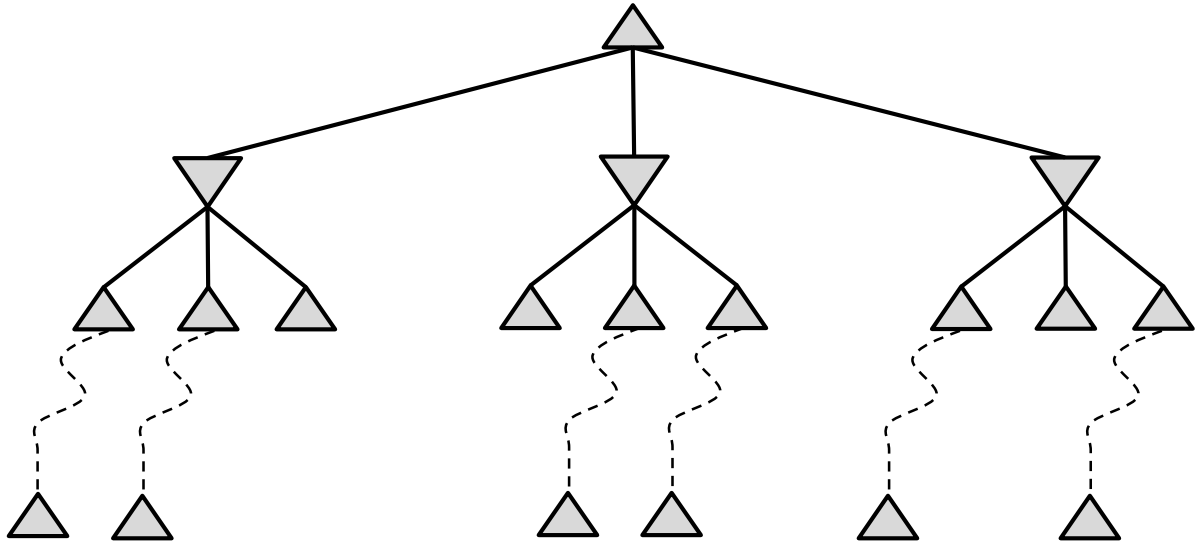
5. Adversarial Search

Contents:

Overview 概述

- The problem with minimax search: 采用minimax搜索的问题：
  - Number of game states is exponential in depth of the tree.
    博弈状态的量随着树的深度呈现指数式增长。
  - We can't eliminate the exponent, but we can effectively cut it in half.
    我们无法消除这种指数，但实际上我们可以将它剪掉一半。



Overview 概述

- The trick to solve the problem: 解决该问题的技巧
  - Compute correct minimax decision without looking at every node in game tree.
    计算正确的minimax决策而不考虑博弈树的每个节点。
  - That is, use “pruning” to eliminate large parts of the tree.
    就是说，采用“剪枝”方法来消除该树的大部分。
- What is alpha–beta pruning 什么是alpha–beta剪枝
  - It is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm.
    是一种搜索算法，旨在削减由minimax算法评价的节点数量。

If you have an idea that is surely bad, don’t take the time to see how truly awful it is.
-- Pat Winston (Director, MIT AI Lab, 1972-1997)

Why Called Alpha-Beta 为何称其为Alpha-Beta

- Alpha–beta pruning gets its name from the following two parameters:

  Alpha-Beta剪枝从如下两个参数得到其名称：

  - α: highest-value we have found so far at any point along the path for MAX.

    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: lowest-value we have found so far at any point along the path for MIN.

    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。

- Alpha–beta search respectively: Alpha-Beta搜索依次完成如下动作：

  - updates the values of α and β as it goes along, and

    边搜索边更新α和β的值，并且

  - prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN.

