CS 4260 and CS 5260
Vanderbilt University

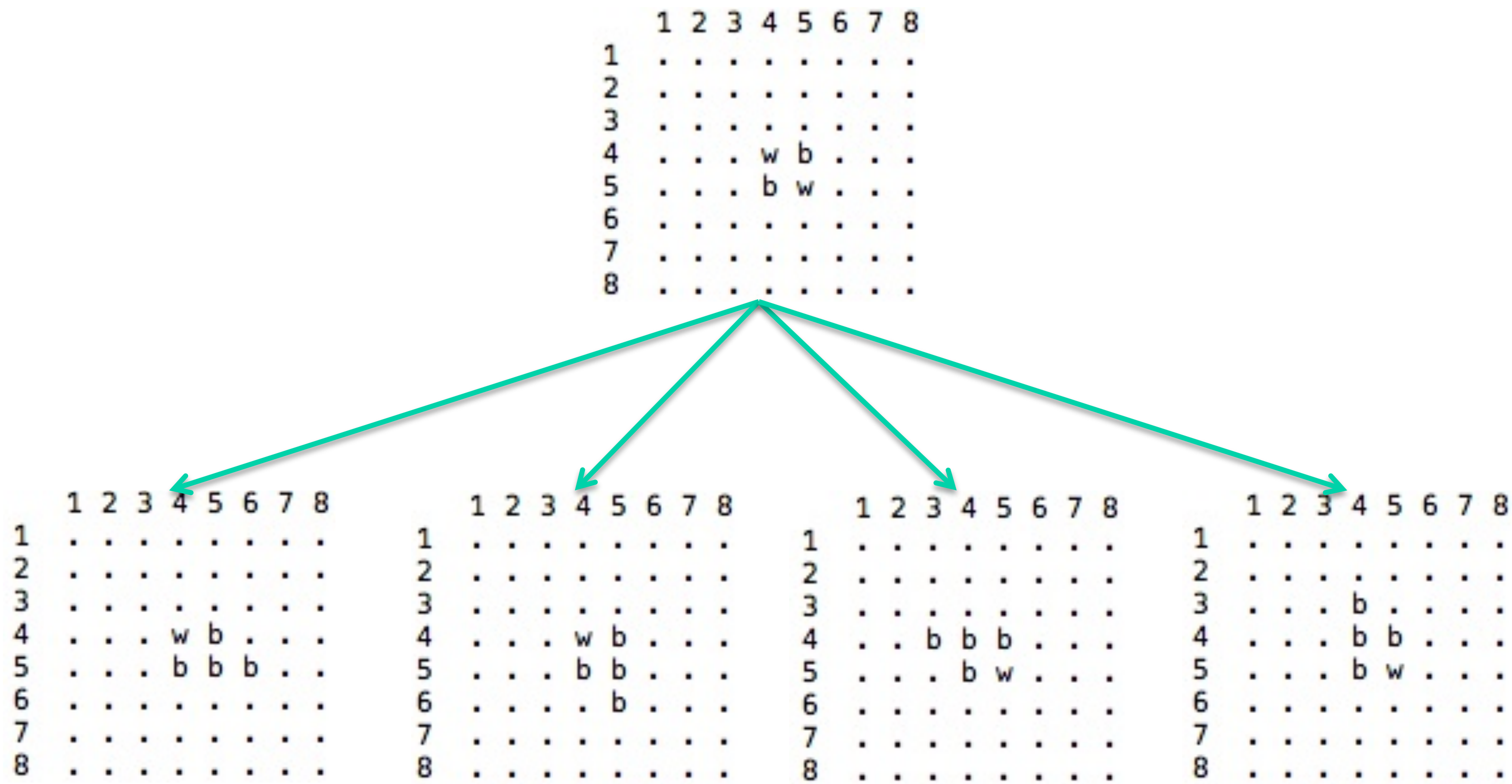
Lecture on Adversarial Search in Games

This lecture covers material in optional reading from section 11.3 of ArtInt

ArtInt: Poole and Mackworth, Artificial Intelligence 2E
at http://artint.info/2e/html/ArtInt2e.html
to include slides at http://artint.info/2e/slides/ch04/lect1.pdf

Douglas H. Fisher

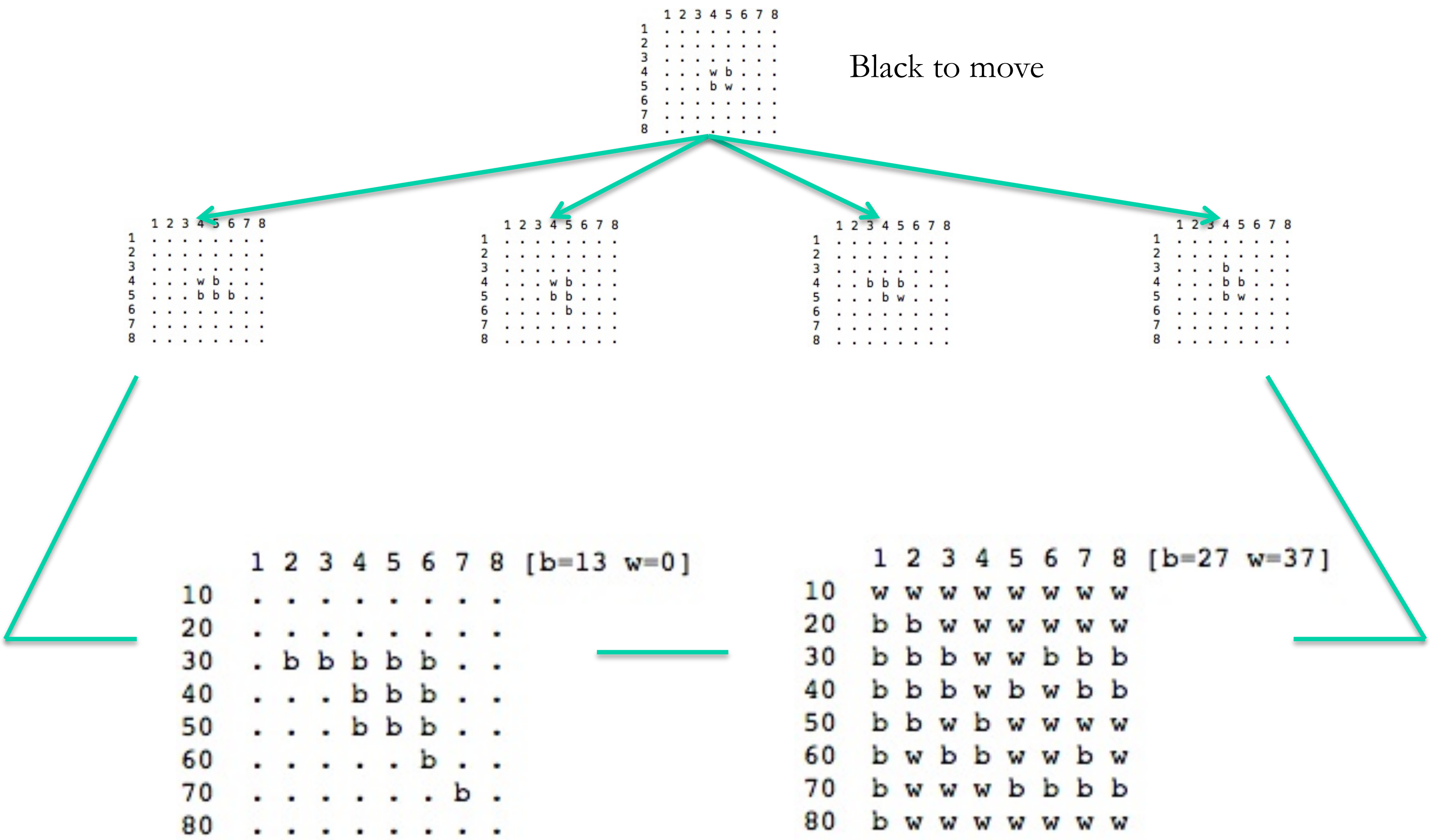
In game of Othello black moves first, but where?



Search ahead

Douglas H. Fisher

Game ends when no player has a move



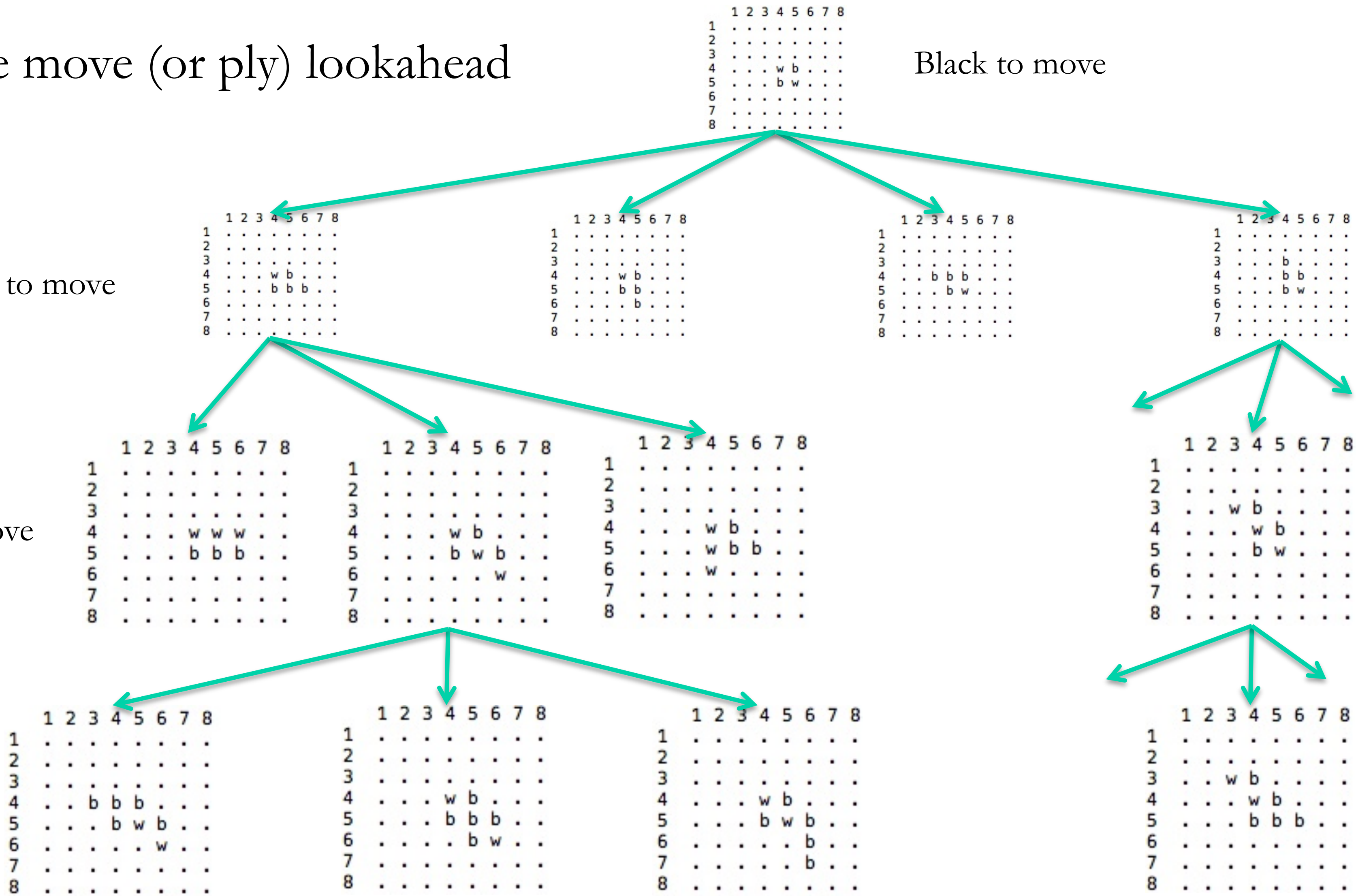
Douglas H. Fisher

Three move (or ply) lookahead

Black to move

White to move

Black to move



5-2 = 3 tile advantage to black

5-2=3

5-2=3

5-2=3

Using simple difference utility function doesn't discriminate at shallow levels

Douglas H. Fisher

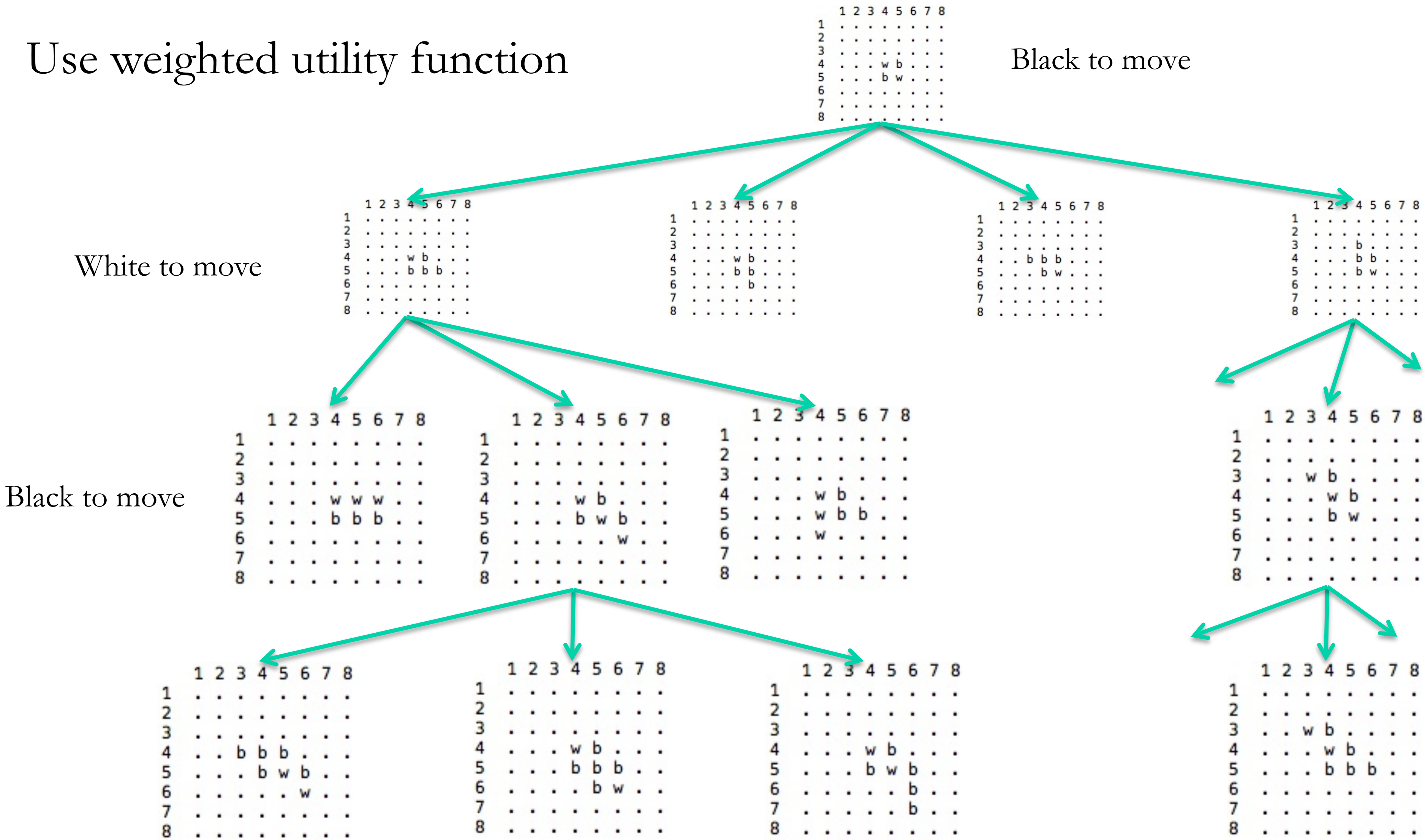
Some squares are more important than others

```
{0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 0, 120, -20,  20,   5,   5,  20, -20, 120,   0,
 0, -20, -40,  -5,  -5,  -5,  -5, -40, -20,   0,
 0,  20,  -5,  15,   3,   3,  15,  -5,  20,   0,
 0,   5,  -5,   3,   3,   3,   3,  -5,   5,   0,
 0,   5,  -5,   3,   3,   3,   3,  -5,   5,   0,
 0,  20,  -5,  15,   3,   3,  15,  -5,  20,   0,
 0, -20, -40,  -5,  -5,  -5,  -5, -40, -20,   0,
 0, 120, -20,  20,   5,   5,  20, -20, 120,   0,
 0,   0,   0,   0,   0,   0,   0,   0,   0,   0}
```

Use a difference of weighted sums

Due to Peter Norvig in “Artificial Intelligence Programming” 1992, Morgan Kaufmann Publishers

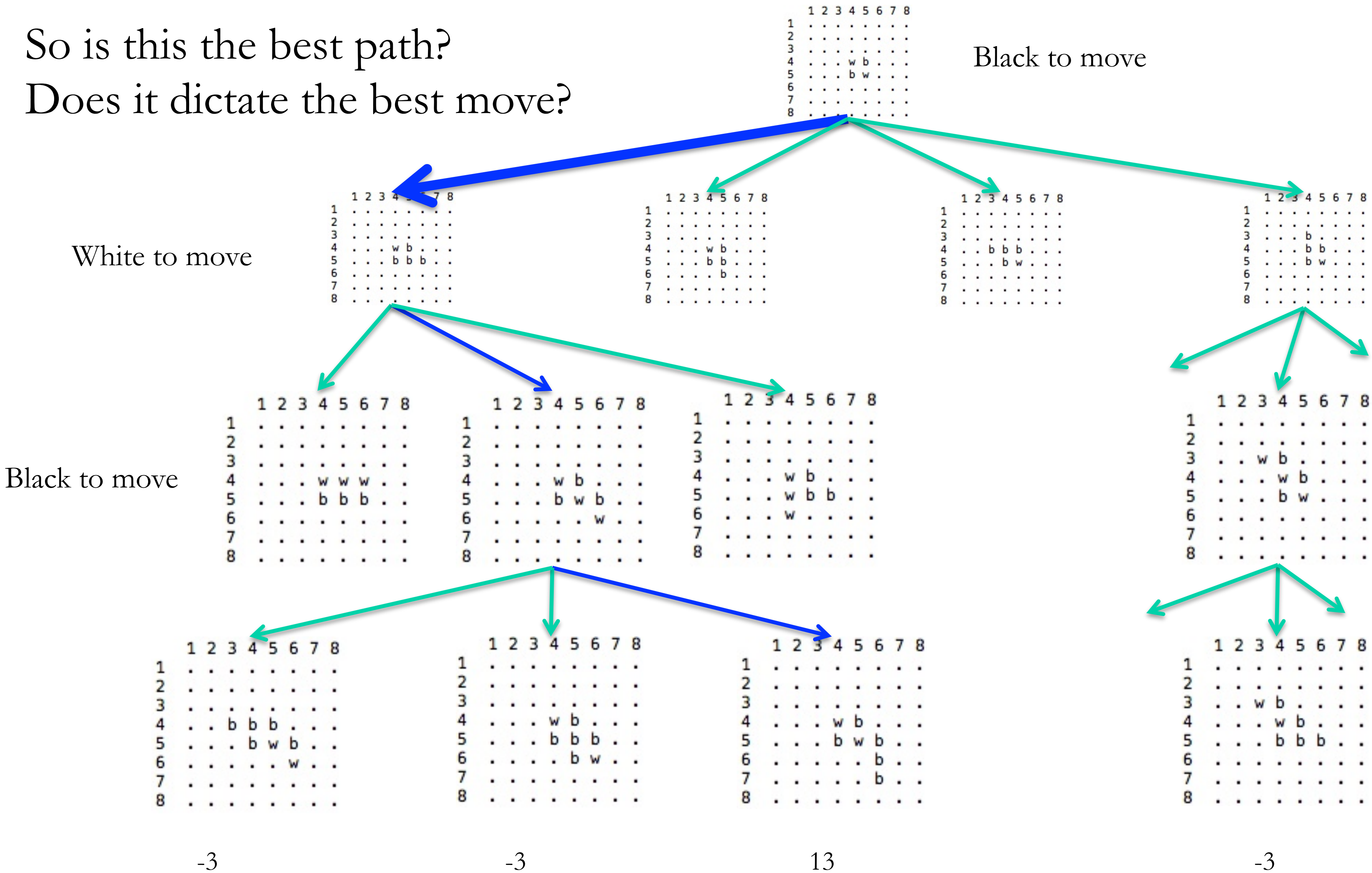
Use weighted utility function



3+3+3 +3-3+3 -15 = -3	-3+3 +3+3+3 +3-15 =-3	-3+3 +3-3+3 +15 -5 = 13	-15+3 -3+3 +3+3+3 =-3

