COMP 472 Artificial Intelligence: Adversarial Search part 4
Alpha-Beta Pruning video 2

- Russell & Norvig: Sections 5.1, 5.2, 5.3

Today

- Adversarial Search
  1. Minimax
  2. Alpha-Beta Pruning 

Alpha-Beta Pruning

是对MiniMax的一个优化

- Optimization over Minimax, that:
  - ignores (cuts off/prunes) branches of the tree that cannot contribute to the solution
  - reduces branching factor
  - allows deeper search with same effort

MiniMax会计算n-ply那一层所有的e(n)

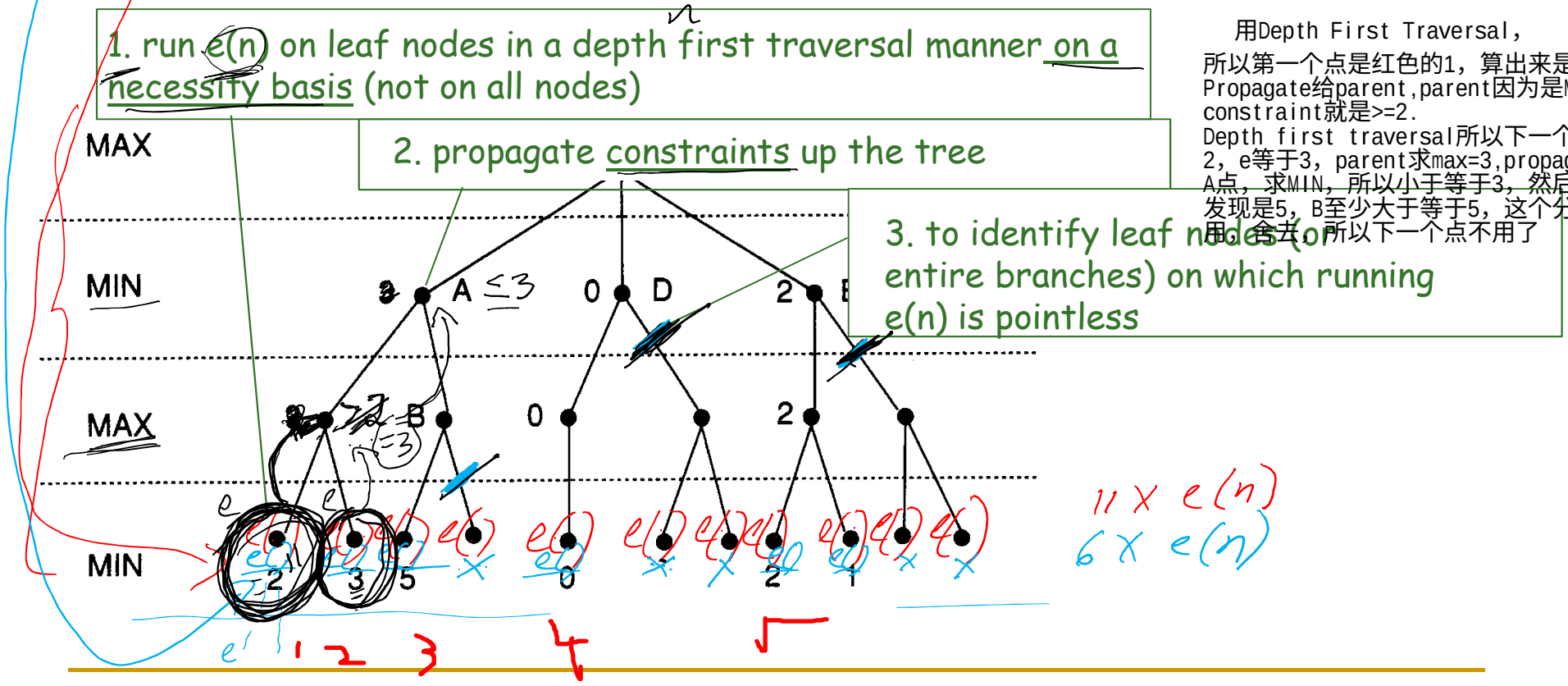
这里提前截断了那些不可能的subtree

因此那些subtree的e(n)既不用算了，减少了en运算量

减少的运算量允许你进行更深的search

Alpha-Beta Pruning: Example 1

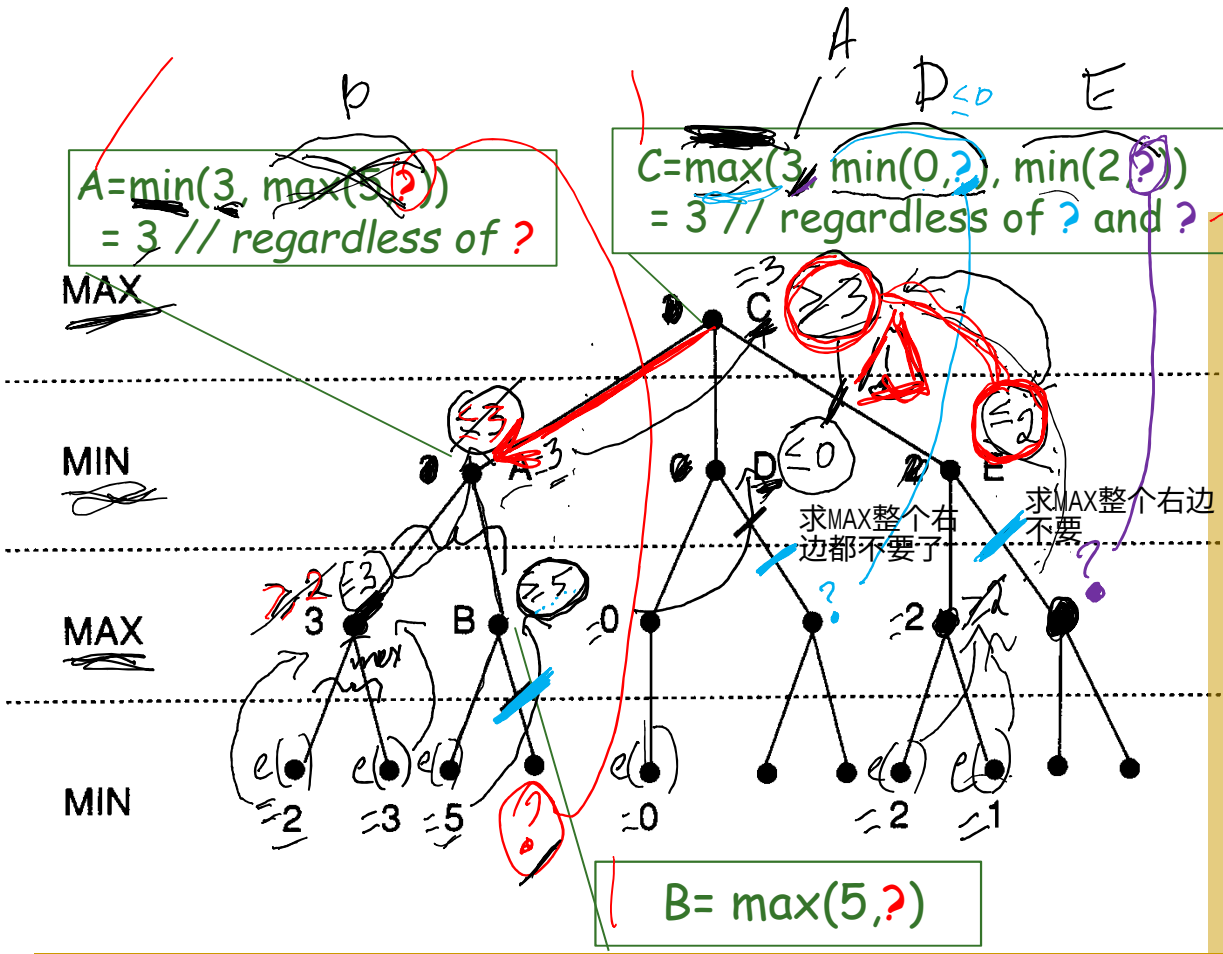
- With Minimax, we look at all nodes at depth *n*
- With α-β pruning, we ignore branches that could not possibly contribute to the final decision



source: G. Luger (2005)

用Depth First Traversal，
所以第一个点是红色的1，算出来是2，Propagate给parent，parent因为是MAX，constraint就是>=2.
Depth first traversal所以下一个点是2，e等于3，parent求max=3，propagate到A点，求MIN，所以小于等于3，然后点3，发现是5，B至少大于等于5，这个分支没用，舍去，所以下一个点不用了

Alpha-Beta Pruning: Example 1



B will be >= 5

So we can ignore B's right branch, because A must be 3

D will be <= 0

But C will be >= 3

So we can ignore D's right branch

E will be <= 2.

So we can ignore E's right branch

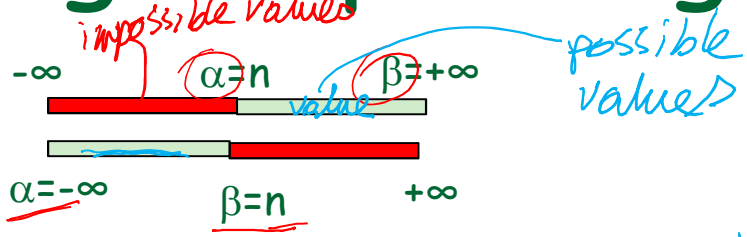
Because C will be 3.

source: G. Luger (2005)

Alpha-Beta Pruning – α pruning

就是前面的大于等于，小于等于

- α = minimum possible value　　value ≥ n
- β = maximum possible value　　value ≤ n



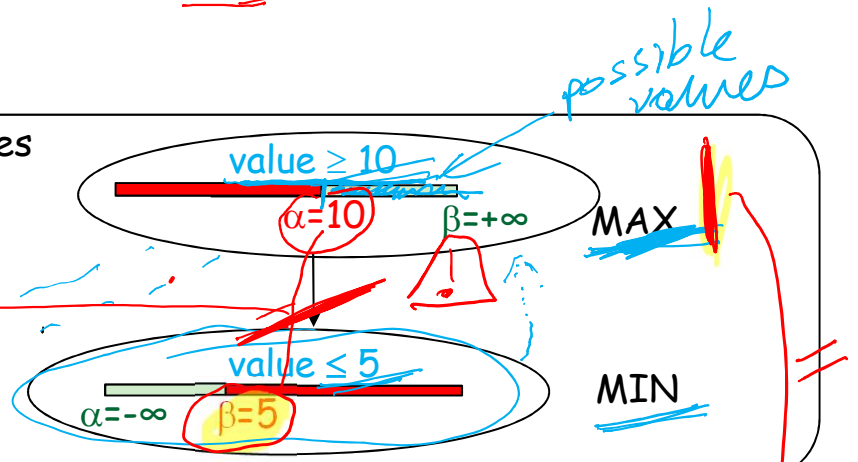
- Alpha pruning – parent is a MAX node

1. if MAX node's α = n, then we can prune branches from a MIN descendant that has a β ≤ n.

   if β_child ≤ α_ancestor → prune

假设parent是个MAX node
他的α=n,也就是说他至少大于等于n
那么child也就是MIN，如果
MIN的β小于等于n，说明
min取的值必然小于等于N
就可以舍去了，成功pruning

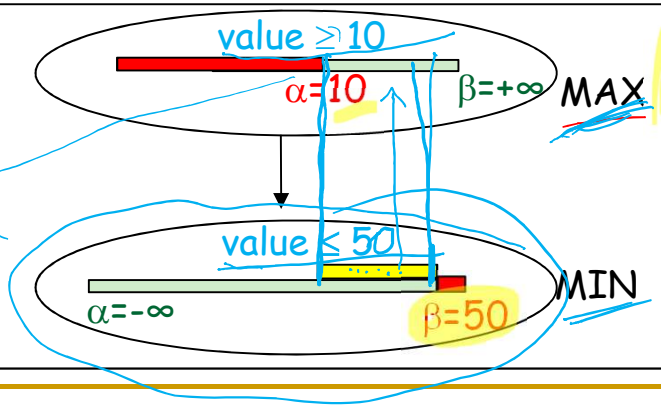
incompatible…
so stop searching this branch;
the value cannot come from there!



2. if β_child > α_ancestor → cannot prune

如果大于，不能舍去
有可能min取出来一个更大的值
MAX就进一步更新

compatible…
we need to search this branch;
the value could come from there!



MAX

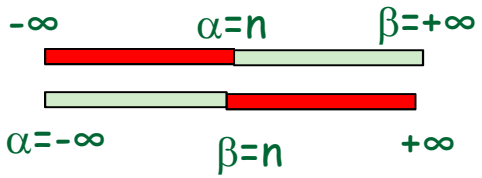
它本身：α大于等于n
要求child:β小于等于n
这时可以舍去

MIN

它本身：β小于等于n
要求child MAX：α大于等于n，
这时可以舍去
如果MAX小于等于n，有希望，我们还要算en

Alpha-Beta Pruning - β pruning

- α = minimum possible value — value ≥ n
- β = maximum possible value — value ≤ n

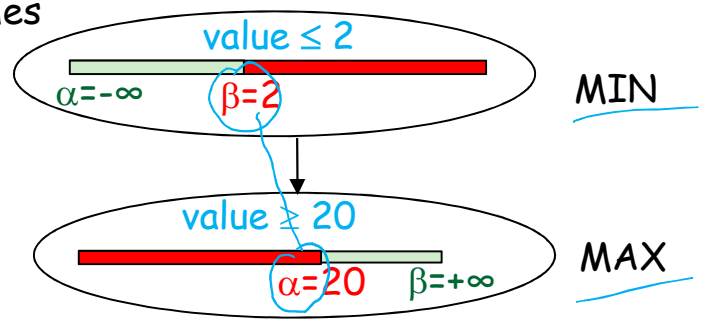


same

- Beta pruning – parent is a MIN node

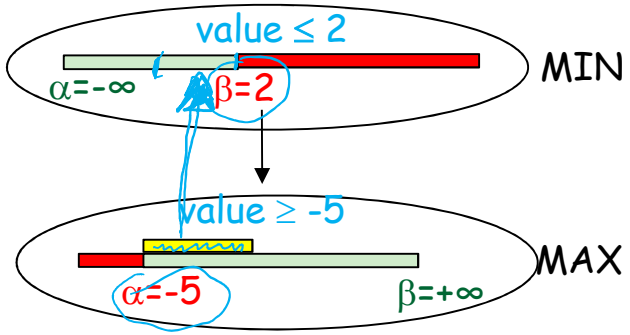
1. if a MIN node's β = n, then we can prune branches from a MAX descendant that has an α ≥ n.
   if $\alpha\_child \geq \beta\_ancestor$ → prune

incompatible…
so stop searching this branch;
the value cannot come from there!



2. if $\alpha\_child < \beta\_ancestor$ → cannot prune

compatible…
we need to search this branch;
the value could come from there!



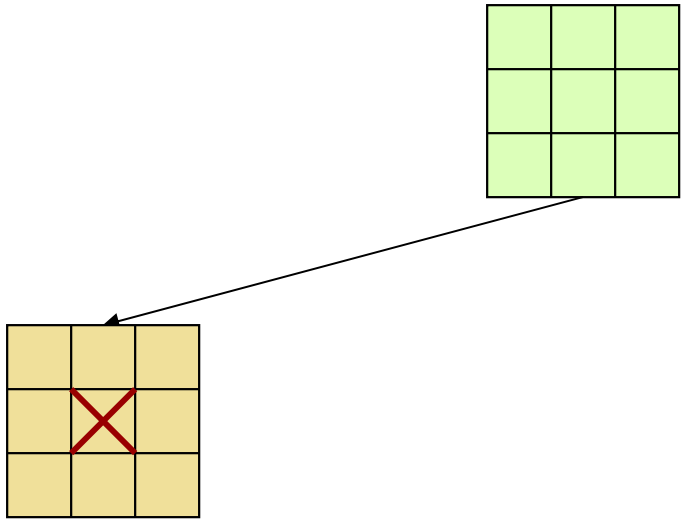
Alpha-Beta Pruning Algorithm

```
01 function alphabeta(node, depth, α, β, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node
04     if maximizingPlayer
05         v := -∞
06         for each child of node
07             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
08             α := max(α, v)
09             if β ≤ α
10                 break (* β cut-off *)
11         return v
12     else
13         v := ∞
14         for each child of node
15             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
16             β := min(β, v)
17             if β ≤ α
18                 break (* α cut-off *)
19         return v
```

Initial call:
`alphabeta(origin, depth, -∞, +∞, TRUE)`

source: http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Example with tic-tac-toe

