Ciência da Computação

Inteligência Artificial

Algoritmos de Busca

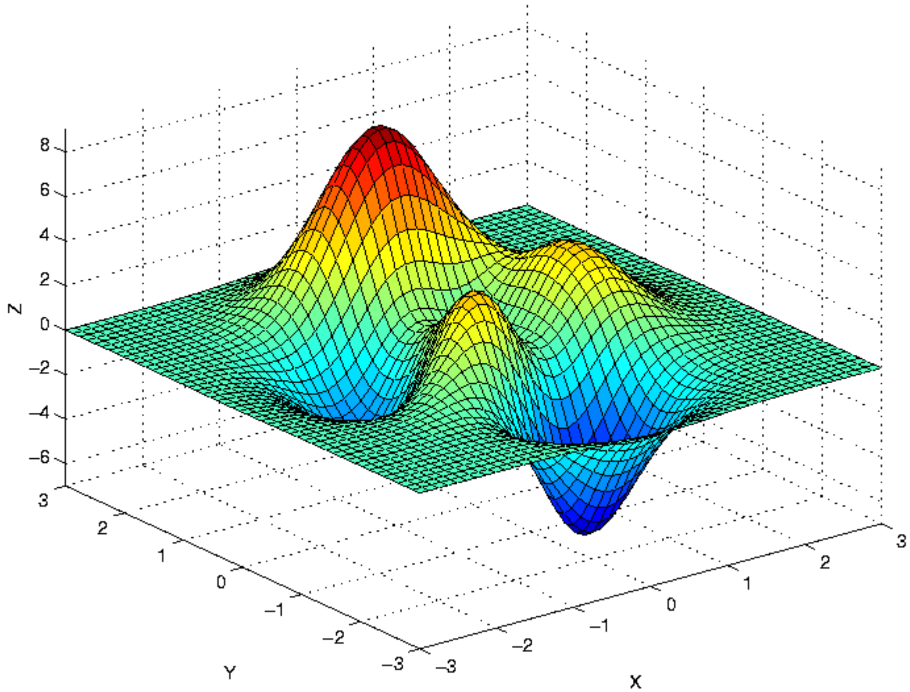
Adriana Postal

André Luiz Brun

unioeste
Universidade Estadual do Oeste do Paraná

Introdução

- A solução de um problema está em seu espaço de busca, composto por todas as soluções possíveis para o problema
- Buscar a solução para um problema é um enorme desafio



Fonte: Michaelis (2022)

Introdução

- O espaço de busca pode (em alguns casos) ser organizado em forma de árvores ou grafos
  - Os nós correspondem a situações de um problema
  - As arestas correspondem a movimentos permitidos ou ações ou passos da solução
  - Especificação de um estado final
  - Especificação de um estado inicial
- Os algoritmos de busca operam sobre árvores
  - Quando a representação utilizar grafos, eles serão modificados para incluir um mecanismo de controle de ciclos

Introdução

<u>Problema dos Jarros de Água</u>

Em posse de dois jarros de água, um com capacidade de 4 litros e outros que suporta até 3 litros, desejo separar uma quantidade referente a dois litros na jarra de 4 litros

Duas variáveis:

X (volume na jarra de 4 litros) $\in \{0,1,2,3,4\}$

Y (volume na jarra de 3 litros) $\in \{0,1,2,3\}$

Introdução

Problema dos Jarros de Água

Restrições:

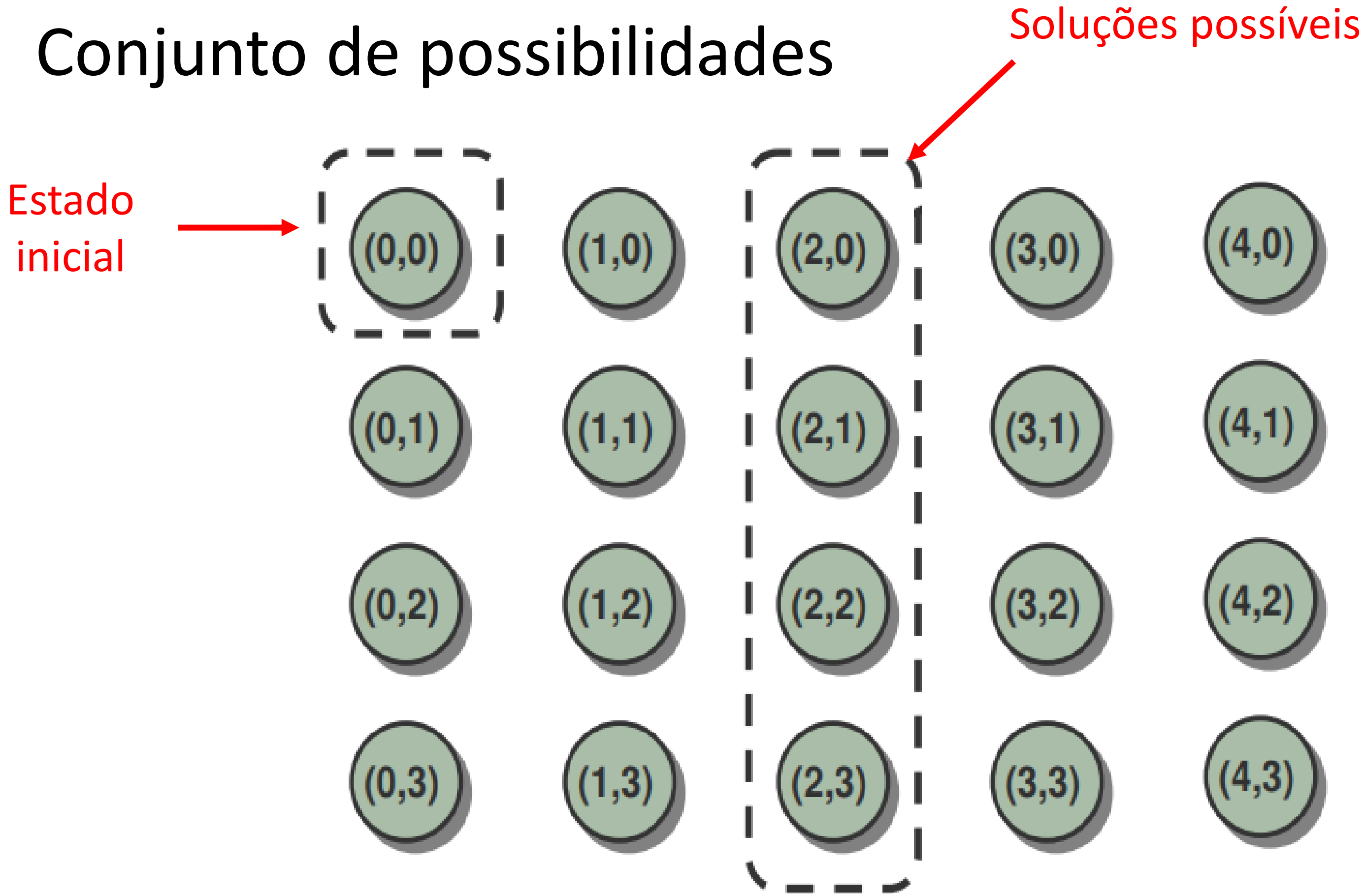
- Não é possível colocar água em um jarro cheio
- Não é permitido valores fracionários

Estado inicial: X=0 e Y=0 → (0,0)

Estado final: X=2 e Y=n → (2,n)

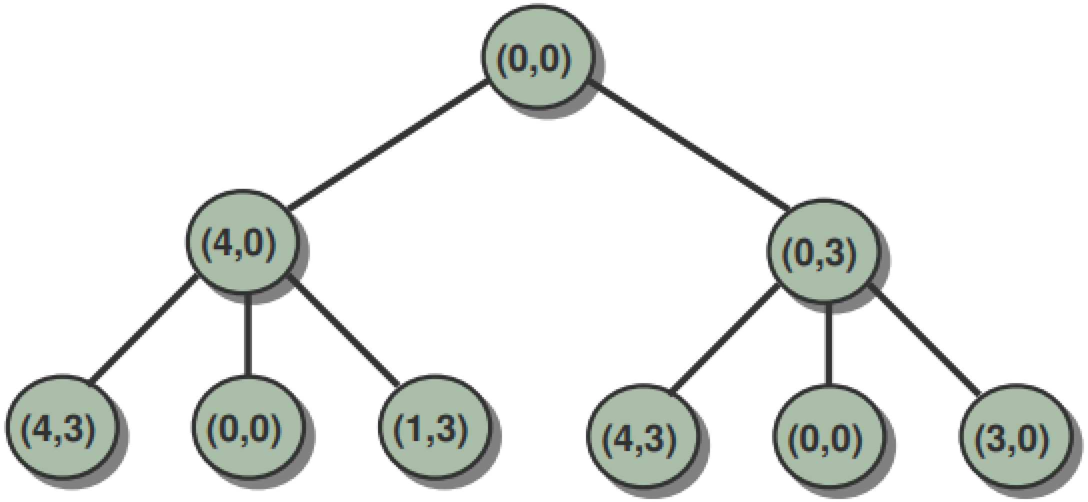
Introdução

Conjunto de possibilidades



Introdução

Mapeamento das ações



Introdução

Mapeamento das ações

(0,0) → (0,3) → (3,0) → (3,3) → (4,2) → (0,2) → (2,0)

Abordagens para busca

- Orientada por dados
  - Encadeamento progressivo
  - Top-down

- Orientada por objetivos
  - Encadeamento regressivo
  - Bottom-up

Abordagens para busca

- Orientada por dados
  - Também chamada de encadeamento para frente (progressivo)
  - Parte de um estado inicial e usa ações permitidas para alcançar o objetivo
  - É útil quando os dados iniciais são fornecidos e não temos clareza sobre o objetivo
  - Um exemplo é a subida da encosta

Abordagens para busca

- Orientadas por objetivo
  - Ou encadeamento para trás (regressivo)
  - Começa de um objetivo e volta para um estado inicial, vendo quais deslocamentos poderiam ter levado ao objetivo
  - Útil em situações nas quais o objetivo pode ser claramente especificado
- Exemplo:
  - Um teorema a ser provado ou encontrar uma saída em um labirinto
  - É a melhor escolha em problemas como diagnósticos médicos: A condição a ser diagnosticada é conhecida (objetivo)

Abordagens para busca

As buscas guiadas por dados e guiadas por objetivo irão produzir o mesmo resultado, entretanto, de acordo com a natureza do problema a ser resolvido, uma das estratégias pode revolvê-lo de forma mais eficiente do que a outra, normalmente devido ao número de testes a ser realizado.

Propriedades dos Métodos de Busca

Para um método de busca ser mais útil, ele deve ter algumas propriedades importantes:

- Completude (ou completeza)
- Complexidade
- Otimalidade
- Irrevogabilidade

Propriedades dos Métodos de Busca

Completude

- Um método é dito completo se ele garantir encontrar um estado objetivo (caso este exista).

- É uma característica importante e desejável: um algoritmo de busca que não encontre uma solução não é útil.

Propriedades dos Métodos de Busca

Complexidade

- É útil descrever quão eficiente é um método em termos de tempo e espaço.
- Complexidade em termos de tempo: relacionada a quanto tempo o método leva para encontrar a solução.
- Complexidade em termos de espaço: relacionada à quantidade de memória que o método precisa utilizar

Propriedades dos Métodos de Busca

<u>Complexidade</u>

- Essas complexidades precisam ser equacionadas:
  - Um método muito rápido nem sempre encontrará a melhor solução
  - Um método que examine cada possível solução garantirá encontrar a melhor solução, mas poderá ser muito ineficiente

Propriedades dos Métodos de Busca

Otimalidade

- Um método de busca é dito ótimo se ele garantir achar a melhor solução que exista.
- O método pode não ser eficiente, mas uma vez que a solução ótima for encontrada, garante-se que ela é a melhor

Propriedades dos Métodos de Busca

Otimalidade

- Em alguns casos, o termo ótimo é utilizado para descrever um algoritmo que encontre uma solução no menor tempo possível
- Neste caso, utilizamos o conceito de admissibilidade no lugar de “quanto a ser ótimo”
- Um algoritmo é considerado admissível se ele garantir encontrar a melhor solução

Propriedades dos Métodos de Busca

Irrevogabilidade

- Métodos que permitem retrocesso (backtracking) são descritos como uma tentativa
- Por outro lado, métodos que não retrocedem, acabam examinando apenas um caminho e são descritos como irrevogáveis
- O nome irrevogável é dado pois uma vez tomada uma decisão ela não pode ser desfeita

Propriedades dos Métodos de Busca

Exemplos de Irrevogabilidade

- Busca em profundidade é busca por tentativa
- Subida de encosta é irrevogável
- Estratégia gulosa é irrevogável
- Programação dinâmica é busca por tentativa

- Métodos irrevogáveis encontrarão, frequentemente, soluções subótimas para o problema

Contrafortes, Platôs e Cristas

Vamos considerar que o espaço de busca segue uma representação bidimensional, de forma que os eixos X e Y correspondem às variáveis que devem ser escolhidas e o eixo Z (altura) corresponde ao resultado

O objetivo nesse caso seria maximizar a altura obtida por cada por ordenado (X,Y)

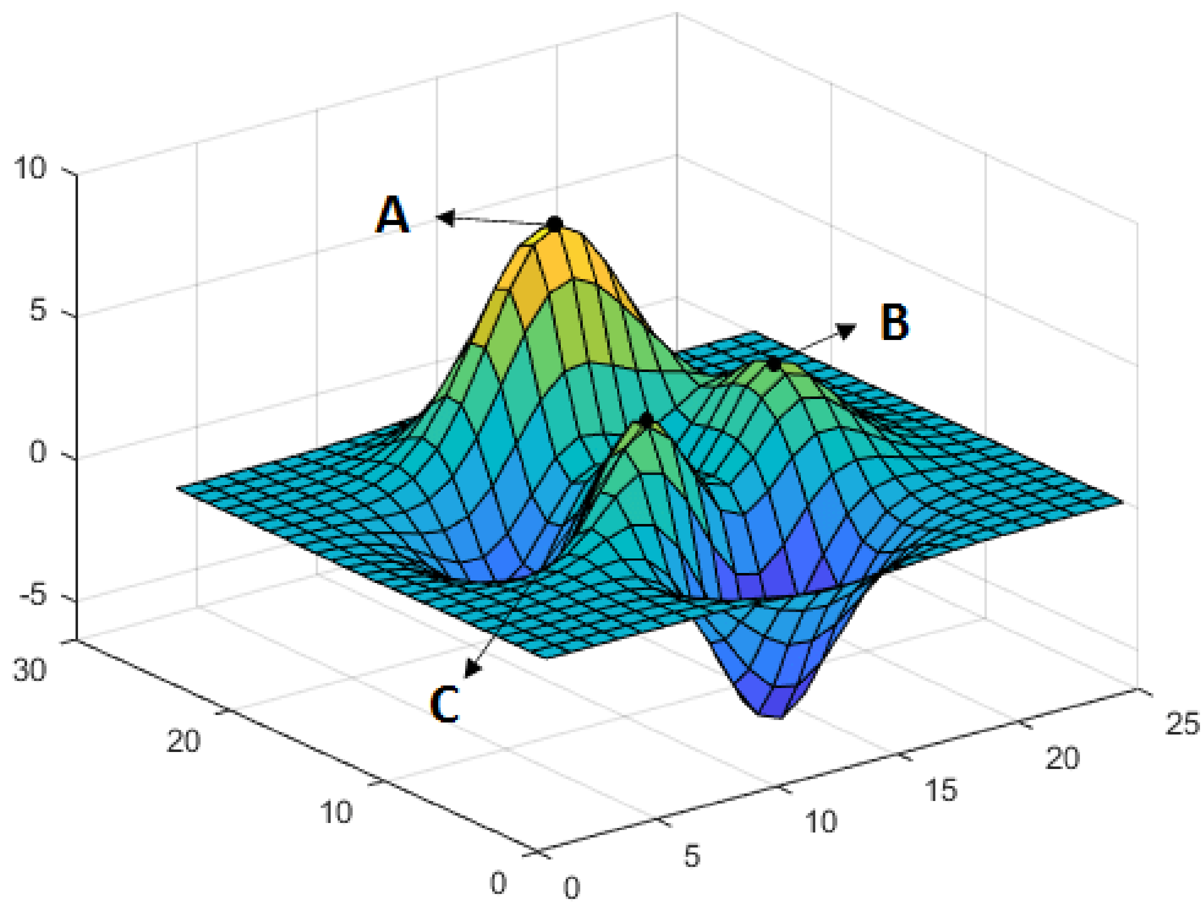
Contrafortes, Platôs e Cristas

Neste tipo de representação o objetivo é maximizar o resultado: nestes casos, os métodos de busca procuram encontrar o ponto mais alto no espaço

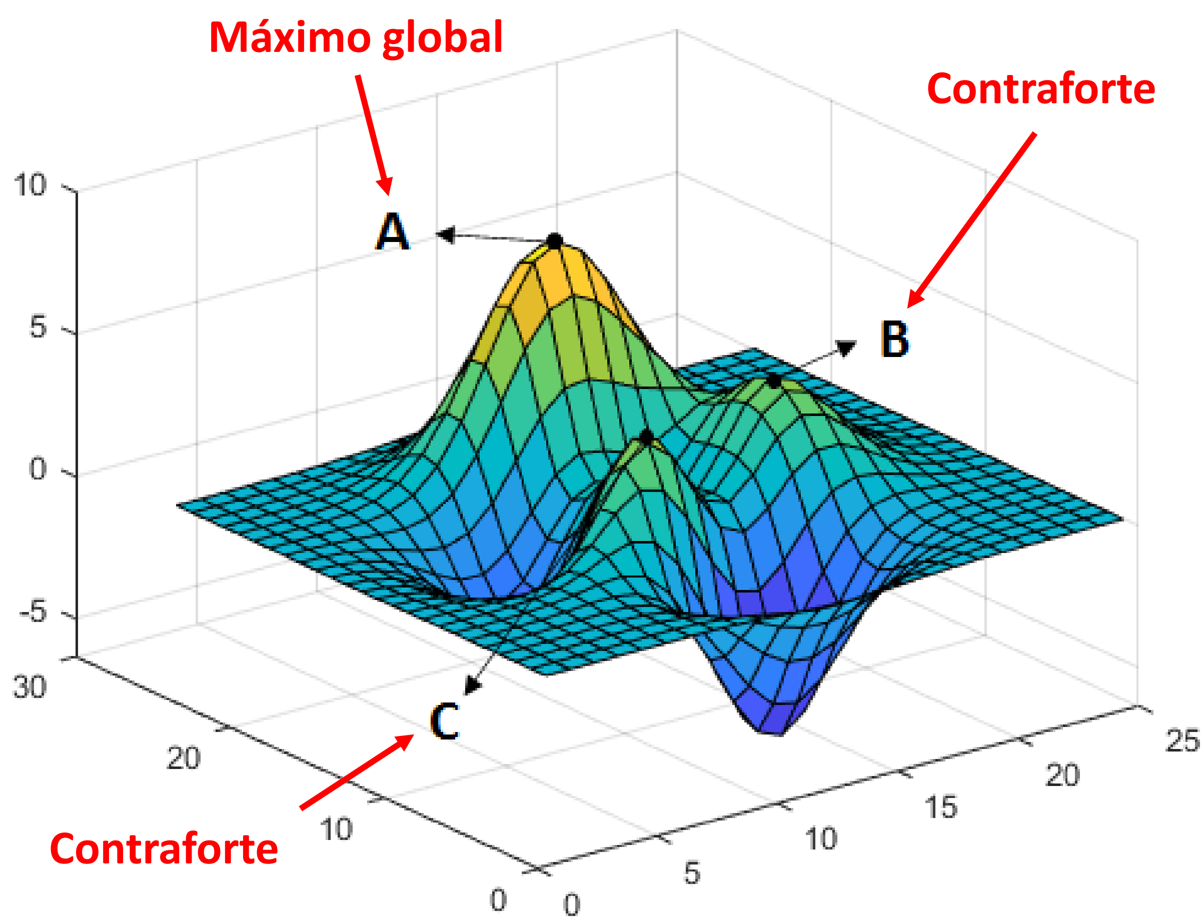
Alguns métodos de busca podem ser induzidos ao erro por 3 fatores:

- Contrafortes (ou máximos locais)
- Platôs
- Cristas

Contrafortes



Contrafortes



Contrafortes

- Ou máximo local
- É uma parte do espaço de busca que parece ser preferível às partes em torno dele
- Técnicas de subida da colina atingirão esse pico
- Para evita-los é interessante que o método permita “saltos” no espaço de busca

Platôs

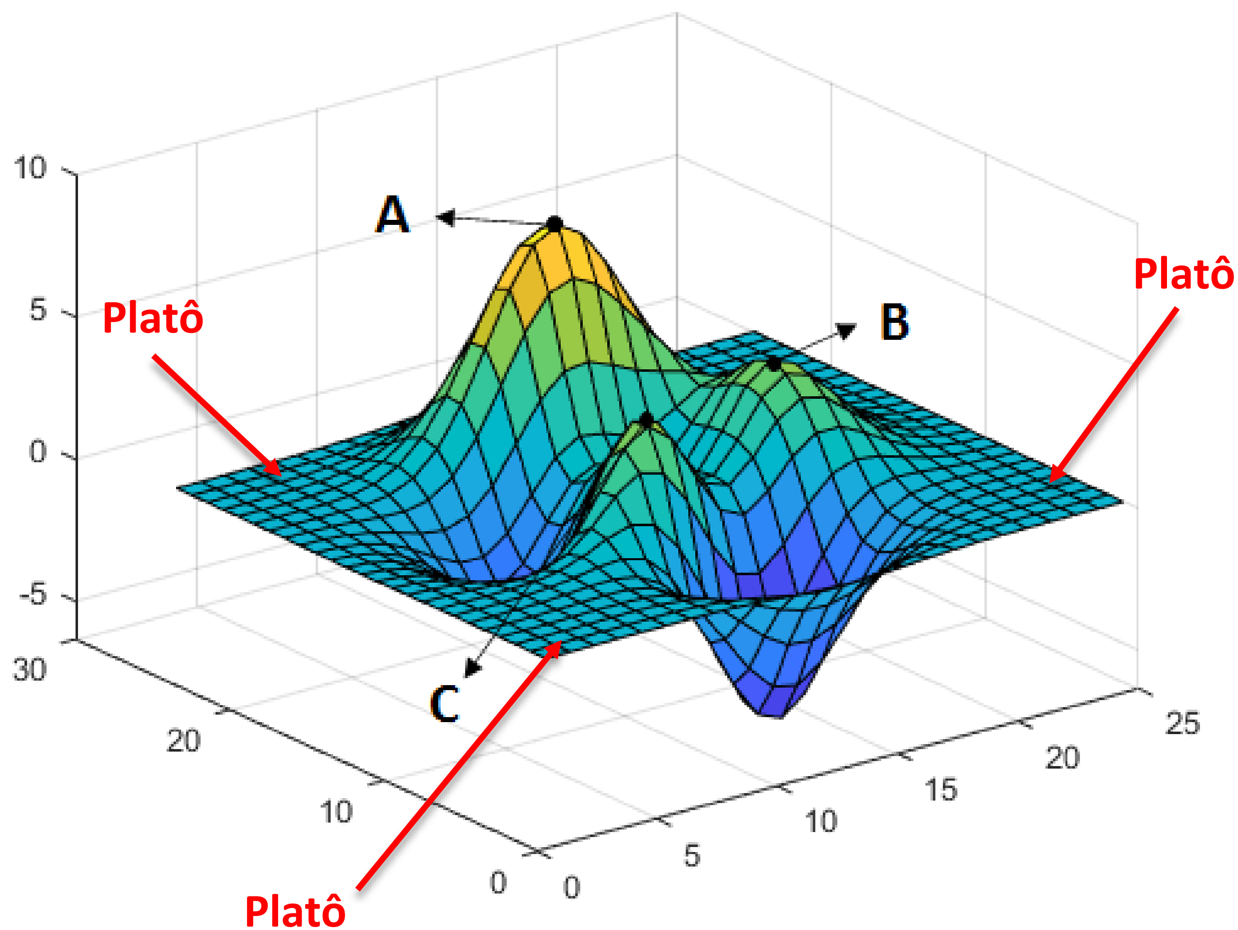
- São regiões em um espaço de busca na qual todos os valores são os mesmos

- Um mesmo espaço de busca pode ter diversos platôs

- Cada platô pode assumir um valor específico

- Platôs podem ser encontrados em pontos de máximo (local ou global)

Platôs



Cristas

- Uma crista é uma região longa e estreita de terras altas com terras baixas em ambos os lados

Métodos de Busca

- Busca Cega
- Busca Heurística

Métodos de Busca

Busca Cega

Também recebe as nomenclaturas de não-informada, uniforme ou busca exaustiva

- Assim chamada porque não leva em conta informações específicas sobre o problema a ser resolvido
- O processo consiste apenas geram sucessores e verificam se o estado objetivo foi atingido

Métodos de Busca

Busca Cega

Os métodos cegos se distinguem pela ordem em que os nós são expandidos já que não há uma orientação na escolha

Alguns métodos que são considerados buscas cegas:

- Testar e gerar (ou força bruta)
- Busca em largura
- Busca em profundidade
- Busca em profundidade limitada
- Busca de aprofundamento iterativo

Métodos de Busca

Busca Cega

A avaliação do desempenho destas estratégias se dá verificando as propriedades citadas anteriormente

Na maioria dos casos, a busca cega é ineficiente. Por exemplo: como encontrar um barco perdido?

Métodos de Busca

Busca Heurística

Ou busca informada

- Os algoritmos são baseados em algum conhecimento que pode acelerar a solução do problema
- Exemplo de informação:
  - Estimativa da distância entre os nós e a meta

Métodos de Busca

Busca Heurística

- Os parâmetros podem ser incertos e não garantem uma solução ótima
- Entretanto, na maioria das vezes, aceleram o processo já que as heurísticas exploram as direções aparentemente boas
- As informações adicionais são usadas nos nós que ainda não foram explorados para decidir quais nós examinar a seguir

Métodos de Busca

<u>Busca Heurística</u>

Métodos heurísticos mais conhecidos:

- Busca pela melhor escolha
- Subindo o morro (ou a colina ou a encosta)
- Têmpera simulada
- Busca menor custo ou A*
- Minimax
- Poda alfa-beta
- Busca gulosa
- Busca com limite superior (IDA*, SMA*)
- ...

Métodos de Busca

Busca Heurística

Função de avaliação: Cada estado a ser avaliado está, de algum modo, associado com um valor que estima o quanto se está distante do estado final

Está diretamente ligada ao problema em que é aplicada

Depende do conhecimento sobre o espaço de busca

Métodos de Busca

Busca Heurística

- Função de custo: mede a dificuldade para ir de um estado para o seu vizinho: mede, com exatidão, o quanto se está distante do estado inicial

- A informação que pode compor uma informação heurística é o custo do caminho:
  - Soma-se a função custo com a função de avaliação

Gerar e Testar

- Busca Exaustiva (ou Força Bruta)
- É a abordagem mais simples
- Custo computacional pode ser fator importante
- Não pressupõe conhecimento adicional

Gerar e Testar

Funcionamento básico da estratégia:

1. Gera cada nó do espaço
2. Testa o nó para verificar se é um nó objetivo:

Em caso positivo, a busca encerra com sucesso

Em caso negativo o procedimento segue para o próximo nó

Para ter sucesso, o método precisa de um gerador adequado

Gerar e Testar

Esse gerador deve satisfazer a 3 propriedades:

1. Ser completo: deve gerar todas as soluções possíveis

2. Não deve ser redundante: não deve gerar a mesma solução 2 vezes

3. Ser bem informado: só deve propor soluções adequadas e que combinem com o espaço de busca

Gerar e Testar

Algoritmo 1: Testar e Gerar
1. Gerar uma solução possível. Para alguns problemas isto significa gerar um ponto em particular no espaço do problema. Para outros, significa gerar um caminho, a partir de um estado inicial. 2. Testar para ver se a solução gerada é realmente uma solução, comparando o ponto escolhido ou o ponto final do caminho escolhido com o conjunto de estados-meta aceitáveis. 3. Se uma solução tiver sido encontrada, saia. Caso contrário, volte à etapa 1.

Busca em Profundidade

- Também conhecida como Primeiro em Profundidade ou Depth-first Search (DFS)
- Segue cada caminho até sua maior profundidade antes de seguir para o próximo caminho
- É um método de busca exaustiva ou força bruta
- Dadas as características da abordagem normalmente é implementada usando-se pilha

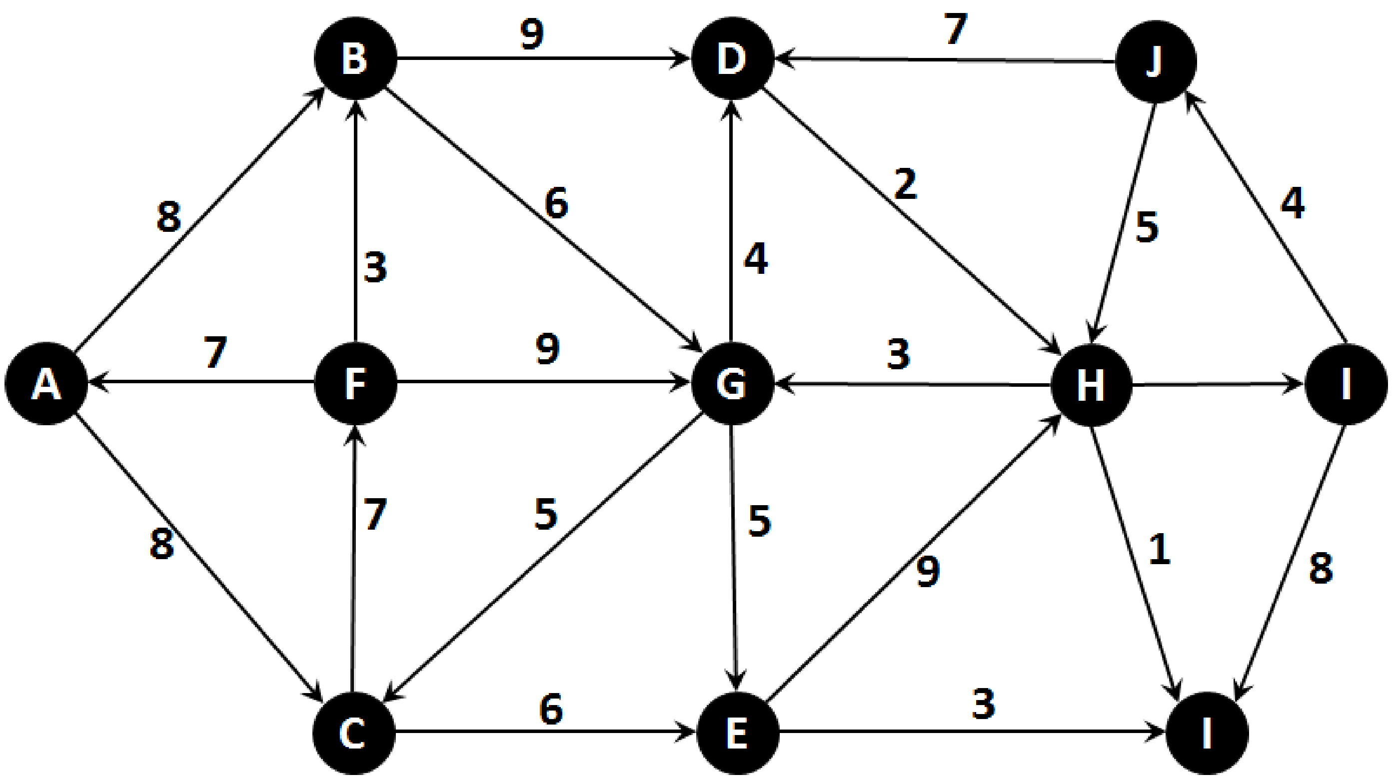
Busca em Profundidade

- Utiliza um método chamado retrocesso cronológico: volta na árvore de busca, uma vez que um caminho sem saída seja encontrado
- Permite a escolha de um caminho diferente do buscado anteriormente
- É assim chamado por desfazer escolhas na ordem contrária ao momento em que foram tomadas

Busca em Profundidade

```
DFS(G)
    for cada vértice u ∈ V[G]
        u.cor = BRANCO
        u.π = NIL
    tempo = 0
    for cada vértice u ∈ V[G]
        if u.cor == BRANCO
            DFS-Visit(G, u)

DFS-Visit(G, u)
    tempo = tempo + 1            // vértice branco u acabou de ser descoberto
    u.d = tempo
    u.cor = CINZENTO
    for cada v ∈ G.Adj[u]        // explorar aresta (u, v)
        if v.cor == BRANCO
            v.π = u
            DFS-Visit(G, v)
    u.cor = PRETO                // pintar u de preto; está terminado
    tempo = tempo + 1
    u.f = tempo
```

Busca em Profundidade



Busca em Profundidade

Vantagens

- Necessita de pouca memória
- Bom para problemas com muitas soluções
- Geralmente, é mais fácil de implementar que a busca em largura

Busca em Profundidade

<u>Desvantagens</u>

Como existem espaços de estados infinitos:

- O algoritmo pode se perder
- Pode seguir um caminho infinito
- Pode entrar em ciclos
- Pode nunca chegar a um nó final

Busca em Profundidade

Para evitar caminhos infinitos (com ciclos), pode-se adicionar um refinamento no algoritmo:

- Definir uma profundidade limite para o algoritmo
- Neste caso, o algoritmo torna-se uma Busca com Profundidade Limitada
- Sobre a busca em profundidade:
- Não é completa: falha em espaços com profundidade infinita ou com ciclos
- Não é ótima

Busca em Largura

- Também conhecida como Busca em Amplitude, Busca em Extensão, Busca em Nível e Breadth-First Search (BFS)
- Começa examinando todos os nós de um nível abaixo do nó inicial
- Em caso negativo, o método busca um nível abaixo e segue tratando um nível por vez
- Normalmente implementado usando-se fila

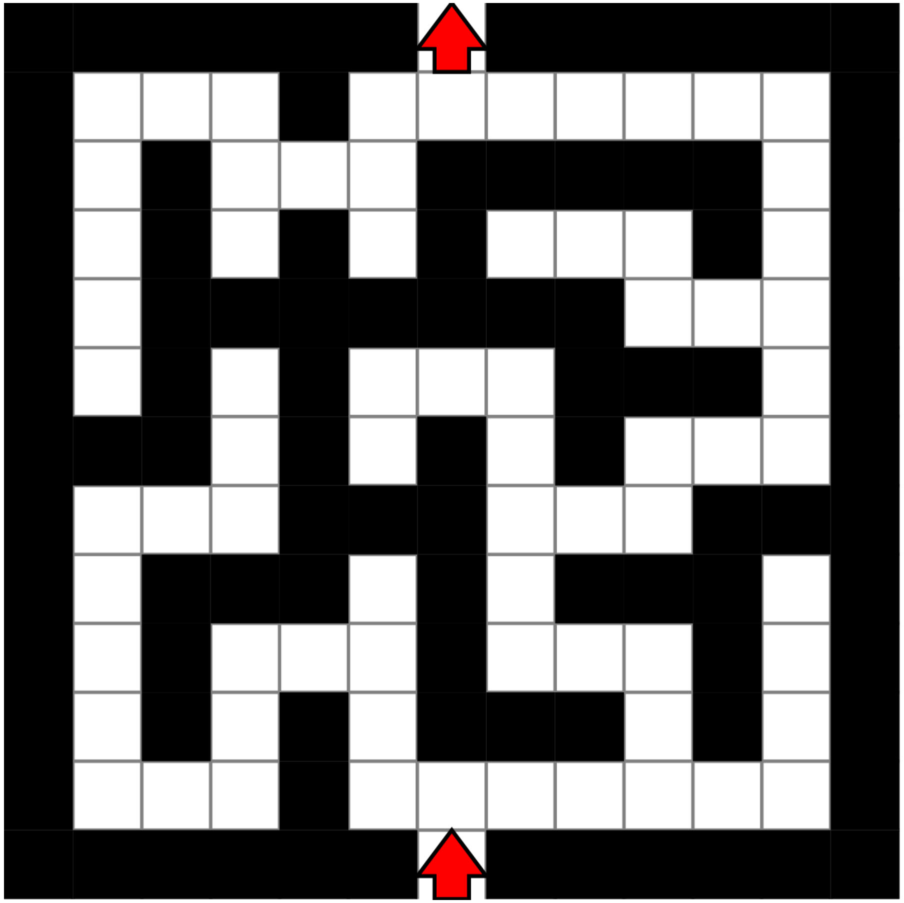
Busca em Largura

```
BFS(G, s)
    for cada vértice u ∈ V[G] − {s}
        u.cor = BRANCO
        u.d = ∞
        u.π = NIL
    s.cor = CINZENTO
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for cada v = Adj[u]
            if v.cor == BRANCO
                v.cor == CINZENTO
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.cor = PRETO
```

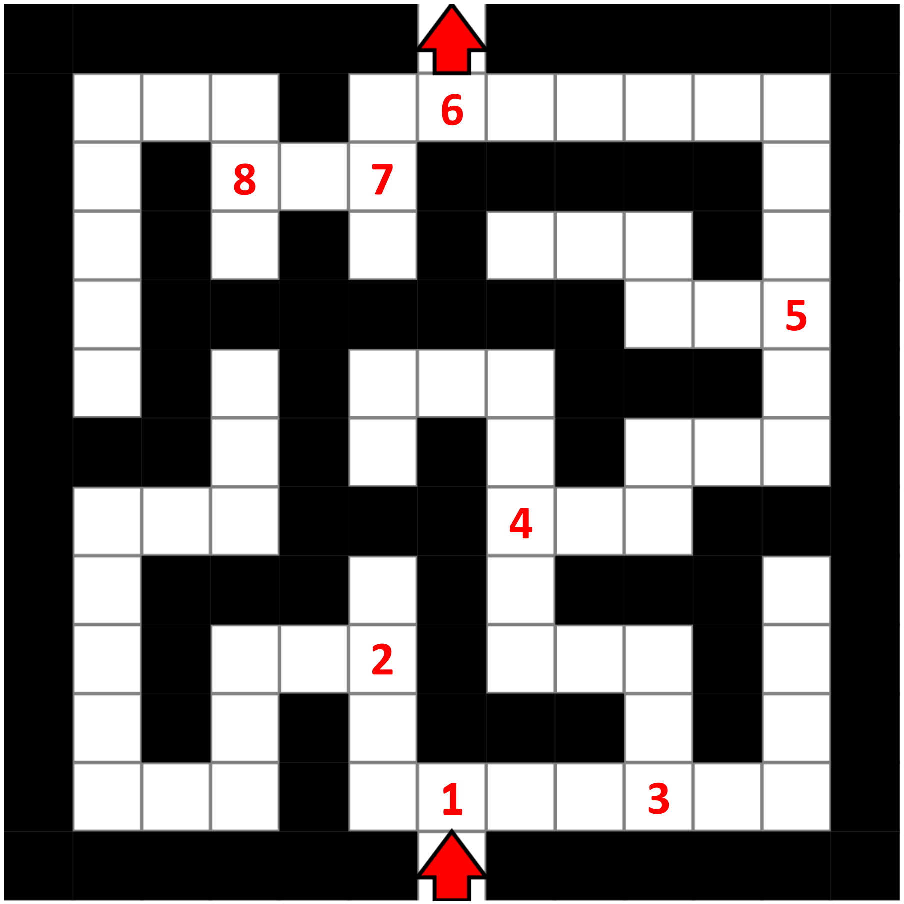
Busca em Largura

- Normalmente, demora muito tempo e sobretudo ocupa muito espaço:
  - É necessário manter um conjunto de nós candidatos alternativos e não apenas um único (profundidade)
- É um método bem melhor para situações em que a árvore pode ter caminhos profundos mas o nó objetivo estiver no nível mais raso da árvore
- Não funciona bem quando o fator de ramificação da árvore é muito alto (tamanho da fila pode aumentar bastante)

