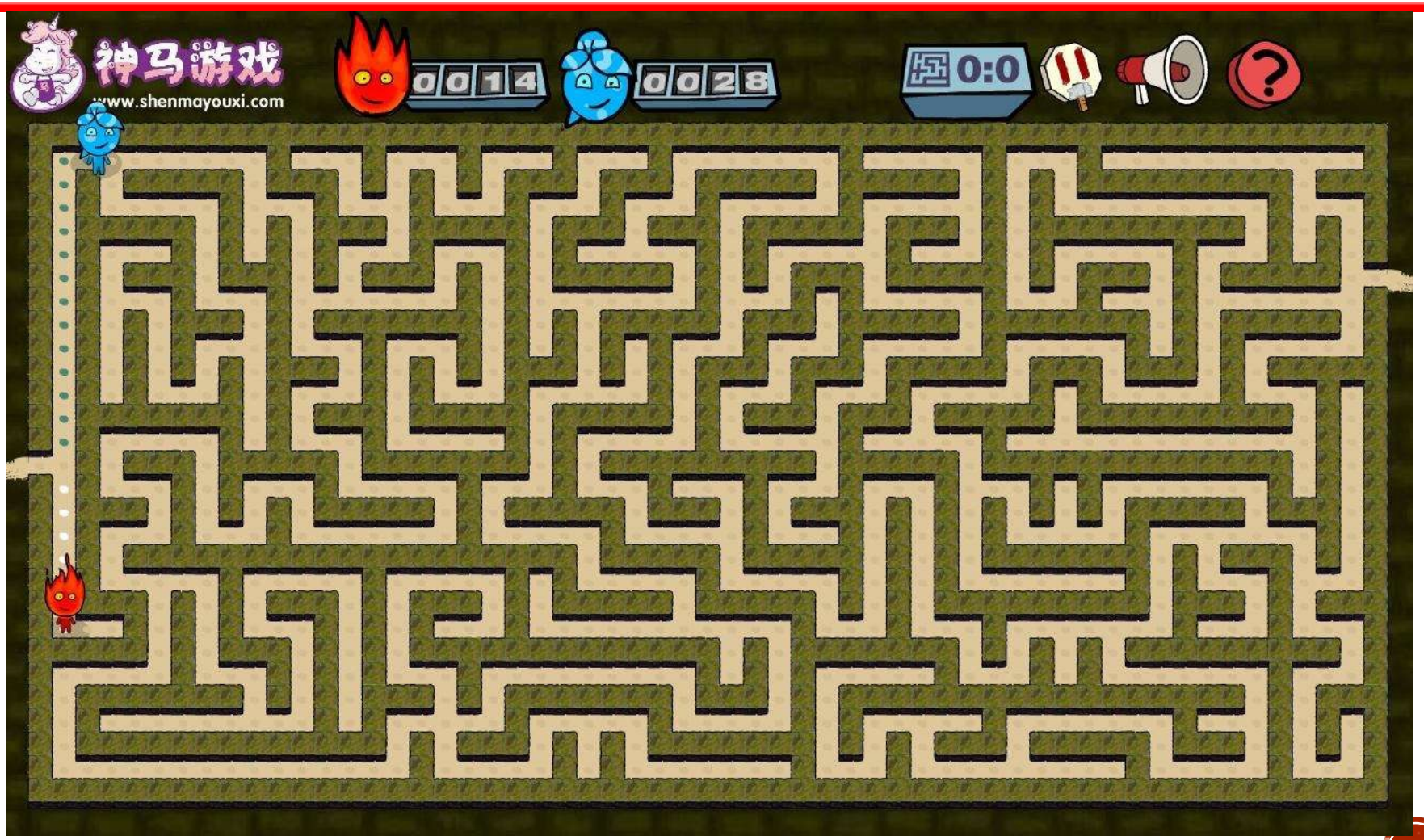
搜索技术

- 什么是搜索？

迷宫问题



旅行商问题TSP

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

计算机下棋


手談対局V
究極の囲碁
[対局名未設定]
2011年10月18日
白の第228手まで
黒6目勝
BLACK
名称未設定
時間:49分11秒
揚浜:2
WHITE
手談(最高級)
時間:13分50秒
揚浜:3
新局
悔回
棋谱
盘面
复盘
退出

什么是搜索

- Search is about choices
- Search is about the choices which you make when you are trying to make decisions

搜索求解策略

- 在求解一个问题时，涉及两个方面：

（1）该问题的表示，如果一个问题找不到合适的表示方法，就谈不上对它求解；

（2）选择一种相对合适的求解方法。由于绝大多数需要人工智能方法求解的问题缺乏直接求解的方法，因此，搜索为一种求解问题的一般方法。

搜索的基本问题与主要过程

- 搜索中需要解决的基本问题：

  (1) 是否一定能找到一个解。

  (2) 找到的解是否是最佳解。

  (3) 时间与空间复杂性如何。

  (4) 是否终止运行或是否会陷入一个死循环。

搜索技术

- 盲目搜索

  在不具有对特定问题的任何有关信息的条件下，按固定的步骤（依次或随机调用操作算子）进行的搜索。

- 启发式搜索

  考虑特定问题领域可应用的知识，动态地确定调用操作算子的步骤（优先选择较适合的操作算子）尽量减少不必要的搜索，以求尽快地到达结束状态

盲目搜索与启发式搜索

- 盲目搜索：
  - 深度优先搜索
  - 宽度优先搜索

- 启发式搜索
  - A算法
  - A*算法

盲目的图搜索策略

- 回溯策略
- 宽度优先搜索策略
- 深度优先搜索策略

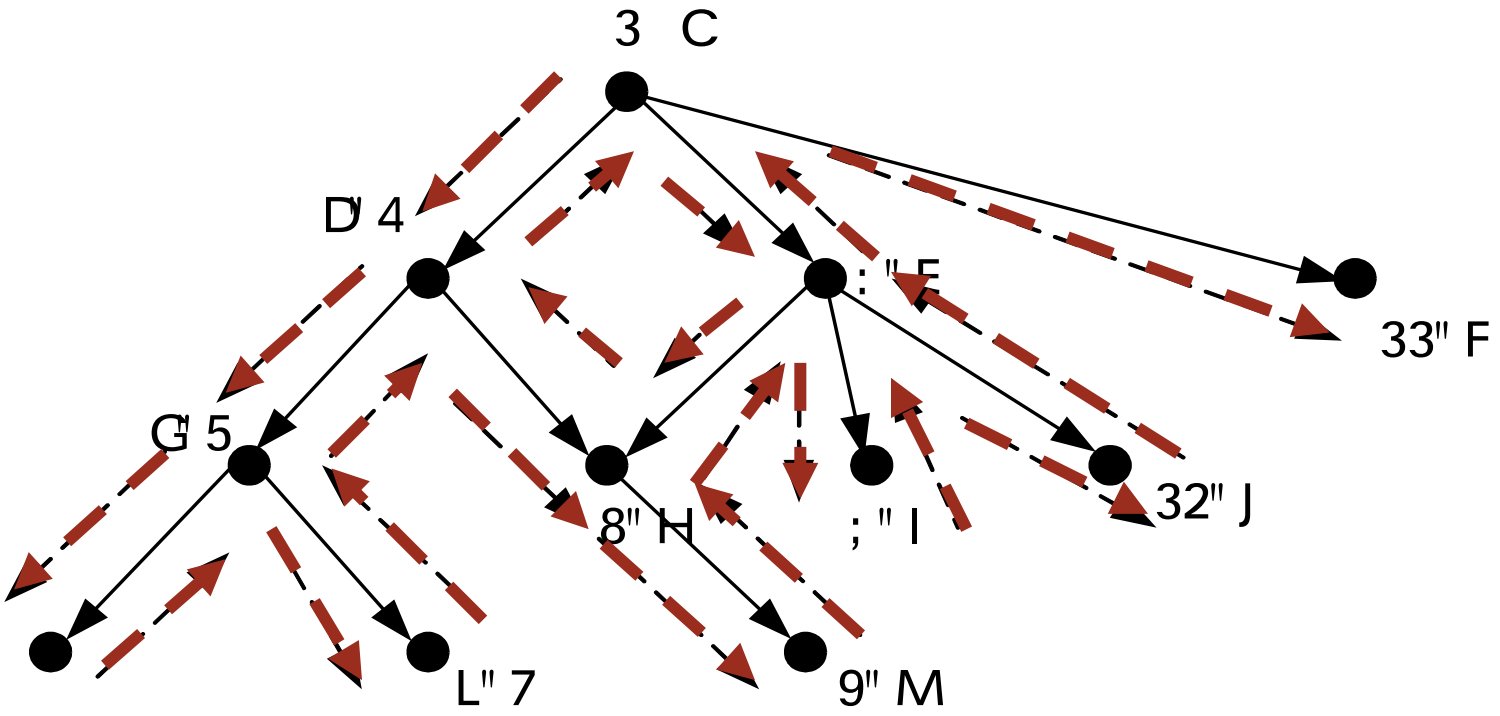
回溯策略

- 带回溯策略的搜索：

从初始状态出发，不停地、试探性地寻找路径，直到它到达目的或“不可解结点”，即“死胡同”为止。

若它遇到不可解结点就回溯到路径中最近的父结点上，查看该结点是否还有其他的子结点未被扩展。若有，则沿这些子结点继续搜索；如果找到目标，就成功退出搜索，返回解题路径。

