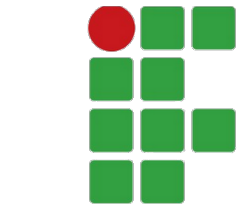
Inteligência Artificial

Métodos de Buscas

Felipe Augusto Lima Reis

felipe.reis@ifmg.edu.br



INSTITUTO FEDERAL
Minas Gerais

Sumário

Introdução

Definição

- Busca é um método utilizado para examinar um espaço de soluções de um problema, em busca de atingir um determinado objetivo [Coppin, 2004];
- Métodos de Busca recebem problema como entrada e retornam uma ou mais soluções, sob a forma de uma sequência de ações [da Silva, 2014].

Busca Informada e Não Informada

- Métodos de busca podem ser subdivididos em relação ao uso de informações sobre nós a serem explorados:
  - Busca Informada: usa informações adicionais sobre nós ainda não explorados para decidir quais nós examinar a seguir
    - Frequentemente utilizam heurísticas para examinar o espaço de busca de forma mais eficiente;
  - Busca Não Informada: métodos que não usam informações sobre nós já / a serem explorados (também conhecida como busca cega) [Coppin, 2004].

Busca baseada em Dados ou Objetivos[1]

- Métodos de busca podem também ser subdivididos em:
  - Busca baseada em dados: começa em um estado inicial e usa ações para seguir em frente até atingir um objetivo;
    - São utilizados em situações onde o objetivo não é conhecido;
  - Busca baseada em objetivos: permite analisar o objetivo e voltar ao estado inicial
    - O método deve possibilitar o algoritmo retorne do objetivo ao estado inicial pelo espaço de buscas;
    - Podem ter melhor desempenho em situações onde o objetivo é previamente conhecido [Coppin, 2004].

[1] Tradução de *Data-driven search* e *Goal-driven search*.

Estratégias de Busca

- Métodos que serão estudados nesta seção:
  - Busca Não Informada:
    - Busca em Largura, em Profundidade, com Profundidade Limitada, com Aprofundamento Iterativo, Busca Bidirecional;
  - Busca Informada:
    - *Best-First Search*, Busca Gulosa (*Greedy Search*), Algoritmo A*, Metaheurísticas[2];

[2] Serão estudados separadamente. Inclui Hill-Climbing, Simulated Annealing, Algoritmos Genéticos, ACO, etc.

Critérios de Avaliação de Algoritmos de Busca

Critérios de Avaliação

- Monotonicidade[3]:
  - Um método de busca é dito monótono se é sempre capaz de encontrar um nó do espaço de busca usando o menor caminho possível [Coppin, 2004].
- Otimalidade:
  - Um método é dito ótimo se garante a seleção da melhor solução, caso exista;
  - O método irá atingir o objetivo com o menor número de etapas/passos;
  - Otimalidade não garante eficiência
    - No entanto, uma vez que o método encontrou uma solução ótima, não existe outra solução melhor [Coppin, 2004];

[3] Denominado também como "Consistente" por [Russel and Norvig, 2013]

Critérios de Avaliação

INSTITUTO FEDERAL
Minas Gerais

- Admissibilidade:
  - Conceito utilizado no lugar de Otimalidade, com o mesmo sentido: garante a seleção da melhor solução;
  - Utilizado devido ao mal uso do conceito de ótimo, que inadequadamente, é usado como sinônimo de algoritmos que executam no menor tempo possível [Coppin, 2004];
- Irrevocabilidade:
  - Métodos que não realizam *backtracking*, ou seja, somente avaliam um caminho e não voltam para examinar soluções diferentes e melhores [Coppin, 2004];

Critérios de Avaliação

- Complexidade:
  - Complexidade de tempo: número de operações a serem executadas (correlacionado ao tempo de execução);
  - Complexidade de espaço: quantidade de memória necessária para execução do algoritmo [Coppin, 2004];
- Completude[4]:
  - Um método é dito completo se ele garante a seleção do estado correspondente ao objetivo, caso exista;
  - Não há garantia de encontrar a melhor solução, porém apenas uma solução possível, caso exista [Coppin, 2004];

[4] Denominado "Completeza" por [Russel and Norvig, 2013].

Busca Básica (Busca Não Informada)

Os métodos desta seção foram separados somente devido a sua ampla utilização.

Gerar e Testar

Gerar e Testar

- Corresponde apenas a gerar um nó no espaço de busca e testar sua viabilidade como objetivo [Coppin, 2004];
  - Considerado o tipo mais simples de busca;
  - Classificado como método de força bruta (busca exaustiva);
  - Classificado como busca não informada (busca cega).

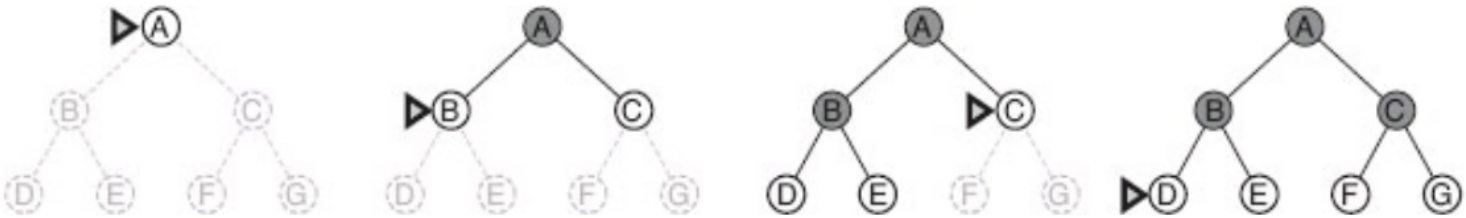
Gerar e Testar

- Contém um gerador de nós, que deve respeitar as seguintes propriedades: [Coppin, 2004]
  - Completude: deve gerar todas as possibilidades possíveis (garante seleção do objetivo);
  - Não redundante: deve gerar cada solução somente 1 vez;
  - Bem informado: somente deve propor soluções válidas no espaço de buscas.

Busca em Largura

Busca em Largura

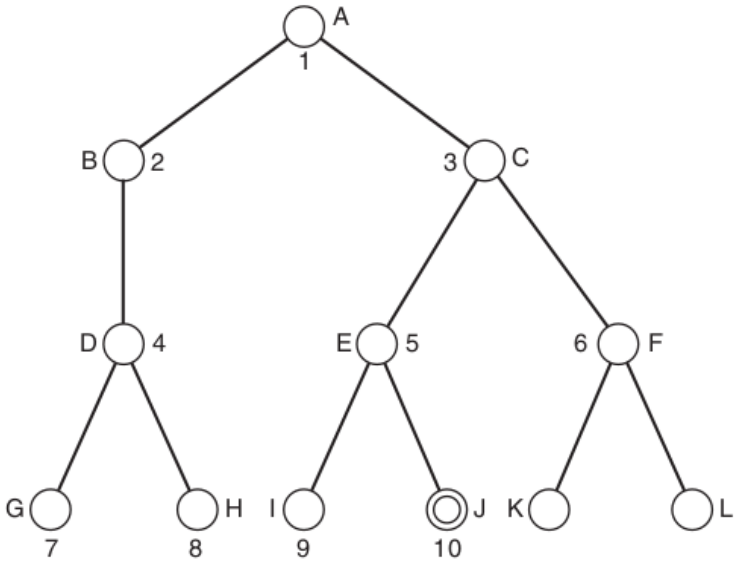
- A Busca em Largura (*Breadth-first search*) realiza busca, a partir de um determinado nó, em todos os sucessores diretos (nós filhos) [Coppin, 2004] [Russel and Norvig, 2013];
  - Todos os nós de um determinado nível d são percorridos, antes que os nós dos níveis seguintes, $d + 1$, sejam expandidos [Russel and Norvig, 2013] [da Silva, 2014];



Fonte: [Russel and Norvig, 2013]

Busca em Largura

- A busca em largura é um algoritmo em que o nó mais raso (próximo ao topo) não expandido é escolhido para expansão
  - Utiliza-se uma fila (FIFO), no qual novos nós, mais profundos, vão para o fim da fila, e nós antigos (mais rasos) são primeiramente expandidos [Russel and Norvig, 2013].



Fonte: [Coppin, 2004]

Busca em Largura

- Características:
  - Completa: um nó de profundidade finita d será encontrado após a expansão dos nós mais rasos[5];
  - Admissível: garante a seleção do melhor valor (apesar de não necessariamente alcançar o objetivo com menor número de passos) [Russel and Norvig, 2013].
- Complexidade:
  - A busca em largura possui baixo desempenho tanto em relação ao tempo quanto espaço [Russel and Norvig, 2013].

