第五章、对抗搜索

- 在有其它智能体计划与我们对抗的世界中如何预先计划
- 博弈的概念
  - 数学中的博弈论把任何多智能体环境看成是一种博弈游戏，其中每个智能体对其它智能体的影响是“显著的”，与智能体是合作的还是竞争的无关。
  - AI中的博弈通常指有完整信息的、确定性的、轮流行动的、两个游戏者的零和游戏。
  - 游戏因为难解而令人感兴趣！

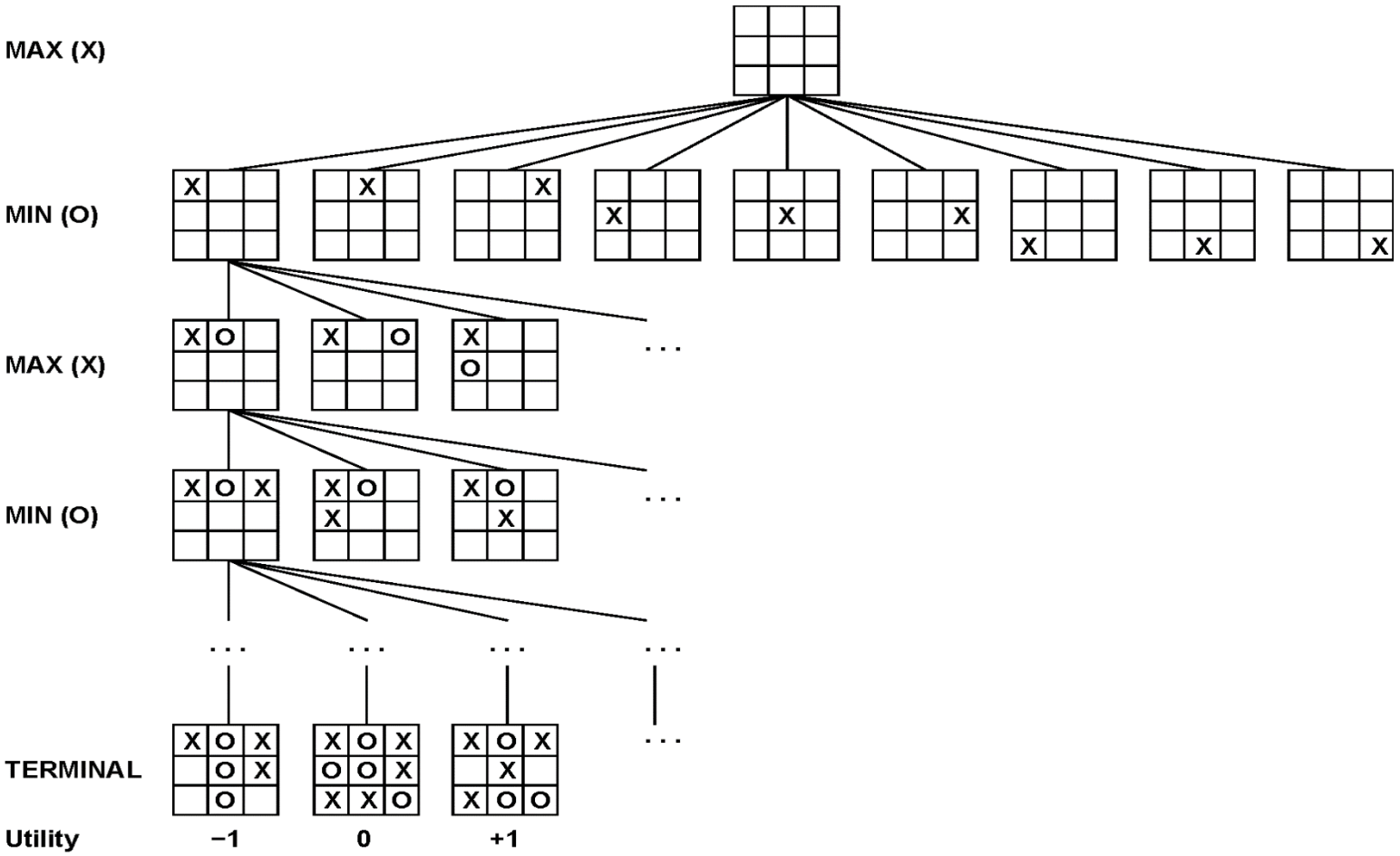
第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

例子：MAX和MIN

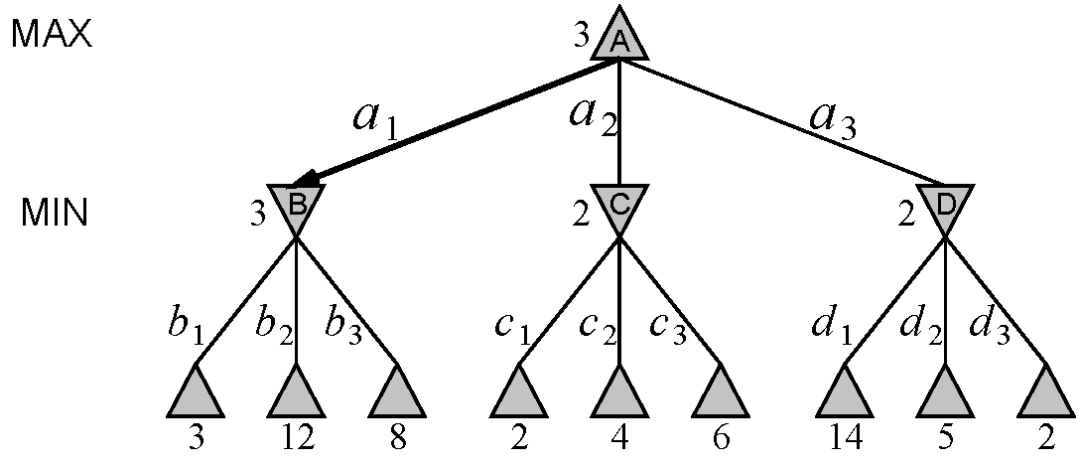
- 问题的表述：MAX先行，然后两人轮流出招，直到游戏结束。在游戏的最后，给优胜者加分，给失败者罚分。
  - 初始状态，包括棋盘局面和确定该哪个游戏者出招。
  - ACTION(s)后继函数，返回(move, state)。
  - TERMINAL-TEST(s)终止测试
  - UTILITY(s,p)效用函数：对终止状态给出一个数值。

例子：井子棋游戏-博弈树



最优策略

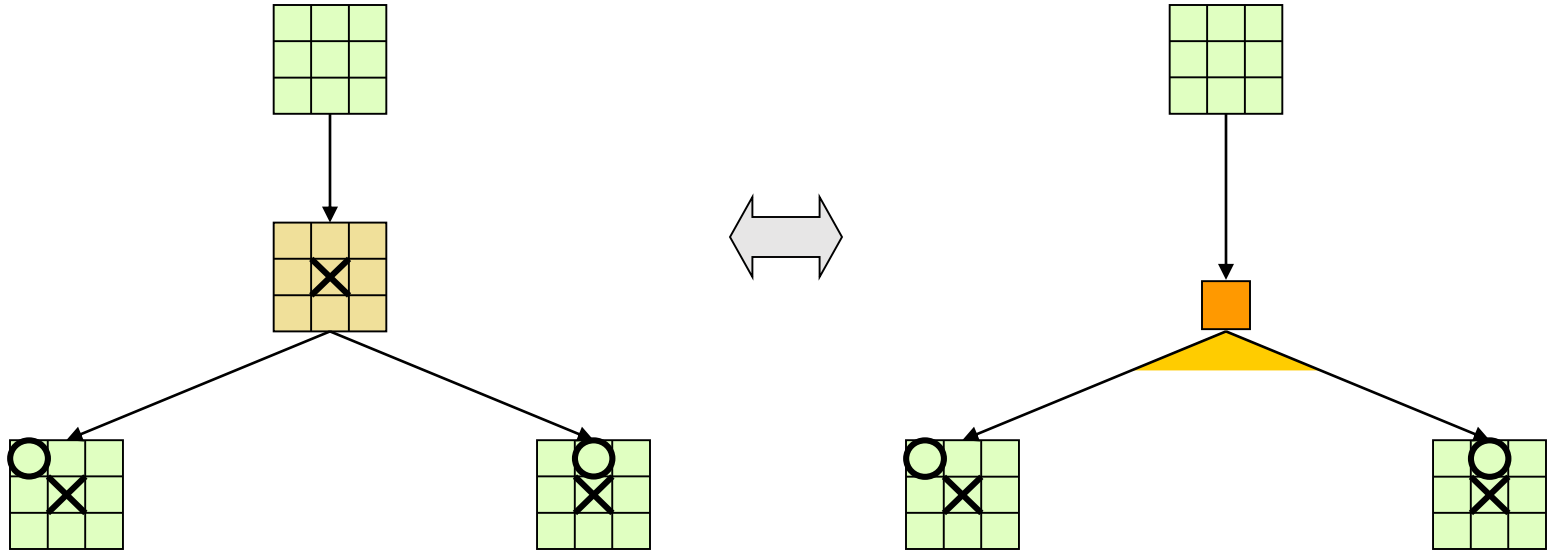
- 最优解是导致取胜的终止状态的一系列招数
- 但，在游戏中MIN还有发言权
- 在对手不犯错误时，最优策略能够导致至少不比任何其它策略差的结果。



一颗两层的博弈树

对手的变数

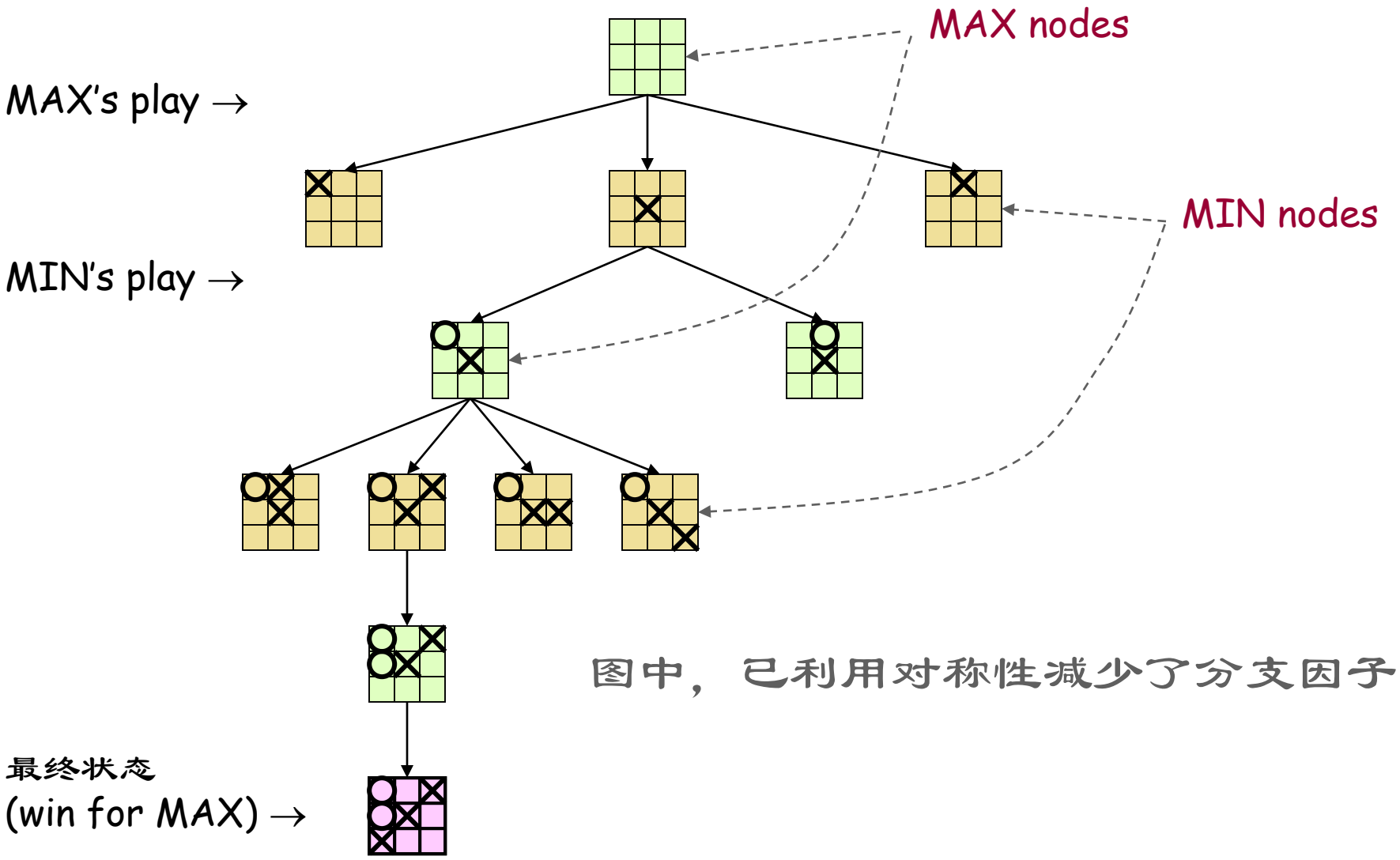
- 由于对手(MIN)的动作带来了问题的不确定性，这是我方(MAX) 做决策时需要考虑的问题



博弈

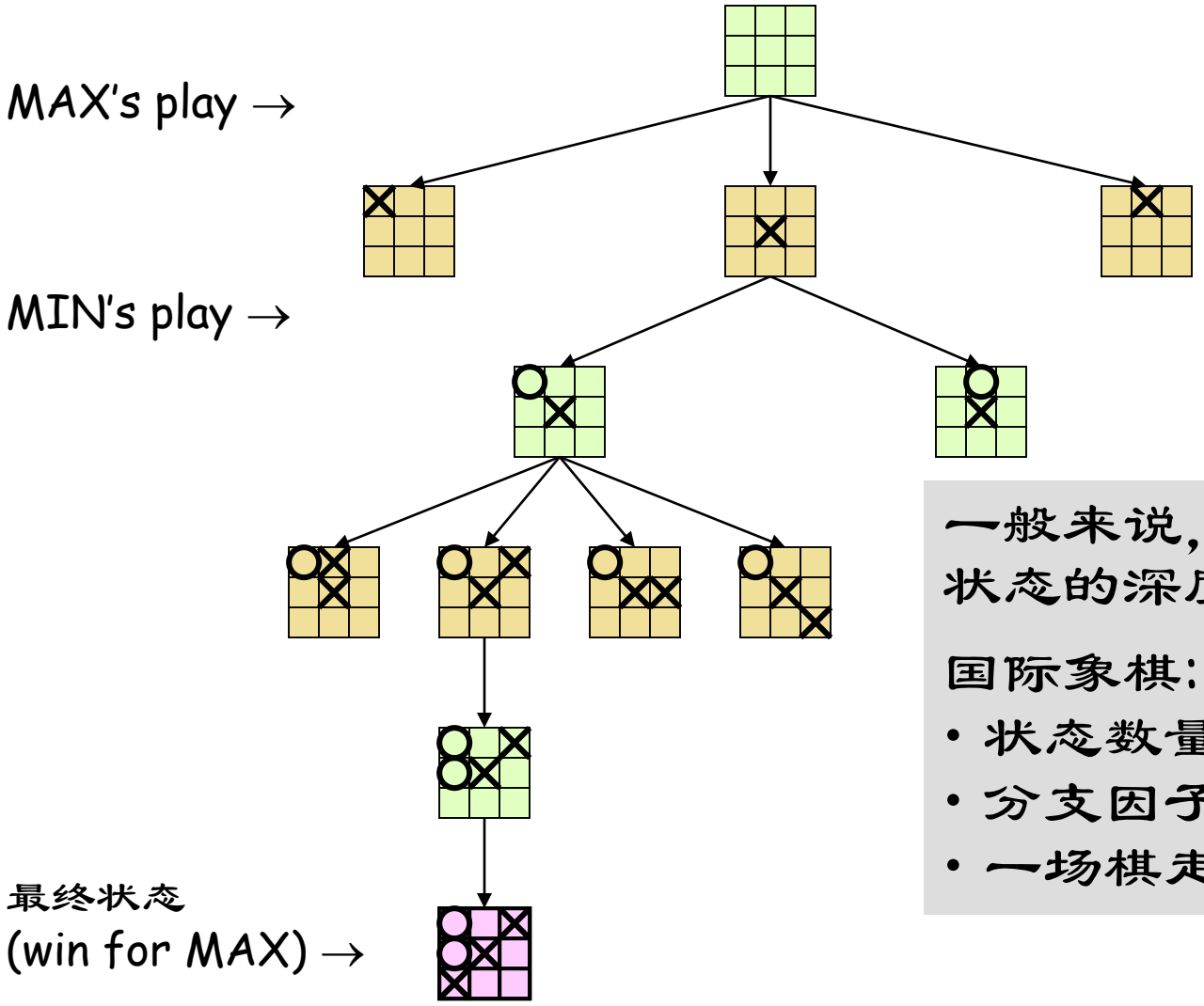
- 由于对手(MIN)的动作带来了问题的不确定性，这是我方(MAX) 做决策时需要考虑的问题
- MIN 希望 MAX 失利 (反之亦然)
- 如果无视MIN的应对，那么MAX没有任何希望能够胜出 (对于 MIN亦然)
- 每一个回合，必须在有限的时间内做出行动决策
- 状态空间非常大：在有限的时间内只能对空间的一小块进行探索

博弈树
Game Tree



图中，已利用对称性减少了分支因子

博弈树
Game Tree



一般来说，分支因子和最终状态的深度是很大的

国际象棋：

- 状态数量：~10^{40}
- 分支因子：~35
- 一场棋走棋步数：~100

选择一步走棋：基本思想

1) 将当前状态作为初始状态，建立一个深度为h的搜索树，h（horizon，称为视野）是在有限时间内能够考虑到的最大深度

2) 对所有叶节点状态进行评价

3) 由叶节点回推至根节点，选择其中最好的一个动作决策（假设对手MIN的应对总是给MAX带来最坏的结果）

→ 极大极小算法 Minimax algorithm

最优策略的确定

- 通过检查每个节点的极小极大值来决定，记为MINMAX-VALUE(n)。

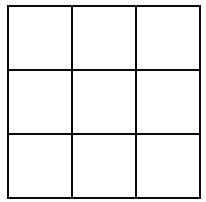
$$MINIMAX(s) = \begin{cases} UTILITY(s) & s\text{为终止状态} \\ \max_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & s\text{为}MAX\text{结点} \\ \min_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & s\text{为}MIN\text{结点} \end{cases}$$

评价函数
Evaluation Function

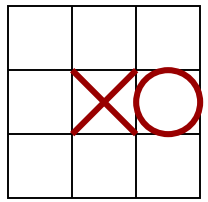
- 函数 e: 状态 s → 数值 e(s)
- e(s) 即为估计状态s对于MAX来说“好”的程度的启发式信息
- e(s) > 0 意味着状态s对于MAX来说是有利的
  (数值越大越有利)
- e(s) < 0意味着状态s对于MIN 来说是有利的
- e(s) = 0 意味着状态 s 是中立的

例子：Tic-tac-Toe

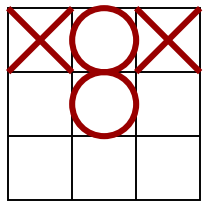
e(s) = 对MAX开放的行，列，对角线数量
－对MIN开放的行，列，对角线数量