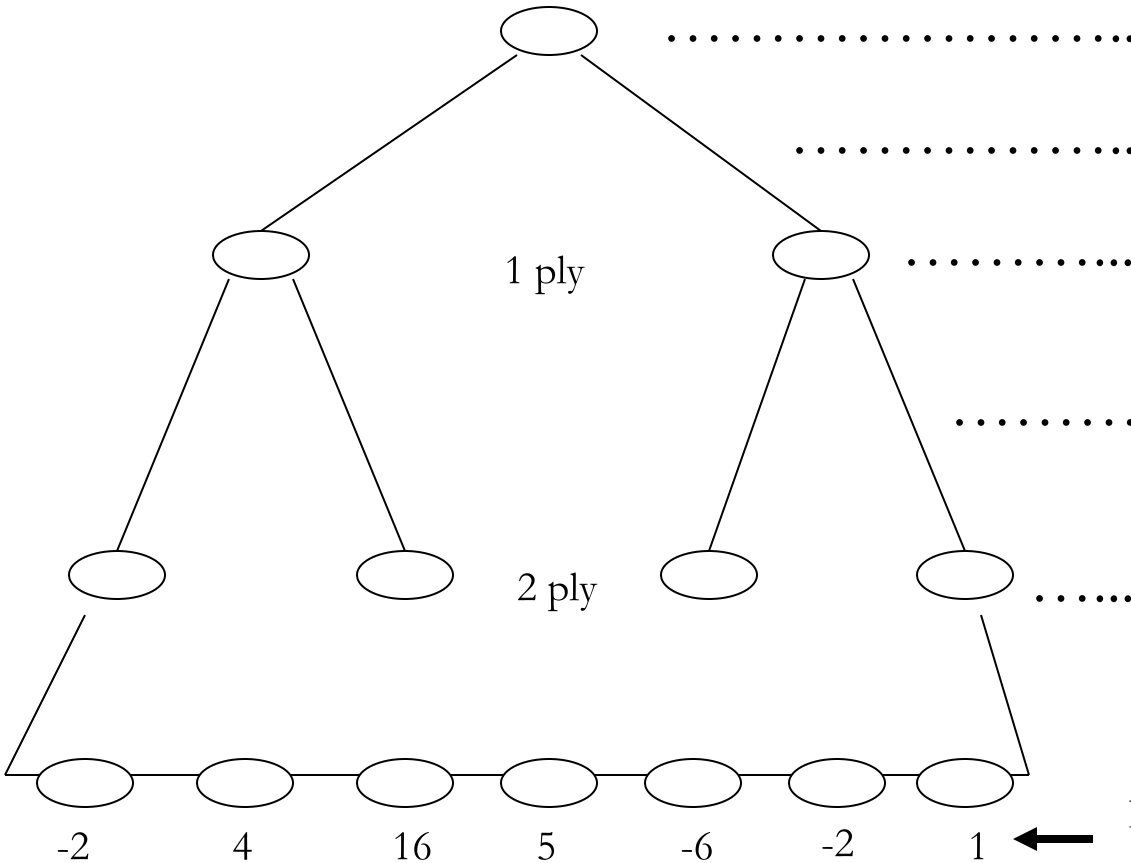
Douglas H. Fisher

So is this the best path?
Does it dictate the best move?



Probably not! Opponent won't cooperate

Douglas H. Fisher



Original board position

Moves available to first player
(the player to move next)

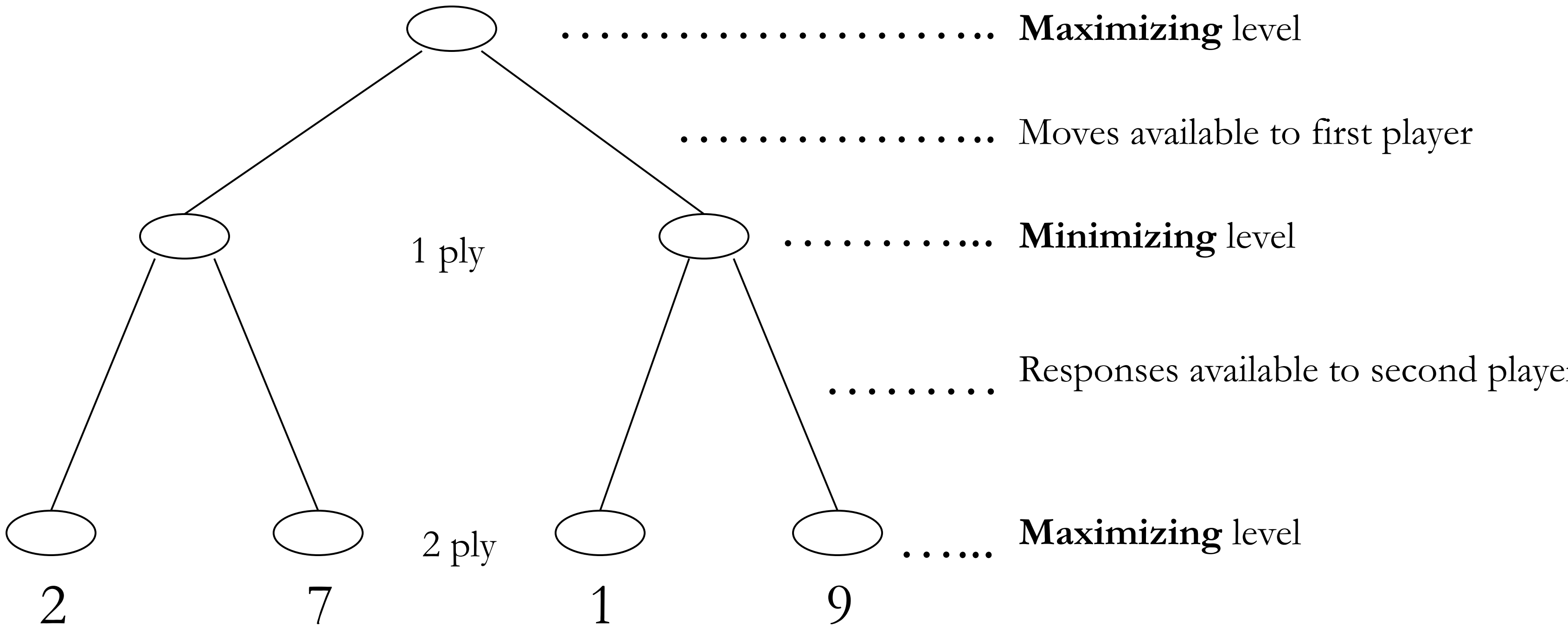
New board situations

Responses available to second player

New board situations

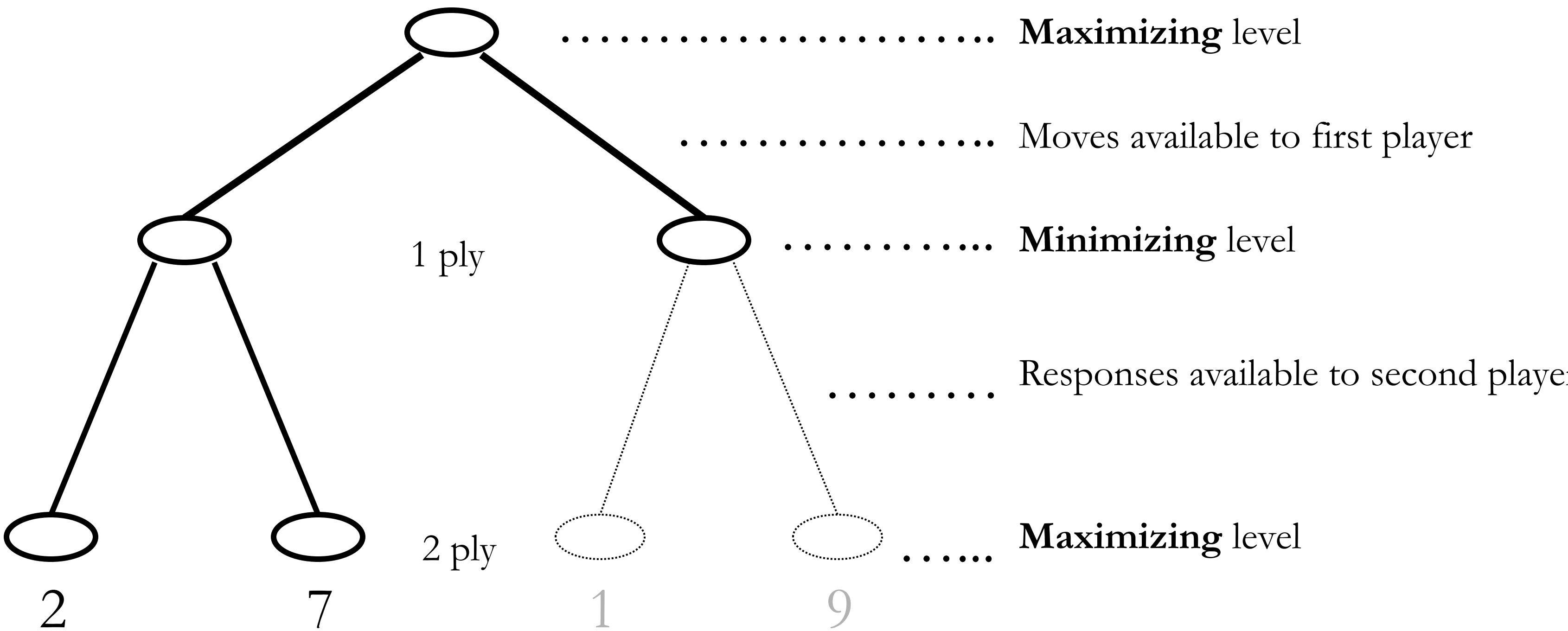
Resulting scores of boards N ply deep
obtained by utility function

Douglas H. Fisher



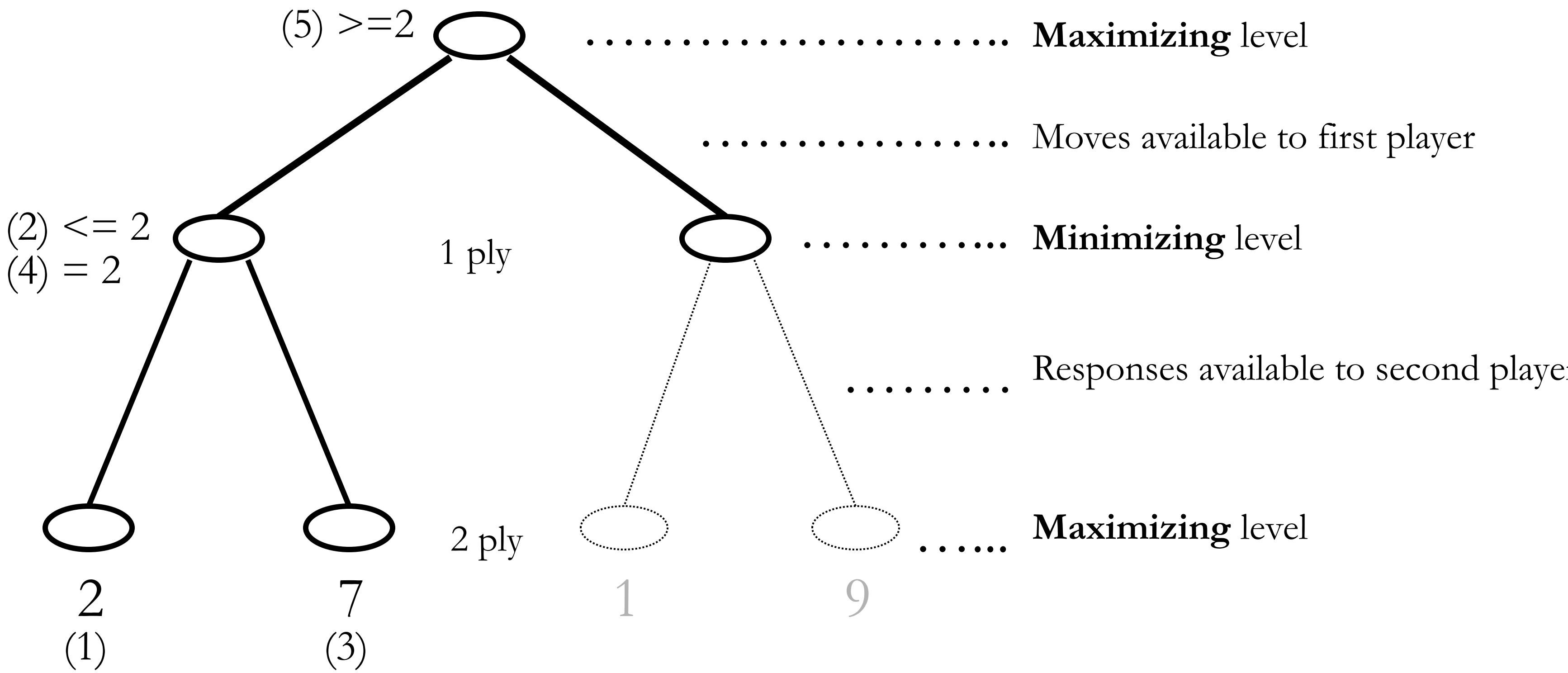
Assumptions: larger scores better for player (Max).
smaller scores better for opponent (Min)
utility function is rational

