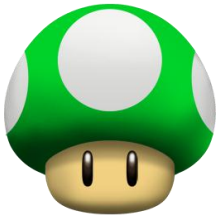
INF 1771 – Inteligência Artificial

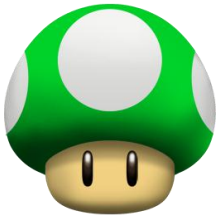
Aula 03 – Resolução de Problemas por Meio de Busca

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

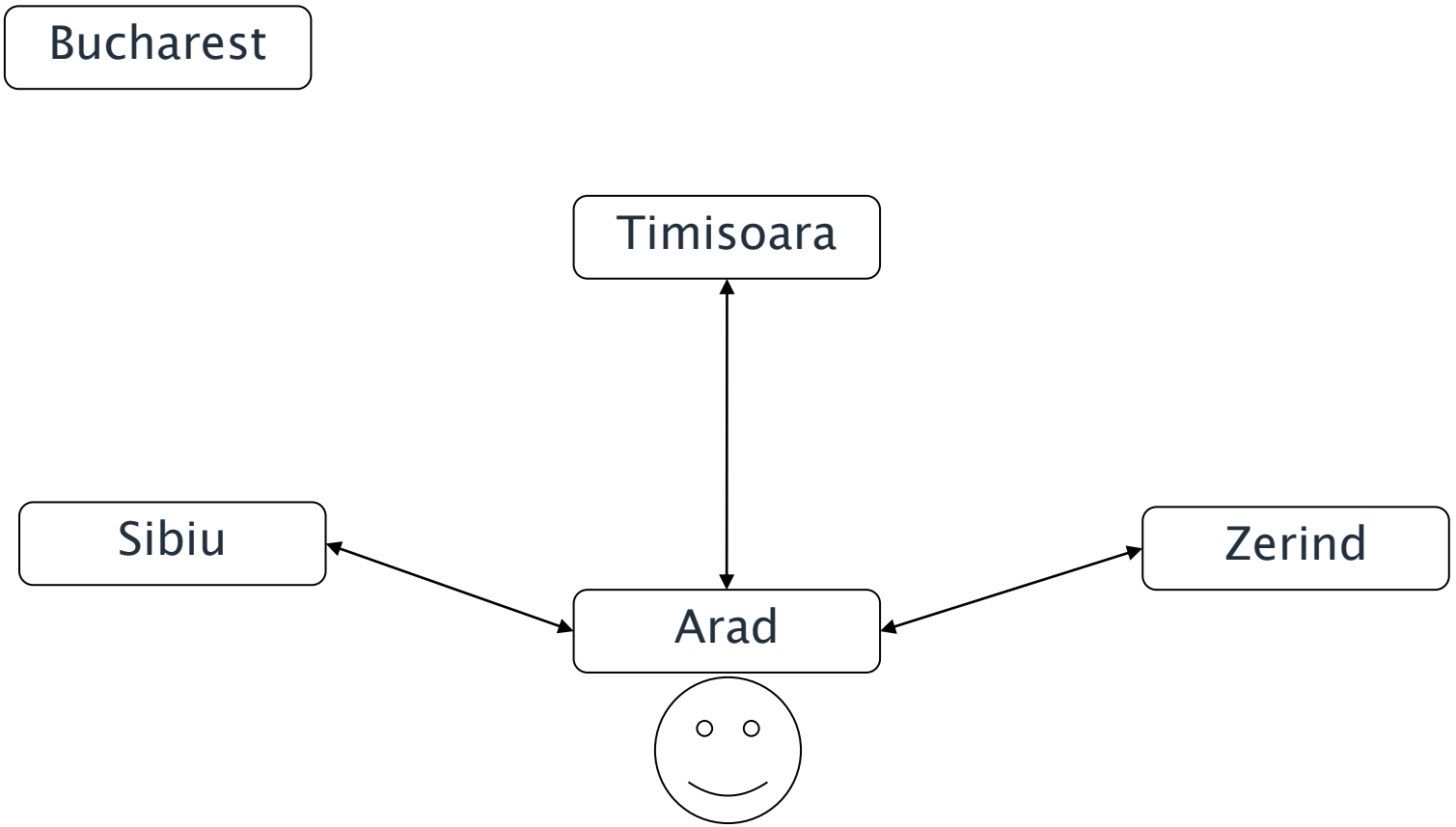


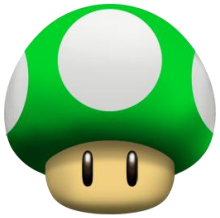
Introdução

- **Agentes Autônomos:**
  - Entidades autônomas capazes de observar o ambiente e agir de forma a atingir determinado objetivo.
- **Tipos de Agentes:**
  - Agentes reativos simples;
  - Agentes reativos baseado em modelo;
  - Agentes baseados em objetivos;
  - Agentes baseados na utilidade;
  - Agentes baseados em aprendizado;

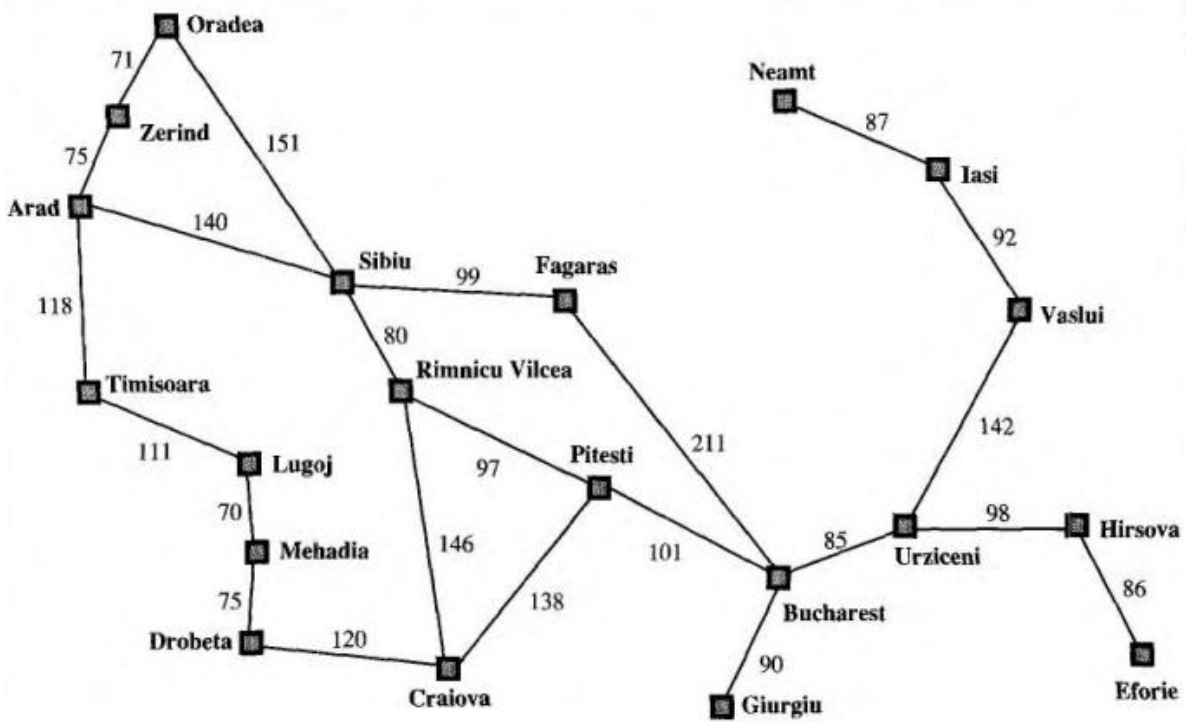


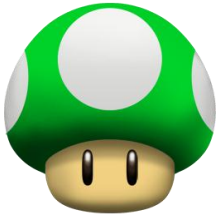
Problema de Busca





Problema de Busca

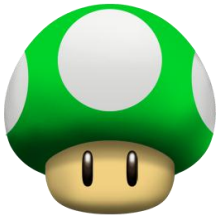




Definição do Problema

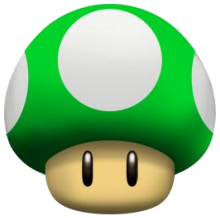
- A **definição do problema** é a primeira e mais importante etapa do processo de resolução de problemas de inteligência artificial por meio de buscas.

- Consiste em analisar o **espaço de possibilidades** de resolução do problema, encontrar sequências de ações que levem a um objetivo desejado.



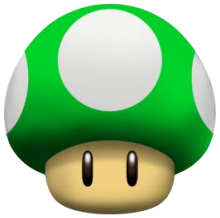
Definição de um Problema

- **Estado Inicial:** Estado inicial do agente.
  - Ex: Em(Arad)
- **Estado Final:** Estado buscado pelo agente.
  - Ex: Em(Bucharest)
- **Ações Possíveis:** Conjunto de ações que o agente pode executar.
  - Ex: Ir(Cidade, PróximaCidade)
- **Espaço de Estados:** Conjunto de estados que podem ser atingidos a partir do estado inicial.
  - Ex: Mapa da Romênia.
- **Custo:** Custo numérico de cada caminho.
  - Ex: Distância em KM entre as cidades.



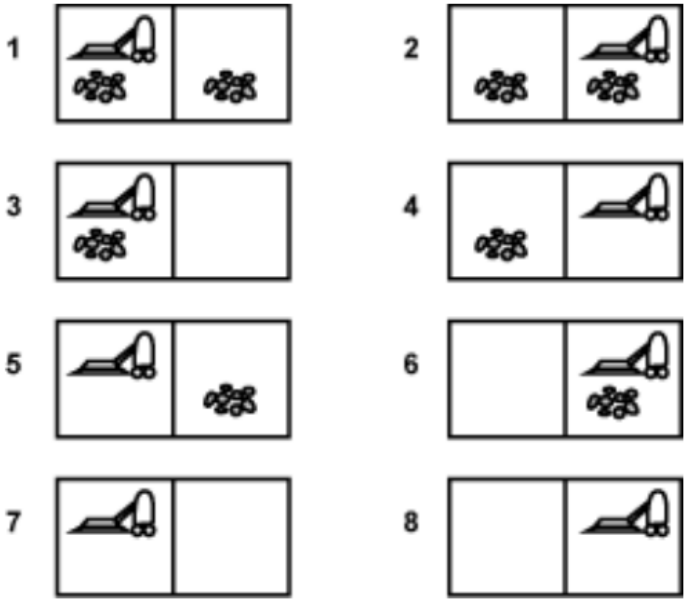
Considerações em Relação ao Ambiente

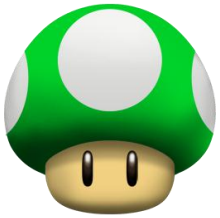
- **Estático:**
  - O Ambiente não pode mudar enquanto o agente está realizando a resolução do problema.
- **Observável:**
  - O estado inicial do ambiente precisa ser conhecido previamente.
- **Determinístico:**
  - O próximo estado do agente deve ser determinado pelo estado atual + ação. A execução da ação não pode falhar.