8－8 = 0

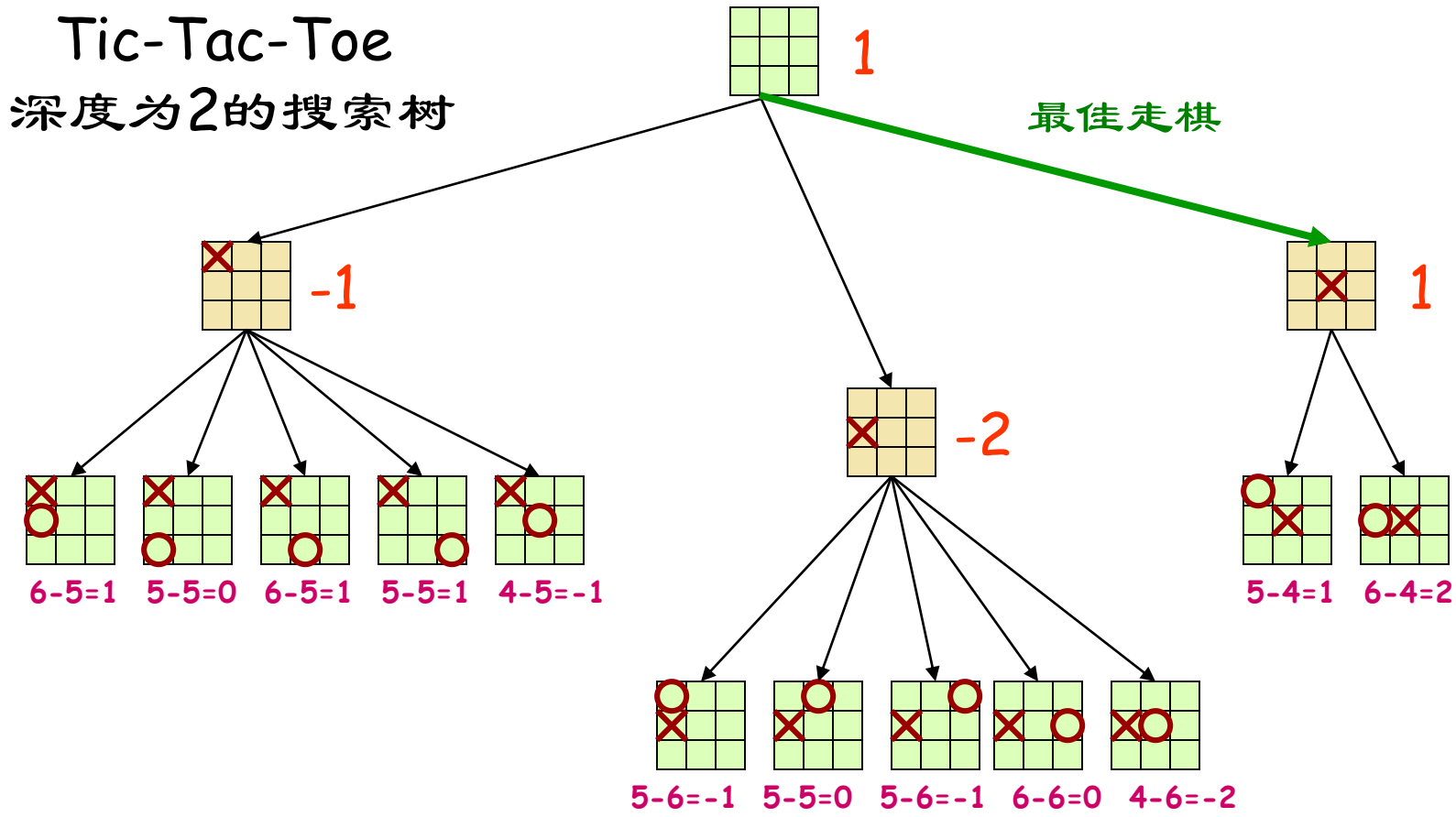


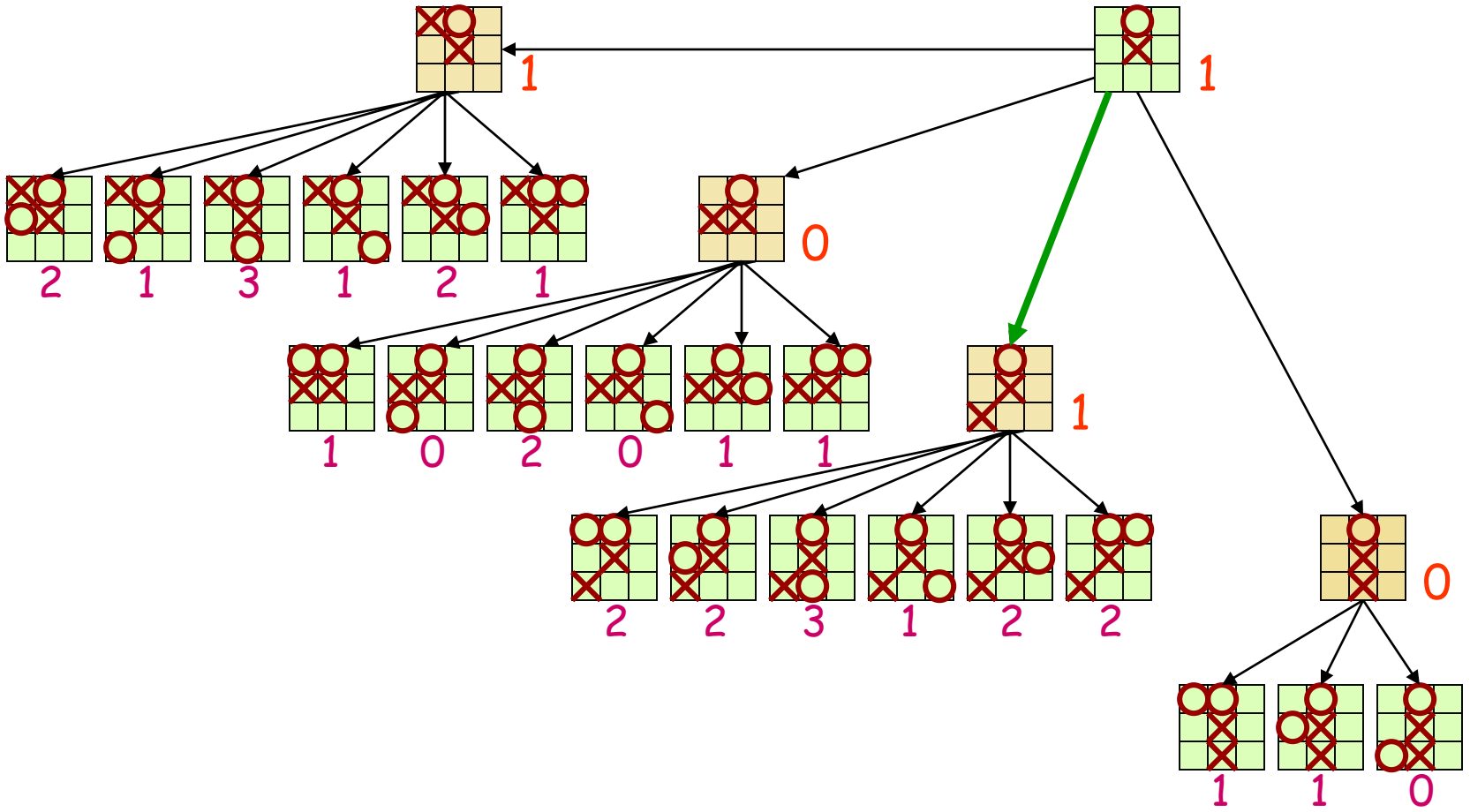
6－4 = 2



3－3 = 0

值的回推


Tic-Tac-Toe
深度为2的搜索树
1
最佳走棋
-1
-2
1
6-5=1
5-5=0
6-5=1
5-5=1
4-5=-1
5-4=1
6-4=2
5-6=-1
5-5=0
5-6=-1
6-6=0
4-6=-2


1
1
2
1
3
1
2
1
0
1
0
2
0
1
1
1
2
2
3
1
2
2
0
1
1
0

为什么使用回推值?

- 在每一个非叶节点N，回推值就是MAX到达深度时能够获得的最大值（总是认为对手MIN的应对是最好的）
- e(STATE(N))可以对节点N的状态"好"的程度进行估计，而回推值则是一个更好的估计

