PROCURA EM ESPAÇO DE ESTADOS

Inteligência Artificial

Joaquim Filipe

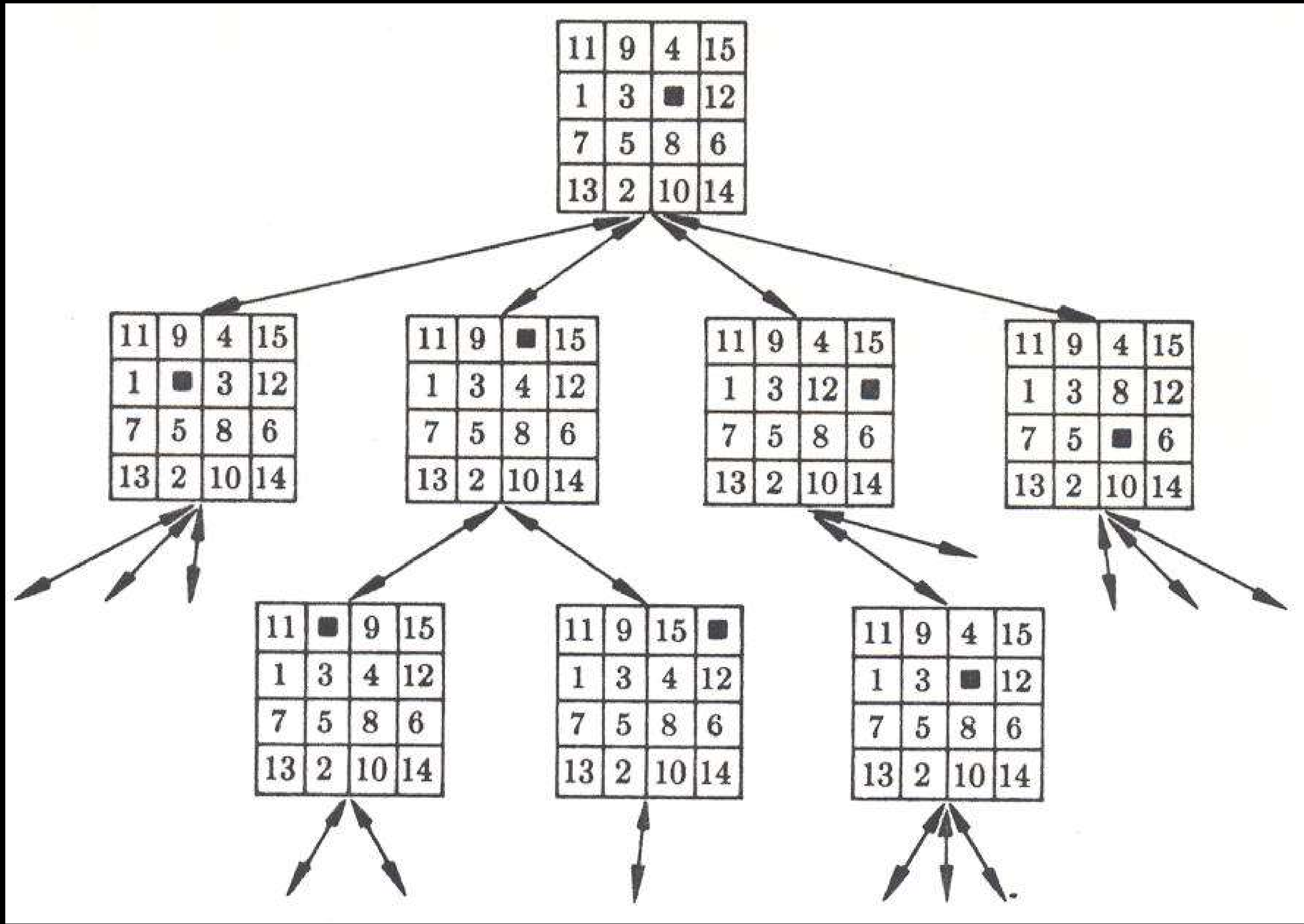
RESOLUÇÃO DE PROBLEMAS

- A inteligência artificial destina-se a construir máquinas para ajudar a resolver problemas em que de momento as pessoas são melhores.
- Daí que um dos tópicos proeminentes seja o estudo e implementação de métodos automáticos de resolução problemas.

ESPAÇO DE ESTADOS

- Uma forma simples de resolver problemas que não se podem resolver com fórmulas ou algoritmos, consiste em explorar o espaço de possibilidades tentando vários caminhos possíveis até encontrar a solução.
- Neste caso um problema terá de ser equacionado em termos de:
  - Estados
  - Operadores (de transição de estados)
- Terá ainda de ser definido
  - O estado final (poderá ser com uma função de teste)

EXEMPLO (PUZZLE DE 15)



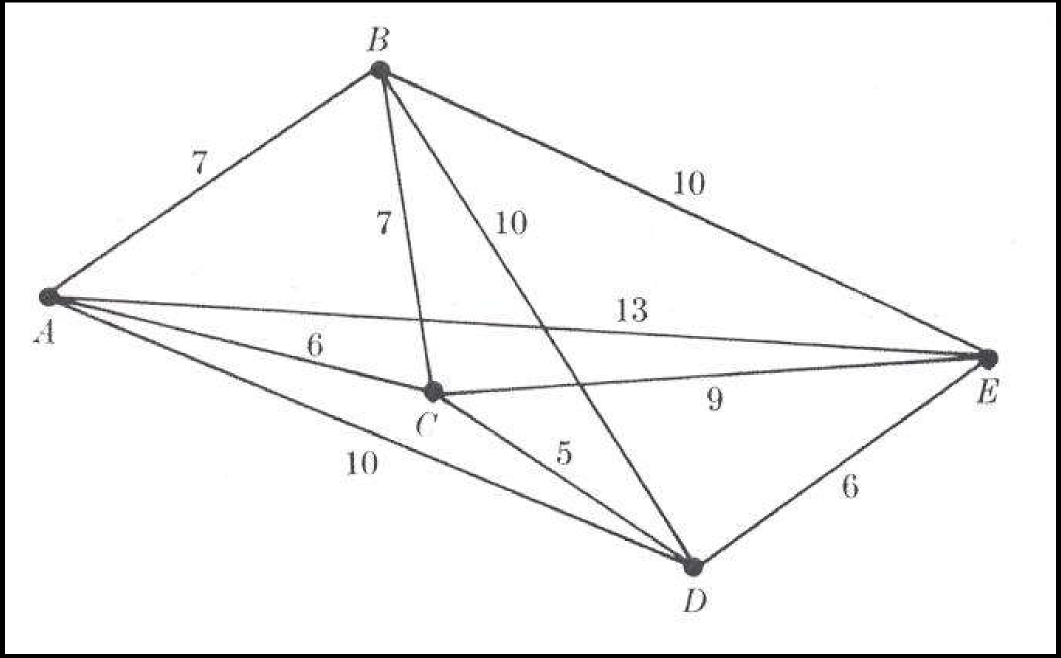
(Nils Nilsson, Problem Solving Methods in AI, p.5)

REPRESENTAÇÃO

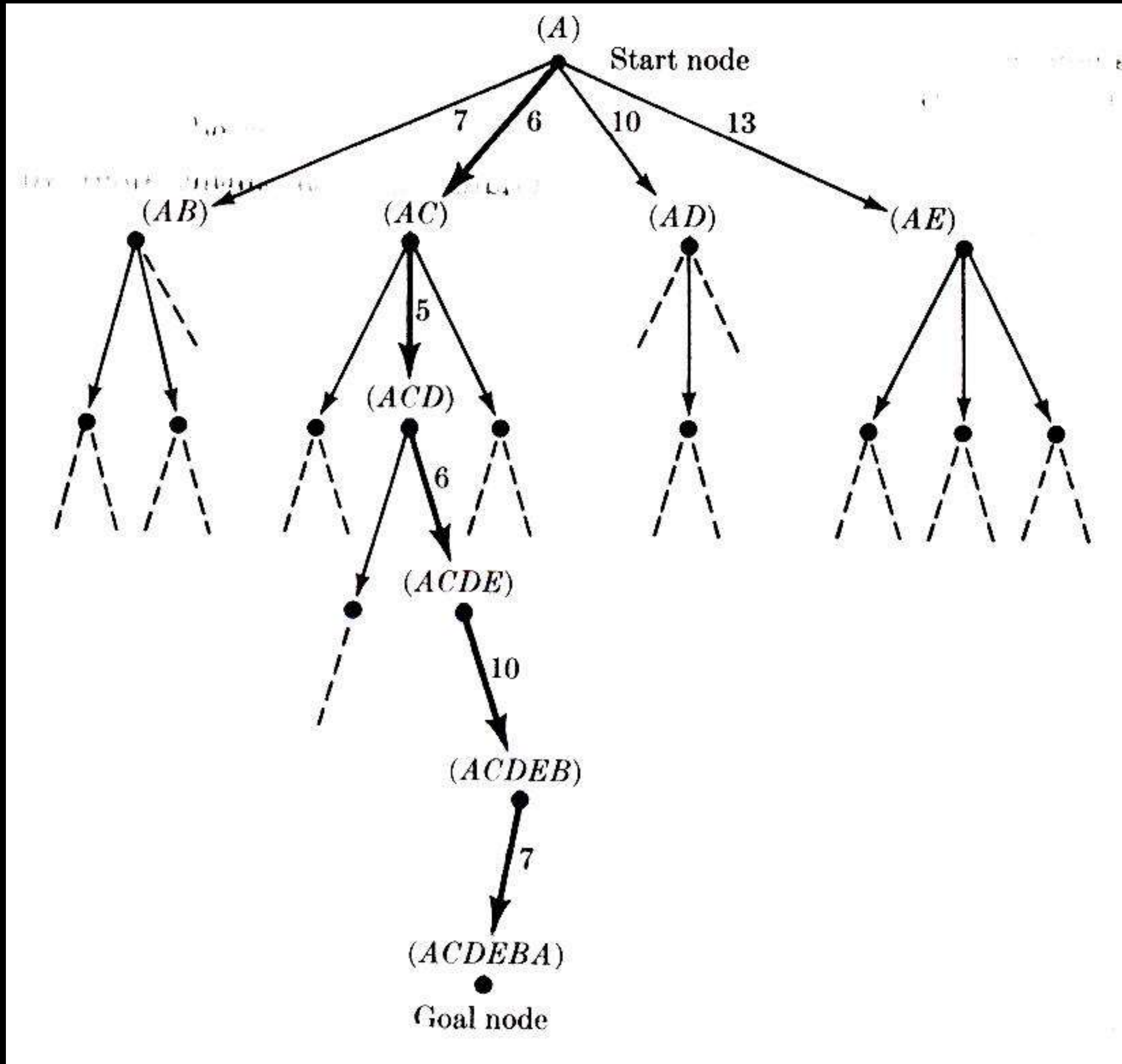
- Grafo:
  - O Espaço de Estados pode ser representado por um grafo dirigido acíclico.
- Nó:
  - cada nó do grafo representa um estado do problema, com alguma informação adicional (pointer para o nó que o gerou; etc.)
- Arco:
  - Cada arco do grafo representa uma transição de estado ao longo do processo de resolução do problema.

EXEMPLO

- Caixeiro Viajante.
    - Problema clássico: um caixeiro viajante tem de planear uma viagem em que visita n cidades apenas 1 vez e regressa à cidade de origem, minimizando um custo (normalmente a distância percorrida).
- Representação gráfica:



GRAFO DO PROBLEMA



(Nils Nilsson, Problem Solving Methods in AI, p.5)

REPRESENTAÇÃO DE UM ESTADO

- Um estado no problema do caixeiro viajante pode ser representado como um conjunto de 2 elementos: {C, V}
  - C = Conjunto de cidades a visitar.
  - V = Conjunto das cidades já visitadas.
- O estado final é um estado em que
  - $C = \emptyset$
- O estado inicial é o estado em que C contém todas as cidades e $V = \emptyset$.

SOLUÇÃO

- A solução é obtida aplicando operadores às descrições dos estados até que surja a descrição do estado final.

ESTRATÉGIAS DE EXPLORAÇÃO DE ÁRVORES

- Conceitos:
  - Nó inicial
  - Sucessores de um nó: nós que são gerados por aplicação de um dos operadores legais.
  - Expansão de um nó: geração de todos os sucessores
  - Ponteiros para o nó pai: para permitir obter imediatamente a solução a partir do estado final
  - Lista de (nós) abertos: lista com os nós que ainda não foram expandidos
  - Lista de (nós) fechados: lista com os nós que já foram expandidos

MÉTODOS DE PROCURA: EM LARGURA PRIMEIRO (*BREADTH-FIRST*)

1. Nó inicial => ABERTOS
2. Se ABERTOS vazia falha.
3. Remove o primeiro nó de ABERTOS (n) e coloca-o em FECHADOS
4. Expande o nó n. Colocar os sucessores no fim de ABERTOS, colocando os ponteiros para n.
5. Se algum dos sucessores é um nó objectivo sai, e dá a solução. Caso contrário vai para 2.

FLUXOGRAMA DO BREADTH-FIRST

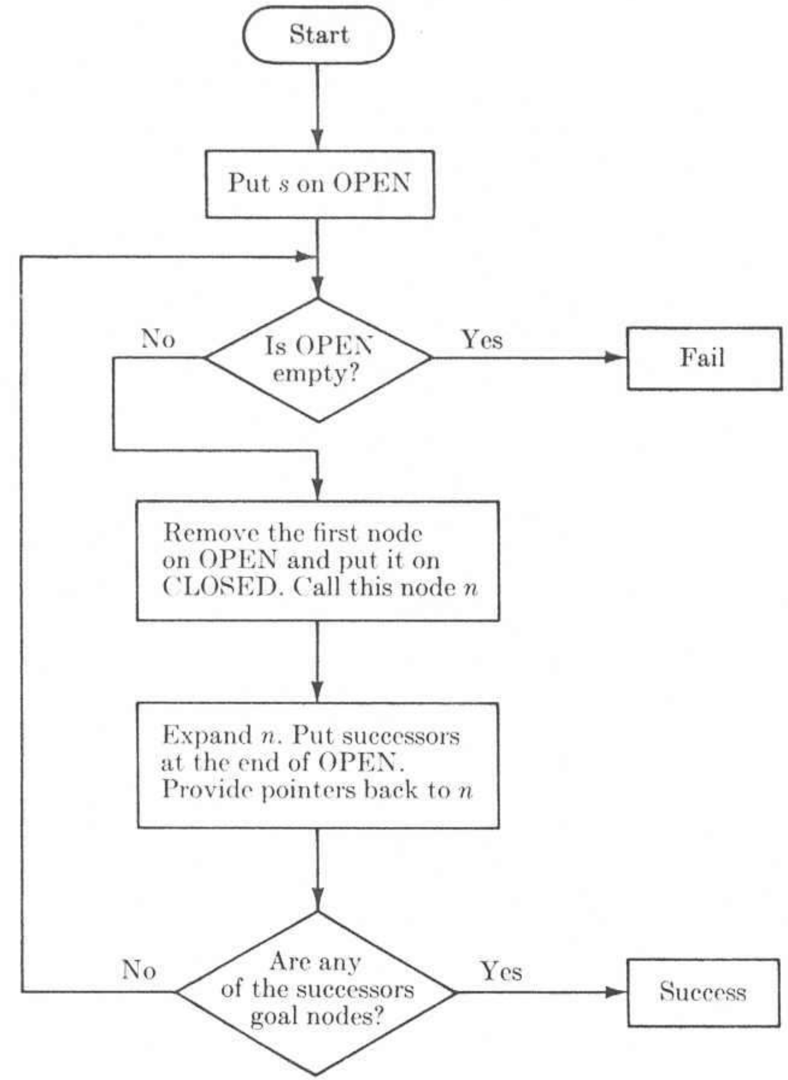


FIG. 3-1 *Flow chart of a breadth-first search algorithm for trees.*

CARACTERÍSTICAS DO BF

- Assume-se que o nó inicial não é um nó objectivo.
- O BF encontra sempre a solução que corresponde ao caminho mais curto.
- Se não houver solução o método termina com falha se o grafo for finito ou não termina se o grafo for infinito.

MÉTODOS DE PROCURA: CUSTO UNIFORME

- Se interessar minimizar o custo em vez da distância, e se os custos associados aos arcos forem diferentes de arco para arco, então é necessário usar uma variante do BF designada "método do custo uniforme" que garante a minimização do custo.

ALGORITMO

1. Nó inicial(s) => ABERTOS. Faz g(s)=0.
2. Se ABERTOS vazia falha.
3. Remove o nó de ABERTOS (n) com menor custo (g) e coloca-o em FECHADOS
4. Se n for um nó objectivo termina e dá a solução.
5. Expande o nó n. Colocar os sucessores em ABERTOS, colocando os ponteiros para n e calculando o g de cada um dos sucessores.
6. Vai para 2.

MÉTODOS DE PROCURA: EM PROFUNDIDADE PRIMEIRO (DEPTH-FIRST)

- Convenciona-se que a profundidade do nó raiz é zero.
- A profundidade de um nó é 1 + a profundidade do antecessor.
- É definido um nível de profundidade máximo a partir do qual os nós não são expandidos

ALGORITMO

1. Nó inicial => ABERTOS
2. Se ABERTOS vazia falha.
3. Remove o primeiro nó de ABERTOS (n) e coloca-o em FECHADOS
4. Se a profundidade de n é maior que d vai para 2.
5. Expande o nó n. Colocar os sucessores no início de ABERTOS, colocando os ponteiros para n.
6. Se algum dos sucessores é um nó objectivo sai, e dá a solução. Caso contrário vai para 2.

FLUXOGRAMA

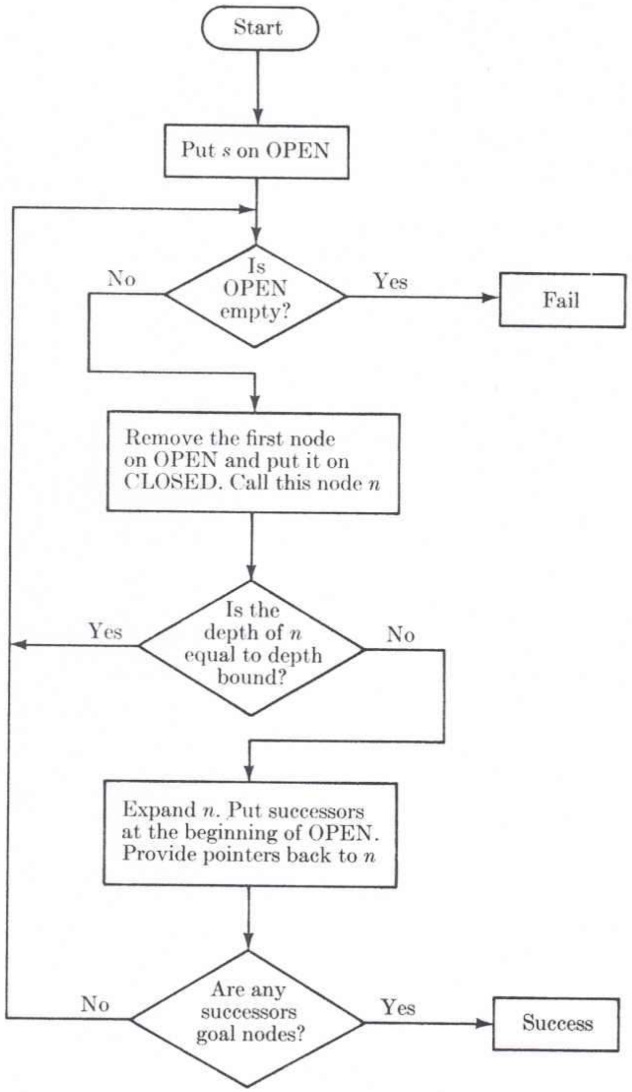


FIG. 3-4 *Flow chart of a depth-first algorithm for trees.*

CONTINUAÇÃO

- DF: quando são gerados sucessores que já estão em ABERTOS ou FECHADOS pode ser necessário recalcular a profundidade dos nós correspondentes.