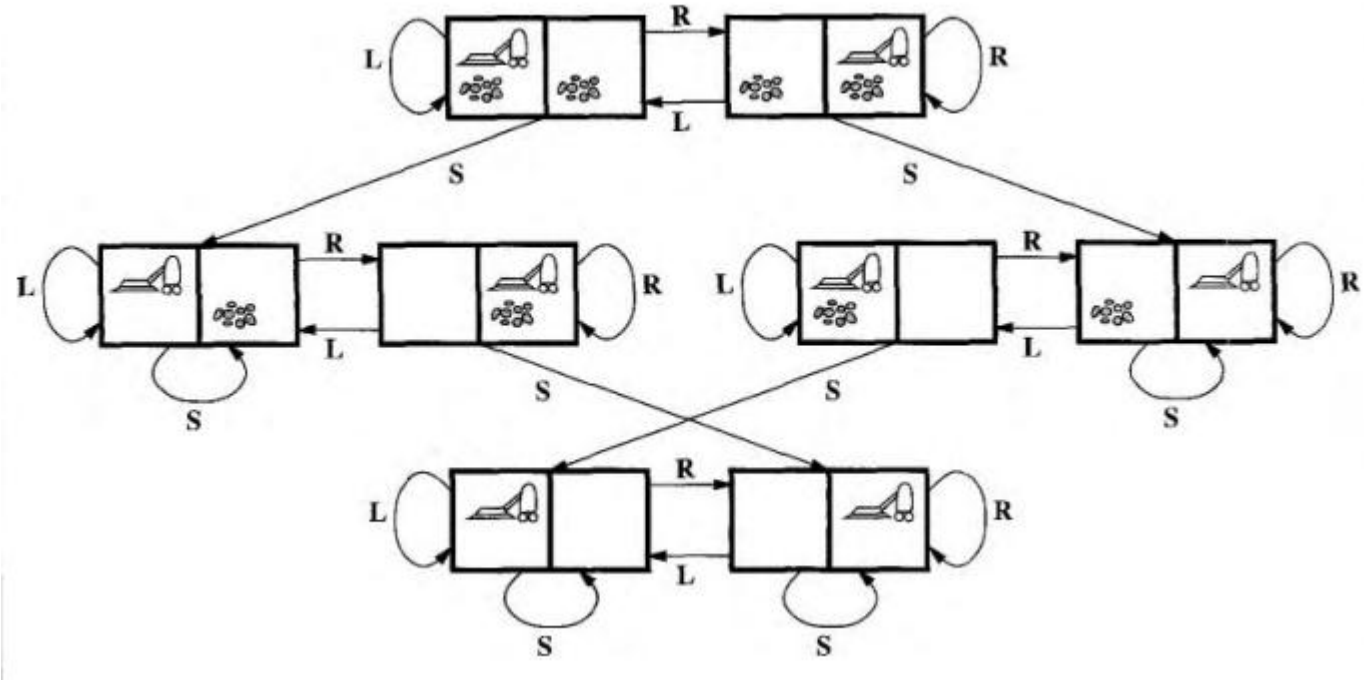
Exemplo: Aspirador de Pó

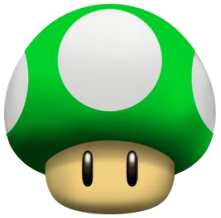
- **Espaço de Estados:** 8 estados possíveis (figura ao lado);
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Estado 7 ou 8 (ambos quadrados limpos);
- **Ações Possíveis:** Mover para direita, mover para esquerda e limpar;
- **Custo:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo numero de passos;





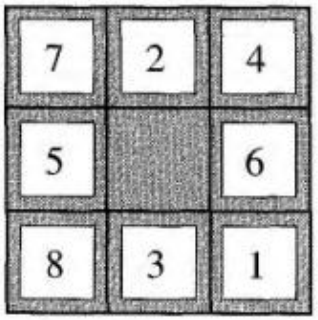
Exemplo: Aspirador de Pó



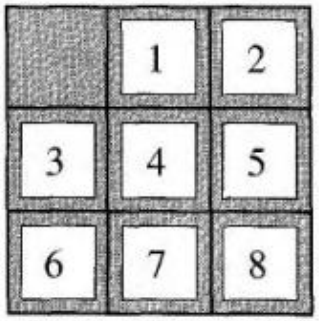


Exemplo: 8–Puzzle

- **Espaço de Estados:** 181.440 possíveis estados;
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Figura ao lado – Goal State;
- **Ações Possíveis:** Mover o quadrado vazio para direita, para esquerda, para cima ou para baixo;
- **Custo:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo numero de passos;

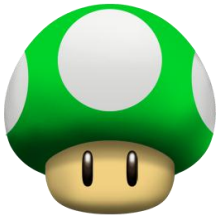


Start State



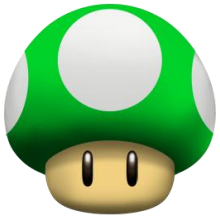
Goal State

- **15-puzzle (4x4) –** 1.3 trilhões estados possíveis.
- **24-puzzle (5x5) –** 10^{25} estados possíveis.



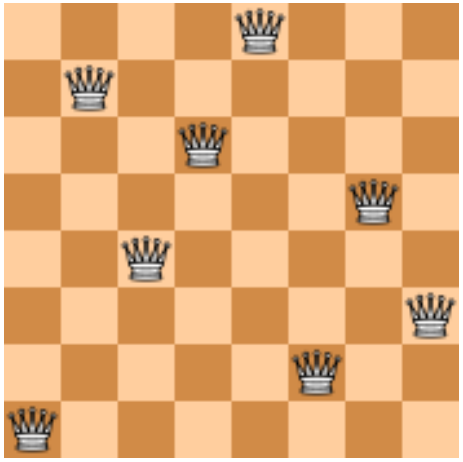
Exemplo: Xadrez

- **Espaço de Estados:** Aproximadamente 10^{40} possíveis estados (Claude Shannon, 1950);
- **Estado Inicial:** Posição inicial de um jogo de xadrez;
- **Estado Final:** Qualquer estado onde o rei adversário está sendo atacado e o adversário não possui movimentos válidos;
- **Ações Possíveis:** Regras de movimentação de cada peça do xadrez;
- **Custo:** Quantidade de posições examinadas;

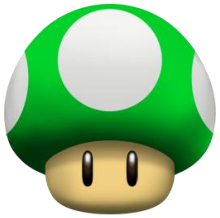


Exemplo: 8 Rainhas (Incremental)

- **Espaço de Estados:** Qualquer disposição de 0 a 8 rainhas no tabuleiro (1.8 x 10^{14} possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Estado Final:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma esta sendo atacada;
- **Ações Possíveis:** Colocar uma rainha em um espaço vaio do tabuleiro;
- **Custo:** Não importa nesse caso;

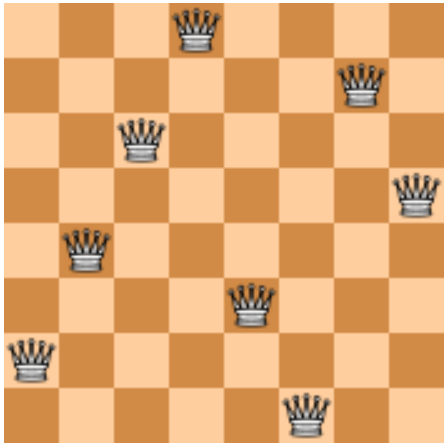


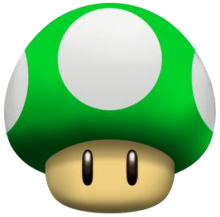
* O jogo possui apenas 92 possíveis soluções (considerando diferentes rotações e reflexões). E apenas 12 soluções únicas.



Exemplo: 8 Rainhas (Estados Completos)

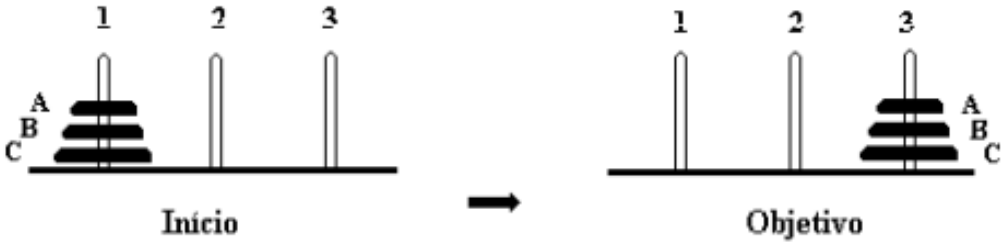
- **Espaço de Estados:** Tabuleiro com n rainhas, uma por coluna, nas n colunas mais a esquerda sem que nenhuma rainha ataque outra (2057 possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Estado Final:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma esta sendo atacada;
- **Ações Possíveis:** Adicionar uma rainha em qualquer casa na coluna vazia mais à esquerda de forma que não possa ser atacada;
- **Custo:** Não importa nesse caso;



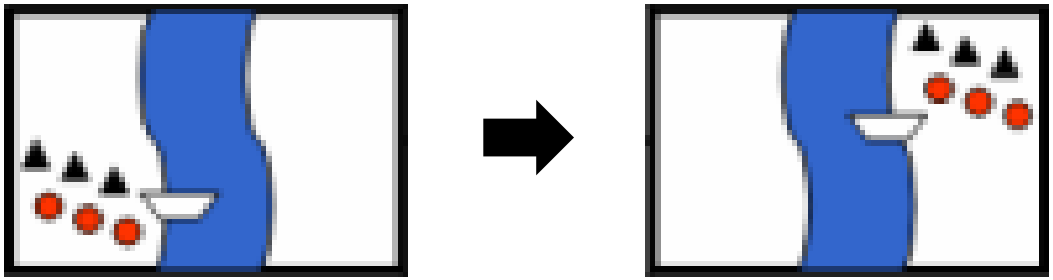


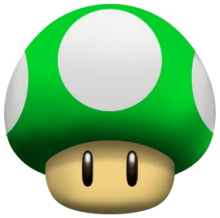
Exercícios

- Torre de Hanói?



- Canibais e Missionários?

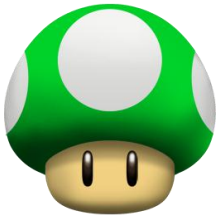




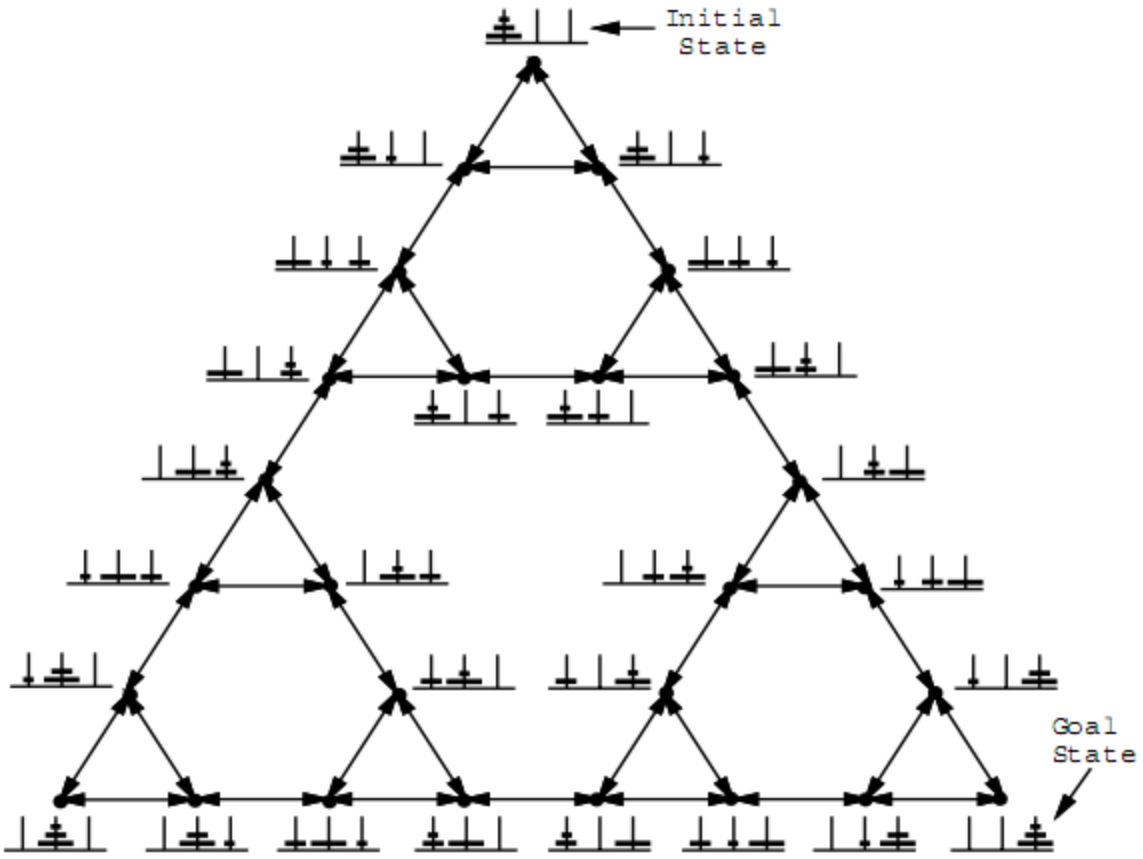
Exercícios

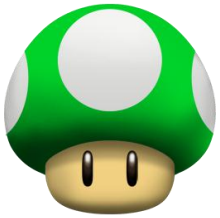
- **Torre de Hanói:**
  - **Espaço de Estados**: Todas as possíveis configurações de argolas em todos os pinos (27 possíveis estados).
  - **Ações Possíveis:** Mover a primeira argola de qualquer pino para o pino da direita ou da esquerda.
  - **Custo**: Cada movimento tem 1 de custo.

