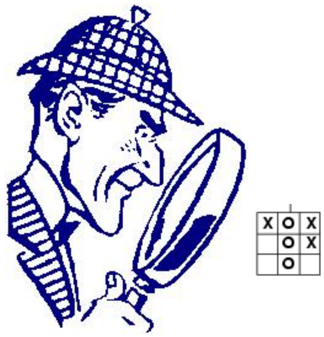
CSE 473

Lecture 7

Playing Games with Minimax and Alpha-Beta Search

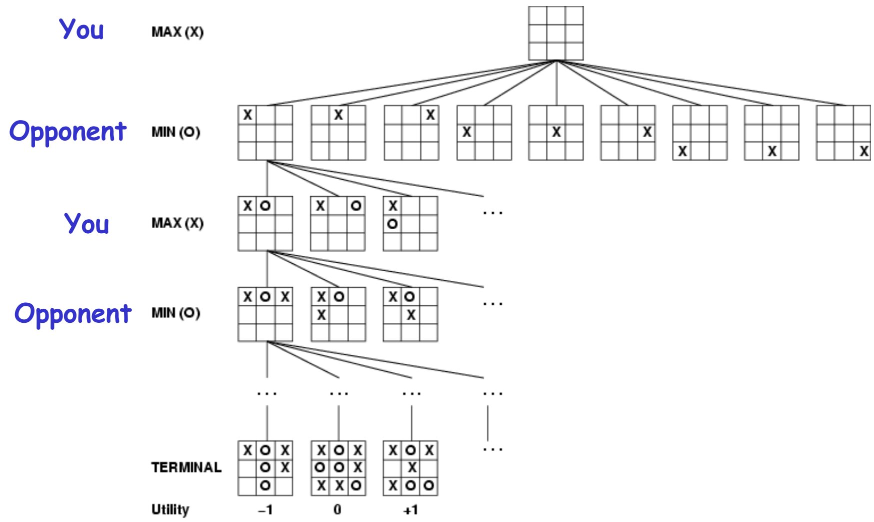


Today

- Adversarial Search
  - Minimax recap
  - α-β search
  - Evaluation functions
  - State of the art in game playing

Recall: Game Trees

From current position, unwind game into the future



Recall: Minimax Search

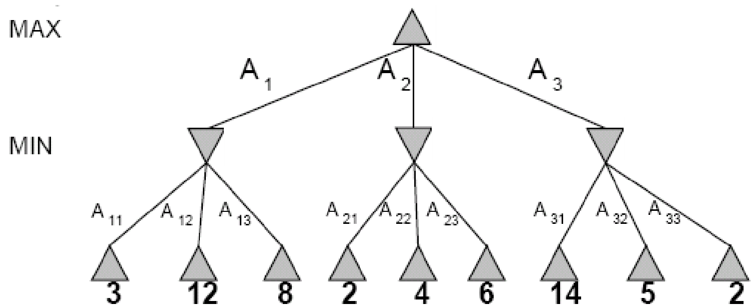
- Find the *best current move* for MAX (you) assuming MIN (opponent) also chooses *its* best move

- Compute for each node *n*:

MINIMAX-VALUE(*n*)=

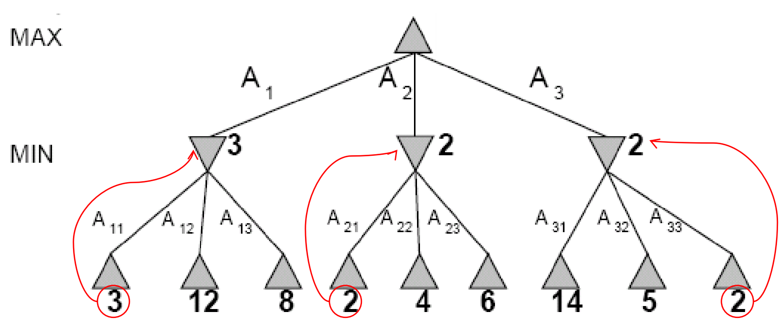
UTILITY(*n*)	if *n* is a terminal
$\max_{s \in succ(n)}$ MINIMAX-VALUE(*s*)	if *n* is a MAX node
$\min_{s \in succ(n)}$ MINIMAX-VALUE(*s*)	if *n* is a MIN node

Example: Two-"Ply" Game Tree

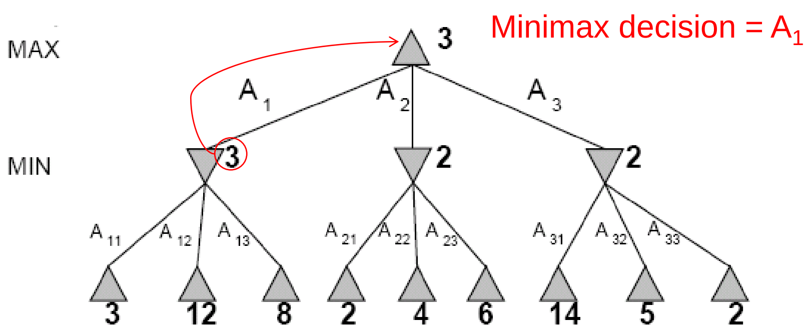


(1 ply = 1 move = 1 layer in tree)

Two-Ply Game Tree



Two-Ply Game Tree



7

What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally

  Maximizes worst-case outcome for MAX

- If MIN does not play optimally, MAX will do even better (utility obtained by MAX will be higher). [Prove it! See Exercise 5.7]

8

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*

  $v \leftarrow$ MAX-VALUE(*state*)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*) **returns** *a utility value*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