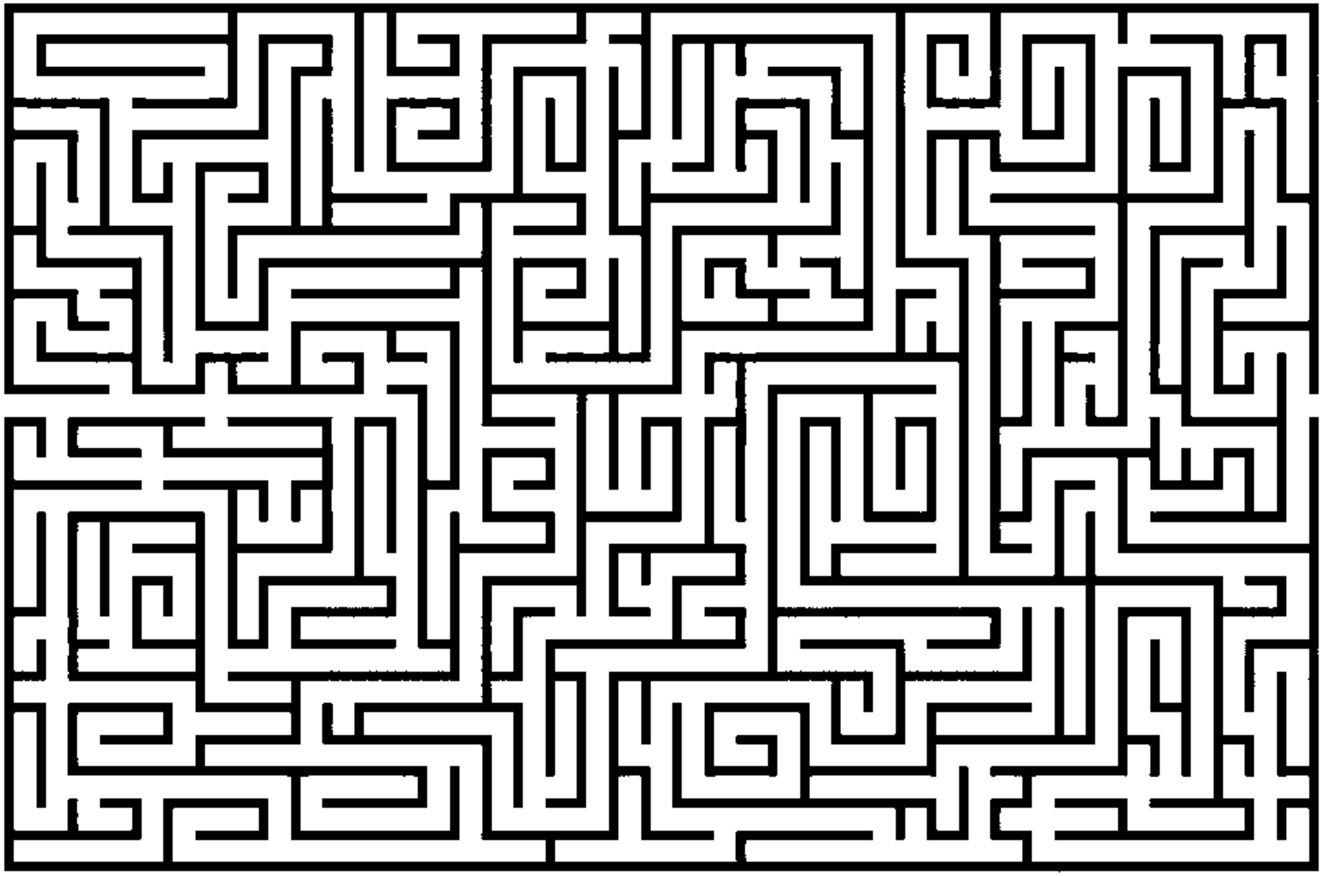
Busca em Largura



Busca em Largura



Busca em Largura



Busca em Largura

- Não tem bom resultado em árvores onde todos os caminhos que levam a um nó objetivo têm comprimentos parecidos:
  - Busca em Profundidade encontraria o objetivo no primeiro caminho examinado
- Maior vantagem: encontra o menor caminho do nó inicial até o nó objetivo (Ford-Fulkerson)
- Sobre a busca em largura:
  - É admissível
  - É completa

DFS vs BFS

Cenário	Busca em Profundidade	Busca em Largura
Alguns caminhos são muito longos ou mesmo infinitos	Funciona mal	Funciona bem
Todos os caminhos têm comprimentos parecidos	Funciona bem	Funciona bem
Todos os caminhos têm comprimentos parecidos e todos os caminhos levam a um estado objetivo	Funciona bem	Desperdício de tempo e memória
Alto fator de ramificação	O desempenho depende de outros fatores	Funciona precariamente

unioeste
Universidade Estadual do Oeste do Paraná

Heurísticas

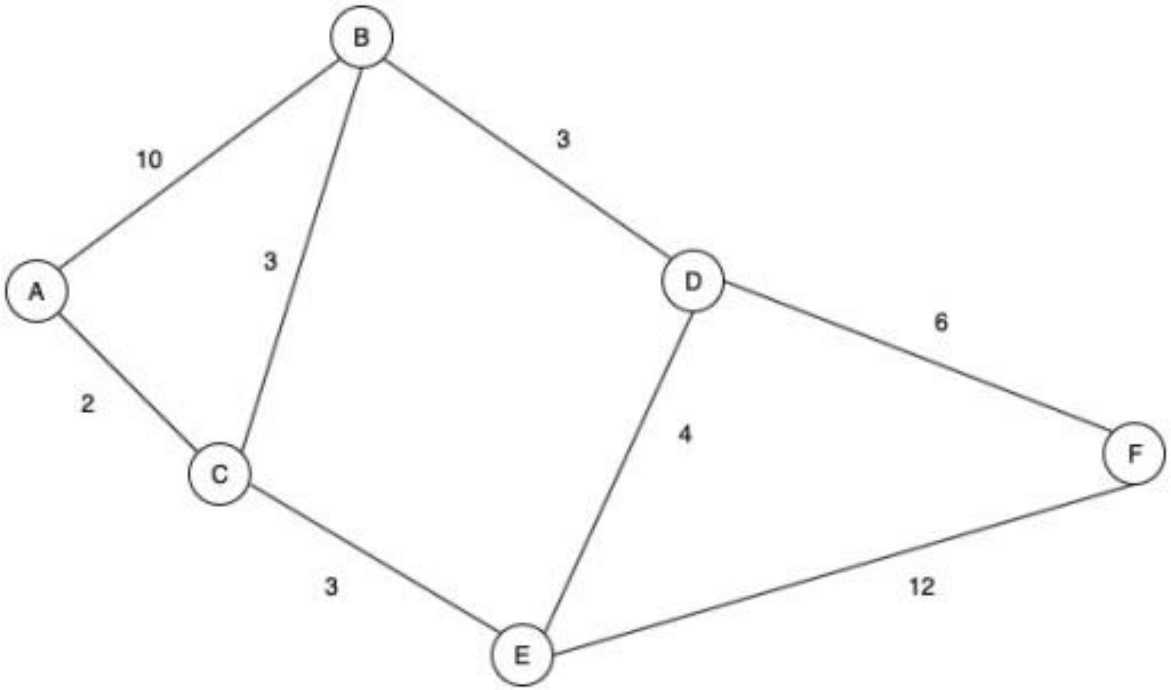
- Método de investigação baseado na aproximação progressiva de um dado problema
- É a utilização de informações que indicam melhor qual caminho a seguir
- Quanto melhor a heurística for, menos nós ela precisará examinar na árvore
- Uma heurística é admissível se ela nunca superestima a distância verdadeira entre um nó e o objetivo

Problema do menor caminho

- Será um exemplo para mostrar as diferenças entre os métodos de buscas (cegos e heurísticos)

- Na figura, cada nó representa uma cidade e as arestas entre os estes representam as estradas que ligam as cidades. O peso das arestas referem-se ao comprimento da rodovia que liga as duas cidades.

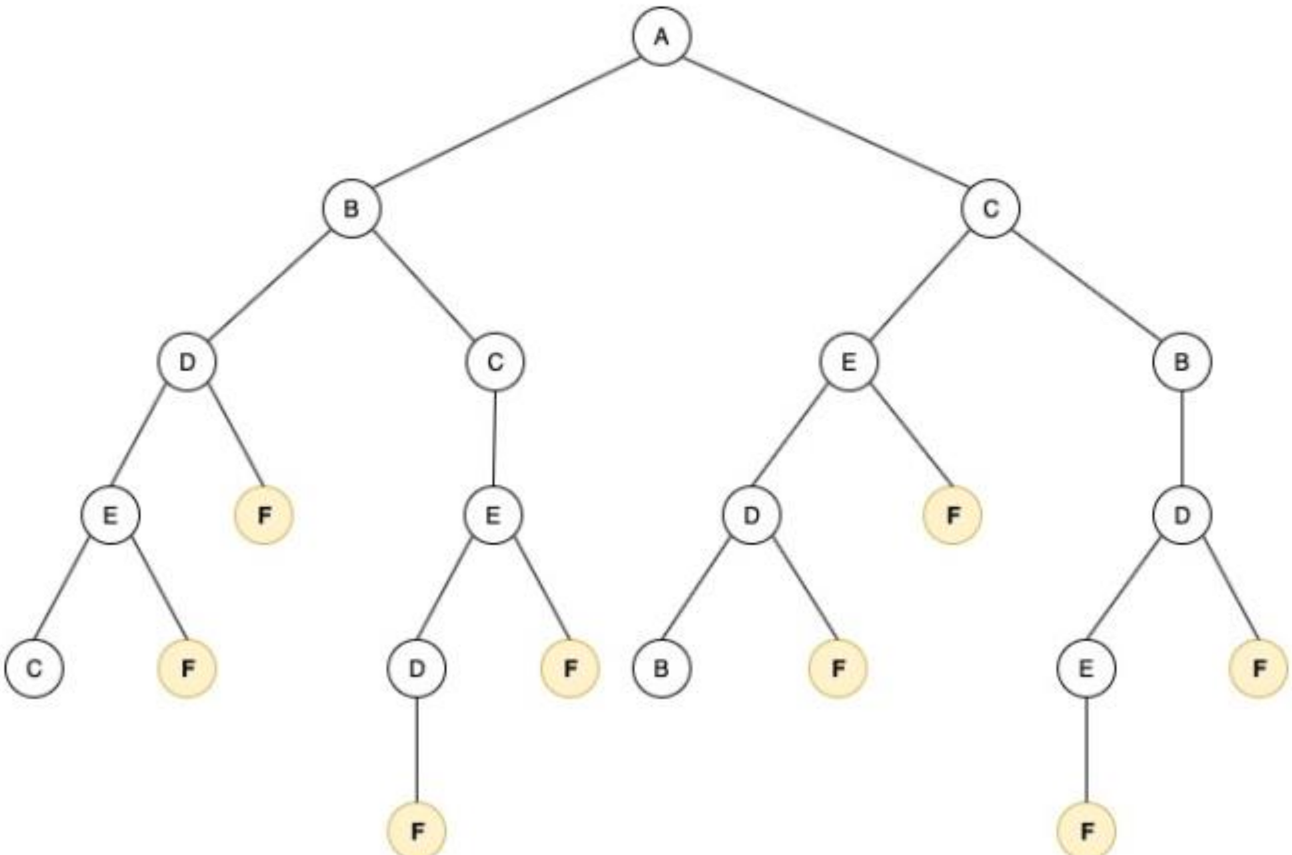
Problema do menor caminho



Problema do menor caminho

- O problema: encontrar o caminho mais curto possível entre duas cidades
- Para mostrar os caminhos obtidos pelos métodos de busca:
  - Nó inicial: A
  - Nó objetivo: F
- O espaço de busca do problema é representado como uma árvore de busca

Problema do menor caminho



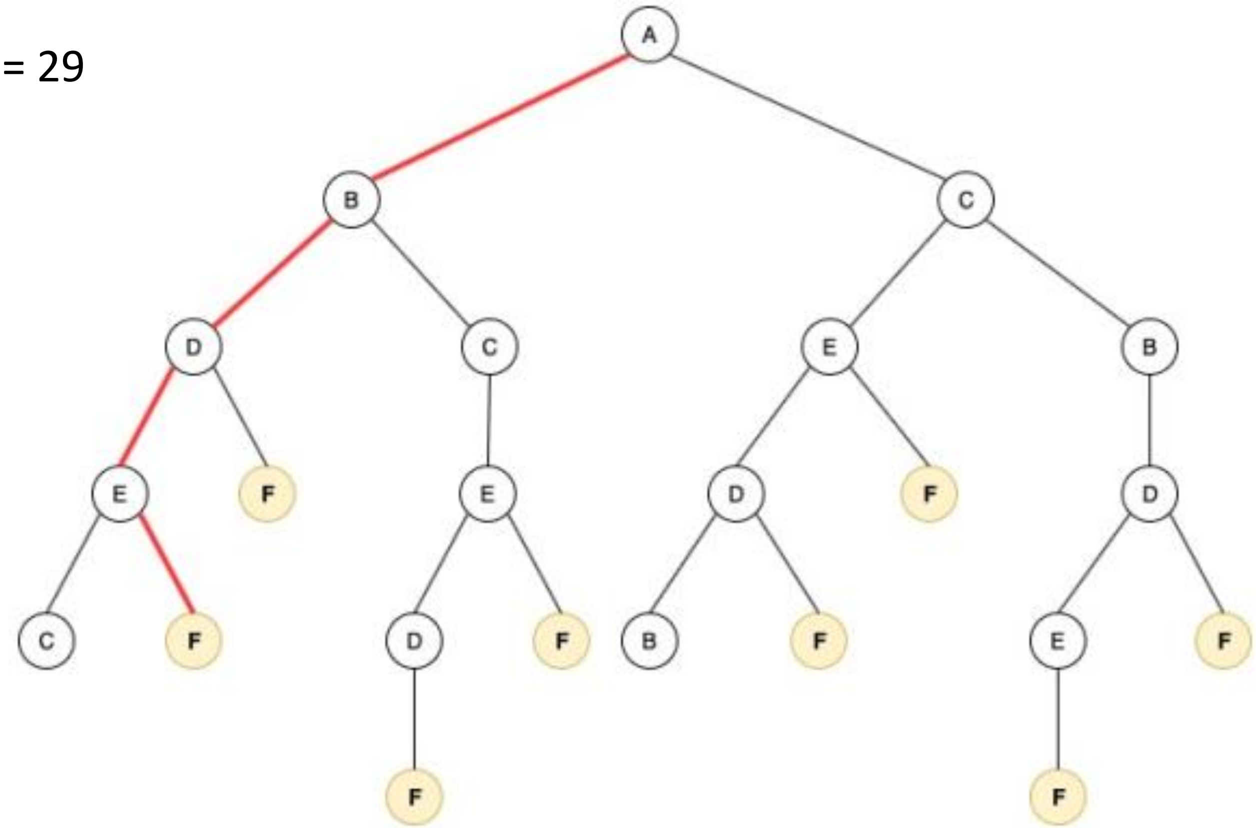
$$\overline{AB} = 10$$
$$\overline{AC} = 2$$
$$\overline{BC} = 3$$
$$\overline{BD} = 3$$
$$\overline{CE} = 3$$
$$\overline{DE} = 4$$
$$\overline{DF} = 6$$
$$\overline{EF} = 12$$

Problema do menor caminho

- O problema possui 8 caminhos possíveis entre o ponto de origem e o destino buscado

Problema do menor caminho

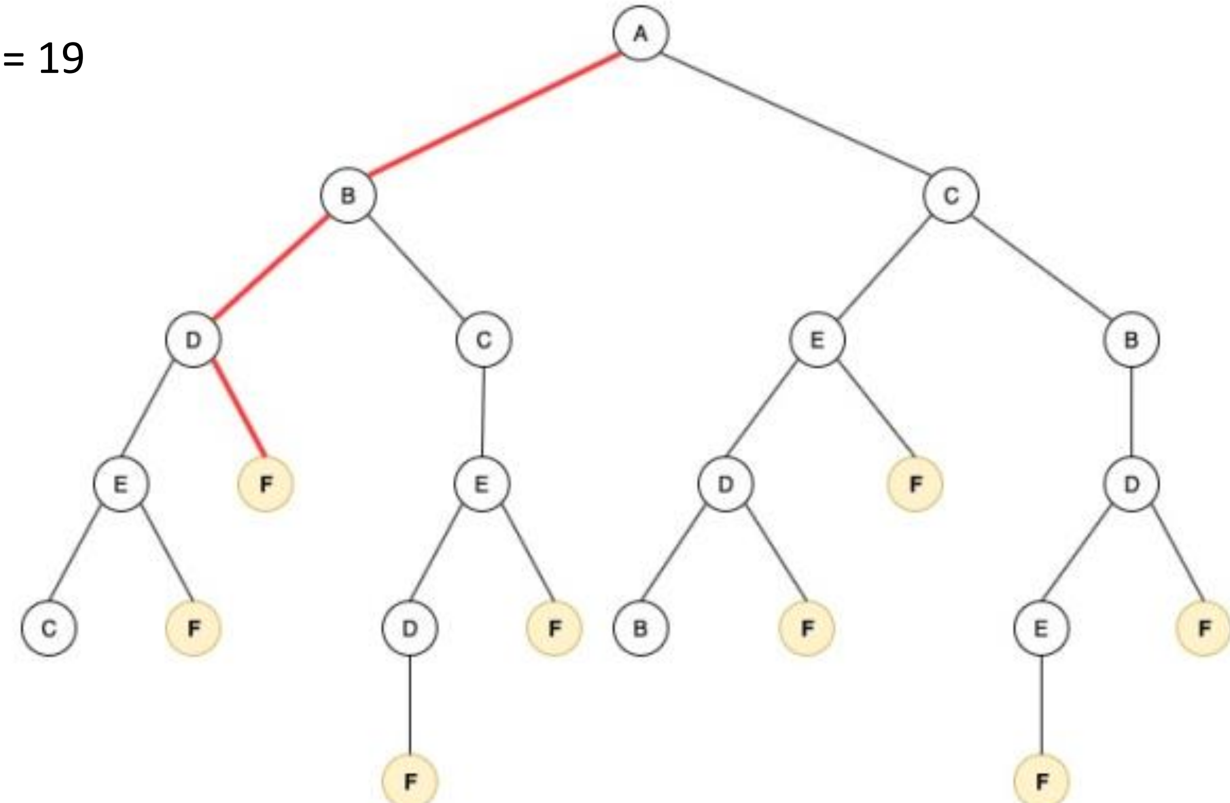
Custo = 29



$$\overline{AB} = 10$$
$$\overline{AC} = 2$$
$$\overline{BC} = 3$$
$$\overline{BD} = 3$$
$$\overline{CE} = 3$$
$$\overline{DE} = 4$$
$$\overline{DF} = 6$$
$$\overline{EF} = 12$$

Problema do menor caminho

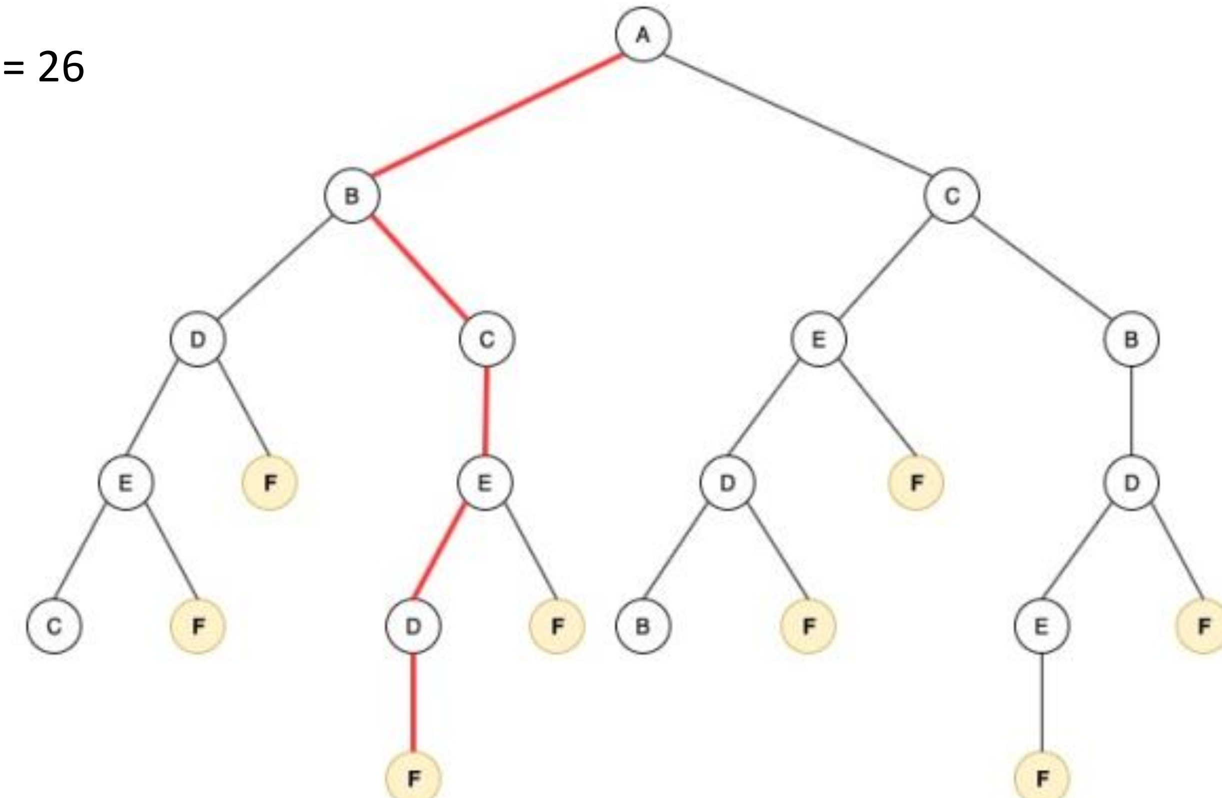
Custo = 19



$\overline{AB} = 10$
$\overline{AC} = 2$
$\overline{BC} = 3$
$\overline{BD} = 3$
$\overline{CE} = 3$
$\overline{DE} = 4$
$\overline{DF} = 6$
$\overline{EF} = 12$

Problema do menor caminho

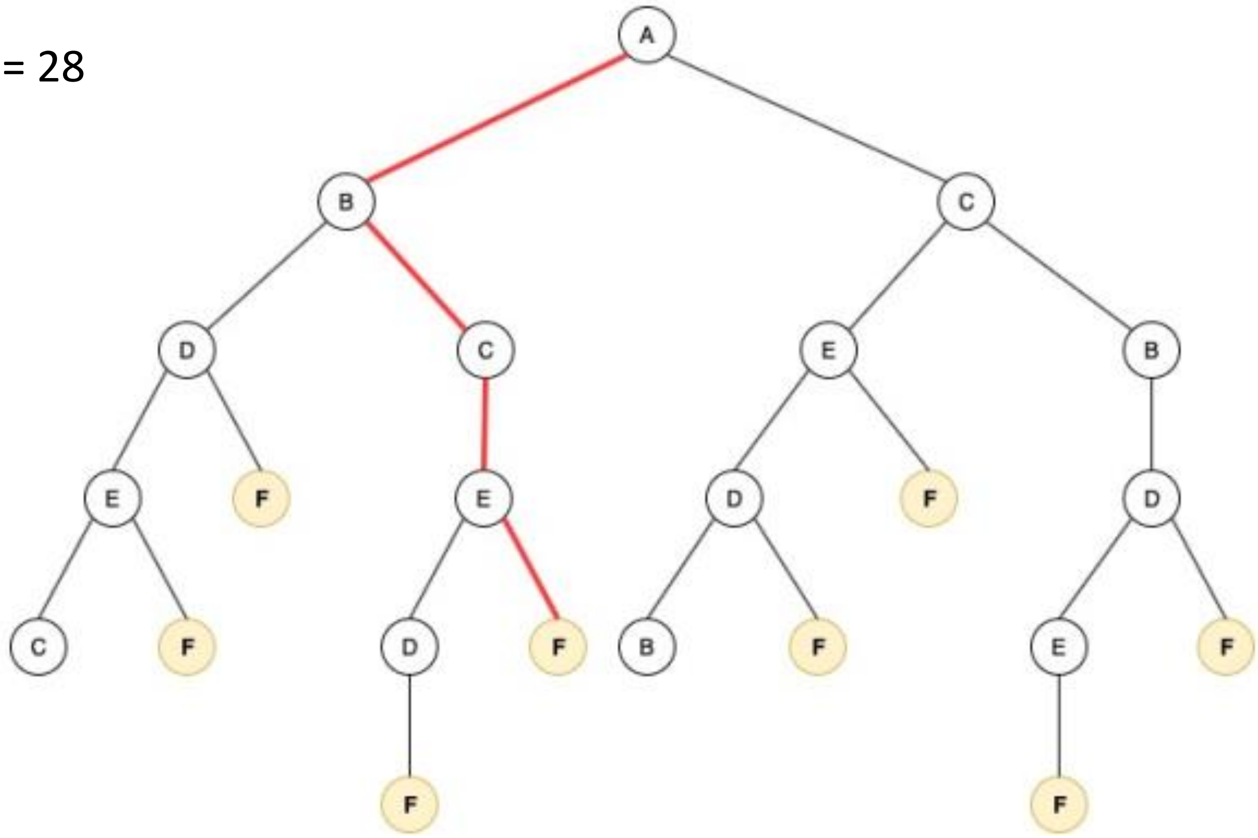
Custo = 26



$$\overline{AB} = 10$$
$$\overline{AC} = 2$$
$$\overline{BC} = 3$$
$$\overline{BD} = 3$$
$$\overline{CE} = 3$$
$$\overline{DE} = 4$$
$$\overline{DF} = 6$$
$$\overline{EF} = 12$$

Problema do menor caminho

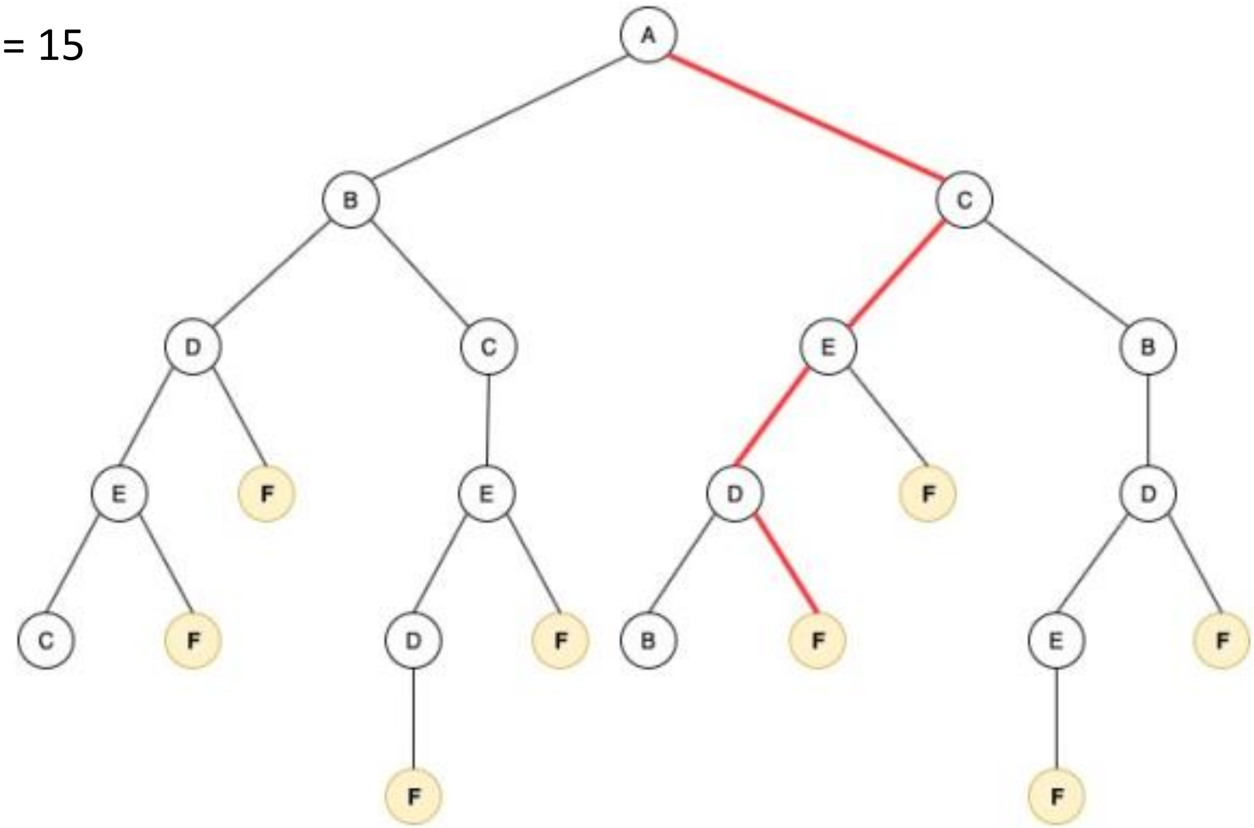
Custo = 28



$$\overline{AB} = 10$$
$$\overline{AC} = 2$$
$$\overline{BC} = 3$$
$$\overline{BD} = 3$$
$$\overline{CE} = 3$$
$$\overline{DE} = 4$$
$$\overline{DF} = 6$$
$$\overline{EF} = 12$$

Problema do menor caminho

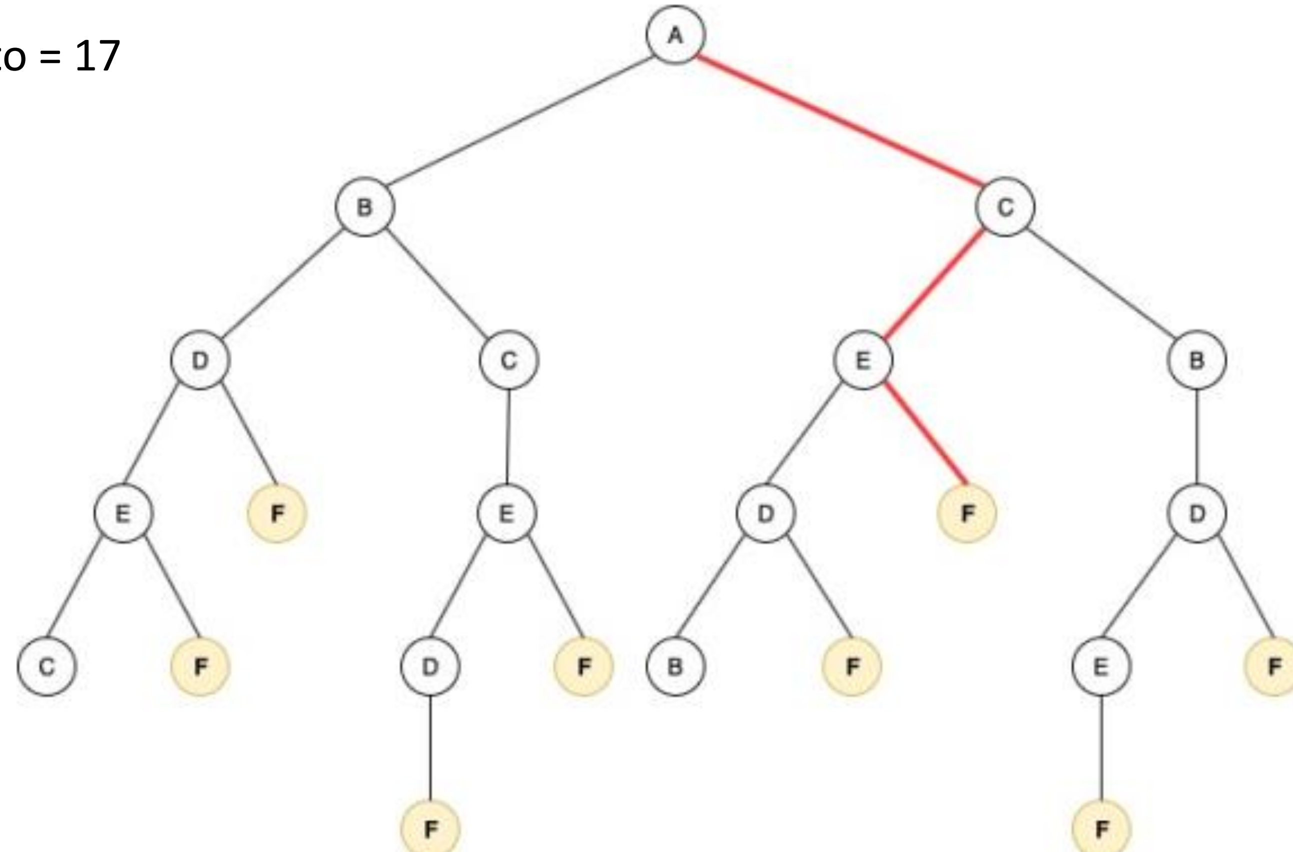
Custo = 15



$\overline{AB} = 10$
$\overline{AC} = 2$
$\overline{BC} = 3$
$\overline{BD} = 3$
$\overline{CE} = 3$
$\overline{DE} = 4$
$\overline{DF} = 6$
$\overline{EF} = 12$

Problema do menor caminho

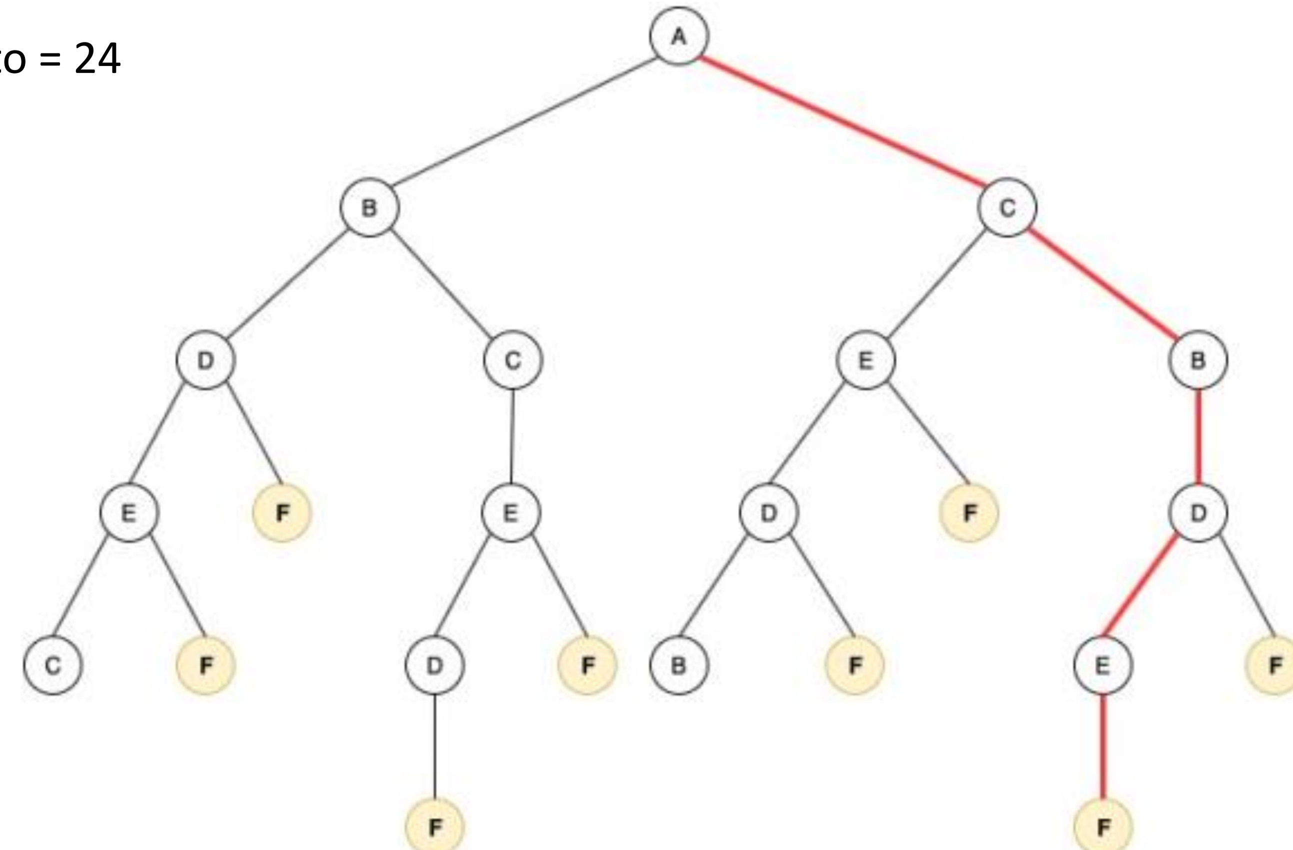
Custo = 17



$\overline{AB} = 10$
$\overline{AC} = 2$
$\overline{BC} = 3$
$\overline{BD} = 3$
$\overline{CE} = 3$
$\overline{DE} = 4$
$\overline{DF} = 6$
$\overline{EF} = 12$