- **Canibais e Missionários:**
  - **Espaço de Estados**: Todas as possíveis configurações validas de canibais e missionários em cada lado do rio (16 possíveis estados).
  - **Ações Possíveis:** Mover 1 ou 2 personagens (canibais ou missionários) para o outro lado do rio. O número de canibais em um determinado lado do rio não pode ser maior do que o número de missionários.
  - **Custo**: Cada movimento tem 1 de custo.

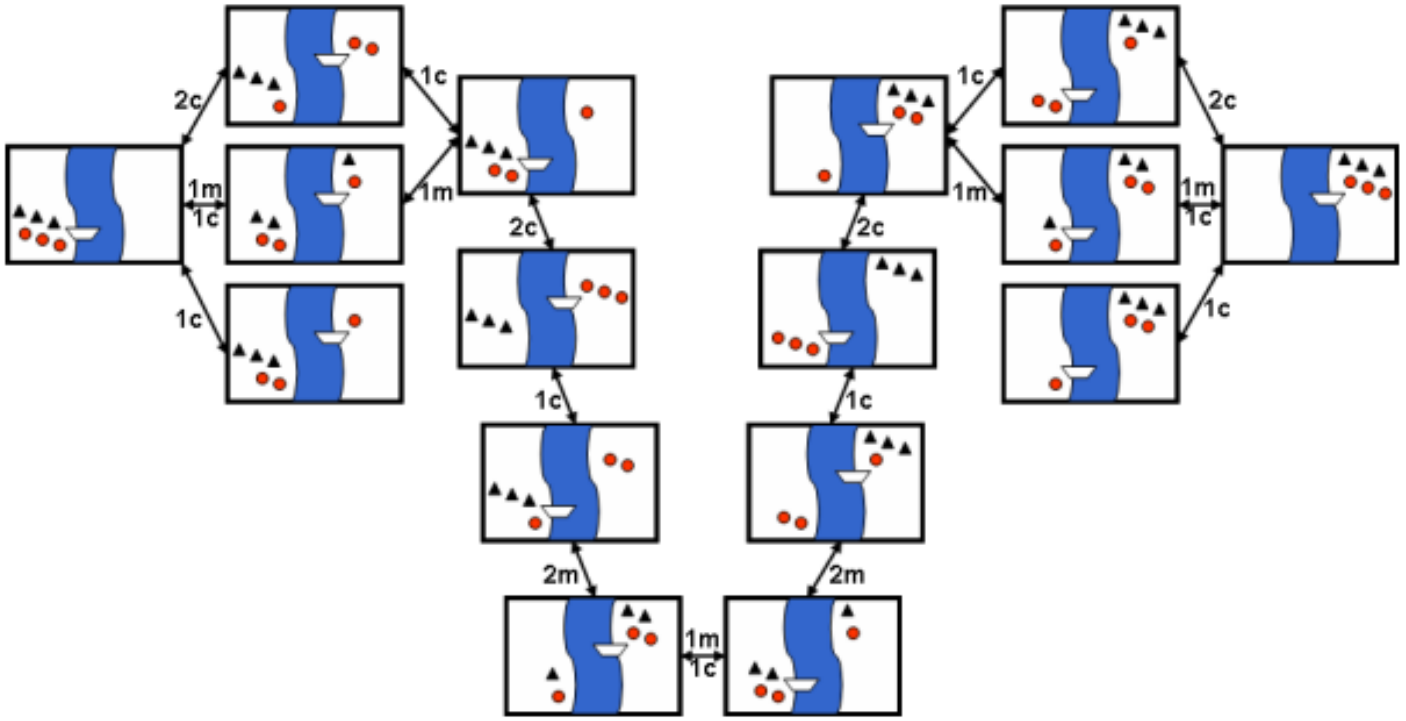


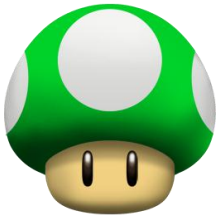
Exercícios





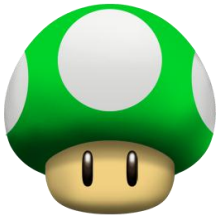
Exercícios





Aplicações em Problemas Reais

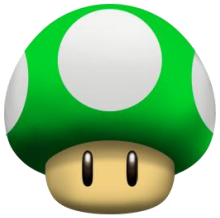
- **Cálculo de Rotas:**
  - Planejamento de rotas de aviões;
  - Sistemas de planejamento de viagens;
  - Caixeiro viajante;
  - Rotas em redes de computadores;
  - Jogos de computadores (rotas dos personagens);
- **Alocação**
  - Salas de aula;
  - Máquinas industriais;



Aplicações em Problemas Reais

- **Circuitos Eletrônicos:**
  - Posicionamento de componentes;
  - Rotas de circuitos;

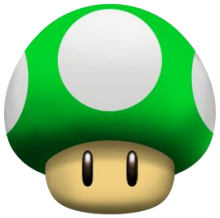
- **Robótica:**
  - Navegação e busca de rotas em ambientes reais;
  - Montagem de objetos por robôs;



Como Encontrar a Solução?

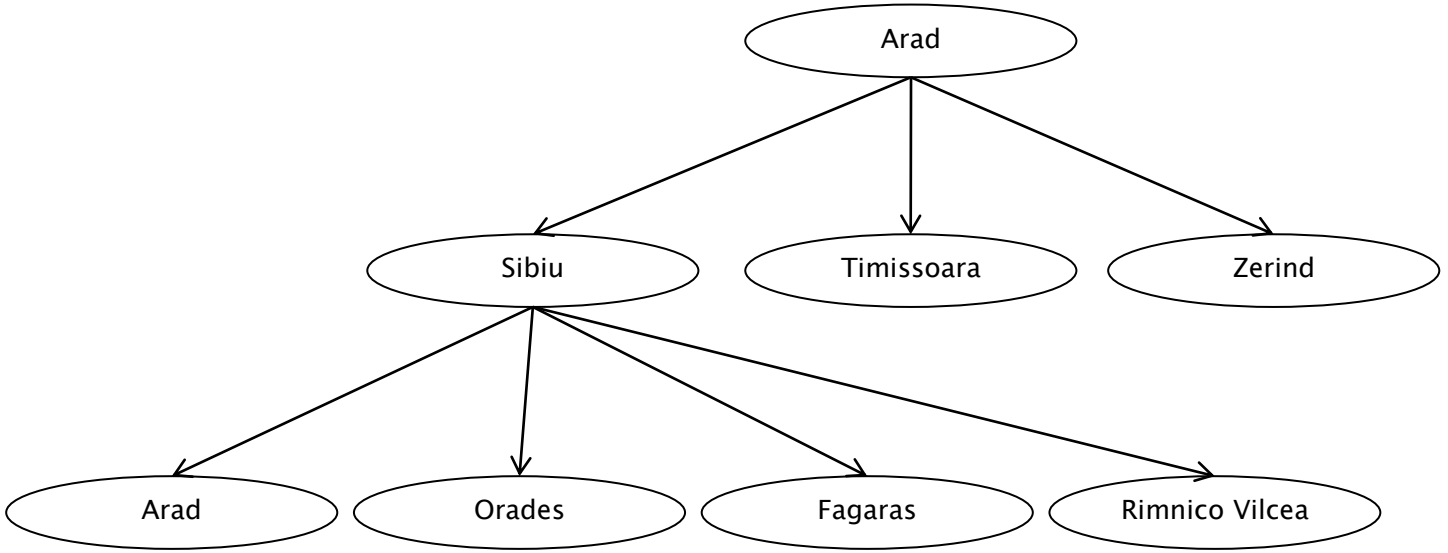
- Uma vez o problema bem formulado, o estado final (objetivo) deve ser "**buscado**" no espaço de estados.

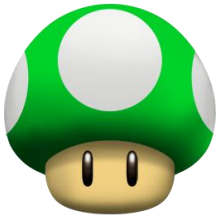
- A busca é representada em uma **árvore de busca**:
  - Raiz: corresponde ao estado inicial;
  - Expande-se o estado corrente, gerando um novo conjunto de sucessores;
  - Escolhe-se o próximo estado a expandir seguindo uma **estratégia de busca**;
  - Prossegue-se até chegar ao estado final (solução) ou falhar na busca pela solução;



Buscando Soluções

- **Exemplo:** Ir de **Arad** para **Bucharest**

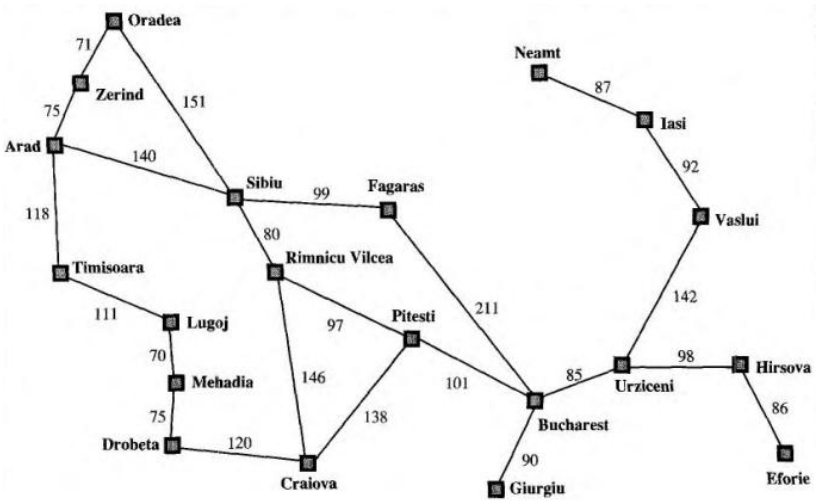


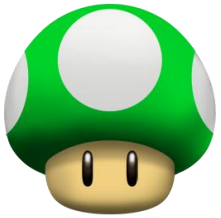


Buscando Soluções

- O espaço de estados é **diferente** da árvore de buscas.

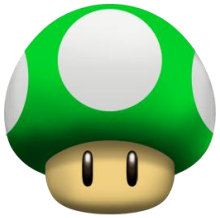
- **Exemplo:**
- 20 estados no espaço de espaços;
- Número de caminhos infinito;
- Árvore com infinitos nós;





Código Descritivo – Busca em Árvore

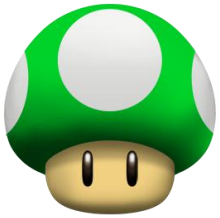
```
Função BuscaEmArvore(Problema, Estratégia) retorna solução ou falha
Inicio
    Inicializa a arvore usando o estado inicial do Problema
    loop do
        se não existem candidatos para serem expandidos então
                retorna falha
        Escolhe um nó folha para ser expandido de acordo com
        a Estratégia
        se Se o nó possuir o estado final então
                retorna solução correspondente
        se não
                expande o nó e adiciona os nós resultantes a
                arvore de busca
Fim
```



Pseudocódigo – Busca em Árvore

```
Função BuscaEmArvore(Problema, fronteira) retorna solução ou falha
Inicio
  fronteira ← InsereNaFila(FazNó(Problema[EstadoInicial]), fronteira)
  loop do
    se FilaVazia(fronteira) então
      retorna falha
    nó ← RemovePrimeiro(fronteira)
    se nó[Estado] for igual a Problema[EstadoFinal] então
      retorna Solução(nó)
    fronteira ← InsereNaFila(ExpandeFronteira(nó, Problema), fronteira)
Fim
```

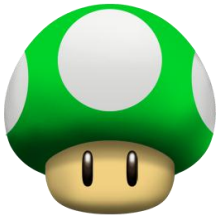
- A função **Solução** retorna a sequência de nós necessários para retornar a raiz da arvore.
- Considera-se *fronteira* uma estrutura do tipo fila.



