Solução de problemas por meio de busca (com Python)

Luis Martí
DEE/PUC-Rio
http://lmarti.com

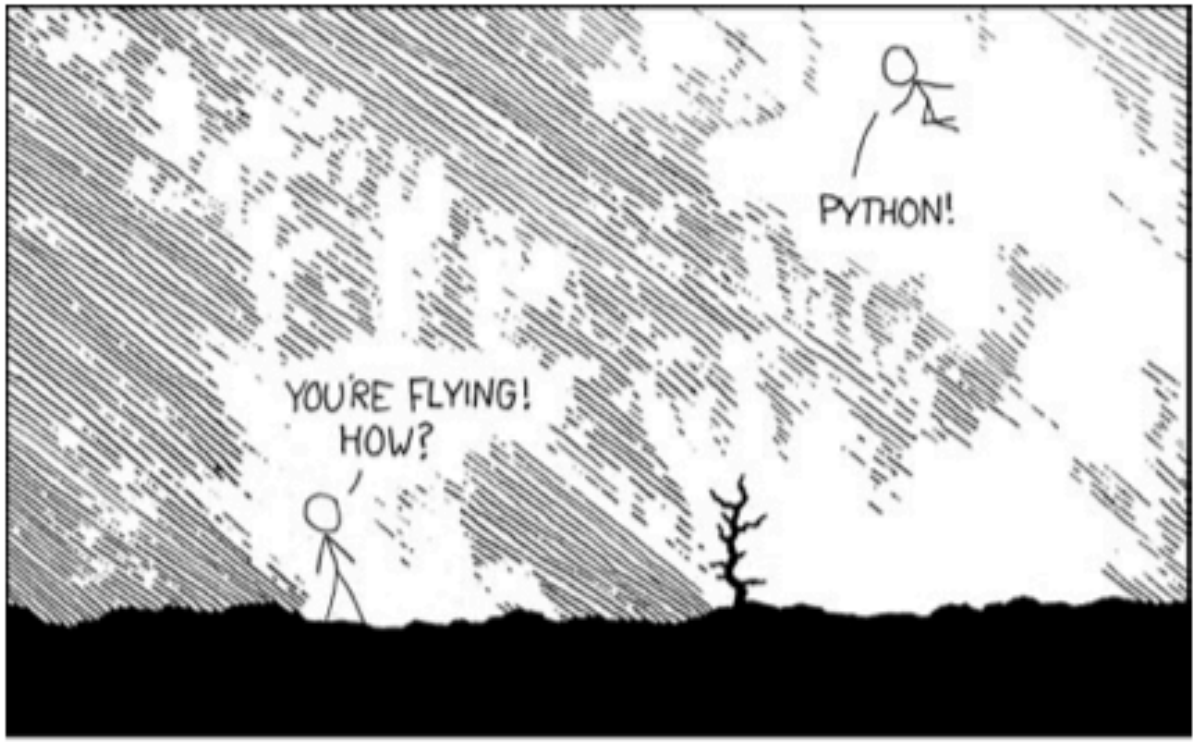
PUC
RIO

Python e AI

- (Re)-introdução ao Python.
- Problemas de busca e principais abordagens.
- Exemplos em Python

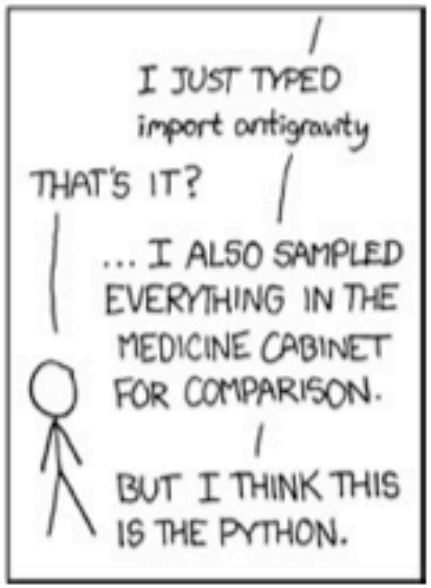
Por que mais uma linguagem?

- Open Source,
- Simplicidade e organização,
- Desempenho,
- Interpretado mas sem código intermeio,
- Programação orientada a objetos,
- Programação funcional,
- Bibliotecas muito completas, etc.


PYTHON!
YOU'RE FLYING! HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
ijustrealized.com


I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?


I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

A única maneira de aprender uma nova linguagem e programar!

Vamos!

Parte II

PROBLEMAS DE BUSCA

Problema: jarros

- Dados uma bica de agua, um jarro de capacidade 3 litros e um jarro de capacidade 4 litros (ambos vazios).
- Como obter 2 litros no jarro de 4?

Solução de problemas por meio de busca

- Desenvolver programas, não com os passos de solução de um problema, mas que produzam estes passos;
- Construir um espaço de estados para encontrar uma sequência de ações cuja aplicação **resolve** um problema;
- Recebe um problema e retorna uma solução

Buscas sem informação

- Algoritmos que não recebem nenhuma informação sobre o problema além de sua formulação.
- É preciso definir
  - problema
  - solução

Formulação de um problema

- Um agente com várias opções imediatas pode decidir o que fazer examinando diferentes sequências de ações possíveis para depois escolher a melhor sequência para executar.
- Formular -> buscar -> executar

Formulação de um problema

- Objetivo
- Estado inicial
- Função sucessor
  - transição de estados (ações)
  - espaço de estados
- Teste de objetivo
- Custo de caminho

Formulação de um problema

- Objetivo:
  - chegar algum lugar: Em(Zerind)
- Estado inicial: ponto de partida
  - ex. partir de Arad: Em(Arad)

- Função transição (sucessor):
  - provê um conjunto de ações possíveis a partir de um determinado estado
  - Sucessor(s) -> {<ação1, sucessor1>, ...