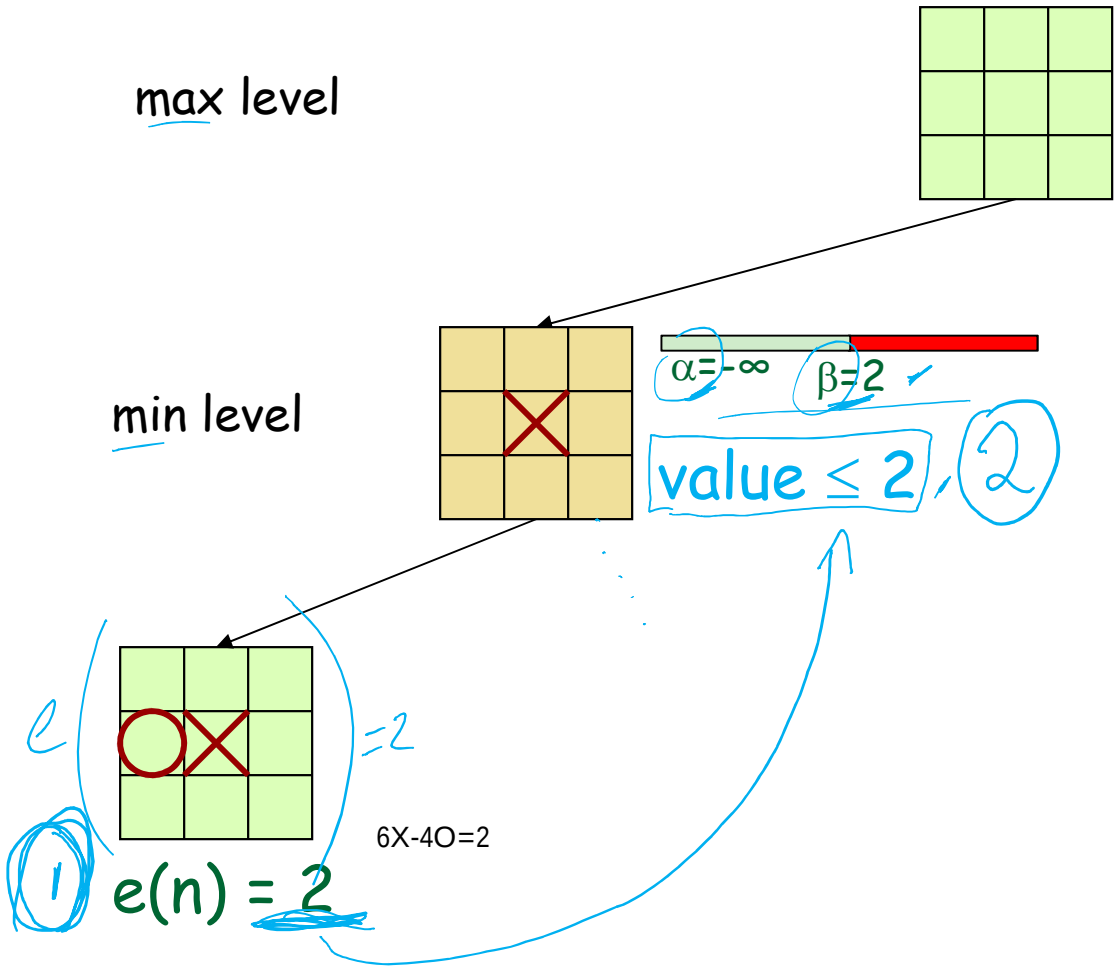


max level

min level

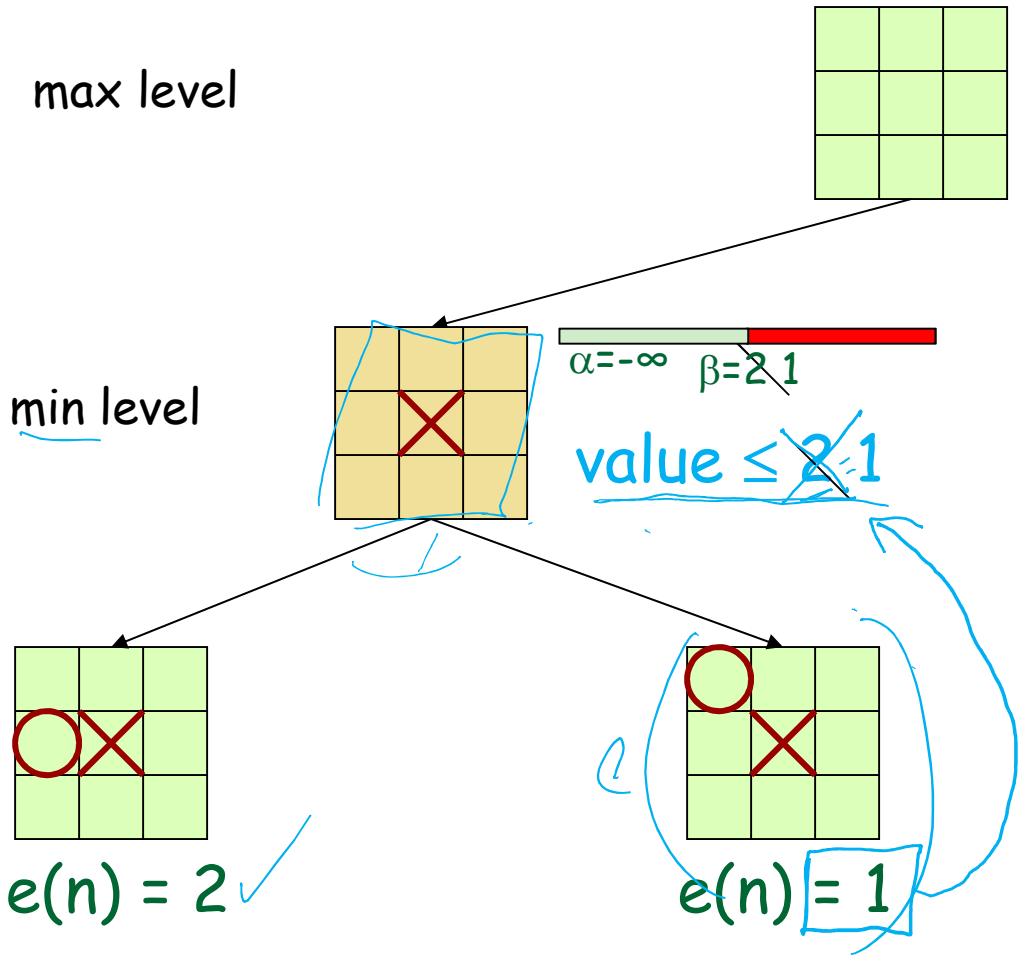
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



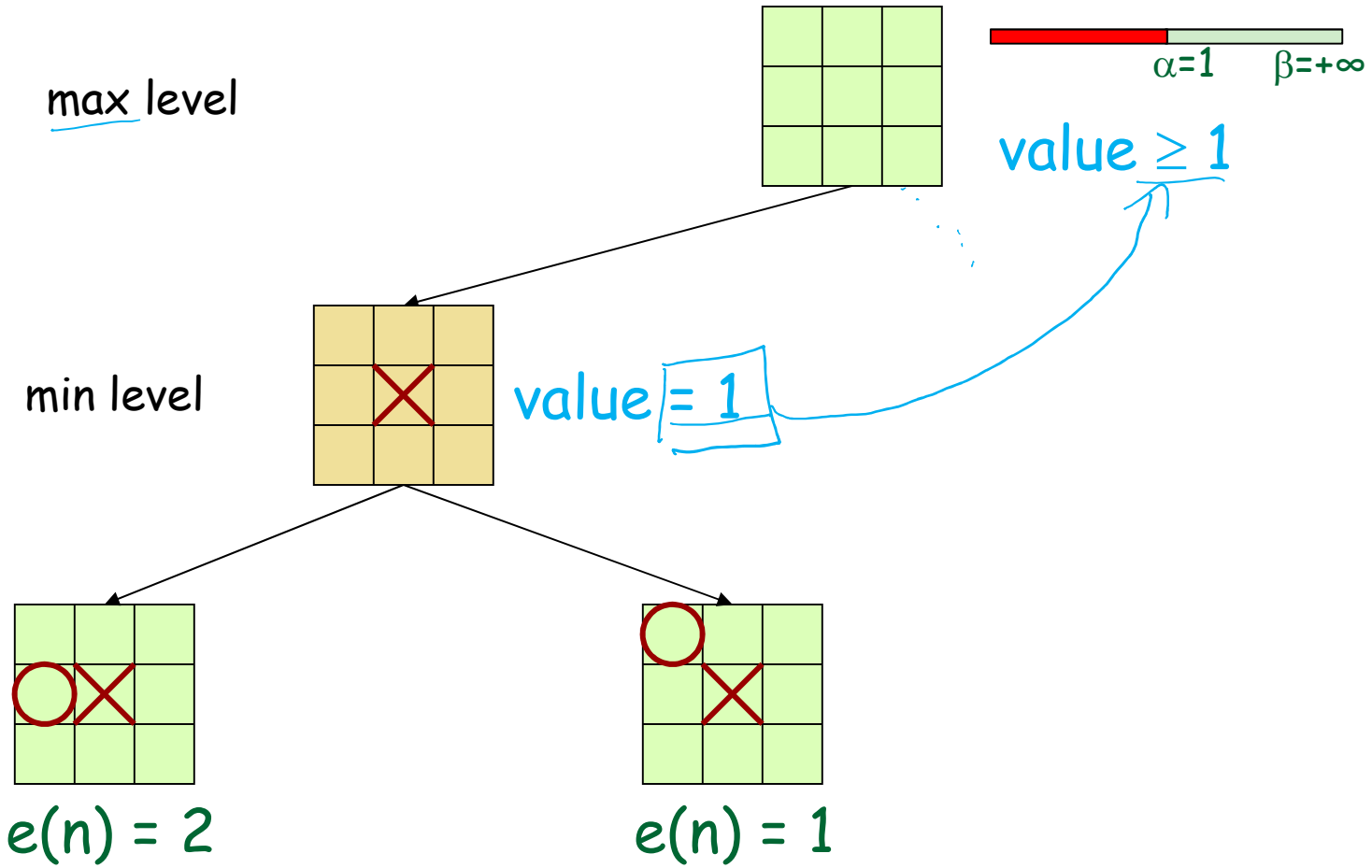
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



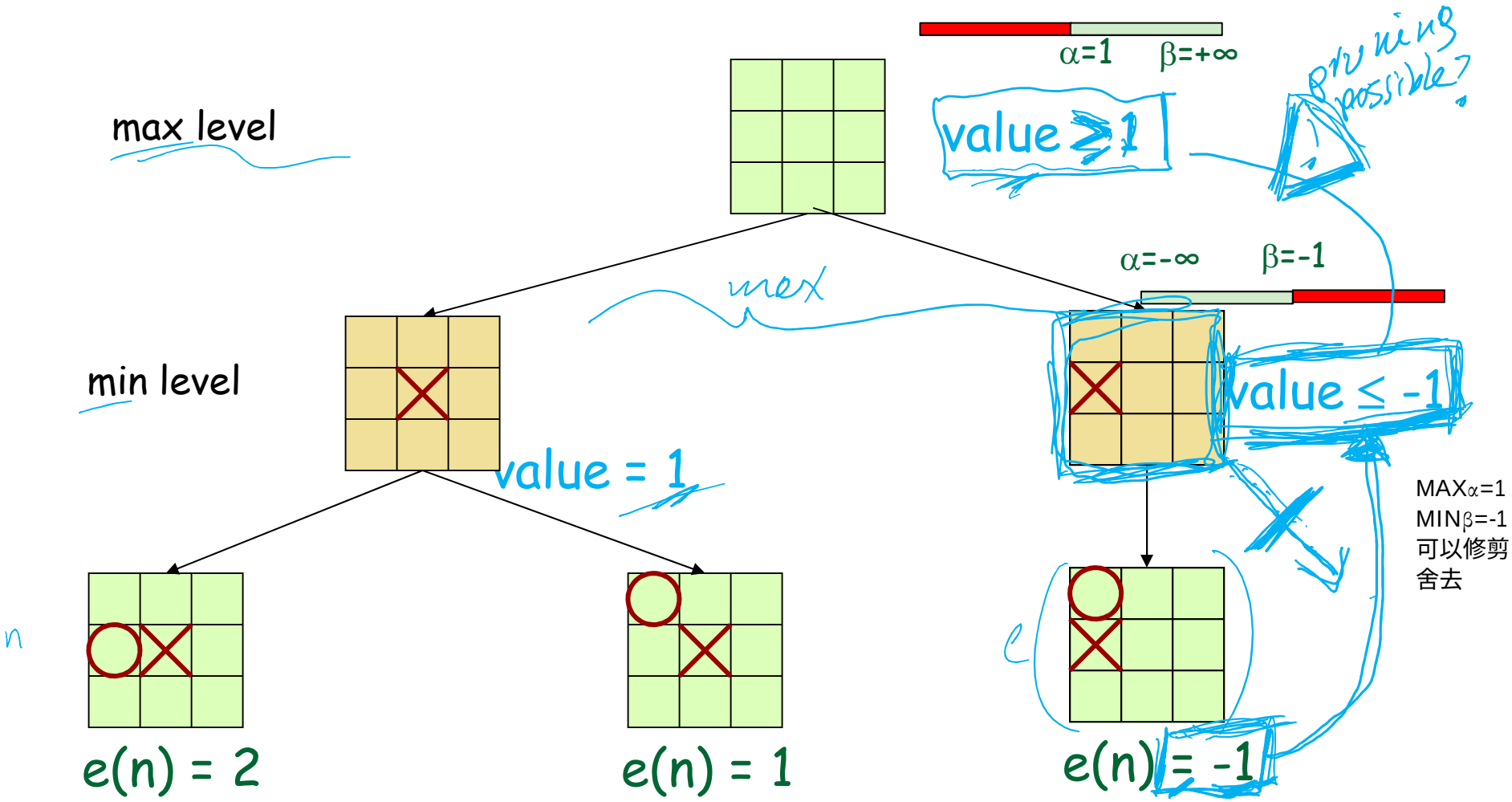
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



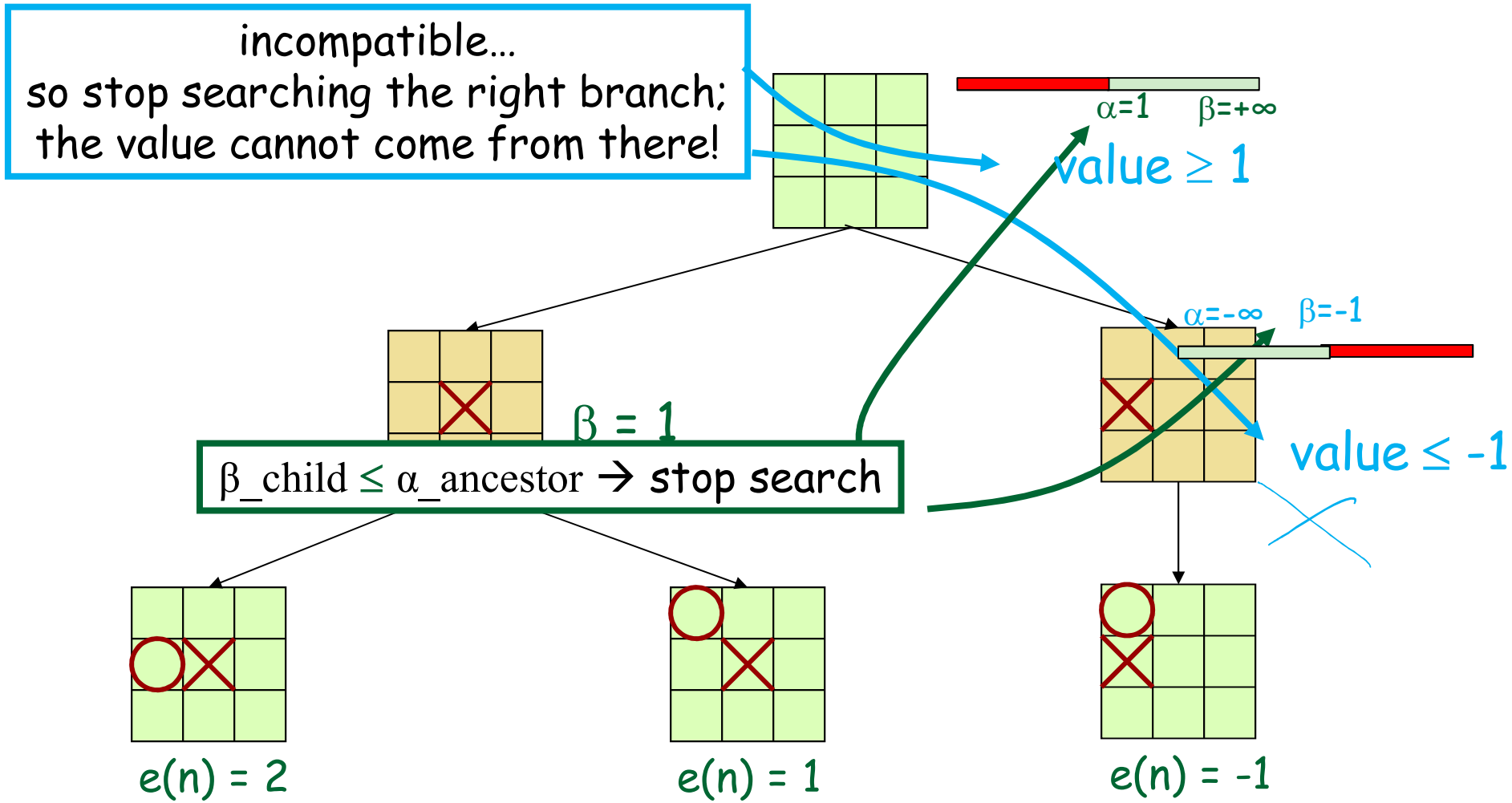
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



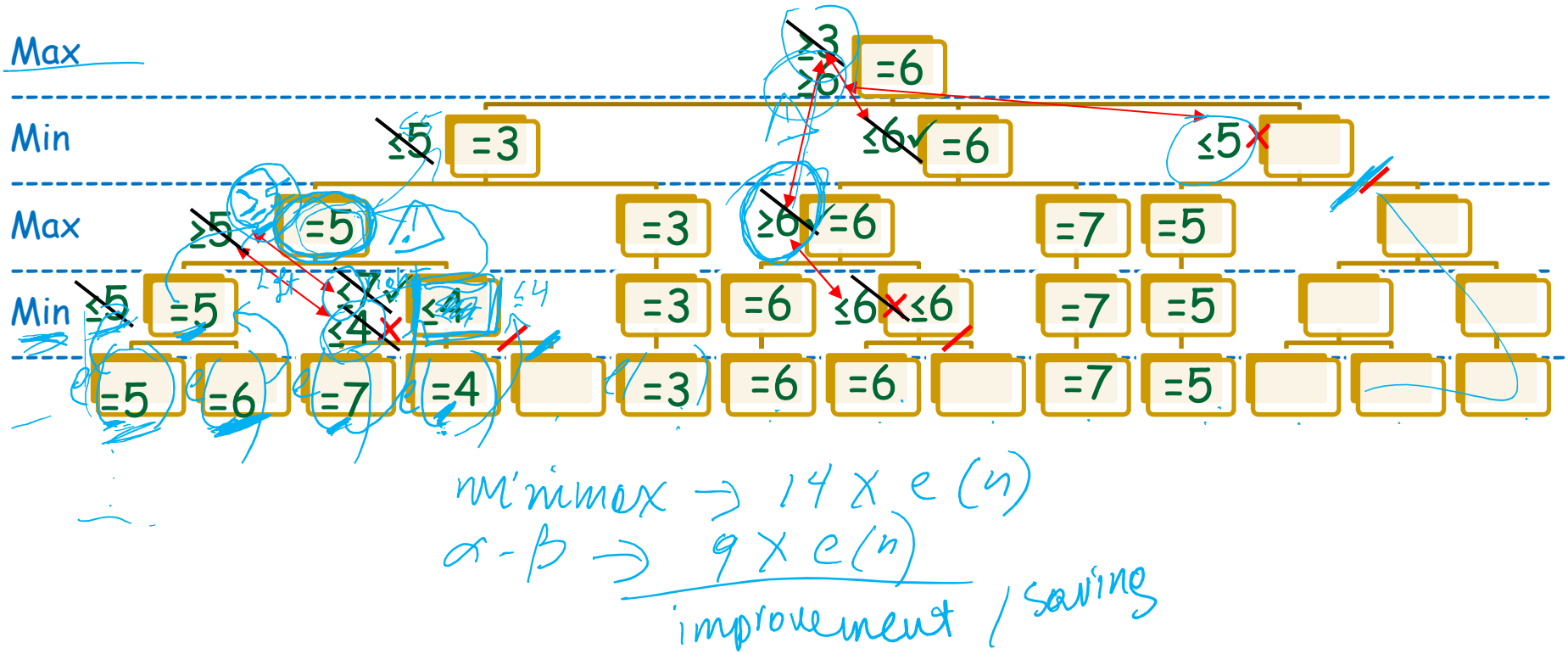
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



