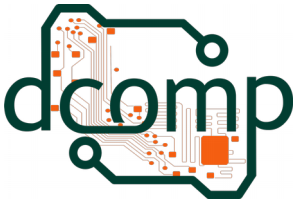
Universidade Federal do Espírito Santo
Centro de Ciências Agrárias – CCENS UFES
Departamento de Computação



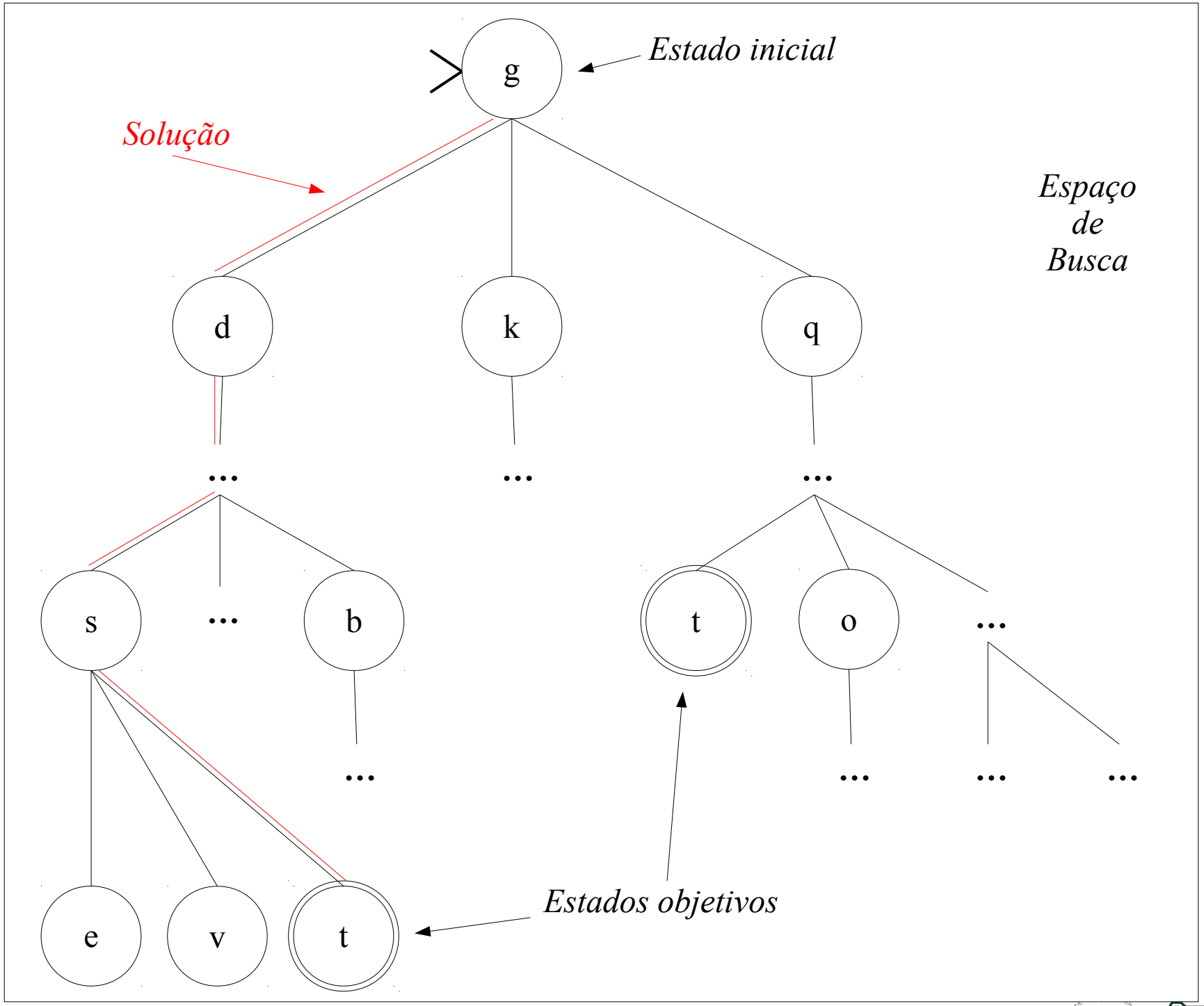
Resolução de Problemas com Métodos de Busca

Inteligência Artificial

Site: http://jeiks.net
E-mail: jacsonrcsilva@gmail.com

Soluções de problemas como Busca

- Ao buscar um objetivo, estamos em um determinado *estado*, sendo:
  - O *estado inicial* é onde iniciamos a busca;
  - O estado que soluciona o problema é o *estado objetivo*.
- Busca:
  - Processo de procurar uma *sequência de ações* que alcançam o estado **objetivo**.
  - Um algoritmo de busca recebe um **problema** como entrada e devolve uma **solução** sob a forma de uma *sequência de ações*.
  - Com a solução encontrada, as ações recomendadas podem ser executadas: Fase de **execução**.
  - O espaço de um problema é seu *Espaço de Busca*.
- Busca com agentes:
  - Criar um projeto para: "formular, buscar, executar".
  - Depois de formular um objetivo e um problema a resolver, o agente chama um procedimento de *busca* para resolvê-lo.
  - Em seguida, ele utiliza a *solução* para orientar suas ações:
    - Efetuando uma ação da sequência fornecida (geralmente a primeira da lista);
    - Removendo esse passo da sequência.
    - Após executar a solução, o agente formulará um novo objetivo.