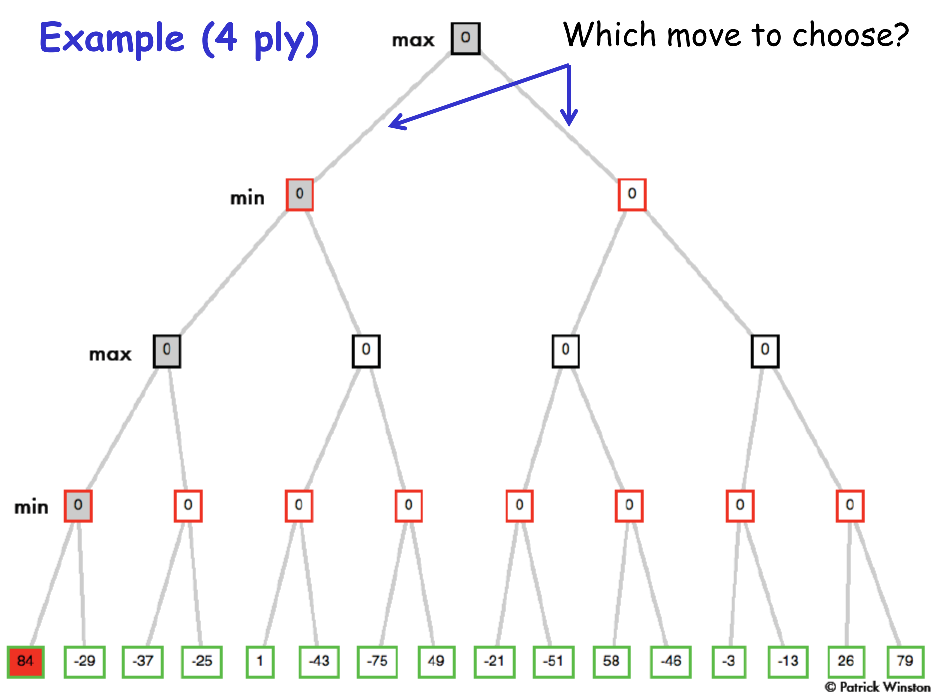
---

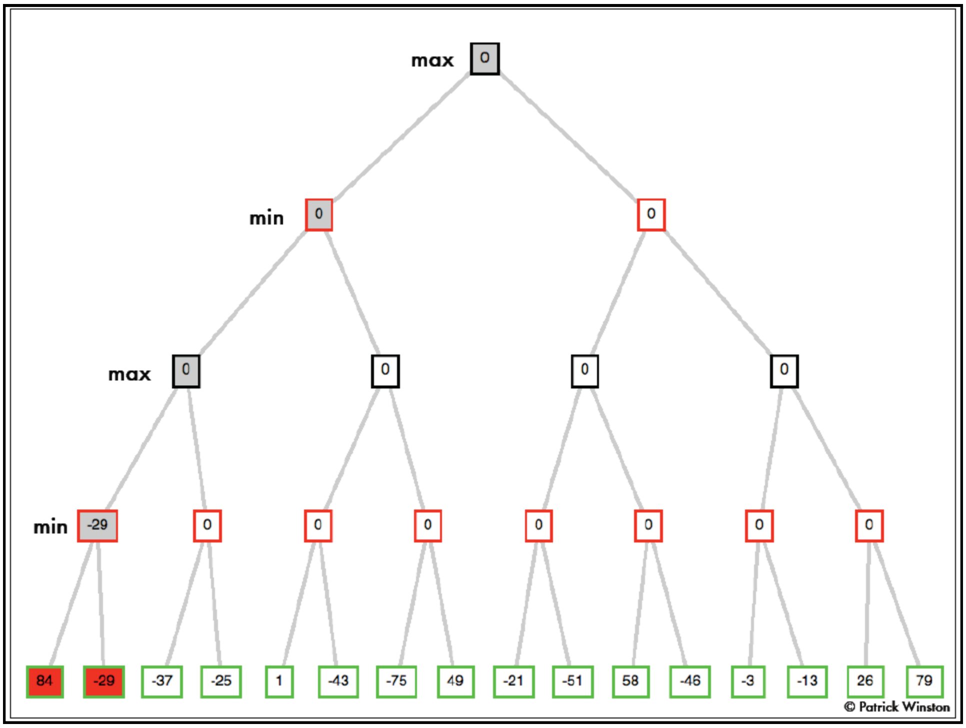
function MIN-VALUE(*state*) **returns** *a utility value*

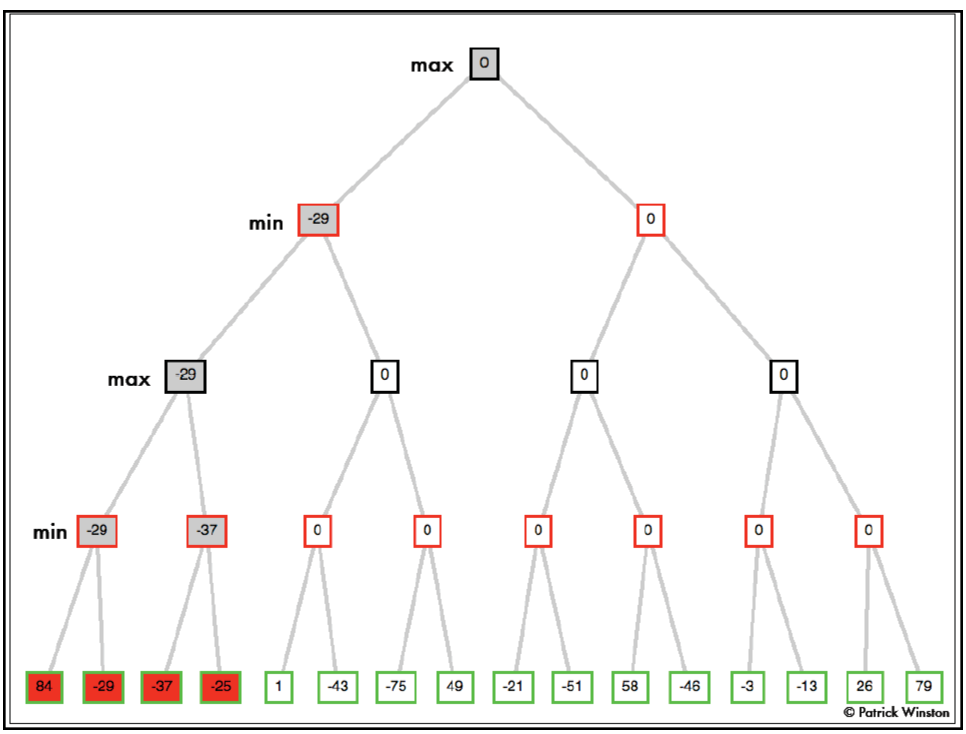
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return v