    ,<açãoN, sucessorN>}
  - ex. Dado um mapa da Romênia:

    Sucessor(Em(arad)) ->

    {<Ir(sibiu), Em(sibiu)>, <Ir(timisoara), Em(timisoara)>, <Ir(zerind), Em(zerind)>}

O conjunto de todos os estado acessíveis a partir de um estado inicial é chamado:

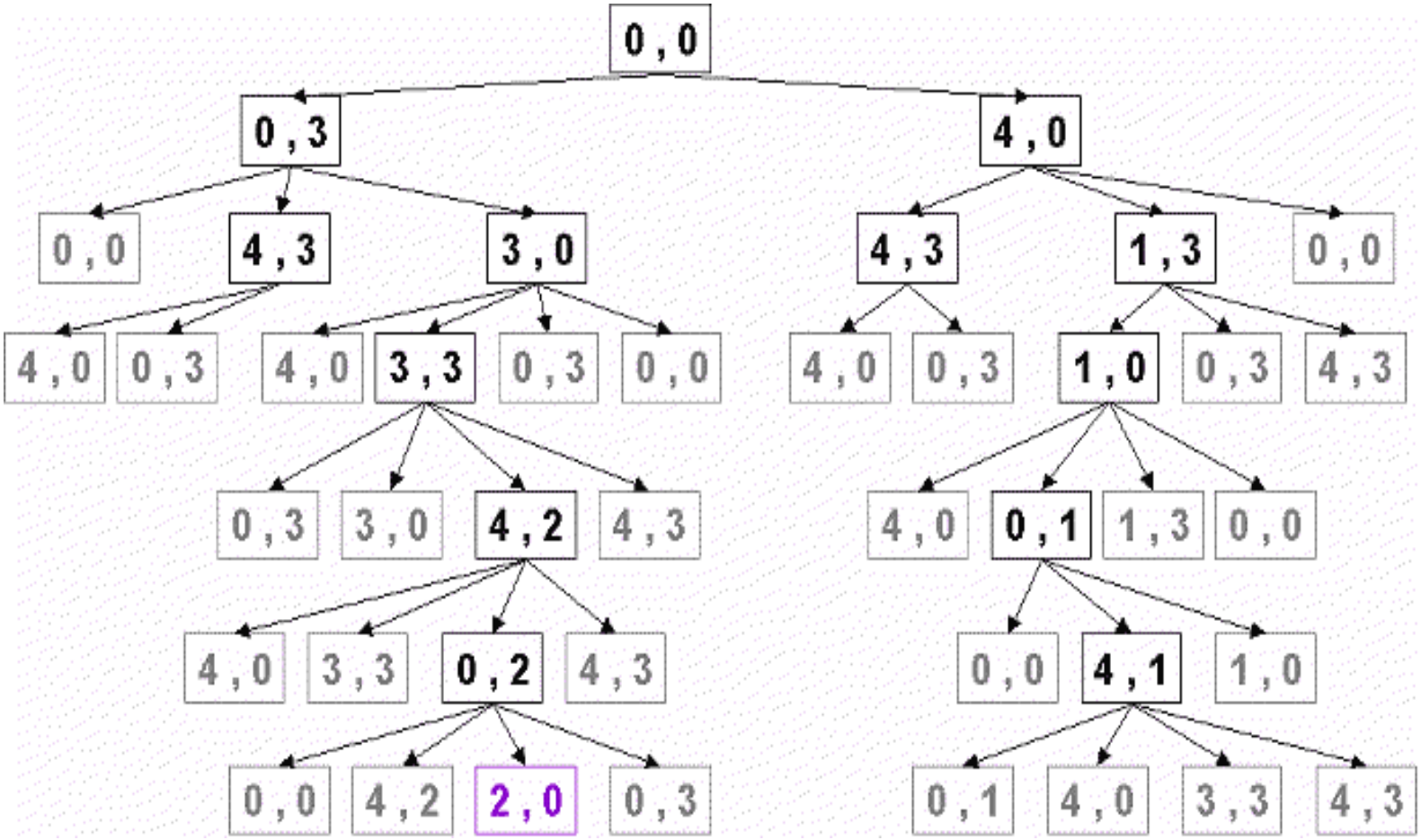
ESPAÇO DE ESTADOS

O espaço de estados pode ser interpretado como um grafo em que os nós são estados e os arcos são ações.

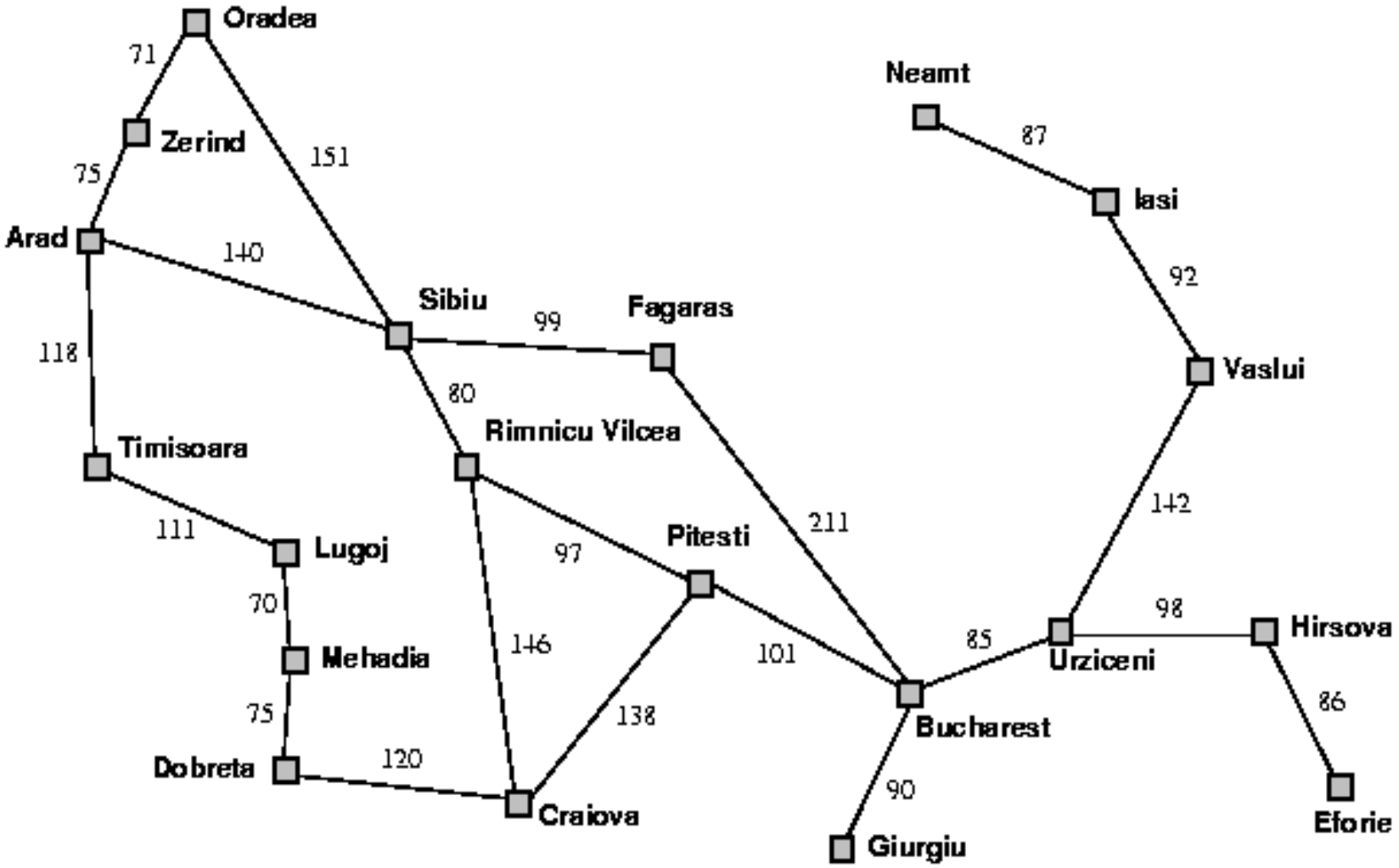
Uma sequência de estados conectados por ações em um espaço de estados chama-se:

UM **CAMINHO** NO ESPAÇO DE ESTADOS

Exemplo da primeira aula (jarros)



Exemplo: viajar pela Romênia



Formulação de um problema

- **Teste de objetivo**: determina se um dado estado gerado no processo de busca é o estado objetivo;
- **Função custo**: atribui um valor numérico para cada caminho

Abstração

- **Abstração *válida***: se qqr solução abstrata pode ser expandida em uma solução mais detalhada;
- **Abstração *útil***: se a execução de cada uma das ações na solução é mais fácil que o problema original.
  - i.e., os passos da solução abstrata não exigem (muito) planejamento adicional

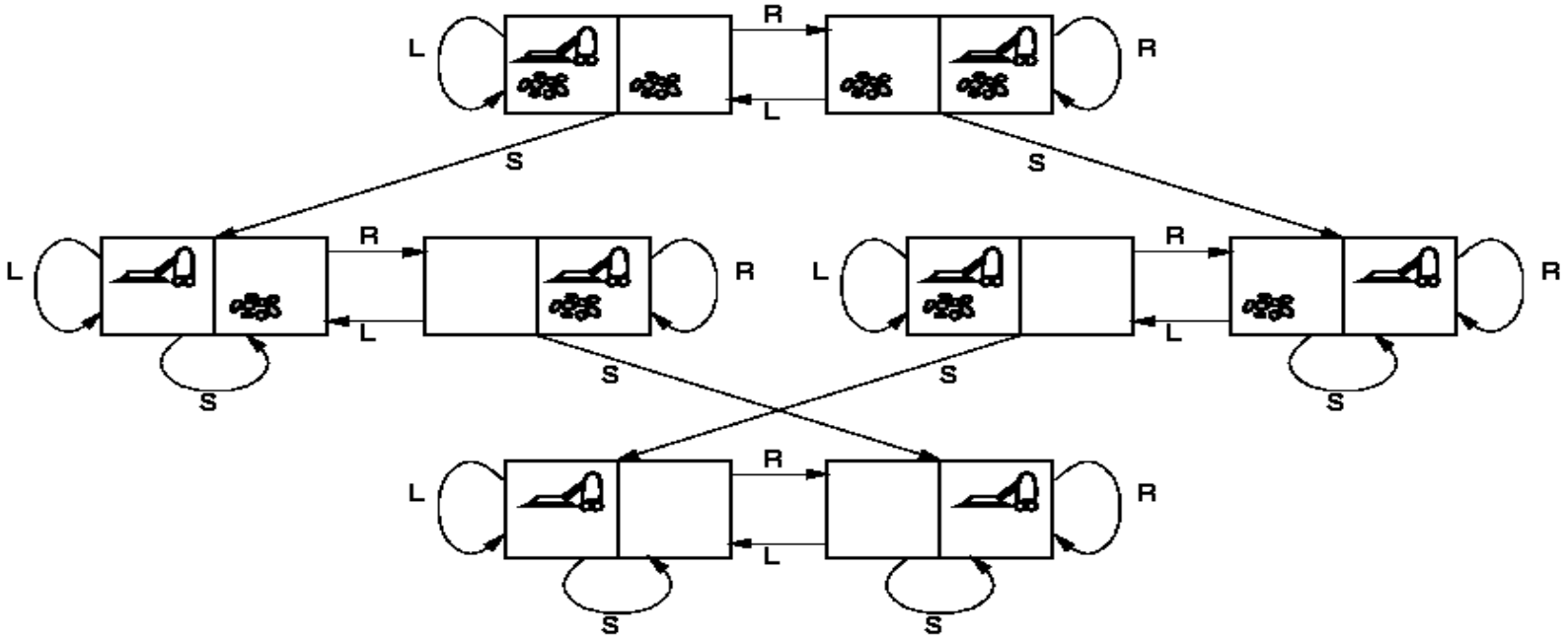
Definição de solução

- **Solução** é um caminho desde o estado inicial até o estado final;
- **Solução ótima**: tem o menor custo de caminho dentre todas as soluções existentes.