g
Estado inicial
Solução
Espaço
de
Busca
d
k
q
s
b
t
o
e
v
t
Estados objetivos

Soluções de problemas como Busca

- Um **problema** pode ser definido formalmente por cinco componentes:
  - O **estado inicial** em que o agente começa.
  - Uma descrição das **ações** possíveis que estão disponíveis para o agente.
    - *AÇÕES(estado)* devolve um conjunto de ações que podem ser executadas em s.
    - Ações que levam do estado atual ao seu estado sucessor.
    - São ações **aplicáveis** no *estado*.
  - Uma descrição do que cada ação faz (**modelo de transição**).
    - *RESULTADO(estado, ação)* devolve o estado que resulta de executar uma ação a em estado.
  - O **teste de objetivo**, que determina se um estado é um estado objetivo.
    - Às vezes existe um conjunto explícito de estados objetivo possíveis,
      e o teste simplesmente verifica se o estado dado é um deles.
  - Uma função de **custo de caminho** que atribui um custo numérico a cada caminho.
    - *CUSTO(estado, ação, sucessor)* devolve o **custo de passo** para alcançar o sucessor com a ação dada.
    - O agente de resolução de problemas escolhe uma função de custo que reflete sua própria medida de desempenho.
- **Solução ótima**:
  - Solução com menor custo de caminho entre todas as soluções.
  - A qualidade da solução é medida pela função de custo de caminho.

Busca guiada por **Dados** ou **Objetivos**

- Abordagens para fazer uma busca:
  - De-cima-para-baixo:
    - Encadeamento para frente;
    - Busca guiada por **Dados**;
    - Parte de um estado inicial e usa ações permitidas para alcançar o objetivo.
  - De-baixo-para-cima
    - Encadeamento para trás;
    - Busca guiada por **Objetivos**;
    - Começa de um objetivo e volta para um estado inicial, vendo quais deslocamentos poderiam ter levado ao objetivo.

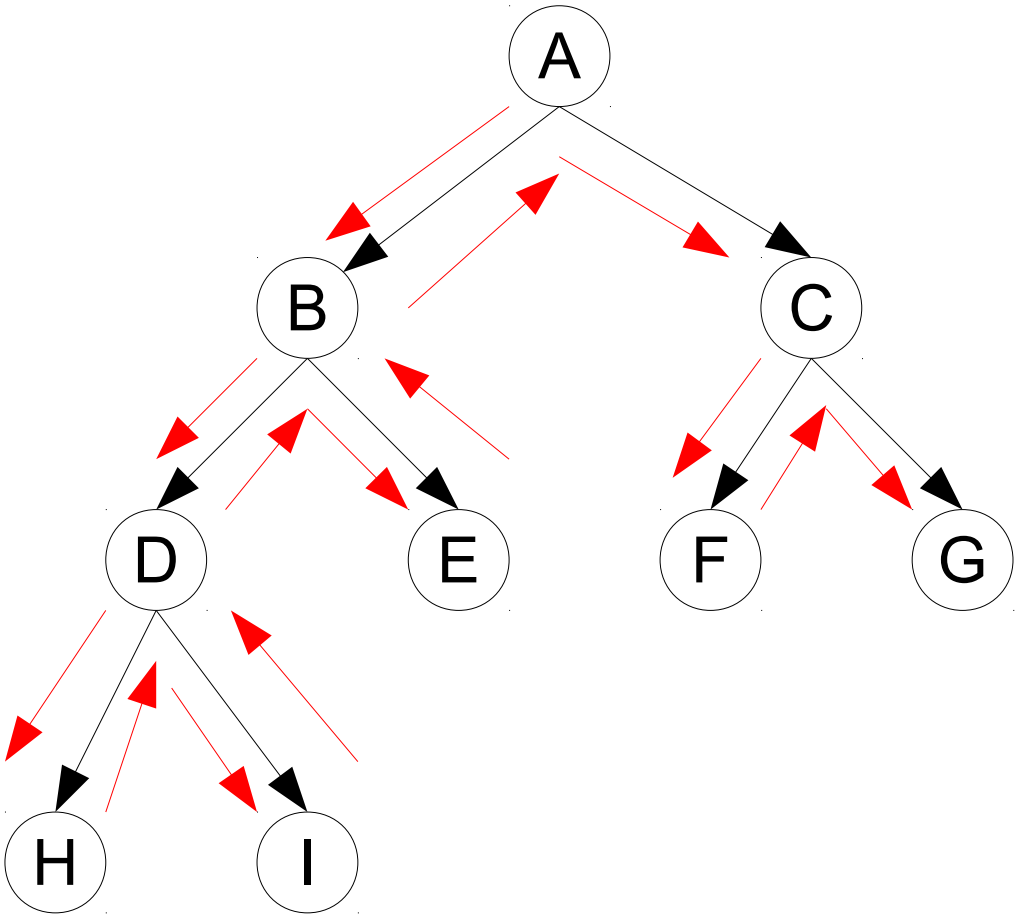
Busca cega (sem heurística)

- **Gerar e Testar**:
  - A mais simples abordagem de busca;
  - Funcionamento: gerar cada nó no espaço de busca e testá-lo para verificar se este é um nó objetivo;
  - É a forma mais simples de *busca de força bruta* ou *busca exaustiva*;
- Precisa de um *Gerador* que satisfaça:
  - *Ele deve ser completo*, garantir que todas as soluções possíveis serão geradas. Pois assim não descartará uma solução adequada;
  - *Ele não deve ser redundante*, não gerando a mesma solução duas vezes;
  - *Ele deve ser bem informado*, só deve propor soluções adequadas e que combinem com o espaço de busca.