Problema do menor caminho

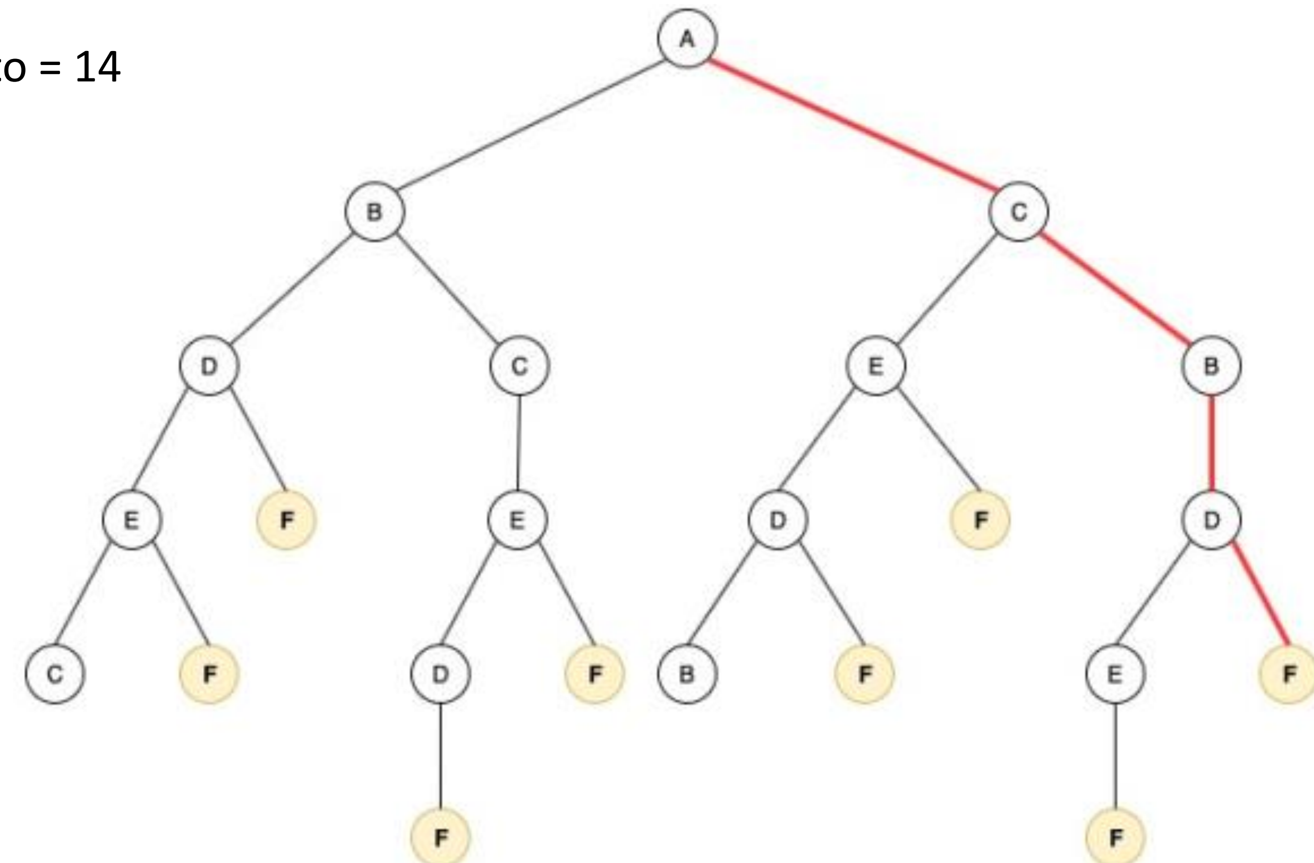
Custo = 24



$$\overline{AB} = 10$$
$$\overline{AC} = 2$$
$$\overline{BC} = 3$$
$$\overline{BD} = 3$$
$$\overline{CE} = 3$$
$$\overline{DE} = 4$$
$$\overline{DF} = 6$$
$$\overline{EF} = 12$$

Problema do menor caminho

Custo = 14



$\overline{AB} = 10$
$\overline{AC} = 2$
$\overline{BC} = 3$
$\overline{BD} = 3$
$\overline{CE} = 3$
$\overline{DE} = 4$
$\overline{DF} = 6$
$\overline{EF} = 12$

Problema do menor caminho

- Busca em profundidade
- Busca em largura
- Subida da encosta

Subida da Encosta

- Também conhecida como:
  - Subindo o Morro
  - Subida da Colina
  - Hill Clambing

- Baseada na ideia de Busca em Profundidade

- Muito usada quando há a disponibilidade de uma boa heurística para avaliar estados, mas nenhum outro conhecimento útil está disponível

Subida da Encosta

- O método expande o estado atual da busca e avalia os seus filhos:
  - O “melhor” filho é selecionado para uma expansão futura
- O algoritmo não registra o histórico do processo de subida:
  - Ele não consegue se recuperar de falhas de sua estratégia (não realiza o back-tracking como a busca em profundidade)

Subida da Encosta

- Se move de forma contínua no sentido do valor crescente
- Não examina antecipadamente valores de estados além de seus vizinhos imediatos
- Termina quando alcança um pico, em que nenhum vizinho tem valor mais alto

Subida da Encosta Simples

Algoritmo 4: Subida de Encosta Simples

```
1. Estado corrente <-- estado inicial
2. Repita até que o estado corrente seja a solução ou não existam operadores a serem
      aplicados ao estado corrente
      a) Escolha um operador que ainda não tenha sido aplicado ao estado corrente e
      aplique-o para produzir um novo estado;
      b) Avalie o novo estado:
            i) Se for a solução (um estado meta) retorne-o e encerre.
            ii) Se for melhor que o estado corrente então
                  Estado corrente <-- novo estado.
```

Subida da Encosta mais íngreme

Algoritmo 5: Subida de encosta para a trilha mais íngreme

```
fila <-- []          // inicializa uma fila vazia
estado <-- no_raiz   // inicializa o estado inicial
enquanto (verdadeiro)
   se eh_objetivo(estado)
      retorne SUCESSO
   senão
      ordenar(sucessores(estado))
      inserir_na_frente_da_fila(sucessores(estado))
   se fila == []
      retorne FALHA
   estado <-- fila[0]    // o novo estado é o primeiro item da fila
   remover_primeiro_item_da(fila)
fim_enquanto
```

Subida da Encosta

Problema das 8 Rainhas

Objetivo: posicionar as 8 rainhas de forma que nenhuma ataque outra

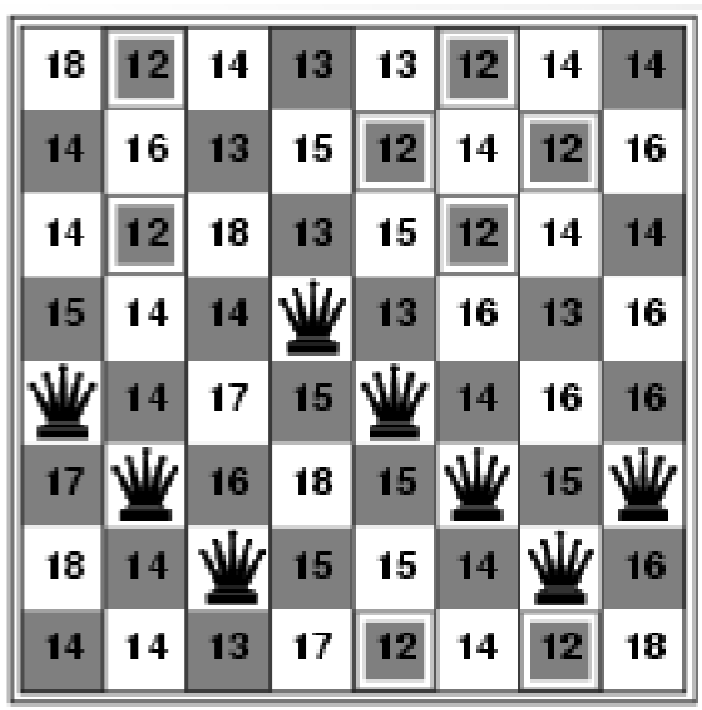
Heurística: número de ataques entre pares de rainhas

Objetivo final: número de ataques = 0

Exploração do espaço de busca dá-se pela mudança de posição de cada uma das rainhas.

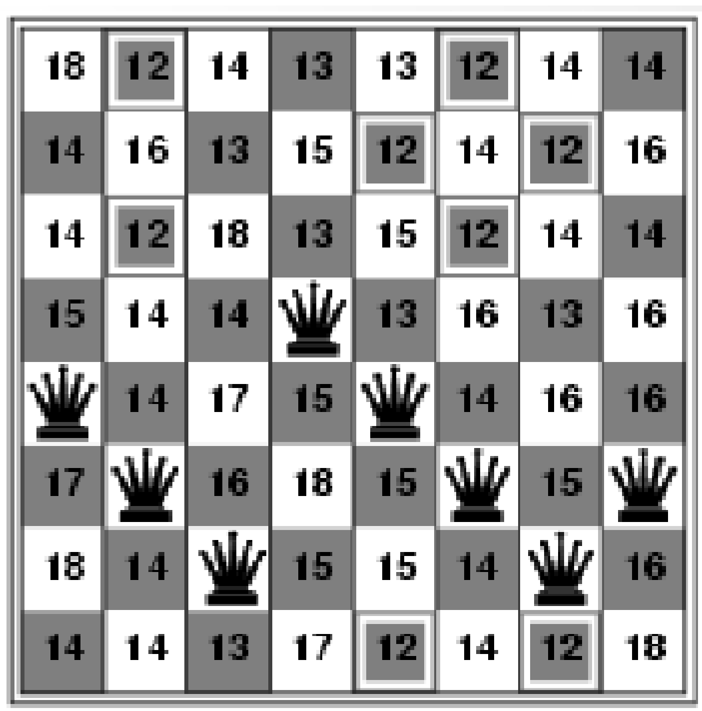
Subida Encosta

Problema das 8 Rainhas



Subida da Encosta

Problema das 8 Rainhas



1,2
1,3
1,5
2,3
2,4
2,6
2,8
3,5
3,7

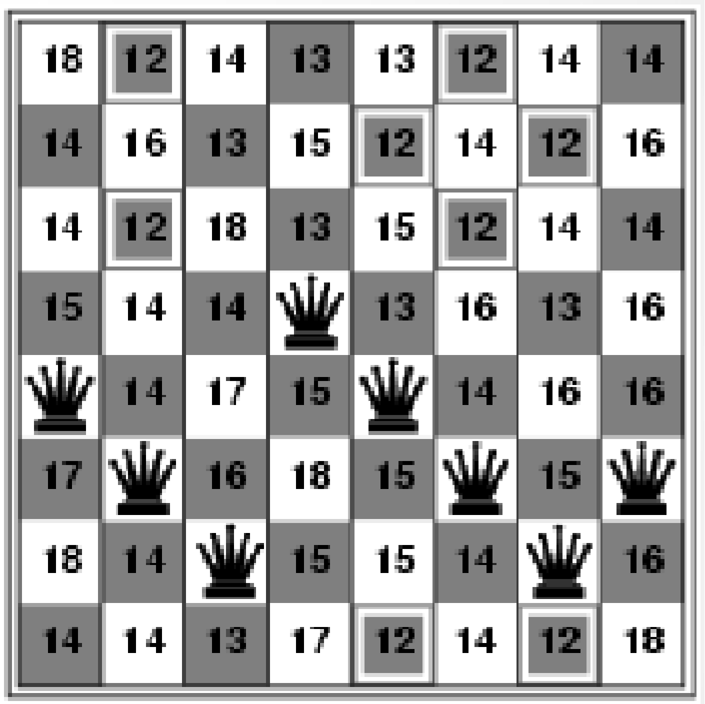
4,5
4,6
4,7
5,6
5,7
6,7
6,8
7,8

H=17

Subida da Encosta

Cada rainha pode ser movida para outras sete linhas. Isso gera uma vizinhança de tamanho 56

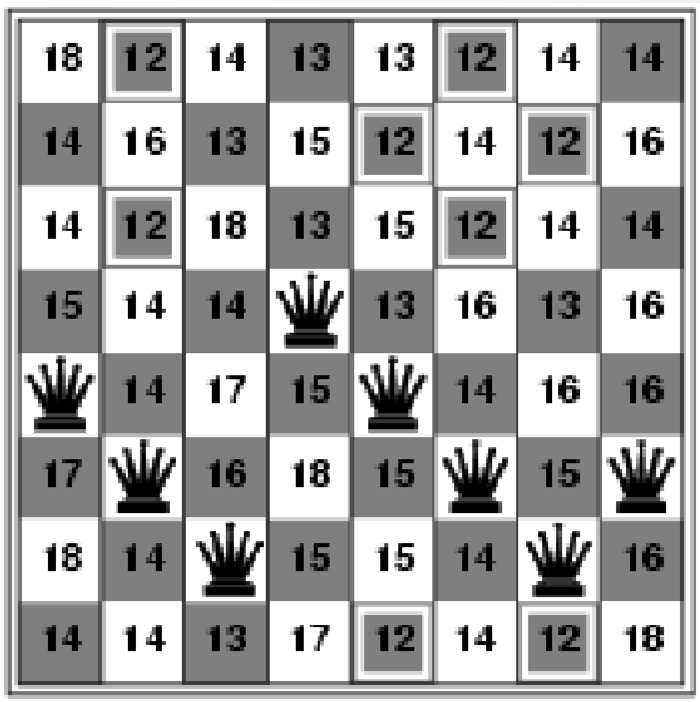
7+7+7+...+7=56



Subida da Encosta

A subida da encosta simples vai pegar a primeira opção que for melhor que o valor atual (H=17).

Nesse caso, seria mover a primeira rainha para a linha 2, o que resultaria em um H=14.

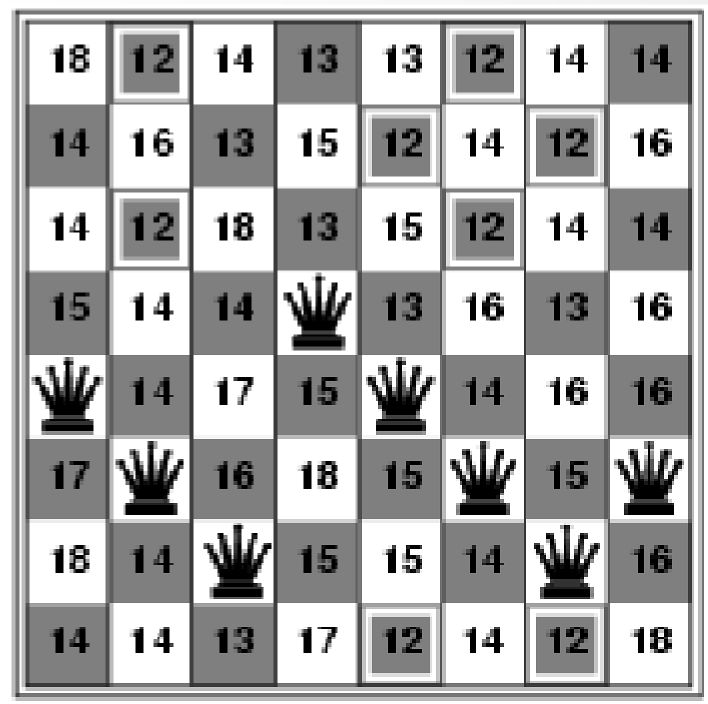


Subida da Encosta

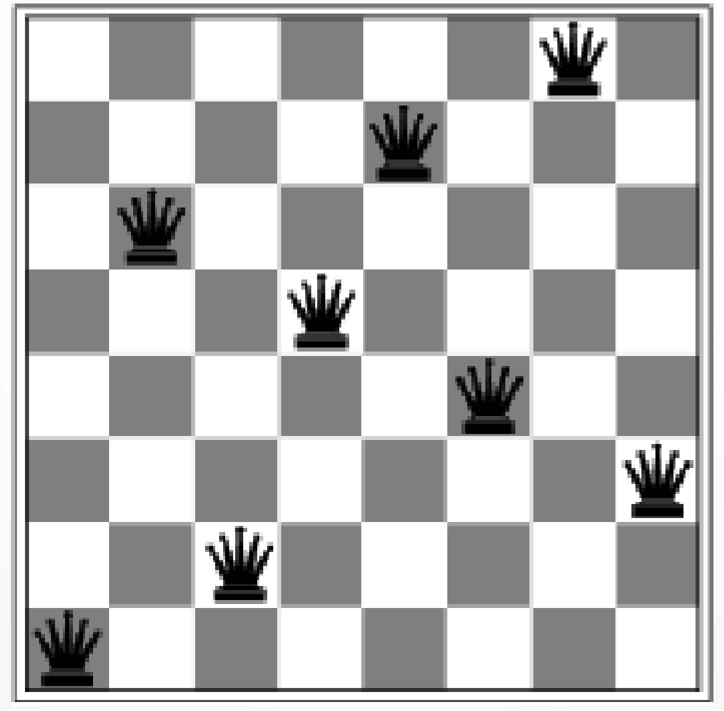
A subida da encosta mais íngreme vai comparar todas as 56 opções e selecionar aquela que levar à melhor solução possível no passo atual

Nesse caso, seria mover a segunda rainha para as linhas 1, 3 ou mover a rainha 5 para as linhas 2, 8 ou mover a rainha 6 para as linhas 1, 3 ou mover a rainha 7 para as linhas 2,8.

Todas as opções resultariam em um H=14.



Subida da Encosta



Subida da Encosta

Problema das 8 Rainhas (FARIA, 2015)

- Usando estados iniciais aleatórios

- Em 86% das vezes busca fica paralisada. Ou seja, resolve apenas 14% das instâncias do problema

- Entretanto, a busca é rápida:
  - 4 passos em média quando tem sucesso
  - 3 passos em média quando fica paralisada
  - Em espaço que tem cerca de 17 milhões de estados

Subida da Encosta

Problema da Convergência Prematura:

- Mudar de estado em quando encontrar um platô
  - Quando estados têm avaliações iguais
  - Colocando um limite de vezes

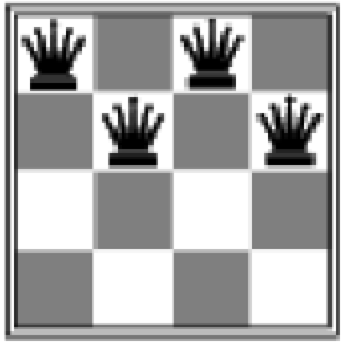
Subida da Encosta

Problema das 8 Rainhas (FARIA, 2015)

- Usando estados iniciais aleatórios
- Usando mudança de estado em platôs
- Passa a resolver 94% das instâncias do problema (14% → 94% 👍)
- Esta abordagem, no entanto, demora mais:
  - 21 passos em média quando tem sucesso
  - 64 passos em média quando falha

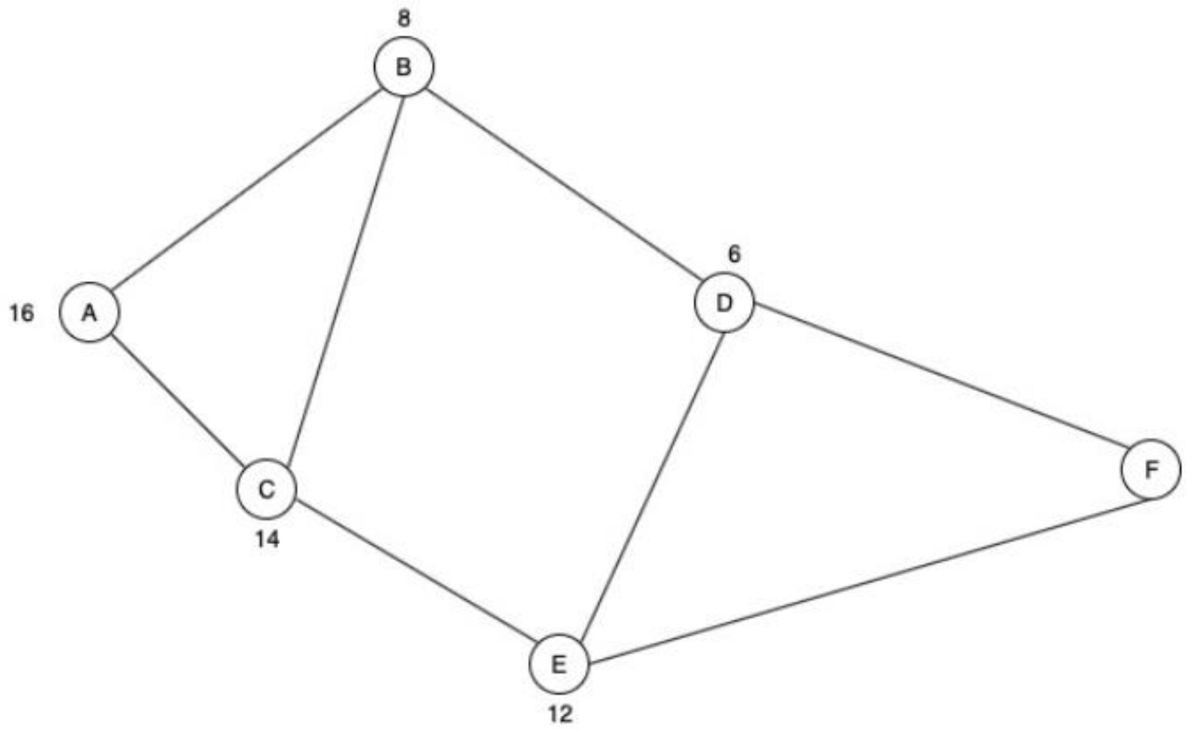
Subida da Encosta

Problema das 4 Rainhas



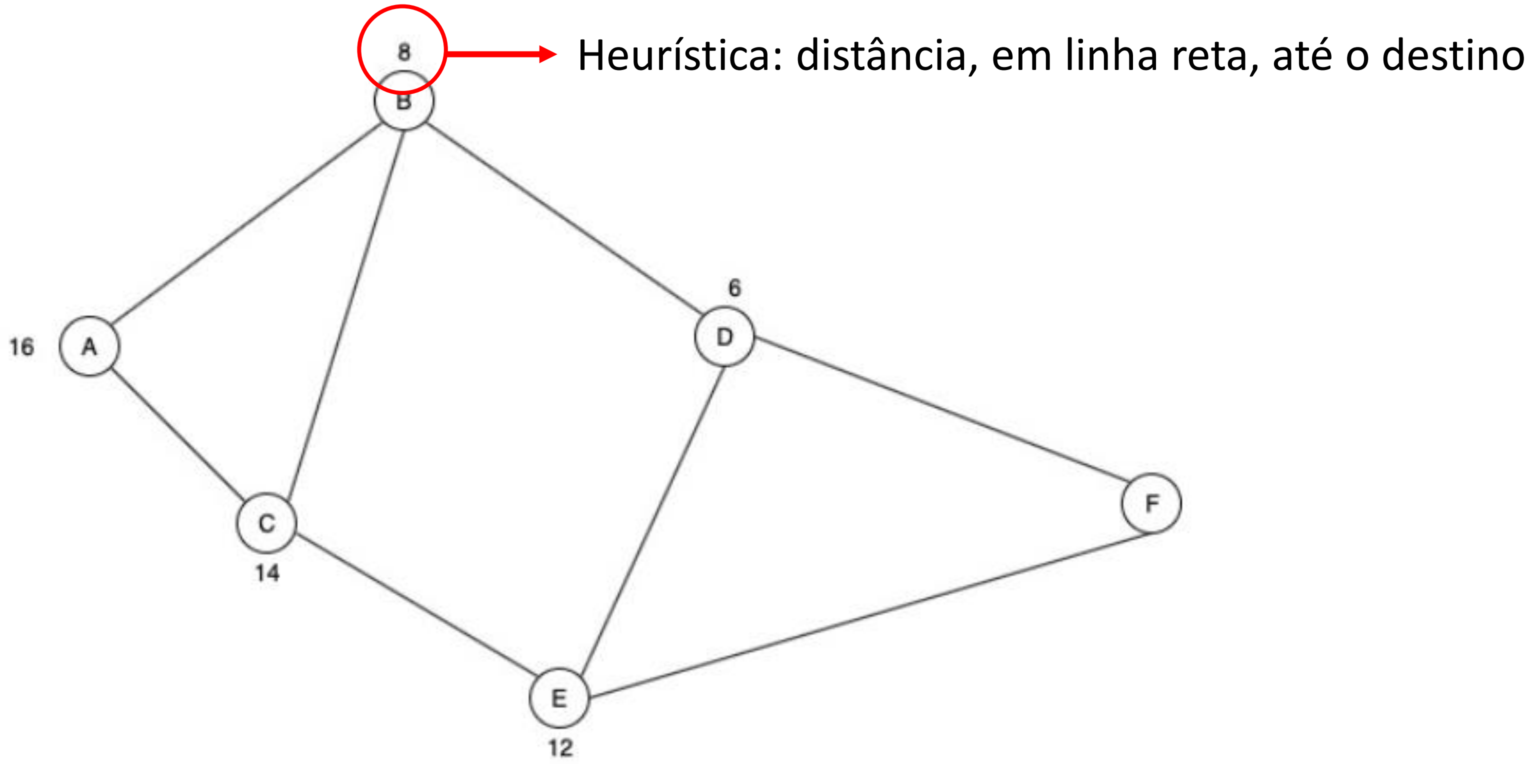
Subida da Encosta

- Menor caminho entre dois pontos



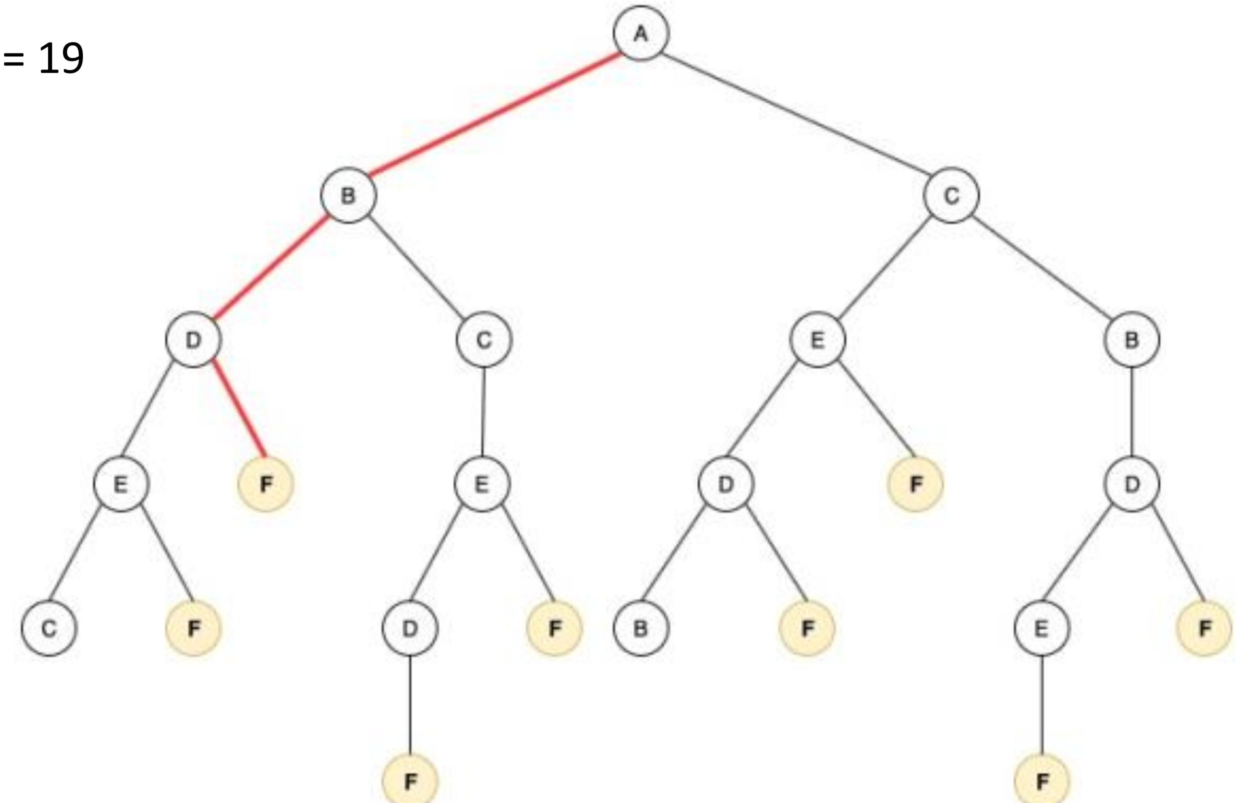
Subida da Encosta

- Menor caminho entre dois pontos



Problema do menor caminho

Custo = 19



$$\overline{AB} = 10$$
$$\overline{BD} = 3$$
$$\overline{DF} = 6$$

Busca pelo Primeiro Melhor

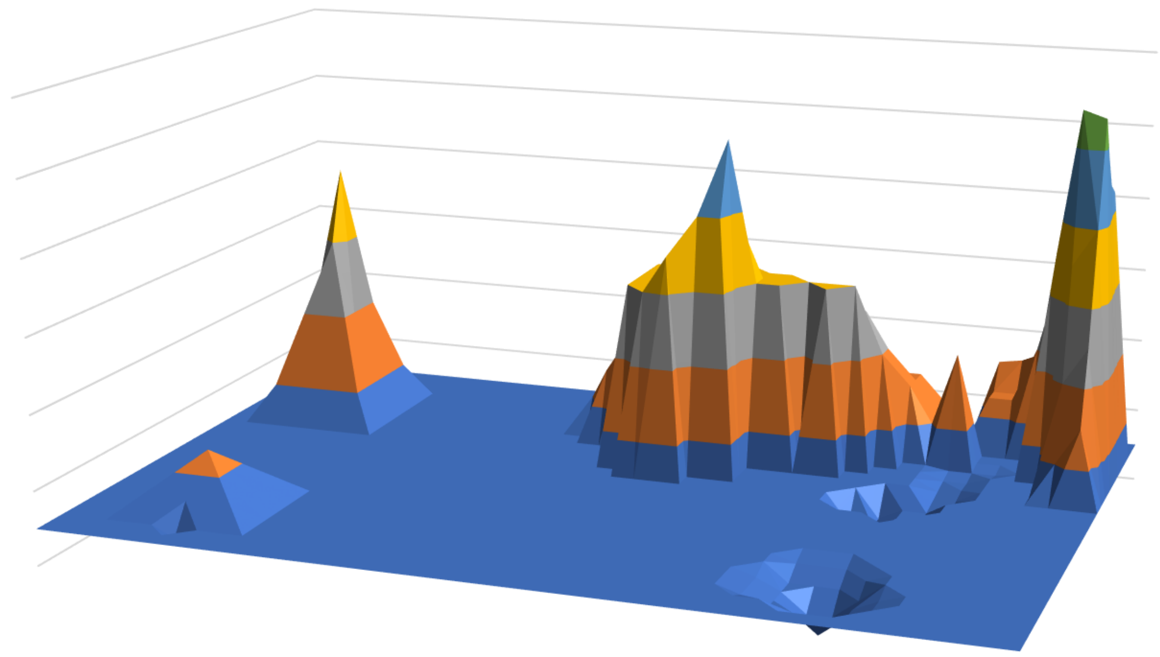
- Busca o-melhor-primeiro ou Busca Pela Melhor Escolha ou Best-First Search (BFS)
- O método A* é derivado deste
- Parecida com Subida de Encosta
- Considera todas as informações disponíveis até aquele instante, não apenas as da última expansão

Busca pelo Primeiro Melhor

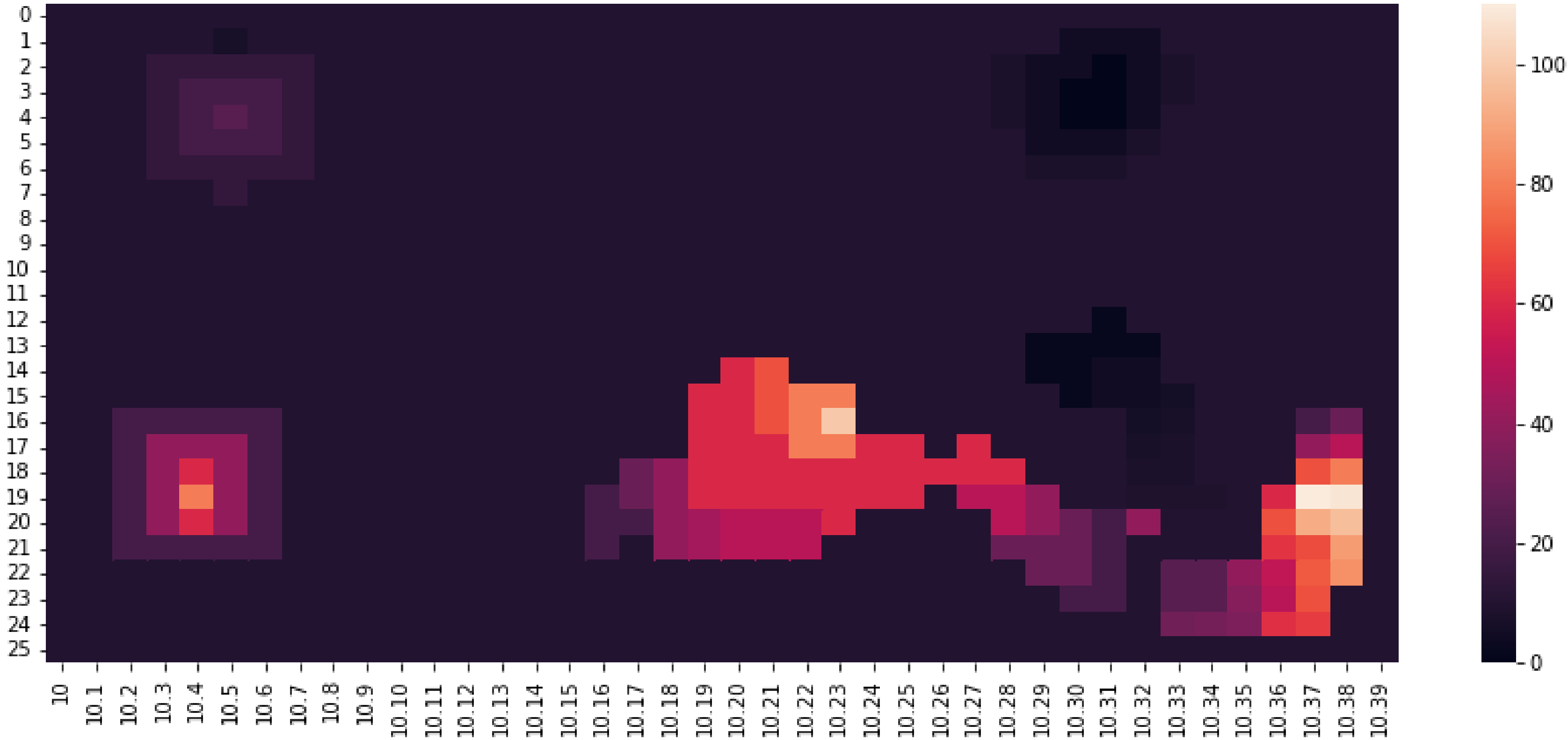
Algoritmo 6: Busca pelo Primeiro Melhor

```
fila <-- []          // inicializa uma fila vazia
estado <-- no_raiz   // inicializa o estado inicial
enquanto (verdadeiro)
   se eh_objetivo(estado)
      retorne SUCESSO
   senão
      inserir_na_frente_da_fila(sucessores(estado))
      ordenar(fila)
   se fila == []
      retorne FALHA
   estado <-- fila[0]    // o novo estado é o primeiro item da fila
   remover_primeiro_item_da(fila)
fim_enquanto
```