回溯策略



回溯搜索示意图

盲目的图搜索策略

- 回溯策略
- 宽度优先搜索策略
- 深度优先搜索策略

宽度优先搜索BFS策略

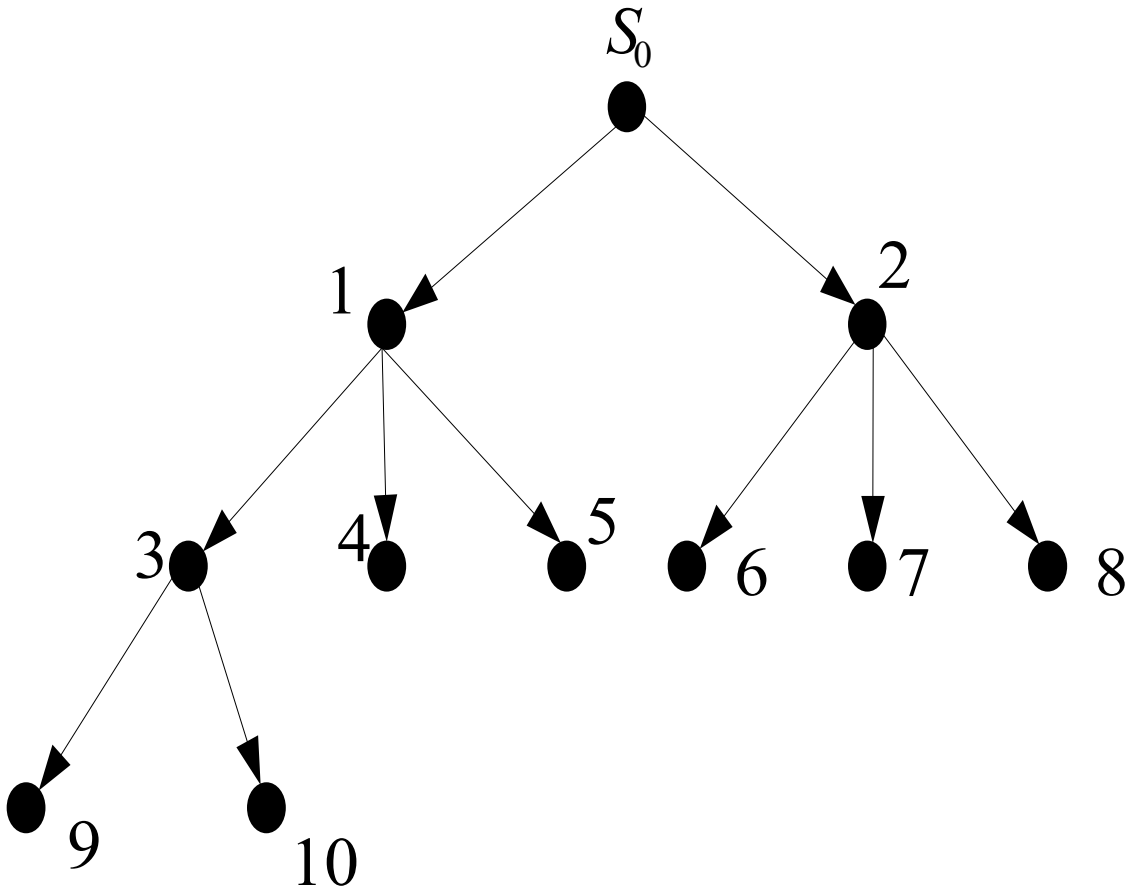
- 搜索策略

  扩展最浅的未扩展节点。

- 实现方法

  使用FIFO队列（队列），把后继节点放在队列的末端；

宽度优先搜索策略

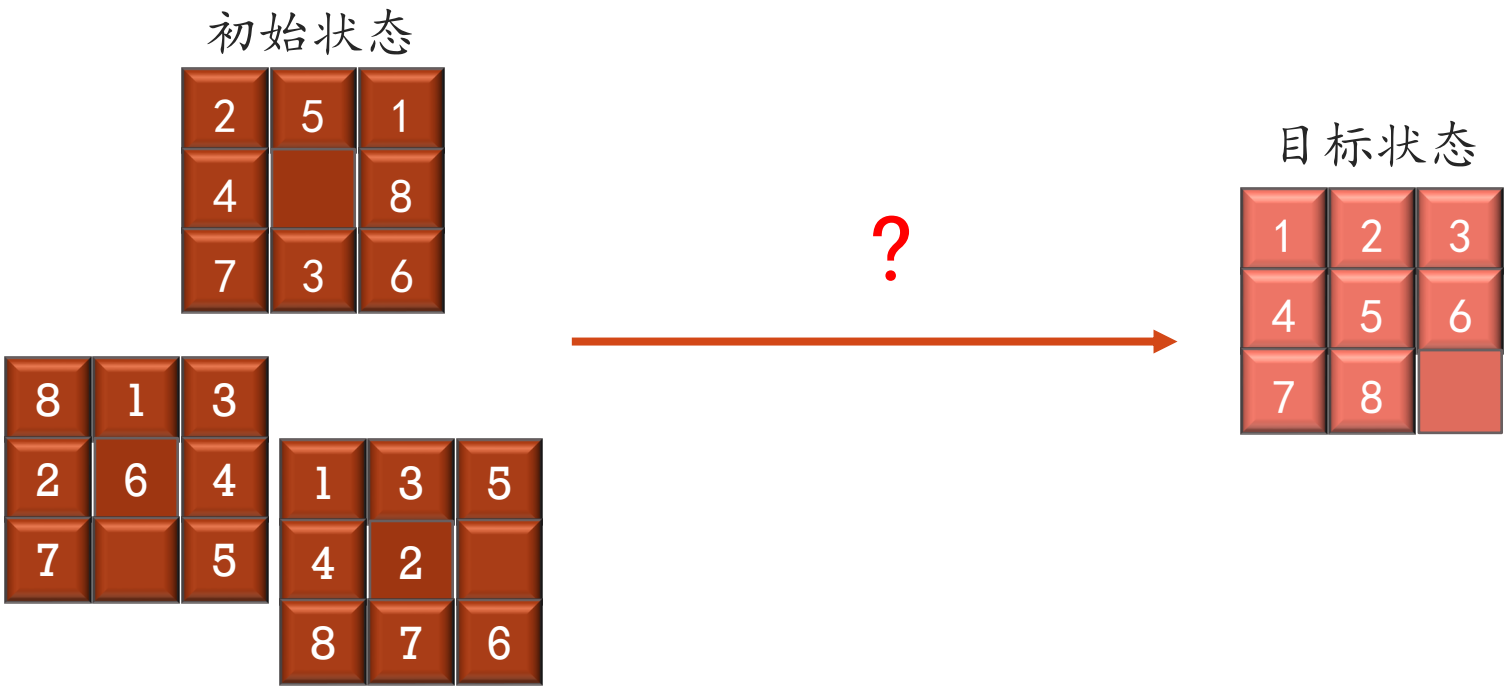


宽度优先搜索法中状态的搜索次序

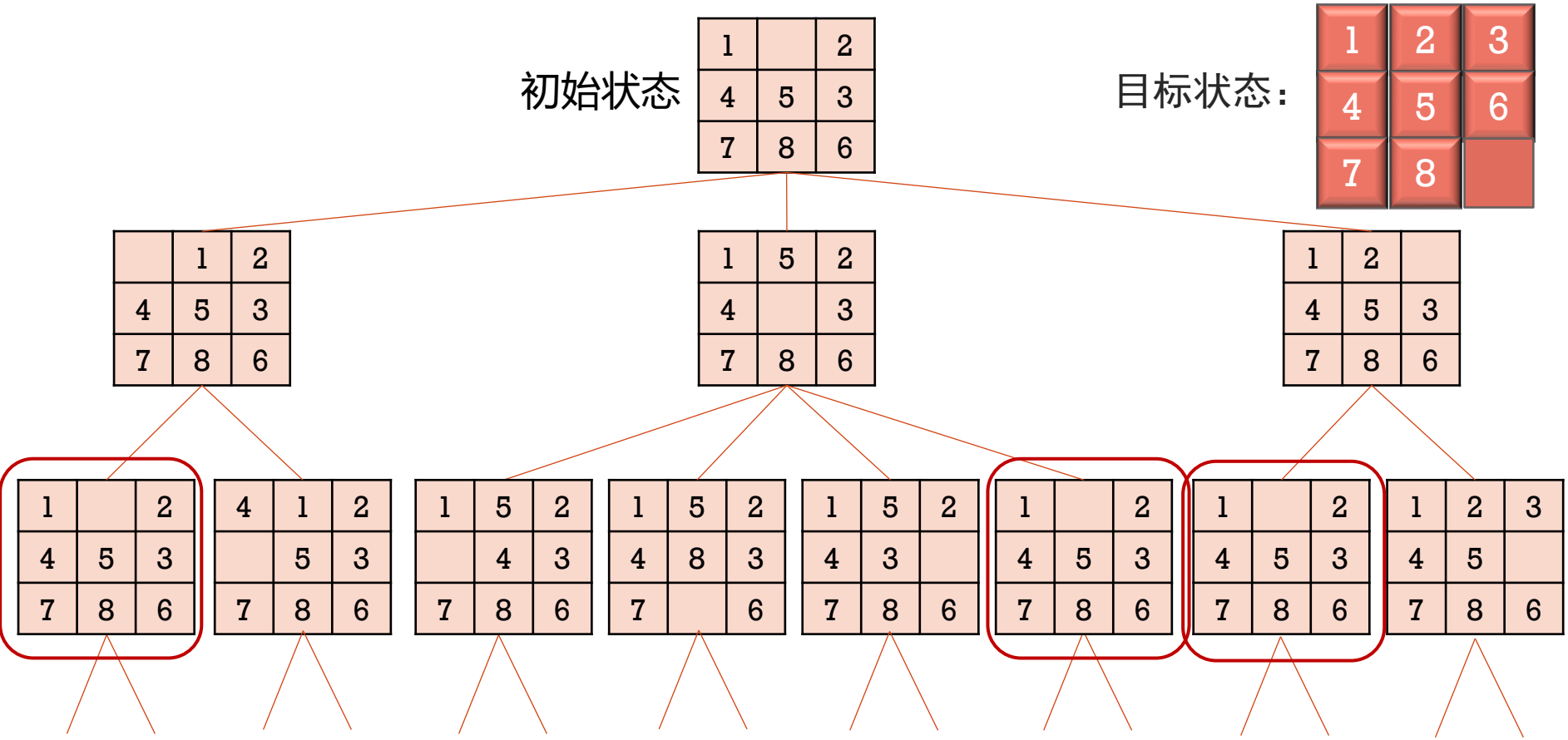
宽度优先搜索策略

- 例子：八数码

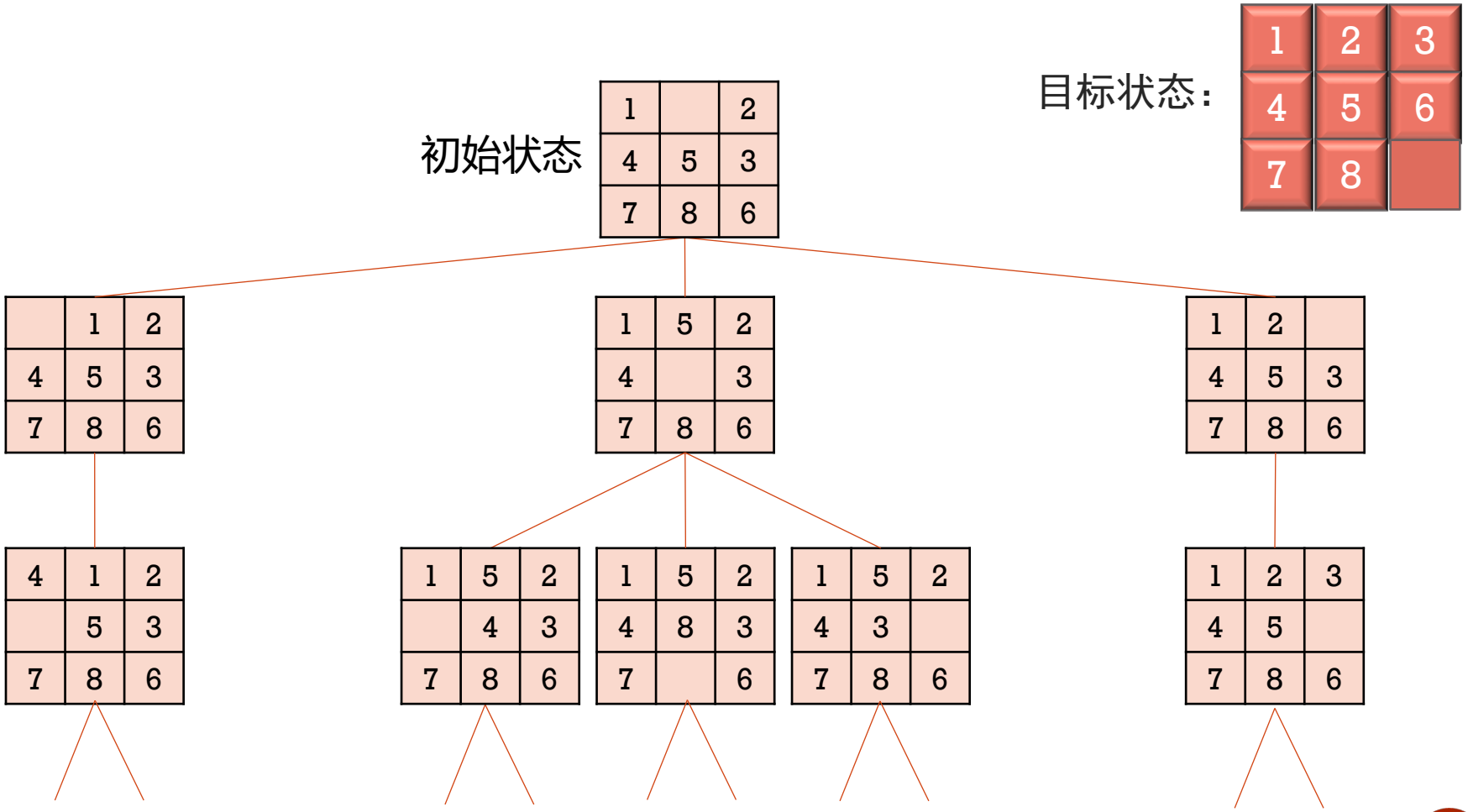
操作算子：上、下、左、右



宽度优先搜索策略

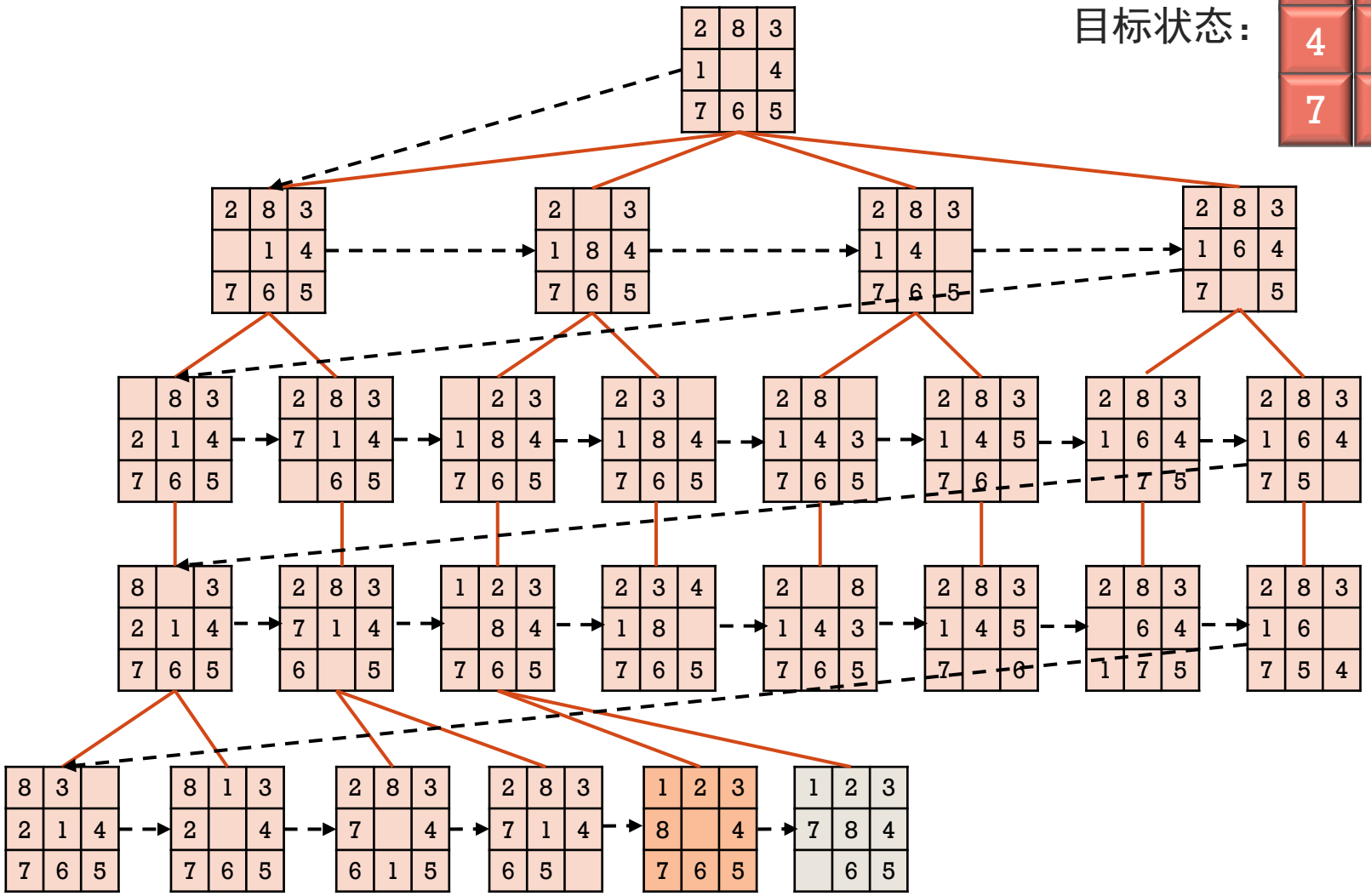


宽度优先搜索策略


目标状态：
1 2 3
4 5 6
7 8
初始状态
1 2
4 5 3
7 8 6
1 2
4 5 3
7 8 6
1 5 2
4 3
7 8 6
1 2
4 5 3
7 8 6
4 1 2
5 3
7 8 6
1 5 2
4 3
7 8 6
1 5 2
4 8 3
7 6
1 5 2
4 3
7 8 6
1 2 3
4 5
7 8 6

宽度优先搜索策略

目标状态：



宽度优先搜索BFS的性质

- 当问题有解时，一定能找到解
- 单位耗散值，且问题有解时，一定能找到最优解
- 方法与问题无关，具有通用性
- 效率较低
- 存储量比较大

盲目的图搜索策略

- 回溯策略
- 宽度优先搜索策略
- 深度优先搜索策略

深度优先搜索策略

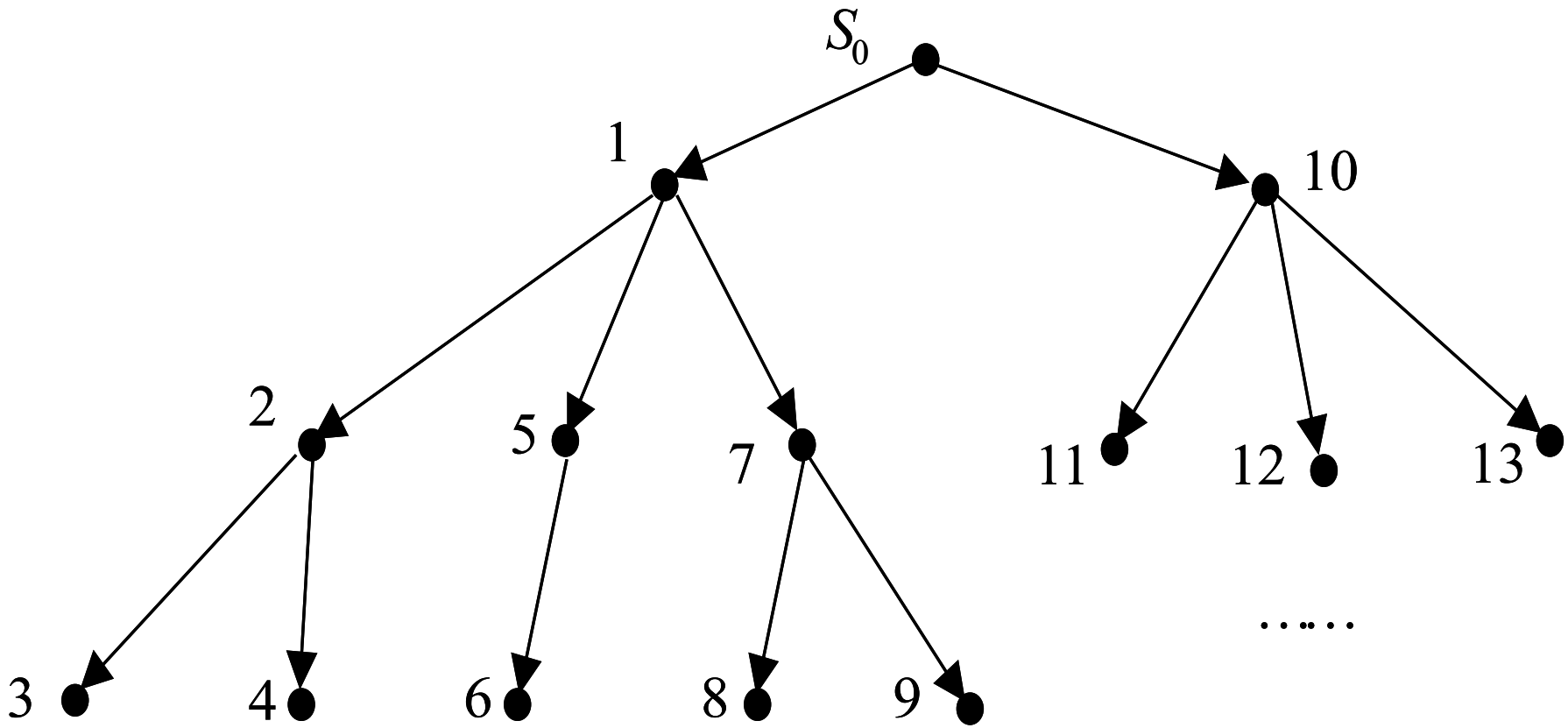
- 搜索策略

  扩展最深的未扩展节点。

- 实现方法

  使用LIFO队列（堆栈），把后继节点放在队列的前端；

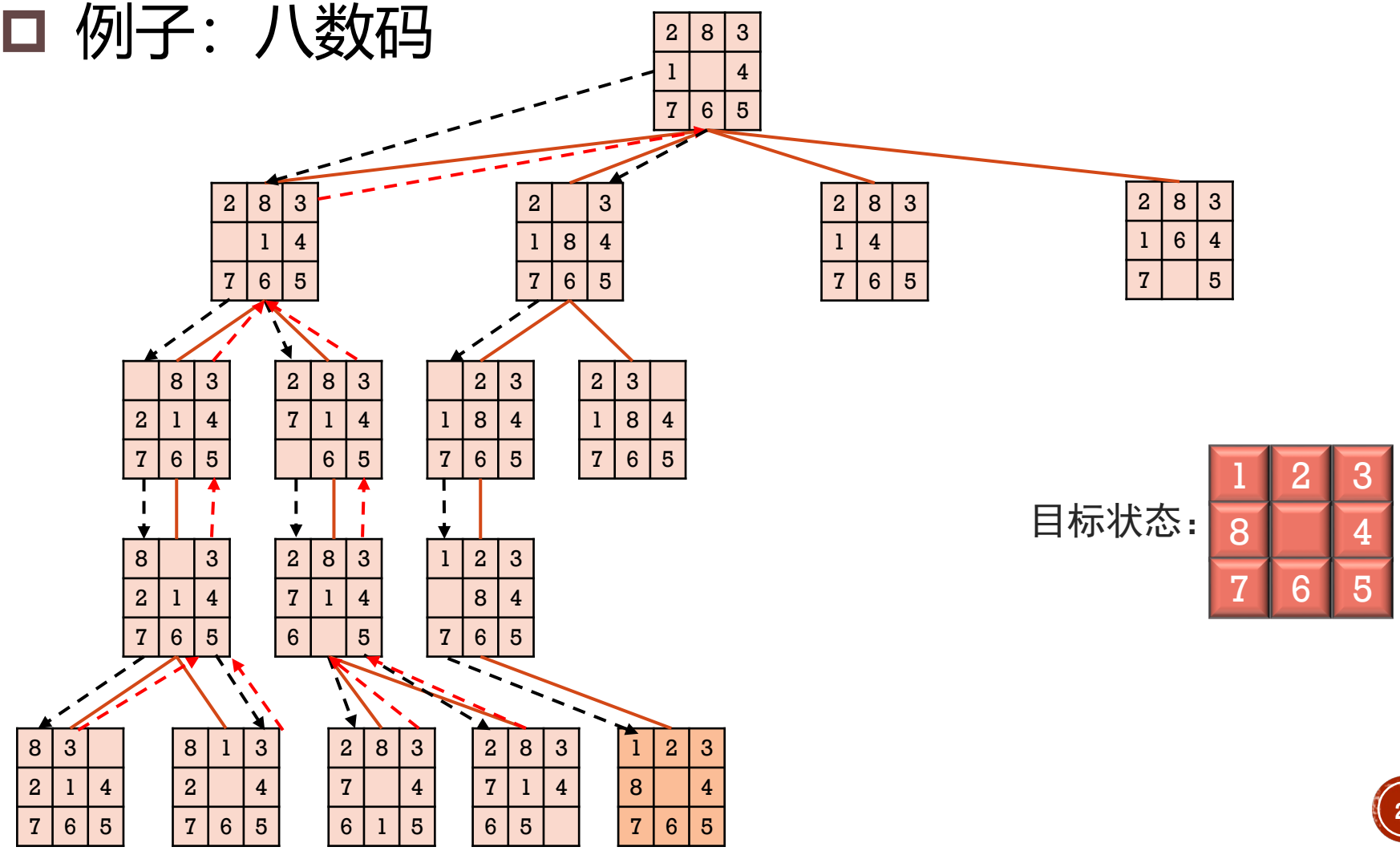
深度优先搜索策略



深度优先搜索法中状态的搜索次序

深度优先搜索策略

- 例子：八数码



目标状态：

1	2	3
8		4
7	6	5

深度优先搜索策略

- 在深度优先搜索中，当搜索到某一个状态时，它所有的子状态以及子状态的后裔状态都必须先于该状态的兄弟状态被搜索。
- 为了保证找到解，应选择合适的深度限制值，或采取不断加大深度限制值的办法，反复搜索，直到找到解。

深度优先搜索的性质

- 一般不能保证找到最优解
- 当深度限制不合理时，可能找不到解，可以将算法改为可变深度限制
- 最坏情况时，搜索空间等同于穷举
- 是一个通用的与问题无关的方法
- 节省内存，只存储从初始节点到当前节点的路径

盲目搜索与启发式搜索

- 盲目搜索：
  - 深度优先搜索
  - 宽度优先搜索

- 启发式搜索
  - A算法
  - A*算法

启发式搜索策略

- 启发式搜索策略
- 启发信息和估价函数
- *A*搜索算法
- *A**搜索算法及其特性分析

启发式搜索

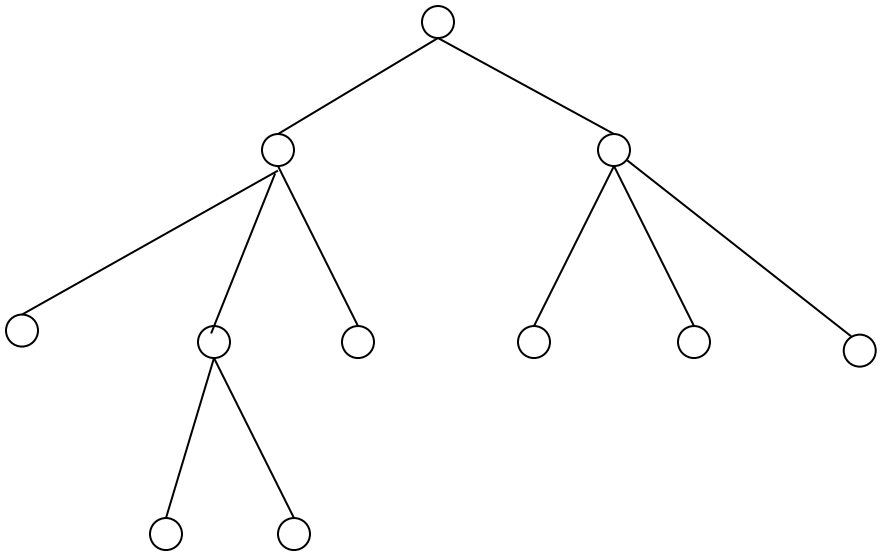
- 在具体求解中，能够利用与该问题有关的信息来简化搜索过程，称此类信息为**启发信息**。
- 在状态空间搜索中，**启发式**被定义成一系列操作算子，并能从状态空间中选择最有希望到达问题解的路径。
- **启发式搜索**：利用与问题有关的启发信息进行搜索。

启发式搜索

- 利用知识来引导搜索，达到减少搜索范围，降低问题复杂度的目的。
- 启发信息的强度
  - 强：降低搜索工作量，但可能导致找不到最优解
  - 弱：一般导致工作量加大，极限情况下变为盲目搜索，但可能可以找到最优解

启发式搜索

- 基本思想：定义一个评价函数f，对当前的搜索状态进行评估，找出一个最有希望的节点来扩展。



启发式策略

- 运用启发式策略的两种基本情况：

（1）一个问题由于在问题陈述和数据获取方面固有的模糊性，可能会使它没有一个确定的解。

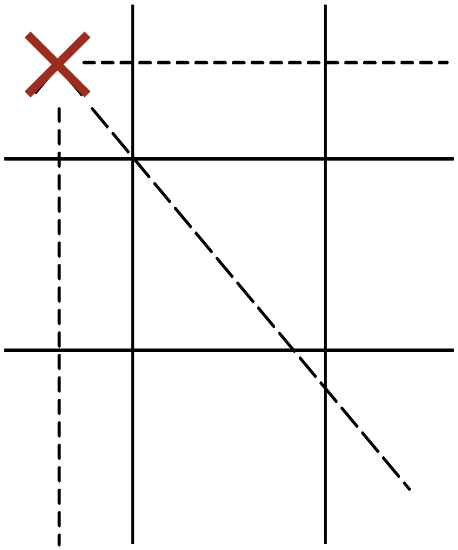
（2）虽然一个问题可能有确定解，但是其状态空间特别大，搜索中生成扩展的状态数会随着搜索的深度呈指数级增长。