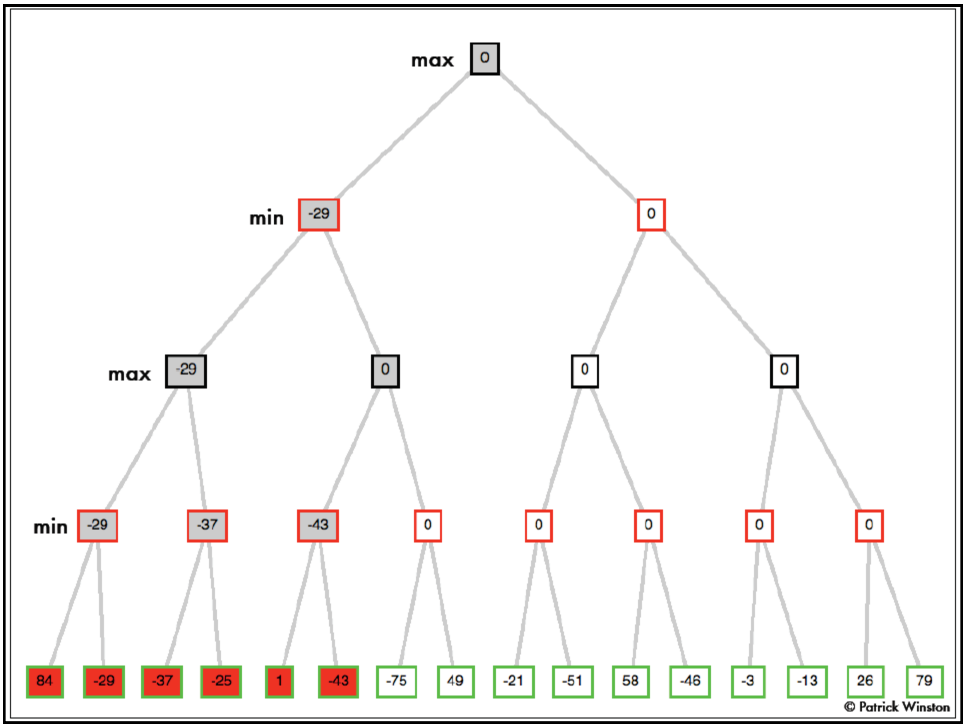
9

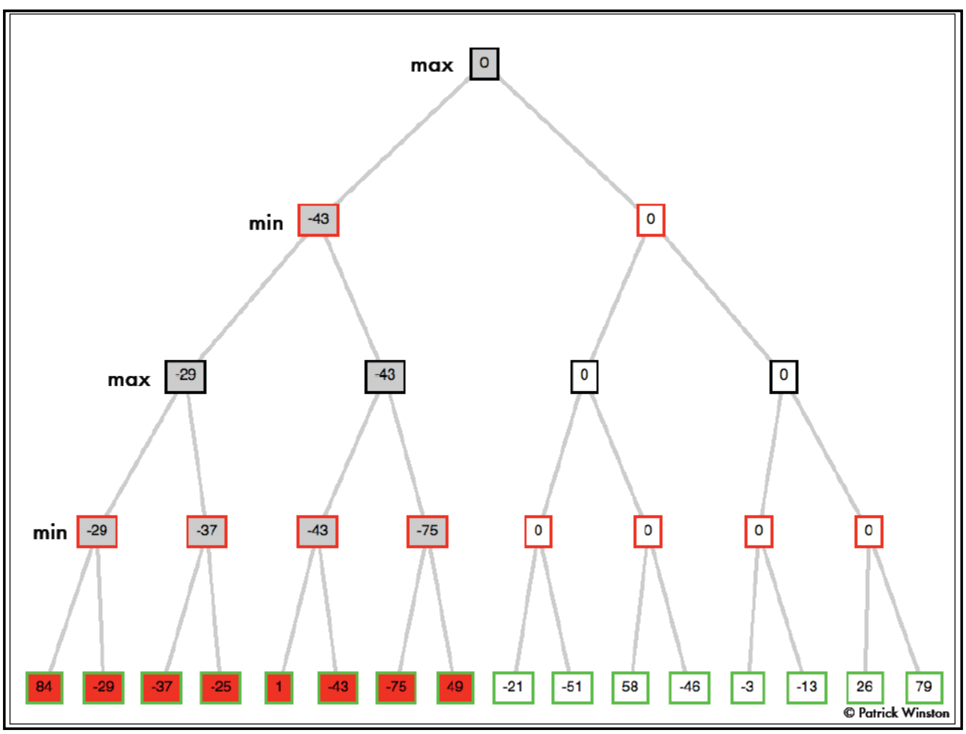
Example (4 ply)