[5] Somente se o número de ramificações b for finito.

Busca em Largura

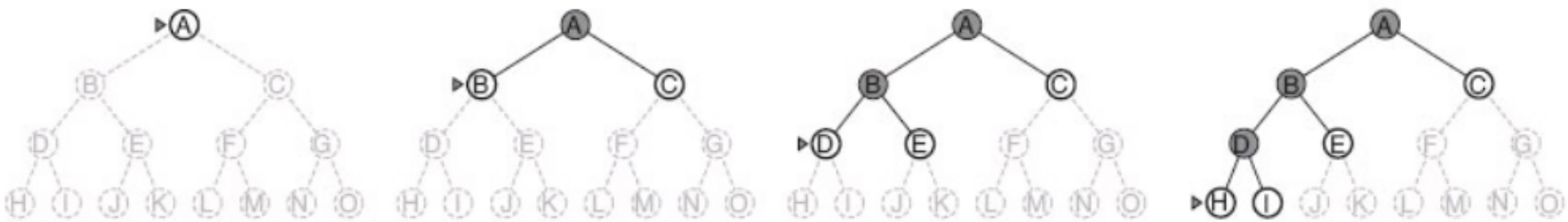
- Complexidade:
  - Supondo uma árvore binária, a primeira expansão tem 1 nó, a segunda 2 nós, a terceira, 4 nós, e assim por diante;
    - Considerando árvores de b ramificações, a primeira expansão tem 1 nó, a segunda b^1 nós, a terceira b^2 nós, ...;
  - O número de nós avaliados em um nível d pode ser dado, então, por uma progressão geométrica $a_d = 1 \times b^{d-1}$;
  - A quantidade total avaliados de nós pode ser dado por uma soma de PG, $S_d = (1 \times (b^d - 1))/(b - 1)$;
  - A complexidade de tempo e espaço são dadas por $\mathcal{O}(b^d)$.

Utiliza-se, comumente, a letra d (*depth* - profundidade), em substituição de n para complexidade de busca.

Busca em Profundidade

Busca em Profundidade

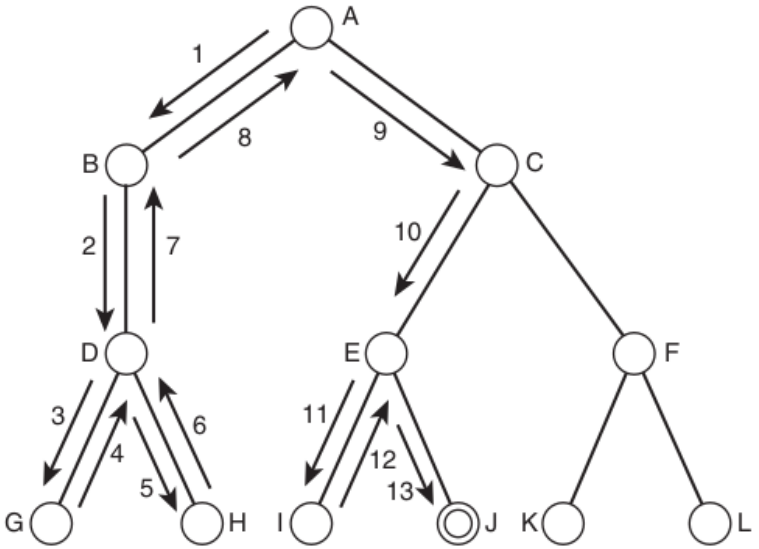
- A Busca em Profundidade (*Depth-first search*) realiza busca, a partir de um determinado nó, até a profundidade máxima da árvore, para depois se mover para um próximo caminho [Coppin, 2004] [Russel and Norvig, 2013];
  - A busca vai até o nível mais profundo da árvore de busca, onde os nós não têm outros sucessores [Russel and Norvig, 2013];



Fonte: Adaptado de [Russel and Norvig, 2013]

Busca em Profundidade

- A busca em profundidade é um algoritmo de busca em grafo em que a expansão ocorre sequencialmente até o nível mais profundo, percorrendo sucessores ainda não explorados.
  - Utiliza-se uma pilha (LIFO), no qual nós mais recentes são expandidos primeiro [Russel and Norvig, 2013].



Fonte: [Coppin, 2004]

Busca em Profundidade

- A busca em profundidade pode ser facilmente implementada usando recursividade [Russel and Norvig, 2013];
- Pode funcionar mais rápido que busca em largura em problemas que possuem várias soluções [da Silva, 2014] [Coppin, 2004];
- Possui como ponto fraco a possibilidade de aprofundar-se em caminhos ruins
  - Em árvores de busca profundas ou infinitas pode nunca retornar uma solução;
  - Pode retornar caminhos sub-ótimos [da Silva, 2014].

Busca em Profundidade

- Utiliza um método chamado de *backtracking* cronológico, para movimentar-se de volta pela árvore, quando um nó folha é encontrado [Coppin, 2004];
  - O nome cronológico é acrescentado ao *backtracking* devido à ordem reversa de volta pelos nós da árvore;
- Pode ser considerada uma busca exaustiva [Coppin, 2004];
- Segundo [Coppin, 2004], a busca em profundidade pode ser usada para:
  - Encontrar arquivos em disco;
  - Percorrer árvores de diretórios;
  - Rastrear a internet (máquinas de busca / *web crawlers*).

Busca em Profundidade

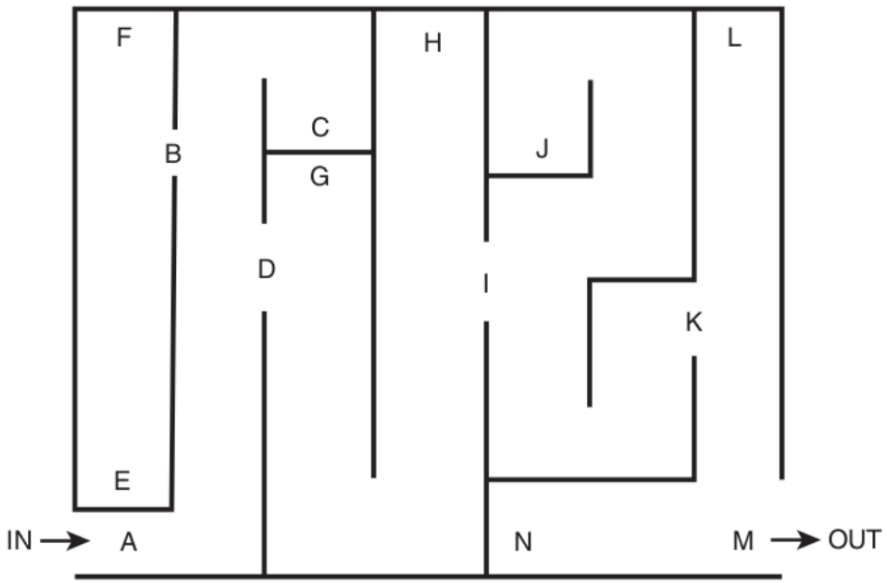
- Características:
  - Não Completa: em grafos com laços, a busca pode não ser completa; somente completa em grafos com estados finitos e sem redundância [Russel and Norvig, 2013];
  - Não-ótima: pode continuar a busca mesmo que encontre um valor ótimo, o que fere a característica de otimalidade;

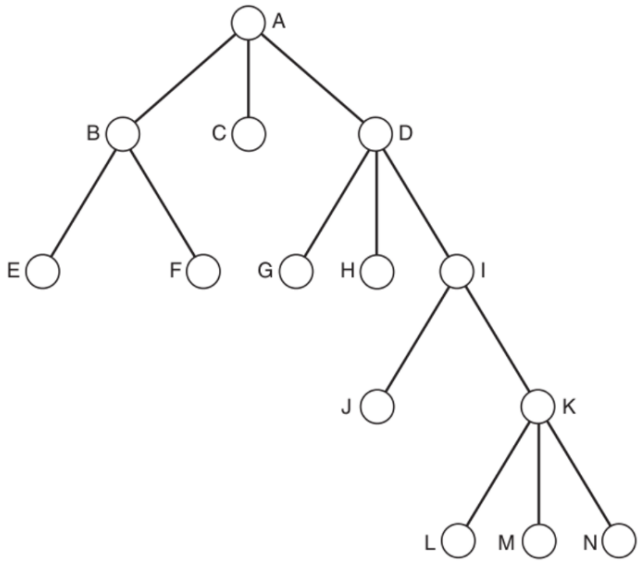
- Complexidade:
  - Possui menor uso de memória, uma vez que necessita apenas de armazenar o caminho da raiz até a folha [da Silva, 2014];
  - Tempo: $\mathcal{O}(b^m)$;
  - Armazenamento: $\mathcal{O}(bm)$.