Douglas H. Fisher



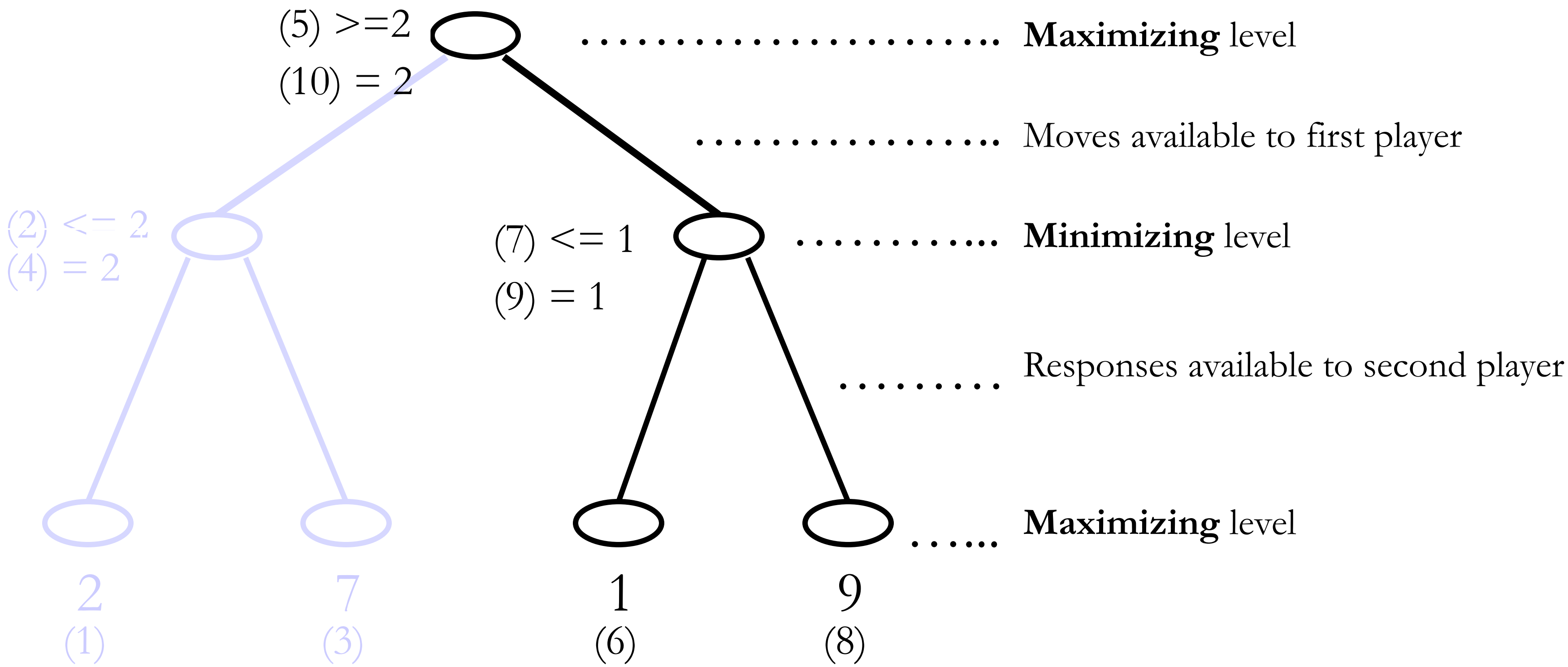
Minimax search with no pruning

Douglas H. Fisher



Minimax search with no pruning

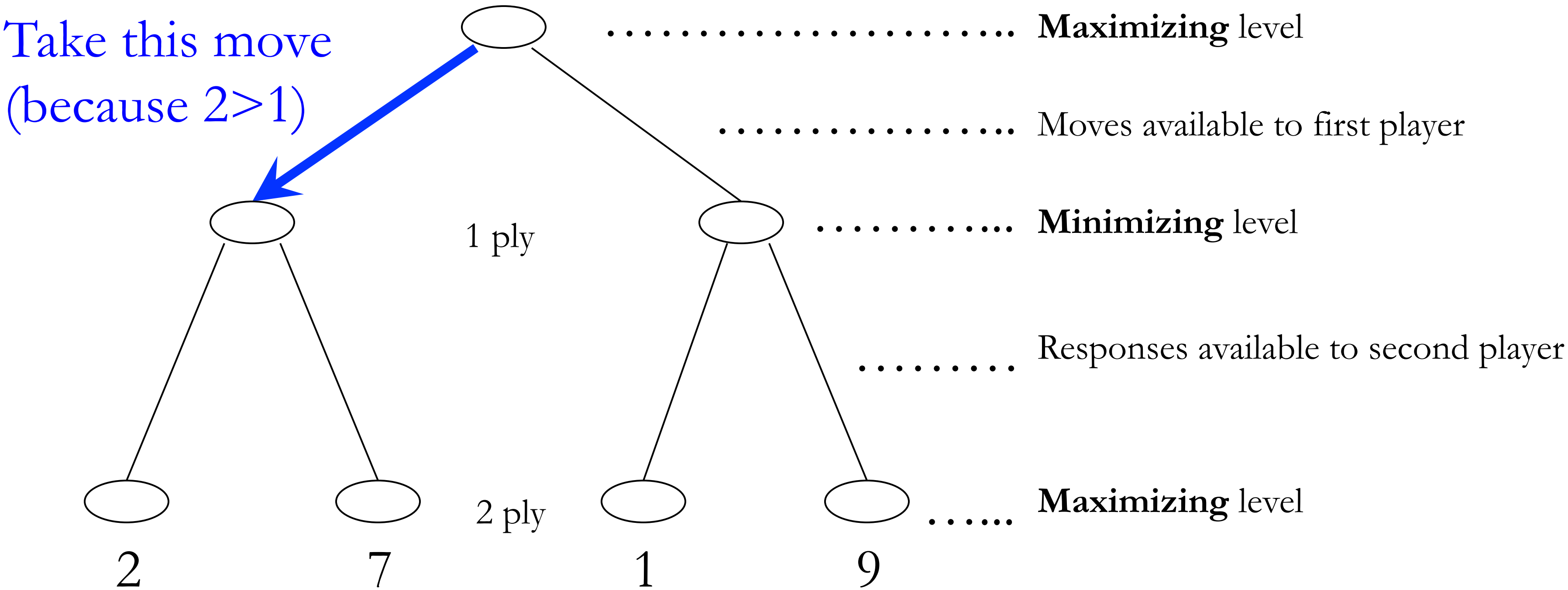
Douglas H. Fisher



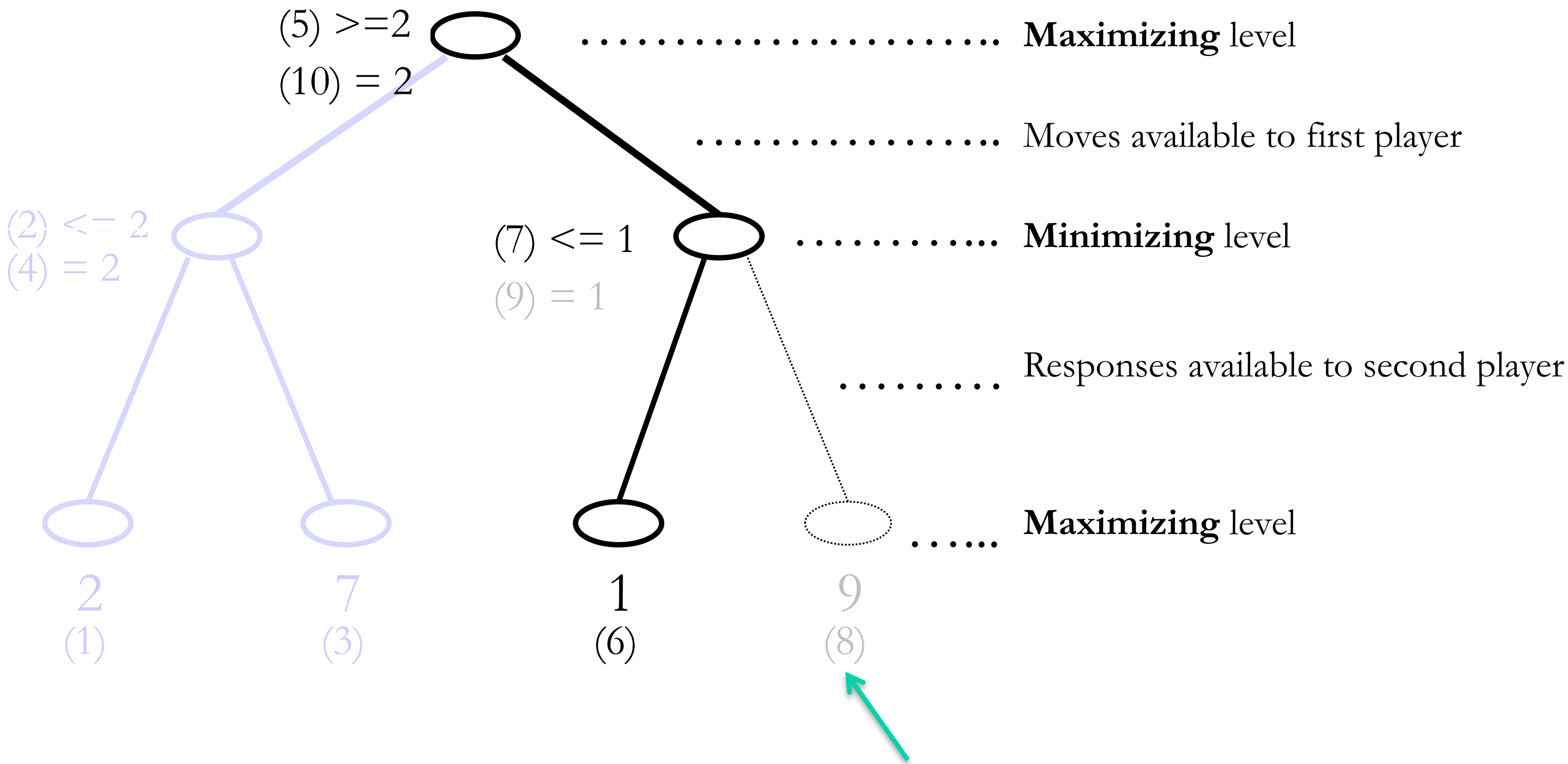
Minimax search with no pruning

All this reasoning is going on in “mind” of AI about to move

Douglas H. Fisher

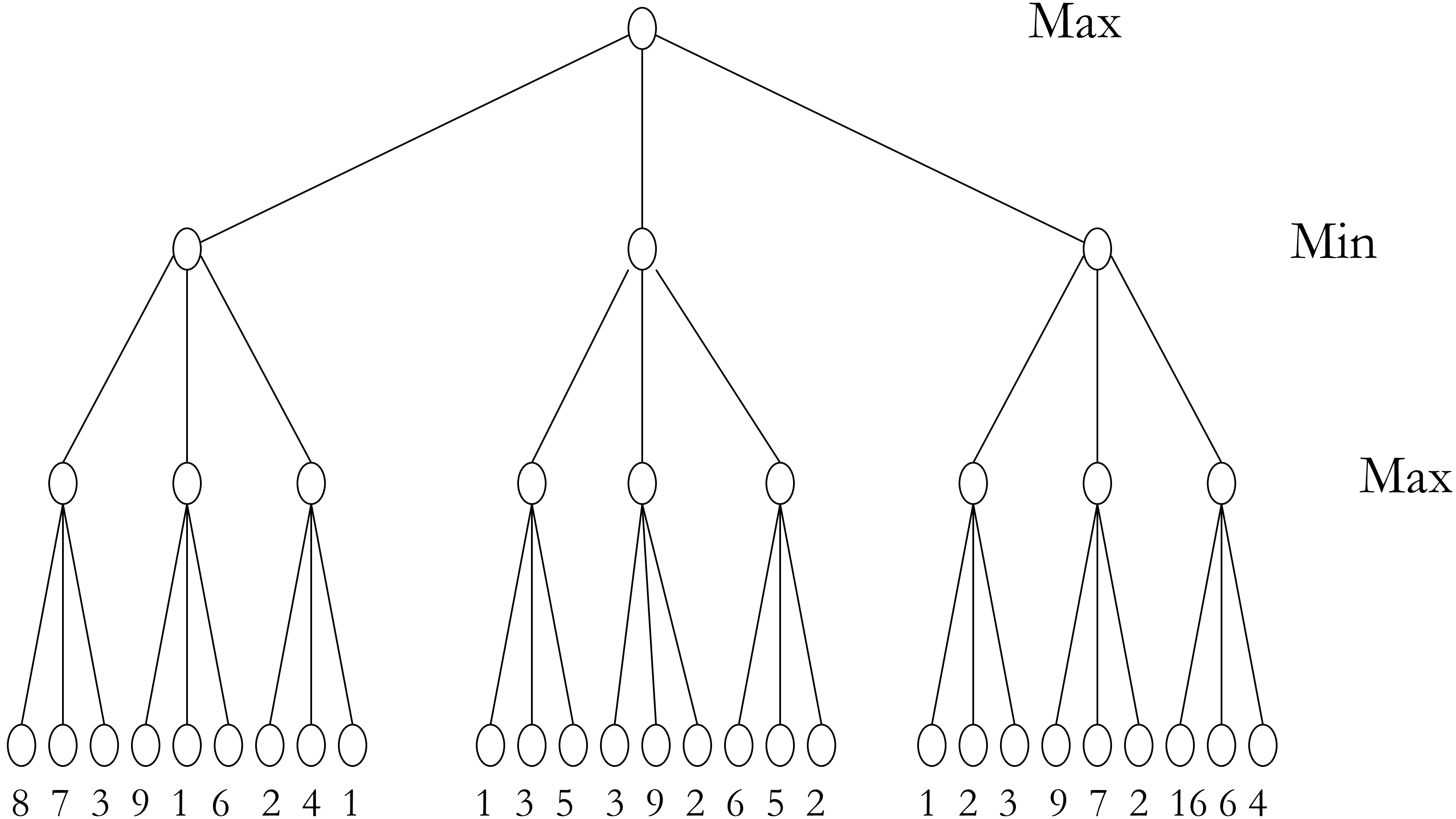


Douglas H. Fisher



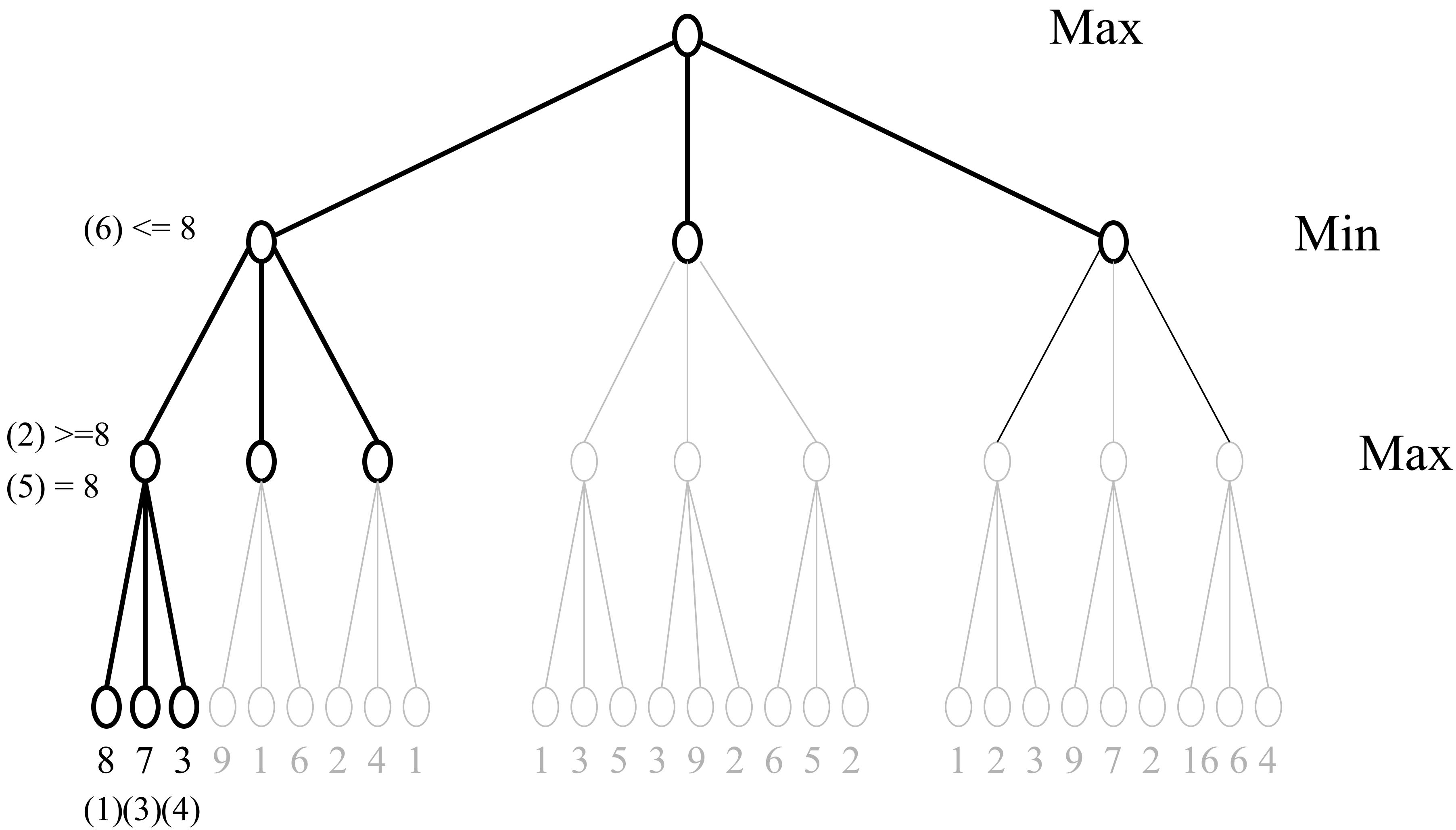
Is there any need to examine this node (i.e., game board)?