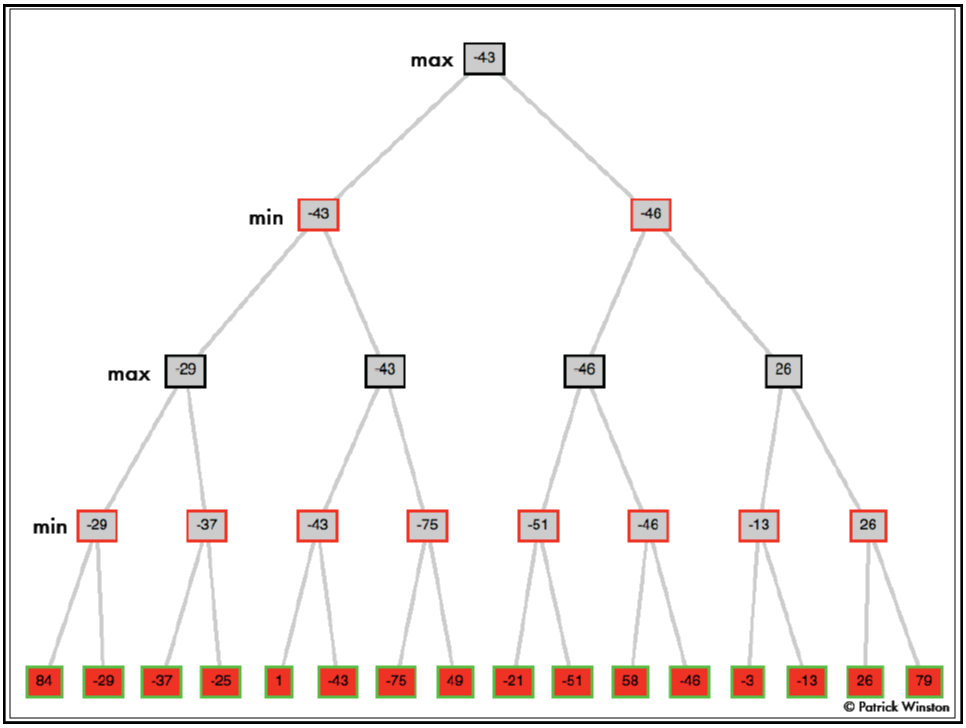


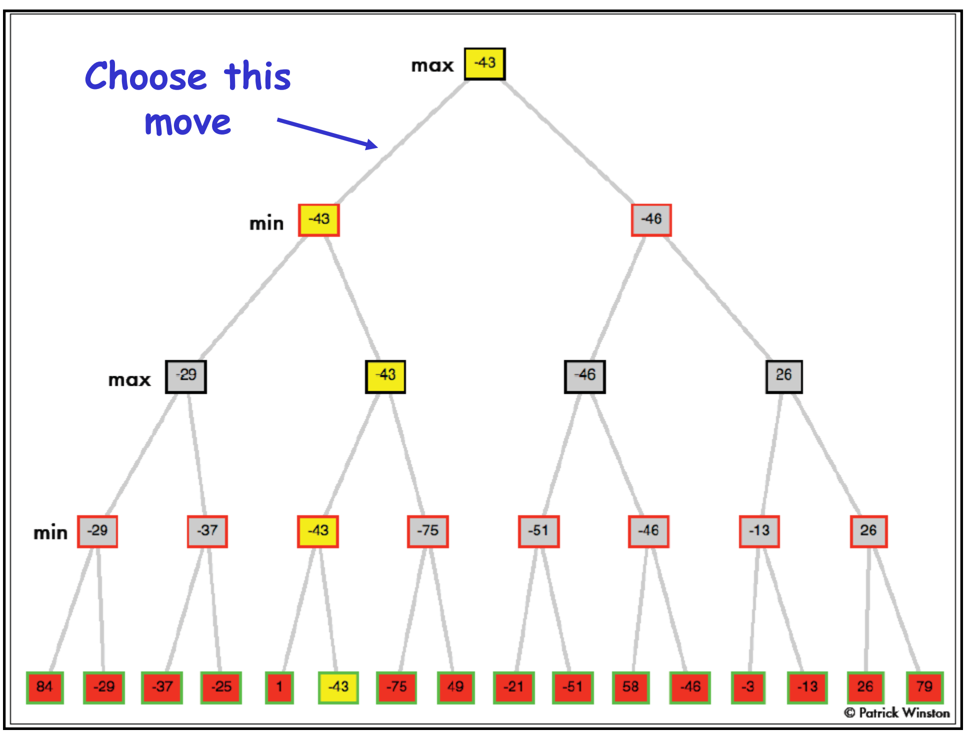

max
min
max
min
0
0
0
0
0
0
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston


max
min
max
min
0
-29
0
-29
0
0
0
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston


max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
-43
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston

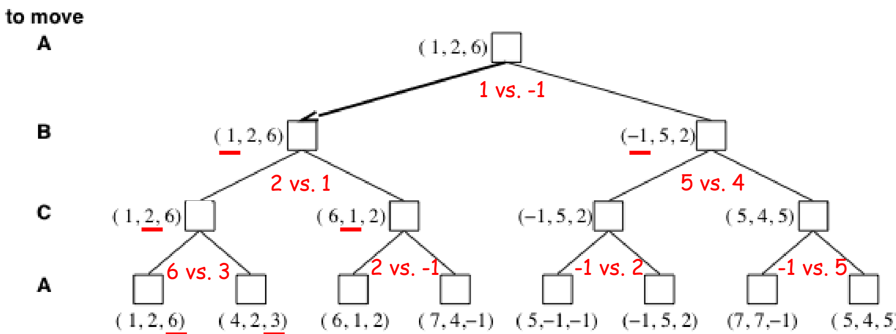

max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston


max
min
max
min
-43
-43
-46
-29
-43
-46
26
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston


Choose this move
max
min
max
min
-43
-43
-46
-29
-43
-46
26
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston

Extension to Multiplayer Games

- More than two players
- Single minimax values become *vectors*
- At each node, apply max to appropriate component of minimax vector



17

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
  Suboptimal opponents: Other strategies may do better but these will do worse for optimal opponents
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

18

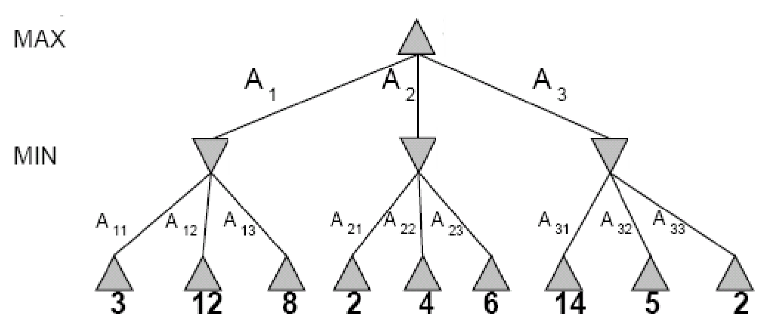
Is Minimax good enough?

- Chess:
  - branching factor b ≈ 35
  - game length m ≈ 100
  - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
  - number of atoms ≈ 10^{78}
  - age ≈ 10^{21} milliseconds

Can we search more efficiently?