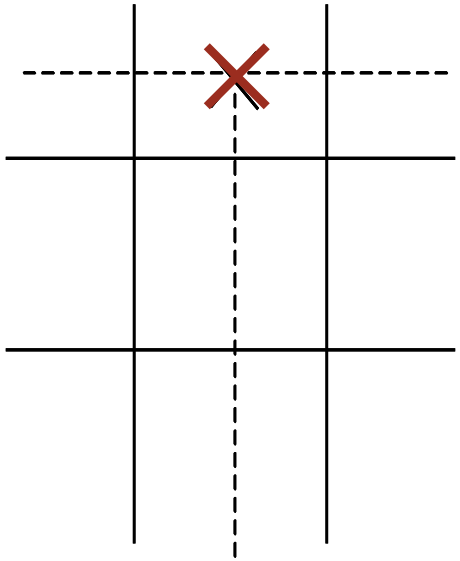
启发式策略

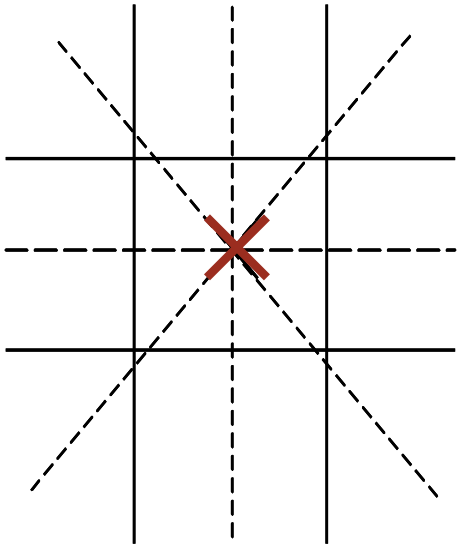
- 例　一字棋。在九宫棋盘上，从空棋盘开始，双方轮流在棋盘上摆各自的棋子 × 或 ○（每次一枚），谁先取得三子一线（一行、一列或一条对角线）的结果就取胜。

- × 和 ○ 能够在棋盘中摆成的各种不同的棋局就是问题空间中的不同状态。
- 9个位置上摆放{空，×，○}有 3^9 种棋局。
- 可能的走法：$9\times8\times7\times\cdots\times1$。

启发式策略







启发式策略的运用

启发信息和估价函数

- 估价函数的任务就是估计待搜索结点的“有希望”程度，并依次给它们排定次序。

➢ 估价函数f(n)：从初始结点经过n结点到达目的结点的路径的最小代价估计值，其一般形式是

$$f(n) = g(n) + h(n)$$

- g(n):从初始结点到n结点的代价值
- h(n): n结点到目标节点路径的代价值

A搜索算法

- 启发式图搜索法的基本特点：如何寻找并设计一个与问题有关的h(n)及构出 f(n)=g(n)+h(n),然后以f(n)的大小来排列待扩展状态的次序，每次选择f(n)值最小者进行扩展。
  - 一般地，在f(n)中，g(n)的比重越大，越倾向于宽度优先搜索方式，而h(n)的比重越大，表示启发性能越强。

A搜索算法

- 例　利用A搜索算法求解八数码问题的搜索树，其估价函数定义为

$$f(n) = d(n) + w(n)$$

- $d(n)$：状态的深度，每步为单位代价。
- $w(n)$：以“不在位”的将牌数作为启发信息的度量。

- $h^*(n)$：为状态 n到目的状态的最优路径的代价。
- $w(n) = h(n) \le h^*(n)$：A搜索算法 $\rightarrow$ A*搜索算法。

启发信息和估价函数

- **例 八数码的估价函数设计方法有多种，并且不同的估价函数对求解八数码问题有不同的影响。**

  - 最简单的估价函数：取一格局与目的格局相比，其位置不符的将牌数目。
  - 较好的估价函数：各将牌移到目的位置所需移动的距离的总和。
  - 第三种估价函数：对每一对逆转将牌乘以一个倍数。
  - 第四种估价函数：.....

最佳图搜索算法A*（A*算法）

- 在A算法中，由于并没有对启发函数作出任何规定，因此，A算法得到的解结果如何也不好评定
- **在A算法中，如果满足条件：**

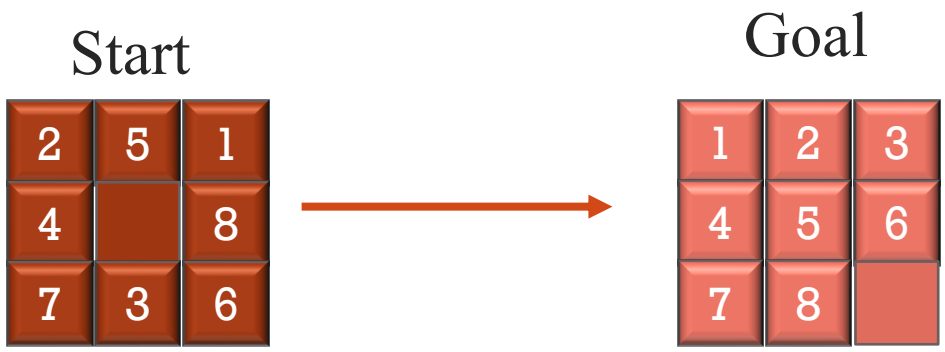
$$h(n) \leqslant h^*(n)$$

则A算法称为A*算法。

最佳图搜索算法A*（A*算法）

- 启发式评估函数是在单智能体寻路问题中，估计一对状态(如某一状态与目标状态)之间最优路径的代价。

- 例如:



$h_1(S)$ =不在目标状态位置上数字的数量

S: state

最佳图搜索算法A*（A*算法）

- 例子

5		8
4	2	1
7	3	6

当前状态

1	2	3
4	5	6
7	8	

目标状态

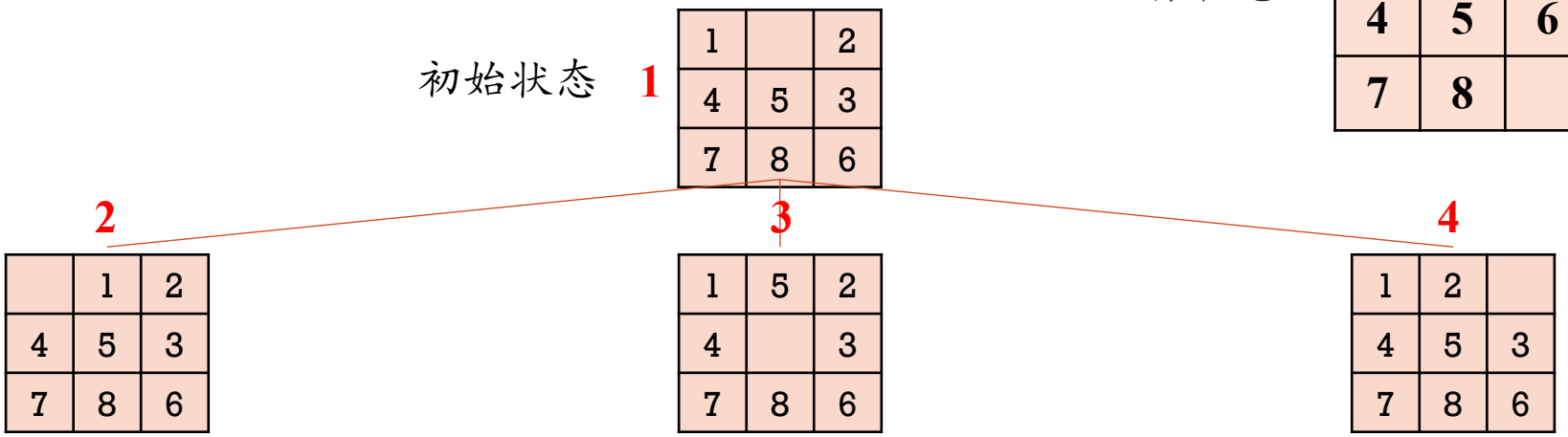
- $h_1(S)$ = 不在目标状态位置上数字的数量= 6
- $h_2(S)$ = 所有数字当前状态到目标状态的Manhattan距离之和
- = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

最佳图搜索算法A*（A*算法）

- A* searching – 例子: 八码数

目标状态：

1	2	3
4	5	6
7	8	



E.g. g(s1)=0
h(s1) =0+1+1+0+0+1+0+0=3;

g(s2) = 1;
h(s2)= 1+1+1+0+0+1+0+0=4;

g(s3) = 1;
h(s3) = 0+1+1+0+1+1+0+0=4;

g(s4) = 1;
h(s4)= 0+0+1+0+0+1+0+0=2;

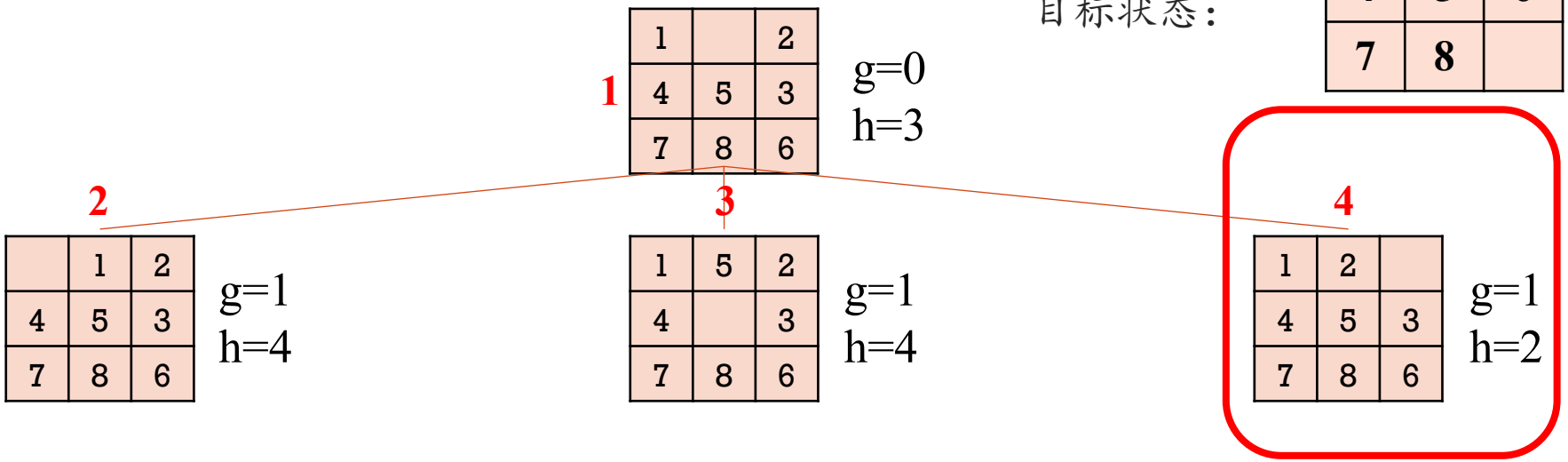
- g(s) = 初始位置到当前位置代价的和
- h(s) =所有数字当前状态到目标状态的Manhattan距离之和

最佳图搜索算法A*（A*算法）

- A* searching – 例子: 八码数

目标状态：

1	2	3
4	5	6
7	8	



E.g. g(s1)=0
h(s1) =0+1+1+0+0+1+0+0=3;

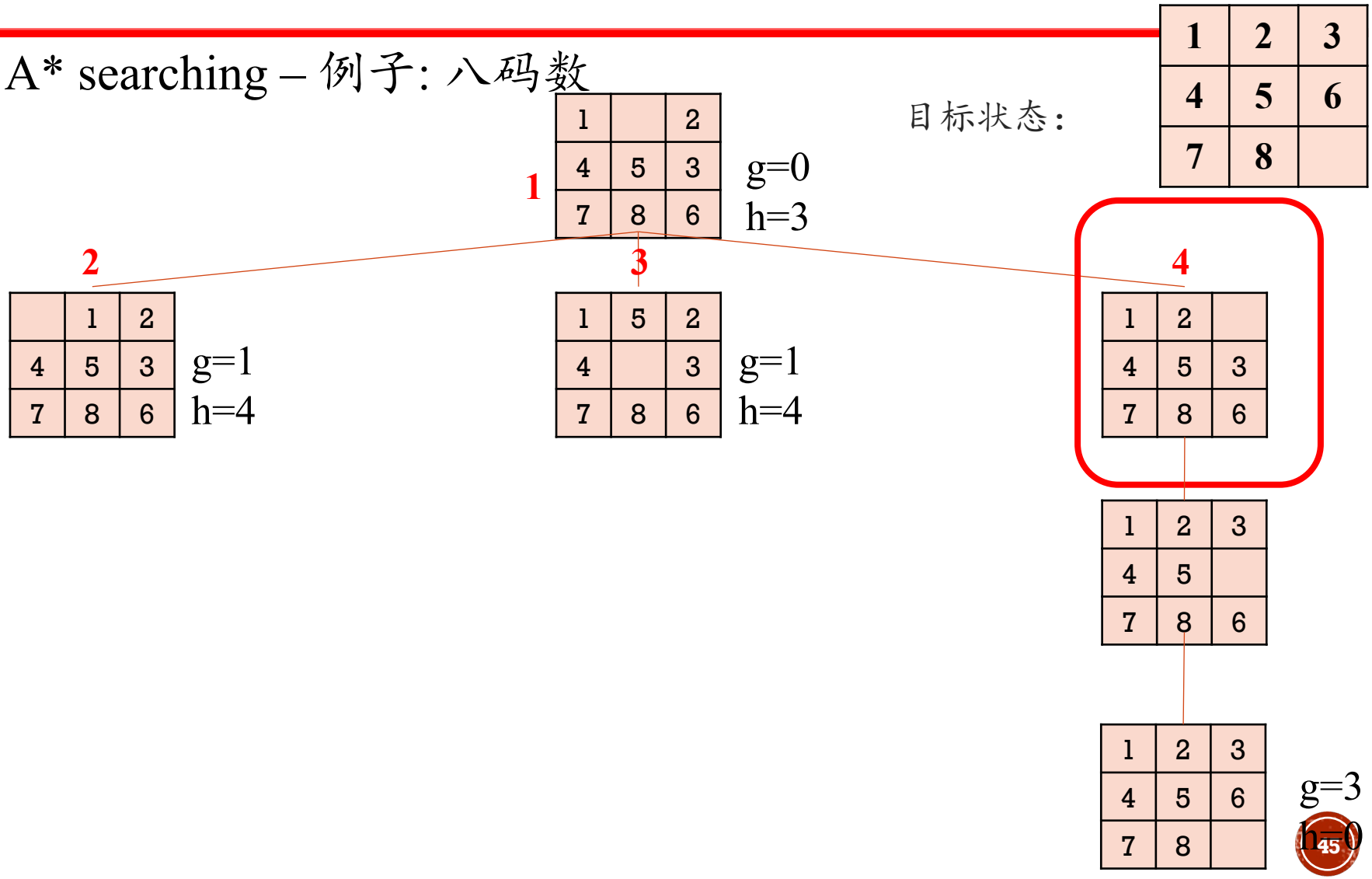
g(s2) = 1;
h(s2)= 1+1+1+0+0+1+0+0=4;

g(s3) = 1;
h(s3) = 0+1+1+0+1+1+0+0=4;

g(s4) = 1;
h(s4)= 0+0+1+0+0+1+0+0=2;

最佳图搜索算法A*（A*算法）

- A* searching – 例子: 八码数



A*算法的两个主要结论

- 定理(可采纳性定理):

  若存在从初始节点s到目标节点t有路径，则A*必能找到最佳解结束。

- 如果某一问题有解，那么利用*A**搜索算法对该问题进行搜索则一定能搜索到解，并且一定能搜索到最优的解而结束。

其他的搜索算法（续1）

- 局部搜索算法

  爬山法 和 束搜索算法

- 动态规划算法

  如果对于任何n，当h(n)=0时，A*算法就成为了动态规划算法。

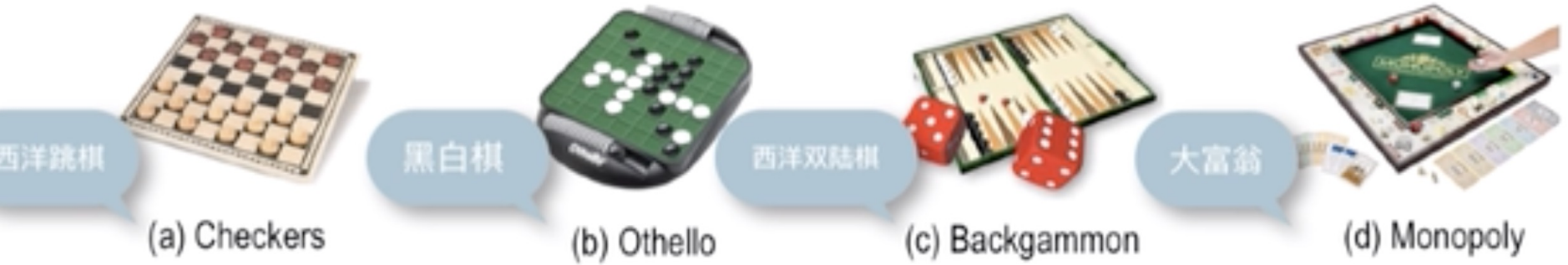
- 对抗搜索
  - α-β剪枝
  - 蒙特卡洛树搜索

博弈搜索

- 在多Agent环境中（竞争环境），每个Agent的目标之间是有冲突的，所以就引出了对抗搜索（Adversarial search problems）（通常称为博弈）。
- Games are a form of multi-agent environment. 人工智能中的博弈通常指博弈论专家们称为有完整信息的，确定性的，轮流行动的，两个游戏者的零和游戏（如象棋）。

博弈搜索

	Deterministic 确定性	Stochastic 随机性
Perfect information (fully observable) 完全信息 （可完全观测）	Chess 国际象棋 Checkers 西洋跳棋 Go 围棋 Othello 黑白棋	Backgammon 西洋双陆棋 Monopoly 大富翁



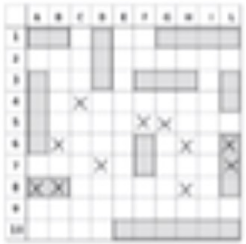
(a) Checkers (b) Othello (c) Backgammon (d) Monopoly

博弈搜索

	Deterministic 确定性	Stochastic 随机性
Imperfect information (partially observable) 不完全信息 （不可完全观测）	Stratego 西洋陆军棋 Battleships 海战	Bridge 桥牌 Poker 扑克 Scrabble 拼字游戏

陆军棋

(e) Stratego



(f) Battleships

(g) Scrabble

博弈搜索算法的起源

1912	Ernst Zermelo 恩斯特·策梅洛	Minimax algorithm 最小最大算法
1949	Claude Shannon 克劳德·香农	Chess playing with evaluation function, selective search 用评价函数和选择性搜索下国际象棋
1956	John McCarthy 约翰·麦卡锡	Alpha-beta search Alpha-beta搜索
1956	Arthur Samuel 亚瑟·塞缪尔	Checkers program that learns its own evaluation function 学习自身的评价函数的西洋跳棋程序

- Ernst Zermelo (1871–1953), a German logician and mathematician.
- Claude Shannon (1916–2001), an American mathematician, and cryptographer known as "the father of information theory".
- John McCarthy (1927-2011), an American computer scientist and cognitive scientist, and one of the founders of AI.
- Arthur Samuel (1901-1990), an American pioneer of computer gaming, AI, and ML.