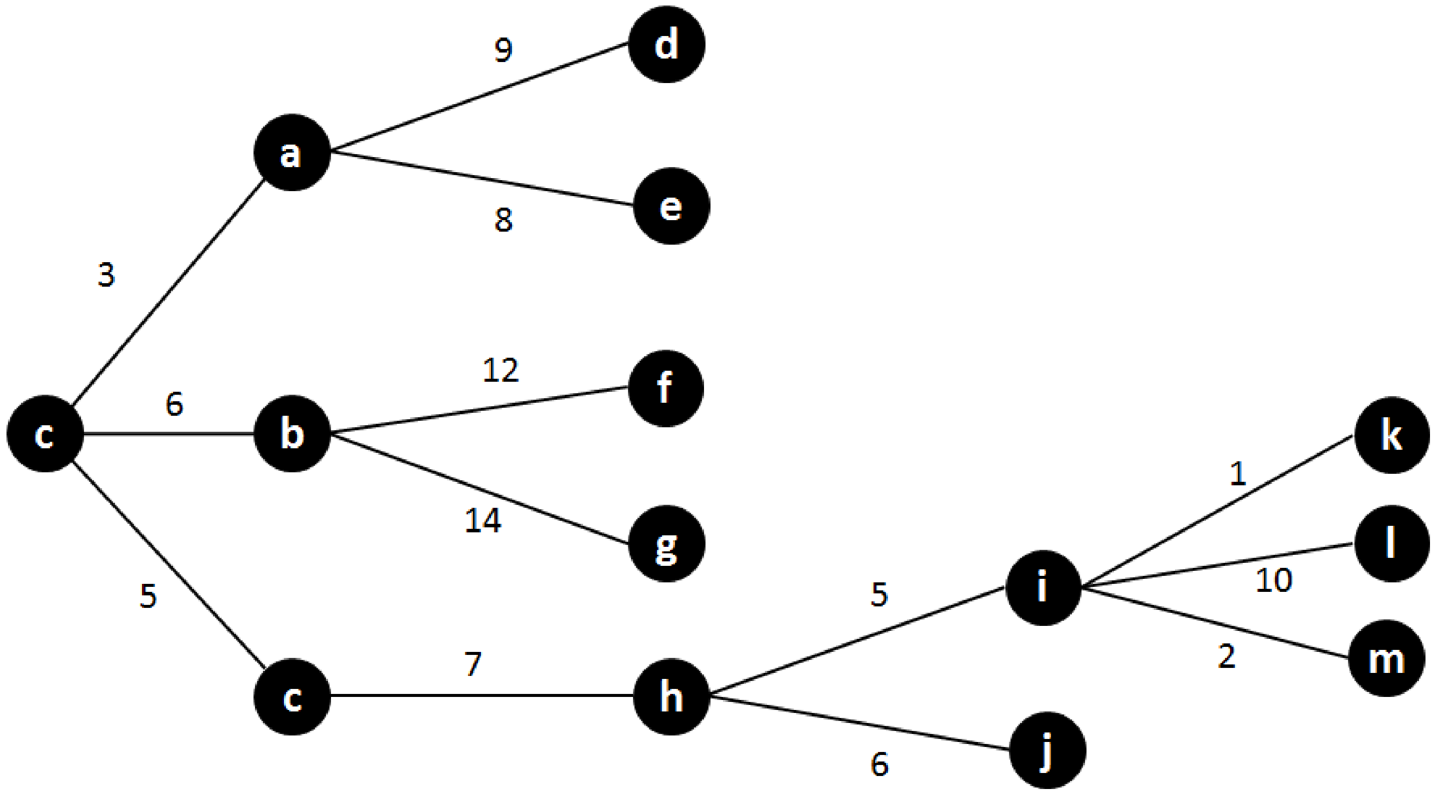
Busca pelo Primeiro Melhor



Busca pelo Primeiro Melhor



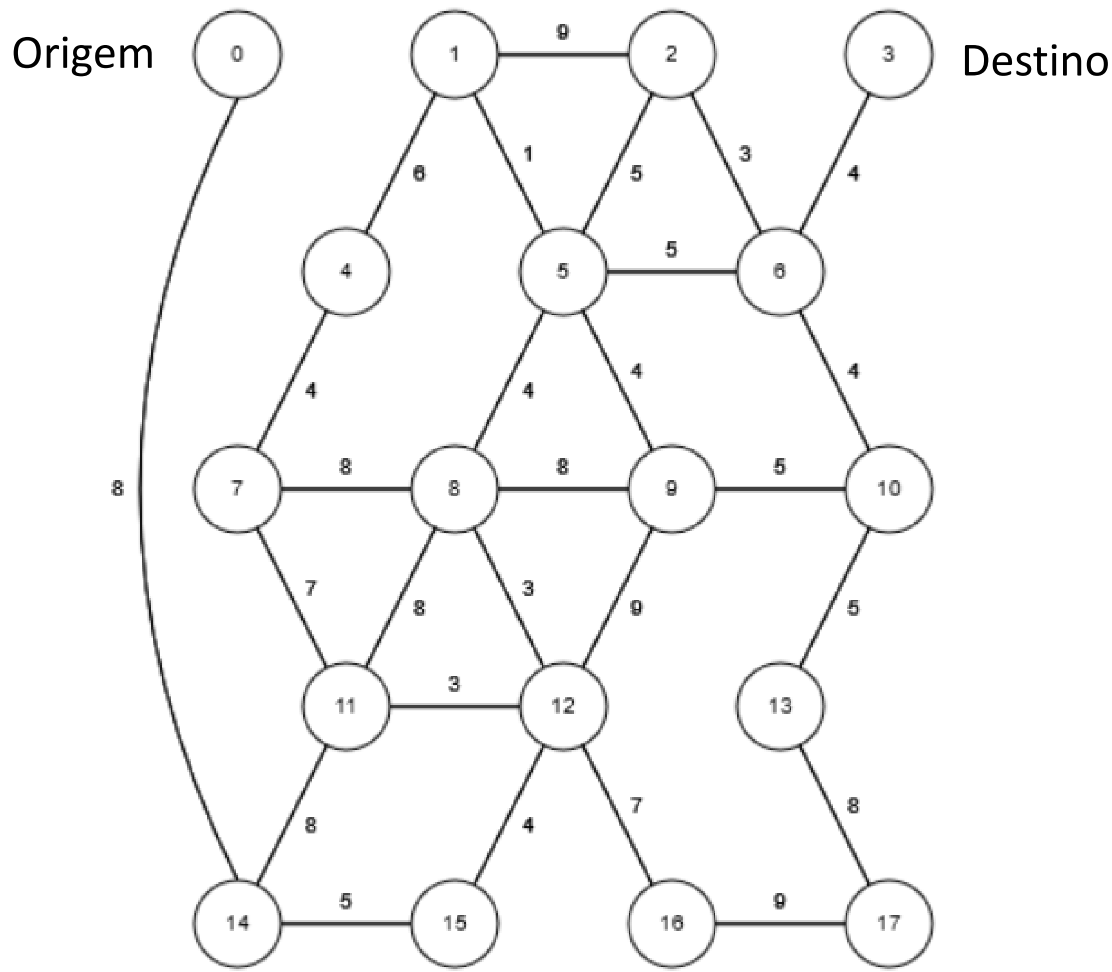
Busca pelo Primeiro Melhor



Busca pelo Primeiro Melhor

10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	7	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	5	5	5	10	10	10	10	10	10	10
10	10	10	15	15	15	15	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	5	5	0	5	8	10	10	10	10	10	10
10	10	10	15	20	20	20	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	5	0	0	5	8	10	10	10	10	10	10
10	10	10	15	20	25	20	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	5	0	0	5	10	10	10	10	10	10	10
10	10	10	15	20	20	20	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	5	5	5	8	10	10	10	10	10	10	10
10	10	10	15	15	15	15	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	8	8	10	10	10	10	10	10	10	10
10	10	10	10	10	15	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	2	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	2	2	2	2	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	60	70	10	10	10	10	10	10	10	2	2	5	5	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	60	60	70	80	80	10	10	10	10	10	10	2	5	5	6	10	10	10	10	10	10
10	10	20	20	20	20	20	10	10	10	10	10	10	10	10	10	10	10	10	60	60	70	80	100	10	10	10	10	10	10	10	10	6	7	10	10	10	20	30	10
10	10	20	40	40	40	20	10	10	10	10	10	10	10	10	10	10	10	10	60	60	60	80	80	60	60	10	60	10	10	10	10	7	8	10	10	10	40	50	10
10	10	20	40	60	40	20	10	10	10	10	10	10	10	10	10	10	30	40	60	60	60	60	60	60	60	60	60	60	10	10	10	8	8	10	10	10	70	80	10
10	10	20	40	80	40	20	10	10	10	10	10	10	10	10	10	20	30	40	60	60	60	60	60	60	60	10	50	50	40	10	10	9	9	9	10	60	110	108	10
10	10	20	40	60	40	20	10	10	10	10	10	10	10	10	10	20	20	40	45	50	50	50	60	10	10	10	10	50	40	30	20	40	10	10	10	70	92	97	10
10	10	20	20	20	20	20	10	10	10	10	10	10	10	10	10	20	10	40	45	50	50	50	10	10	10	10	10	30	30	30	20	10	10	10	10	63	69	88	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	30	30	20	10	25	25	40	52	72	85	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	20	20	10	25	25	37	50	70	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	32	33	35	62	65	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Busca pelo Primeiro Melhor



Busca com Limite Superior

- É uma forma de busca em largura, mas que usa uma heurística
- Utiliza um limiar, de tal modo que apenas os poucos melhores caminhos são considerados a cada nível
- É eficiente na utilização de memória

Busca com Limite Superior

- Especialmente útil para explorar um espaço de busca que tenha um elevado fator de ramificação
- <u>Desvantagem</u>: não realiza uma busca exaustiva na árvore e pode falhar até em achar um nó objetivo

Busca com Limite Superior

Algoritmo 7: Busca com Limite Superior

```
fila <-- []          // inicializa uma fila vazia
estado <-- no_raiz   // inicializa o estado inicial
enquanto (verdadeiro)
   se eh_objetivo(estado)
      retorne SUCESSO
   senão
      inserir_no_final_da_fila(sucessores(estado))
      selecionar_melhores_caminhos(fila, n)   // n é o limite (limiar)
      // remove todos os caminhos, menos os n melhores
   se fila == []
      retorne FALHA
   estado <-- fila[0]   // o novo estado é o primeiro item da fila
   remover_primeiro_item_da(fila)
fim_enquanto
```

A*

- É similar ao busca pelo primeiro melhor. Entretanto, utiliza uma heurística um pouco mais complexa
- Procura evitar expandir nós que já são “custosos”. Para isso leva em conta a custo já computado para cada possível solução
- Utiliza a seguinte heurística:

$$f(nó) = g(nó) + h(nó)$$

A*

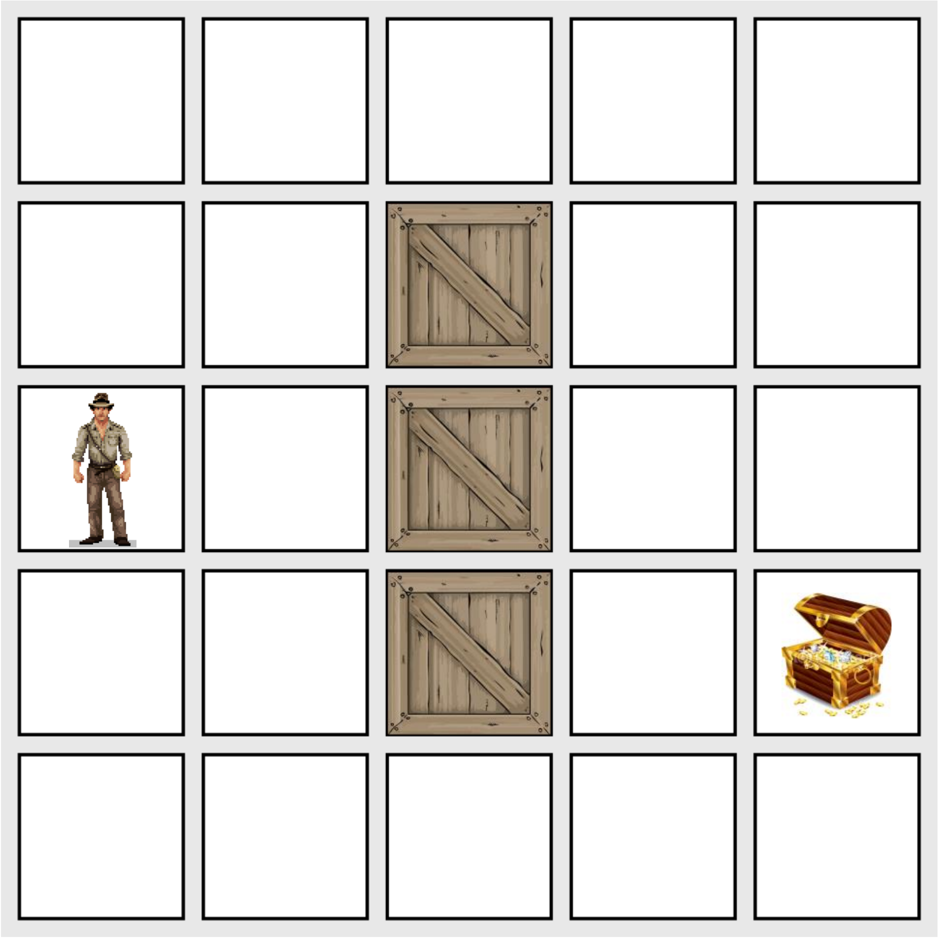
$$f(nó) = g(nó) + h(nó)$$

g(nó): representa o custo do caminho que leva ao nó atual

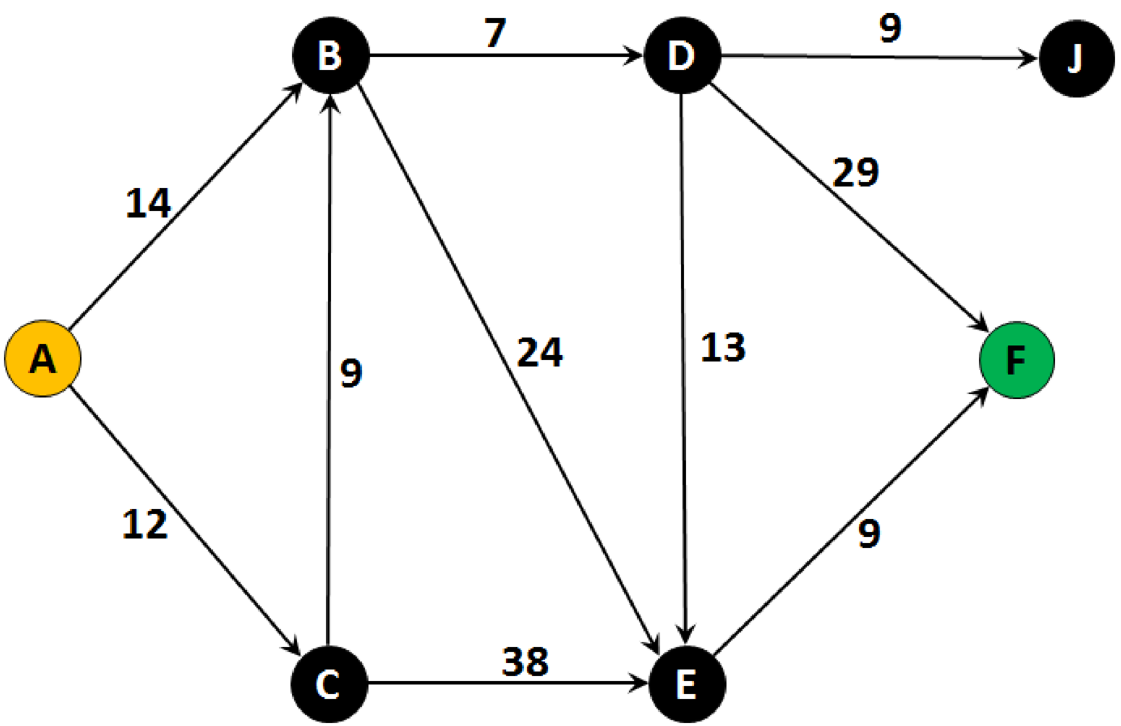
h(nó): subestimativa da distância deste nó até um estado objetivo

f(nó): função de avaliação baseada em caminho

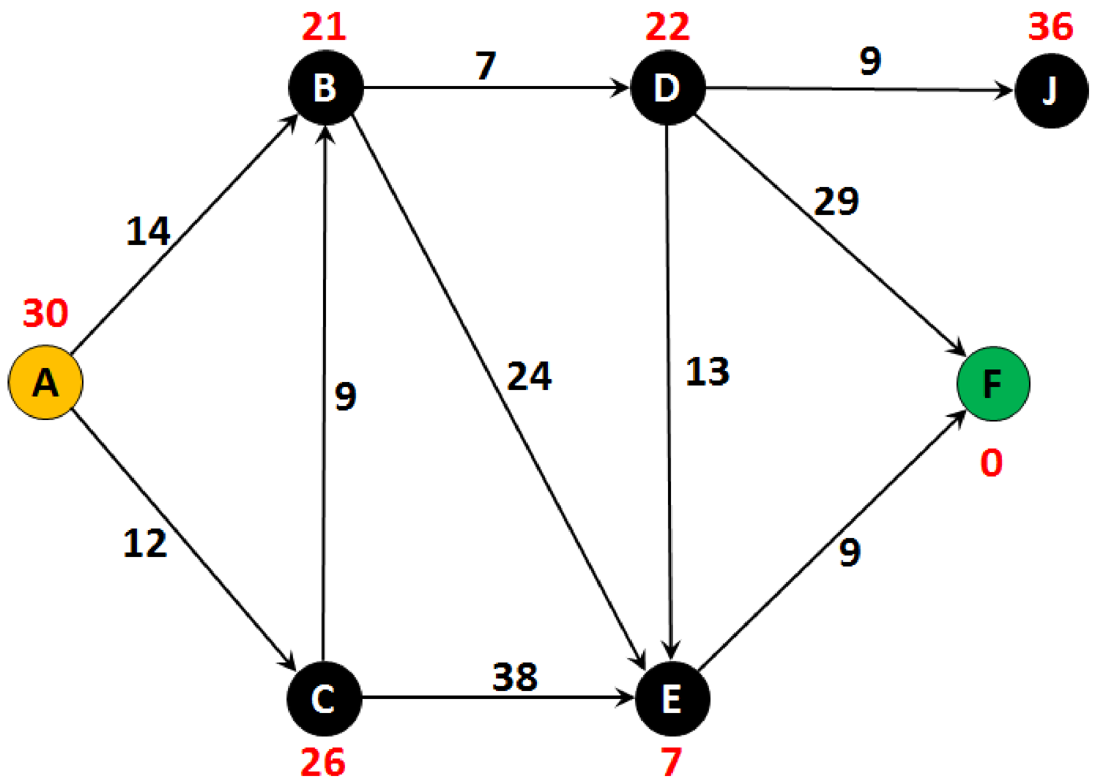
A*



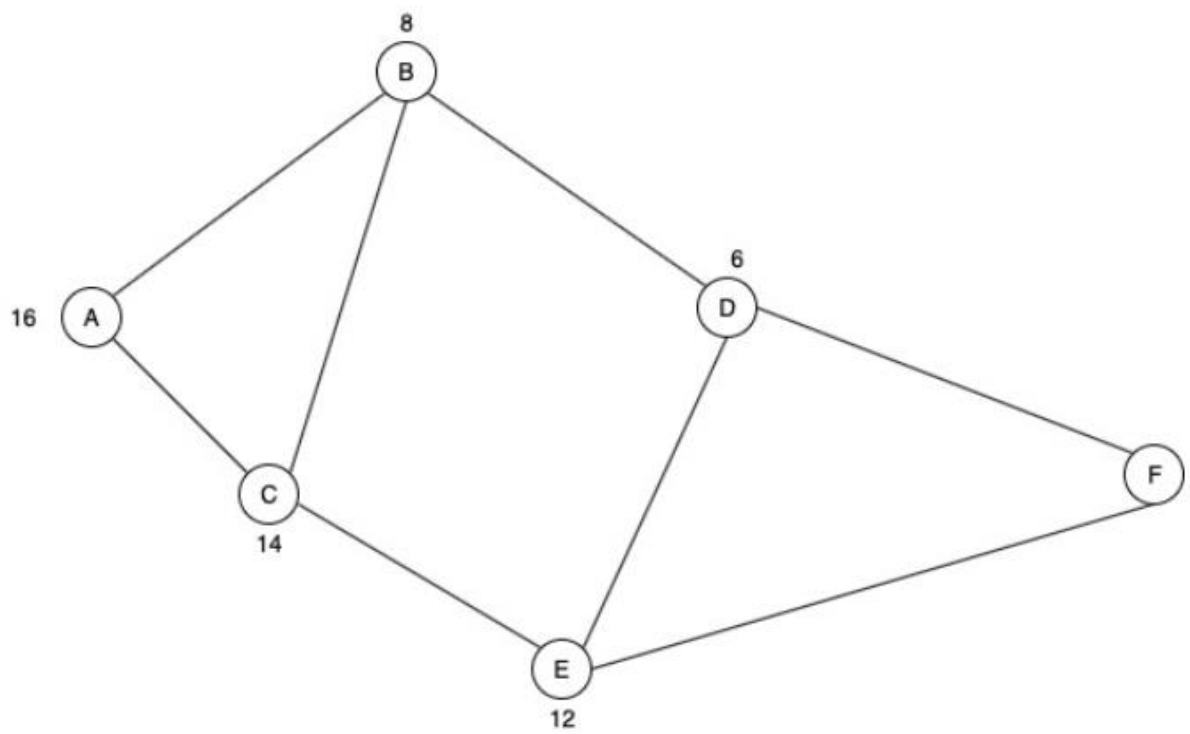
A*



A*



A*



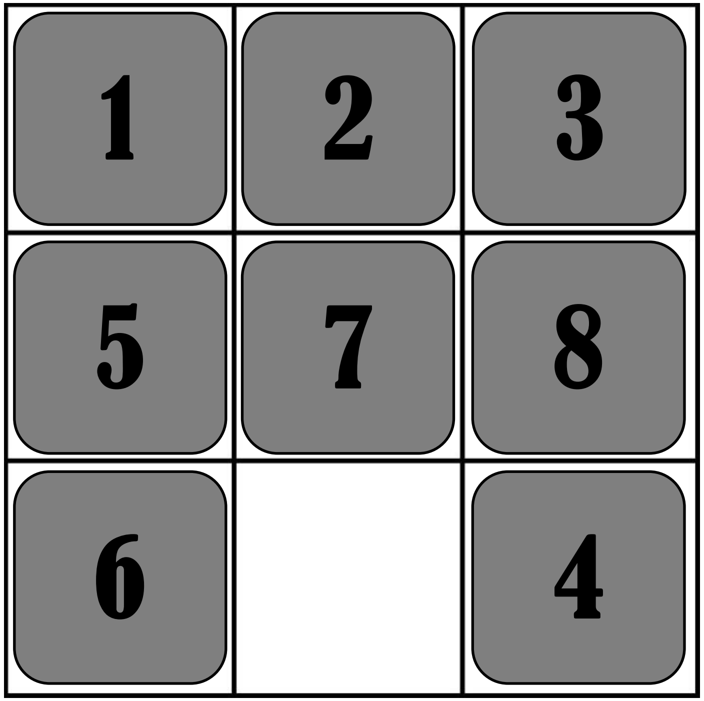
$$\overline{AB} = 10$$
$$\overline{AC} = 02$$
$$\overline{BC} = 03$$
$$\overline{BD} = 03$$
$$\overline{CE} = 03$$
$$\overline{DE} = 04$$
$$\overline{DF} = 06$$
$$\overline{EF} = 12$$

A*

Algoritmo 8: Algoritmo A*

```
faça uma lista aberta contendo apenas o nó inicial
faça uma lista fechada vazia
enquanto (o nó de destino não foi alcançado):
   considere o nó com a menor pontuação f na lista aberta
   se (este nó é o nó de destino):
      Terminou com SUCESSO
   senão:
      coloque o nó atual na lista fechada e observe todos os seus vizinhos
      para (cada vizinho do nó atual):
         se (vizinho tem um valor g menor que o atual e está na lista fechada):
             - Substitua o vizinho pelo novo valor mais baixo de g
             - O nó atual agora é o pai do vizinho
         caso contrário
             se (o valor atual de g for menor e esse vizinho estiver na lista aberta):
                 - Substitua o vizinho pelo novo valor mais baixo de g
                 - Alterar o pai do vizinho para o nosso nó atual
             caso contrário
                 se esse vizinho não estiver nas duas listas:
                     - Adicione-o à lista aberta e defina seu g.
```

A*



A*

- Importante: o valor estimado para h deve ser uma subestimação do valor real. Caso essa característica seja respeitada, o método garante a melhor solução.

Têmpera Simulada

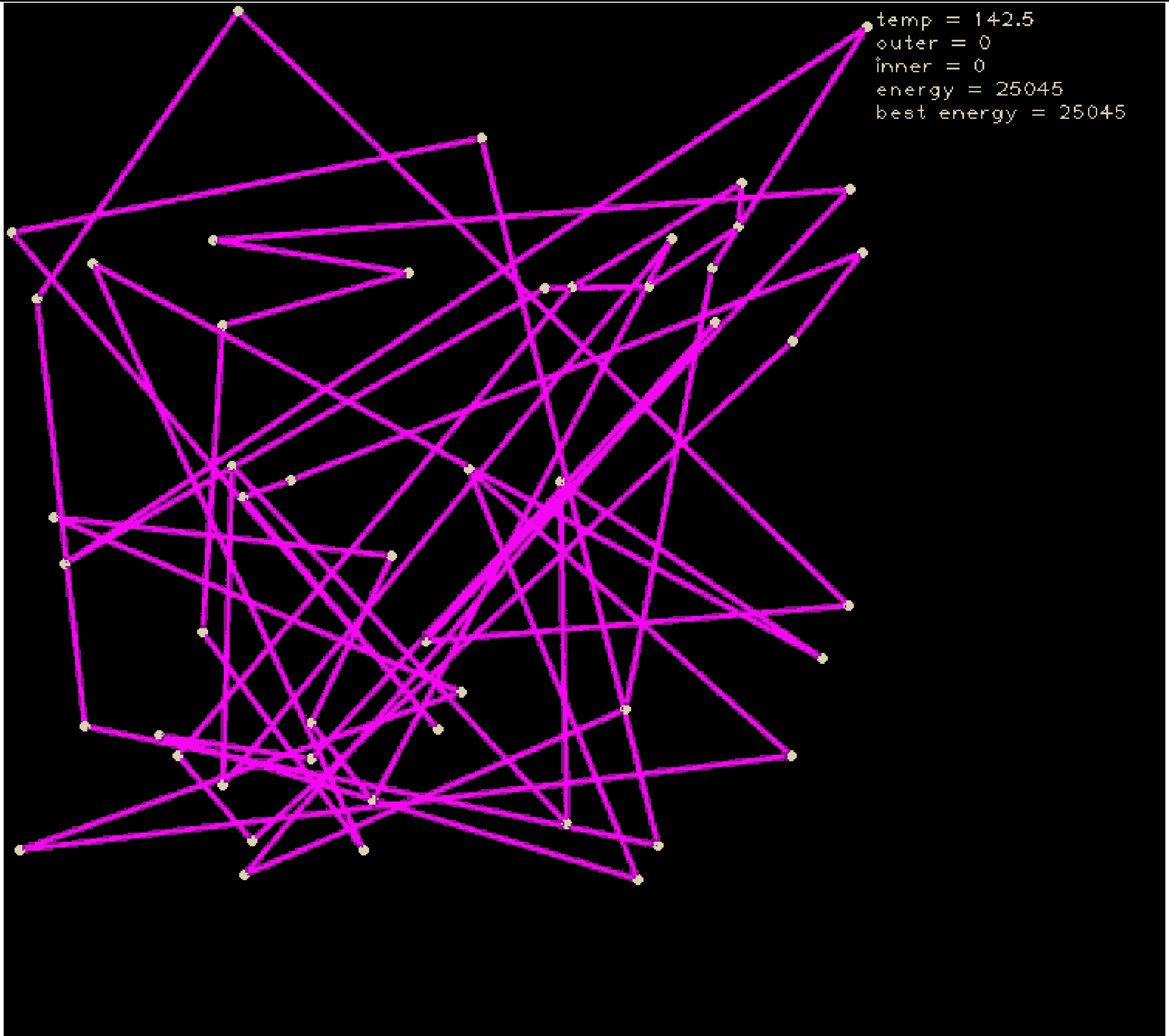
- Também conhecido como Simulated Annealing (SA)
- Técnica de otimização proposta por Kirkpatrick et al. em 1983 aplicado em projetos de circuitos
- Baseado em conceitos da mecânica estatística que foca em saber o que acontece com um sistema em baixa temperatura
- A ideia é aquecer o material até próximo seu ponto de fusão

Têmpera Simulada

- Nesse ponto o material contém um valor de energia próximo ao seu máximo. Isso faz com que as moléculas estejam em um estado bastante volátil, permitindo sua remodelagem

- Conforme o material vai esfriando, essa energia vai se dissipando e as moléculas vão perdendo mobilidade, rearranjando a estrutura do material

Têmpera Simulada - TSP



Fonte: https://github.com/TobyPDE/simulated-annealing-tsp

Têmpera Simulada

A cada iteração do método a temperatura do material é reduzida. O processo é executado até atingir seu ponto de mínimo.

Um processo muito acelerado pode levar a máximos locais, não atingindo a melhor solução global. Esse processo faz com que a temperatura chegue ao seu valor mínimo, mas a organização da estrutura do material não apresenta a melhor configuração.

Para evitar esse problema o SA permite que as soluções parciais sejam pioradas, permitindo que o método escape de máximos locais.

Têmpera Simulada

A aceitação de soluções piores é definida por uma certa probabilidade (caracterizada pela temperatura). Quanto maior a temperatura, maior a chance de aceitar soluções inferiores à atual.

Conforme o processo de resfriamento ocorre, a temperatura diminui, fazendo com que soluções inferiores tenham cada vez menos chance de serem aceitas.

Ao término do processo praticamente são aceitas apenas soluções melhores que a atual. Nesse ponto considera-se que se o método estava em um ótimo local ele já conseguiu escapar.

Têmpera Simulada

- Para cada nova possível solução (vizinhos da solução atual) calcula-se:

- $\Delta \mathrm{E} = \mathrm{E}_{\mathrm{i+1}} - \mathrm{E}_{\mathrm{i}}$

- Em que $\Delta \mathrm{E}$ corresponde à variação da solução atual E_{i} para uma possível nova solução $\mathrm{E}_{\mathrm{i+1}}$.

- Se o valor de $\Delta \mathrm{E}$ for negativo significa que uma solução melhor foi encontrada (minimização) e ela é aceita como possível resposta

- Caso contrário, a solução será aceita com base em uma probabilidade P

Têmpera Simulada

- A probabilidade de ΔE ocorrer é dada pela equação

- $P(\Delta E) = e^{-\frac{\Delta E}{T}}$

- Em que ΔE corresponde à variação da nova solução e T refere-se à temperatura atual. Um valor menor de T faz com que a chance da solução ser aceita diminui.

Têmpera Simulada

- Critério de Metrópolis *et al.* (1953)

- Um número aleatório x uniformemente distribuído no intervalo $[0, 1]$ é calculado e comparado com $P(\Delta E)$:

- Se $x < P(\Delta E)$ então a solução é aceita

- Caso contrário, a solução é rejeitada

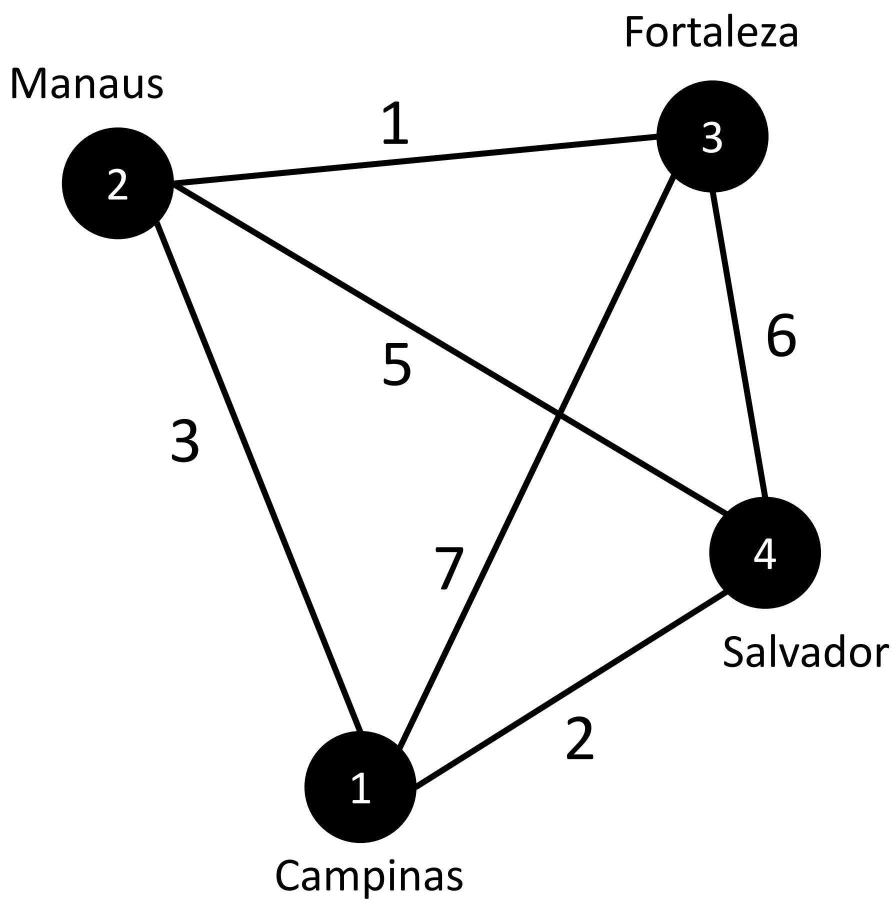
Têmpera Simulada

procedimento $SA(f(.), N(.), \alpha, SAmax, T_0, s)$

$s^\star \leftarrow s$;
$IterT \leftarrow 0$;
$T \leftarrow T_0$;
<u>enquanto</u> $(T > 0)$ <u>faça</u>
    <u>enquanto</u> $(IterT < SAmax)$ <u>faça</u>
        $IterT \leftarrow IterT + 1$;
        Gere um vizinho qualquer $s' \in N(s)$;
        $\Delta = f(s') - f(s)$;
        <u>se</u> $(\Delta < 0)$
            <u>então</u>
                $s \leftarrow s'$;
                <u>se</u> $(f(s') < f(s^\star))$ <u>então</u> $s^\star \leftarrow s'$;
            <u>senão</u>
                Tome $x \in [0, 1]$;
                <u>se</u> $(x < e^{-\Delta/T})$ <u>então</u> $s \leftarrow s'$;
        <u>fim-se</u>;
    <u>fim-enquanto</u>;
    $T \leftarrow \alpha \times T$;
    $IterT \leftarrow 0$;
<u>fim-enquanto</u>;
$s \leftarrow s^\star$;
Retorne s;

fim SA;

Exemplo do TSP



Exemplo do TSP

- Função Objetivo $f(.)$

- Como o objetivo do problema é traçar uma rota passando por todos os vértices do grafo (uma única vez) e retornar o vértice de origem, a função objetivo consiste em somar as distâncias dos trajetos percorridos.

Exemplo do TSP

- Espaço de Busca $N(.)$
- Consiste no conjunto de soluções possíveis dentro do escopo do problema a ser resolvido. Nesse caso, é o conjunto de rotas que podem ser traçadas

  - 12341, 12431, 13241, 13421, 14231, 14321
  - 21342, 21432, 23142, 23412, 24132, 24312
  - 31243, 31423, 32143, 32413, 34123, 34213
  - 41234, 41324, 42134, 42314, 43124, 43214

Exemplo do TSP

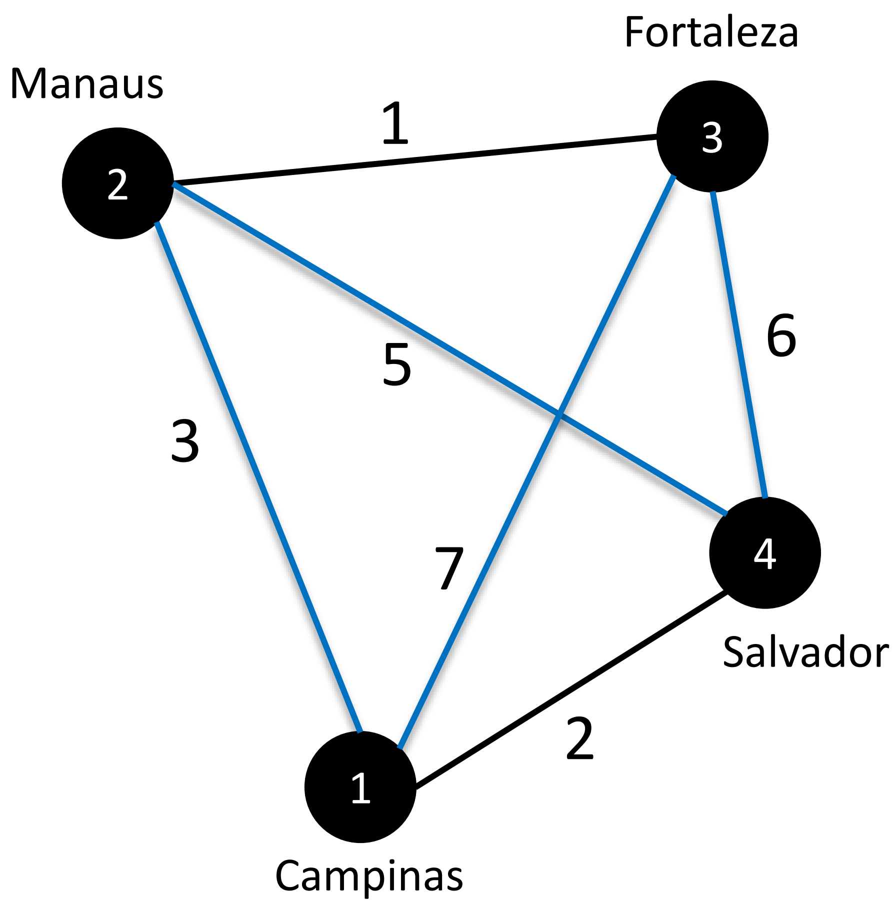
- Espaço de Busca $N(.)$
- Consiste no conjunto de soluções possíveis dentro do escopo do problema a ser resolvido. Nesse caso, é o conjunto de rotas que podem ser traçadas

  - 12341, 12431, 13241, **13421**, 14231, 14321
  - 21342, 21432, 23142, 23412, 24132, 24312
  - 31243, 31423, 32143, 32413, 34123, 34213
  - 41234, 41324, 42134, 42314, 43124, 43214

Exemplo do TSP

- Fator de atenuação da temperatura $\alpha = 0{,}75$
- Nº de soluções buscadas por temperatura SA_{max} =3
- Temperatura inicial $T_0 = 5$
- Solução inicial (13421) $s = 21$

Exemplo do TSP



Exemplo do TSP

$s^* = 21\ (13421)$

$SA_{max}=3$

$$\mathbf{T = 5}$$

IterT=1

Nova solução: 31423 (trocando-se 1 e 3)

f(31423)=15

$$\Delta E = E_{i+1} - E_i$$
$$\Delta E = 15 - 21$$
$$\Delta E = -6$$

$s^* = 15\ (31423)$

Exemplo do TSP

$s^* = 15$ (31423)

$SA_{max}=3$

$$\mathbf{T = 5}$$

IterT=2

Nova solução: 34123 (trocando-se 1 e 4)

f(34123)=12

$$\Delta E = E_{i+1} - E_i$$
$$\Delta E = 12 - 15$$
$$\Delta E = -3$$

$s^* = 12$ (34123)

Exemplo do TSP

$s^* = 12\ (34123)$

$SA_{max}=3$

$\mathbf{T = 5}$

IterT=3

Nova solução: 24312 (trocando-se 3 e 4)

f(24312)=21

$$\Delta E = E_{i+1} - E_i$$
$$\Delta E = 21 - 12$$
$$\Delta E = 9$$

Nesse caso deve-se testar o critério de Metropolis

Exemplo do TSP

$$P(\Delta E) = e^{-\frac{\Delta E}{T}}$$
$$P(\Delta E) = e^{-\frac{9}{5}} = 0{,}1653$$

Sorteia-se um valor aleatório x do intervalo [0,1]. Caso o valor sorteado seja inferior a $P(\Delta E)$, a solução é aceita. Caso contrário ela é descartada e um novo vizinho é formado.

Vamos supor que o valor de x é de 0,8172. Nesse caso, a solução s* = 12 (23412) permanece

Exemplo do TSP

- Dado que o valor de SAmax foi atingido, antes da próxima sequência de vizinhos deve-se diminuir a temperatura T.

- Esse processo é feito até que T seja zero ou valor muito próximo disso.

Estratégias de Resfriamento

$$T_k = \alpha T_{k-1} \quad \forall k \geq 1$$

$$0 < \alpha < 1$$

$$T_k = \frac{T_{k-1}}{1 + \gamma\sqrt{T_{k-1}}} \quad \forall k \geq 1$$

$$0 < \gamma < 1$$

Estratégias de Resfriamento

$$T_k = \begin{cases} \beta T_{k-1} & se\ k = 1 \\ \dfrac{T_{k-1}}{1 + \gamma T_{k-1}} & se\ k \geq 2 \end{cases}$$

Onde:

$$0 < \beta < 1$$

$$\gamma = \frac{T_0 - T_{k-1}}{(k-1) T_0 T_{k-1}}$$

Temperatura Inicial

Média dos custos dos vizinhos

1) Gerar uma solução inicial válida

2) Gerar um determinado número de vizinhos

3) Para cada vizinho calcular seu custo

4) A temperatura inicial será a média dos custos dos vizinhos criados

Temperatura Inicial

Empiricamente

1) Gerar uma solução inicial válida

2) Empregar uma temperatura inicial baixa e contar quantos vizinhos são aceitos em SA_{max}

3) Se o percentual de vizinhos aceito for grande (superior a 90%) essa temperatura pode ser usada como inicial

4) Caso contrário, aumente uma pouco a temperatura ($\pm 10\%$) e repita o processo

Mochila Binária

- Para exemplificar a aplicação do método de SA ao problema da mochila vamos considerar as seguintes configurações do problema:

- Pesos $P = (6, 10, 9, 5, 12, 4)$
- Benefícios $V = (8, 5, 10, 15, 7, 5)$
- Capacidade de Mochila $C = 30$

Mochila Binária

- O primeiro passo é encontrar uma solução inicial válida para podermos determinar a temperatura inicial T_0.

- O conjunto de possíveis soluções basicamente é a combinação das presenças e ausências de todos os seis itens disponíveis, totalizando assim, 64 possibilidades (2^6).

