

Algoritmos de procura

Puzzle 8

Procura não informada

Problema Puzzle8

- O problema consiste em resolver um puzzle (de 8 peças) em que o algoritmo tem de descobrir a sequência de operações que permite passar do estado inicial até ao estado final
- Os dois estados poderiam ser representados desta forma (matriz de 3*3):

Estado Inicial:

1	2	5
3	4	
6	7	8

Estado final:

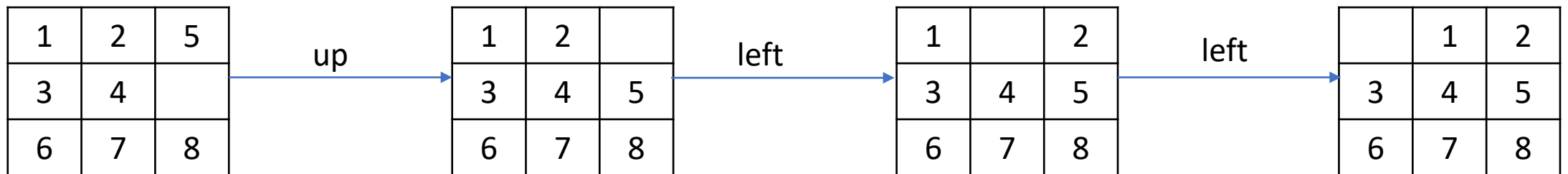
	1	2
3	4	5
6	7	8

Problema Puzzle8

- As operações que se podem operar neste problema são, por exemplo, as seguintes:
 - Mover o espaço em branco para baixo (*down*)
 - Mover o espaço em branco para cima (*up*)
 - Mover o espaço em branco para a direita (*right*)
 - Mover o espaço em branco para a esquerda (*left*)

Problema Puzzle8

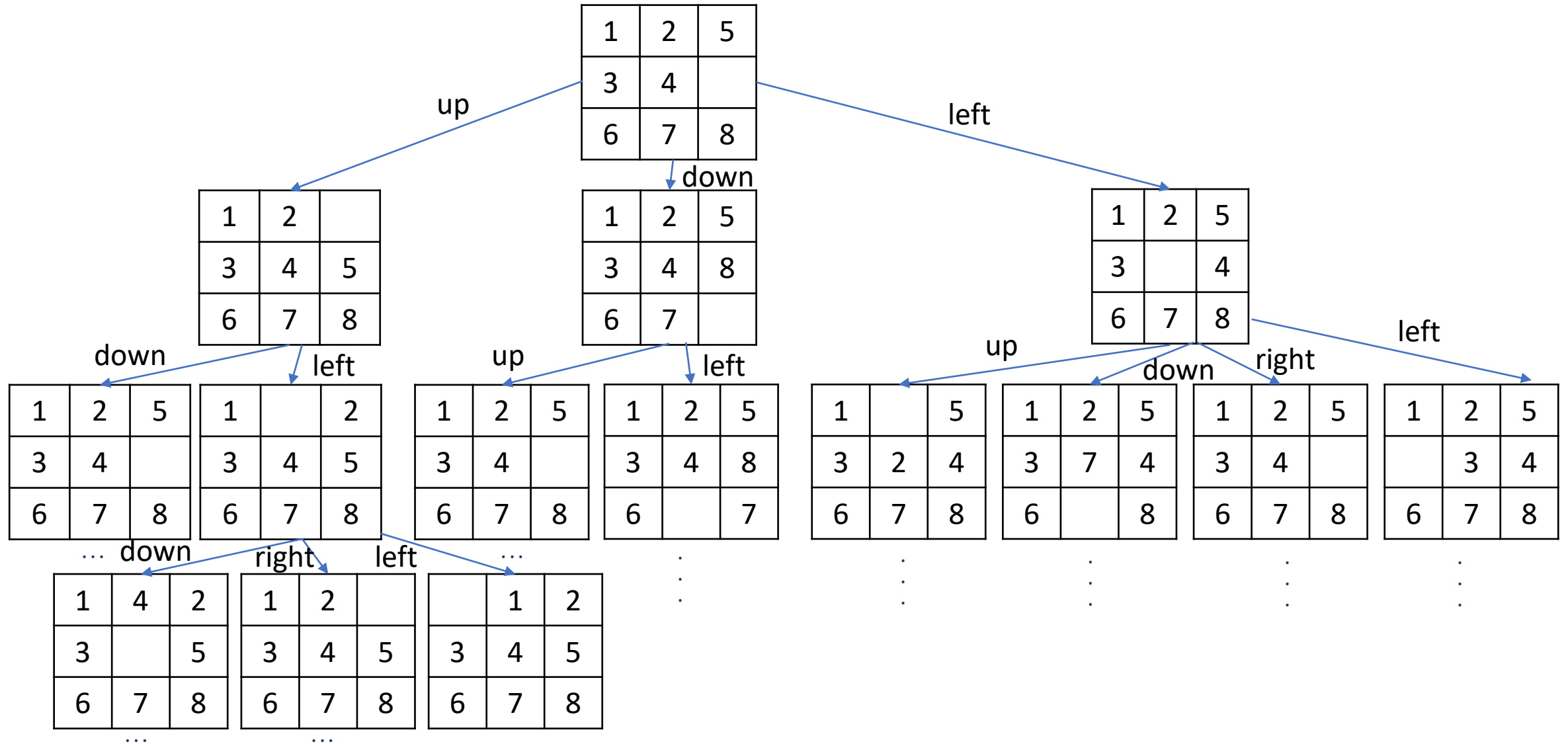
- A resolução consiste de uma sequência de operações que causam modificações sucessivas ao estado inicial, conduzindo a um estado final ou objetivo
- Relativamente ao problema representado anteriormente, uma resolução possível é a seguinte sequência de ações: mover o espaço para cima, mover o espaço para a esquerda, e tornar a mover o espaço para a esquerda



Problema Puzzle8

- O Espaço de Estados ou Espaço de Procura de um problema é o conjunto de todos os estados que representam configurações do mundo relativamente aos aspetos relevantes para o problema
- O espaço de procura pode ser representado por um grafo, em que os nós estão associados a um estado e os arcos de um grafo representam ações
- Quanto maior for o espaço de estados, maior serão em média os recursos (tempo e memória) que um algoritmo de procura consumirá para resolver um dado problema
- No caso do 8-puzzle, um estado é representado por uma matriz de 3×3 , portanto existe um número de estados igual ao número de permutações dos nove números que representam um estado, isto é, 362.880 estados

Árvore de procura para o puzzle-8



Procura no espaço de estados

- Após a formulação do problema, o estado final deve ser “procurado”
- Para isto, deve-se usar um método de procura para encontrar a ordem correta de aplicação dos operadores (ações) que levam do estado inicial ao estado final
- Se a procura termina com sucesso é possível executar a solução (= conjunto ordenado de ações/operadores a aplicar)

- No projeto fornecido:

Agente = **Formular** (estado inicial e objetivo) + **Procurar** (encontrar uma solução com um método de procura) + **Executar** (sequência de ações que aplicadas a um estado inicial permitem chegar ao objetivo)

Procura no espaço de estados

function GRAPH-SEARCH(problem) returns a solution, or failure
initialize the frontier using the initial state of problem
initialize the explored set to be empty
while(frontier is not empty)
remove the first node from the frontier
if the node contains a goal state then return the corresponding solution
add the node to the explored set
expand the node, adding the resulting nodes to the frontier only if
not in the frontier or explored set
return failure

Algoritmos de procura cega

- Largura Primeiro (*Breadth First Search*)
 - Uniforme (*Uniform Search*)
 - Profundidade Primeiro (*Depth First Search*)
 - Profundidade Limitada (*Depth Limited Search*)
 - Aprofundamento Progressivo (*Iterative Deepening Search*)
-
- O método de procura é idêntico nos métodos de procura. Ou seja, o primeiro nó da fronteira é o nó a ser explorado até se encontrar um nó objetivo
 - A maior diferença entre os métodos de procura é a forma em como os sucessores de um nó são adicionados à fronteira