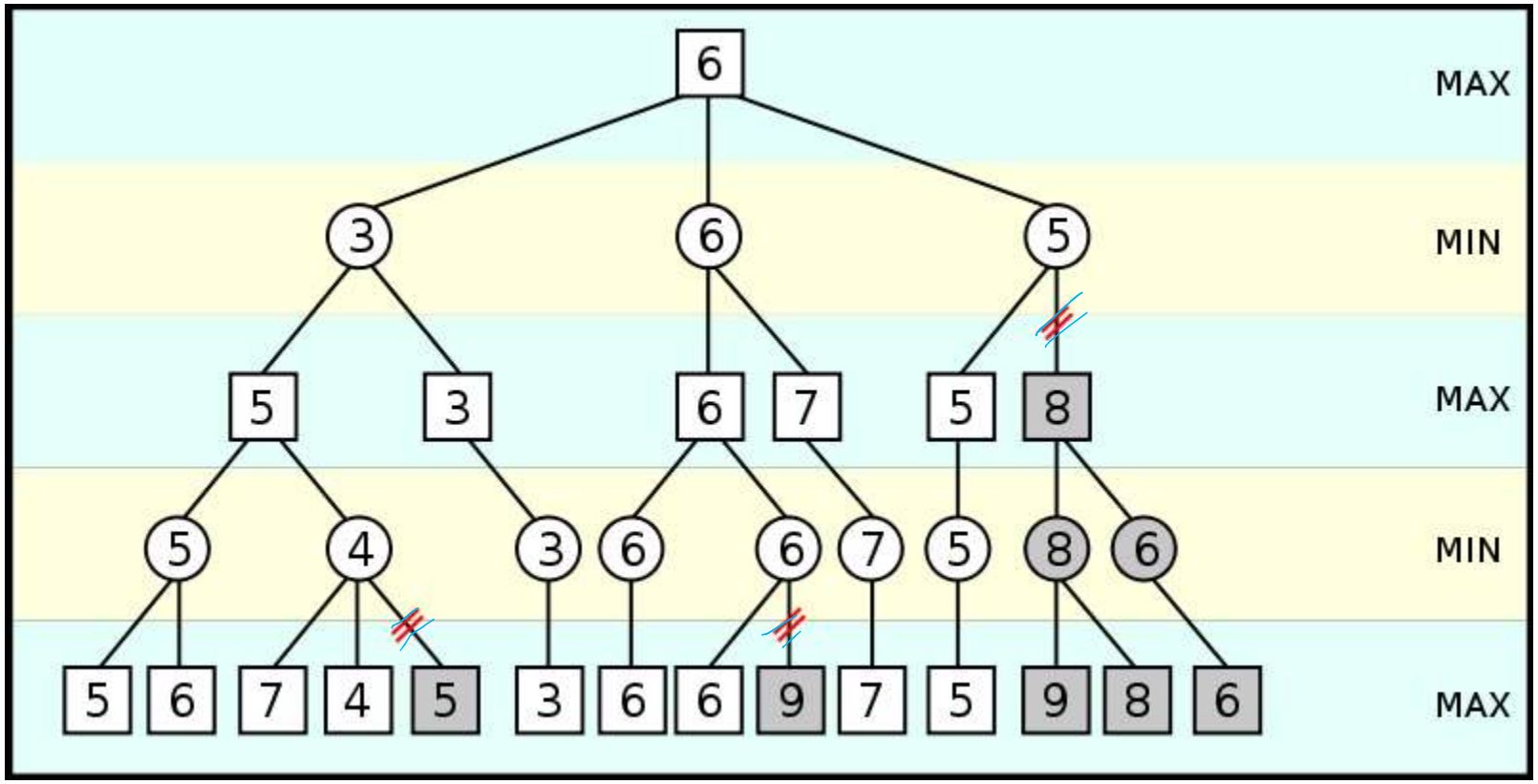
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Alpha-Beta Pruning: Example 2



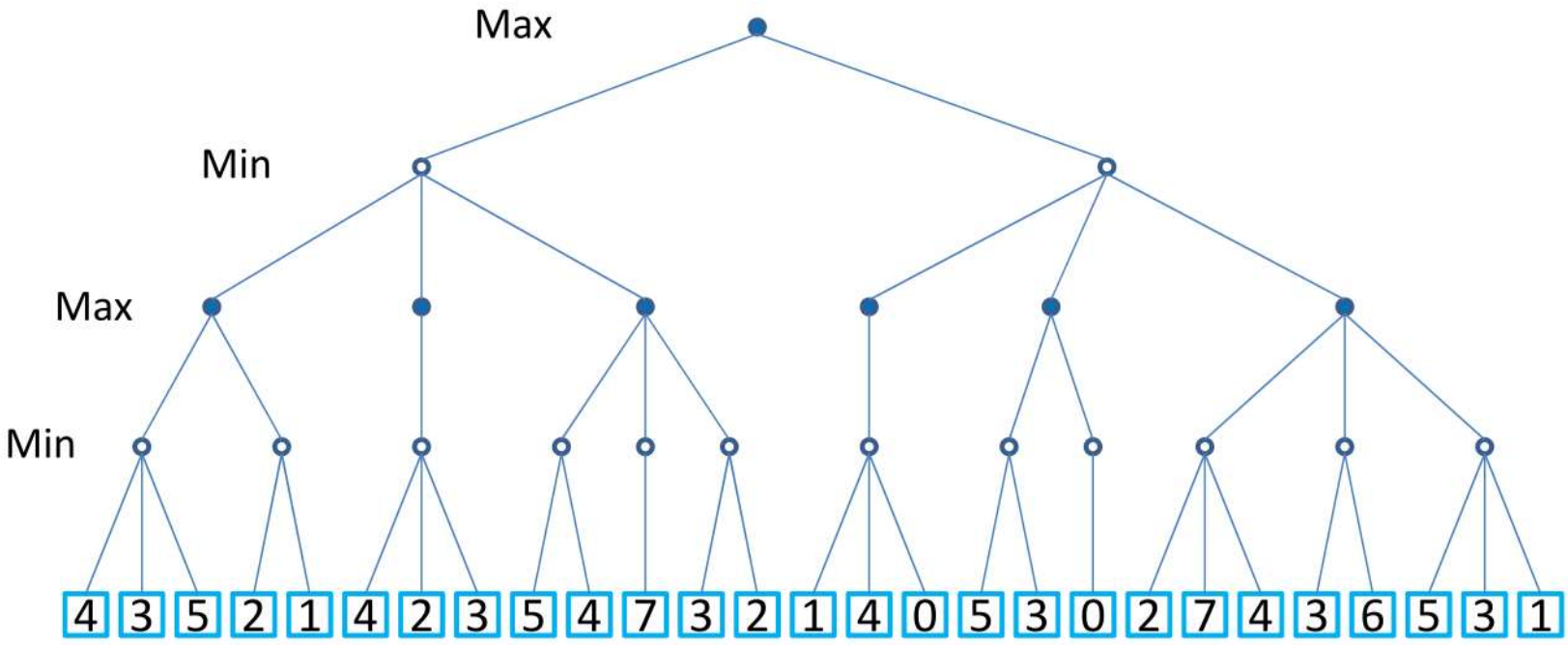
source: http://en.wikipedia.org/wiki/File:AB_pruning.svg

