CSE 4355 | Week 5 | Class 1 | Saturday · 9:30 AM – 10:50 AM | 80 min

Minimax and alpha–beta pruning

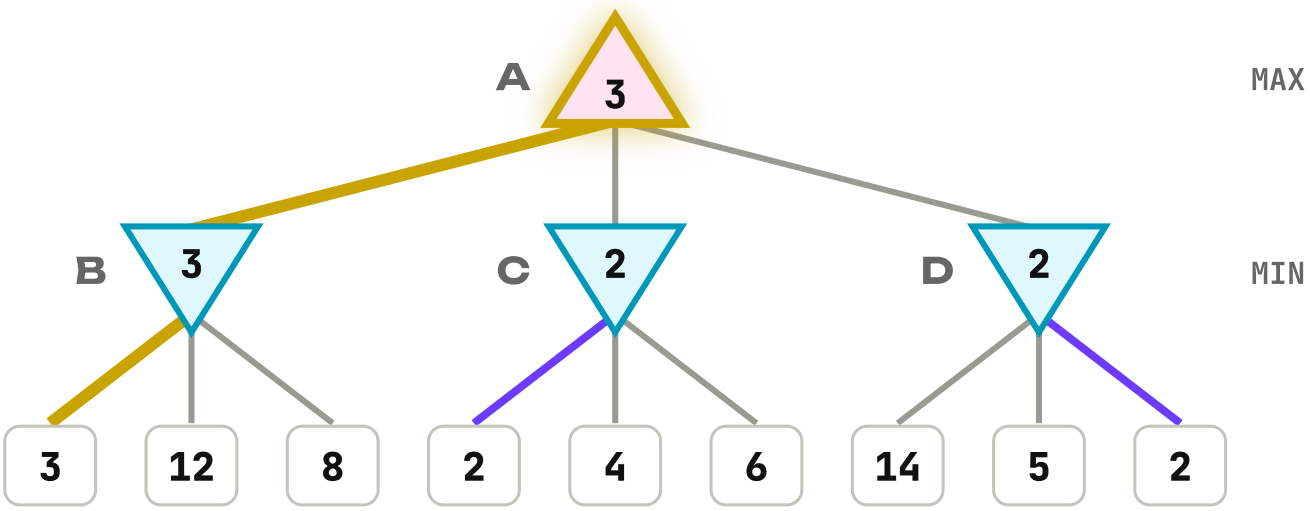
Minimax looks at every leaf. Most of them cannot change the decision. Alpha–beta proves which ones, skips them, and, with good move ordering, searches twice as deep in the same time.

---

01 COLD OPEN 0–4 min

Do you need to look at every leaf?

Last class's tree. Minimax read all 9 leaves to find A = 3.



THINK DEEPER

You have finished B (= 3). Under C, the first leaf is 2. Do you need to look at C's other two leaves?

No. C ≤ 2 < 3, so C is out of the race. The leaves 4 and 6 could be anything, even 1,000,000, and the decision would not change. Today: a rule that finds such cuts automatically.

A STRONG ANSWER MENTIONS

- C is a MIN node: its value can only be 2 or less.
- MAX already has 3 from B.
- So MAX will never choose C, whatever the 4 and the 6 are.

02 CONCEPT 4–10 min

Alpha and beta: what each side already has

- **α** = the best (highest) value MAX can already guarantee on the path to this node. Starts at −∞, only goes up.
- **β** = the best (lowest) value MIN can already guarantee on that path. Starts at +∞, only goes down.
- The true value that matters lies in the window **[α, β]**. When the window closes, **α ≥ β**, the rest of the node's children cannot matter: **prune**.

$$\text{MIN node: cut when } v \le \alpha \qquad \text{MAX node: cut when } v \ge \beta$$

READ IT AS A CONVERSATION

MAX at the root: "I already have a move worth 3." **MIN at C:** "Here I can hold you to 2." **MAX:** "Then I'm not coming to C. Stop looking."

PRO TIP

α and β travel **down** the tree as arguments; values come back **up**. A child's window is its parent's current window.

03 CONCEPT 10–14 min

Alpha–beta in fifteen lines

```
function MAX-VALUE(s, α, β):
  if TERMINAL(s): return UTILITY(s)
  v ← -∞
  for each a in ACTIONS(s):
    v ← max(v, MIN-VALUE(RESULT(s, a), α, β))
    if v ≥ β: return v        // MIN above will avoid this node
    α ← max(α, v)
  return v

function MIN-VALUE(s, α, β):
  if TERMINAL(s): return UTILITY(s)
  v ← +∞
  for each a in ACTIONS(s):
    v ← min(v, MAX-VALUE(RESULT(s, a), α, β))
    if v ≤ α: return v        // MAX above will avoid this node
    β ← min(β, v)
  return v
```

Call `MAX-VALUE(s₀, -∞, +∞)` and keep the action with the best value.

GOTCHA

Values returned from a cut node are **bounds**, not exact: C returns "2", meaning "≤ 2". Only the root's value is guaranteed exact.

PRO TIP

It is minimax plus two lines per function. Still depth-first: **O(bm)** memory.

04 WORKED EXAMPLE 14–20 min

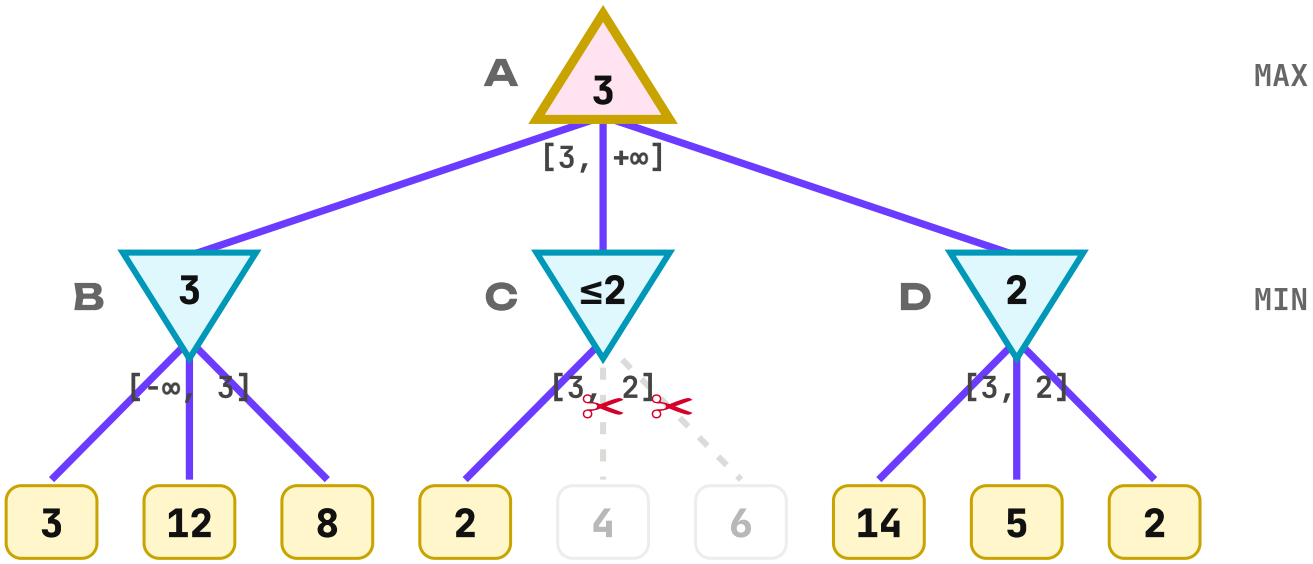
Worked example: pruning by hand

GIVEN

Last class's tree (Russell & Norvig): MAX root, leaves 3 12 8 | 2 4 6 | 14 5 2.

FIND

The root value, each node's [α, β] window, the pruned leaves.



A returns 3.

Method: ALPHA-BETA-SEARCH (Russell & Norvig)

SOLUTION

1. Root A starts with the window [α, β] = [−∞, +∞]. Visit B (MIN) with the same window. Leaves 3, 12, 8: B = 3. Back at A: **α = 3**. MAX can now guarantee at least 3.

2. Visit C (MIN) with [3, +∞]. First leaf: **2**. So C ≤ 2. But MAX already has 3 via B: whatever C's other leaves are, MAX will never choose C.

$$v_C \le 2 \le \alpha = 3 \;\Rightarrow\; \text{prune } 4, 6$$

3. Visit D (MIN) with [3, +∞]. Leaves 14 (D ≤ 14), 5 (D ≤ 5), 2 (D = 2). No cut was possible: the small leaf came last.

   > Had D's children been ordered 2, 5, 14, the first one would have pruned the other two.

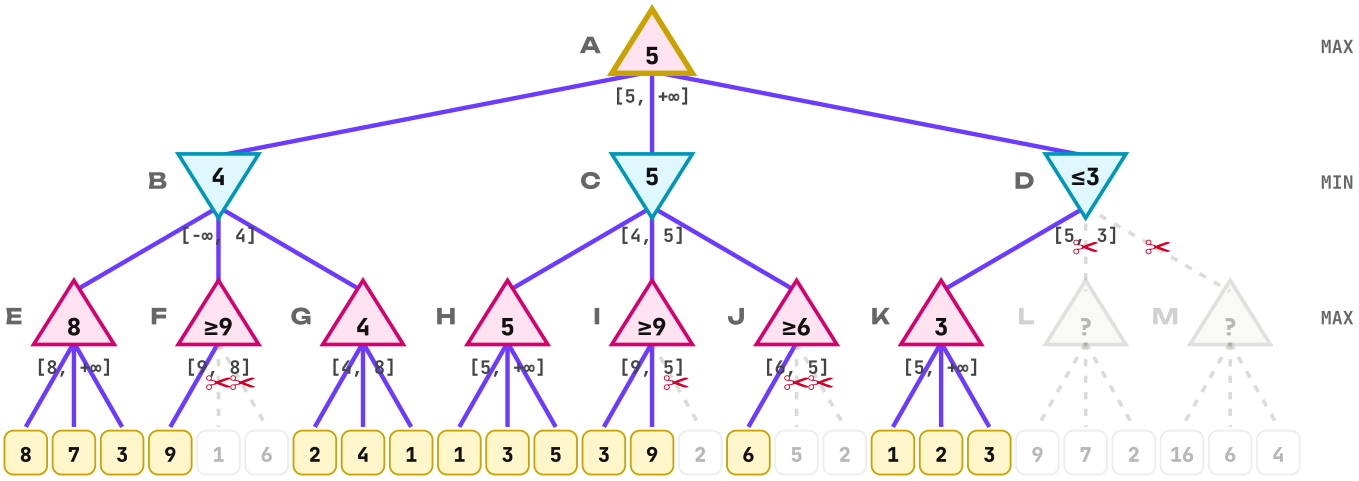
4. A = max(3, ≤2, 2) = **3**, the same value and move as plain minimax, after 7 leaves instead of 9.

ANSWER

A = 3 (move to B). Leaves 4 and 6 under C are pruned; 7 of 9 leaves evaluated.

05 LIVE LAB 20–27 min

Live lab: pruning, and why order matters



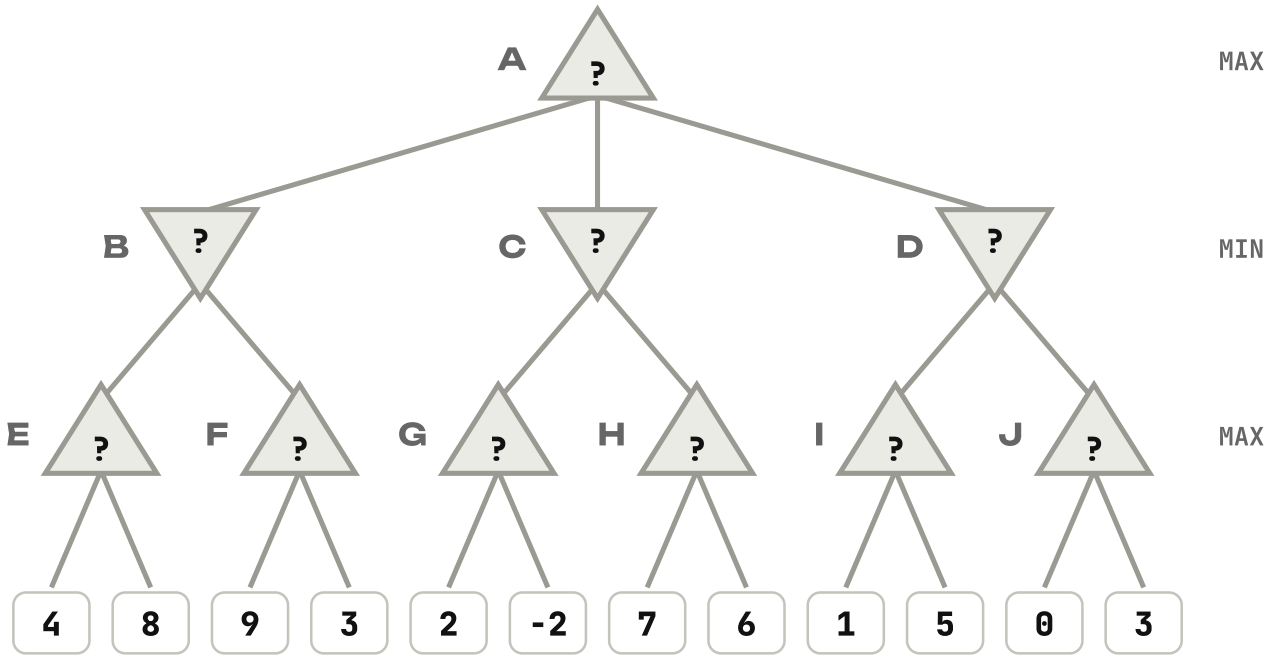
A returns 5.

06 YOUR TURN 27–33 min

Your turn · 1: alpha–

beta on a three-ply tree

1 EXAM CQ



Run alpha–beta left to right. Give the [α, β] window of every node when it is visited, and list the pruned leaves.

SOLUTION

1. E (MAX) under B: leaves 4, 8 → E = 8. B: β = 8.
2. F (MAX) gets [−∞, 8]. First leaf 9: v = 9 ≥ β = 8 → **prune leaf 3**. F ≥ 9, so B = min(8, ≥ 9) = 8. Root: α = 8.
3. C (MIN) gets [8, +∞]. G: leaves 2, −2 → G = 2. C: v = 2 ≤ α = 8 → **prune H** (leaves 7, 6).
4. D (MIN) gets [8, +∞]. I: leaves 1, 5 → I = 5. D: 5 ≤ 8 → **prune J** (leaves 0, 3). Root A = **8**.

> 7 of 12 leaves evaluated. The good first move (B) made α high early, so C and D were cut after one grandchild each.

ANSWER

A = 8, move to B. Pruned: leaf 3 (under F), subtree H (7, 6), subtree J (0, 3). 7 of 12 leaves evaluated.

07 CONCEPT 33–38 min

Move ordering decides everything

ORDERING	NODES EXAMINED	EFFECTIVE B
Worst first	b^m (no pruning)	b
Random	$\approx b^{3m/4}$	$b^{3/4}$

ORDERING	NODES EXAMINED	EFFECTIVE B
Best first (perfect)	$\approx b^{m/2}$	$\sqrt{b}$

Chess: b ≈ 35 becomes ≈ 6 with perfect ordering. We cannot know the best move in advance (that is the search), but cheap guesses come close:

- try **captures and checks** first; then **killer moves** (moves that caused cuts elsewhere at this depth);
- **iterative deepening**: the best move from depth d − 1 is searched first at depth d;
- **transposition table**: a hash table of positions already evaluated (different move orders reach the same position).

PRO TIP

Iterative deepening (Week 3) looks wasteful, but here it pays for itself: the shallow searches produce the move ordering for the deep one.

---

08 WORKED EXAMPLE 38–42 min

Worked example: how deep can it see?

GIVEN

Chess, b ≈ 35. The computer can examine 10^9 positions per move.

FIND

The search depth for minimax, for alpha–beta with perfect ordering, and with random ordering.

Method: complexity of alpha-beta (Russell & Norvig; Knuth & Moore)

SOLUTION

1. Minimax looks at b^d nodes:

$$35^d = 10^9 \;\Rightarrow\; d = \frac{9}{\log_{10} 35} = \frac{9}{1.544} \approx 5.8 \text{ ply}$$

2. Alpha–beta with perfect ordering looks at about $b^{d/2}$:

$$35^{d/2} = 10^9 \;\Rightarrow\; d \approx 11.7 \text{ ply}$$

3. With random ordering, about $b^{3d/4}$: d ≈ 7.8 ply. Real engines, with good ordering, get close to the perfect case.

> Same hardware, same time: twice the look-ahead. That is the difference between a club player and a grandmaster.

ANSWER

Minimax ≈ 5.8 ply; alpha–beta: ≈ 11.7 ply (perfect), ≈ 7.8 ply (random).

---

09 THINK DEEPER 42–46 min

Critical thinking · what

does pruning cost?

THINK DEEPER

Alpha–beta skipped leaf 3 under F. Suppose that leaf was actually 100. Would MAX have played differently?

No. F is worth at least 9 to MAX, so MIN at B avoids F whatever it is. Pruning is **exact**: it never changes the root decision, it only refuses to compute numbers nobody will use.

A STRONG ANSWER MENTIONS

- F is a MAX node: with 9 already, F ≥ 9. With 100 it would be 100.
- B is MIN: it chooses E (8) over F (≥ 9) either way.
- Only a value that changes a choice somewhere on the path to the root can matter; the cut proves this one cannot.

THINK DEEPER

Then why do some engines also use forward pruning (skipping moves that merely look bad), which can be wrong?

Because $\sqrt{b}$ is still exponential. Forward pruning trades **safety for depth**: it can miss a hidden sacrifice, but seeing deeper everywhere usually wins more games than it loses.

A STRONG ANSWER MENTIONS

- Alpha–beta alone cannot beat $b^{m/2}$.
- Forward pruning (beam search, ProbCut, late move reductions) gives up the guarantee for more depth.
- It is a bet: occasionally missing a brilliant sacrifice versus seeing several ply deeper everywhere else.

10 CONCEPT 46–51 min

Dice and cards: expectiminimax

In **Ludo**, backgammon or any game with dice, the tree has a third kind of node: **chance**, whose children are the outcomes of the roll, each with a probability.

$$\mathrm{EMM}(s) = \begin{cases} \mathrm{UTILITY}(s) & \text{terminal} \\ \max_a \mathrm{EMM}(\mathrm{RESULT}(s,a)) & \text{MAX} \\ \min_a \mathrm{EMM}(\mathrm{RESULT}(s,a)) & \text{MIN} \\ \sum_r P(r)\,\mathrm{EMM}(\mathrm{RESULT}(s,r)) & \text{CHANCE} \end{cases}$$

! SCALE MATTERS NOW

Minimax only compares values, so any order-preserving change of EVAL gives the same move. An **average** is different: EVAL must be proportional to the expected payoff (e.g. a win probability), not just ranked correctly.

✦ PRO TIP

Time grows to $O(b^m n^m)$ for n dice outcomes: a Ludo die adds a factor of 6 per turn. Alpha–beta pruning works only weakly on chance nodes.

11 WORKED EXAMPLE 51–56 min

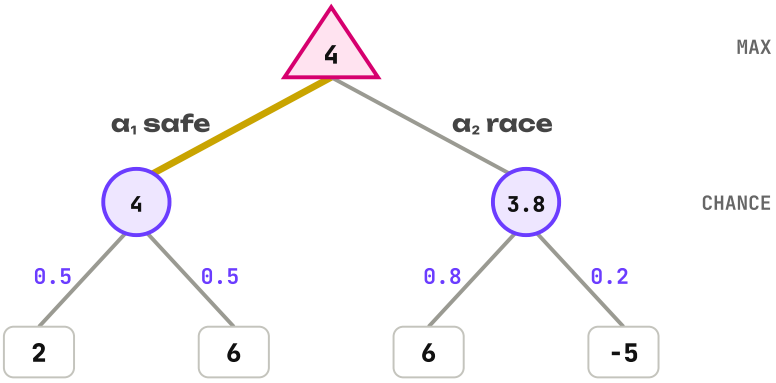
Worked example: safe move or race?

GIVEN

Ludo. (a_1) Move to a safe square: utility 2 or 6, probability ½ each (the opponent's roll). (a_2) Race ahead: utility 6 with probability 0.8, −5 if the opponent rolls the capture (0.2).

FIND

The expectiminimax value of each move and the best move.



Method: EXPECTIMINIMAX (Russell & Norvig)

SOLUTION

1. Chance node after a_1:

$$0.5(2) + 0.5(6) = 4$$

2. Chance node after a_2:

$$0.8(6) + 0.2(-5) = 4.8 - 1.0 = 3.8$$

3. MAX takes the larger average: **a_1 (safe), 4 > 3.8**. Note a_2 has the more likely good outcome; the rare capture is costly enough to outweigh it.

ANSWER

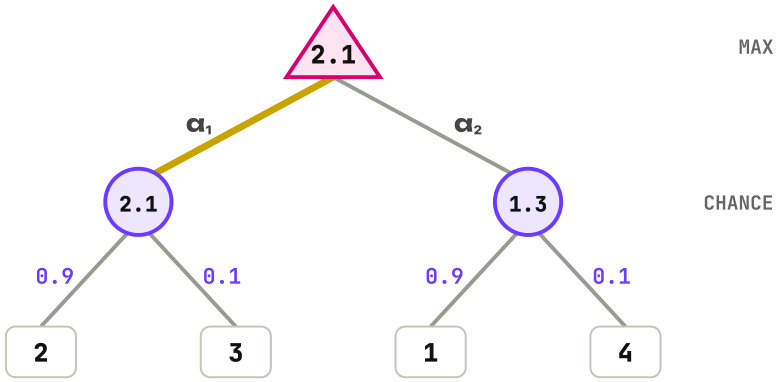
a_1 = 4, a_2 = 3.8: play the safe move.

12 YOUR TURN 56–61 min

Your turn · 2: same

order, different move

2 CRITICAL THINKING



Find MAX's best move. Then relabel the leaves 1 → 1, 2 → 20, 3 → 30, 4 → 400 (the order is unchanged) and find it again. What does this tell you about evaluation functions?

SOLUTION

1. Original:

$$a_1 = 0.9(2) + 0.1(3) = 2.1, \qquad a_2 = 0.9(1) + 0.1(4) = 1.3 \;\Rightarrow\; a_1$$

2. Relabelled (same order):

$$a_1 = 0.9(20) + 0.1(30) = 21, \qquad a_2 = 0.9(1) + 0.1(400) = 40.9 \;\Rightarrow\; a_2$$

3. The best move **changed**. In plain minimax it could not: max and min care only about order.

> With chance nodes, EVAL must be a positive linear transformation of the expected utility (e.g. probability of winning).

ANSWER

Original: a_1 (2.1 vs 1.3). Relabelled: a_2 (21 vs 40.9). Order-preserving changes of EVAL can change expectiminimax decisions.

---

13 REAL WORLD 61–65 min

Real world: what engines actually do

STOCKFISH

The strongest open-source chess engine: alpha–beta with iterative deepening, transposition tables, killer moves, late-move reductions and quiescence search, with a small neural network (NNUE) as EVAL. Tens of millions of positions per second on a laptop.

ALPHAZERO (2017)

Replaced alpha–beta by **Monte Carlo tree search** guided by a deep network, trained only by self-play. It examined about 1,000× fewer positions than Stockfish and still beat it.

LUDO ON YOUR PHONE

Mobile Ludo bots run expectiminimax a few plies deep (or simple rules). Sholo Guti has no dice, so a minimax player with alpha–beta and a material-count EVAL is a weekend project: a good presentation topic.

> Go's b ≈ 250 makes even $\sqrt{b}$ too large, and no simple EVAL exists. MCTS samples promising lines deeply (random playouts or a value network) instead of trying every move to a fixed depth.

---

14 POP QUIZ 65–70 min

Pop quiz

1. **Alpha–beta pruning, compared with minimax on the same tree, returns…**

   A. a better root value

   B. the same root value and move, usually after fewer leaves ✓

   C. an approximate value

   D. a different move when pruning happens

   Pruning removes only branches that cannot influence the root decision. The root value is identical; some internal nodes return only bounds.

2. **At a MIN node, the remaining children are pruned when…**

   A. $v \geq \beta$

   B. $v \leq \alpha$ ✓

   C. $\alpha = -\infty$

   D. $v = 0$

   MIN's value can only go down from v. If $v \leq \alpha$, MAX higher up already has a better option, so this node will never be chosen.

3. **With perfect move ordering, alpha–beta examines about…**

   A. b^m nodes

   B. $b^{m/2}$ nodes ✓

   C. $m \cdot b$ nodes

   D. b^{2m} nodes

   The effective branching factor drops from b to $\sqrt{b}$, so the same time searches twice as deep.

4. **In the textbook tree [[3, 12, 8], [2, 4, 6], [14, 5, 2]], how many leaves does alpha–beta skip?**

   A. 0

   B. 2 ✓

   C. 4

   D. 6

   Under C, after the first leaf $2 \leq \alpha = 3$, the leaves 4 and 6 are pruned. D cannot be cut because its 2 comes last.

5. **The value of a chance node in expectiminimax is…**

   A. the best child

   B. the worst child

   C. the probability-weighted average of its children ✓

   D. the most likely child

   EXPECTIMINIMAX(chance) = Σ P(r) · value(child r). Values must therefore be on a meaningful scale, not just in the right order.

---

15 RECAP 70–80 min

Three things to remember

01

Alpha–beta: carry [α, β] down; cut a MIN node when v ≤ α, a MAX node when v ≥ β. Same root value as minimax.

02

Ordering: worst b^m, random ≈ $b^{3m/4}$, best ≈ $b^{m/2}$: twice the depth. Iterative deepening, killer moves and transposition tables get close to the best case.

03

Chance nodes average: Σ P(r)·value. EVAL must then have a meaningful scale, not just the right order.

HW BEFORE NEXT CLASS

(a) Run alpha–beta on [[3, 2, 1], [4, 8, 9], [5, 6, 7]] (MAX root, three MIN children). Which leaves are pruned? (b) Reorder the children (not the values within a leaf set) so that alpha–beta prunes as much as possible. Check both in the lab.

Assignment 2 is due next class. The midterm (Weeks 1–5, CQ only) follows this week.

SOLUTION

1. (a) B = min(3, 2, 1) = 1, α = 1. C: 4, 8, 9: v stays above α, no cut; C = 4, α = 4. D: 5, 6, 7: no cut; D = 5. Root 5, **all 9 leaves**.
2. (b) Put the best MAX move first and each MIN node's smallest leaf first: [[5, 6, 7], [4, 8, 9], [3, 2, 1]]. B = 5, α = 5; C: 4 ≤ 5 → prune 8, 9; D: 3 ≤ 5 → prune 2, 1. **5 leaves**.

ANSWER

(a) Root 5, no pruning (9 leaves). (b) Reordered [[5, 6, 7], [4, 8, 9], [3, 2, 1]]: root 5 after 5 leaves.

S. M. Sakeef Sani
Adjunct Lecturer, CSE · Northern University Bangladesh
smsakeefsani.hbitcorp.com