# Inteligência Artificial
# IA

Prof. João Luís Garcia Rosa

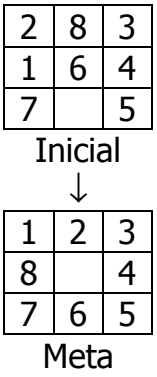
# I. MÉTODOS DE BUSCA

2004 V1.4

## Sistema de Produção

- Sistema de Produção:
  - Base de dados global
  - Regras de produção
  - Estratégia de controle
- Exemplo: tabuleiro de 8 pedras

## Sistema de Produção





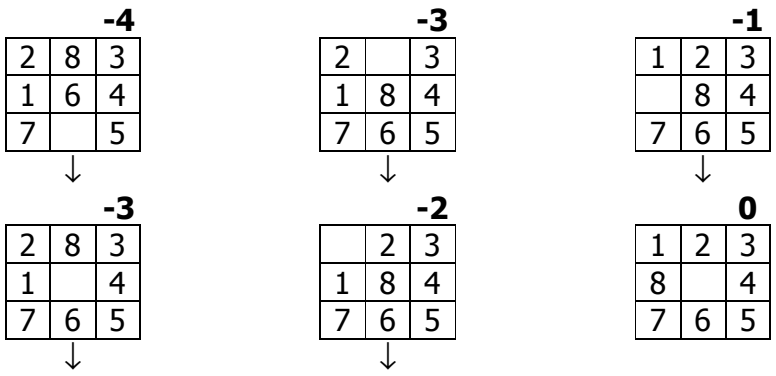

## Sistema de Produção

- Procedimento *PRODUÇÃO*
  - 1 *DADOS* ← base de dados inicial
  - 2 *até DADOS* satisfazer a condição de terminação, faça:
  - 3 *início*
  - 4 selecione alguma regra, R, no conjunto de regras que possa ser aplicada a *DADOS*
  - 5 *DADOS* ← resultado da aplicação de R a *DADOS*
  - 6 *fim*

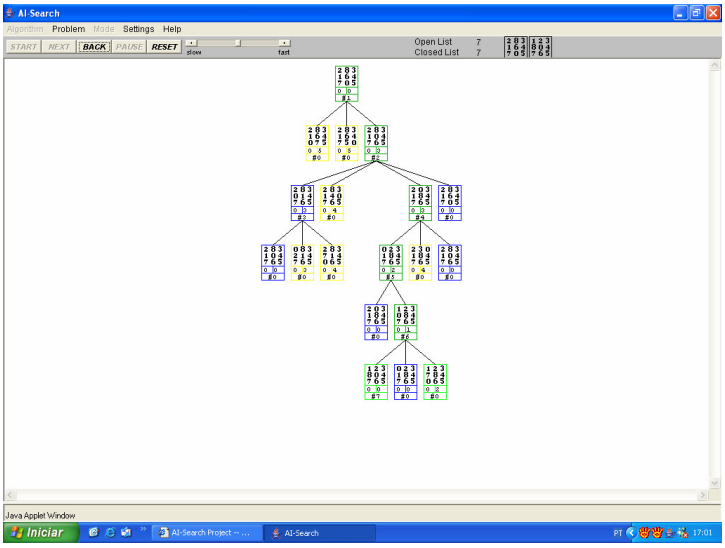
- Controle
  - Passo 4: maior problema
  - Estratégias não-informadas
  - Estratégia informada (heurística)
    - *hill-climbing* para o tabuleiro de 8 pedras







## http://www.cs.rmit.edu.au/AI-Search/Product/





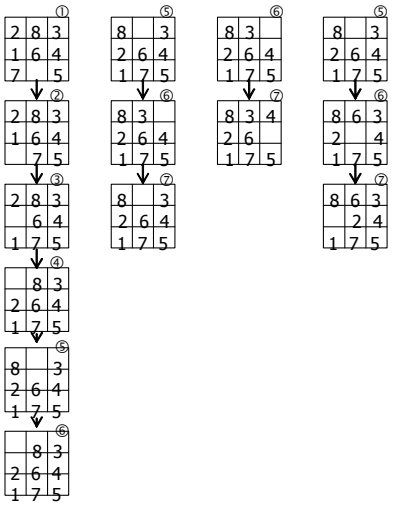

## Sistema de Produção

- Regime de controle:
  - A forma como é conduzido um processo de busca
  - Dois tipos principais:
    - Irrevogável
    - tentativo