Alpha-Beta Pruning: Example 2

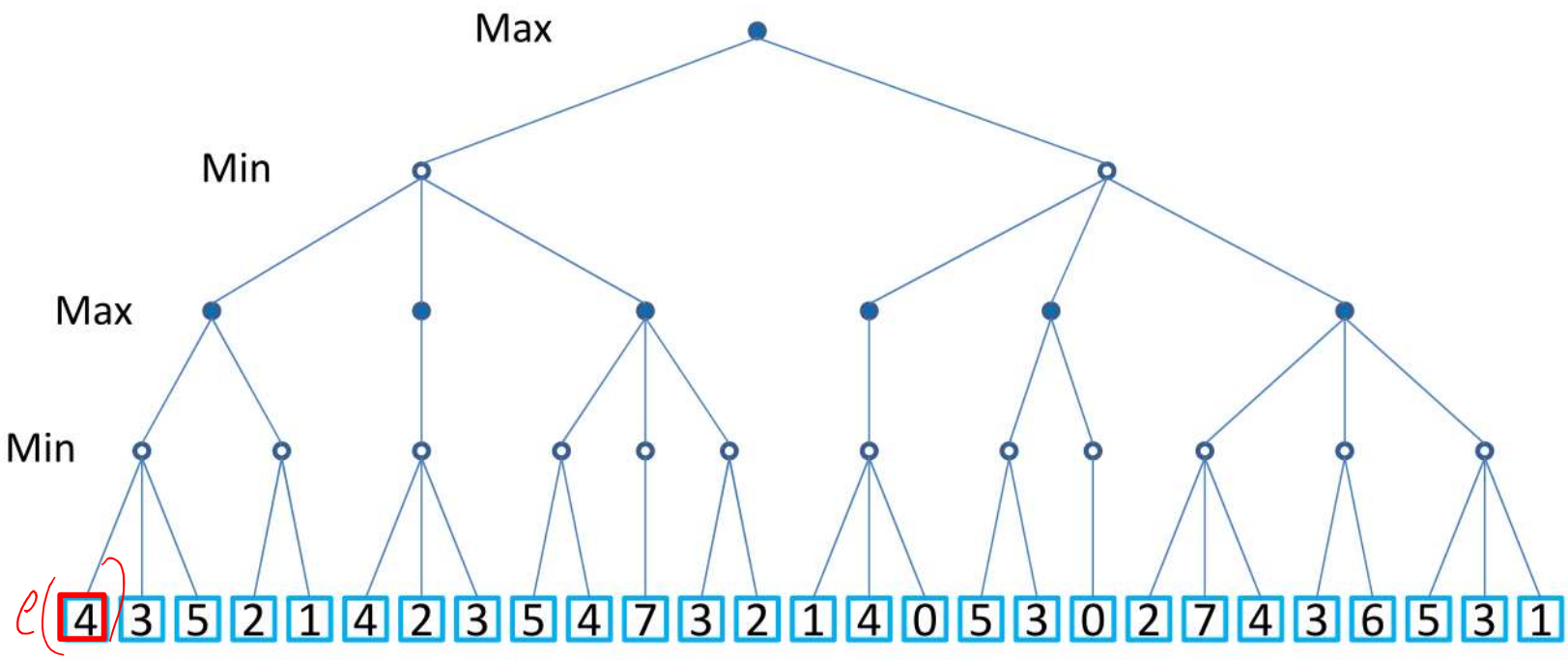


source: http://en.wikipedia.org/wiki/File:AB_pruning.svg

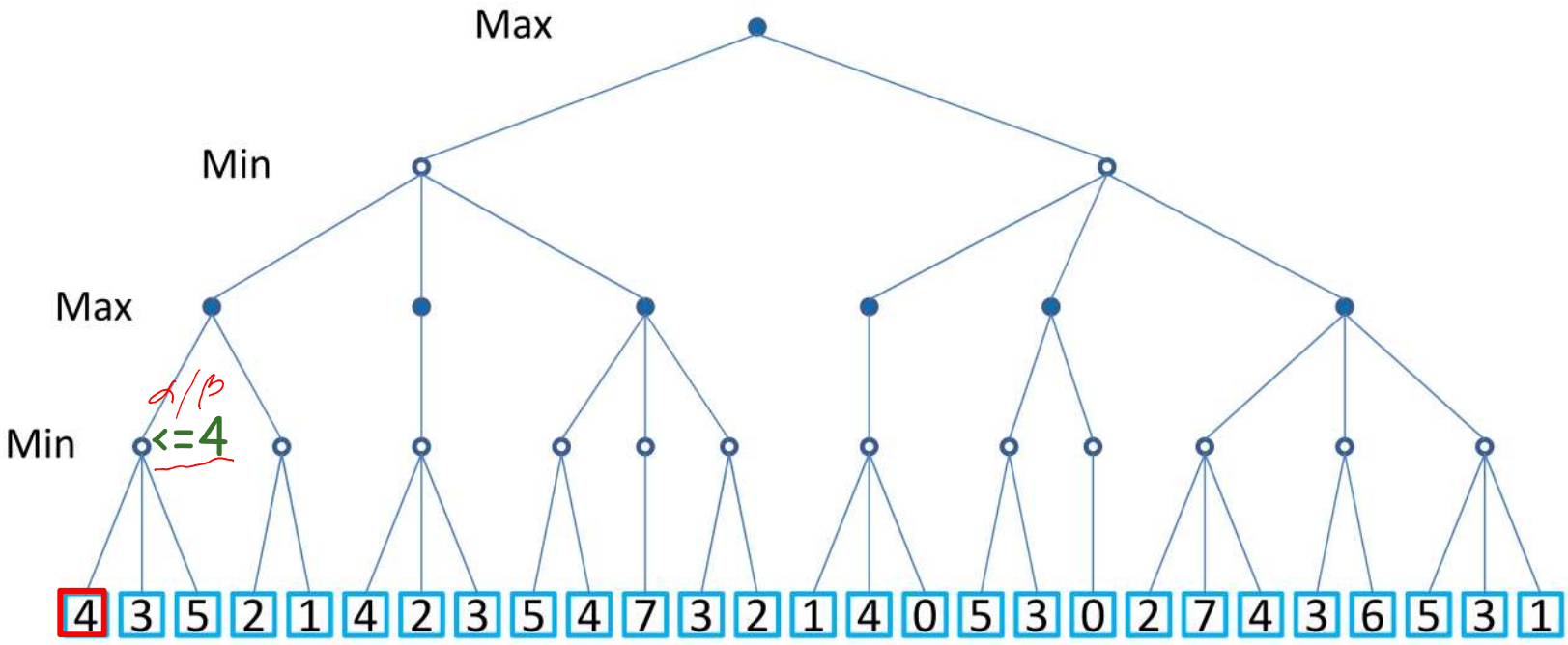
Alpha-Beta Pruning: Example 3



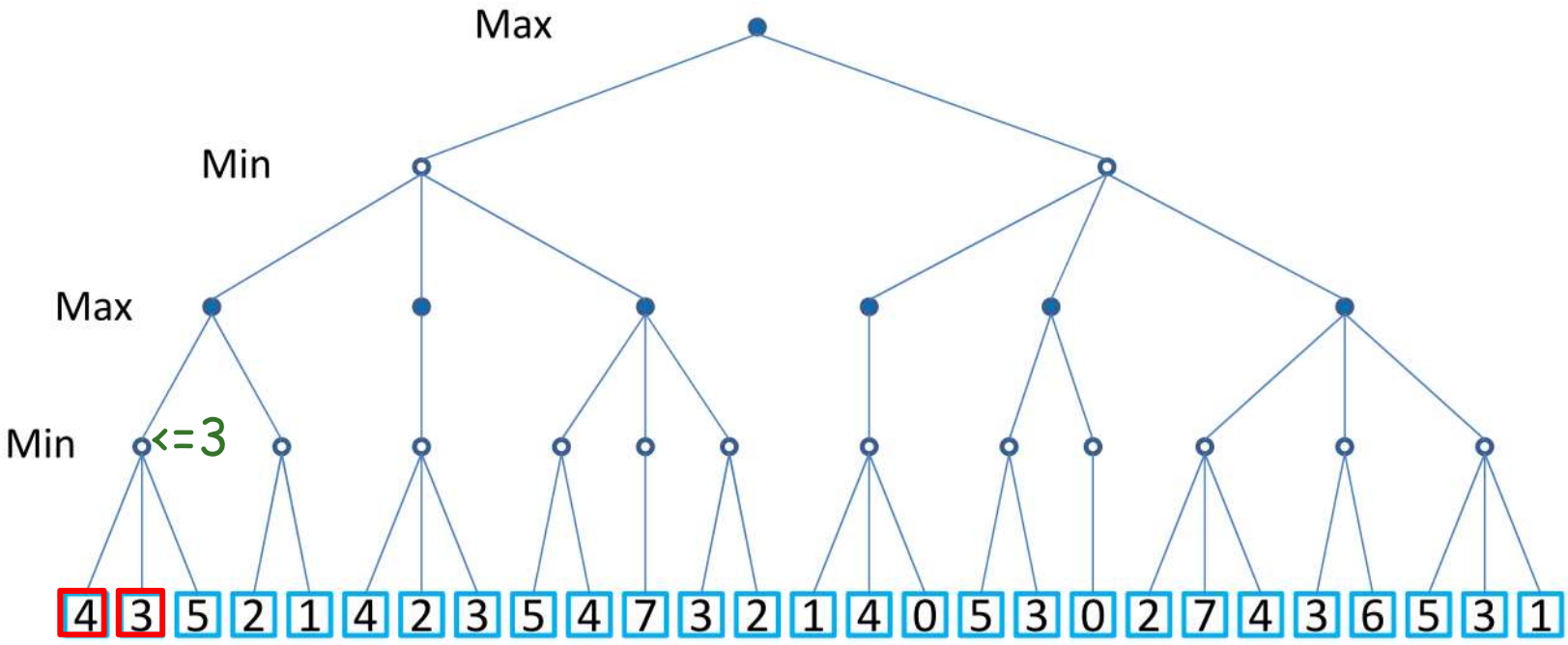
Alpha-Beta Pruning: Example 3



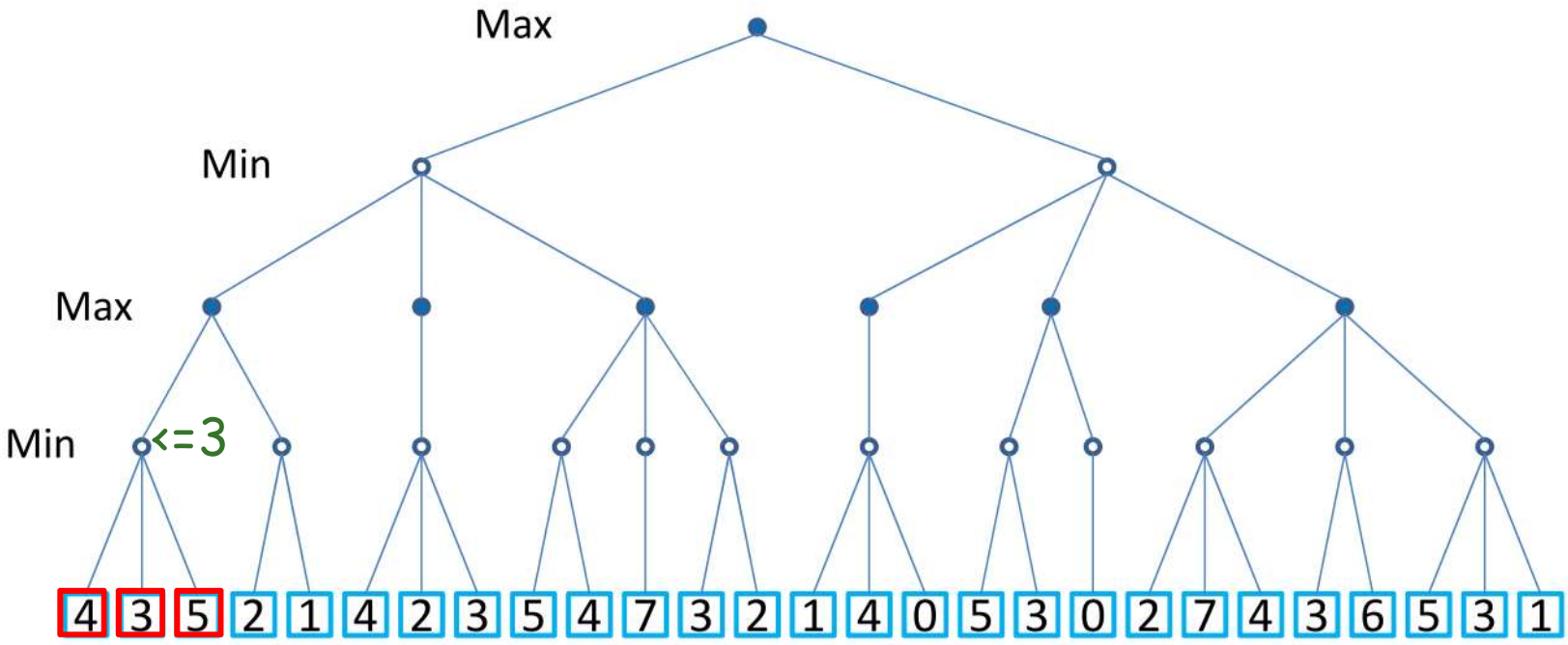
Alpha-Beta Pruning: Example 3



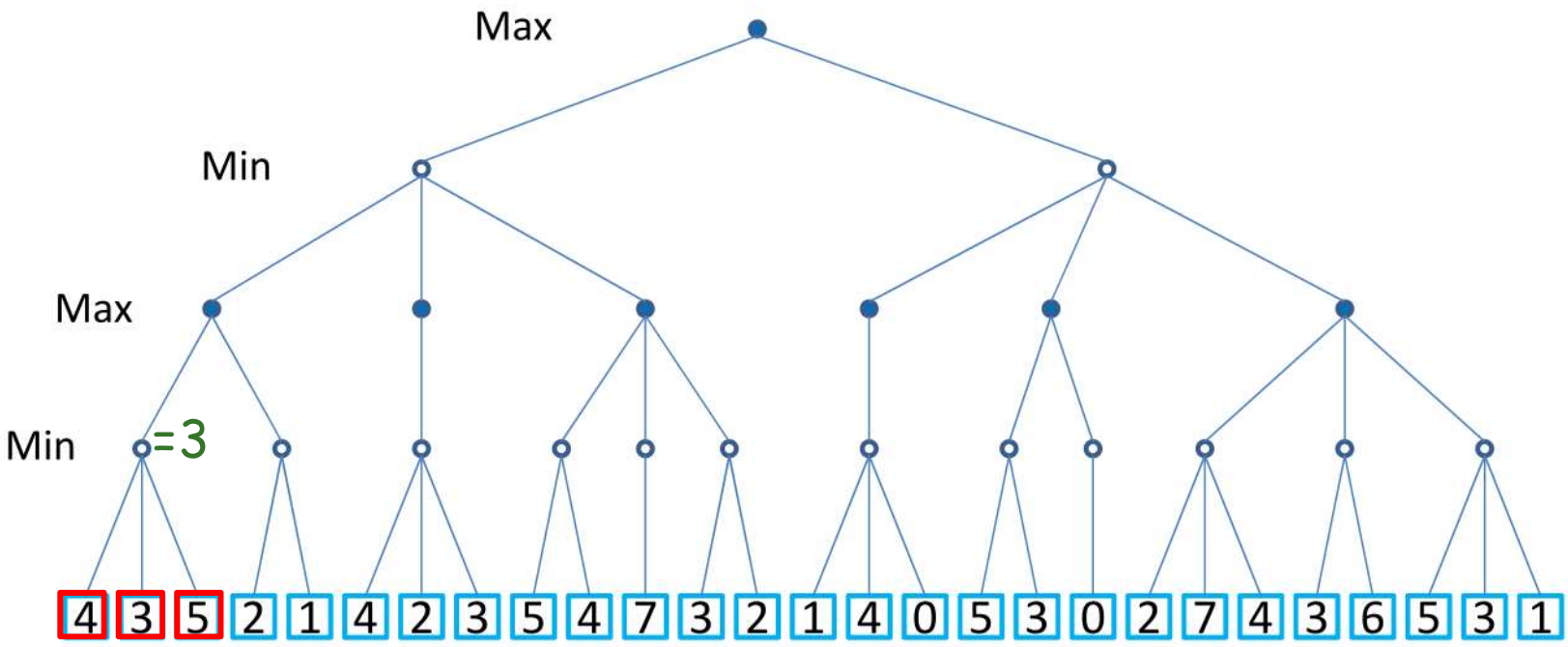
Alpha-Beta Pruning: Example 3



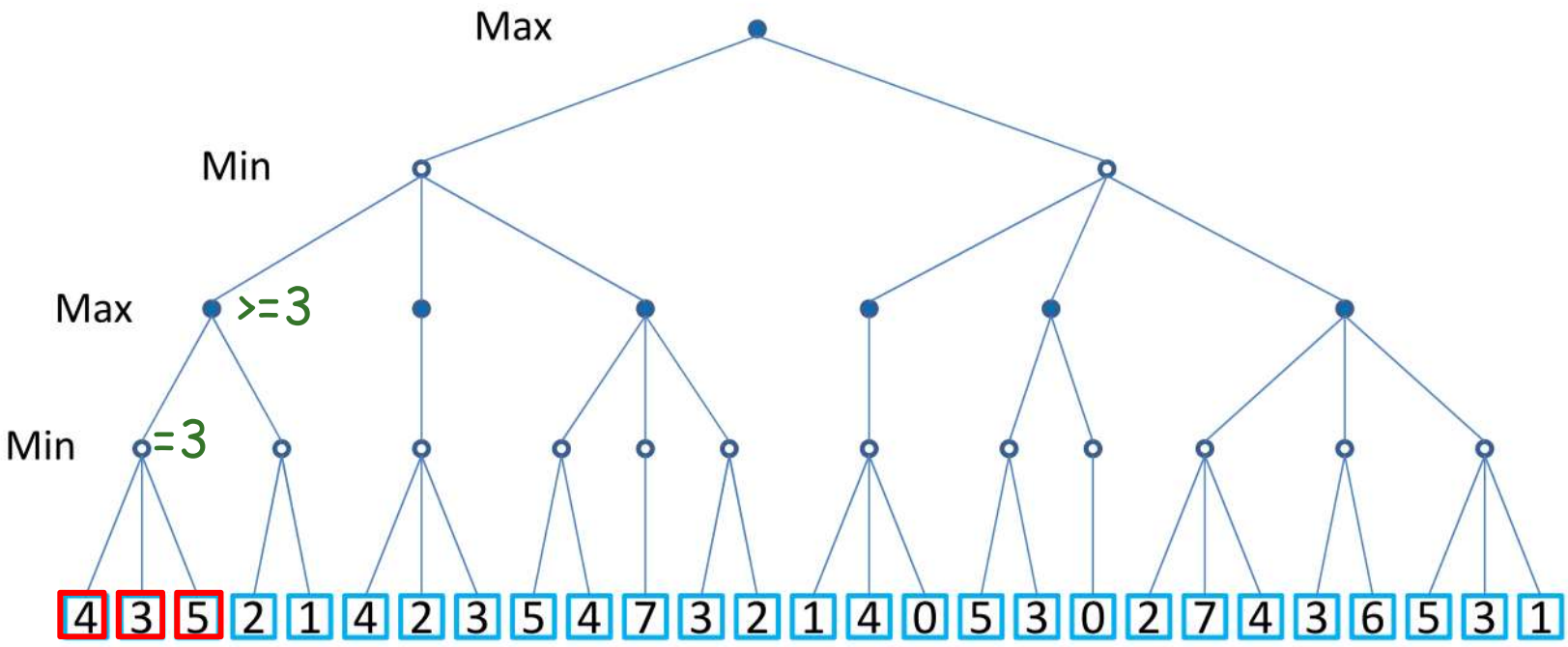
Alpha-Beta Pruning: Example 3



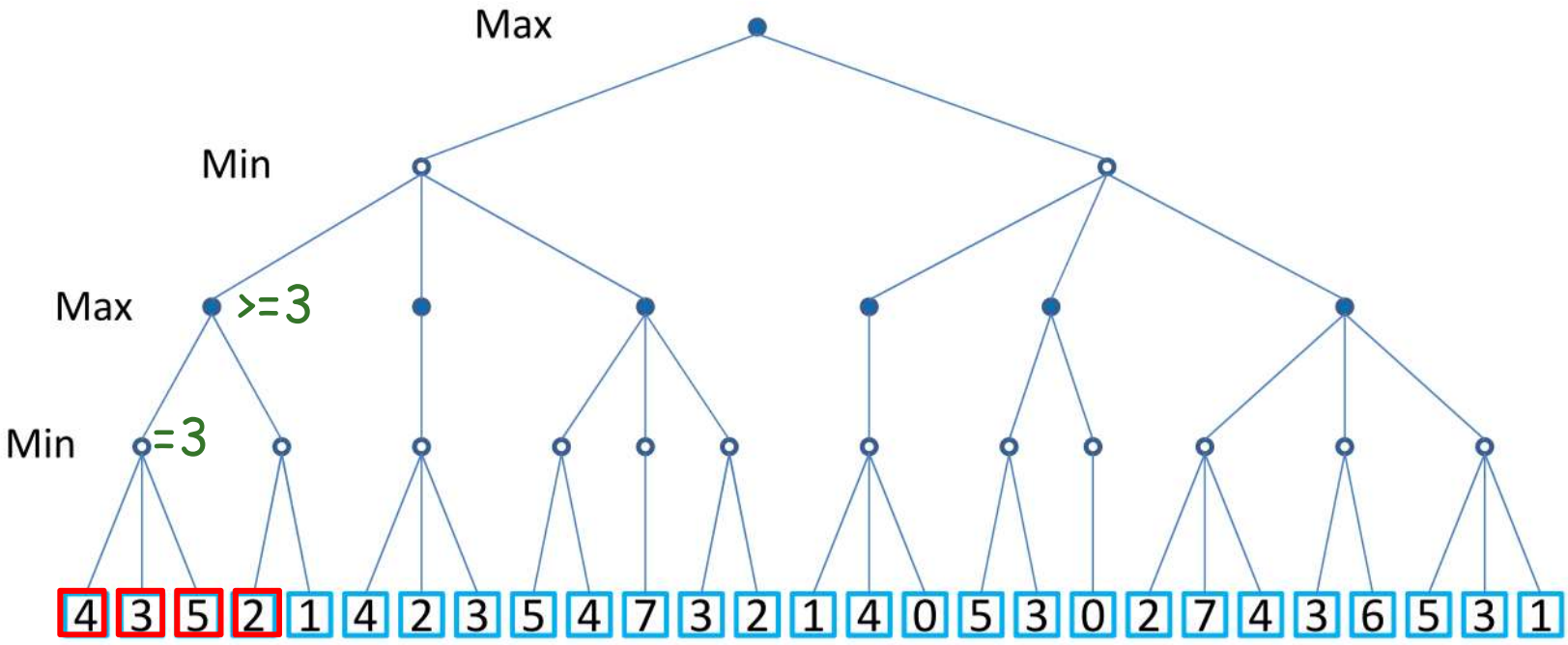
Alpha-Beta Pruning: Example 3