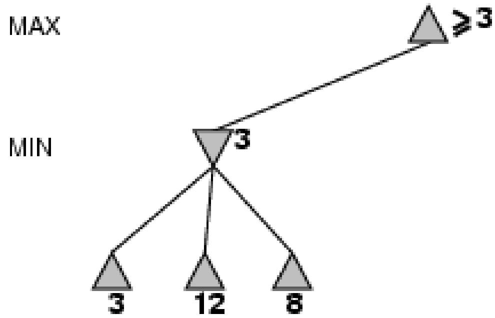
19

Back to Two-Ply Game Tree



20

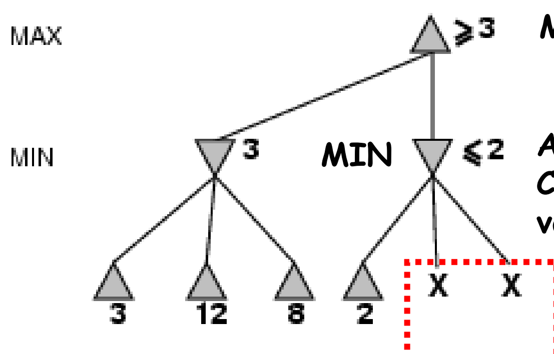
Pruning trees



Minimax algorithm explores depth-first

21

Pruning trees



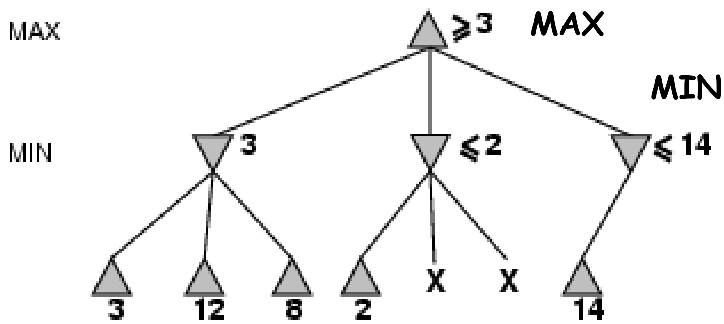
MAX

At MIN node:
Current best MAX value α = 3 > 2

No need to look at these nodes!! (these nodes can only decrease MIN value from 2)

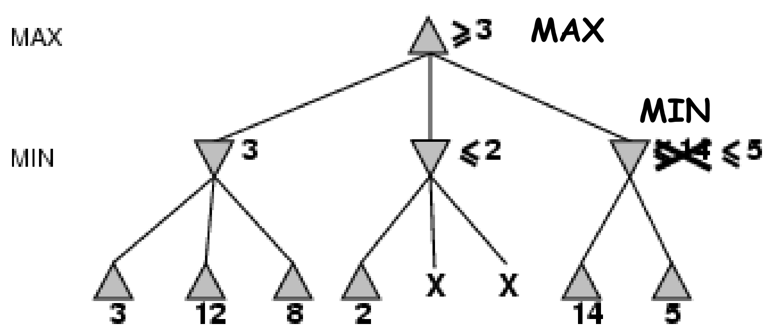
22

Pruning trees



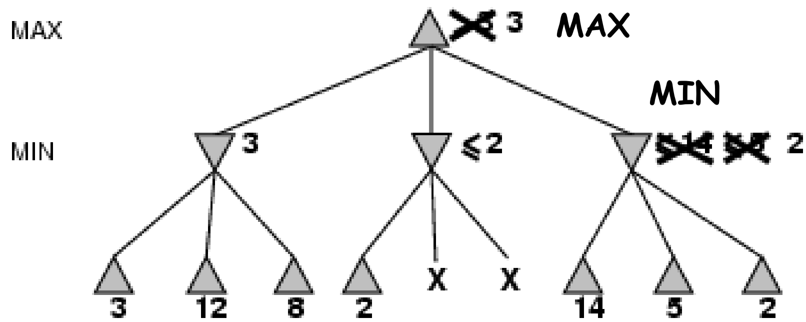
23

Pruning trees



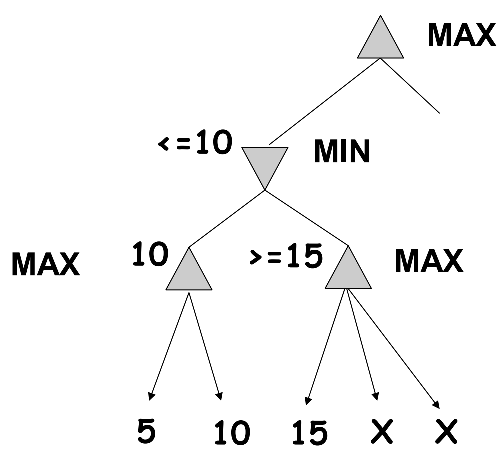
24

Pruning trees



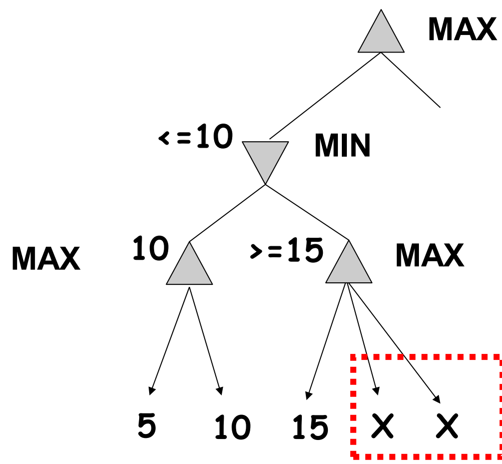
25

One more example



26

One more example



At MAX node: Current best MIN value β = 10 < 15

No need to look at these nodes!! (these nodes can only increase MAX value from 15)

27

This form of tree pruning is known as alpha-beta pruning

alpha = highest (best) value for MAX along current path from root

beta = lowest (best) value for MIN along current path from root

The α-β algorithm

(minimax with four lines of added code)

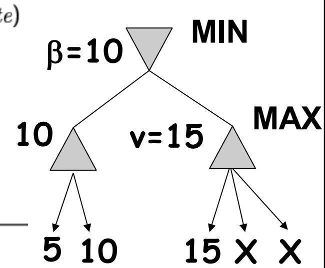
function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

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*

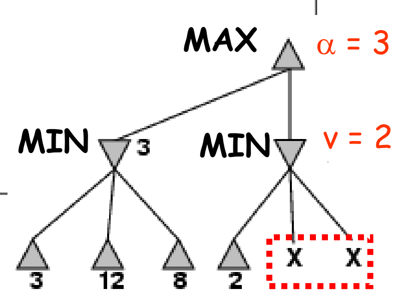
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
New   **if** $v \geq \beta$ **then return** v → Pruning
    $\alpha \leftarrow$ MAX(α, v)
  return v



The α-β algorithm (cont.)

function MIN-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*)
  $v \leftarrow +\infty$
  for s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v → Pruning
    $\beta \leftarrow$ MIN(β, v)
  return v



30

Properties of α-β

- Pruning does not affect final result
- Good *move ordering* improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → allows us to search deeper – doubles depth of search
- α-β search is a simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

31

Good enough?

- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - α-β search space $b^{m/2} \approx 35^{50} \approx 10^{77}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{21}$ milliseconds

32

Transposition Tables

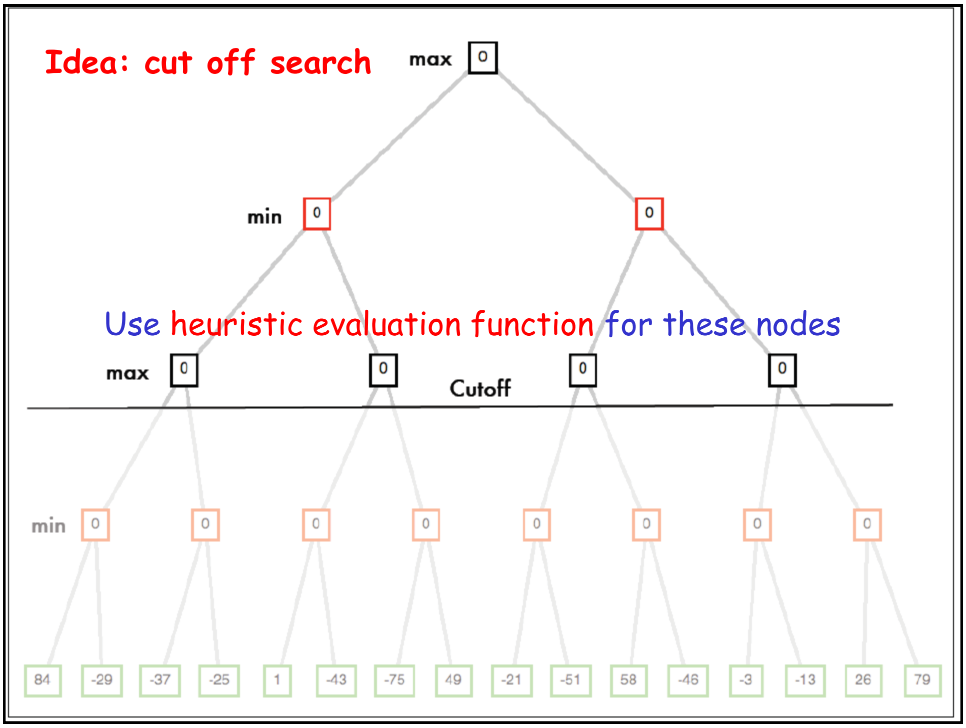
- Game trees contain repeated states
- In chess, e.g., the game tree may have 35^{100} nodes, but there are only 10^{40} different board positions
- Similar to *explored set* in graph-search, maintain a transposition table
  - Got its name from the fact that the same state is reached by a transposition of moves.
- 10^{40} is still huge!

33

Can we do better?

- Strategies:
  - search to a fixed depth ("cut off" search)
  - *iterative deepening* (most common)

34



Heuristic Evaluation Functions

- Motivation: When search space is too large, create game tree up to a certain depth only.
- Art is to estimate utilities of positions that are not terminal states.
- Example of simple evaluation criteria in chess:
  - Material worth: pawn=1, knight =3, rook=5, queen=9.
  - Other: king safety, good pawn structure

eval(s) =

w1 * material(s) +
w2 * mobility(s) +
w3 * king safety(s) +
w4 * center control(s) + ...

36

Cutting off search

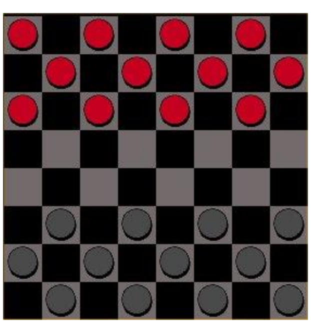
- Does it work in practice?
  Suppose $b^m = 10^6$ and $b=35 \Rightarrow m=4$

- 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov

37

Game Playing State-of-the-Art

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in **1994**. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!



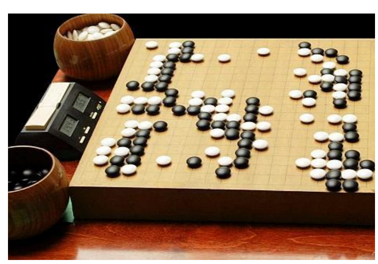
Game Playing State-of-the-Art

- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in **1997**. Deep Blue examined 200 million positions per second, used very sophisticated evaluation functions and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Game Playing State-of-the-Art

- **Othello**: Human champions refuse to play against computers because software is too good

- **Go**: Human champions refuse to play against computers because software is too bad.
  - In Go, b > 300, so need pattern databases and Monte Carlo search
  - Human champions are now beginning to be challenged by machines.