博弈算法进展

- Computers are better than humans 计算机优于人类

Checkers 西洋跳棋	Solved in 2007 2007年已解决
Chess 国际象棋	IBM Deep Blue defeated Kasparov in 1997 IBM深蓝于1997年战胜了卡斯帕罗夫
Go 围棋	Google AlphaGo beat Lee Sedol, a 9 dan professional in Mar. 2016 谷歌AlphaGo于2016年3月战胜了9段职业棋手李世乭

- Computers are competitive with top human players 计算机与顶级人类玩家媲美

Backgammon	TD-Gammon used reinforcement learning to learn evaluation function TD-Gammon使用了强化学习方法来得到评价函数
Bridge	Top systems use Monte-Carlo simulation and alpha-beta search 顶级的系统使用蒙特卡罗仿真和alpha-beta搜索

两个玩家博弈

- 特点：

  确定性、完整信息、轮流、两个玩家、零和

- 两个玩家分别称为

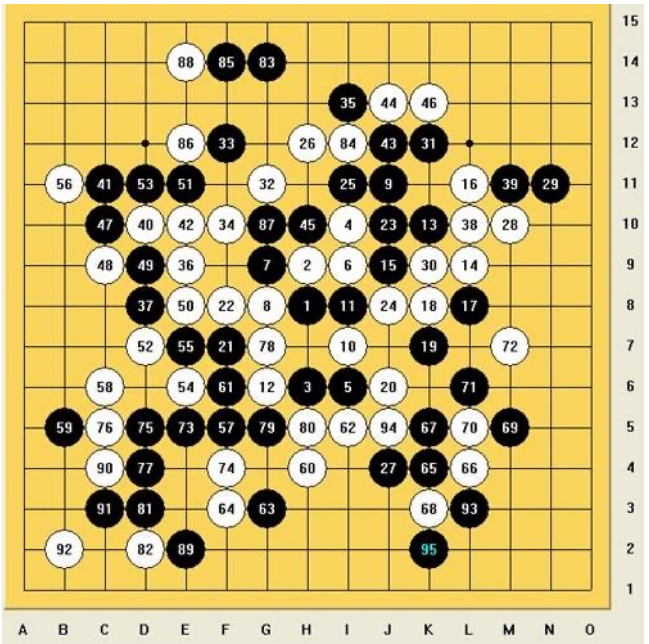
  max和min

- Max先走棋，然后轮流走棋，直到博弈结束

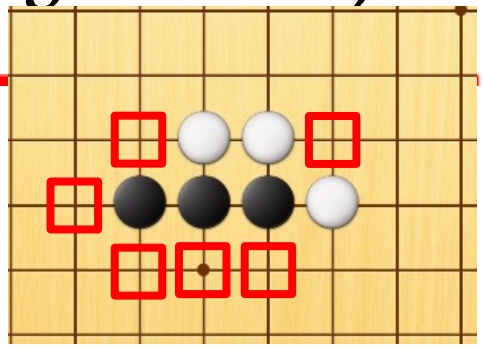
博弈搜索

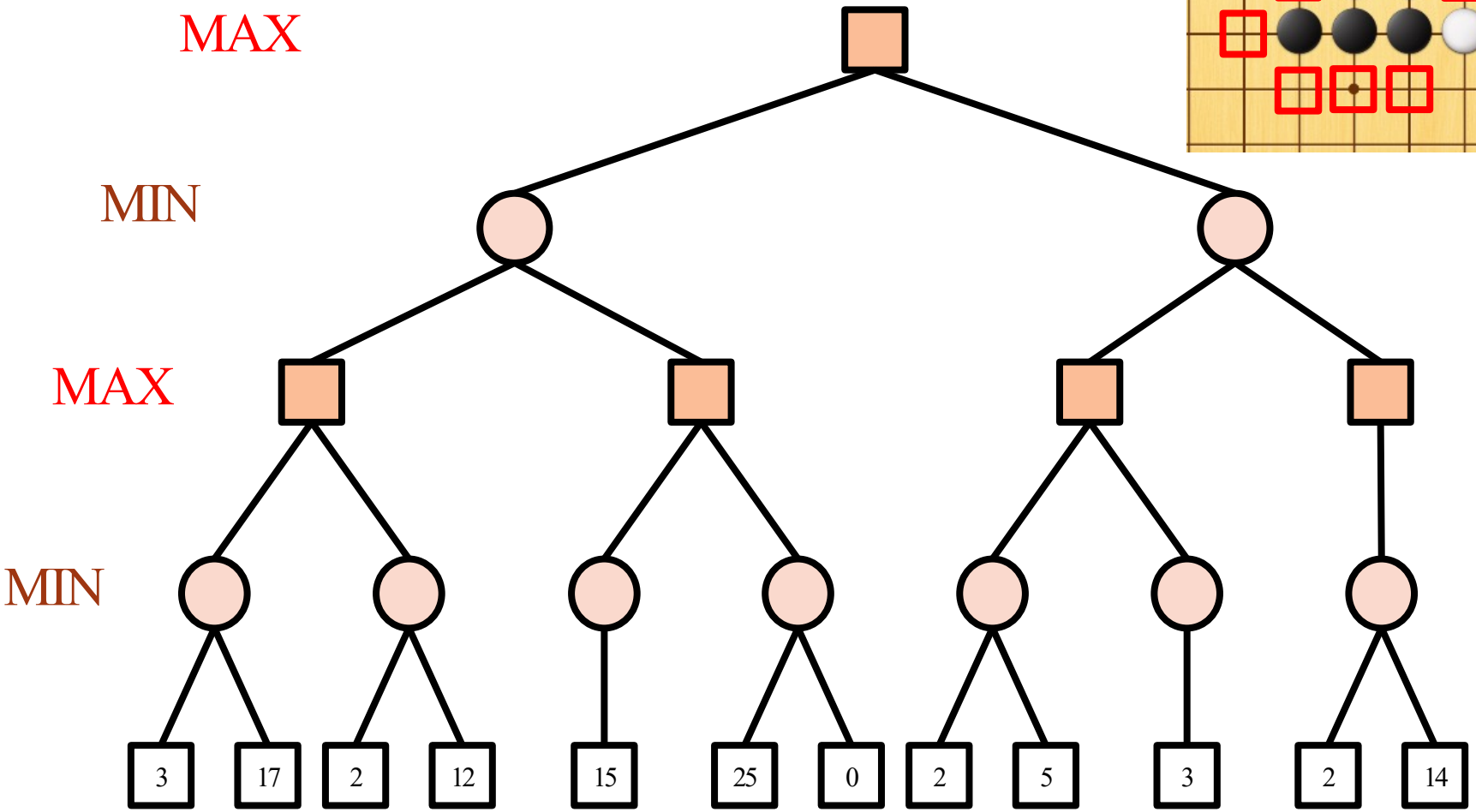
极大极小算法（The Minimax Algorithm）

- 无论Min做什么，Max必须找到一种策略来找到获胜状态，得到更高的分数；
- MIN也会做同样的事情，或者至少会阻止MAX获胜，得到更小的分数；
- 所以, Max会选择分数更高的操作算子；
- Min总是选择分数更小的操作算子。