Exemplos de problemas

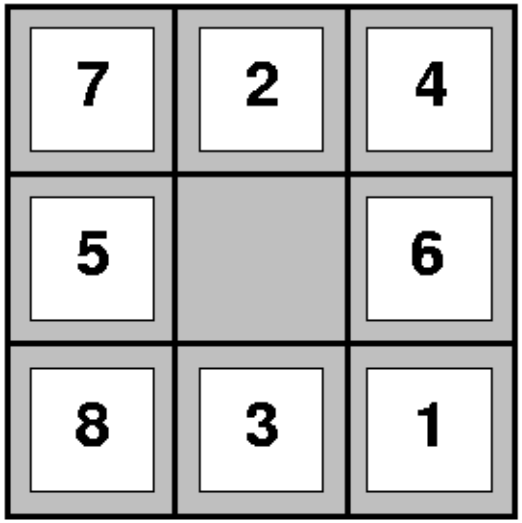
- Miniproblemas (micro-worlds):
  - descrição concisa e exata
  - ilustrar, exercitar ou comparar solucionadores de problemas
- Problemas no mundo real
  - problemas com interesse não acadêmico (em principio)

Miniproblemas: aspirador de pó

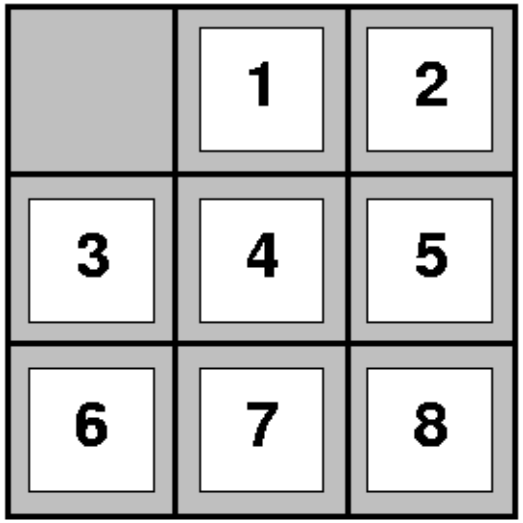


- **Estados:** duas posições para o agente
- **Estado inicial**: qqr
- **Função sucessor**: ações esq., dir., aspirar
- **Teste**: todos os quadrados limpos
- **Custo**: cada passo=1

Miniprob.:quebra-cabeças 8 peças



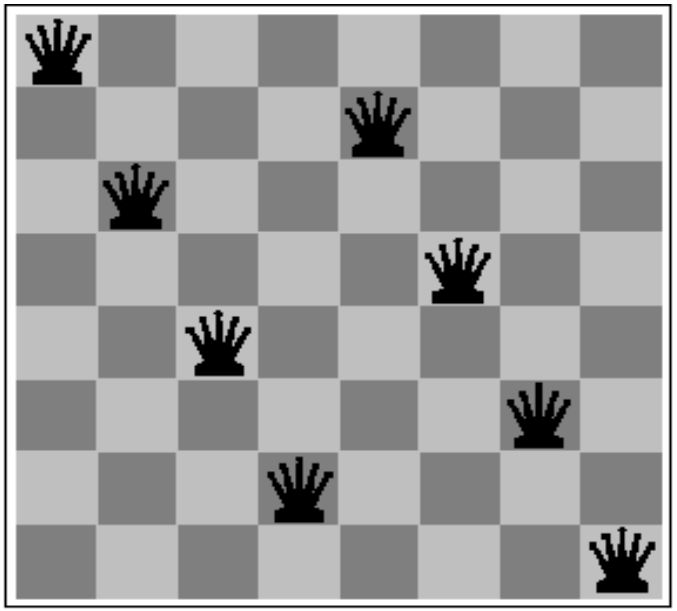
Start State



Goal State

- **Estados:** posição das 8 peças e do vazio
- **Estado inicial**: qqr estado
- **Função sucessor**: esq., dir., acima, abaixo
- **Teste**: verifica se o estado corrente é o objetivo
- **Custo**:cada passo=1; caminho=num. d passos

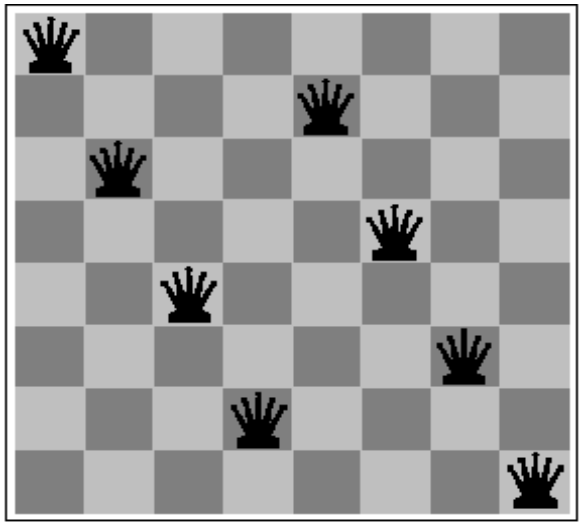
Miniprob.:8 rainhas: *formulação incremental*



Quasi solução

- **Estados:** qqr disposição de rainhas
- **Estado inicial**: nenhuma rainha
- **F. sucessor**: colocar 1 rainha em qqr vazio
- **Teste**: 8 rainhas no tab., nenhuma atacada
- 64x63x...57 = $3x10^{14}$ seq. para investigar

Miniprob.:8 rainhas: *formulação de estados completos*



Quasi solução

- **Estados:** disposições de n rainhas, uma por coluna, sem que nenhuma rainha ataque outra
- **Função sucessor**: adicionar uma rainha a qqr quadrado na coluna vazia mais à esquerda, de tal modo que ela não seja atacada
- tamanho do espaço de estados: 2.057

Uma boa formulação do problema é essencial!