Considere b o número de ramificações na árvore (*branching factor*) e m a profundidade máxima da árvore.

Busca em Profundidade - Exemplo [Coppin, 2004]

- Considere o algoritmo para solução de um labirinto, usando busca em profundidade
  - Para isso, suponha que a solução seja dada por um indivíduo percorrendo o labirinto e tocando parede esquerda com a mão.





Fonte: [Coppin, 2004]

Busca em Largura x Busca em Profundidade

Comparação - Largura x Profundidade

- As busca em Largura e em Profundidade podem ser resumidas na tabela abaixo:

Critério	**Profundidade**	**Largura**
Caminhos longos ou infinitos	Funcional Mal	Funciona Bem
Caminhos com comprimentos parecidos	Funciona Bem	Funciona Bem
Caminhos com comprimentos parecidos; todos levam a um estado objetivo	Funciona Bem	Desperdício Tempo/Memória
Branching factor alto	Depende de outros fatores	Precário

Fonte: [Coppin, 2004] e [da Silva, 2014]

Busca Não Informada

Busca com Profundidade Limitada

Busca com Profundidade Limitada

- A Busca com Profundidade Limitada tem como objetivo melhorar ao algoritmo de busca em profundidade, ao impor um limite L de profundidade de busca
  - Essa restrição almeja resolver o problema de caminhos muito longos ou infinitos [Russel and Norvig, 2013];
- O limite de profundidade pode ser baseado em conhecimento prévio acerca do problema
  - Se, ao saber previamente que o comprimento da maior solução tem no máximo 19 itens, pode-se definir $L = 19$;
  - Esse limite de profundidade é chamada de diâmetro do espaço de estados [Russel and Norvig, 2013].

Busca com Profundidade Limitada

- Características:
  - Não Completa: além de manter a não completude da busca em profundidade tradicional, a escolha de um $L < d$ restringe ainda mais a solução [Russel and Norvig, 2013];
  - Não-ótima: o algoritmo pode continuar a busca mesmo que encontre um valor ótimo;
- Complexidade:
  - Tempo: $\mathcal{O}(b^L)$;
  - Armazenamento: $\mathcal{O}(bL)$.

Busca com Aprofundamento Iterativo

Busca com Aprofundamento Iterativo

- A Busca de Aprofundamento Iterativo (IDS)[6] é uma estratégia que combina os benefícios das buscas em Largura e em Profundidade [da Silva, 2014] [Russel and Norvig, 2013]
  - Possibilita economia de memória (busca em profundidade);
  - Capaz de gerar caminhos que envolvam a menor quantidade de passos para encontrar uma solução (busca em largura);
  - A ordem de expansão dos estados é similar à busca em largura;

[6] *Iterative Deepening Search (IDS)* ou *Depth-First Iterative Deepening (DFID)* [Coppin, 2004].

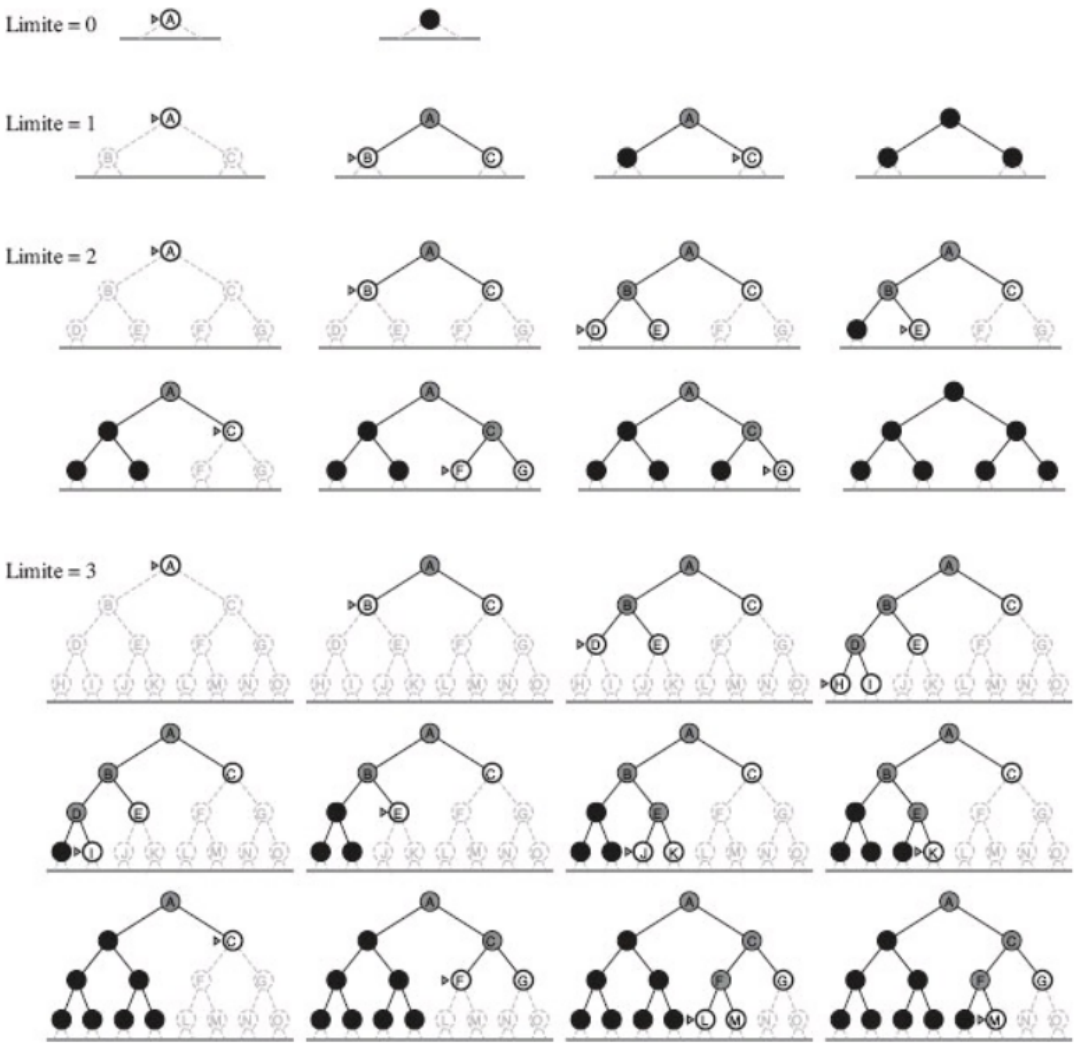
Busca com Aprofundamento Iterativo

- O método aumenta gradualmente o limite de busca, até encontrar um nó-objetivo [Russel and Norvig, 2013].
- Por muitas vezes é usada em conjunto com a busca em profundidade
  - A busca em profundidade é usada para encontrar o melhor limite de profundidade;
- O IDS é a técnica não informada com melhor desempenho quando o espaço de busca é grande e a profundidade da solução não é conhecida [Russel and Norvig, 2013];

Busca com Aprofundamento Iterativo

- No IDS, cada nó pode ser examinado mais de uma vez [Coppin, 2004] [Russel and Norvig, 2013]
  - Nós de profundidade d são gerados uma vez, de $d-1$ são gerados duas vezes, e assim adiante, até os filhos da raiz, que são gerados d vezes [Russel and Norvig, 2013].
- Devido aos custos de gerar nós de forma repetida, o método perde desempenho, em relação a tempo e espaço;
- No entanto, possui como pontos positivos a manutenção de qualidades da Busca em Largura [Russel and Norvig, 2013].

Busca com Aprofundamento Iterativo



Fonte: [Russel and Norvig, 2013]

Busca com Aprofundamento Iterativo

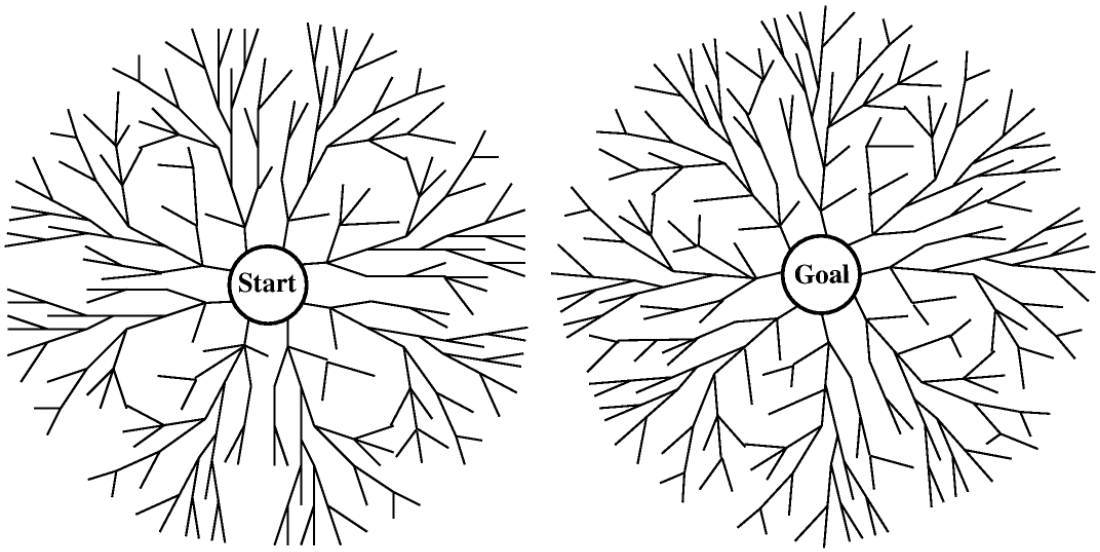
INSTITUTO FEDERAL
Minas Gerais

- Características:
  - Completa: somente se o número de ramificações for finito;
  - Não Ótima: otimalidade ocorre somente se os custos dos passos forem idênticos [Coppin, 2004];
- Complexidade:
  - Tempo: $\mathcal{O}(b^d)$;
  - Armazenamento: $\mathcal{O}(bd)$.

Busca Bidirecional

Busca Bidirecional

- A Busca Bidirecional executa duas buscas simultaneamente
  - Uma busca começa no estado inicial, e a outra, parte, de forma reversa, do objetivo em direção ao início;
  - Espera-se que as duas buscas se encontrem em um ponto intermediário [Russel and Norvig, 2013];



Fonte: [Russel and Norvig, 2013]

Busca Bidirecional

- Vantagens:
  - Reduz consideravelmente área de busca (de b^d para $2b^{\frac{d}{2}}$) [Russel and Norvig, 2013];
- Desvantagens:
  - Exige o conhecimento prévio do nó objetivo [Coppin, 2004];
  - Requer que as ações no espaço de estado sejam reversíveis [Russel and Norvig, 2013];
  - Requer o cálculo de nós predecessores, o que pode não ser trivial em muitos cenários [Russel and Norvig, 2013];

Busca Bidirecional

- Características:
  - Completa: garante a seleção do estado objetivo;
  - Não Ótima: mesmo que seja utilizada a busca em largura, não é garantida a otimalidade[7] [Russel and Norvig, 2013];
- Complexidade:
  - Tempo: $\mathcal{O}(b^{\frac{d}{2}})$;
  - Armazenamento: $\mathcal{O}(b^{\frac{d}{2}})$.

[7] Otimalidade ocorre somente se todos os passos tiverem custos idênticos.

Busca de Custo Uniforme (Branch and Bound)

Busca de Custo Uniforme (*Branch and Bound*)

- Inventada por Dijkstra, em 1959[8][9] [Coppin, 2004];
- A Busca de Custo Uniforme possui as seguintes características:
  - Ao invés expandir todos os nós de uma profundidade d, o algoritmo pode expandir um nó n de custo mais baixo $g(n)$ na profundidade $d+1$ [Aluisio, 2020][10];
  - Para gerenciar expansões, possui uma fila de prioridade, ordenada pelo custo do caminho [Russel and Norvig, 2013];
  - Ao adicionar nós, o custo sempre aumenta em relação ao caminho anterior, ou seja $g(sucessor) \geq g(n)$ [Aluisio, 2020].

[8] Conhecido também como Algoritmo de Dijkstra [Coppin, 2004].

[9] [Russel and Norvig, 2013] afirma que o algoritmo de caminhos mais curtos de Dijkstra é a origem do BCU.

[10] Consequentemente pode expandir outros nós na profundidade $d+k$, antes de expandir nós na profundidade d.

Busca de Custo Uniforme

- [Russel and Norvig, 2013] consideram o algoritmo como uma adaptação da Busca em Largura, enquanto [Coppin, 2004] considera o algoritmo como uma alteração na Busca A*
  - É semelhante à Busca em Largura quando todos os custos são iguais [Hruschka, 2014];
  - É equivalente à Busca A* quando $h(n)$ é definido como zero [Coppin, 2004].

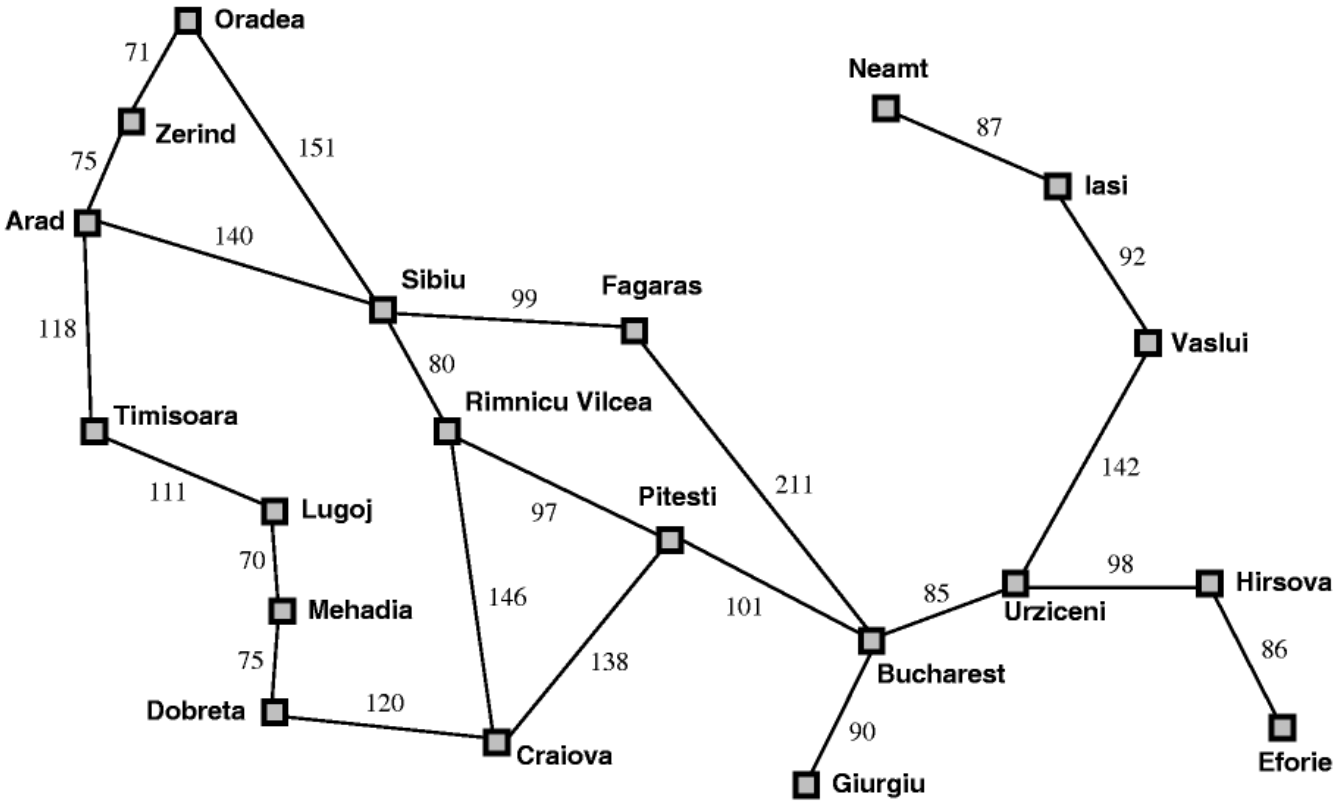
Busca de Custo Uniforme

- O algoritmo da Busca de Custo Uniforme possui, ainda, as seguintes alterações em relação à Busca em Largura:
  - O teste de objetivo é aplicado quando um nó é selecionado para a expansão e não quando ele é gerado pela primeira vez
    - Isso evita que o primeiro nó objetivo gerado esteja abaixo de um caminho ótimo;
  - São adicionados testes, aos nós da solução atual, para análise de possíveis caminhos melhores [Russel and Norvig, 2013][11];

[11] Chamado por [Russel and Norvig, 2013] por nós de borda.

BCU - Exemplo [Russel and Norvig, 2013]

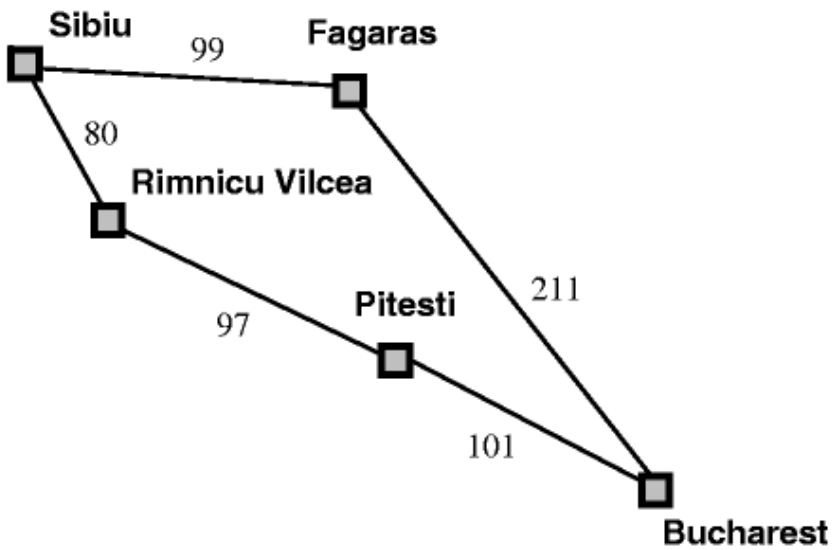
- O grafo abaixo contém as distâncias reais entre algumas cidades.



Fonte: [Johnson, 2008]

BCU - Exemplo [Russel and Norvig, 2013]

- Consideremos que uma pessoa deseja sair de Sibiu e chegar em Bucareste.



Fonte: [Russel and Norvig, 2013]

BCU - Exemplo [Russel and Norvig, 2013]

INSTITUTO FEDERAL Minas Gerais

- Ordem de expansão:
  1. Expandir a cidade Rimnicu Vilcea (80), cuja distância é menor que Fagaras (99);
  2. Fazer Rimnicu Vilcea (80) + Pitesti (97) = 177
     - Como 177 é maior que 99 (Fagaras), expandir Fagaras;
  3. Fazer Fagaras (99) + Bucareste (211) = 310.
     - Apesar de chegar ao objetivo, a busca continua, uma vez que 177 é menor que 310;
  4. Expandir Pitesti, fazendo 80+97+101=278.
     - Como a distância até Bucareste por este caminho é menor que a solução anterior, ele será retornado pelo método.

Busca de Custo Uniforme

- Características:
  - Completa: um nó de profundidade finita d será encontrado, após a expansão dos nós mais rasos;
  - Ótimo: somente se o custo de passo for positivo, a expansão ocorrerá monotonicamente na ordem de menor custo [Russel and Norvig, 2013] [Hruschka, 2014].
- Complexidade:
  - Uma vez que o algoritmo é baseado no custo, a complexidade não pode ser definida em termos do número de ramificações b e profundidade d [Russel and Norvig, 2013].

Comparação - Busca Não Informada

Comparação - Busca Não Informada

- Em relação à complexidade, os métodos podem resumidos pela tabela abaixo:

Método	Tempo	Espaço
Busca em Largura	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$
Busca em Profundidade	$\mathcal{O}(b^m)$	$\mathcal{O}(bm)$
Busca com Prof. Limitada	$\mathcal{O}(b^L)$	$\mathcal{O}(bL)$
Aprofundamento Iterativo	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$
Busca Bidirecional	$\mathcal{O}(b^{\frac{d}{2}})$	$\mathcal{O}(b^{\frac{d}{2}})$
Busca de Custo Uniforme	$\mathcal{O}(b^{1+(C^*/\epsilon)})$	$\mathcal{O}(b^{1+(C^*/\epsilon)})$

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Legenda:

- b: Número de ramificações (*branching factor*);
- d: Nível de profundidade da árvore;
- m: Profundidade máxima da árvore;
- L: Limite de pesquisa na árvore (parâmetro).
- C^*: custo da solução ótima;
- ϵ: custo de cada uma das ações, ao percorrer o grafo.

Comparação - Busca Não Informada

- Em relação à completude/completeza temos:

Método	Completo
Busca em Largura	Somente se número de ramificações for finito.
Busca em Profundidade	Não.
Busca com Prof. Limitada	Não.
Aprofundamento Iterativo	Somente se número de ramificações for finito.
Busca Bidirecional	Somente se número de ramificações for finito e ambos os sentidos usarem busca em largura.
Busca de Custo Uniforme	Sim

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Comparação - Busca Não Informada

INSTITUTO FEDERAL
Minas Gerais

- Em relação à otimalidade temos:

Método	Ótimo
Busca em Largura	Somente se os custos dos passos forem idênticos.
Busca em Profundidade	Não.
Busca com Prof. Limitada	Não.
Aprofundamento Iterativo	Somente se os custos dos passos forem idênticos.
Busca Bidirecional	Somente se os custos dos passos forem idênticos e ambos os sentidos usarem busca em largura.
Busca de Custo Uniforme	Somente se custo do passo for positivo.

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Busca Informada

Best-First Search (Busca Pela Melhor Escolha)

Best-First Search (BFS)

- A Busca Pela Melhor Escolha ou *Best-First Search (BFS)*[12]. é um método de busca que expande primeiro os nós com melhor valor objetivo [Russel and Norvig, 2013] [da Silva, 2014];
  - Essa expansão gulosa, busca gerar rapidamente uma solução;
  - Para isso, considera que a função de busca $f(n)$ é igual à heurística $h(n)$ [Russel and Norvig, 2013];

$$\boxed{f(n) = h(n)}$$

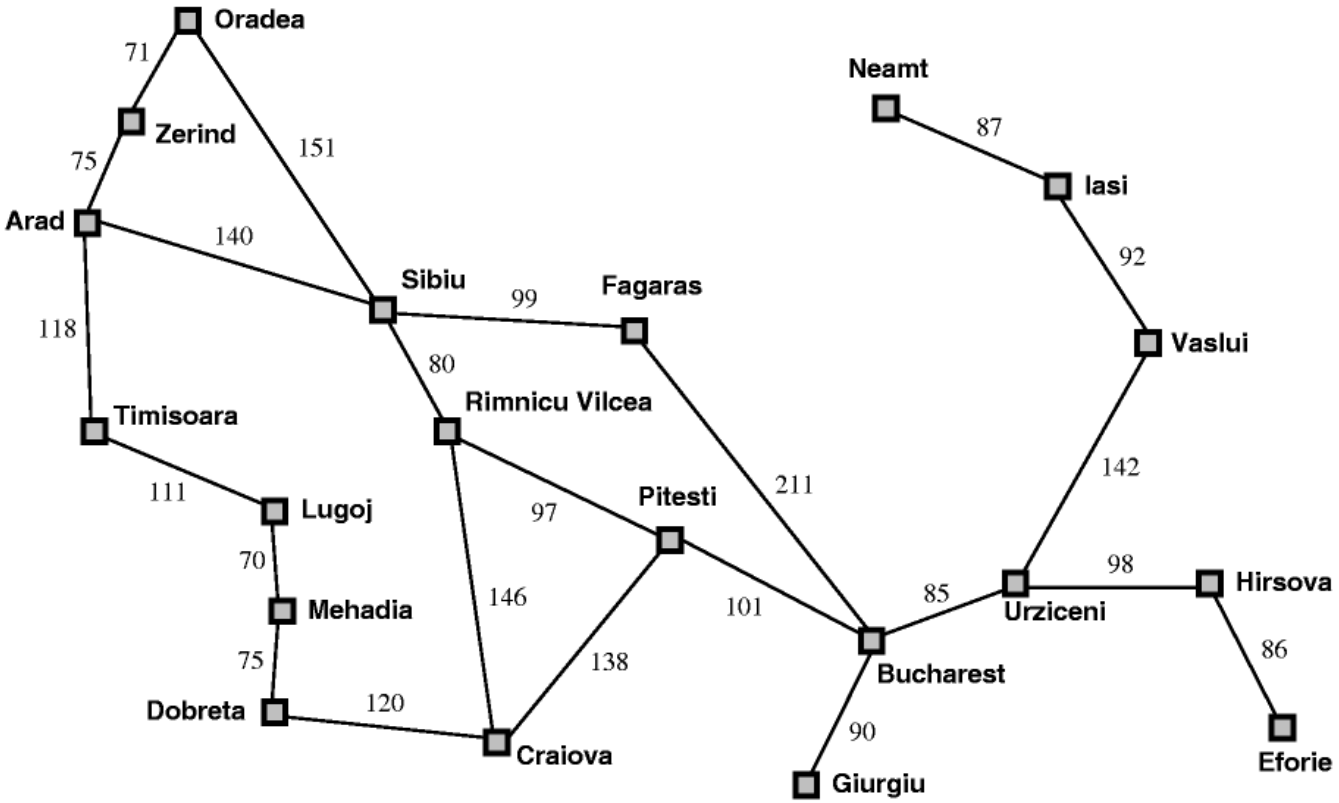
[12] Denominada Busca Gulosa de Melhor Escolha ou Busca Gulosa em [Russel and Norvig, 2013]

Best-First Search (BFS)

- A Busca Pela Melhor Escolha procura o melhor caminho possível a partir da posição atual na árvore [Coppin, 2004]
  - No passo seguinte, busca novamente a melhor opção (nó mais próximo ao objetivo).
- A função heurística $h(n)$ corresponde ao caminho mais barato até o objetivo [da Silva, 2014]
  - Se $h(n) = 0$, o método atingiu o objetivo.

BFS - Exemplo [Russel and Norvig, 2013]

- O grafo abaixo contém as distâncias reais entre algumas cidades.



Fonte: [Johnson, 2008]

BFS - Exemplo [Russel and Norvig, 2013]

INSTITUTO FEDERAL Minas Gerais

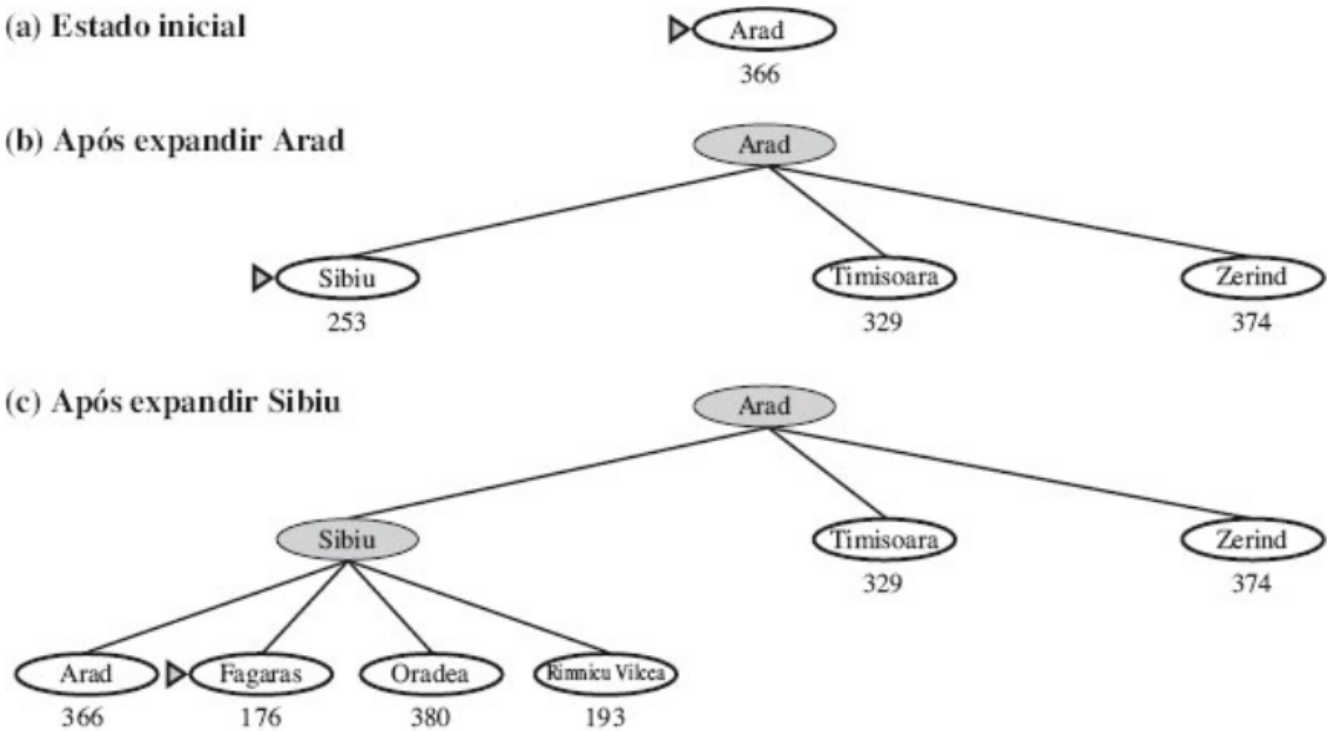
- A tabela abaixo contém a distância Euclidiana de algumas cidades até Bucareste
  - Essas distâncias podem ser usadas como heurística $h(n)$;
- Consideremos que uma pessoa deseja sair de Arad e chegar em Bucareste.

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

Fonte: [Russel and Norvig, 2013]

BFS - Exemplo [Russel and Norvig, 2013]

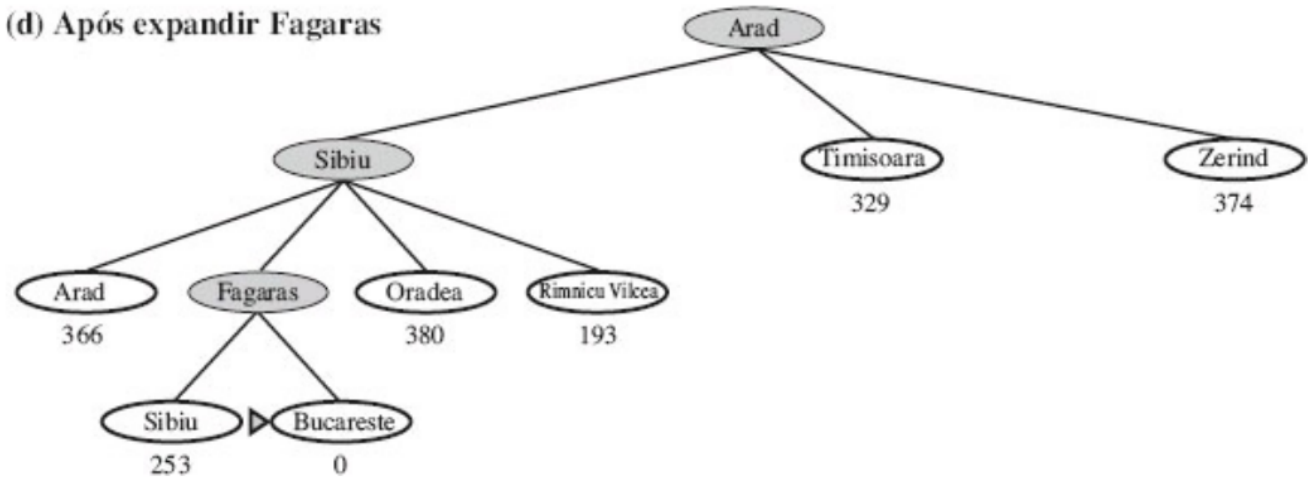
- Ordem de execução do BFS.



Fonte: [Russel and Norvig, 2013]

BFS - Exemplo [Russel and Norvig, 2013]

- Ordem de execução do BFS.



Fonte: [Russel and Norvig, 2013]

Best-First Search (BFS)

- Características:
  - Não Completa: pode aprofundar em um caminho infinito, sem atingir o objetivo;
  - Não Ótima: não há garantia de que o caminho aprofundado seja o menor [Russel and Norvig, 2013] [da Silva, 2014];
- Complexidade:
  - Tempo: $\mathcal{O}(b^m)$;
  - Armazenamento: $\mathcal{O}(b^m)$.

Variável m corresponde à profundidade máxima da árvore no espaço de buscas.

Busca A*

O método é pronunciado como "Busca A estrela" [Russel and Norvig, 2013].

Busca A*

- A Busca A* é semelhante ao BFS, porém mais complexa na escolha do caminho[13] [Coppin, 2004];
  - O BFS não considera o custo do caminho atual até o objetivo;
  - A busca A*, no entanto, avalia o custo $g(n)$, para alcançar o nó, e $h(n)$, para ir do nó ao objetivo [Russel and Norvig, 2013];
  - A função de custo $f(n)$ é dada, então, por:

$$\boxed{f(n) = g(n) + h(n)}$$

[13] [Russel and Norvig, 2013] considera o método como uma variação do BFS.

Busca A*

- A função $f(n)$ é chamada de função de avaliação baseada no caminho [Coppin, 2004];
- Se o valor da heurística $h(n)$ for sempre subestimada, o algoritmo A* é ótimo [Coppin, 2004][14];
  - Nesse caso, a heurística é chamada de admissível;
  - Heurísticas admissíveis são otimistas, uma vez que imaginam que o custo para solução do problema é sempre inferior à realidade [Russel and Norvig, 2013];

[14] A heurística também será ótima caso respeite a propriedade de monotonicidade [Russel and Norvig, 2013].

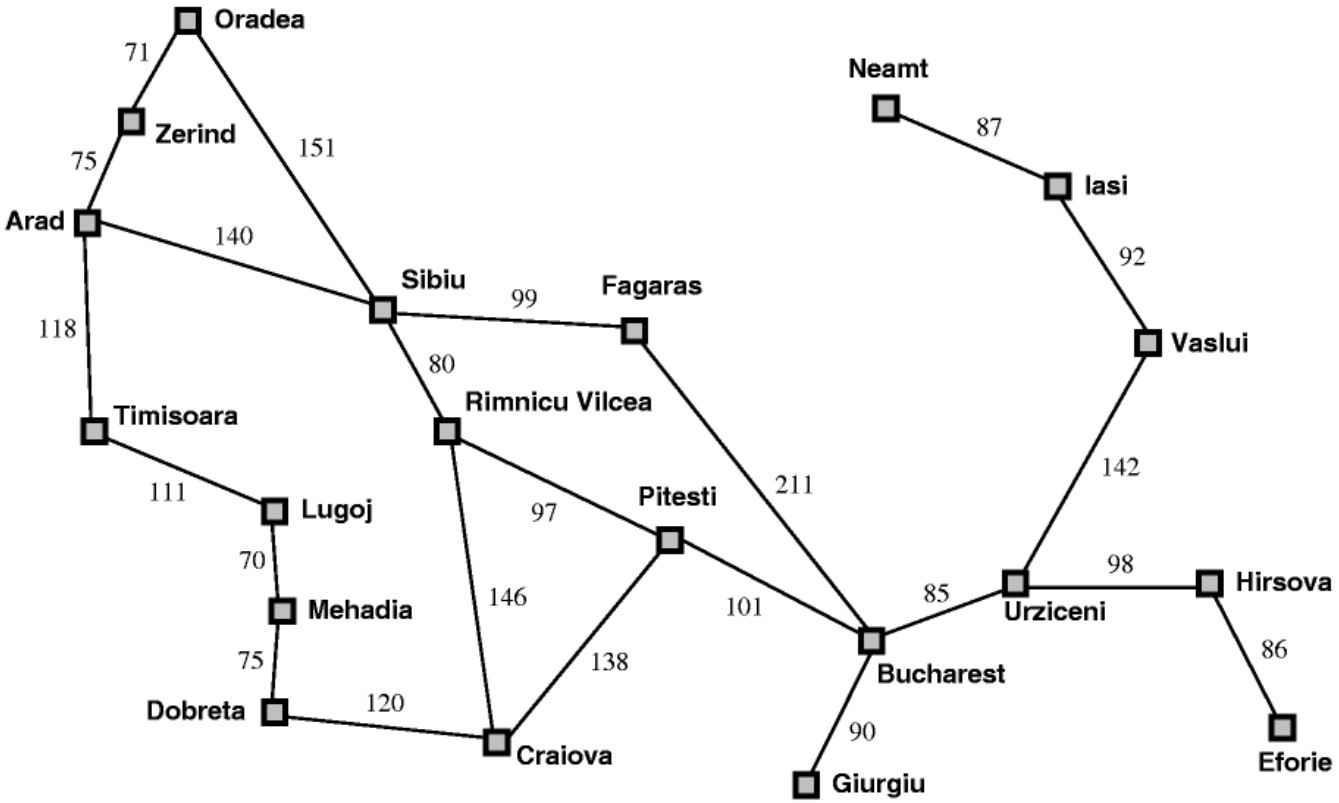
Busca A*

- A busca A* é otimamente eficiente para qualquer dado heurístico consistente[15] [Russel and Norvig, 2013]
  - Não é garantido que nenhum outro algoritmo ótimo expanda menos nós do que A* (exceto em casos de desempate);
  - Algoritmos que estendem caminhos de busca a partir da raiz e utilizam a mesma informação heurística também são otimamente eficientes [Russel and Norvig, 2013];
- Em alguns casos, a complexidade A* torna impraticável a busca por uma solução ótima
  - Variantes do algoritmo buscam por soluções subótimas, o que é mais veloz [Russel and Norvig, 2013].

[15] "Uma heurística $h(n)$ será consistente se, para cada nó n e para todo sucessor n', o custo estimado de alcançar o objetivo de n não for maior do que o custo de chegar a n' somado ao custo estimado de alcançar o objetivo de n'" [Russel and Norvig, 2013].

Busca A* - Exemplo [Russel and Norvig, 2013]

- O grafo abaixo contém as distâncias reais entre algumas cidades.



Fonte: [Johnson, 2008]

Busca A* - Exemplo [Russel and Norvig, 2013]

INSTITUTO FEDERAL Minas Gerais

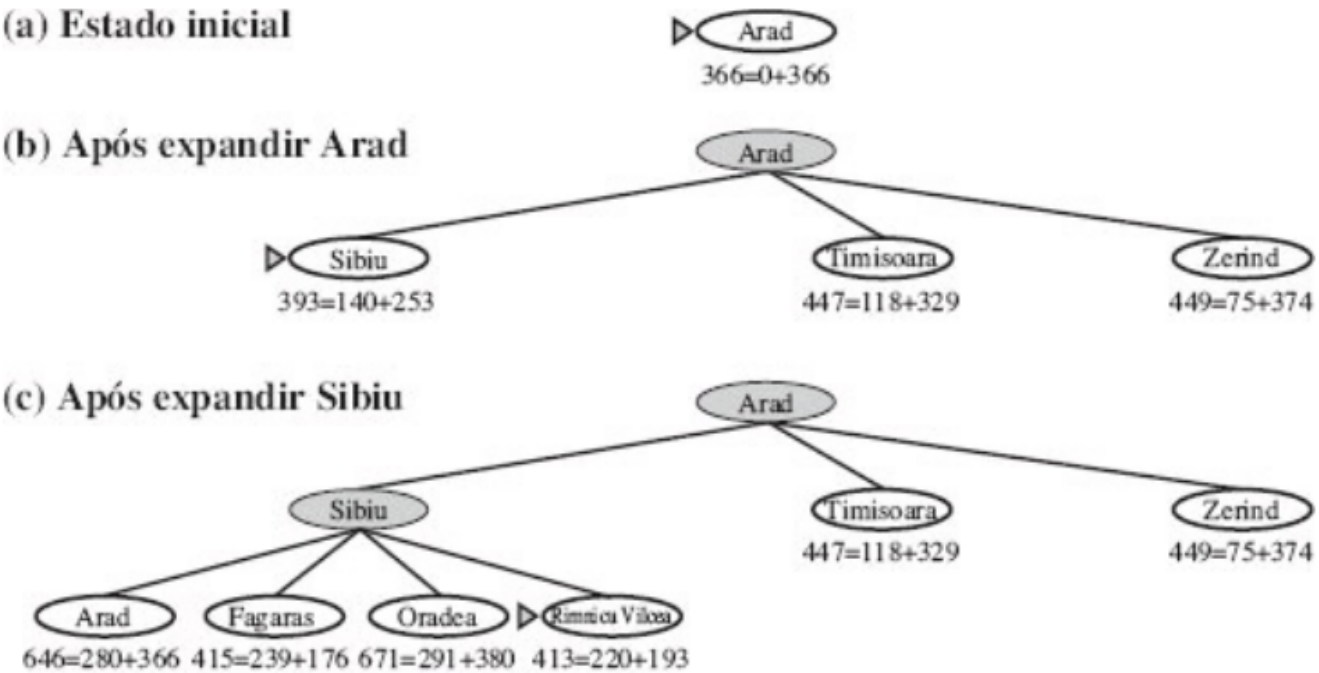
- A tabela abaixo contém a distância Euclidiana de algumas cidades até Bucareste
  - Essas distâncias podem ser usadas como heurística $h(n)$;
- Consideremos que uma pessoa deseja sair de Arad e chegar em Bucareste.