极大极小算法

Minimax Algorithm

1. 从当前状态（MAX走棋）开始，扩展博弈树到深度h
2. 对博弈树的每一个叶节点计算评价函数值
3. 由叶节点开始至根节点计算回推值：

   a
   b

   视野：因为需要在时间限制内做出决策
4. 选择能够得到最大回推值的走棋

博弈过程（MAX）

到达最终状态前重复以下步骤

1. 采用极大极小方法选择一步走棋
2. 按1的决策走棋
3. 观察MIN的应对走棋

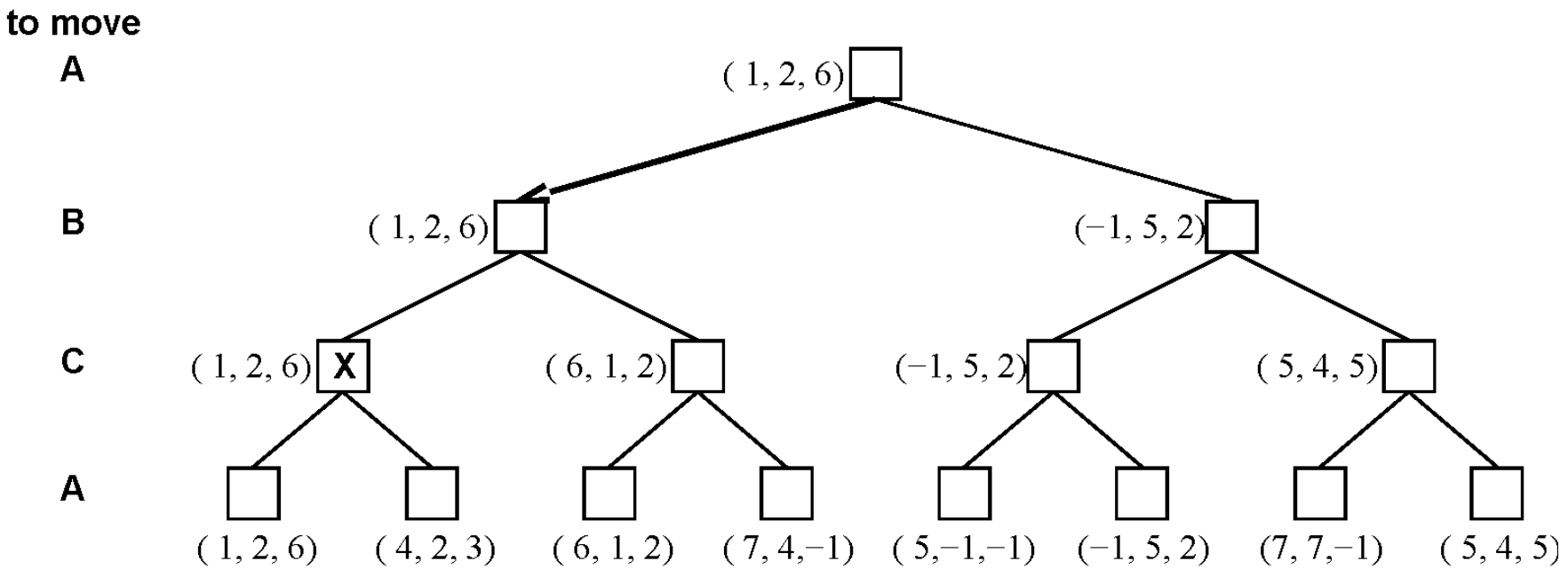
注意每一回合建造深度为h的博弈树仅为选择当前的一步走棋

下一个回合重复所有的步骤（上一回合中深度为h-2的子树可以拿来重复使用）

极小极大值的递归算法

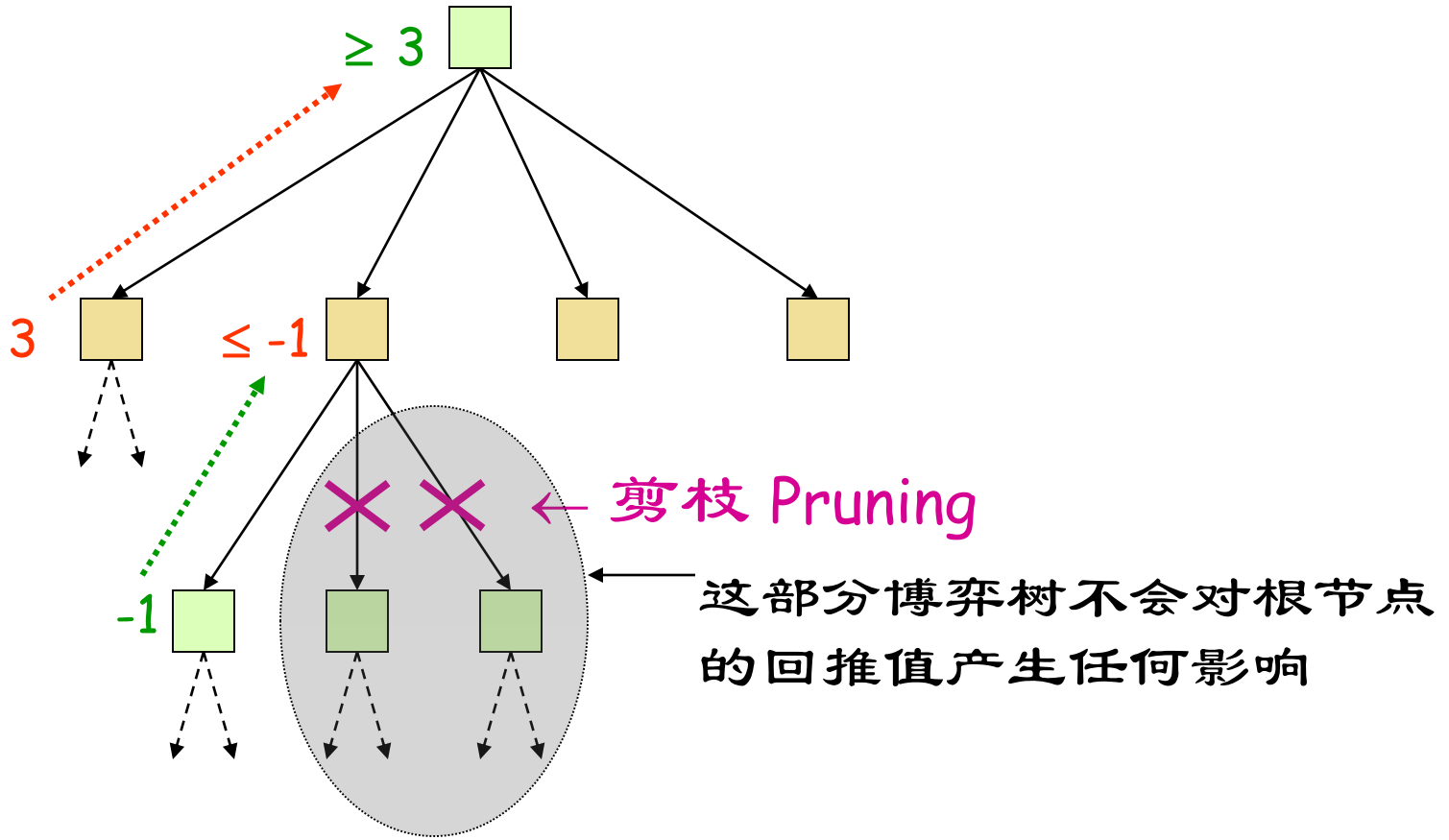
```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

多人游戏中的最优决策


to move
A
B
C
A
(1, 2, 6)
(1, 2, 6)
(−1, 5, 2)
(1, 2, 6)
X
(6, 1, 2)
(−1, 5, 2)
(5, 4, 5)
(1, 2, 6)
(4, 2, 3)
(6, 1, 2)
(7, 4,−1)
(5,−1,−1)
(−1, 5, 2)
(7, 7,−1)
(5, 4, 5)

还能做得更好吗？

是的！ 能够做得更好！

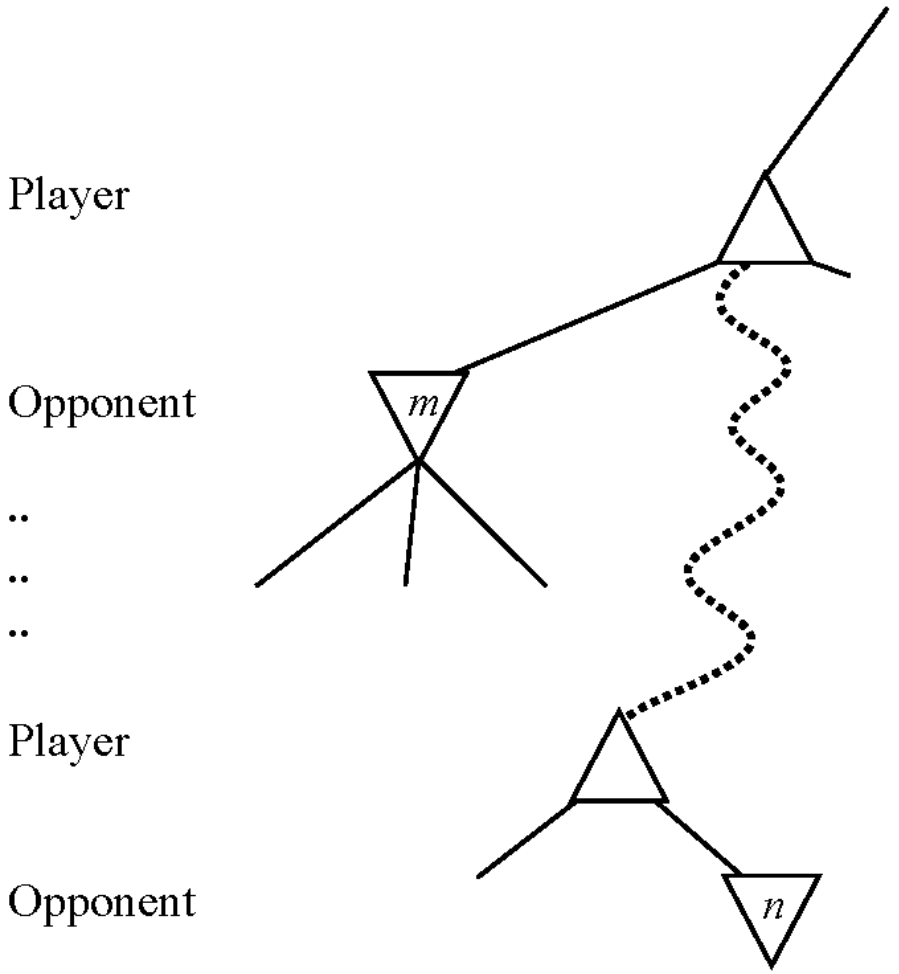


第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

α–β剪枝