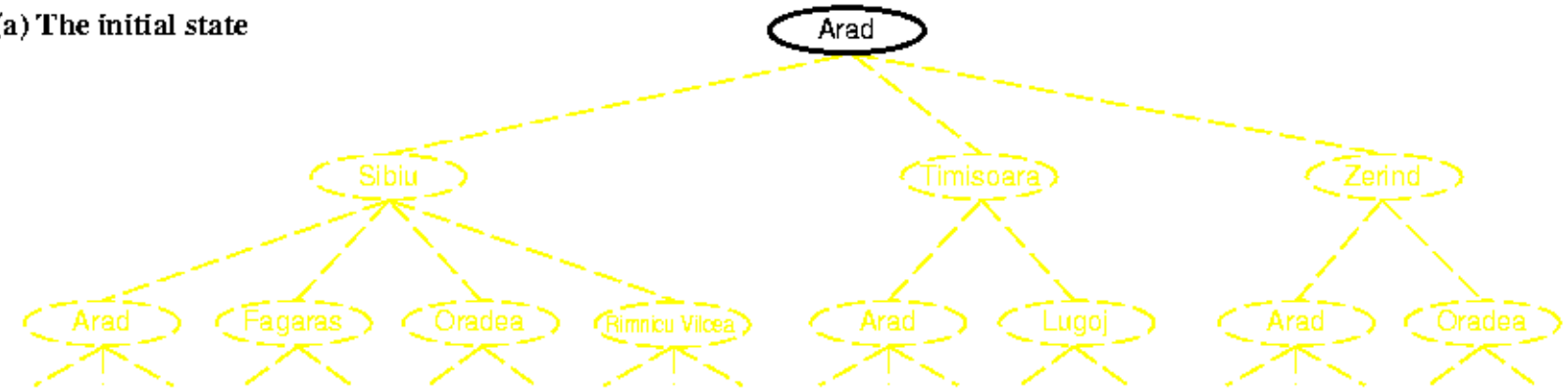
Problemas do mundo real

- Problema de roteamento
  - encontrar a melhor rota de um ponto a outro (aplicações: redes de computadores, planejamento militar, planejamento de viagens aéreas)
- Problemas de tour
  - visitar cada ponto pelo menos uma vez
- Caixeiro viajante
  - visitar cada cidade exatamente uma vez
  - encontrar o caminho mais curto

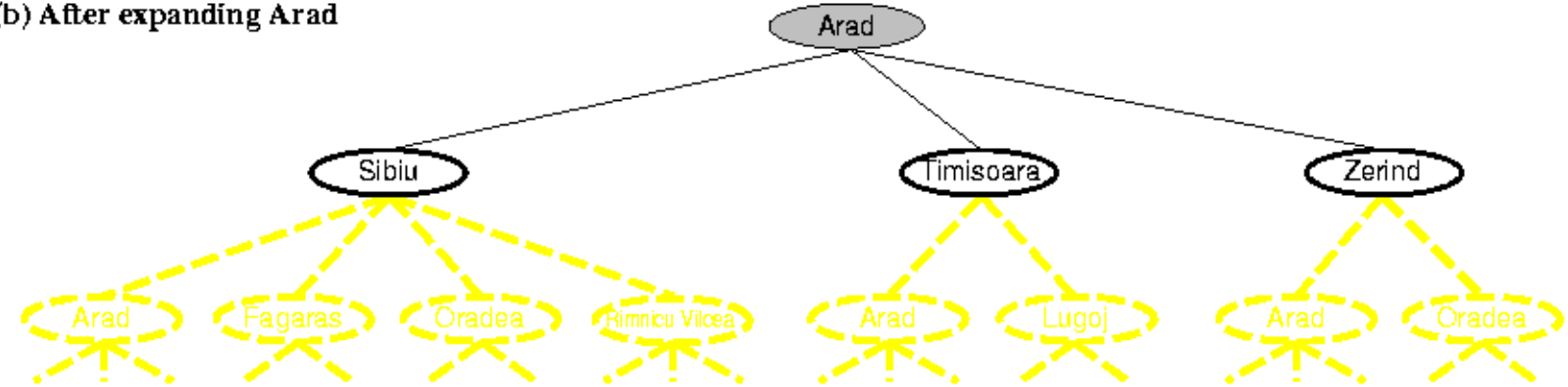
Busca de soluções

- Percorrer o espaço de estados a partir de uma ***árvore de busca***;
- *Expandir* o estado atual aplicando a função sucessor, *gerando* novos estados;
- Busca: seguir um caminho, deixando os outros para depois;
- A *estratégia de busca* determina qual caminho seguir.