- Estratégias de controle
  - Dentro do regime de controle existem as *estratégias de controle*.
  - Dois tipos de estratégia de controle dentro do regime de controle tentativo:
    - *Backtracking*
    - Busca em grafos





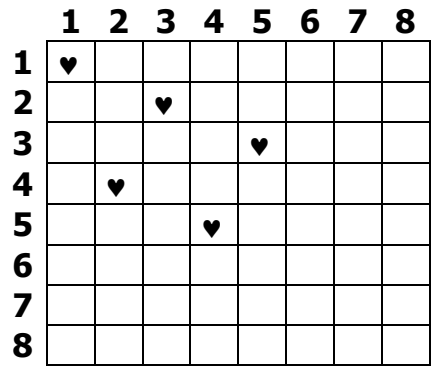


- *Procedimento Recursivo BACKTRACK (DADOS) (Nilsson, 1982)*
  - *1 se TERMO(DADOS), retorne*
  - *2 se DEADEND(DADOS), retorne FALHA*
  - *3 REGRAS ⇐ REGRASAPL(DADOS)*
  - *4 LOOP: se NULL(REGRAS), retorne FALHA*
  - *5 R ⇐ PRIMEIRA(REGRAS)*
  - *6 REGRAS ⇐ CAUDA(REGRAS)*
  - *7 RDADOS ⇐ R(DADOS)*
  - *8 CAMINHO ⇐ BACKTRACK(RDADOS)*
  - *9 se CAMINHO = FALHA, vá para LOOP*
  - *10 retorne CONC(R,CAMINHO)*

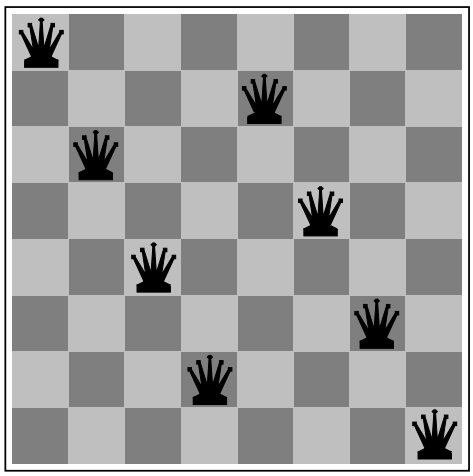




- Problema das 8 rainhas

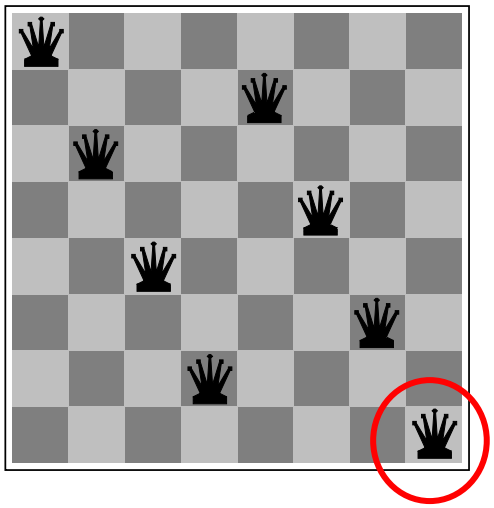


## Problema das 8 rainhas



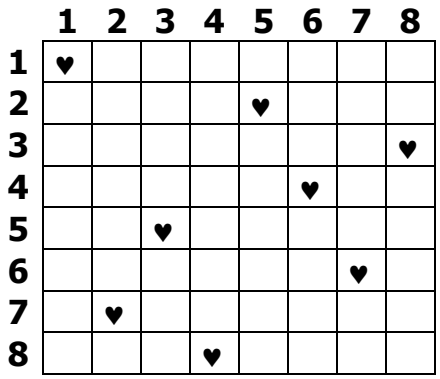


## Problema das 8 rainhas



## Problema das 8 rainhas

Uma solução depois de 91 *backtrackings*:






## Busca em grafos

- Busca em grafos
  - No *backtracking:*
    - o sistema de controle armazena somente o caminho correntemente sendo estendido.
    - Um procedimento mais flexível envolve o armazenamento explícito de todos os caminhos de tal forma que qualquer um deles pode ser candidato a futura extensão: BUSCA EM GRAFOS.