No – the maximizing player will not move in that direction anyways

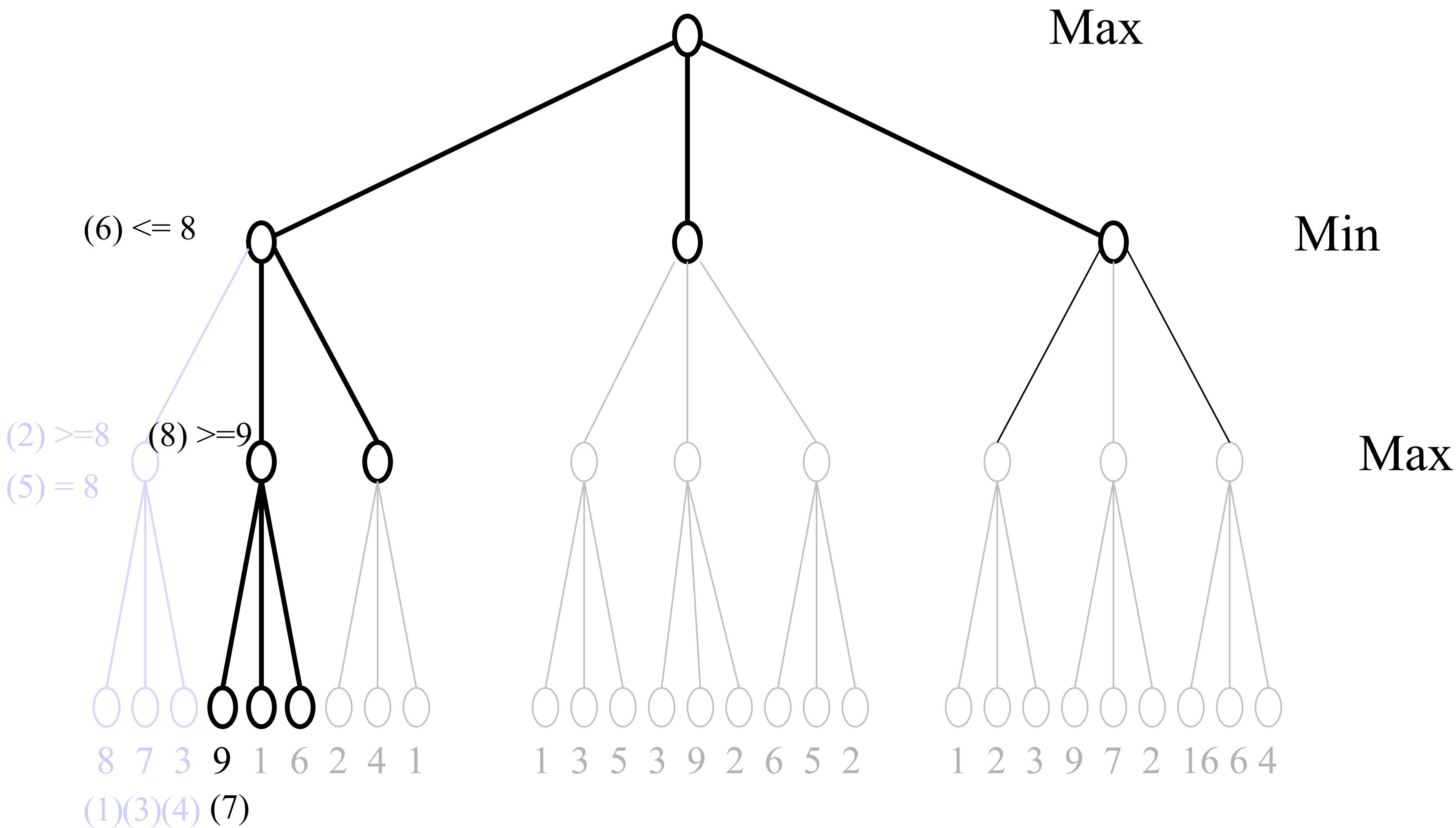


Minimax search with (apha beta) pruning

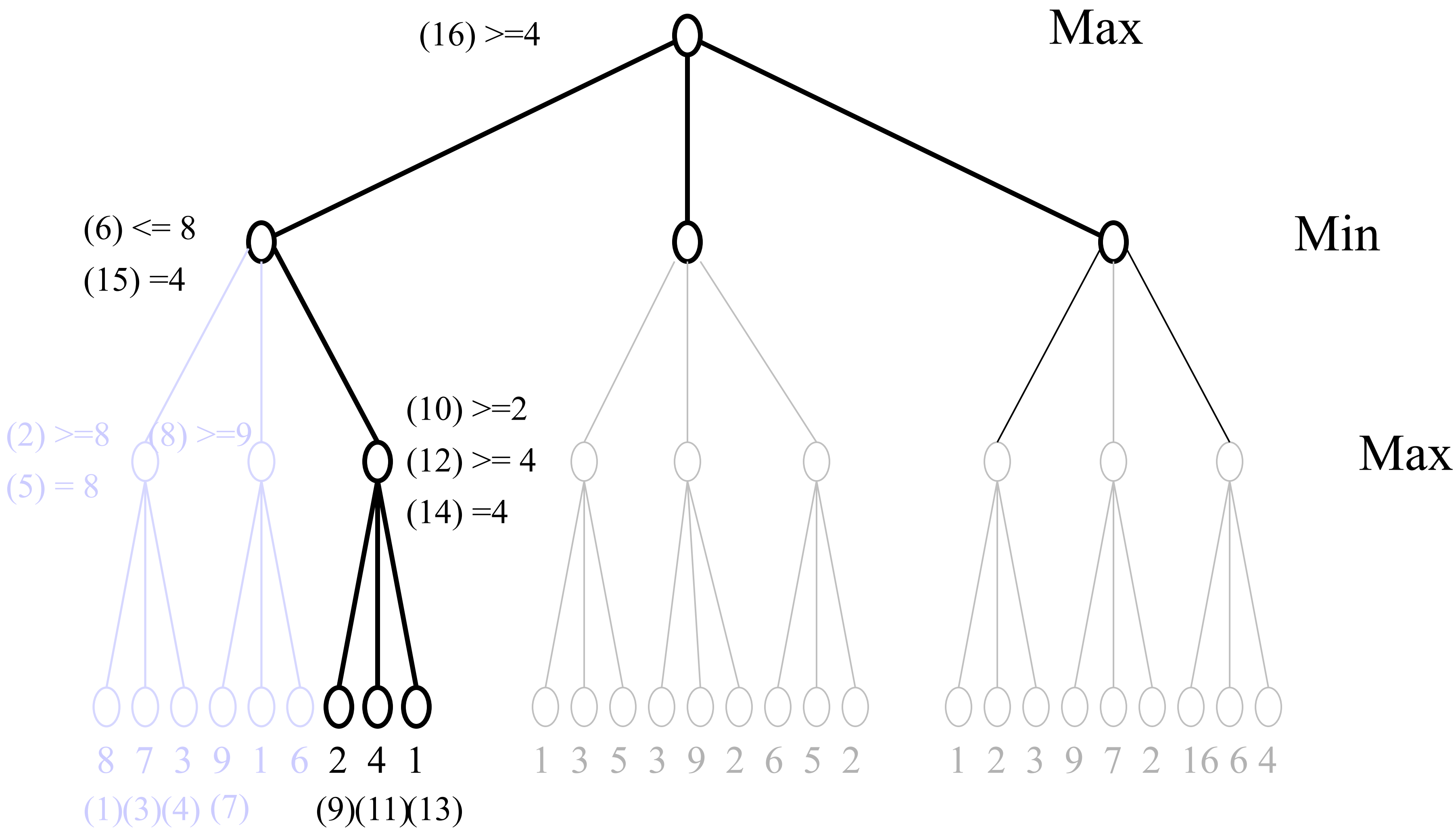
Douglas H. Fisher



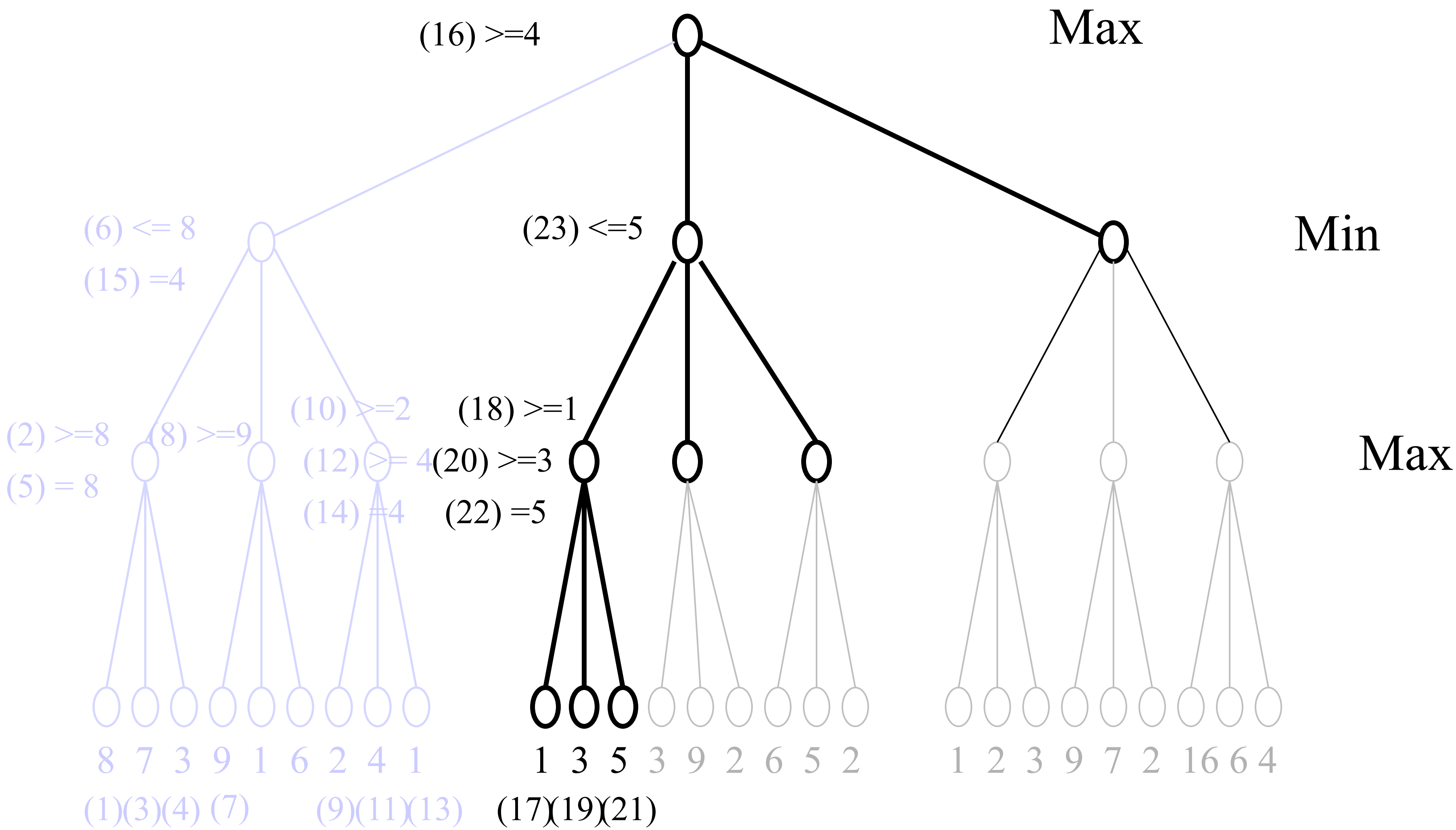
Douglas H. Fisher



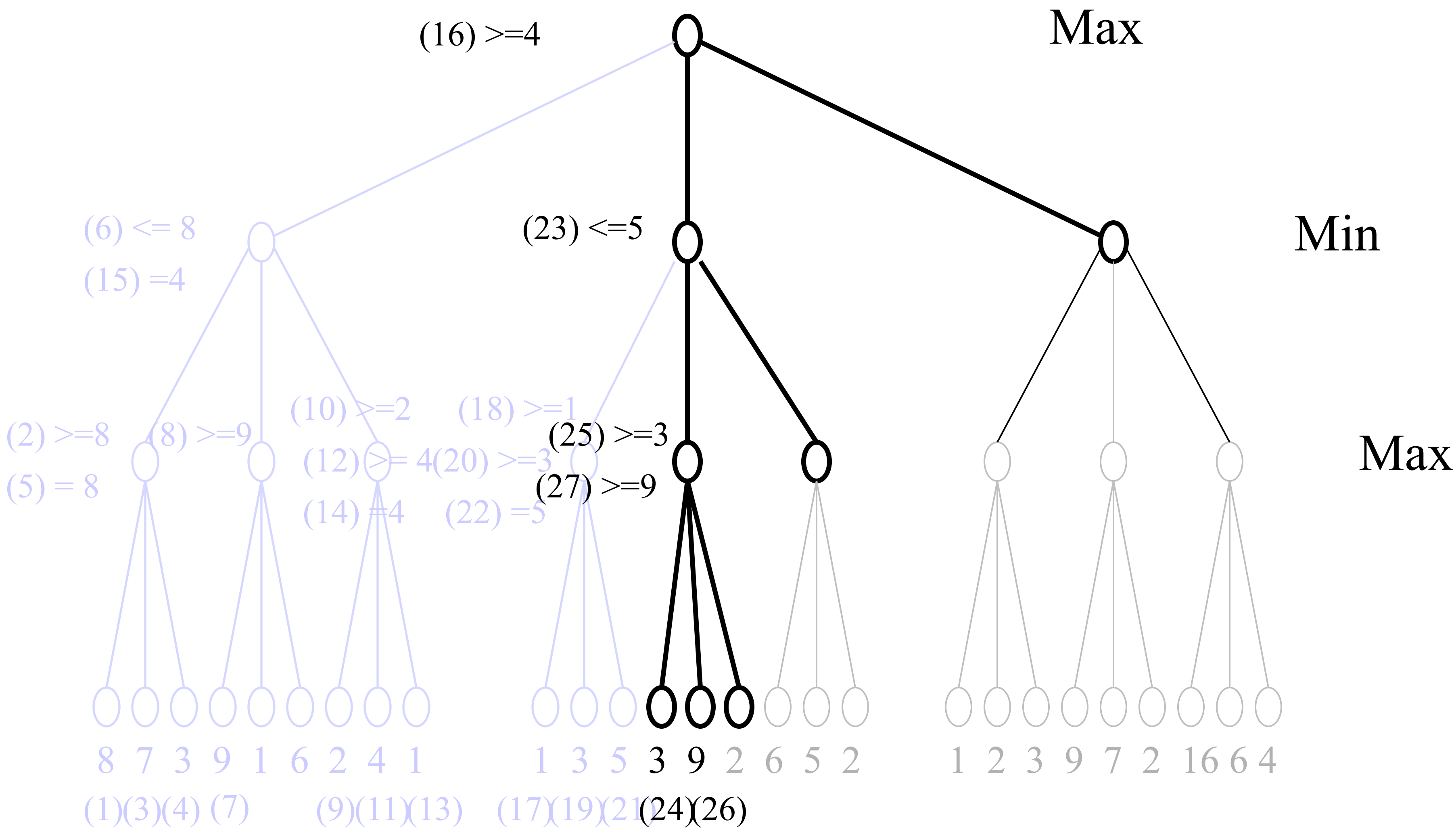
Douglas H. Fisher



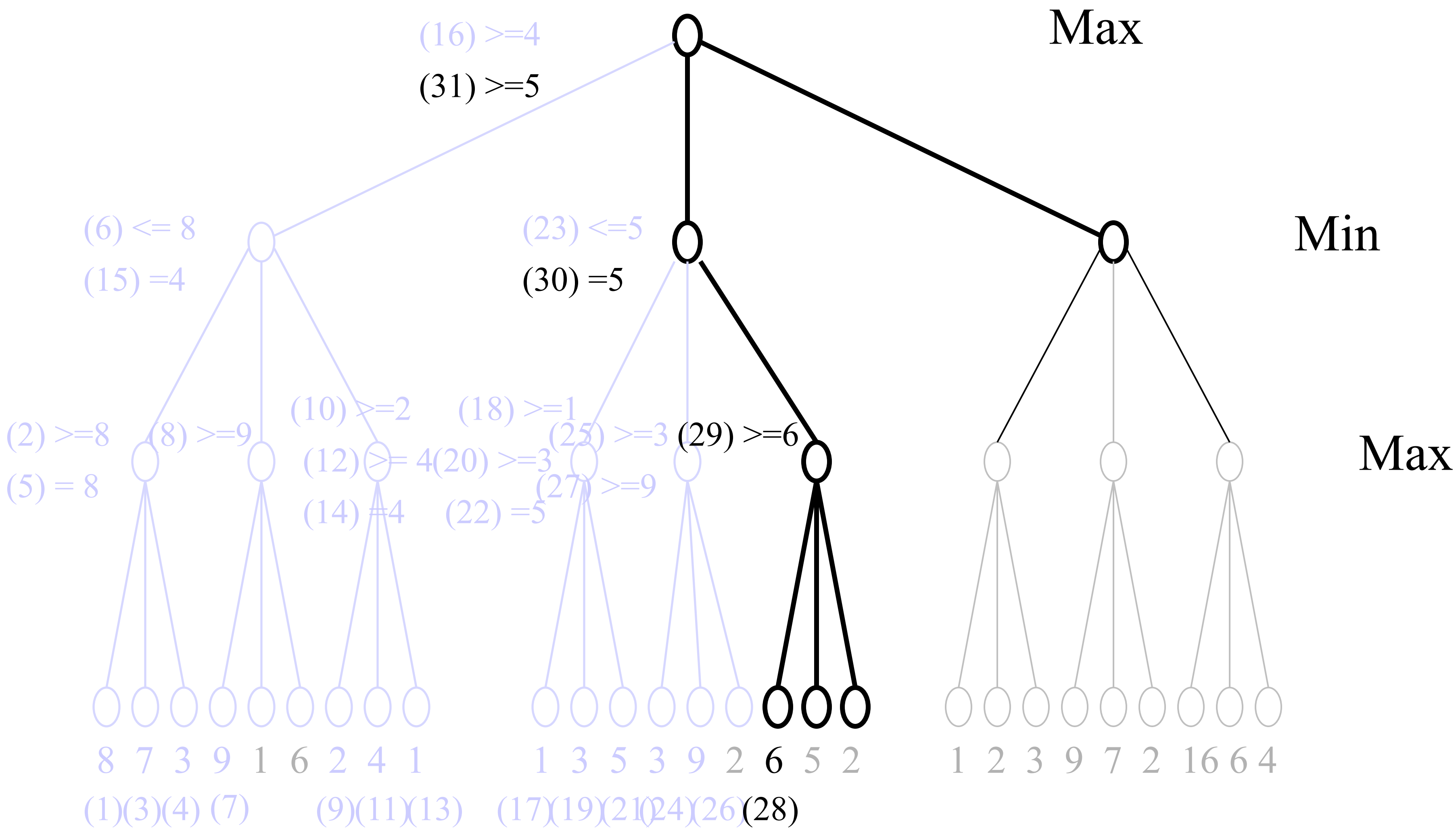
Douglas H. Fisher



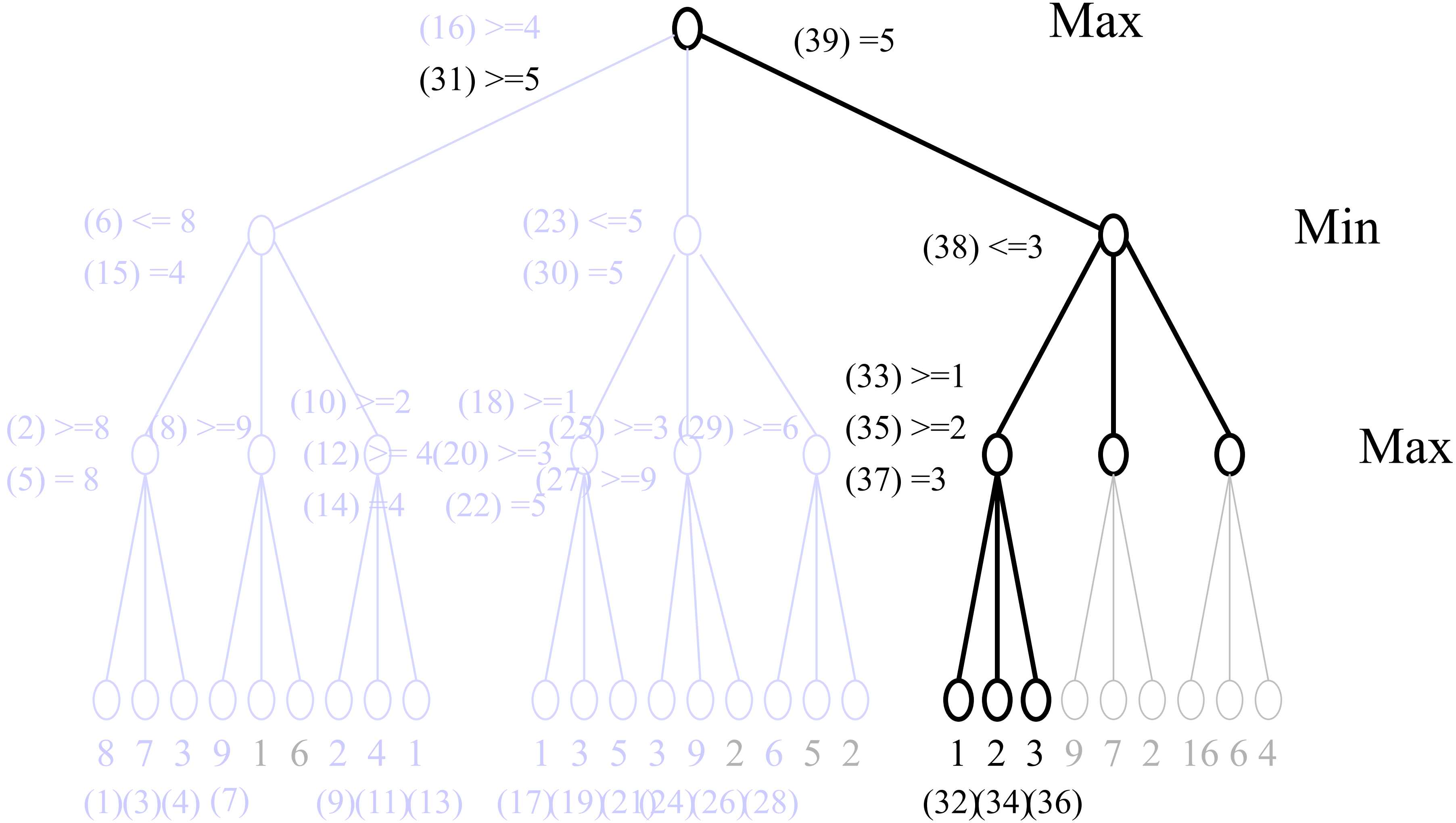
Douglas H. Fisher



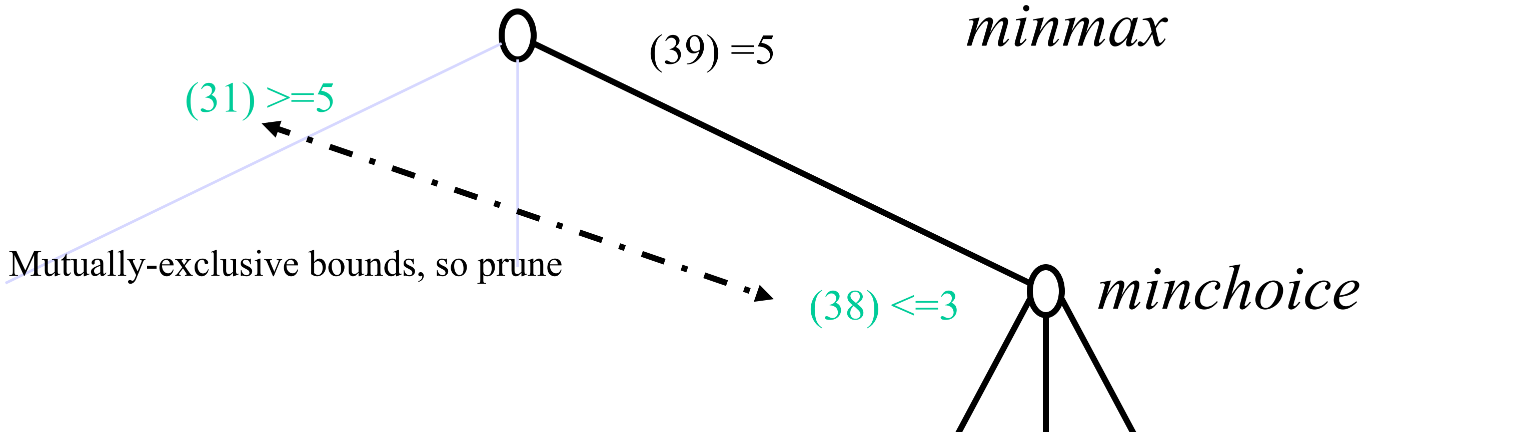
Douglas H. Fisher

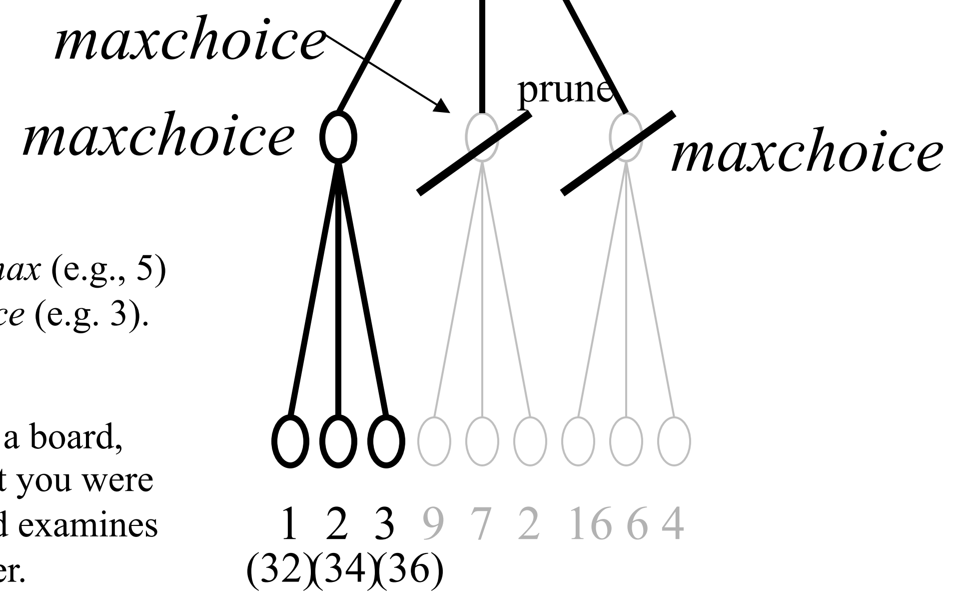


Douglas H. Fisher



Take middle move



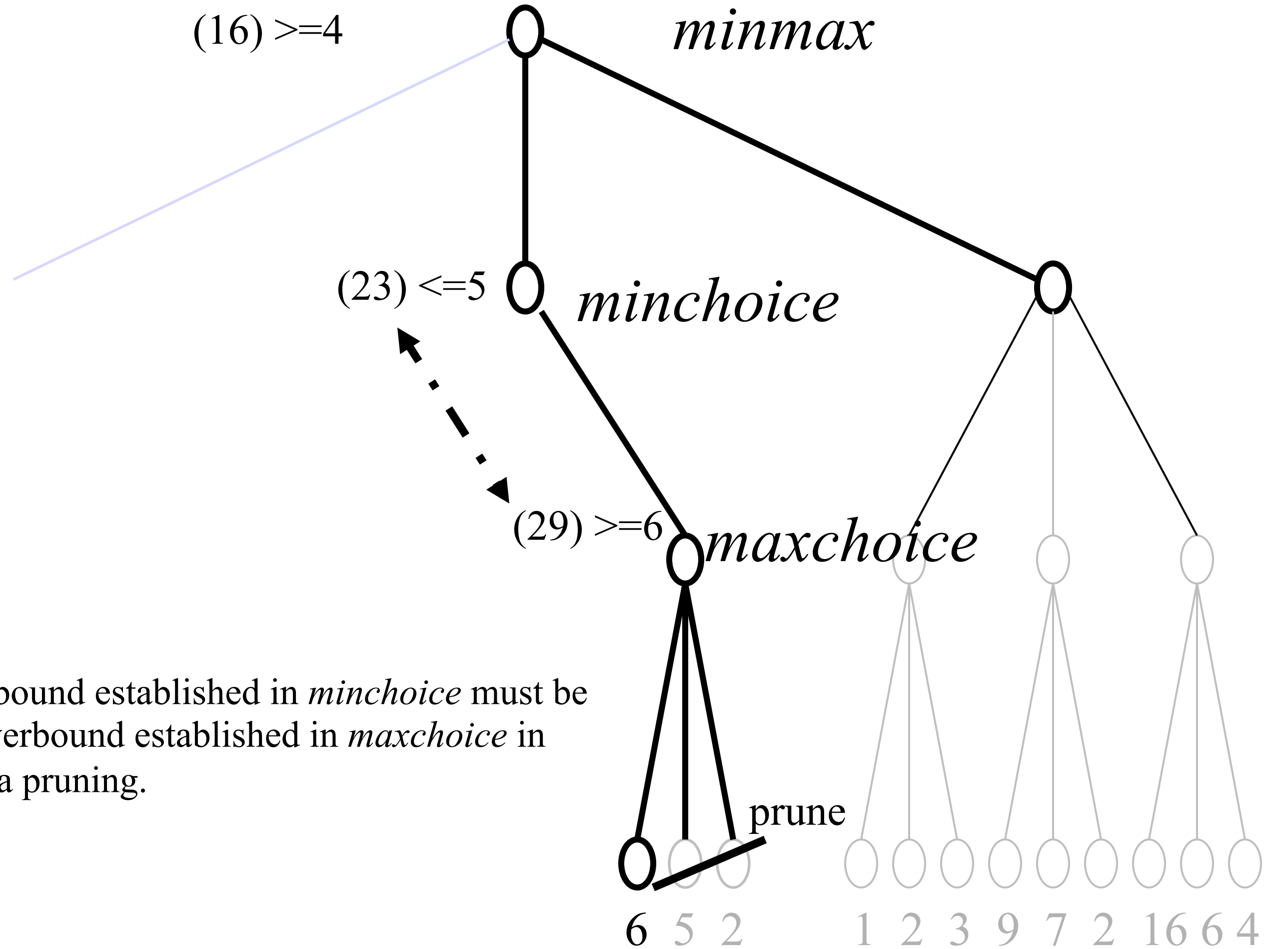


ASIDE: In sample Othello code, *minmax* corresponds to the top-level call, and minchoice and maxchoice are mutually recursive. Implementing alpha-beta pruning requires, in part, that the lower bound established in *minmax* (e.g., 5) be compared with the upperbound established in *minchoice* (e.g. 3).

In class, I talked about generating all successor boards of a board, then examining one successor after another. The code that you were given, actually gets a queue of moves, then generates and examines the boards that result from these moves, one after the other.

This slide is specific to sample Othello code – you won' be examined on it!

This is a C implementation of a variation on a Lisp version by Peter Norvig in "Artificial Intelligence Programming" 1992, Morgan Kaufmann Publishers

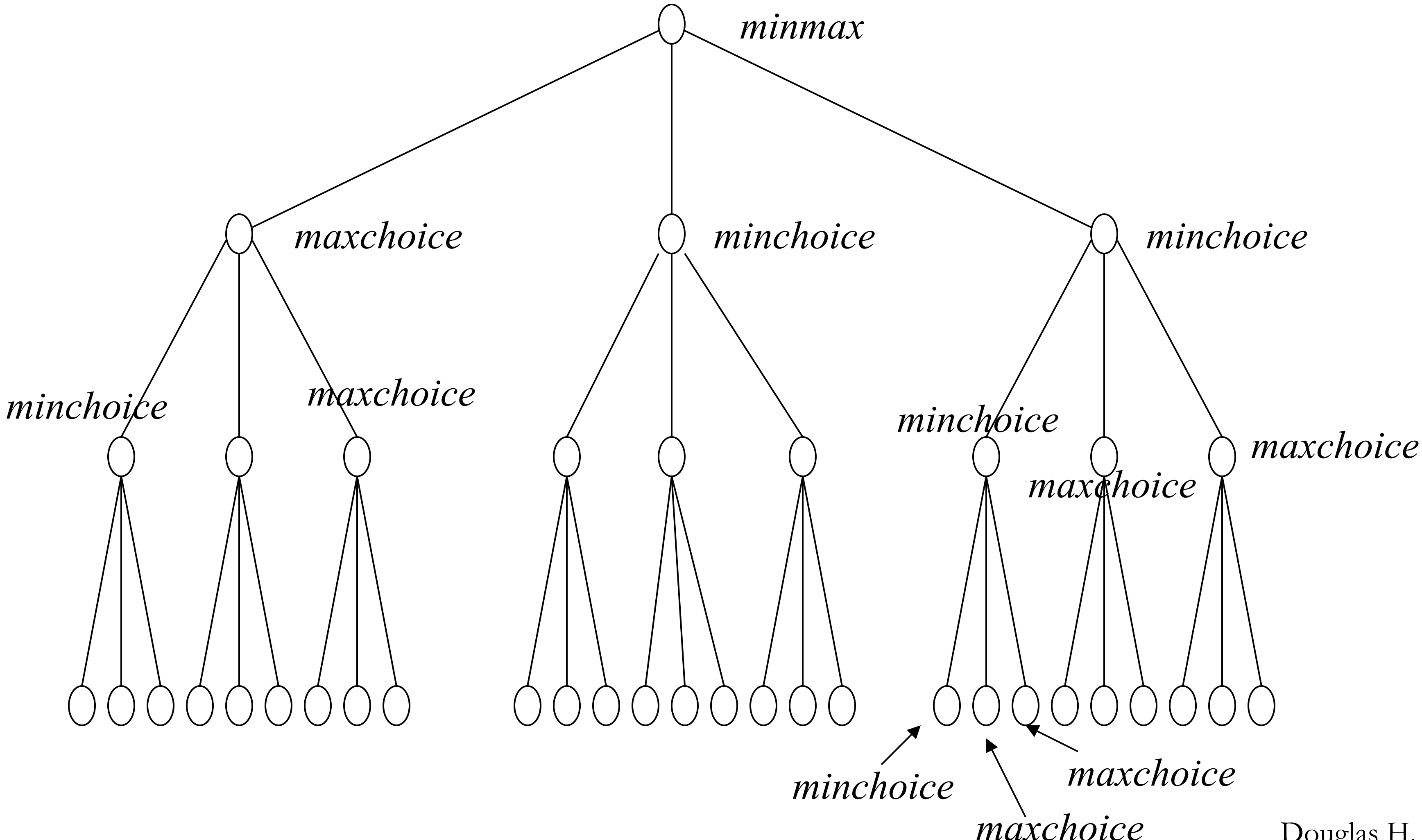


Similarly, the upperbound established in *minchoice* must be compared to the lowerbound established in *maxchoice* in order to do alphabeta pruning.

This slide is specific to sample Othello code – you won' be examined on it!

This slide is specific to sample Othello code – you won' be examined on it!

Finally, remember that in Othello, player **i** may not have any legal moves, so the other player, **k**, continues to move until player **i** can once again move. As a result, *minchoice* will sometimes call *minchoice* and *maxchoice* will sometimes call *maxchoice*.



This slide is specific to sample Othello code – you won' be examined on it!

If you run roundrobin with the current code (minmax with no pruning), then be prepared to wait quite a while. The following will eventually materialize (which I have indented for a bit better readability):

```
random wins=1 boards=0          || diff/1 wins=9 boards=1824
random wins=2 boards=0          || diff/3 wins=8 boards=174938
random wins=0 boards=0          || diff/5 wins=10 boards=14970716
random wins=3 boards=0          || wdiff/1 wins=7 boards=2455
random wins=1 boards=0          || wdiff/3 wins=9 boards=293989
random wins=0 boards=0          || wdiff/5 wins=10 boards=33035881
diff/1 wins=0 boards=1799       || diff/3 wins=10 boards=155657
diff/1 wins=0 boards=1763       || diff/5 wins=10 boards=16361232
diff/1 wins=4 boards=2134       || wdiff/1 wins=6 boards=2397
diff/1 wins=2 boards=2038       || wdiff/3 wins=7 boards=268585
diff/1 wins=0 boards=1736       || wdiff/5 wins=10 boards=24682590
diff/3 wins=3 boards=200450     || diff/5 wins=6 boards=19828063
diff/3 wins=3 boards=202781     || wdiff/1 wins=7 boards=2301
diff/3 wins=1 boards=208165     || wdiff/3 wins=9 boards=242318