GRAFOS EM VEZ DE ÁRVORES

- Os métodos anteriores presumem que o espaço de estados tem uma estrutura do tipo árvore.
- Se o espaço de estados for um grafo é preciso modificar os algoritmos:
    - Breadth-first: reconhecer se um estado sucessor já está em ABERTOS ou em FECHADOS e nesse caso não colocar o nó correspondente em ABERTOS.
    - Custo uniforme:
        - Se o sucessor (n_{SUC}) está em ABERTOS
            - n_{SUC} não é adicionado se $g(n_{SUC})>g(n_a)$
            - caso contrário n_{SUC} substitui n_a.
        - Se o sucessor está em FECHADOS ignora-se n_{SUC}.

CONT.

Depth-first

- Se está em abertos um nó com o mesmo estado, isso quer dizer que esse nó tem um custo g menor ou igual ao do nó gerado agora. Abandona o nó gerado.
- Se está em fechados e tem um custo maior ou igual:
  - Elimina o nó antigo e coloca o nó gerado em abertos e coloca pointers nos sucessores do nó antigo para o nó gerado.
- Caso contrário:
  - abandona o nó gerado.

EXPLOSÃO COMBINATÓRIA

- Os métodos BF, Custo uniforme e DF fazem uma procura exaustiva, pelo que se designam por métodos cegos ou não informados.
- Para muitos problemas esta procura exaustiva torna-se pouco prática, não resolvendo o problema da explosão combinatória.
- Necessário usar uma alternativa mais inteligente.

MÉTODOS HEURÍSTICOS OU "INFORMADOS"

- Por vezes é possível usar regras empíricas para acelerar a procura.
  - A ideia central é evitar considerar todas as alternativas, focando a atenção apenas nas que têm mais interesse.
  - Necessário avaliar o "interesse" dos nós: *funções de avaliação*.
  - Estas regras são específicas do problema em causa e nem sempre resultam.
- Métodos que usam este tipo de conhecimento: métodos de procura heurísticos. Também designados: métodos de procura informados.

USO DE FUNÇÕES DE AVALIAÇÃO

- Considera-se que é possível definir uma função de avaliação do interesse dos nós f(n).
- Por convenção a lista de nós ABERTOS é ordenada por ordem crescente de f(n), em que f(n) é o valor da função de avaliação aplicada ao nó n.
- Um algoritmo que use a convenção anterior para fazer procura em espaço de estados cuja estrutura seja do tipo grafo acíclico, consiste na sequência de passos indicada no slide seguinte.

ALGORITMO DE PROCURA ORDENADA

1. Nó inicial(s) => ABERTOS. Faz f(s)=0.
2. Se ABERTOS vazia falha.
3. Remove o nó de ABERTOS (n) com menor custo (f) e coloca-o em FECHADOS
4. Expande o nó n. Calcula o f de cada um dos sucessores.
5. Colocar os sucessores que ainda não existem em ABERTOS nem FECHADOS na lista de ABERTOS, por ordem de f colocando os ponteiros para n.
6. Se algum sucessor for um nó objectivo termina e dá a solução.
7. Associa aos sucessores já em ABERTOS ou FECHADOS o menor dos valores de f (existente ou agora calculado). Coloca em ABERTOS os sucessores que estavam em FECHADOS cujos valores de f baixaram. Redirecciona para n os ponteiros de todos os nós cujos valores de f baixaram.
8. Vai para 2.

ALGORTIMO A*

- O algoritmo anterior não especifica o tipo de função de avaliação. Se esta consistir em **f(n)=g(n)+h(n)** em que g(n) é o custo do nó n e h(n) é o seu valor heurístico, designa-se essa família de algoritmos por A.
- Se se modificar o algoritmo de procura ordenada por forma a que o teste de estado objectivo seja feito sobre o nó n que é seleccionado depois de colocar todos os sucessores em ABERTOS, tem-se o algoritmo A*.

ALGORITMO DE PROCURA ÓPTIMO

- A função de avaliação f'(n)=g(n)+h'(n) dá uma estimativa do custo total do caminho de custo mínimo que passa por n.
- Nota: O algoritmo de custo uniforme encontra a solução ótima (de menor custo).
- h'(n)≡0 => o algoritmo A* coincide com o do custo uniforme, e encontra a solução ótima.
- Pode demonstrar-se que **se h' for um limite inferior de h**, o algoritmo A* continua a encontrar a solução ótima. Ver pp. 60 e seguintes Nils Nilsson.

ADMISSIBILIDADE

- Um algoritmo diz-se admissível se, para qualquer grafo, descobre sempre o caminho ótimo para o objetivo, desde que esse caminho exista.
- Se h' é um limite inferior de h então o algoritmo A* é admissível.
- A admissibilidade implica que:
  - Quando o A* expande um nó n já encontrou um caminho ótimo para n.
  - Quando o A* expande um nó n a função de avaliação f' não é maior que o custo real f.

INFORMAÇÃO HEURÍSTICA

- Usar h'(n)≡0 reflete a ausência total de conhecimento acerca do domínio de aplicação pelo que embora o algoritmo seja admissível é pouco prático.
- Um algoritmo A é mais informado do que um algoritmo B sse $h_A > h_B$ para todos os estados exceto os objetivo.
- Exemplo: 8-puzzle com h=W em que W é o número de peças erradas. Este algoritmo é mais informado do que o do custo uniforme (h'=0).

CONSISTÊNCIA

- Pode demonstrar-se que se a heurística for consistente o A* nunca expande mais nós do que um algoritmo A com informação heurística menor ou igual.
  - Consistente ⇔ $h'(m) - h'(n) \leq h(m) - h(n)$, ou seja: a estimativa do custo do caminho entre dois nós é um limiar inferior (ou igual) do custo real.
    - Esta característica é normalmente verificada desde que a heurística não mude ao longo do processo de procura. Exemplo: h=W é consistente.

EXEMPLO DE APLICAÇÃO

- Considere-se o seguinte problema do 8-puzzle:

2	1	6
4		8
7	5	3

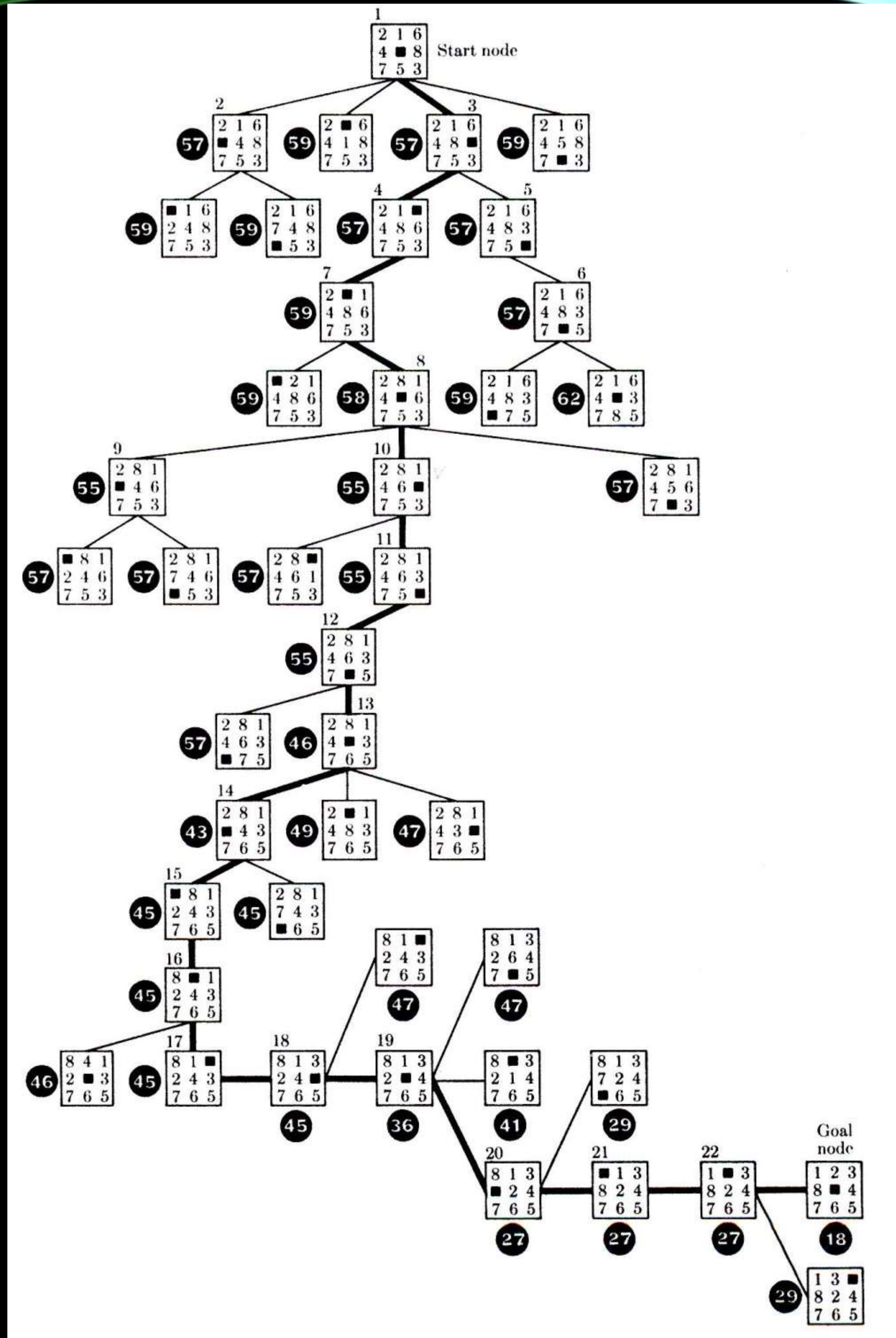
$\longrightarrow$

1	2	3
8		4
7	6	5

- Considere-se a função heurística **h'=P+3S** em que
  - P é o somatório das distâncias de cada peça à sua casa certa assumindo que não há obstáculos,
  - S é o somatório de um score para cada peça em, ao circular em torno da casa central, se soma 2 se a peça não estiver seguida do seu sucessor correcto e 0 caso contrário; soma-se 1 por uma peça errada na casa central.

VERIFICAR O GRAFO SEGUINTE

- Os valores em bolas pretas representam a função de avaliação f' = g + h'
- Os números fora das bolas representam a ordem de expansão dos nós do grafo
- A heurística usada é admissível?
- É garantido que se encontra a solução de menor custo(ótima)?



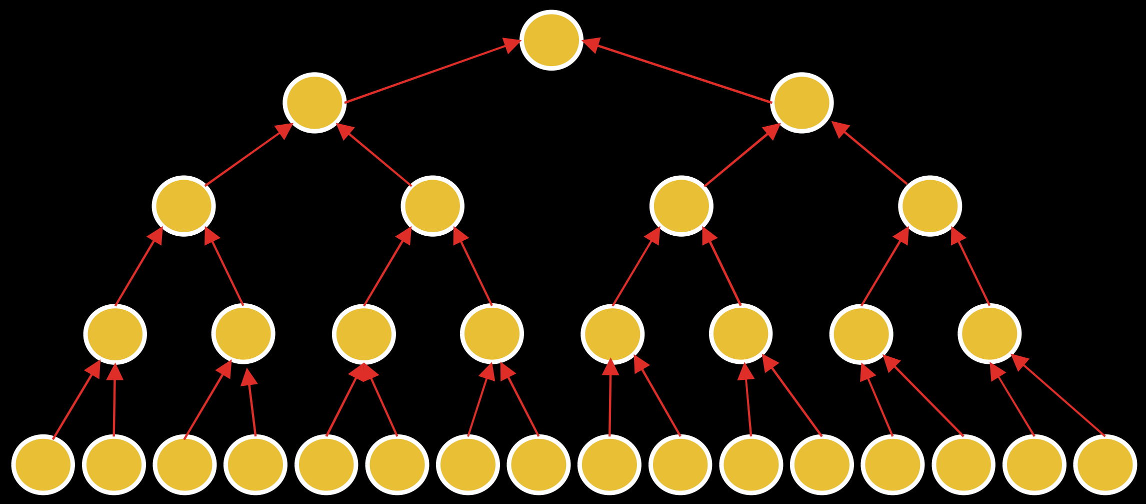
(Nils Nilsson, Problem Solving Methods in AI, p.67)

COMPARAÇÃO COM BF

- No grafo anterior, o A* gerou um total de 43 nós e a solução tem 18 níveis.
- Quantos nós geraria o algoritmo breadth-first para encontrar a mesma solução?
  - Pode assumir-se um fator de ramificação média B=2.
  - Com esse valor de B o número de nós em cada nível é B^n.
  - O valor esperado (média) para o número total de nós gerados pelo BF é dado por $T_m=(MIN+MAX)/2$ em que MIN é o número mínimo de nós gerados e MAX é o número máximo de nós gerados.
  - $MIN = 2^1+ 2^2+\ldots+ 2^{17}+2$
  - $MAX = 2^1+ 2^2+\ldots+ 2^{18}$
  - $T_m = [2*(2^1+ 2^2+\ldots+ 2^{17})+(2+2^{18})]/2$
- O BF apresentaria um grafo com mais de 150,000 nós, em vez dos 43 apresentados no grafo anterior pelo A* com a heurística P+3S.

COMPARAÇÃO DO BF COM O DF

- Pode interessar também comparar os dois algoritmos não informados, anteriormente estudados (breadth-first e depth-first). Para facilitar os cálculos considera-se que a solução existe no nível L=4 e que se tem um factor de ramificação B=2.



1 … $2^1 = 2$

2 … $2^2 = 4$

3 … $2^3 = 8$

4 … $2^4 = 16$

BF VS. DF (COM D=L)

- Aplicando, no caso do BF, o método descrito no slide anterior, obtém-se os seguintes valores:
  - MIN = 2 + 4 + 8 + 2 = 16
  - MAX= 2 + 4 + 8 + 16 = 30
  - T_m = (16 + 30)/2 = 23

- De forma similar, para o DF, se d=4, tem-se:
  - MIN = 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 = 30
  - T_m = (8 + 30)/2 = 19

- Aparentemente o DF é melhor que o BF.

DF (COM D>L)

- Se agora o nível de profundidade máxima, d, for superior a L, a relação BF-DF altera-se.
- Considere d=5:
  - MIN = 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 + 32 – 2 = 60
  - $T_m = (8 + 60)/2 = 34$
- Considere d=6:
  - MIN = 2 + 2 + 2 + 2 = 8
  - MAX= 2 + 4 + 8 + 16 + 32 + 64 – 2 – 4 = 120
  - $T_m = (8 + 120)/2 = 64$
- Em ambas as situações o DF é pior que o BF.

ALGORITMO DE DIJKSTRA

Este algoritmo soluciona o problema do caminho mais curto.

PSEUDOCÓDIGO:

1. Atribuir valor zero à estimativa do custo mínimo do nó **s** (a raiz da árvore) e infinito às estimativas para todos os outros nós do grafo;
2. Em cada passo, encontrar o nó **u**, que ainda não foi processado, que possua a menor distância a **s**.
3. Ver para cada nó **v**, vizinho de **u**, se é melhor manter a distância atual de **v** ou atualizar fazendo o caminho S→u e depois u→v. De notar que o caminho S→u já foi fixado e possivelmente tem conexões no meio.

Edsger Dijkstra
1956

COMPARAÇÃO COM OUTROS ALGORITMOS DE PROCURA EM ESPAÇO DE ESTADOS

- Se os arcos tiverem valor unitário, o algoritmo de Dijkstra é igual ao BF.

- Se os arcos do grafo tiverem pesos arbitrários positivos este algoritmo determina o caminho de custo mínimo. Nesse caso, o algoritmo de Dijkstra é igual ao algoritmo de custo uniforme.

- O algoritmo de Dijkstra é um caso especial do A* em que a heurística é zero.

MEDIDAS DE DESEMPENHO

- Há certas medidas que embora não determinem completamente o poder heurístico podem ser úteis para comparar várias técnicas de procura.
  - Penetrância: $P = \frac{L}{T}$

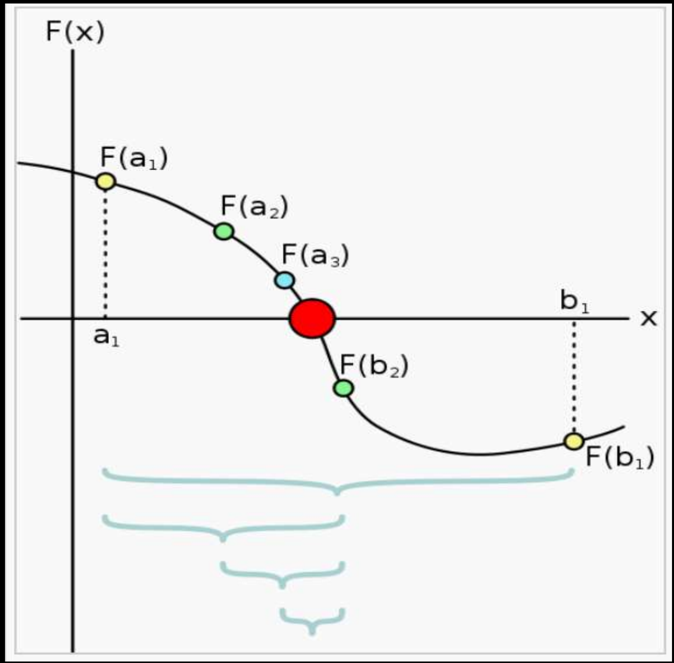
    em que L é o comprimento do caminho até ao objetivo e T é o número total de nós gerados

  - Factor de ramificação média

$$B + B^2 + \ldots + B^L = T \quad \text{ou} \quad \frac{B}{B-1}(B^L - 1) = T$$

RESOLUÇÃO DE EQUAÇÕES DE ORDEM SUPERIOR

- Método da bissecção



- Método de Newton-Raphson

http://faculty.washington.edu/dbp/SAPACLISP-1.x/basic-math.lisp

ALGORITMOS DE PROCURA EM ESPAÇO DE ESTADOS

Procura com Memória Limitada

IDA*
ITERATIVE DEEPENING A*

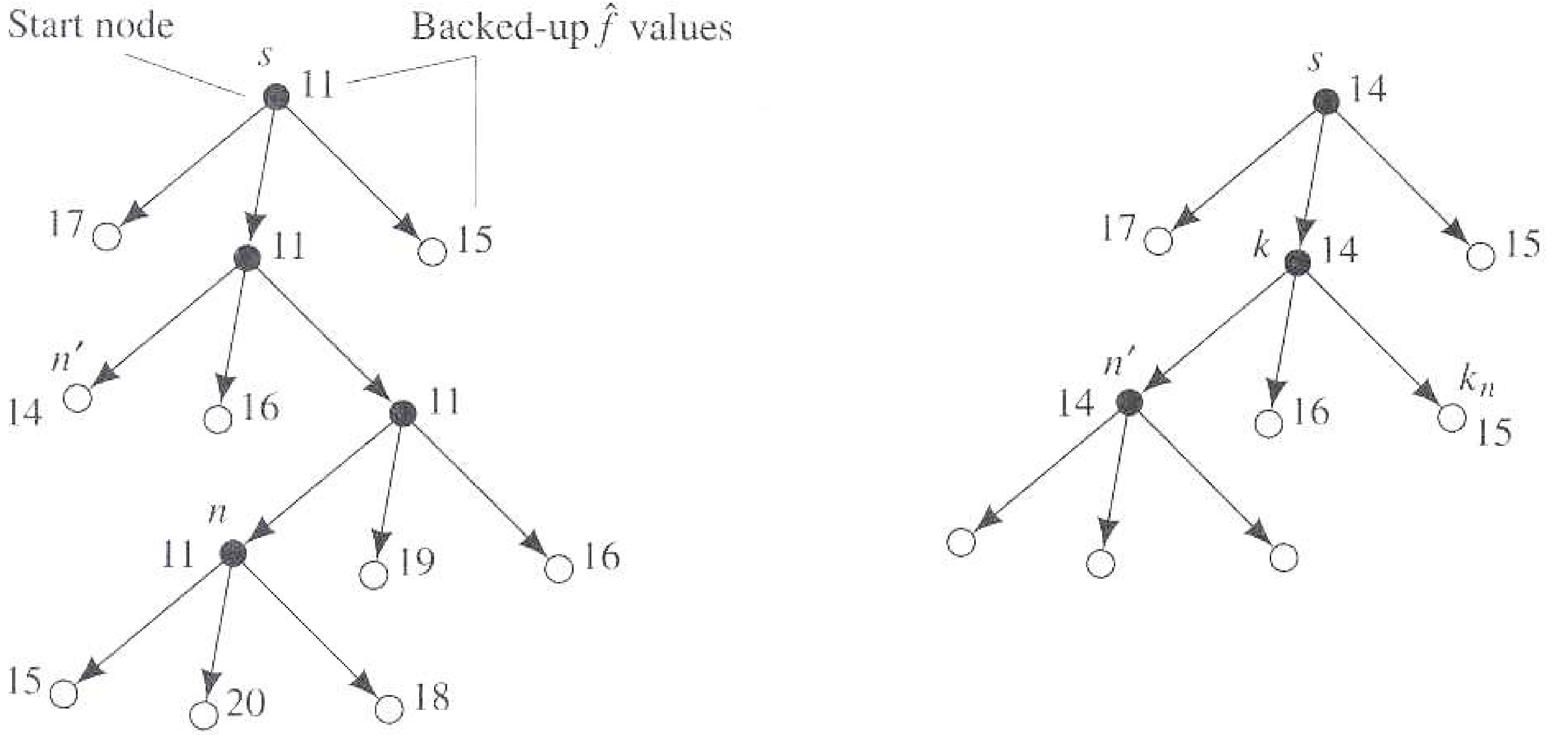
- Os requisitos de memória dos métodos não informados aumentam exponencialmente com a profundidade do estado objectivo no espaço de procura.
- A utilização de heurísticas não evita este problema, apesar de reduzir o factor de ramificação.
- IDA* surge em 1985 (Korf)
- Pode ser objecto de implementação paralela (Powley, Ferguson e Korf 1993)
- O IDA* garante a descoberta da solução óptima, desde que se use uma heurística admissível.

ALGORITMO IDA*

- Aplica-se uma série de vezes o método de procura em profundidade, com limiares de profundidade variáveis, mas em que o limite ("cost cutt-off") é dado em termos do f.
- Na primeira pesquisa o limiar L é dado por f'(n0) = g(n0) + h'(n0) = h'(n0), em que n0 é o nó inicial.
- O custo do caminho óptimo pode ser igual ao limiar mas não maior dado que a heurística tem de ser admissível, i.e. h(n0) >= h'(n0)
- Só se expandem nós com f'(n)<=L
- Se a solução não for encontrada passa-se a usar um novo limiar L1 tal que L1=min(F(n)) em que F(n) é o conjunto de nós visitados mas que ainda não foram expandidos.

DESEMPENHO

- Dado que o IDA* é tipo procura-em-profundidade apenas necessita de guardar em memória um número de nós igual ao maior ramo explorado.
- Num problema em que os valores de f' são diferentes para todos os nós de um espaço de estados o número de iterações pode ser igual ao número de nós com f' menor que o custo do caminho óptimo.
- Neste caso, sendo a solução dada por uma sequência de N nós o A* explora O(N) enquanto o IDA* requer 1+2+...+N ou seja $O(N^2)$.



(a) RBFS has just expanded node n but has not yet backed up the $\hat{f}$ values of its successors

(b) $\hat{f}$ values have been backed up, the subtree below k_n has been discarded, and search continues below n'

Figure 9.9

Recursive Best-First Search

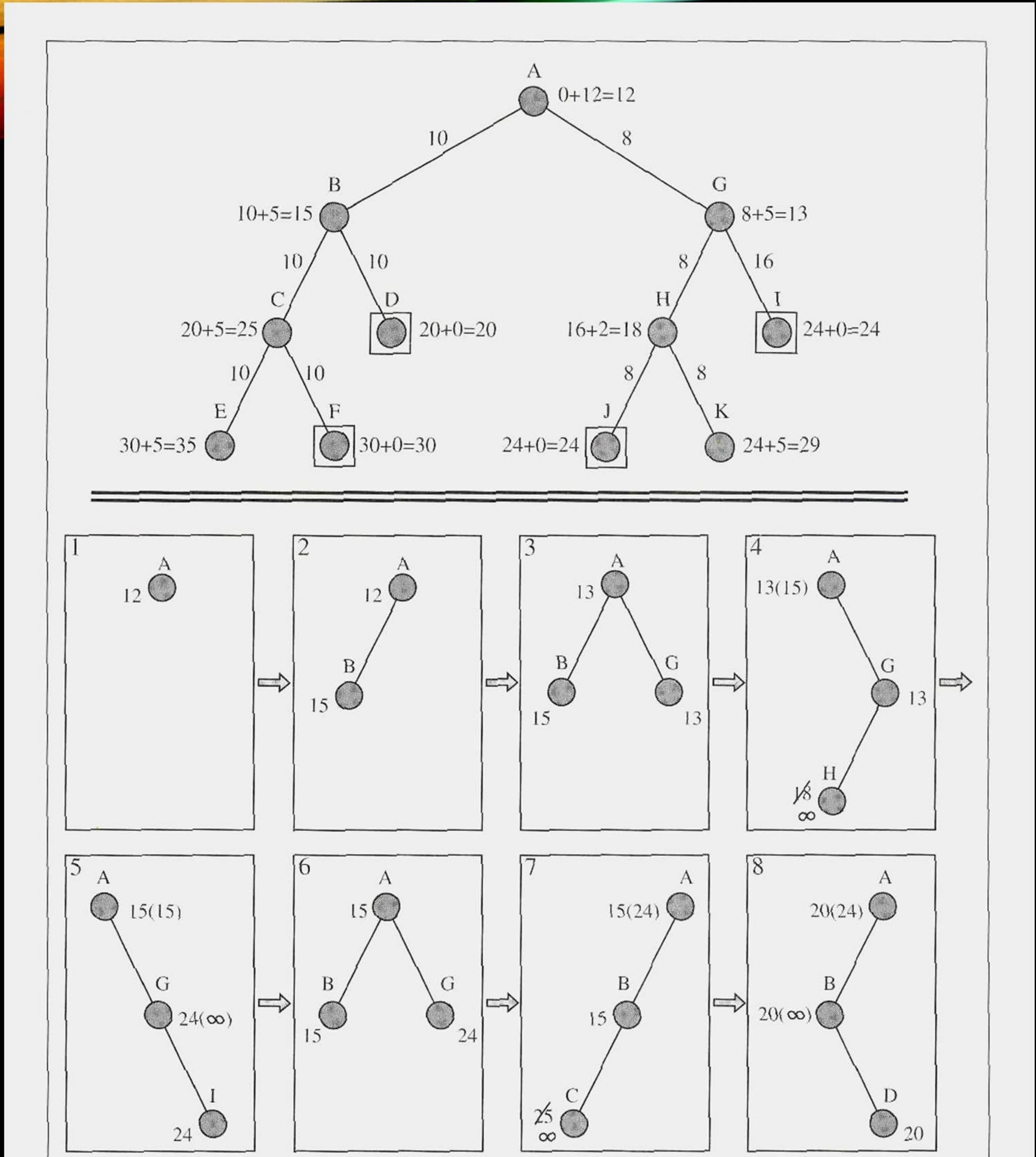
RBFS - RECURSIVE BEST-FIRST SEARCH

- É uma variante do IDA* em que
  - os valores f’ dos sucessores de um nó n são calculados e
  - de acordo com isso se recalculam os valores de f’ do nó n e de todos os seus antecessores (backup do valor f’ ótimo)
- O valor recalculado do nó n com sucessores n_i é

  $f'(n) = \min[n_i]\, f'(n_i)$
- Se um dos sucessores do nó n, n_i, tem o valor de f’ mais baixo de ABERTOS então esse é o próximo nó expandido.
- Se um outro nó de ABERTOS, m, tem o valor de f’ mais baixo, então o algoritmo elimina todos os nós até ao antecessor comum, exceto os seus sucessores directos, continuando a procurar a partir do nó m.

SMA* SIMPLIFIED MEMORY-BOUNDED A*

- O SMA* pode usar toda a memória disponível para realizar a procura, o que aumenta a eficiência.
  - Evita repetir estados, tanto quanto a memória lho permite
  - É completo se a memória for suficiente para guardar o caminho mais curto para a solução.
  - É óptimo se a memória for suficiente para guardar o caminho de menor custo (óptimo) para a solução.
  - Se tiver memória suficiente para guardar toda a árvore de estados a procura é equivalente ao A*.


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
D
H
I
20+5=25
20+0=20
16+2=18
24+0=24
10
10
8
8
E
F
J
K
30+5=35
30+0=30
24+0=24
24+5=29
1
A
12
2
A
12
B
15
3
A
13
B
G
15
13
4
A
13(15)
G
13
H
18
∞
5
A
15(15)
G
24(∞)
I
24
6
A
15
B
G
15
24
7
A
15(24)
B
15
C
25
∞
8
A
20(24)
B
20(∞)
D
20

ALGORITMO

- A ideia base do SMA* é a de que quando é necessário gerar um sucessor mas não há memória disponível é preciso abrir espaço, "esquecendo" um dos nós anteriormente gerados.
- Regras:
    - O SMA* prefere esquecer o nó com f' mais elevado. Em caso de empate retira o de nível mais baixo.
    - Antes de remover uma sub-árvore, guarda no nó antecessor dessa sub-árvore informação acerca da qualidade do melhor caminho na sub-árvore esquecida.
    - Quando se gera um sucessor, o valor deste é propagado para cima (back-up), mediante a regra do mínimo, à semelhança do RBFS.
    - Um nó cuja profundidade N seja igual ao limite de memória disponível (em termos de número de nós) é imediatamente valorizado com $f=\infty$

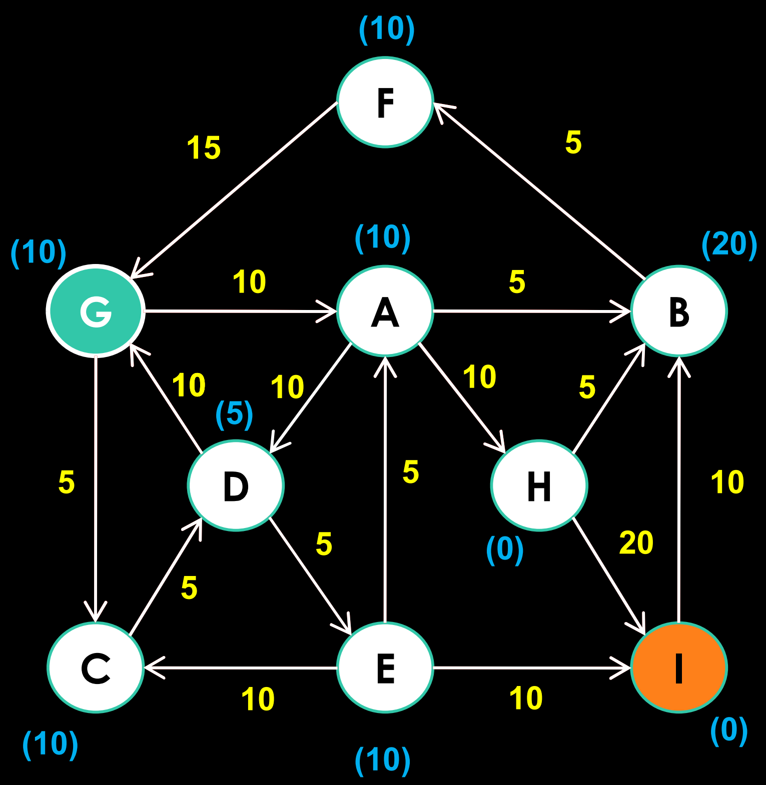
APRECIAÇÃO FINAL SOBRE O SMA*

- O SMA* pode ser melhor que o A* em problemas com espaços de estados fortemente conectados.
- O SMA* pode ser incapaz de resolver problemas que o A* consegue resolver se for necessário gerar repetidamente as mesmas sub-árvores ao oscilar entre caminhos candidatos.

COMPARAÇÃO DE ALGORITMOS

- Eficácia:
  - Chegar a uma solução
    - Solução ótima.
- Eficiência:
  - Usar os recursos de forma mais económica
    - Espaço mínimo
    - Tempo mínimo

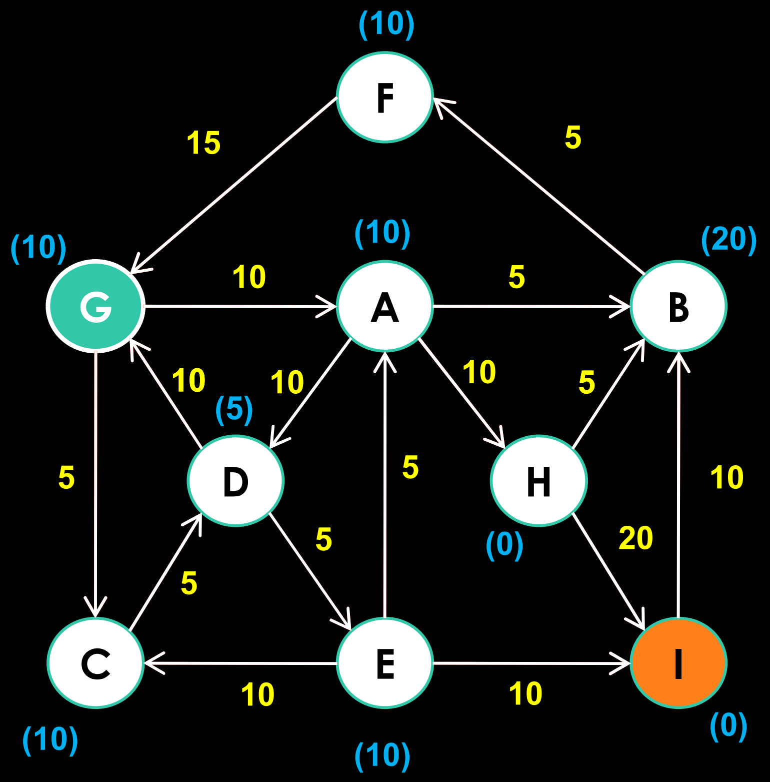
ANÁLISE COMPARATIVA SIMULAÇÃO

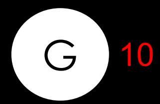


- A*
- IDA* - Iterative Deepening A*
- RBFS – Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

Este slide e seguintes, referentes à simulação, são baseados em exemplo fornecido pelo Prof. Cédric Grueau