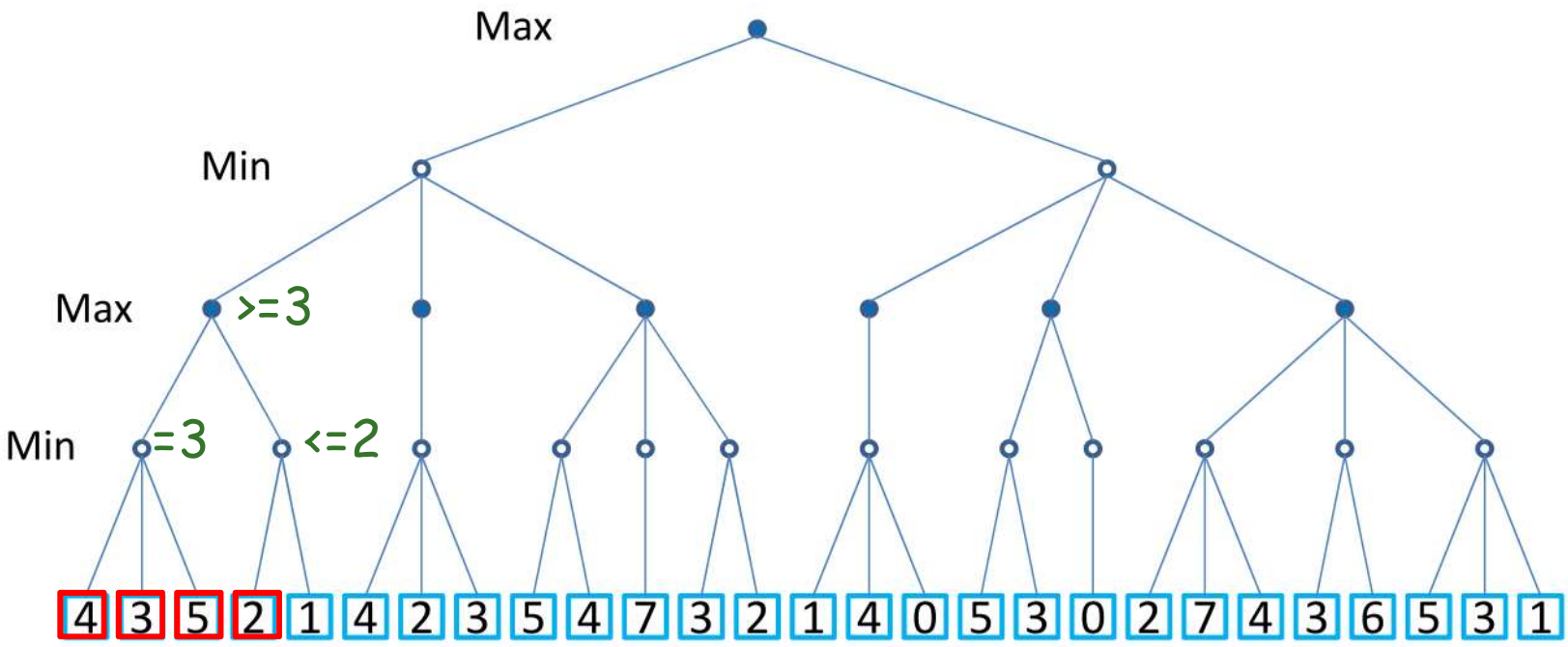
Alpha-Beta Pruning: Example 3



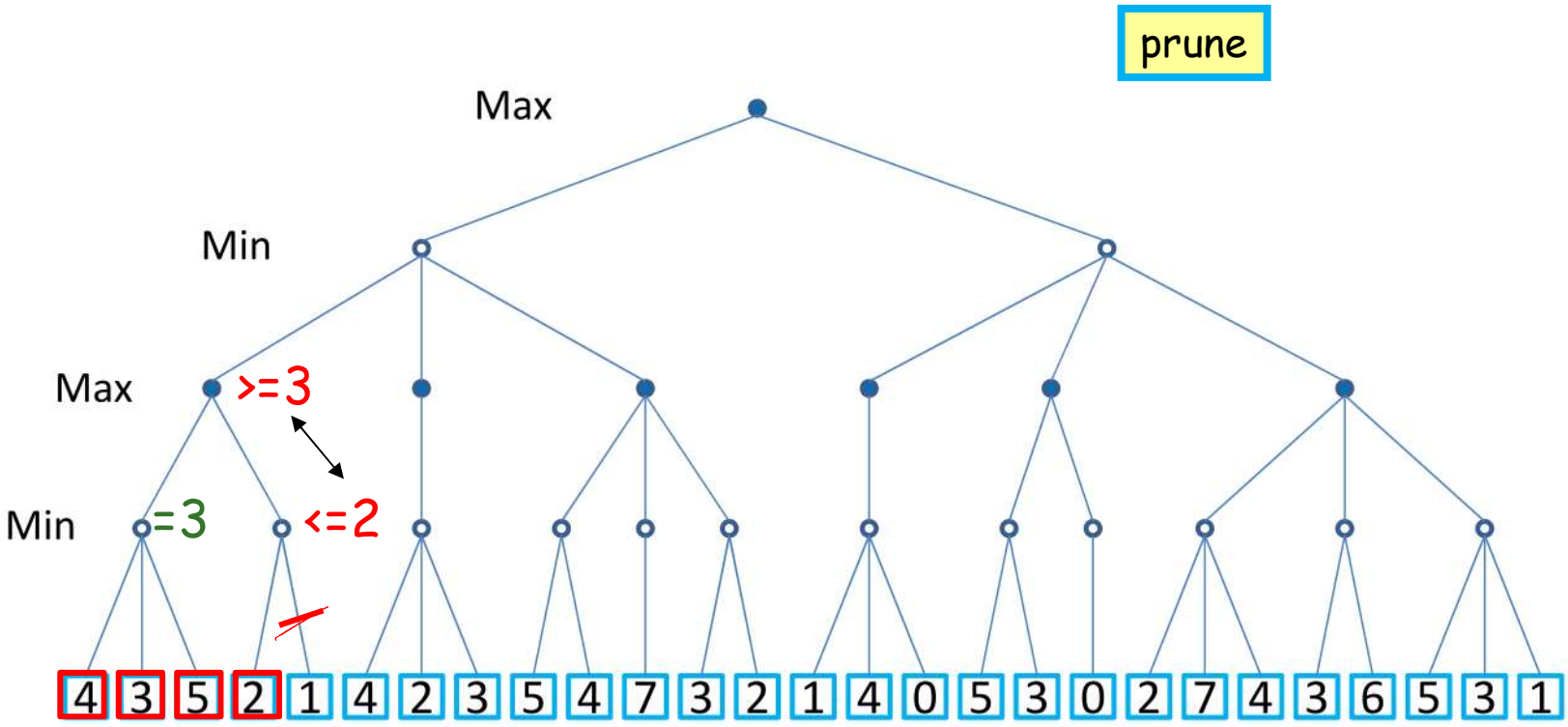
Alpha-Beta Pruning: Example 3



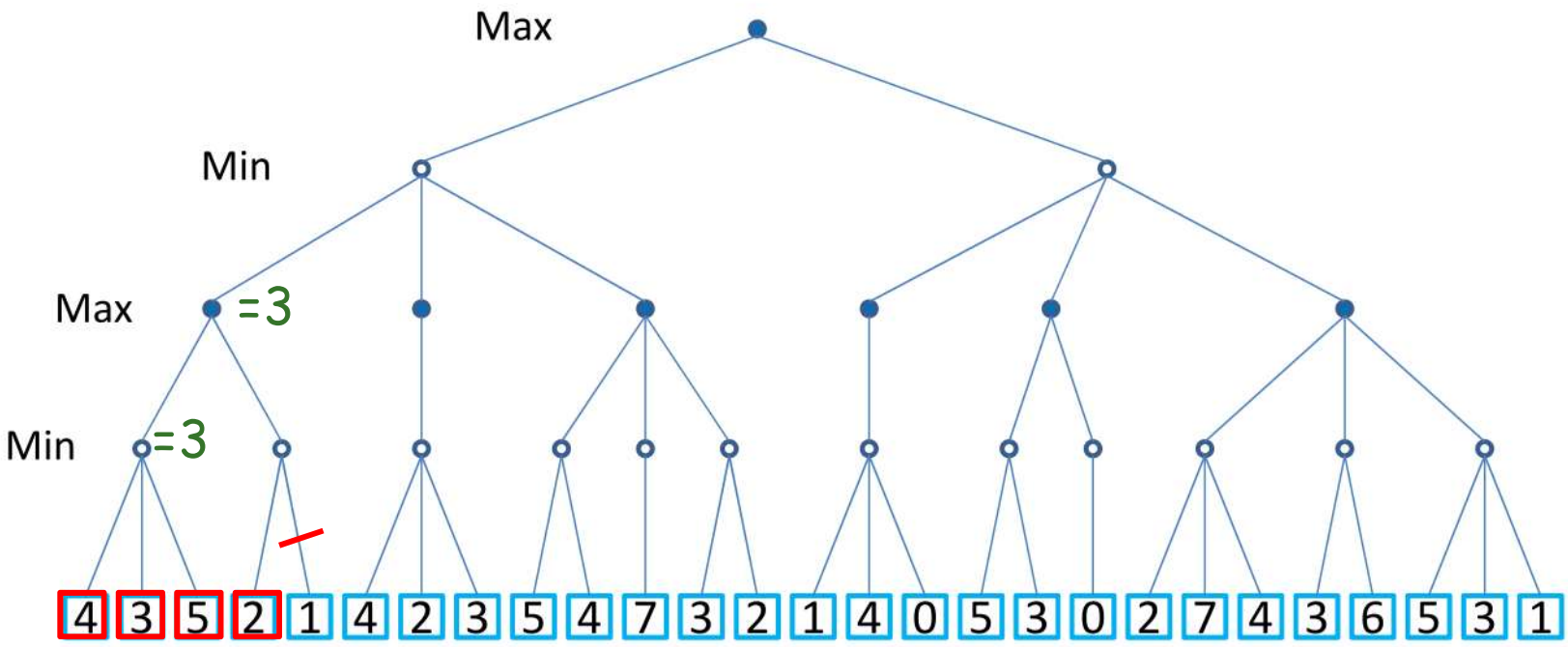
Alpha-Beta Pruning: Example 3



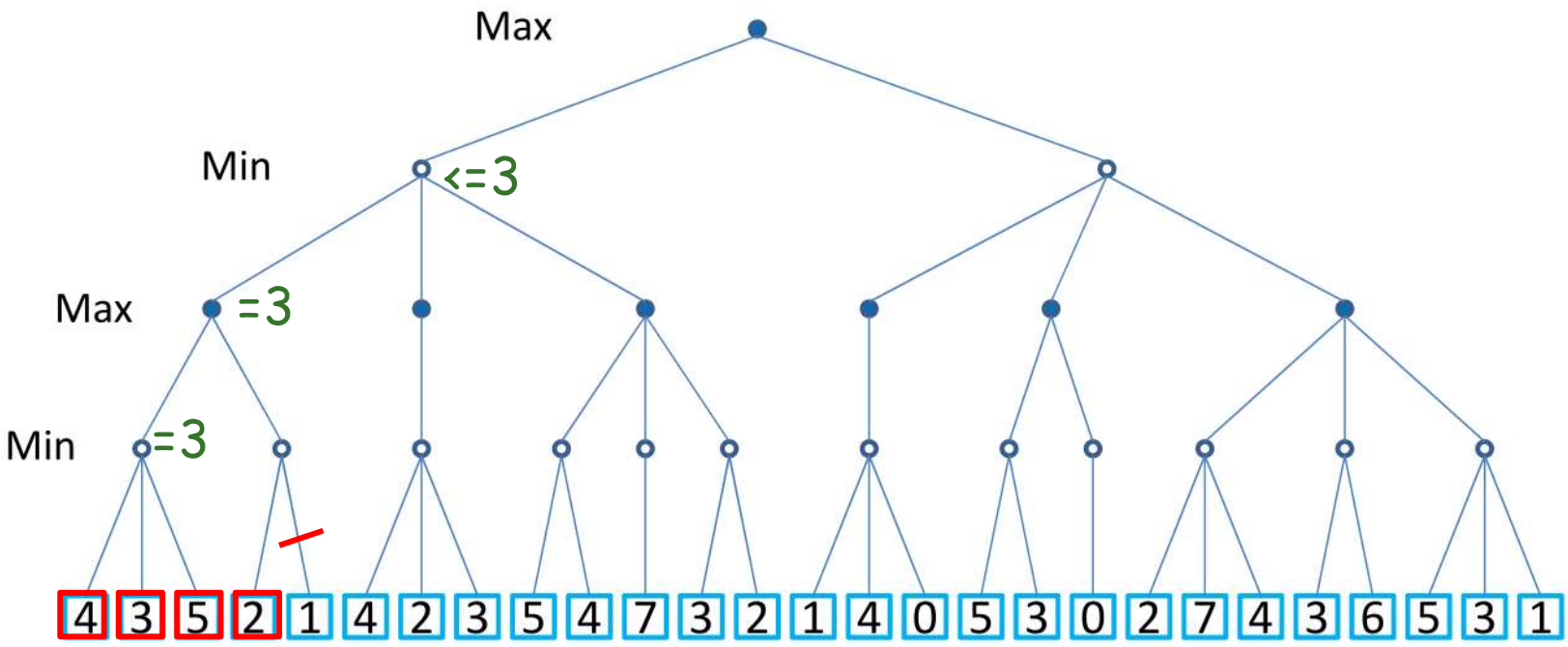
Alpha-Beta Pruning: Example 3



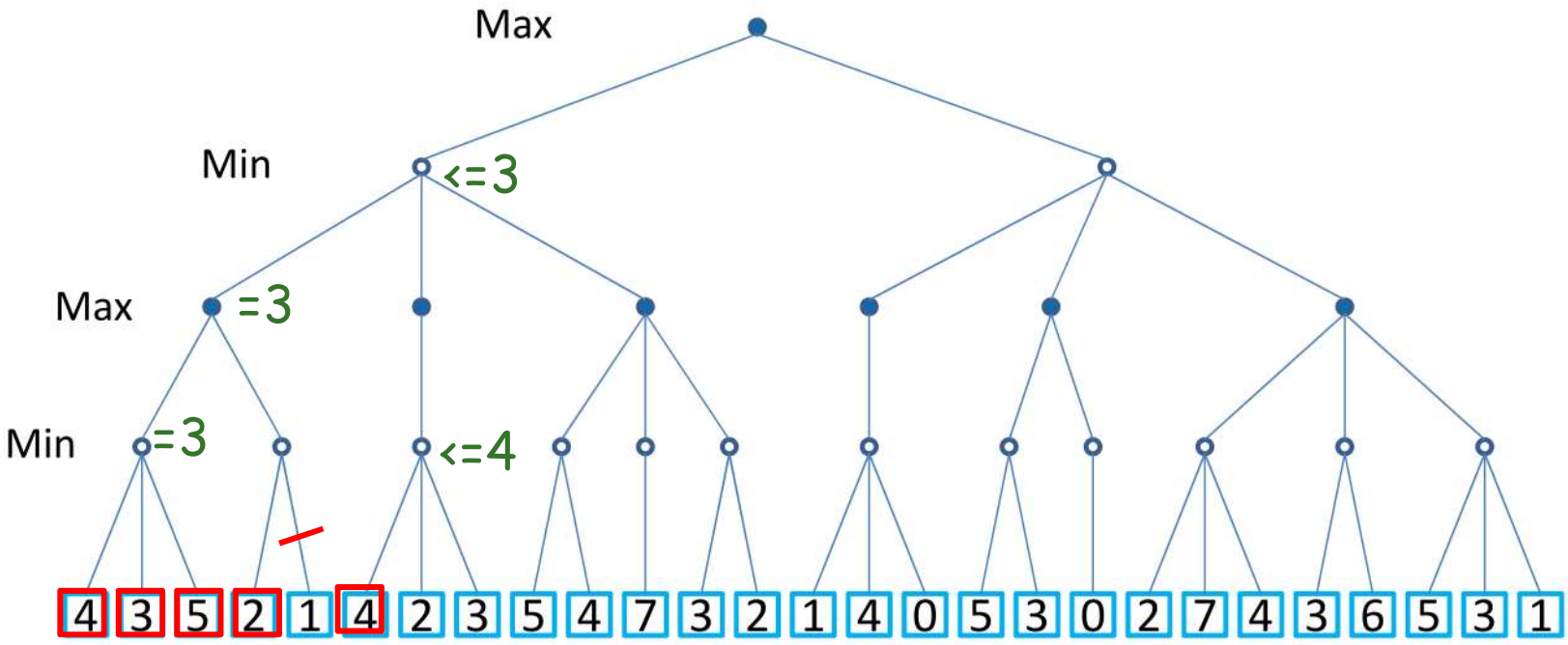
Alpha-Beta Pruning: Example 3



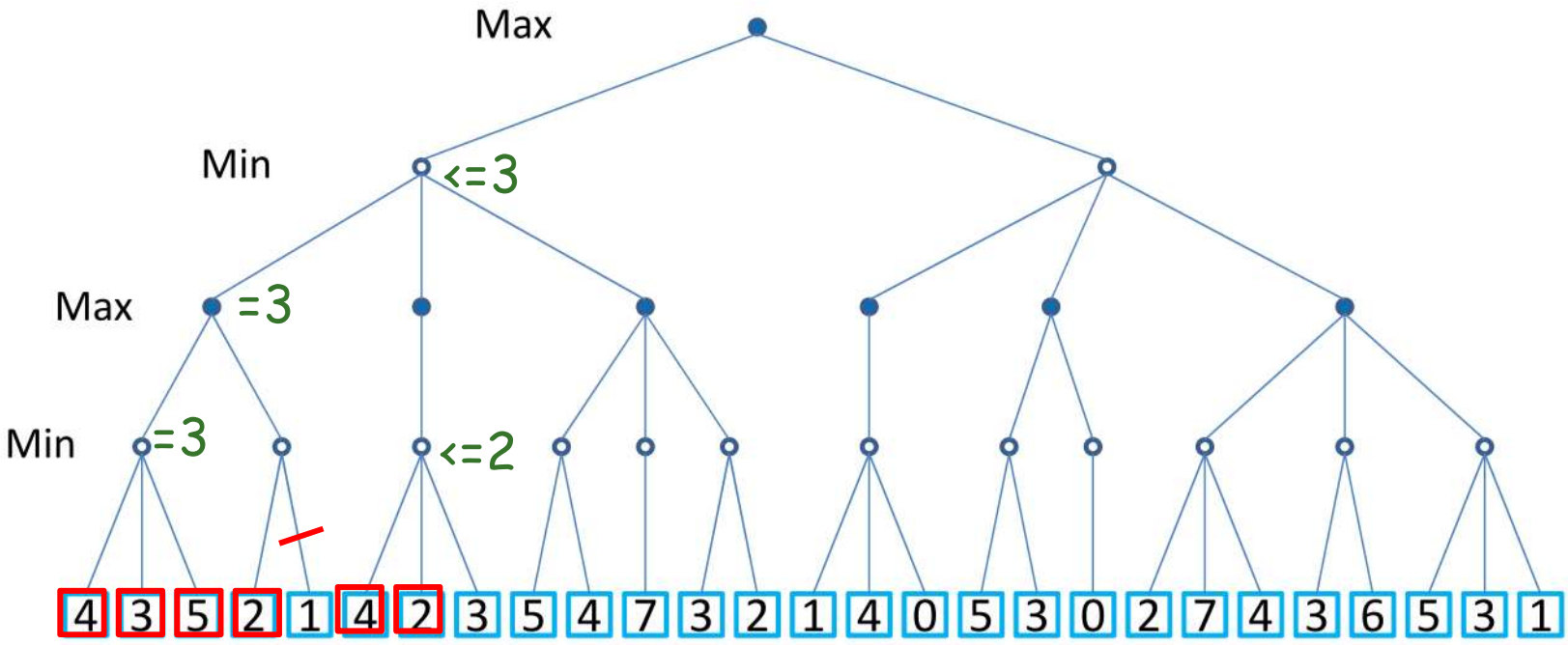
Alpha-Beta Pruning: Example 3



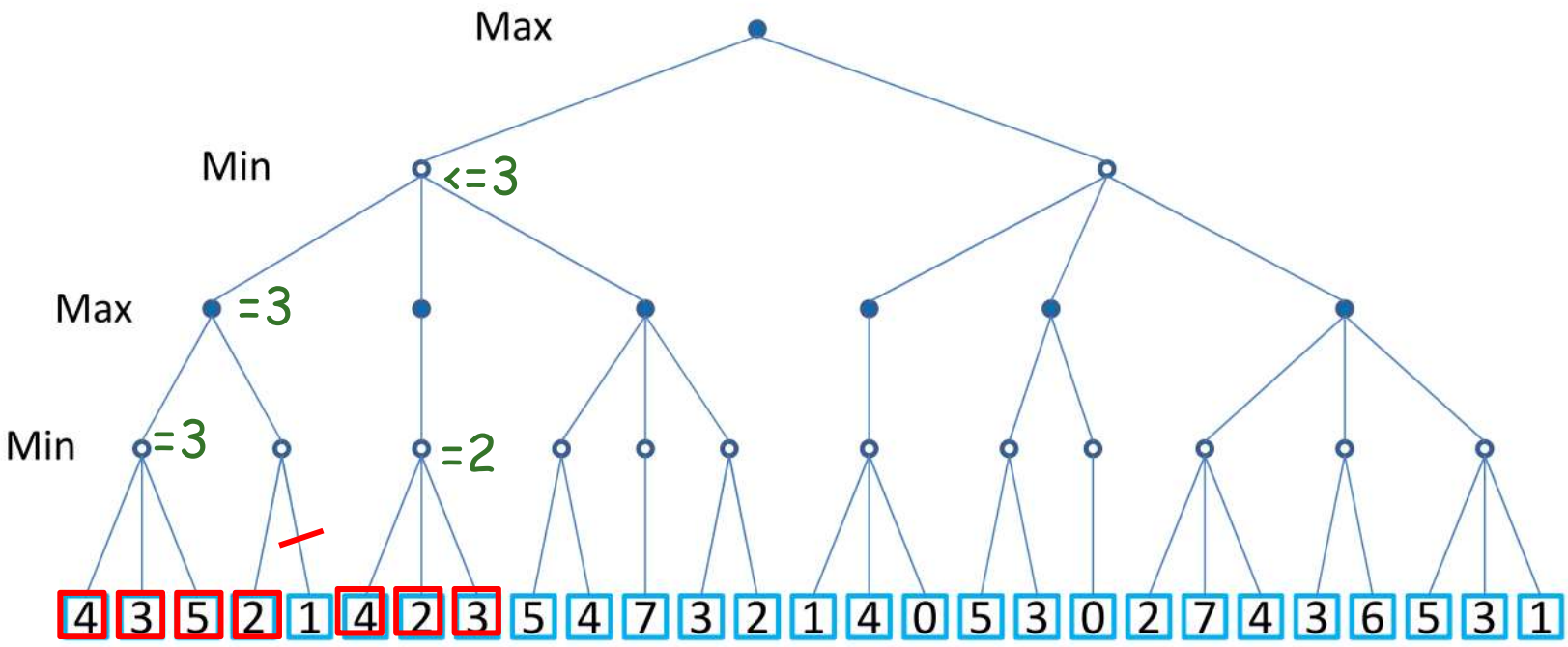
Alpha-Beta Pruning: Example 3



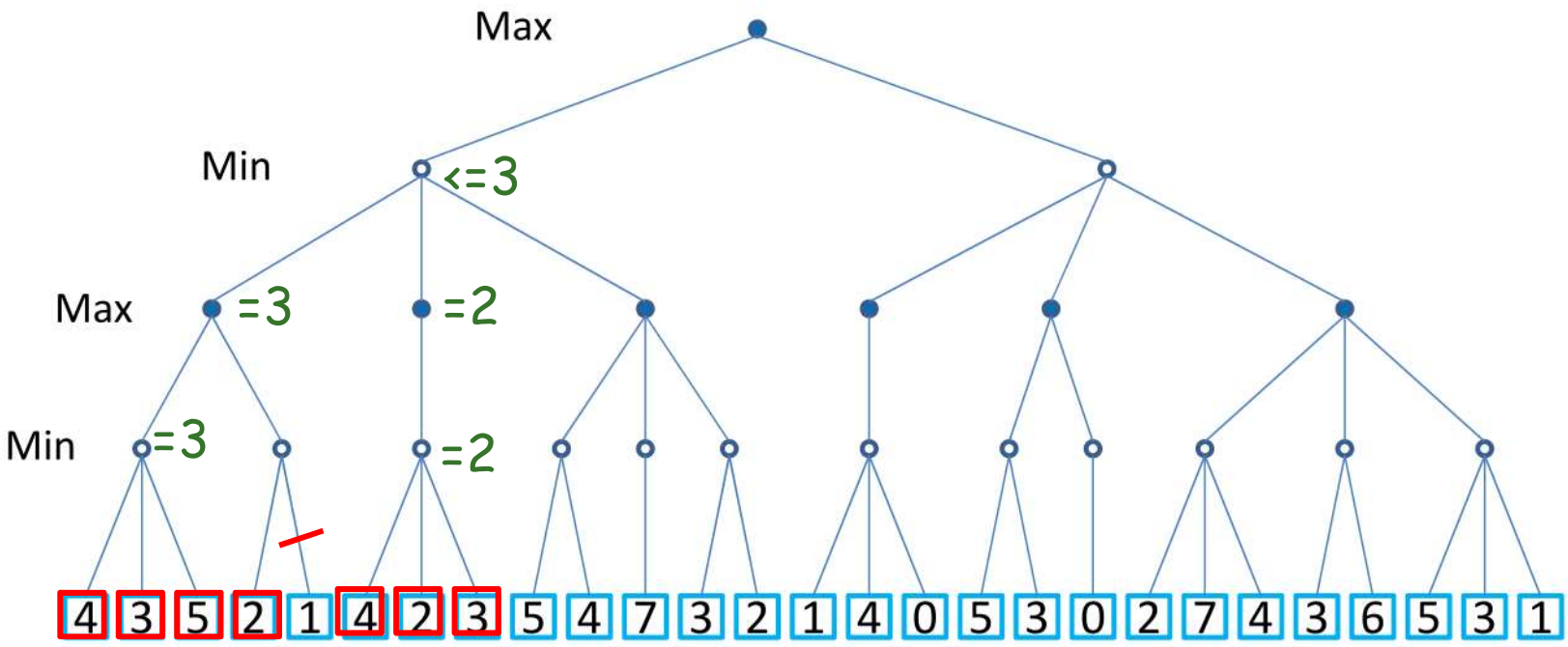