- **Pacman:** The reigning champion is <your CSE 473 program here>

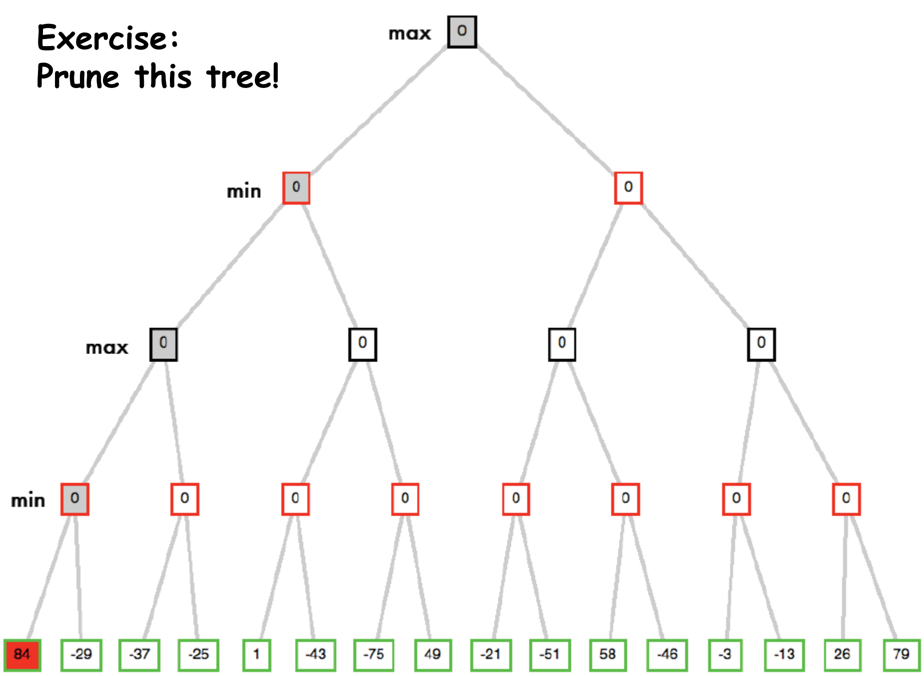
Next Time

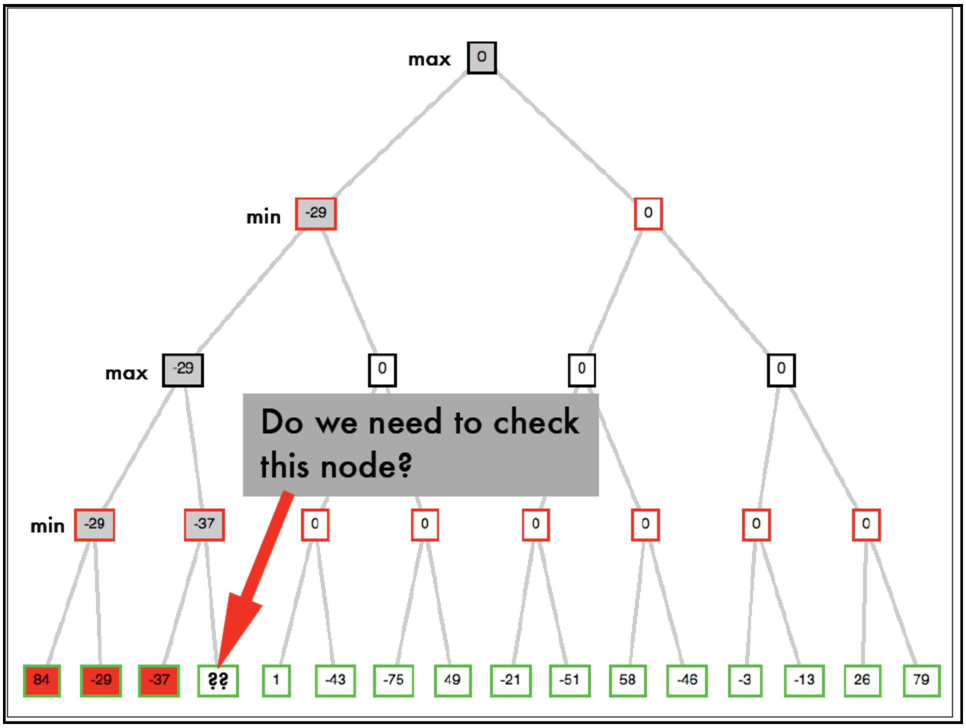
- Rolling the dice
- Expectiminimax search

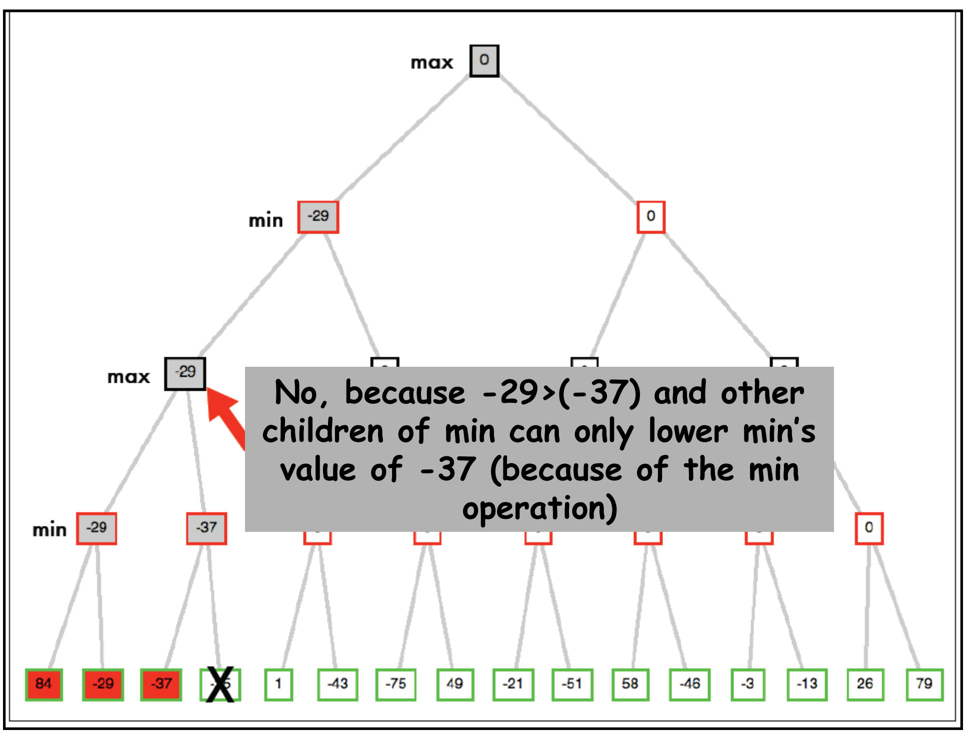
- To Do: Project #1 (due this Sunday!)

