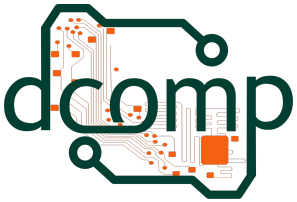
Universidade Federal do Espírito Santo
Centro de Ciências Agrárias – CCA UFES
Departamento de Computação



Métodos de Busca

Inteligência Artificial

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

Solução de problemas como Busca

- Um problema pode ser considerado como um objetivo
- Um conjunto de ações podem ser praticadas para alcançar esse objetivo
- Ao buscar um objetivo, estamos em um determinado *estado*
  - O *estado inicial* é quando iniciamos a busca
  - O estado que satisfaz a meta é o *estado objetivo*
- Busca
  - Método que examina o espaço de um problema, buscando um objetivo
- O espaço de um problema é seu *Estado de Busca*

Busca guiada por Dados ou Objetivos

- Abordagens para fazer uma árvore de 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 guiada por Dados ou Objetivos

- Ambas atingem o mesmo resultado;
- Um dos métodos pode ser mais rápido que o outro, porém:
  - Depende da natureza do problema

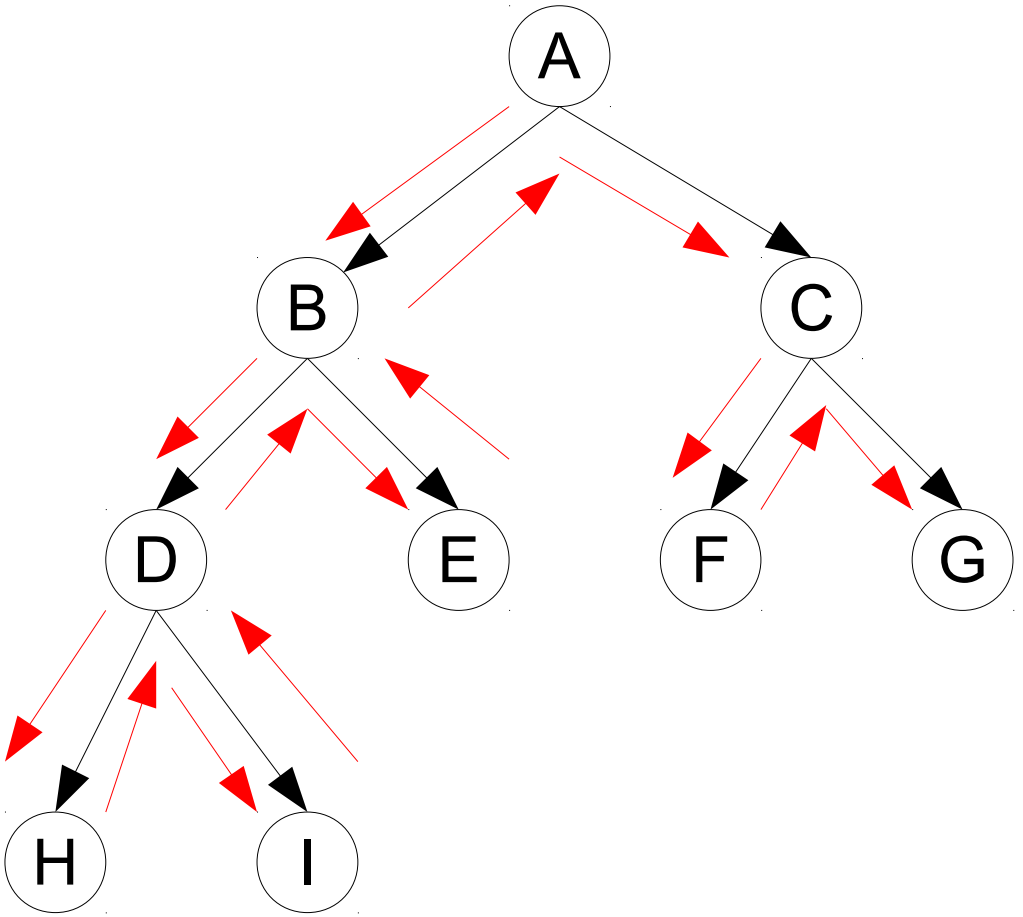
Metodologias

- **Gerar e Testar** – técnica de *busca cega*
  - 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 em 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