    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枝。

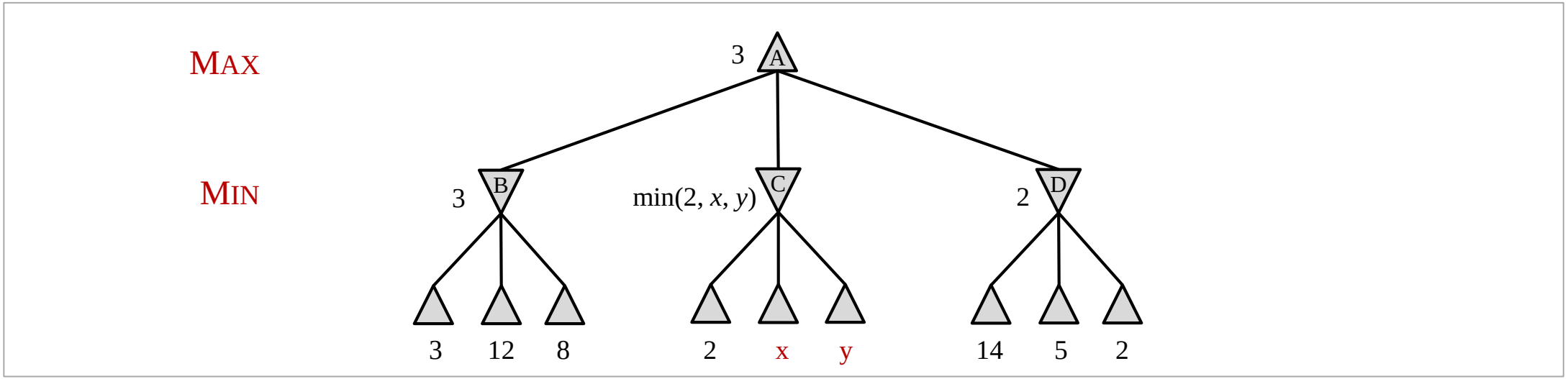
Alpha-Beta Search Algorithm Alpha-Beta搜索算法