Medida de Desempenho

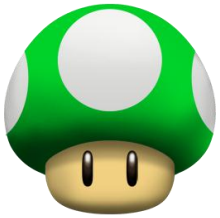
- **Desempenho do Algoritmo:**
  - (1) O algoritmo encontrou alguma solução?
  - (2) É uma boa solução?
    - Custo de caminho (qualidade da solução).
  - (3) É uma solução computacionalmente barata?
    - Custo da busca (tempo e memória).

- **Custo Total**
  - Custo do Caminho + Custo de Busca.



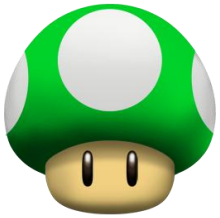
Métodos de Busca

- **Busca Cega ou Exaustiva:**
  - Não sabe qual o melhor nó da fronteira a ser expandido. Apenas distingue o estado objetivo dos não objetivos.
- **Busca Heurística:**
  - Estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas.
- **Busca Local:**
  - Operam em um único estado e movem-se para a vizinhança deste estado.



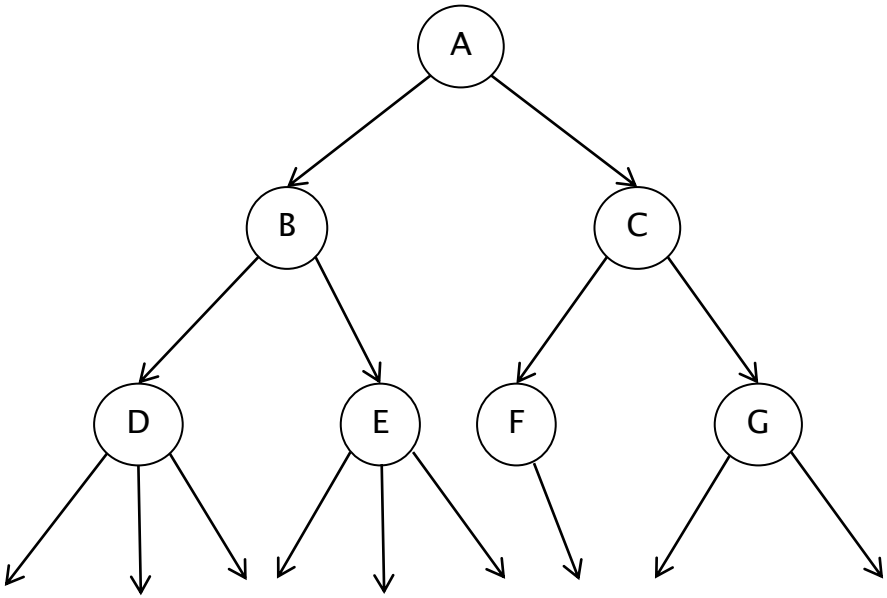
Busca Cega

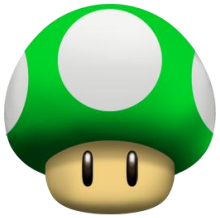
- **Algoritmos de Busca Cega:**
  - Busca em largura;
  - Busca de custo uniforme;
  - Busca em profundidade;
  - Busca com aprofundamento iterativo;



Busca em Largura

- **Estratégia:**
  - O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.





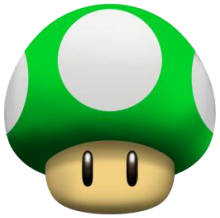
Busca em Largura

- Pode ser implementado com base no pseudocódigo da função "BuscaEmArvore" apresentado anteriormente. Utiliza-se uma estrutura de fila (first-in-first-out) para armazenar os nós das fronteira.

- **Complexidade:** $O(b^{d+1})$

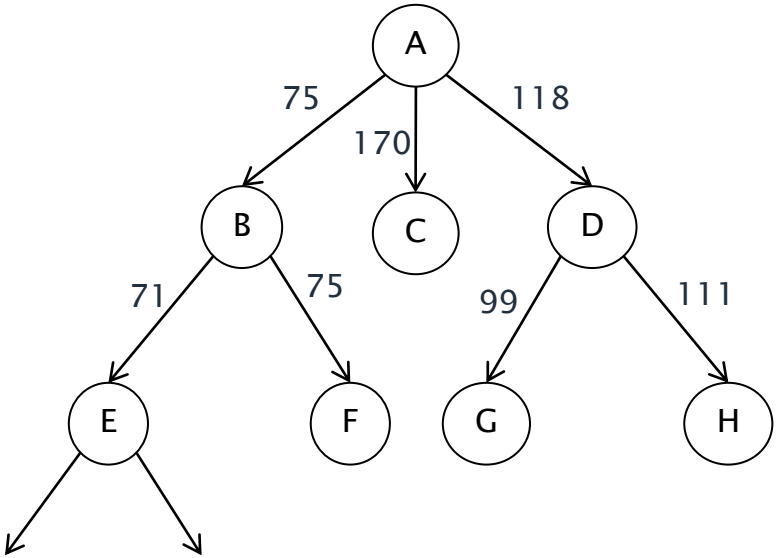
Profundidade (*d*)	**Nós**	**Tempo**	**Memória**
2	1100	0.11 ms	107 KB
4	111,100	11 ms	10.6 MB
6	10^7	1.1 seg	1 GB
8	10^9	2 min	103 GB
10	10^{11}	3 horas	10 TB
12	10^{13}	13 dias	1 PB
14	10^{15}	3.5 anos	99 PB

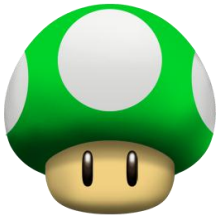
** Considerando o numero de folhas b = 10 e cada nó ocupando 1KB de memória*



Busca de Custo Uniforme

- **Estratégia:**
  - Expande sempre o nó de menor custo de caminho. Se o custo de todos os passos for o mesmo, o algoritmo acaba sendo o mesmo que a busca em largura.



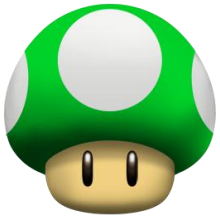


Busca de Custo Uniforme

- A primeira solução encontrada é a **solução ótima** se custo do caminho sempre aumentar ao logo do caminho, ou seja, não existirem operadores com custo negativo.

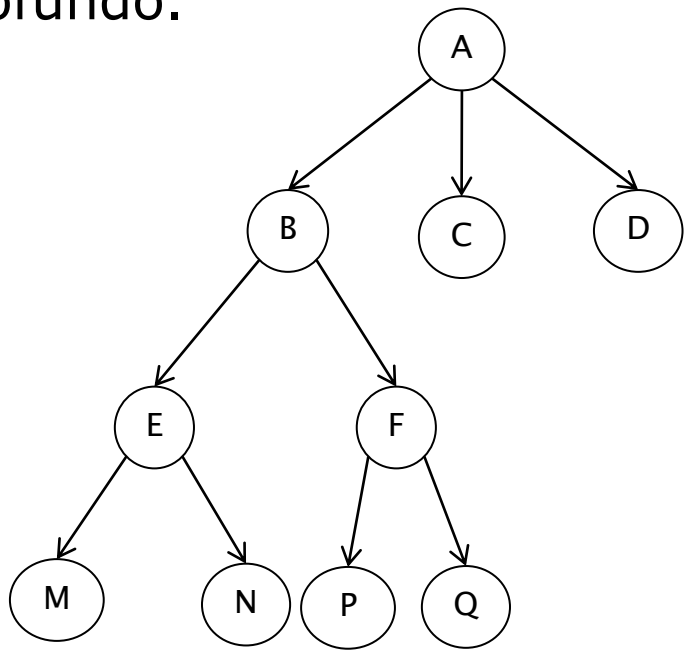
- Implementação semelhante a busca em largura. Adiciona-se uma **condição de seleção** dos nós a serem expandidos.

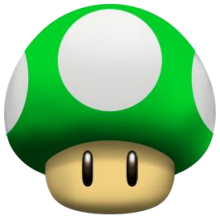
- **Complexidade:** $O(b^{1+(C/\alpha)})$
  - Onde:
    C = custo da solução ótima;
    α = custo mínimo de uma ação;



Busca em Profundidade

- **Estratégia:**
  - Expande os nós da vizinhança até o nó mais profundo.



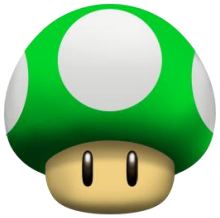


Busca em Profundidade

- Pode ser implementado com base no pseudocódigo da função “BuscaEmArvore” apresentado anteriormente. Utiliza-se uma estrutura de pilha (last-in-first-out) para armazenar os nós das fronteira.

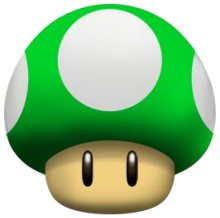
- Pode também ser implementado de forma recursiva.

- **Consome pouca memória**, apenas o caminho de nós sendo analisados precisa armazenado. Caminhos que já foram explorados podem ser descartados da memória.