diff/3 wins=1 boards=174285     || wdiff/5 wins=9 boards=26257282
diff/5 wins=2 boards=18595500   || wdiff/1 wins=8 boards=2383
diff/5 wins=4 boards=19102429   || wdiff/3 wins=5 boards=202807
diff/5 wins=0 boards=16349918   || wdiff/5 wins=10 boards=22520544
wdiff/1 wins=0 boards=2311      || wdiff/3 wins=10 boards=292058
wdiff/1 wins=1 boards=1944      || wdiff/5 wins=9 boards=29422051
wdiff/3 wins=1 boards=234330    || wdiff/5 wins=8 boards=28452315
```

Each line above shows the results of 10 matches against two strategies. For example, the last line shows the results of the strategy using 3-ply lookahead with weighteddiffeval (wdiff/3) as the utility function against 5-ply lookahead with the same utility function (wdiff/5). Over 10 games, wdiff/5 wins 8, wdiff/3 wins 1, and there is one tie. Over 10 games, wdiff/5 (in this last case) generates 28,452,315 boards (or an average of 2,845,232 boards generated per game)!

This slide is specific to sample Othello code – you won' be examined on it!

```
random wins=1 boards=0          || diff1 wins=9 boards=1824
random wins=2 boards=0          || diff3 wins=8 boards=44887
random wins=0 boards=0          || diff5 wins=10 boards=944422
random wins=3 boards=0          || wdiff1 wins=7 boards=2455
random wins=1 boards=0          || wdiff3 wins=9 boards=97418
random wins=0 boards=0          || wdiff5 wins=10 boards=2839502
diff1 wins=0 boards=1799        || diff3 wins=10 boards=44741
diff1 wins=0 boards=1763        || diff5 wins=10 boards=1038063
diff1 wins=4 boards=2134        || wdiff1 wins=6 boards=2397
diff1 wins=2 boards=2038        || wdiff3 wins=7 boards=90605
diff1 wins=0 boards=1736        || wdiff5 wins=10 boards=2643711
diff3 wins=3 boards=59175       || diff5 wins=6 boards=1160722
diff3 wins=3 boards=51624       || wdiff1 wins=7 boards=2301
diff3 wins=1 boards=58869       || wdiff3 wins=9 boards=84970
diff3 wins=1 boards=48521       || wdiff5 wins=9 boards=2947041
diff5 wins=2 boards=1142881     || wdiff1 wins=8 boards=2383
diff5 wins=4 boards=1063738     || wdiff3 wins=5 boards=76715
diff5 wins=0 boards=1124236     || wdiff5 wins=10 boards=2683518
wdiff1 wins=0 boards=2311       || wdiff3 wins=10 boards=89526
wdiff1 wins=1 boards=1944       || wdiff5 wins=9 boards=2635355
wdiff3 wins=1 boards=80708      || wdiff5 wins=8 boards=2844060
```

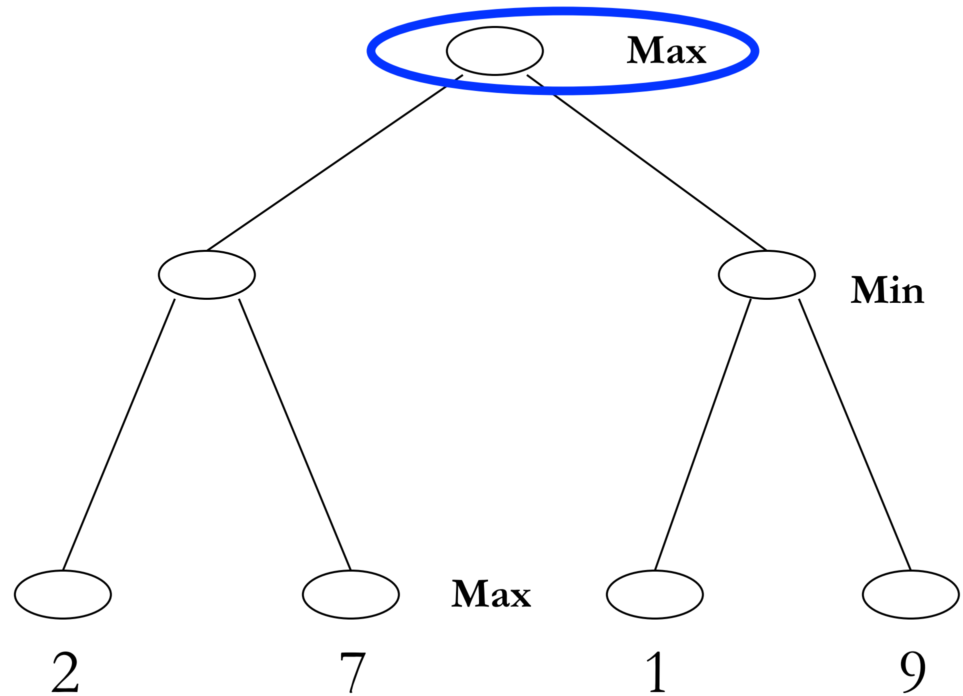
Notice that the win/loss results do not change at all (minmax with alphabeta pruning chooses exactly the same moves as minmax without alphabeta pruning), but the number of generated boards is vastly reduced. For example, when wdiff5 is pitted against wdiff3, wdiff5 with pruning generates 2,844,060 boards over 10 games or an average of 284,406 boards per game, as opposed to about 10 times this without pruning.

Exercise: understand the book's algorithm on examples

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

$$MinimaxAlphaBeta\,(R, -\infty, \infty)$$



From ArtInt (Poole and Mackworth) Section 11.3

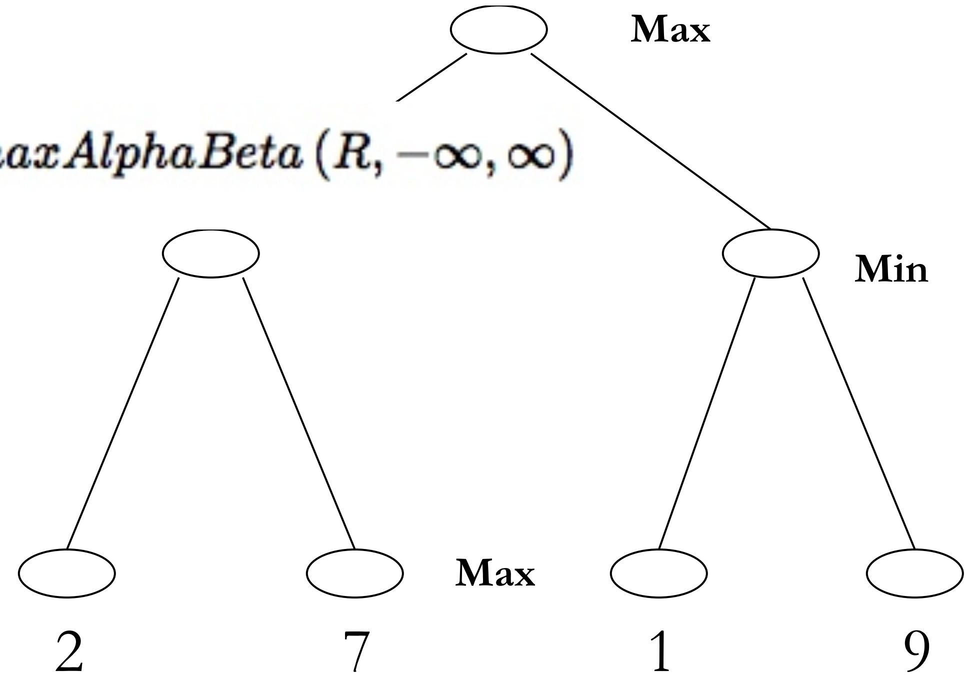
Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
     else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
     else  // n is a MIN node
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