- Uma solução é considerada válida, ou viável, se ela emprega um item de forma inteira e se a soma dos itens adicionados à mochila não ultrapassam sua capacidade.

Mochila Binária

- Dessa forma, vamos considerar a seguinte configuração como solução inicial:

- Pesos $P = (6, 10, 9, 5, 12, 4)$
- Benefícios $V = (8, 5, 10, 15, 7, 5)$

- Solução Inicial
- $S_0 = (0, 1, 0, 0, 1, 0)$
- $P_{S_0} = 22\ (22 \leq 30)$
- $V_{S_0} = 12$

Busca de Custo Uniforme

- Também chamada ramificar e limitar
- É uma variação da busca em largura
- Ao invés de expandir o primeiro nó na fila deve-se expandir o nó com o menor custo de caminho
- Variação da busca pelo 1º melhor:
  - Usa a função g(n), assim como A*
  - Mas o valor de h(n) = 0 (não é levada em conta)
- É completa e ótima, desde que o custo de um caminho cresça monotonicamente

Busca de Custo Uniforme

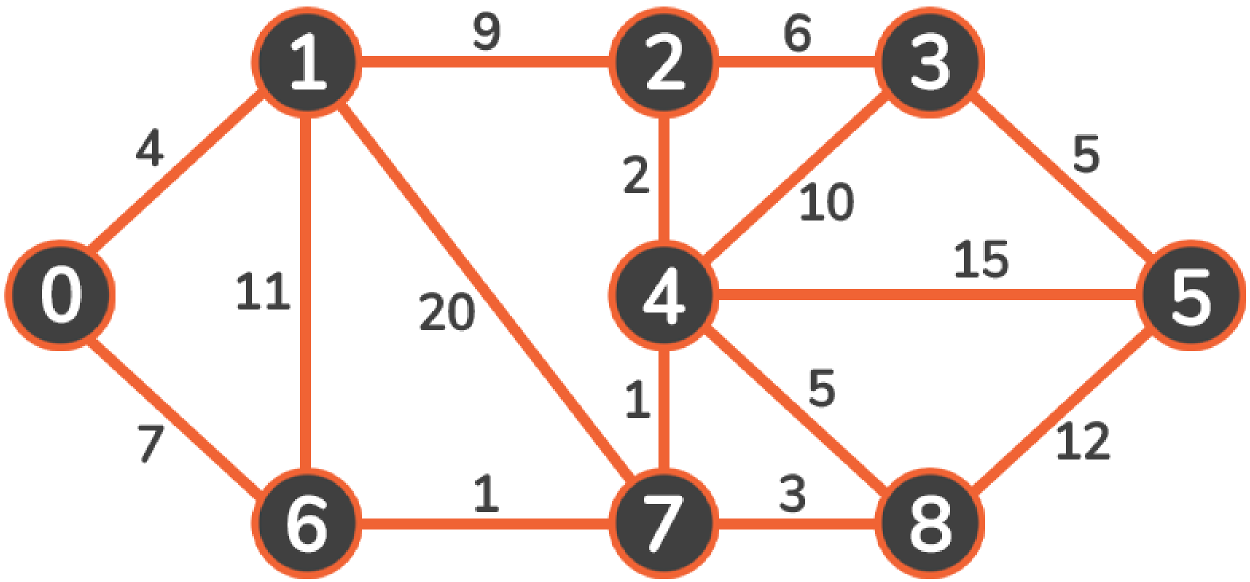
- Foi criada por Dijkstra em 1959 e também é conhecida como Algoritmo de Dijkstra

Busca de Custo Uniforme

Algoritmo 9: Algoritmo de Dijkstra (Busca de Custo Uniforme)

```
função dijkstra(G, S)

   para cada vértice V em G
      distancia[V] <-- infinito
      anterior[V] <-- NULL
      se V != S
         adicione V à fila de prioridade Q
      distancia[S] <-- 0

      enquanto (Q ≠  ∅) faça
         U <-- extraia o menor em Q
         para cada vizinho V não visitado  U
            tempDistance <-- distancia[U] + peso_aresta(U, V)
            se tempDistance < distancia[V]
               distancia[V] <-- tempDistance
               anterior[V] <-- U
retorne distancia[], anterior[]
```

Busca de Custo Uniforme



Minimax

Quando a busca não tem um adversário (só o próprio problema a ser resolvido)

- A solução consiste de um método para encontrar o objetivo, geralmente usando alguma heurística
- Função de avaliação: estimativa de custo ou benefício ao longo das escolhas realizadas

Minimax

Quando o problema trata-se de um jogo:

- Os objetivos dos agentes são conflitantes
- A solução passa a ser uma estratégia que especifica um movimento para cada resposta possível dada pelo oponente
- Exemplos: xadrez, damas, go, gamão, jogo da velha
- Jogos de soma zero
- Informação completa
- Os agentes interagem de forma alternada

Minimax

- O algoritmo é recursivo e é usado para escolher um movimento ideal para um jogador, assumindo que o outro jogador também esteja jogando de maneira ideal
- Para isso ele considera o estado atual do jogo e quais os movimentos que o adversário pode realizar
- Os jogadores recebem a nomenclaturas de Max e Min. O primeiro quer maximizar seu ganho enquanto o segundo quer minimizá-lo

Minimax

- Todo estado do tabuleiro do jogo tem um valor associado a ele:
  - Se o maximizador tiver vantagem a pontuação do tabuleiro tenderá a ter algum valor positivo
  - Se o minimizador tiver vantagem o estado tenderá a ter algum valor negativo

Minimax

- Árvore do Jogo: estrutura na forma de uma árvore que consiste em todos os movimentos possíveis que permitem passar de um estado do jogo para o próximo estado

Um jogo pode ser definido como um problema de pesquisa com os seguintes componentes:

- Estado inicial: compreende a posição do tabuleiro e mostra de quem é o movimento
- Função sucessora: define quais são os movimentos legais que um jogador pode fazer

Minimax

- Estado terminal: é a posição do tabuleiro quando o jogo termina
- Função de utilidade: é uma função que atribui um valor numérico ao resultado de um jogo:

  Ex.: no xadrez ou no jogo da velha, o resultado é uma vitória (+1), uma perda (-1) ou um empate (0)

Minimax

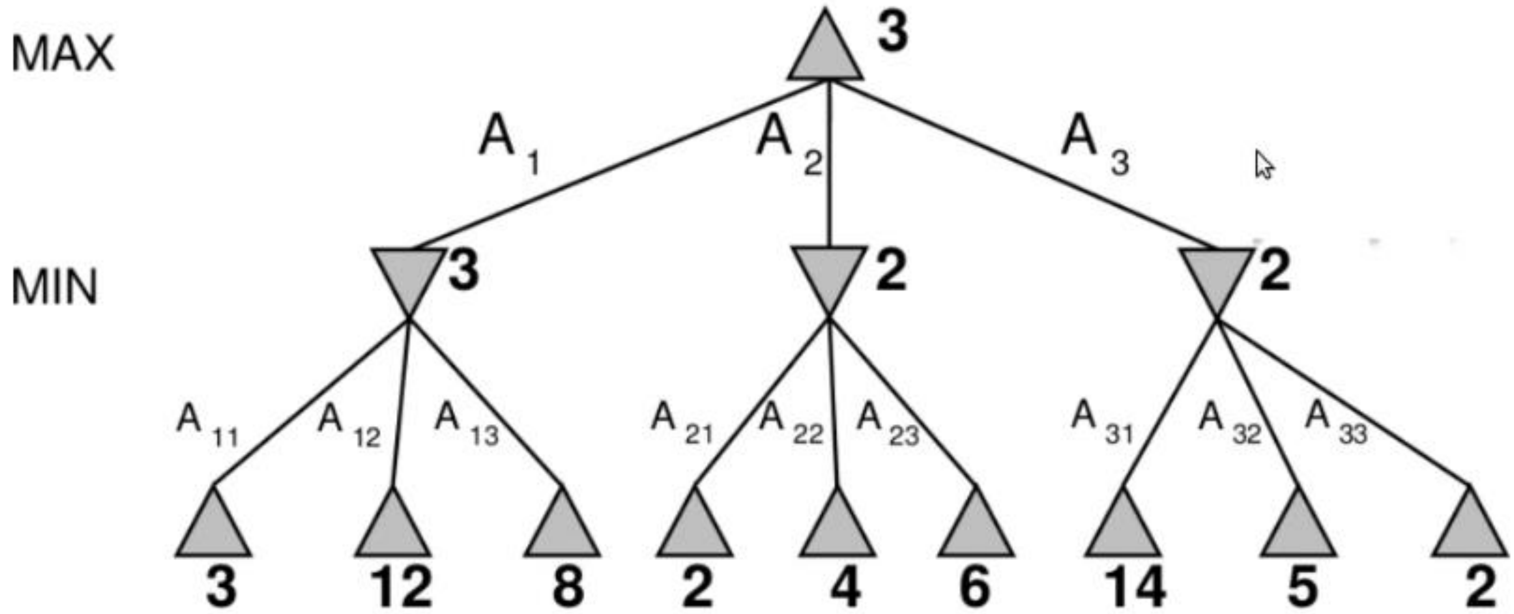
Como o algoritmo funciona?

- Processo geral do algoritmo:

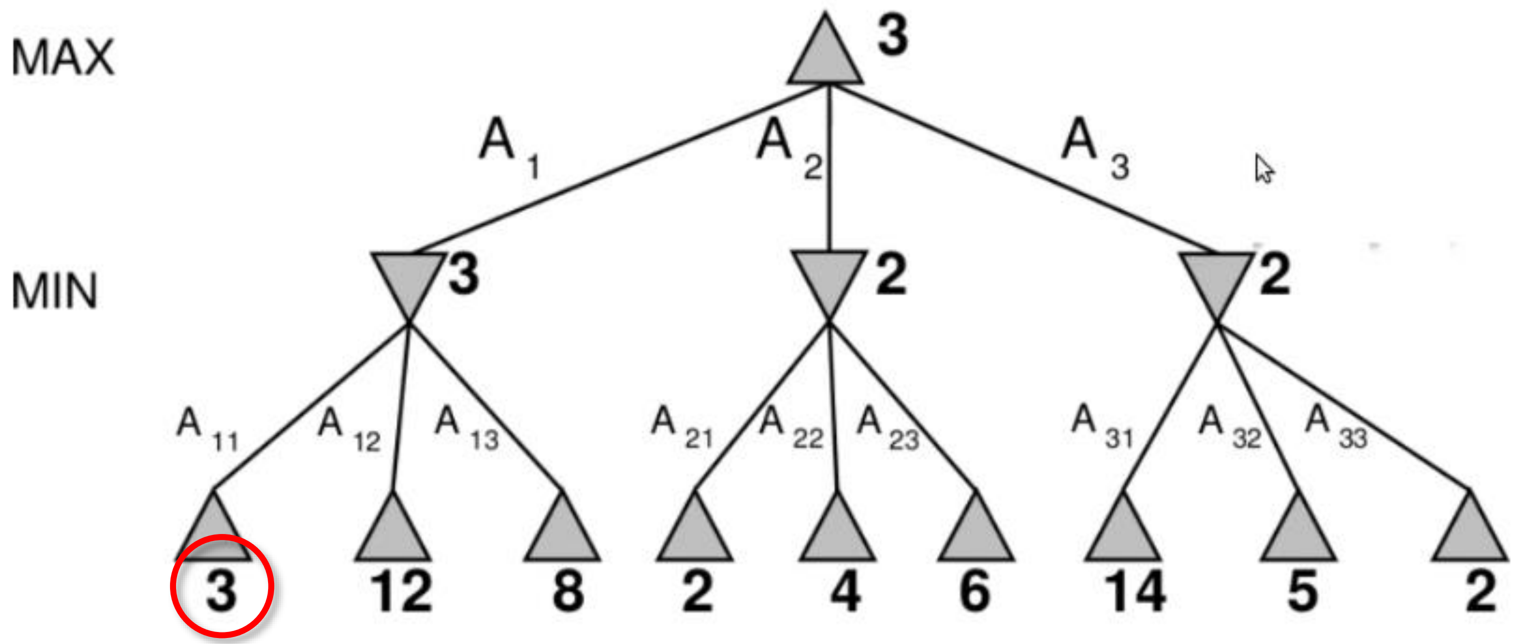
Etapa 1: gere toda a árvore do jogo, começando com a posição inicial até os estados terminais.

Exemplo:

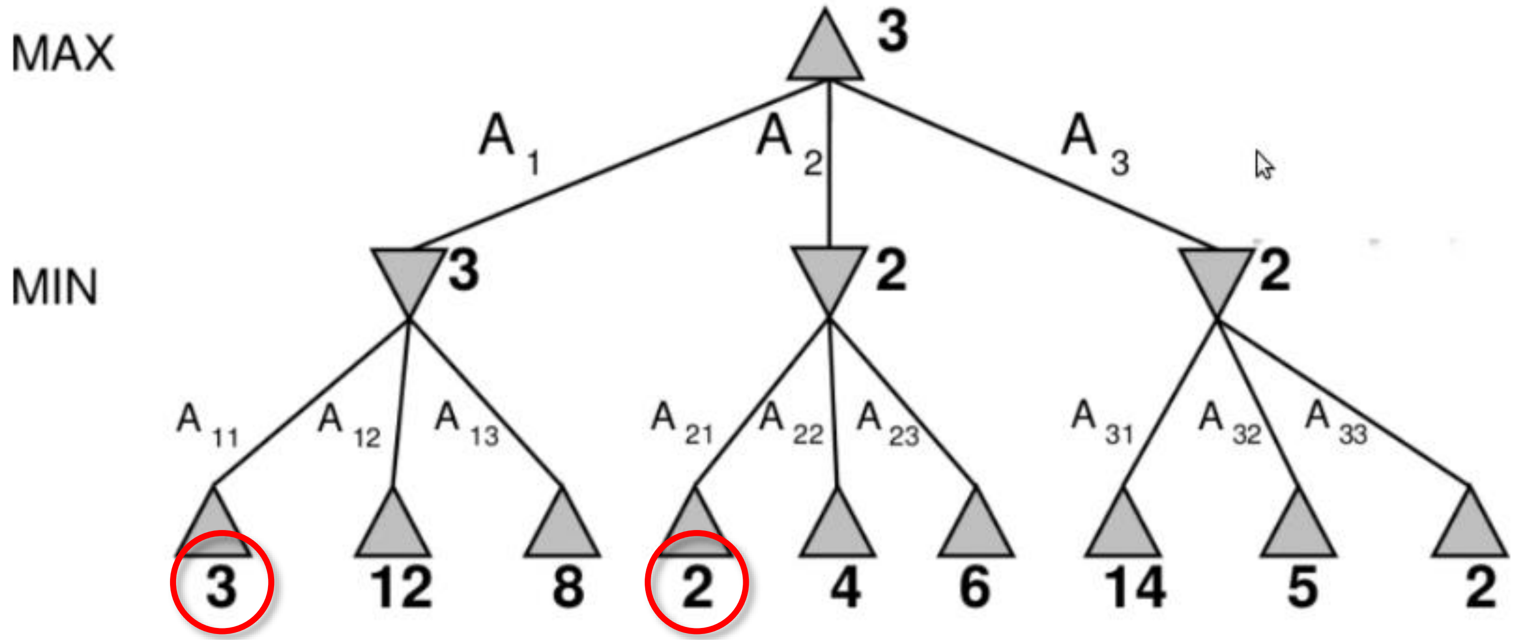
Minimax


MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

Minimax

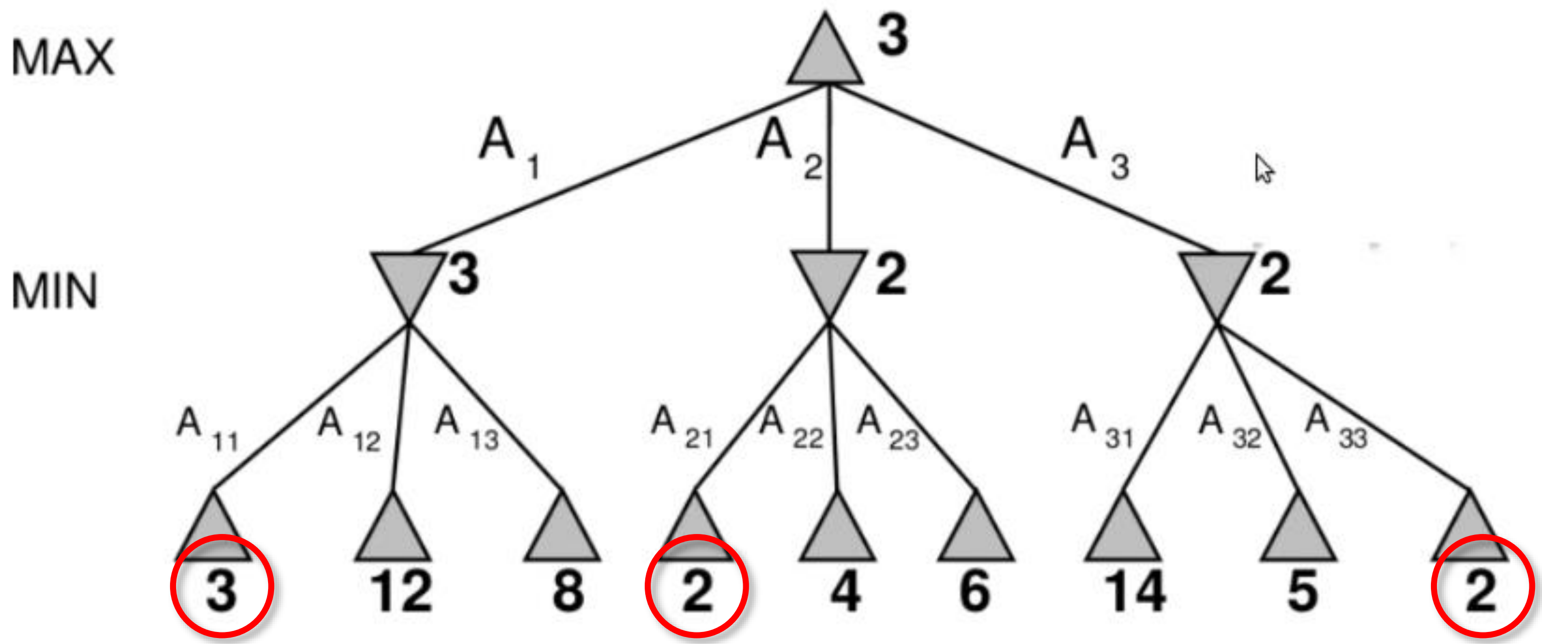

MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

Minimax



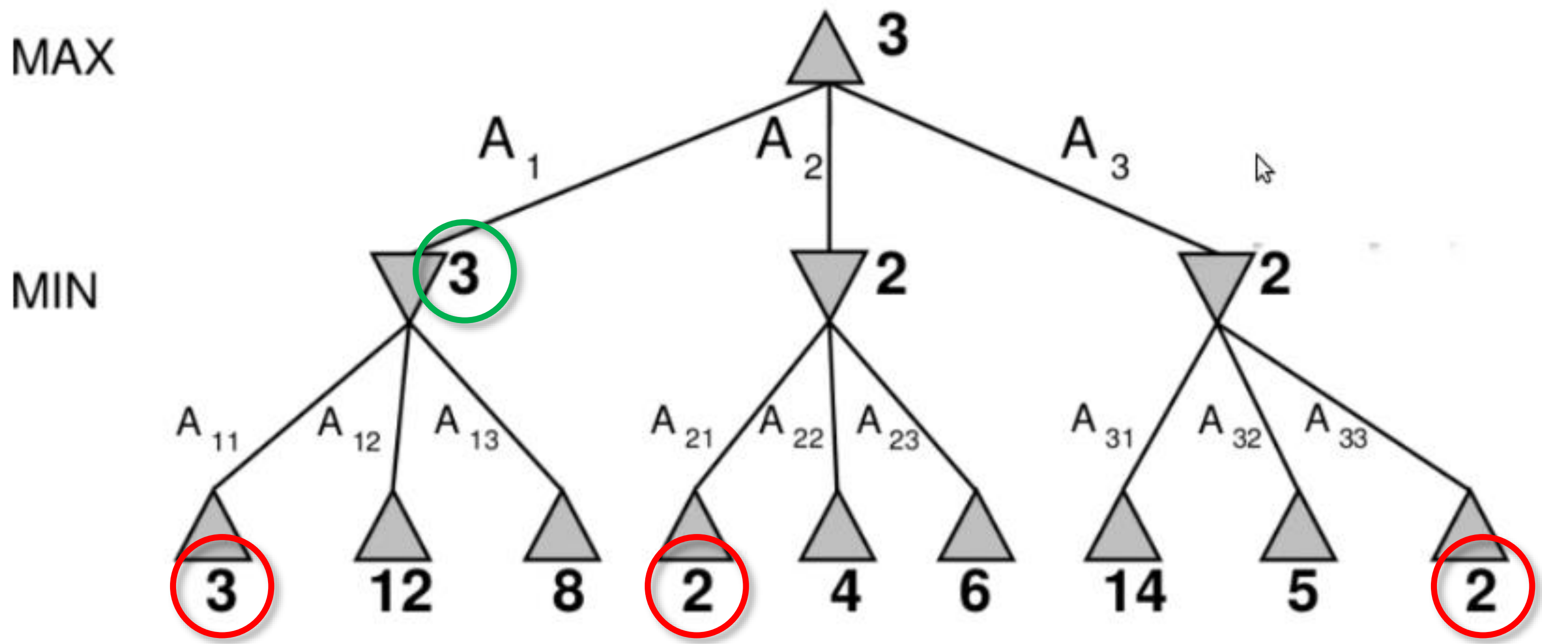
MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

Minimax



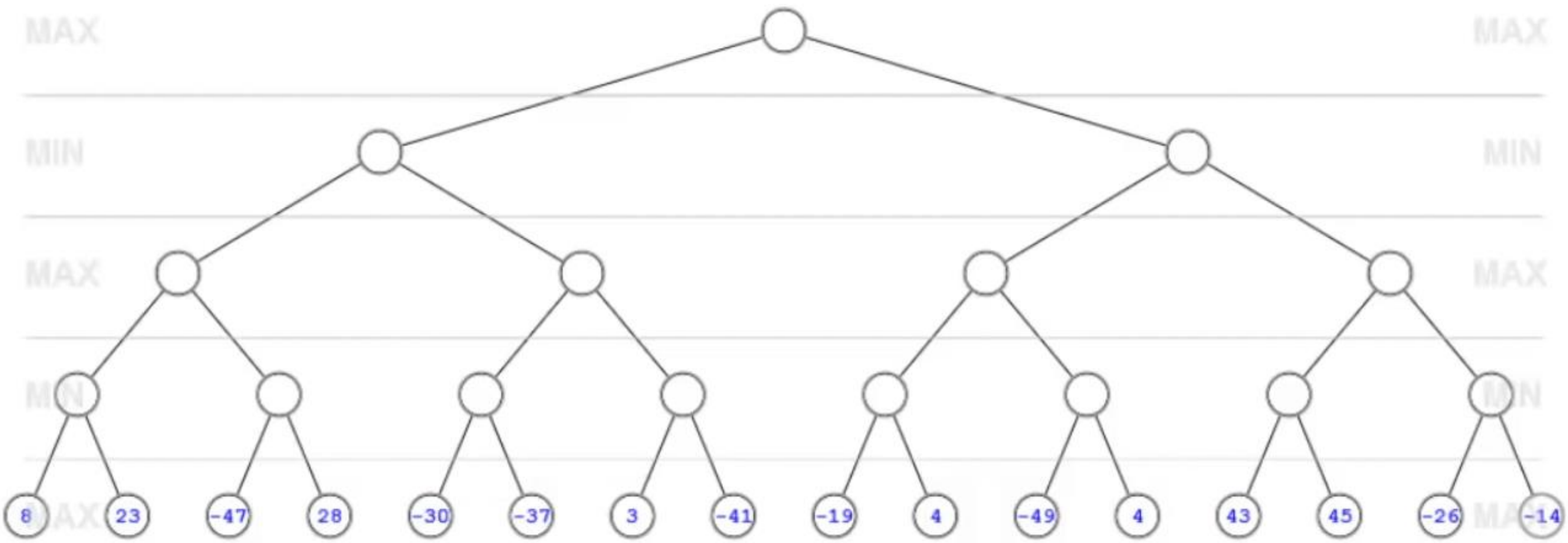
MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

Minimax

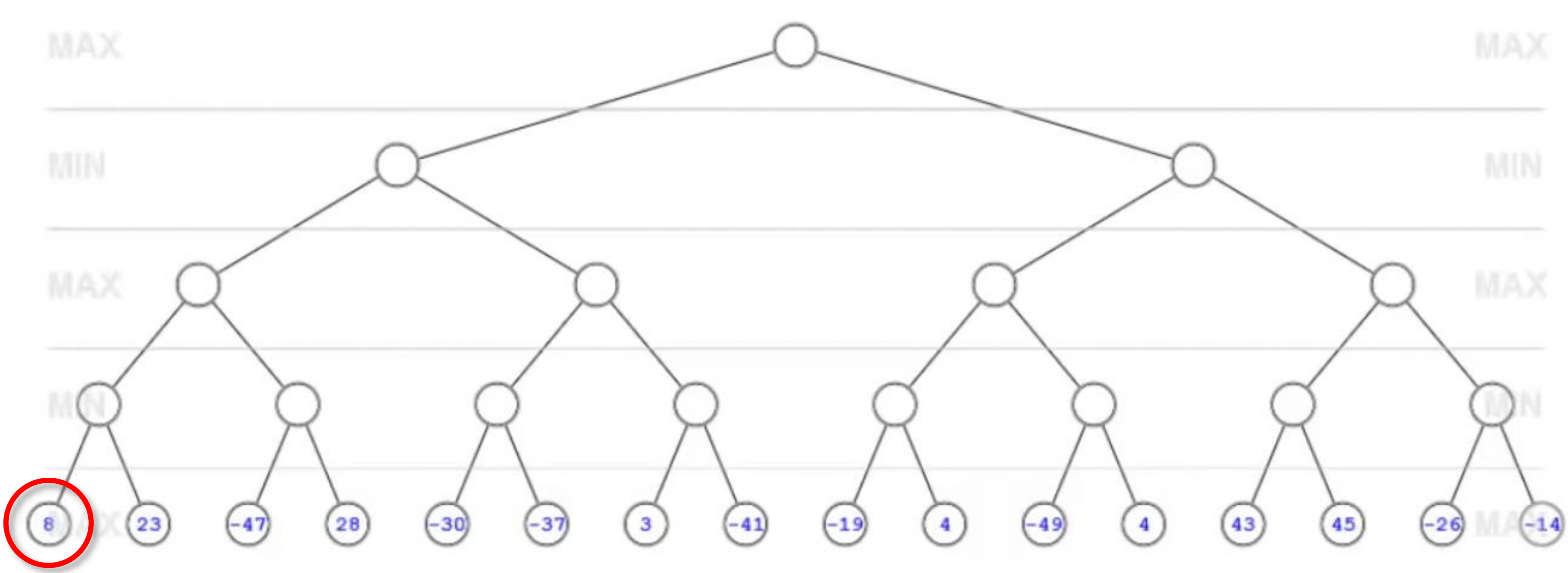


MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

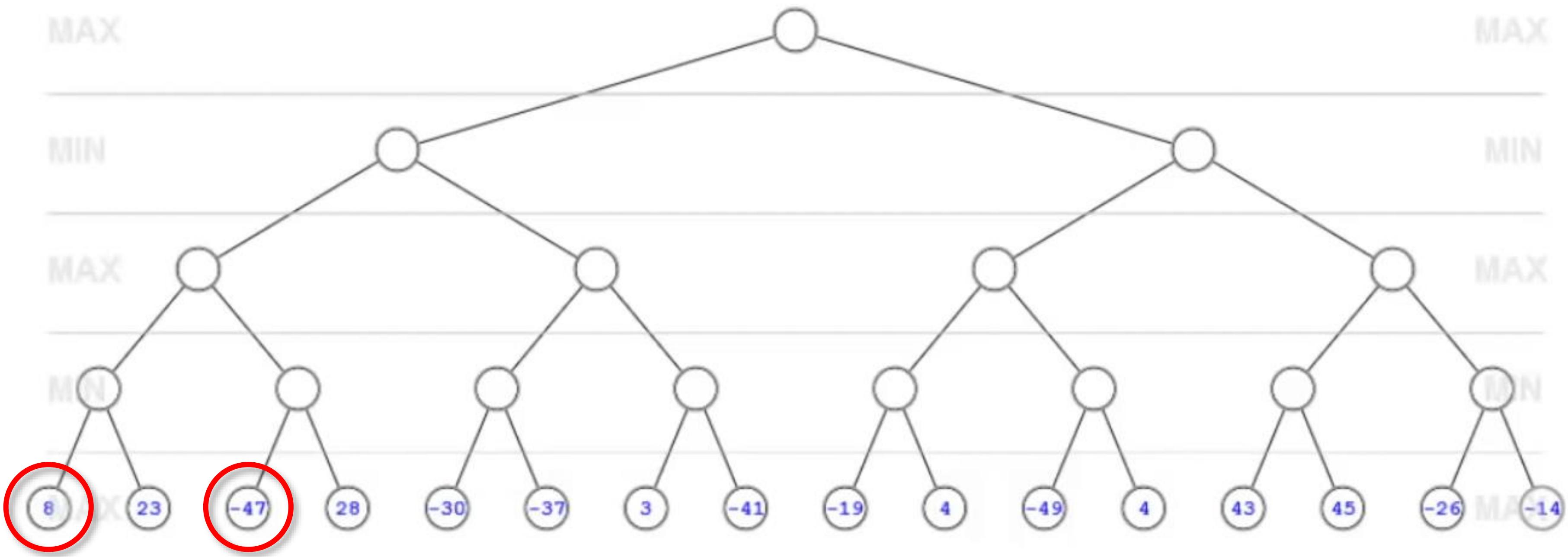
Minimax



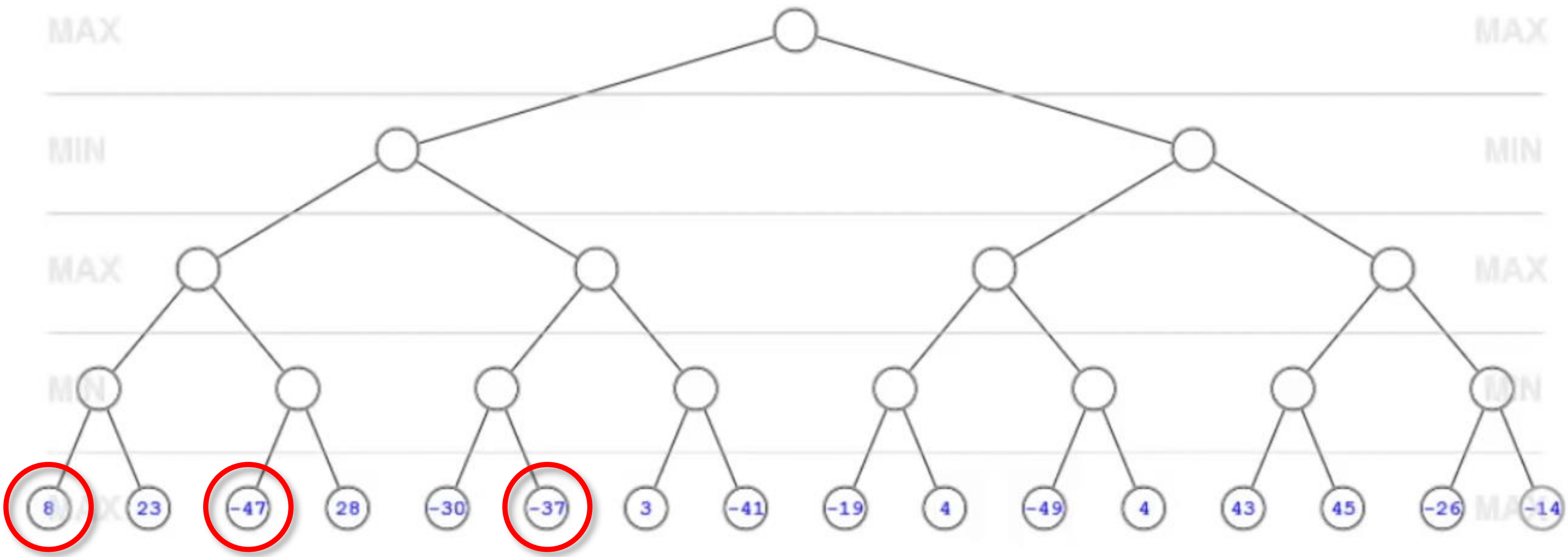
Minimax


MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
MAX
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

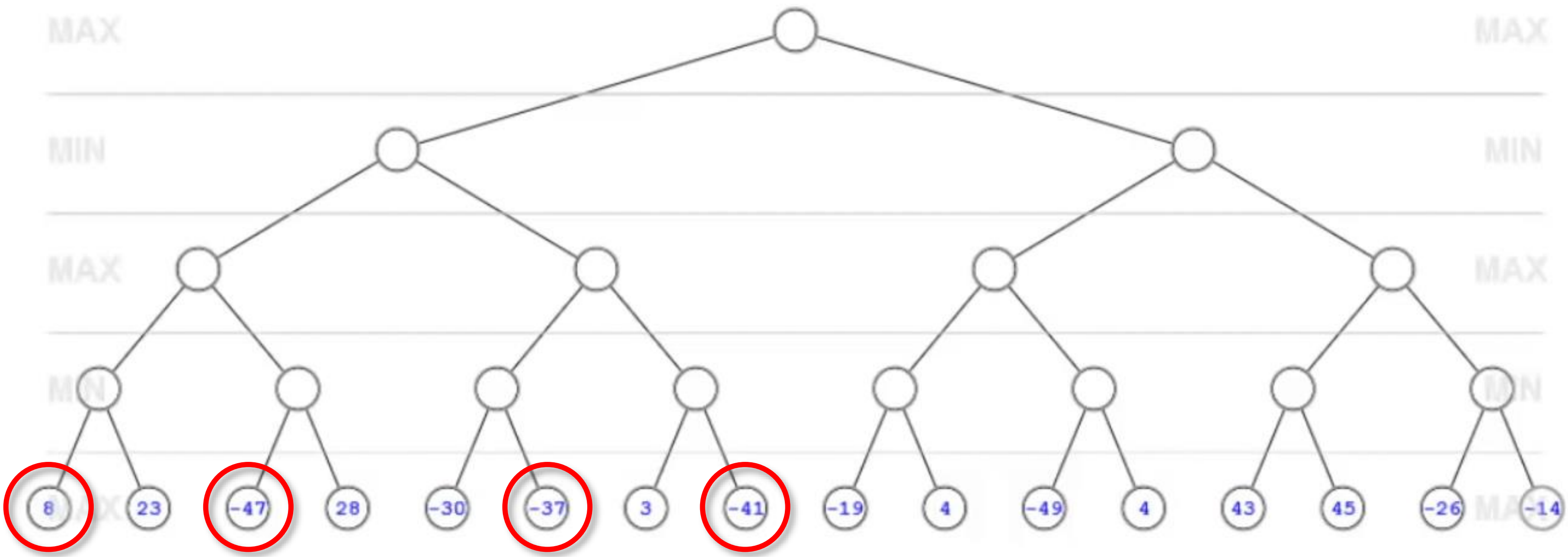
Minimax



Minimax

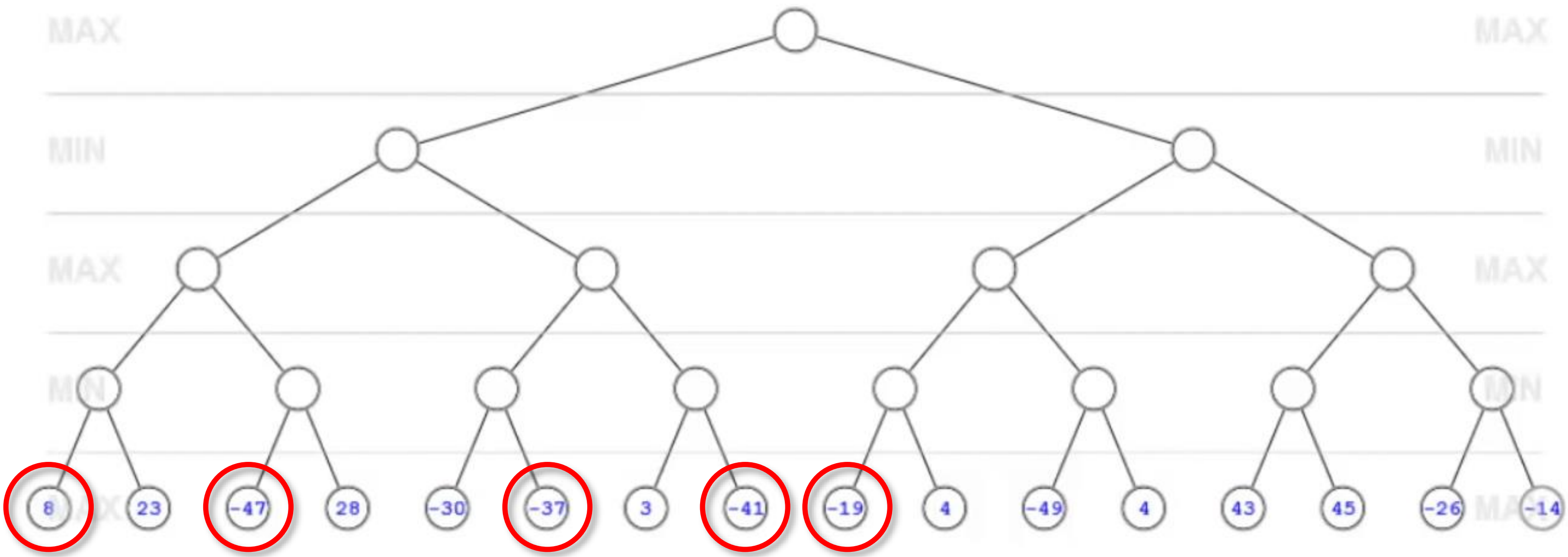


Minimax

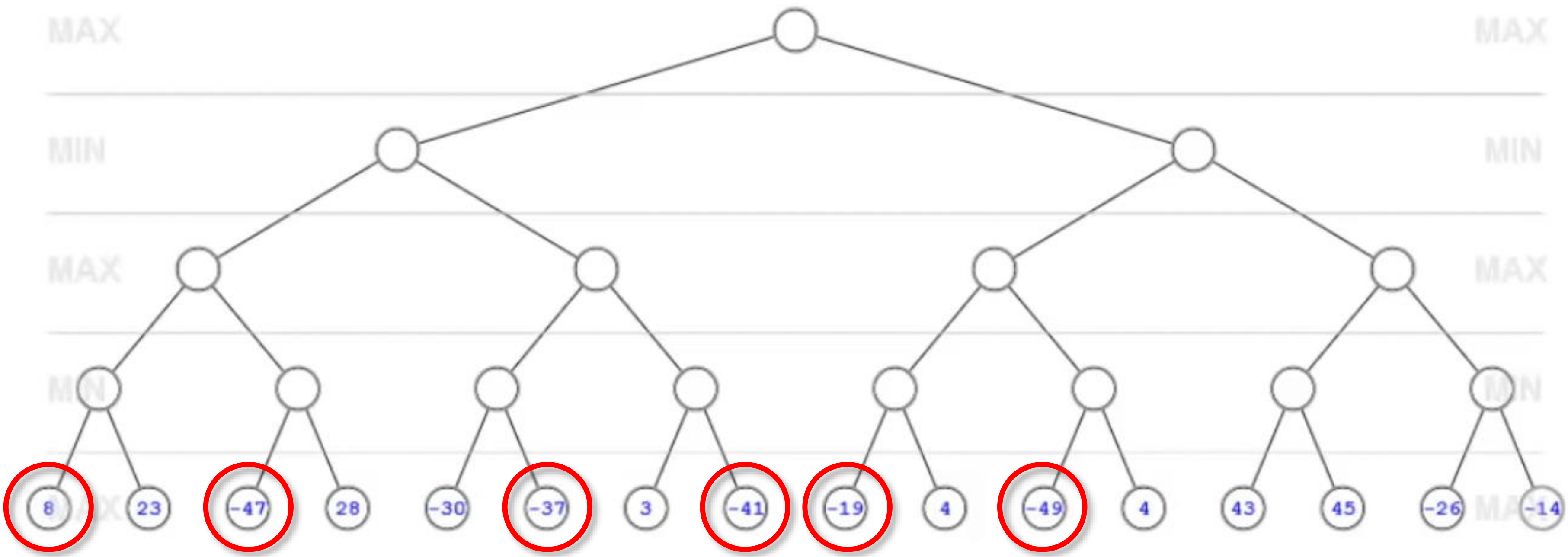


MAX
MAX
MIN
MIN
MAX
MAX
MIN
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