Alpha-Beta Pruning: Example 3



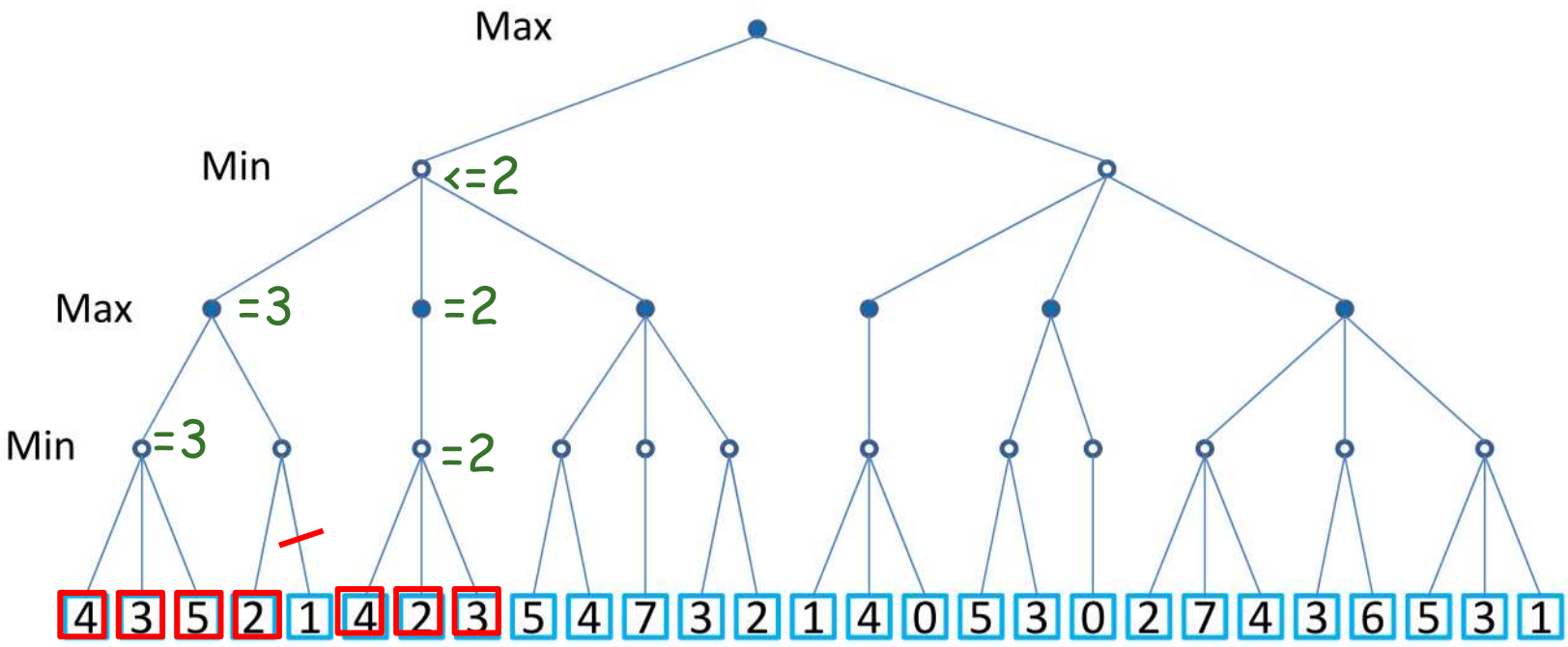
Alpha-Beta Pruning: Example 3



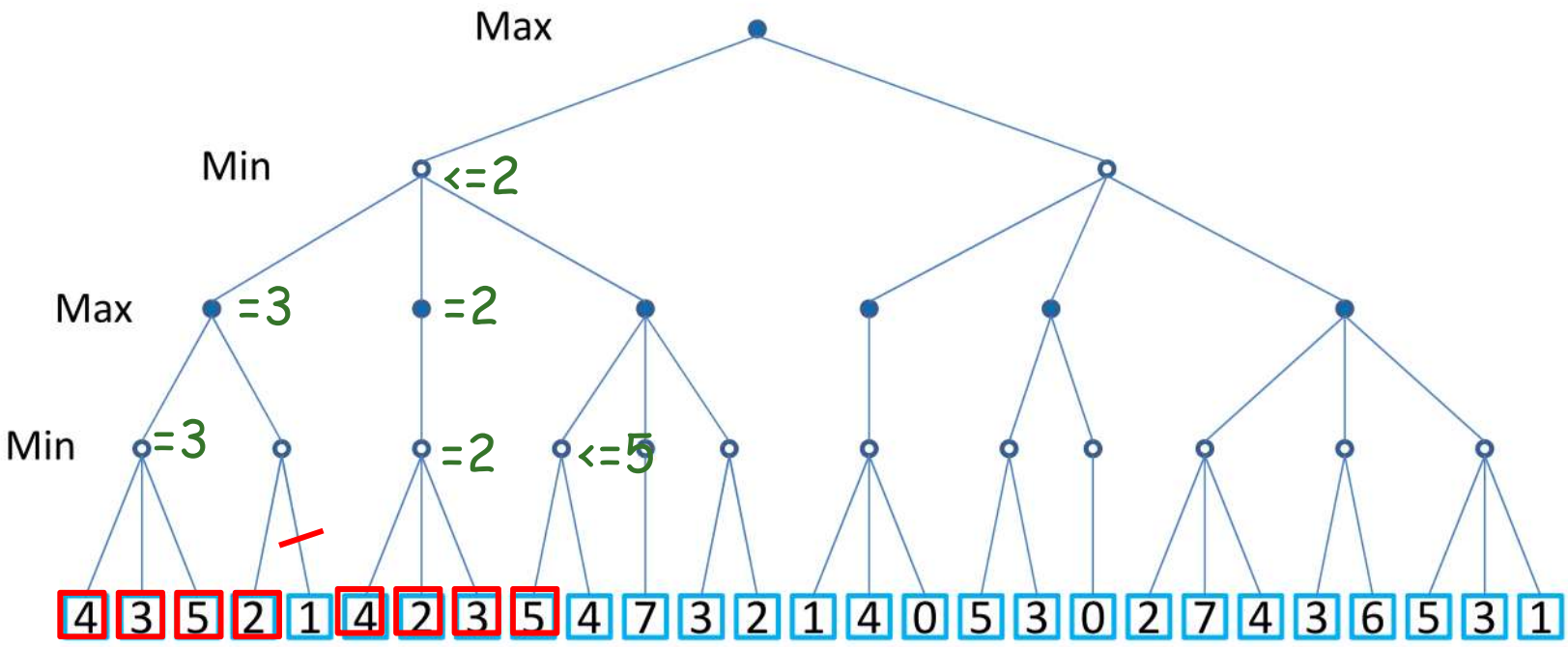
Alpha-Beta Pruning: Example 3



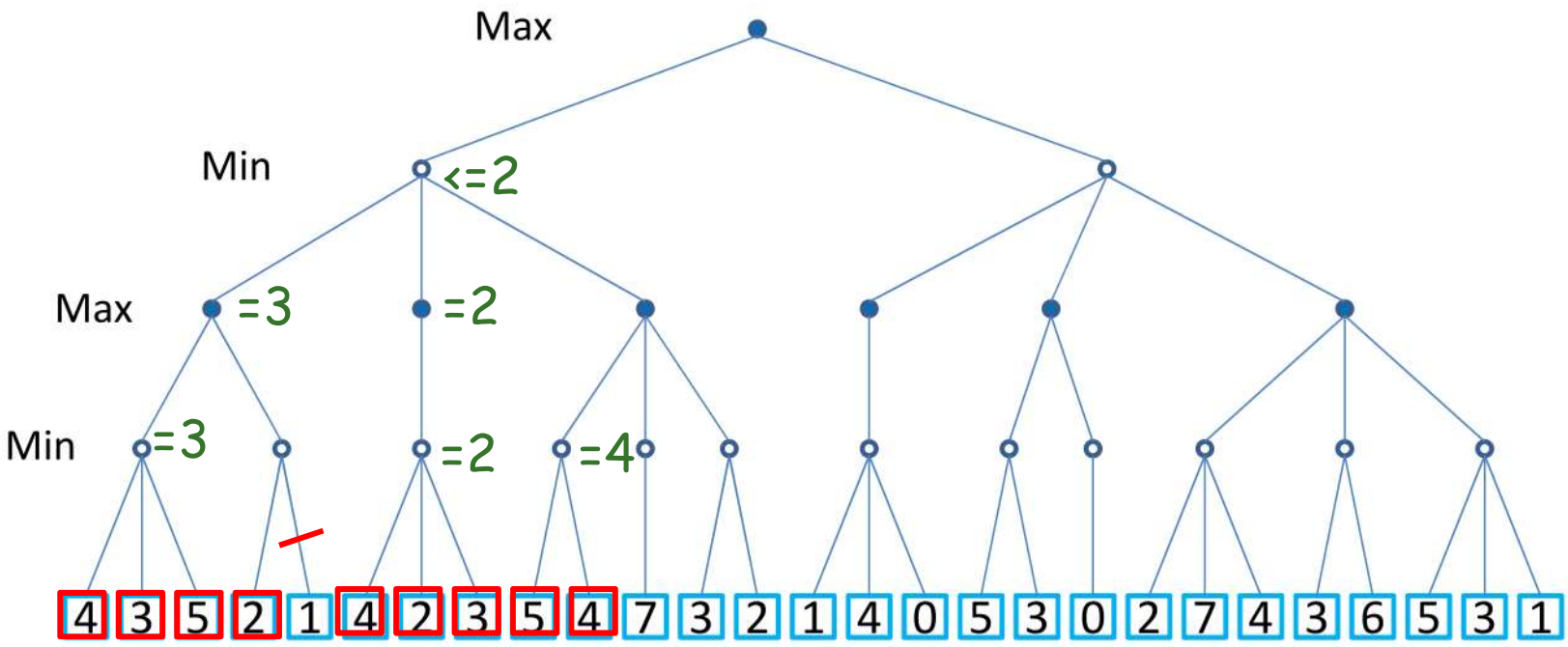
Alpha-Beta Pruning: Example 3



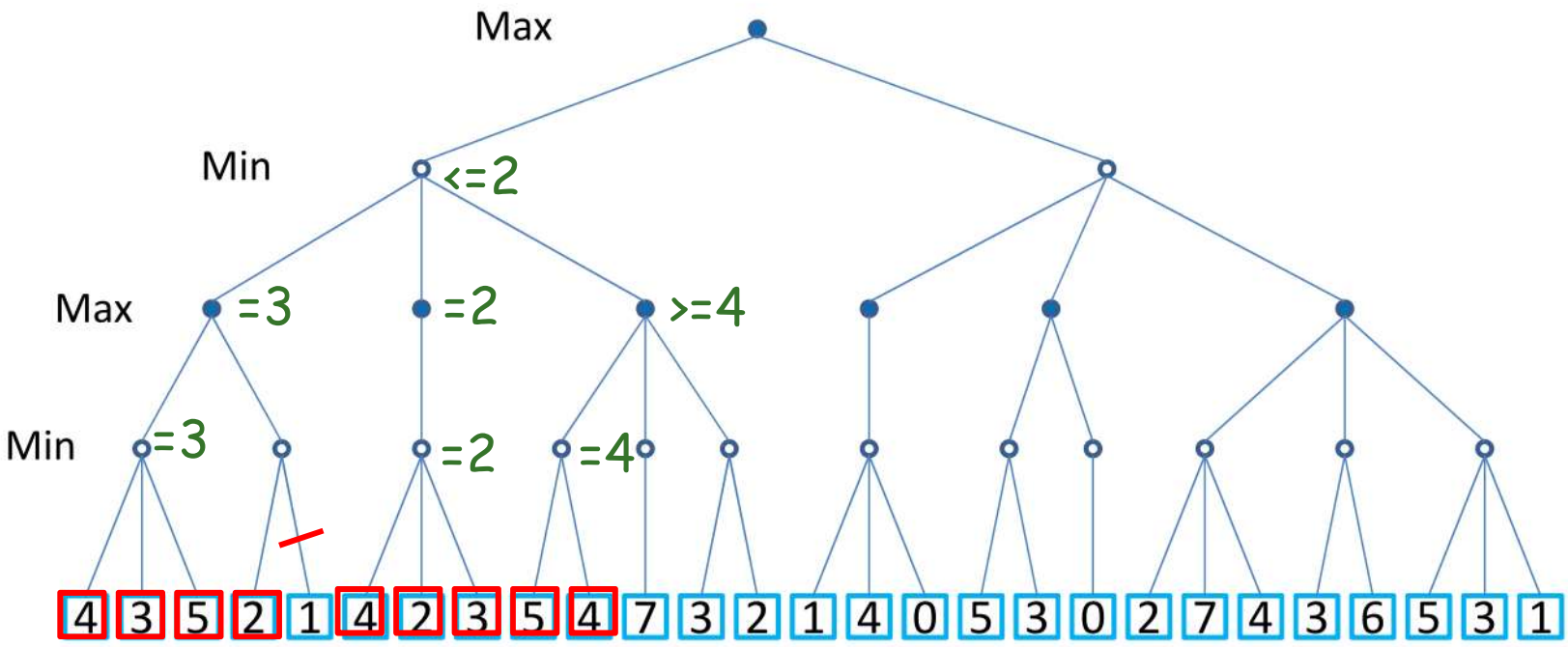
Alpha-Beta Pruning: Example 3



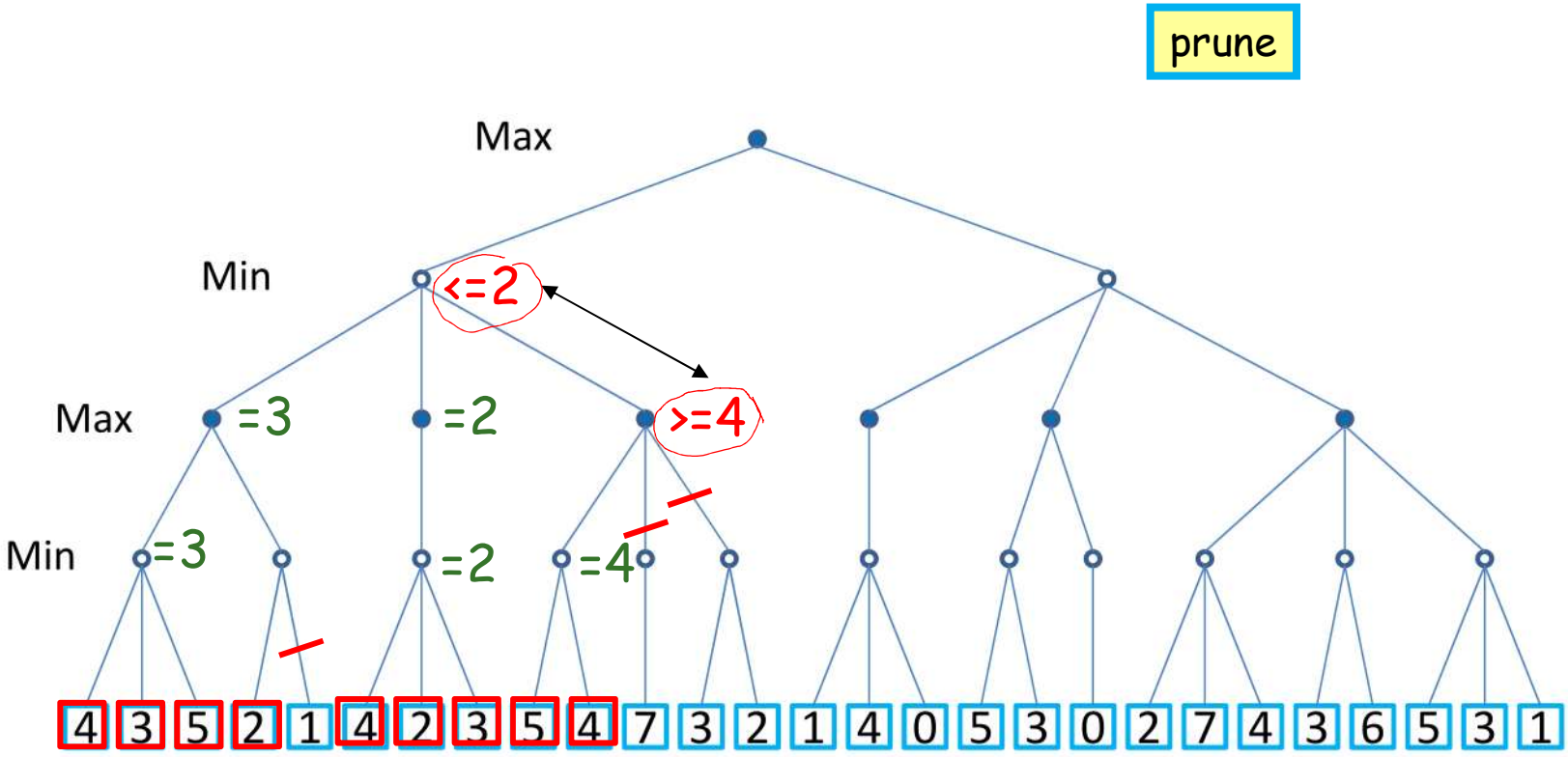
Alpha-Beta Pruning: Example 3



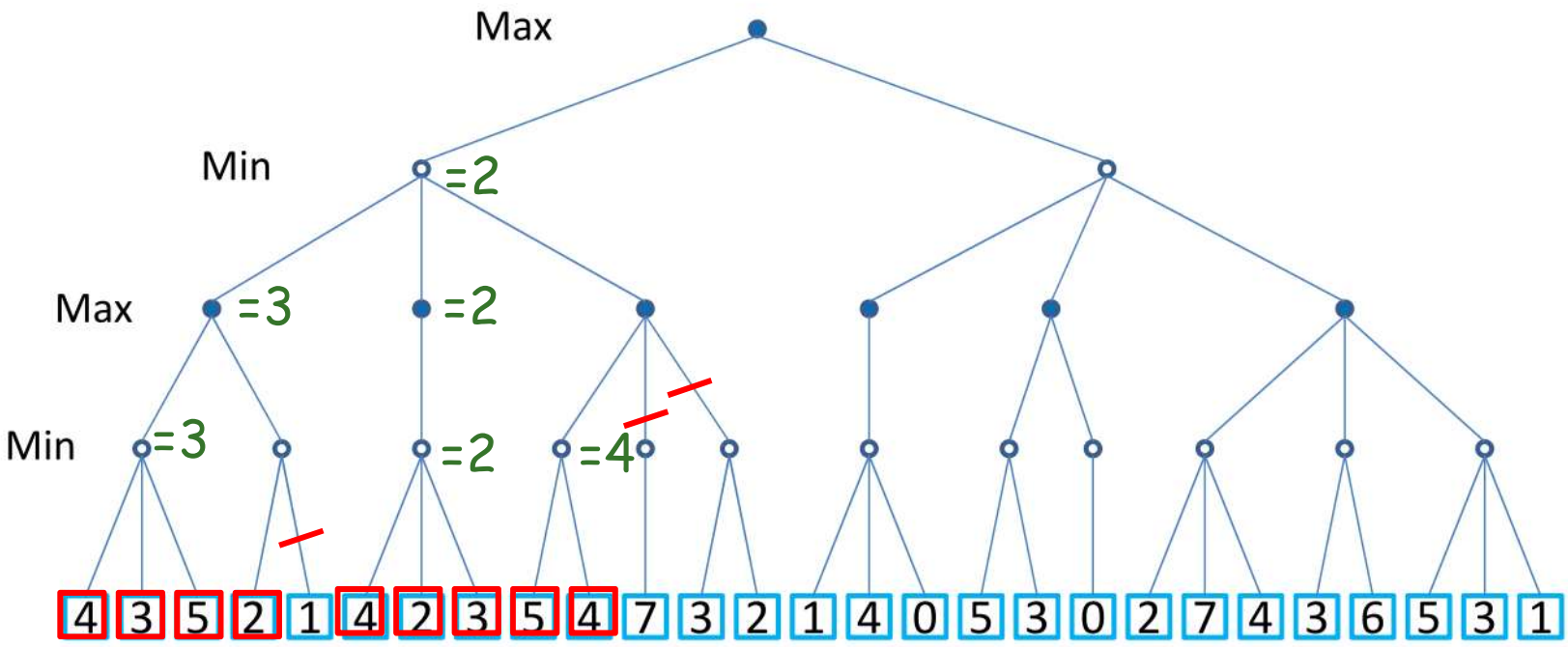
Alpha-Beta Pruning: Example 3



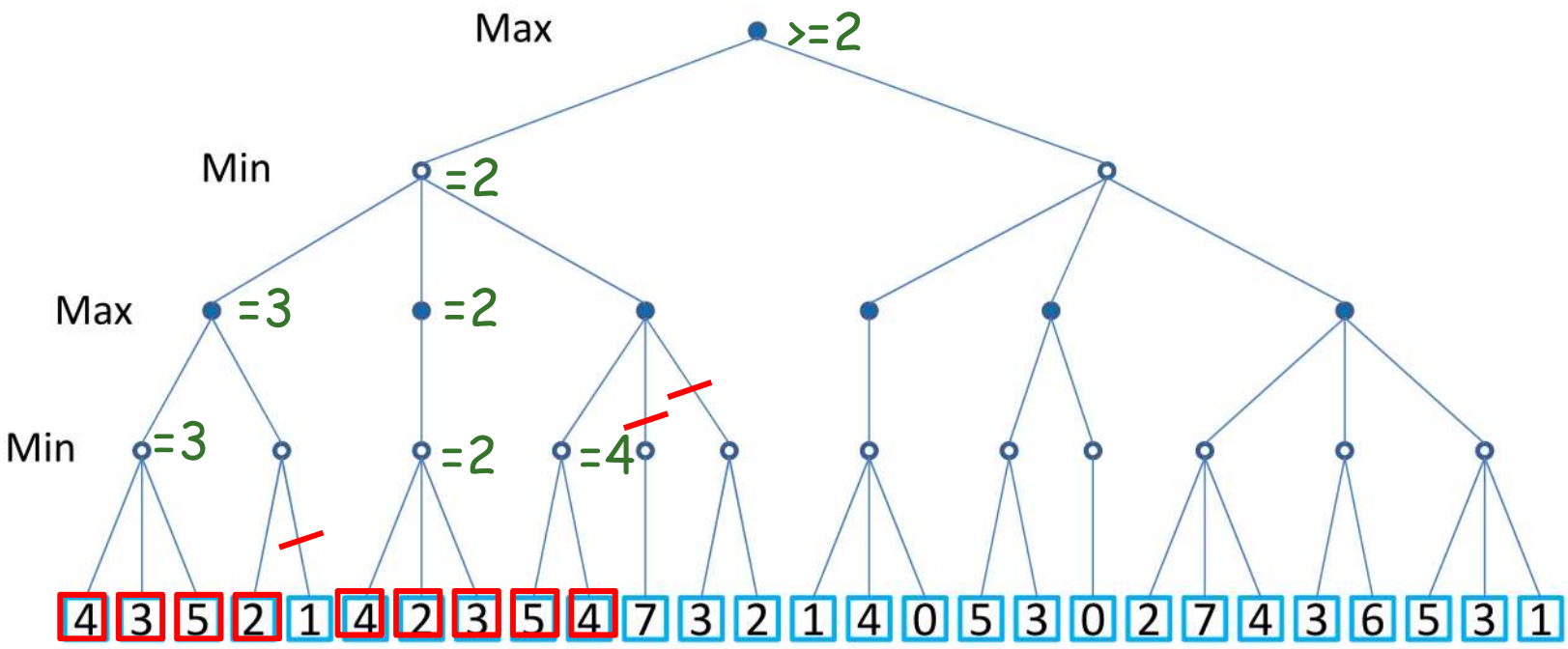
Alpha-Beta Pruning: Example 3



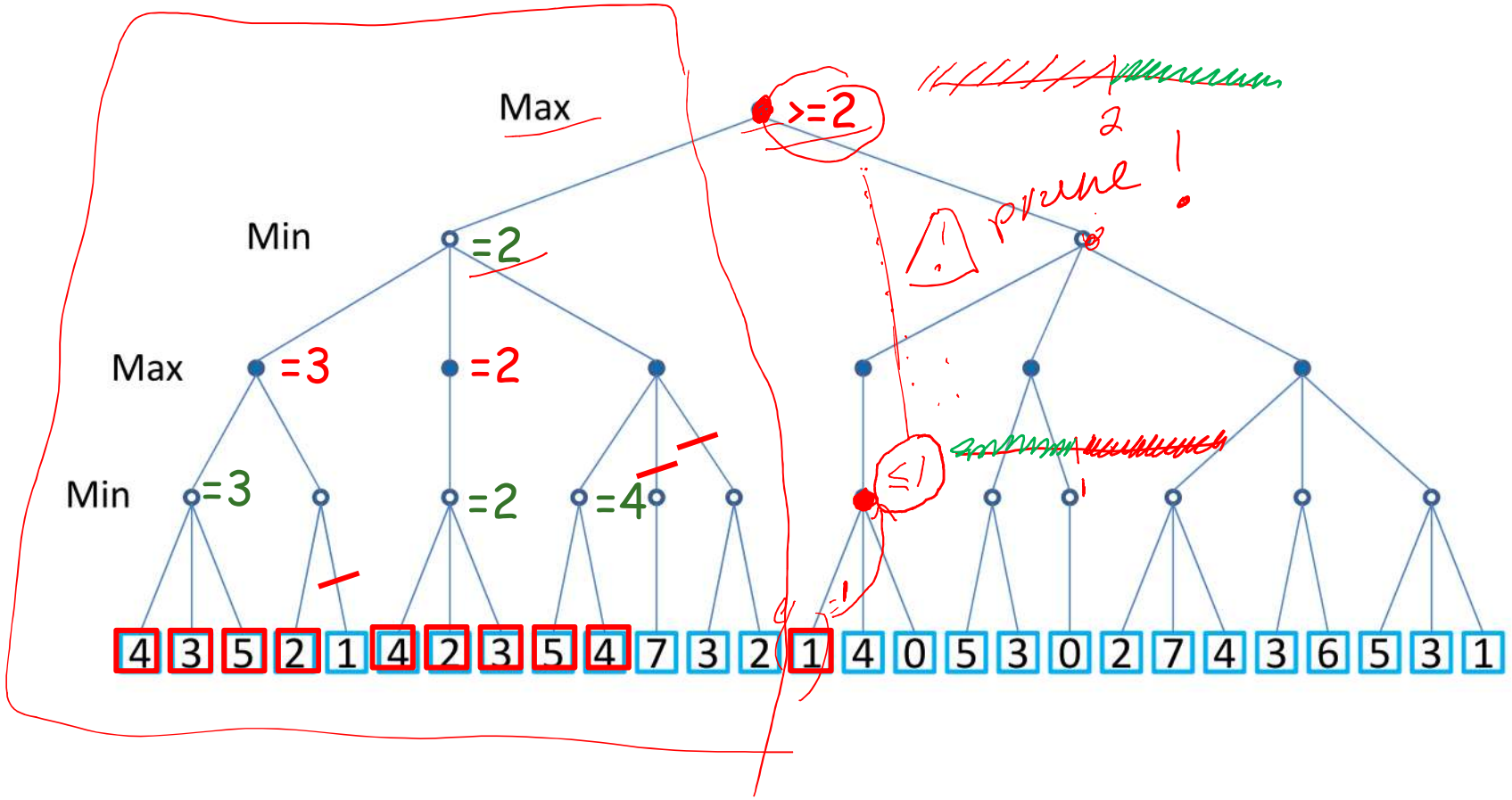
Alpha-Beta Pruning: Example 3



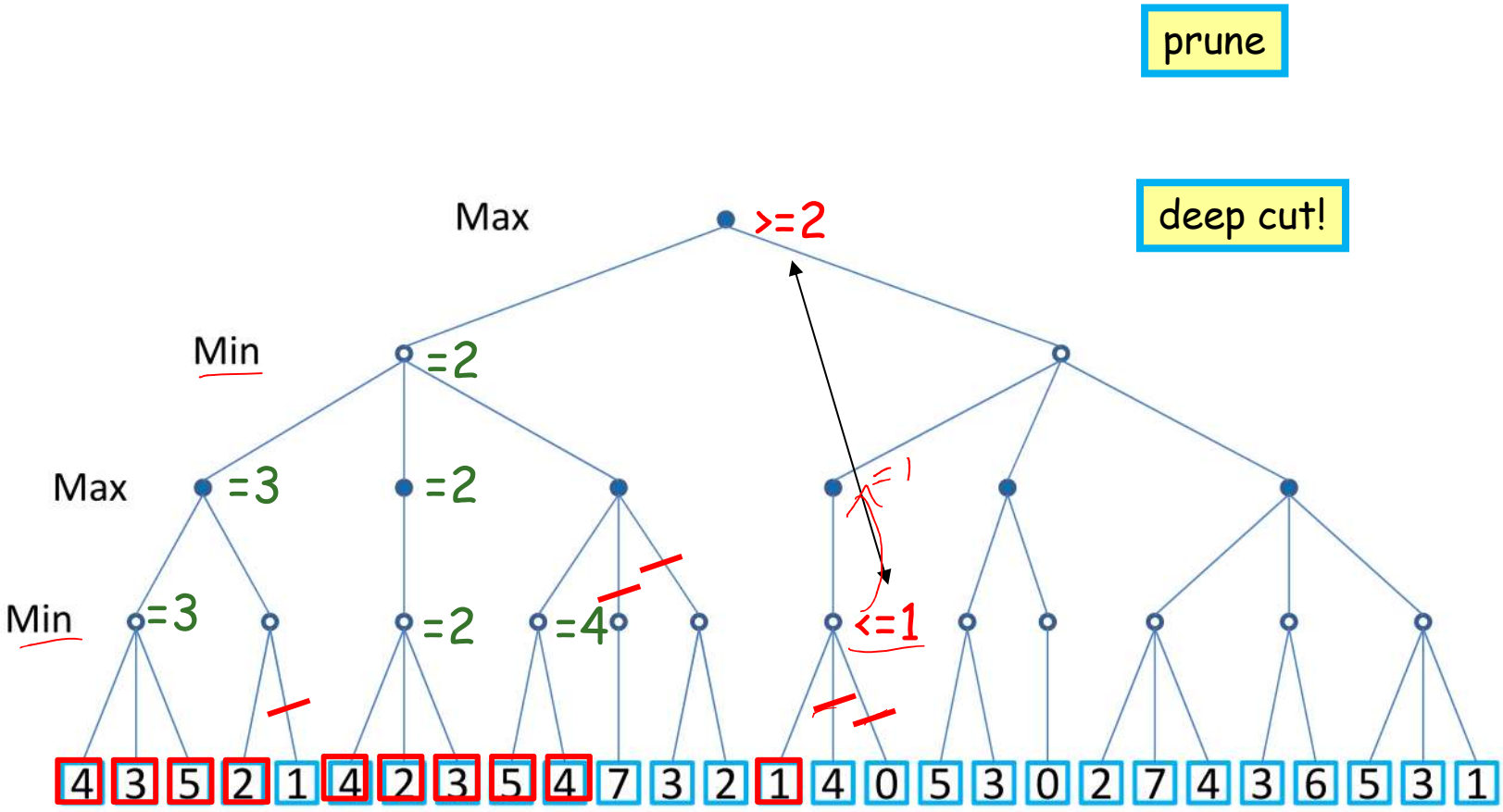
Alpha-Beta Pruning: Example 3



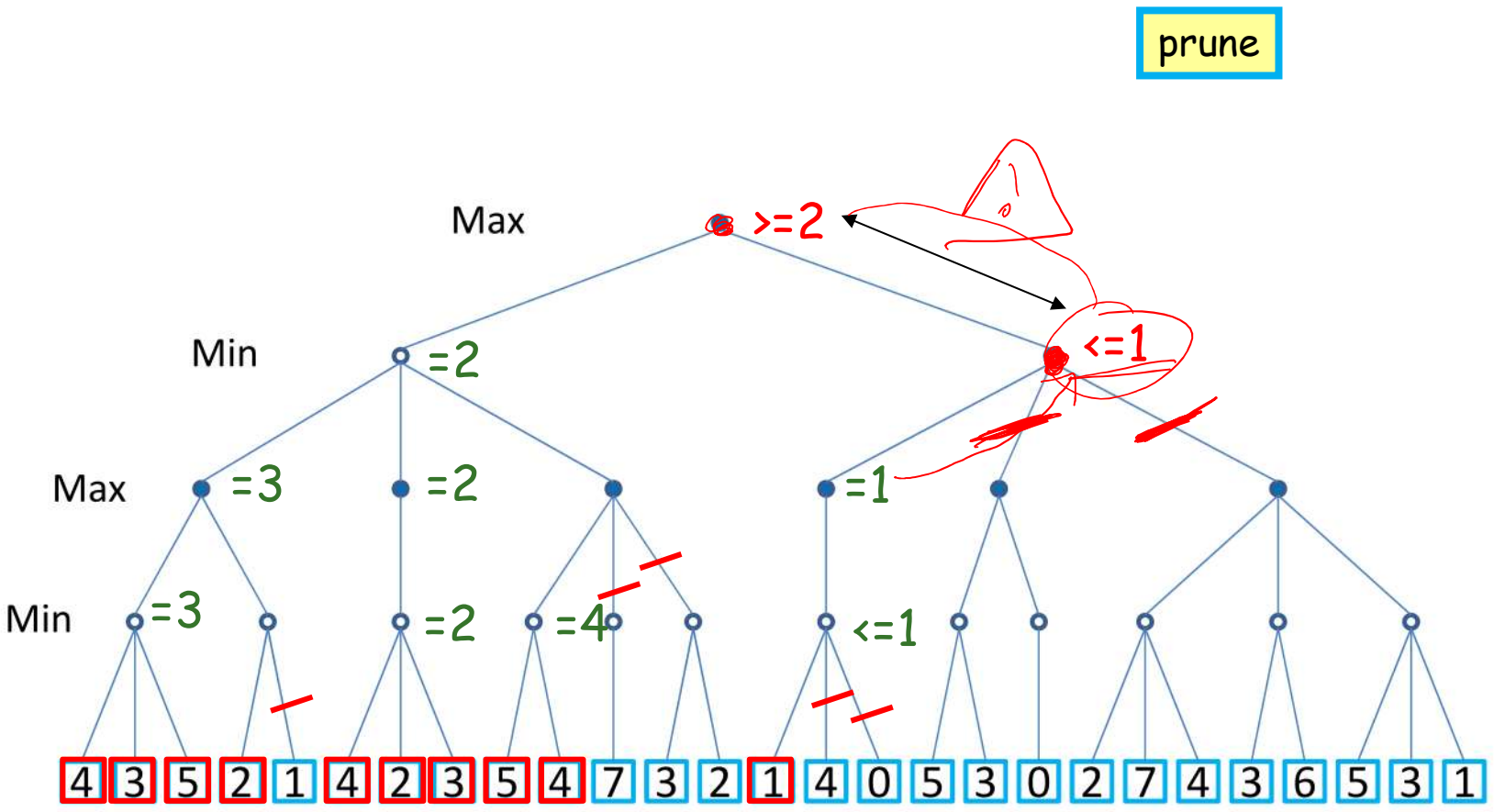
Alpha-Beta Pruning: Example 3



Alpha-Beta Pruning: Example 3



Alpha-Beta Pruning: Example 3

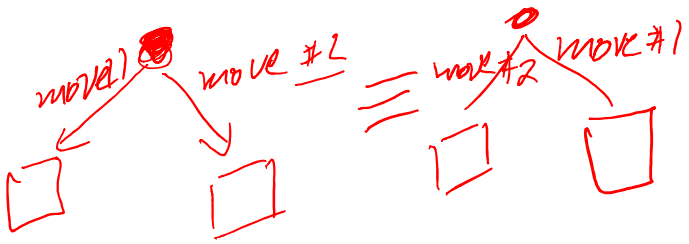


10 nodes explored out of 27

Efficiency of Alpha-Beta Pruning

效率取决于sibling的排列顺序

- Depends on the order the siblings
  - which is an arbitrary choice ;-(



试图generate最优化的tree

- In worst case: 最差没用
  - alpha-beta provides no pruning
  - plus extra overhead cost ;-(