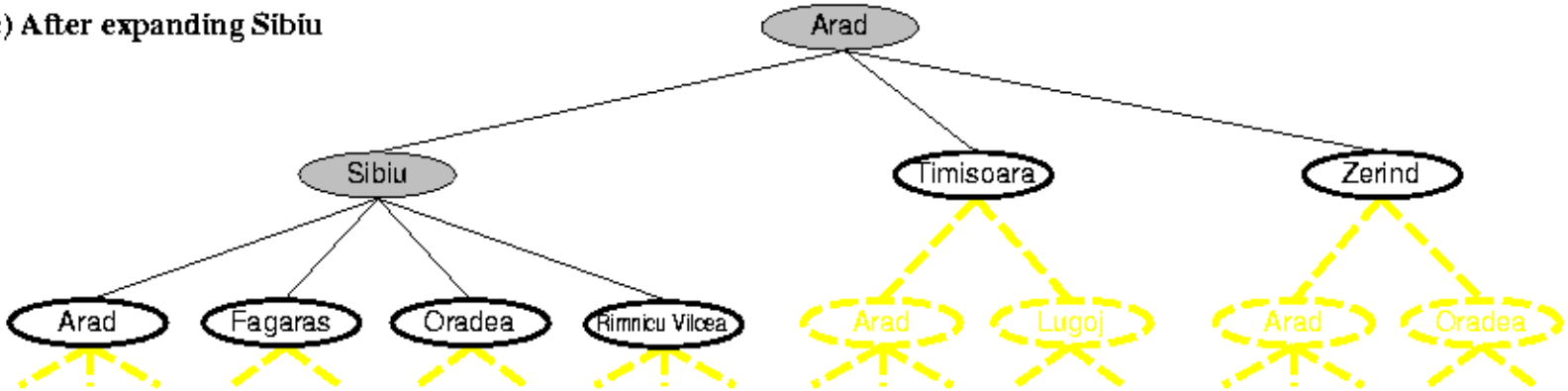
(a) The initial state



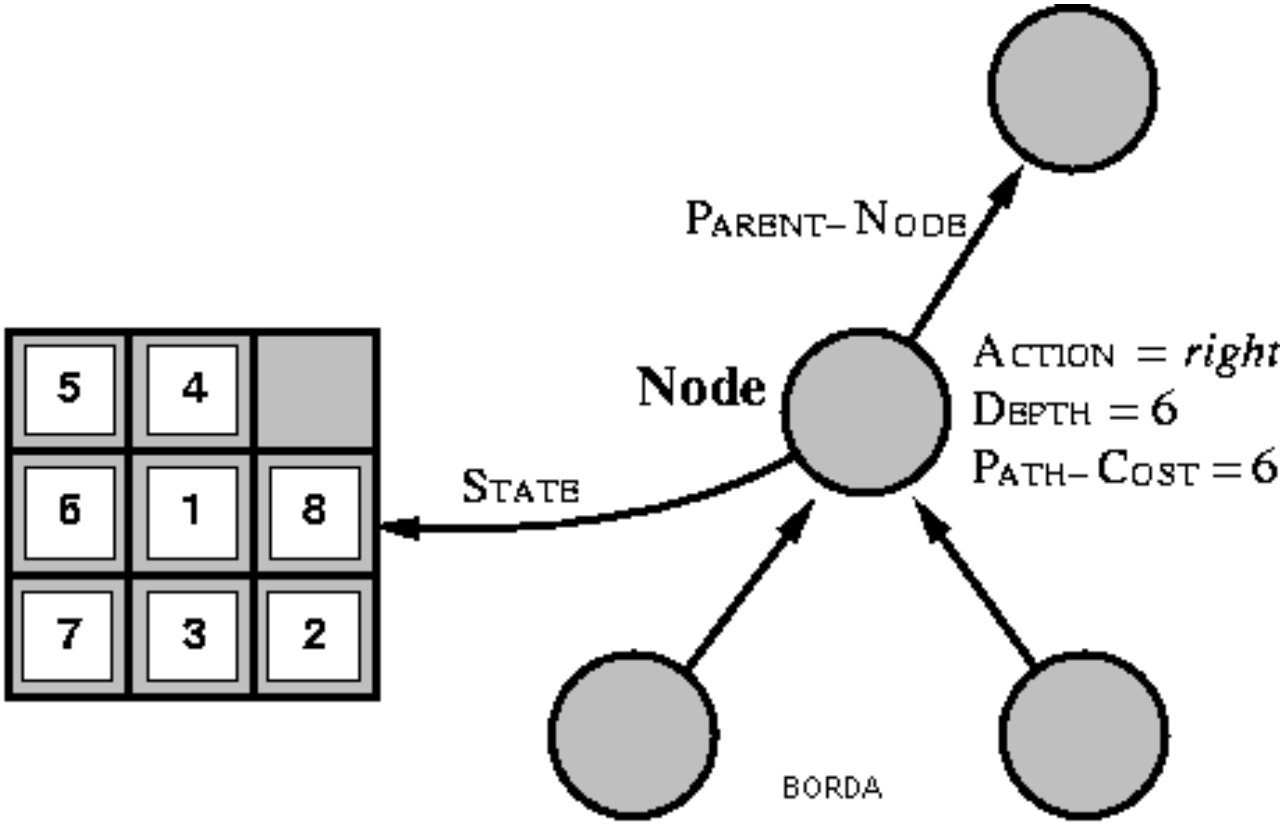
(b) After expanding Arad



(c) After expanding Sibiu



Algumas definições



Árvore de busca não é equivalente à espaço de estados!

- Há 20 estados no mapa da Romênia (espaço de estados),
- mas infinitos caminhos a percorrer.
- Portanto a árvore de busca, neste caso, tem tamanho infinito.

Medição de desempenho de um algoritmo

- Completeza: sempre encontra uma solução se esta existir;
- Otimização: encontra a solução ótima
- Complexidade de tempo: tempo gasto para encontrar uma solução;
- Complexidade de espaço: memória necessária para encontrar uma solução.

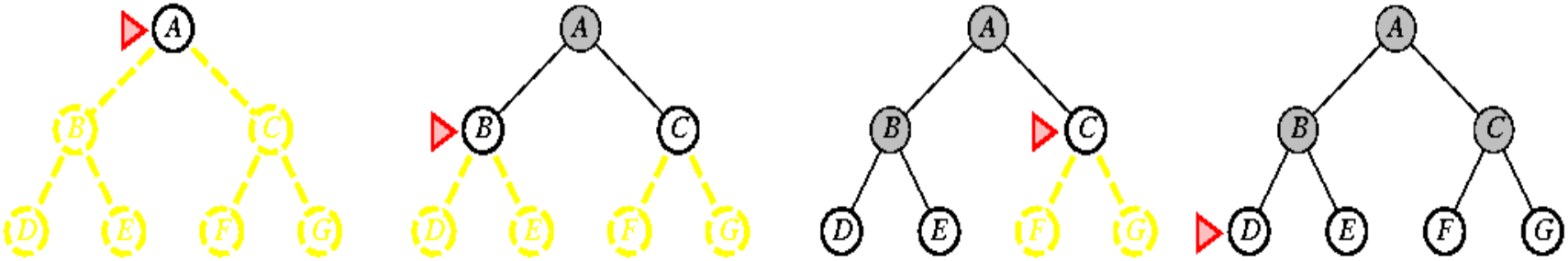
Análise de algoritmos de busca

- Fator de ramificação: **b** (número máximo de sucessores de qualquer nó);
- Profundidade do nó objetivo menos profundo: **d**
  - tempo: medido em termos do número de nós gerados durante a busca
  - espaço: número máximo de nós armazenados.

Busca em extensão

- O nó raiz é expandido primeiro e, em seguida, todos os sucessores dele, depois todos os sucessores desses nós...
- i.e., todos os nós em uma dada profundidade são expandidos antes de todos os nós do nível seguinte.

Busca em extensão em uma árvore binária



Busca em extensão: análise

- Completa: sempre encontra o objetivo que esteja em uma profundidade d (desde que b seja finito);
- Ótima??