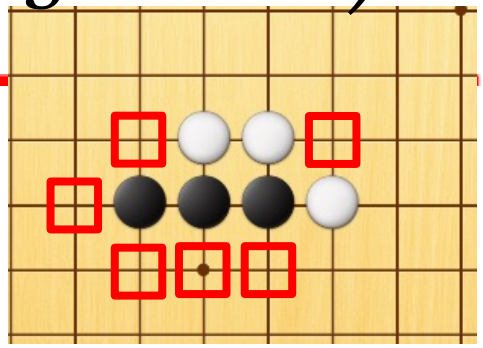


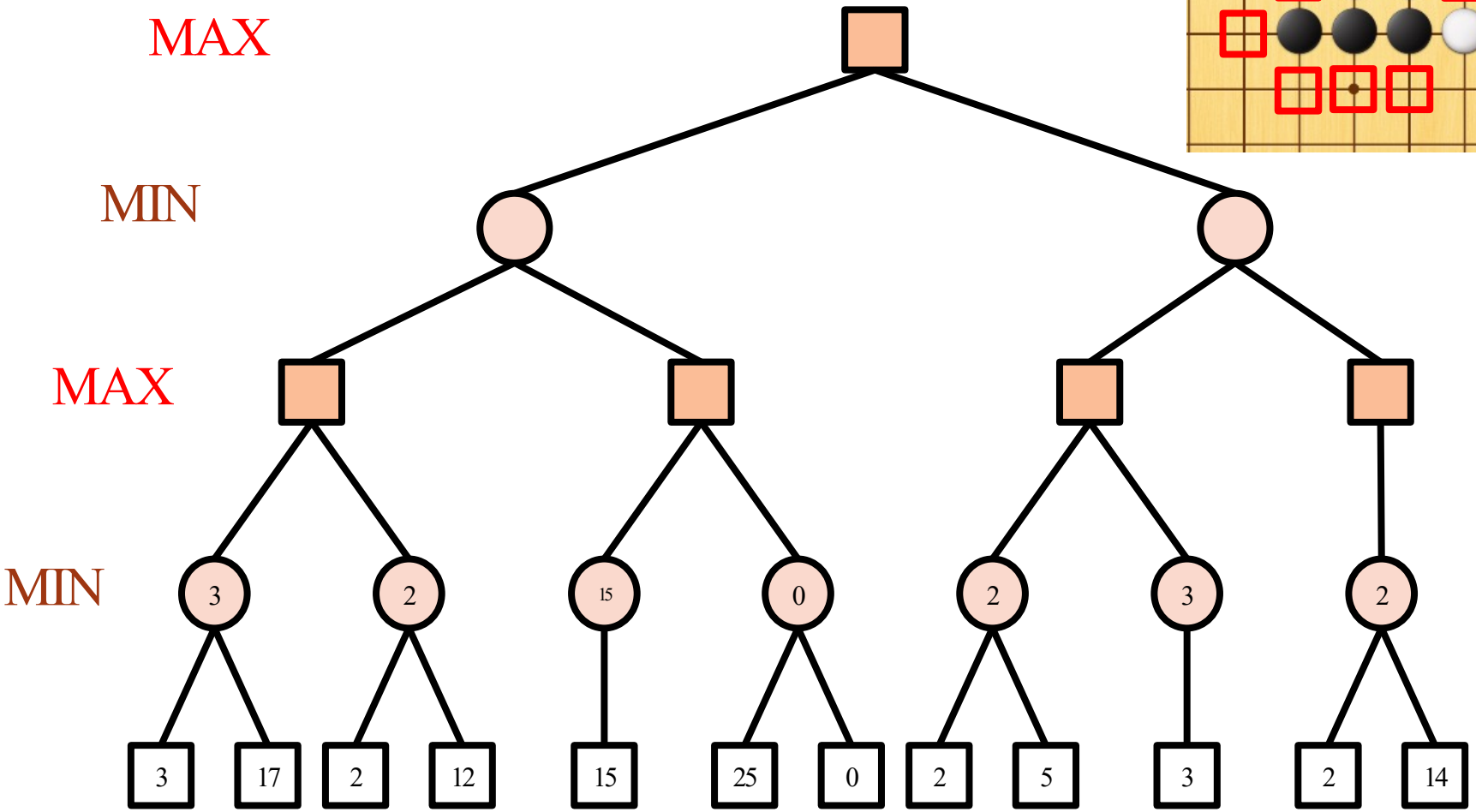
极大极小算法（The Minimax Algorithm）



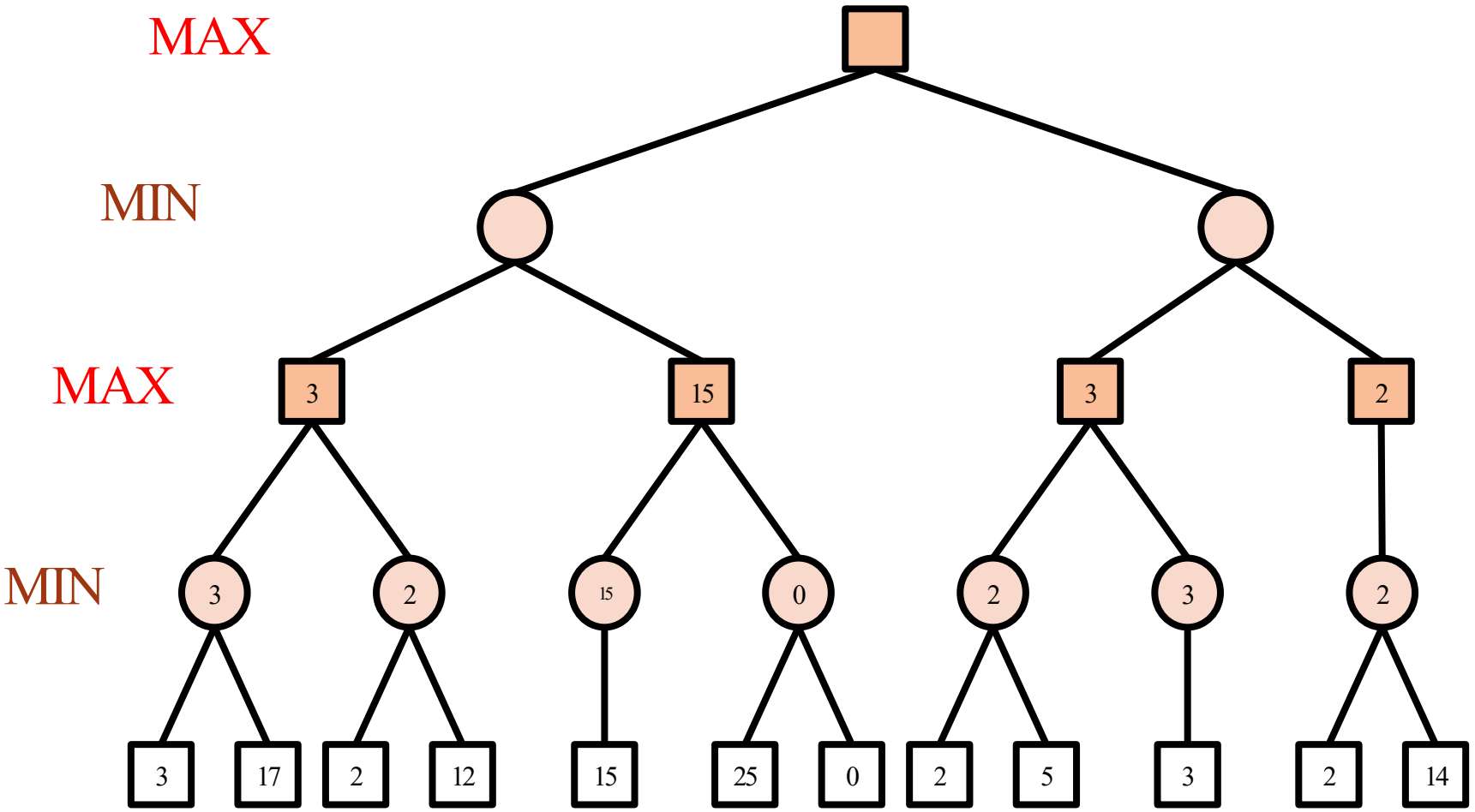


极大极小算法（The Minimax Algorithm）

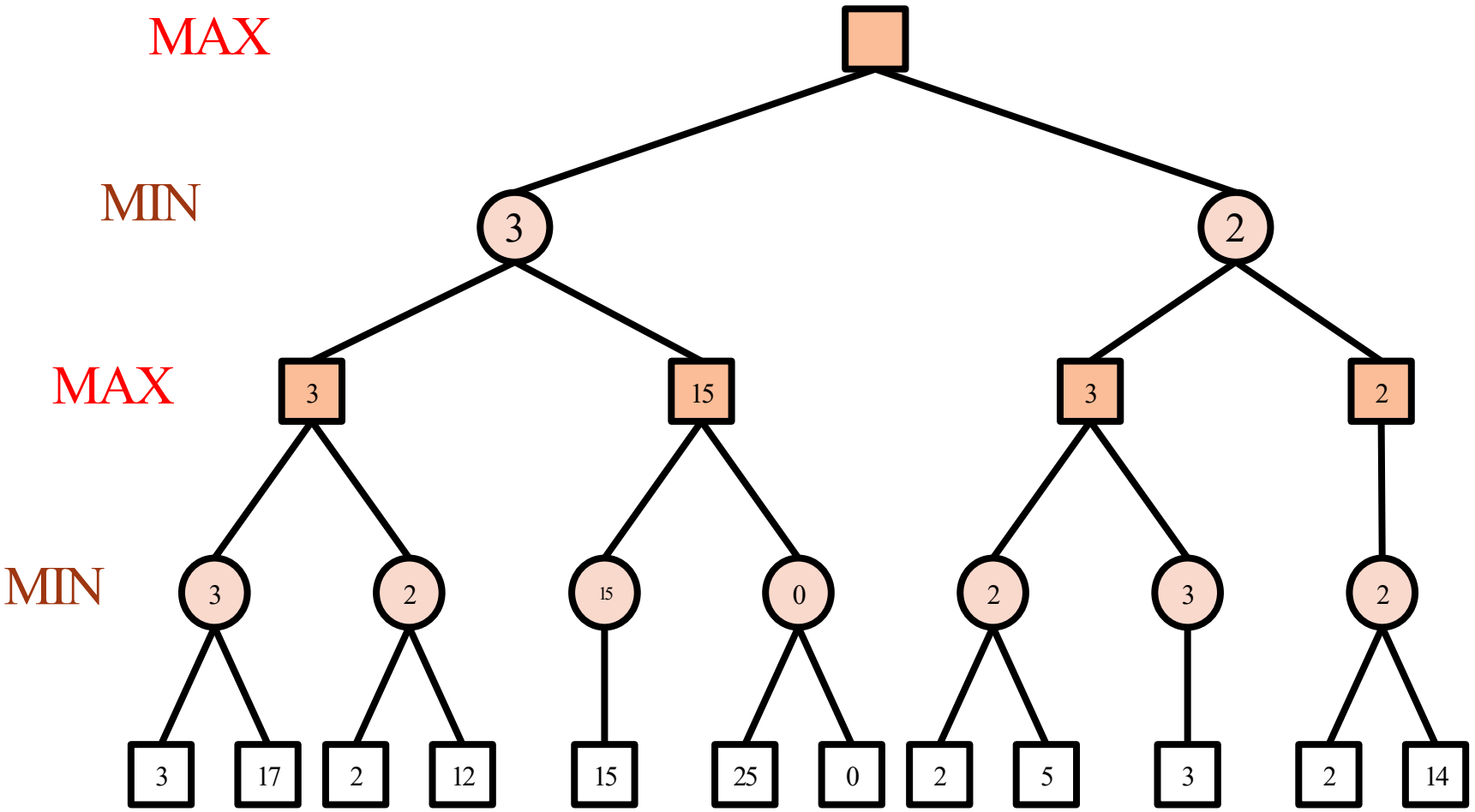




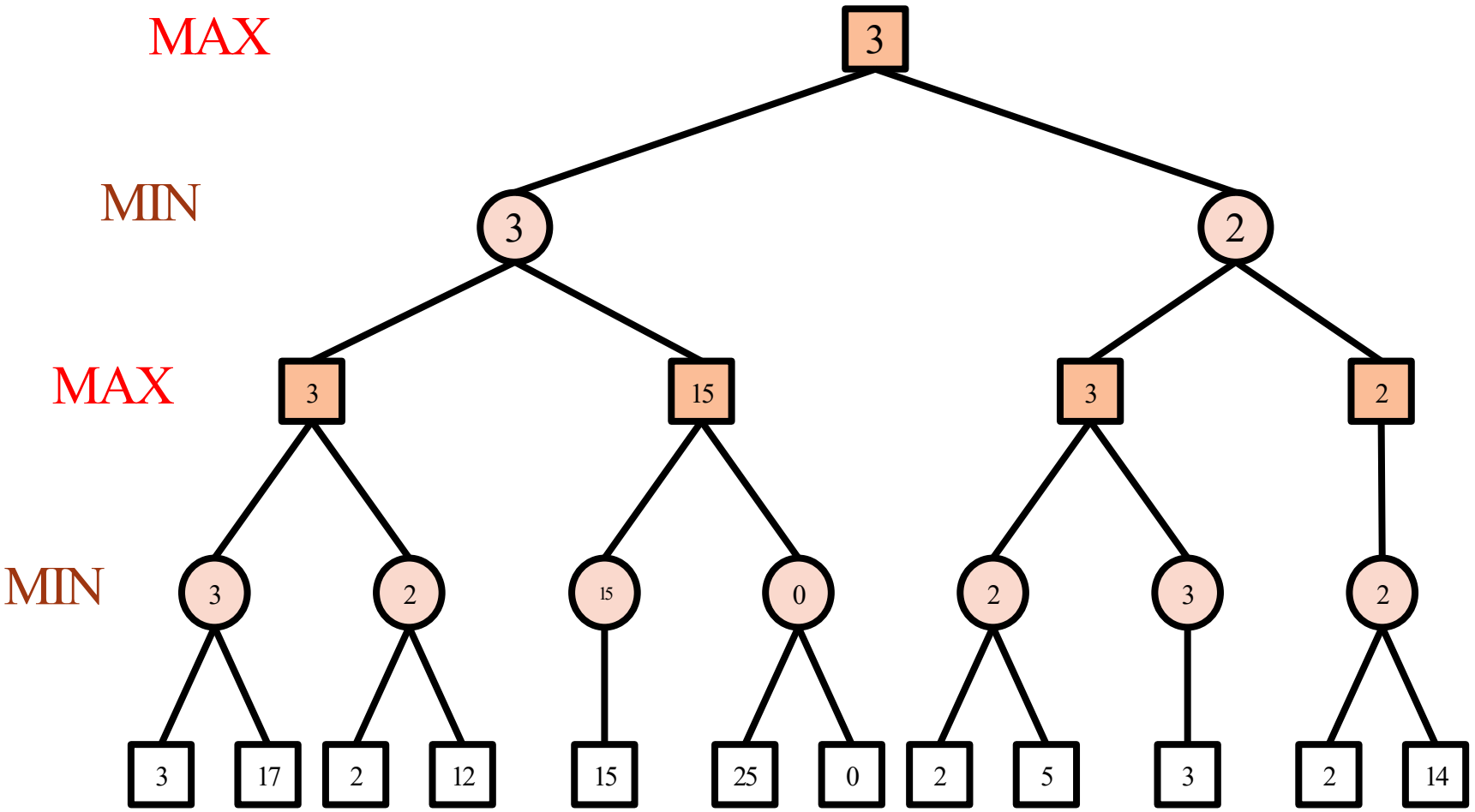
极大极小算法（The Minimax Algorithm）



极大极小算法（The Minimax Algorithm）



极大极小算法（The Minimax Algorithm）



博弈搜索

- 极大极小算法（The Minimax Algorithm）

1. 在对弈中，可利用棋局估值函数对任何一个局面进行估值，估值越大表明对当前玩家越有利，估分越小则表明越不利；

2. 选择落子时，要考虑对自己第一步越有利的，对对手下一步越不利的，对自己第二步越有利的，以此类推；

3. 这样选择落子时就表现为一棵极大极小搜索树，逐层搜索，对自己轮的层就选最大值的局面，对对手轮就选最小值的局面，直到一定深度。

博弈搜索

- 博弈状态的数量随着树的深度呈现指数式增长；
- 我们无法消除这种指数，但实际上我们可以将它（树枝）剪掉一半；

----采用剪枝的方法来消除博弈树的大部分

α-β剪枝算法

A-B剪枝算法

- 什么是α-β剪枝算法?

  它是一种搜索算法，旨在削减有minmax算法评价的节点数量

- 为何称为α-β剪枝算法?

  · α：沿着max路径上的任意选择点，迄今为止我们已经发现的最高值;

  · β：沿着min路径上的任意选择点，迄今为止我们已经发现的最低值

A-B剪枝算法

- α-β搜索以此完成如下动作

·每个节点需更新α和β 值;

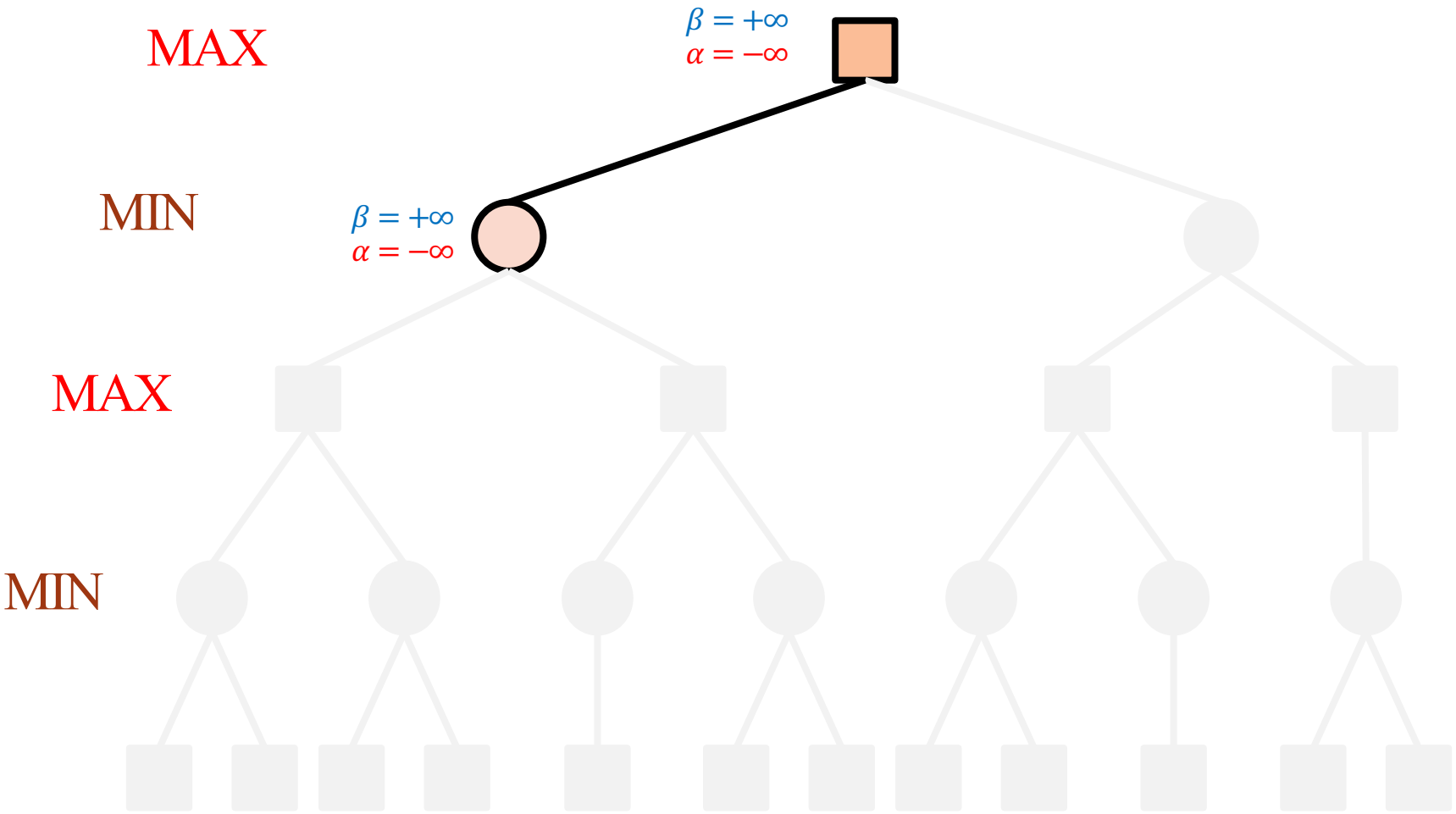
·一旦得知当前节点的值比当前max或min的α或β值更差，则在该节点剪去其余的分支。

·β：沿着min路径上的任意选择点，迄今为止我们已经发现的最低值

A-B剪枝算法

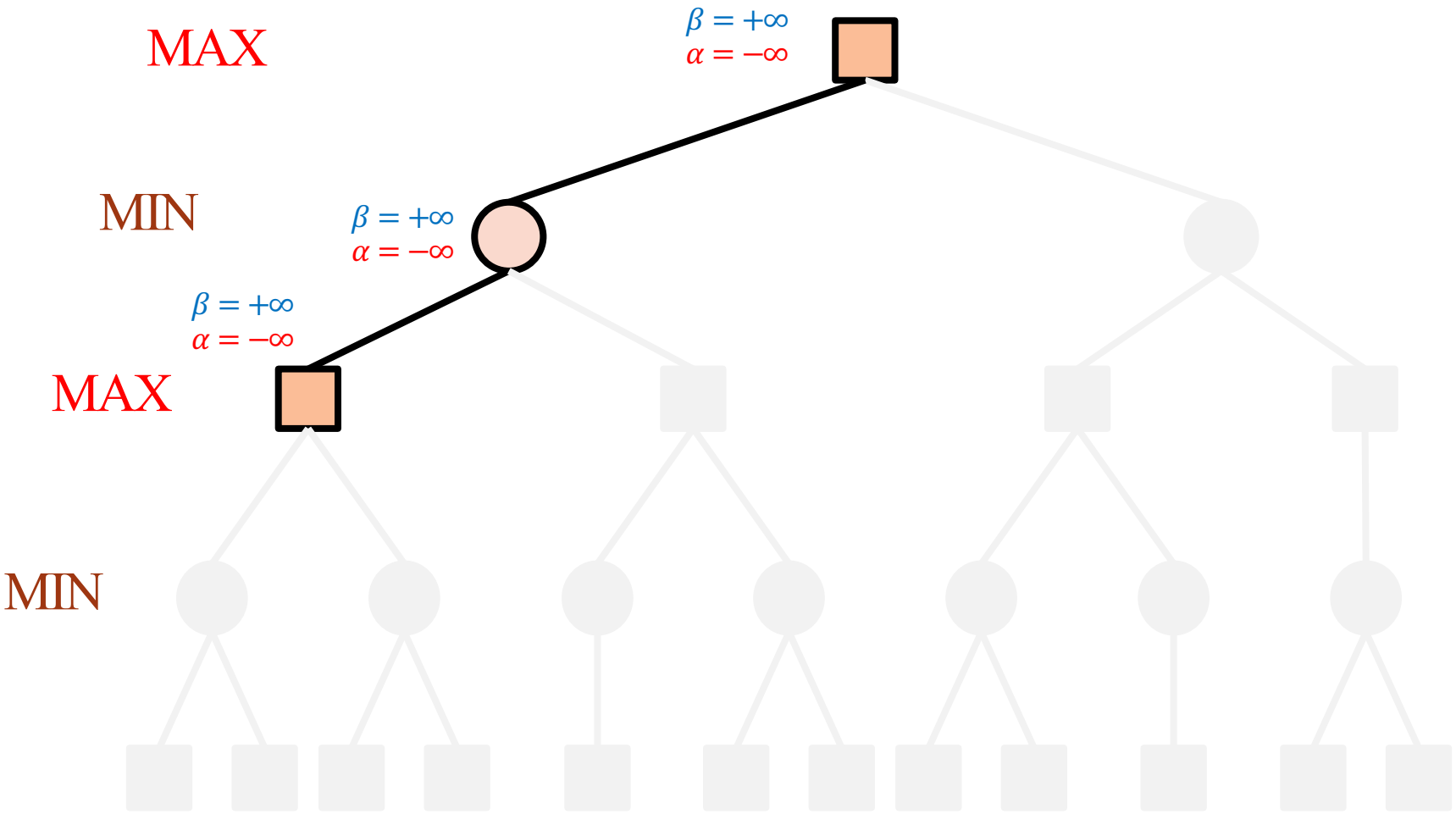
- α-β搜索以此完成如下动作
  - At every leaf node the evaluation is calculated.
  - For every maximum node the current largest child value is saved in α.
  - For every minimum node the current smallest child value is saved in β.
  - If at a minimum node k the current value $\beta \leq \alpha$, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
  - If at a maximum node l the current value $\alpha \geq \beta$, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.

A-B剪枝算法



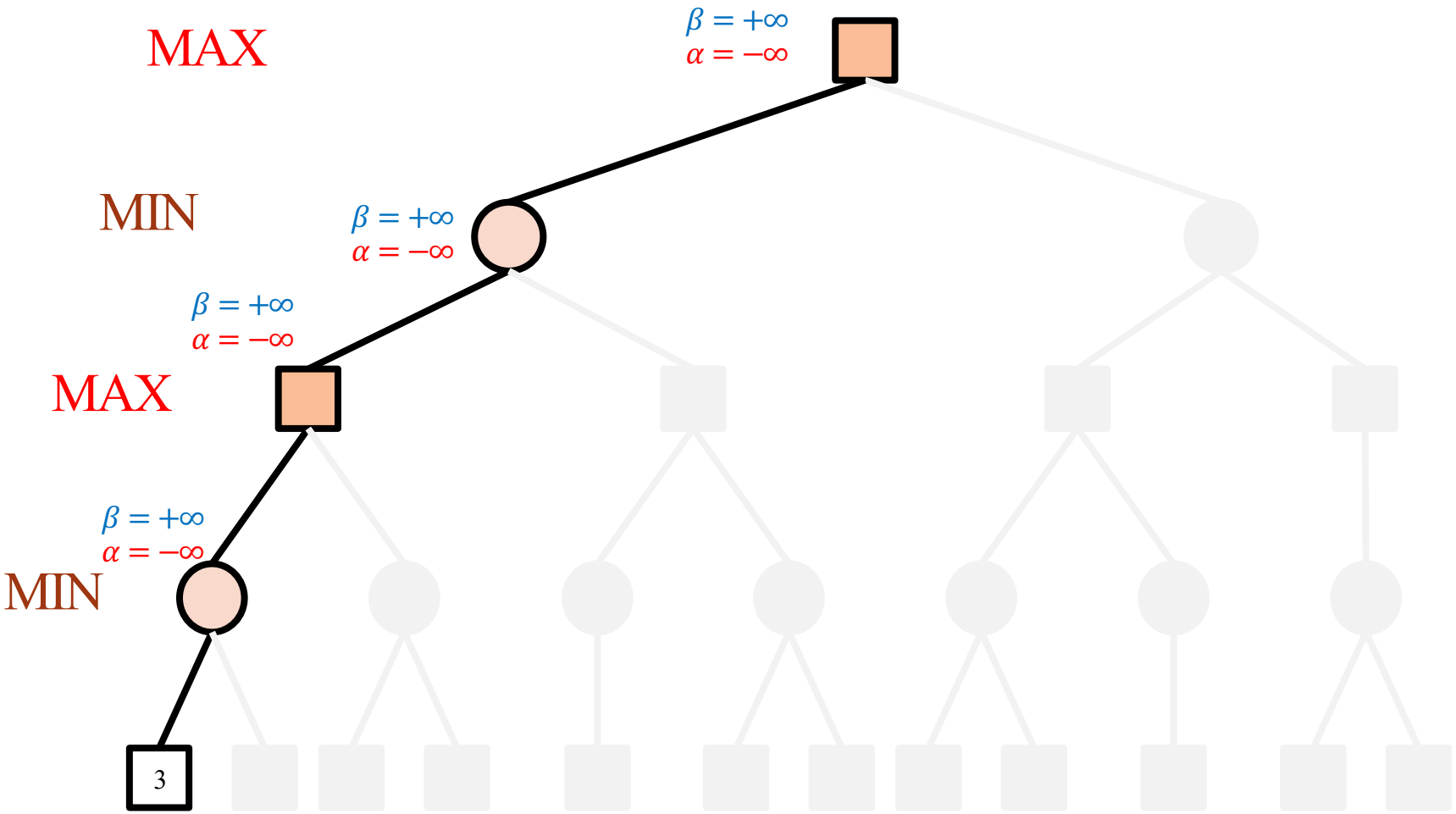
- At every leaf node the evaluation is calculated.

A-B剪枝算法



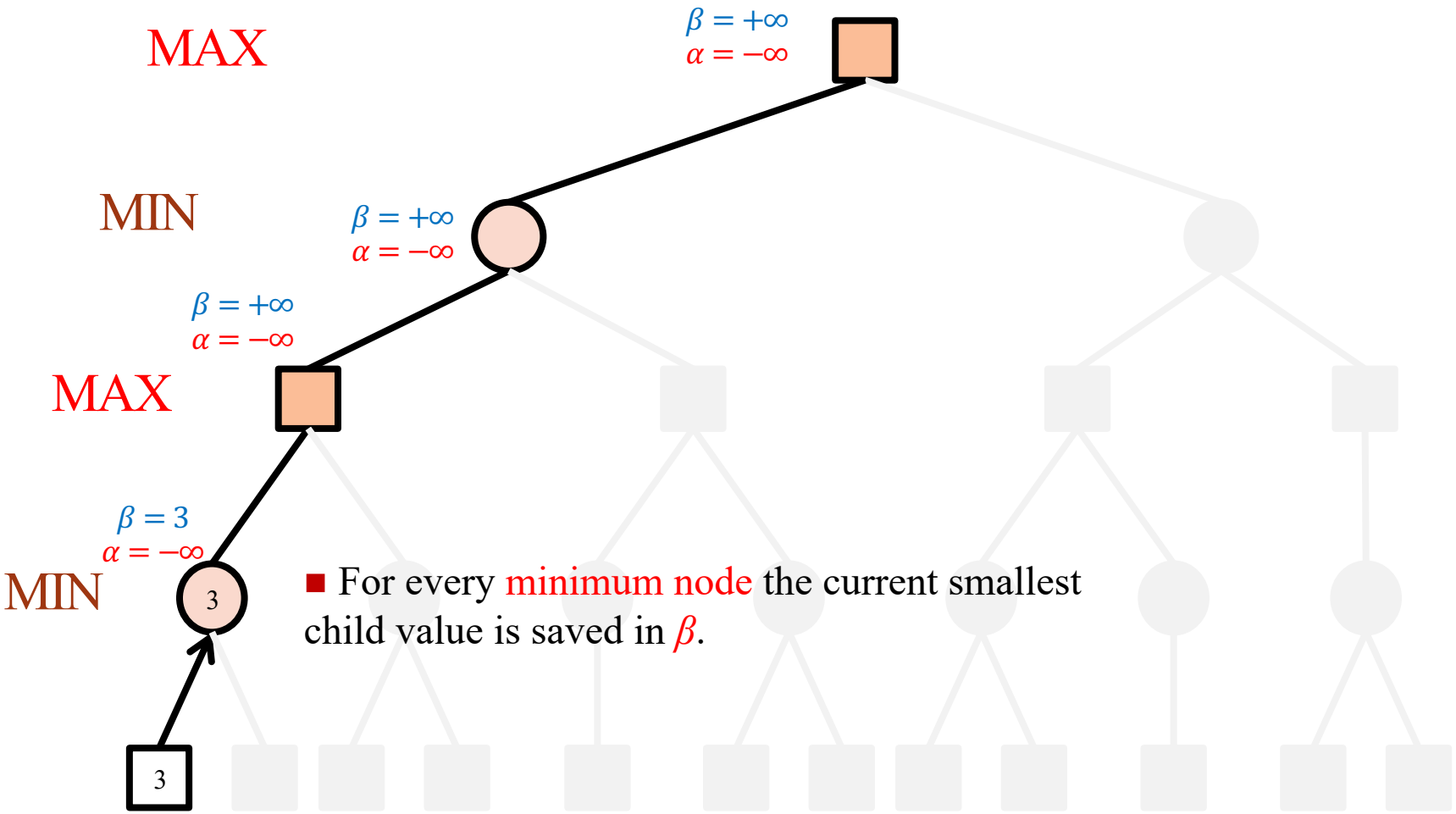
- At every leaf node the evaluation is calculated.