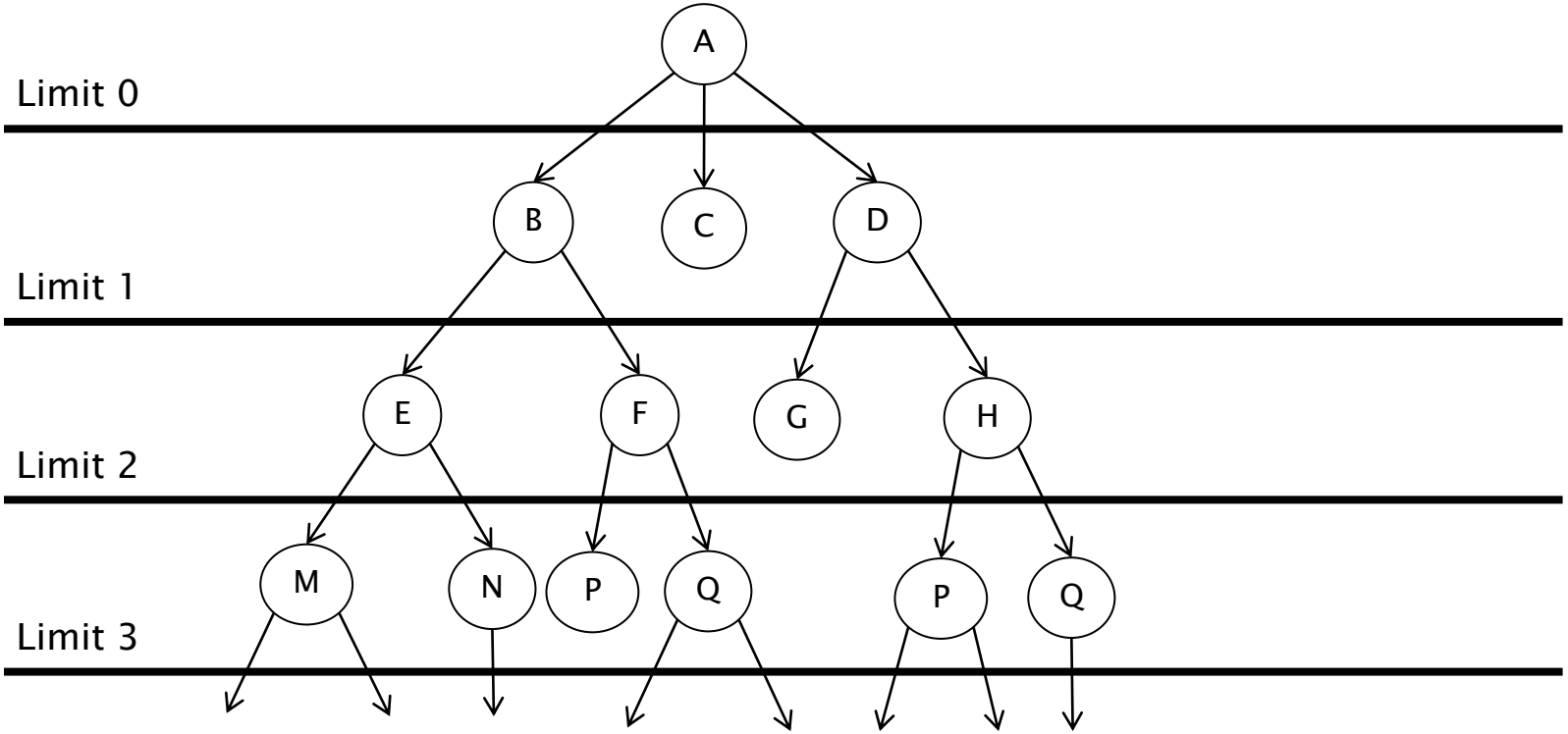
Busca em Profundidade

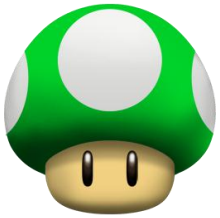
- Uso de memória pela **busca em largura** em uma arvore com 12 de profundidade: 1000 TB.
- Uso de memória pela **busca em profundidade** em uma arvore com 12 de profundidade: 118 KB.
- **Problema:** O algoritmo pode fazer uma busca muito longa mesmo quando a resposta do problema esta localizado a poucos nós da raiz da árvore.



Busca com Aprofundamento Iterativo

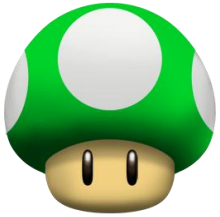
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.





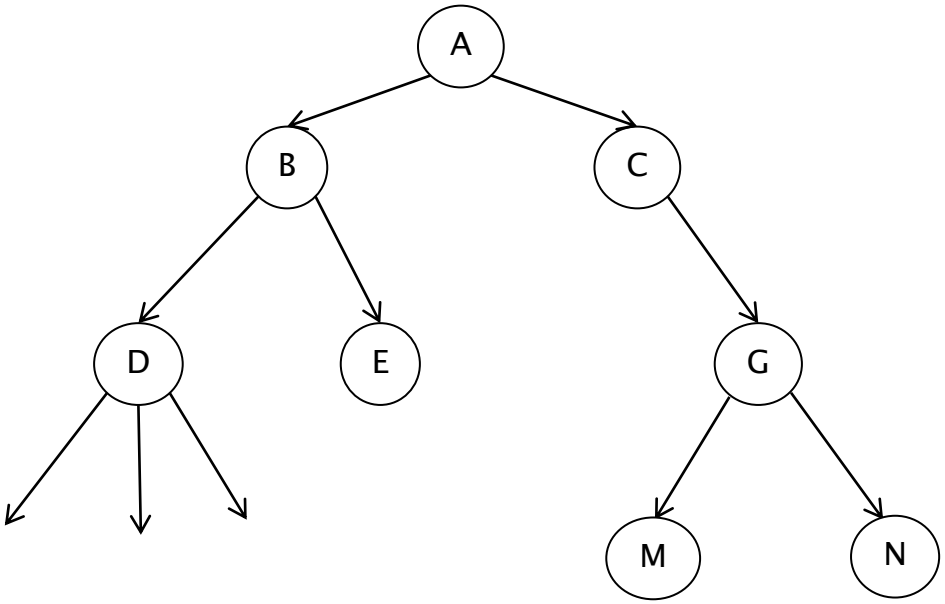
Busca com Aprofundamento Iterativo

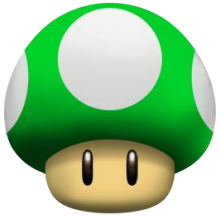
- Combina os benefícios da busca em largura com os benefícios da busca em profundidade.
- Evita o problema de caminhos muito longos ou infinitos.
- A repetição da expansão de estados não é tão ruim, pois a maior parte dos estados está nos níveis mais baixos.
- Cria menos estados que a busca em largura e consome menos memória.



Busca Bidirecional

- **Estratégia:**
  - A busca se inicia ao mesmo tempo a partir do estado inicial e do estado final.

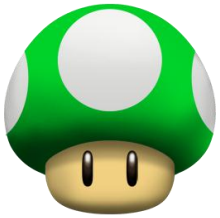




Comparação dos Metodos de Busca Cega

Criterio	Largura	Uniforme	Profundidade	Aprofundamento Iterativo	Bidirecional
Completo?	Sim [1]	Sim [1],[2]	Não	Sim [1]	Sim [1], [4]
Ótimo?	Sim [3]	Sim	Não	Sim [3]	Sim [3], [4]
Tempo	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(b^{m})$	$O(b^{d})$	$O(b^{d/2})$
Espaço	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(bm)$	$O(bd)$	$O(b^{d/2})$

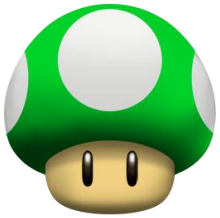
b = fator de folhas por nó.
d = profundidade da solução mais profunda.
m = profundidade máxima da árvore.
[1] completo se b for finito.
[2] completo se o custo de todos os passos for positivo.
[3] ótimo se o custo de todos os passos for idêntico.
[4] se ambas as direções usarem busca em largura.



Como evitar estados repetidos?

- Estados repetidos sempre vão ocorrer em problema onde os estados são reversíveis.

- Como evitar?
  - Não retornar ao estado “pai”.
  - Não retorna a um ancestral.
  - Não gerar qualquer estado que já tenha sido criado antes (em qualquer ramo).
    - Requer que todos os estados gerados permaneçam na memória.

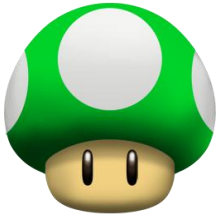


Busca Heurística

- **Algoritmos de Busca Heurística:**
  - Busca Gulosa
  - A*

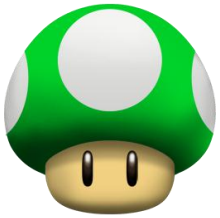
- A busca heurística leva em conta o **objetivo** para decidir qual caminho escolher.

- Conhecimento extra sobre o problema é utilizado para **guiar o processo de busca**.



Busca Heurística

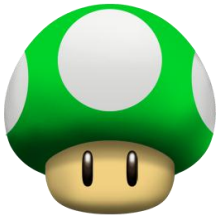
- Como encontrar um barco perdido?
  - **Busca Cega** -> Procura no oceano inteiro.
  - **Buca Heurística** -> Procura utilizando informações relativas ao problema. Ex: correntes marítimas, vento, etc.



Busca Heurística

- **Função Heurística** (*h*)
  - Estima o custo do caminho mais barato do estado atual até o estado final mais próximo.
  - São específicas para cada problema.

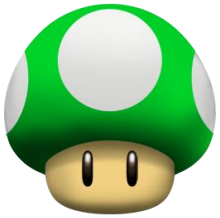
- **Exemplo:**
  - Encontrar a rota mais curta entre duas cidades:
    - *h*(*n*) = distância em linha reta direta entre o nó *n* e o nó final.



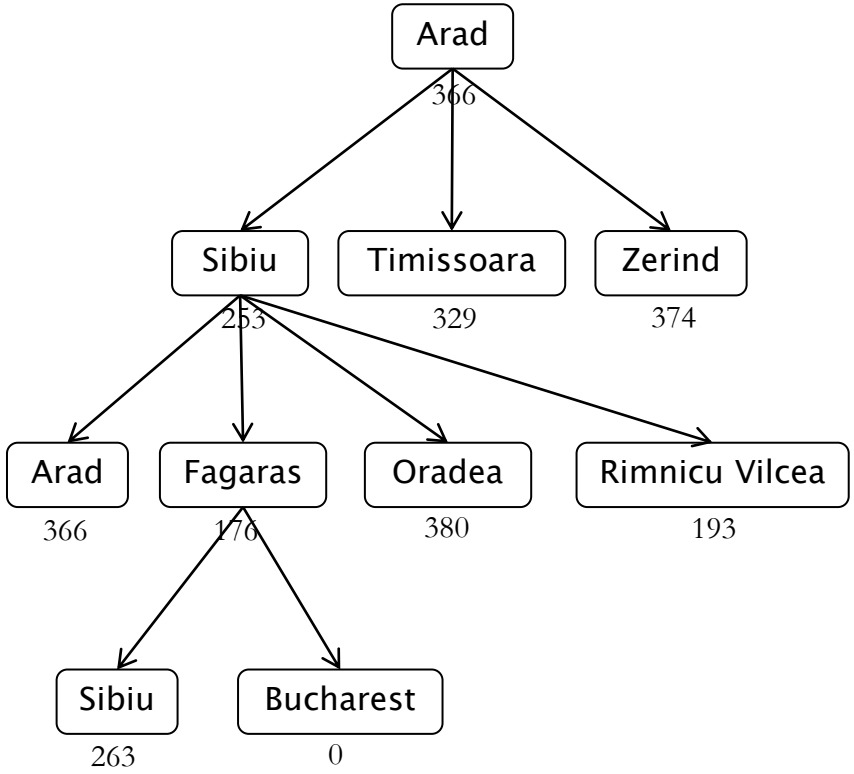
Busca Gulosa

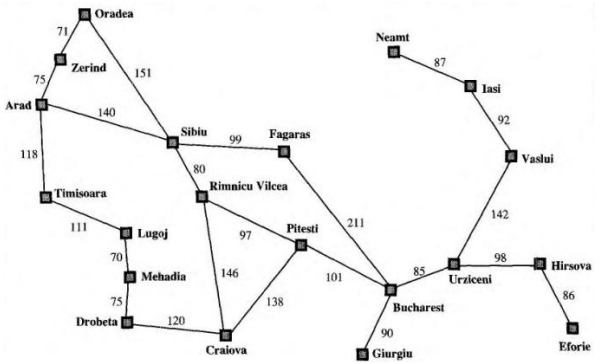
- **Estratégia:**
  - Expande os nós que se encontram mais próximos do objetivo (uma linha reta conectando os dois pontos no caso de distancias), desta maneira é provável que a busca encontre uma solução rapidamente.

- A implementação do algoritmo se assemelha ao utilizado na busca cega, entre tanto utiliza-se uma função heurística para decidir qual o nó deve ser expandido.