Busca em extensão: análise

- Completa: sempre encontra o objetivo que esteja em uma profundidade d (desde que b seja finito);
- Ótima??
  - O nó objetivo mais raso não é necessariamente o nó ótimo
  - será ótimo se o custo do caminho for uma função **não-decrescente** da profundidade do nó.

Busca em extensão: análise

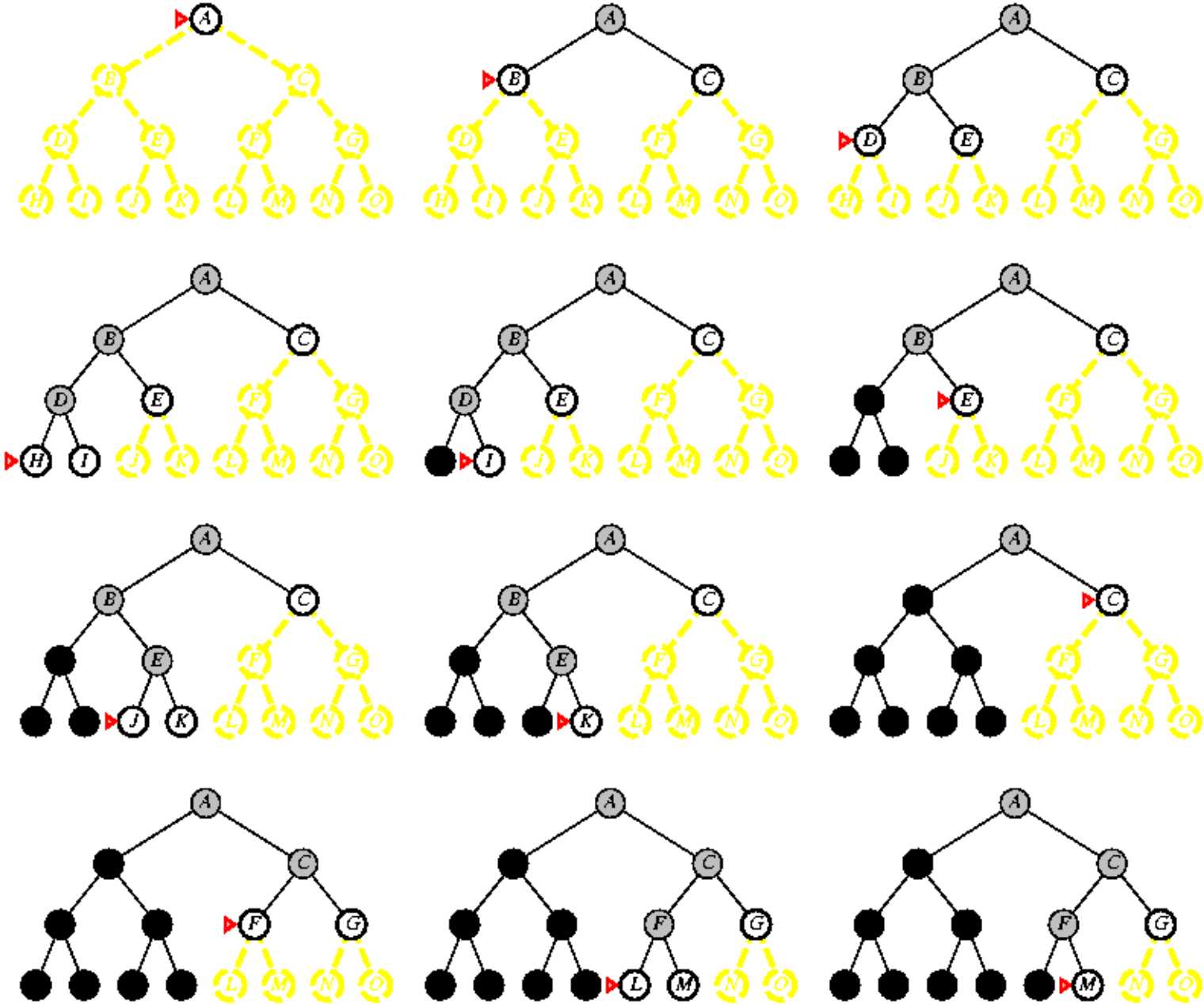
- Complexidade de tempo e espaço:
  - Espaço de estados em que cada estado tem b sucessores (portanto, a uma profundidade **n**, $\mathbf{b^n}$ estados são gerados).
  - Supondo que a solução esteja a uma profundidade **d**:

$$b + b^2 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

- No pior caso a solução está no último nó de *d*, portanto todos os irmão do objetivo tem seus filhos gerados (daí vem o d+1);
- checa o nó, sem checar se filhos (daqui o "-b").

Busca em profundidade

- Expande o nó mais profundo na borda atual da árvore;
- Não havendo mais sucessores, a busca retorna à próxima profundidade acima que não foi explorada.


A
B C
D E F G
H I J K L M N O

Busca em profundidade: análise

- Só precisa armazenar um único caminho da raiz até um nó folha, e os nós irmãos não expandidos;
- Nós cujos descendentes já foram completamente explorados podem ser retirados da memória;
- Logo para ramificação *b* e profundidade máxima *m,* a ***complexidade espacial*** é: O(*bm*)

Busca em profundidade: análise

- Pode fazer uma escolha errada e ter que percorrer um caminho muito longo (as vezes infinito), quando uma opção diferente levaria a uma solução rapidamente (ex. nó C na fig. anterior);
- Portanto, a busca em profundidade não é completa e nem ótima!
- Complexidade temporal: no pior caso, todos os nós são gerados, portanto:

$$O(b^m)$$

Busca em profundidade limitada

- Para resolver o problema de busca em profundidade em árvores infinitas, um limite **L** restringe a busca. I.e., nós na profundidade **L** são tratados como se não tivessem sucessores.
- Resolve caminhos infinitos, porém adiciona mais incompleteza;

Busca em profundidade limitada

- Limites de profundidade podem ser conhecidos a priori:
  - ex. caminho mais longo no mapa da romênia tem **L** = 19, porém qqr cidade pode ser alcançada a partir de qqr outra em **L** = 9.
- Dois tipos de falhas terminais:
  - *falha*: nenhuma solução encontrada;
  - *corte*: nenhuma solução dentro de **L**;