## Busca em grafos

- Notação de grafos:
  - *Grafo:*
    - conjunto (não necessariamente finito) de *nós*.
  - *Arcos direcionados*:
    - Certos pares de nós são conectados por *arcos*, e estes arcos são *direcionados* de um membro do par ao outro (*grafo direcionado).*
  - *Sucessor/pai*:
    - Se um arco é direcionado do nó n*i* para o nó n*j*, então o nó n*j* é o *sucessor* do nó n*i*, e o nó n*i* é o *pai* do nó n*j*.
  - *Árvore:*
    - caso especial de um grafo no qual cada nó tem, no máximo, um pai.
  - *Raiz*:
    - Um nó na árvore que não tem pai é chamado de *nó raiz*.
  - *Folha*:
    - Um nó na árvore que não tem sucessores é chamado de *nó folha*.



## Busca em grafos

  - *Profundidade*:
    - Diz-se que o nó raiz é de *profundidade zero*. A profundidade de qualquer outro nó na árvore é, por definição, a profundidade de seus pais mais 1.
  - *Caminho*:
    - Uma seqüência de nós (n*i*1,n*i*2,...,n*ik*), com cada n*ij* um sucessor de n*ij*-1- para *j* = 2,...,*k*, é chamada de um *caminho de comprimento k* do nó n*i*1 para o nó n*ik*. Se um caminho existe entre o nó n*i* para o nó n*j*, então o nó n*j* é acessível a partir do nó n*i*.
  - *Descendente e ancestral*:
    - O nó n*j* é então um *descendente* do nó n*i*, e o nó n*i* é um *ancestral* do nó n*j*.
  - *Custo*:
    - Freqüentemente é conveniente atribuir custos positivos aos arcos, para representar o custo da aplicação da regra correspondente. Usa-se a notação c(n*i*, n*j*) para denotar o custo de um arco direcionado do nó n*i* para o nó n*j*. O custo de um caminho entre dois nós é a soma dos custos de todos os arcos que conectam os nós no caminho. Em alguns problemas, deseja-se achar o caminho que tenha custo *mínimo* entre dois nós.

## Busca em grafos

- Para este modelo, os nós são rotulados por base de dados e os arcos são rotulados por regras.
- Note que o problema de achar uma seqüência de regras para transformar uma base de dados em outra é equivalente ao problema de achar um caminho num grafo.

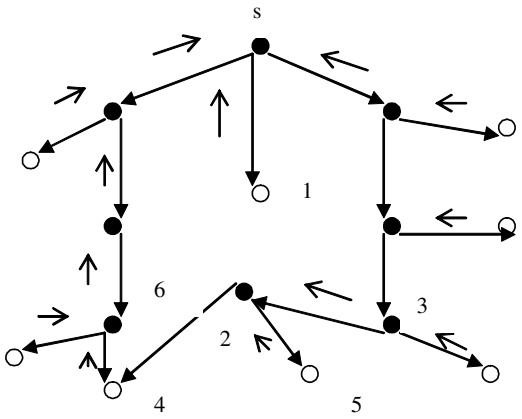
- No tipo mais simples de problema, deseja-se achar um caminho (talvez tendo custo mínimo) entre um nó dado *s*, representando a base de dados inicial e um outro nó dado *t*, representando alguma outra base de dados. A situação mais usual envolve achar um caminho entre um nó s e *qualquer* membro do conjunto de nós {*ti*} que representa as bases de dados que satisfazem a condição terminal. Chama-se o conjunto {*ti*} o conjunto meta e cada nó *t* em {*ti*} é um *nó meta*.



## Busca em grafos

- Procedimento BUSCA-EM-GRAFOS (Nilsson, 1982)
  - 1 Crie um grafo de busca, G, consistindo somente do nó inicial, *s*. Ponha *s* numa lista chamada ABERTOS.
  - 2 Crie uma lista chamada FECHADOS que está inicialmente vazia.
  - 3 LOOP: se ABERTOS está vazia, saia com falha.
  - 4 Selecione o primeiro nó em ABERTOS, remova-o de ABERTOS, o ponha-o em FECHADOS. Chame este nó de *n*.
  - 5 Se *n* é um nó meta, saia com sucesso com a solução obtida traçando um caminho entre os ponteiros de *n* para *s* em G. (Os ponteiros são estabelecidos no passo 7.)
  - 6 Expanda o nó *n*, gerando o conjunto, M, de seus sucessores que não são ancestrais de *n*. Instale estes membros de M como sucessores de *n* em G.
  - 7 Estabeleça um ponteiro para *n* a partir daqueles membros de M que ainda não estejam nem em ABERTOS nem em FECHADOS. Adicione estes membros de M a ABERTOS. Para cada membro de M que já esteja em ABERTOS ou FECHADOS, decida se direciona ou não seu ponteiro para *n*. Para cada membro de M já em FECHADOS, decida para cada um de seus descendentes em G se redireciona ou não o seu ponteiro.
  - 8 Reordene a lista ABERTOS, de acordo com alguma esquema arbitrário ou de acordo com mérito heurístico.
  - 9 Vá para LOOP.