- 极小极大值搜索的时间复杂度是$O(b^m)$
- 剪枝：剪裁掉那些不影响最后决策的分支。
- 一般原则：考虑在树中某处的节点n，如果游戏者在n的父节点或上层的任何节点有一个更好的选择m，那么在实际的游戏中就永远不会到达n，即n被剪裁掉。



α–β剪枝：如果**m**比**n**好，我们就不会走到**n**。

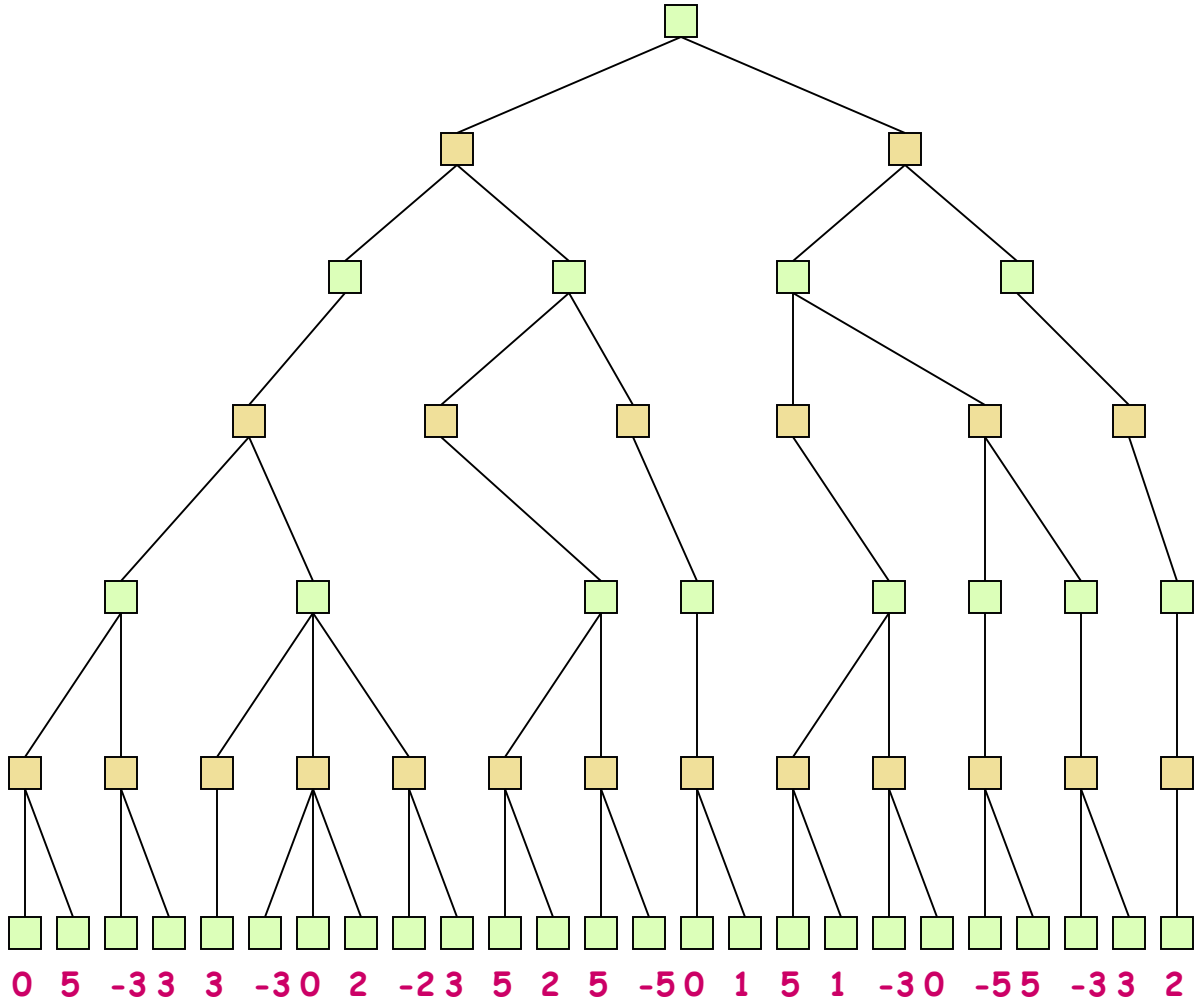
$\alpha-\beta$ 剪枝
Alpha-Beta Pruning

- 采用深度优先方法探索博弈树
- 只要有可能就回推 α 和 β 的值
- 把那些不会改变最终决策的分枝剪去

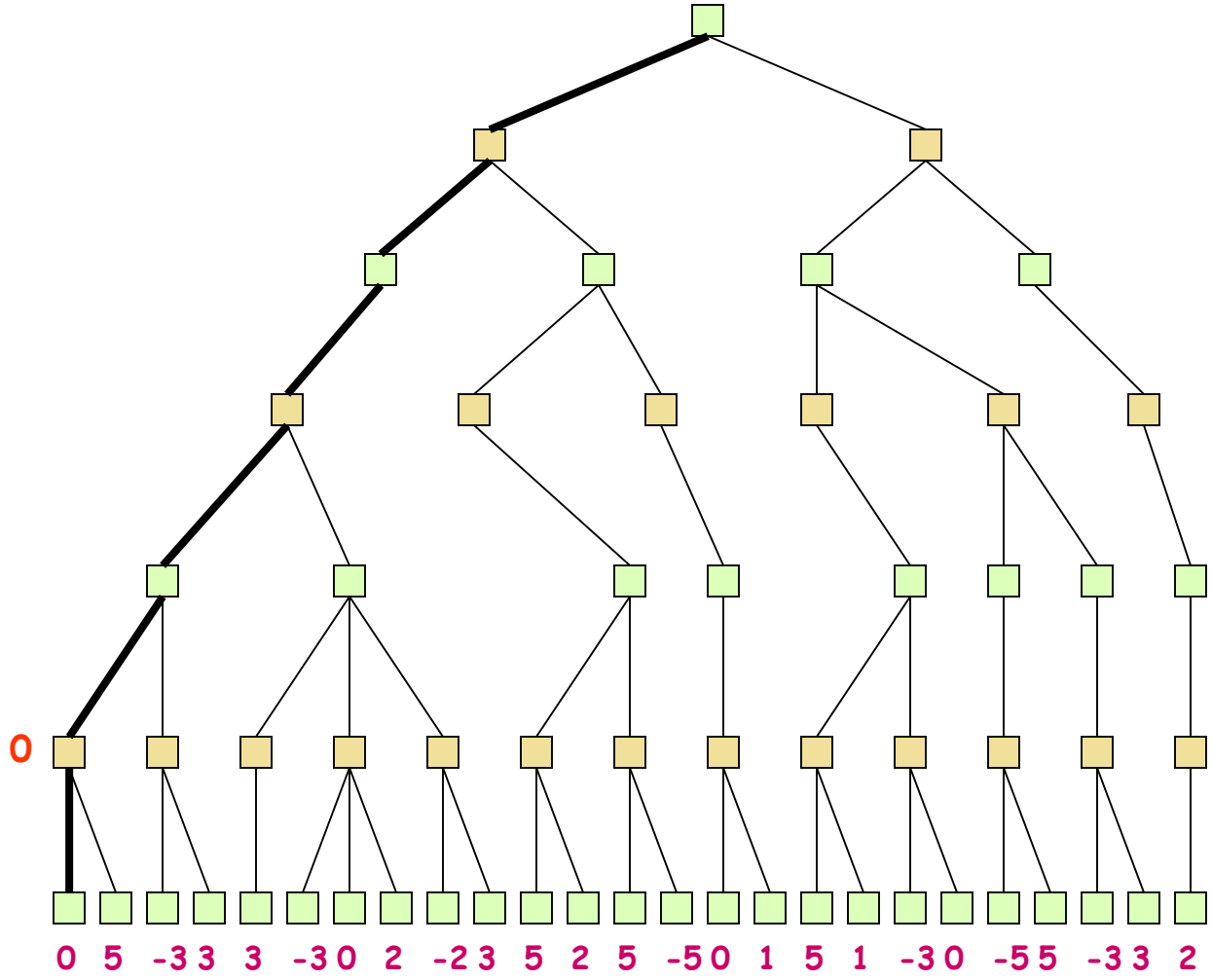
α－β 算法

- 当节点N以下的搜索完成（或剪枝中止）时，更新N的父节点的α或β值
- 当一个MAX节点N的α 值≥ N 的MIN祖先节点的β值，则节点 N 以下的搜索中止
- 当一个MIN节点N的β值≤ N 的MAX祖先节点的α 值，则节点 N 以下的搜索中止

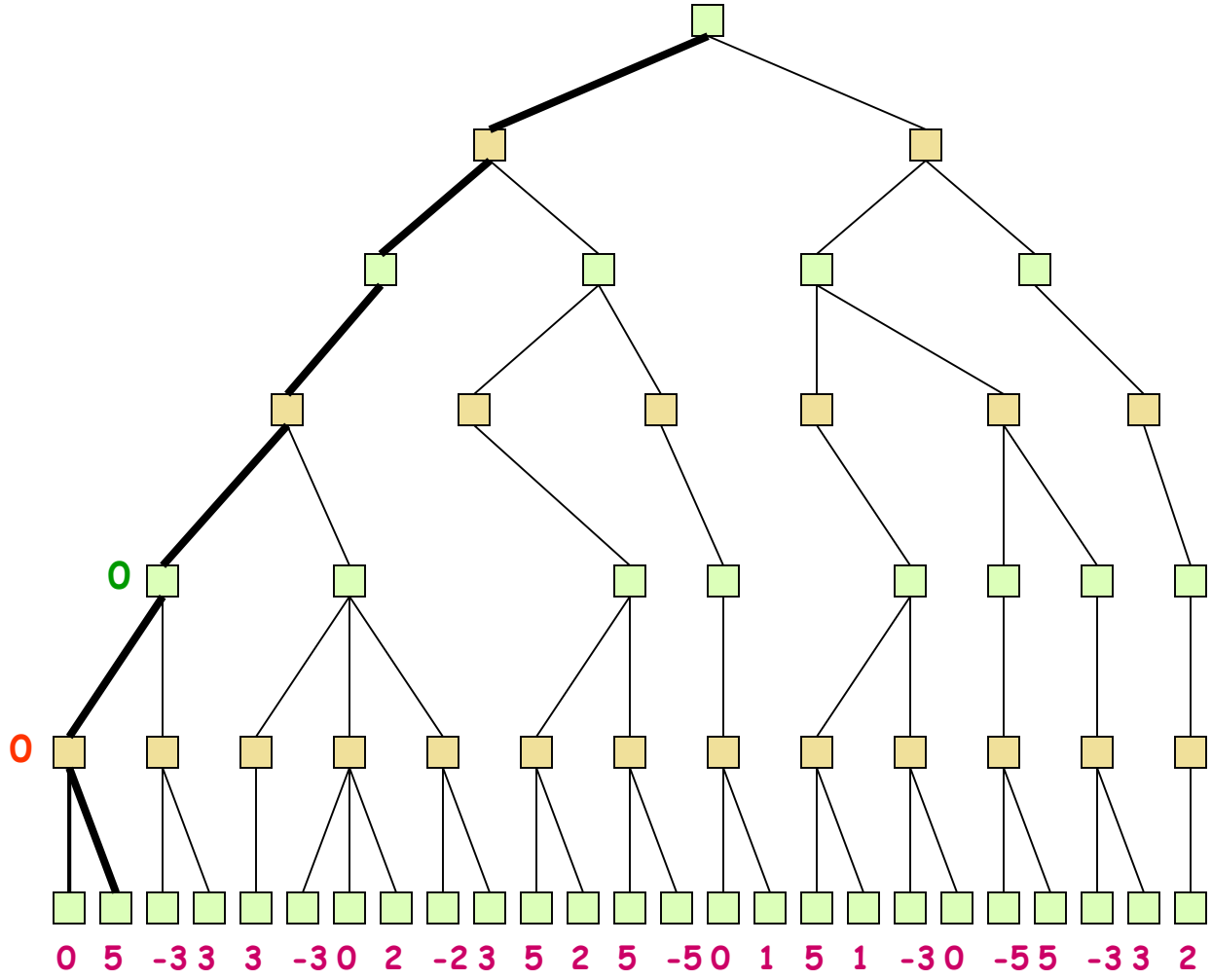
Example



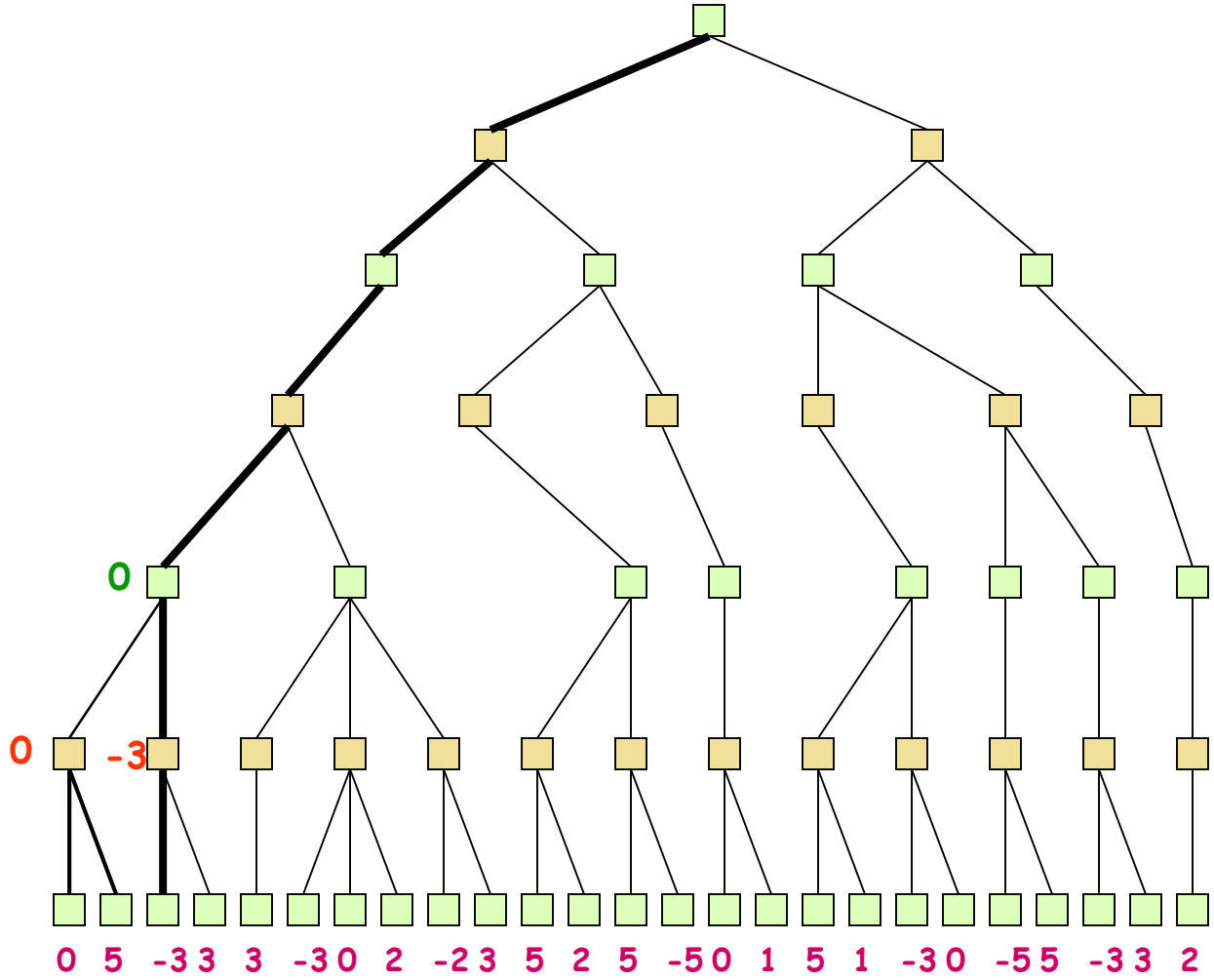
Example



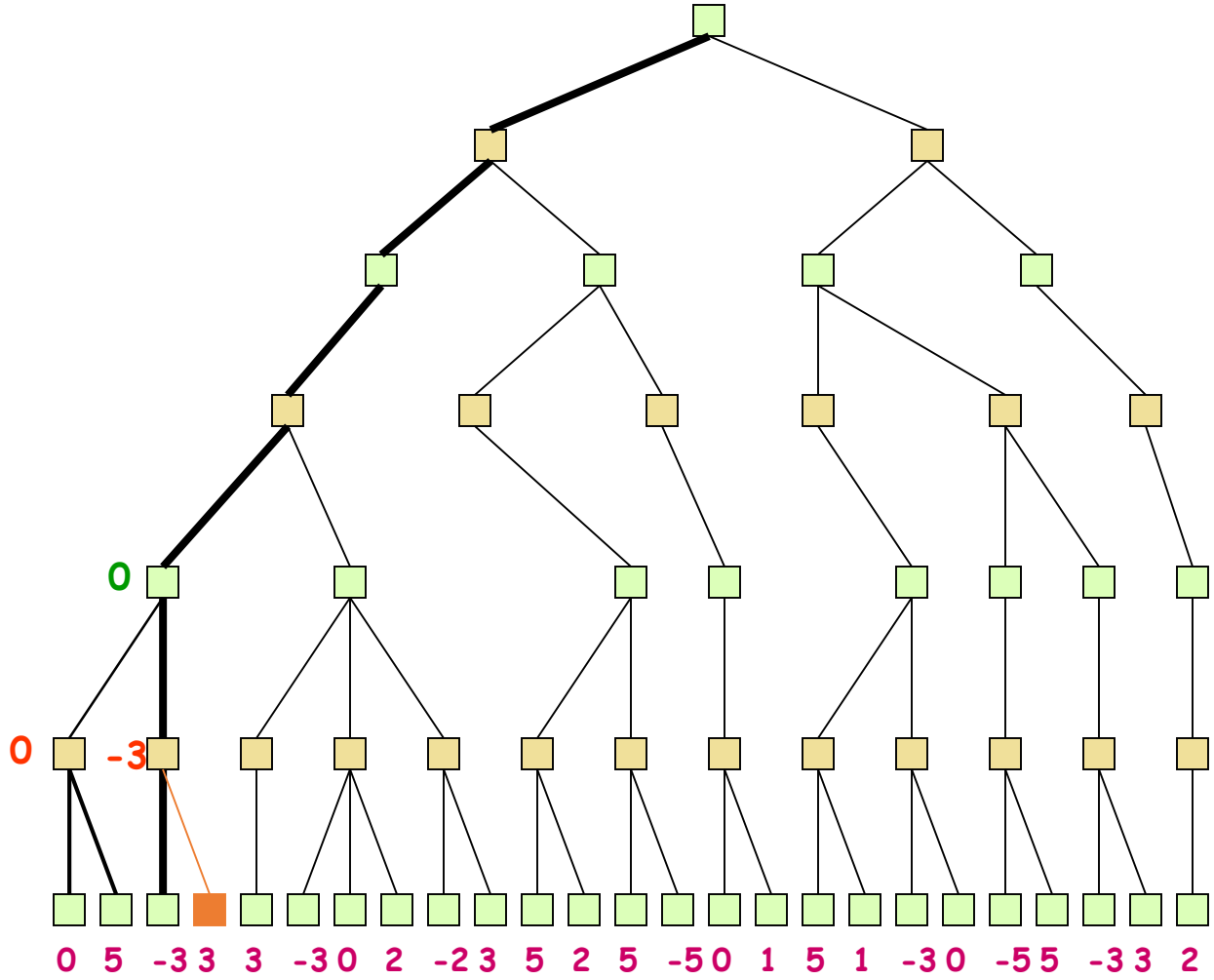
Example



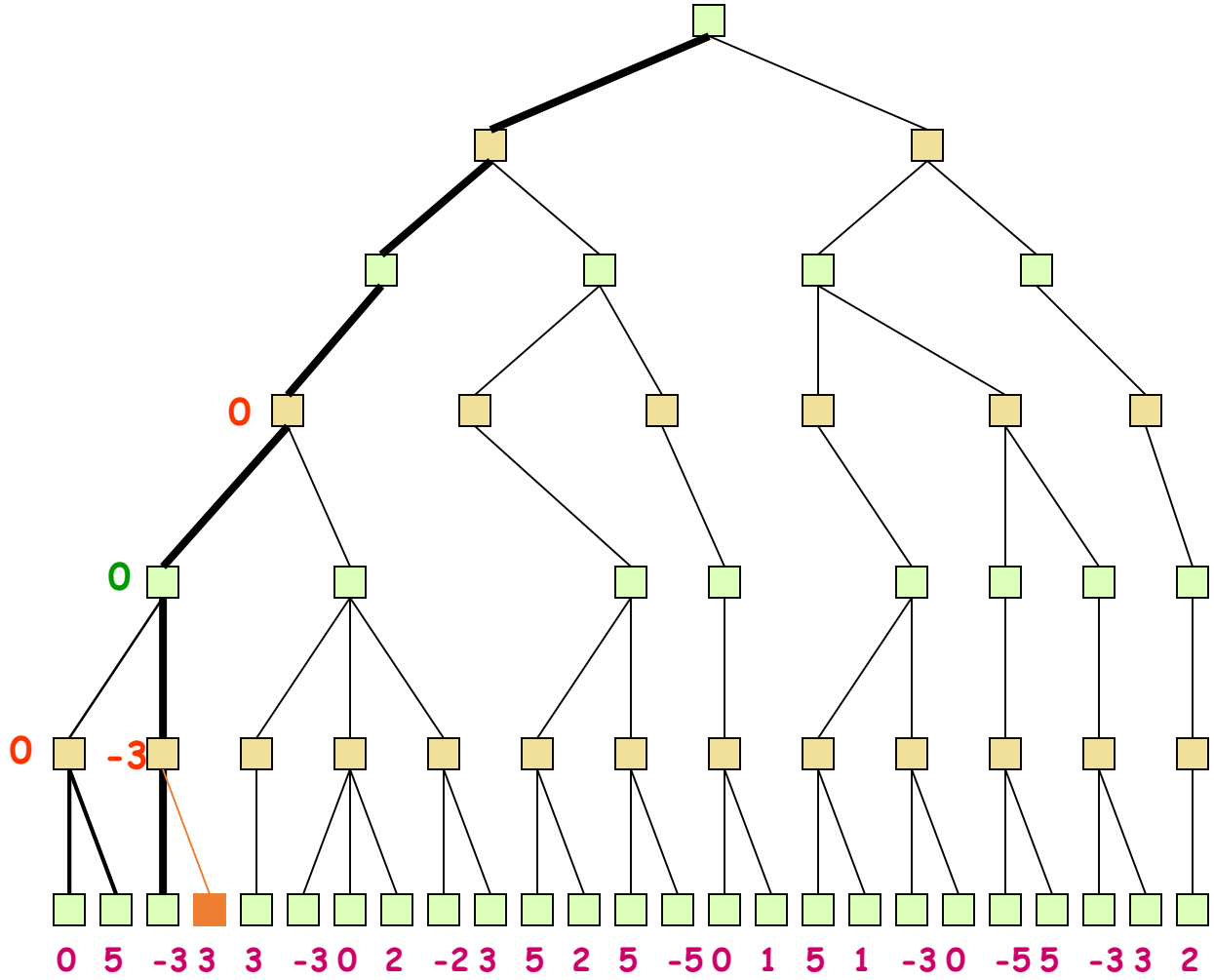
Example



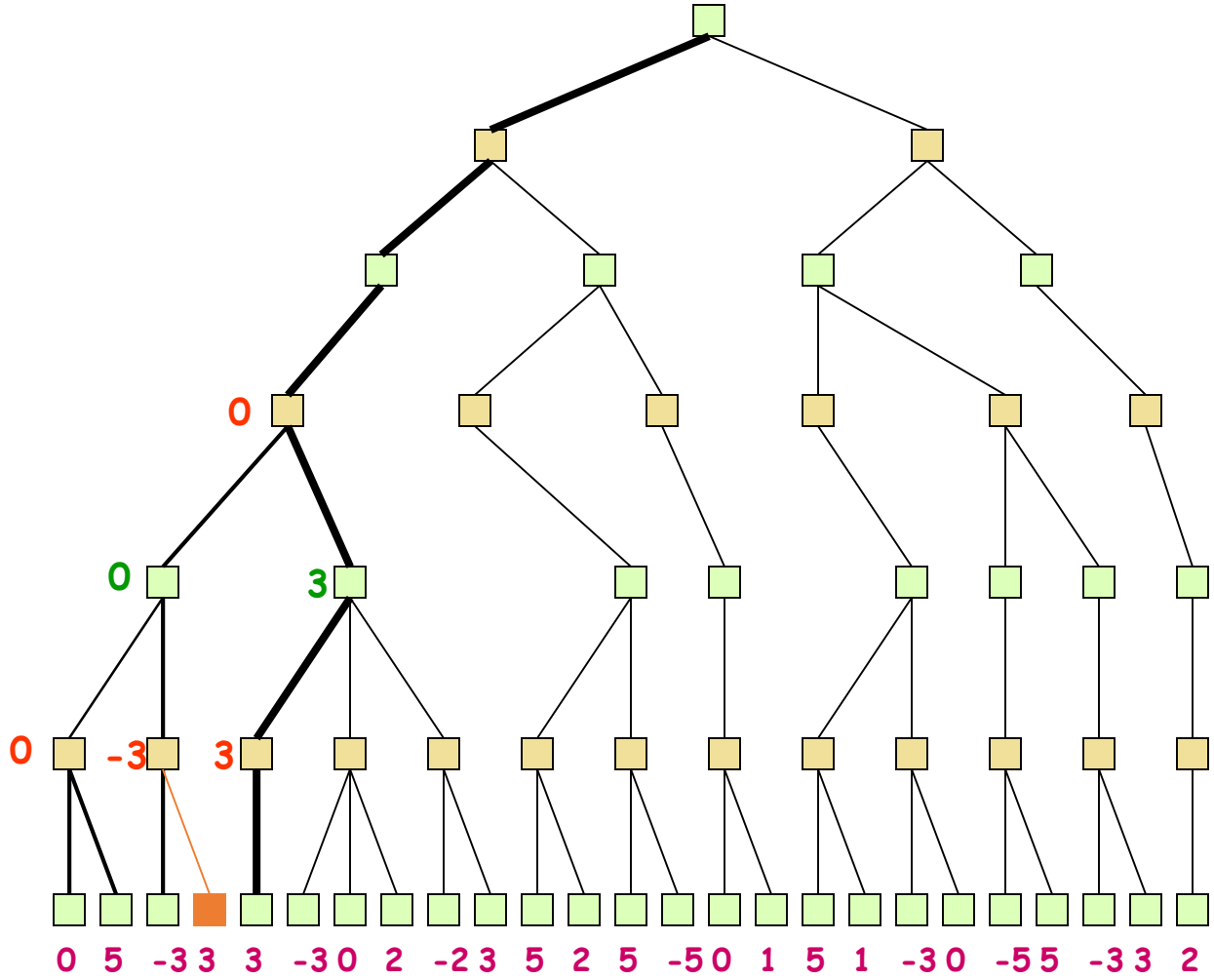
Example



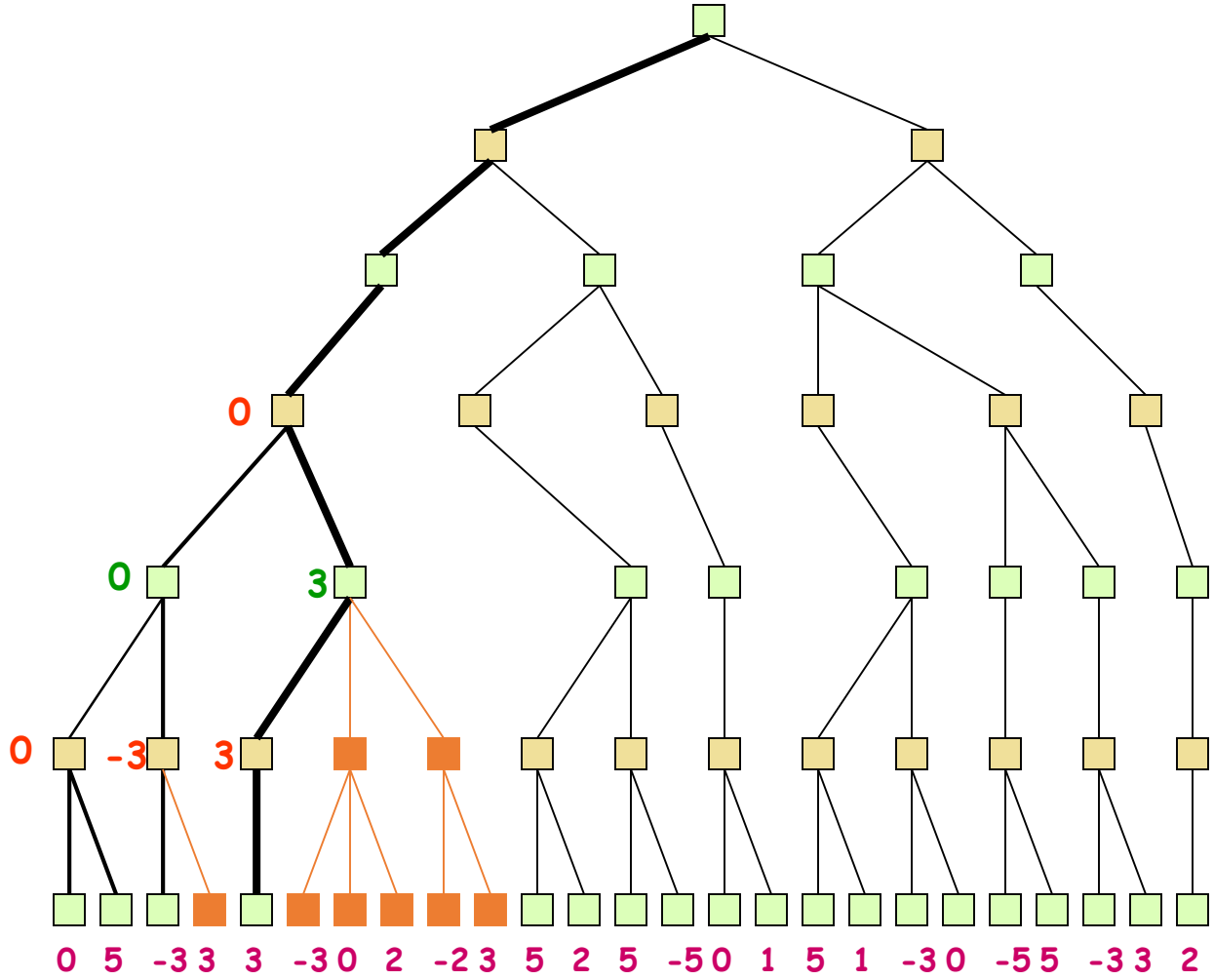