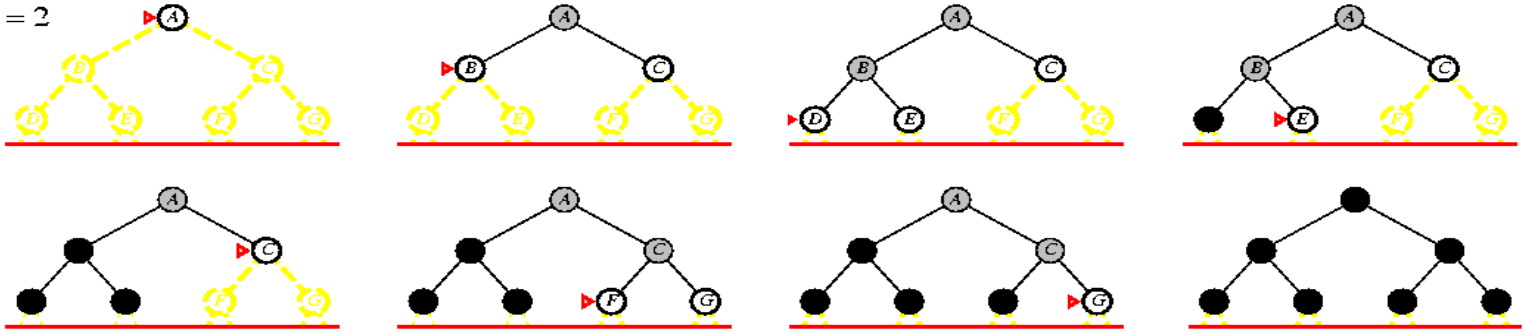
Busca por aprofundamento iterativo

- Combina busca em profundidade com busca em extensão;
- Faz busca em profundidade aumentando gradualmente o limite de profundidade;
- Método de busca preferido quando se tem espaço de busca grande e profundidade não conhecida;

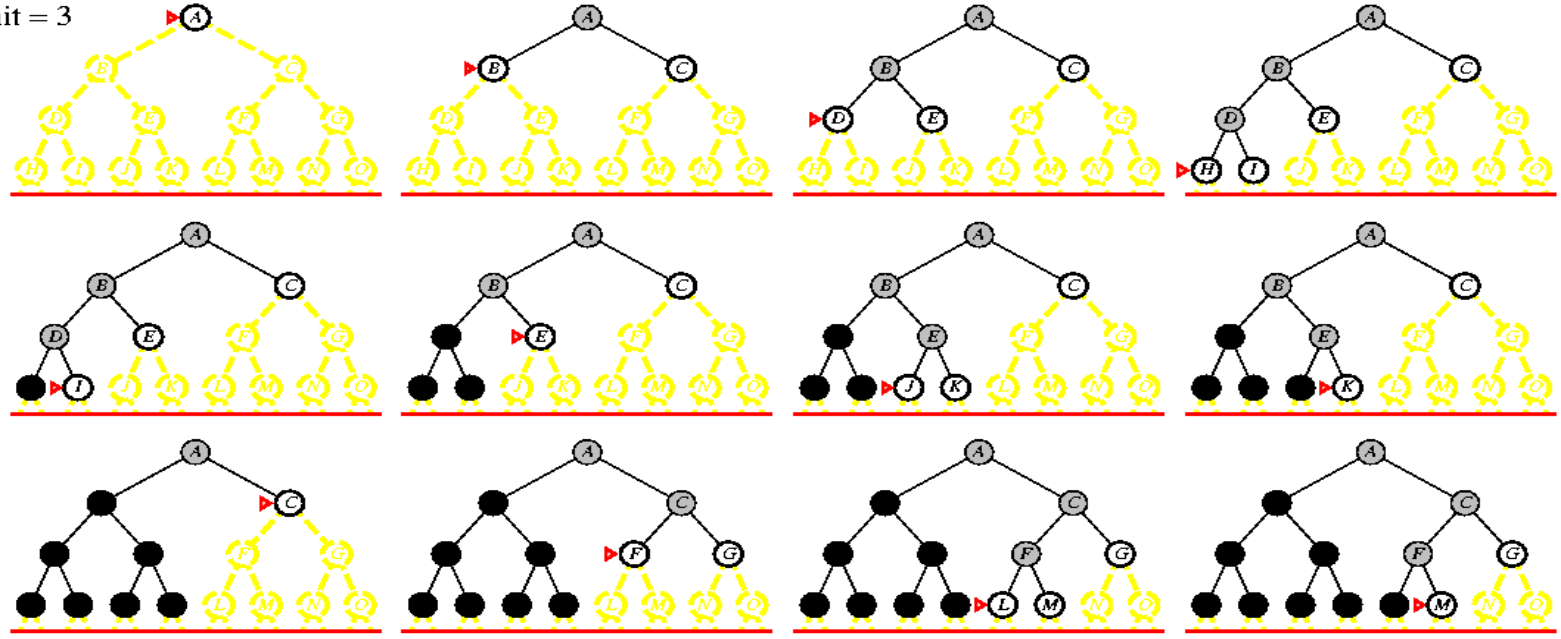
Limit = 0

Limit = 1

Limit = 2



Limit = 3



Como evitar estados repetidos

- Um processo de busca pode perder tempo expandindo nós já explorados antes;
- Estados repetidos podem levar a laços infinitos;
- **Detecção de estados repetidos:** comparar os nós prestes a serem expandidos com nós já visitados. Se o nó já tiver sido visitado, será descartado em vez de expandido.

- É um procedimento ótimo para buscas de custo uniforme, pois, nesse caso, a primeira vez que um estado aparecer será sempre "mais ótimo" que a segunda.
- Pode excluir um caminho ótimo, caso contrário...

Conclusões até agora

- Embora algoritmos de busca sejam o que há de mais antigo em IA,
- eles ainda ocupam uma posição fundamental nesta área,
- havendo poucas abordagens em Inteligência Artificial que não tenham alguma relação com algum tipo de busca em estados.

Voltamos para o Python!