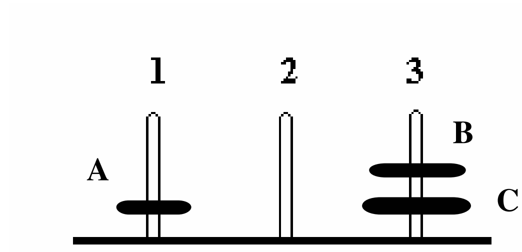
Exemplo de representação

- (i j k) para descrever um estado do jogo, onde
  - **i** representa o pino da argola C (a maior)
  - **j** representa o pino da argola B (a intermediária)
  - **k** representa o pino da argola A (a menor)

O estado **(3 3 1)**, por exemplo, representa a argola **C** no pino 3, **B** no pino 3 (acima de **C**) e **A** no pino 1.

C	B	A	posição: argola
(i	j	k)	número: pino

Exemplo: (3 3 1)
C B A

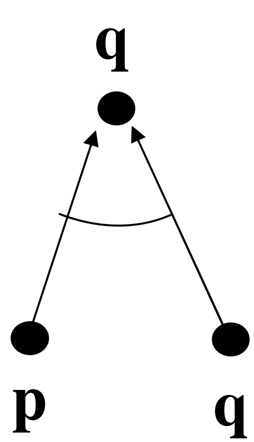


Exemplo de representação

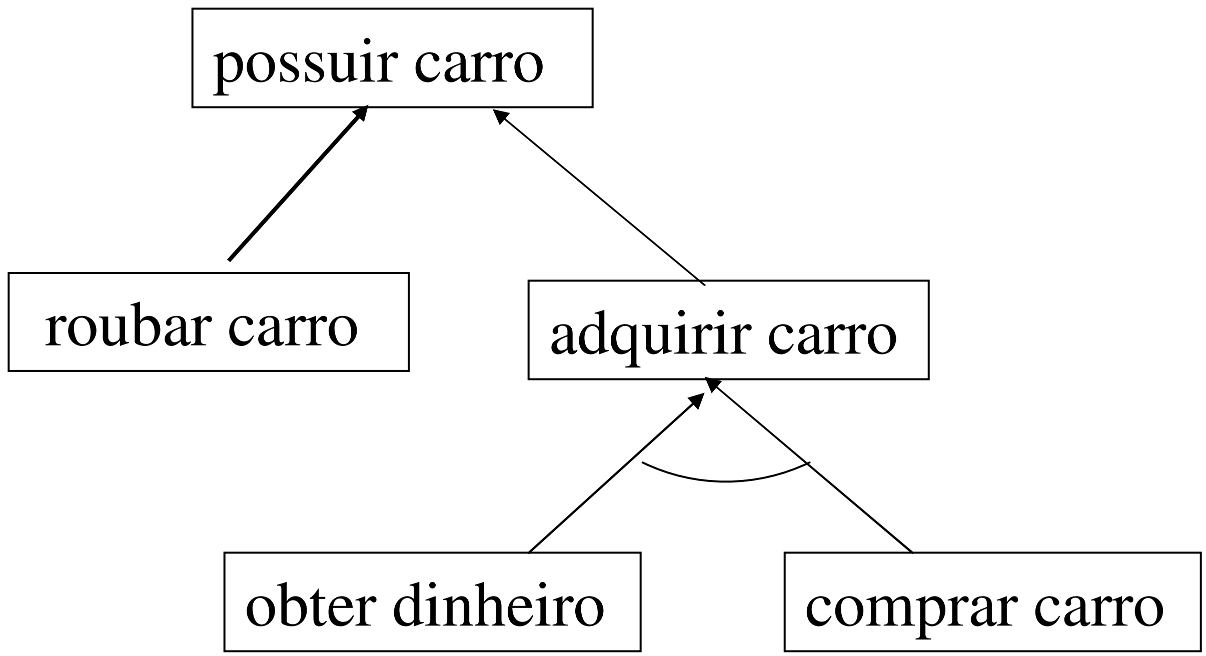
- descrição do problema: (111) $\rightarrow$ (333)
- os três subproblemas:
  - 1. (111) $\rightarrow$ (122)
  - 2. (122) $\rightarrow$ (322)
  - 3. (322) $\rightarrow$ (333)
- problemas primitivos: movimentação de uma argola “livre”

Grafos E/OU

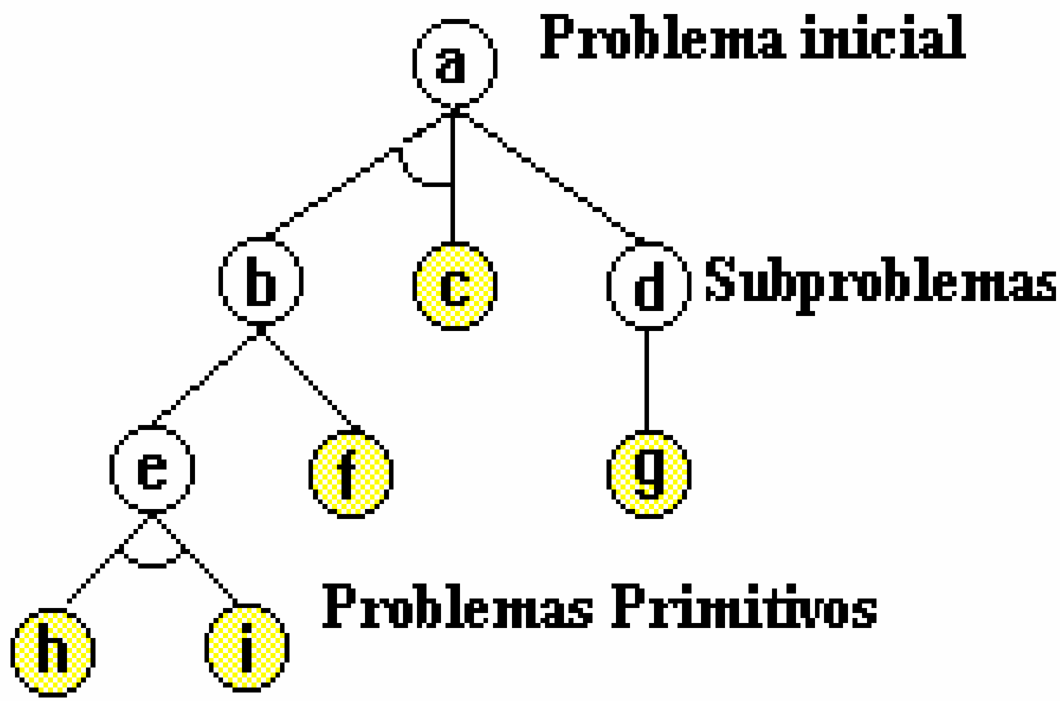
Podemos representar a redução de problemas através de árvores onde cada nó representa um subproblema.