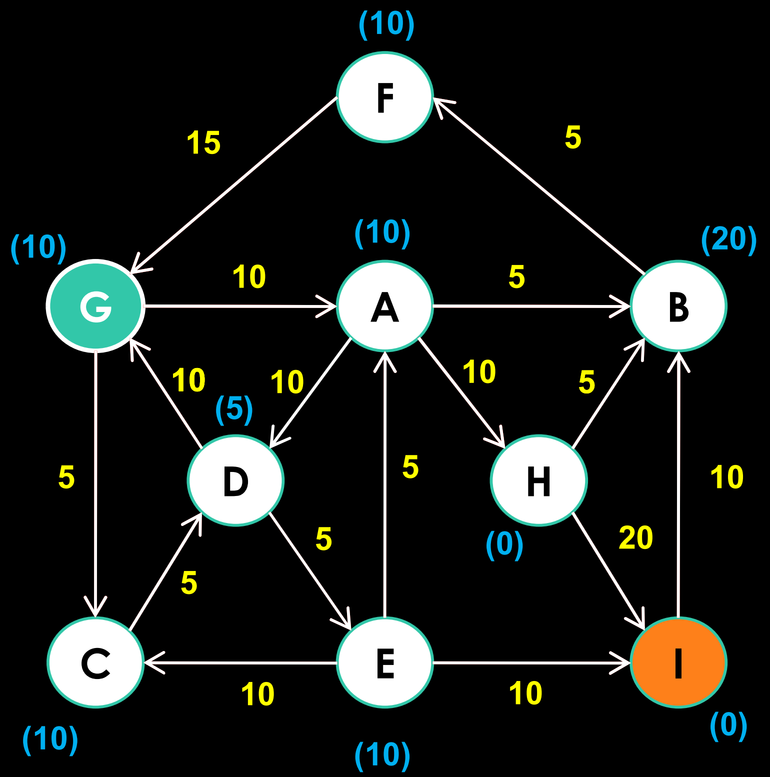
ALGORITMO A*

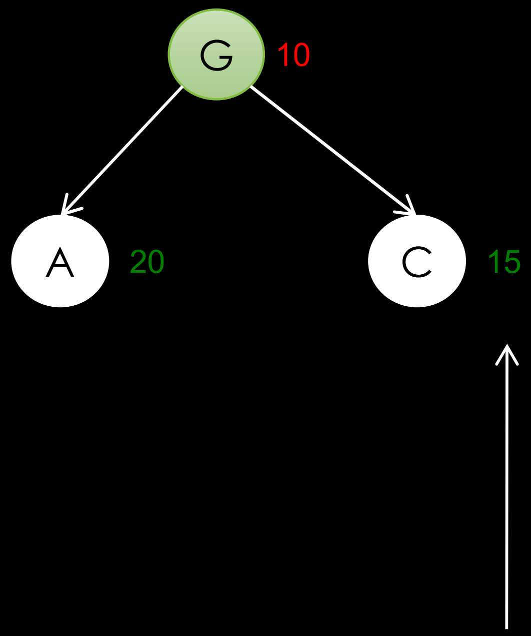




Nota: os valores $h(n)$ são identificados a azul entre parêntesis

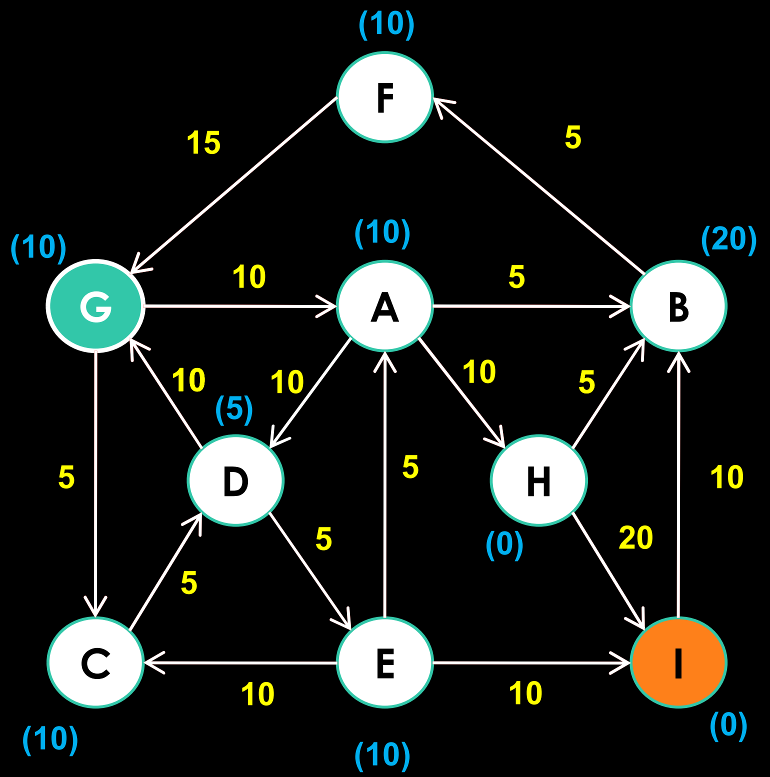
ALGORITMO A*

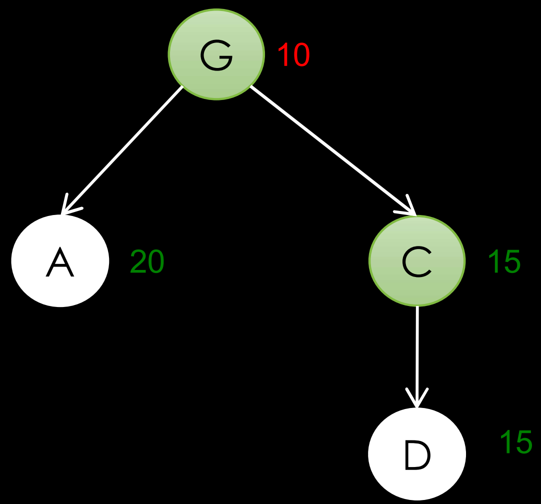




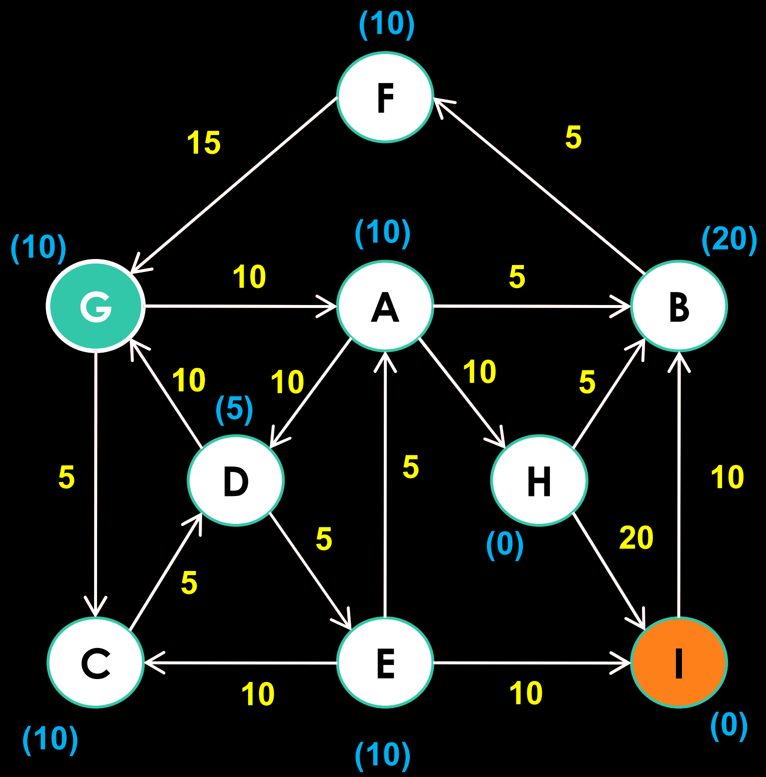
Nota: os valores *f(n)* são identificados em verde

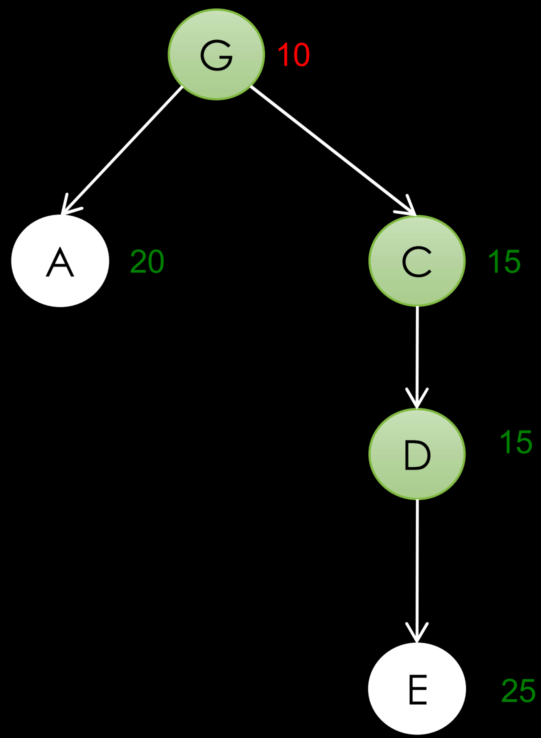
ALGORITMO A*


(10)
F
15
5
(10)
(10)
(20)
G
10
A
5
B
10
10
10
5
(5)
D
5
5
H
10
5
(0)
20
5
C
E
I
10
10
(10)
(10)
(0)

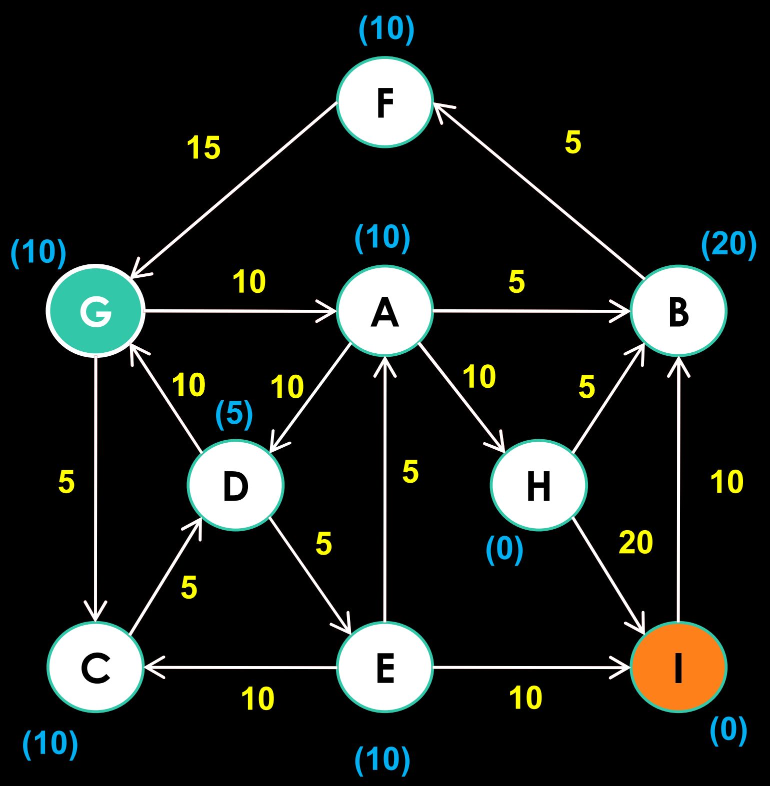

G
10
A
20
C
15
D
15

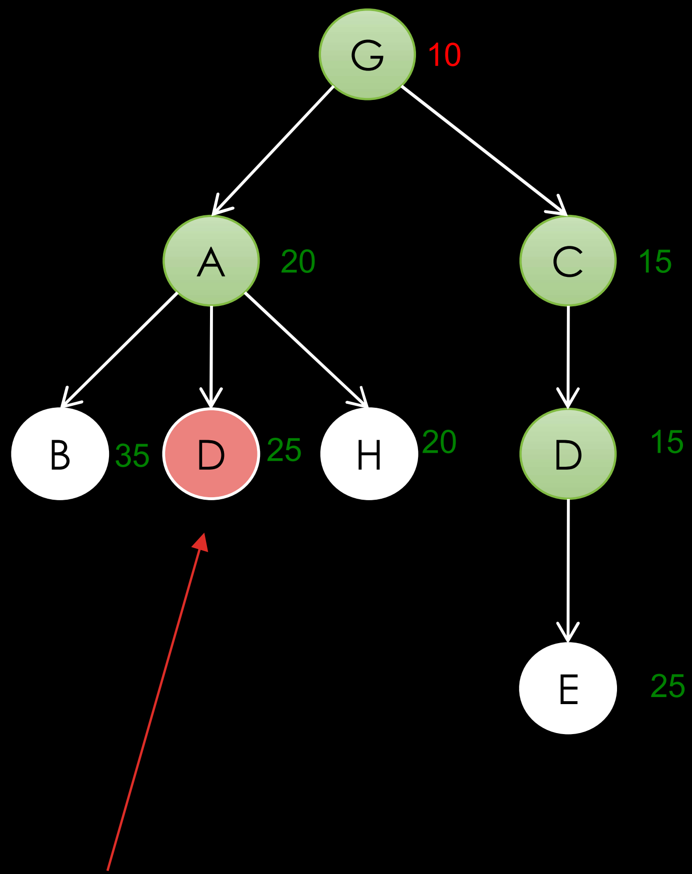
ALGORITMO A*


(10)
F
15
5
(10)
G
10
(10)
A
5
(20)
B
10
10
10
5
(5)
D
5
5
H
10
(0)
20
5
5
C
10
E
10
I
(10)
(10)
(0)


G 10
A 20
C 15
D 15
E 25

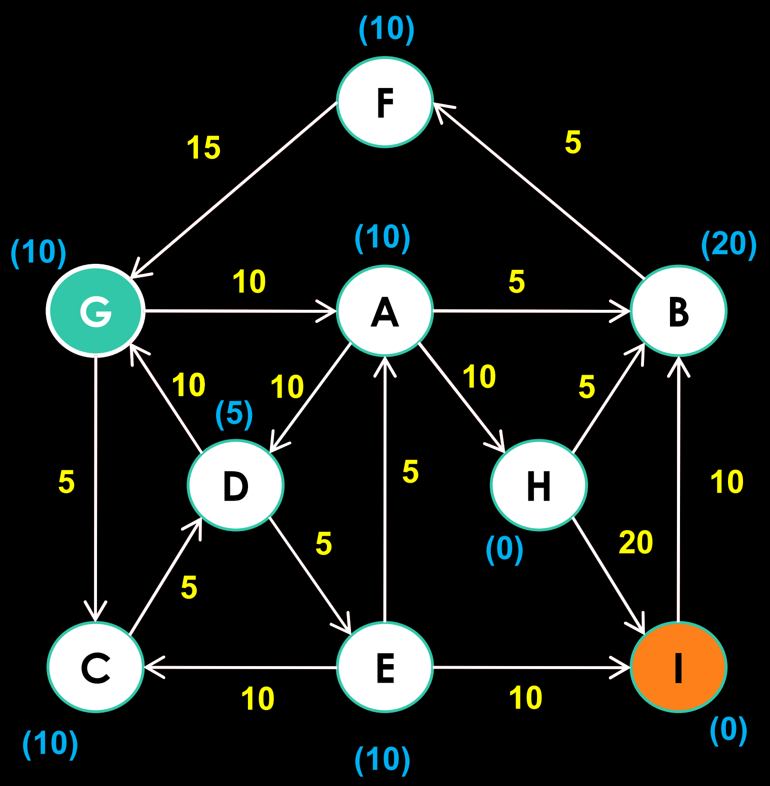
ALGORITMO A*

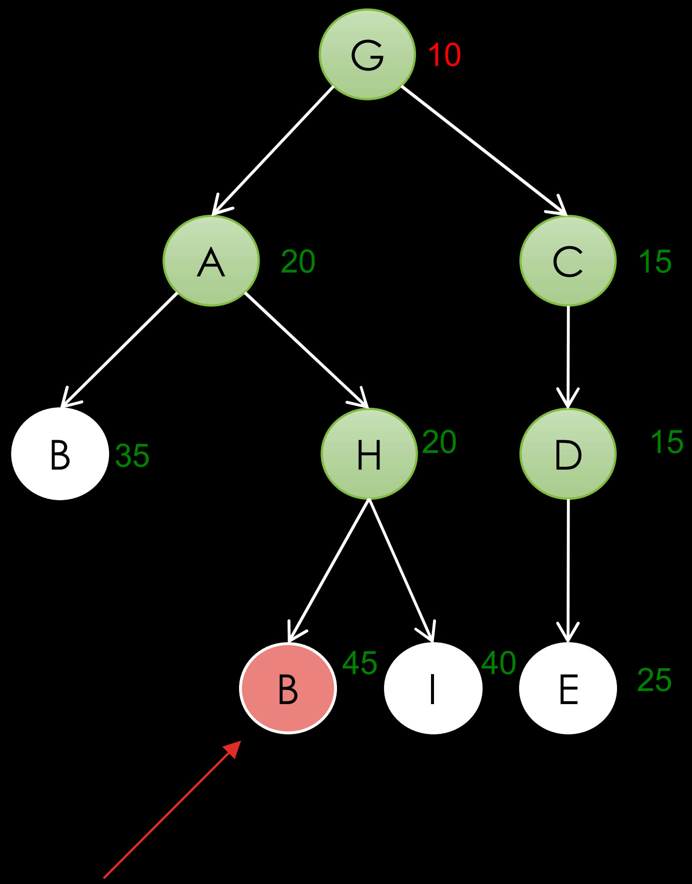




Nota: Não se coloca o nó D em abertos porque já está em fechados e tem menor custo

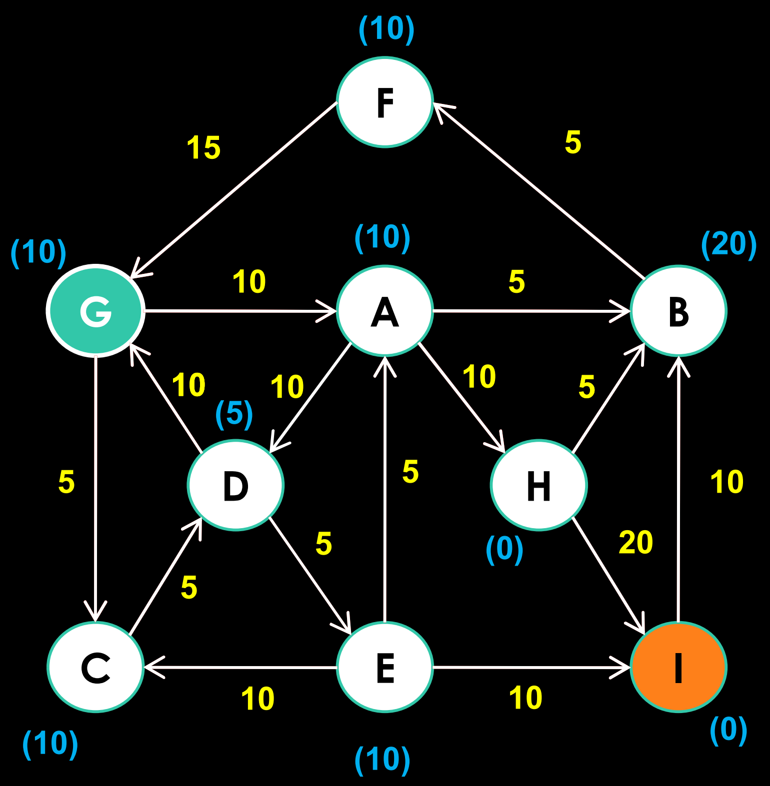
ALGORITMO A*

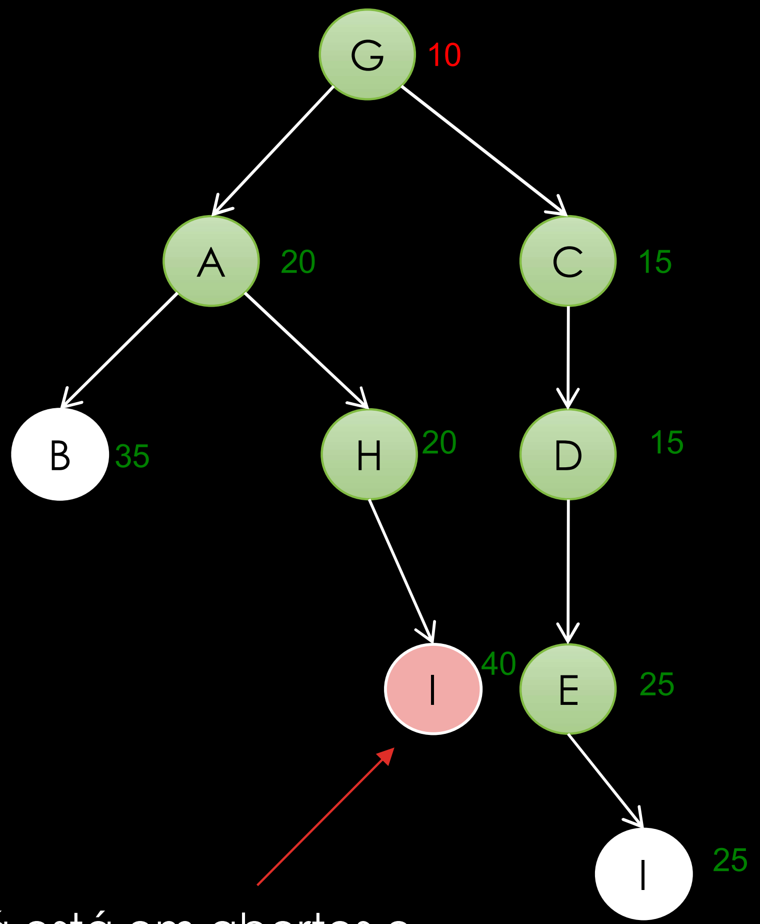




Nota: Não se coloca o nó B em abertos porque já está em abertos e tem menor custo