Example



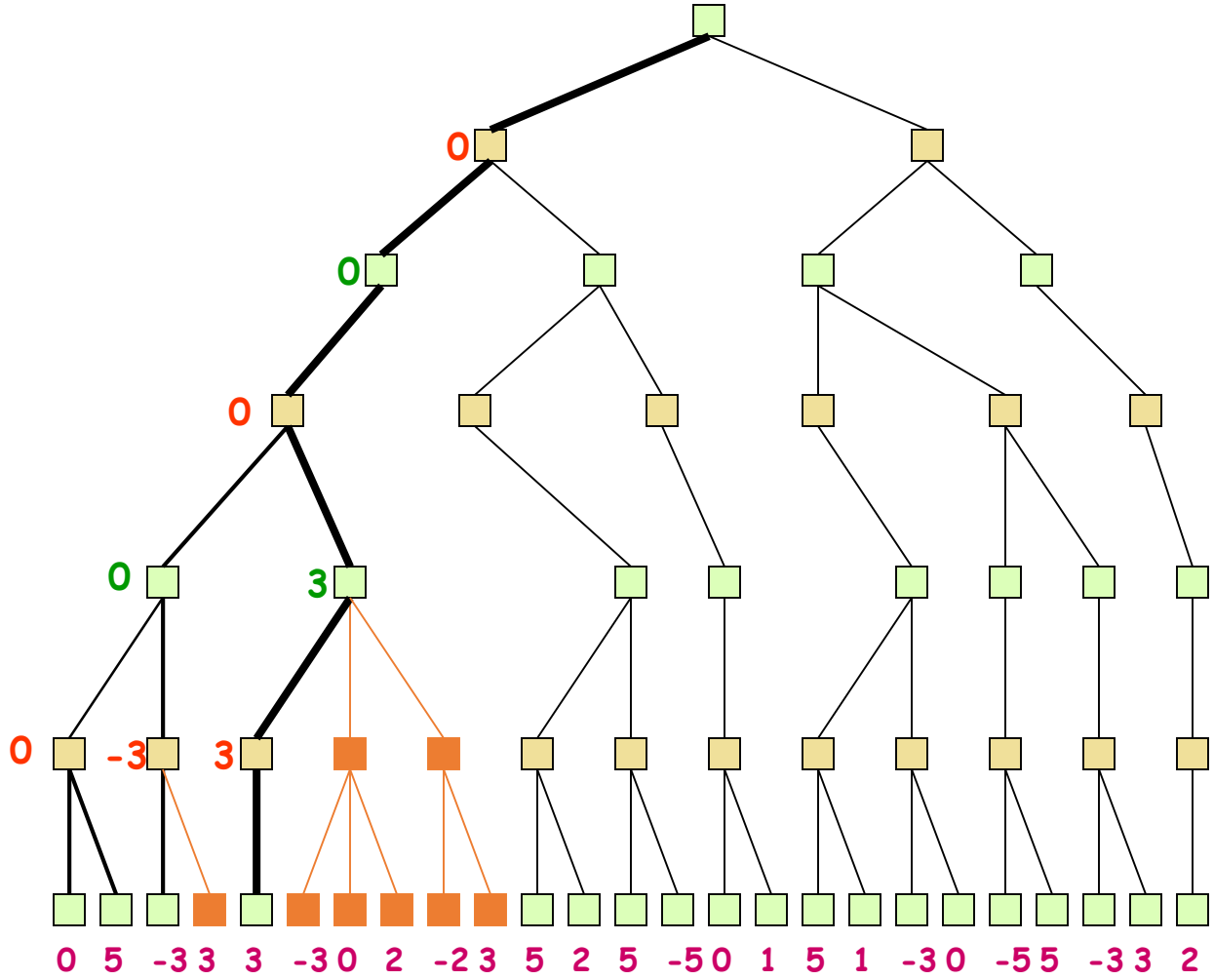
Example



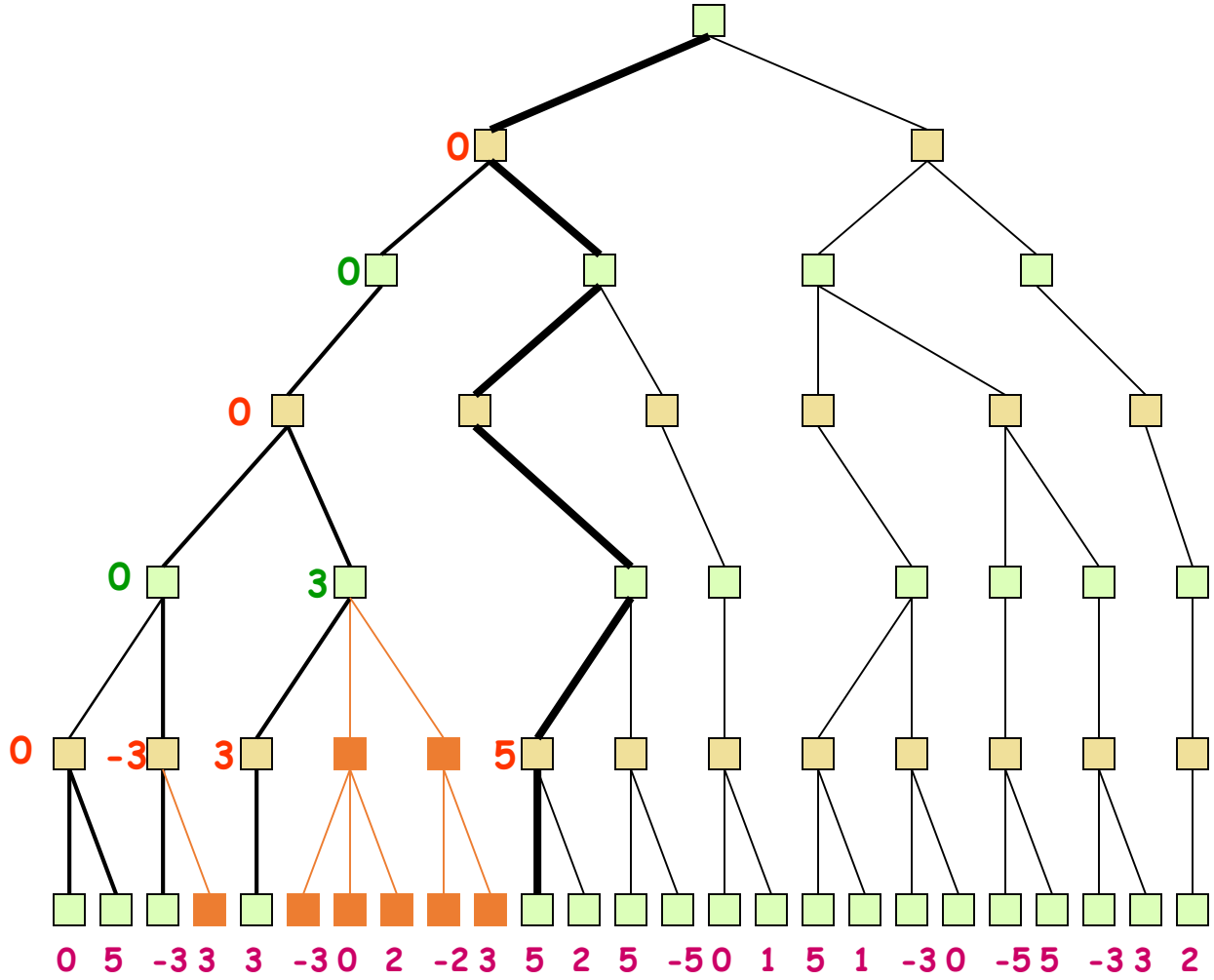
Example



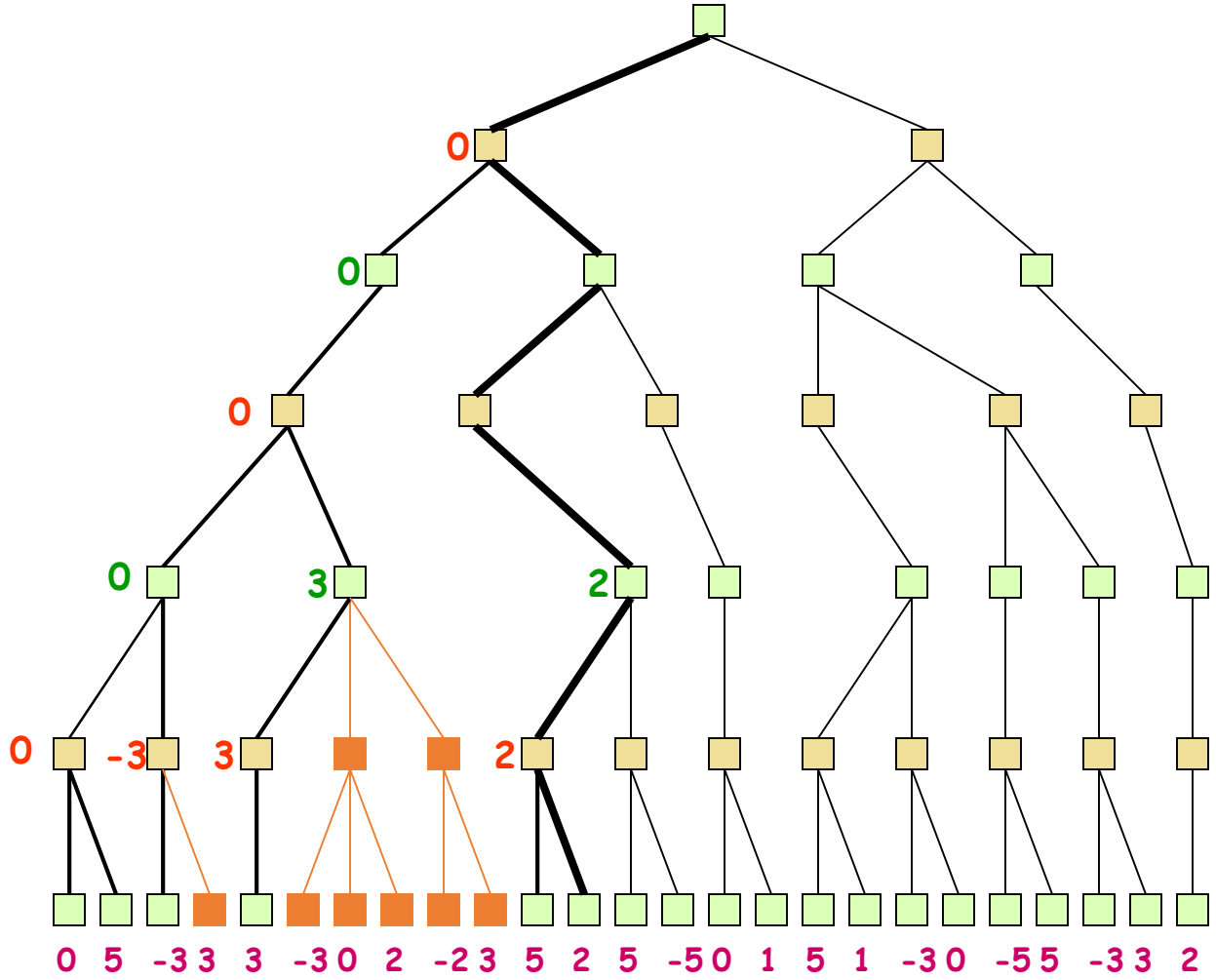
Example



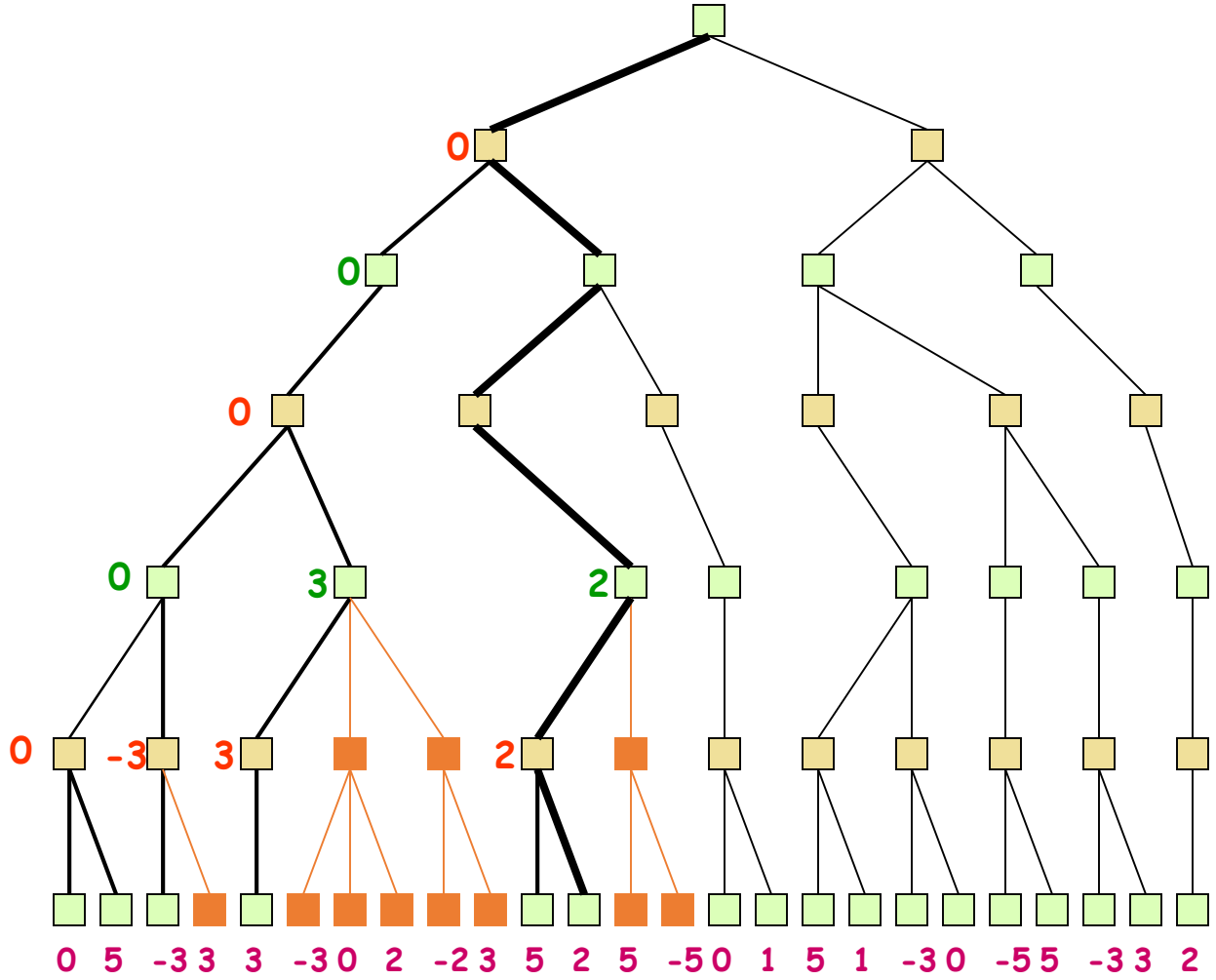
Example



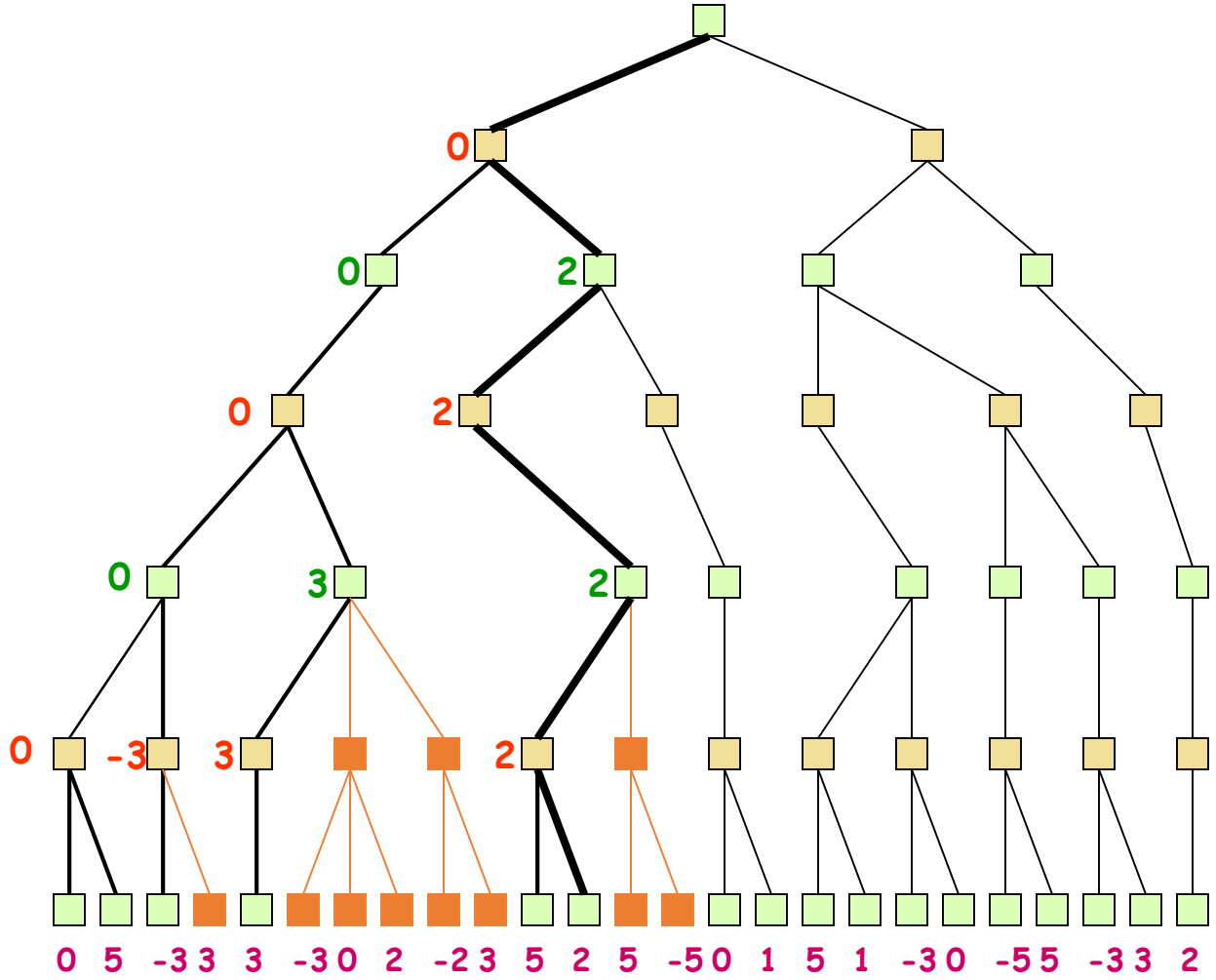
Example



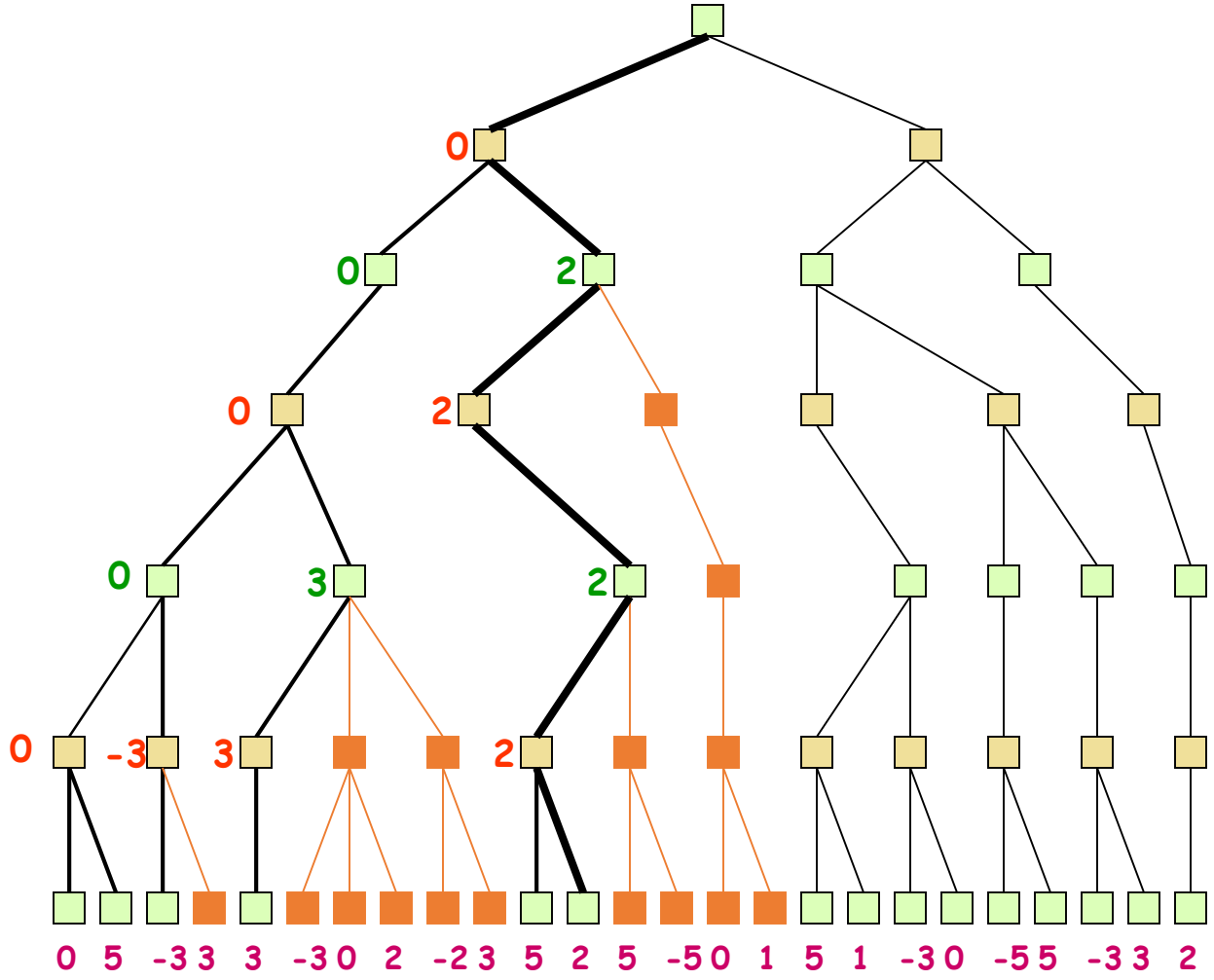
Example



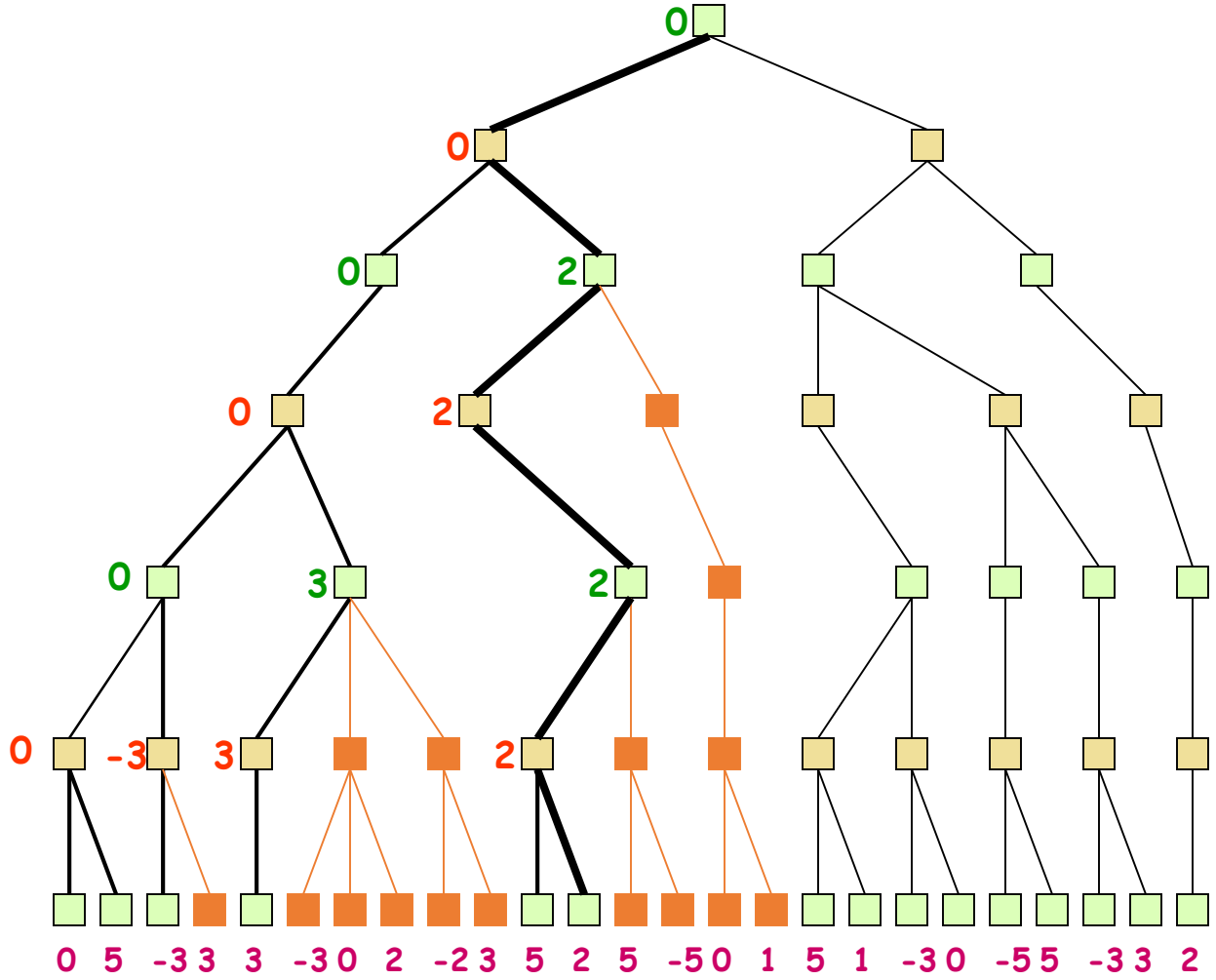
Example



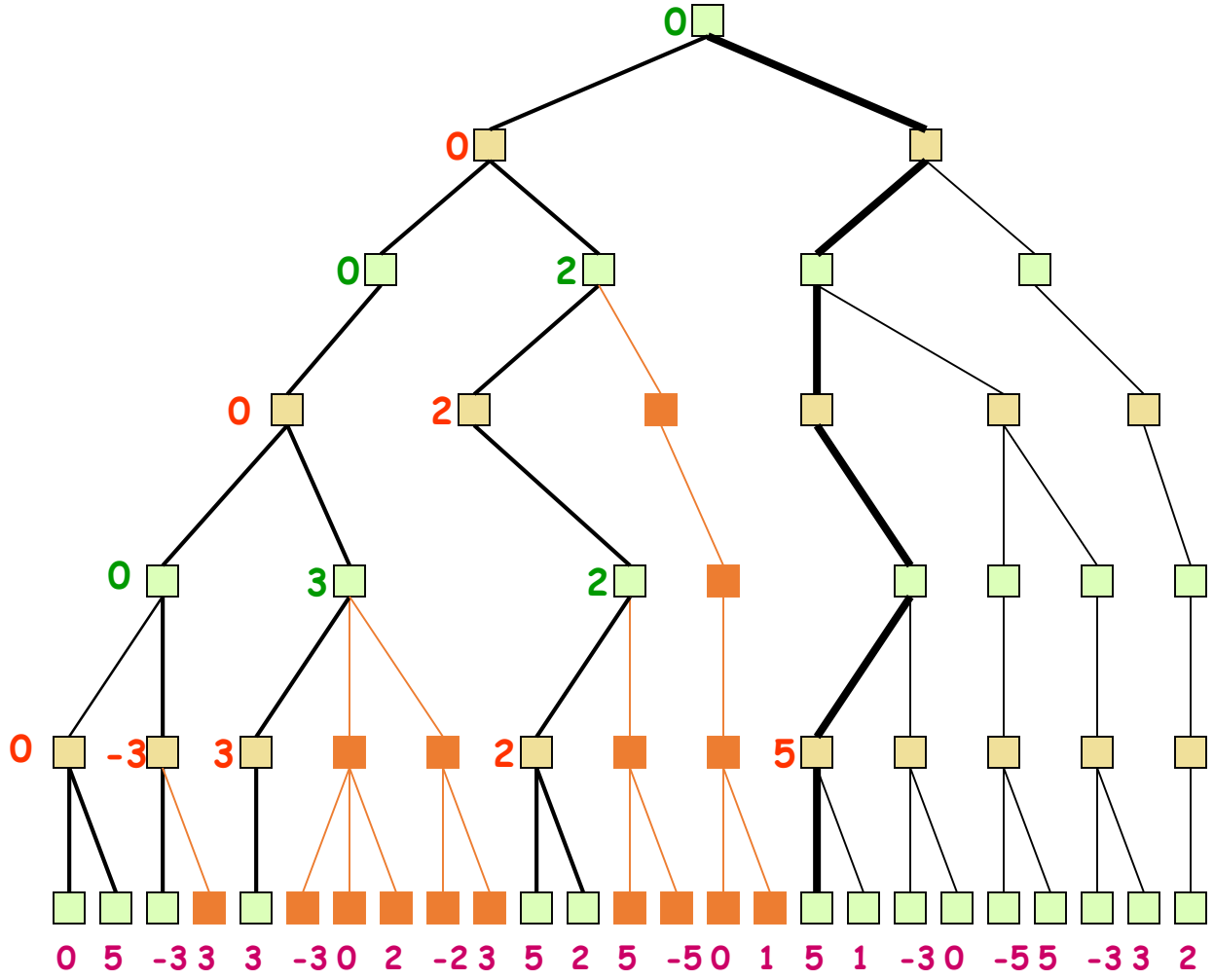
Example



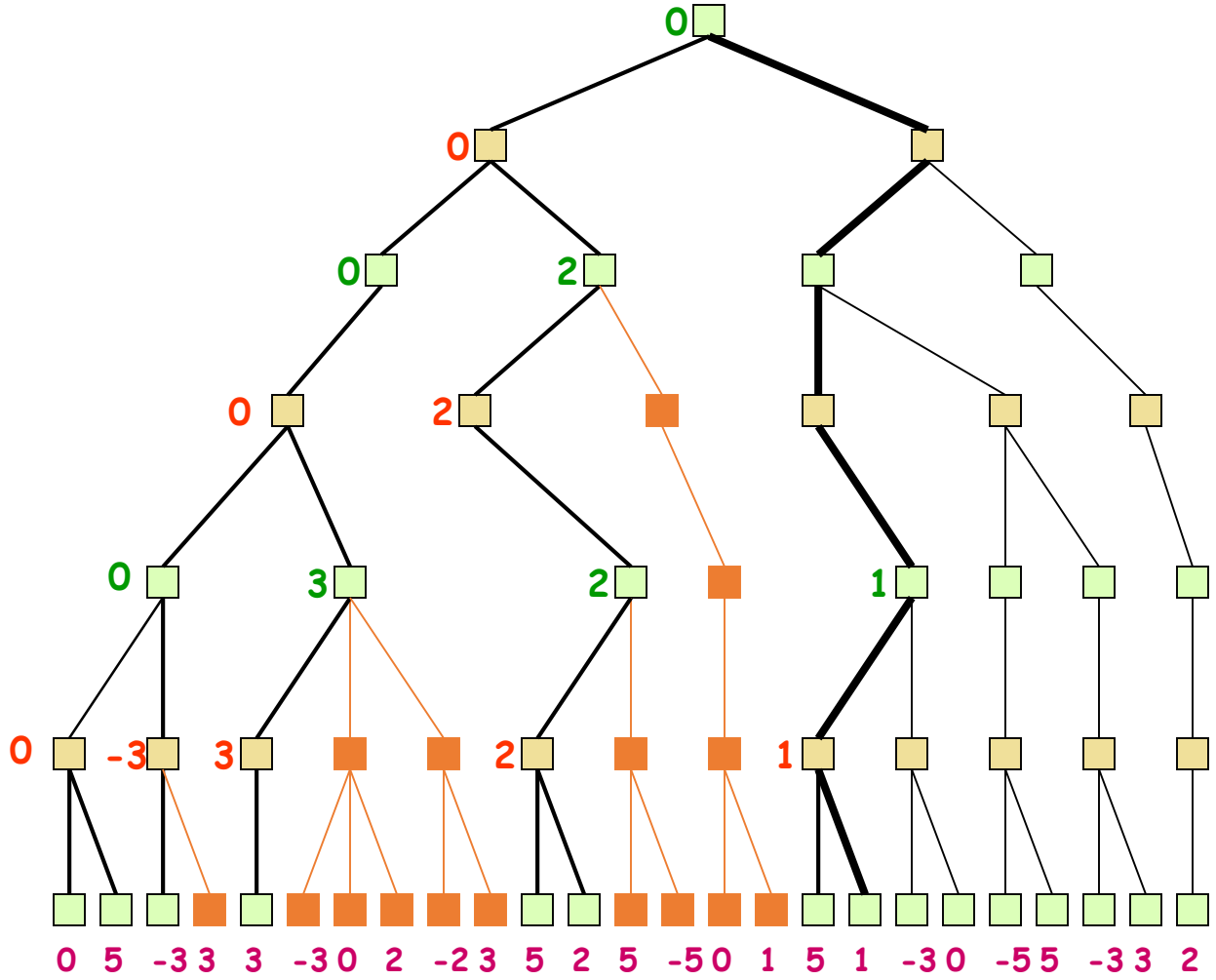
Example



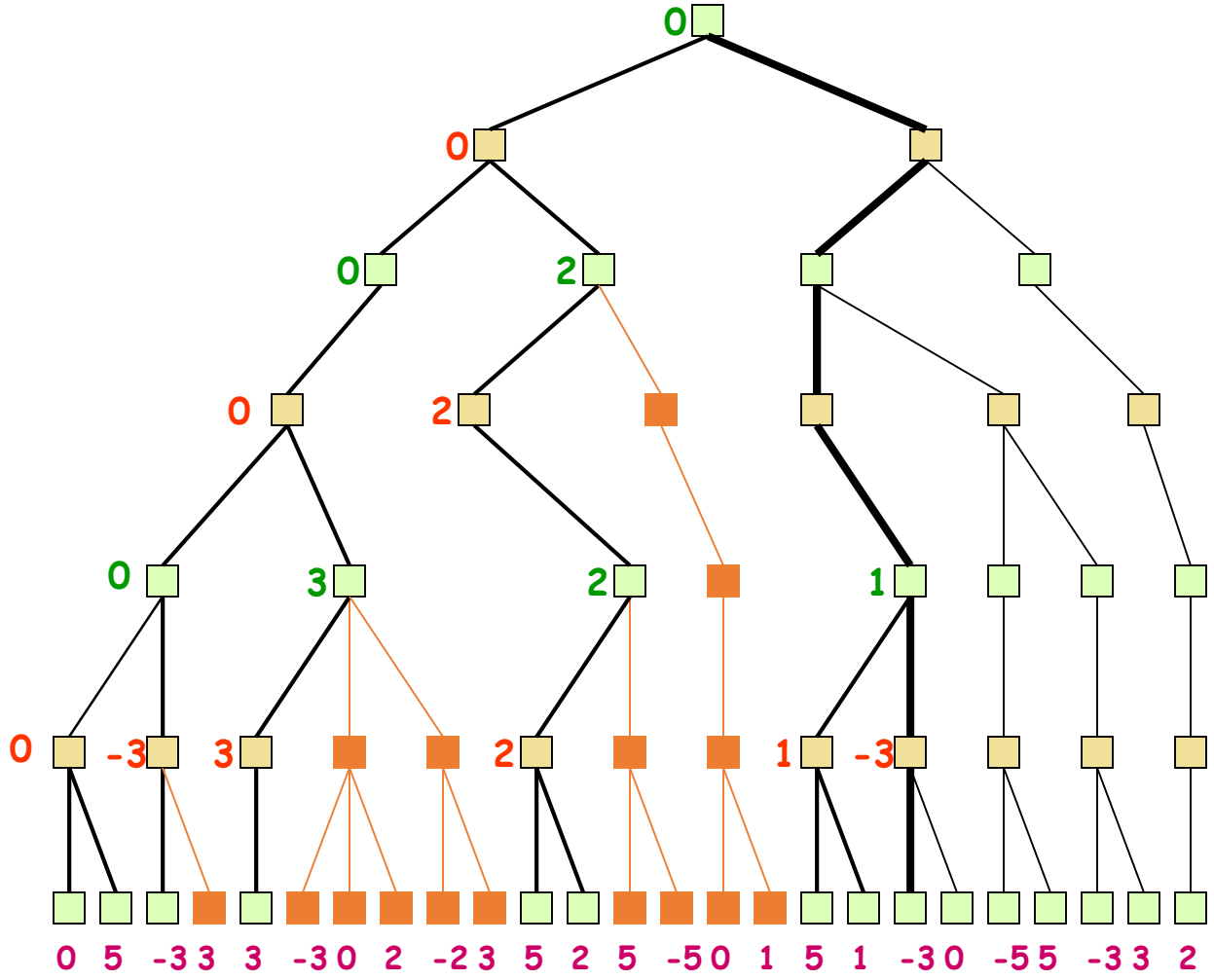
Example



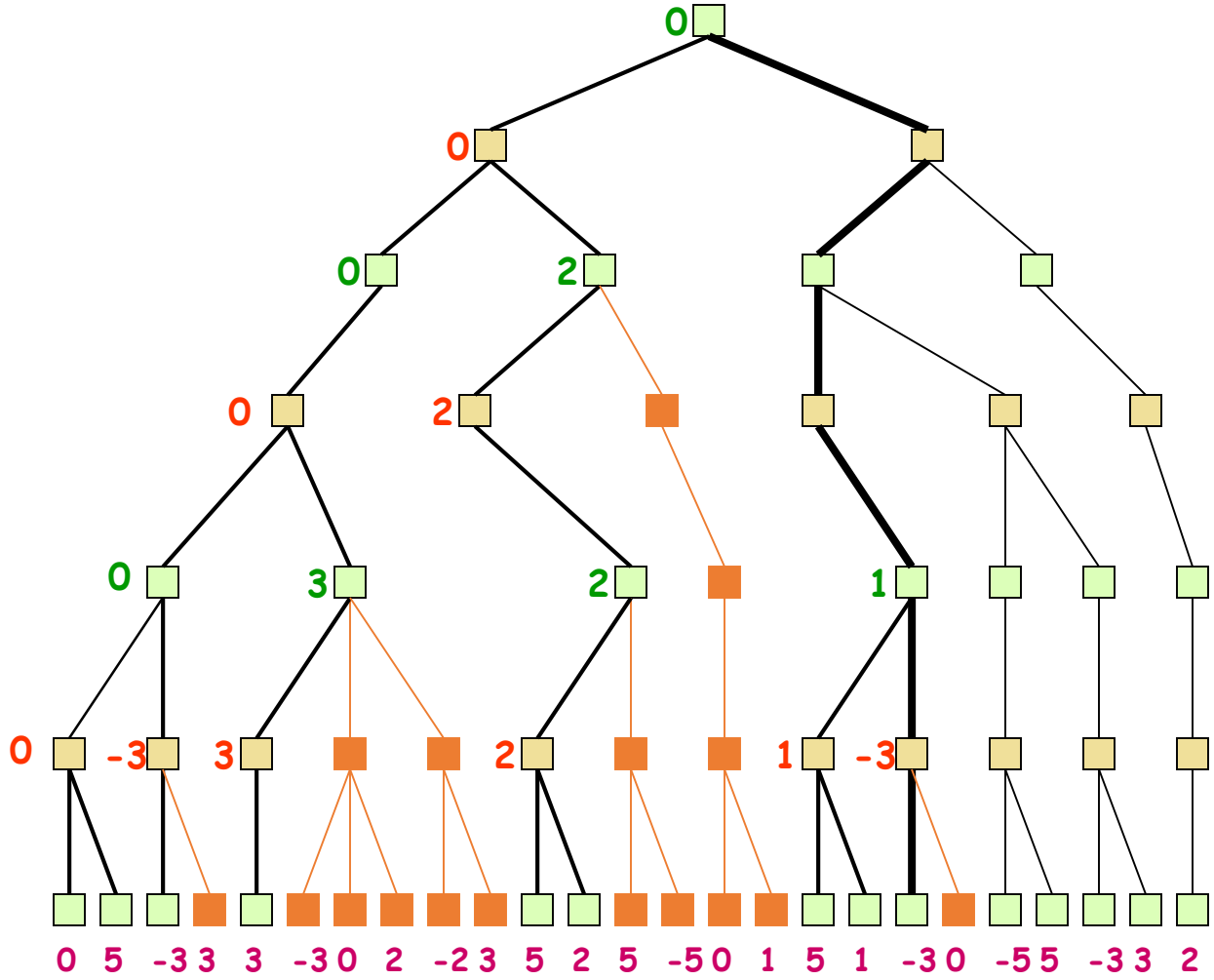
Example



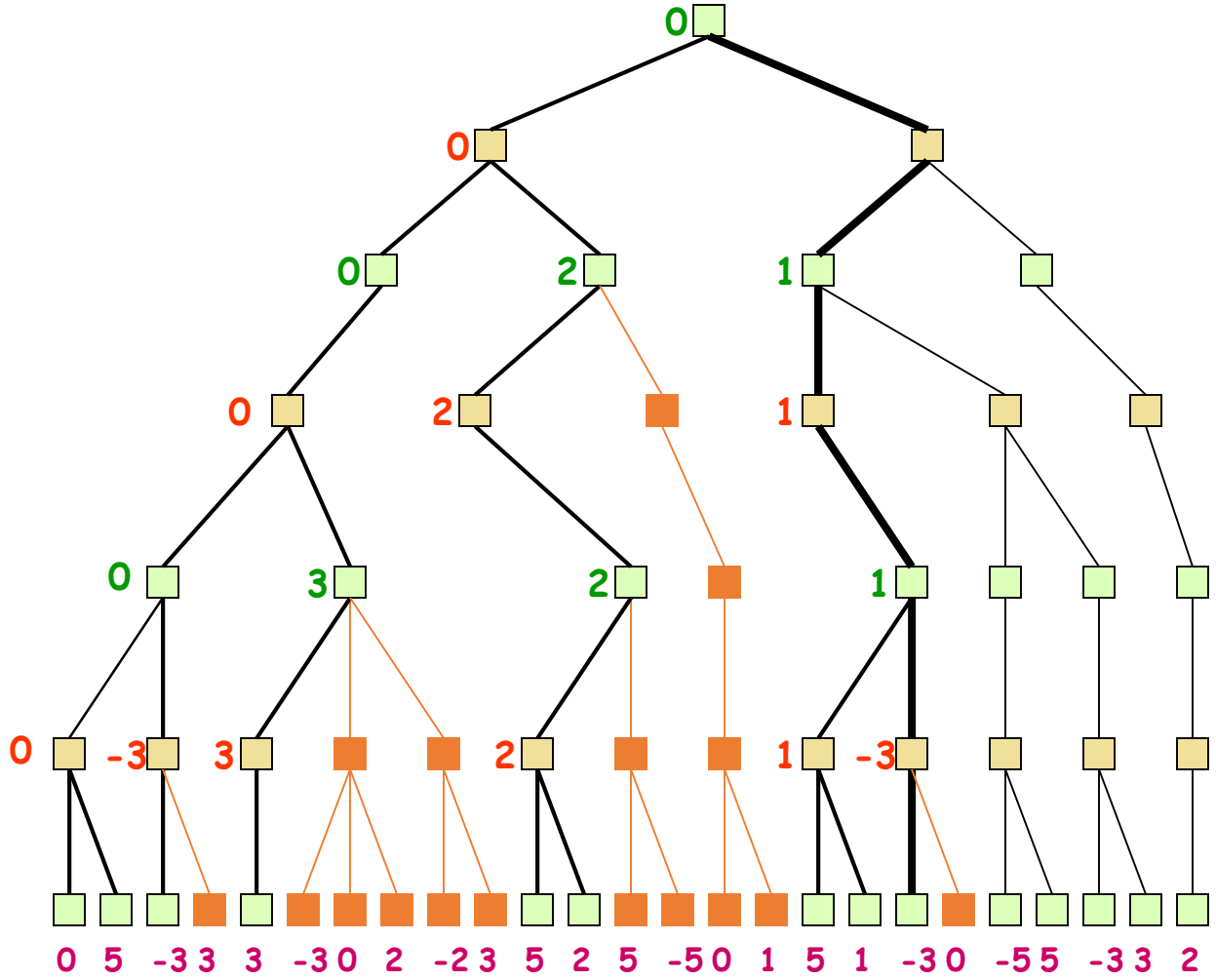
Example



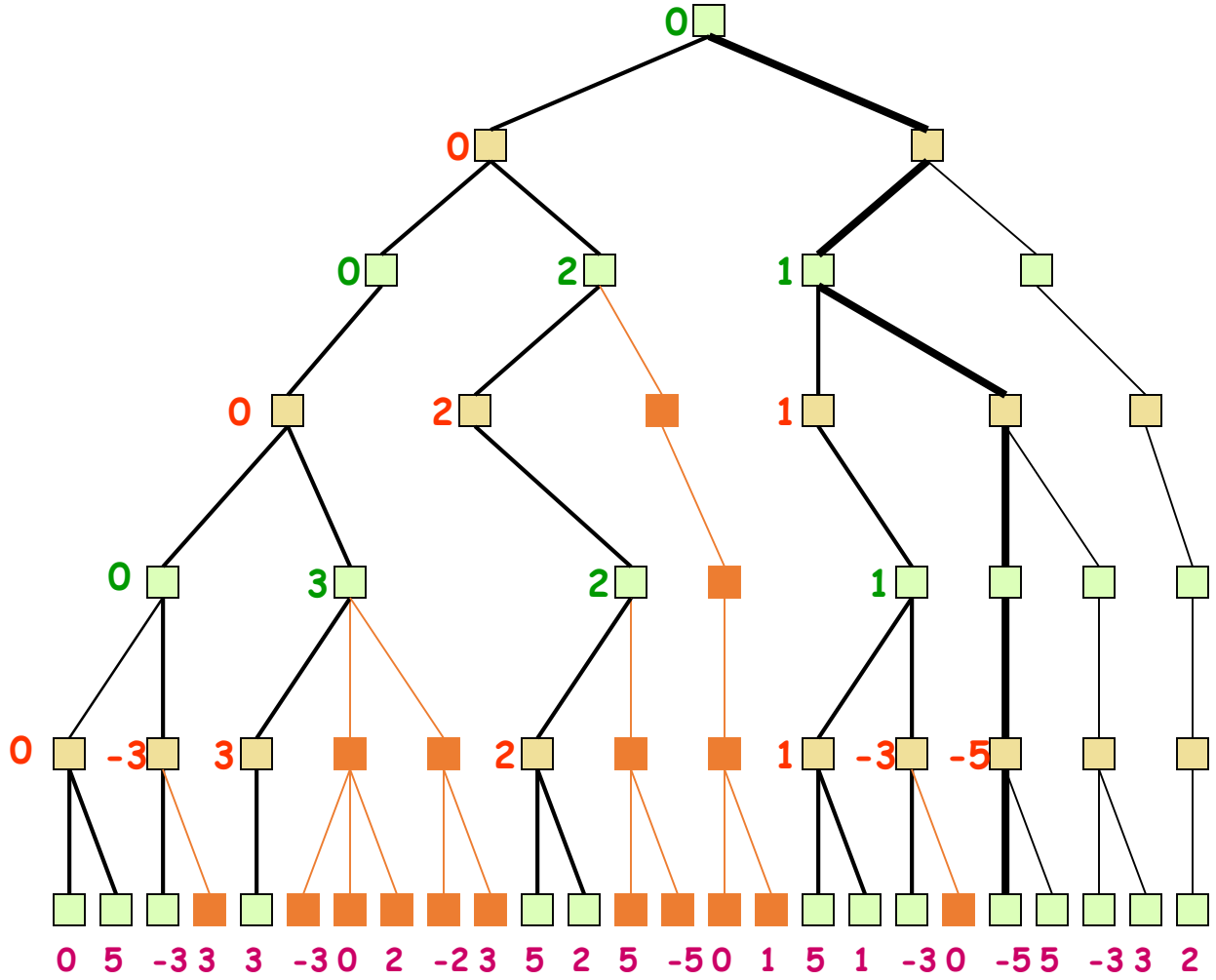
Example



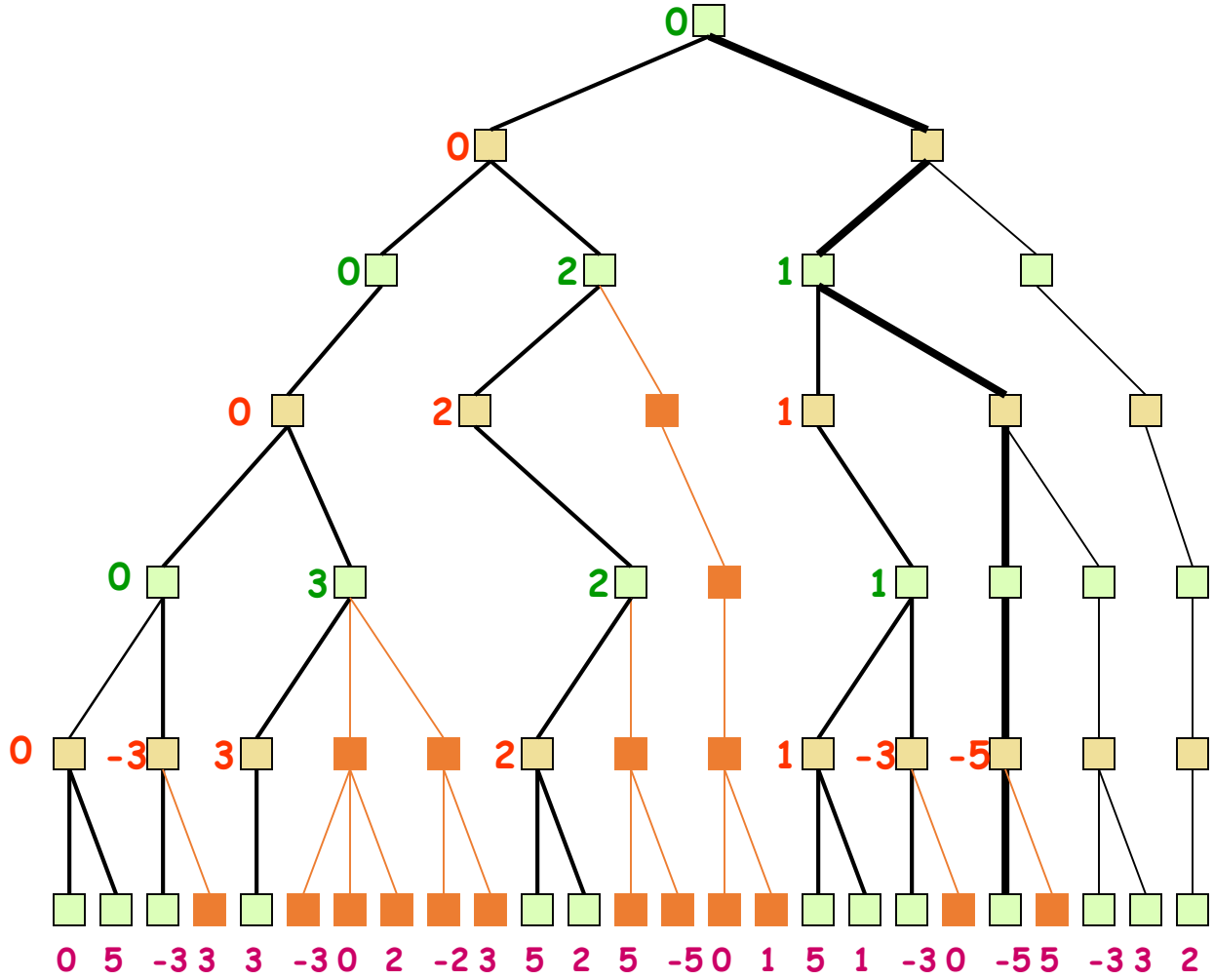
Example