Busca cega (sem heurística)

- Profundidade:
  - Segue cada caminho até sua maior profundidade antes de seguir para o próximo caminho
  - Se a folha não representar um estado objetivo,
    - A busca retrocederá ao primeiro nó anterior que tenha um caminho não explorado
  - Utiliza um método chamado de **retrocesso cronológico**:
    - Volta na árvore de busca, uma vez que um caminho sem saída seja encontrado
    - É assim chamado por desfazer escolhas na ordem contrária ao momento em que foram tomadas
  - É um método de *busca exaustiva* ou de *força bruta.*

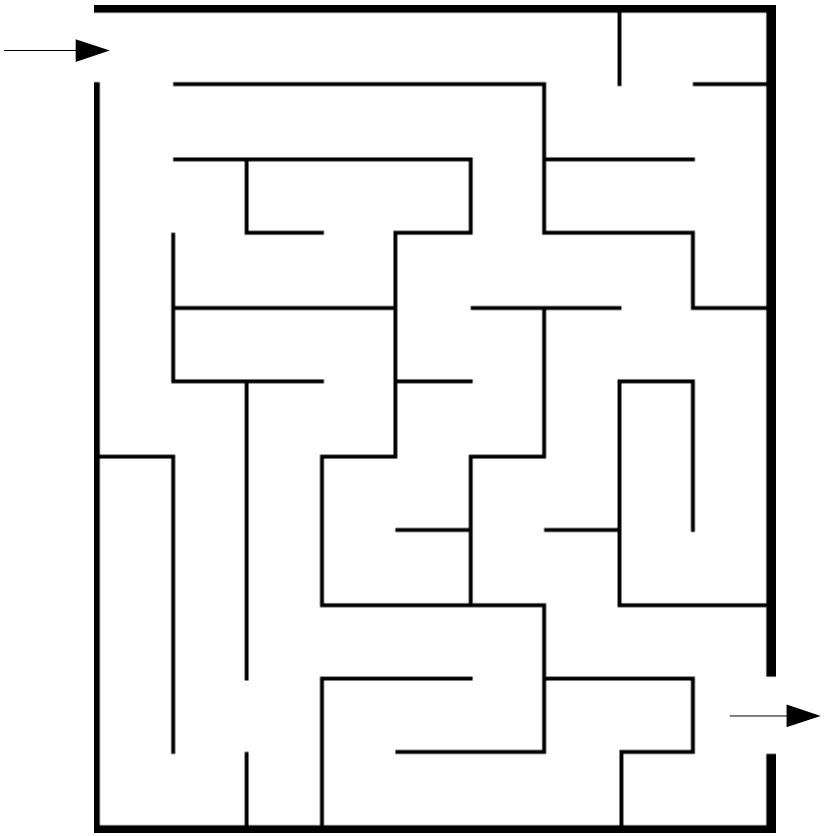
Exemplo



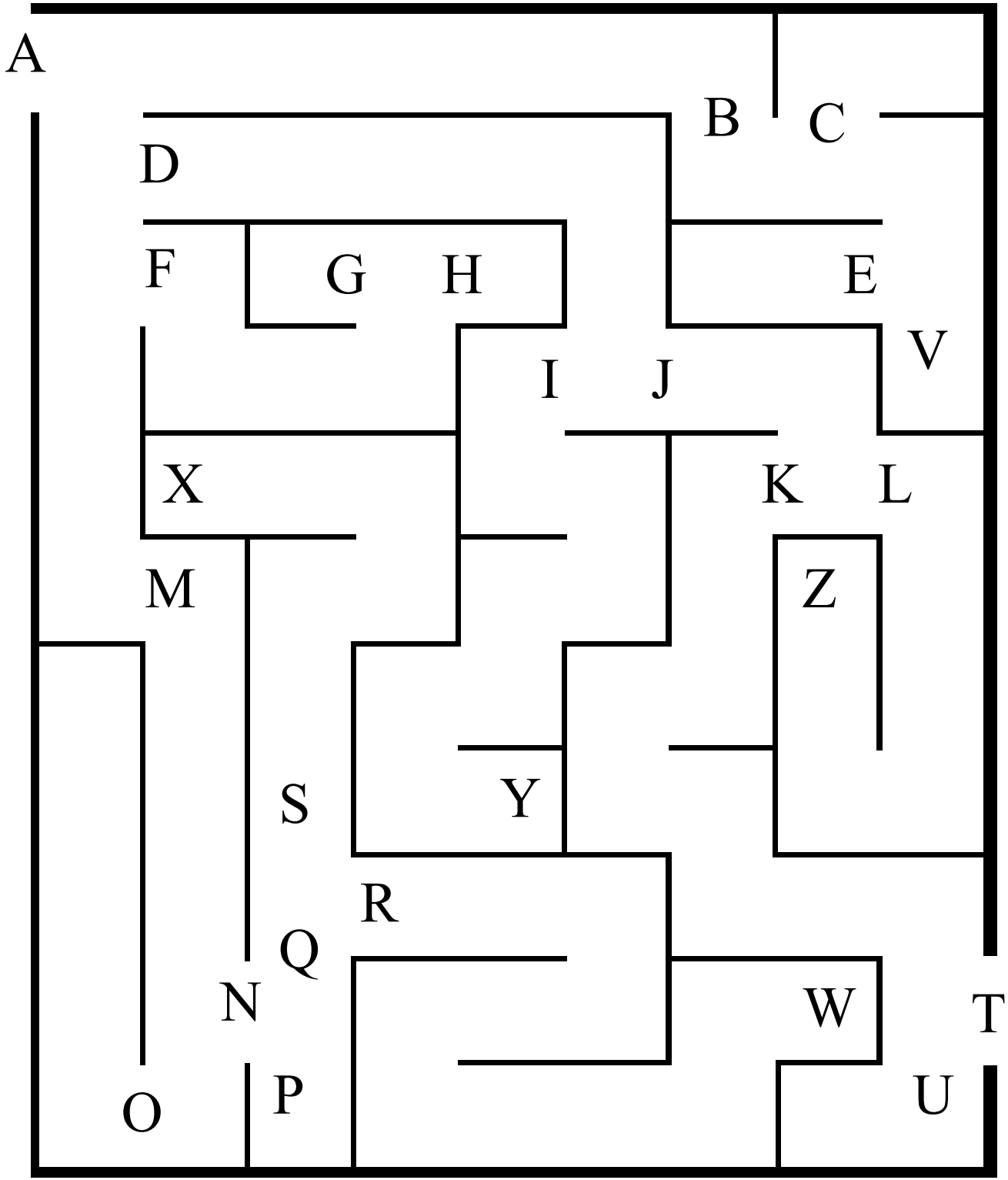
Ordem: A, B, D, H, I, E, C, F, G

Humanos utilizam busca em profundidade

- É o modo mais fácil e natural;
- Exemplos:
  - Percorrendo um labirinto;



  - Comprando um presente em um shopping;



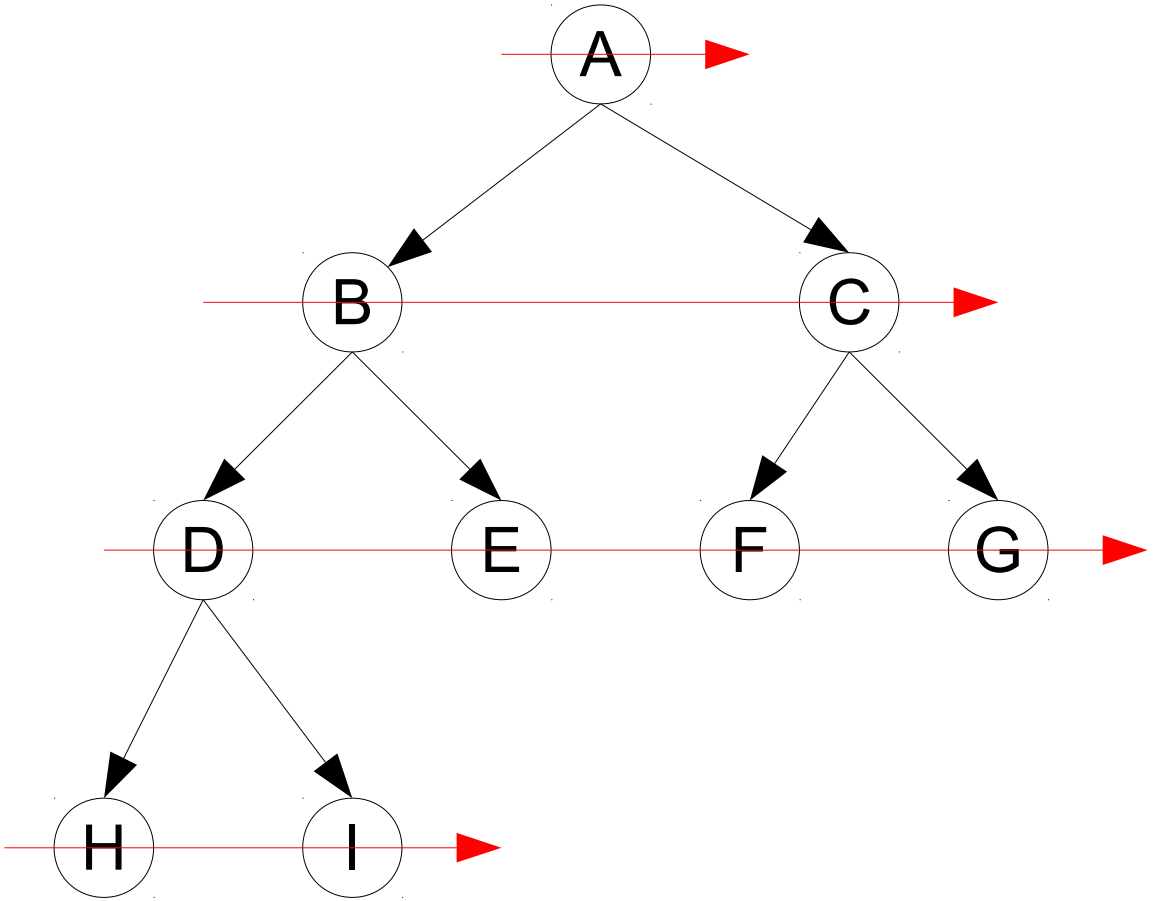
Mapa em Prolog:

```
conexao(a,b).
conexao(a,d).
conexao(a,f).
conexao(a,m).
...
...
...
```