A-B剪枝算法



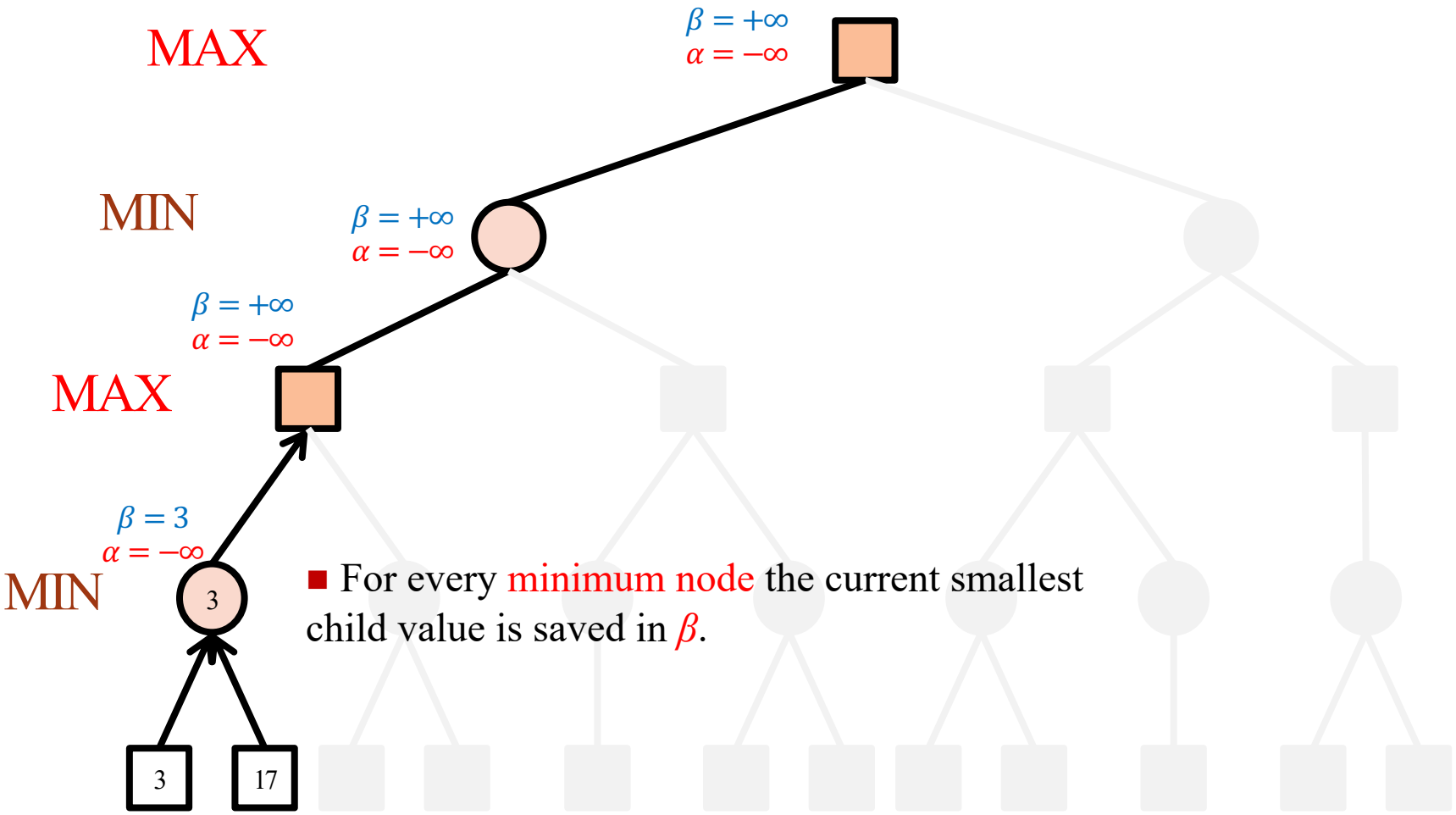
- At every leaf node the evaluation is calculated.

A-B剪枝算法



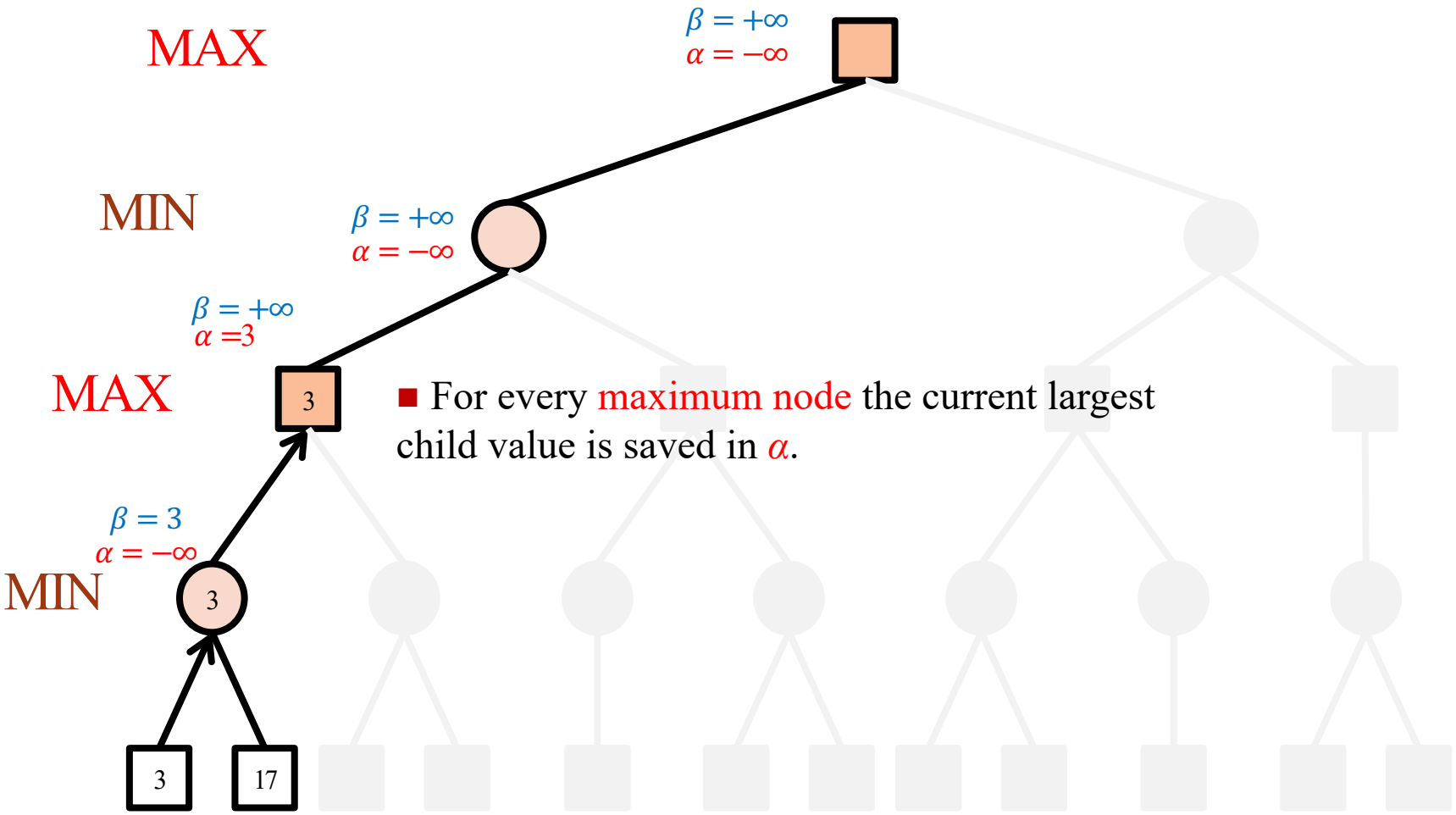
- For every minimum node the current smallest child value is saved in β.

A-B剪枝算法



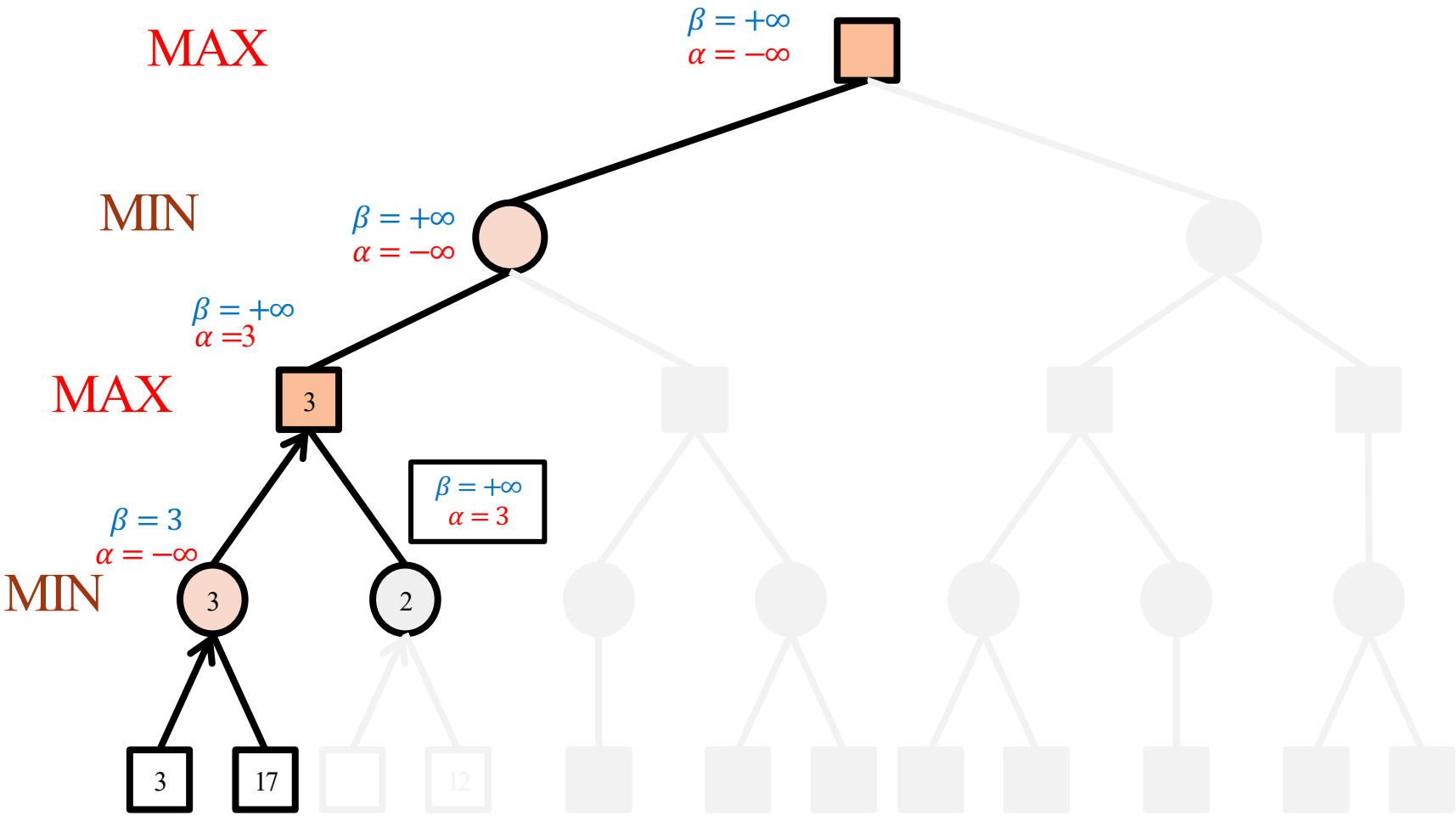
- For every minimum node the current smallest child value is saved in β.

A-B剪枝算法

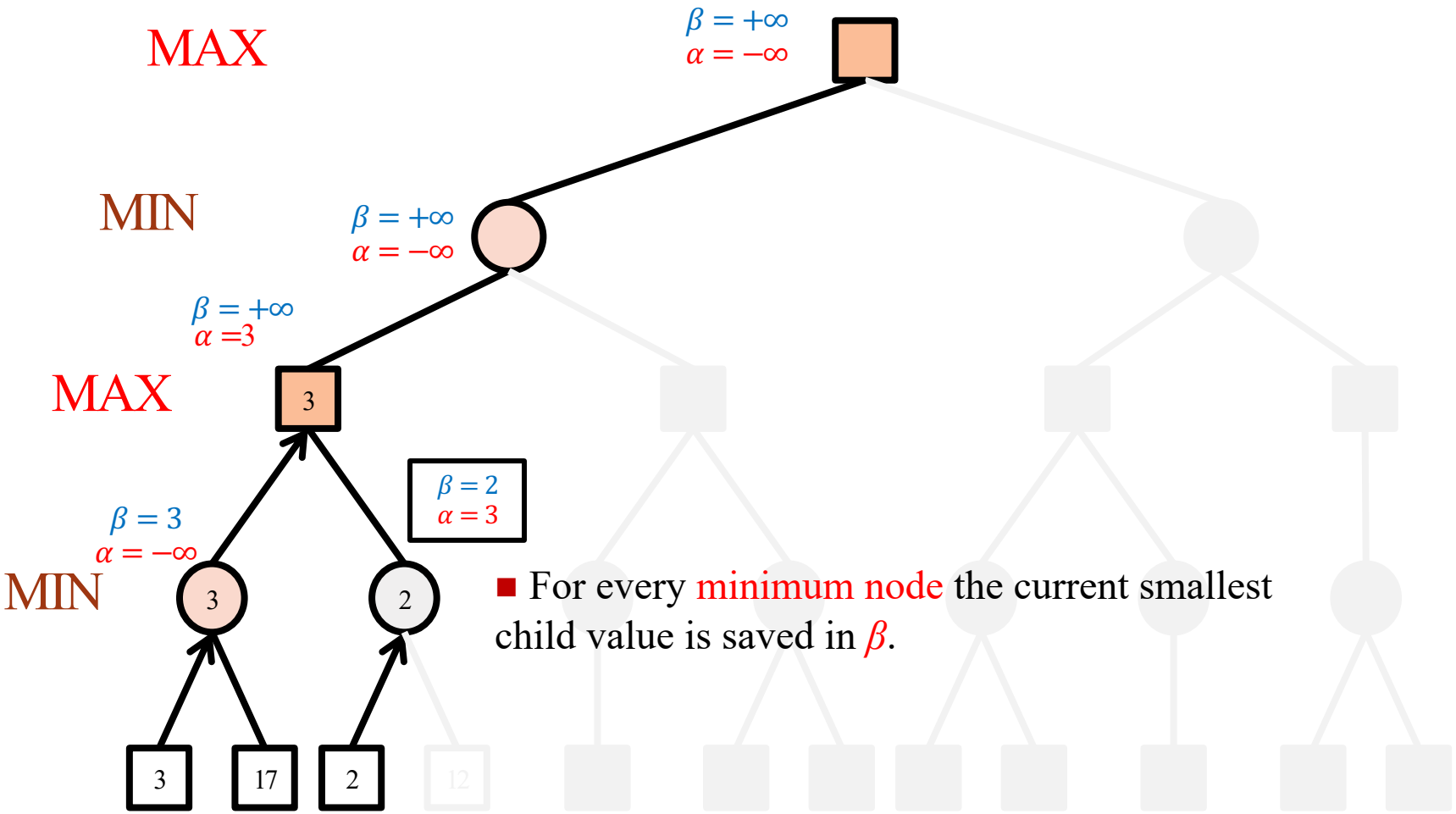


- For every maximum node the current largest child value is saved in α.

A-B剪枝算法

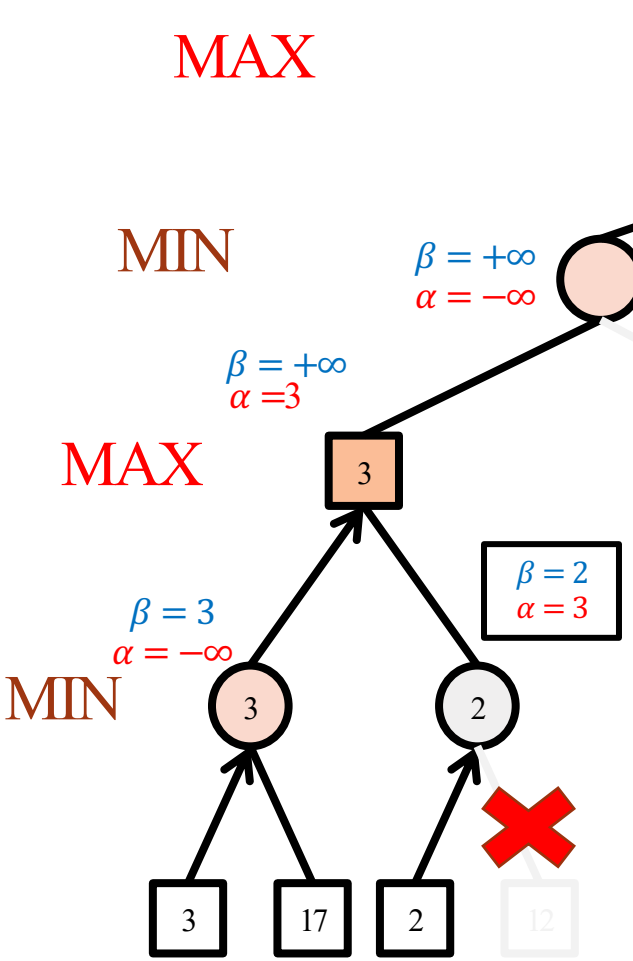

MAX
$\beta = +\infty$
$\alpha = -\infty$
MIN
$\beta = +\infty$
$\alpha = -\infty$
$\beta = +\infty$
$\alpha = 3$
MAX
3
$\beta = +\infty$
$\alpha = 3$
$\beta = 3$
$\alpha = -\infty$
MIN
3
2
3
17
12

A-B剪枝算法



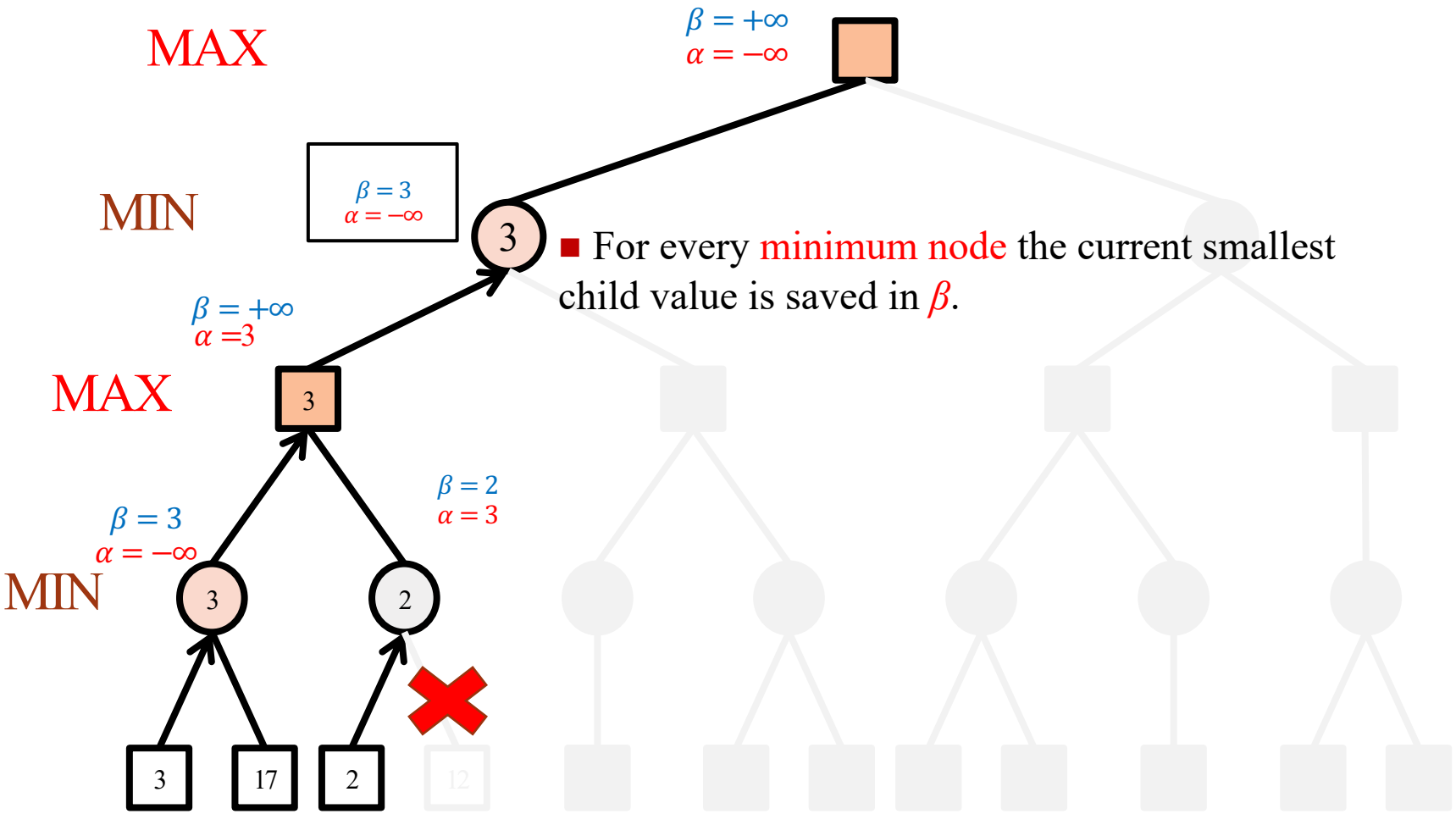
- For every minimum node the current smallest child value is saved in β.