$MinimaxAlphaBeta\,(R, -\infty, \infty)$

$MinimaxAlphaBeta\,(R, -\infty, \infty)$

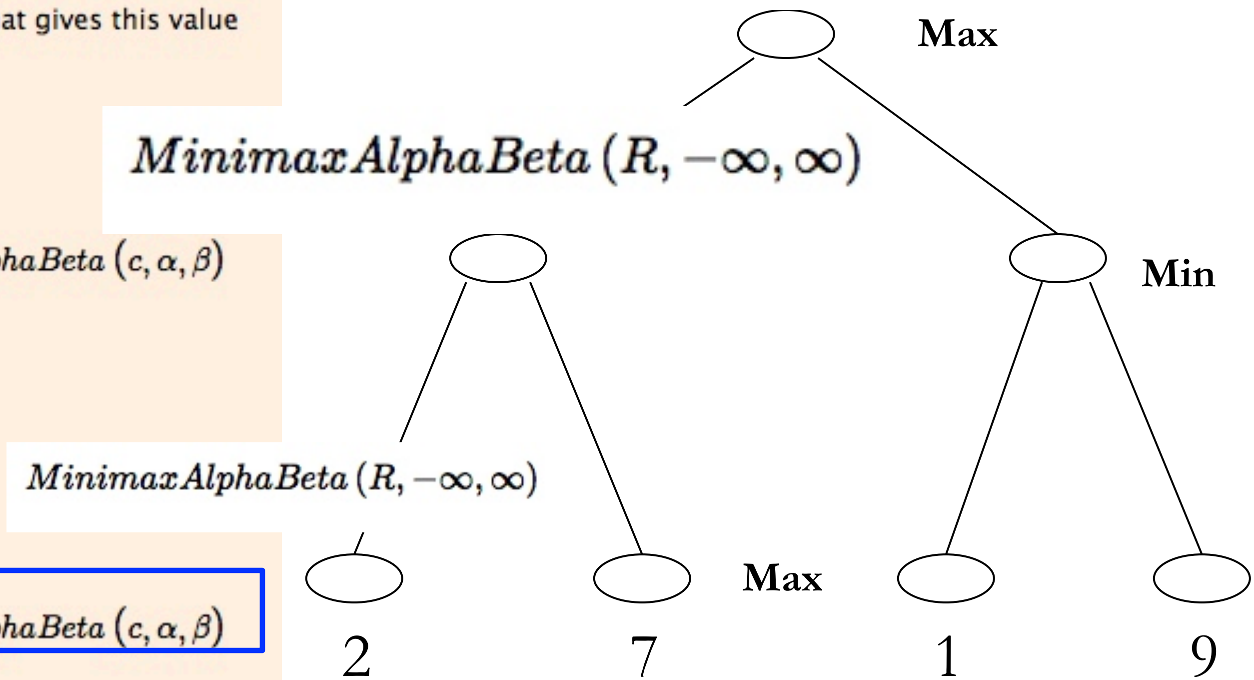


Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else   // n is a MIN node
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

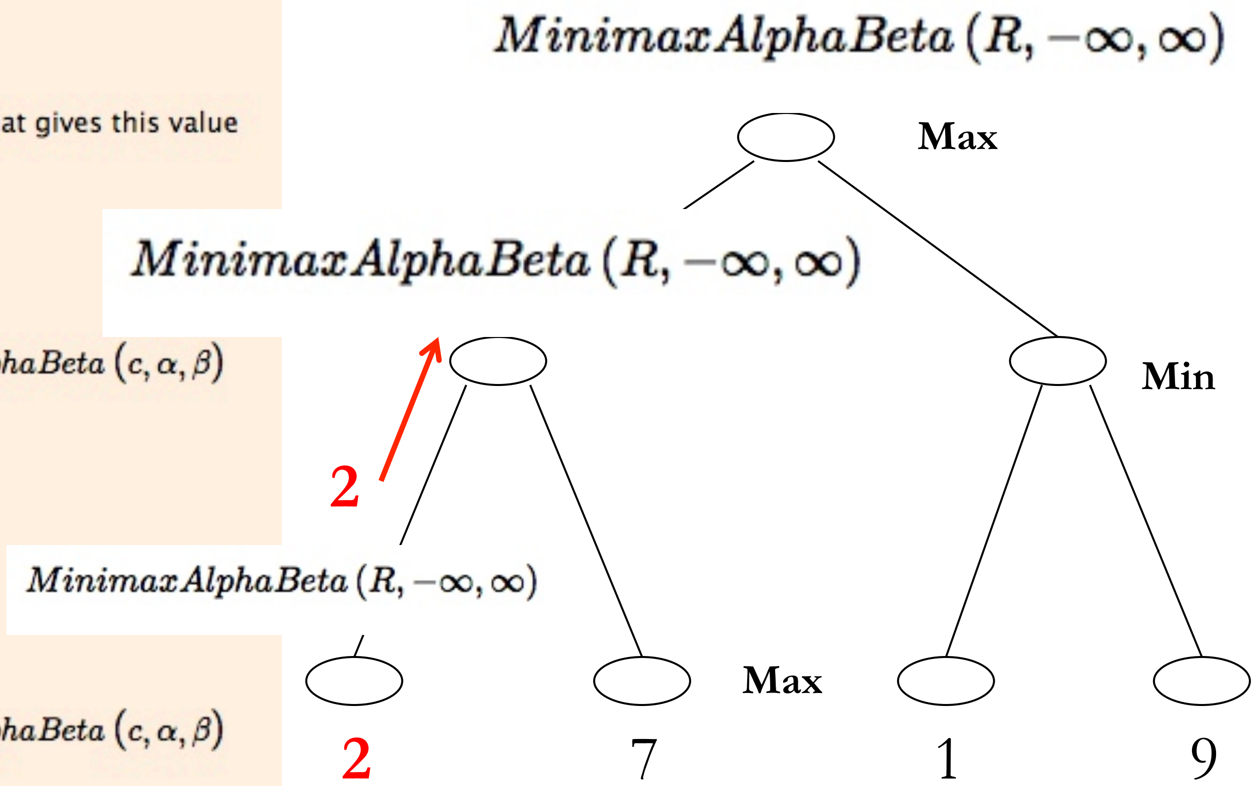
$MinimaxAlphaBeta(R, -\infty, \infty)$



Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

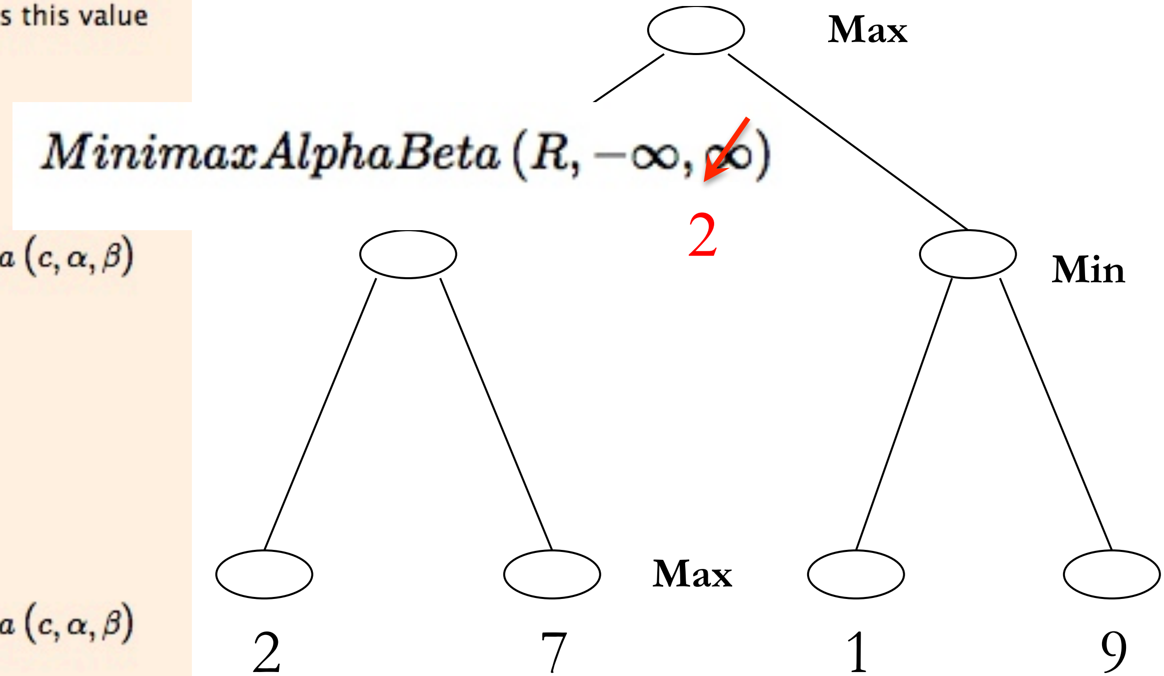
Figure 11.6: Minimax with α–β pruning



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

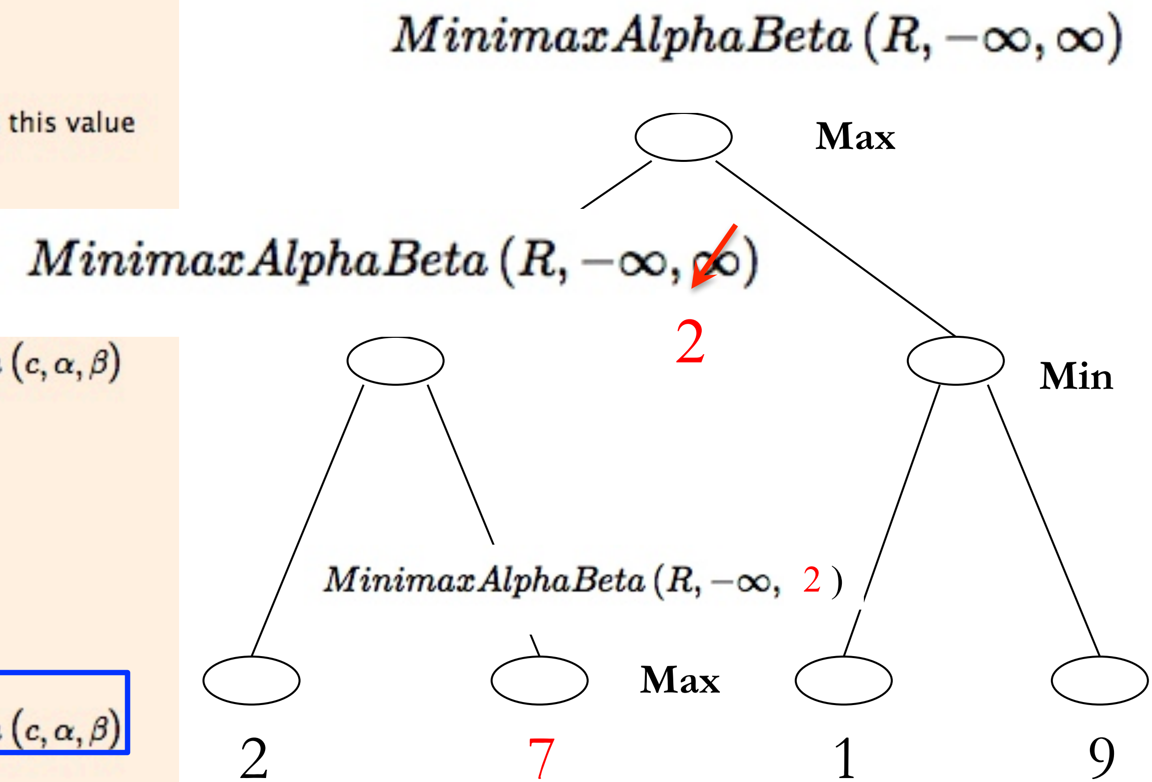
Figure 11.6: Minimax with α–β pruning

$MinimaxAlphaBeta\,(R, -\infty, \infty)$



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

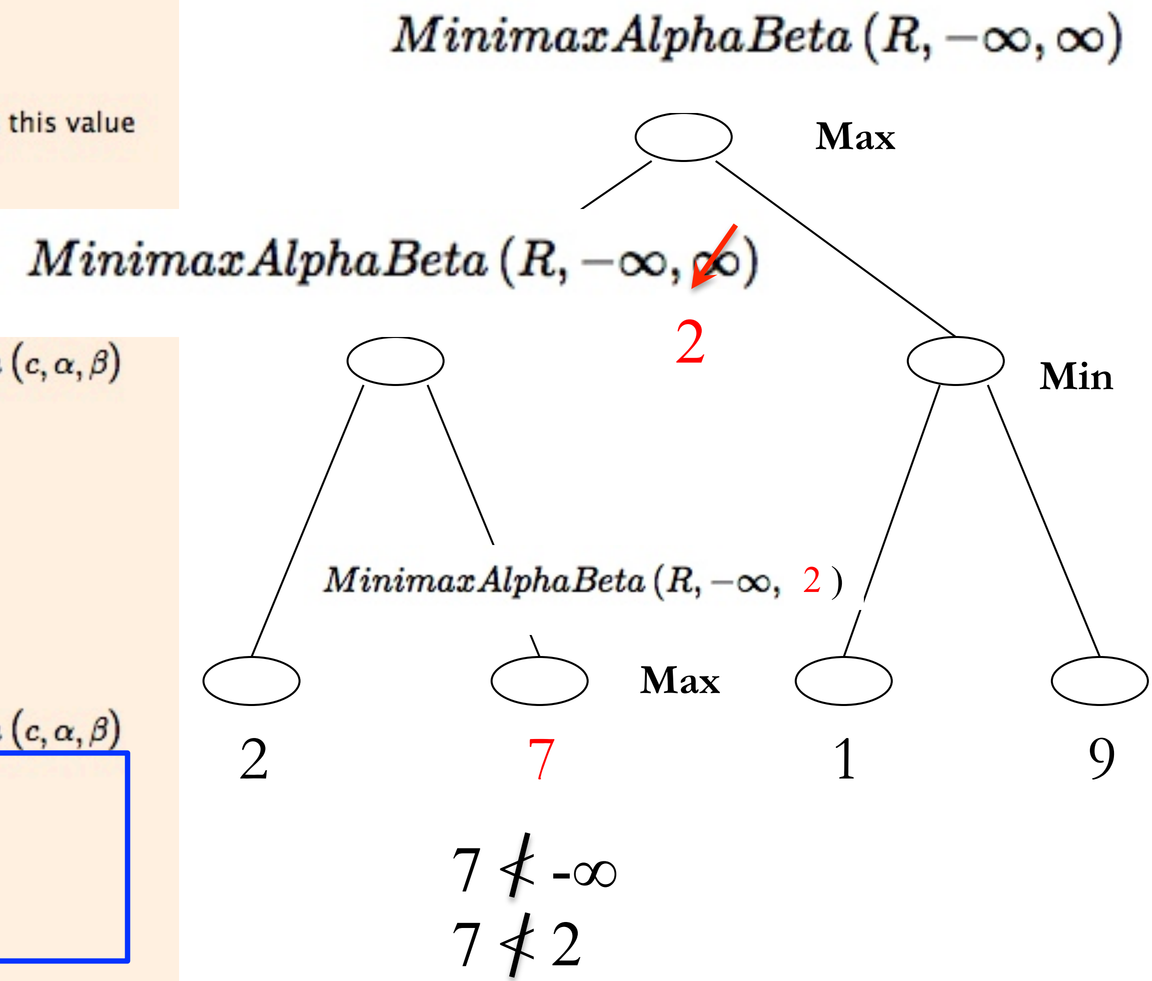
Figure 11.6: Minimax with α–β pruning



Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

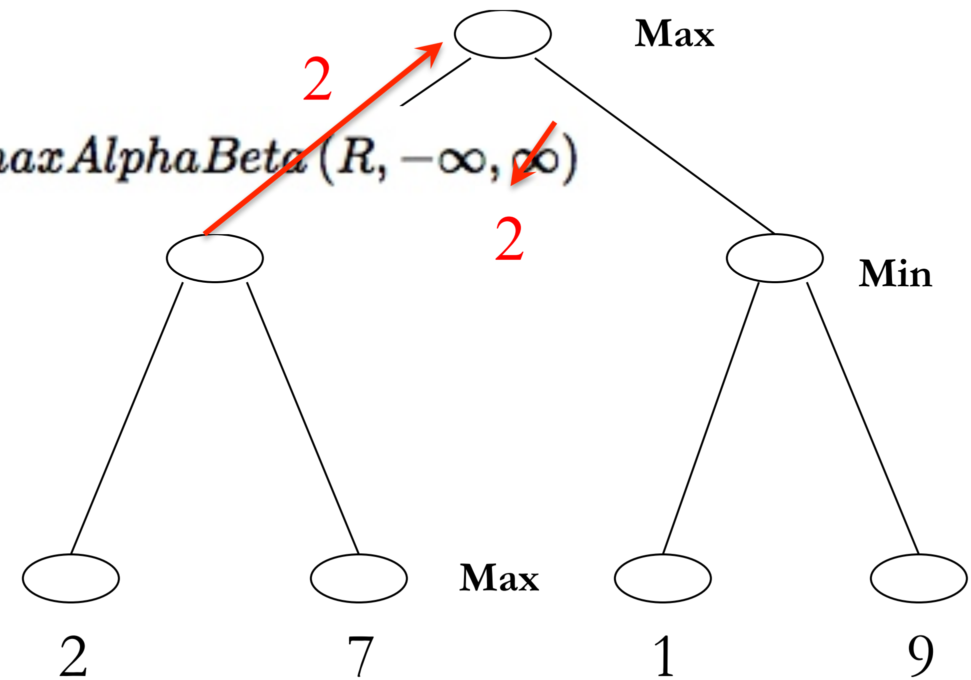
Figure 11.6: Minimax with α–β pruning



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
     else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
     else   // n is a MIN node
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