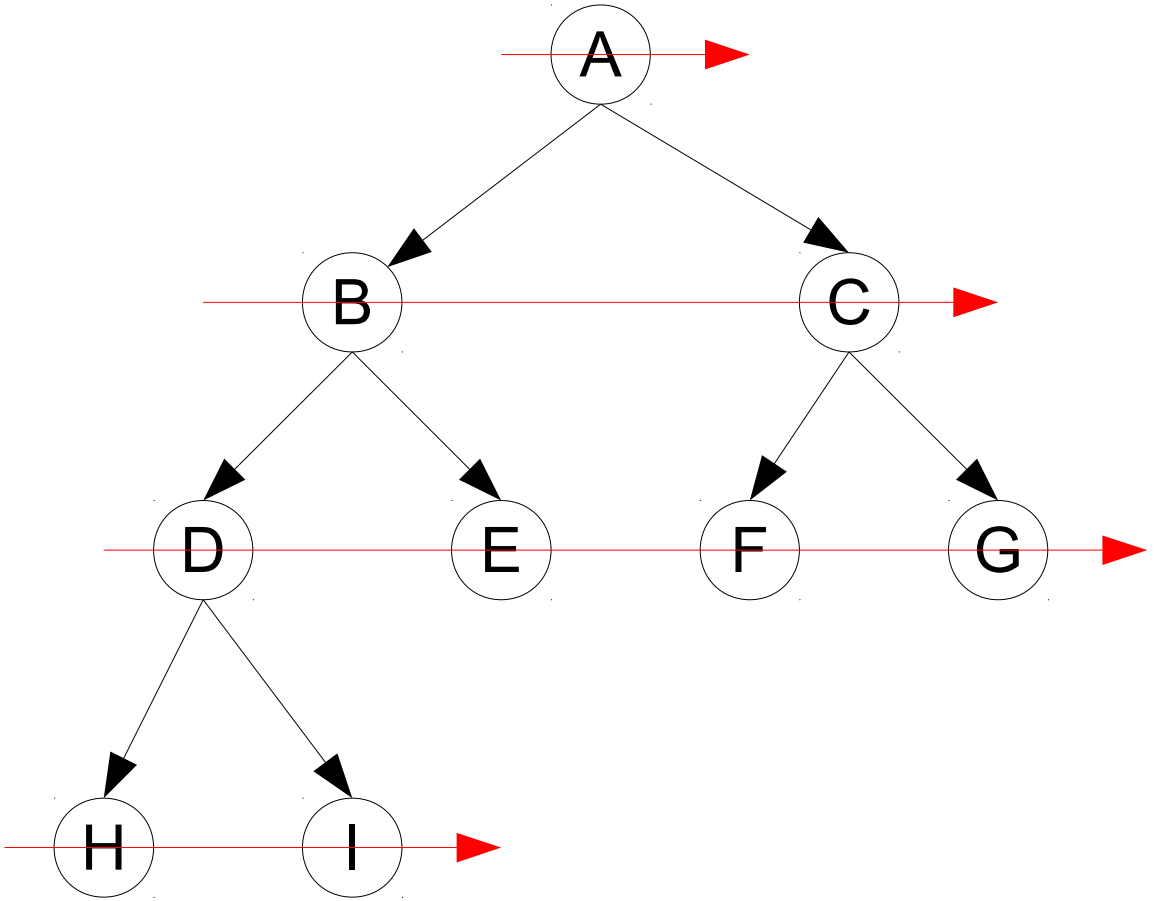


Ordem: A, B, D, H, I, E, C, F, G

Busca em 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

Comparação

Cenário	Profundidade	Largura
Caminhos muito longos ou infinitos	Funciona mal	Funciona bem
Caminhos com comprimentos parecidos	Funciona bem	Funciona bem
Todos caminhos tem comprimentos parecidos e todos levam a um estado objetivo	Funciona bem	Desperdício de tempo e memória
Alto fator de ramificação	O desempenho depende de outros fatores	Funciona precariamente