A-B剪枝算法



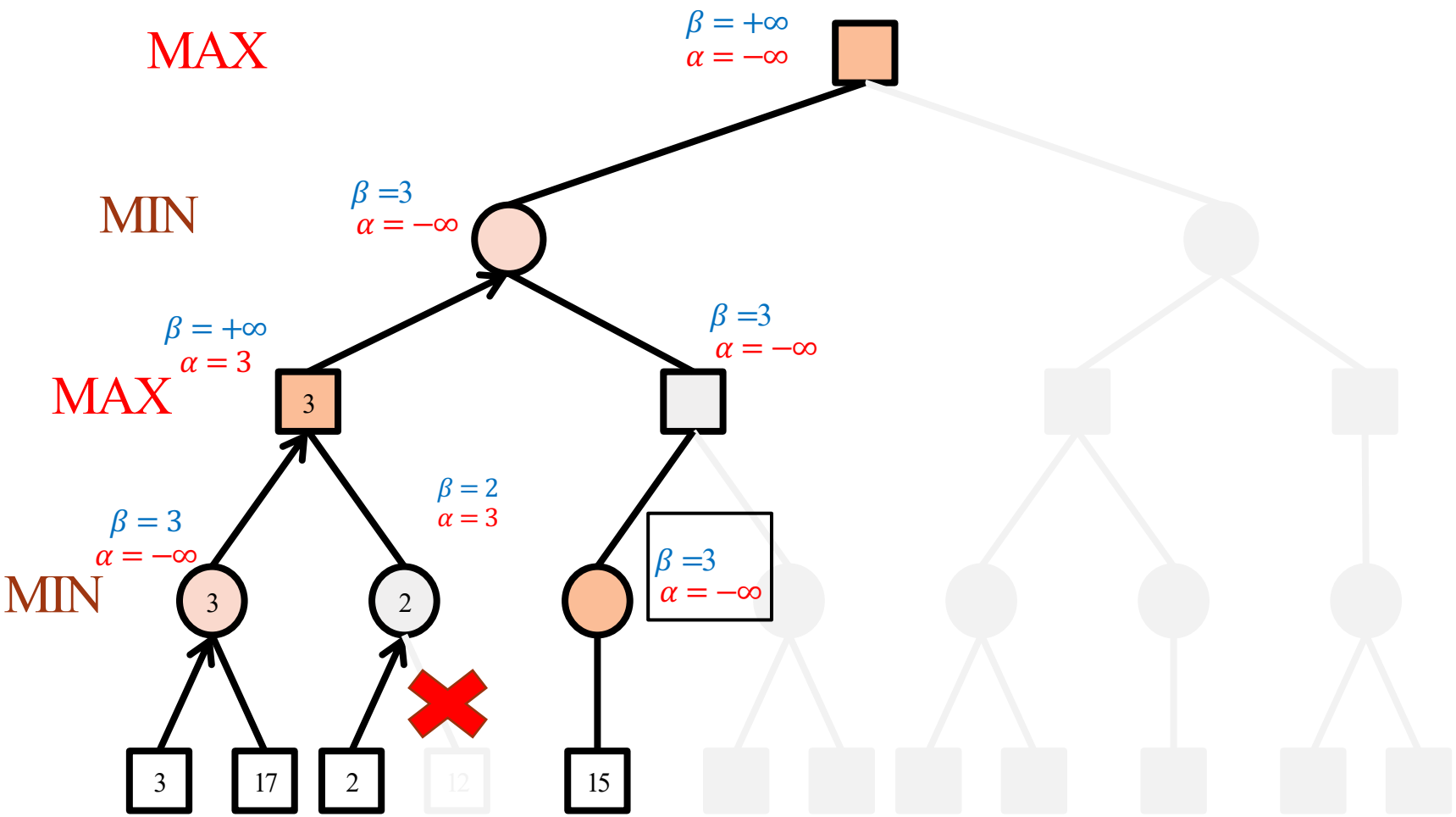
■If at a minimum node k the current value $\beta \leq \alpha$, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.

A-B剪枝算法

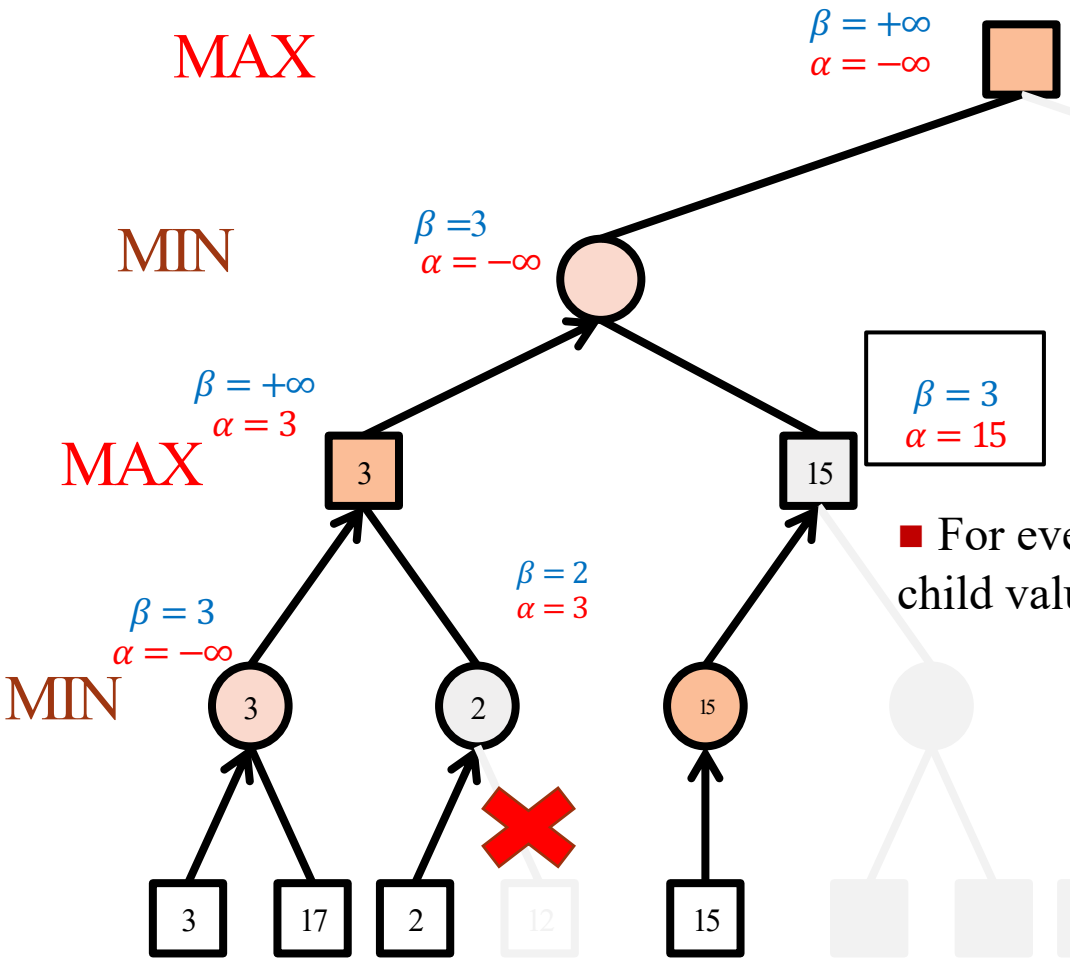


- For every minimum node the current smallest child value is saved in β.

A-B剪枝算法

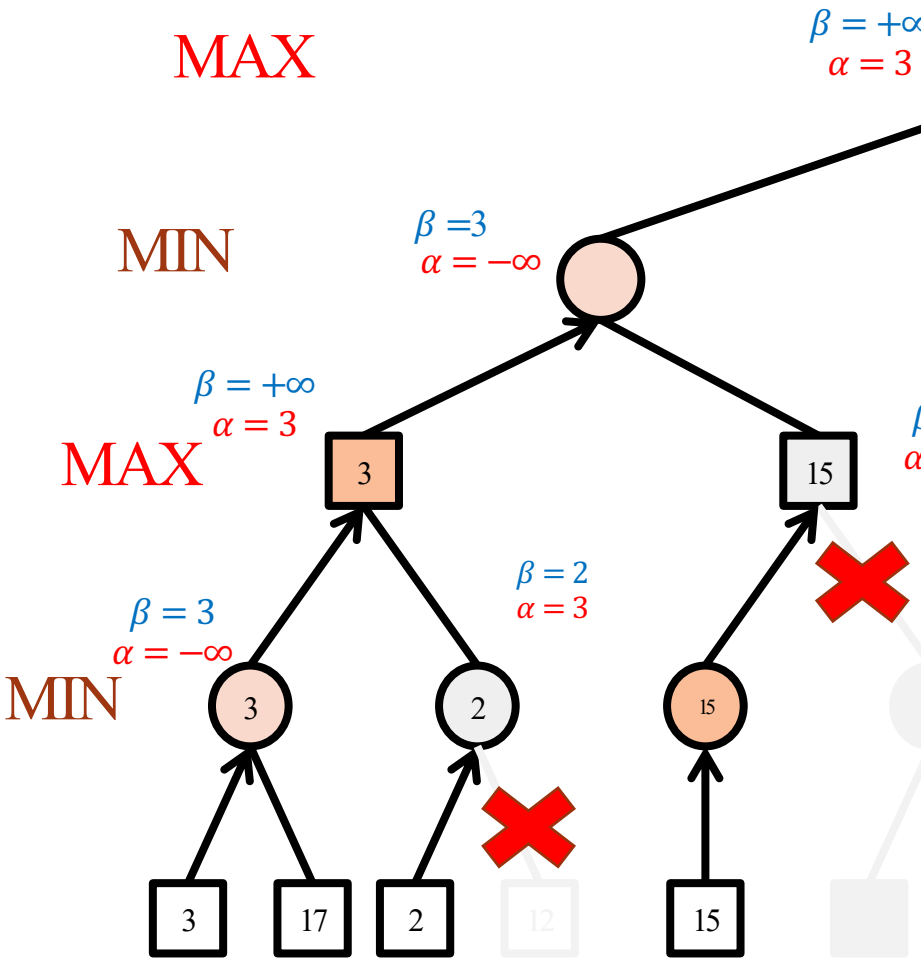


A-B剪枝算法



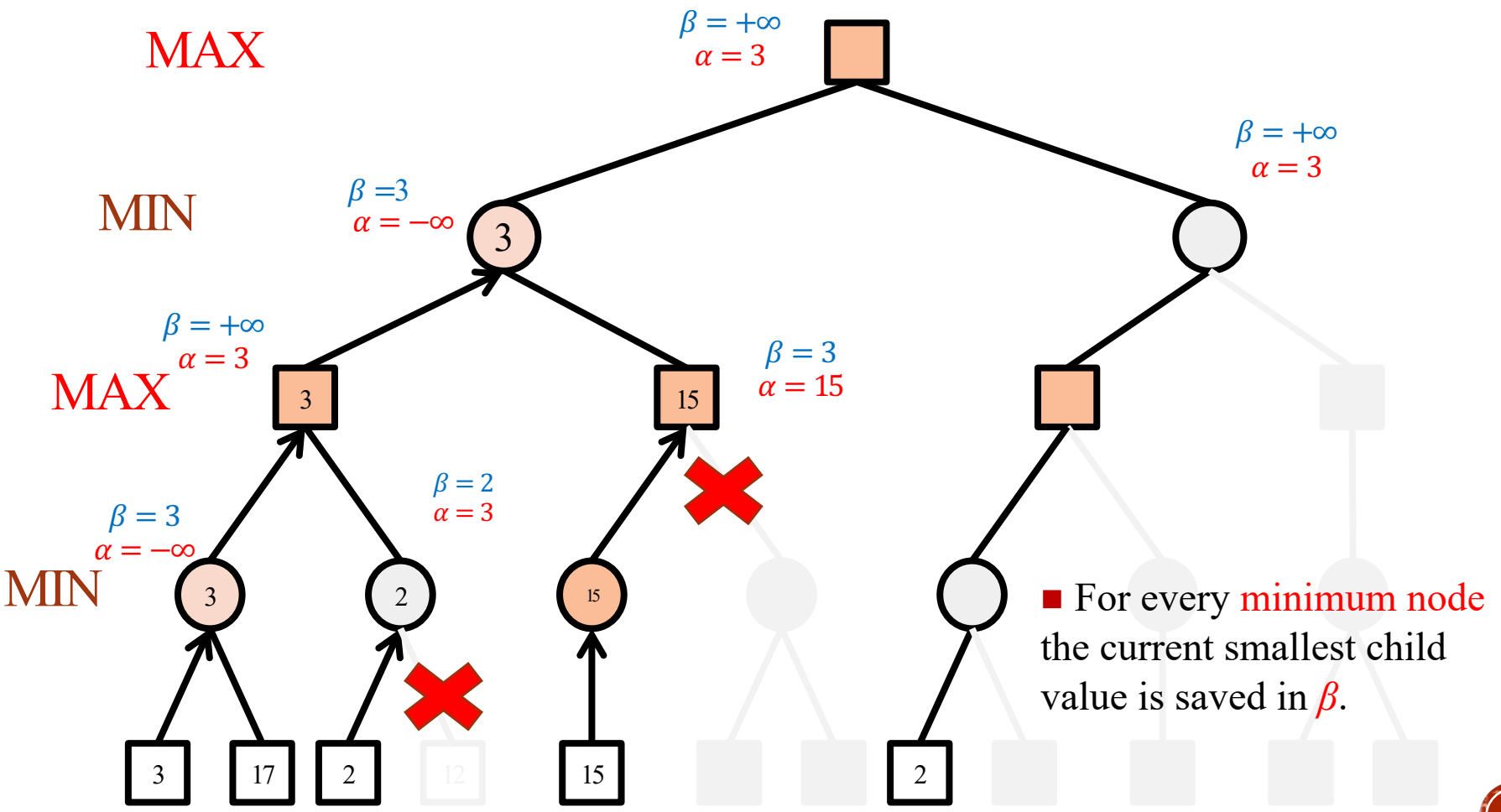
- For every maximum node the current largest child value is saved in α.

A-B剪枝算法



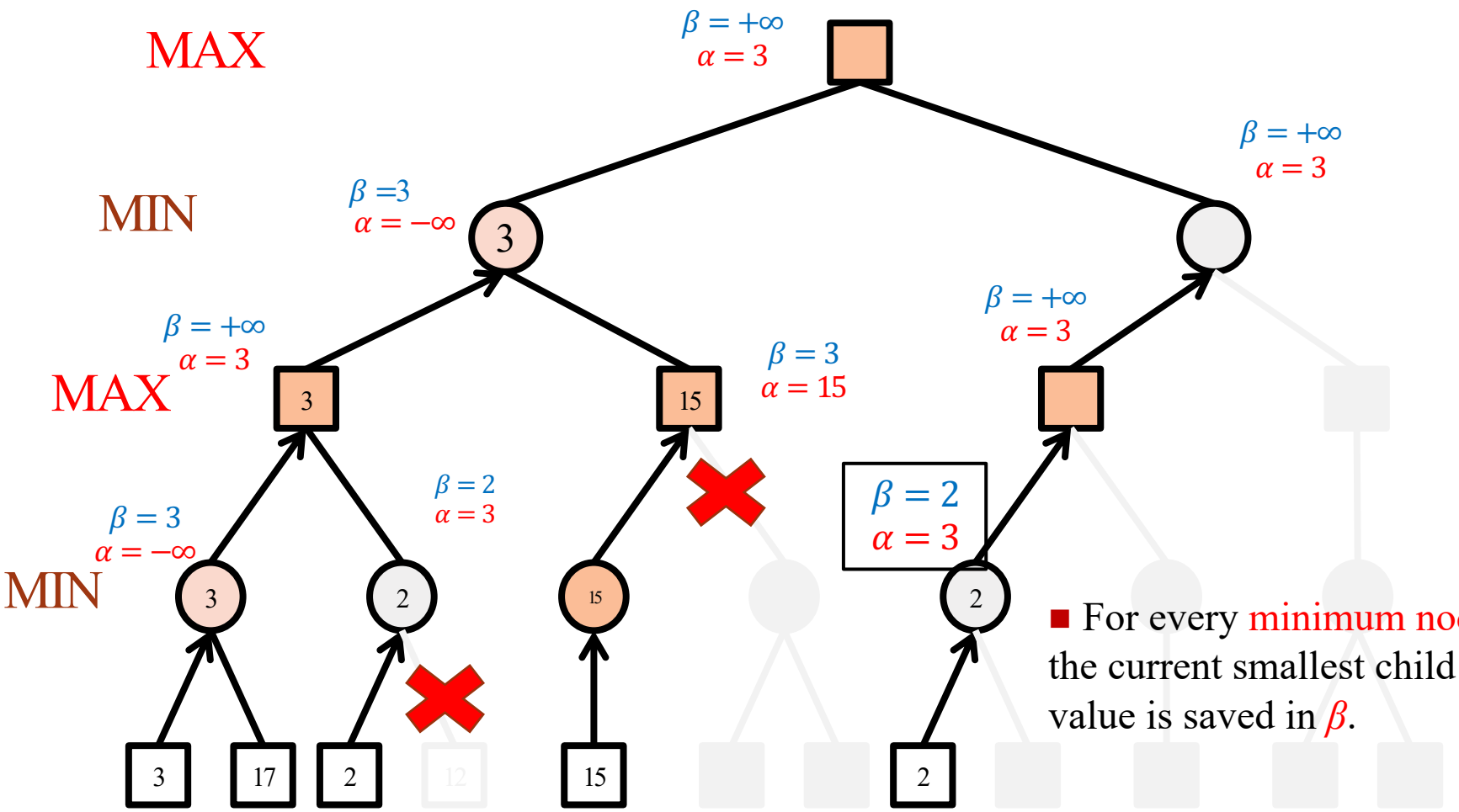
■If at a maximum node l the current value $\alpha \geq \beta$, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.