$$p \wedge q \rightarrow p$$



Grafos E/OU

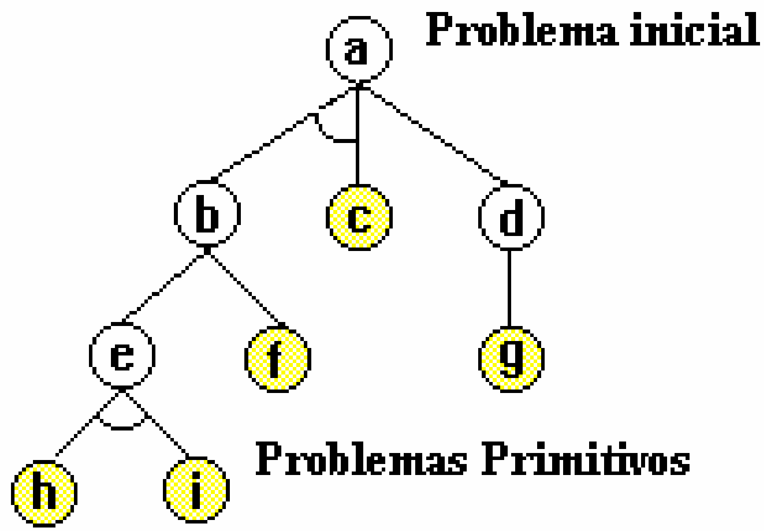

a
Problema inicial
b
c
d
Subproblemas
e
f
g
h
i
Problemas Primitivos

Solução por redução de problemas

- Achar o grafo de solução corresponde a aplicar os operadores que transformam problemas em subproblemas até chegar à solução.
- Nós resolvíveis:
  - os nós terminais (problemas primitivos) são resolvíveis
  - um nó com sucessores **OU** é resolvível se pelo menos um dos sucessores é resolvível
  - um nó com sucessores **E** é resolvível se todos os seus sucessores forem resolvíveis

Métodos de busca por redução de problema

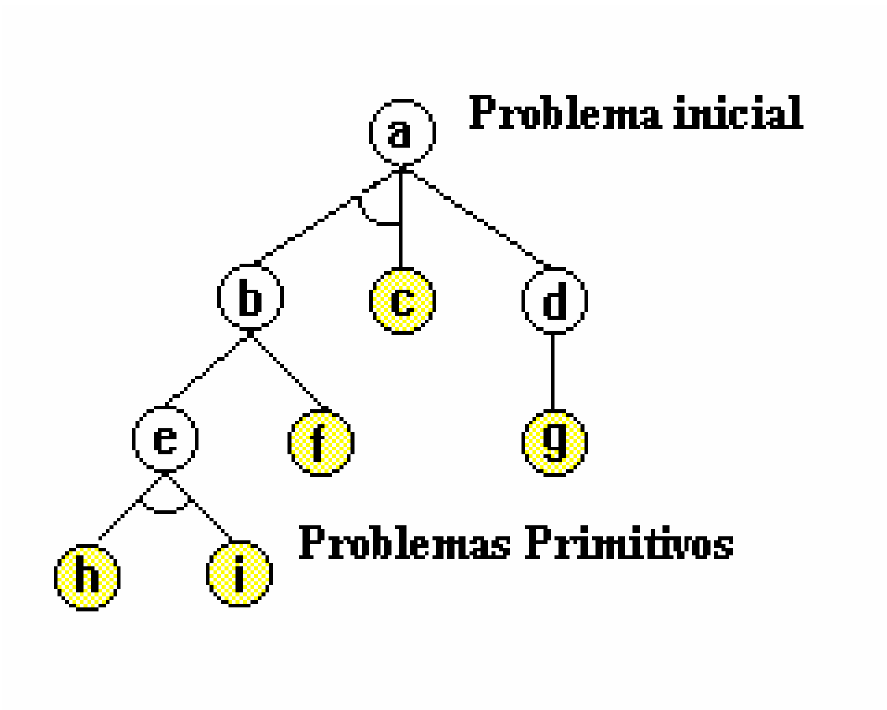
- busca em largura (amplitude)



- ordem: a, b, c, d, e, f, g

Métodos de busca por redução de problema

- busca em profundidade



- ordem: a, b, e h, i, c

Métodos de busca por redução de problema

- busca heurística: os arcos são rotulados com o custo de cada operador de redução utilizado

- Caminhos e custos:

a) a,b,c,e,h,i 2+1+3+1+1=8

b) a,b,c,f 2+1+5=8

c) a,d,g 10+8=18