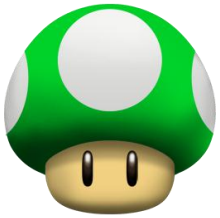


Busca Gulosa



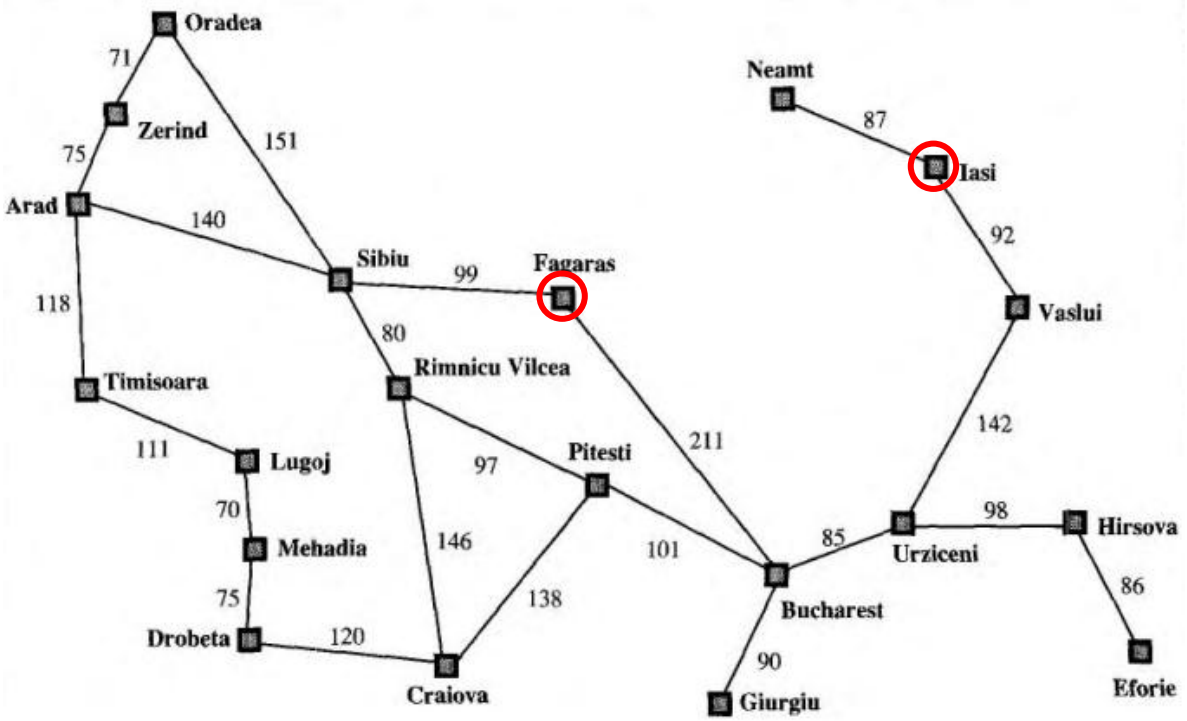


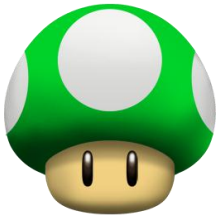
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374
Hirsova	151	Urziceni	80



Busca Gulosa

- Ir de Iasi para Fagaras?

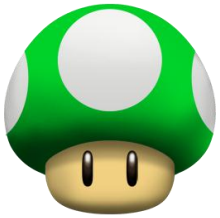




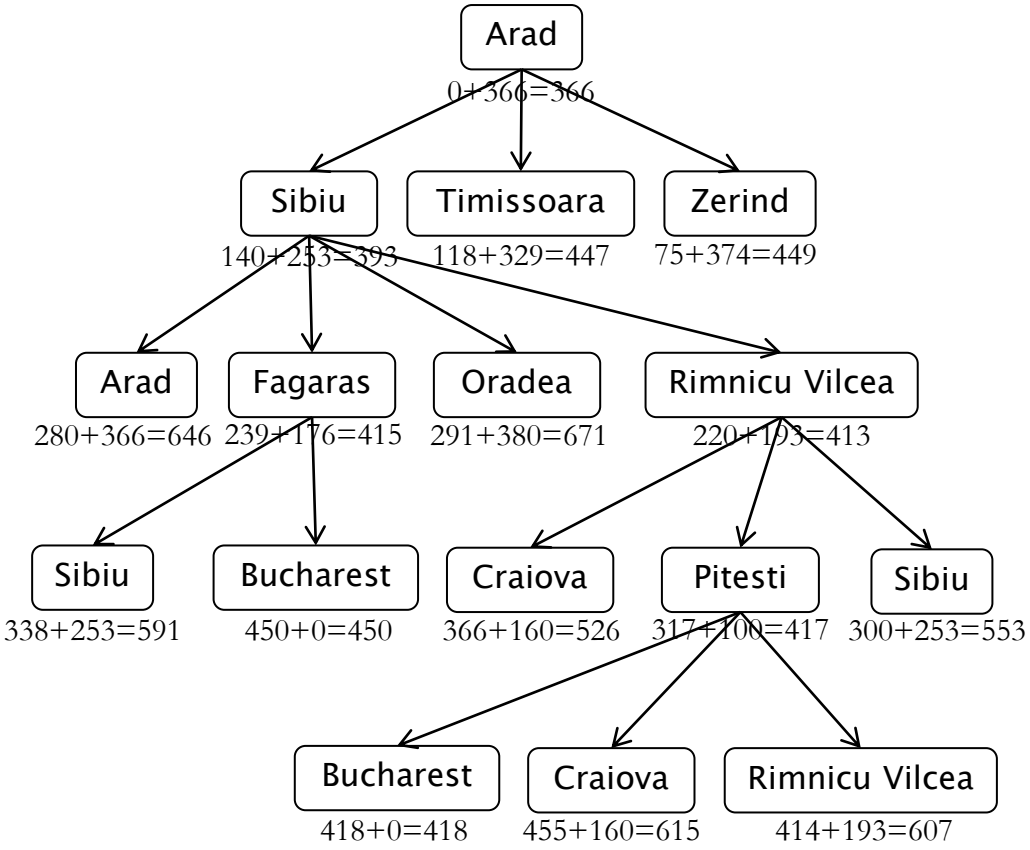
A*

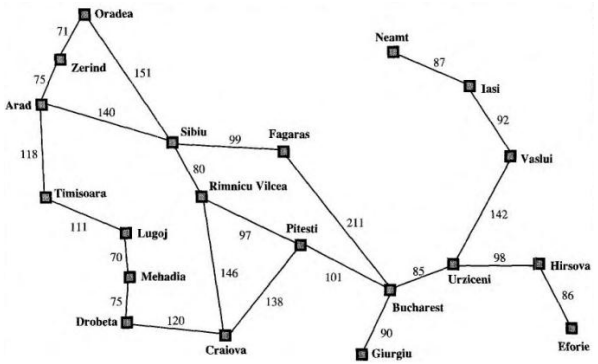
- **Estratégia:**
  - Combina o custo do caminho $g(n)$ com o valor da heurística $h(n)$
  - $g(n)$ = custo do caminho do nó inicial até o nó n
  - $h(n)$ = valor da heurística do nó n até um nó objetivo (distancia em linha reta no caso de distancias espaciais)
  - $f(n) = g(n) + h(n)$

- **É a técnica de busca mais utilizada.**

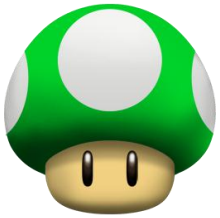


A*



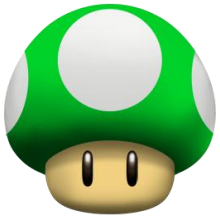


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374
Hirsova	151	Urziceni	80



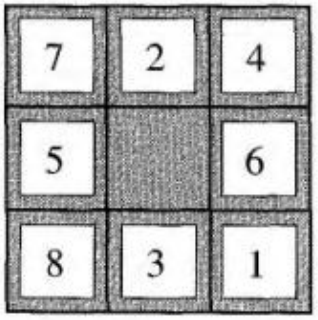
A*

- A estratégia é **completa** e **ótima**.
- **Custo de tempo:**
  - Exponencial com o comprimento da solução, porém boas funções heurísticas diminuem significativamente esse custo.
- **Custo memória:** $O(b^d)$
  - Guarda todos os nós expandidos na memória.
- Nenhum outro algoritmo ótimo garante expandir menos nós.

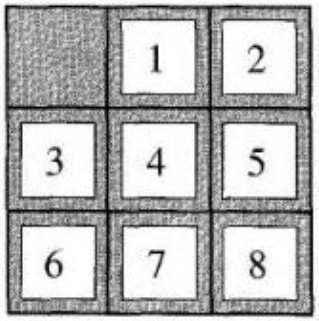


Definindo Heurísticas

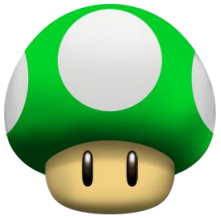
- Cada problema **exige** uma função heurística diferente.
- Não se deve superestimar o custo real da solução.
- Como escolher uma boa função heurística para o jogo 8-Puzzle?



Start State

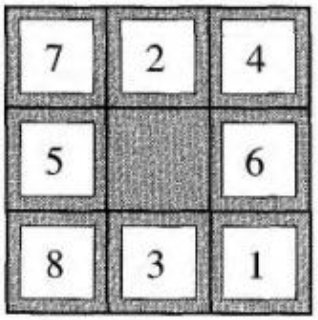


Goal State

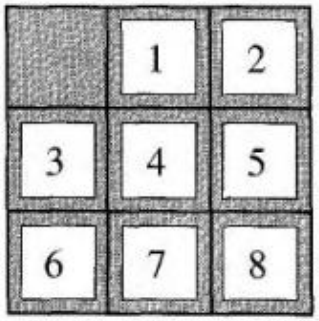


Definindo Heurísticas

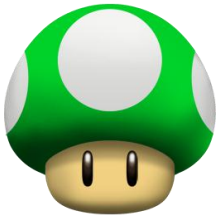
- Como escolher uma boa função heurística para o jogo 8-Puzzle?
  - h^1 = número de elementos fora do lugar.
  - h^2 = soma das distâncias de cada número à sua posição final (movimentação horizontal e vertical).
- Qual das heurísticas é melhor?



Start State

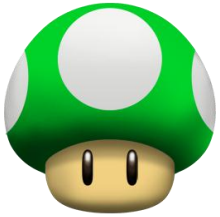


Goal State



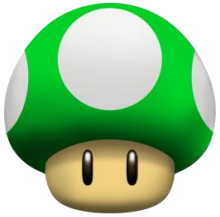
Exemplo – A*

	1	2	3	4	5
1	☺				X
2					
3					
4					



Exemplo – A*

- Qual é o espaço de estados?
- Quais são as ações possíveis?
- Qual será o custo das ações?

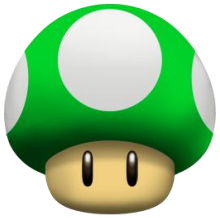


Exemplo – A*

- **Heurística do A*:** $f(n) = g(n) + h(n)$
  - $g(n)$ = custo do caminho
  - $h(n)$ = função heurística

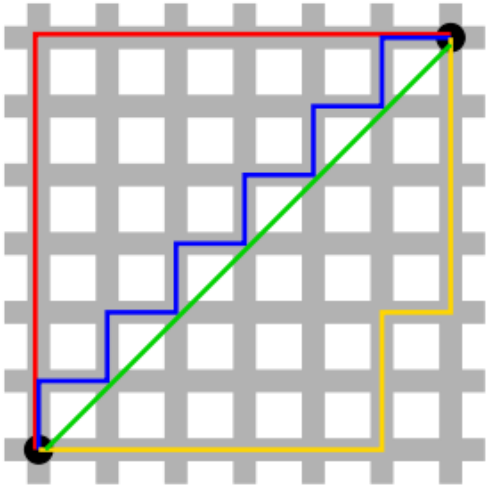
- Qual seria a função heurística $h(n)$ mais adequada para este problema?

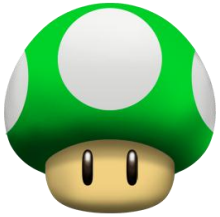
  - A distancia em linha reta é uma opção.



Exemplo – A*

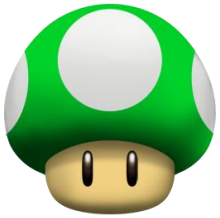
- Como calcular a heurística $h(n)$?
  - Distancia de Manhattan



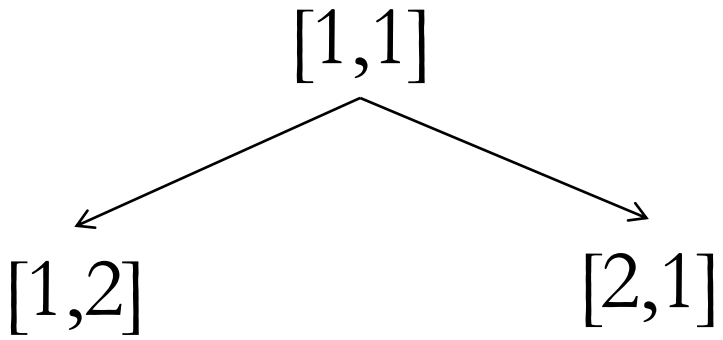


Exemplo – A*

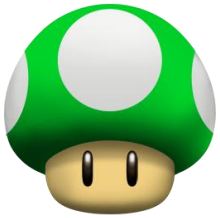
- O próximo passo é gerar a árvore de busca e expandir os nós que tiverem o menor valor resultante da função heurística $f(n)$.
  - $f(n) = g(n) + h(n)$



Exemplo – A*

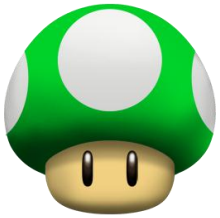


- [1,2] = $f(n)$ = ?? + ??
- [2,1] = $f(n)$ = ?? + ??

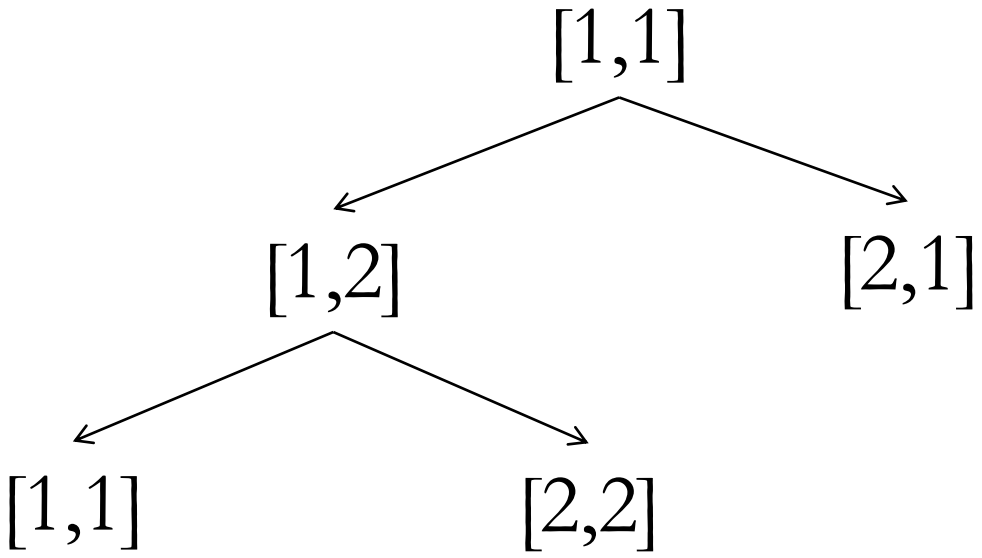


Exemplo – A*

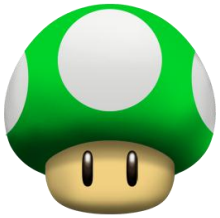
	1	2	3	4	5
1	☺				X
2					
3					
4					



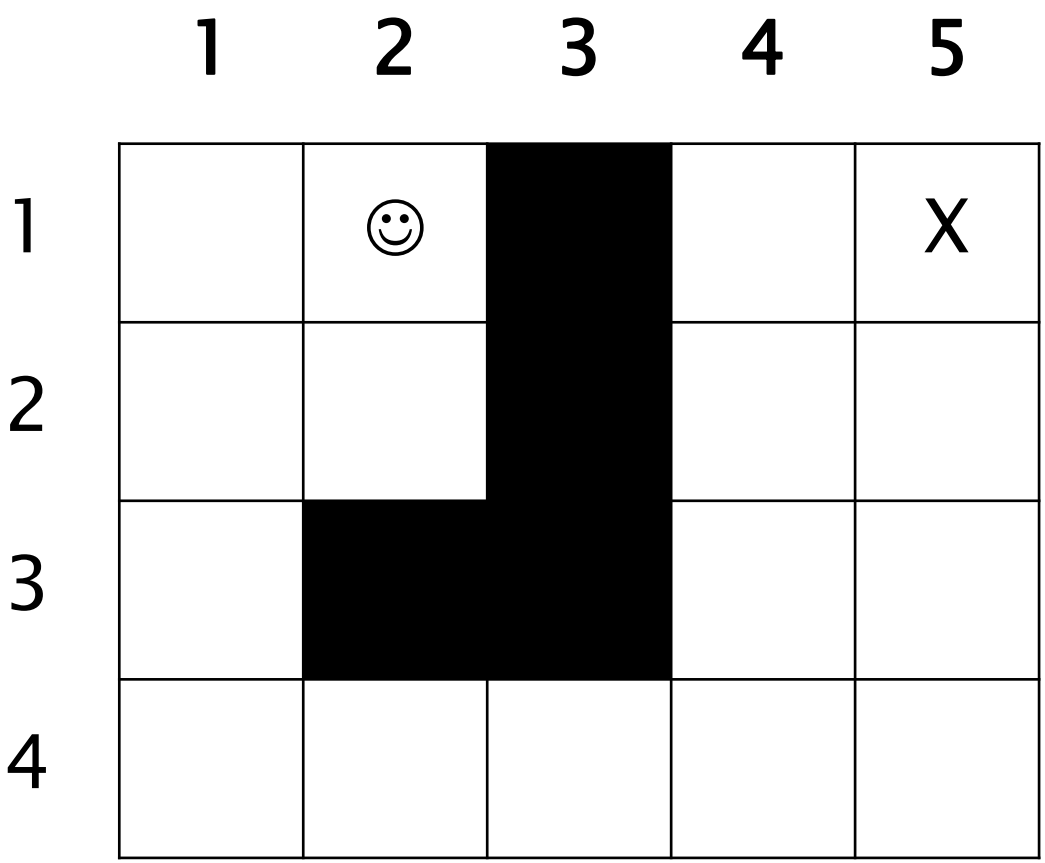
Exemplo – A*

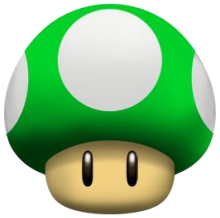


- [1,1] = $f(n)$ = ?? + ??
- [2,2] = $f(n)$ = ?? + ??



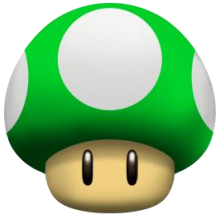
Exemplo – A*





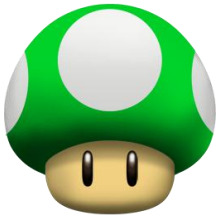
Busca Local

- Em muitos problemas o **caminho para a solução é irrelevante**.
  - **Jogo das n-rainhas:** o que importa é a configuração final e não a ordem em que as rainhas foram acrescentadas.
  - **Outros exemplos:**
    - Projeto de Circuitos eletronicos;
    - Layout de instalações industriais;
    - Escalonamento de salas de aula;
    - Otimização de redes;



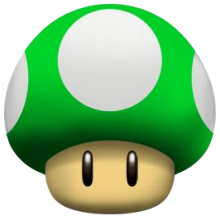
Busca Local

- Algoritmos de busca local operam sobre um **unico estado corrente**, ao invés de vários caminhos.
- Em geral se movem apenas para os vizinhos desse estado.
- O caminho seguido pelo algoritmo não é guardado.



Busca Local

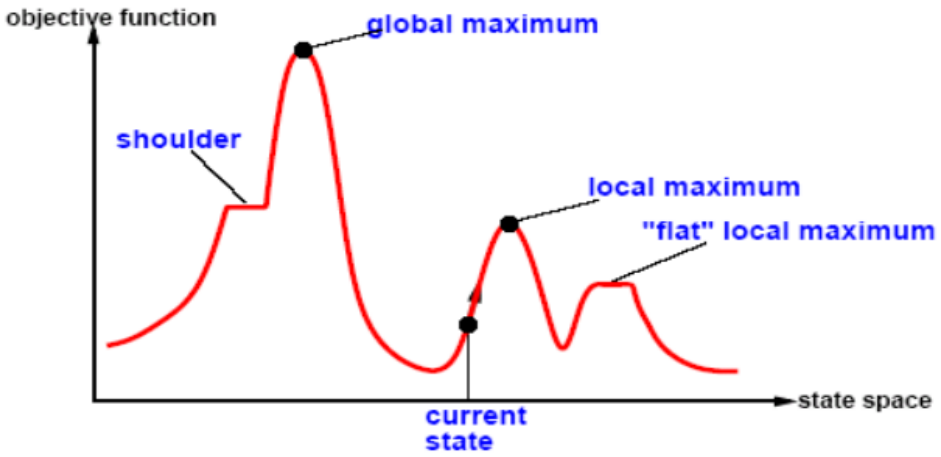
- **Vantagens:**
  - Ocupam pouquíssima memória (normalmente constante).
  - Podem encontrar soluções razoáveis em grandes ou infinitos espaços de estados.
- **São uteis para resolver problemas de otimização.**
  - Buscam por estados que atendam a uma função objetivo.

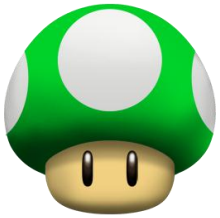


Busca Local

- **Panorama do Espaço de Estados**:

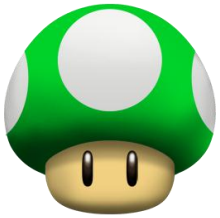
- Location = Estado;
- Elevation = Valor de custo da função heurística;
- Busca-se o maximo ou minimo global;





Subida de Encosta (Hill-Climbing)

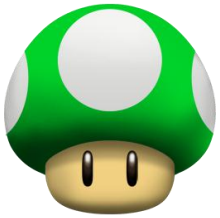
- **Estratégia:**
  - Se **move** de forma contínua no sentido do valor crescente da heurística;
  - **Termina** ao alcançar um pico em que nenhum vizinho possui um valor mais alto;
  - **Não mantém nenhuma árvore de busca**, somente o estado e o valor da função objetivo;
  - Não examina antecipadamente valores de estados além de seus vizinhos imediatos;
  - "É como tentar encontrar o topo do monte Everest em meio a um denso nevoeiro e sofrendo de amnésia".



Subida de Encosta (Hill-Climbing)

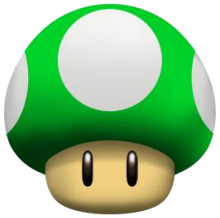
Processo:

- Inicialize (aleatoriamente) o ponto x no espaço de estados do problema.
- A cada iteração, um novo ponto x’ é selecionado aplicando-se uma pequena perturbação no ponto atual, ou seja, selecionando-se um ponto x’ que esteja na vizinhança de x.
- Se este novo ponto apresenta um melhor valor para a função de avaliação, então o novo ponto torna-se o ponto atual.
- O método é terminado quando nenhuma melhora significativa é alcançada, um número fixo de iterações foi efetuado, ou um objetivo foi atingido.