Busca cega (sem heurística)

- Largura (extensão):
  - Percorre a árvore em largura ao invés de profundidade.
  - Começam examinando todos os nós de um nível abaixo do nó raiz.
  - Se não encontrar o objetivo, buscam um nível abaixo.
  - Melhor em árvores que tenham caminhos mais profundos.
  - Utilizado em **árvores de jogos**.

Exemplo



Ordem: A,B,C,D,E,F,G,H,I

Implementando a busca...

```
Profundidade:
   Lista  = []
   Estado = no_raiz;
   repita:
      se eh_objetivo( estado )
         retorne SUCESSO
      senão
         inserirNaFrenteDaLista(sucessores(estado))

      se Lista estiverVazia
         retorne FALHA

      Estado = Lista[0]
      RemoverPrimeiroItemDaLista
```

Implementando a busca...

Profundidade:

```
Lista  = []
Estado = no_raiz;
repita:
   se eh_objetivo( estado )
      retorne SUCESSO
   senão
      inserirNaFrenteDaLista(sucessores(estado))

   se Lista estiverVazia
      retorne FALHA

   Estado = Lista[0]
   RemoverPrimeiroItemDaLista
```

Largura:

substituir a função *inserirNa**Frente**DaLista*
por *inserir**Atrás**DaLista*

Propriedades dos métodos de busca

- Complexidade:
  - Ligado ao tempo e espaço utilizados na busca;
- Completude:
  - Se é completo, ou seja, se sempre acha um objetivo (qualquer objetivo);
  - Obs.: se houver objetivo;
- Quanto a ser ótimo:
  - Garantir achar o melhor objetivo que exista;
  - Não garante que seja pelo menor caminho ou tempo.
- Monótono:
  - Sempre chega a um dado estado pelo caminho mais curto possível.
- Admissibilidade:
  - Garantir achar a melhor solução pelo melhor caminho.
- Irrevogabilidade:
  - Não retrocedem, examinando assim somente um caminho.

Medição de desempenho

- Completeza:
  - O algoritmo oferece a garantia de encontrar uma solução quando ela existir?
- Otimização:
  - A estratégia encontra a solução ótima*?

  **menor custo entre todas as soluções*
- Complexidade de tempo:
  - Quanto tempo ele leva para encontrar uma solução?
- Complexidade de espaço:
  - Quanta memória é necessária para executar a busca?

Medição de desempenho

- A complexidade de tempo e a complexidade de espaço de memória:
  - São consideradas em relação a alguma medida da dificuldade do problema.
  - Em computação:
    - Medida típica: tamanho do grafo do espaço de estados, | Vértices | + | Arestas |.
    - Apropriado quando o grafo for uma estrutura de dados explícita.
  - Especificamente em IA:
    - Grafo sempre representado por:
      - estado inicial, ações, modelo de transição, frequentemente infinito.
    - Por essas razões, a complexidade é expressa em termos de três quantidades:
      - b, o fator de ramificação ou número máximo de sucessores de qualquer nó;
      - d, a profundidade do nó objetivo menos profundo; e
      - m, o comprimento máximo de qualquer caminho no espaço de estados.
    - Medição do tempo: número de nós gerados durante a busca.
    - Medição do espaço: número máximo de nós armazenados na memória.

Busca Informada

- Possuem **heurísticas** que ajudam na busca.
- É fornecida uma orientação (guia) sobre onde procurar por soluções.
- *Heurística* pode ser definida como:
  - A utilização de informações que indicam melhor qual caminho a seguir.

    Ex: pesquisar em todas as lojas por calças, ou somente nas lojas que trabalham com tecidos?
  - Regras para escolher as arestas que tem maior probabilidade de levar a uma solução do problema.
- A busca informada possui uma *função de avaliação heurística*:
  - É aplicada a um nó e retorna um valor que representa:
    - uma boa estimativa da distância entre o nó e o objetivo
    - Ex: Se para dois nós *m* e *n*, a função retorna $f(m) < f(n)$, então deve ser o caso que *m* é mais provável de estar em um *caminho ótimo* para o nó objetivo

Busca Informada

- Em quais problemas utilizar busca informada:
  - Quando o problema pode não ter uma solução exata por causa das ambiguidades:
    - Na formulação do problema; ou
    - Nos dados disponíveis.
  - Quando o problema tiver uma solução exata, mas com o custo computacional de encontrá-la alto, proibitivo.

Subida da colina (de encosta) (*Hill-Climbing*)

- Caso de estudo:
  - Se tentar escalar uma montanha em dia de neblina,

    com um altímetro, mas sem mapa,

    você utilizaria uma abordagem de subida da colina
  - É uma Abordagem *Gerar e Testar*;
  - Como proceder:
    - Verificar a altura a alguns centímetros de sua posição.
    - Você pode olhar primeiro a norte, depois sul, depois oeste e depois leste.
    - Assim que encontrar a **primeira** posição que o leve para uma altura maior que a atual, vá para lá e repita esses passos.
    - Note que se a posição for a norte, não serão nem avaliados o sul, oeste ou leste.
    - Se todas as posições o levam para mais baixo de onde está, você assume que chegou ao topo.

Você sempre chegará ao topo?

Você sempre chegará ao topo?

Subida da Colina pela Encosta de Maior Aclive

Funciona da mesma forma que a Subida da Colina, porém

sempre verifica **todas** as quatro **posições** em volta e escolhe a posição que seja mais alta

Subida da Colina pela Encosta de Maior Aclive (Ben Coppin)

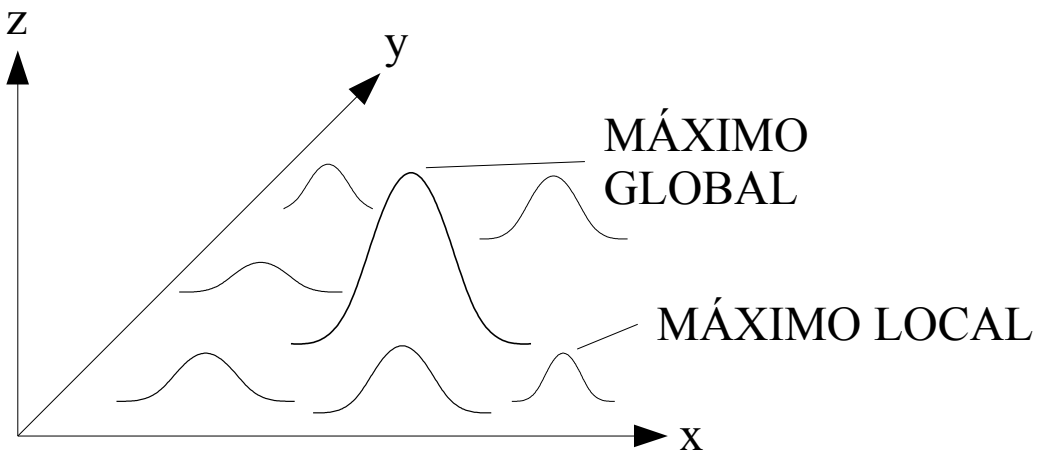
```
colina:
   lista = []
   estado = no_raiz
   repita:
      se É_Objetivo(estado)
         retorne SUCESSO
      senão
         aux = Ordenar( sucessores(estado) )
         InserirNaFrenteDaLista( aux )

      se lista == []
         retorne FALHA

      Estado = fila[0]
      RemoverPrimeiroItemDa ( lista )
```

Subida da encosta (Russel, Norvig)

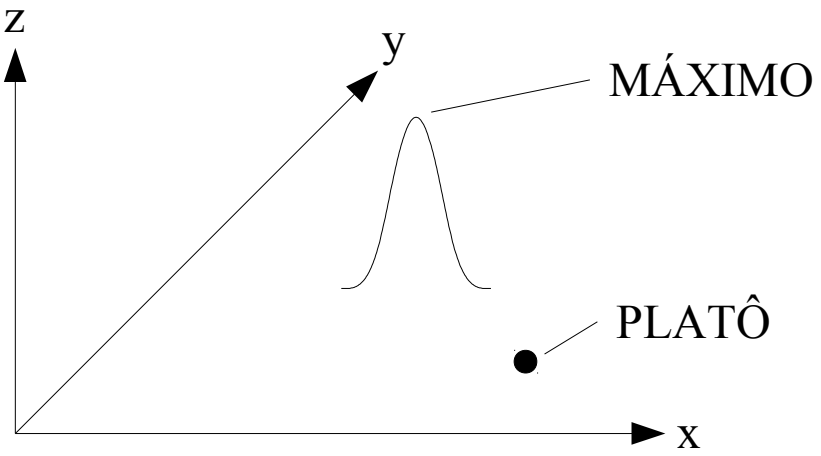
```
Subida_na_Encosta:
   estadoAtual = no_raiz
   repita:
      aux = Ordenar( sucessores(estadoAtual) )
      vizinho = aux[0]
      se valor(vizinho) ≤ valor(estadoAtual):
         retorna estadoAtual
      senão
         estadoAtual ← vizinho
```

Problemas encontrados

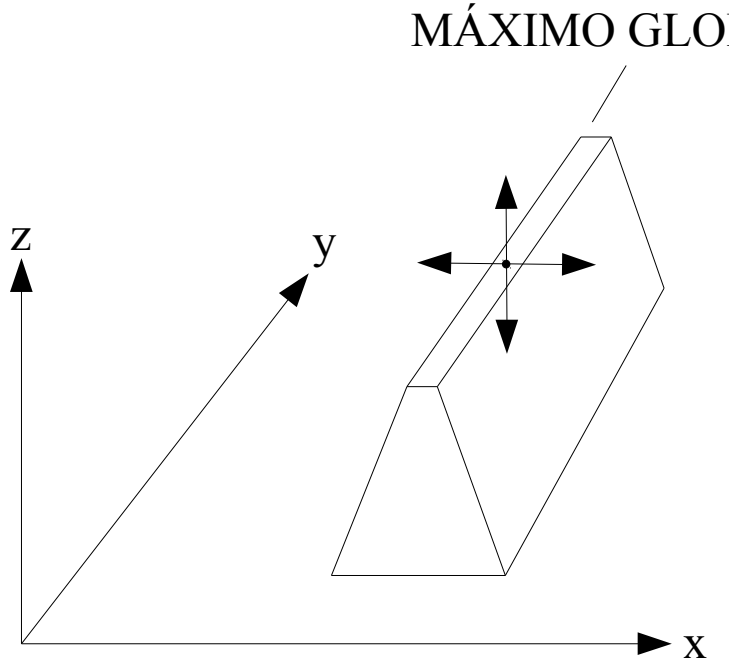
- Contrafortes:
  - Máximo local;
  - Parte de um espaço de busca que parece ser preferível as partes em torno dele.
- Platôs:
  - Região em um espaço de busca na qual todos os valores são os mesmos.
- Cristas:
  - É uma região longa e estreita de terras altas com terras baixas em ambos os lados.



CONTRAFORTES



PLATÔS



CRISTAS ou CORDILHEIRAS

Melhorando o método

- Problemas do Subida da Colina:
  - Nunca faz movimentos “encosta abaixo” em direção a estados com valor mais baixo;
  - É incompleto e pode ficar preso em um máximo local.
  - Mas um percurso puramente aleatório, mesmo sendo completo, é extremamente ineficiente.
- Têmpera simulada:
  - Combina a subida da coluna com um percurso aleatório;
  - Busca ter eficiência e completeza.
  - Em metalurgia, a têmpera é o processo usado para temperar ou endurecer metais e vidro aquecendo-os a alta temperatura e depois esfriando-os gradualmente, permitindo assim que o material alcance um estado cristalino de baixa energia.

Têmpera Simulada

- Vamos mudar nosso ponto de vista de subida de encosta para descida de gradiente (minimização do custo).
- Caso de estudo:
  - Imagine a tarefa de colocar uma bola de pingue-pongue na fenda mais profunda em uma superfície acidentada.
  - Se simplesmente deixarmos a bola rolar, ela acabará em um mínimo local.
  - Se agitarmos a superfície, poderemos fazer a bola quicar para fora do mínimo local.
  - O artifício é agitar com força suficiente para fazer a bola sair dos mínimos locais, mas não o bastante para desalojá-la do mínimo global.
- Comparação com a Têmpera Simulada:
  - A solução é começar a agitar com força (alta temperatura) e
  - Depois reduzir gradualmente a intensidade da agitação (baixar a temperatura).

```
Subida_na_Encosta(problema):
   estadoAtual = estadoInicial(problema)
   repita:
      aux = Ordenar( sucessores(estadoAtual) )
      próximo = aux[0]
      se valor(próximo) ≤ valor(estadoAtual):
         retorna estadoAtual
      senão
         estadoAtual ← próximo
```

```
Têmpera_Simulada(problema, escalonamento):
   //escalonamento: mapeamento de tempo para temperatura
   estadoAtual = estadoInicial(problema)
   para t = 1 até ∞ faça:
      T = escalonamento[t] //diminui com o tempo
      se T == 0:
         retorna estadoAtual
      próximo ← Aleatório( sucessores(estadoAtual) )
      ΔE ← valor(próximo) - valor(estadoAtual)
      r  ← [0,1]
      se ΔE > 0 ou probabilidade e^(ΔE/T) > r:
         estadoAtual ← próximo
```

Busca pelo Primeiro Melhor (Busca gulosa)

- Parecido com à Subida na Colina, porém
  - Avalia os estados usando somente a função heurística:

    f(estado) = h(estado)
  - A lista inteira de próximas posições é ordenada após receber a inserção de novos caminhos,
  - *em vez de inserir um conjunto de dados ordenados.*

- Significado:
  - ele segue o melhor caminho disponível na árvore.

Implementação do Primeiro Melhor

```
Colina:
   lista = []
   estado = no_raiz
   repita:
      se É_Objetivo(estado)
         retorne SUCESSO
      senão
         InserirNaFrenteDaLista( sucessores(estado) )
         Ordenar( lista )

      se lista == []
         retorne FALHA

      Estado = fila[0]
      RemoverPrimeiroItemDa ( lista )
```

Busca com Limite Superior (Feixe Local – Local Beam)

- Utiliza um limiar de tal modo que apenas os poucos melhores caminhos são considerados a cada nível;
- Muito eficiente na utilização de memória;
- Seria útil para explorar um espaço de busca com alto fator de ramificação.

Busca com Limite Superior

```
colina:
   lista = []
   estado = no_raiz
   repita:
      se É_Objetivo(estado)
         retorne SUCESSO
      senão
         InserirNoFinalDaLista( sucessores(estado) )
         SelecionarMelhoresCaminhos( lista, n )
         //remove todos, exceto os n melhores caminhos da lista
         //na versão estocástica: escolhe n sucessores aleatórios

      se lista == []
         retorne FALHA
      Estado = fila[0]
      RemoverPrimeiroItemDa ( lista )
```

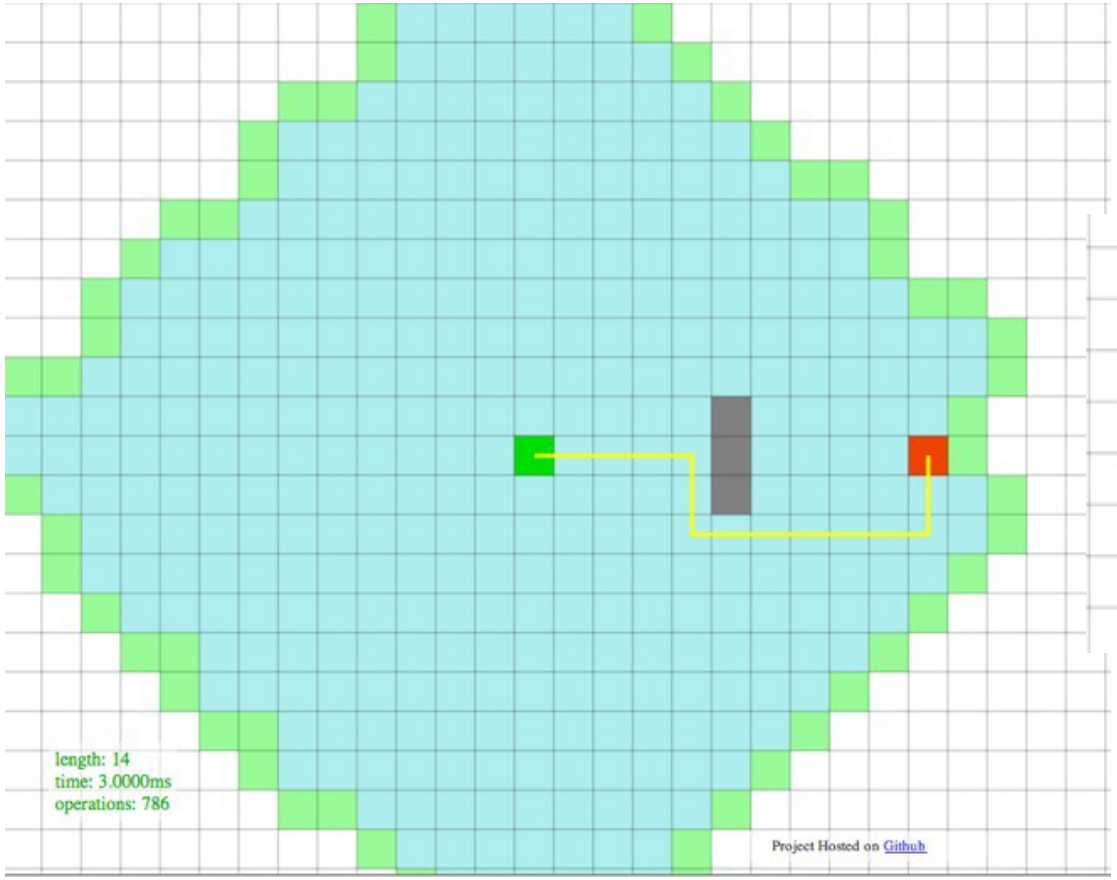
Algoritmo A*

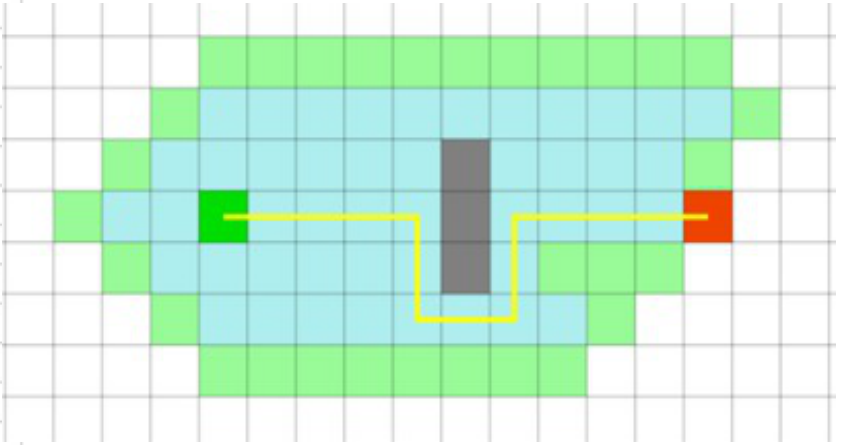
- Semelhante à busca pelo primeiro melhor, mas utiliza a seguinte função para avaliar nós:

  f (nó) = g (nó) + h (nó)
  g (nó) → custo do caminho que leva ao nó atual
  h (nó) → subestimativa da distância desse nó até um estado objetivo.
  É uma heurística que prevê a distância desse nó até o nó objetivo

  - f (nó) = *função de avaliação baseada em caminho*
- Se h(nó) for sempre uma subestimativa com valores corretos, A* será ótimo:
  - pois será garantido encontrar o caminho mais curto.

Visualização dos Métodos de Busca





Origem: <http://qiao.github.io/PathFinding.js/visual>