A-B剪枝算法



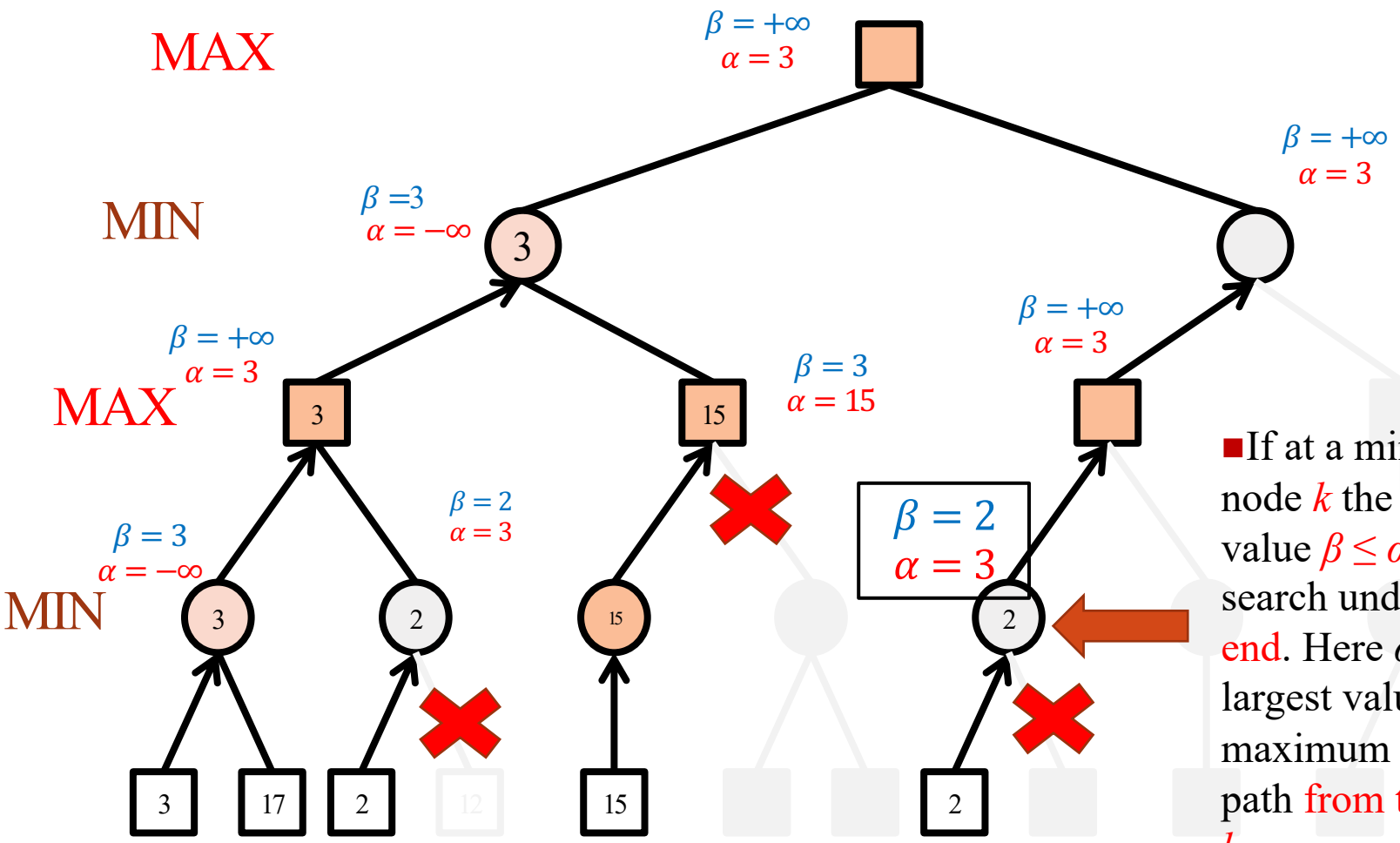
- For every minimum node the current smallest child value is saved in β.

A-B剪枝算法



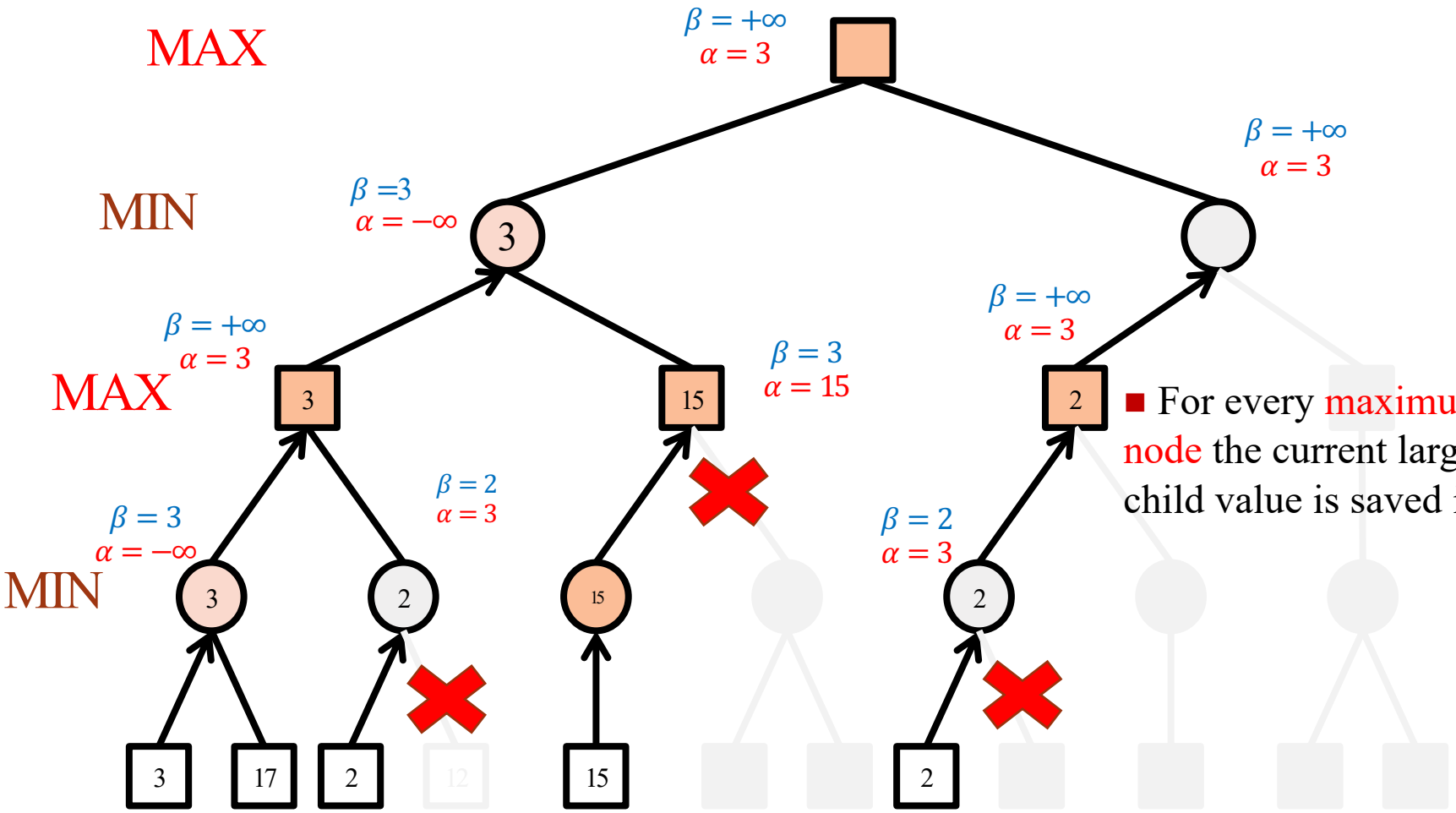
■ For every minimum node the current smallest child value is saved in β.

A-B剪枝算法



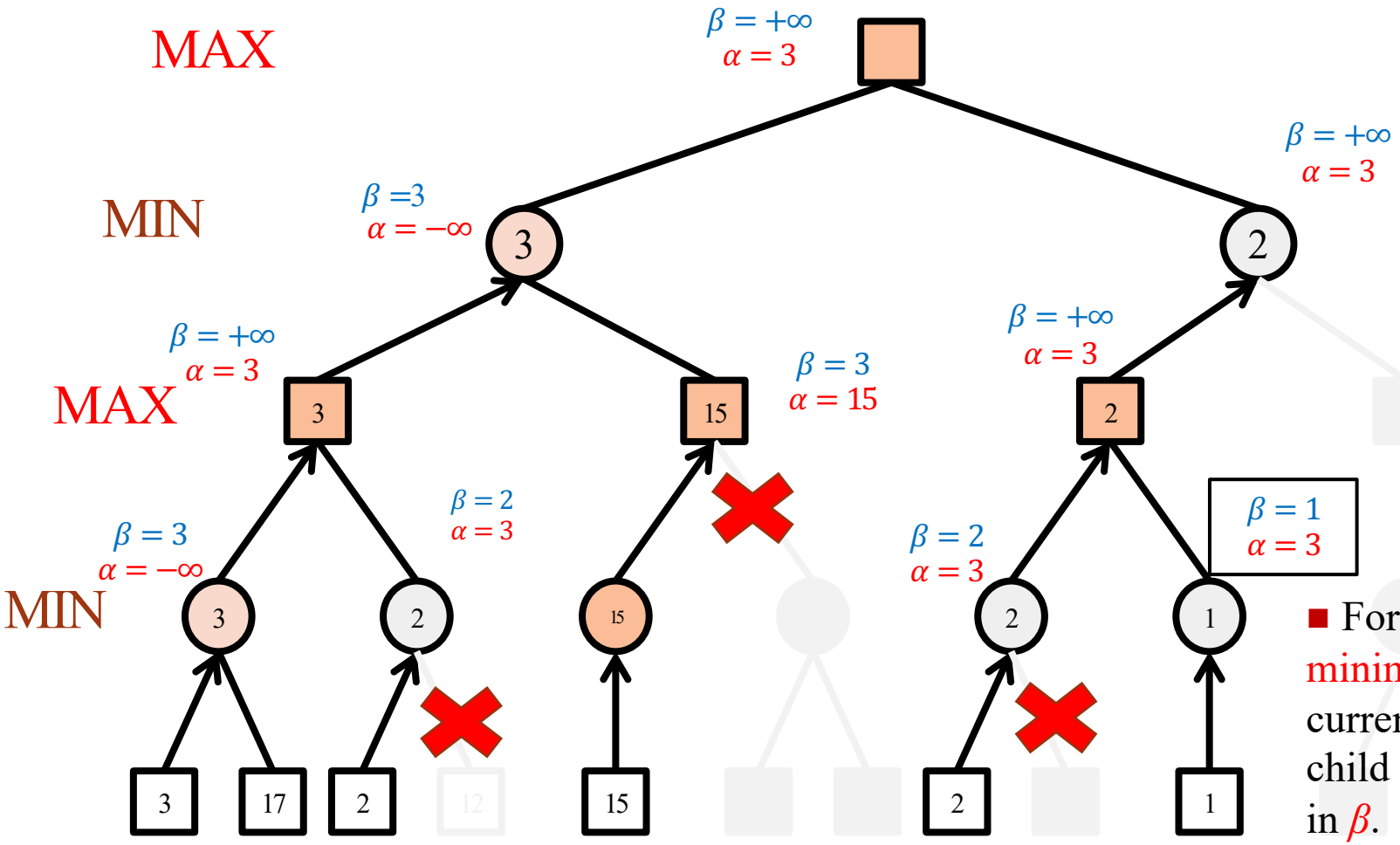
■If at a minimum node k the current value $\beta \leq \alpha$, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.

A-B剪枝算法



- For every maximum node the current largest child value is saved in α.

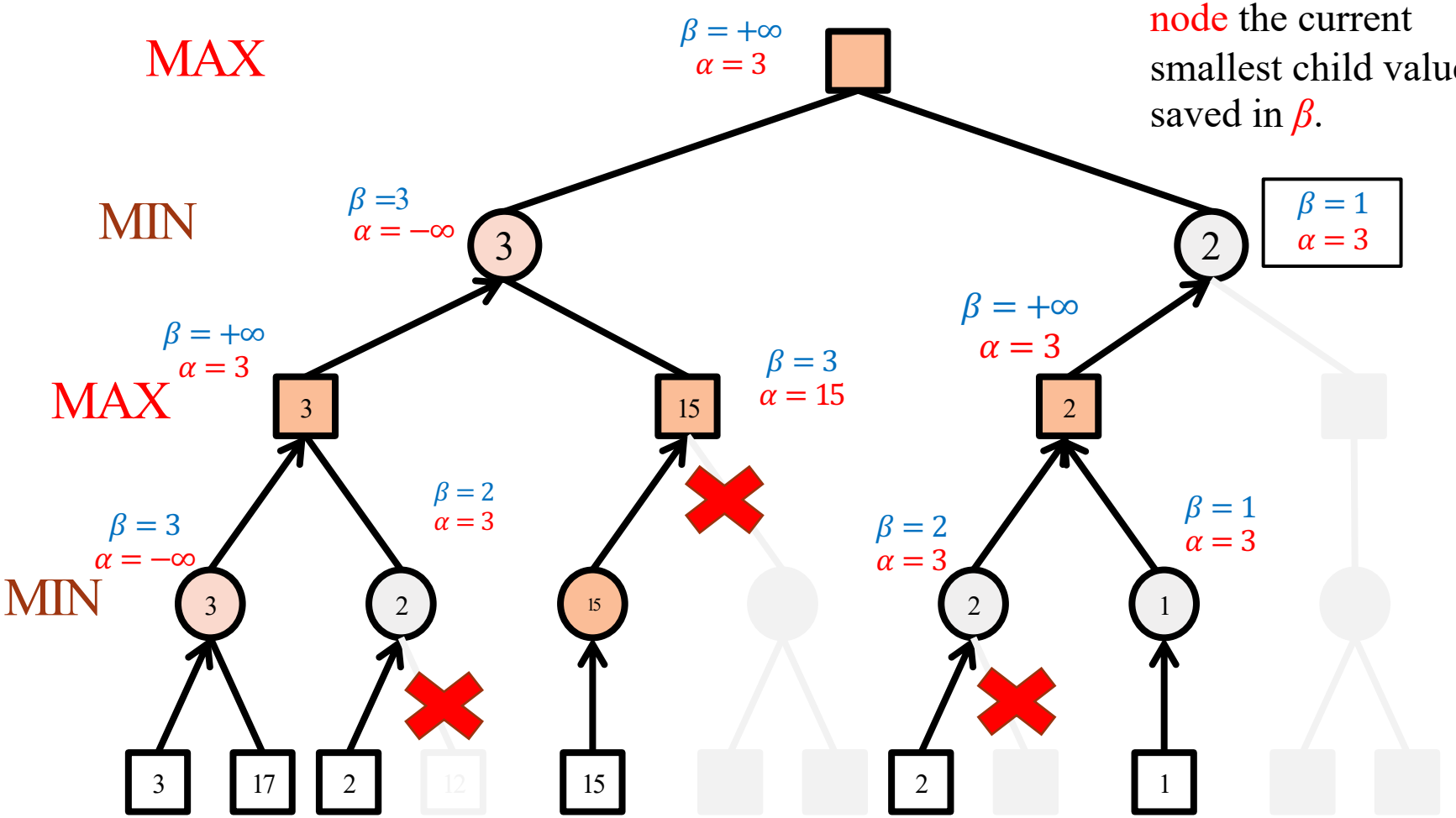
A-B剪枝算法



■ For every minimum node the current smallest child value is saved in β.

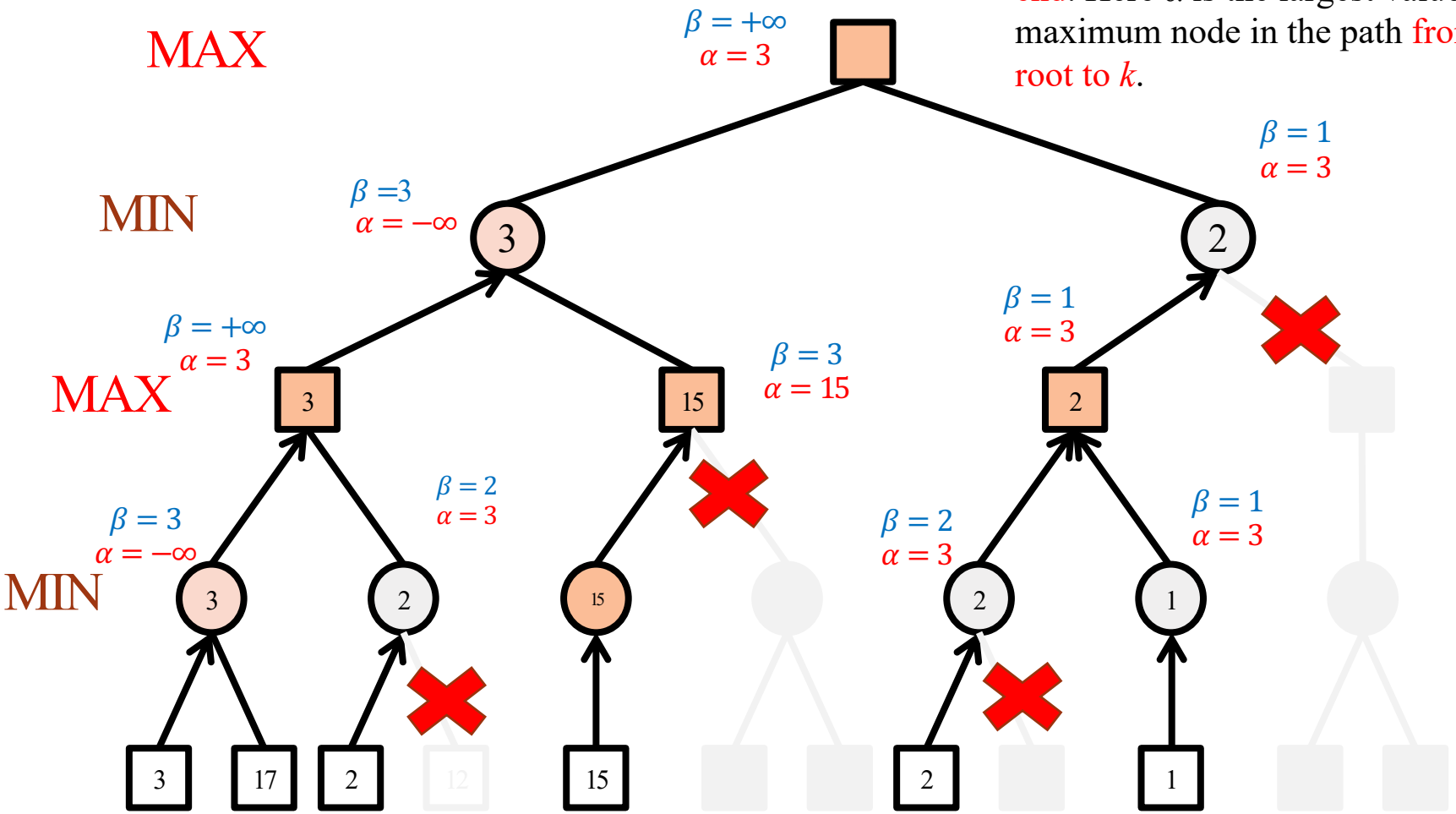
A-B剪枝算法

- For every minimum node the current smallest child value is saved in β.



A-B剪枝算法

■If at a minimum node k the current value $\beta \leq \alpha$, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.



A-B剪枝算法

- For every maximum node the current largest child value is saved in α.