## Busca em grafos






## Estratégias não-informadas de busca em grafos

- Procedimentos não-informados de busca em grafos
  - As estratégias não-informadas usam apenas a informação disponível na definição do problema
    - Busca em largura
    - Busca em profundidade
    - Outras.

## Estratégias não-informadas de busca em grafos

- Estratégias de busca:
  - Uma estratégia é definida através da *ordem da expansão dos nós*
  - Avaliação:
    - Completeza: sempre acha a solução se ela existir?
    - Complexidade temporal: número de nós gerados / expandidos
    - Complexidade espacial: número máximo de nós na memória
    - Otimalidade: sempre acha a solução de custo mínimo?

    - $b$ = fator de ramificação máximo
    - $d$ = profundidade da solução de menor custo
    - $m$ = profundidade máxima do espaço de estados (pode ser $\infty$)



## Estratégias não-informadas de busca em grafos

- Propriedades da Busca em Largura:
  - Completa? Sim (se $b$ é finito)
  - Tempo? $1 + b + b^2 + ... + b^d + b(b^d - 1) = O(b^{d+1})$
  - Espaço? $O(b^{d+1})$ (mantém todo nó na memória)
  - Ótima? Sim (se custo = 1 por passo); não ótima em geral

  - *Espaço* é o maior problema: pode facilmente gerar nós a 10MB/s, tal que 24h = 860GB.

## Estratégias não-informadas de busca em grafos

- Propriedades da Busca em Profundidade:
  - Completa? Não: falha em espaços de profundidade infinita, espaços com loops. Modificar para evitar estados repetidos no caminho → completa em espaços finitos
  - Tempo? $O(b^m)$ terrível se $m$ é muito maior que $d$, mas se soluções são densas, pode ser mais rápido que a busca em largura.
  - Espaço? O($bm$) i. e. espaço linear!
  - Ótima? Não.



## Estratégias heurísticas de busca em grafos

- Procedimentos heurísticos de busca em grafos
  - A palavra *heurística* vem da palavra grega *heuriskein*, que significa "descobrir", que é também a origem de *eureka*, a famosa exclamação de Arquimedes ("Eu achei")

- *Problema do caixeiro viajante:*
  - Um vendedor tem uma lista de cidades, que ele deve visitar apenas uma vez. Existem estradas diretas ligando cada par de cidades da lista. Ache a rota que o vendedor deve seguir para a viagem mais curta possível que começa e termina em uma das cidades.





- Heurística do vizinho mais próximo:
  - 1. Arbitrariamente selecione uma cidade inicial.
  - 2. Para selecionar a próxima cidade, olhe todas as cidades ainda não visitadas e selecione a mais próxima a cidade corrente. Vá a ela.
  - 3. Repita o passo 2 até que todas as cidades tenham sido visitadas.
  - Este procedimento é executado em tempo proporcional a $N^2$, uma melhora significativa sobre $N!$, que seria o tempo gasto caso fosse usada uma estratégia não informada (que geraria uma *explosão combinatorial*).

## Heurísticas

- Argumentos a favor do uso de heurística:
  - Raramente precisamos da solução ótima; uma boa aproximação normalmente serve muito bem. De fato, existe alguma evidência de que as pessoas, quando resolvem problemas, ao invés de serem otimizadoras, elas procuram a satisfação. Em outras palavras, elas procuram qualquer solução que satisfaça algum conjunto de requisitos, e assim que elas encontram um elas terminam.
  - Ainda que as aproximações produzidas por heurísticas possam não ser muito boas no pior caso, os piores casos raramente acontecem na vida real.
  - A tentativa de entender por que uma heurística funciona, ou por que ela não funciona, freqüentemente leva a um entendimento mais aprofundado do problema.



## Métodos de Busca

- Exercício:
  - Especifique uma base de dados global, regras e uma condição de terminação para um sistema de produção para resolver o seguinte problema dos jarros de água:
    - Dado um jarro de 5 litros cheio de água e um jarro de 2 litros vazio, como se pode obter precisamente 1 litro no jarro de 2 litros? A água pode ser desperdiçada ou transferida de um jarro para outro; entretanto, só os 5 litros iniciais estão disponíveis.