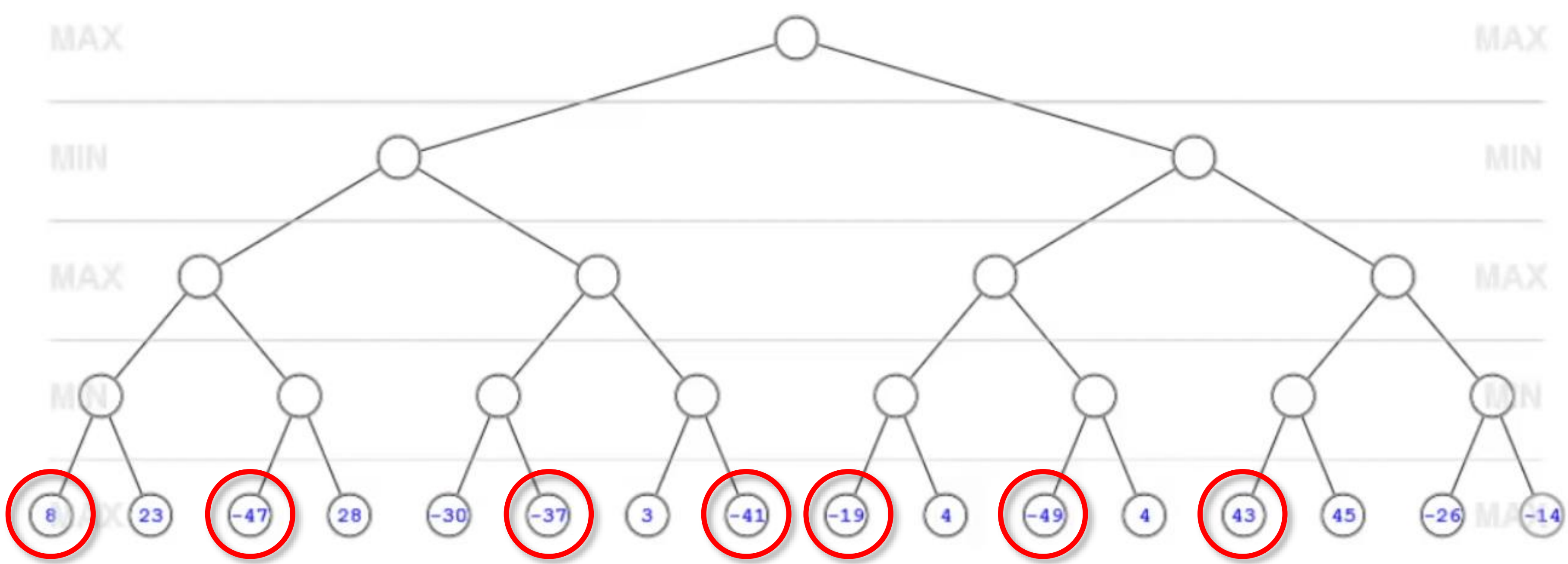
Minimax



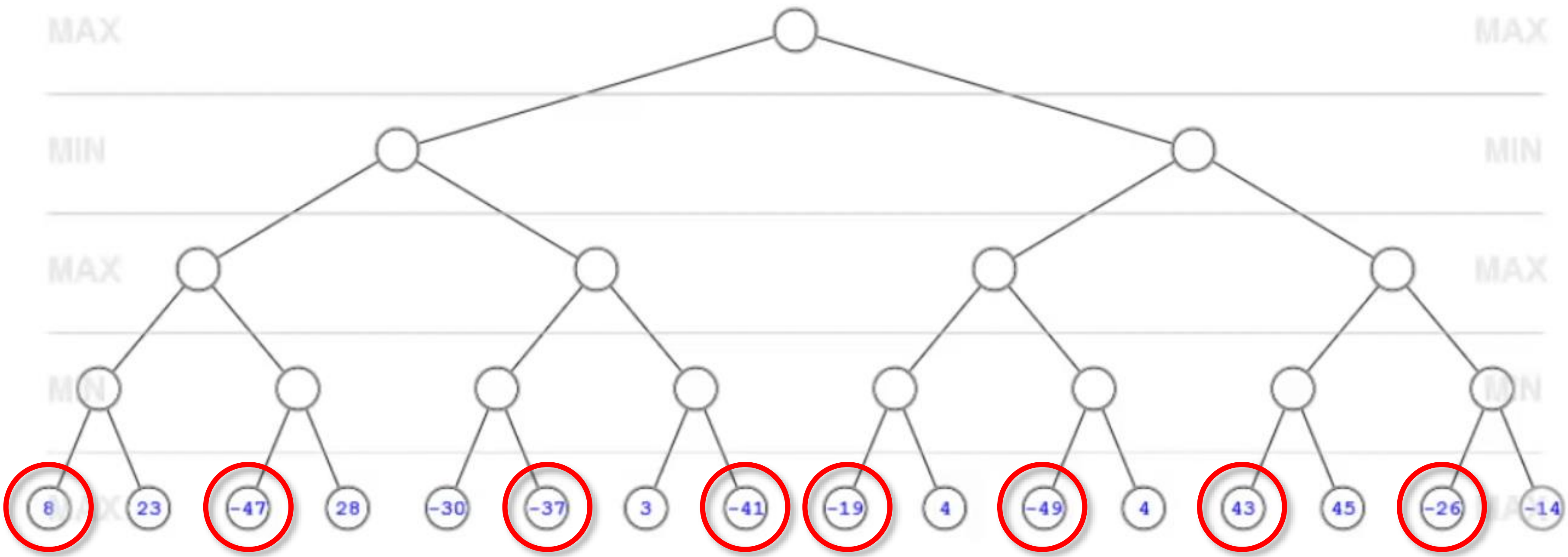
Minimax


MAX
MAX
MIN
MIN
MAX
MAX
MIN
MIN
MAX
MAX
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

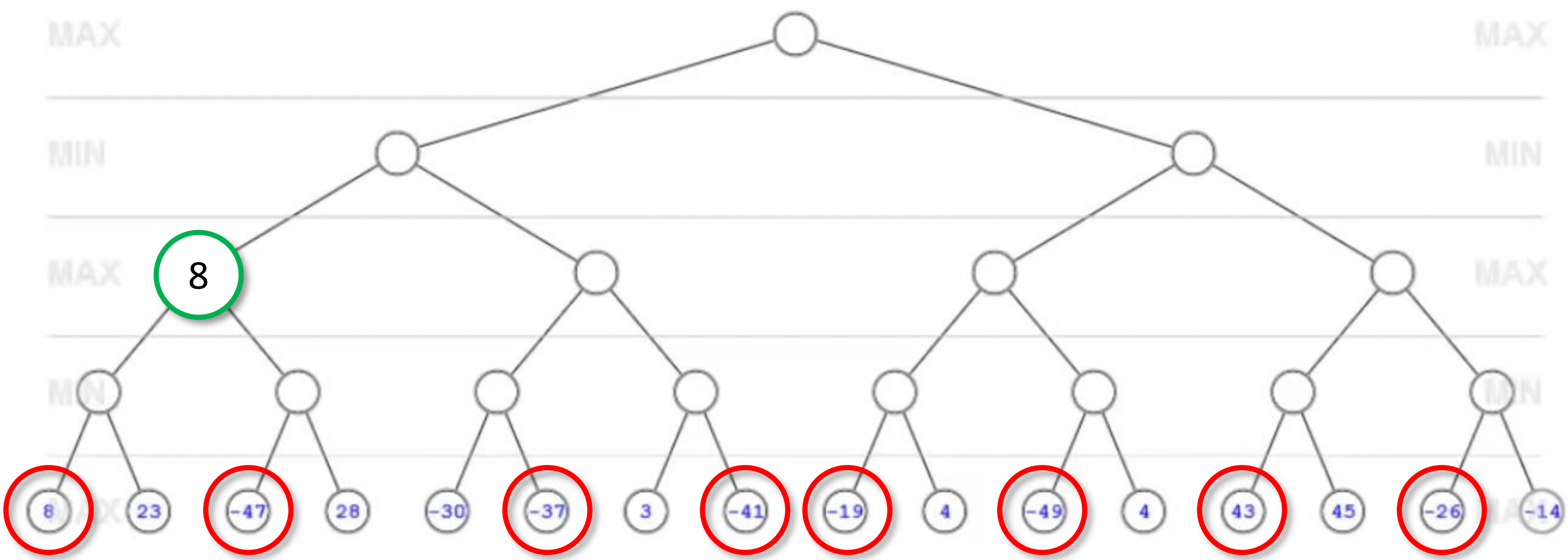
Minimax


MAX
MAX
MIN
MIN
MAX
MAX
MIN
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax

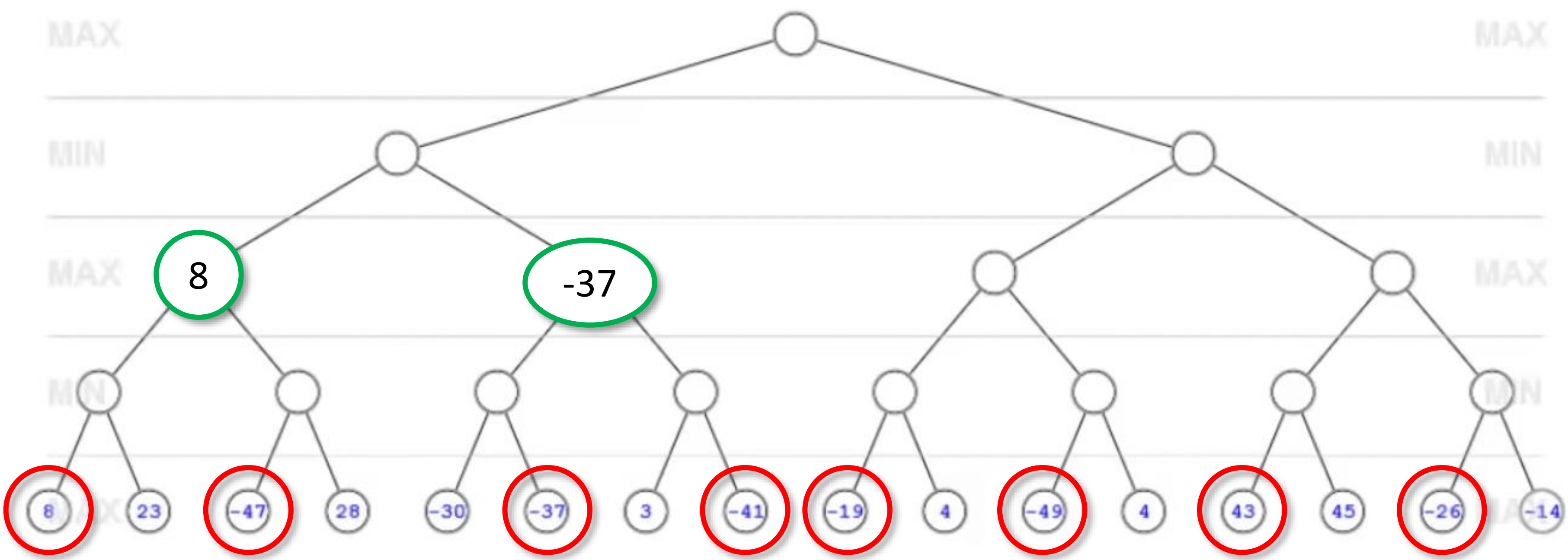

MAX
MIN
MAX
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14
MAX
MIN
MAX
MIN

Minimax

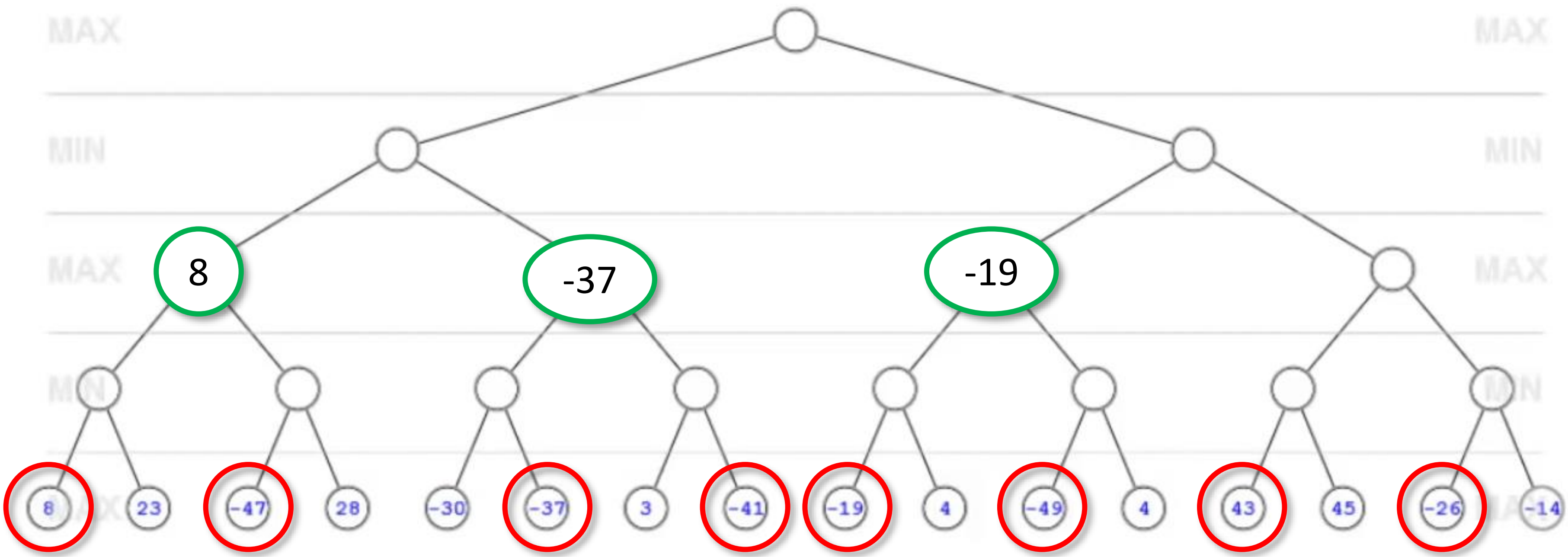


MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
8
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax

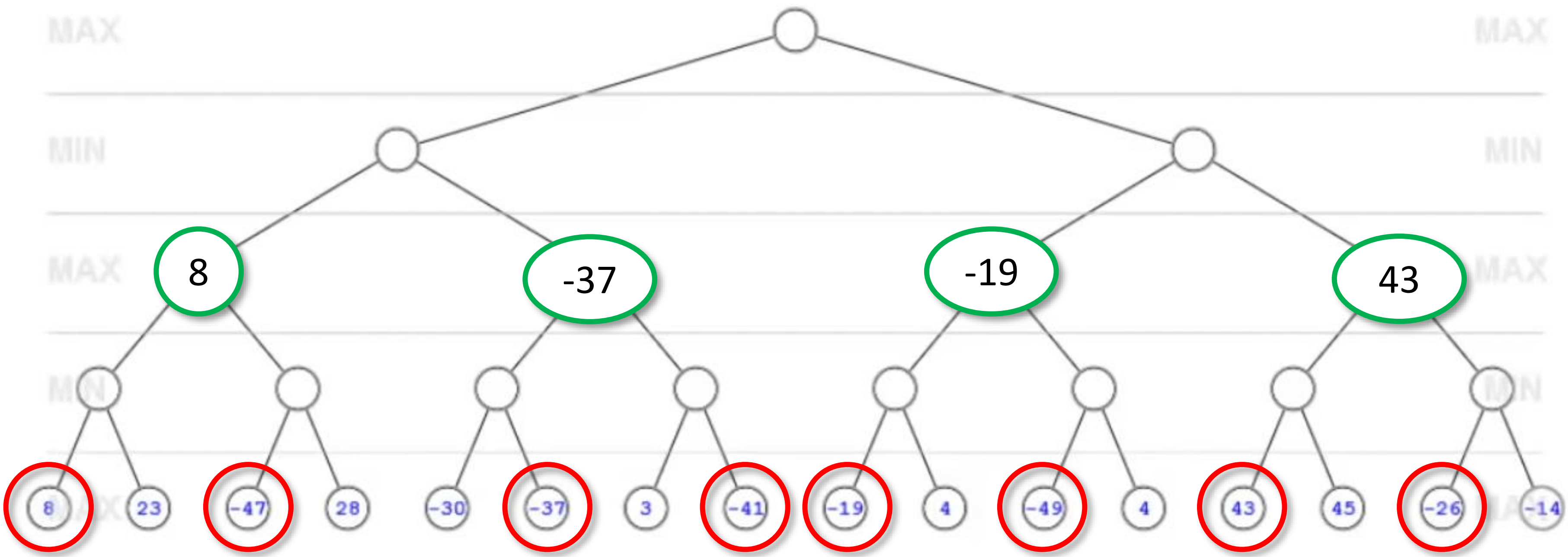

MAX
MAX
MIN
MIN
8
-37
MAX
MAX
MIN
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax

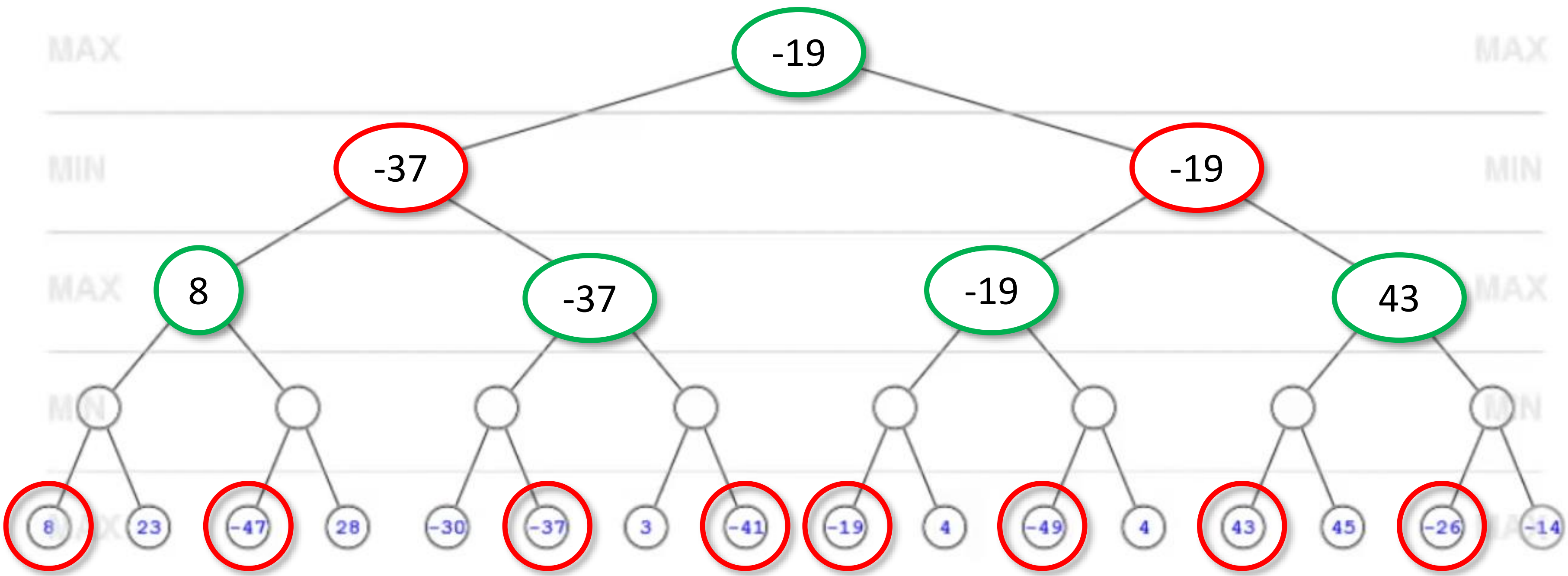


MAX
MAX
MIN
MIN
MAX
MAX
8
-37
-19
MIN
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax


MAX
MAX
MIN
MIN
8
-37
-19
43
MAX
MAX
MIN
MIN
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax

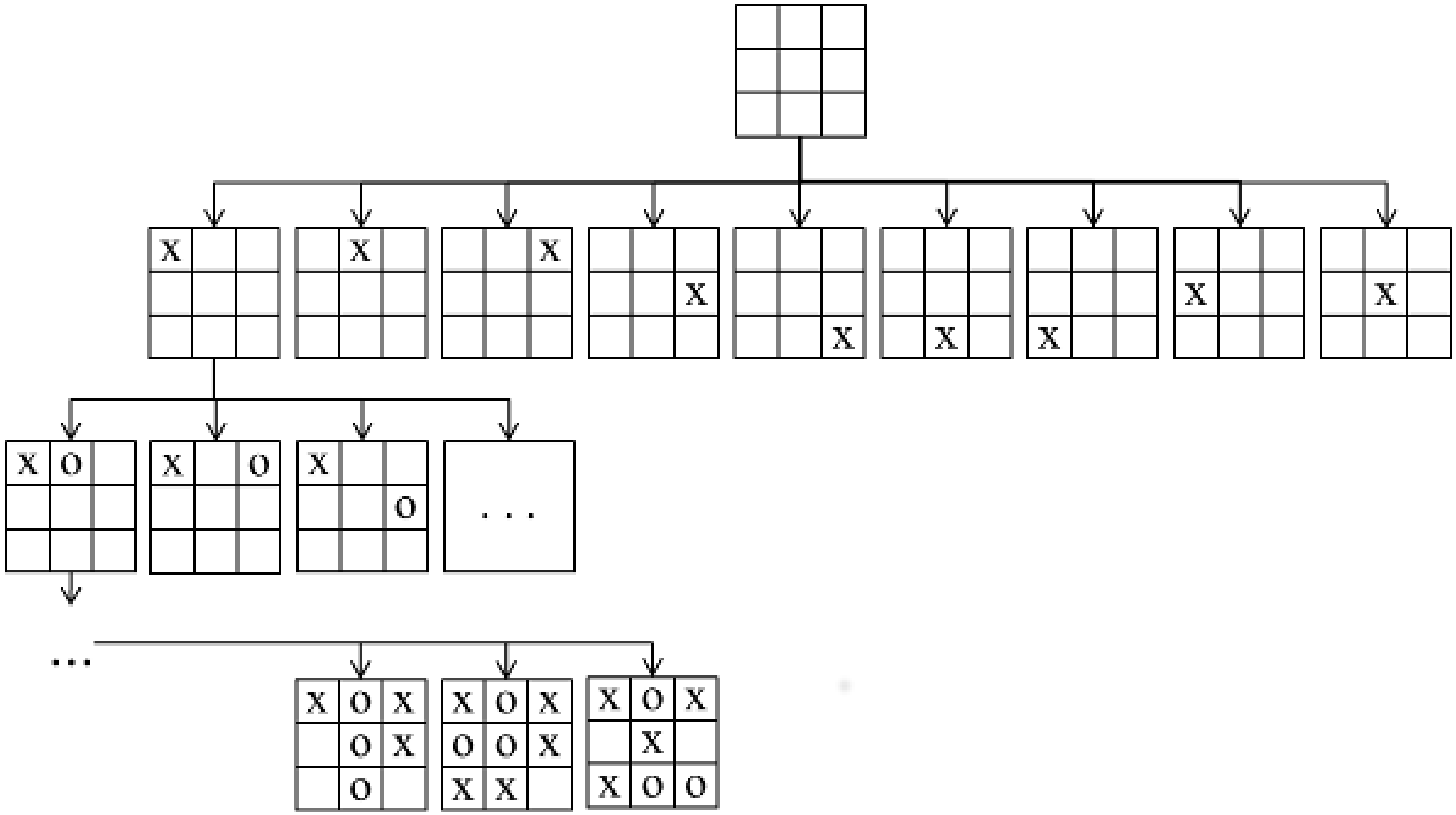

MAX
MIN
MAX
MIN
MAX
MIN
MAX
-19
-37
-19
8
-37
-19
43
8
23
-47
28
-30
-37
3
-41
-19
4
-49
4
43
45
-26
-14

Minimax

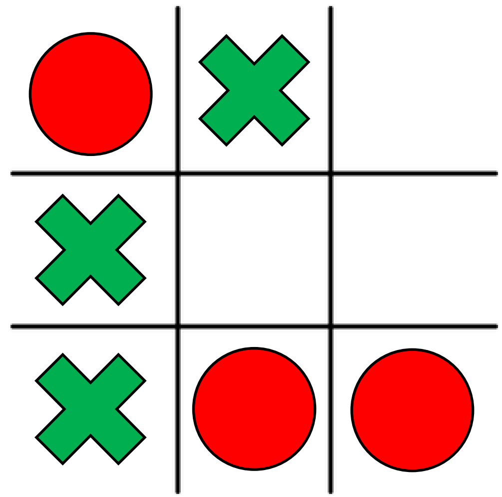
Algoritmo 11: Algoritmo Minimax

```
função minimax(no, profundidade, MAX)
   se (profundidade == 0) ou (no for é um nó terminal)
      retorne valor_utilidade(no)

   se MAX   //MAX está jogando
      melhorValor <-- -∞
      para cada filho de no
         v <-- minimax(filho, profundidade - 1, Falso)
         melhorValor <-- max(melhorValor, v)
      retorne melhorValor

   senão   //MIN está jogando
      melhorValor <-- +∞
      para cada filho de no
         v <-- minimax(filho, profundidade - 1, Verdadeiro)
         melhorValor <-- min(melhorValor, v)
      retorne melhorValor
```

Minimax



Minimax



Jogo das Moedas

- Dado um conjunto composto por N moedas, cada jogador pode retirar 1, 2 ou 3 moedas em cada movimento
- As jogadas são alternadas entre os competidores
- Perde quem retirar a última moeda

Jogo das Moedas

- Estado inicial: conjunto com as N moedas
- Ações possíveis:
  - Retirar 1 moeda
  - Retirar 2 moedas
  - Retirar 3 moedas
- Estado final: conjunto vazio
- Função de utilidade:
  - Retorna -1 se A (Min) vence
  - Retorna +1 se B (Max) vence

unioeste
Universidade Estadual do Oeste do Paraná

Jogo das Moedas

Exemplo: Conjunto composto por 5 moedas

Jogo Nim

- Dois oponentes se enfrentam utilizando um pilha de cartas
- A cada movimento um jogador deve dividir a pilha de fichas em duas pilhas não vazias de tamanhos diferentes
- Aquele que não puder realizar um movimento de divisão com tamanho diferentes perde o jogo

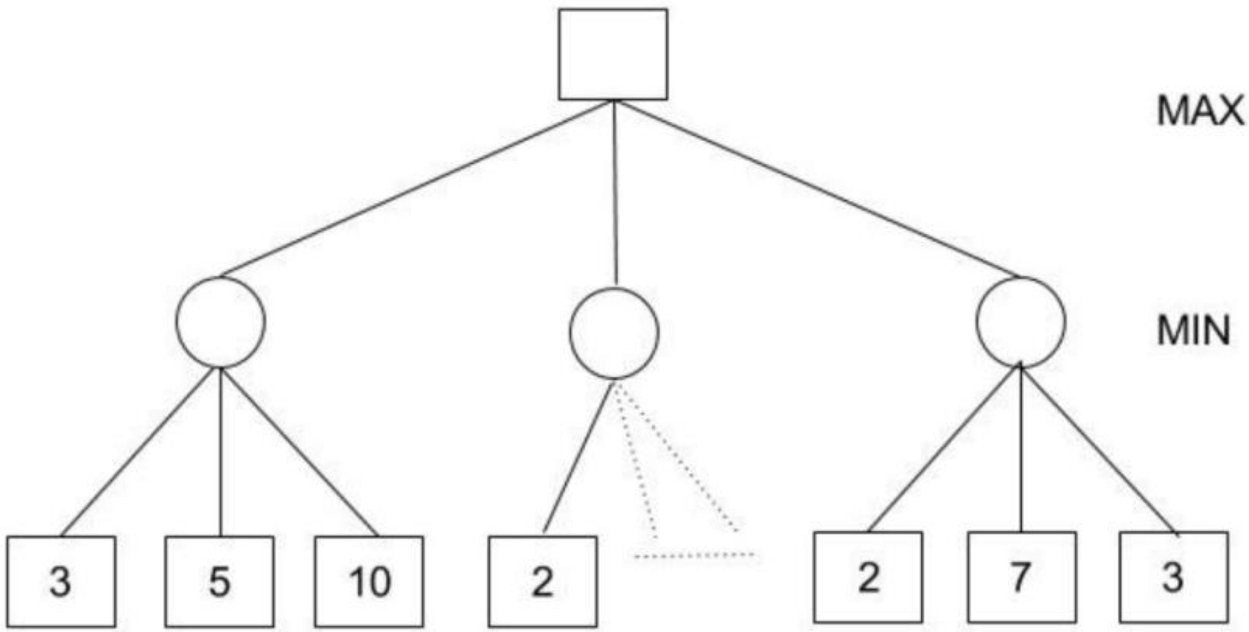
Jogo Nim

Exemplo: conjunto composto por 7 cartas

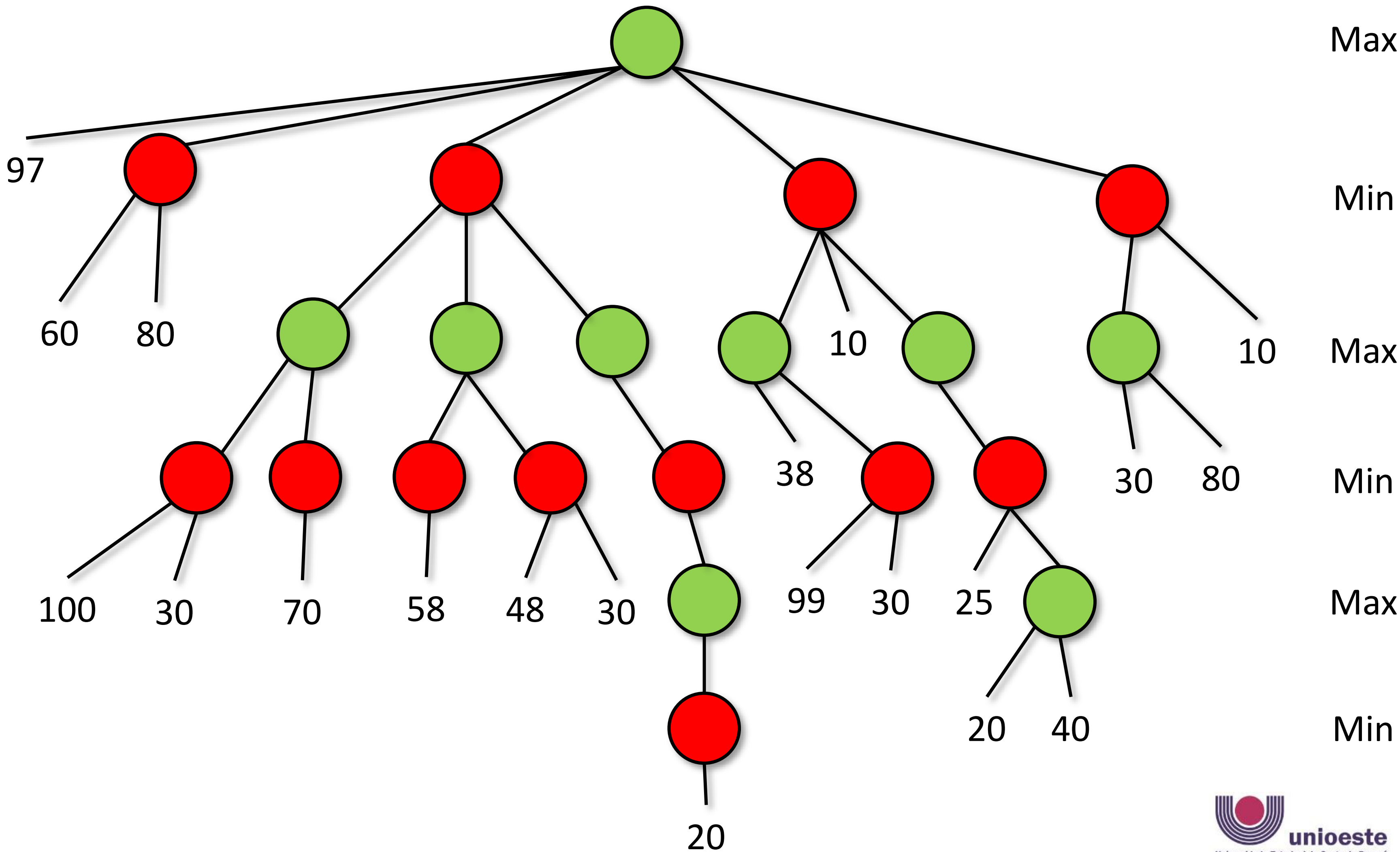
Poda Alfa-Beta

- Em geral, as árvores de jogos consomem muito tempo para construir
- É possível encontrar a decisão minimax real sem olhar para todos os nós da árvore do jogo através do processo de poda
- Aplicar a poda-alfa a um algoritmo minimax padrão fará com que ele faça o mesmo movimento padrão mas sem visitar nós que não afetam a decisão final