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
        if v ≥ β then return v else α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
        if v ≥ α then return v else β ← MIN(β, v)
    return v
```

Example: Game Tree Using Minimax 采用Minimax的博弈树

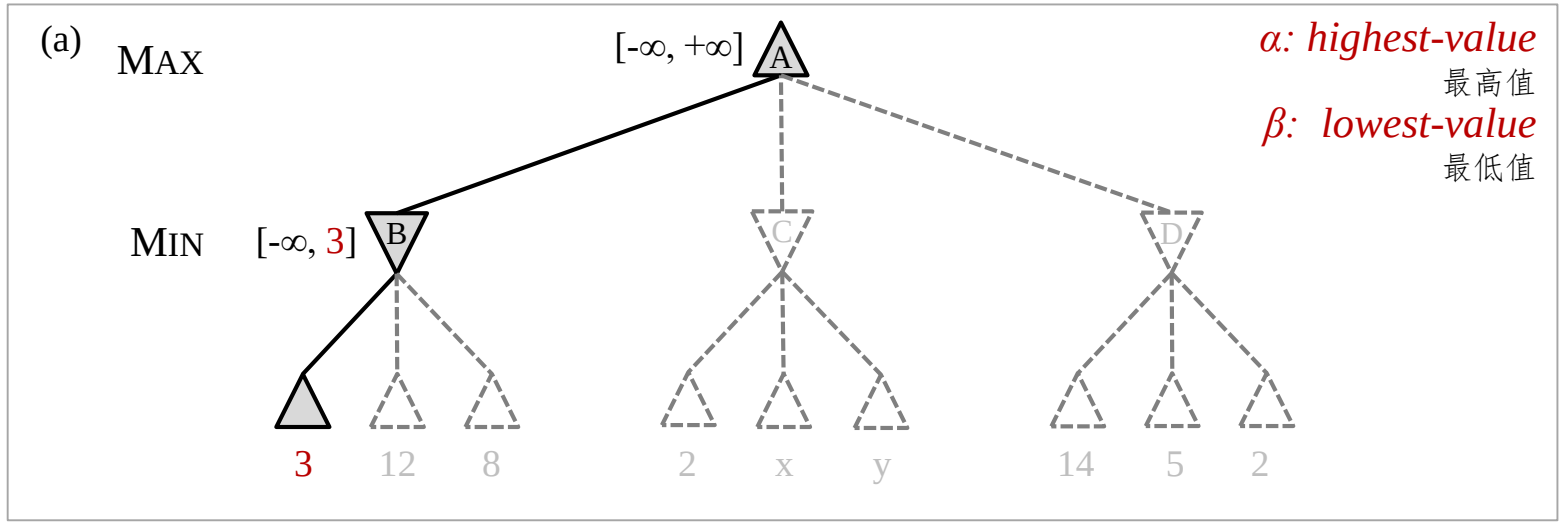


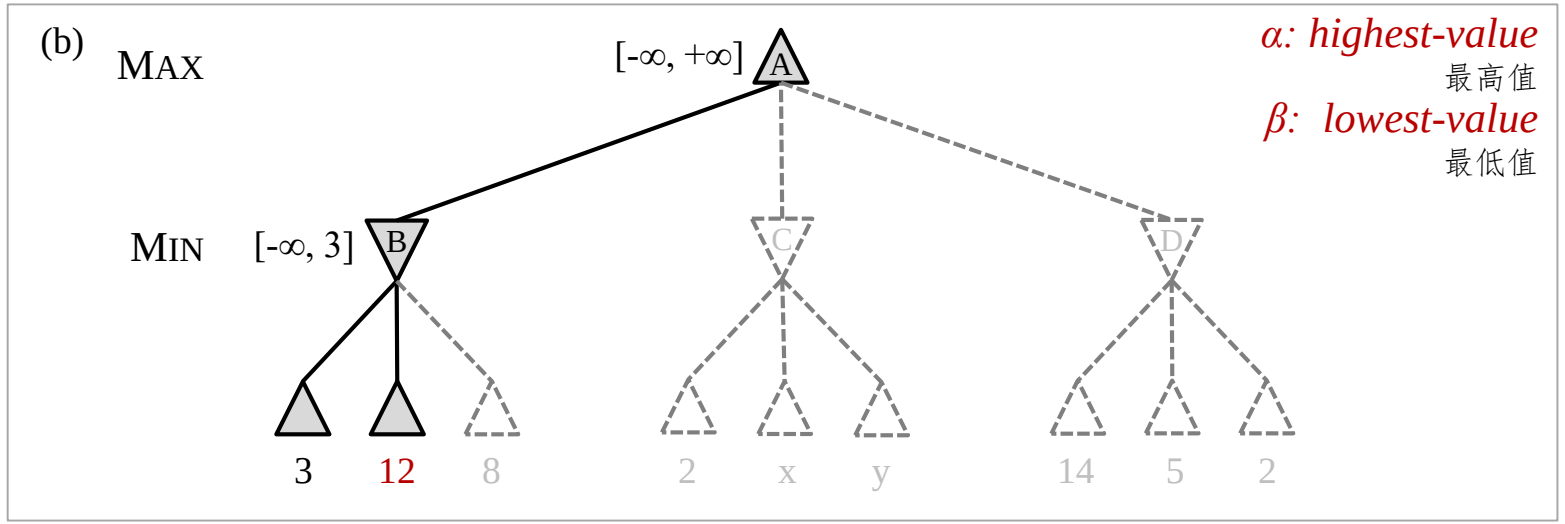
- The value of root node is given by: 根节点的值由如下方法得出：

$$\begin{aligned} \text{MINIMAX}(root) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \le 2 \\ &= 3. \end{aligned}$$

no pruning!
无剪枝！

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树





Initial value: 初始值：

$A[\alpha=-\infty, \beta=+\infty]$

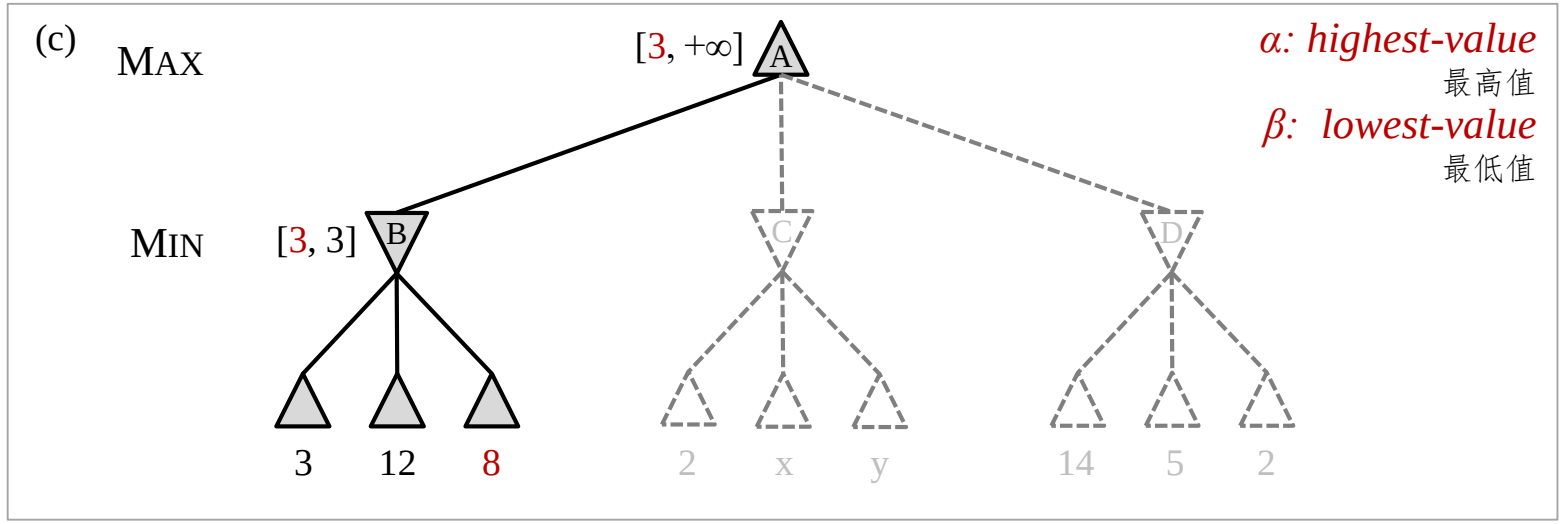