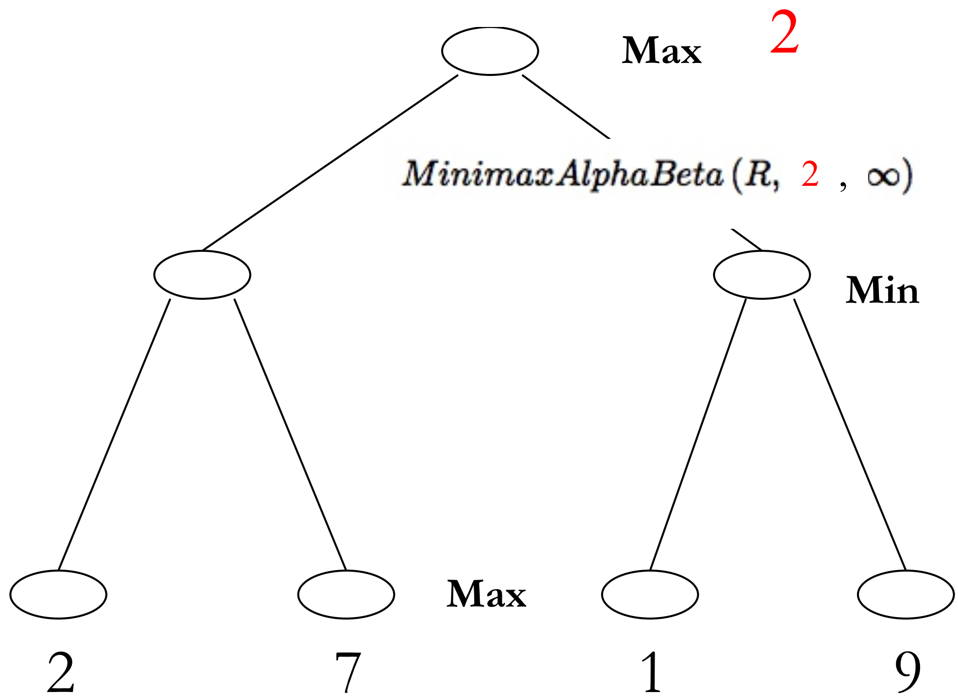
$MinimaxAlphaBeta(R, -\infty, \infty)$



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

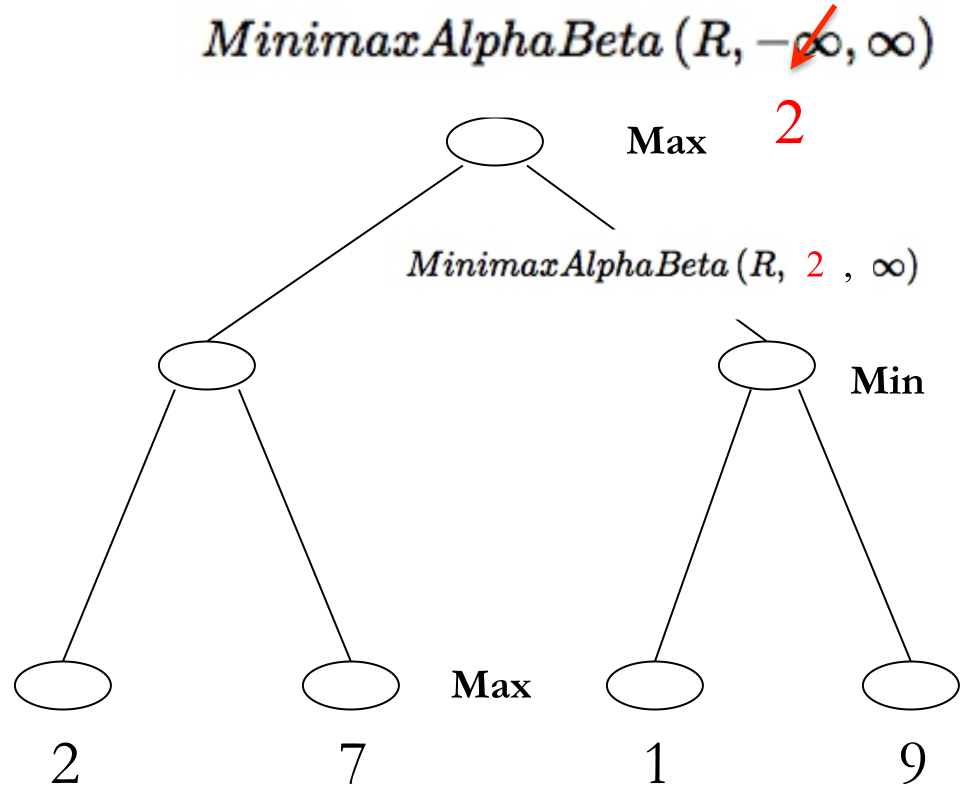
$MinimaxAlphaBeta(R, -\infty, \infty)$



Douglas H. Fisher

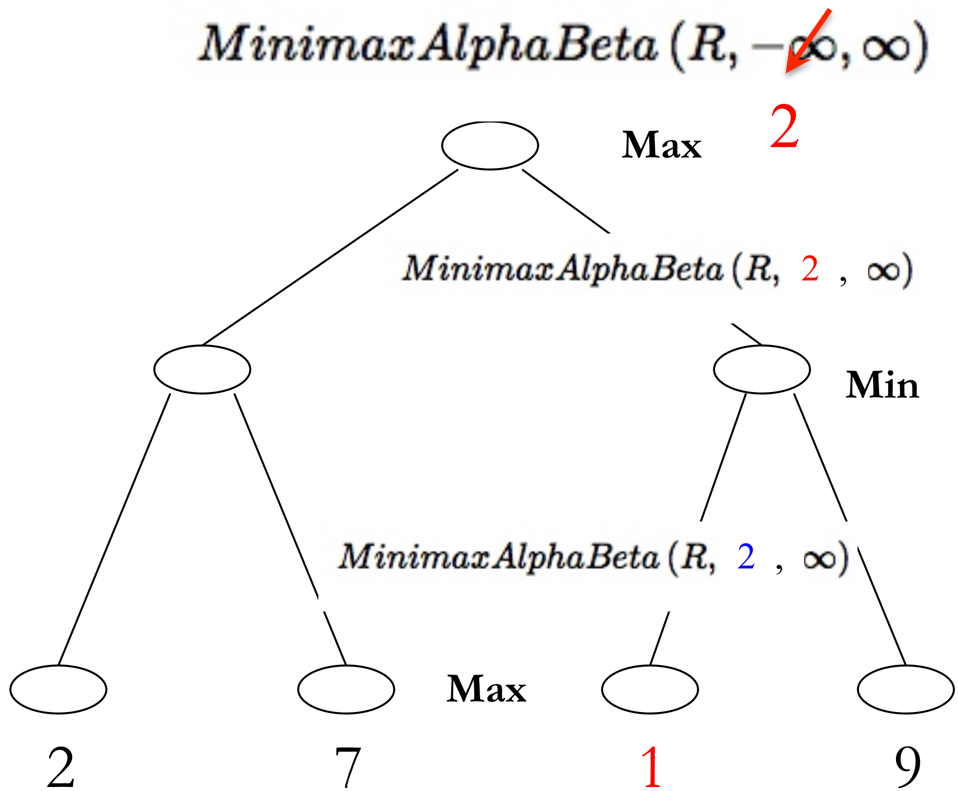
```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

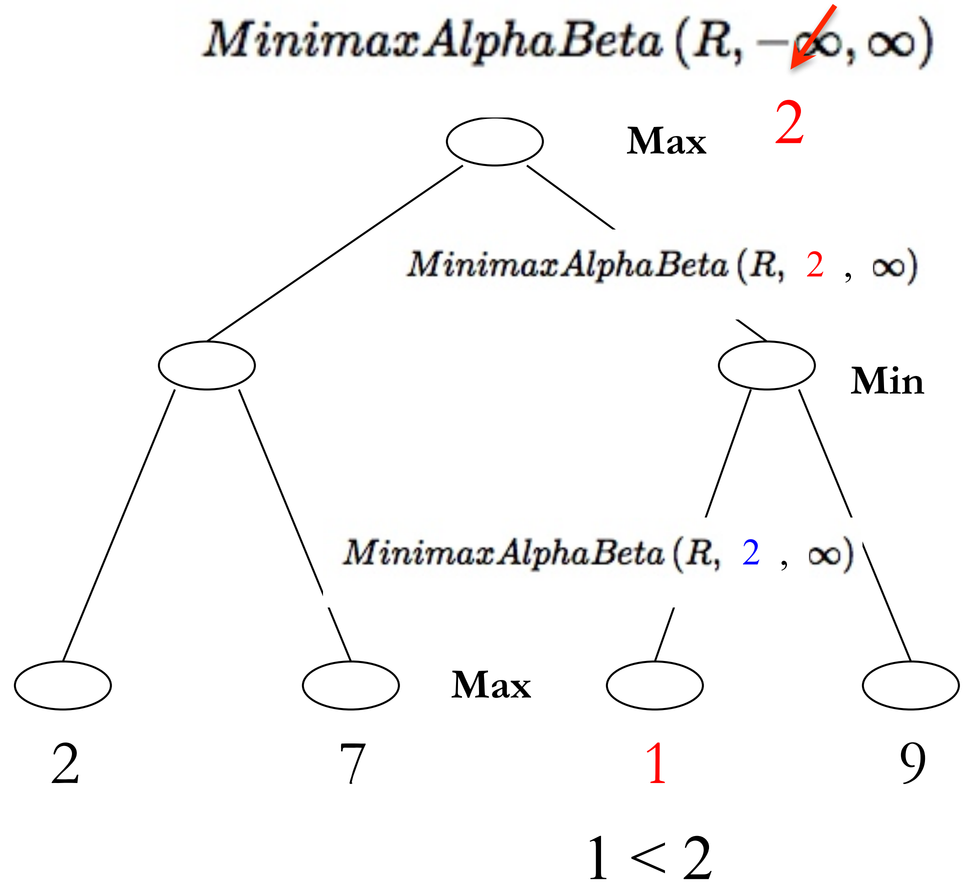
Figure 11.6: Minimax with α–β pruning



Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

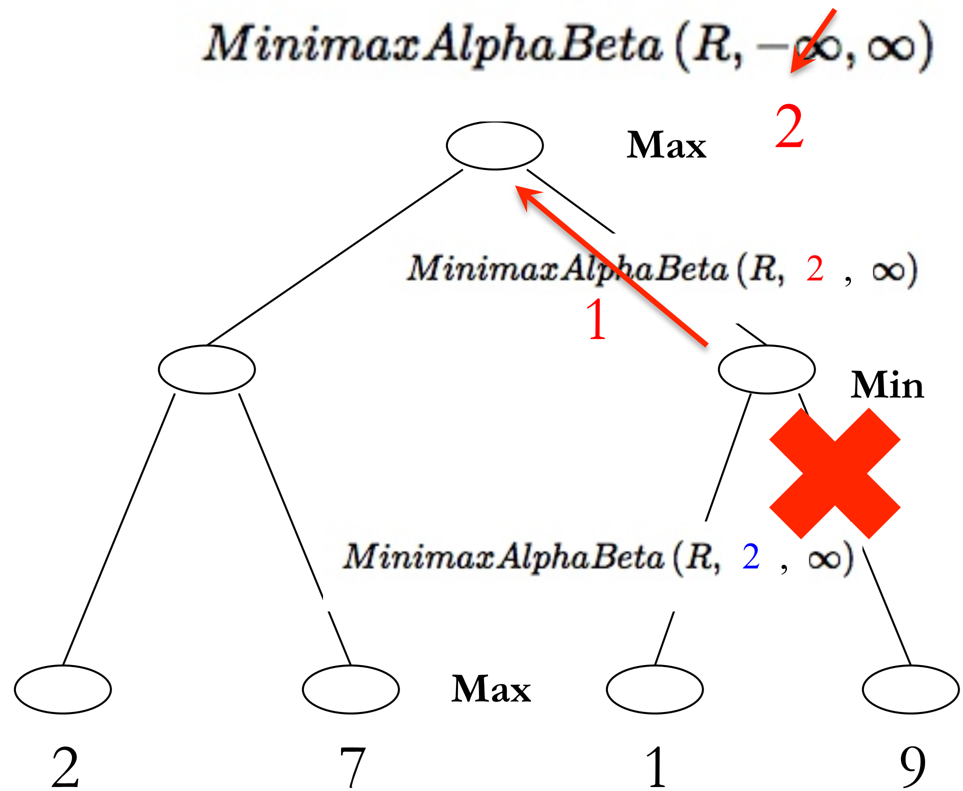


Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate (n) , None
     else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta (c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
     else
        for each child c of n do
            score, path := MinimaxAlphaBeta (c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

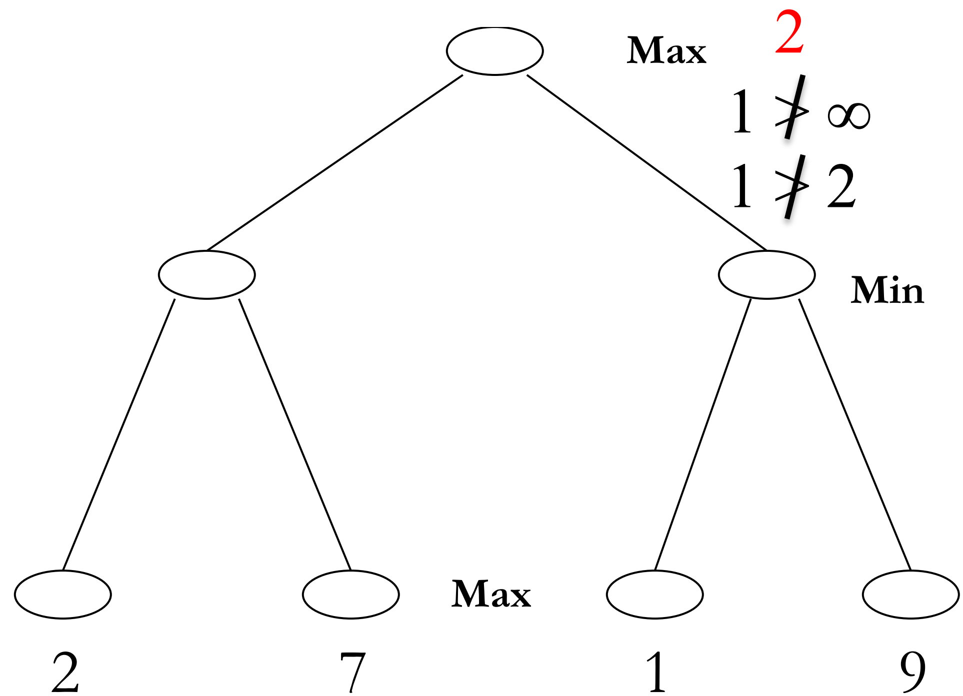
Effectively prune



```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
    else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
    else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

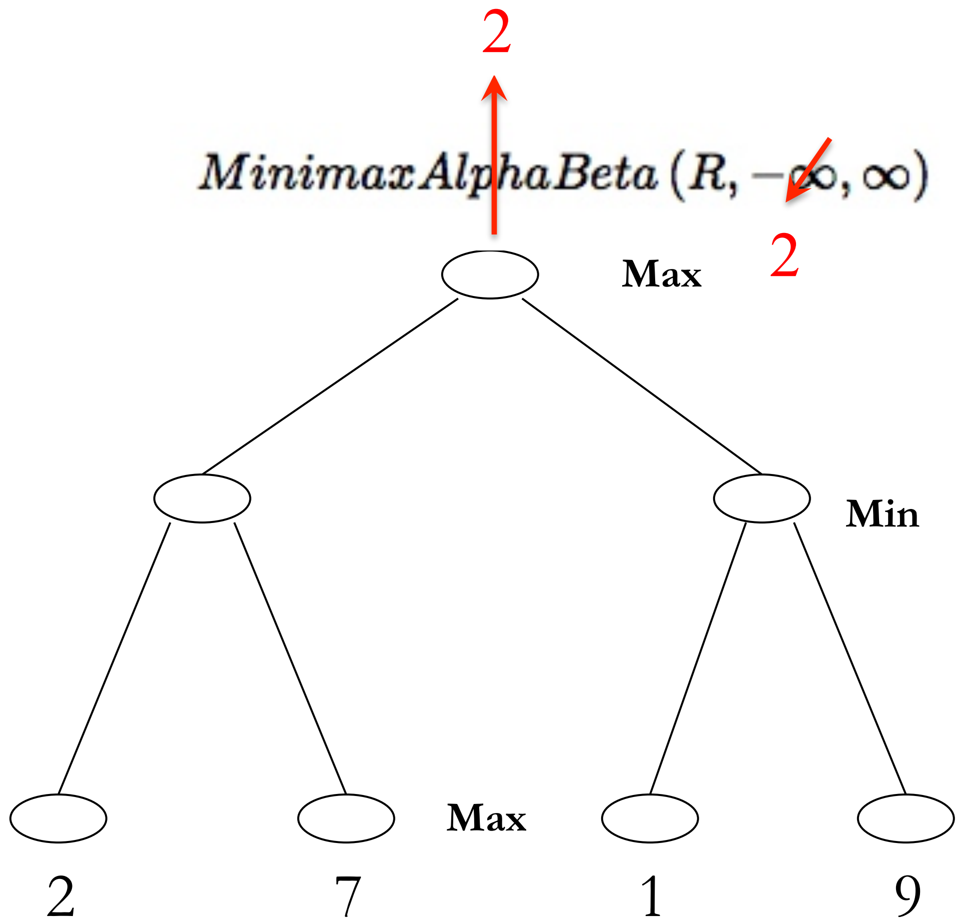
$MinimaxAlphaBeta\,(R, -\cancel{\infty}, \infty)$



Douglas H. Fisher

```
procedure Minimax_alpha_beta(n, α, β)
    Inputs
        n a node in a game tree
        α, β real numbers
    Output
        A pair of a value for node n, path that gives this value
    best := None
    if n is a leaf node then
        return evaluate(n), None
     else if n is a MAX node then
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≥ β then
                return score, None
            else if score > α then
                α := score
                best := c : path
        return α, best
     else
        for each child c of n do
            score, path := MinimaxAlphaBeta(c, α, β)
            if score ≤ α then
                return score, None
            else if score < β then
                β := score
                best := c : path
        return β, best
```

Figure 11.6: Minimax with α–β pruning

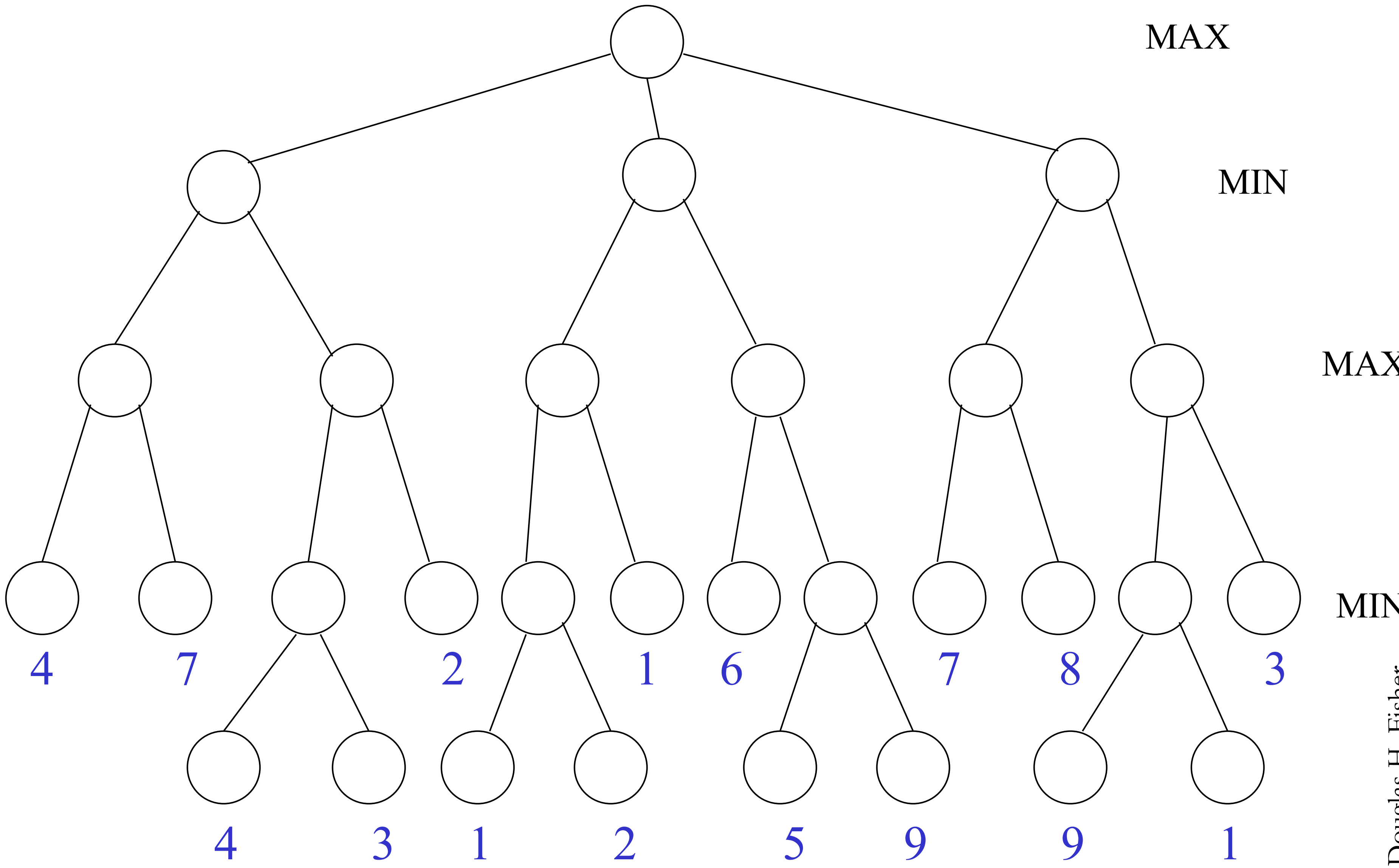


Douglas H. Fisher

Game Tree Search Problems

Given the following game tree, compute the utility value of the root (which is at a maximizing level) as computed by minimax search, and write the utility value by the root