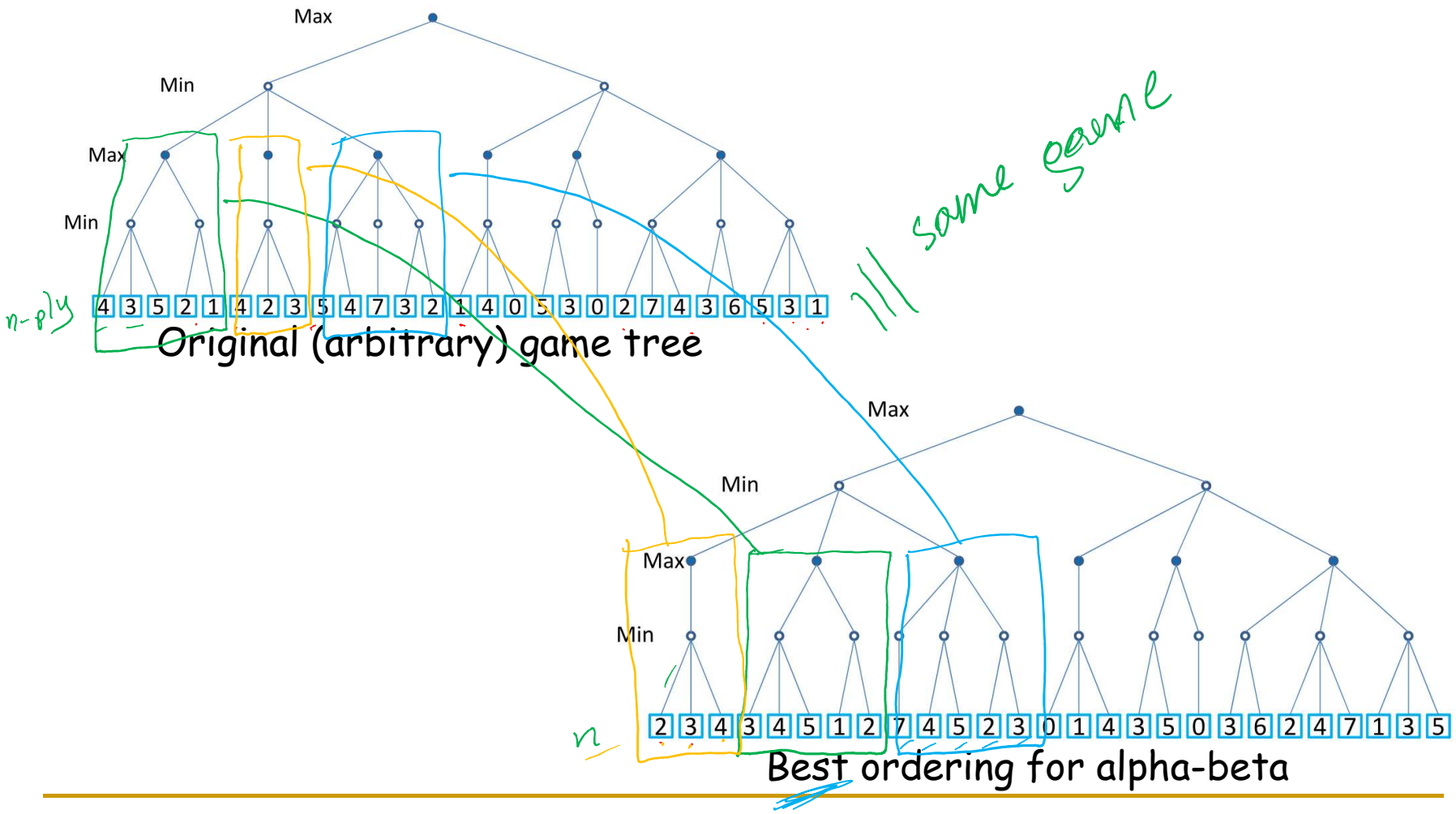
最好开根

- In best case:
  - branching factor is reduced to its square root

$36^{e(n)}$

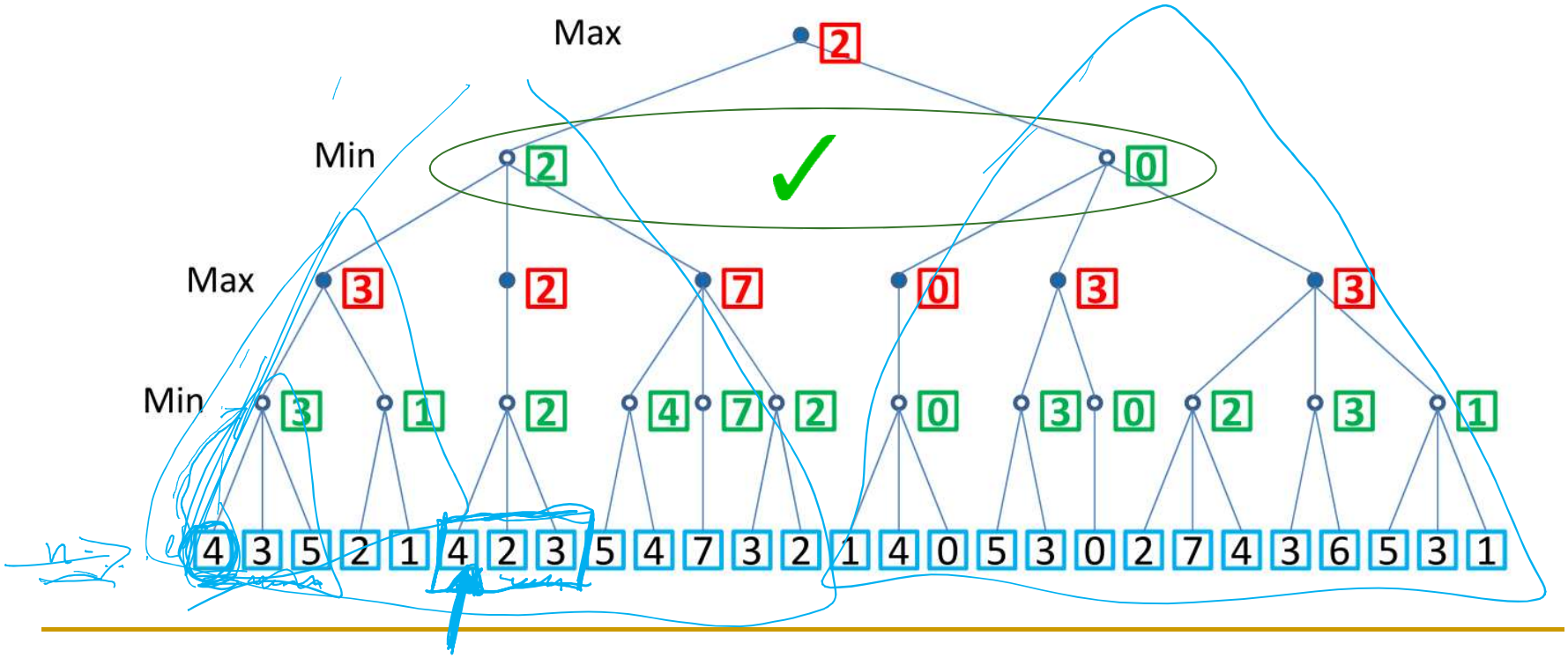
$\Rightarrow 6^{e(n)}$

Alpha-Beta: Best ordering



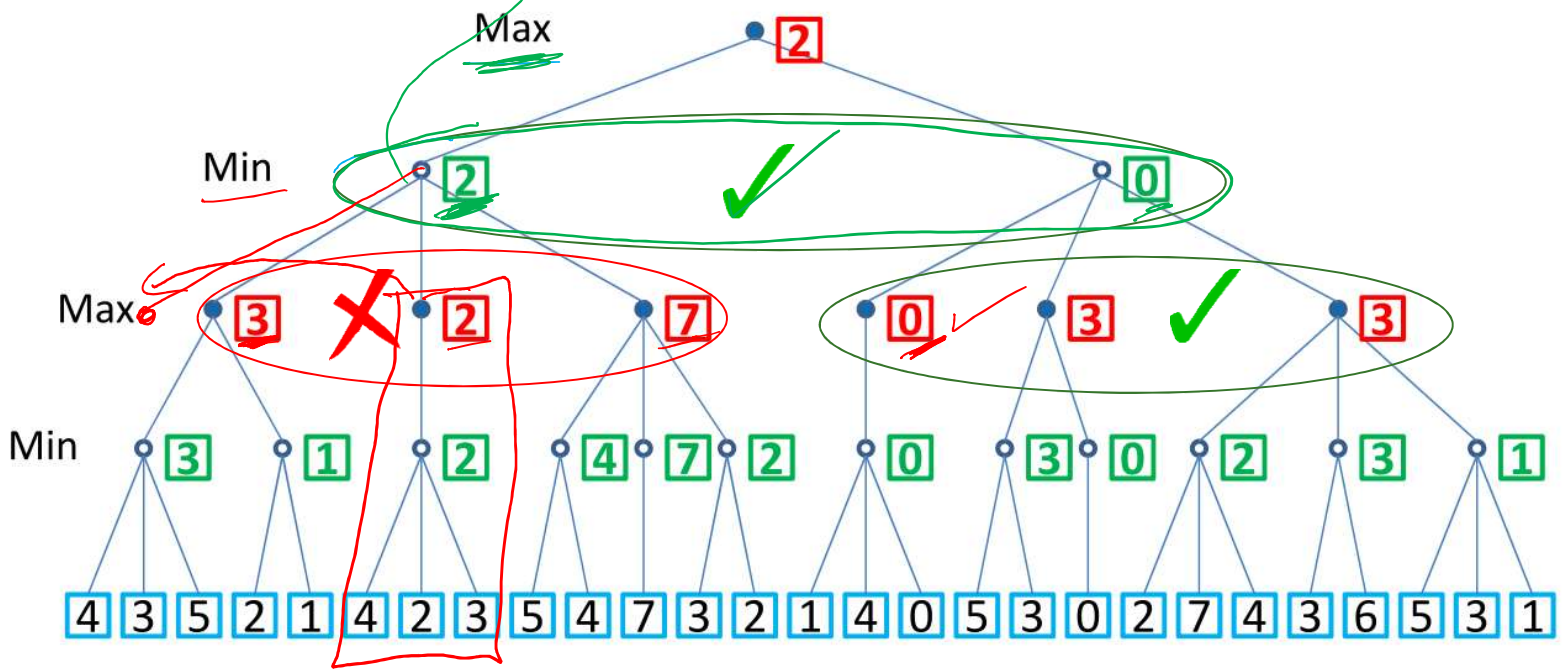
Alpha-Beta: Best ordering

- best ordering:
  - strongest constraint placed first, ie:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first



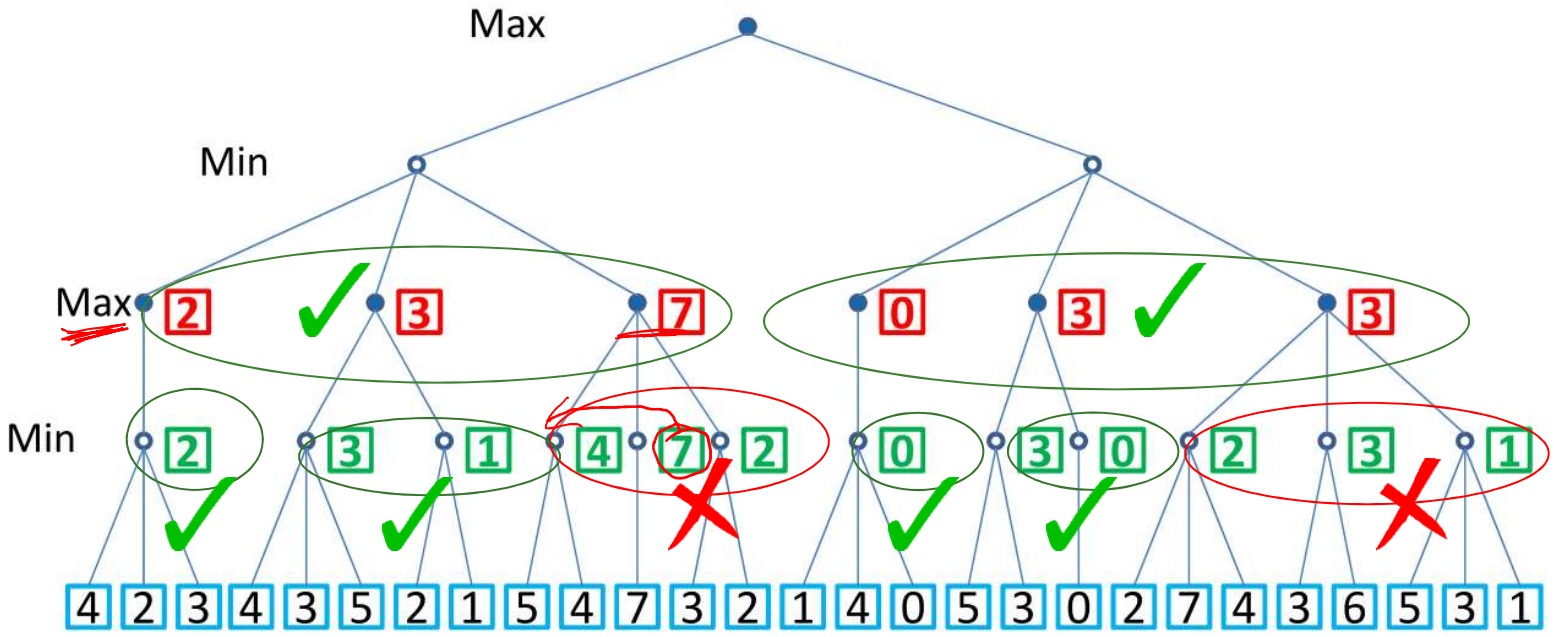
Alpha-Beta: Best ordering

- best ordering:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first

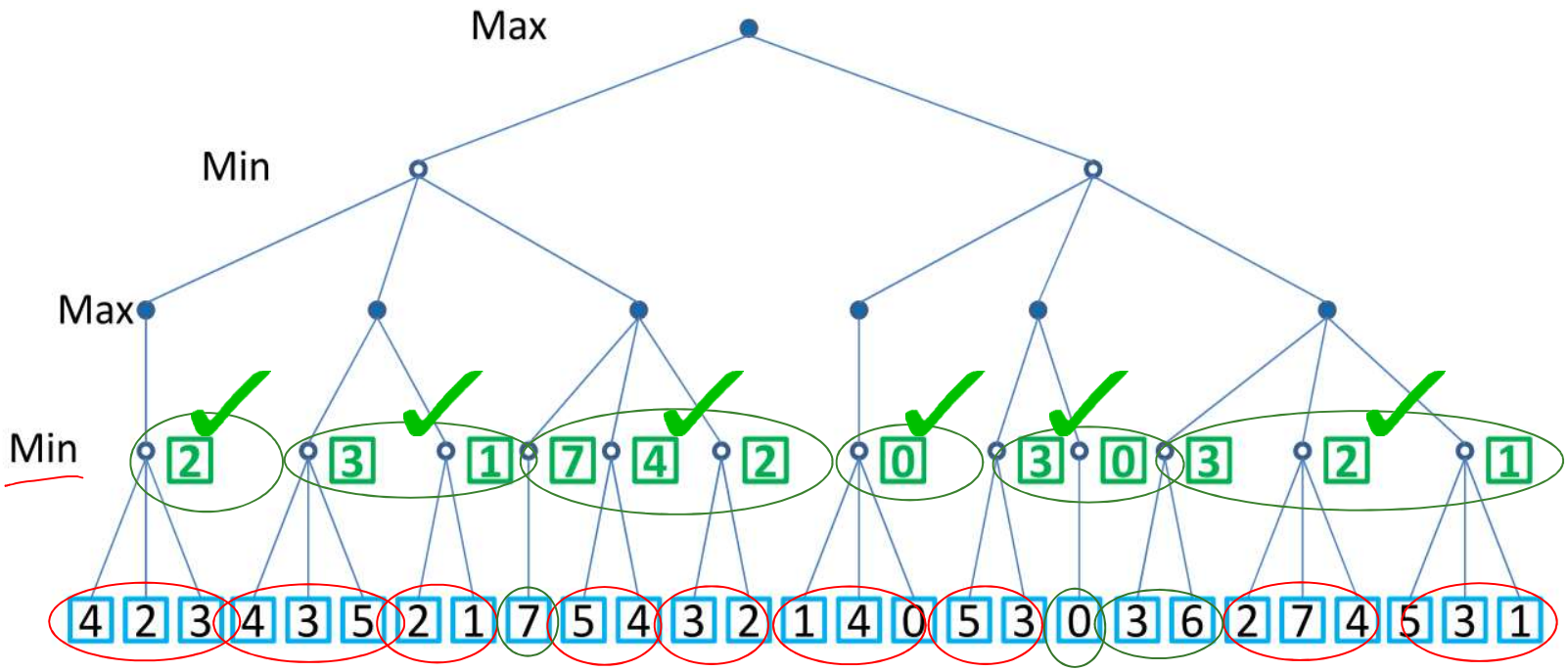


Alpha-Beta: Best ordering

- best ordering:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first

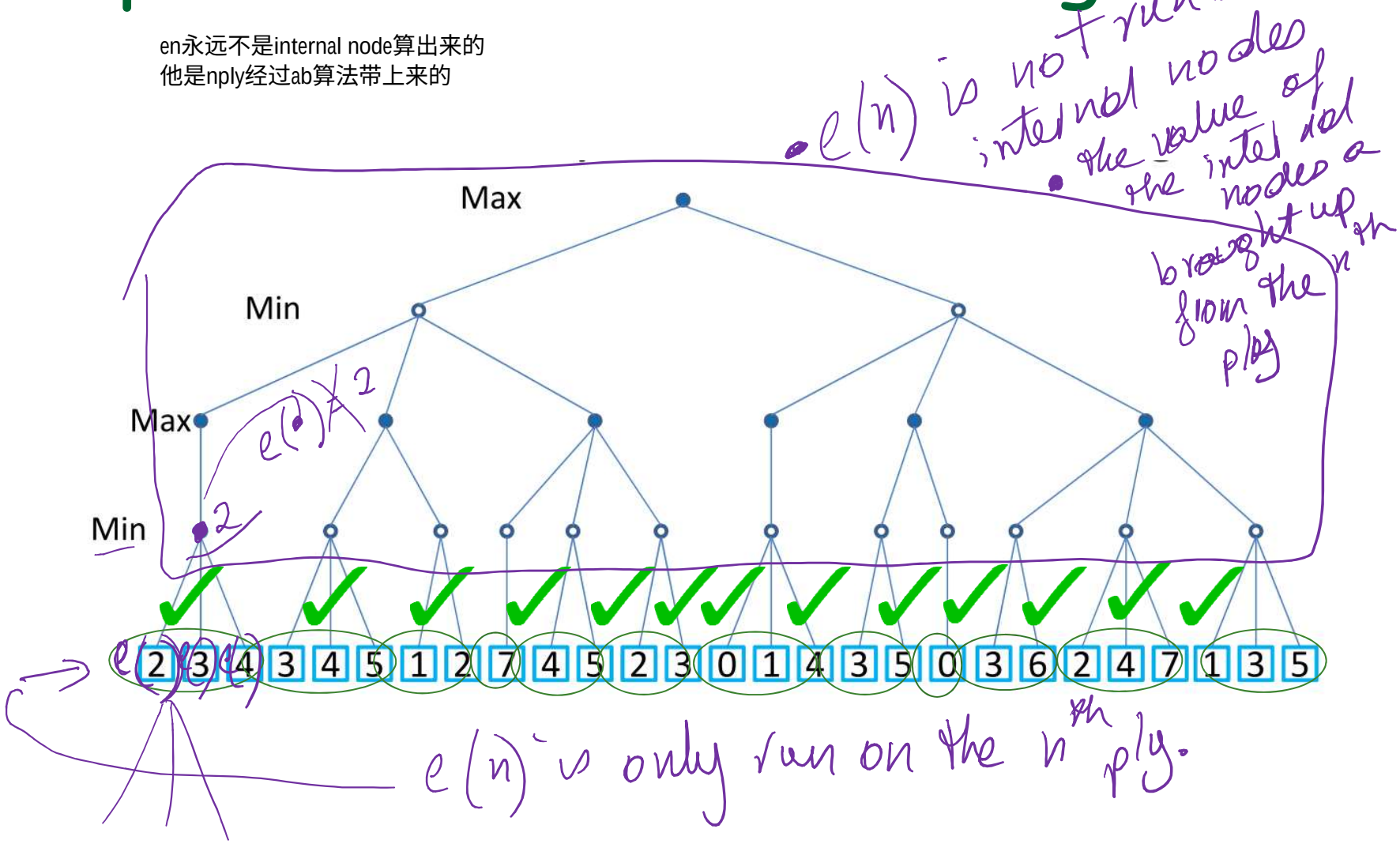


Alpha-Beta: Best ordering



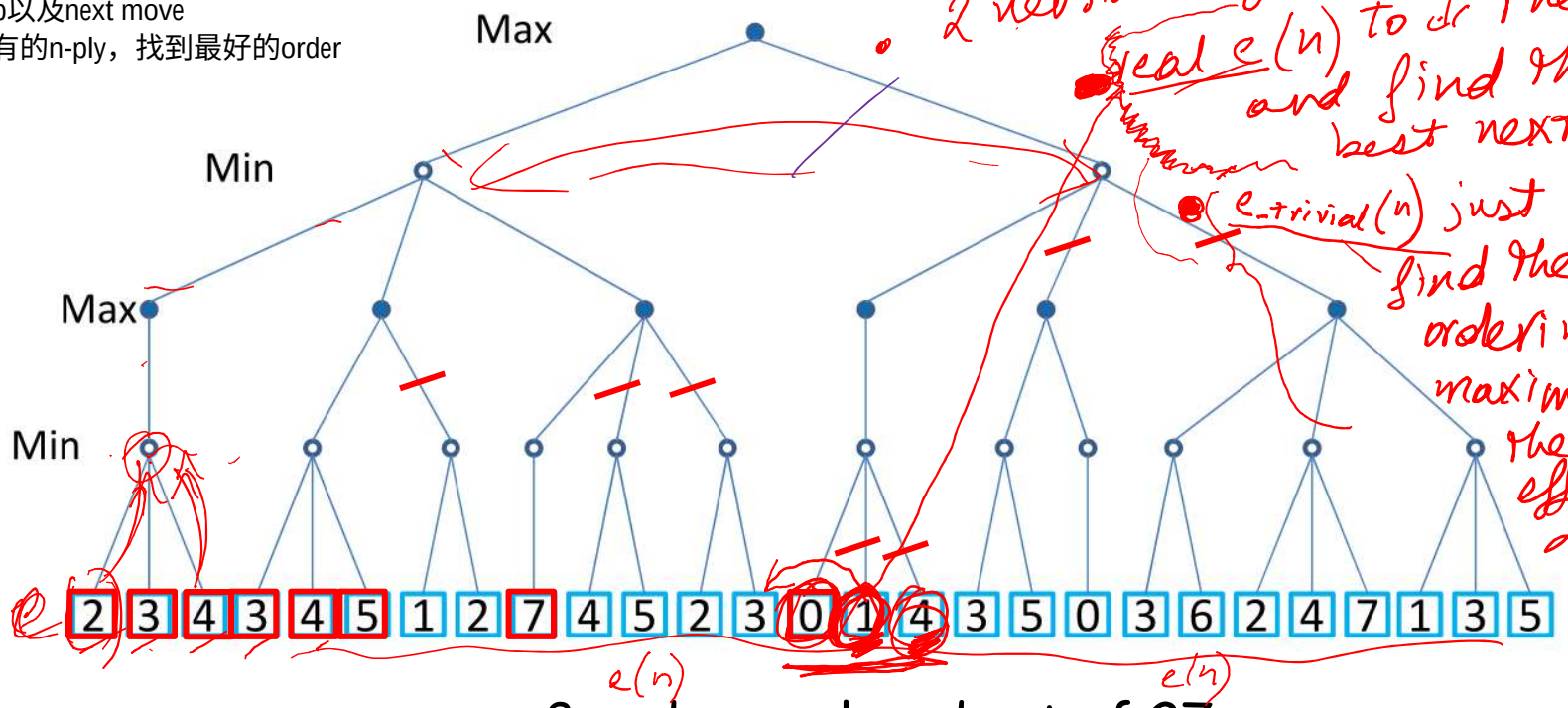
Alpha-Beta: Best ordering

en永远不是internal node算出来的
他是nply经过ab算法带上来的



Alpha-Beta: Best ordering

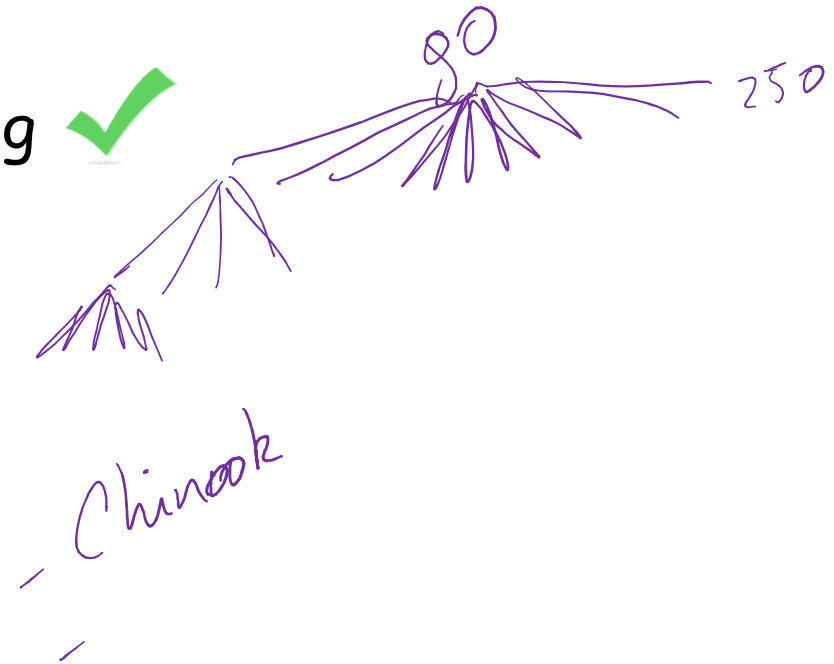
问题在于，想要进行排列，我们先要把所有en算出来，这样是矛盾的
因此我们设计两个版本的en
real en:用来进行a-b以及next move
e-trival n，遍历所有的n-ply，找到最好的order



8 nodes explored out of 27

Today

- Adversarial Search
  1. Minimax
  2. Alpha-beta pruning



Up Next

- Part 5: Natural Language Processing