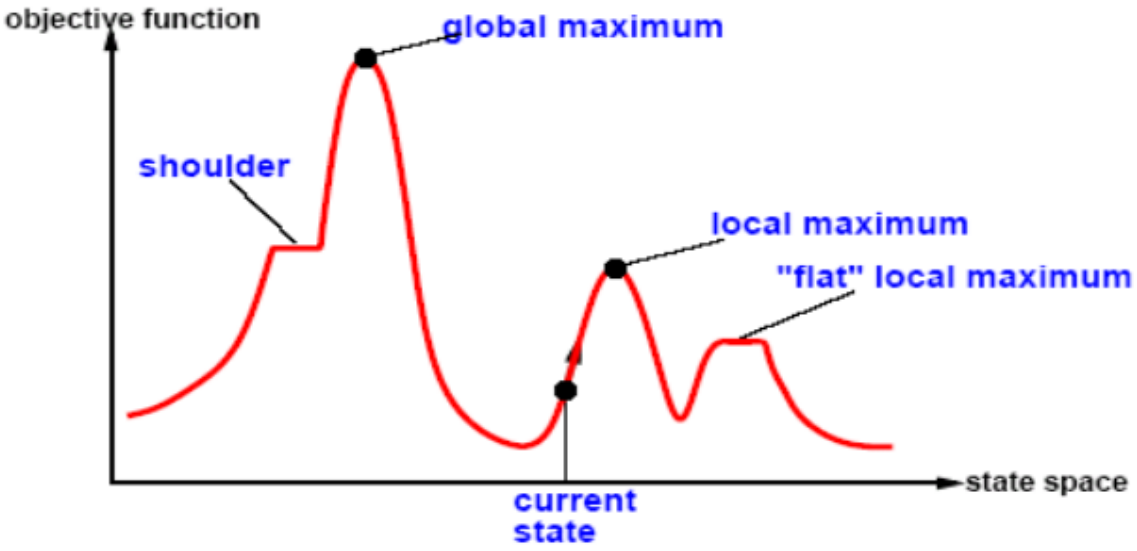
Pseudocódigo – Hill-Climbing

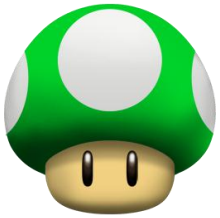
```
Função Hill-Climbing(Problema) retorna um estado que é o maximo local
Inicio
  EstadoAtual ← FazNó(Problema[EstadoInicial])
  loop do
    Vizinho ← SucessorDeMaiorValor(EstadoAtual)
    se Vizinho[Valor] for menor ou igual EstadoAtual[Valor] então
      retorna EstadoAtual
    EstadoAtual ← Vizinho
Fim
```



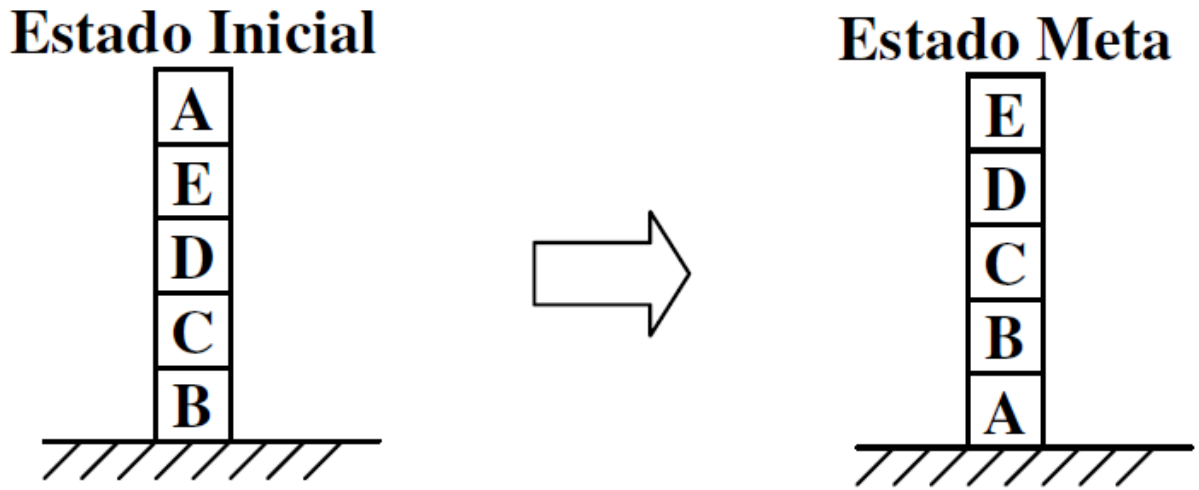
Subida de Encosta (Hill-Climbing)

- Problemas:
  - **Máximos Locais**:

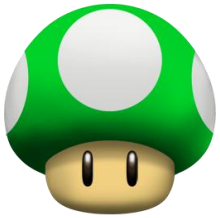




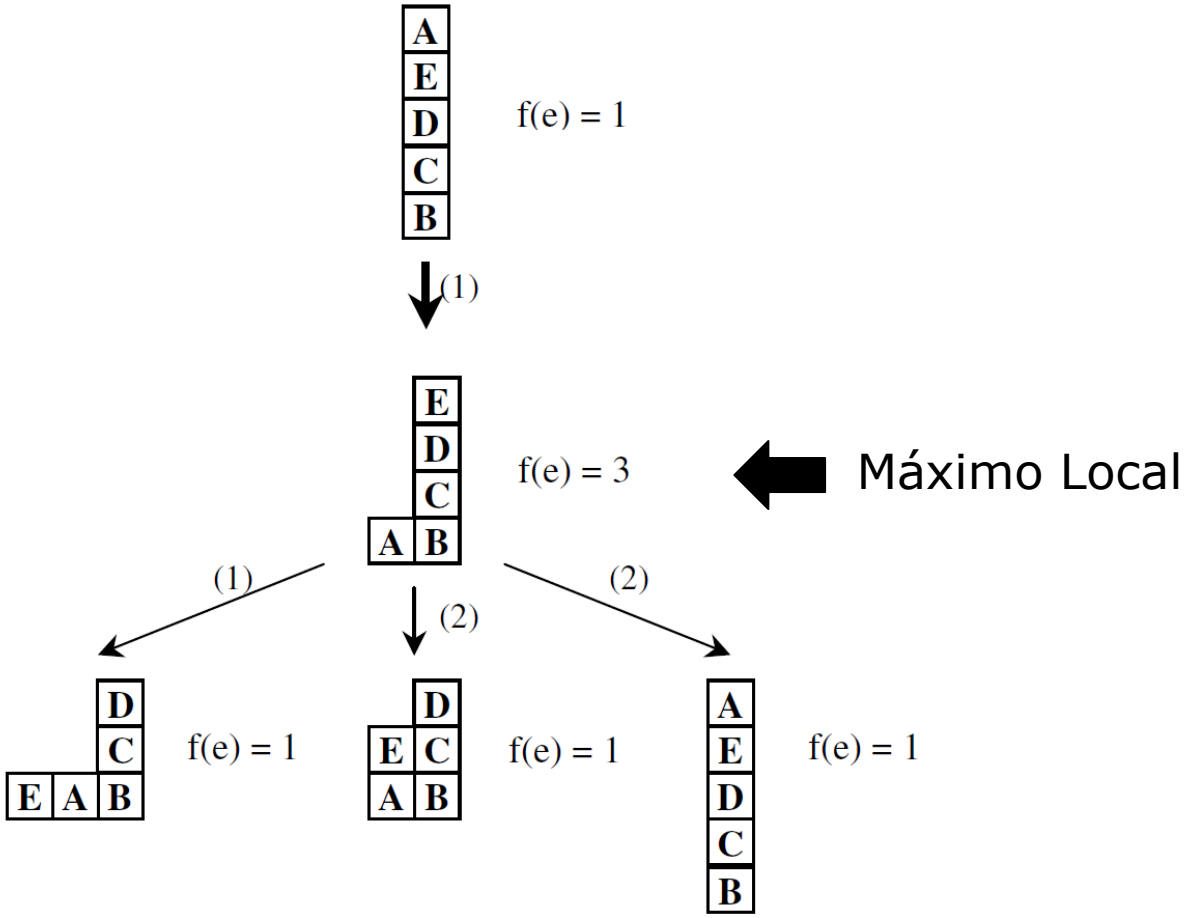
Subida de Encosta – Exemplo

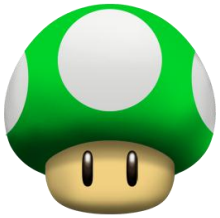


- **Ações Possíveis:**
  - Pegar um bloco e colocar ele sobre a mesa.
  - Pegar um bloco e colocar ele sobre outro bloco.
- **Heurística:**
  - +1 para cada bloco em cima do bloco onde ele deve estar.
  - -1 para cada bloco em cima do bloco errado.



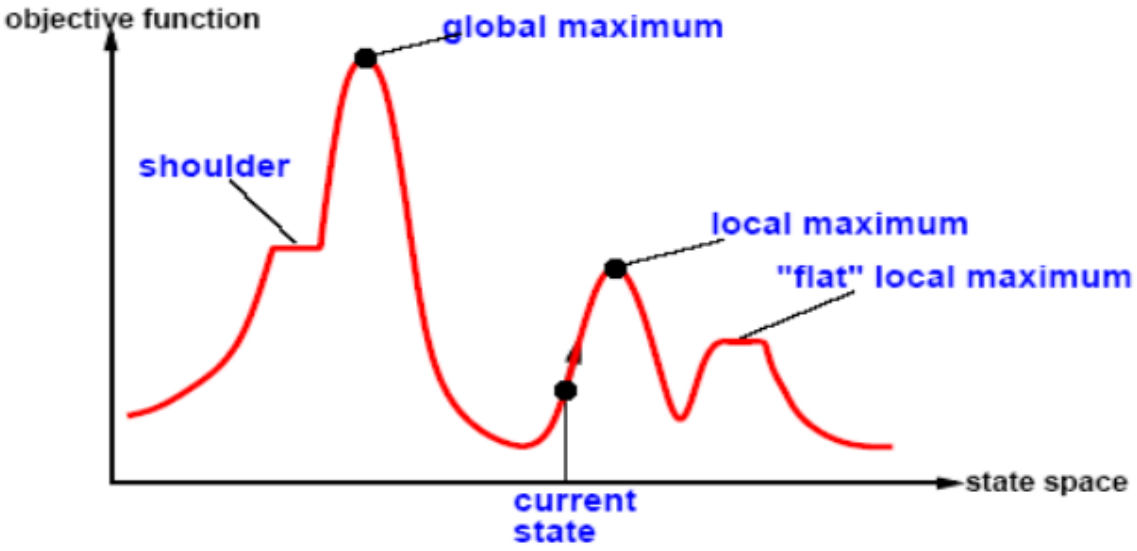
Subida de Encosta – Exemplo

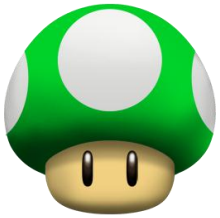




Subida de Encosta (Hill-Climbing)

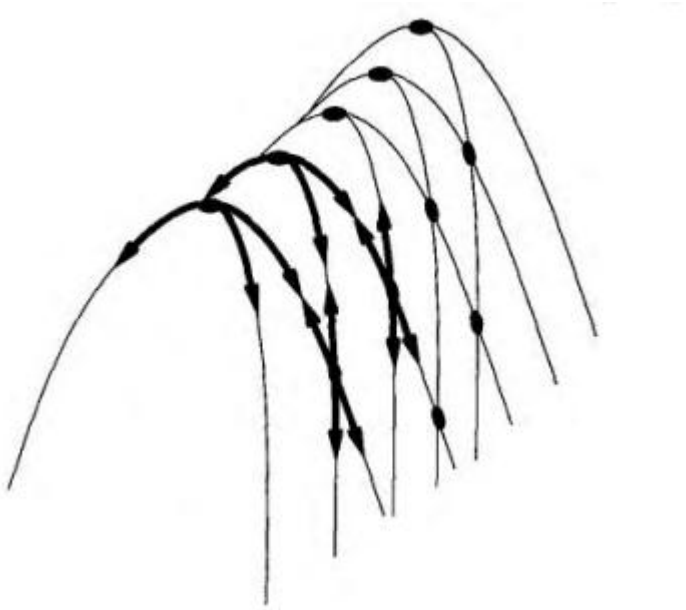
- Problemas:
  - **Planícies**:

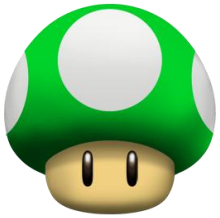




Subida de Encosta (Hill-Climbing)

- Problemas:
  - **Encostas e Picos**:





Subida de Encosta (Hill-Climbing)

- **Não é ótimo e não é completo.**
- **Variações:**
  - Random-Restart Hill-Climbing;