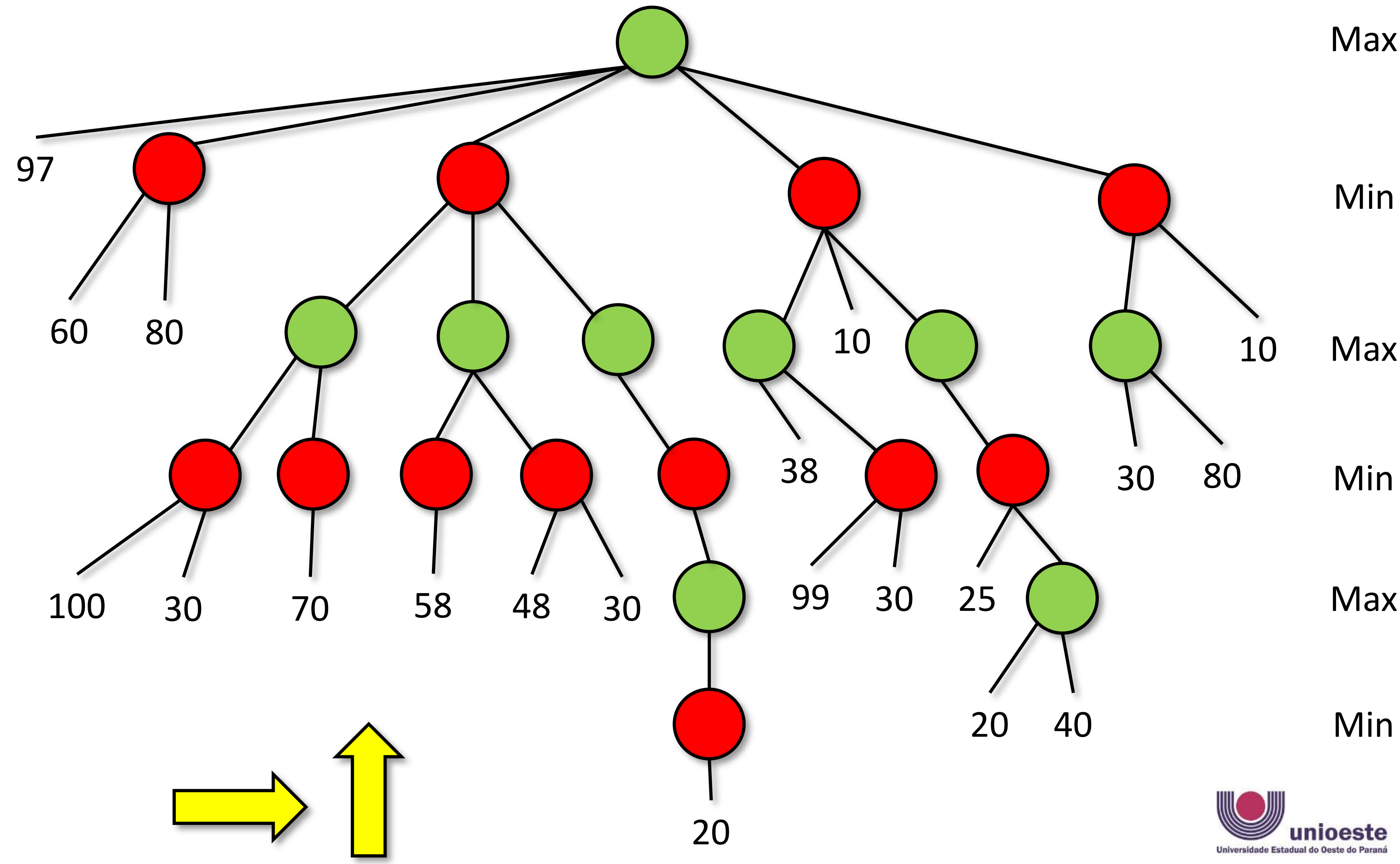
Poda Alfa-Beta



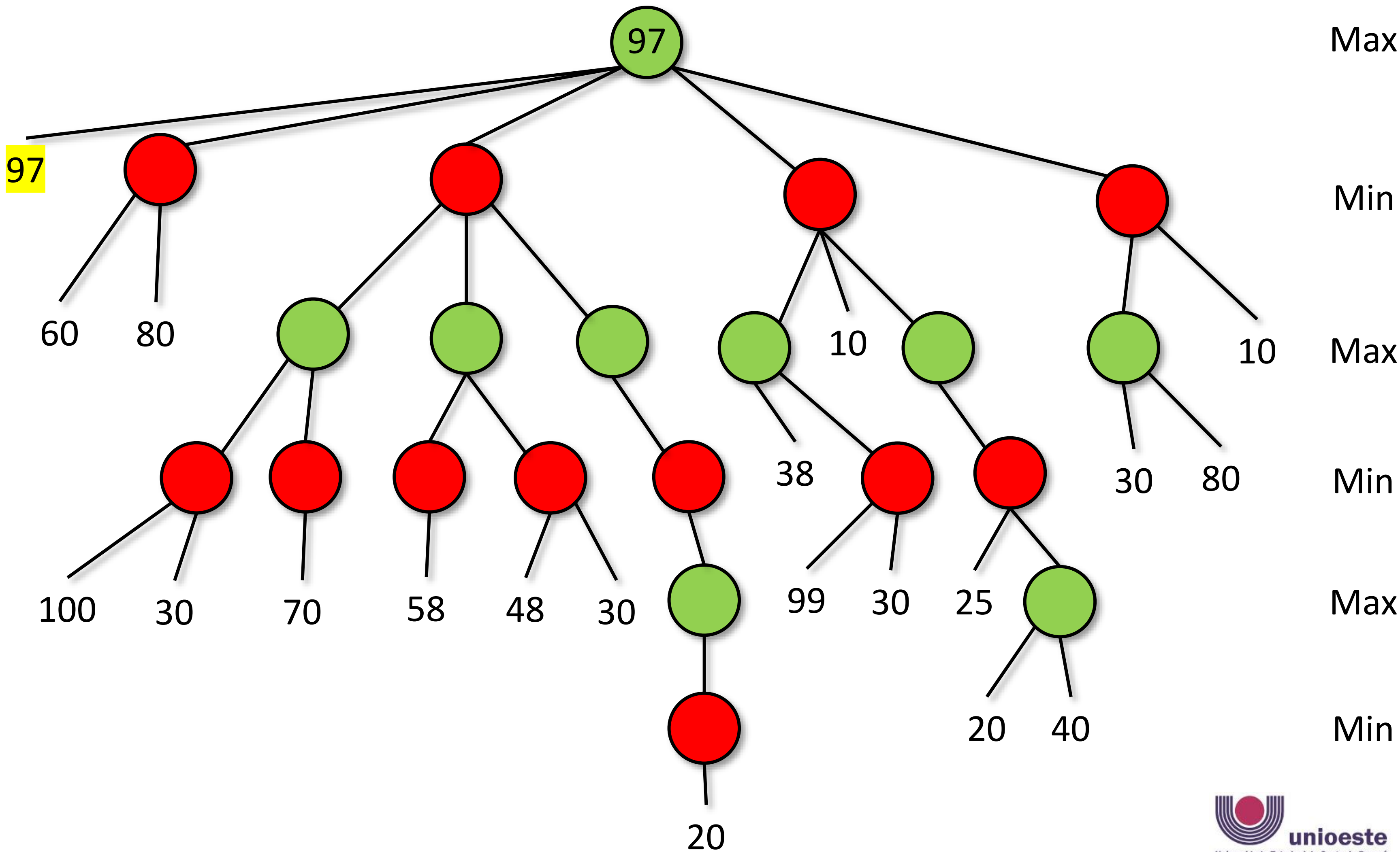
Poda Alfa-Beta



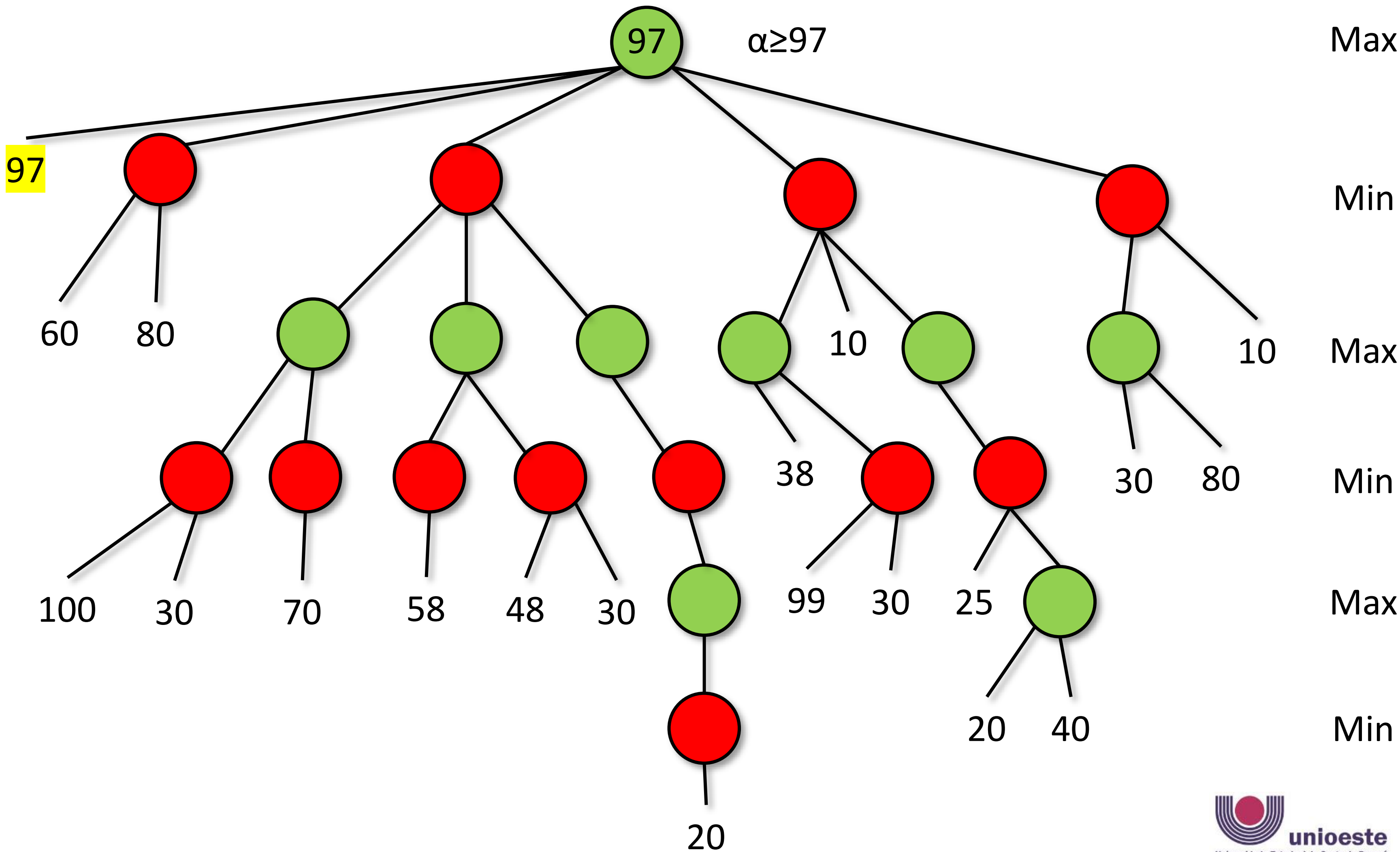
Poda Alfa-Beta



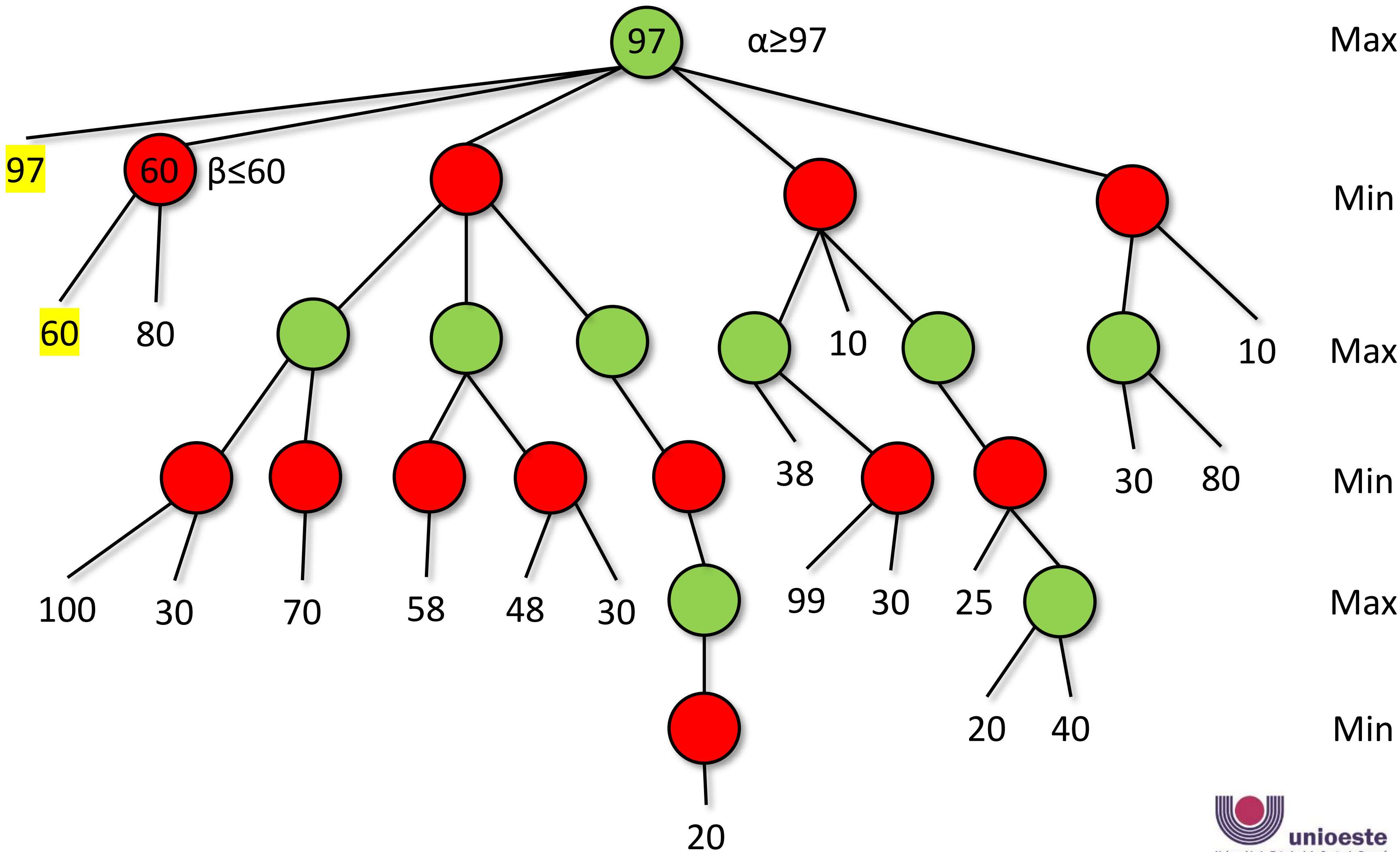
Poda Alfa-Beta



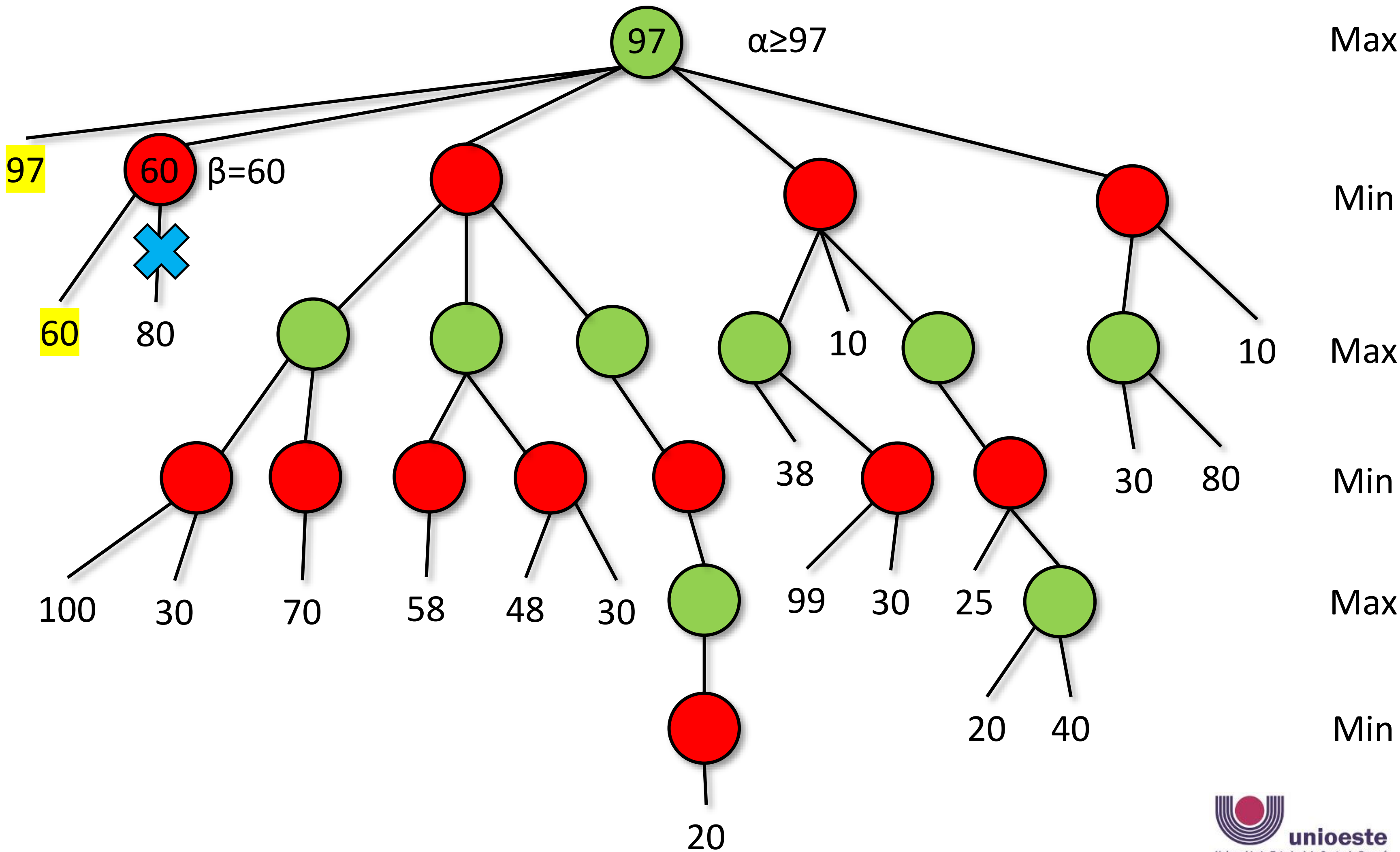
Poda Alfa-Beta



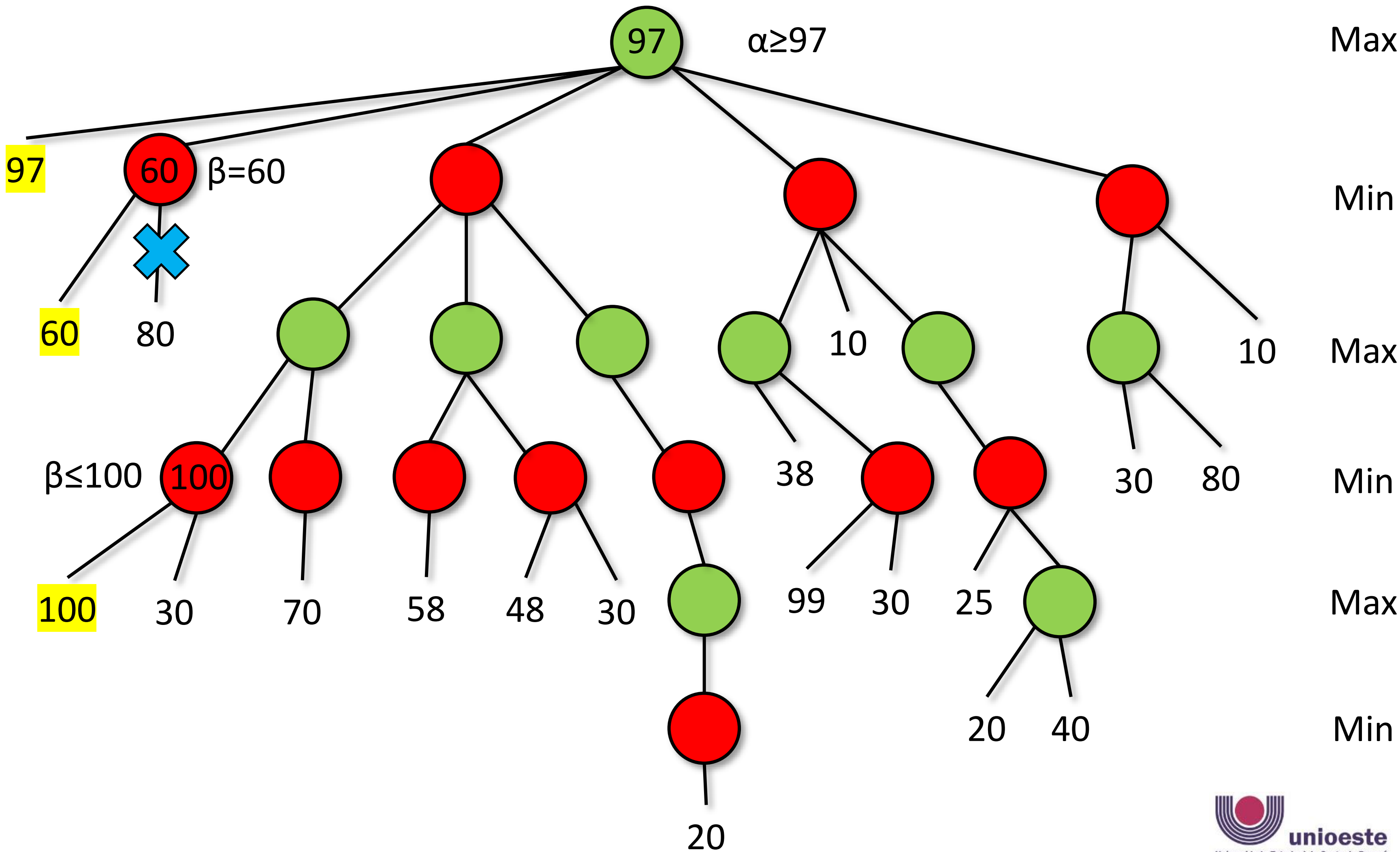
Poda Alfa-Beta



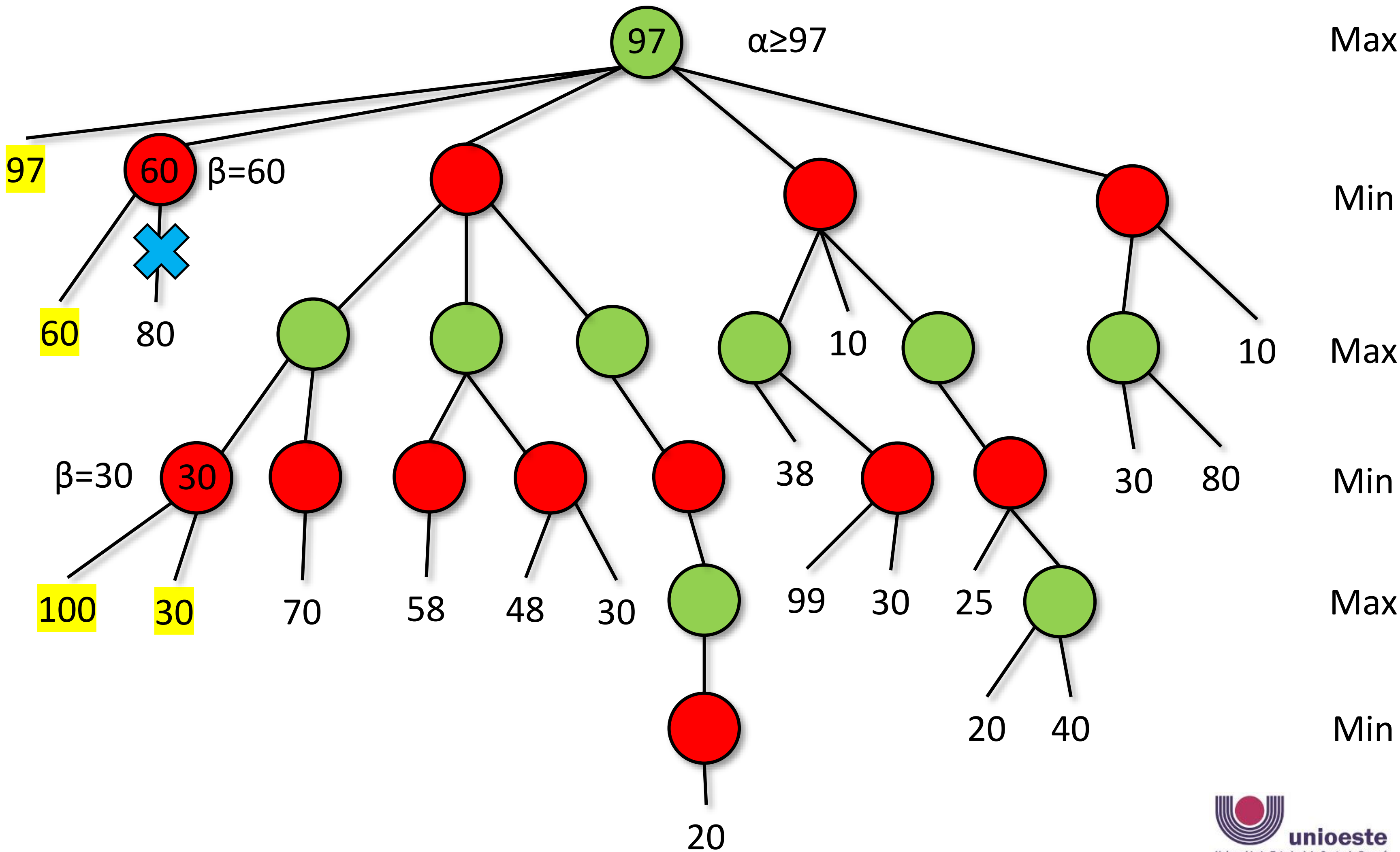
Poda Alfa-Beta



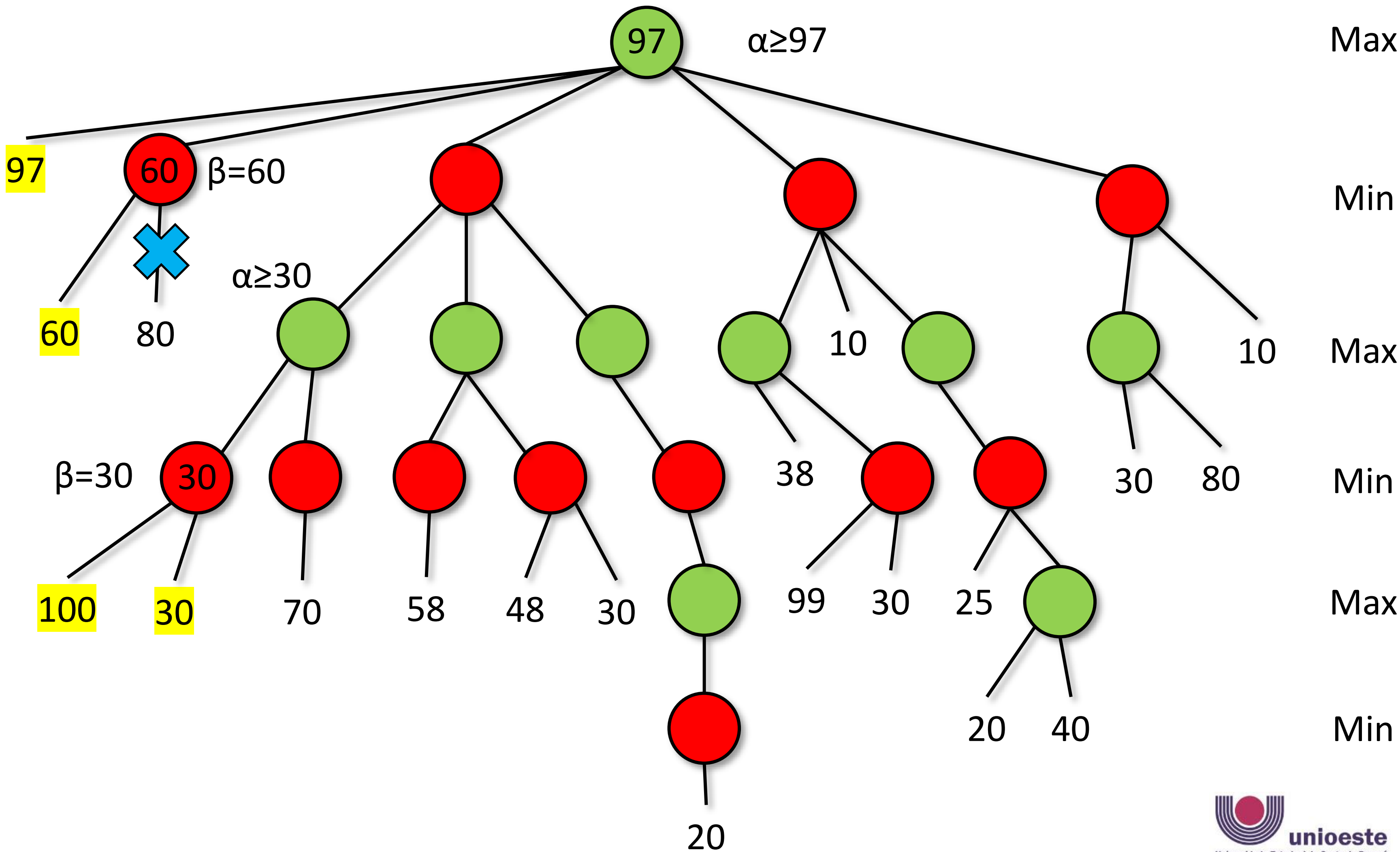
Poda Alfa-Beta



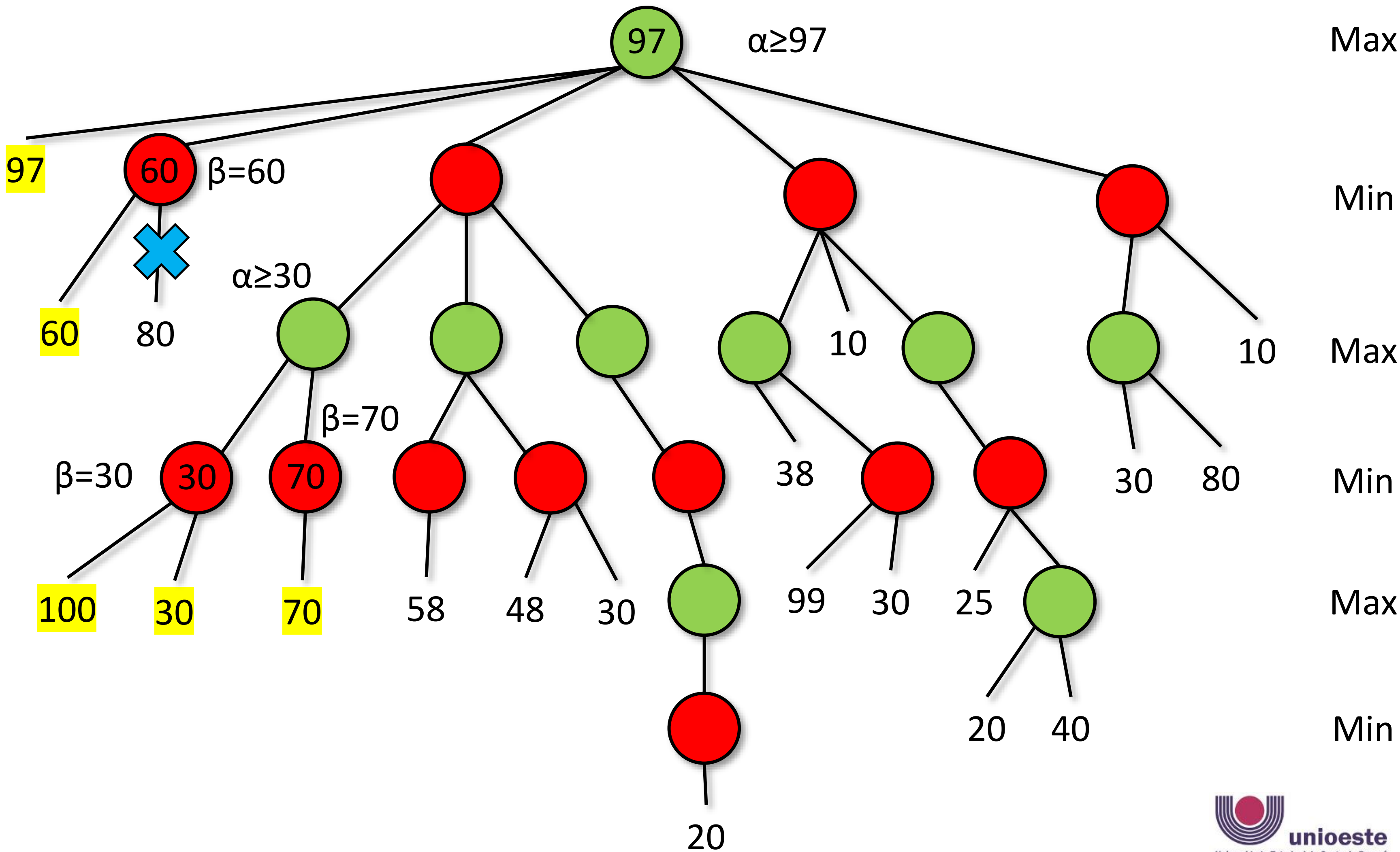
Poda Alfa-Beta



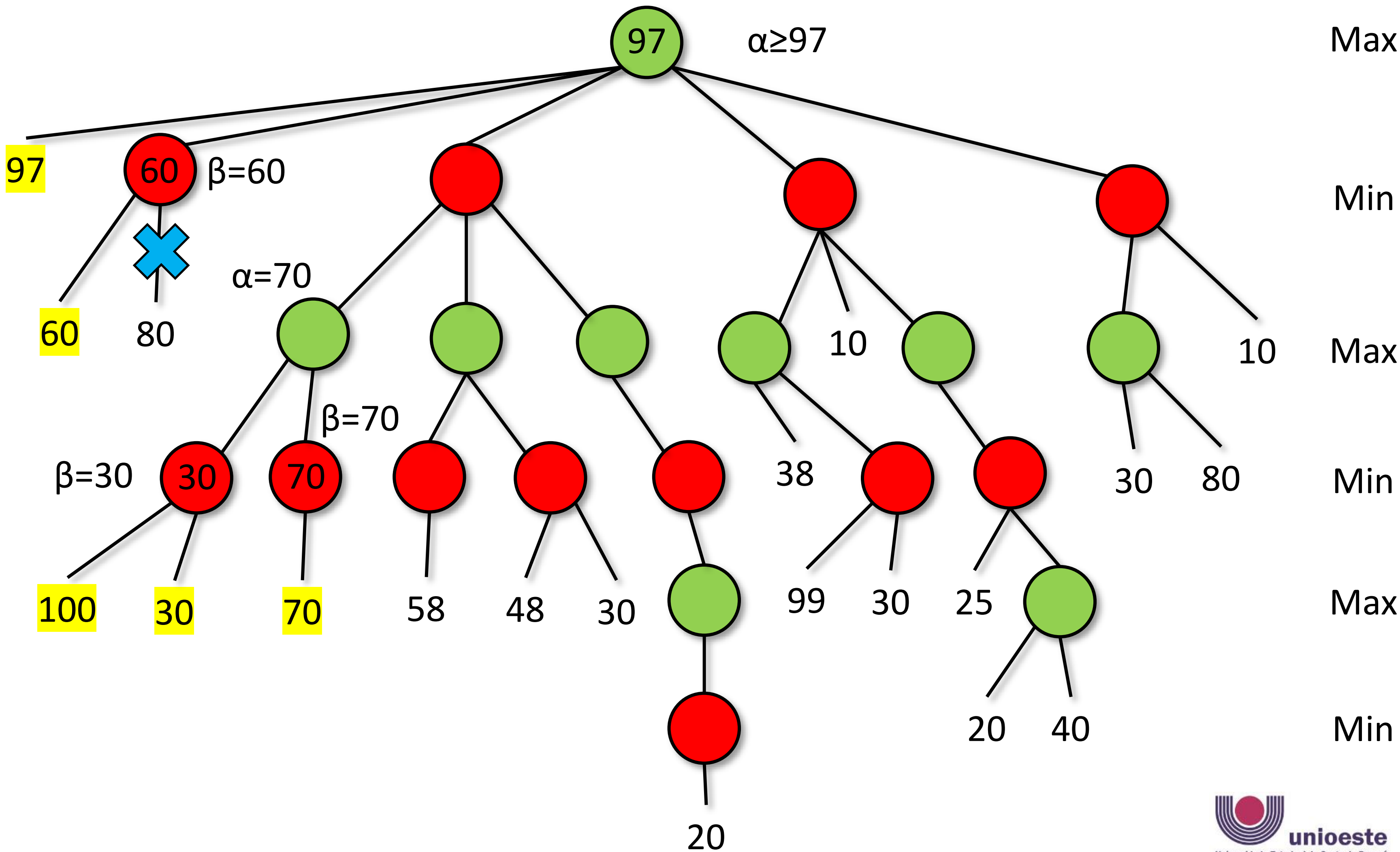
Poda Alfa-Beta



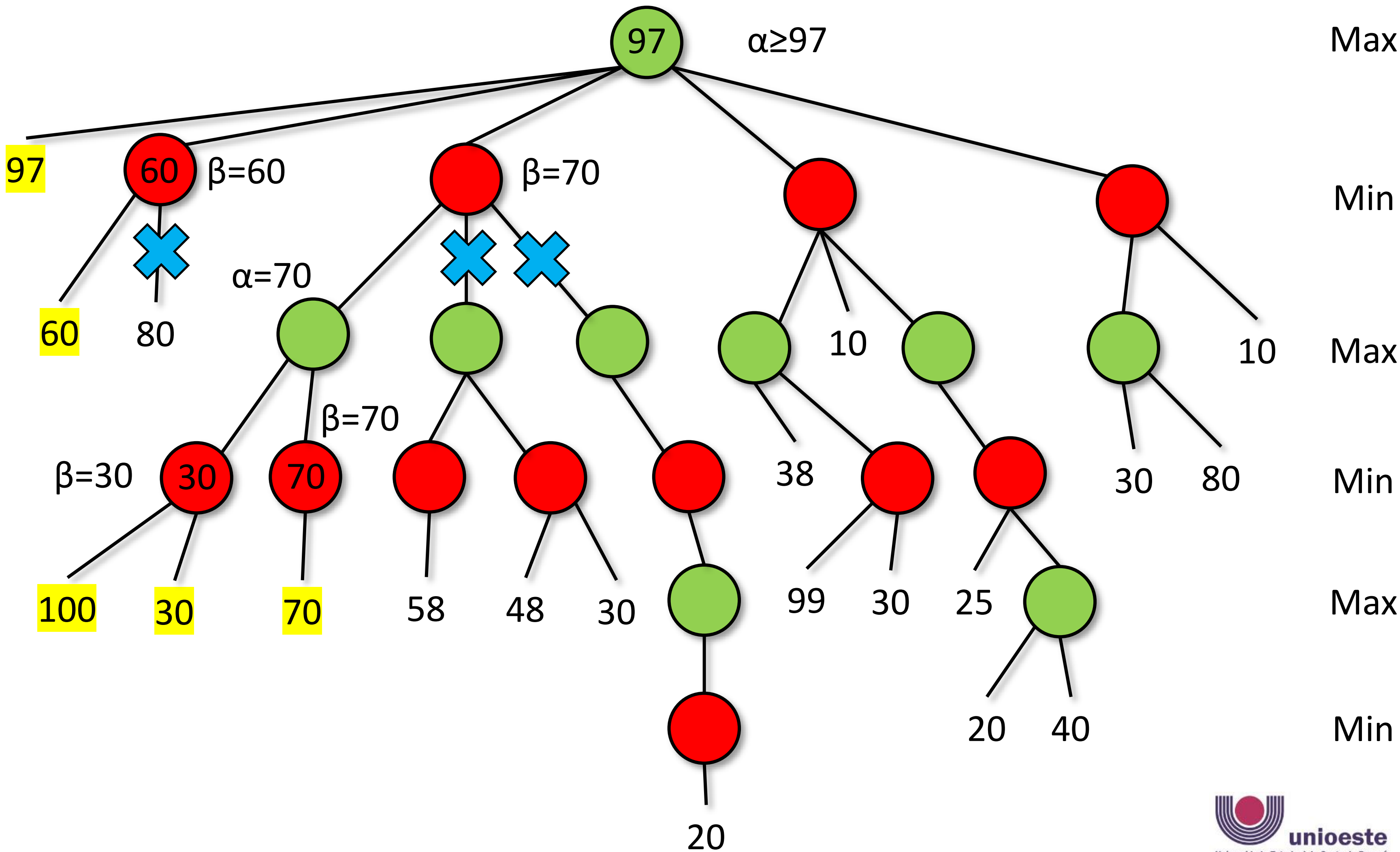
Poda Alfa-Beta



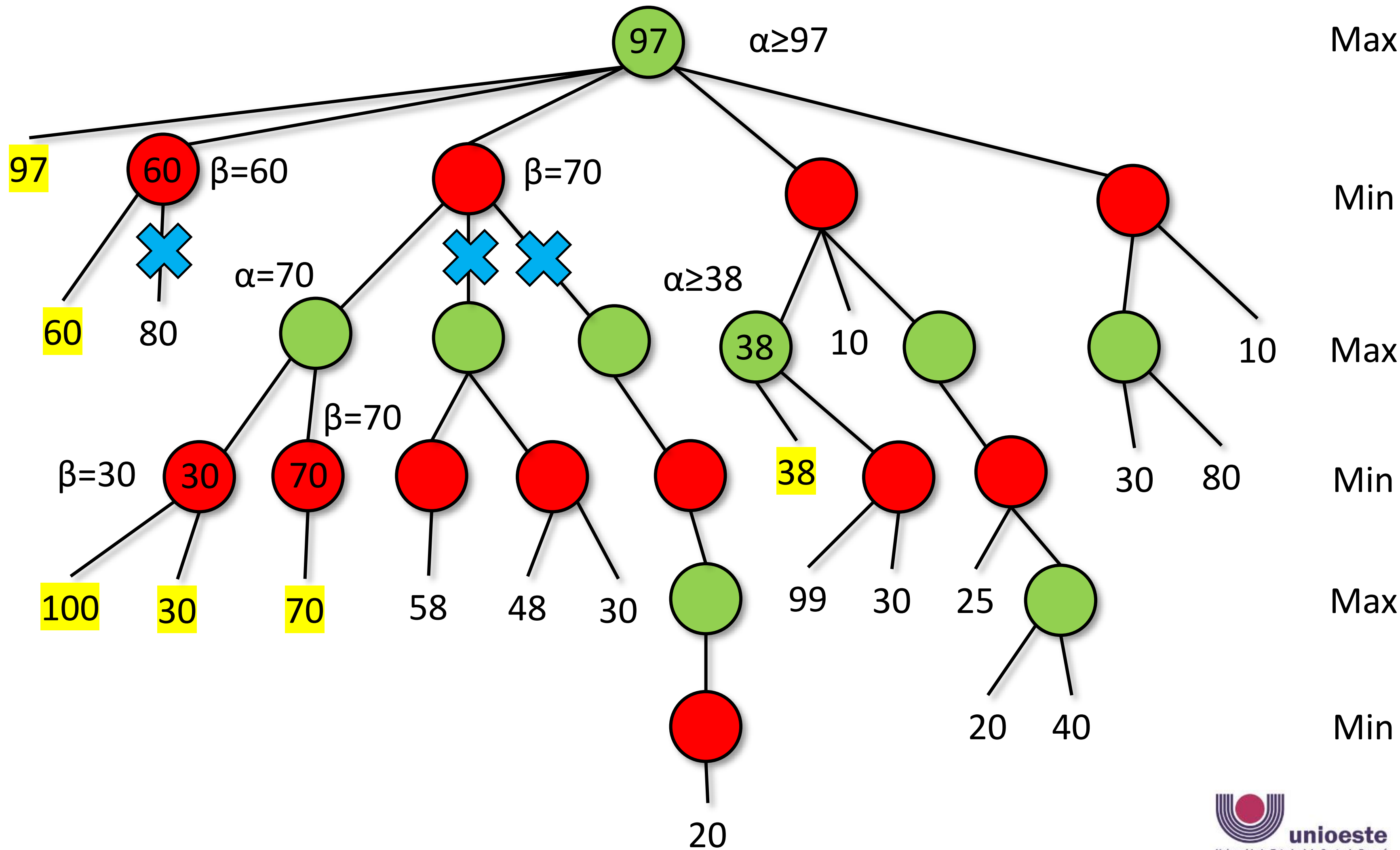
Poda Alfa-Beta



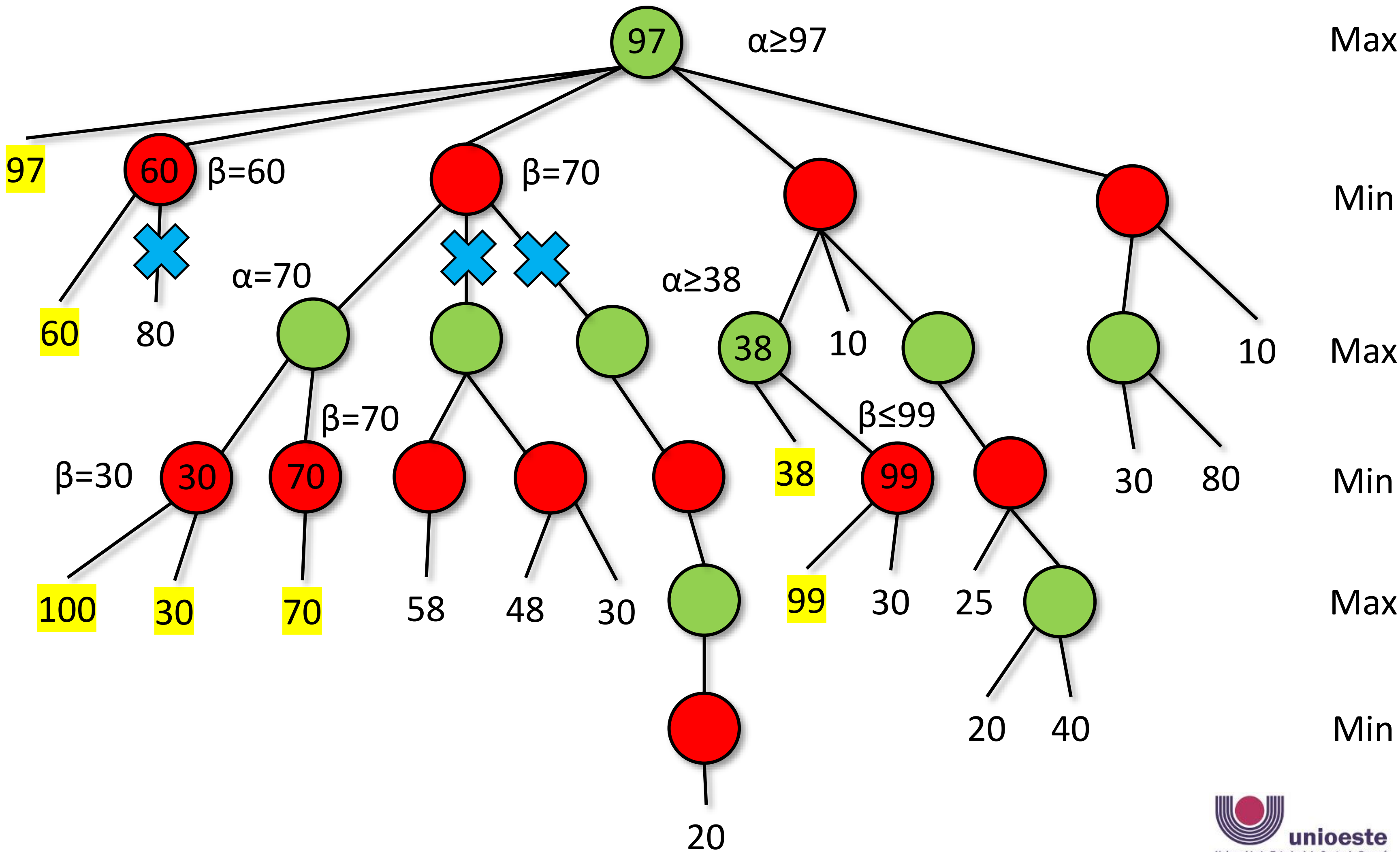
Poda Alfa-Beta



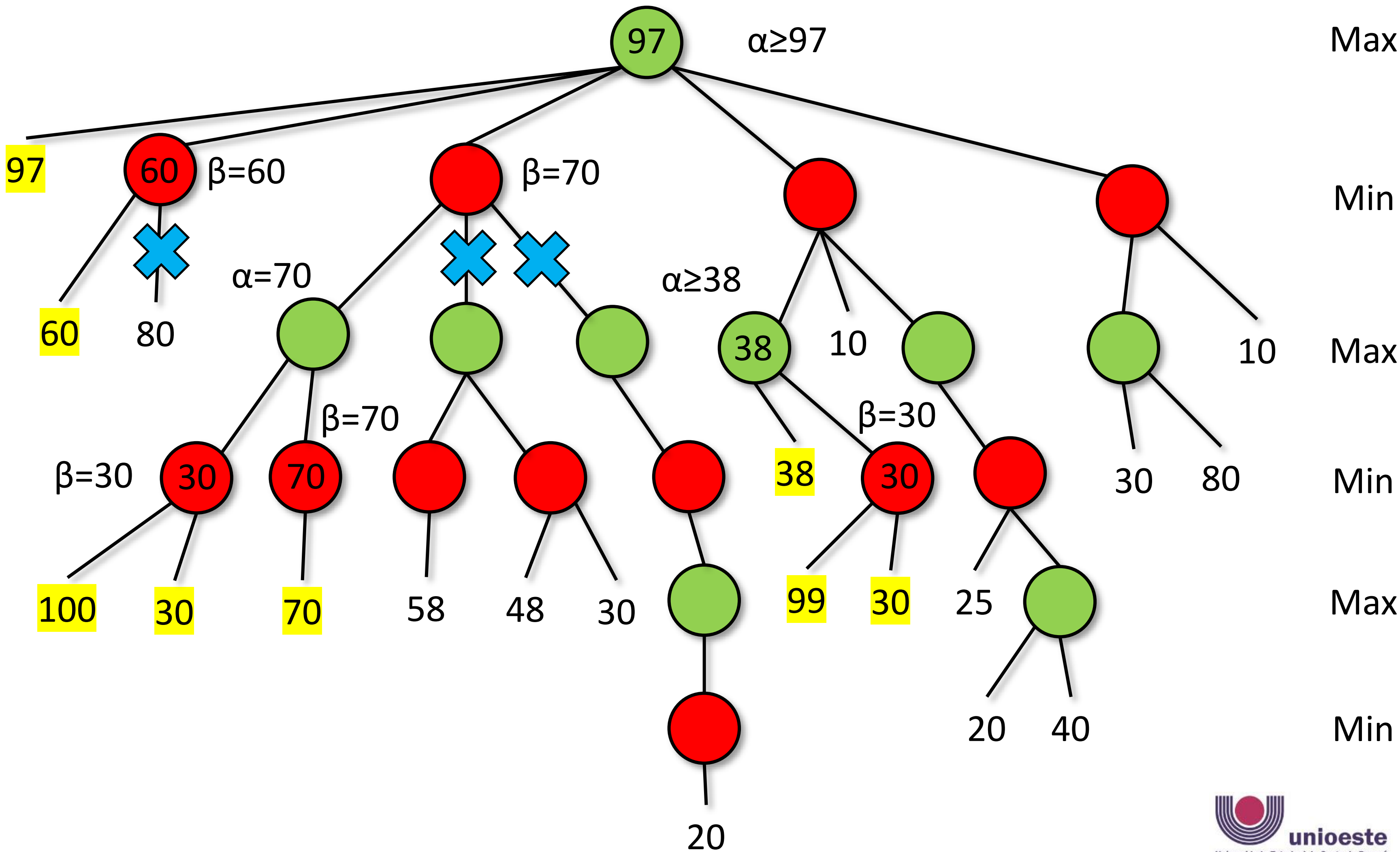
Poda Alfa-Beta



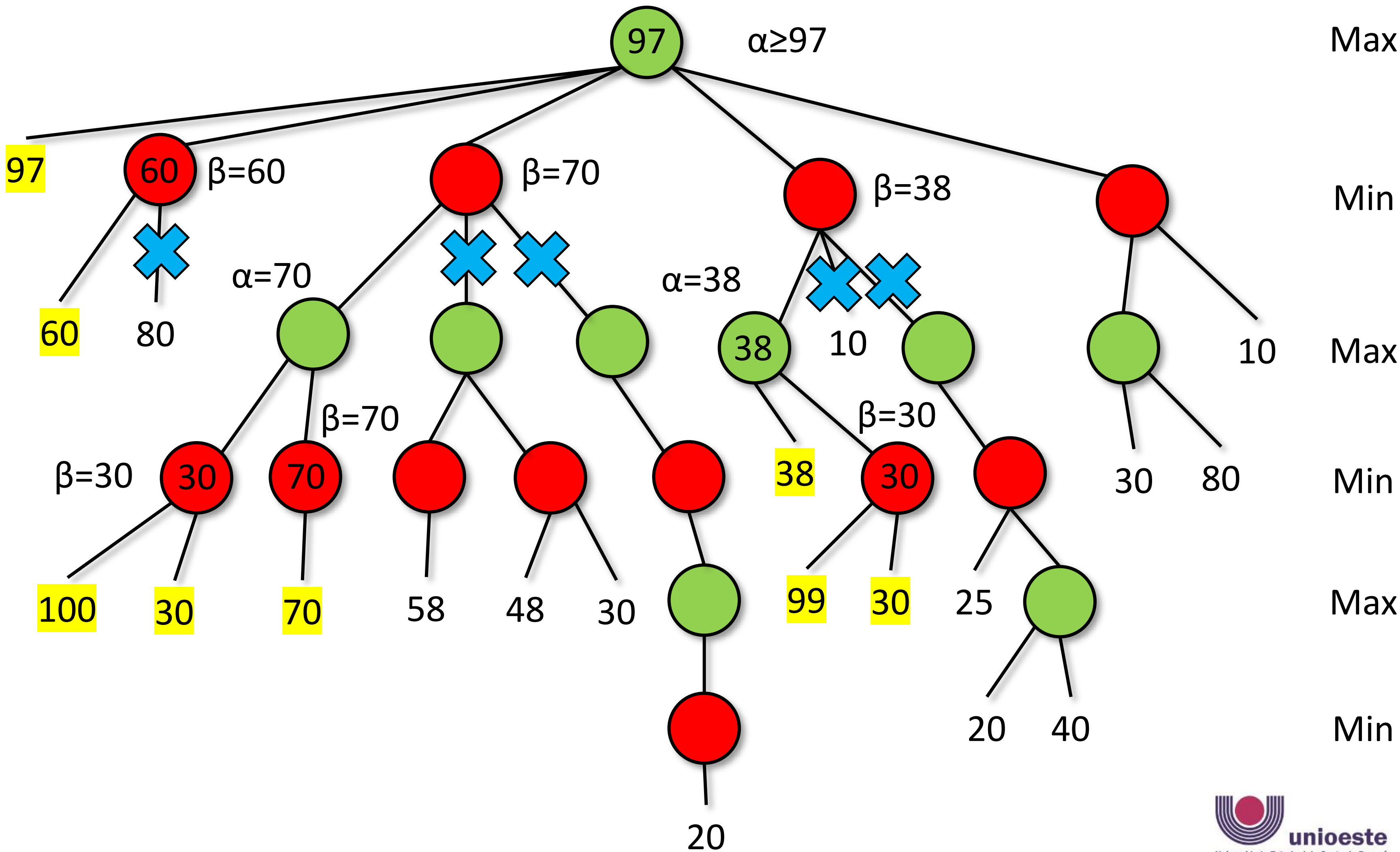
Poda Alfa-Beta



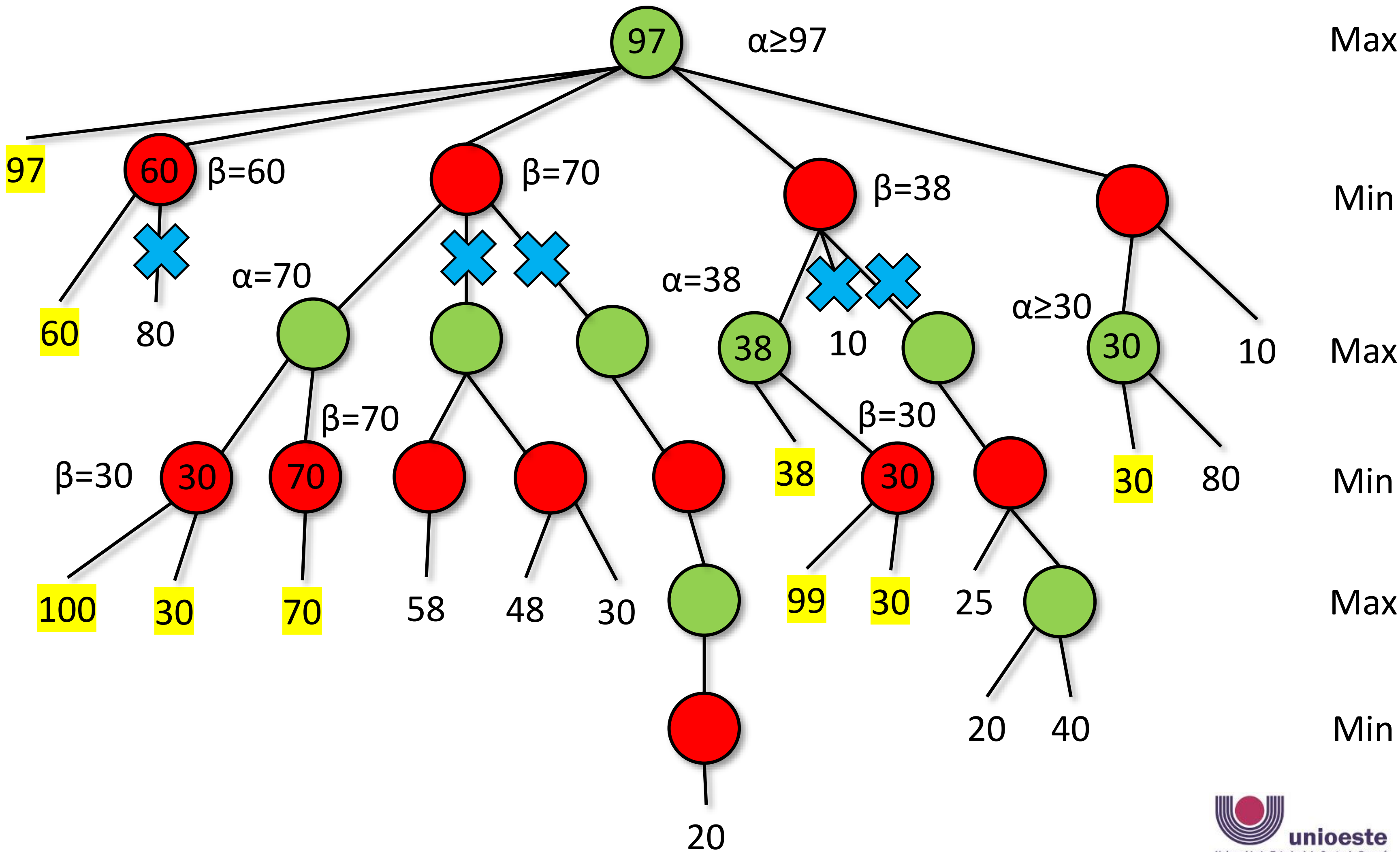
Poda Alfa-Beta



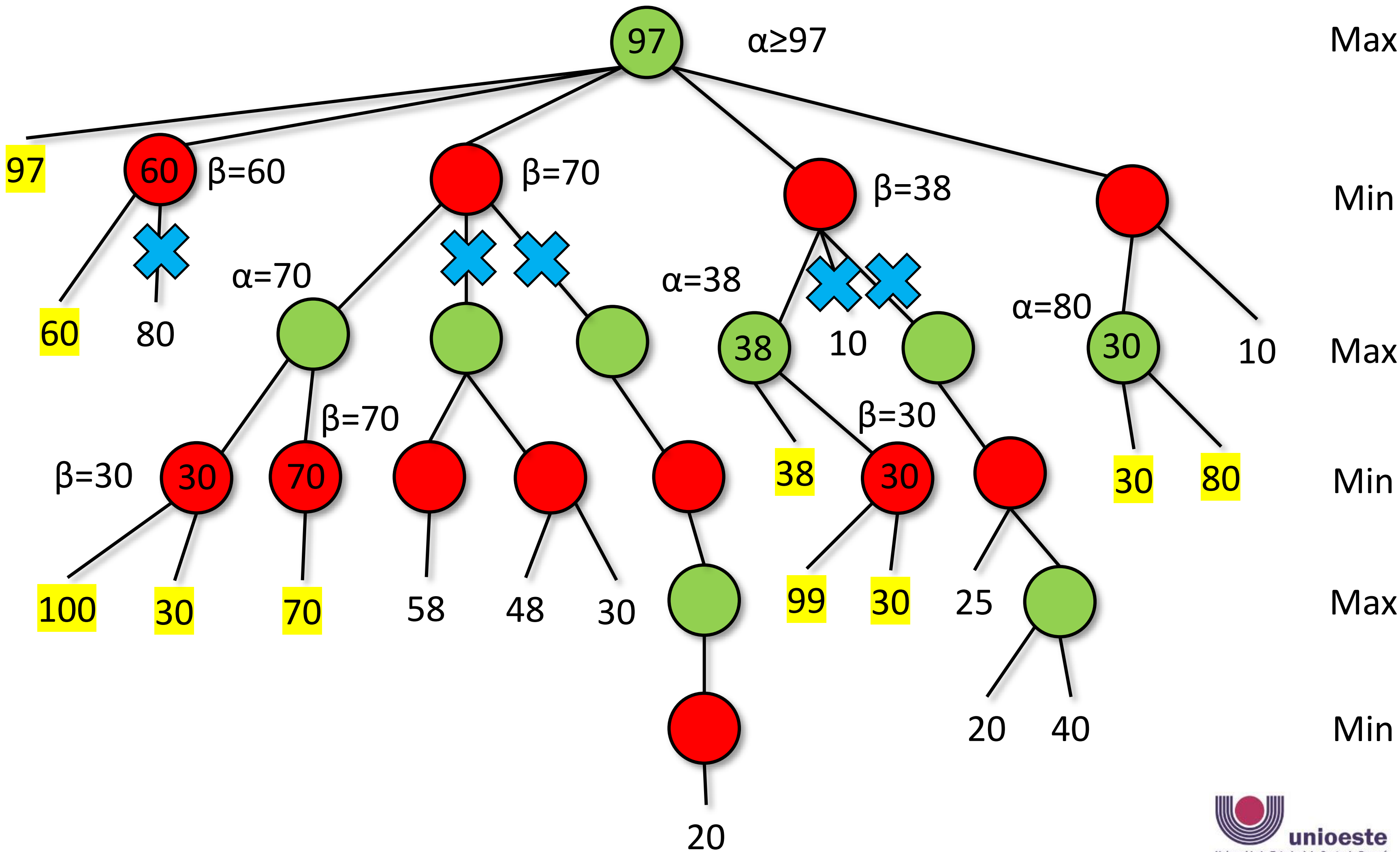
Poda Alfa-Beta



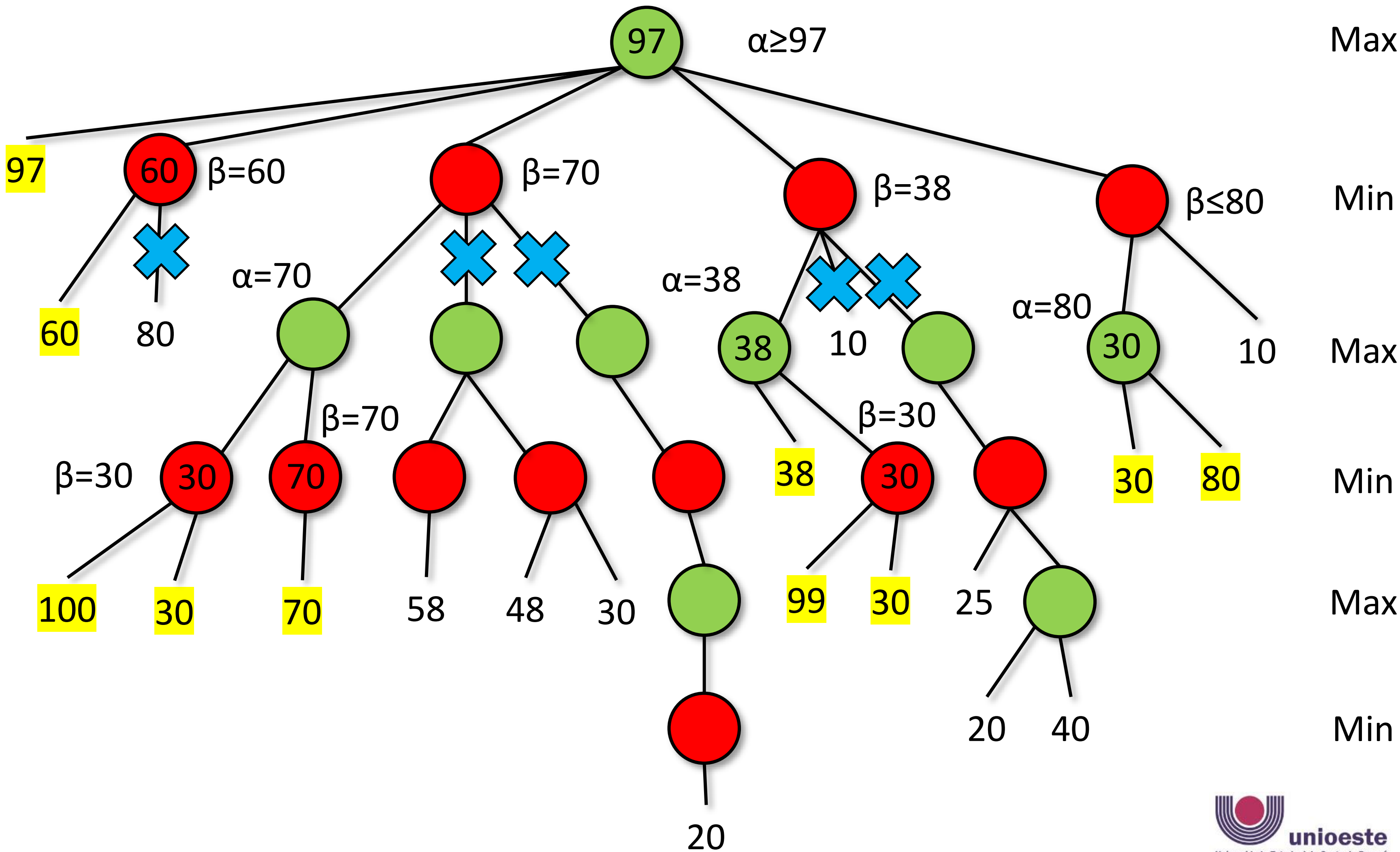
Poda Alfa-Beta



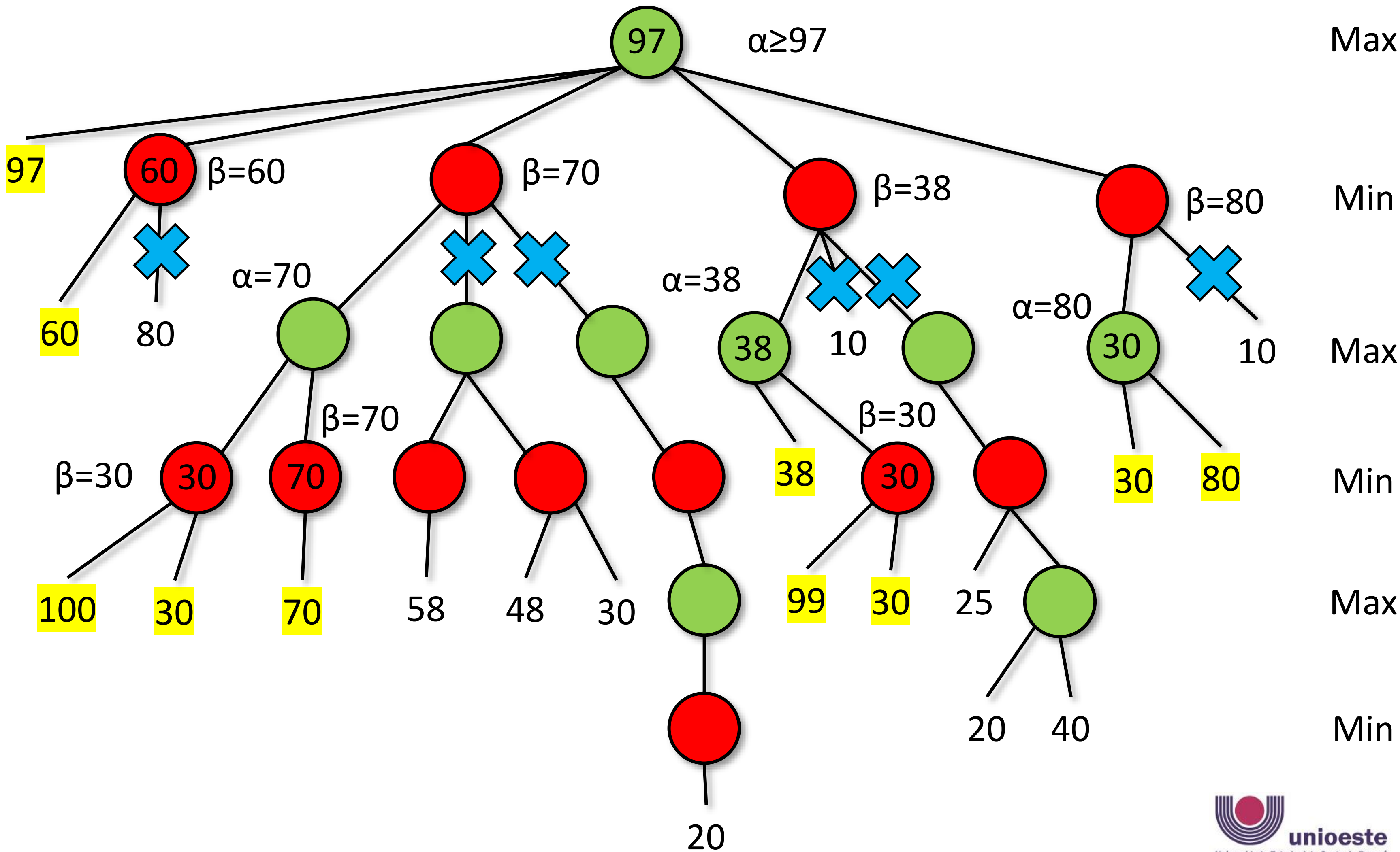
Poda Alfa-Beta



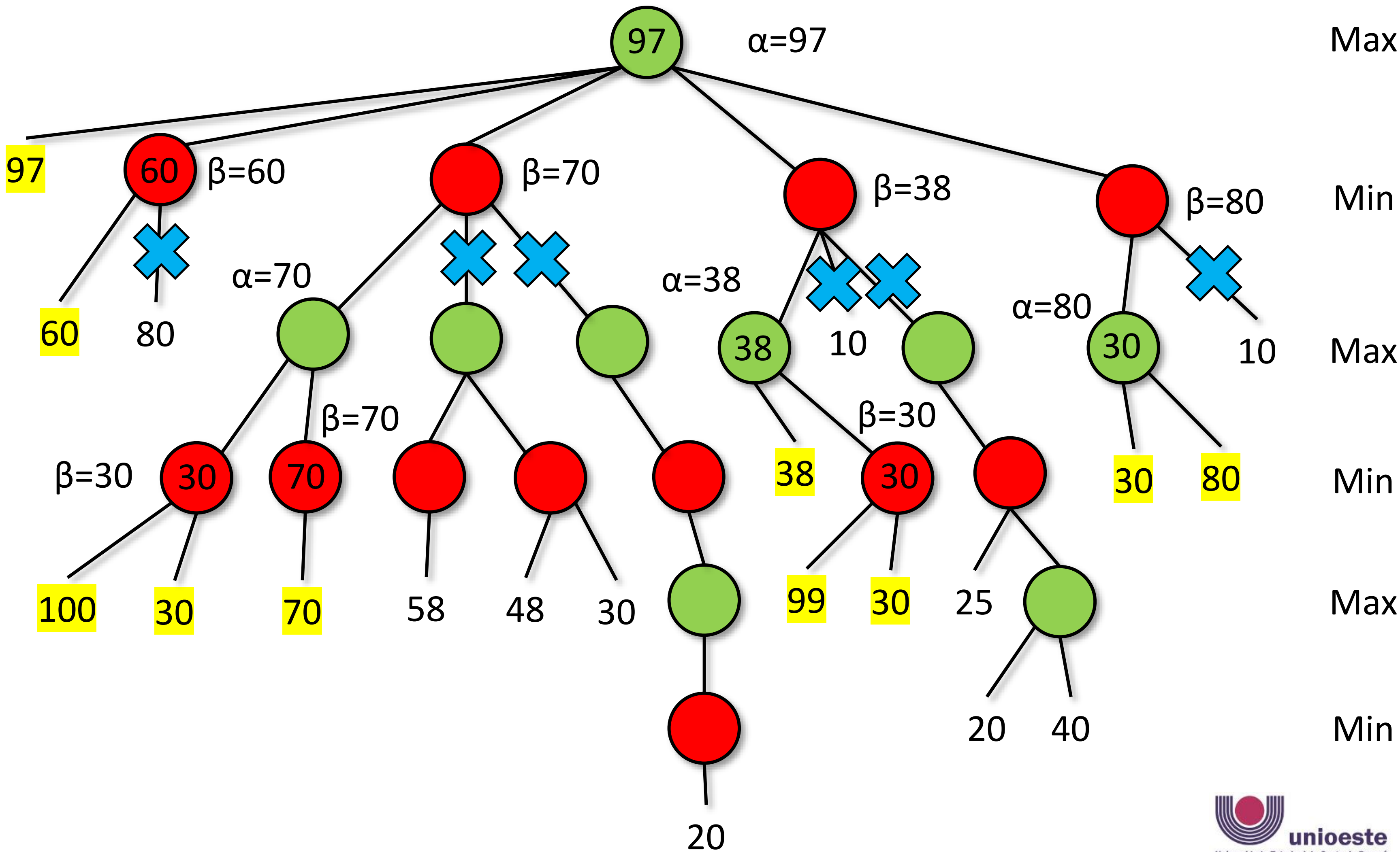
Poda Alfa-Beta



Poda Alfa-Beta



Poda Alfa-Beta



Poda Alfa-Beta

20,33,-45,31,24,25,-10,20,46,-25,10,-42,24,-19,36,-41

5 níveis

Métricas

<u>Distância de Manhattan</u>

- É a heurística padrão em grades quadradas

```
função Manhattan(atual)
dx = abs(atual.x – objetivo.x)
dy = abs(atual.y – objetivo.y)
retorne D*(dx+dy)
```

D = distância entre vizinhos

Métricas

Distância de Manhattan

Métricas

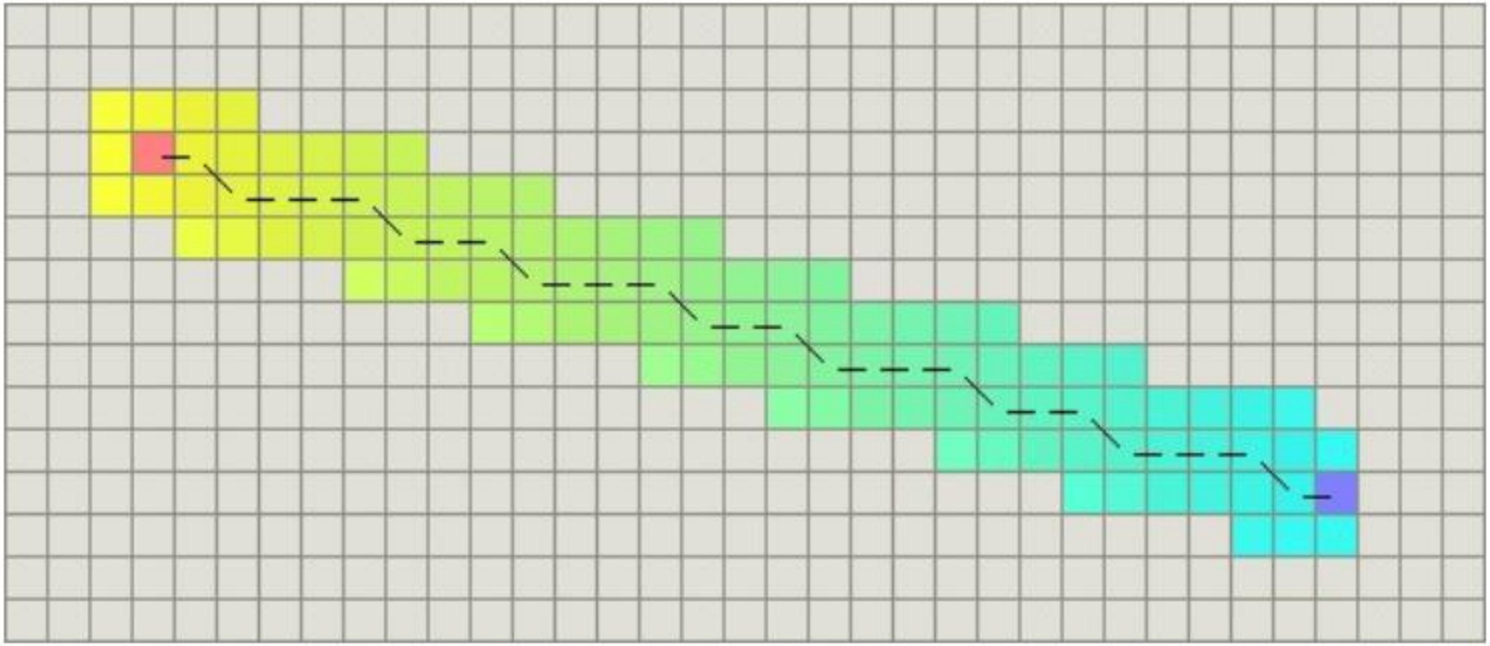
<u>Distância Diagonal</u>

- Usada quando é possível mover-se na diagonal dentro da grade
- O valor D2 é o custo de mover-se na diagonal.

```
função Diagonal(atual)
dx = abs(atual.x – objetivo.x)
dy = abs(atual.y – objetivo.y)
retorne D*(dx+dy)+(D2-2*D)*min(dx, dy)
```

Métricas

Distância Diagonal



Métricas

Distância Euclidiana

- Quando você pode mover-se para qualquer direção, a linha reta é a mais adequada

função Euclidiana(atual)

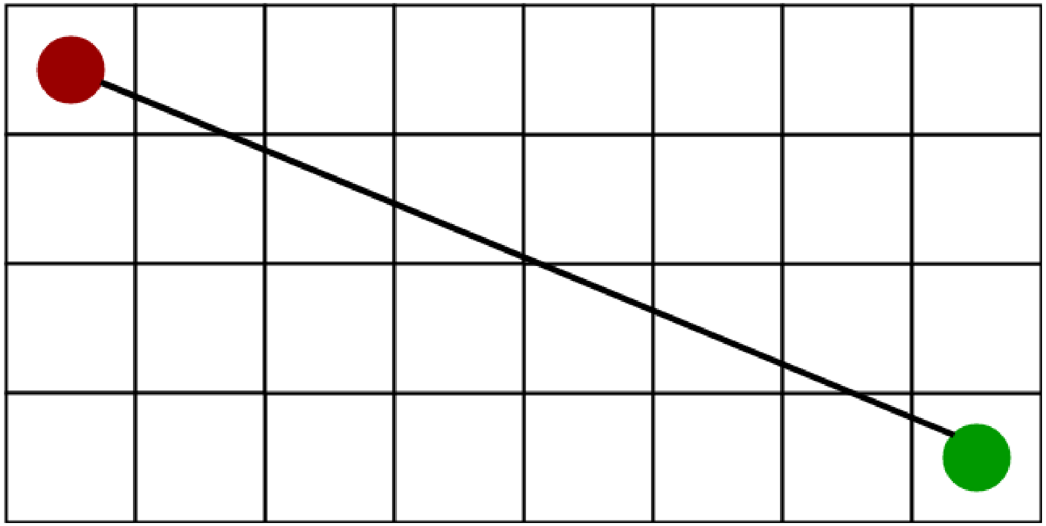
dx = abs(atual.x – objetivo.x)

dy = abs(atual.y – objetivo.y)

retorne D*sqrt(dx² + dy²)

Métricas

Distância Euclidiana



Referências

FARIA, F. A. Além da busca clássica. Notas de Aula. Universidade Federal de São Paulo, 2015.