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

Fonte: [Russel and Norvig, 2013]

Busca A* - Exemplo [Russel and Norvig, 2013]

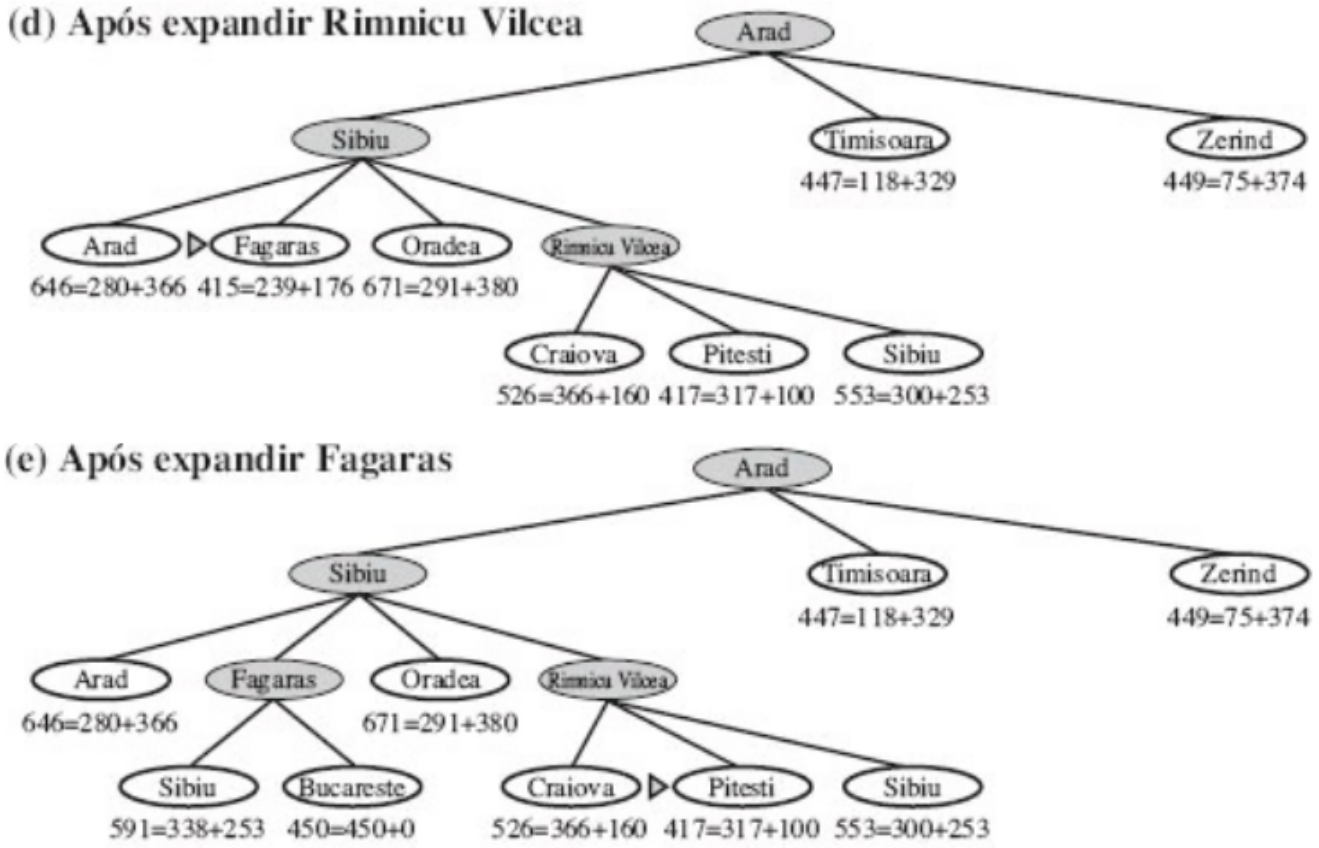
- Ordem de execução da Busca A*.



Fonte: [Russel and Norvig, 2013]

Busca A* - Exemplo [Russel and Norvig, 2013]

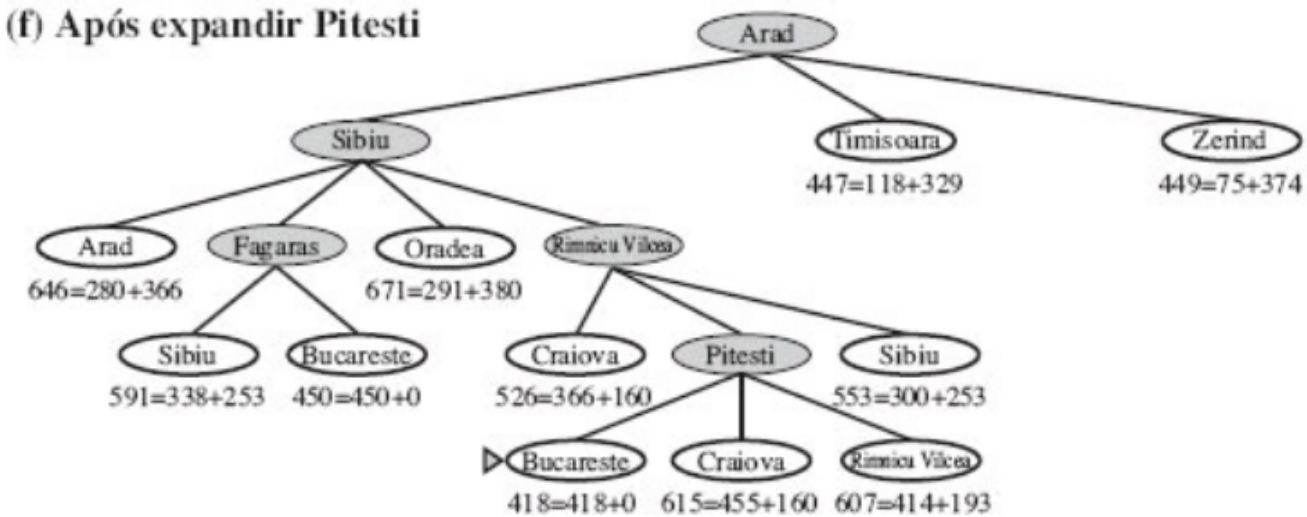
- Ordem de execução da Busca A*.



Fonte: [Russel and Norvig, 2013]

Busca A* - Exemplo [Russel and Norvig, 2013]

- Ordem de execução da Busca A*.



Fonte: [Russel and Norvig, 2013]

Busca A*

- Características:
  - Completa: garante a seleção do estado-objetivo;
  - Ótima: se a heurística $h(n)$ for admissível [Coppin, 2004] [Russel and Norvig, 2013];
- Complexidade:
  - Indefinida, uma vez que depende da heurística $h(n)$ utilizada, e pode variar de acordo com o erro absoluto ou do erro relativo da mesma [Russel and Norvig, 2013].

Comparação - Busca Informada

Comparação - Busca Informada

- Em relação à complexidade, os métodos podem resumidos pela tabela abaixo:

Método	Tempo	Espaço
Best-First Search	$\mathcal{O}(b^m)$	$\mathcal{O}(b^m)$
Busca A*	-	-

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Legenda:

- b: Número de ramificações (*branching factor*);
- m: Profundidade máxima da árvore;

Comparação - Busca Informada

- Em relação à completude/completeza temos:

Método	Completo
Best-First Search	Não
Busca A*	Sim

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Comparação - Busca Informada

- Em relação à otimalidade temos:

Método	Ótimo
Best-First Search	Não.
Busca A*	Somente se a heurística $h(n)$ for admissível.

Fonte: Próprio autor, baseado em [Russel and Norvig, 2013].

Referências I

Aluisio, S. M. (2020).
Busca uniforme.
[Online]; acessado em 13 de Outubro de 2020. Disponível em:
https://sites.icmc.usp.br/sandra/G2_t2/Busca.html.

Coppin, B. (2004).
Artificial intelligence illuminated.
Jones and Bartlett illuminated series. Jones and Bartlett Publishers, 1 edition.

da Silva, D. M. (2014).
Inteligência Artificial - Slides de Aula.
IFMG - Instituto Federal de Minas Gerais, Campus Formiga.

Hruschka, E. R. (2014).
Inteligência artificial - resolução de problemas via busca.
[Online]; acessado em 13 de Outubro de 2020. Disponível em:
http://wiki.icmc.usp.br/images/1/11/Aula2_Busca.pdf.

Johnson, M. (2008).
Informed search methods.
[Online]; acessado em 06 de JUlho de 2021. Disponível em:
https://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html.

Russel, S. and Norvig, P. (2013).
Inteligência artificial.
Campus - Elsevier, 3 edition.