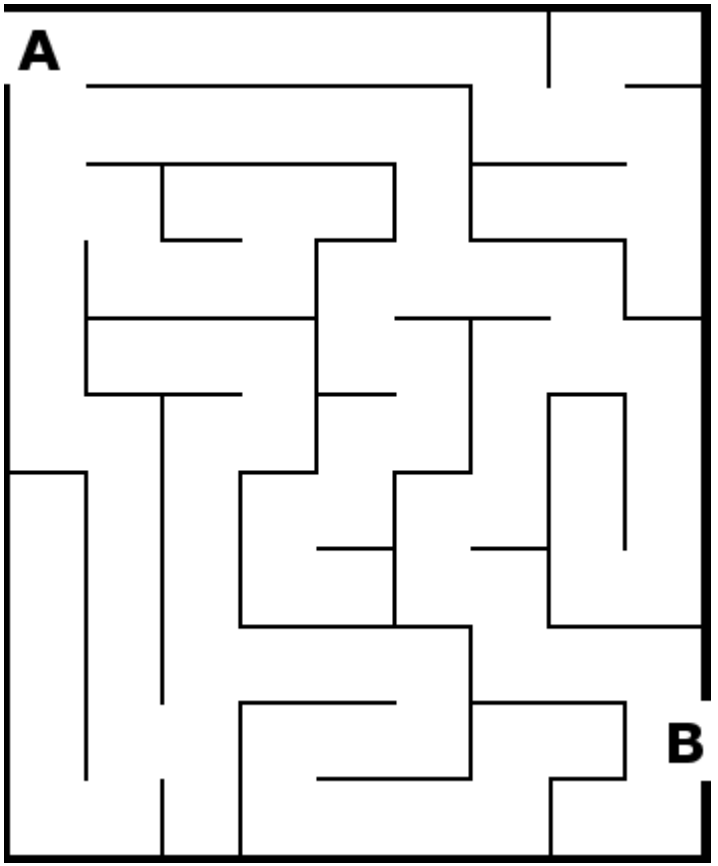
O problema de caminhos infinitos pode ser evitado na busca em profundidade pela aplicação de um **limiar de profundidade**

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 a melhor solução (objetivo) que exista;
  - Não garante que seja pelo menor caminho ou tempo.
- Admissibilidade:
  - Garantir achar a melhor solução pelo melhor caminho.
- Irrevogabilidade:
  - Não retrocedem, examinando assim somente um caminho.

Humanos utilizam busca em profundidade

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



  - Comprando um presente em um shopping;

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*

Derivações dos algoritmos de busca

Busca em profundidade com Aprofundamento Iterativo (BPAI)

- Também chamado de:
  - Busca com Aprofundamento Iterativo (BAI),
  - *Depth-First Iterative Deepening* (DFID),
  - *Iterative Deepening Search* (IDS).
- Técnica exaustiva;
- Combina buscas em profundidade e em largura;
- Conduz buscas com profundidade limitada:
  1. com profundidade de um;
  2. depois com profundidade de dois;
  3. depois com profundidade de três; e
  4. assim por diante, até encontrar um nó objetivo.

Busca em profundidade com Aprofundamento Iterativo (BPAI)

Árvore com fator de ramificação ***b*** e profundidade ***d***

Profundidade

- Total de nós:
  - $1 + b + b^2 + \ldots + b^d$

    Prog. Geom.:

    $(1 - b^{d+1}) / (1 - b)$

  - Ex: com $d = 2$ e $b = 2$:

    $(1 - 8)/(1 - 2) = 7$ nós

BPAI

- Como os nós devem ser examinados mais de uma vez, temos:

  $(d+1)+b(d)+b^2(d-1)+b^3(d-2)+\ldots+b^d$

- Complexidade de $O(b^d)$
  - Ex: com $d = 2$ e $b = 2$:

    $(2+1) + 2 \times 2 + 4 = 11$ nós

Busca em profundidade com Aprofundamento Iterativo (BPAI)

- Ruim para árvores pequenas e com bons resultados em árvores grandes

- Ex: profundidade 4 e fator de ramificação 10

  Profundidade:

  $(1 – 105) / (1 – 10) = 11.111$ nós

  BPAI:

  $(4+1)+10x4+100x3+1.000x2+10.000 = 12.345$ nós

Heurísticas de Busca

- *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?
- 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

Heurísticas de Busca

- Métodos Informados:
  - Utilizam informação adicional sobre os nós não explorados para decidir qual escolher.
  - Que utilizam Heurísticas.
- Métodos Não Informados *ou* Cegos:
  - Não utilizam informação adicional sobre os nós não explorados.
  - Que não utilizam Heurísticas.
- Quanto melhor a heurística for, menos nós ela precisará examinar na árvore.

Monotonicidade

- Método de busca é monótono
  - se ele sempre chega a um dado nó pelo caminho mais curto possível
- Uma heurística *monotônica* é uma heurística com essa propriedade
- *Heurística admissível*
  - Heurística que nunca superestime a distância verdadeira entre um nó e o objetivo

Métodos de busca informados

Subida na colina

- Caso de estudo
  - Se tentar escalar uma montanha em dia de neblina,
    com um altímetro, mas sem mapa,
    você utilizaria uma abordagem de subida na colina
  - Abordagem *Gerar e Testar*;
  - Como proceder:
    - Verificar a altura a alguns centímetros de sua posição em cada direção: norte, sul, oeste e leste.
    - Assim que encontrar uma posição que o leve para uma altura maior que a atual, vá para lá e repita esses passos.
    - Se todas as posições o levam para mais baixo de onde está, assuma que chegou ao topo.

Você sempre chegará ao topo?

Você sempre chegará ao topo?

Subida na Colina pela Encosta de Maior Aclive

Funciona da mesma forma que a Subida na Colina, porém

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

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.

Implementação da Subida na Colina

```
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 )
```

Busca pelo Primeiro Melhor

- Parecido com à Subida na Colina, porém
  - 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

- 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.

Implementação do Primeiro Melhor

```
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

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

Identificando os Melhores Caminhos

Identificando o Caminho Ótimo

- Caminho Ótimo
  - Aquele que tem o menor custo ou a menor distância entre o *nó inicial* e o *nó objetivo*
- Existem diversos métodos que identificam o *caminho ótimo* em uma árvore de busca
- Método mais simples
  - *Procedimento do Museu Britânico*:

    Envolve:
    - examinar cada caminho na árvore de busca,
    - retornar pelo melhor caminho que foi encontrado.

Identificando o Caminho Ótimo

- Algumas técnicas mais sofisticadas
  - A*
  - Busca de custo uniforme (Ramificar e Limitar)
  - Busca Gulosa

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.

Busca de Custo Uniforme

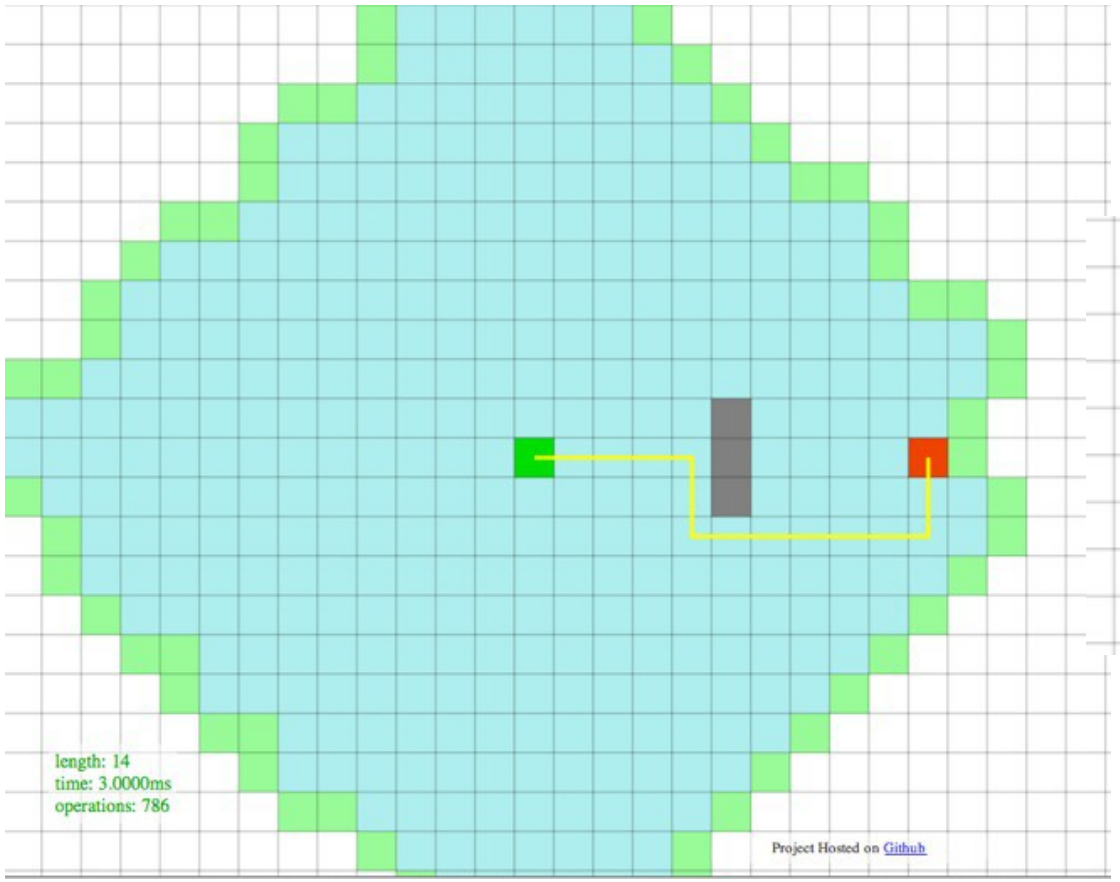
- Algoritmo de Dijkstra.
- Variação da busca pelo primeiro melhor.
  - Usa a função g(n), assim como A*,
  - porém, zera o valor de h(n).

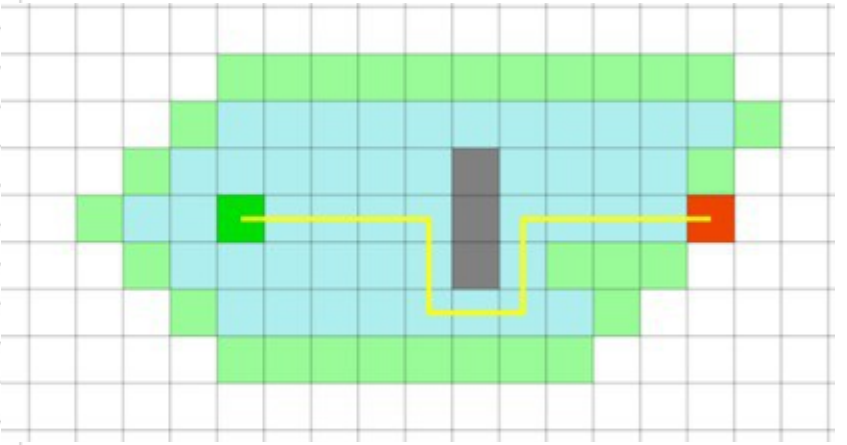
- Se para cada nó *m* que tenha um sucessor *n* for verdade que g(m) < g(n), então, a busca é ótima.

Busca Gulosa

- Variação do A*,
  - onde g(nó) é zerada e
  - Somente h(nó) é utilizada.
- Sempre seleciona o caminho que tenha
  - o menor valor heurístico, ou
  - a menor distância (custo) estimada.

Visualização dos Métodos de Busca





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