(a) The 1st leaf below *B* has the value 3. Hence, *B*, as a MIN node, $B[\beta=3]$.

*B*下面第一个叶节点的值为3。因此，*B*作为MIN节点，$B[\beta=3]$。

(b) The 2nd leaf below *B* has a value of 12; MIN would avoid this move, still $B[\beta=3]$.

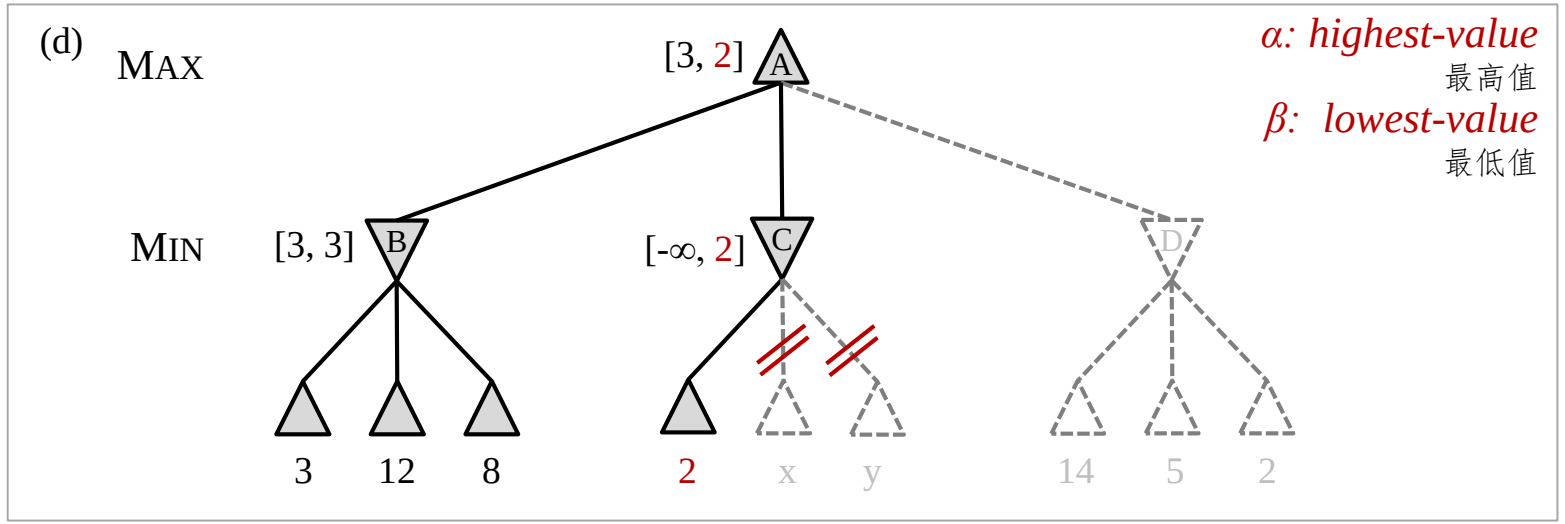
*B*下面第二个叶节点的值为12； MIN将回避这个移动，仍然是$B[\beta=3]$。

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



(c) The 3rd leaf below *B* has a value of 8; so exactly MIN node *B*[β=3]. Now, we can infer *B*[α=3], because MAX has *A*[$\alpha \geq 3$].

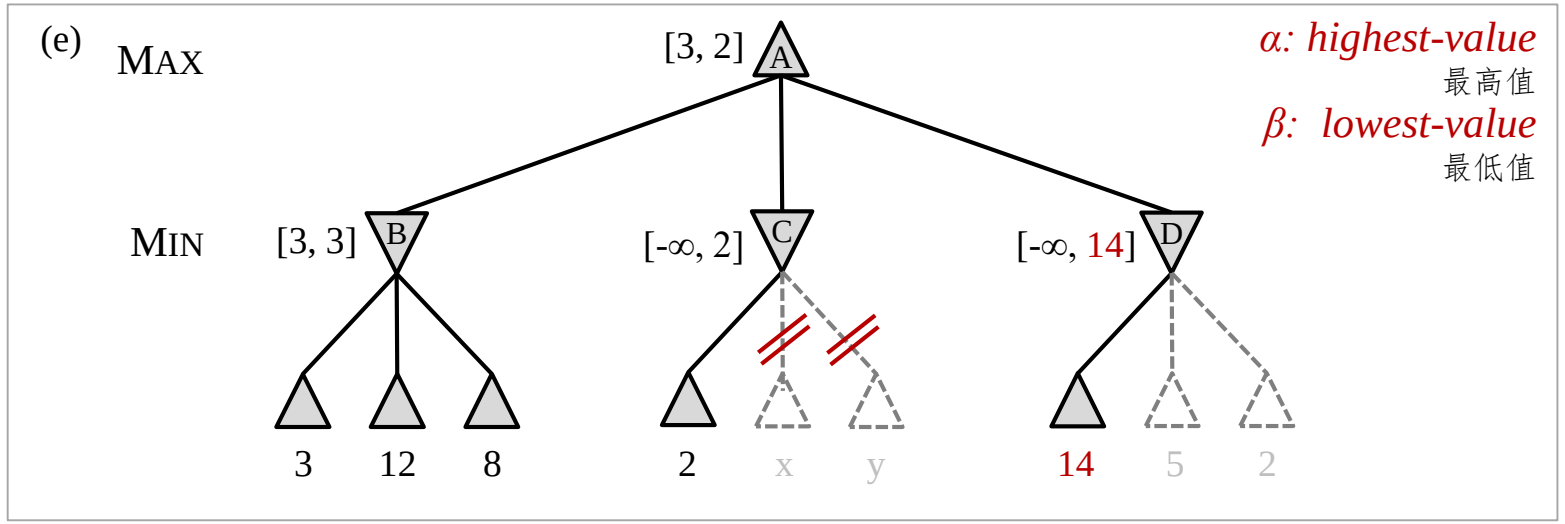
*B*下面第三个叶节点的值为8；故MIN节点正是*B*[β=3]。现在，因为MAX为*A*[$\alpha \geq 3$]，我们能够推出*B*[α=3]。