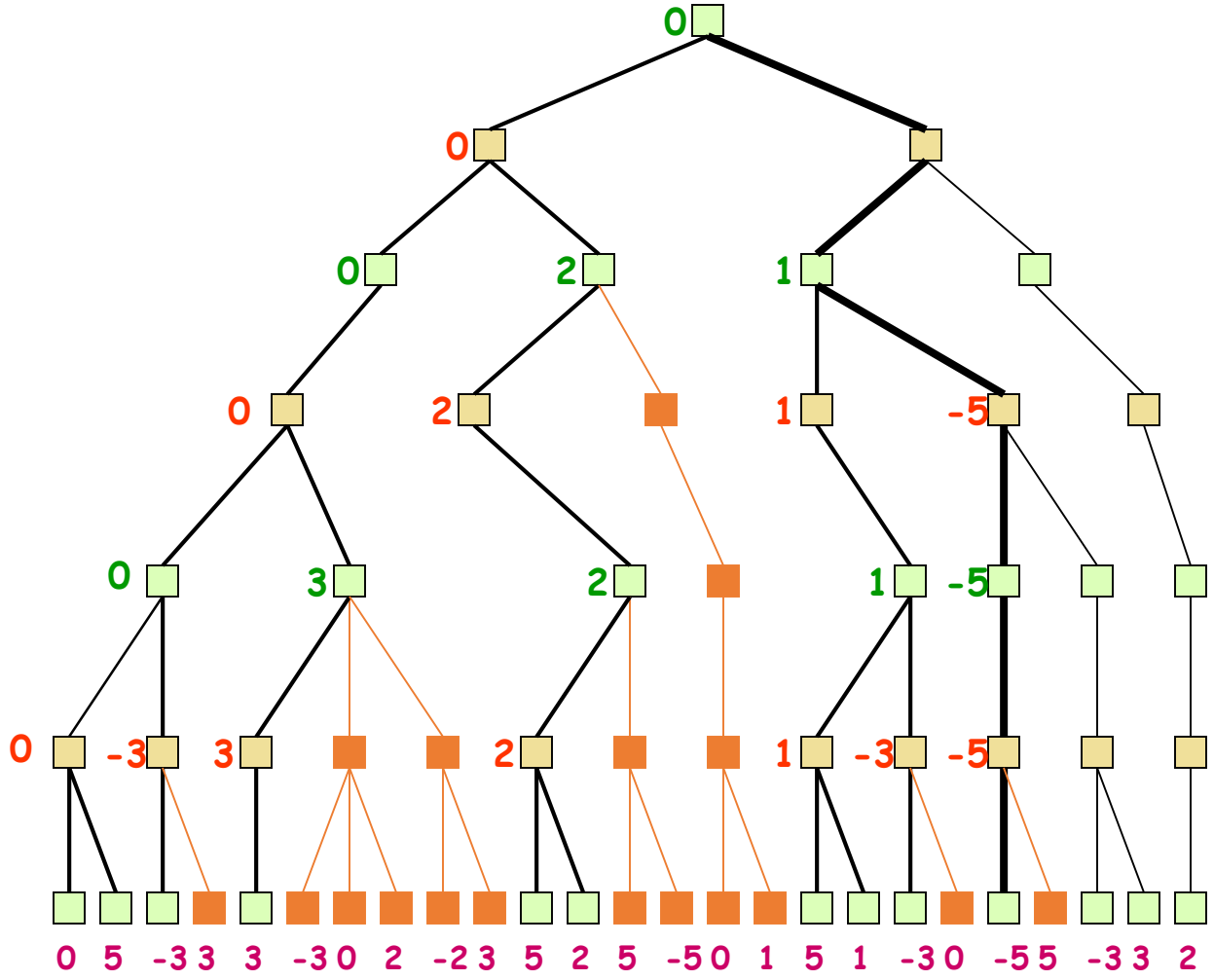
Example



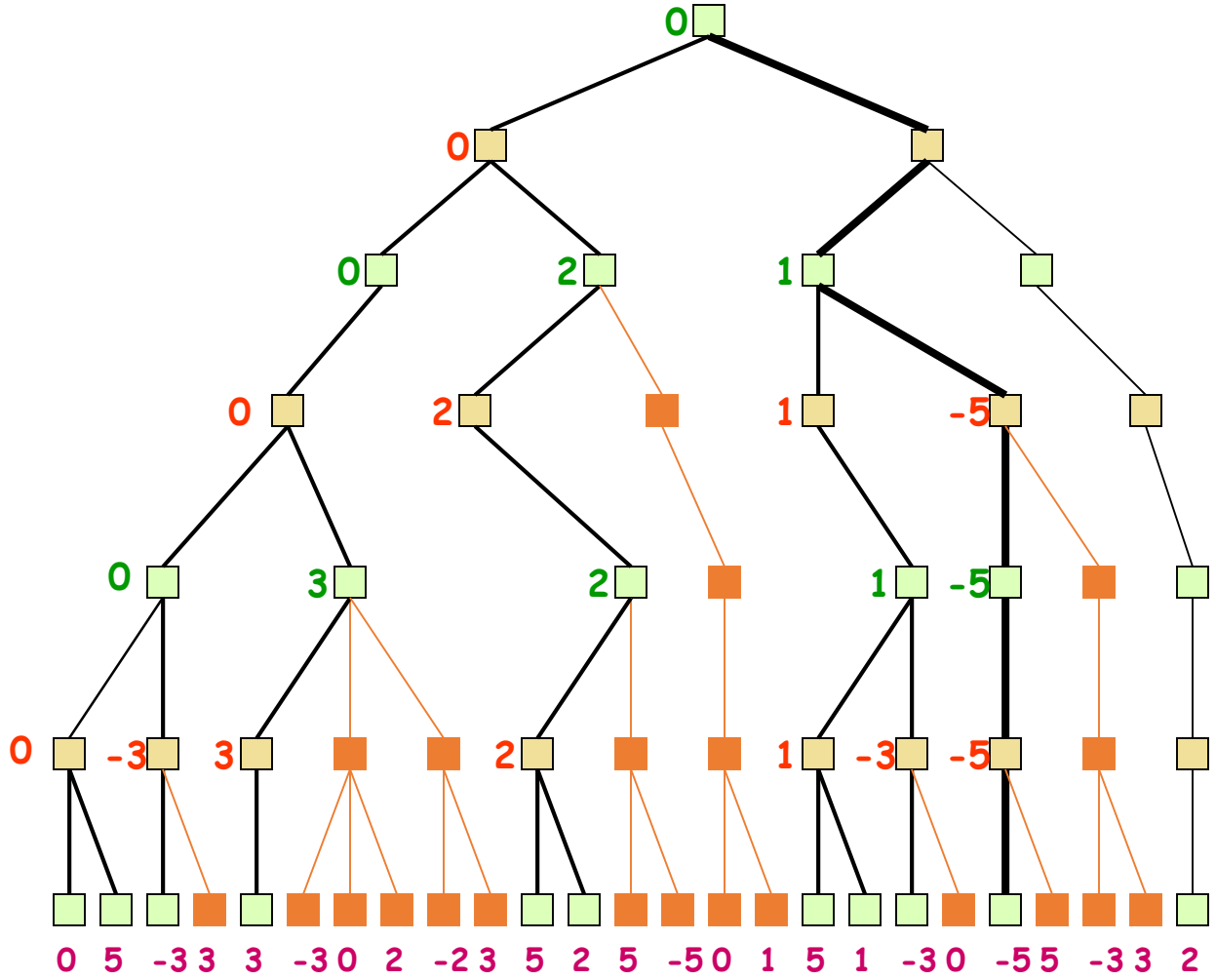
Example



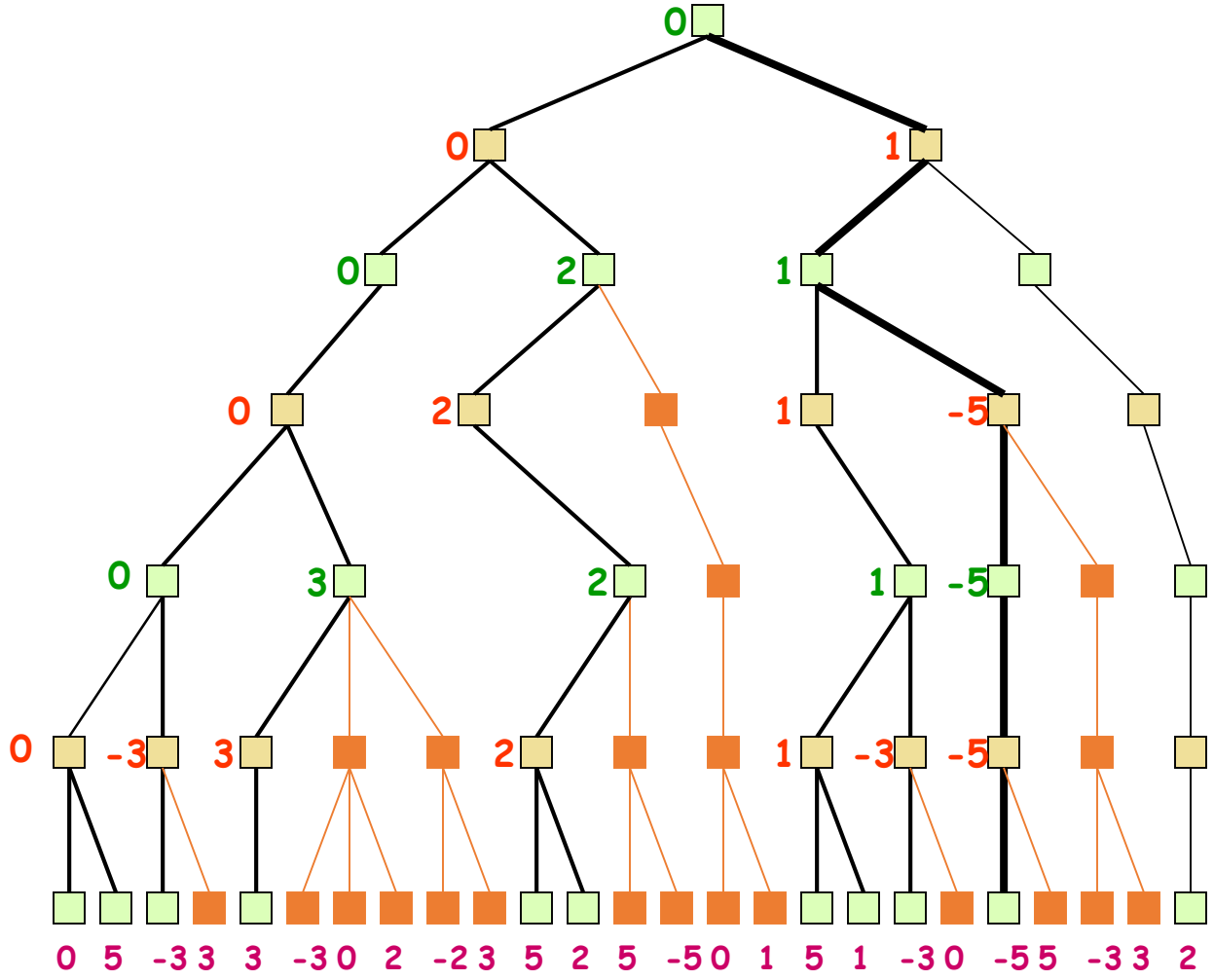
Example



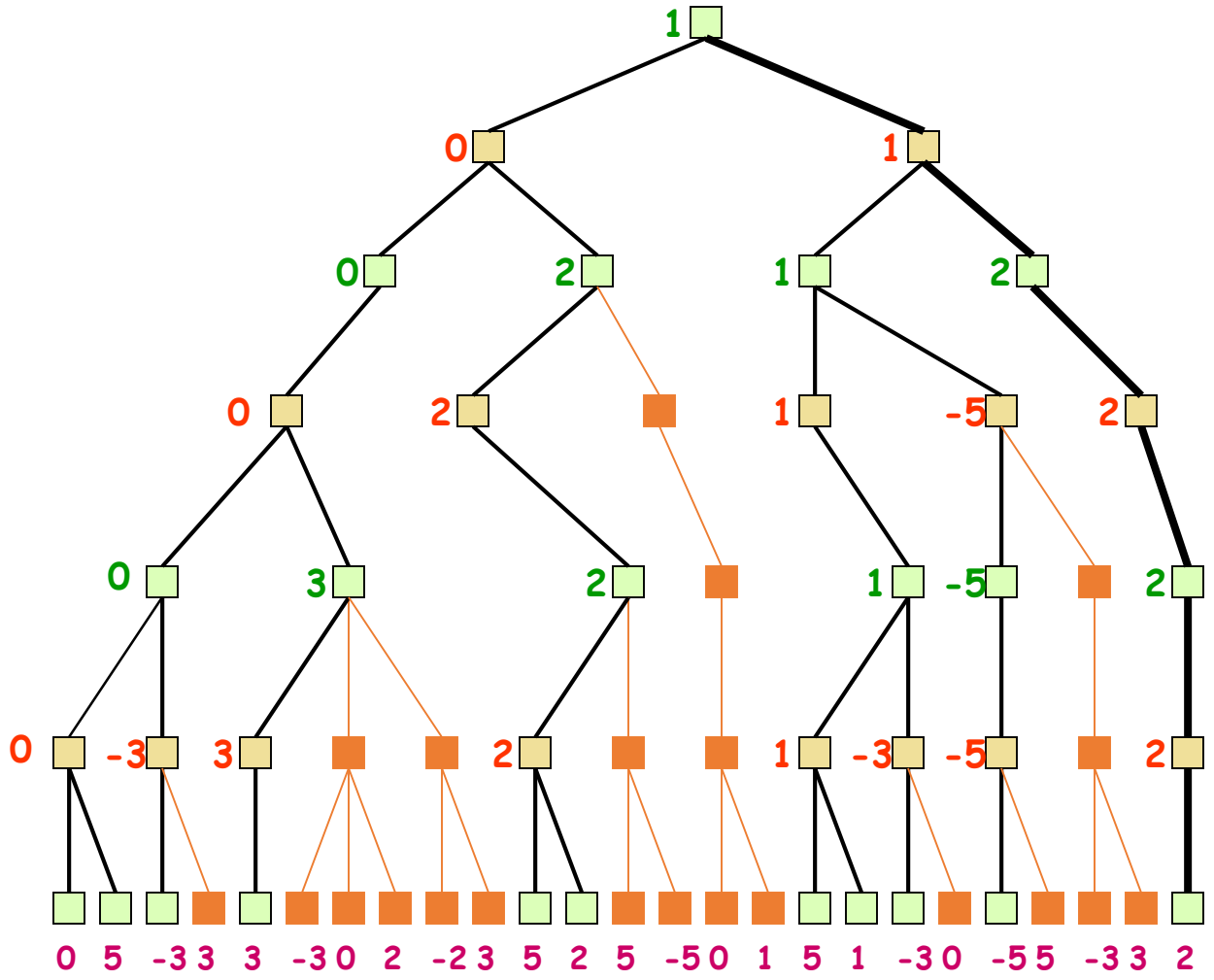
Example



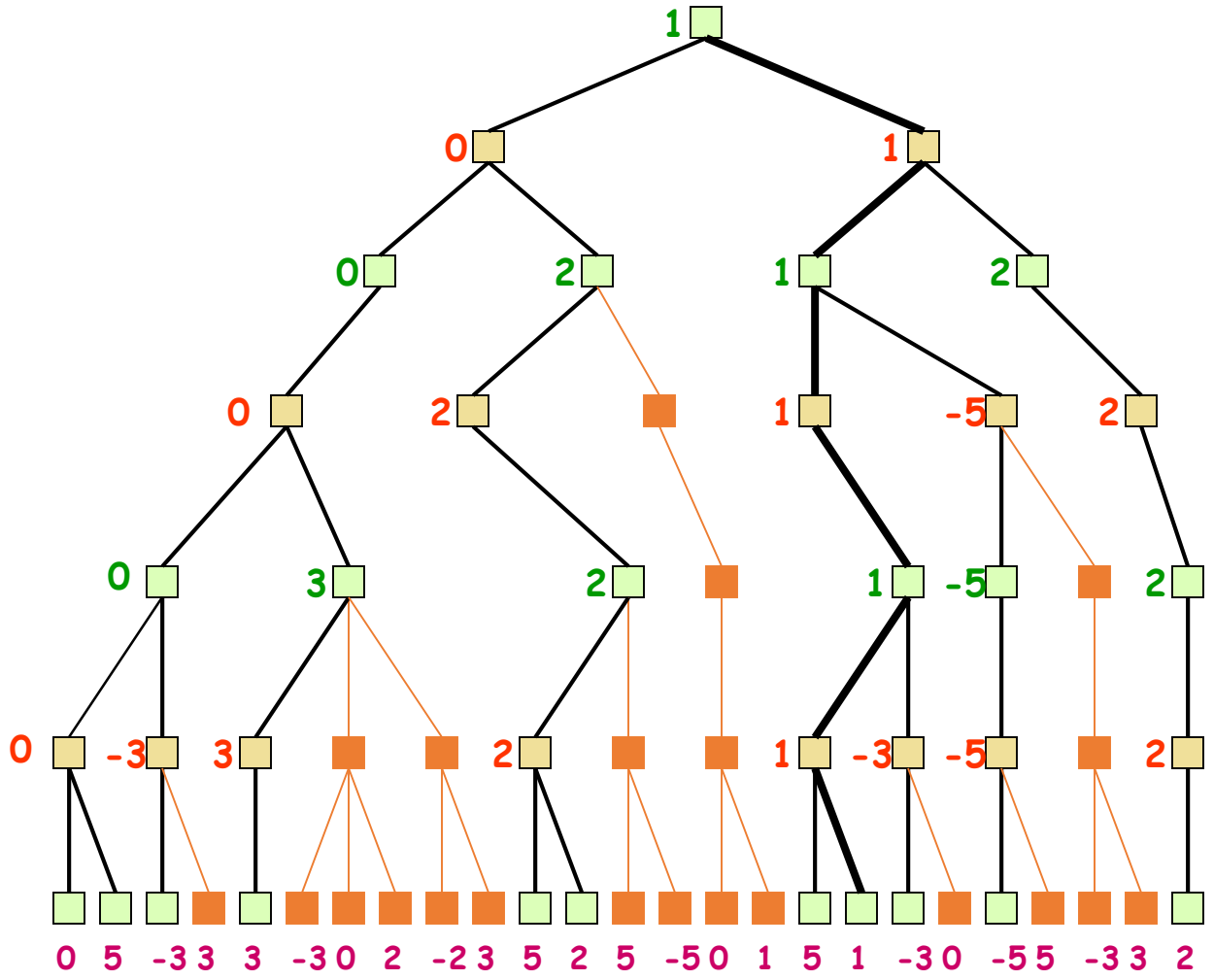
Example



Example

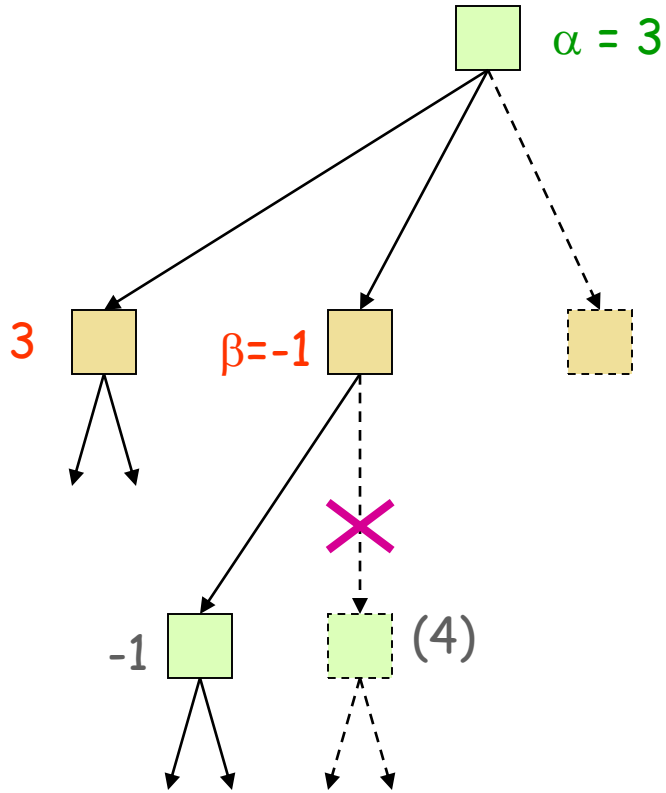

1
0 1
0 2 1 2
0 2 1 -5 2
0 3 2 1 -5 2
0 -3 3 2 1 -3 -5 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

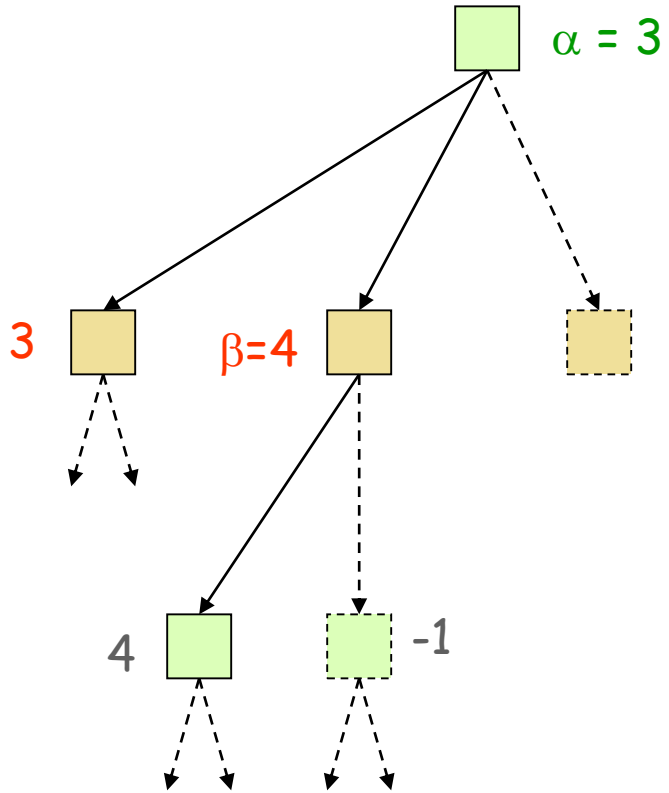
Example


1
0 1
0 2 1 2
0 2 1 -5 2
0 3 2 1 -5 2
0 -3 3 2 1 -3 -5 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

剪枝让我们得了多少好处？

考虑以下两种情况：





剪枝让我们得了多少好处？

- 假设博弈树有着均一的分支因子b
- 极大极小检查$O(b^h)$个节点，这也是$\alpha-\beta$的最坏情况
- $\alpha-\beta$ 带来的好处最大，当：
  - MAX节点的MIN子节点按照回推值递减顺序排列
  - MIN 节点的MAX子节点按照回推值递增顺序排列
- $\alpha-\beta$ 检查$O(b^{h/2})$个节点 [Knuth and Moore, 1975]
- 但这几乎不可能做到（如果能够把节点按上述规律进行排序，那么我们也就无需再对博弈树进行搜索了）
- 如果节点是随机的顺序排列，则$\alpha-\beta$ 检查的节点数平均为$\sim O(b^{3h/4})$

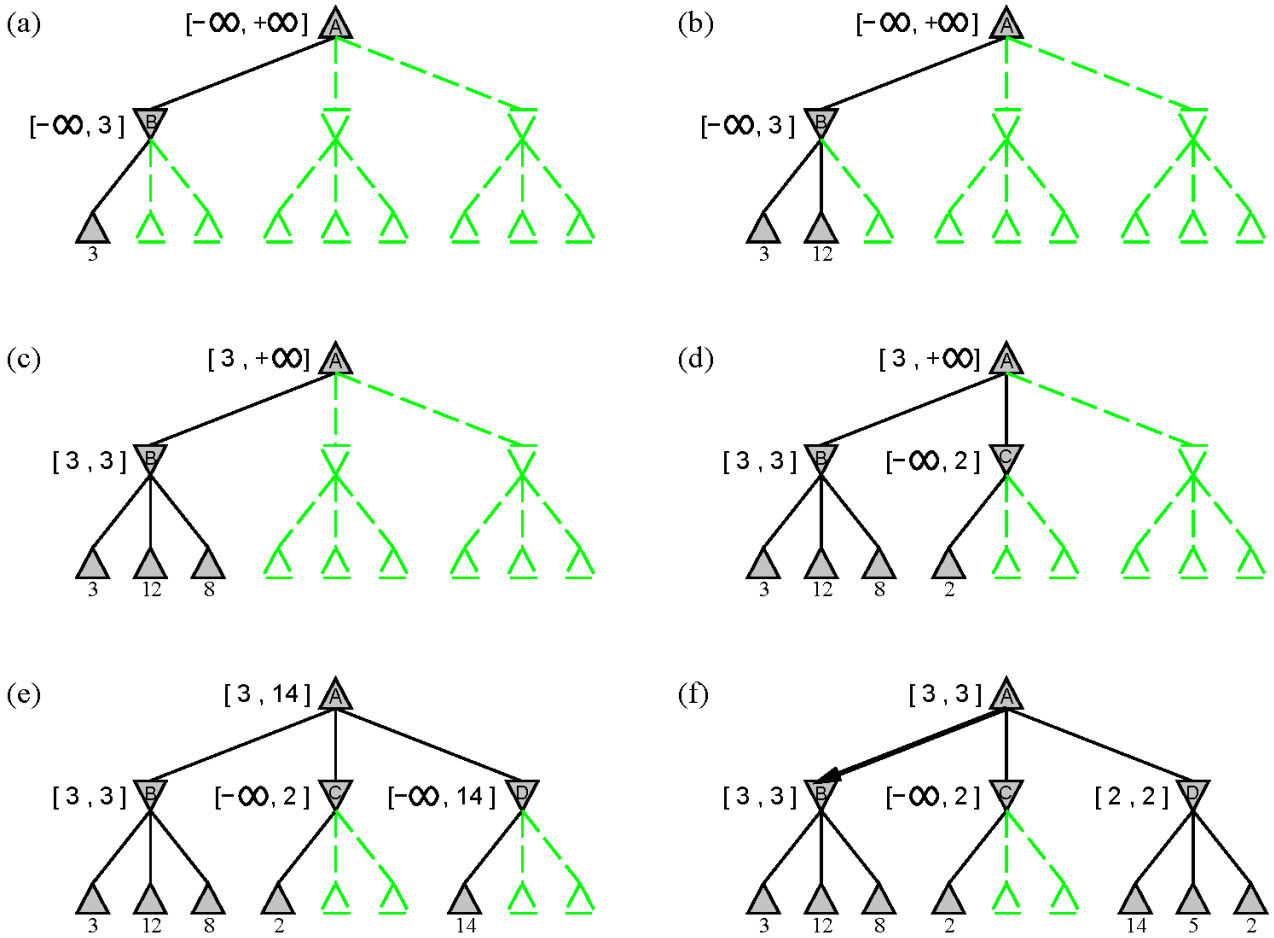
节点的启发式排序
Heuristic Ordering of Nodes

- 将根节点以下的节点按照前次迭代得到的回推值进行排序

其他一些提高性能的方法

- 自适应地平线（h）+ 迭代深度
- ……

例子


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3 , +∞]
A
[3 , 3]
B
3
12
8
(d)
[3 , +∞]
A
[3 , 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3 , 14]
A
[3 , 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3 , 3]
A
[3 , 3]
B
[−∞, 2]
C
[2 , 2]
D
3
12
8
2
14
5
2

α–β剪枝

- α＝到目前为止我们在路径上的任意选择点发现的MAX的最佳选择
- β＝到目前为止我们在路径上的任意选择点发现的MIN的最佳选择
- α–β剪枝的效率很大程度上取决于检查后继的顺序
  - 如果能够先检查那些可能最好的后继，那么算法的时间复杂度为$O(b^{d/2})$。

α–β剪枝算法

```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

处理搜索树中的重复状态

- 在游戏中重复状态的频繁出现往往是因为调换—导致同样棋局的不同行棋序列的排列
  - 例如，[a_1, b_1, a_2, b_2]和[a_2, b_2, a_1, b_1]都结束于同样的棋局。
- 解决办法：第一次遇见某棋局时将对它的评价存储在哈希表中（该哈希表称为调换表）。

第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

不完整的实时决策：对MINIMAX或α–β算法的改进

- α–β算法依然要搜索至少一部分空间直到终止状态，这样的搜索不现实。
- Shannon提出：
  - 用估计棋局效用值的启发式评价函数EVAL代替效用函数
  - 用可以决策什么时候运用EVAL的截断测试取代终止测试

不完整的实时决策

- 因此得到如下的启发式极小极大值，s为状态，d为最大深度：

H-MINIMAX(s,d)=

$$\begin{cases} EVAL(s) & \text{如果} CUTOFF-TEXT(s,d)\text{为真} \\ \max_{a\in Actions(s)} H-MINIMAX(RESULT(s,a),d+1) & s\text{为MAX结点} \\ \min_{a\in Actions(s)} H-MINIMAX(RESULT(s,a),d+1) & s\text{为MIN结点} \end{cases}$$

如何设计评价函数？

- 应该以和真正的效用函数同样的方式对终止状态进行排序
- 评价函数的计算不能花费太多的时间
- 对于非终止状态，评价函数应该和取胜的实际机会密切相关。

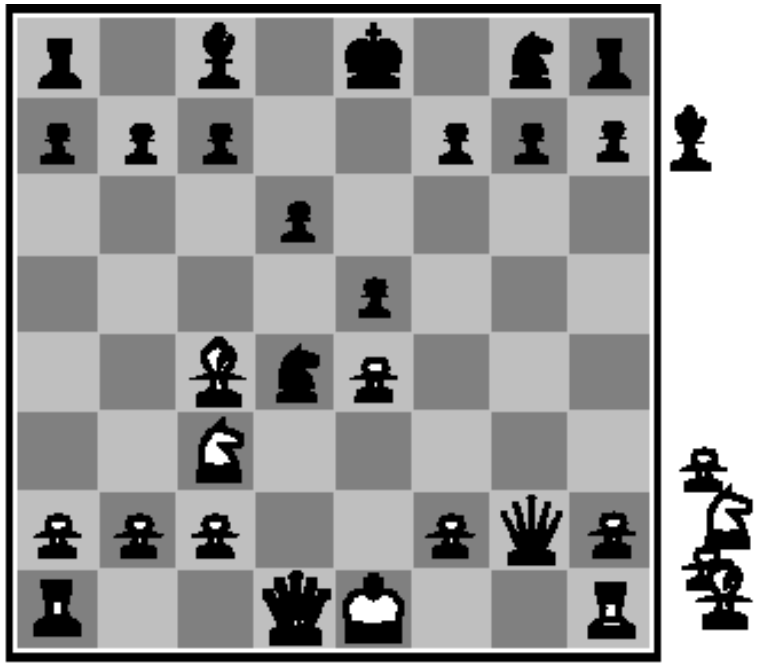
评价函数（续）

- 在计算能力有限情况下，评价函数能做到最好的就是猜测最后的结果
- 大多数评价函数的工作方式是计算状态的不同，那么对状态的一个合理评价
  是加权平均值
  - 例如，国际象棋中EVAL通常取为加权线性函数（假设每个特征的贡献独立于其它特征的值），

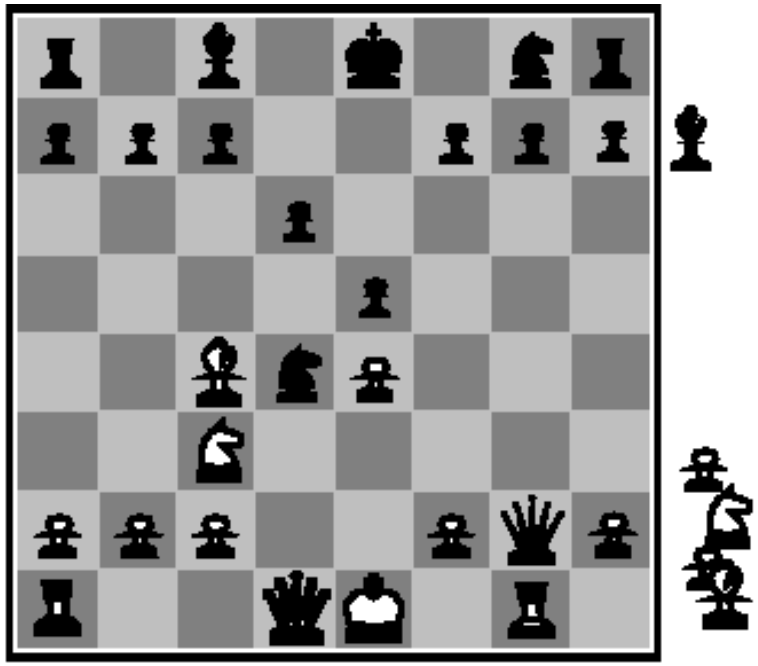
$$\sum_{i=1}^{n} w_i f_i(s)$$

截断搜索

- 用if CUTOFF-TEST(state, depth) then return EVAL(state)代替α–β算法中TERMINAL-TEST所在的两行，或
- 使用迭代搜索：当时间用完时，程序就返回目前完成最深的搜索所选择的招数。
  - 由于评价函数的近似性，上述方法可能导致错误。



(a) White to move



(b) White to move

（a）黑棋有1个马、2个 兵的优势，能够取胜。

（b）黑棋会被白棋吃掉皇后，从而失败。

截断搜索（续）

- 需要更为复杂的截断测试，评价函数应该只用于那些静止的棋局（静态棋局）。
- 非静止的棋局可以进一步扩展直到静止的棋局，这种额外的搜索称为静态搜索。

截断搜索（续）

- 地平线效应：指对手招数导致我方严重损失并且从理论上基本无法避免。
- 单步延伸是避免地平线效应的一种策略

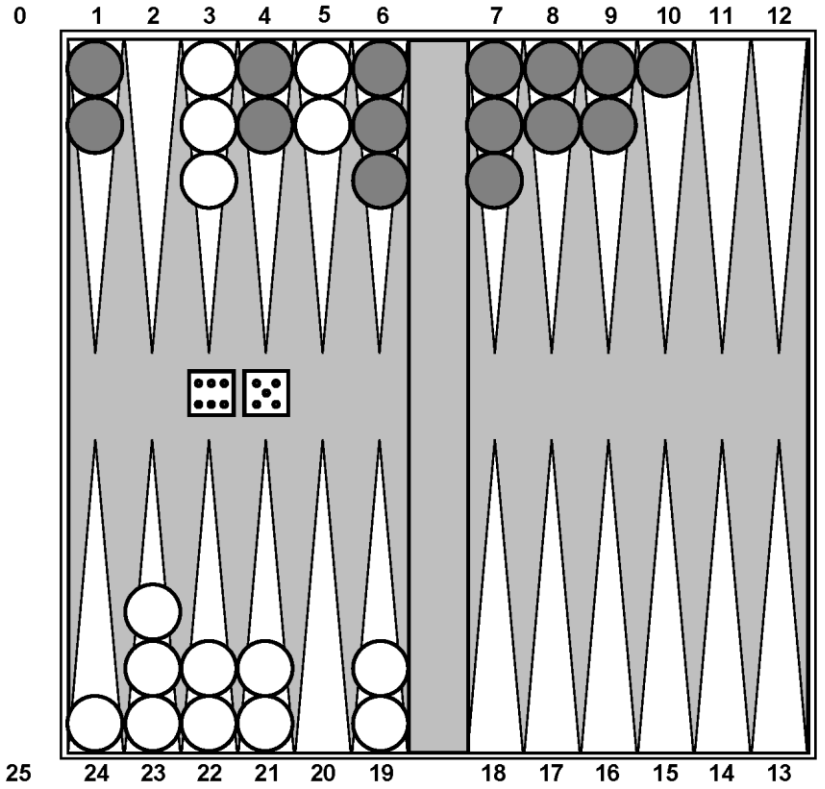
这样做可能超过深度限制，但由于单步延伸很少，不会增加太多开销。

第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

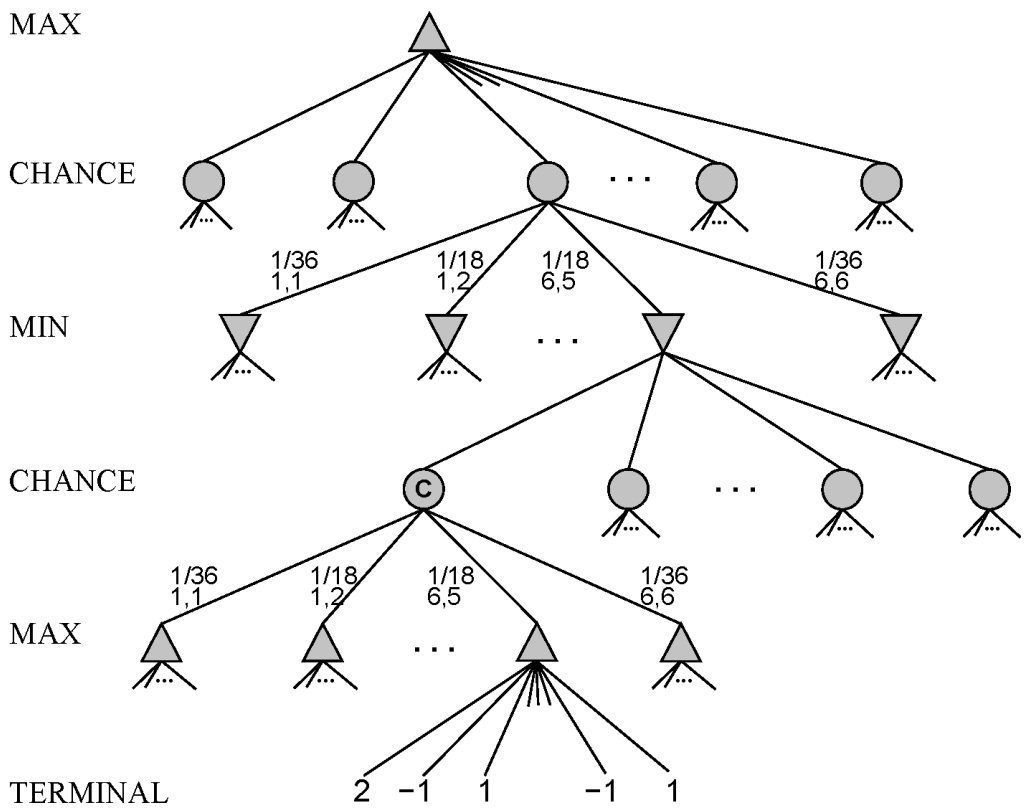
例子：西洋双陆棋

- 目标：将自己的棋子全部移出棋盘
- 白棋顺时针向25移动
- 黑棋逆时针向0移动
- 每个棋子按照掷出的骰子数移动到任意位置除非那里有多个对方棋子，如果只有一个，这个棋子就被吃了，要重新开始。



双陆棋的博弈树（包括几率节点）

- 两个骰子总共有21种骰法，其中：
  - 6个有相同骰子数的组合（1/36概率）
  - 15个不同骰子数的组合（1/18概率）

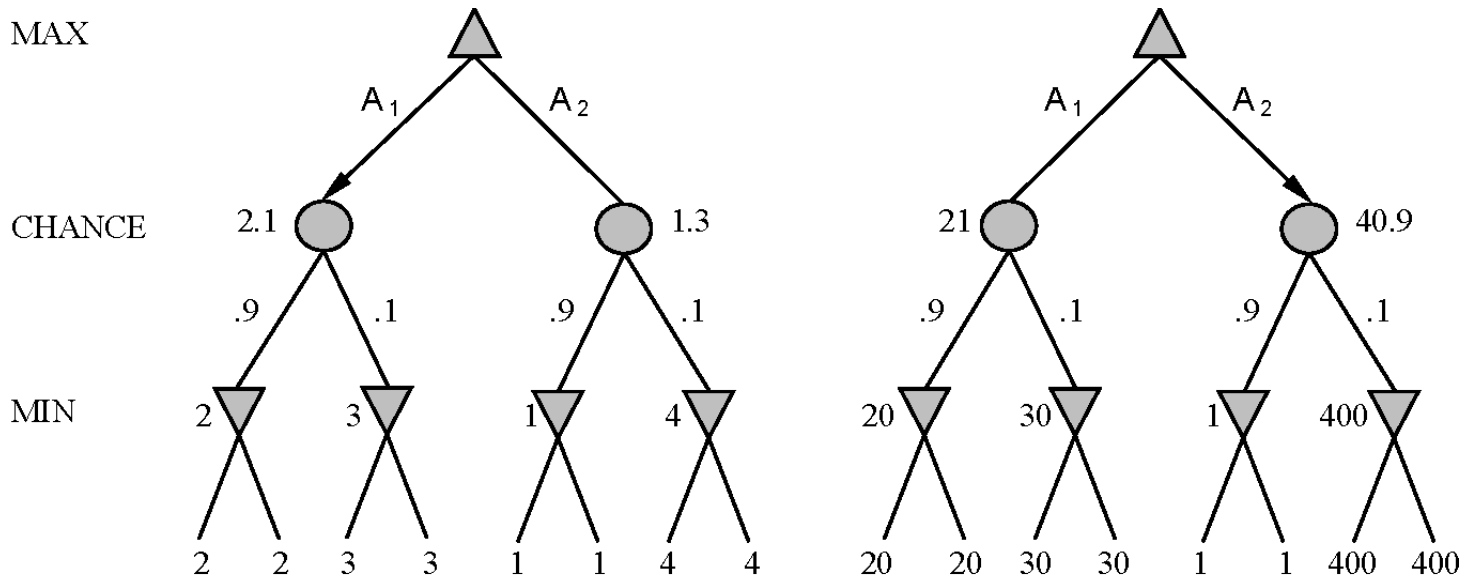


极小极大值→期望极小极大值

$$\text{expectimin imax}(n) =$$

$$\begin{cases} \text{utility}(n) & n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{expectimin imax}(s) & n \text{ is a MAX state} \\ \min_{s \in Successors(n)} \text{expectimin imax}(s) & n \text{ is a MIN state} \\ \sum_{s \in Successors(n)} p(s)\,\text{expectimin imax}(s) & n \text{ is a chance state} \end{cases}$$

有机率节点的游戏中的局面评价



在保持顺序不变的情况下，叶节点赋值的变换导致了最佳招数的改变。

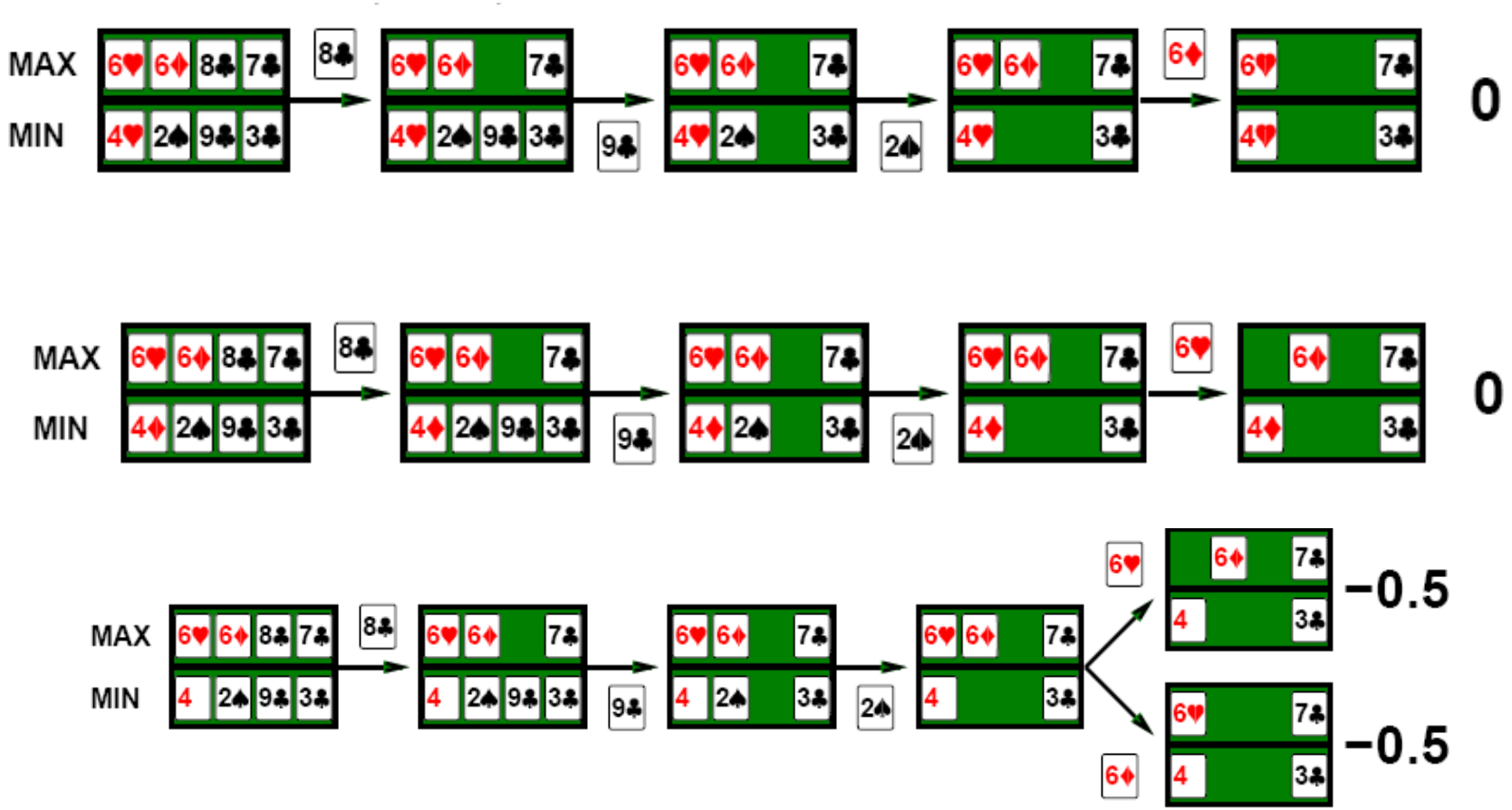
期望极小极大值的复杂度

- 算法的时间复杂度为$O(b^m n^m)$，其中n为不同的掷骰子结果的数目。
- 可以对有几率节点的博弈树使用类似α-β剪枝的技术吗？
  - 限制效用函数的取值范围⇒不用看几率节点的子节点就可以设置几率节点的值的上界

第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

牌类游戏



牌类游戏

- 考虑未知牌的所有可能应对策略 假设应对策略s发生的概率是p(s)

$$\arg\max_{a} \sum P(s)MINIMAX(RESULT(s,a))$$

- 若使用蒙特卡洛近似，随机采样N个样本，设组合s在样本中出现的概率与p(s)成正比：

$$\arg\max_{a} \frac{1}{N} \sum_{i=1}^{n} P(s)MINIMAX(RESULT(s,a))$$

第五章、对抗搜索

- 博弈中的优化决策
- α–β剪枝：允许我们忽略那些不影响最后决定的部分搜索树
- 不完整的实时决策
- 包含几率因素的游戏
- 部分可观察博弈
- 博弈程序的当前发展水平

博弈程序发展现状

- 国际象棋：IBM深蓝打败了国际象棋大师
- 西洋跳棋：Chinook在1990年的简化比赛中战胜了长久以来的人类冠军
- 翻转棋：1997年的Logistello程序 以6比0击败了人类冠军
- 西洋双陆棋：TD-GAMMON稳定的排在世界前列
- 围棋：围棋就有点惨，也就能达到个业余选手的水平。