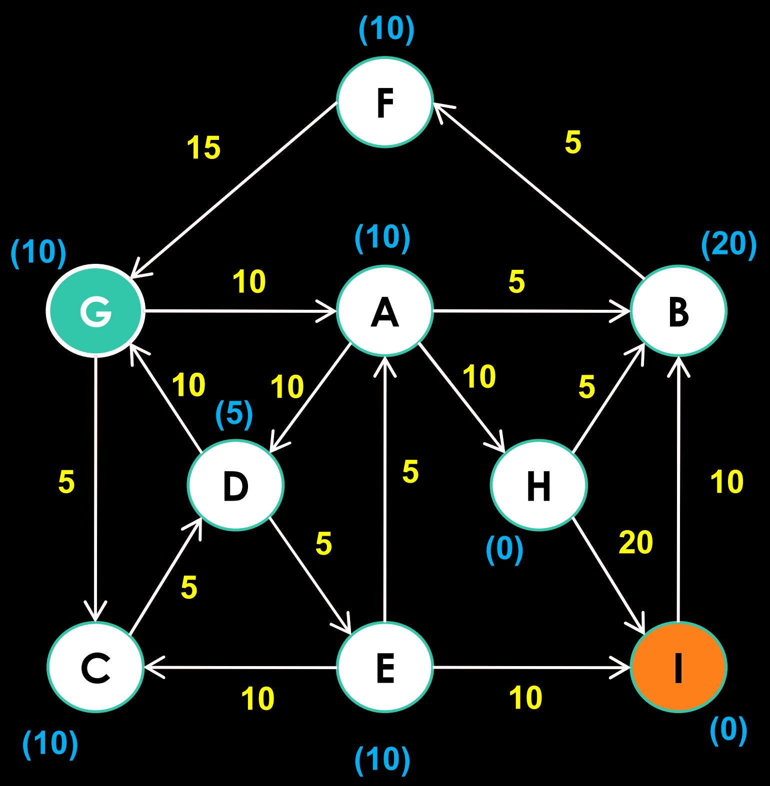
ALGORITMO A*



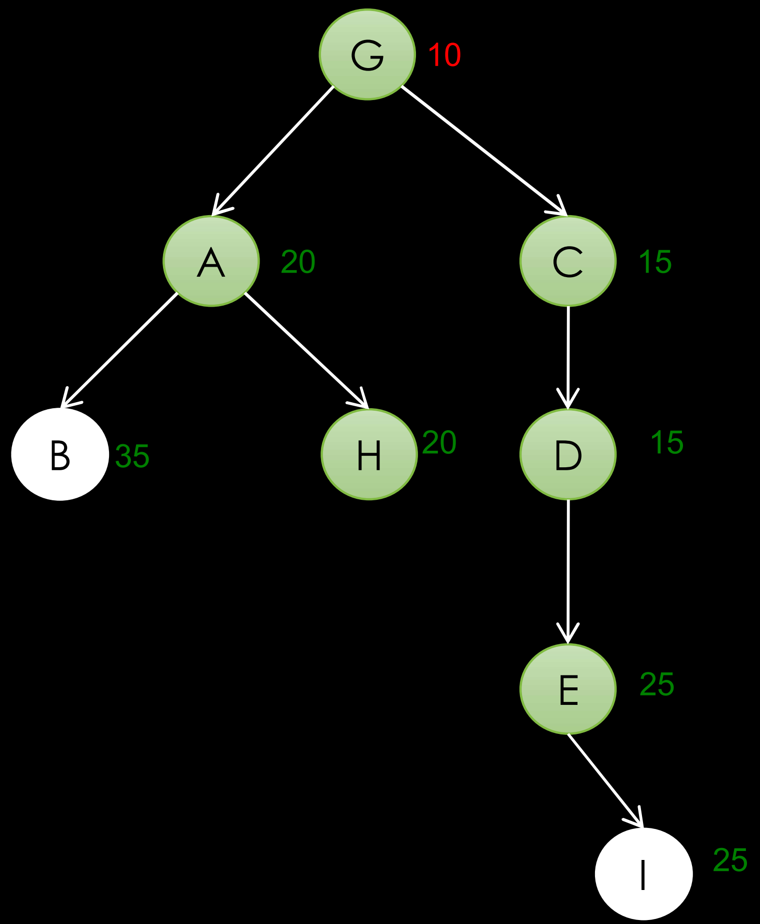


Nota: O nó I já está em abertos e tem maior custo. Neste caso é removido o nó antigo e conserva-se o de menor custo

ALGORITMO A*

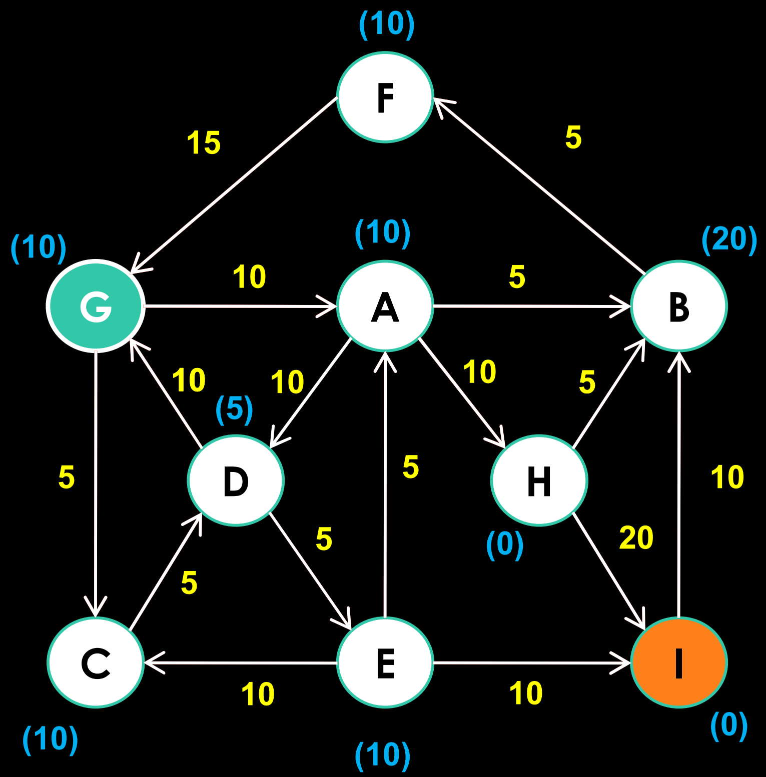


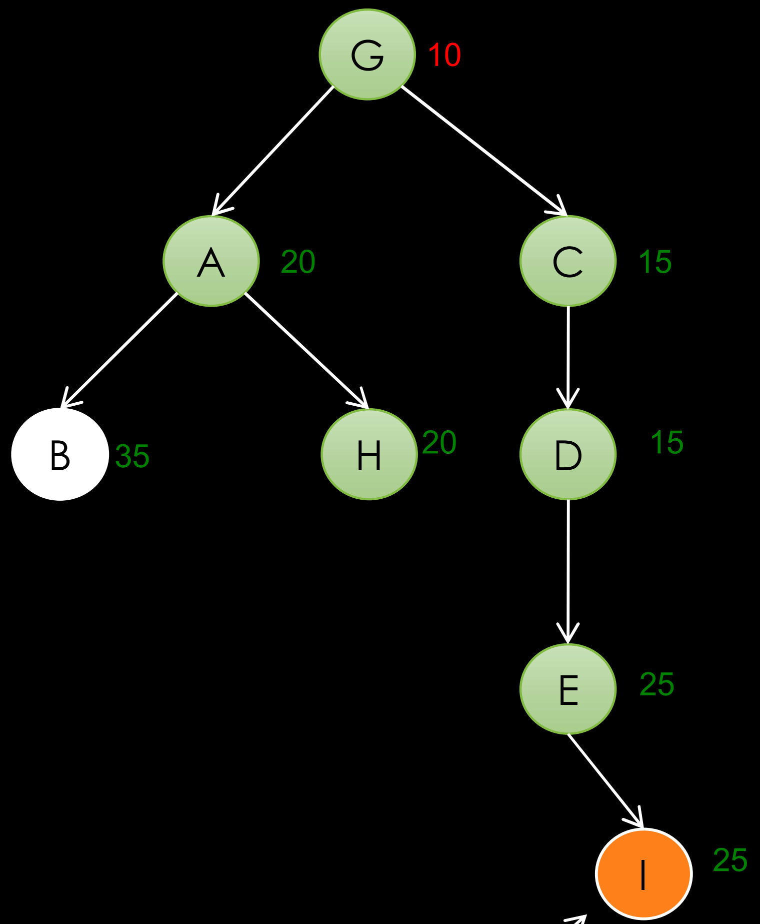
(10)
F
15
5
(10)
G
10
(10)
A
5
(20)
B
10
10
(5)
10
5
D
H
5
5
10
5
(0)
20
5
C
E
I
10
10
(10)
(10)
(0)



G
10
A
20
C
15
B
35
H
20
D
15
E
25
I
25

ALGORITMO A*



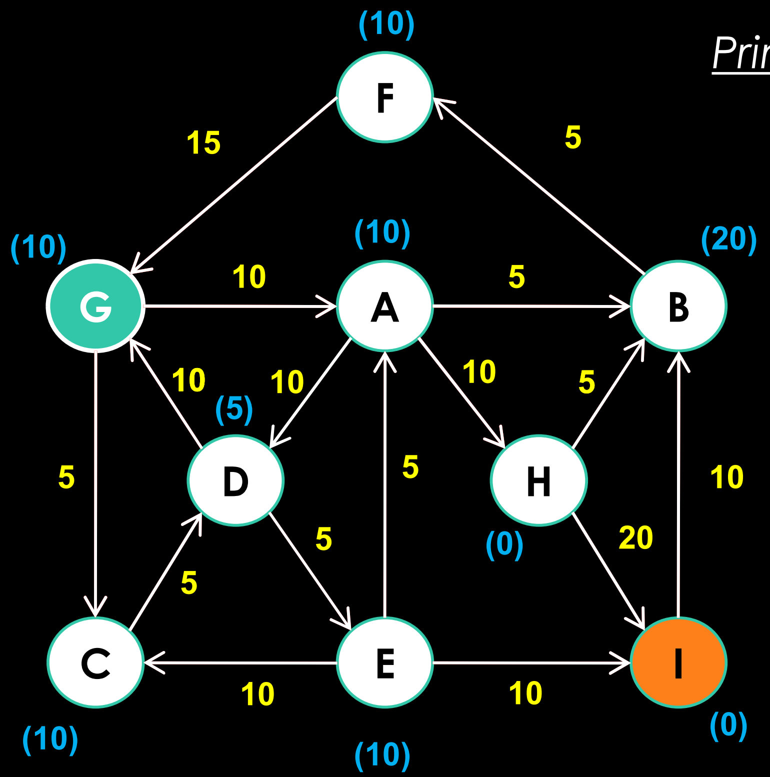


Pára e dá a solução: G, C, D, E, I

ANÁLISE COMPARATIVA SIMULAÇÃO

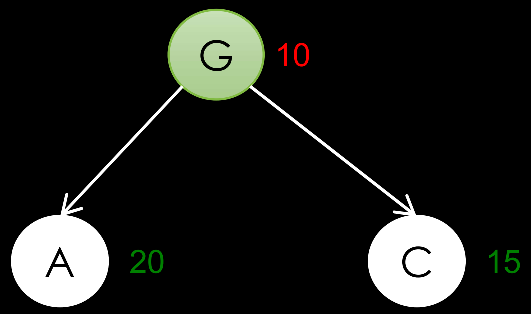
- A*
- IDA* - Iterative Deepening A*
- RBFS – Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

ALGORITMO IDA*

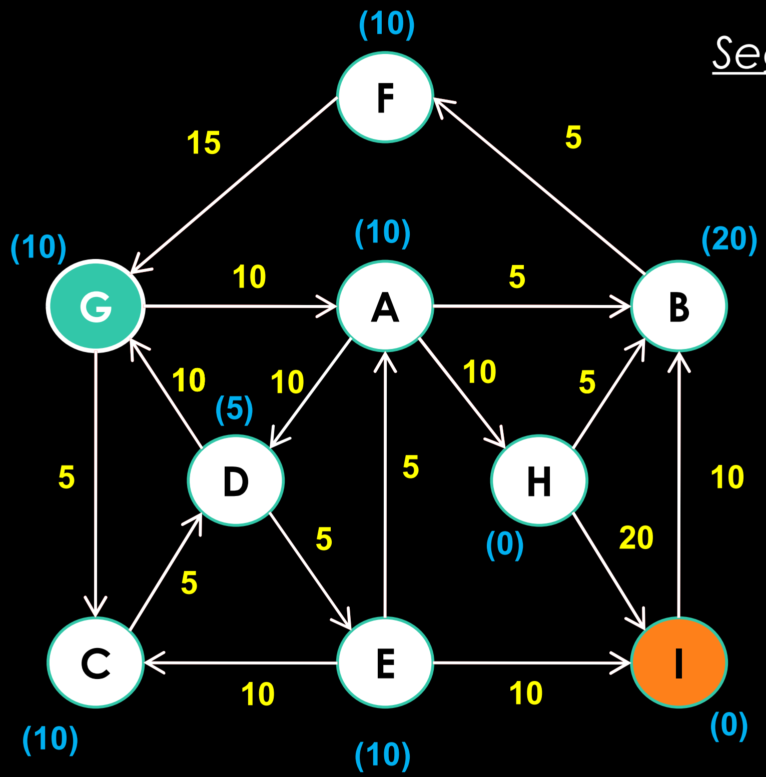


Primeira iteração:

limite = 10

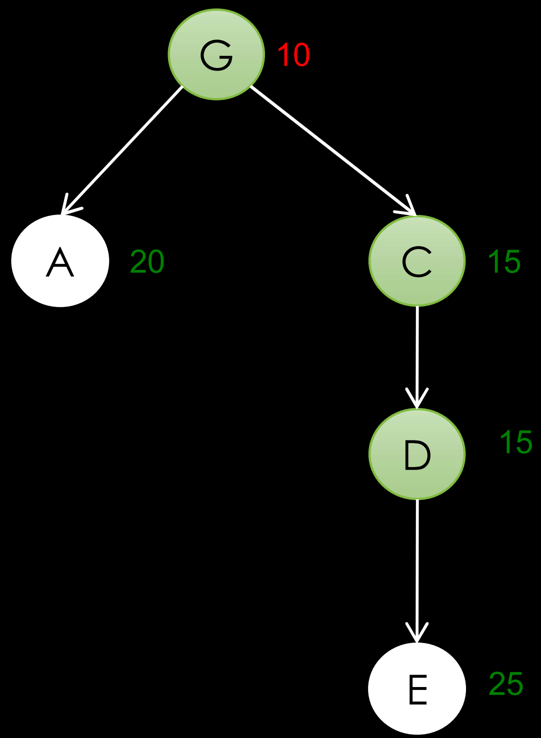


ALGORITMO IDA*

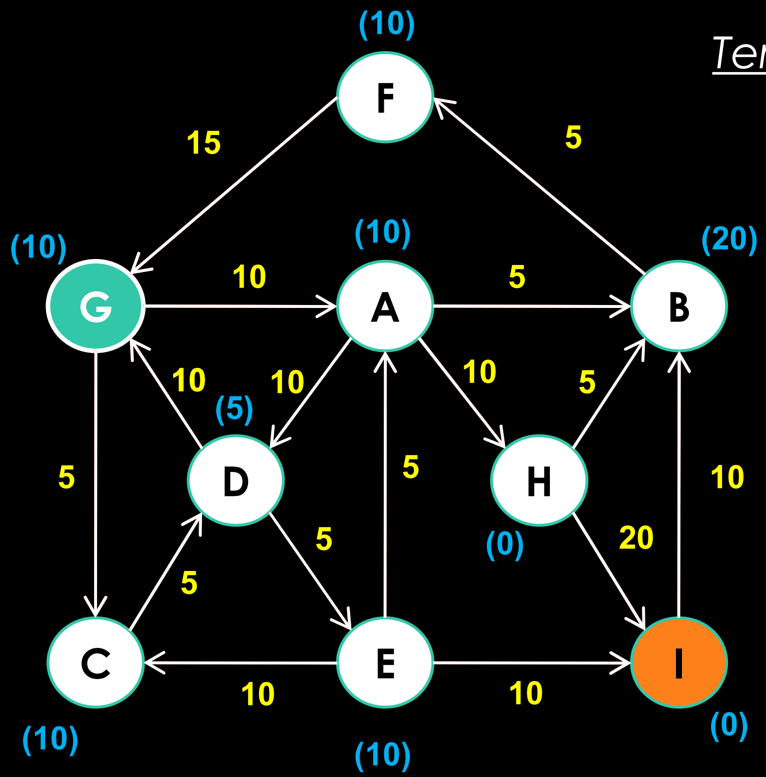


<u>*Segunda iteração*</u>:

limite = 15

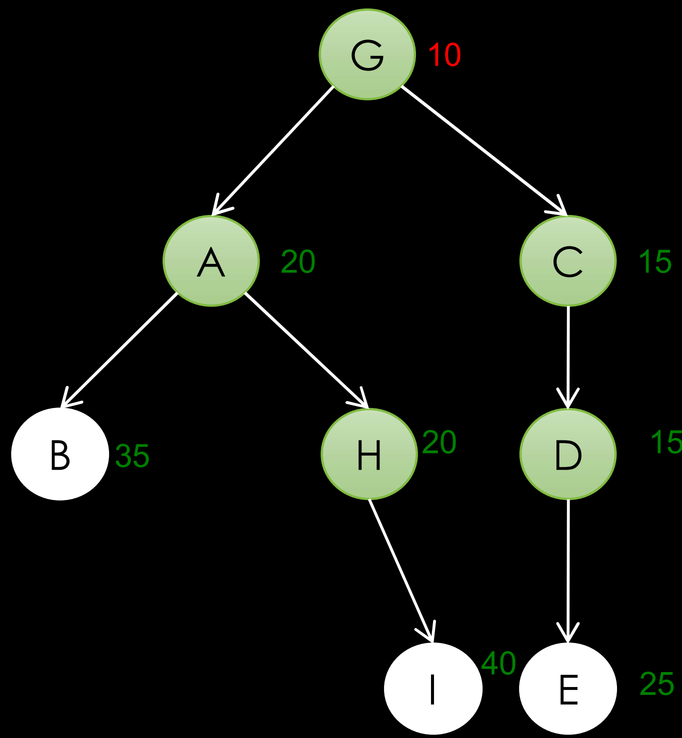


ALGORITMO IDA*

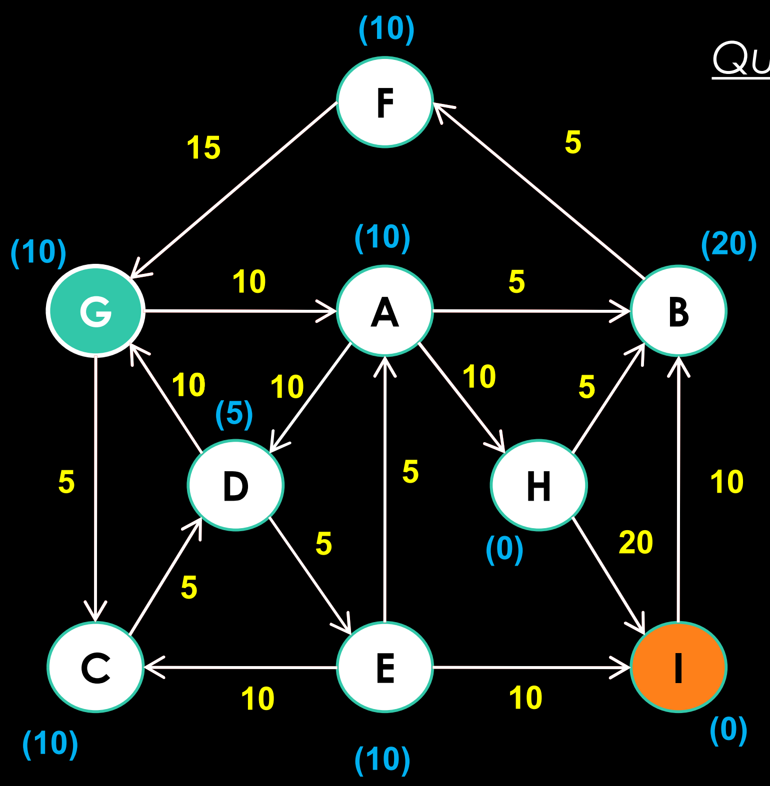


Terceira iteração:

limite = 20

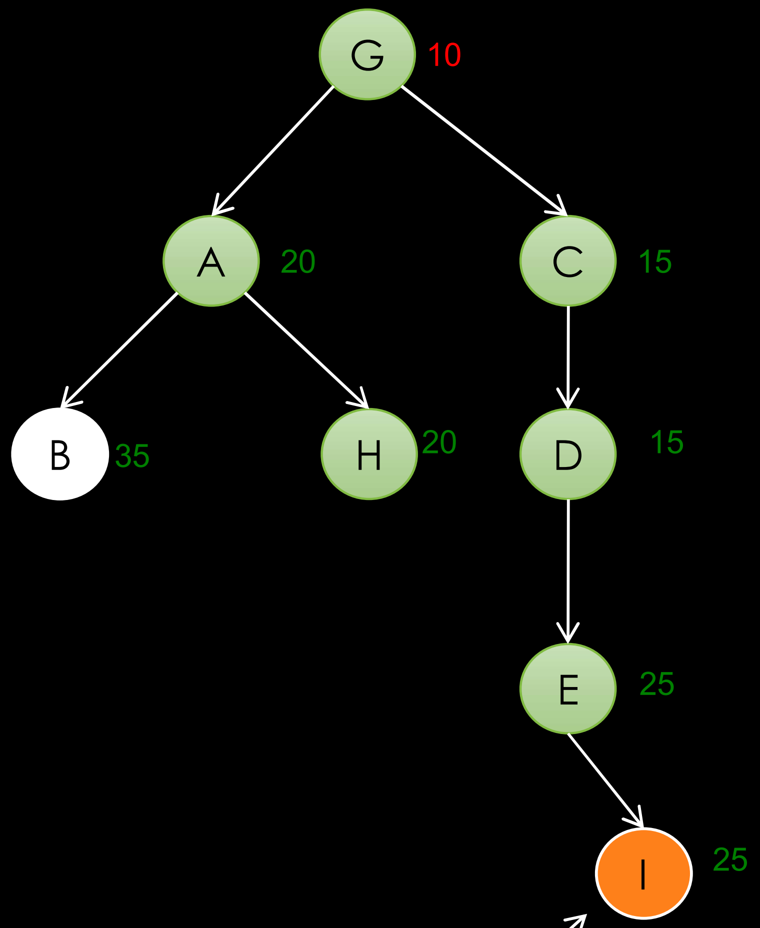


ALGORITMO IDA*



Quarta iteração:

limite = 25

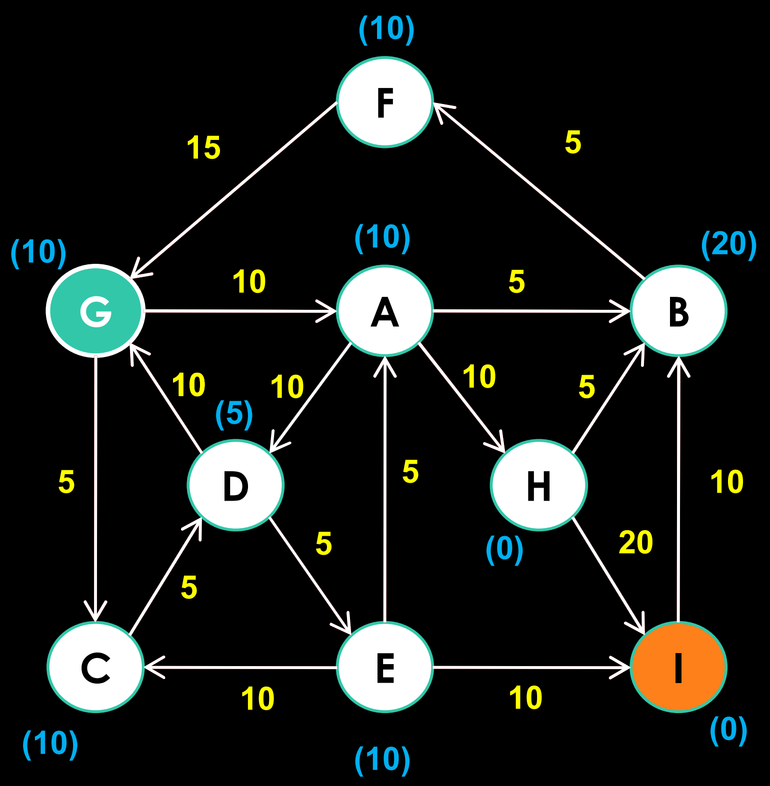


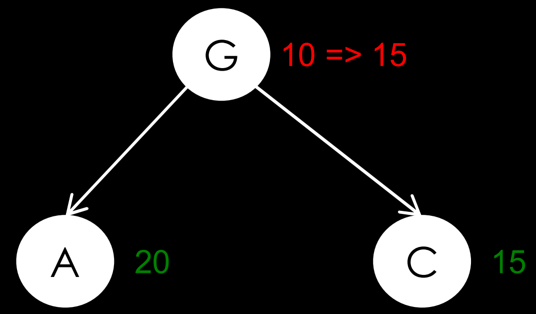
Pára e dá a solução: G, C, D, E, I

ANÁLISE COMPARATIVA SIMULAÇÃO

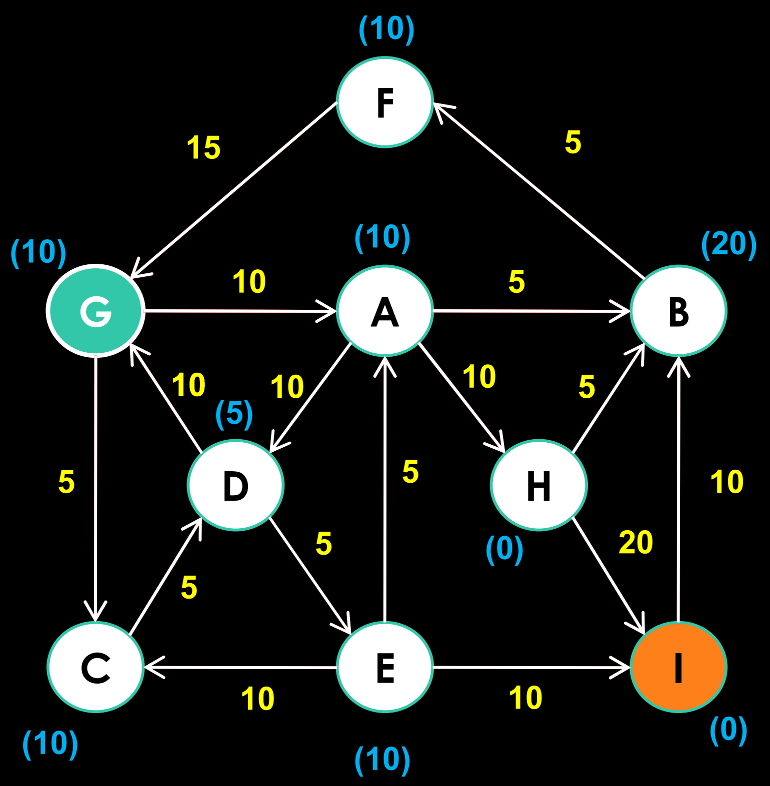
- A*
- IDA* - Iterative Deepening A*
- RBFS – Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

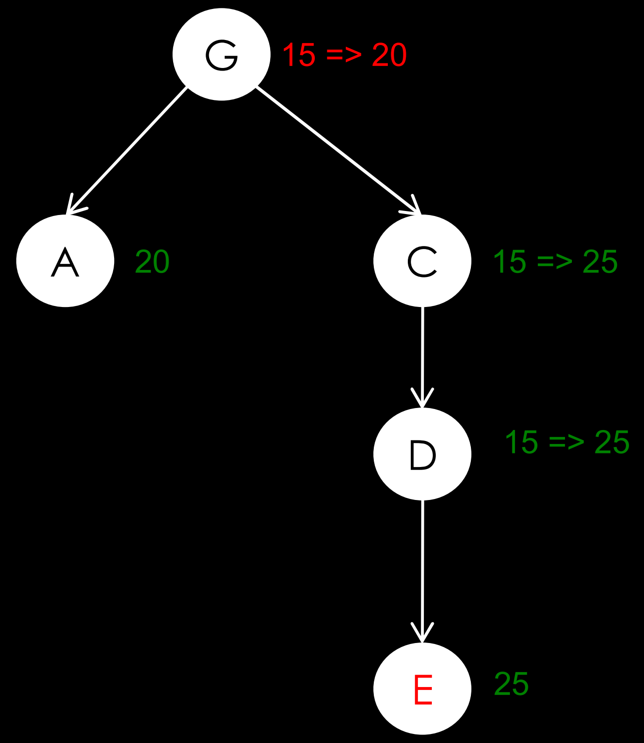
ALGORITMO RBFS



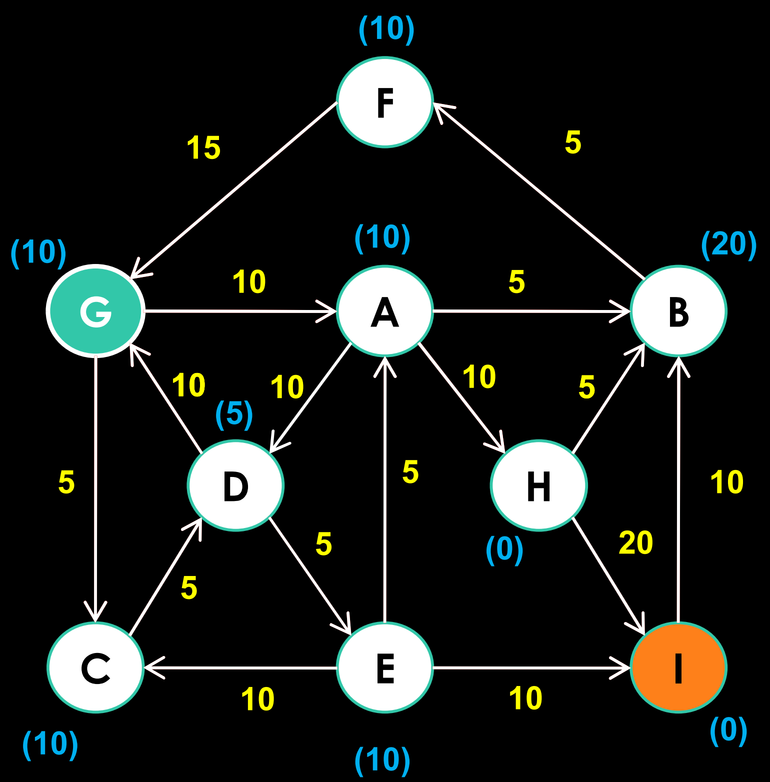


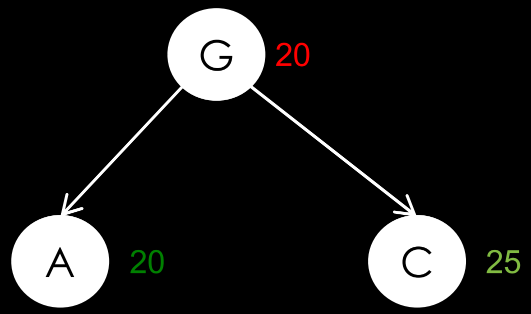
ALGORITMO RBFS



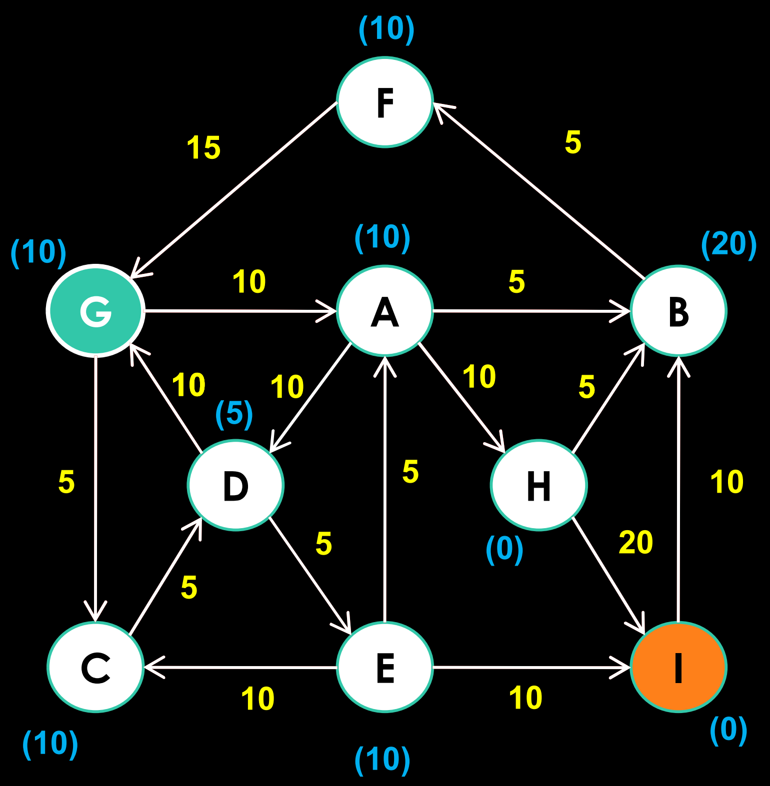


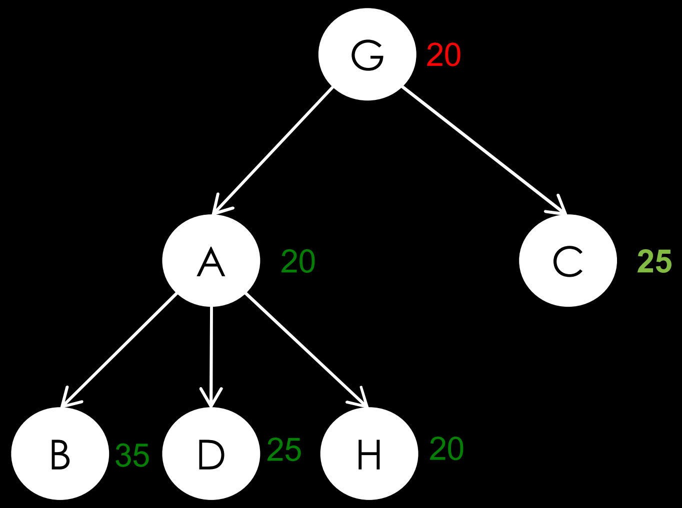
ALGORITMO RBFS



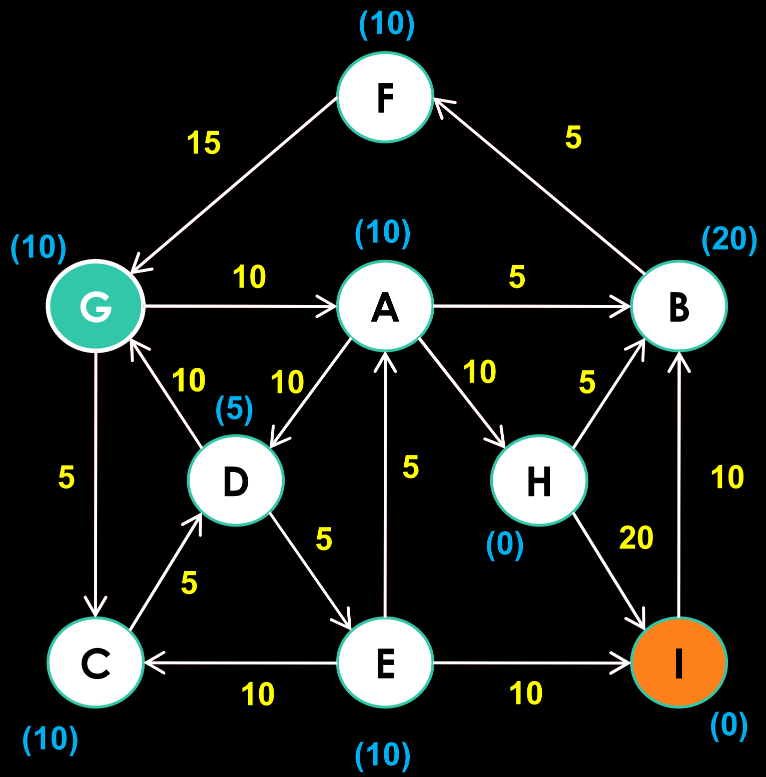


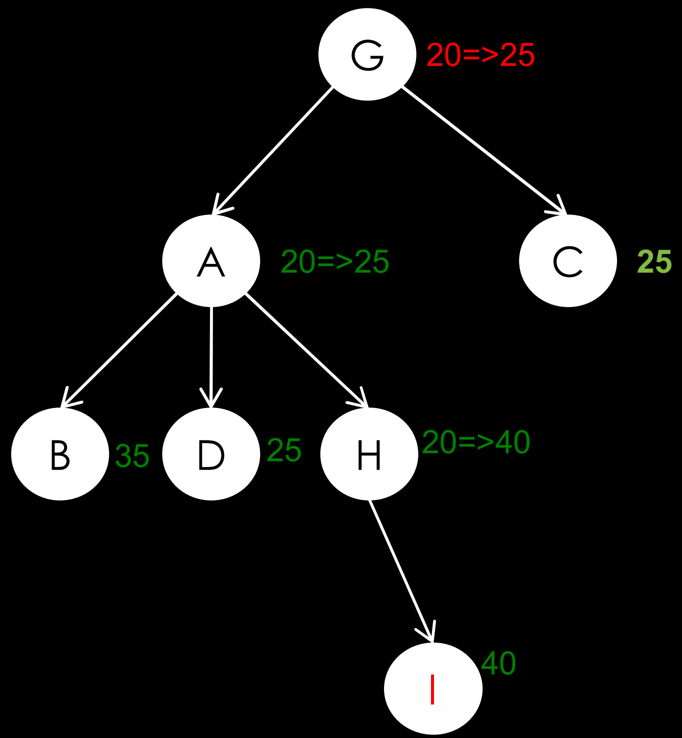
ALGORITMO RBFS



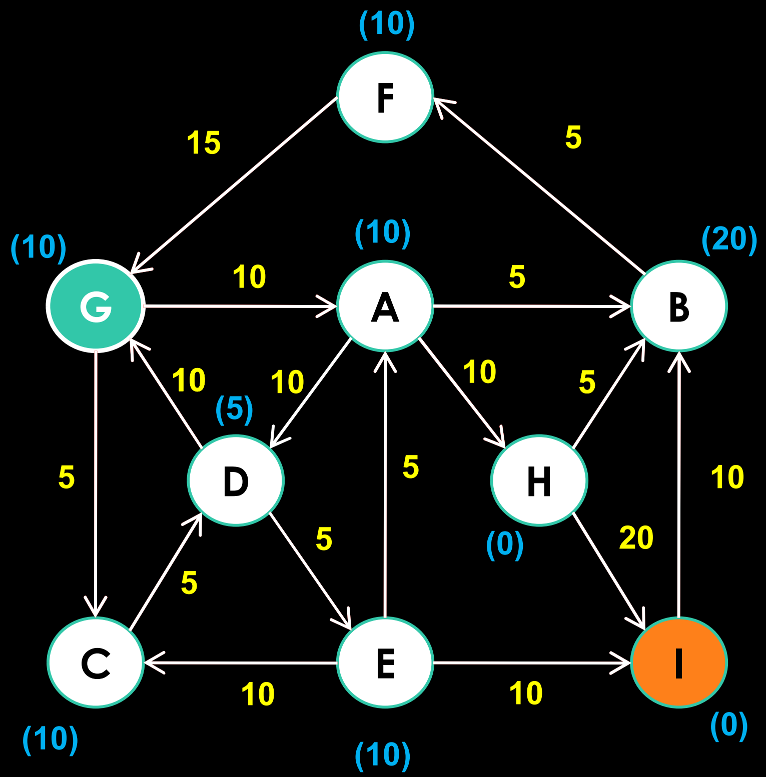


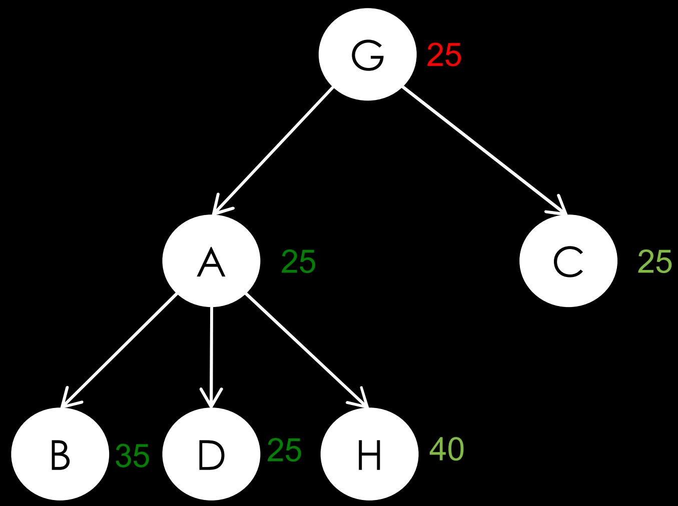
ALGORITMO RBFS



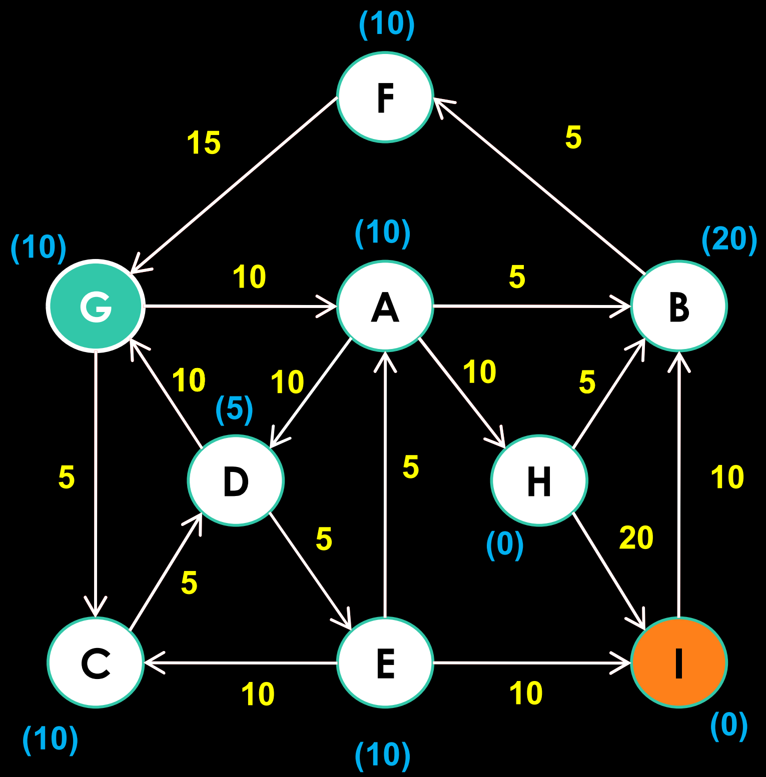


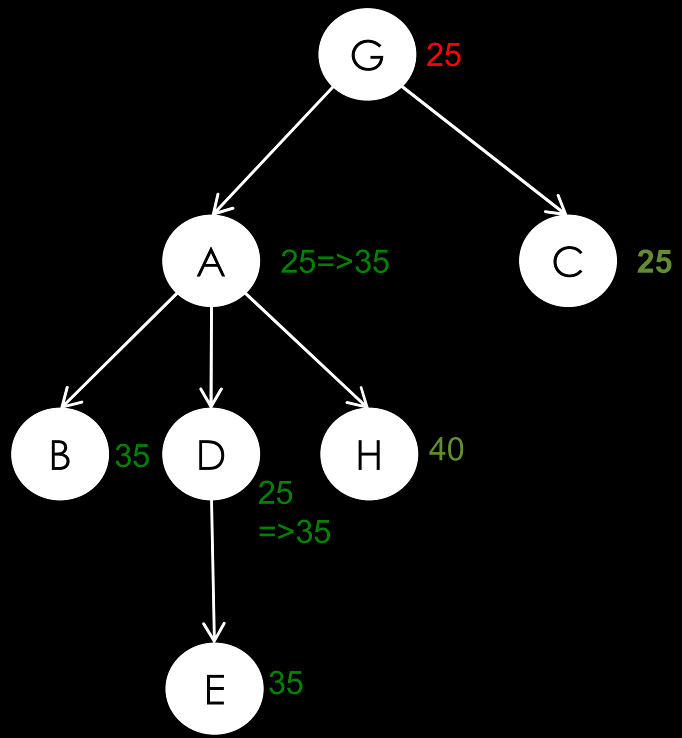
ALGORITMO RBFS



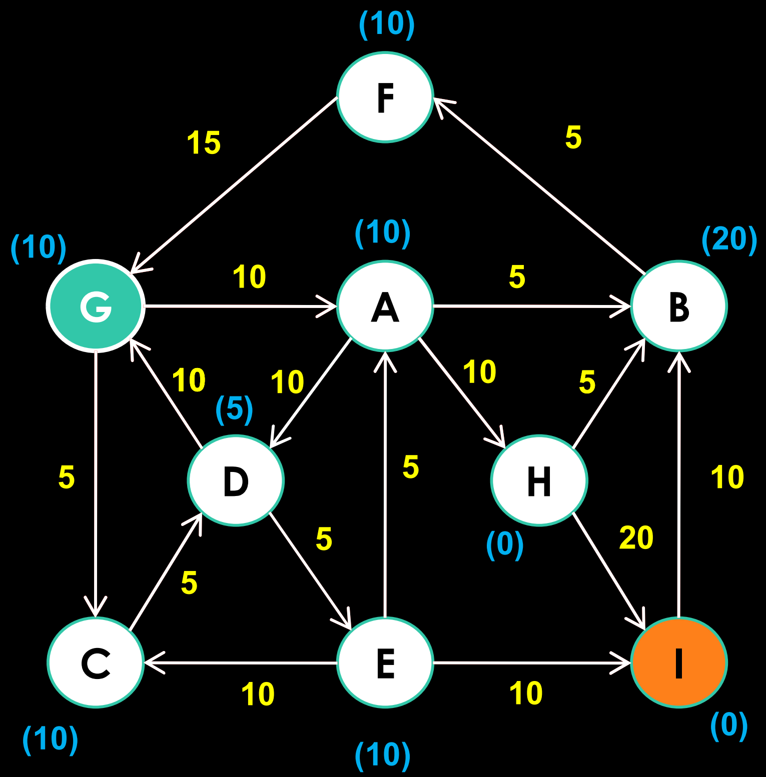


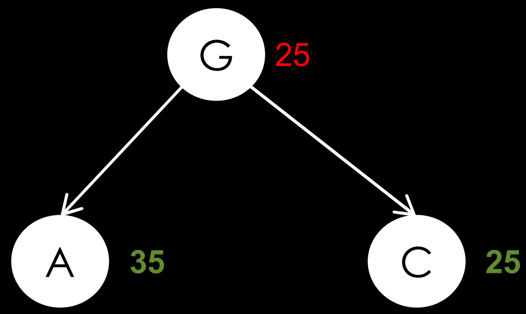
ALGORITMO RBFS



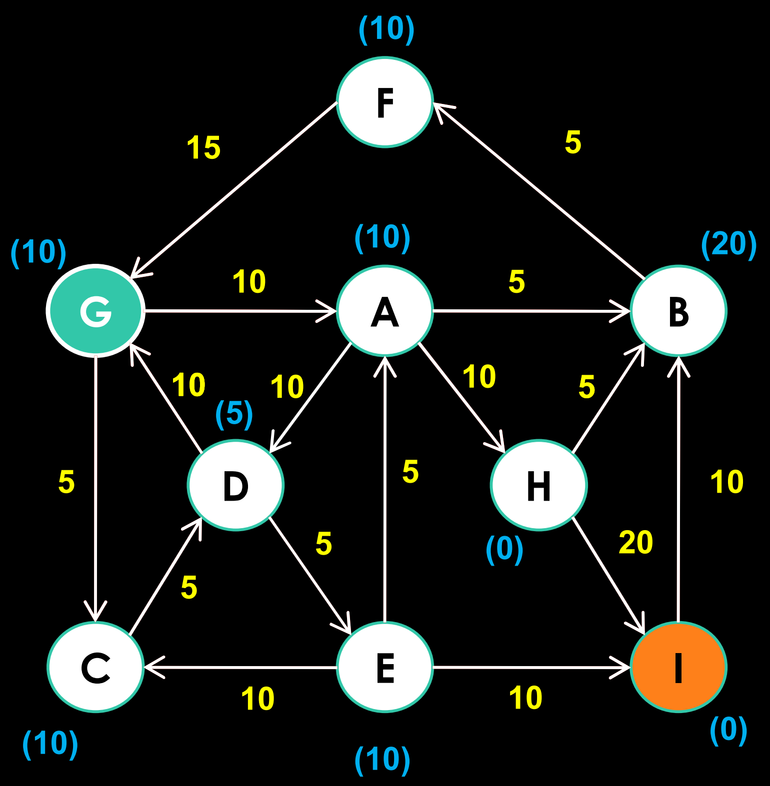


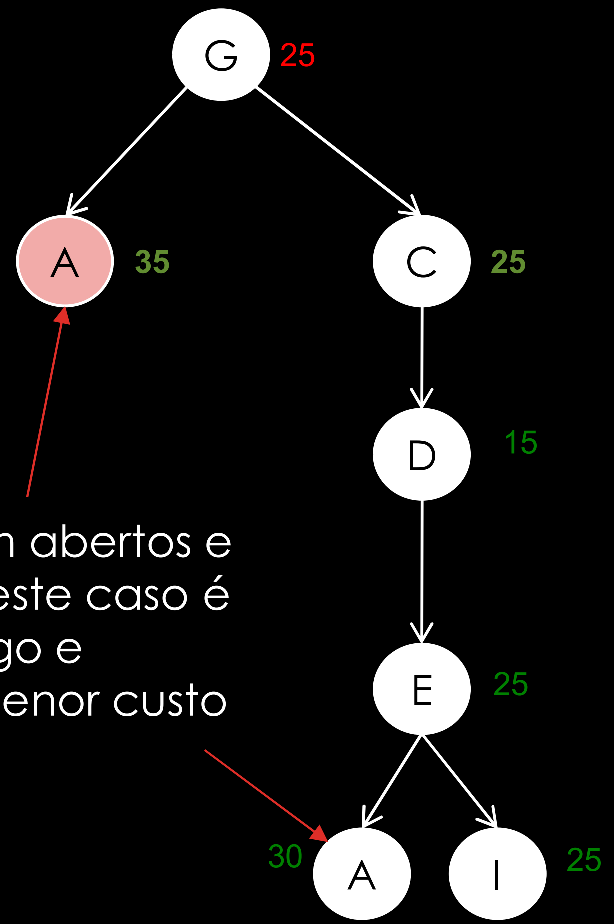
ALGORITMO RBFS





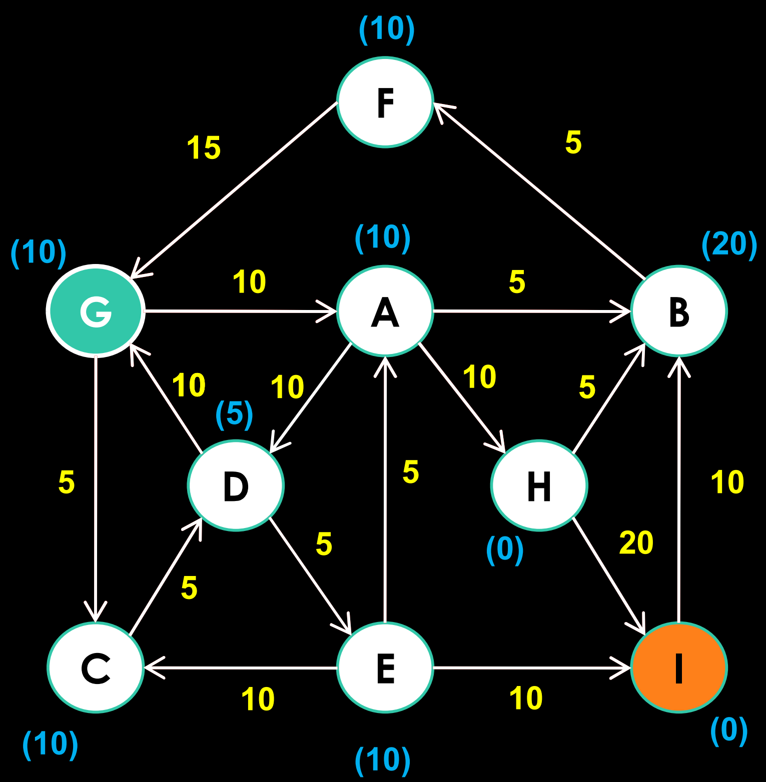
ALGORITMO RBFS

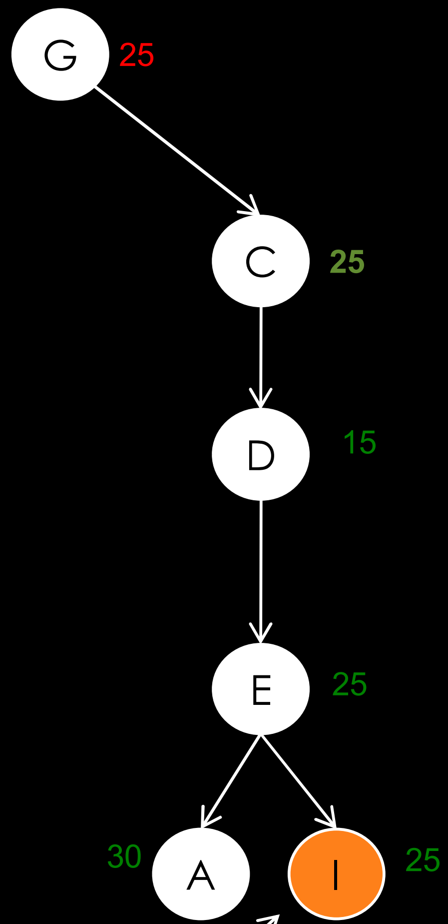




Nota: O nó A já está em abertos e tem maior custo. Neste caso é removido o nó antigo e conserva-se o de menor custo

ALGORITMO RBFS



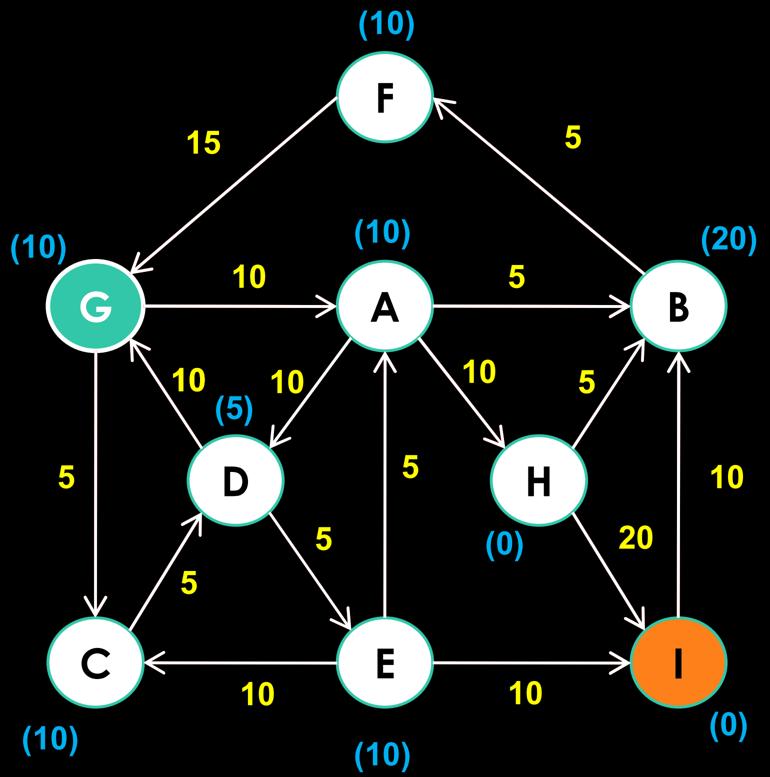


Pára e dá a solução: G, C, D, E, I

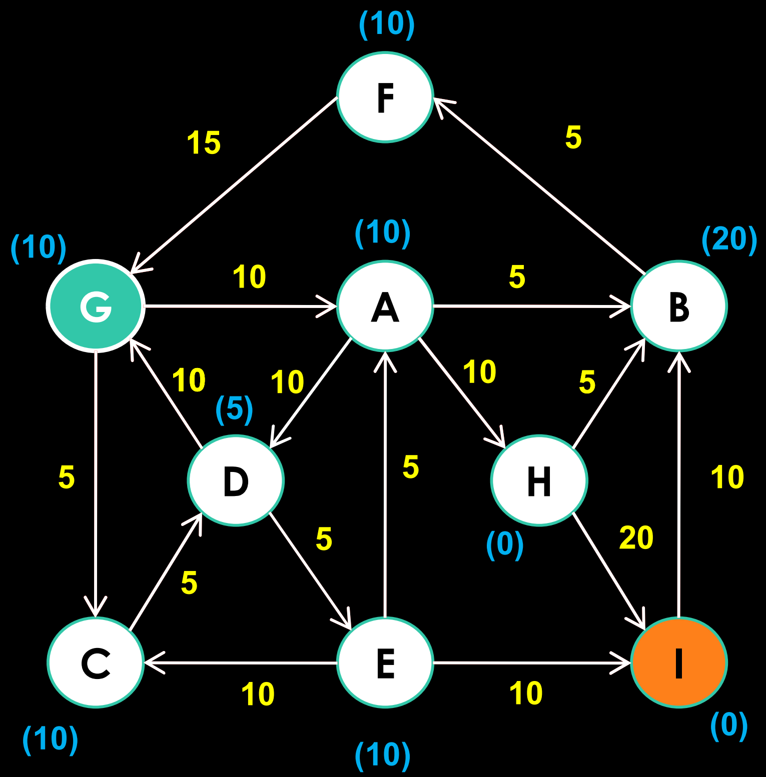
ANÁLISE COMPARATIVA SIMULAÇÃO

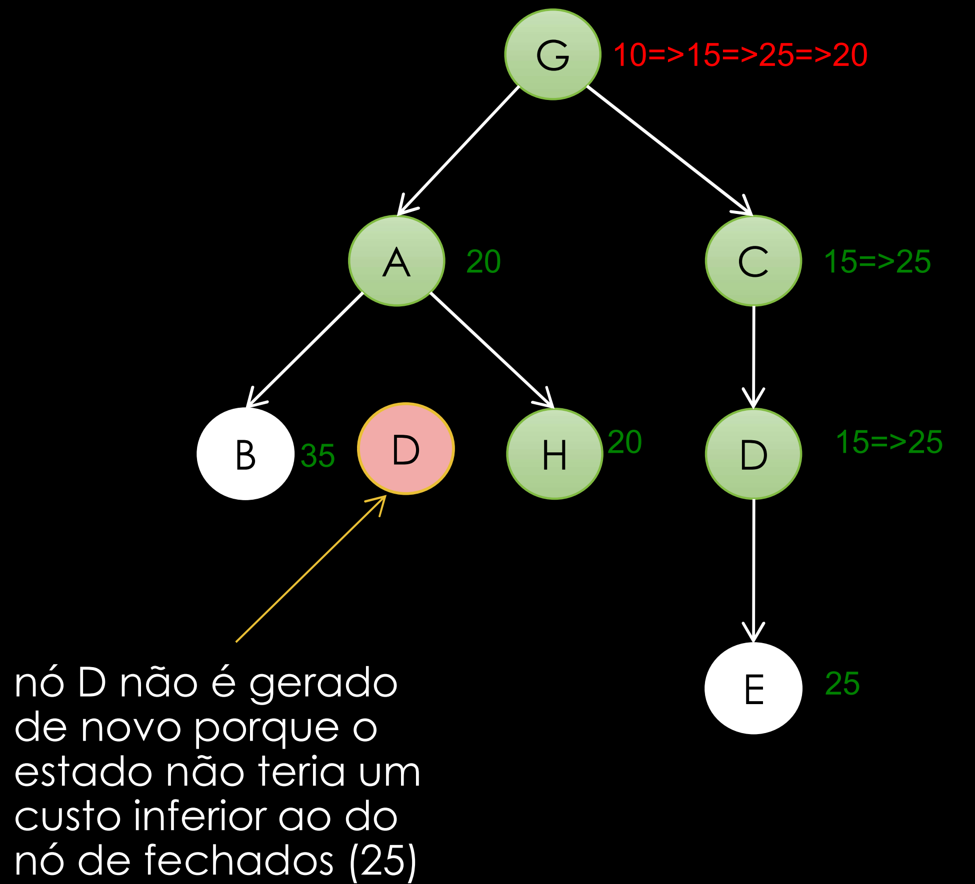
- A*
- IDA* - Iterative Deepening A*
- RBFS – Recursive Best First Search
- SMA* - Simplified Memory-Bound A*