(d) The 1st leaf below *C* has the value 2, hence, as a MIN node *C*[β=2], and *B*[β=3]>*C*[β=2], so MAX would never choose *C*. Therefore just prune all successor of *C* (*α–β* pruning).

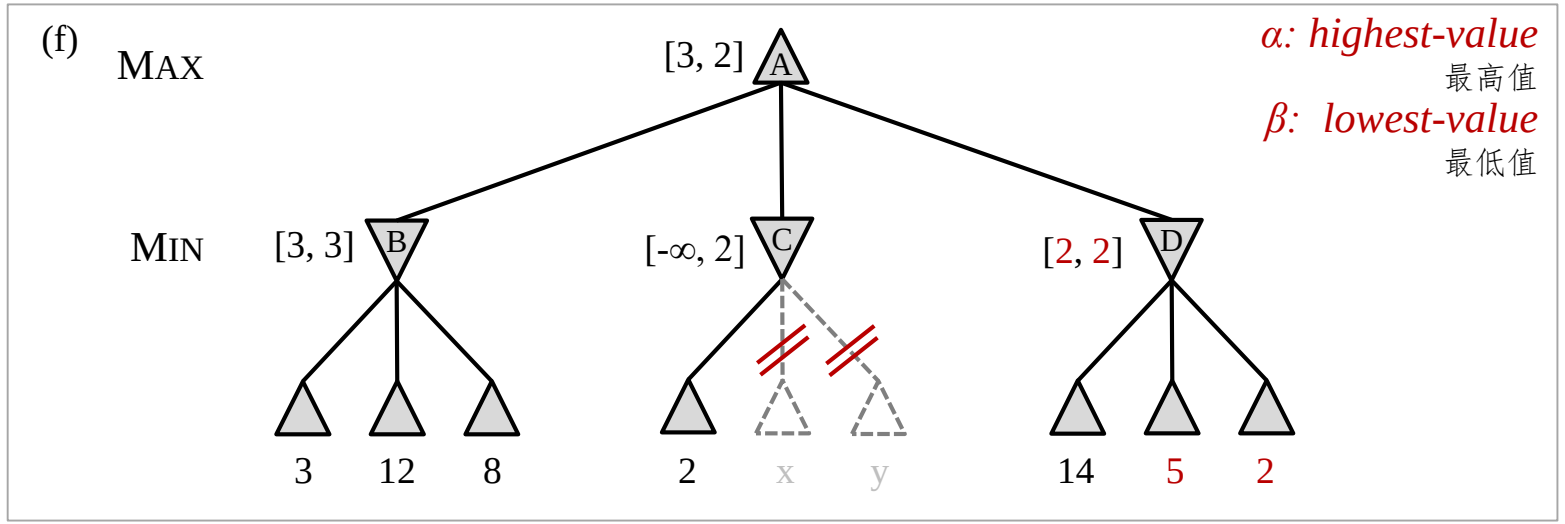
C下面第一个叶节点的值为2，因此，由于MIN节点*C*[β=2], 且*B*[β=3]>*C*[β=2]，故MAX将不会选择C，所以只需剪掉C的所有后继节点 (*α–β* pruning)。

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



(e) The 1st leaf below *D* is 14, $D[\beta \leq 14]$, so we need to keep exploring *D*'s successor states. We now have bounds on all of root's successors, so $A[\beta \leq 2]$.

*D*下面第一个叶节点为14，$D[\beta \leq 14]$，故我们需要不断搜索D节点的后继状态。到此我们已经遍布了根节点的所有后继节点，故$A[\beta \leq 2]$。



(f) The 2nd successor of *D* is worth 5, so keep exploring. The 3rd successor is worth 2, so $D[\beta=2]$. MAX's decision at the root keeps $A[\beta=2]$.

*D*的第2个后继节点的值等于5，故不断搜索，第3个后继节点等于2，故$D[\beta=2]$。根节点MAX的抉择保持$A[\beta=2]$。

General Principle of Alpha-Beta Pruning Alpha-Beta剪枝的一般原则

- Alpha–beta pruning can be applied to trees of any depth, and often possible to prune entire subtrees rather than just leaves.

  Alpha-Beta剪枝可被用于任意深度的树，并且常常可以剪去整个子树而不仅仅是叶节点。

- The general principle: 一般原则：
  - Consider a node *n* somewhere in the tree, such that Player has a choice of moving to that node.

    设某个节点*n*位于树的某处，于是玩家选择移向那个节点。
  - If Player has a better choice *m* at parent node of *n*, or at any choice point further up, then *n will never be reached* in actual play.

    若玩家在位于*n*的父节点或更上层处有更好的选择*m*，则在实战中完全没必要抵达*n*。