Procura em Largura (Breadth First Search)

- O nó de menor profundidade, mais à esquerda é escolhido para gerar sucessores
- Os nós sucessores são adicionados à fronteira pela ordem com que são gerados (ou seja no fim)
- Os nós de profundidade d são explorados antes dos nós de profundidade $d+1$
- É completo e ótimo (se custo for igual para todos as ações)
- Utiliza uma lista de nós explorados
- Um nó só é adicionado à fronteira se o estado correspondente:
 - Não esteja já na fronteira e não tenha sido explorado

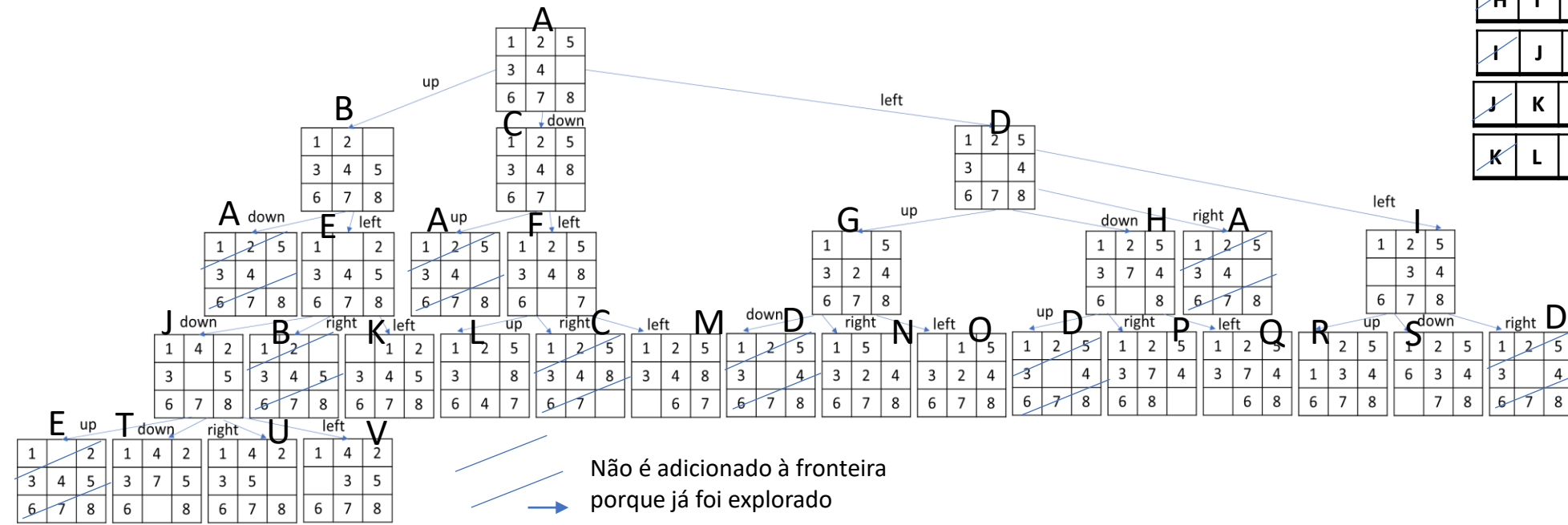
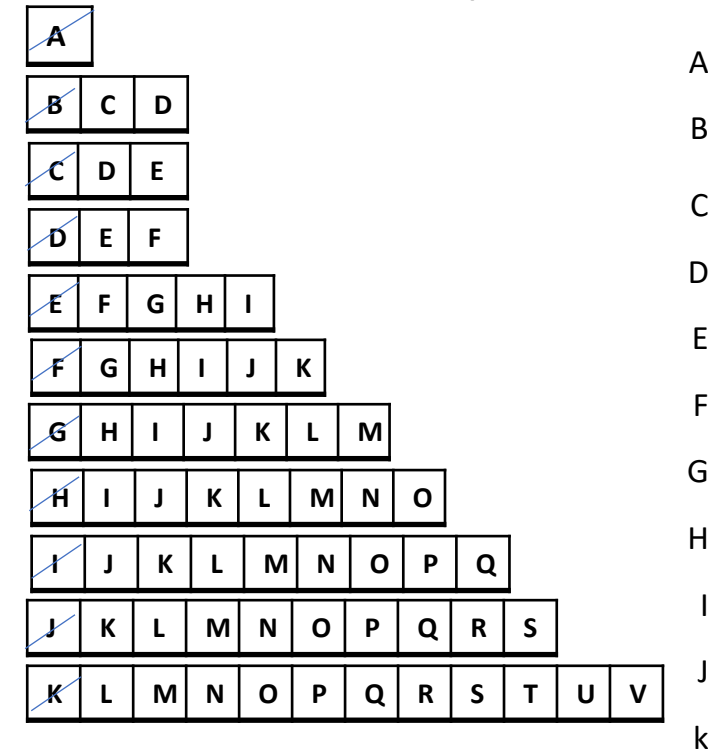
Procura em Largura (*BreadthFirstSearch*)