ALGORITMO SMA* (MAX = 8 NÓS)

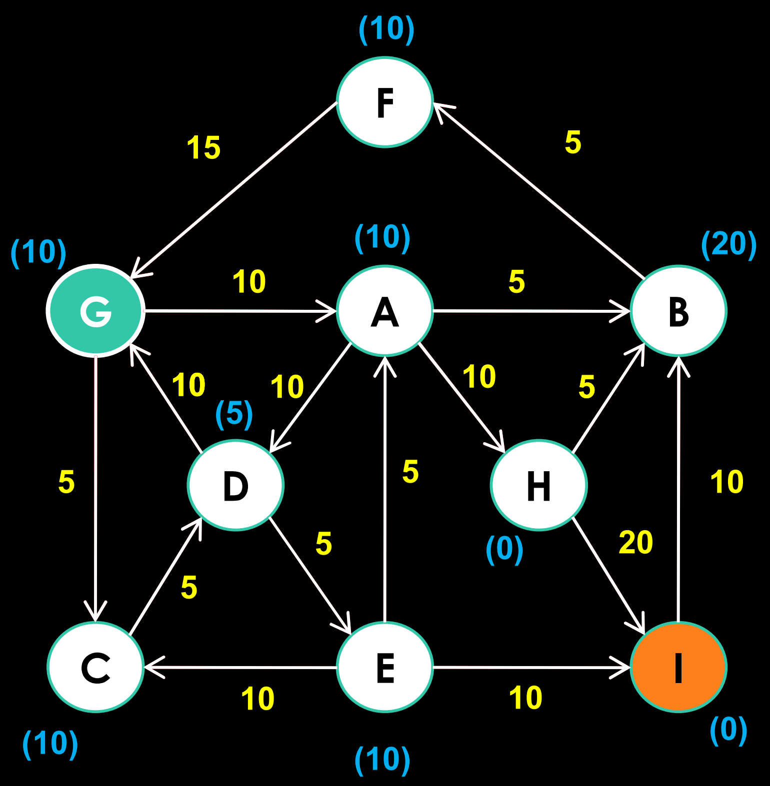


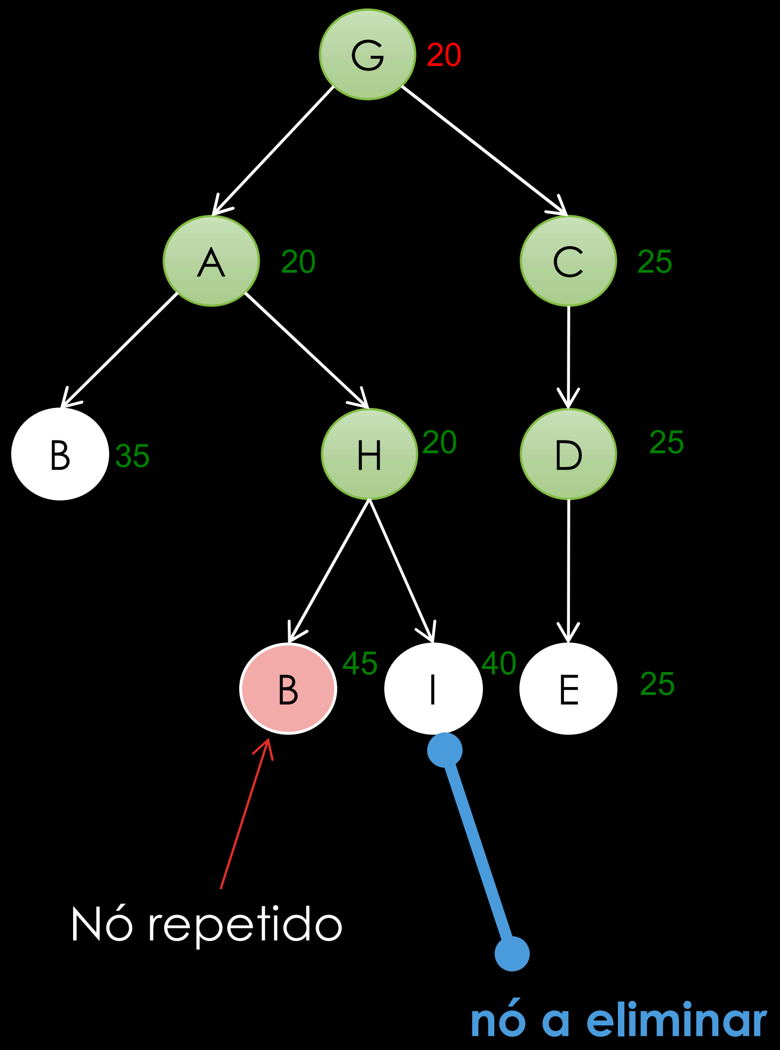
ALGORITMO SMA* (MAX = 8 NÓS)





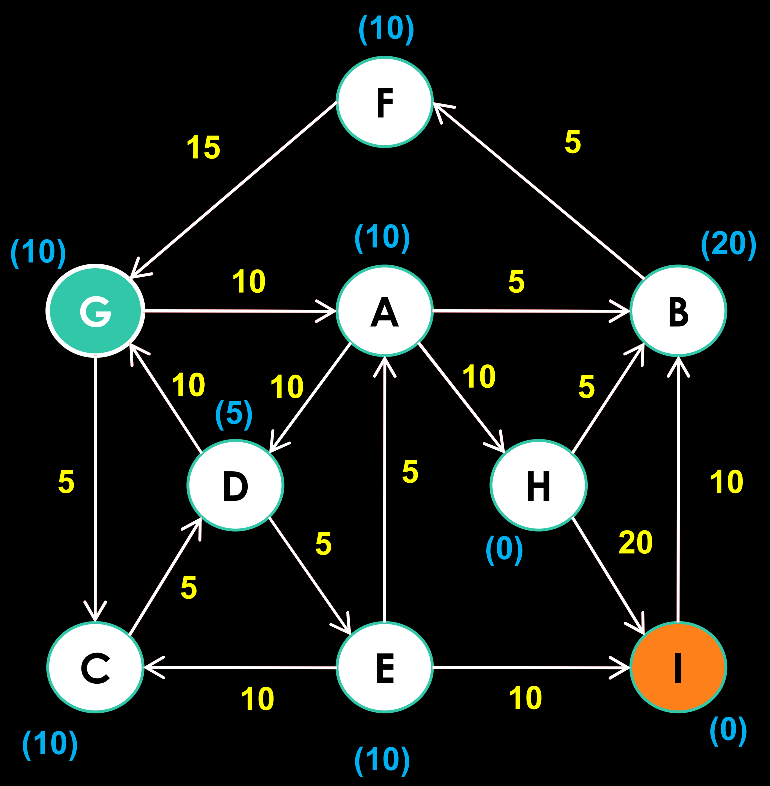
ALGORITMO SMA* (MAX = 8 NÓS)

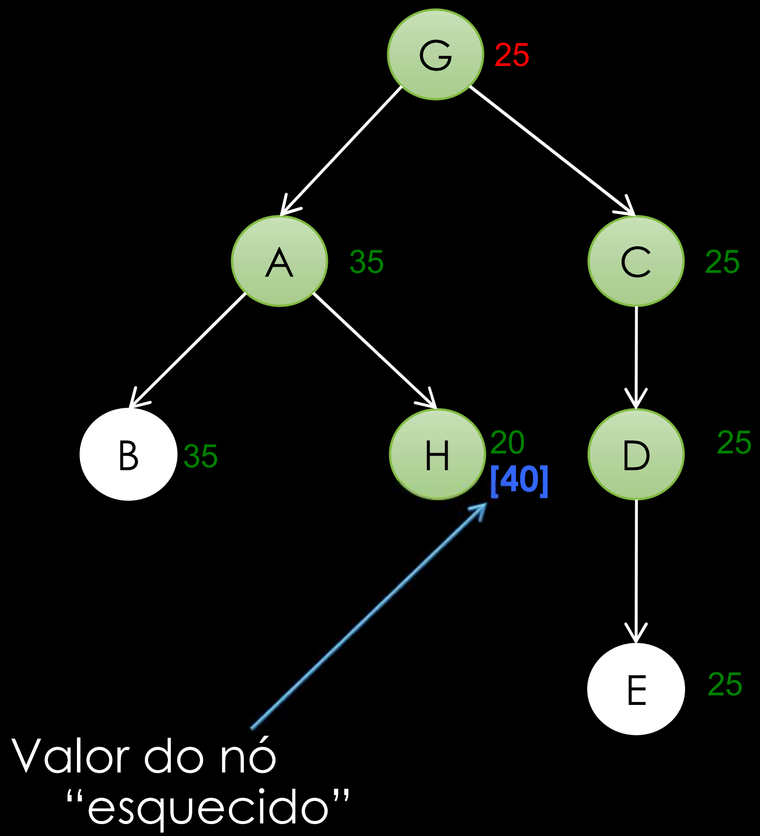




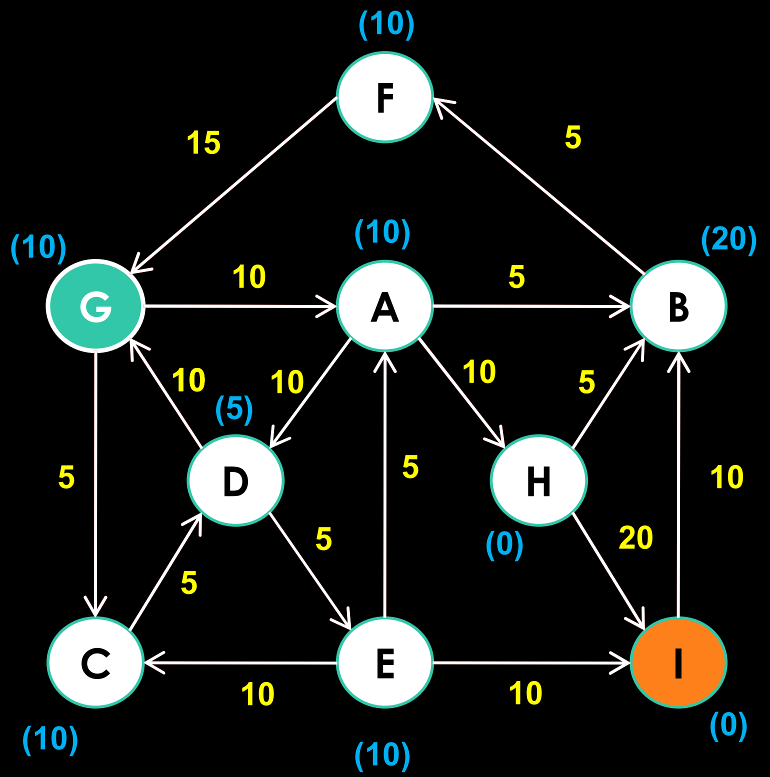
Até agora observou-se o comportamento normal do A* mas atingiu-se a capacidade de memória máxima.

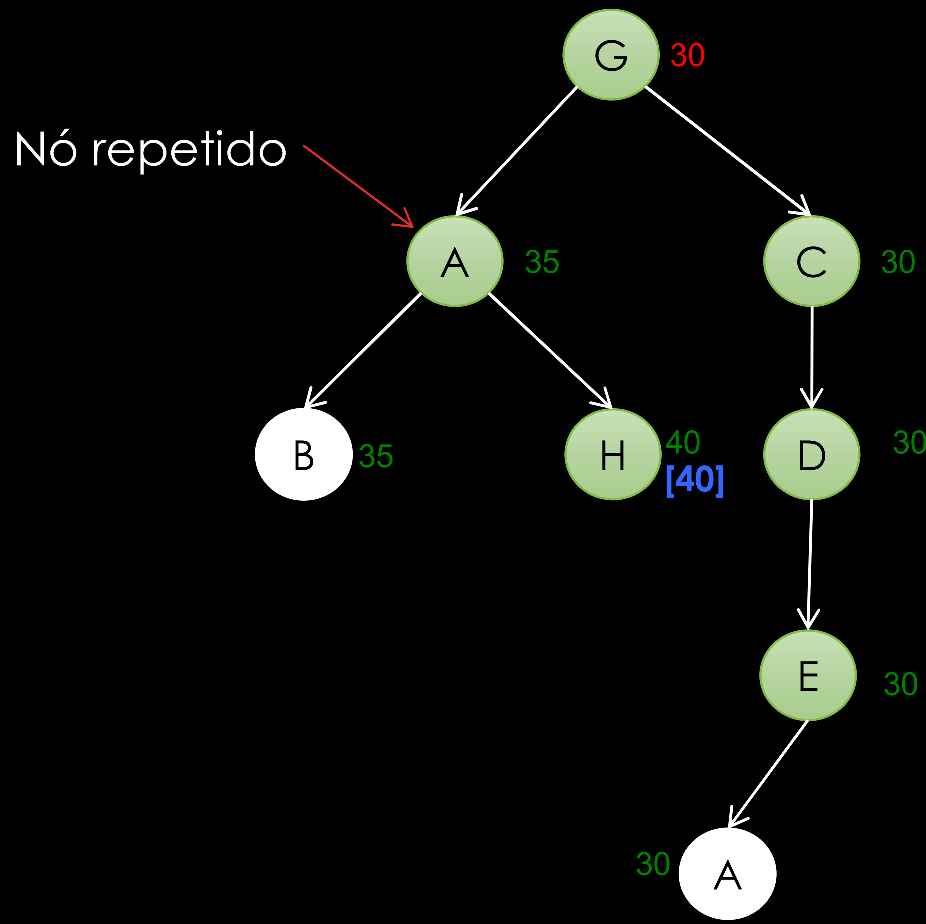
ALGORITMO SMA* (MAX = 8 NÓS)



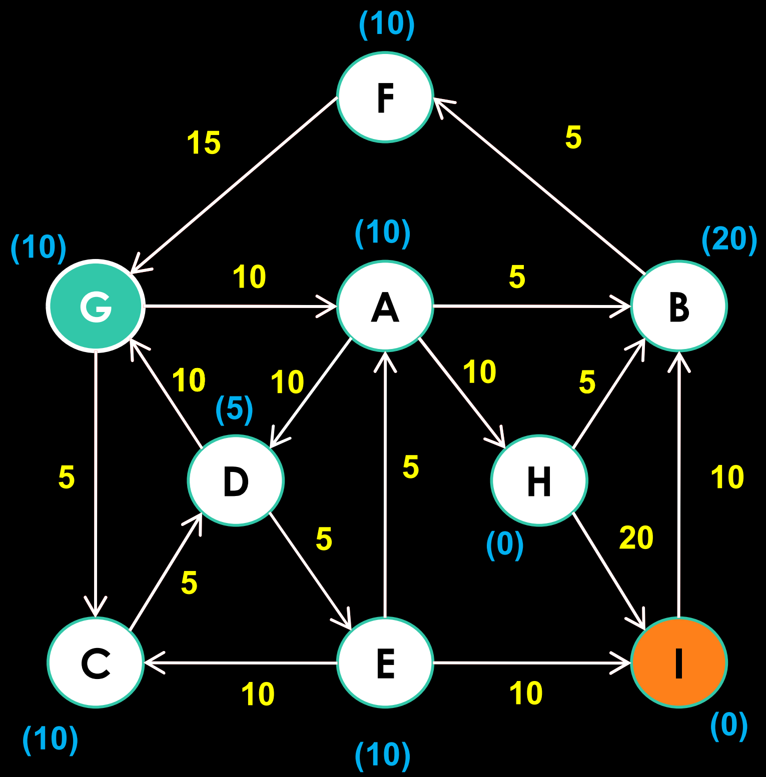


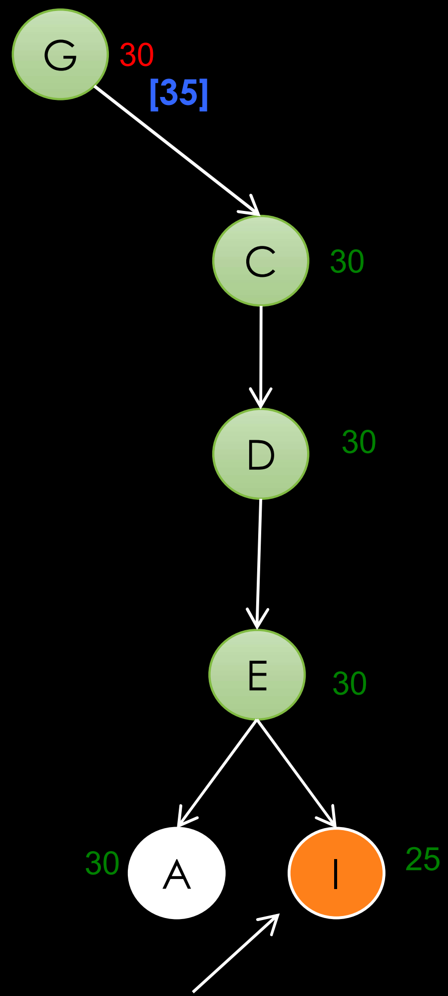
ALGORITMO SMA* (MAX = 8 NÓS)





ALGORITMO SMA* (MAX = 8 NÓS)





Pára e dá a solução: G, C, D, E, I

Exercicio 1: A ponte

- Quatro pessoas querem passar de um lado para o outro de uma ponte. Na ponte só conseguem passar duas pessoas de cada vez. É de noite e para passar a ponte é necessário uma tocha. Só existe uma tocha. Cada uma das pessoas demora um tempo diferente a percorrer a ponte: um demora 1 minuto, outro 2 minutos, outro 5 minutos e o último 10 minutos.
- Apresentar
  - uma descrição para o **estado** do problema.
  - Apresentar a lista dos **operadores**
  - uma regra para avaliação de estado final (considere que o tempo mínimo é 19 minutos).

CONT.

- Resolver no papel o problema indicando o **estado inicial**, o **estado objectivo**, os **operadores** e **estados** desde o estado inicial até ao estado objectivo.
- Indicar os 4 primeiros passos do algoritmo A* usando a seguinte função f'(n):
- f'(n) = g(n) + h'(n), em que:

  g(n) = Tempo já gasto

  h'(n) = Estimativa do tempo que se gastará
  igual à soma dos tempos das pessoas na margem inicial.

Esta heurística é admissível? Justifique.

EXERCICIO 2: TORRES DE HANÓI

- Considere o problema das torres de Hanói em que se pretende passar n discos de tamanhos distintos empilhados de uma posição para outra, existindo uma posição intermédia. Só se pode mover um disco de cada vez e não se pode colocar um disco maior sobre um mais pequeno.
- Defina o espaço de estados do problema, os operadores e uma função de avaliação de solução.
- Escreva a árvore de procura usando o algoritmo BF com n=4 a partir da seguinte posição inicial: ((1 3) (2 4) ())