Value of Root: _______

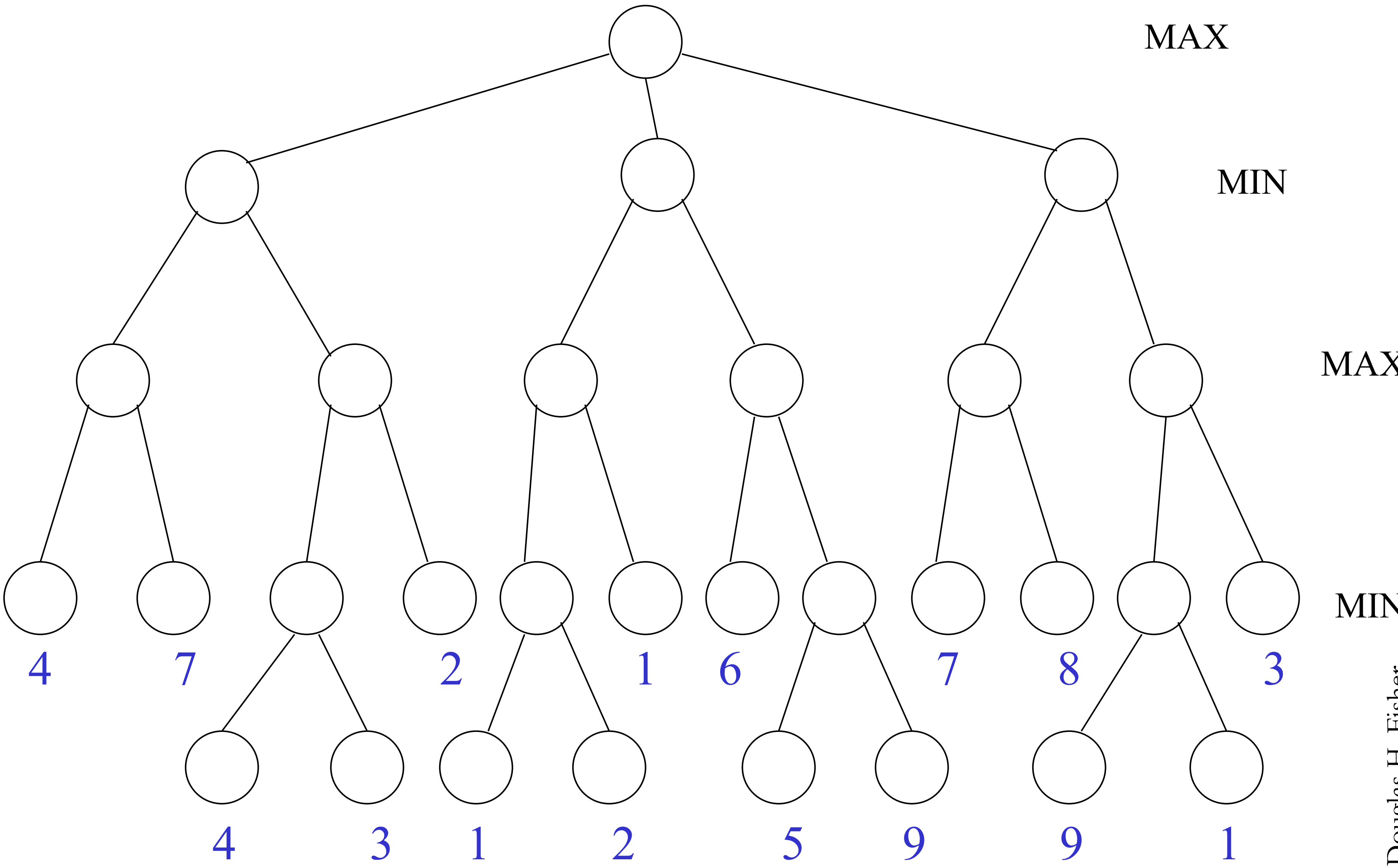


Douglas H. Fisher

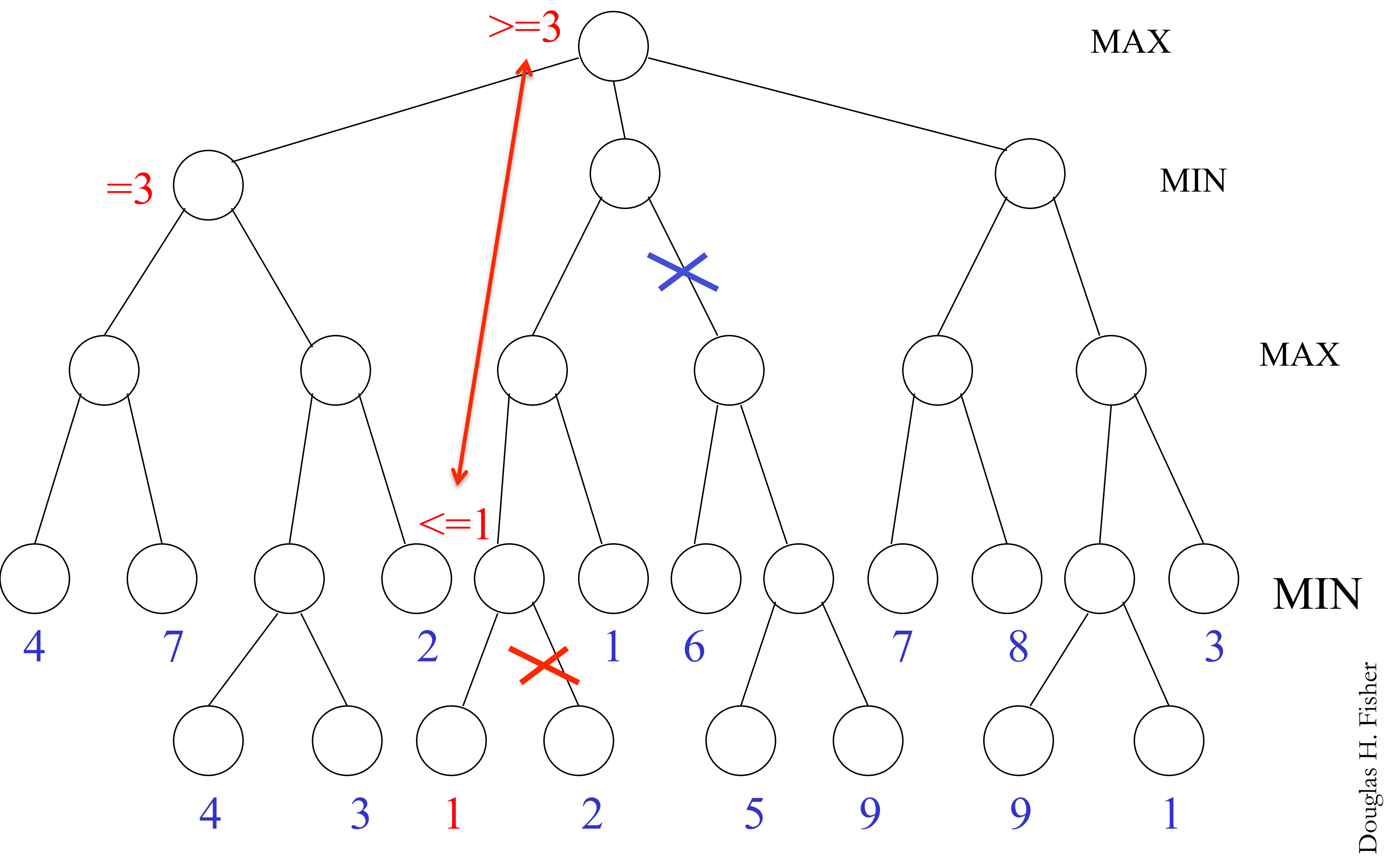
Game Tree Search Problems

Put an 'X' through an arc if it and its entire subtree would be pruned when using alpha beta pruning.

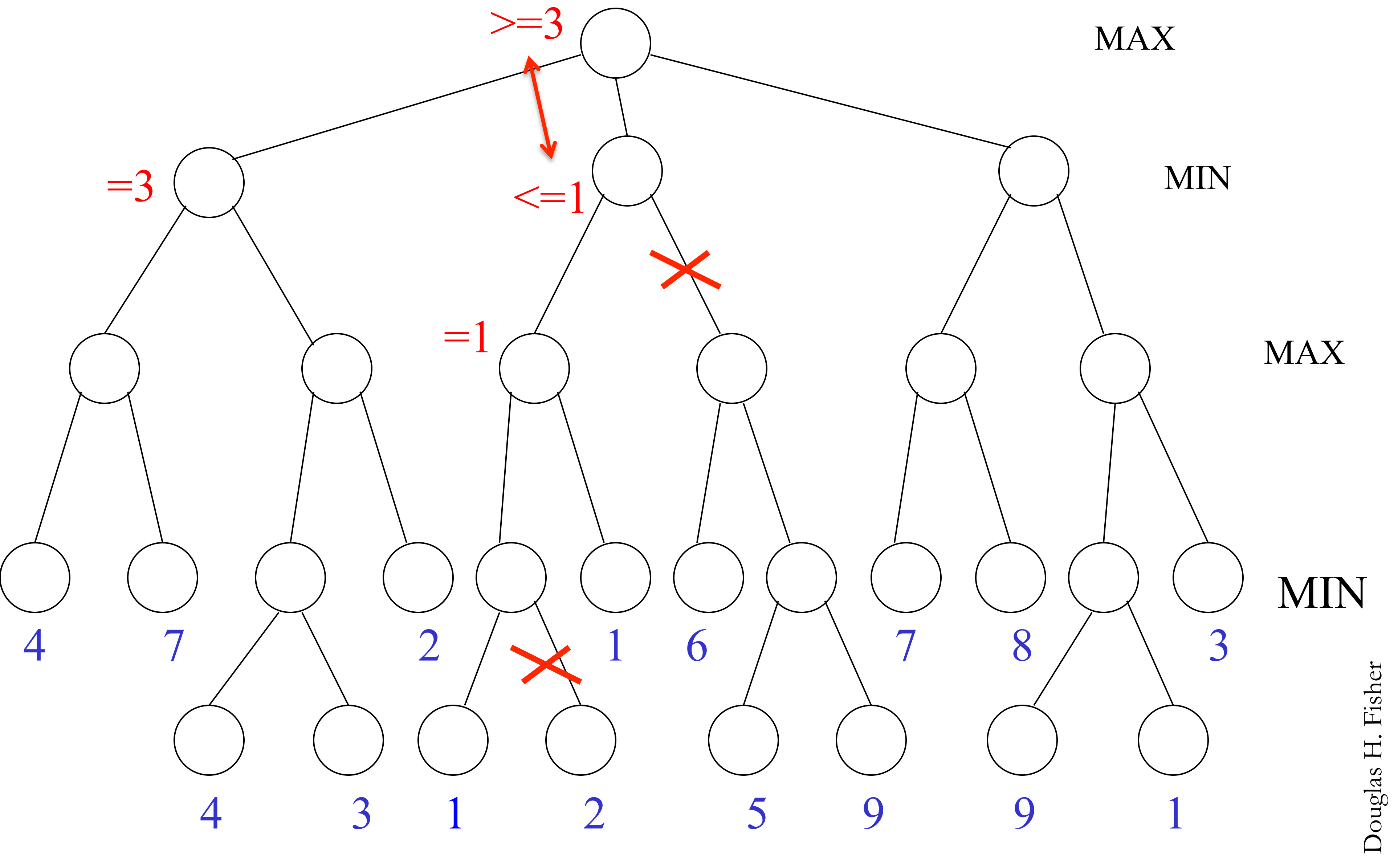
Value of Root: 3



Douglas H. Fisher

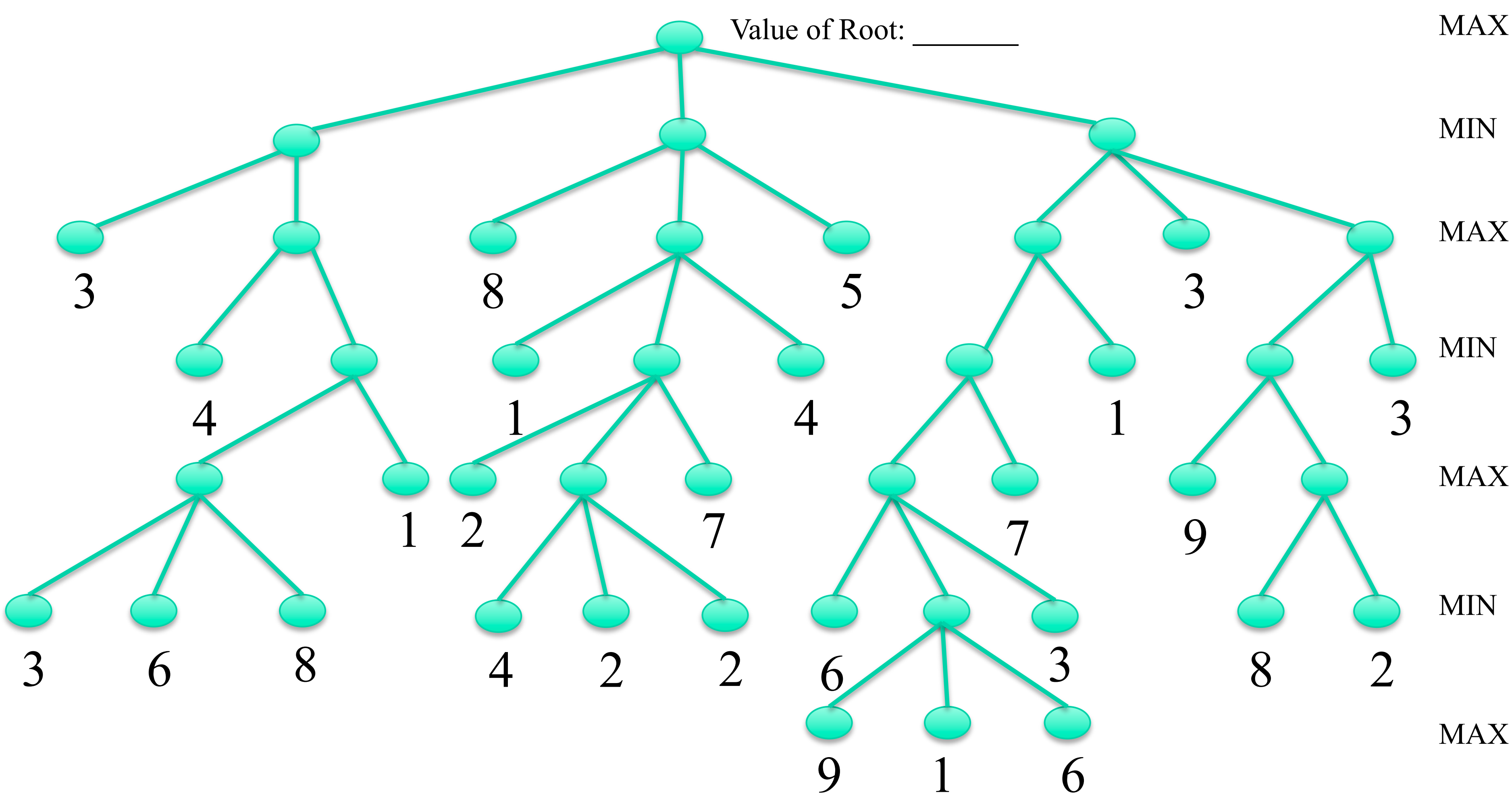


>=3
MAX
=3
MIN
MAX
<=1
MIN
4
7
2
1
6
7
8
3
4
3
1
2
5
9
9
1
Douglas H. Fisher



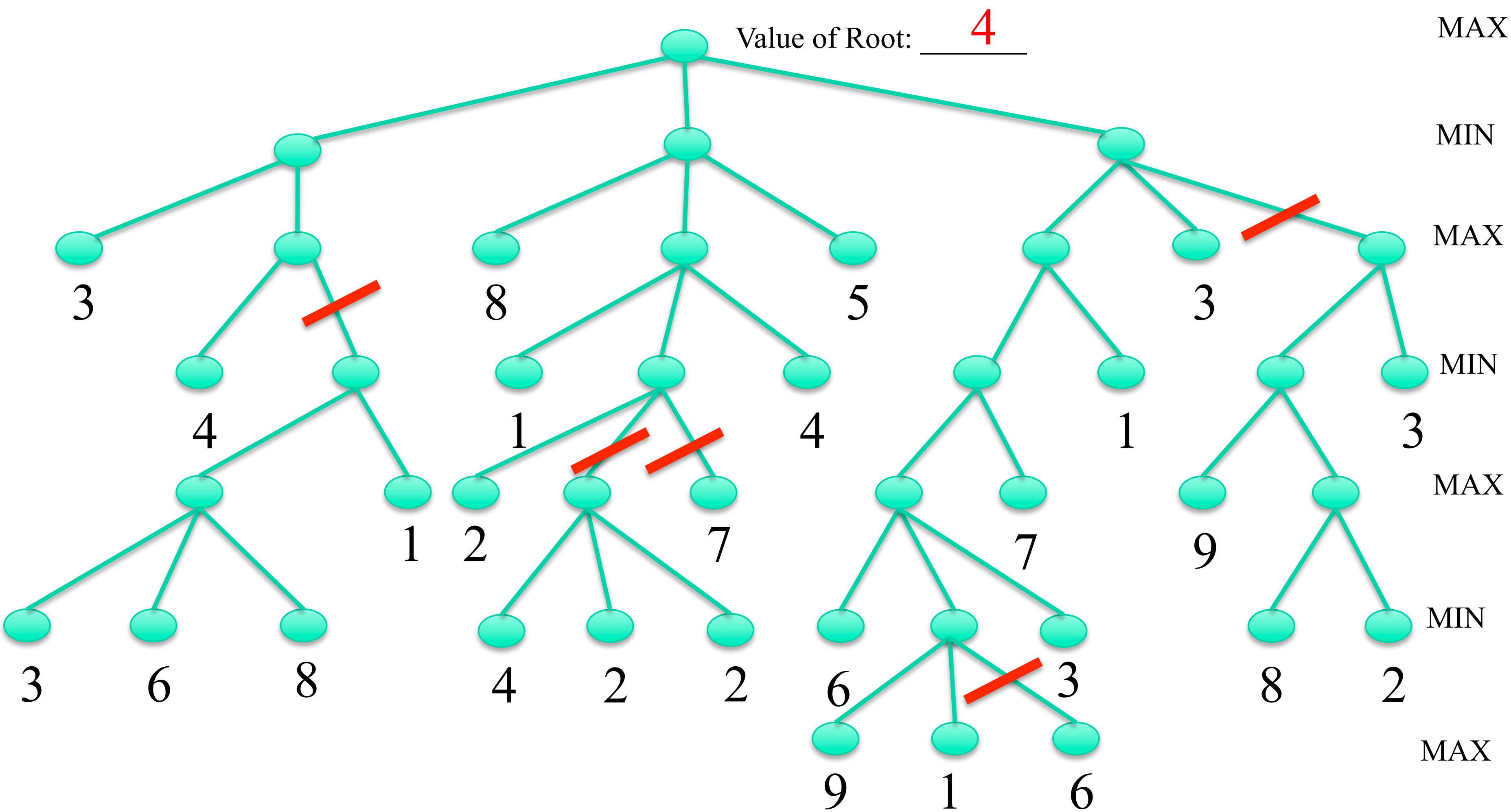
>=3
MAX
=3
<=1
MIN
=1
MAX
MIN
4
7
2
1
6
7
8
3
4
3
1
2
5
9
9
1
Douglas H. Fisher

Consider the game tree below. The numbers at leaves are scores obtained by an evaluation function. Show the value of the root obtained through minimax search. Additionally, put an 'X' through an arc if it and its entire subtree would be pruned when using alpha beta pruning.



Douglas H. Fisher

Consider the game tree below. The numbers at leaves are scores obtained by an evaluation function. Show the value of the root obtained through minimax search. Additionally, put an 'X' through an arc if it and its entire subtree would be pruned when using alpha beta pruning.



Douglas H. Fisher