Sempre que um nó explorado não é objetivo, é adicionado à lista de explorados

A	B	C	D	E	F	G	H	I	J
---	---	---	---	---	---	---	---	---	---

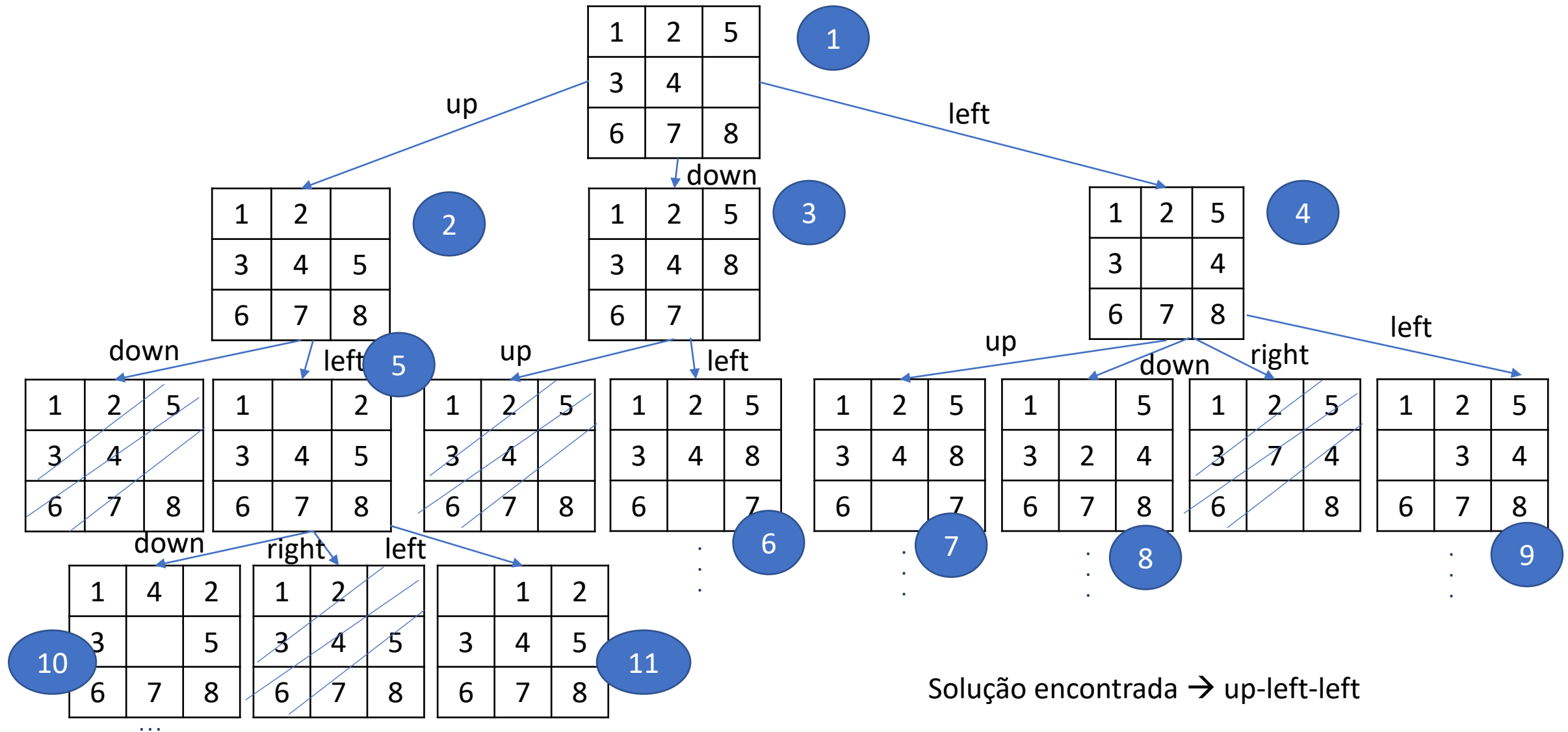
Puzzle8

- Fronteira:
- Nós explorados:



Não é adicionado à fronteira porque já foi explorado

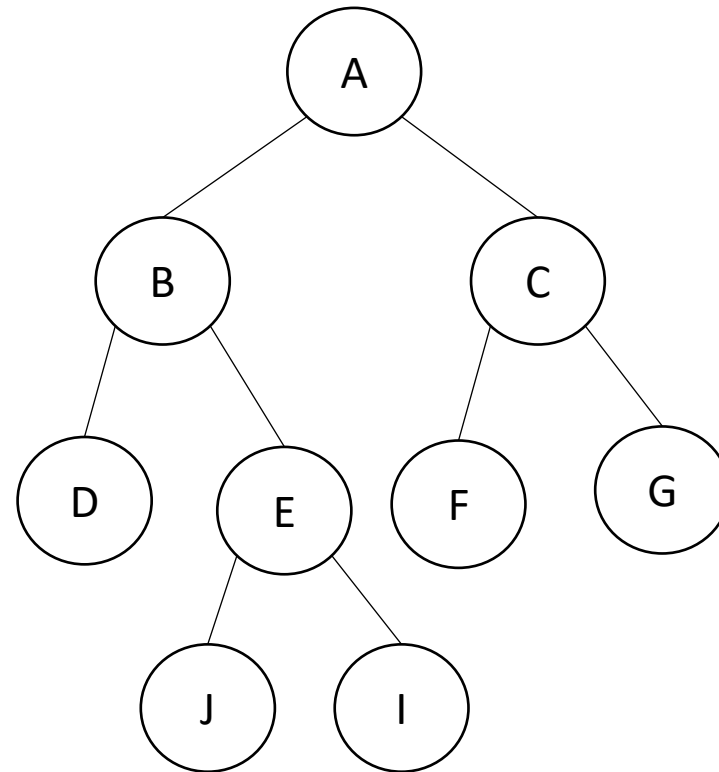
Procura em Largura – Nós explorados



Considere o seguinte exemplo:

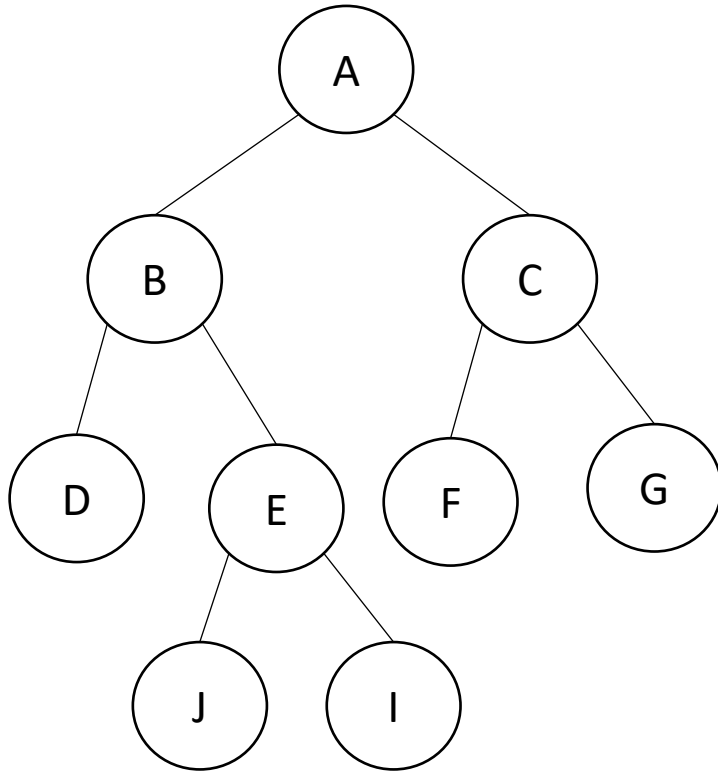
Nó inicial - A

Nó objetivo - F

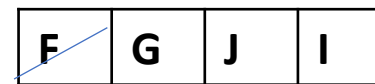
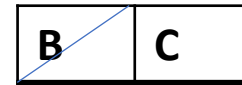


Procura em Largura (*BreadthFirstSearch*)

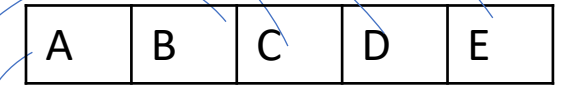
Sempre que um nó explorado não é objetivo, é adicionado à lista de explorados



- Fronteira:



- Nós explorados:



A

B

C

D

E

F

Procura Uniforme (*UniformSearch*)

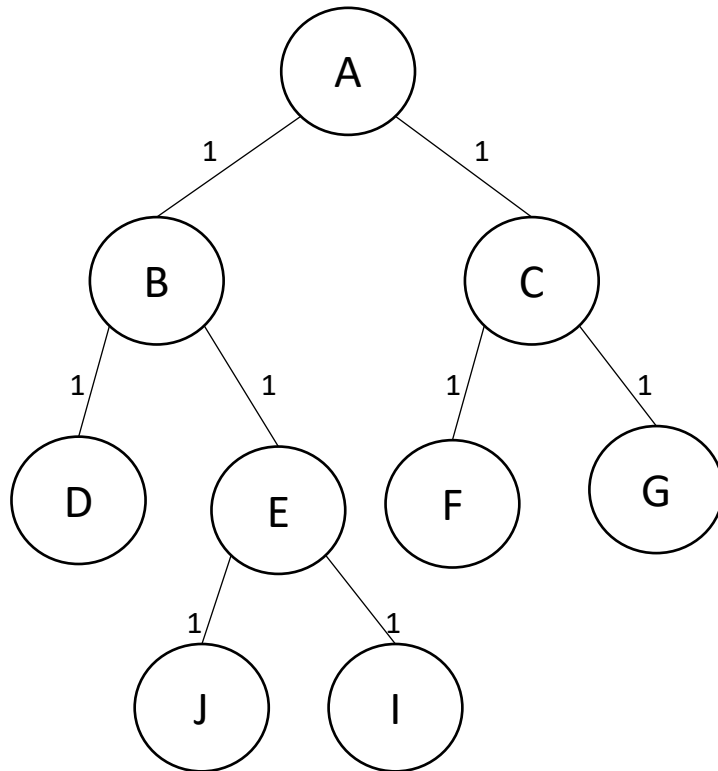
$$f = g(n)$$

- É ótimo
- Tal como na procura em largura utiliza uma lista de nós explorados
- A fronteira é uma lista de nós ordenada de forma crescente pelo valor de f , em que neste caso o valor f de um nó é o custo desde o nó inicial até chegar a esse nó (custo = g)
- Um nó só é adicionado à fronteira se o estado correspondente:
 - Não esteja já na fronteira e não tenha sido explorado
 - Já está na fronteira mas o custo do caminho até chegar a esse nó a partir do nó inicial (ou seja, o valor de g) for inferior ao do estado que já está na fronteira. Neste caso, o nó que está na fronteira deve ser eliminado e o novo nó é adicionado (ver método `addSuccessorsToFrontier` da procura uniforme)

Procura Uniforme (*UniformSearch*)

Sempre que um nó explorado não é objetivo, é adicionado à lista de explorados

Neste exemplo vamos considerar que o custo de todas as ações é igual a 1



$$f = g(n)$$

• Fronteira:

~~A₀~~

~~B₁~~ C₁

~~C₁~~ D₂ E₂

~~D₂~~ E₂ F₂ G₂

~~E₂~~ F₂ G₂

~~F₂~~ G₂ J₃ I₃

• Nós explorados:

A B C D E

A

B

C

D

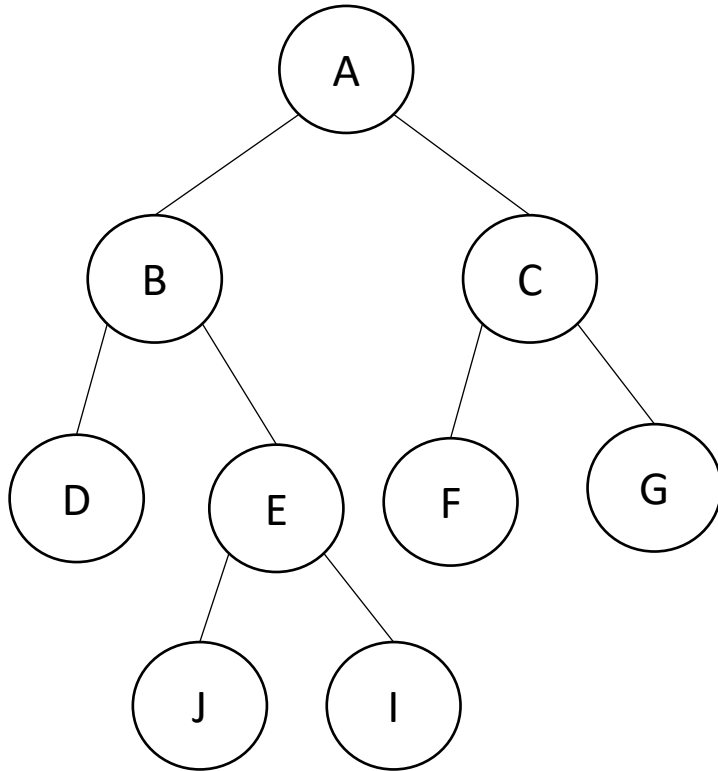
E

F

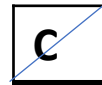
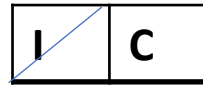
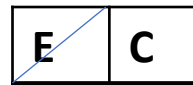
Procura em Profundidade (*DepthFirstSearch*)

- Começa no nó raiz e a partir daí expande o nó no nível mais profundo da árvore
- Os nós sucessores são adicionados ao início da fronteira
- É completo mas não é ótimo
- Não utiliza uma lista de nós explorados
- Um nó só é adicionado à fronteira se o estado correspondente:
 - Não esteja já na fronteira e não for igual a um dos pais até ao topo da árvore

Procura em Profundidade (*DeapthFirstSearch*)



- Fronteira:



- Nós explorados:

A

B

D

E

J

I

C

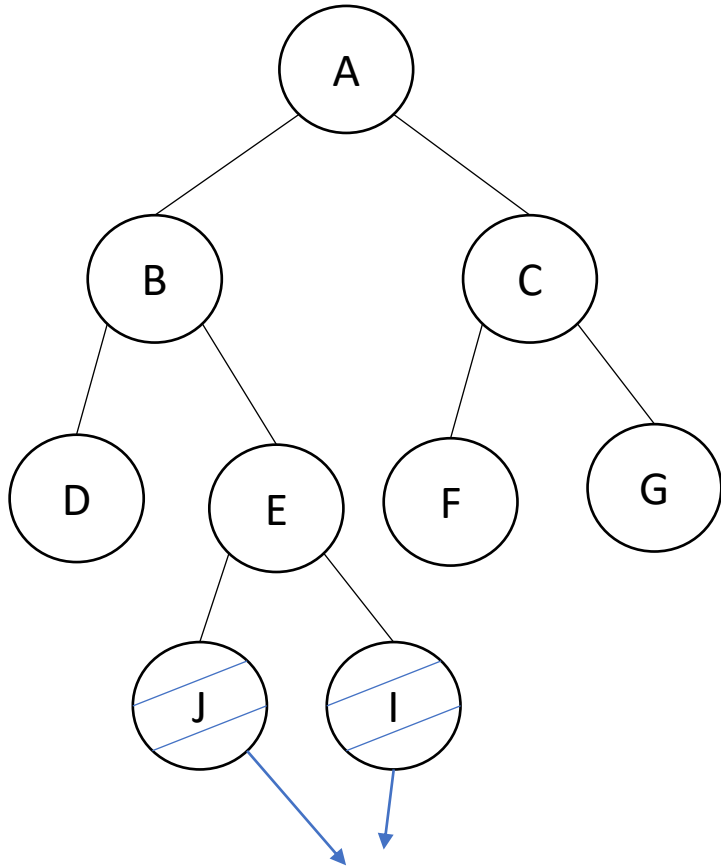
F

Procura em Profundidade Limitada (*DepthLimitedSearch*)

- Método de procura idêntico ao da profundidade primeiro
- Impõe um limite à profundidade da árvore
- Não é completo e não é ótimo

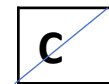
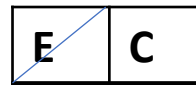
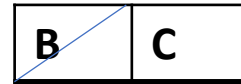
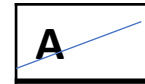
Procura em Profundidade Limitada (*DepthLimitedSearch*)

Limite=2



Todos os nós com limite>2 não serão adicionados à fronteira

• Fronteira:



• Nós explorados:

A

B

D

E

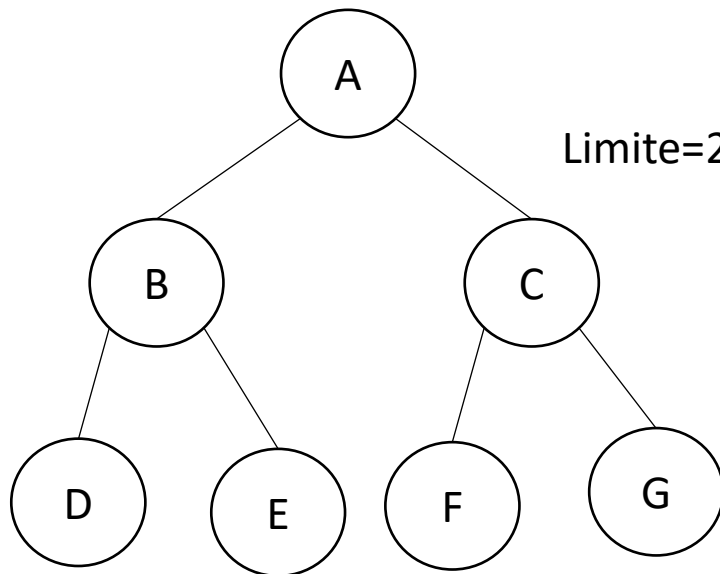
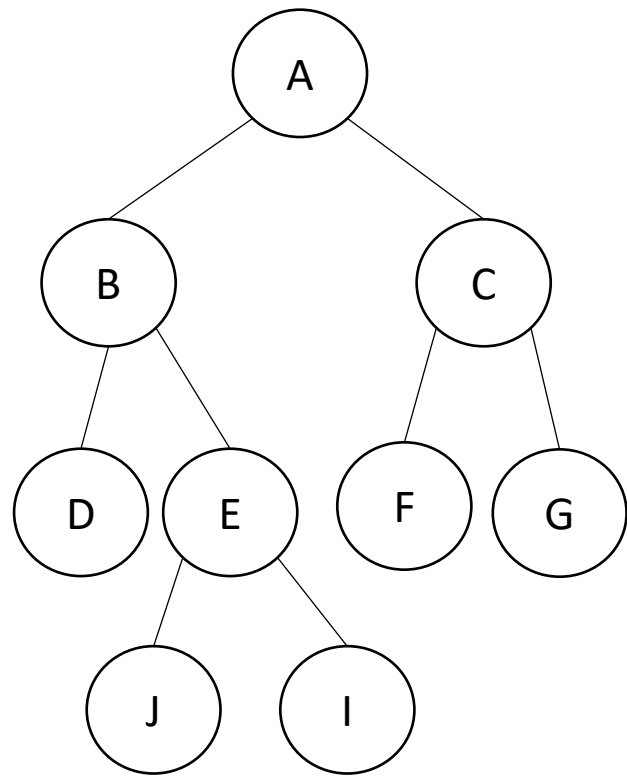
C

F

Procura por Aprofundamento Progressivo (*IterativeDeepeningSearch*)

- É aplicado múltiplas vezes o algoritmo de procura em profundidade limitada do limite 0 até ao limite n até encontrar um solução para o problema
- Tal como vai ser implementado nas aulas, se não encontrar uma solução entra em loop
- É completo e é ótimo (se custo for igual para todos as ações)

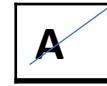
Procura por Aprofundamento Progressivo (*IterativeDeepeningSearch*)



• Fronteira:

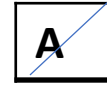
• Nós explorados:

Limite=0

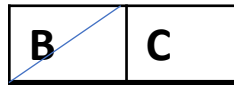


A

Limite=1



A



B

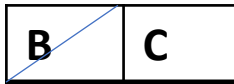


C

Limite=2



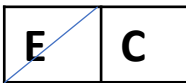
A



B



D



E



C

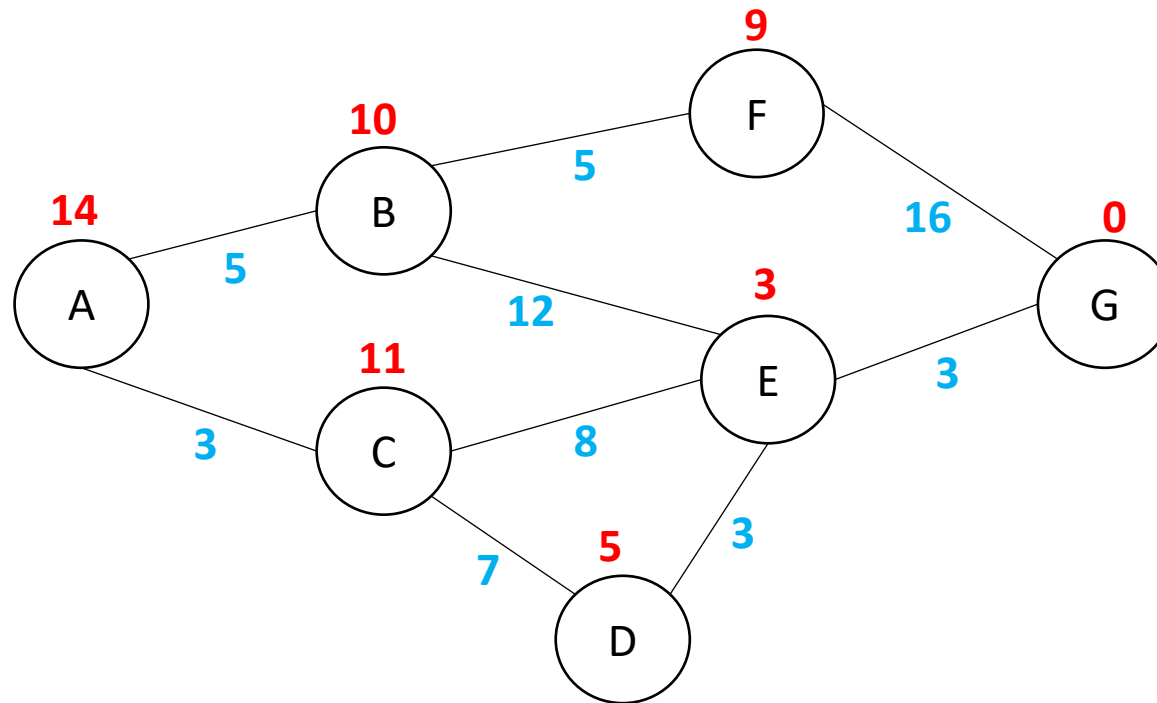


F

Considere o seguinte exemplo:

vermelho – valor da heurística

azul – custo do caminho

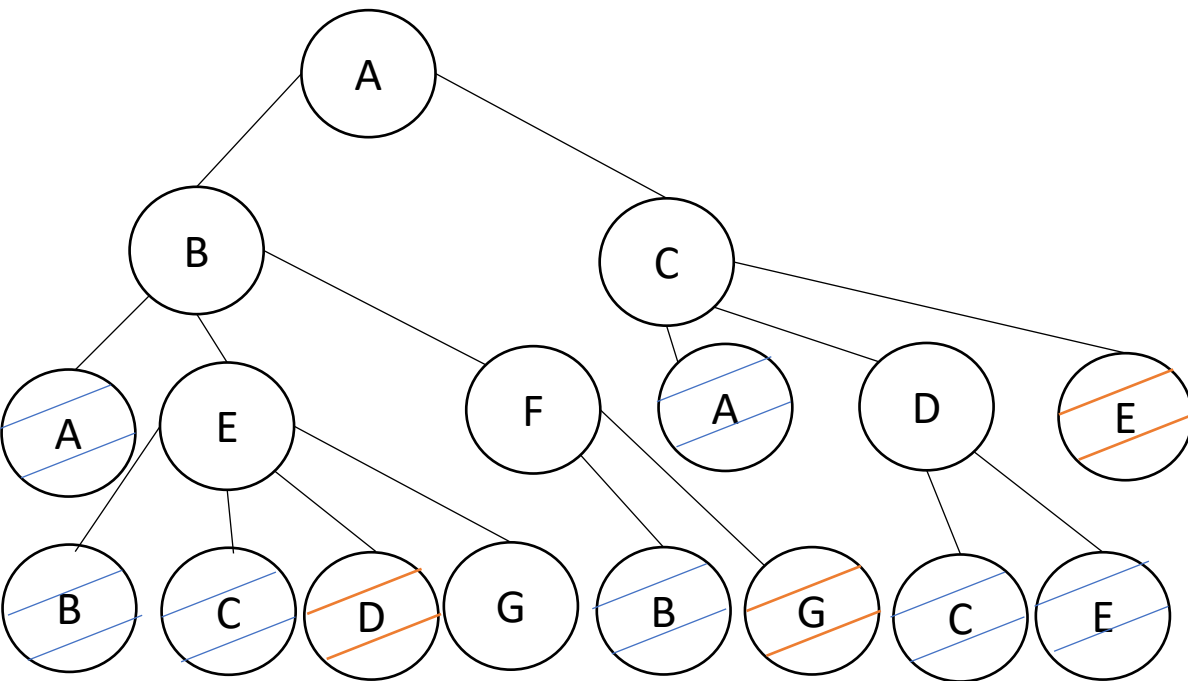
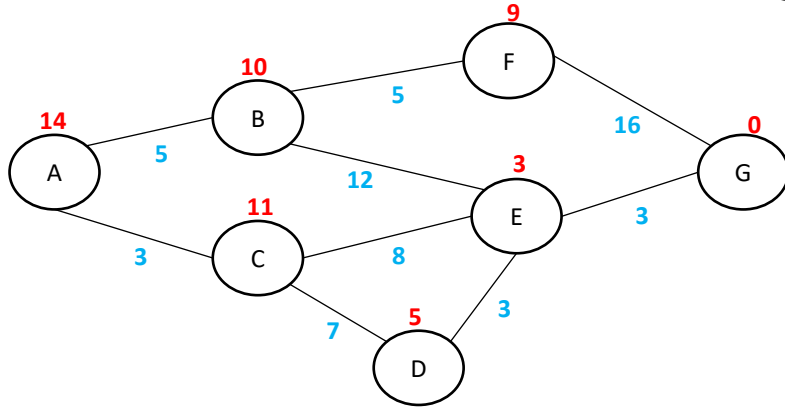


Nó inicial – A

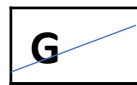
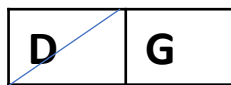
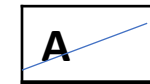
Nó objetivo - G

Procura em Largura (*BreadthFirstSearch*)

Sempre que um nó explorado não é objetivo, é adicionado à lista de explorados



• Fronteira:



• Nós explorados:



A

B

C

E

F

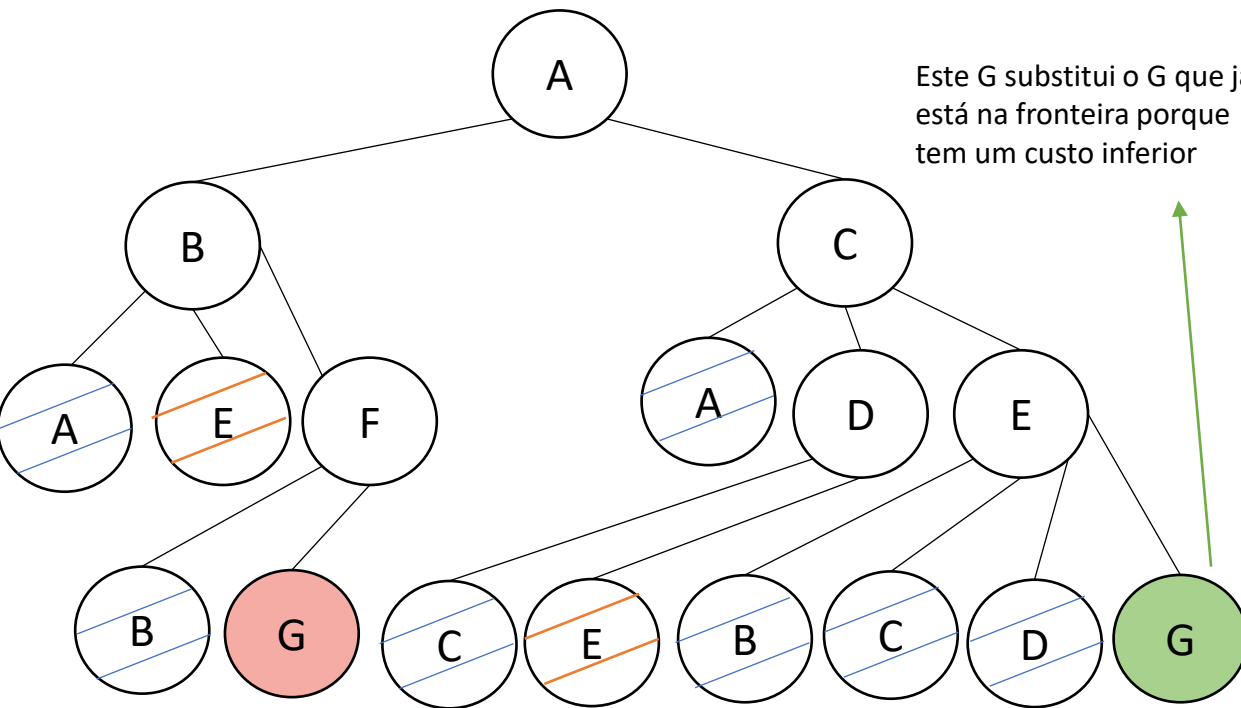
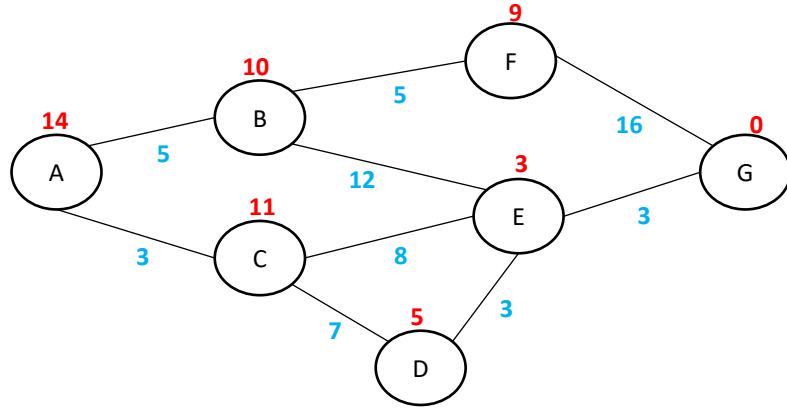
D

G

Procura Uniforme (*UniformSearch*)

$$f = g(n)$$

Sempre que um nó explorado não é objetivo, é adicionado à lista de explorados



Este G substitui o G que já está na fronteira porque tem um custo inferior

• Fronteira:

~~A₀~~

~~C₃~~ | B₅

~~B₅~~ | D₁₀ | E₁₁

~~D₁₀~~ | F₁₀ | E₁₁

~~F₁₀~~ | E₁₁

~~E₁₁~~ | G₂₆

~~G₁₄~~

• Nós explorados:

A C B D F E

A

C

B

D

F

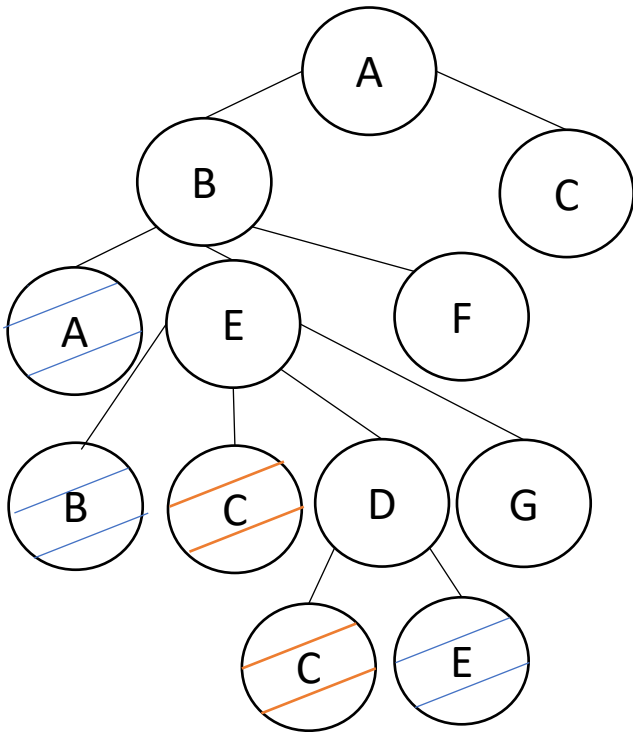
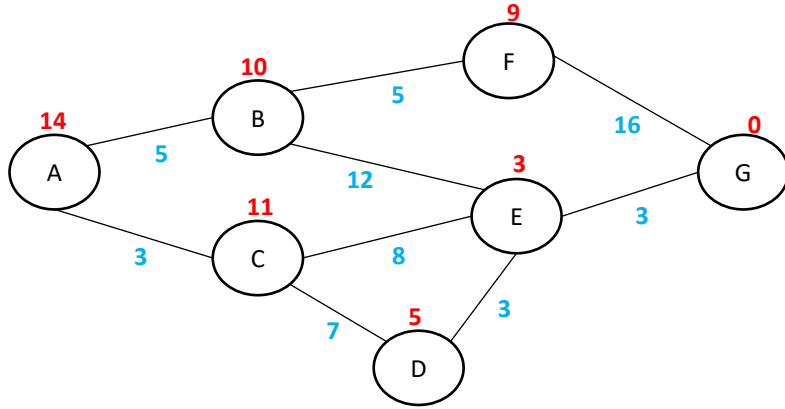
E

G

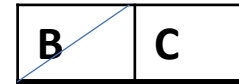
— Não é adicionado à fronteira
— porque já foi explorado

— Não é adicionado à fronteira
— porque já está na fronteira e tem um custo superior

Procura em Profundidade (*DeapthFirstSearch*)



• Fronteira:



• Nós explorados:

A

B

E

D

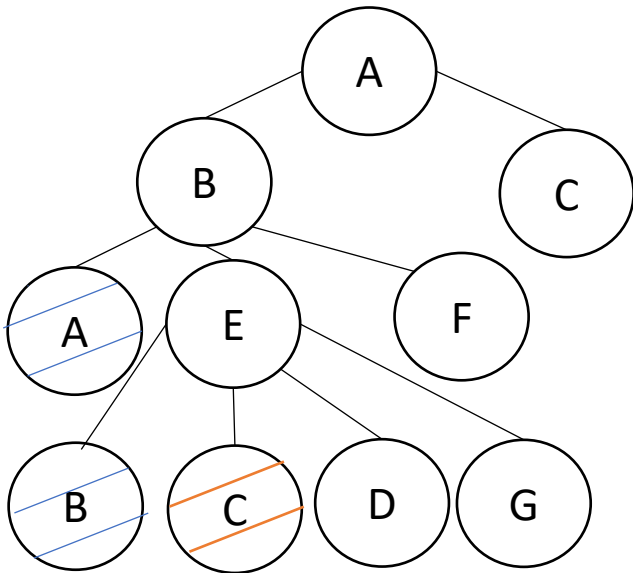
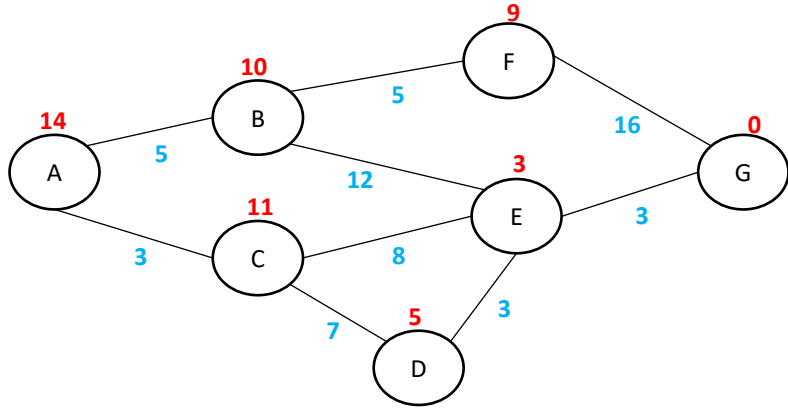
G

↖ ↗
→ Não é adicionado à fronteira porque um dos pais desse nó é igual a esse nó

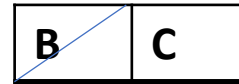
↖ ↗
→ Não é adicionado à fronteira porque já está na fronteira

Procura em Profundidade Limitada (*DepthLimitedSearch*)

Limite=3



• Fronteira:



• Nós explorados:


A


B

E

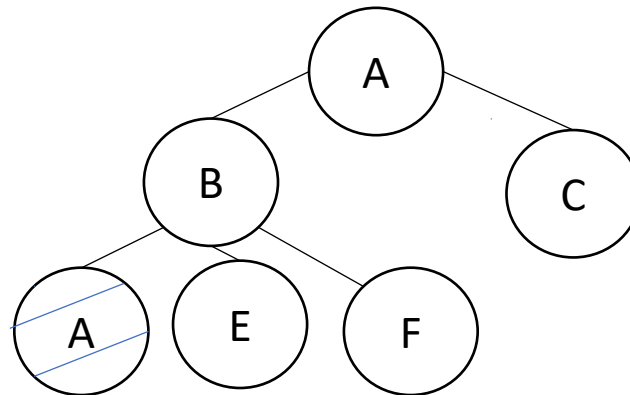
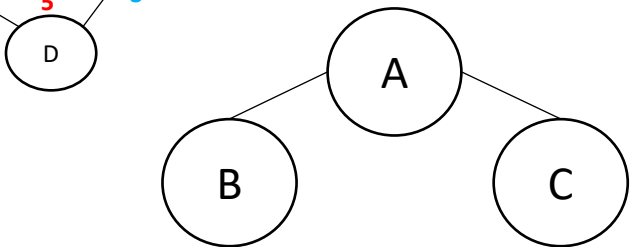
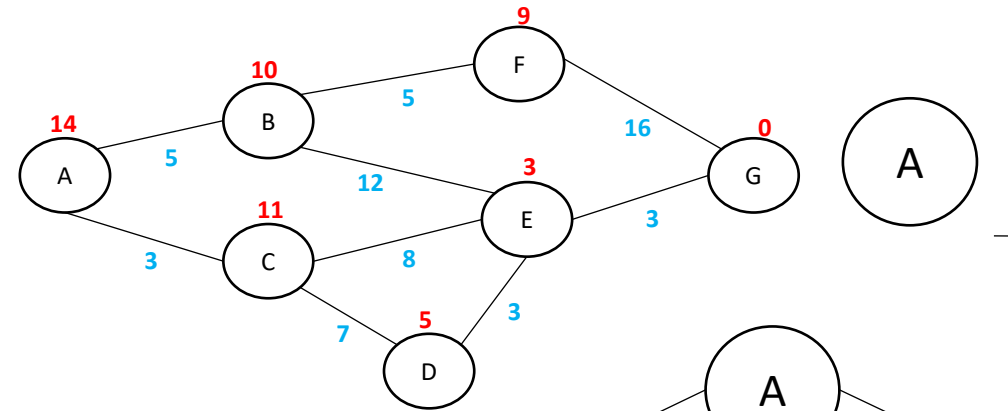
D

G

 Não é adicionado à fronteira porque um dos pais desse nó é igual a esse nó

 Não é adicionado à fronteira porque já está na fronteira

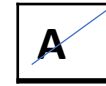
Procura por Aprofundamento Progressivo (*IterativeDeepeningSearch*)



• Fronteira:

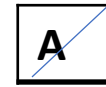
• Nós explorados:

Limite=0



A

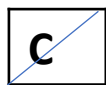
Limite=1



A

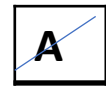


B

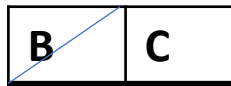


C

Limite=2



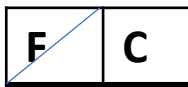
A



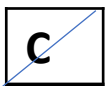
B





E





F



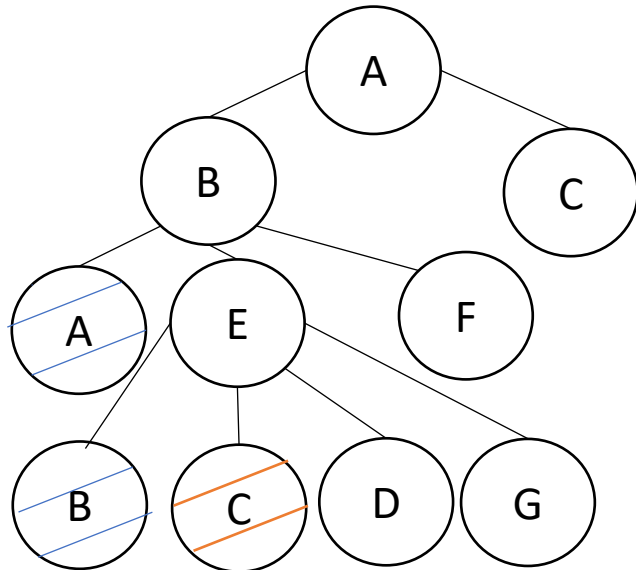
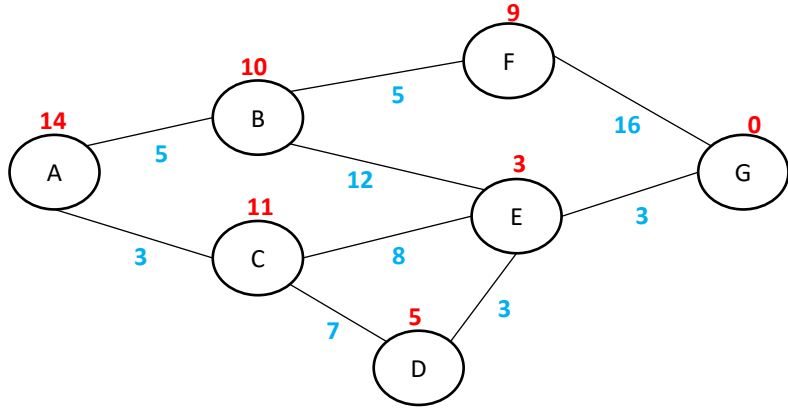
C

 Não é adicionado à fronteira porque um dos pais desse nó é igual a esse nó


 Não é adicionado à fronteira porque já está na fronteira


Procura por Aprofundamento Progressivo (*IterativeDeepeningSearch*)

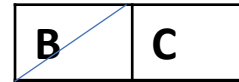
continuação



Limite=3

• Fronteira:

...



• Nós explorados:

...


A


B

E

D

G

 Não é adicionado à fronteira porque um dos pais desse nó é igual a esse nó

 Não é adicionado à fronteira porque já está na fronteira

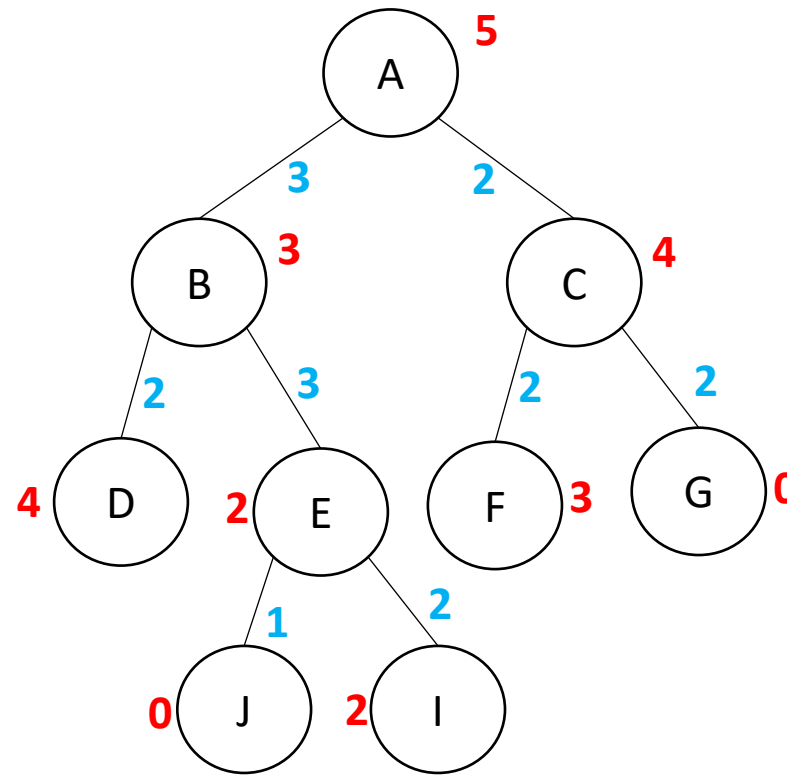
Considere o seguinte exemplo:

vermelho – valor da heurística

azul – custo do caminho

A – nó inicial

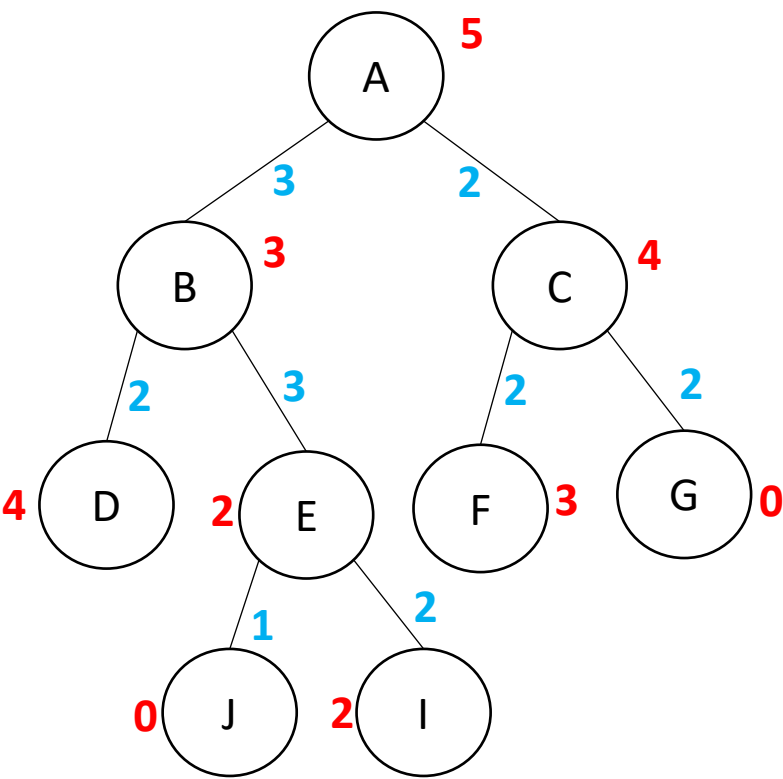
G e J – nós objetivo



Considere o seguinte exemplo:

vermelho – valor da heurística

azul – custo do caminho



A – nó inicial
G e J – nós objetivo

Largura:

~~A~~
A
~~B C~~
B
~~C D E~~
C
~~D E F G~~
D
~~E F G~~
E
~~F G J I~~
F
~~G J I~~
G

Profundidade:

~~A~~
A
~~B C~~
B
~~D E C~~
D
~~E C~~
E
~~F I C~~
J

Profundidade Limitada:
(limite =2)

~~A~~
A
~~B C~~
B
~~D E C~~
D
~~E C~~
E
C
F
G

Uniforme:

~~A~~₀
A
~~C~~₂ B₃
C
~~B~~₃ F₄ G₄
B
F₄ G₄ D₅ E₆
F
G₄ D₅ E₆
G

Aprofundamento progressivo:

~~A~~
A Limite=0
~~A~~
A
~~B C~~
B
~~C~~
C Limite=1
~~A~~
A
~~B C~~
B
~~D E C~~
D
~~E C~~
E
C
F
G
32
G Limite=2