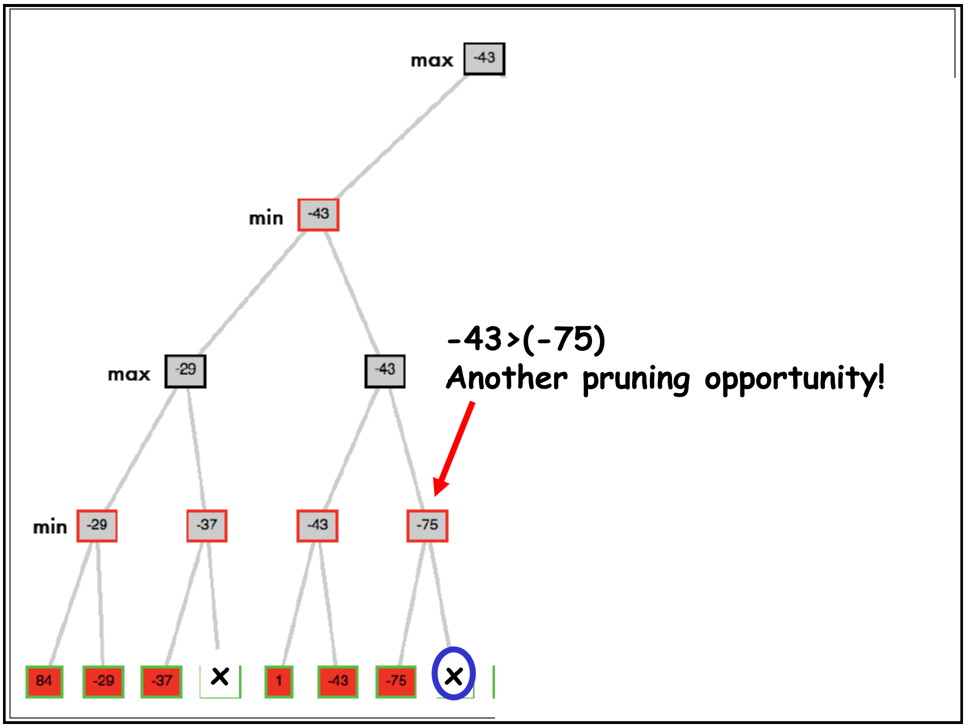
41

Exercise:
Prune this tree!

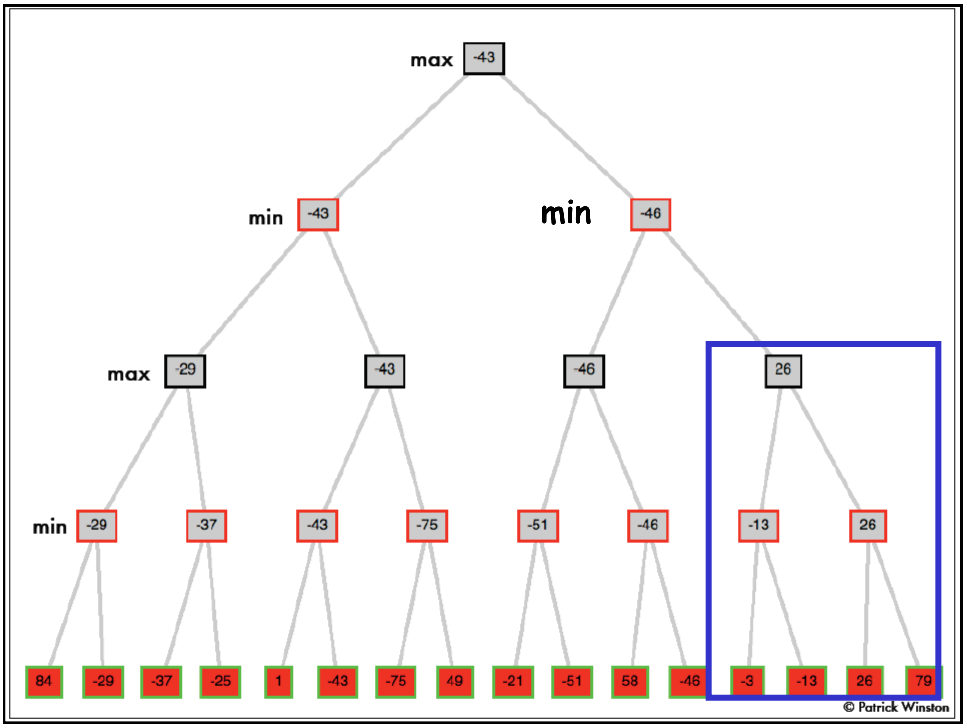


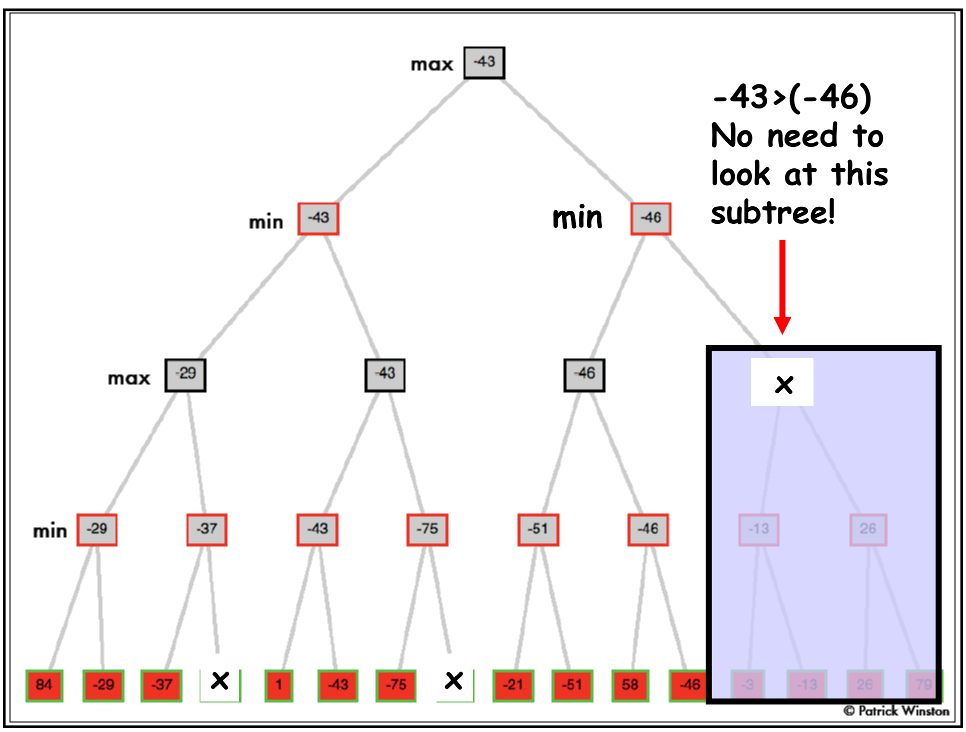

max
0
min
-29
0
max
-29
0
0
0
Do we need to check this node?
min
-29
-37
0
0
0
0
0
0
84
-29
-37
??
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-29
0
max
-29
No, because -29>(-37) and other children of min can only lower min's value of -37 (because of the min operation)
min
-29
-37
0
84
-29
-37
X
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79



Pruning can eliminate entire subtrees!


max
-43
min
-43
min
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston


max
-43
min
-43
min
-46
-43>(-46)
No need to
look at this
subtree!
max
-29
-43
-46
X
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
X
1
-43
-75
X
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston