Adversarial Search for Games

CS 640 Lecture Notes

Slides by Margrit Betke, Zheng Wu, and Mahir Patel based on materials from Patrick Winston's book Artificial Intelligence, and Wikipedia pages

Chess World Championship, November 9, 1985

fide_chess • Follow ...

fide_chess 35 years ago, on November 9, 1985, Garry Kasparov won the 24th game of his second match against Anatoly Karpov and, at the age of 22, became the youngest-ever world chess champion.

Commemorating this day, the 13th world chess champion shared a great memory on his Twitter *Kasparov63*, he wrote:
"A short story about the night before this fateful game that made me the world champion 35 years ago today.
The evening of November 8, I received two phone calls from two former world champions, both named Mikhail.
Mikhail Botvinnik, my old teacher, the Patriarch of Soviet chess, was typically

Liked by **imperator.rima** and **others**

49 MINUTES AGO

Add a comment... Post

Chess

16 pieces per player:
1 king
1 queen
2 rooks
2 bishops
2 knights
8 pawns

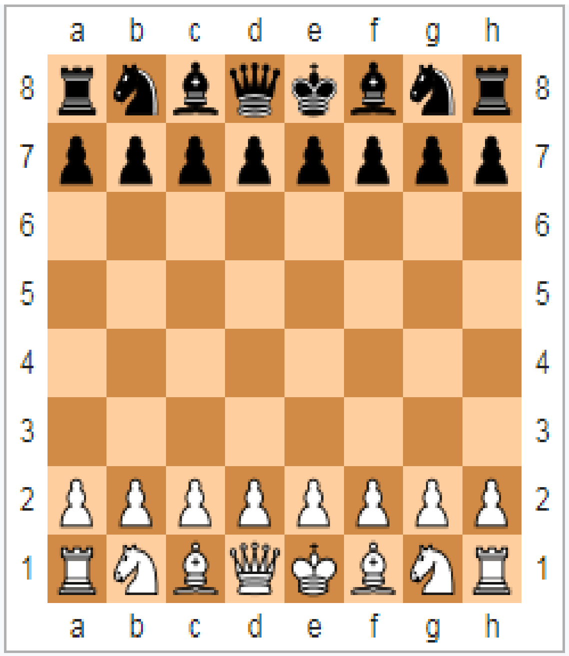
8 x 8 board

Pawn, rook, knight, bishop, queen, king

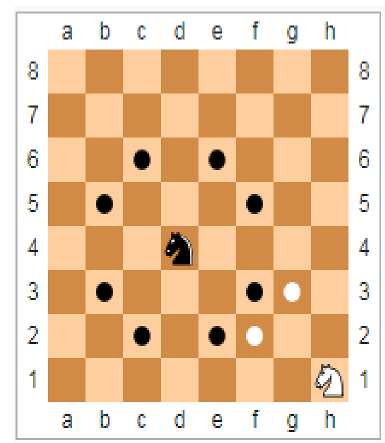
Abstract strategy two-player game

Image Credits: Wikipedia: Bubba73, F. Camaratta

Initial Board and Moves of Pieces



Chess starting position. Squares are referenced using algebraic notation.



The black knight may move to any of eight squares (black dots). The white knight in this case is limited to two squares (white dots).

Image Credit: Wikipedia

Chess History

- 1400 years ago from India & Persia; through Arabia to Europe to US
- "Chaturanga" = army
- 1886: 1st World Championship
- Current Champion: Ding Liren from China
- Gary Kasparov was ranked world No. 1 for 255 months overall in his career (1985-2005)

Chess History & AI

- First Question: Why research algorithms for solving an abstract strategy game?

Source: Wikipedia

Chess History & AI

- First Question: Why research AI algorithms for game theory?
- Lot of real-world problems can be categorized as adversarial or cooperative system.

Source: Wikipedia

Chess History & AI

- First Question: Why research AI algorithms for game theory?
- Lot of real-world problems can be categorized as adversarial or cooperative system.
- Chess provides a well-defined (deterministic) yet complex environment.

Source: Wikipedia

Chess History & AI

- In 1997 Gary Kasparov became the first world champion to lose a match to a computer under standard time controls, when he lost to the IBM supercomputer Deep Blue in a highly publicized match.

Source: Wikipedia

1996: Kasparov vs IBM DeepBlue

The 1996 match

Game #	White	Black	Result	Comment
1	**Deep Blue**	Kasparov	1–0	
2	**Kasparov**	Deep Blue	1–0	
3	Deep Blue	Kasparov	½–½	Draw by mutual agreement
4	Kasparov	Deep Blue	½–½	Draw by mutual agreement
5	Deep Blue	**Kasparov**	0–1	Kasparov offered a draw after the 23rd move.
6	**Kasparov**	Deep Blue	1–0	
Result: **Kasparov–Deep Blue: 4–2**				

Source: Wikipedia

1997: Kasparov vs IBM DeepBlue

The 1997 rematch

Game #	White	Black	Result	Comment
1	**Kasparov**	Deep Blue	1–0	
2	**Deep Blue**	Kasparov	1–0	
3	Kasparov	Deep Blue	½–½	Draw by mutual agreement
4	Deep Blue	Kasparov	½–½	Draw by mutual agreement
5	Kasparov	Deep Blue	½–½	Draw by mutual agreement
6	**Deep Blue**	Kasparov	1–0	
Result: **Deep Blue–Kasparov: 3½–2½**				

Source: Wikipedia

1997: Kasparov vs IBM DeepBlue

Three matches ended in draws, with Kasparov appearing to weaken psychologically. Deep Blue went on to win the decisive sixth game, marking the first time in history that a computer defeated the World Champion in a match of several games. From this experience, particularly the second game of the match, Kasparov accused the IBM team of cheating. He suspected that a human player was used during the games to improve the strategic strength of the computer.

2016 Podcast with Kasparov: Change of heart in his views of this match

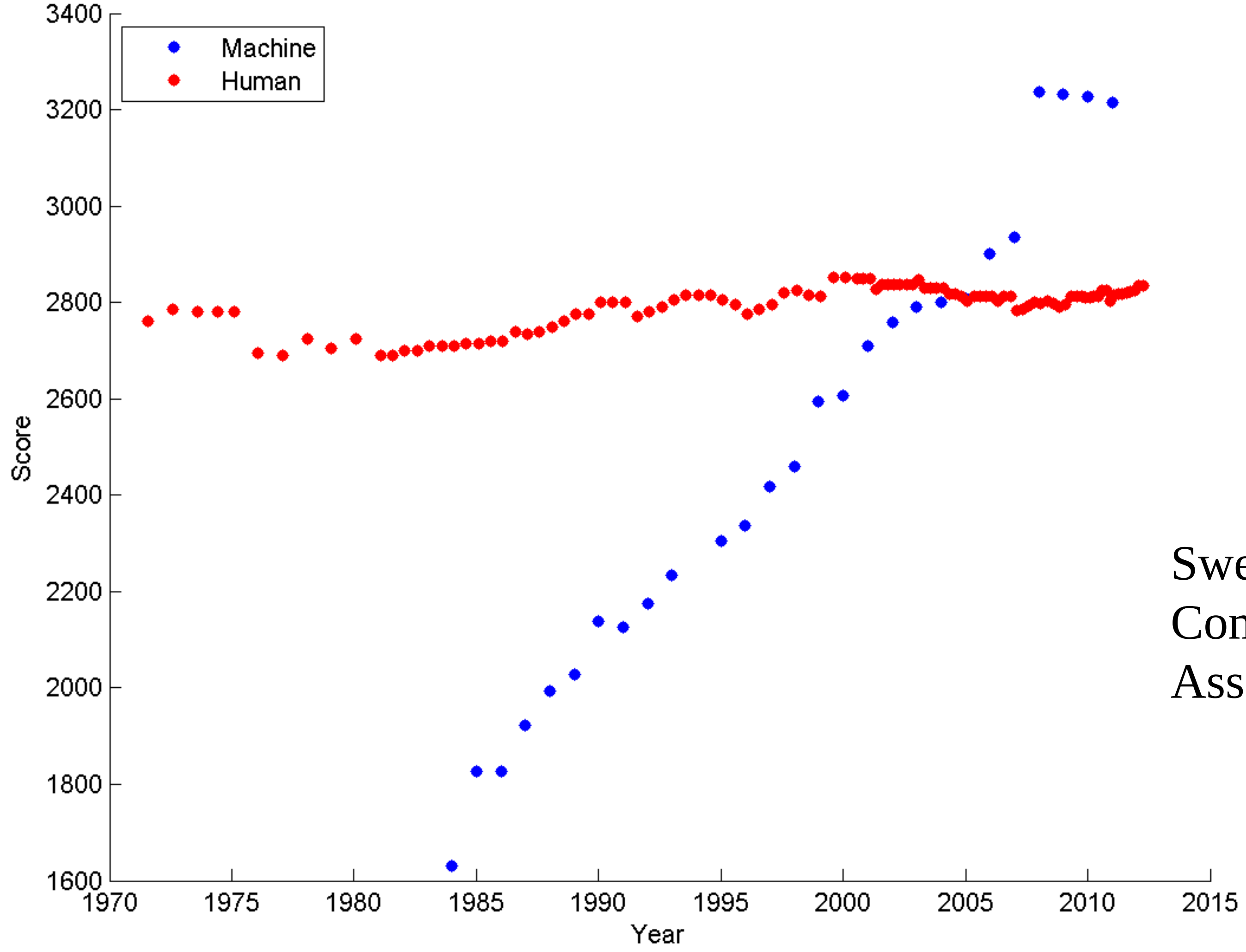
"I did a lot of research – analysing the games with modern computers, also soul-searching – and I changed my conclusions. I am not writing any love letters to IBM, but my respect for the Deep Blue team went up, and my opinion of my own play, and Deep Blue's play, went down. Today you can buy a chess engine for your laptop that will beat Deep Blue quite easily."

Source: Wikipedia

Famous AI Chess Programs

- IBM Deep Blue:
  - IBM dismantled DeepBlue due to dispute
  - Evaluation function: 8,000 parts
  - Opening: 4,000 positions from 700,000 grand master games
  - End game: 6 pieces left; dedicated database
  - IBM did not let Kasparov know about database of games to study opponent
- Deep Fritz:
  - https://en.wikipedia.org/wiki/Fritz_(chess)
  - In 1995, Fritz 3 won the World Computer Chess Championship in Hong Kong
  - March 2022: Fritz 18 Neuronal release

Machine vs. Human Chess Champions



Swedish Chess Computer Association data

https://srconstantin.wordpress.com/2017/01/28/performance-trends-in-ai/

State of the Art in 2023: Alpha-beta technique

- Stockfish (strongest CPU chess engine in the world)

  Rating: 3546

  Open-source engine available for various desktop and mobile platforms. It is based on another open-source chess engine named Glaurung and uses the alpha-beta procedure.

- Hannibal

  Rating: 3229

  Hannibal is a Universal Chess Interface (UCI) engine that incorporates ideas from earlier engines, *Twisted Logic*, and *LearningLemming*. It uses the alpha-beta technique with many other chess specific heuristics and relies on a selective search method.

Board Games like Chess

AI system evaluates board configurations with a Static Evaluator :

Compute a number = Static Evaluation Score

The static evaluator checks the merit of a move by looking ahead (potentially several moves)

Minimax Procedure

- Static Evaluation Score reflects board quality in a single number
- 1st player is the Maximizer: Player aiming for maximum score
- 2nd player is the Minimizer: Player aiming for minimum score
- Game Tree: nodes, branches, depth
- Game = Path through Game Tree

Chess

16 pieces per player:
1 king
1 queen
2 rooks
2 bishops
2 knights
8 pawns

8 x 8 board

Pawn, rook, knight, bishop, queen, king

IBM Deep Blue static evaluator:
Pawn: 1
Knight: 3, Bishop: 3.25
Rook: 5
Queen: 9
Piece count
King safety

Abstract strategy two-player game

Image Credits: Wikipedia: Bubba73, F. Camaratta

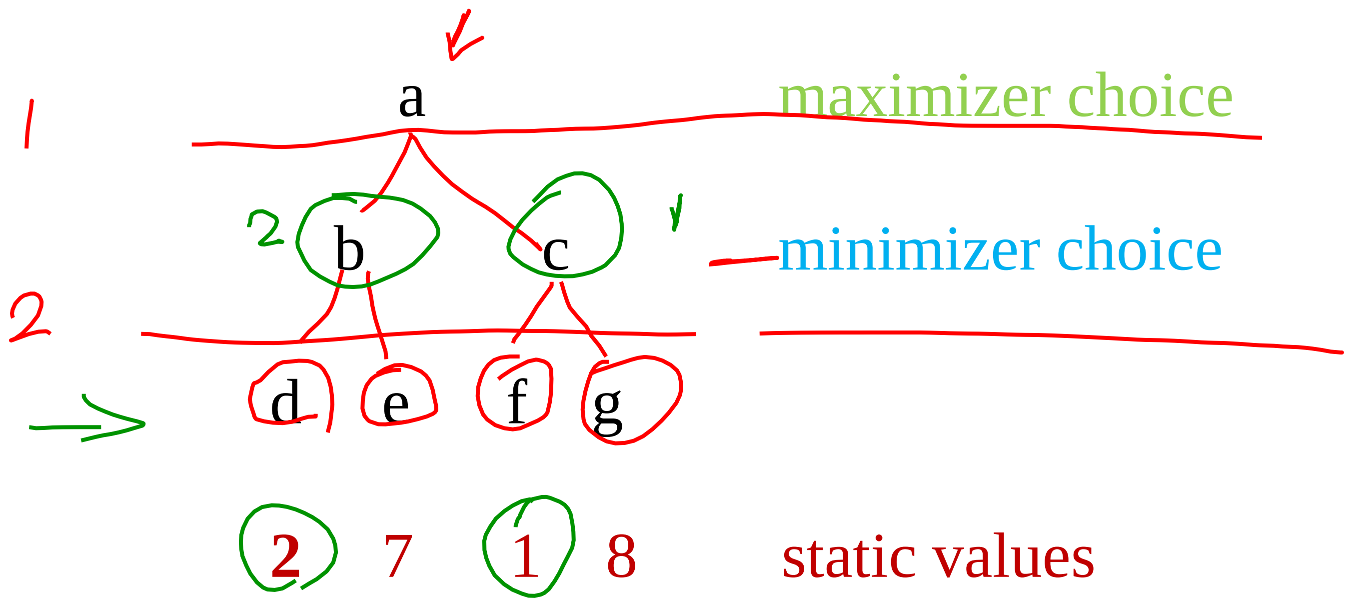
Minimax Procedure

- If at Look-ahead Depth:

  Compute & report static value

- Otherwise:
  - ❖ If choice of minimizer: Recursive call on children & Report minimum
  - ❖ If choice of maximizer: Recursive call on children & Report maximum

Examples on Black Board



Lookahead depth: maximizer 2, minimizer 1

Examples on Black Board

a maximizer choice

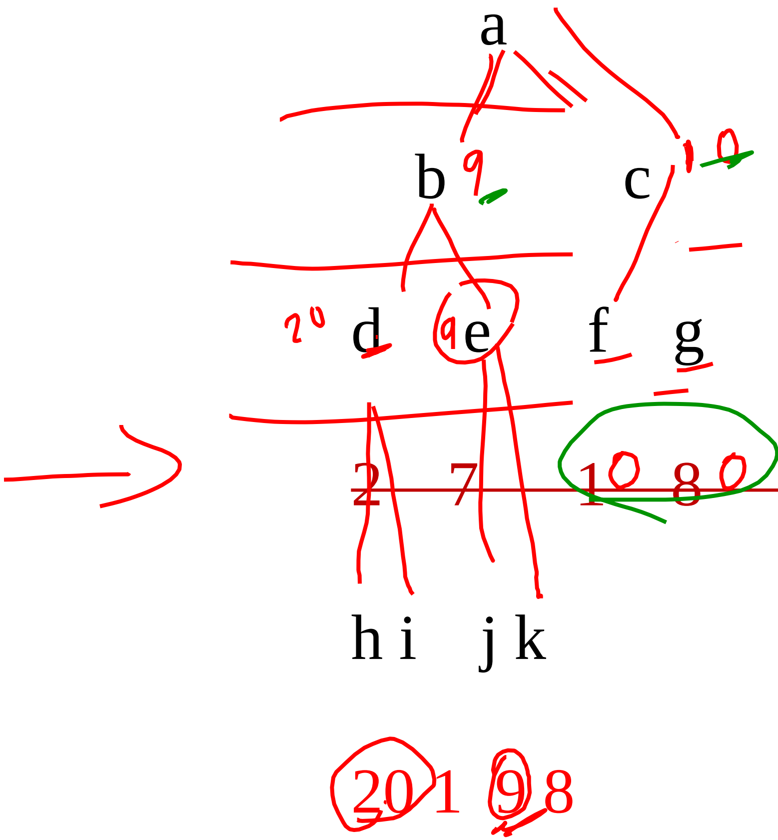
b c minimizer choice

d e f g

2 7 1 8 static values

Lookahead depth: maximizer 2, minimizer 1

Examples on Black Board



maximizer choice

minimizer choice

maximizer choice

~~static values~~

static values

Lookahead depth: maximizer 2, minimizer 2

Examples on Black Board

a maximizer choice

b c minimizer choice

d **e** f g maximizer choice

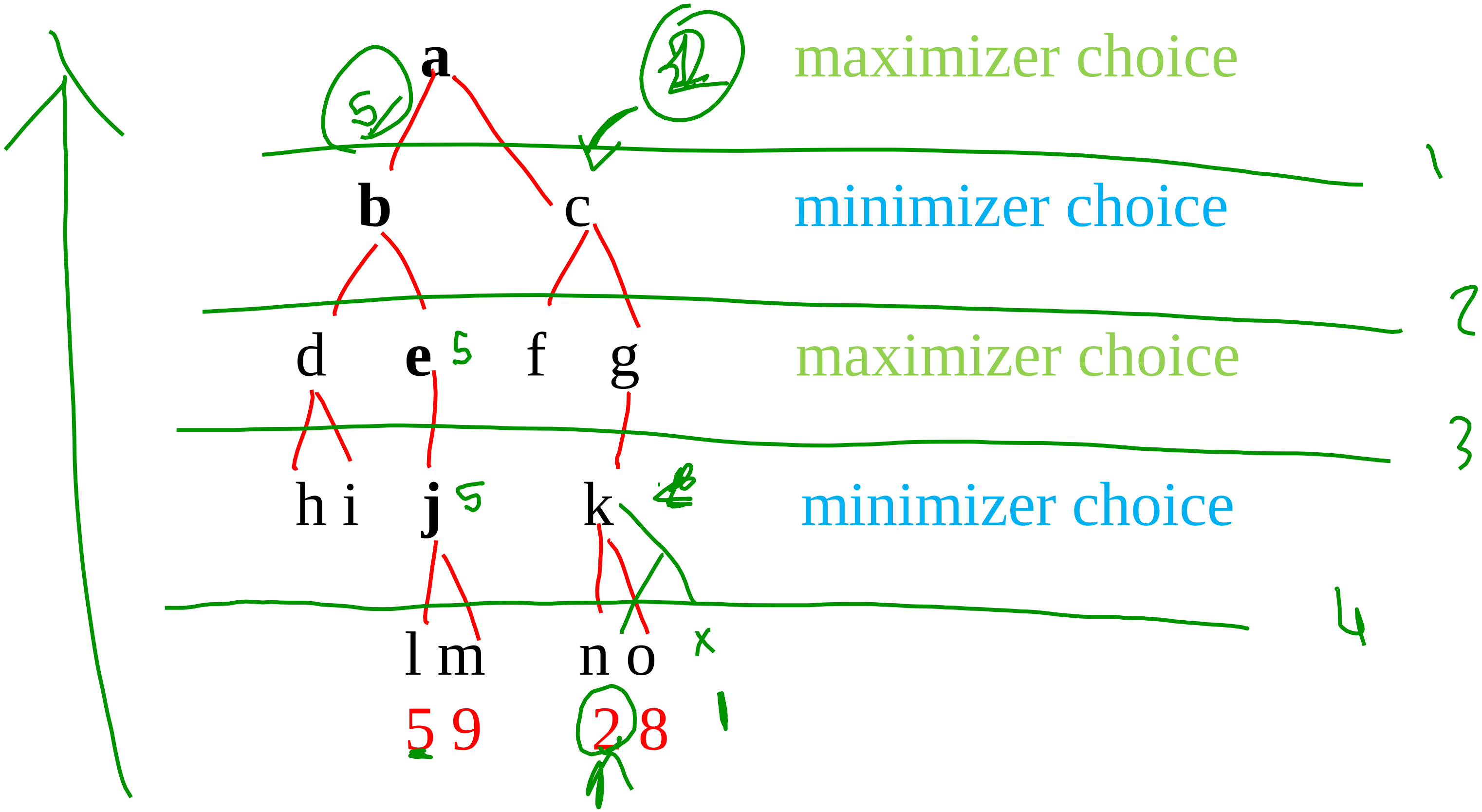
~~2 7 1 8 static values~~

h i **j** k

20 1 **9** 8 static values

Lookahead depth: maximizer 2, minimizer 2

Examples on Black Board



Examples on Black Board

		a						maximizer choice
	b				c			minimizer choice
d	**e**			f		g		maximizer choice
h i	**j**				k			minimizer choice
		l m			n o			
		5 9			2 8			

Examples on Black Board

a maximizer choice

b c minimizer choice

d **e** f g maximizer choice

h i **j** k minimizer choice

l m n o

5 9 2 8

Opportunity to prune
since 2 < 5

Alpha-Beta Procedure

- Principle

  If you have an idea that is surely bad, do not take time to see how truly awful it is
- Key idea

  The best move score at some child node provides a bound at parent node, which can be used to possibly prune other children branches

Alpha-Beta Procedure

- Use alpha to keep track of the lower bound for the maximizer
- Use beta to keep track of the upper bound for the minimizer
- For the maximizing level, the node might update its alpha, and return its alpha
- For the minimizing level, the node might update its beta, and return its beta
- The alpha and beta can be interpreted as the minimum risk for the maximizer and minimizer, respectively

void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score < beta, beta = *score; // update upper bound

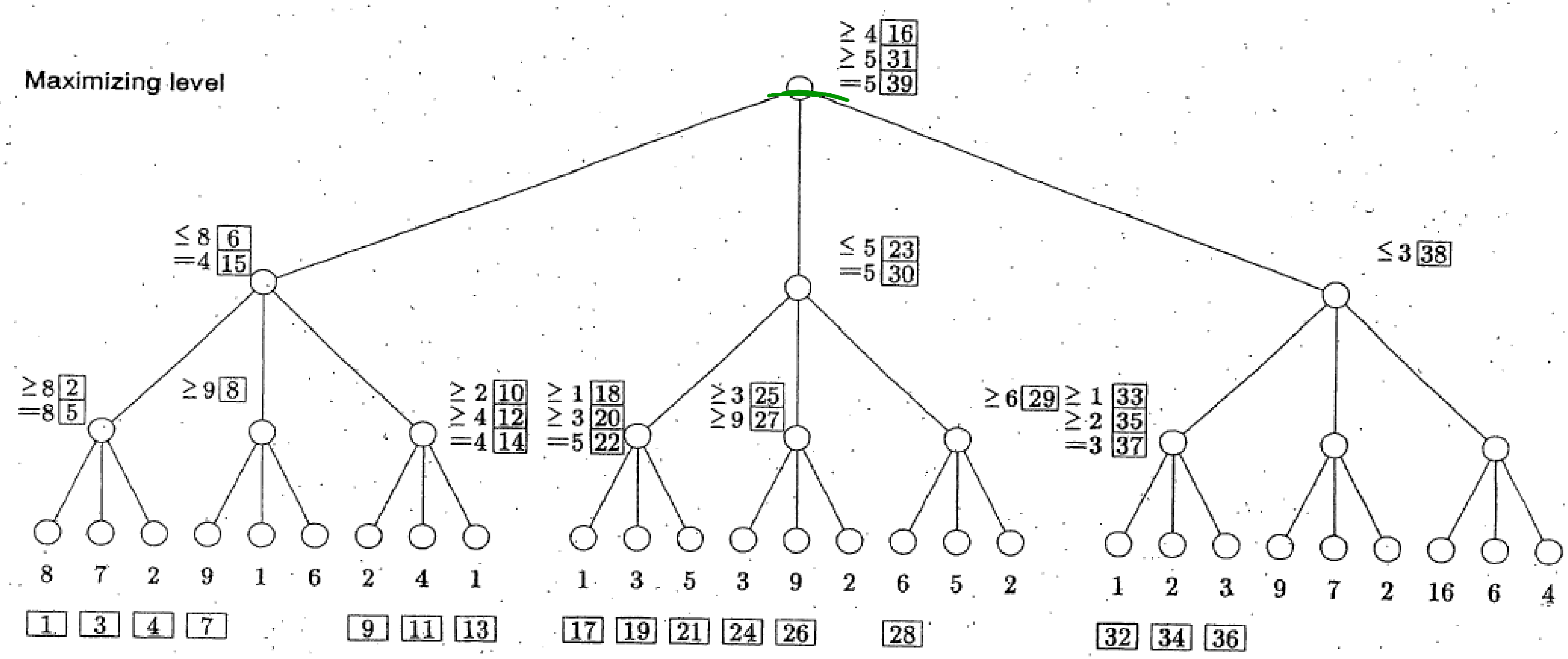
   *score = beta;
4. Else if at a maximizing level,

   until all children are examined or alpha >= beta,

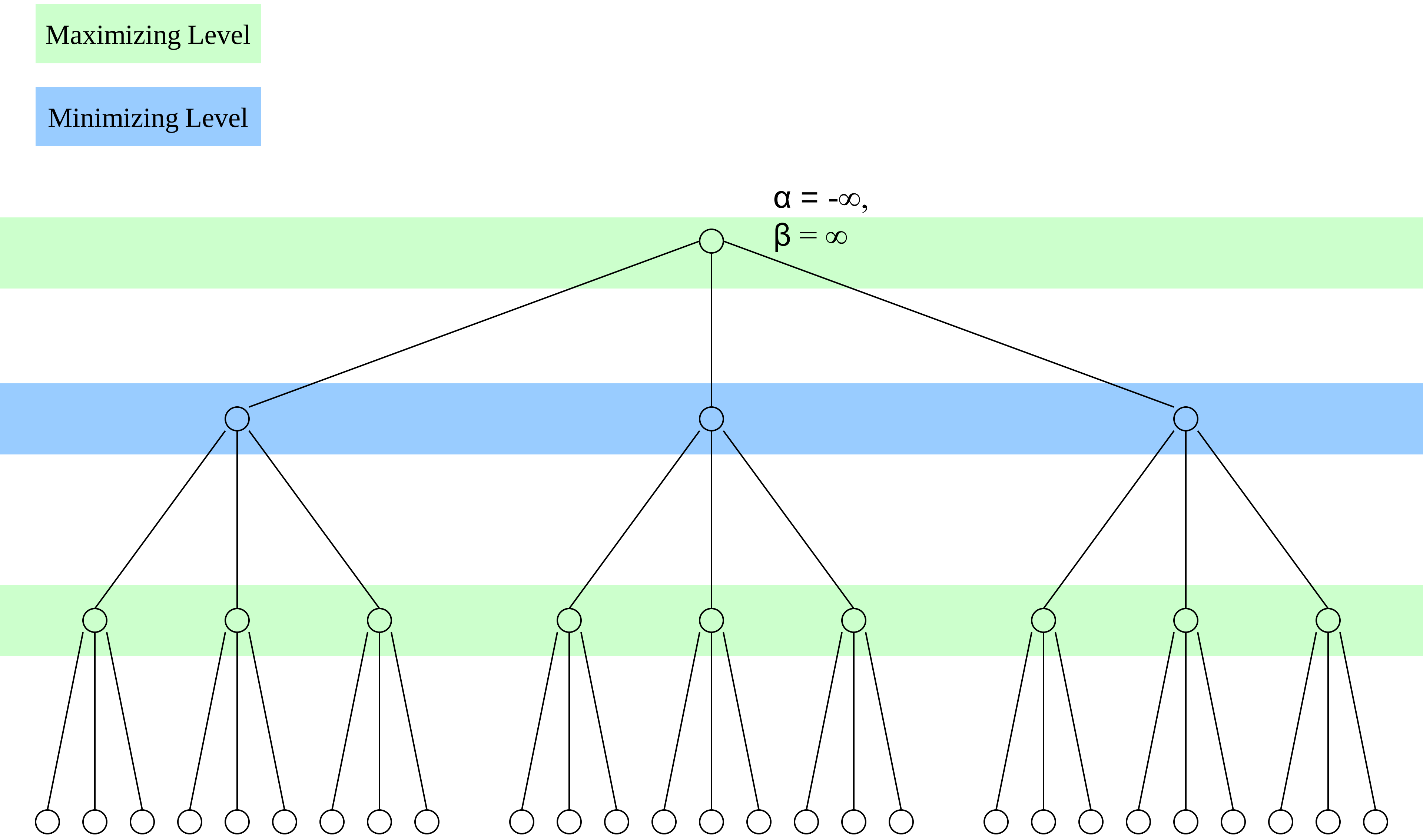
      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;



Example from Patrick Winston's AI book


Maximizing Level
Minimizing Level
α = -∞,
β = ∞

void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

   (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;
4. Else if at a maximizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

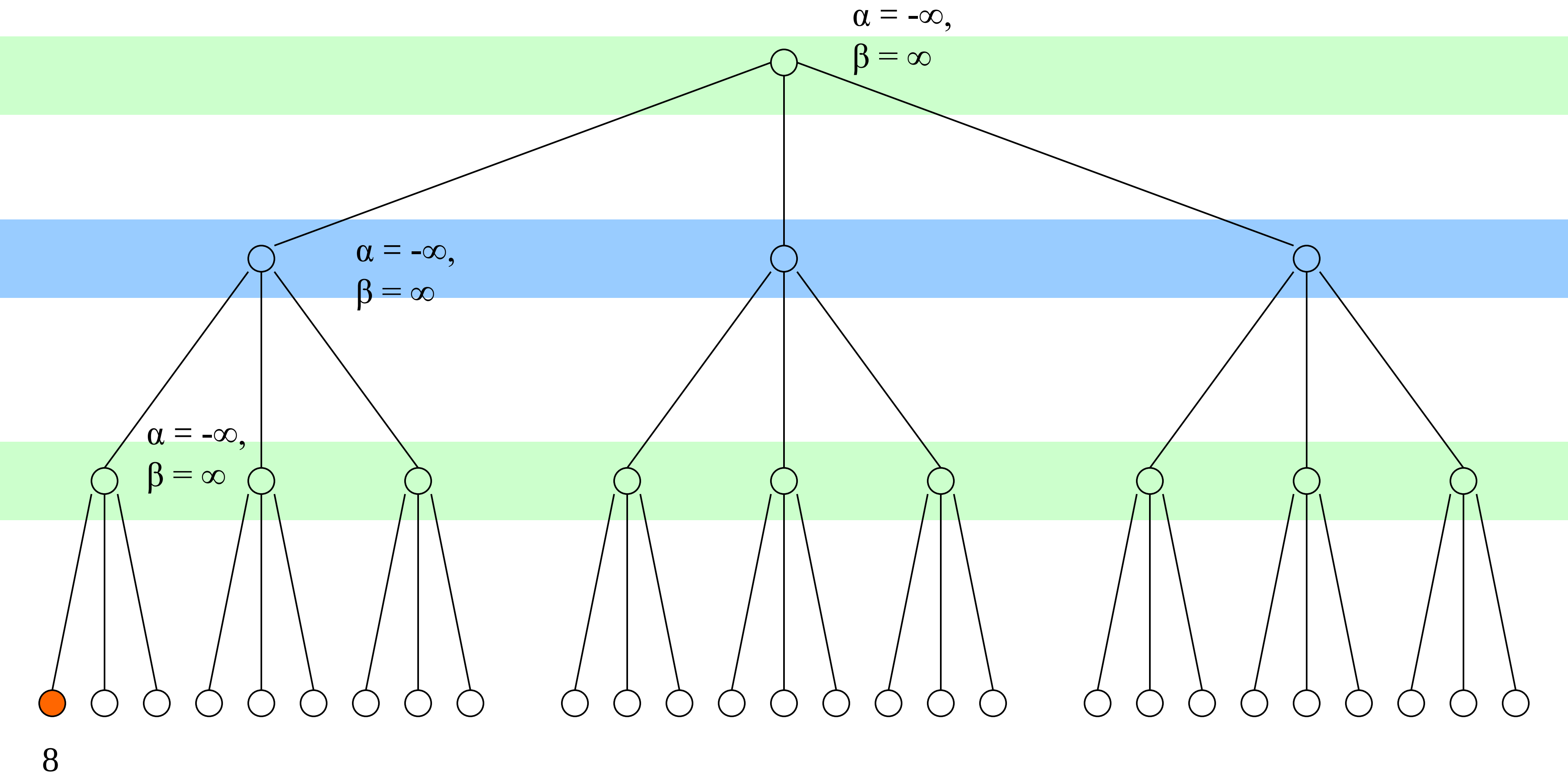
   (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Maximizing Level

Minimizing Level

Step 1



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;

4. Else if at a maximizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

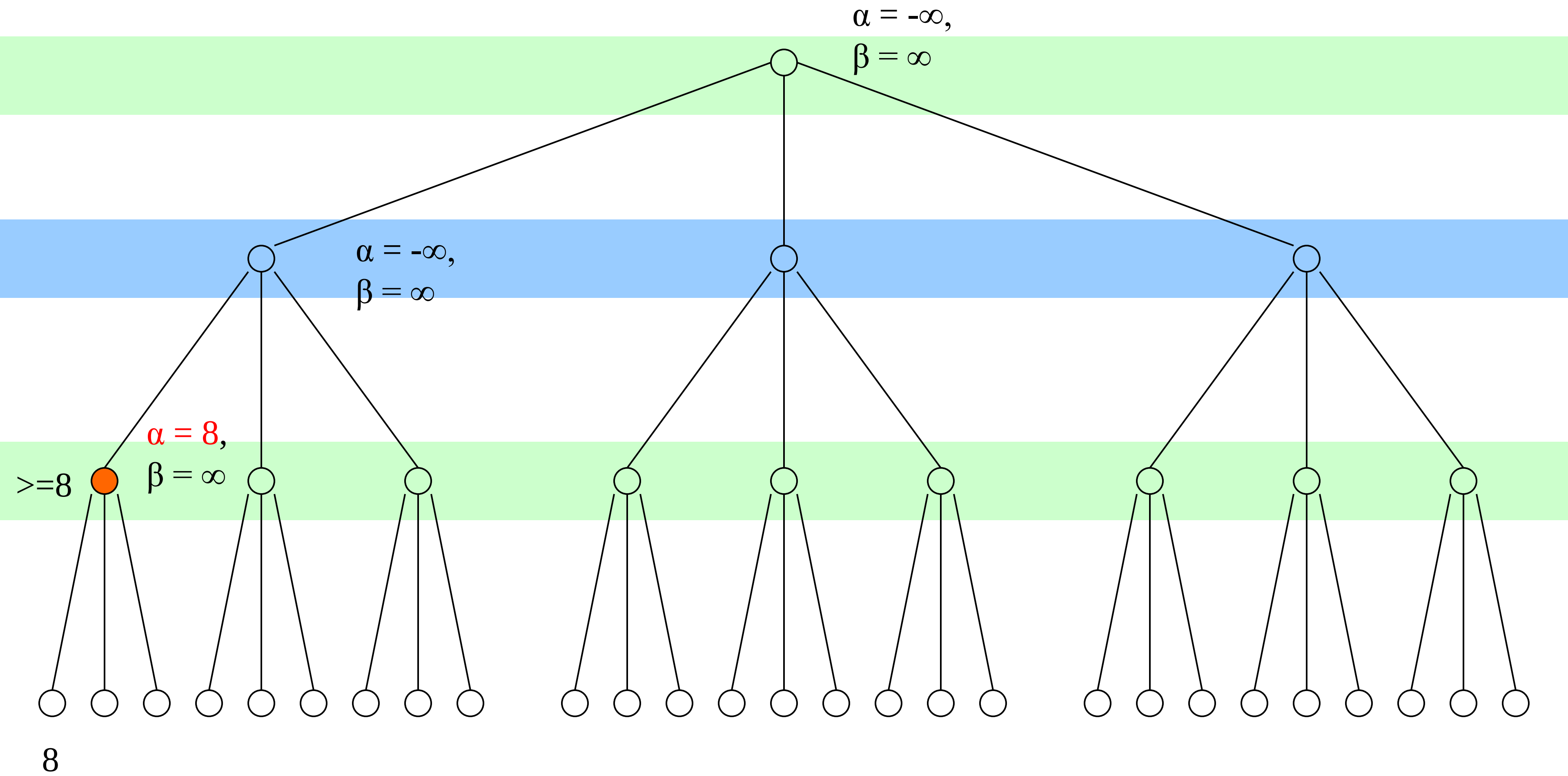
      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Maximizing Level

Minimizing Level

Step 2



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=$-\infty$, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

   (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;
4. Else if at a maximizing level,

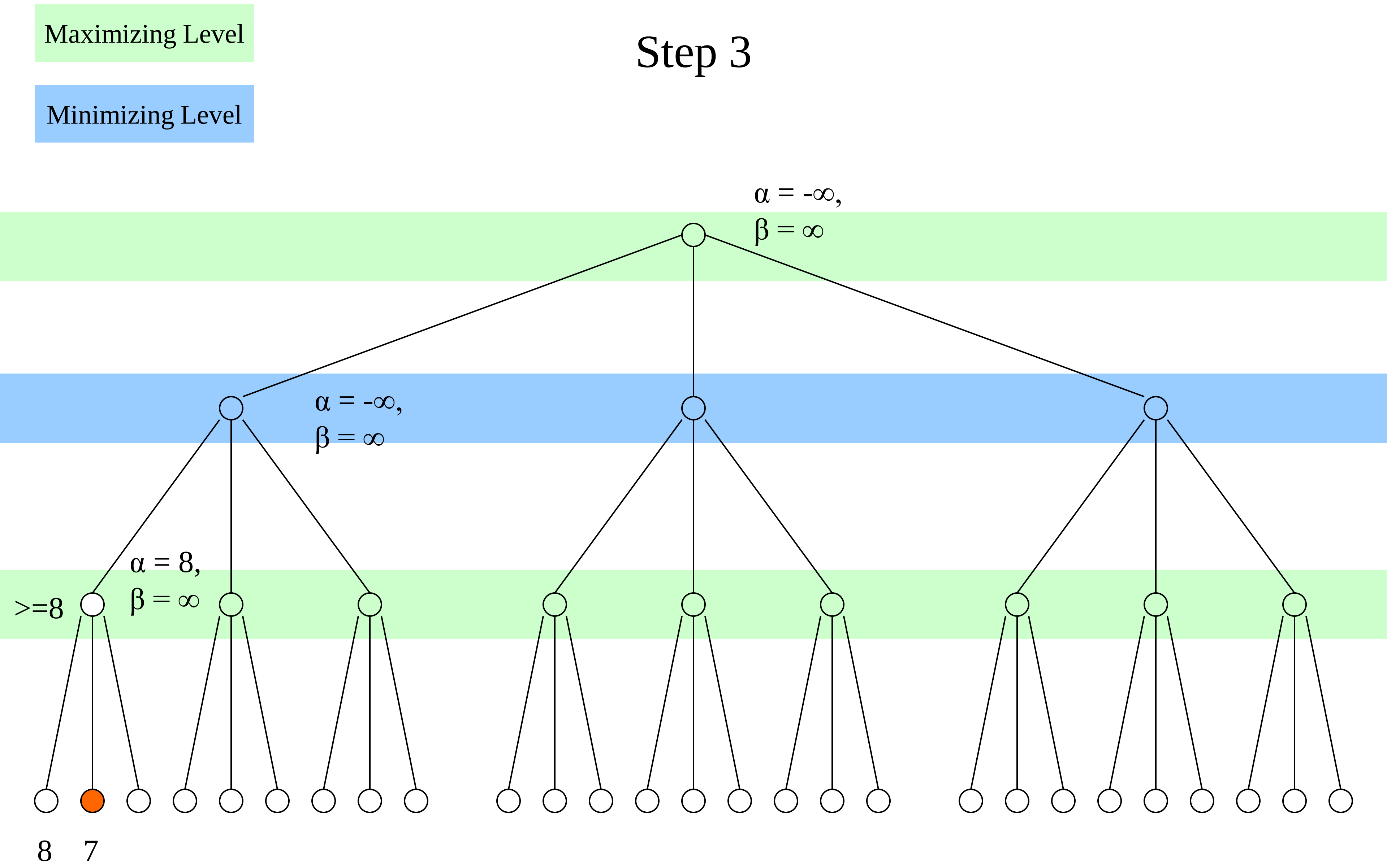
   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;   In loop

   (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

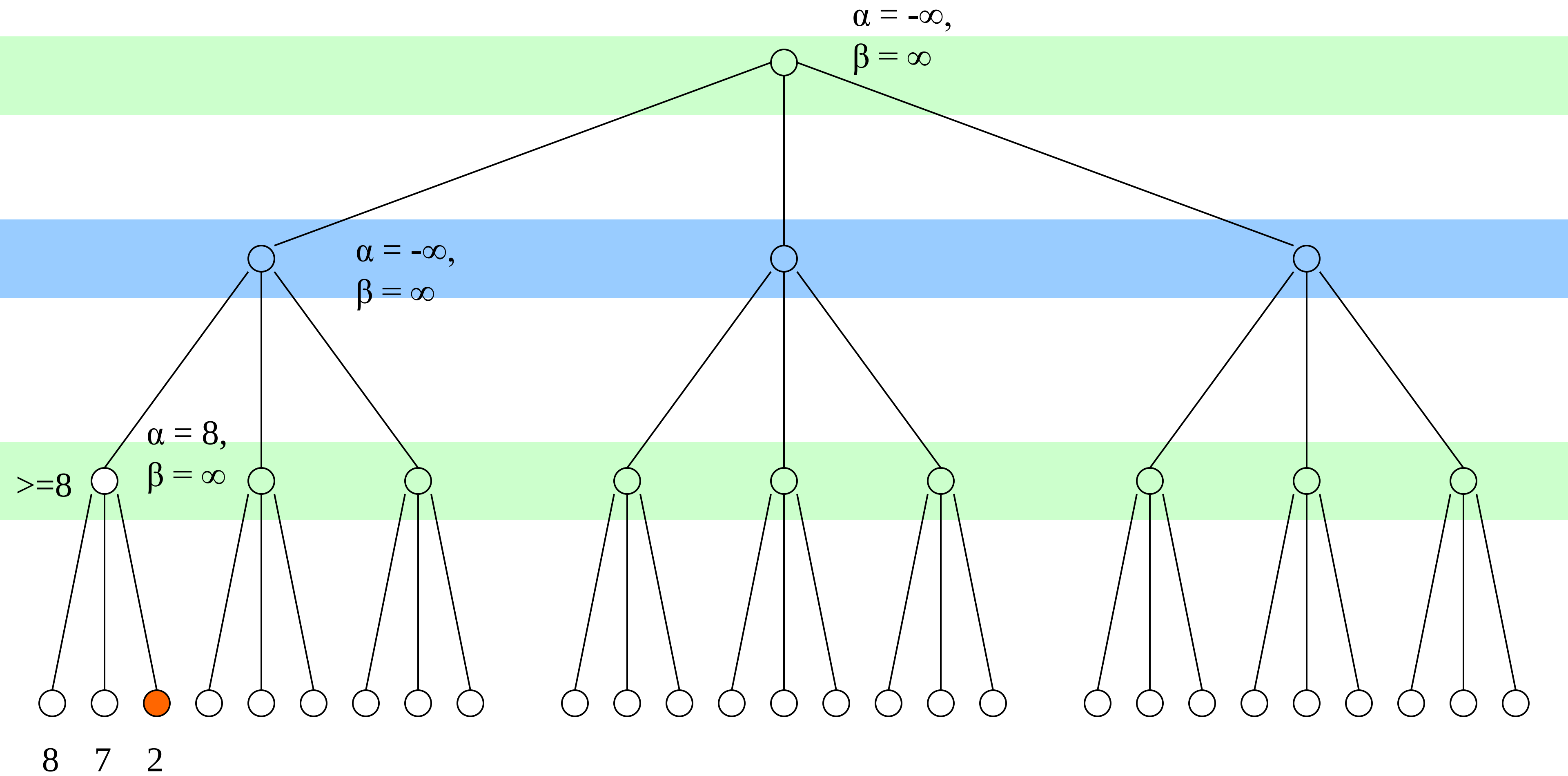
Step 3



Step 4

Maximizing Level

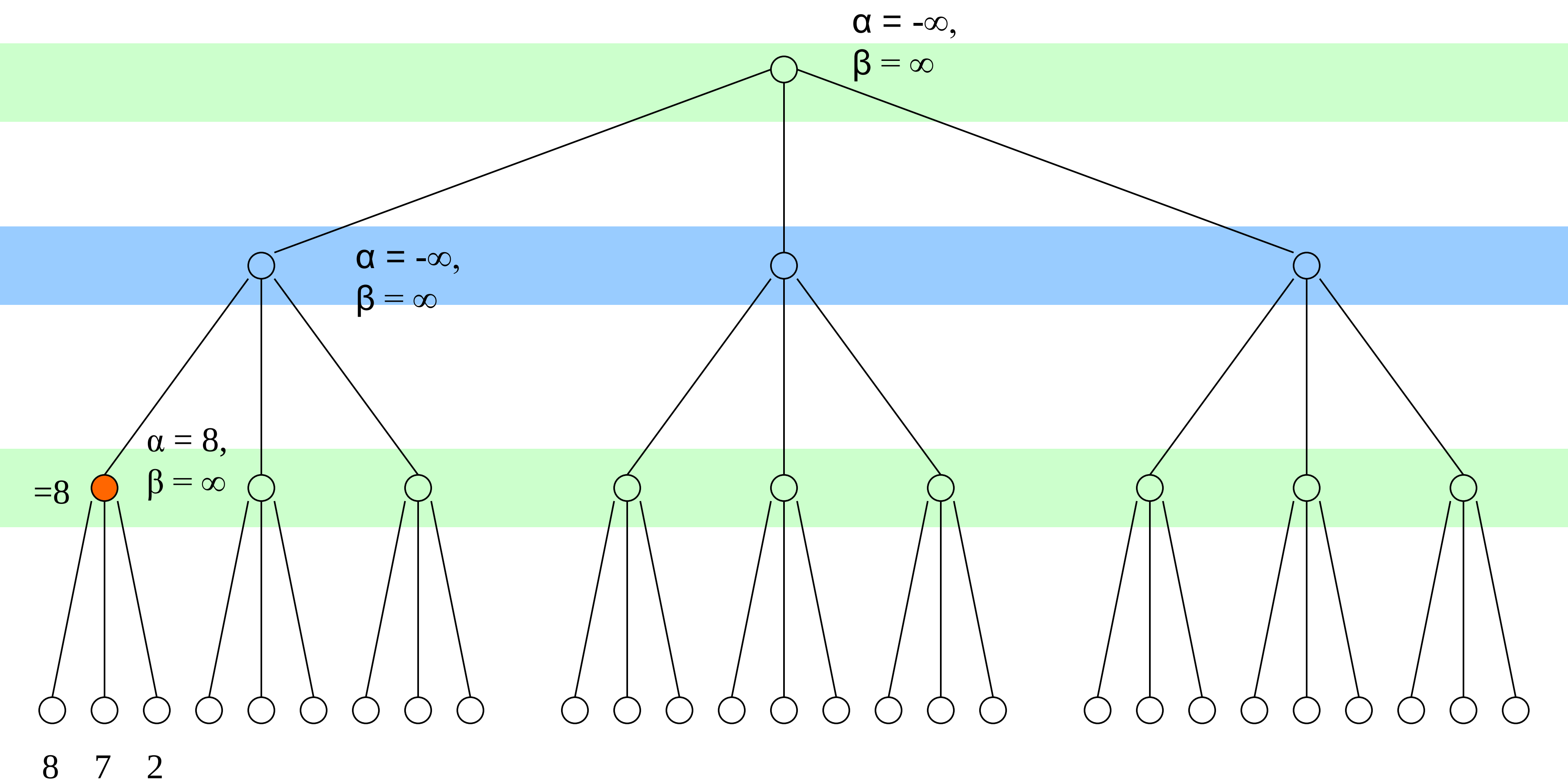
Minimizing Level



Maximizing Level

Minimizing Level

Step 5



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child; (second child)

      (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;
4. Else if at a maximizing level,

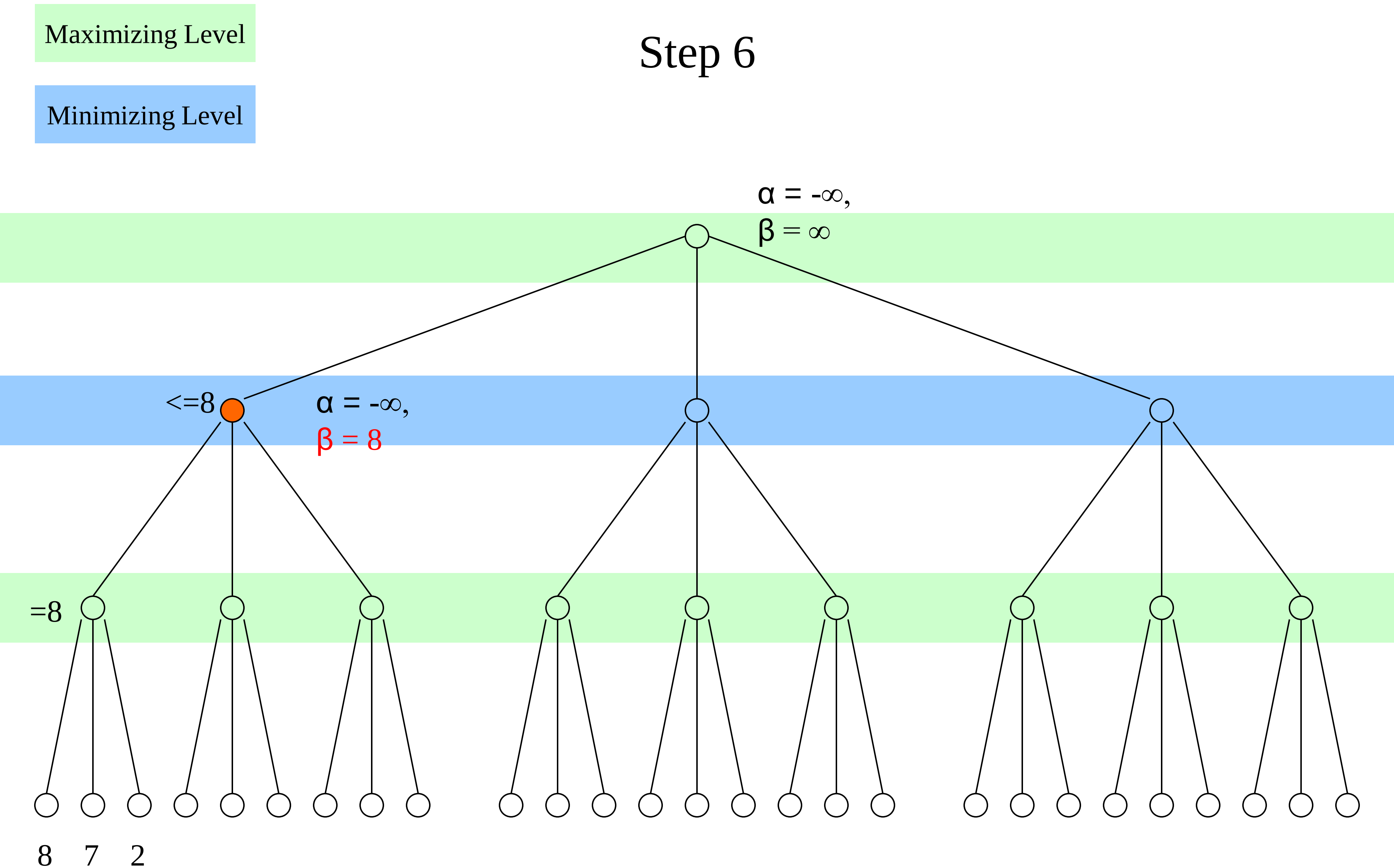
   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

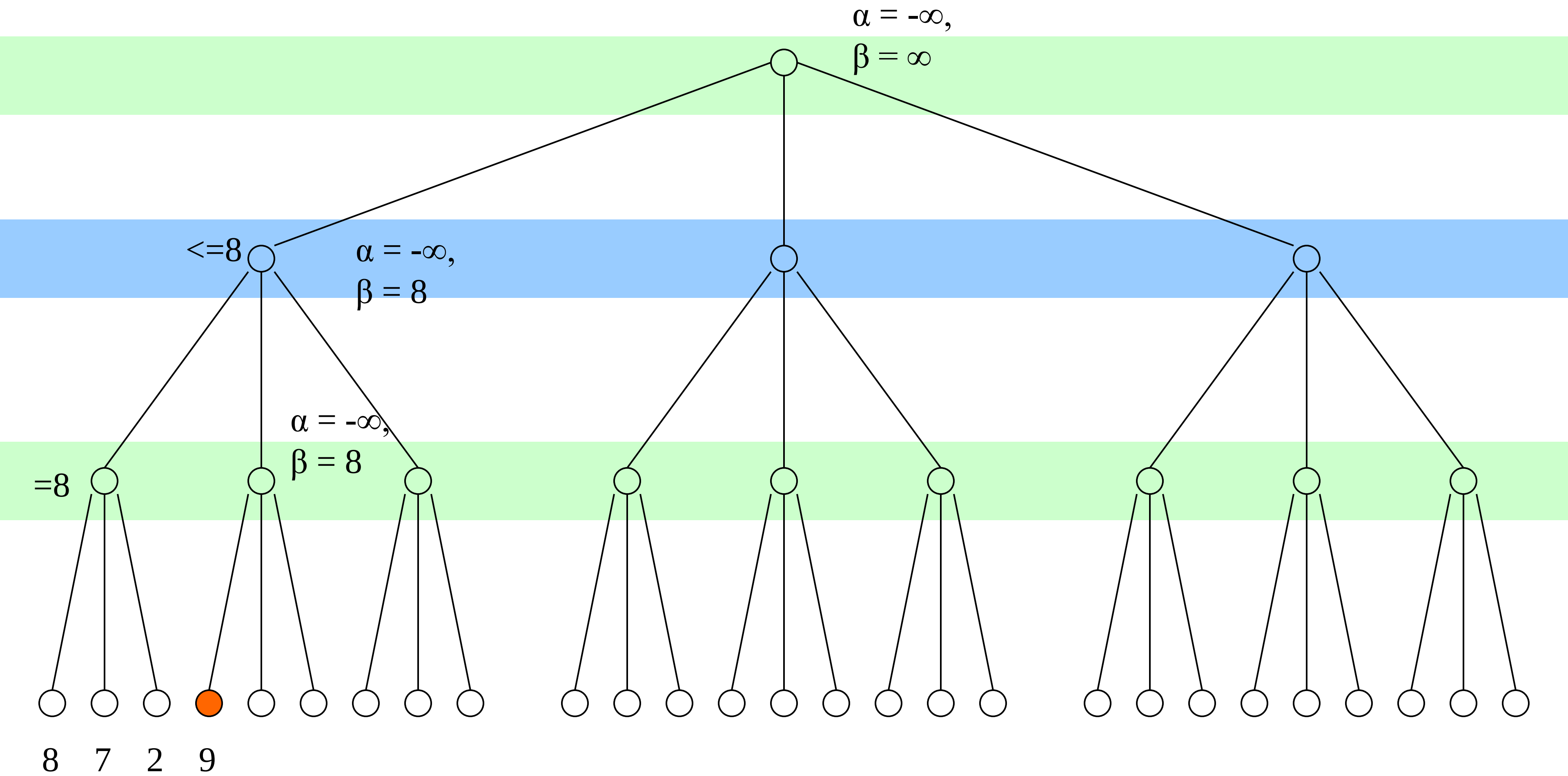
Step 6



Maximizing Level

Minimizing Level

Step 7



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

   (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;

4. Else if at a maximizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

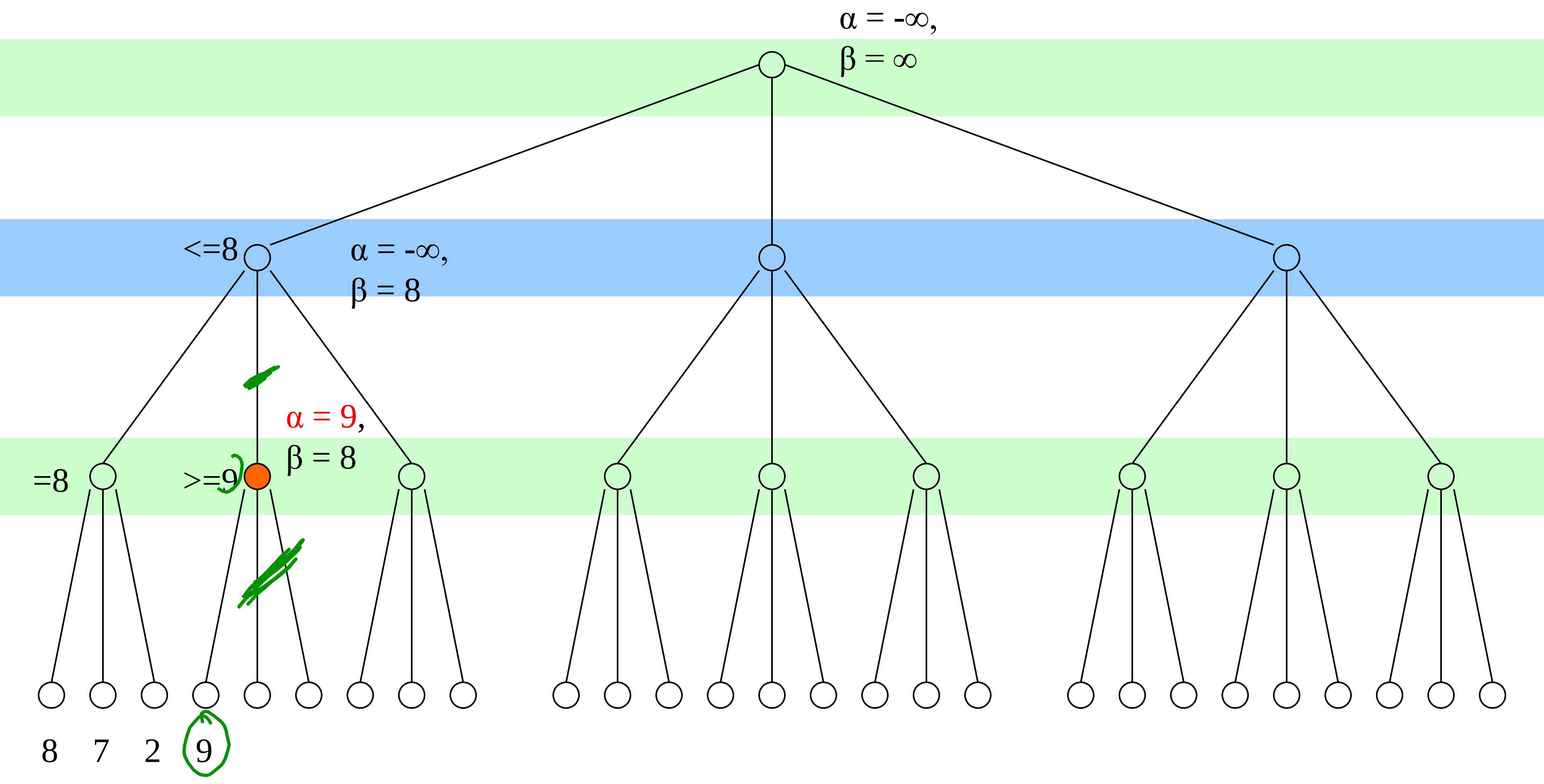
   (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Step 8

Maximizing Level

Minimizing Level



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,   PRUNE now

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;

4. Else if at a maximizing level,

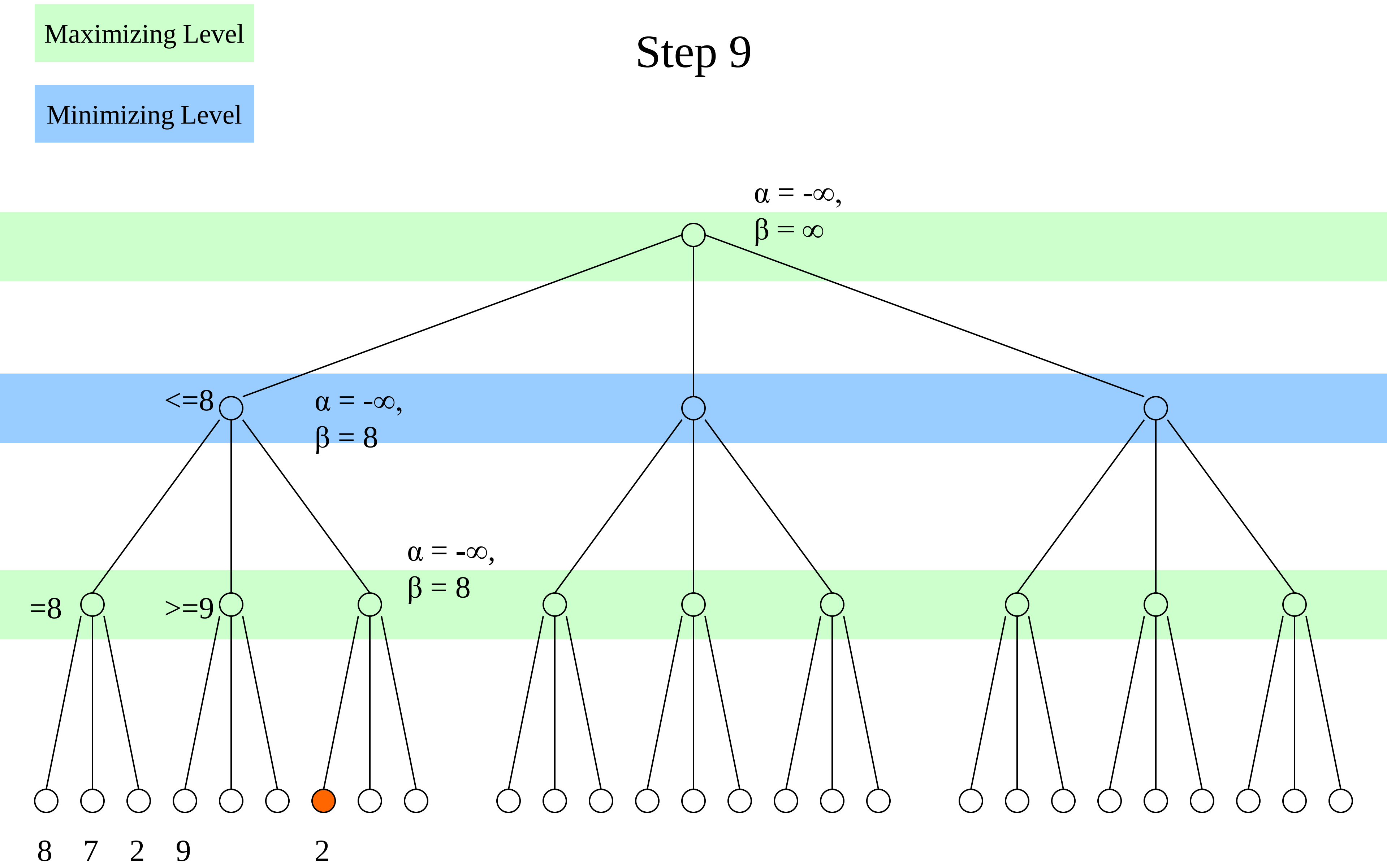
   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Step 9



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;
4. Else if at a maximizing level,

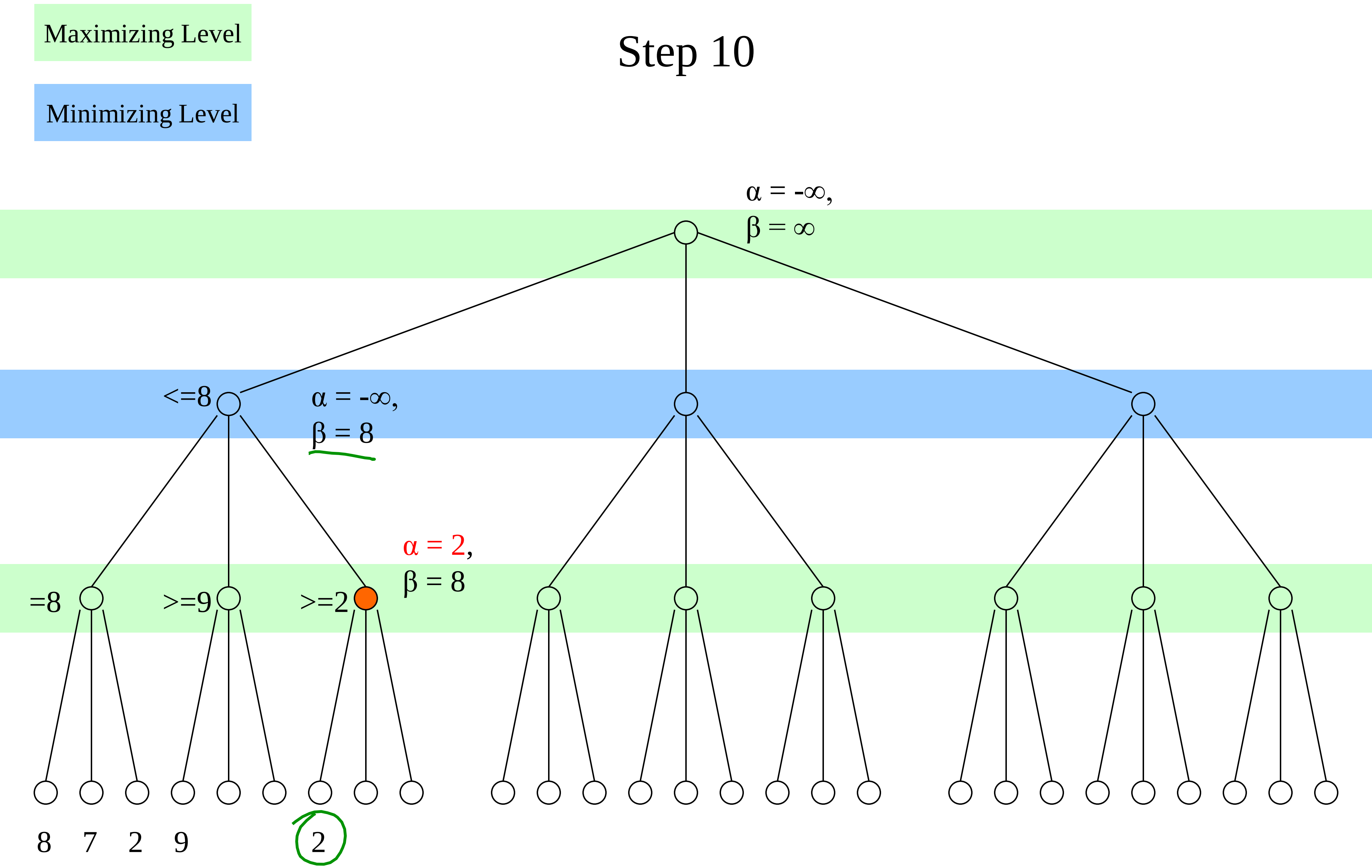
   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

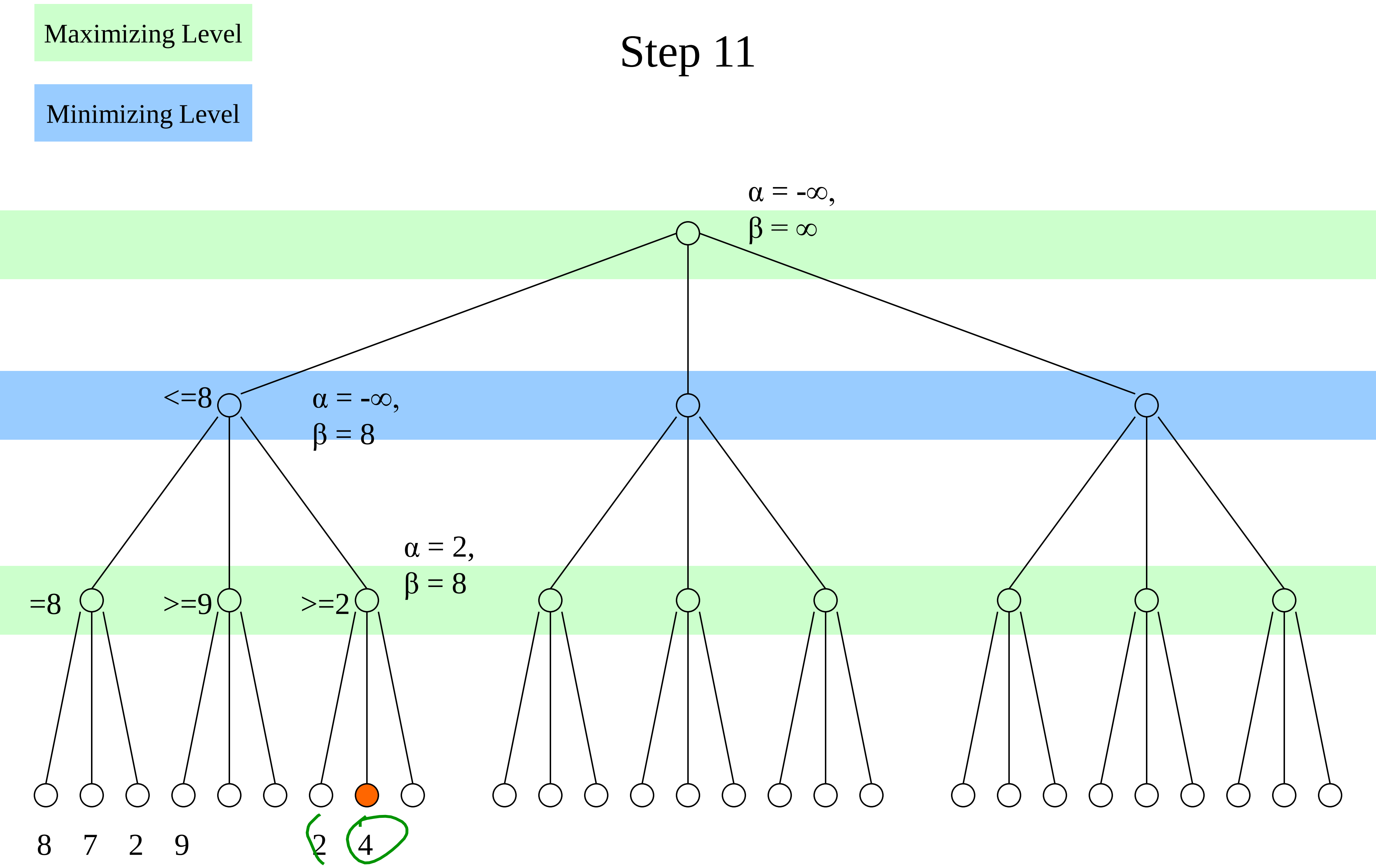
      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

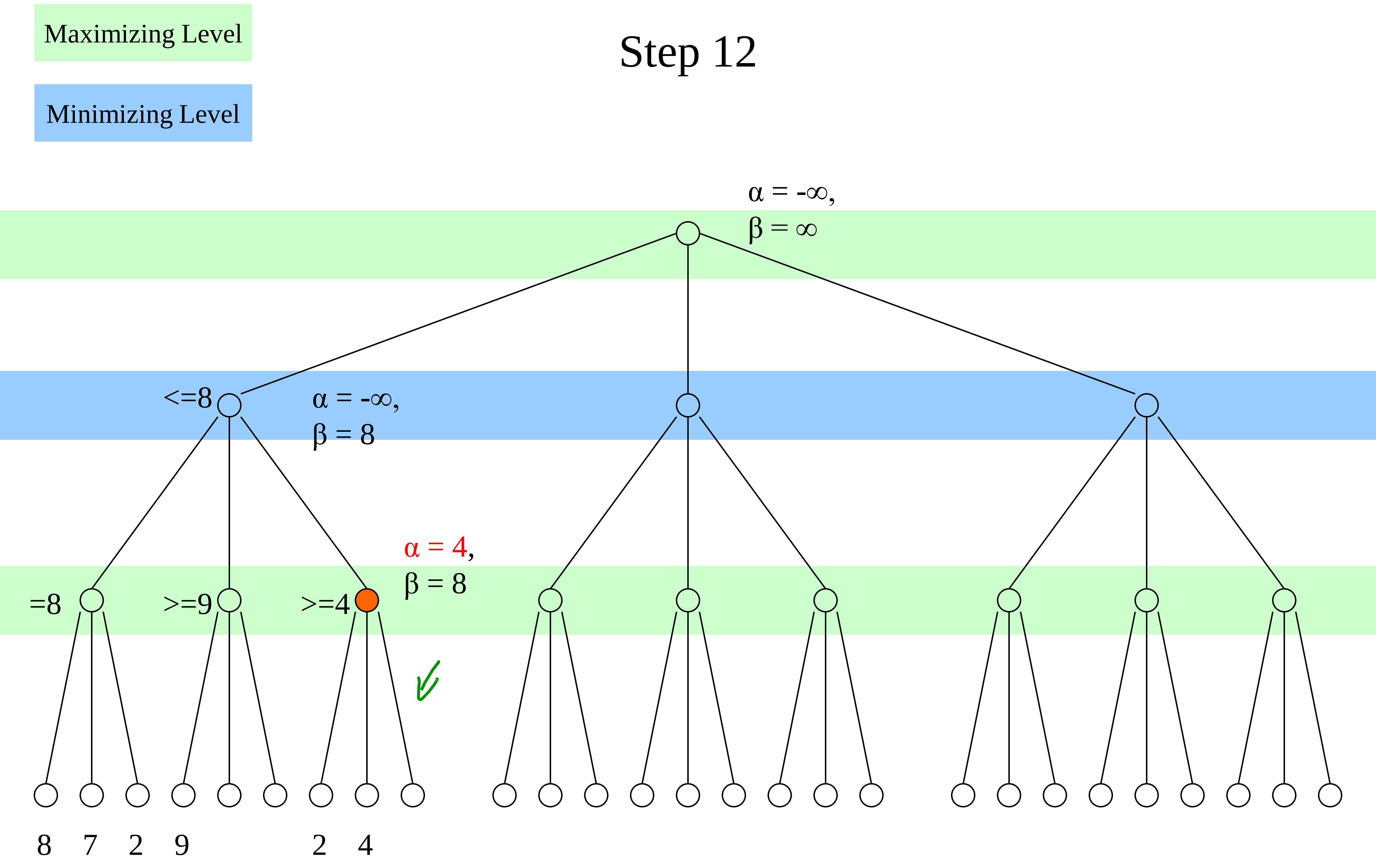
Step 10



Step 11



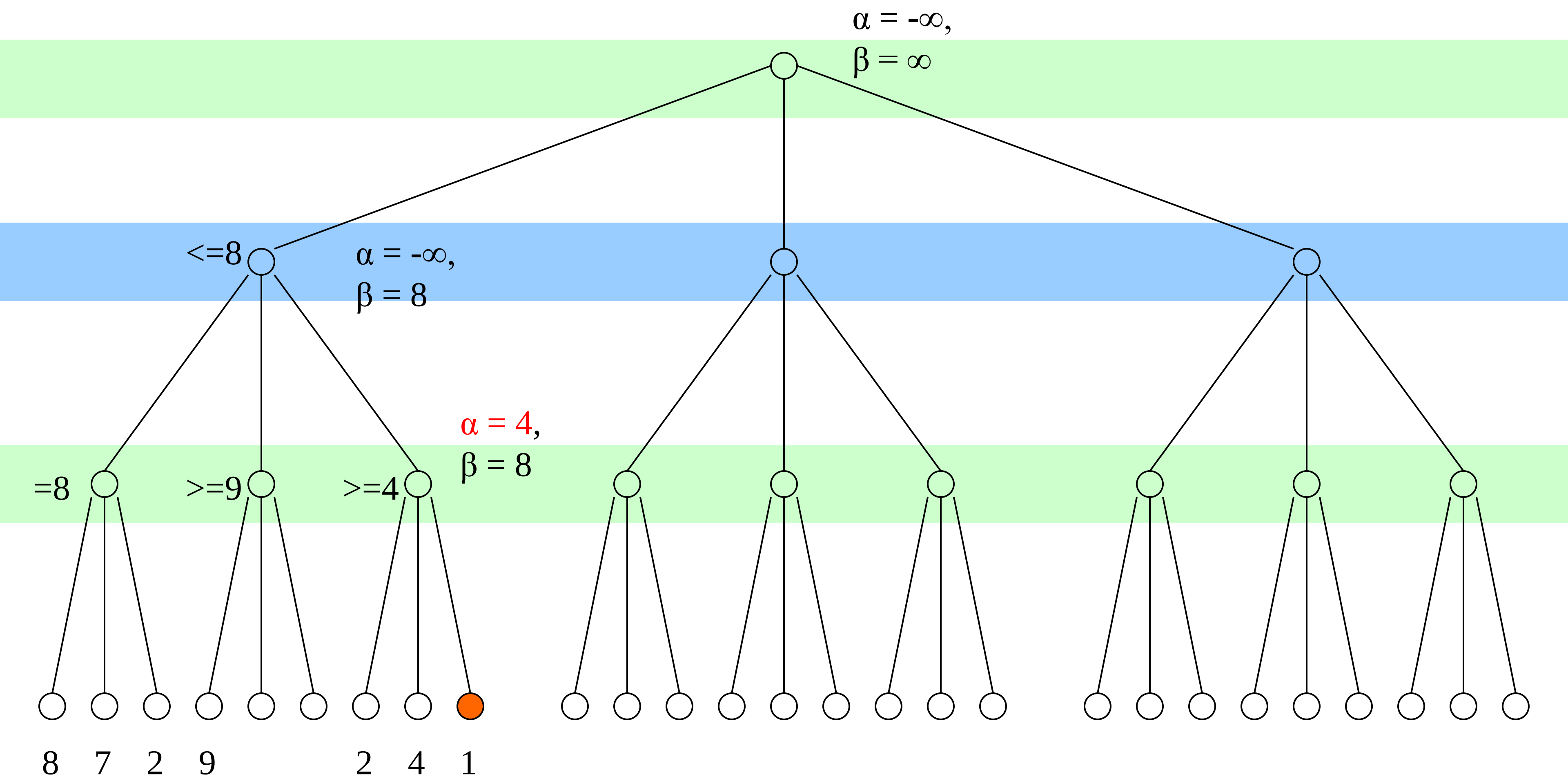
Step 12



Maximizing Level

Minimizing Level

Step 13



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=-∞, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;

4. Else if at a maximizing level,

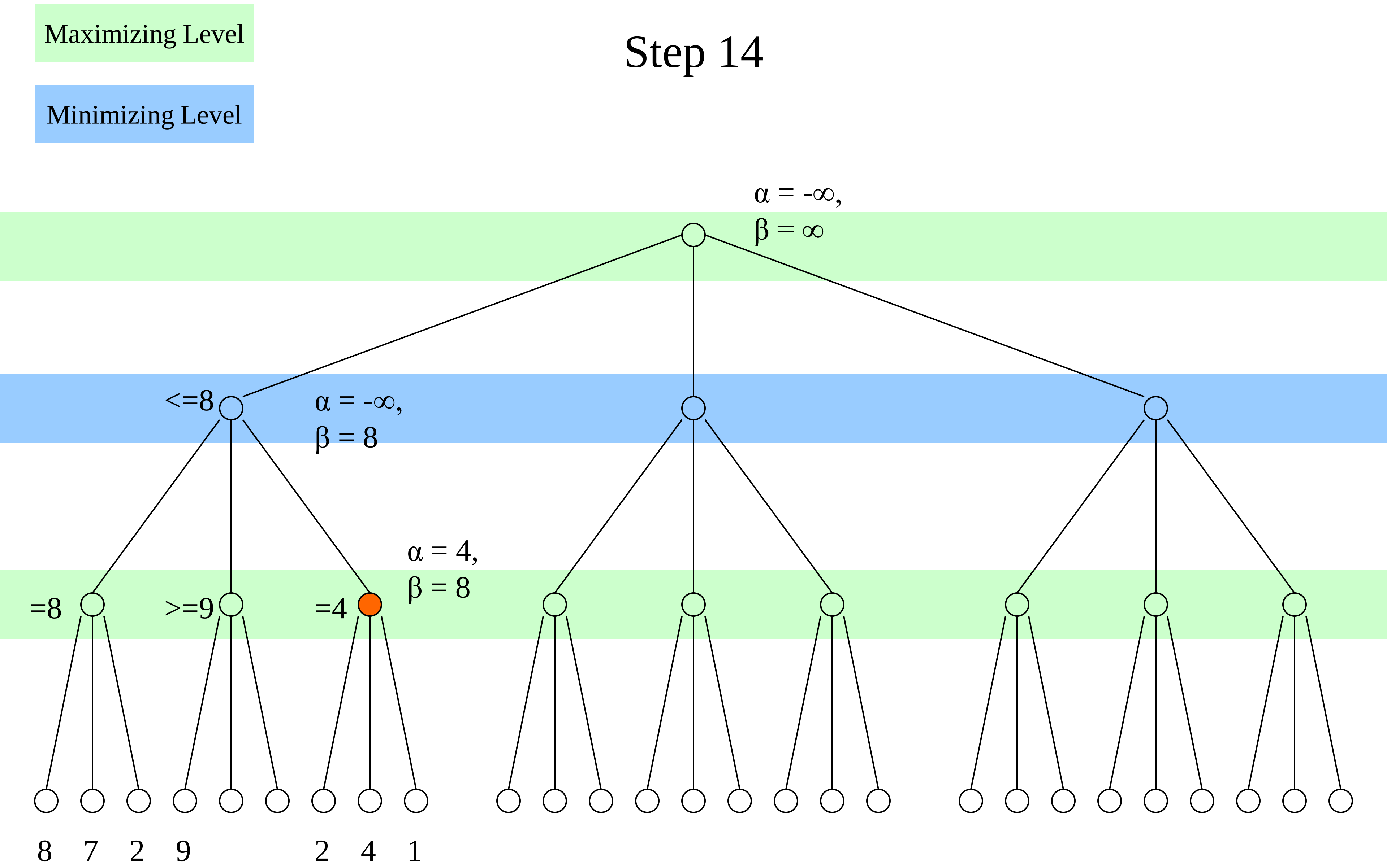
   until all children are examined or alpha >= beta,

      (a) Recursive call Alpha_Beta_Procedure on a child;

      (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Step 14



Maximizing Level
Minimizing Level
α = -∞,
β = ∞
<=8
α = -∞,
β = 8
α = 4,
β = 8
=8
>=9
=4
8
7
2
9
2
4
1

void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha= $-\infty$, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

    until all children are examined or alpha >= beta,

        (a) Recursive call Alpha_Beta_Procedure on a child;

        (b) If *score < beta, beta = *score; // update upper bound

    *score = beta;

4. Else if at a maximizing level,

    until all children are examined or alpha >= beta,

        (a) Recursive call Alpha_Beta_Procedure on a child;

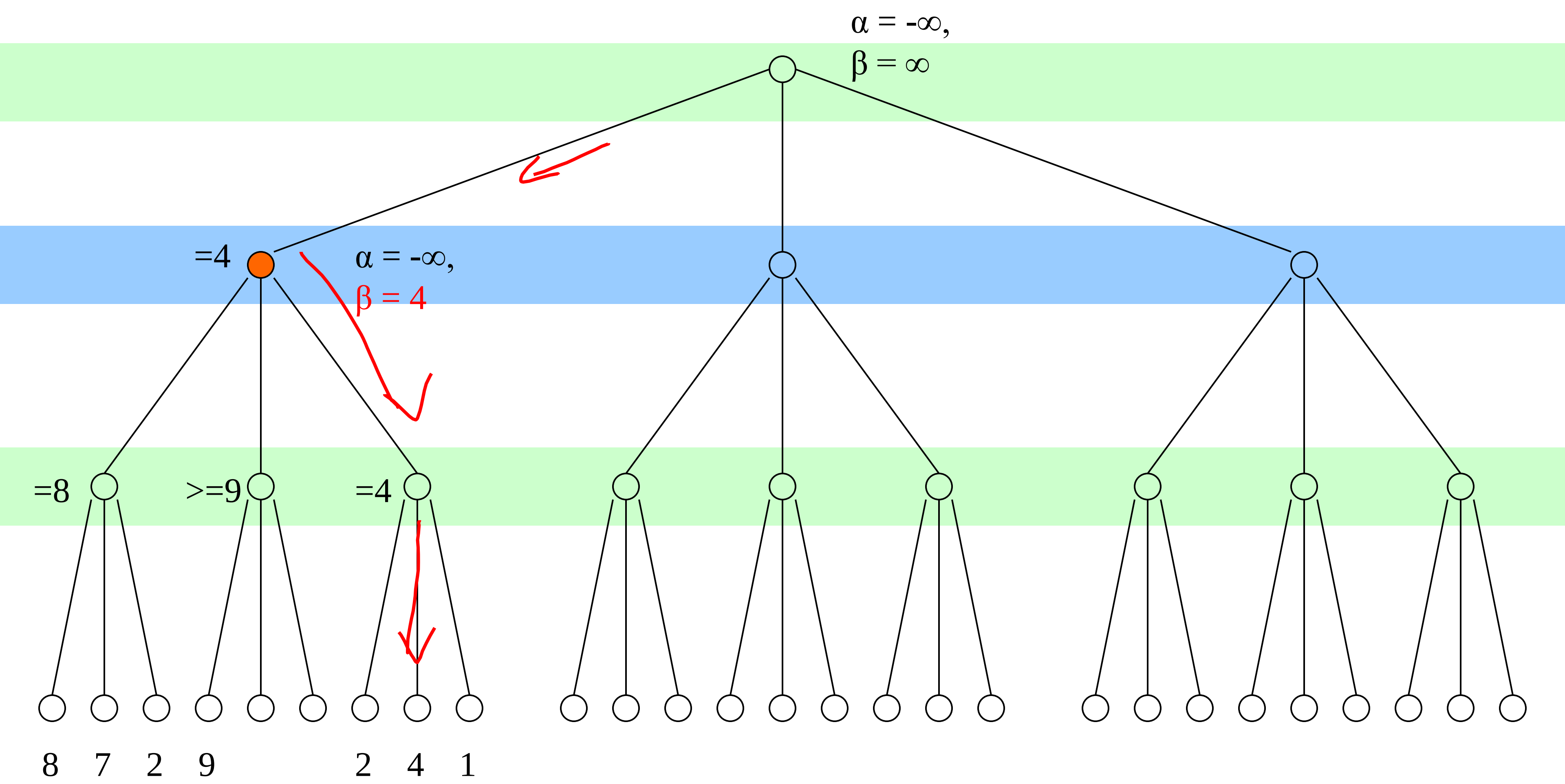
        (b) If *score > alpha, alpha = *score; // update lower bound

    *score = alpha;

Step 15

Maximizing Level

Minimizing Level



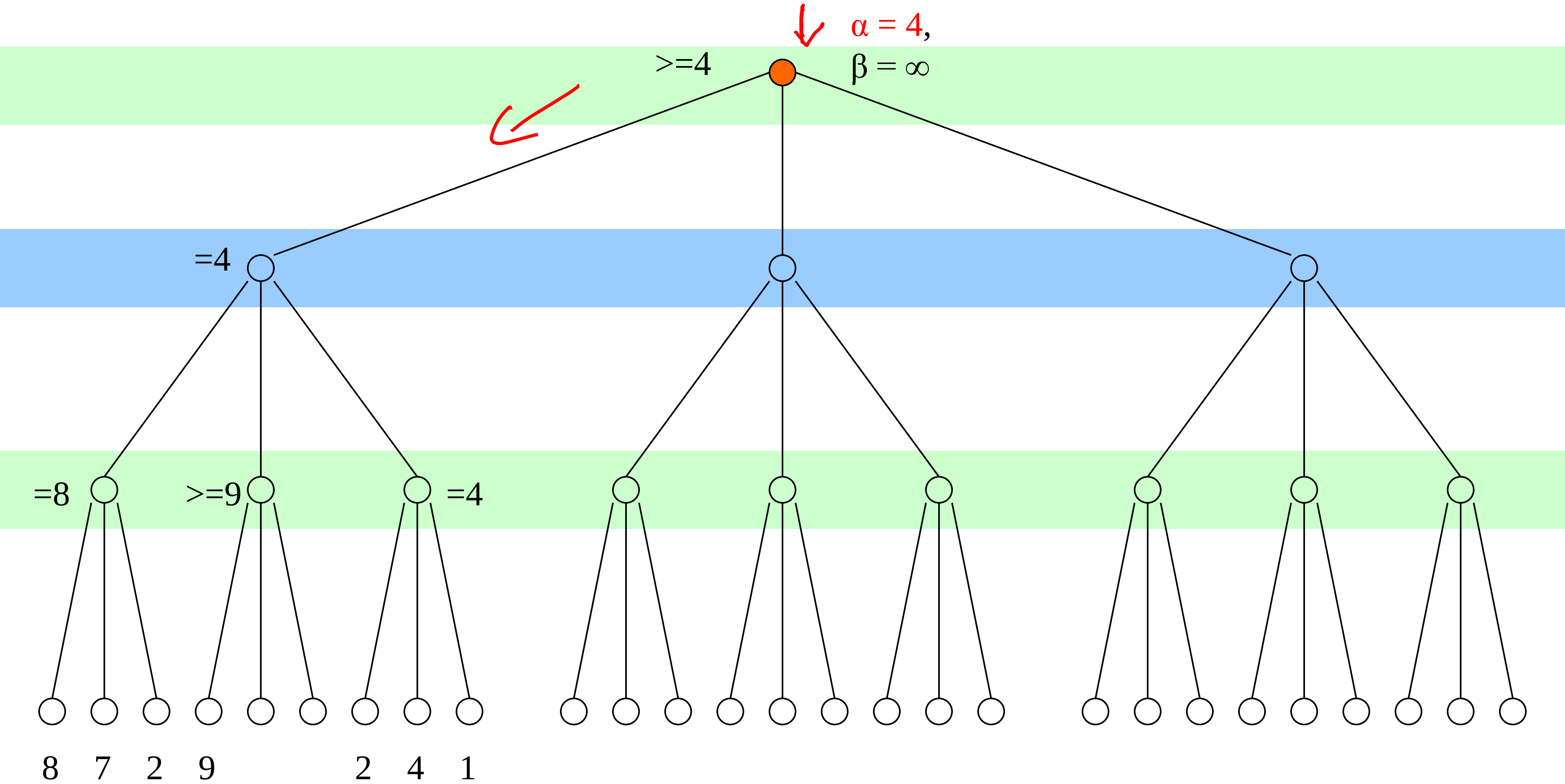
void Alpha_Beta_Procedure(alpha, beta, &score)

```
1.  If at the root, set alpha=-∞, beta = ∞
2.  If at the leaf, *score = static_evaluator(current_board, role); return
3.  If at a minimizing level,
      until all children are examined or alpha >= beta,
         (a) Recursive call Alpha_Beta_Procedure on a child;
         (b) If *score < beta, beta = *score;  // update upper bound
      *score = beta;
4.  Else if at a maximizing level,
     until all children are examined or alpha >= beta,
         (a) Recursive call Alpha_Beta_Procedure on a child;
         (b) If *score > alpha, alpha = *score;  // update lower bound
      *score = alpha;
```

Maximizing Level

Minimizing Level

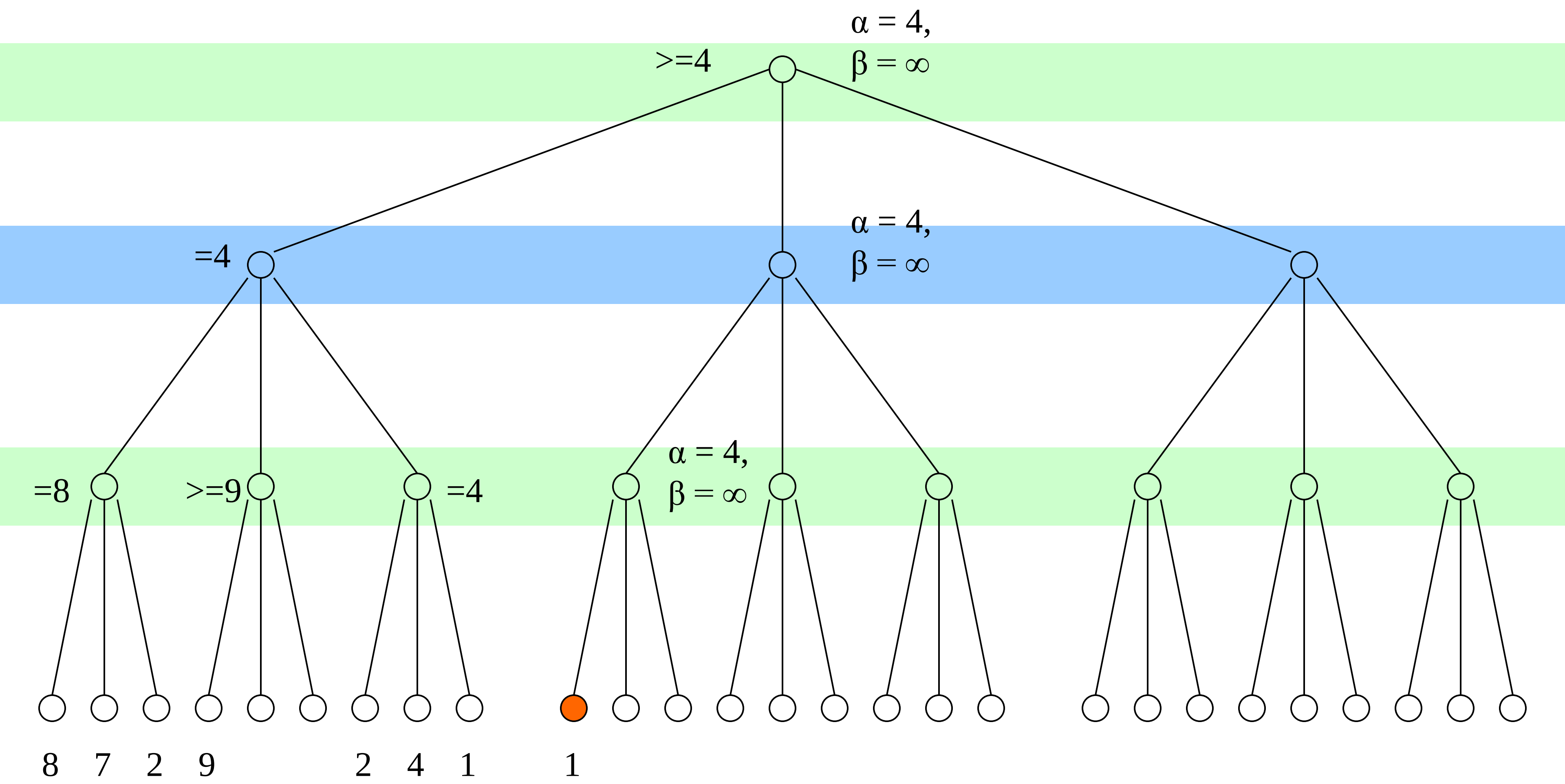
Step 16



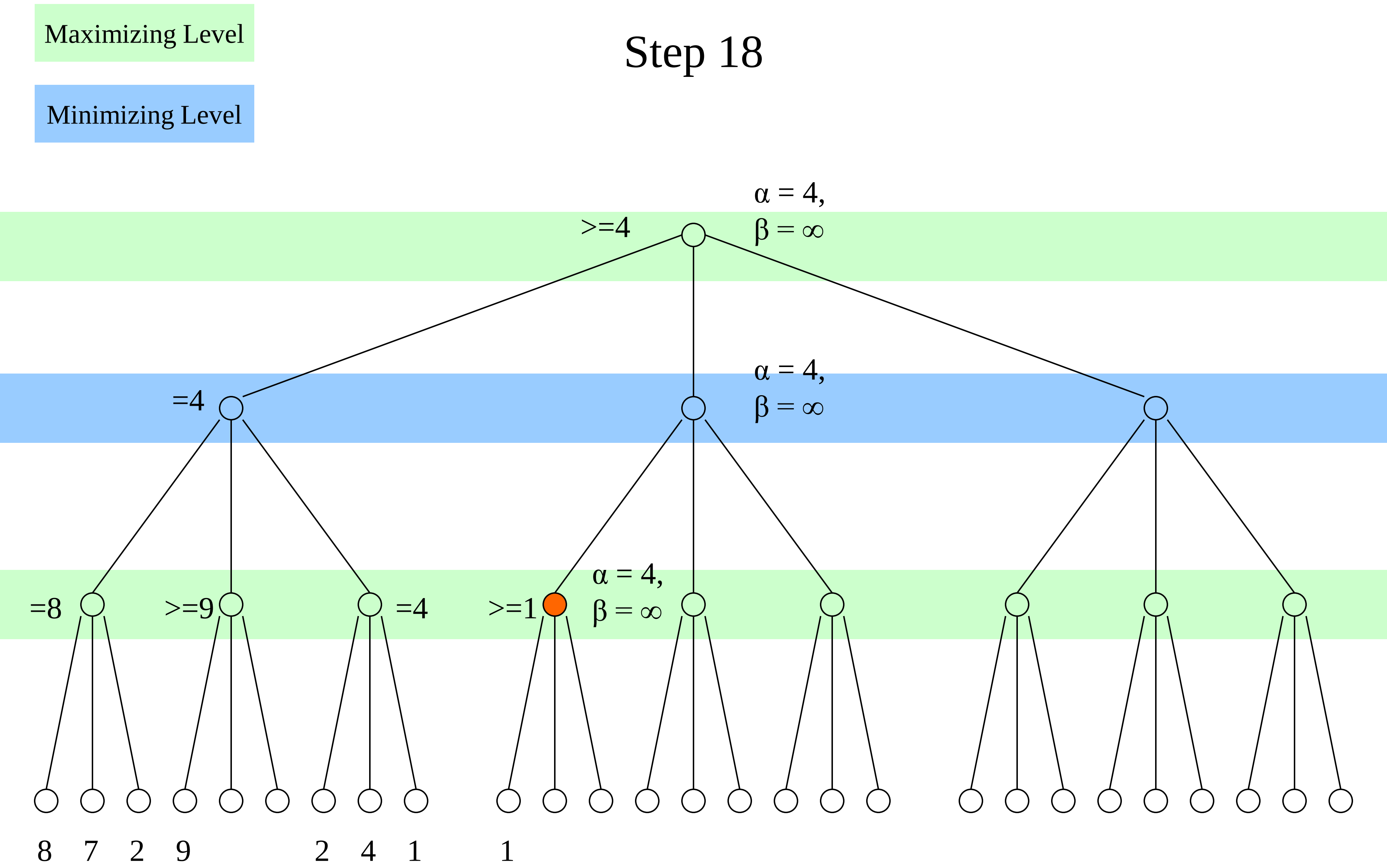
Maximizing Level

Minimizing Level

Step 17



Step 18

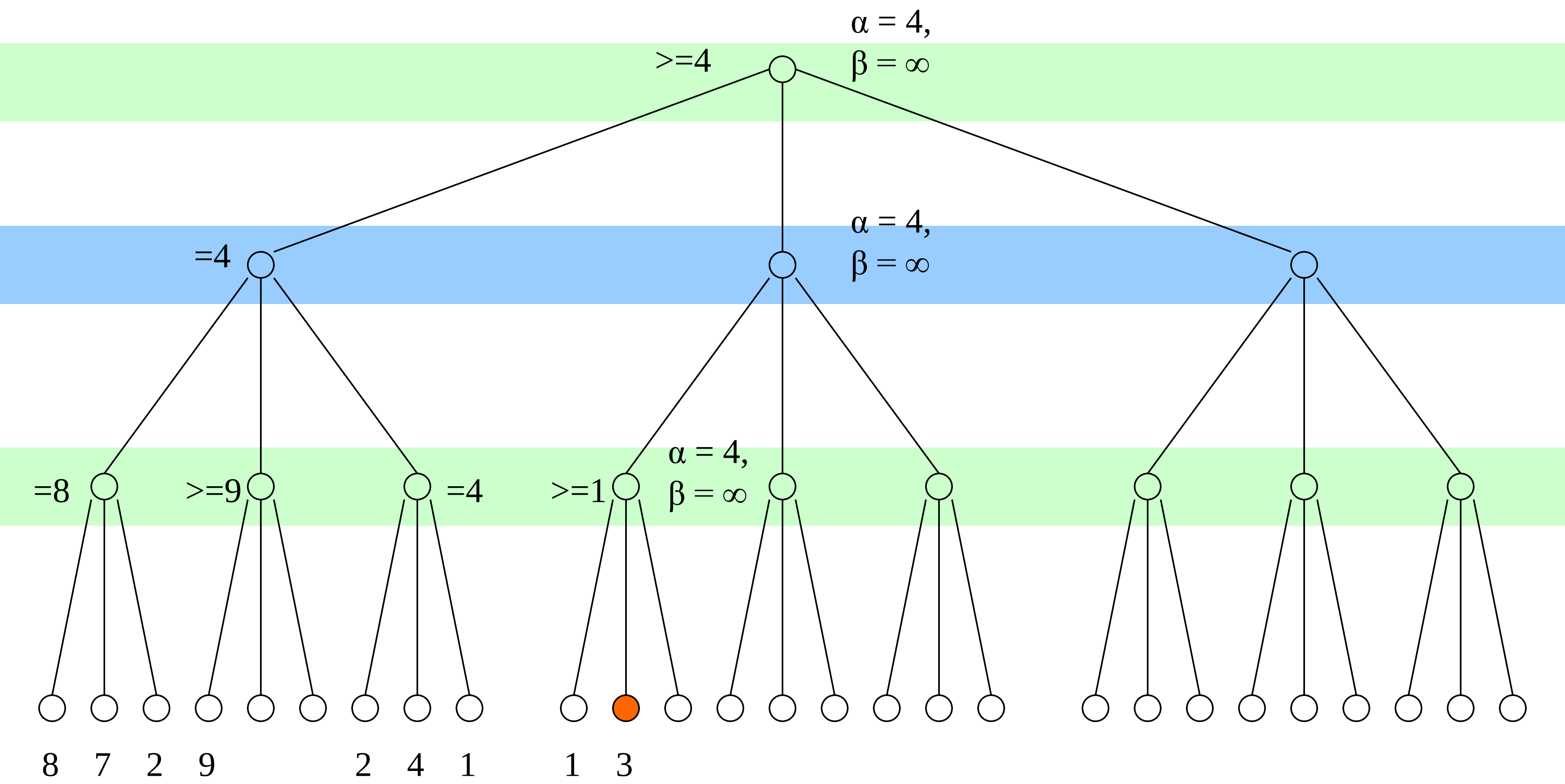


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = ∞
=4
=8
>=9
=4
>=1
α = 4,
β = ∞
8
7
2
9
2
4
1
1

Maximizing Level

Minimizing Level

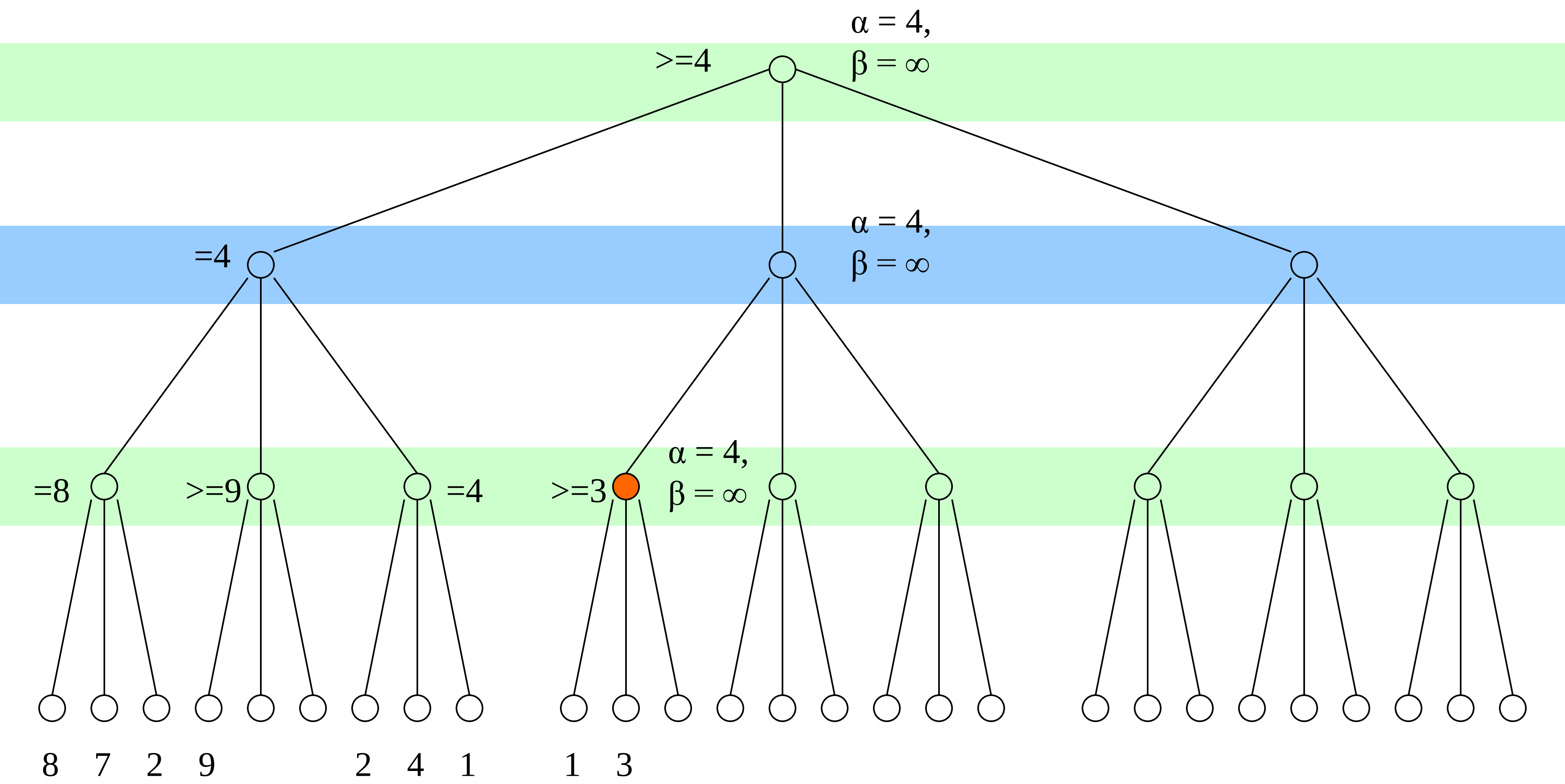
Step 19



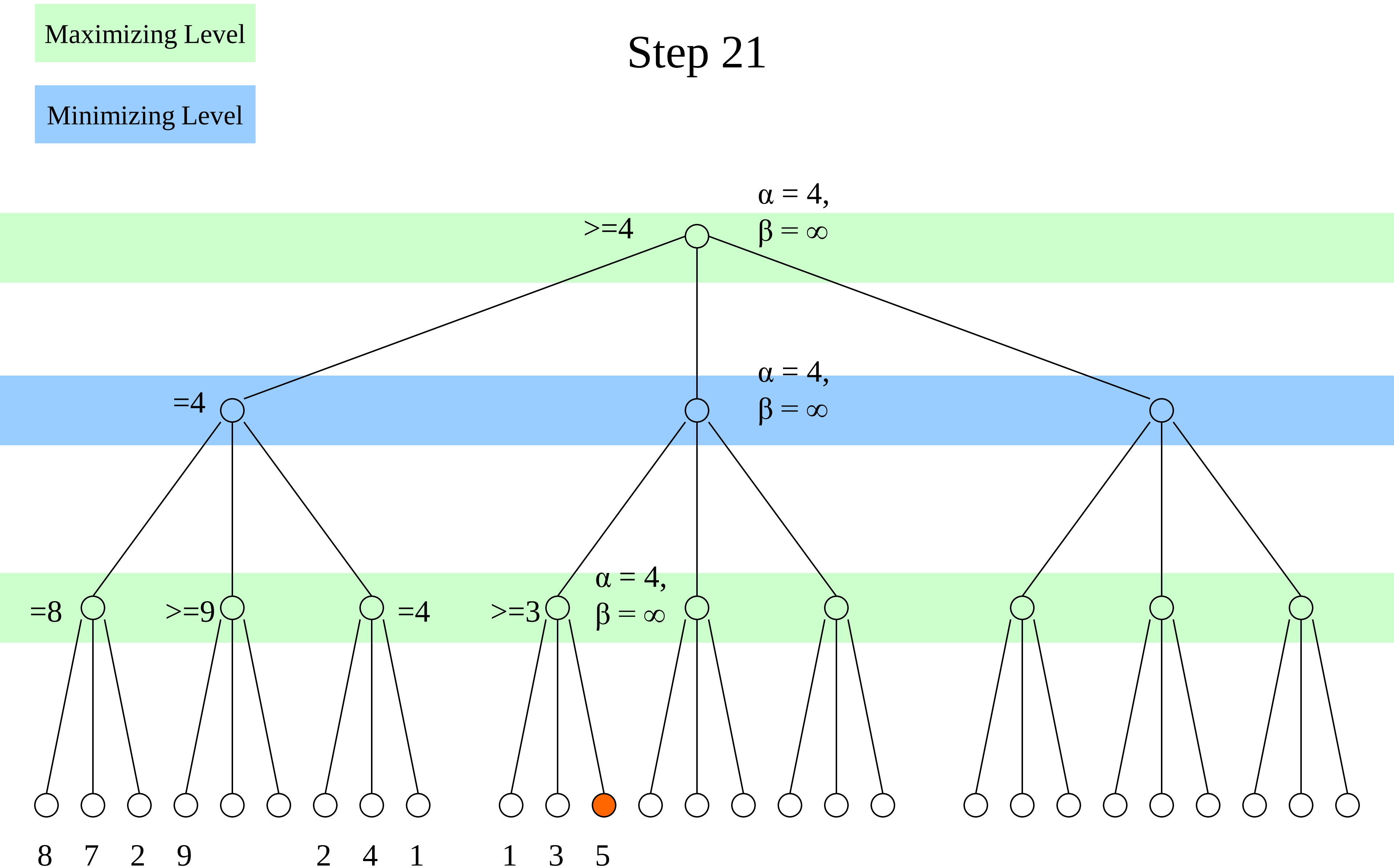
Maximizing Level

Minimizing Level

Step 20



Step 21

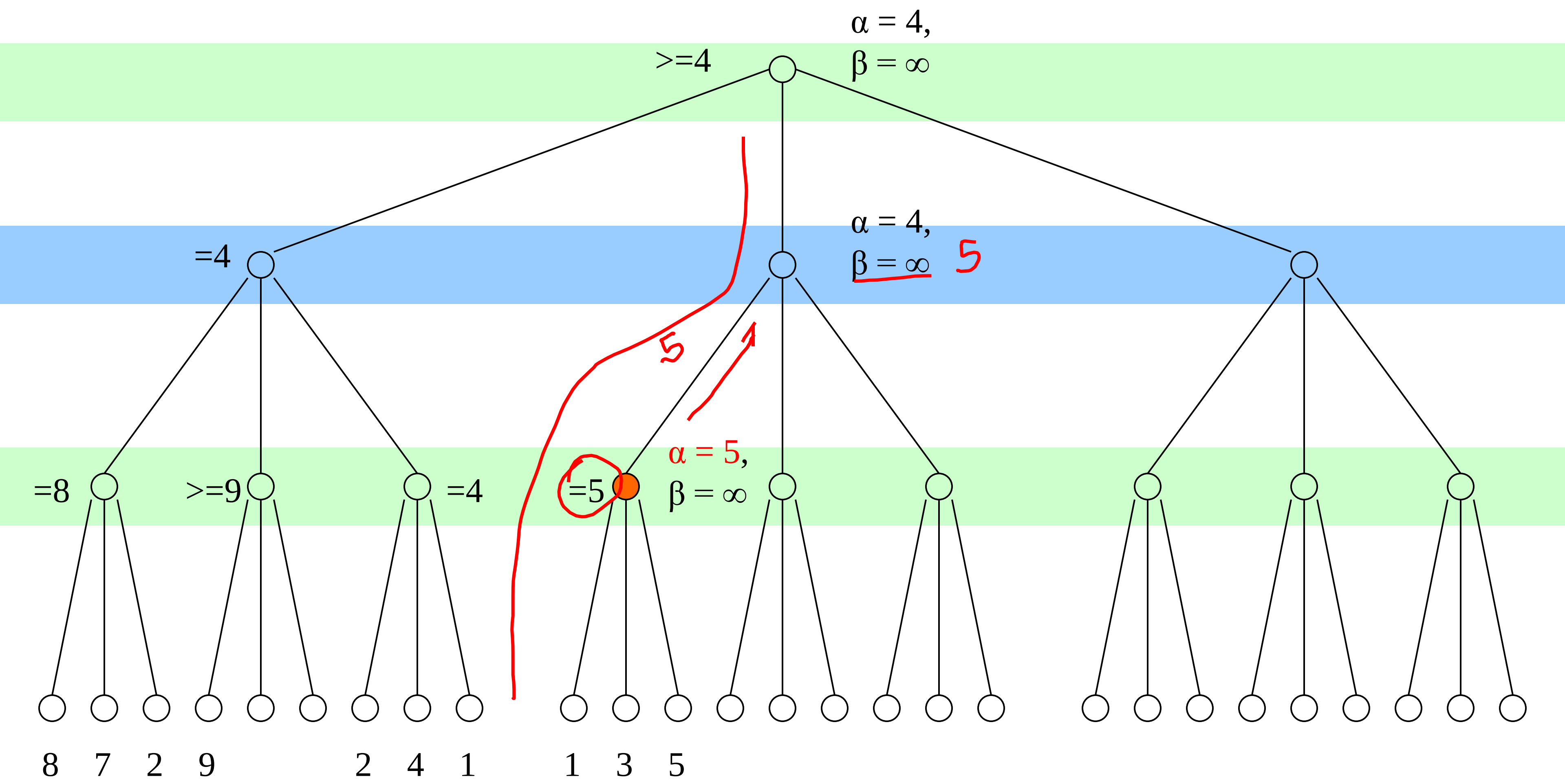


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = ∞
=4
=8
>=9
=4
>=3
α = 4,
β = ∞
8 7 2 9 2 4 1 1 3 5

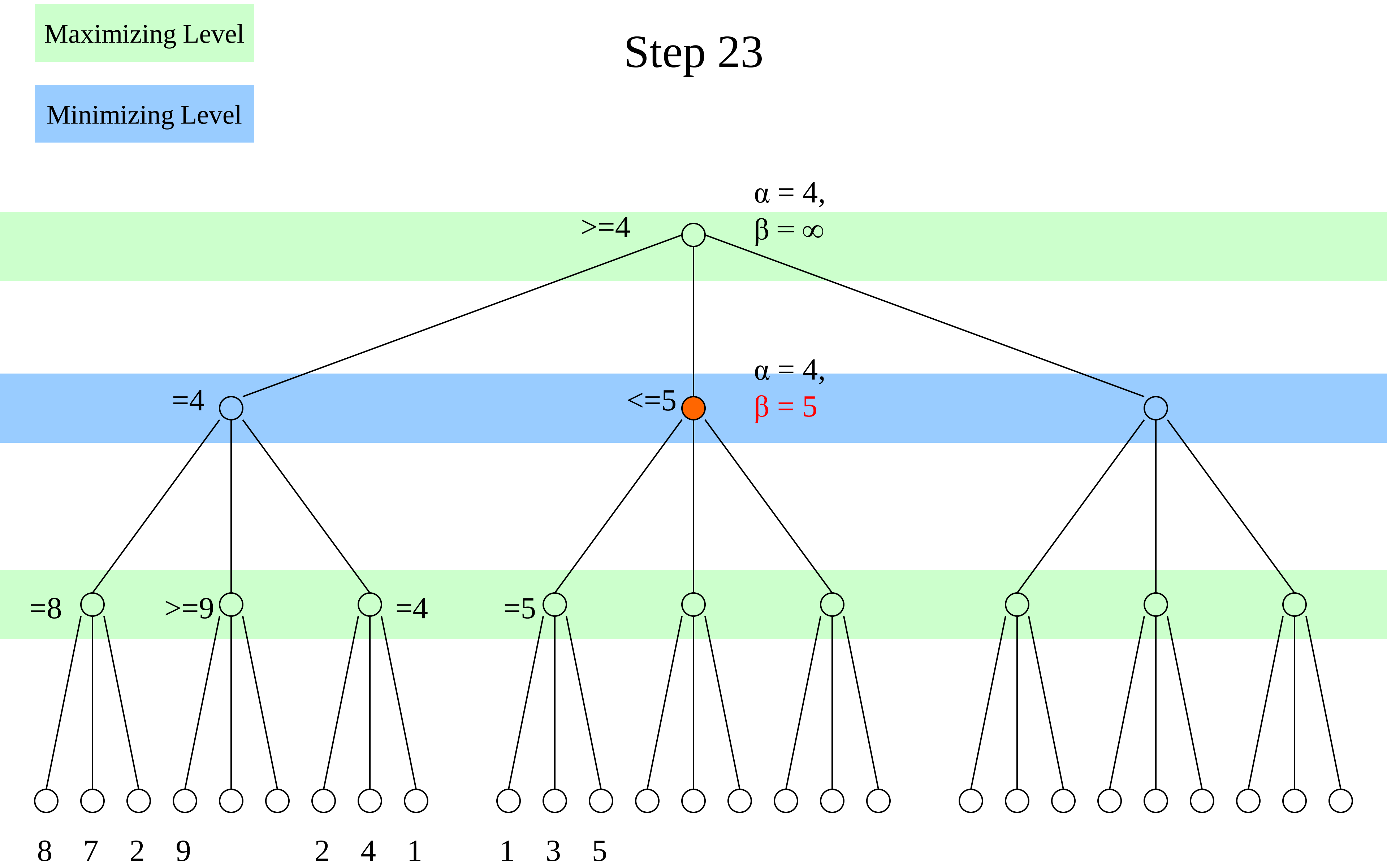
Maximizing Level

Minimizing Level

Step 22



Step 23

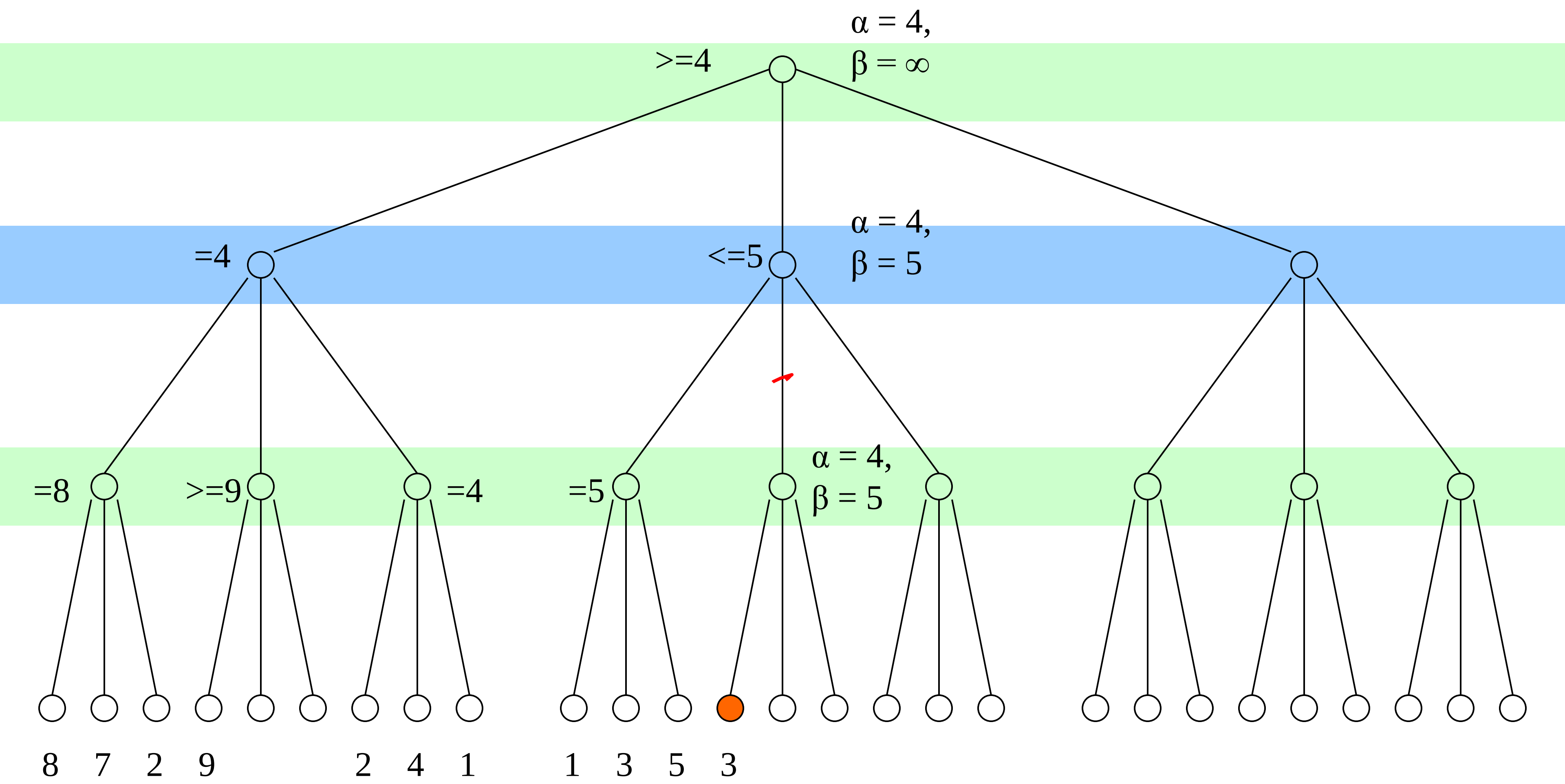


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
=4
<=5
α = 4,
β = 5
=8
>=9
=4
=5
8
7
2
9
2
4
1
1
3
5

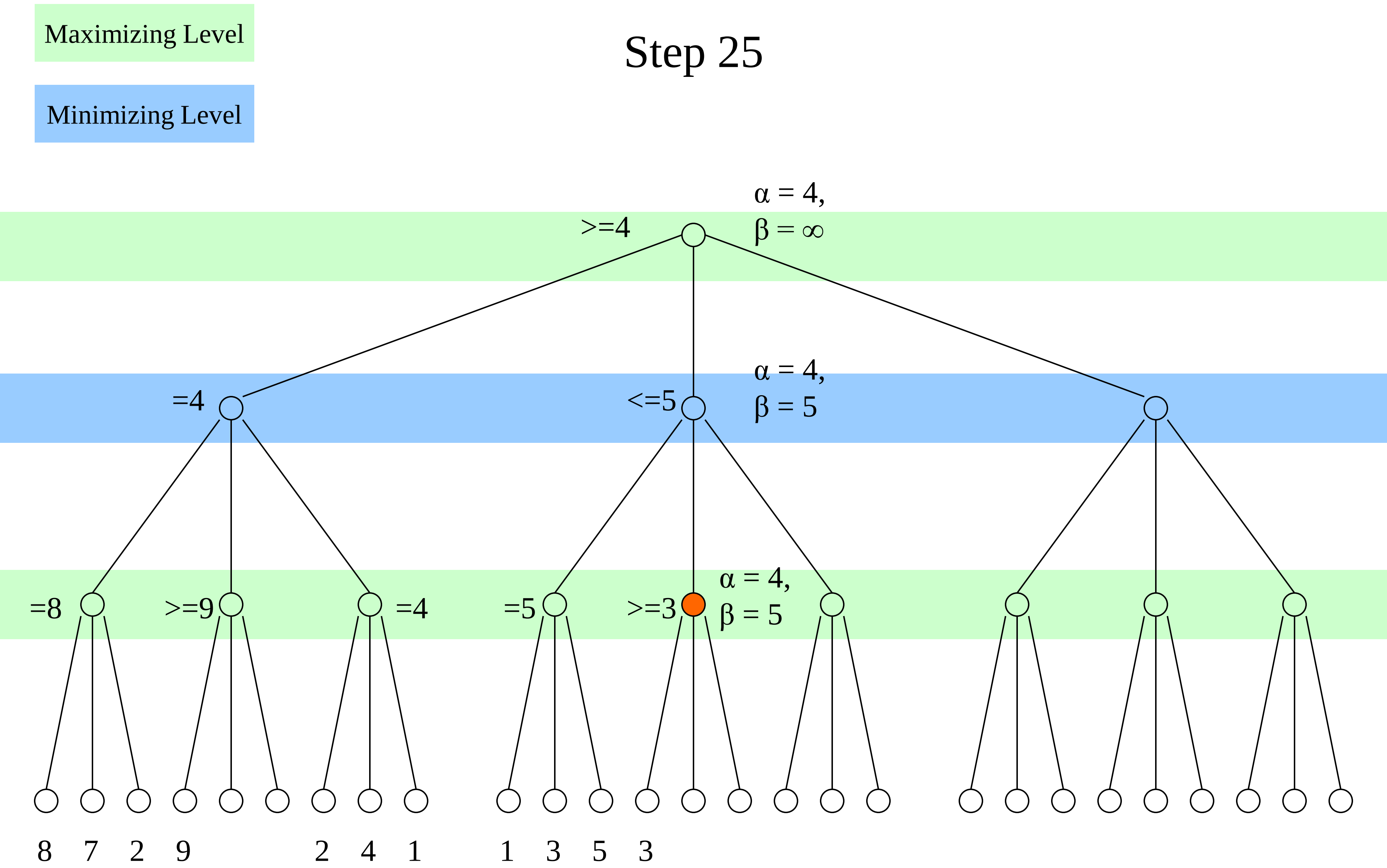
Maximizing Level

Minimizing Level

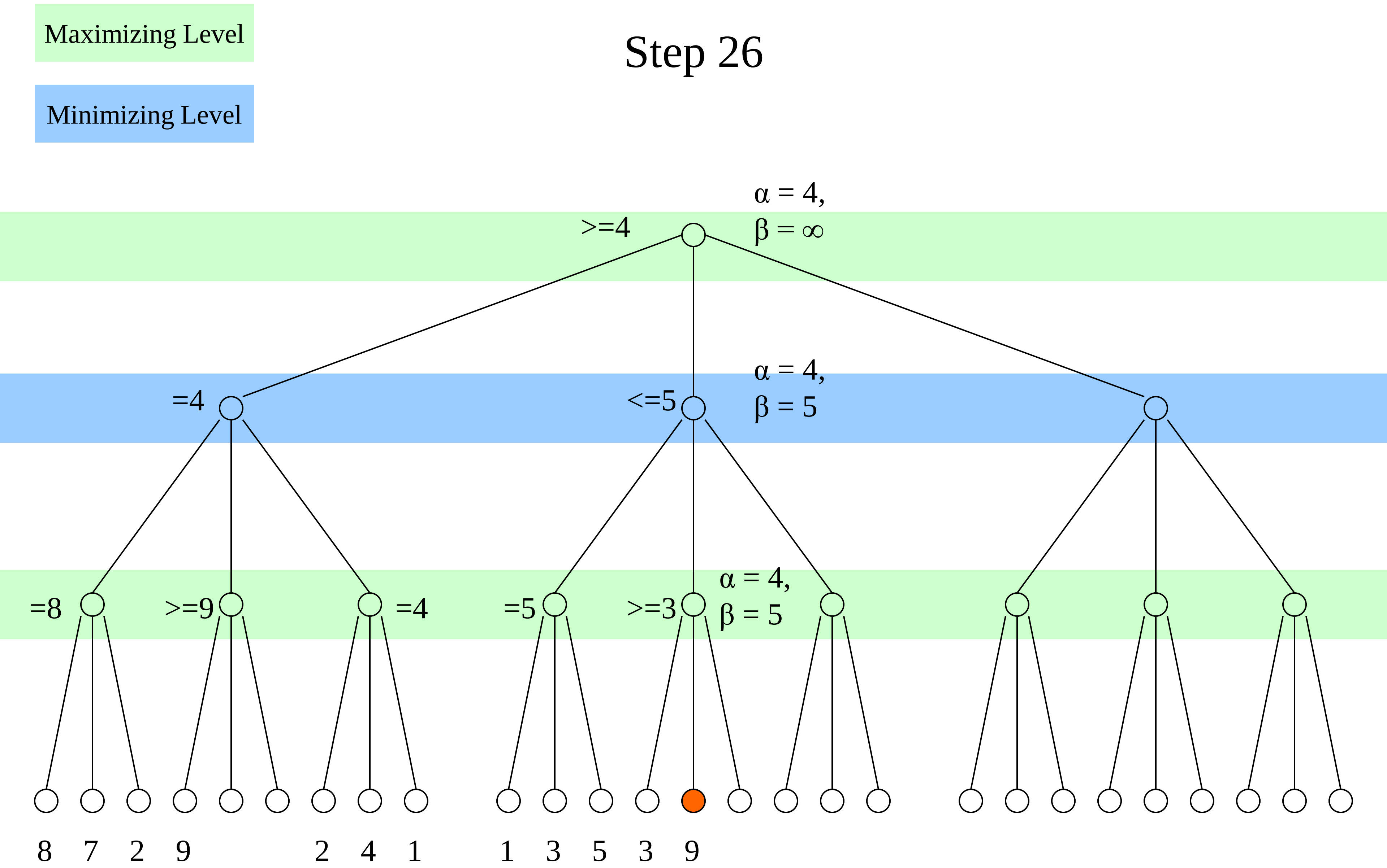
Step 24



Step 25


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = 5
=4
<=5
α = 4,
β = 5
=8
>=9
=4
=5
>=3
8
7
2
9
2
4
1
1
3
5
3

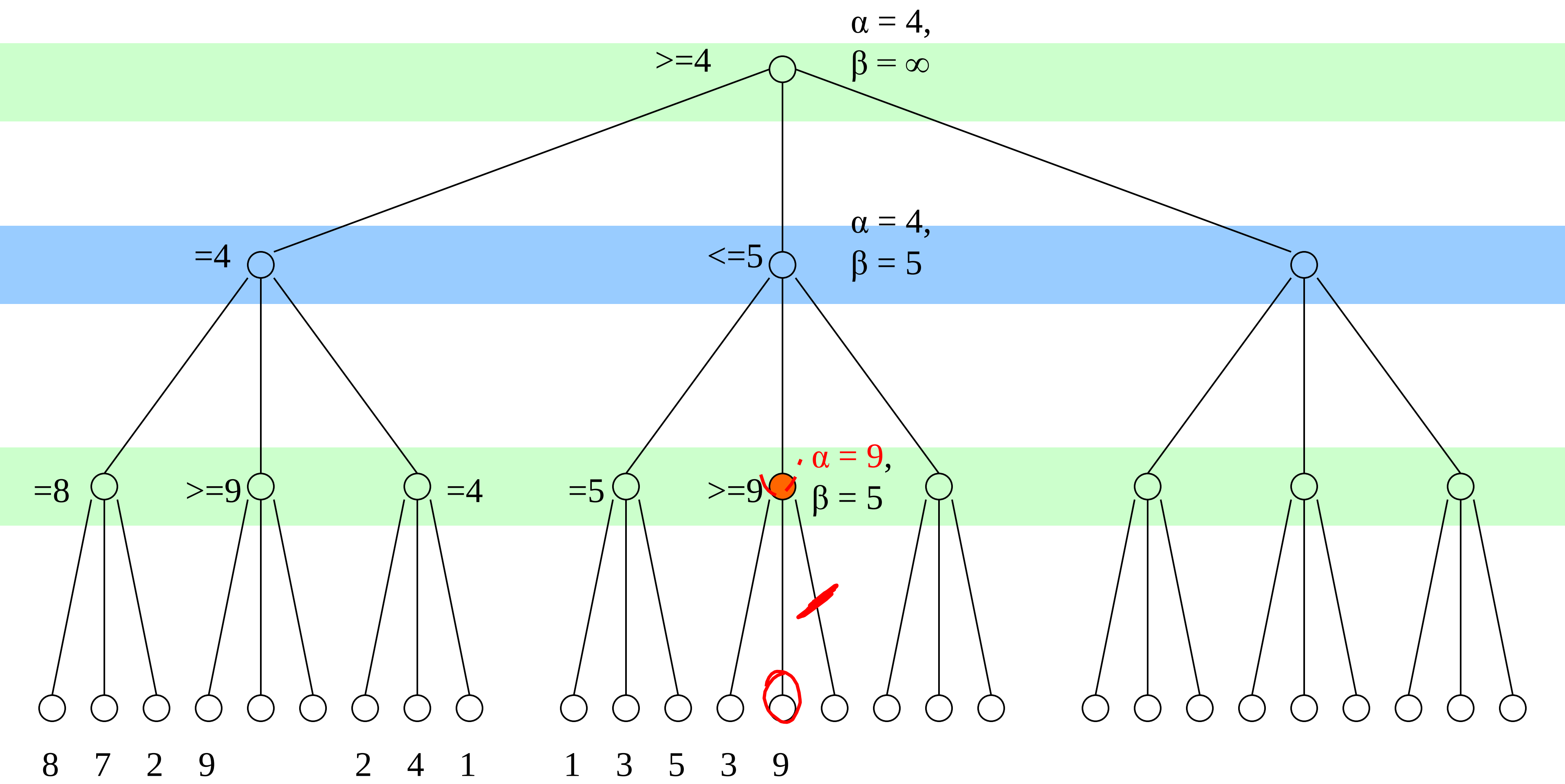
Step 26


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = 5
=4
<=5
α = 4,
β = 5
=8
>=9
=4
=5
>=3
8 7 2 9 2 4 1 1 3 5 3 9

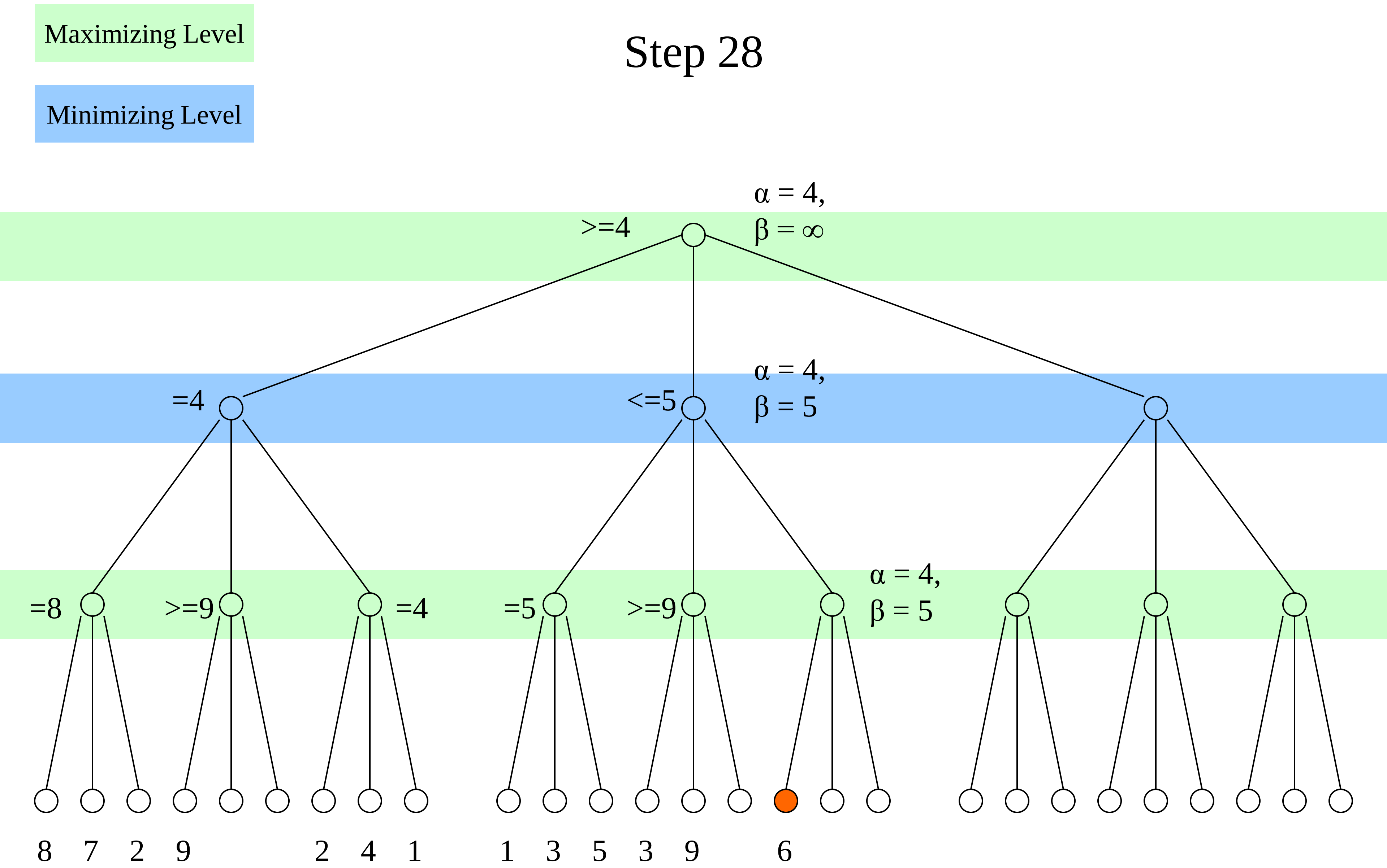
Maximizing Level

Minimizing Level

Step 27

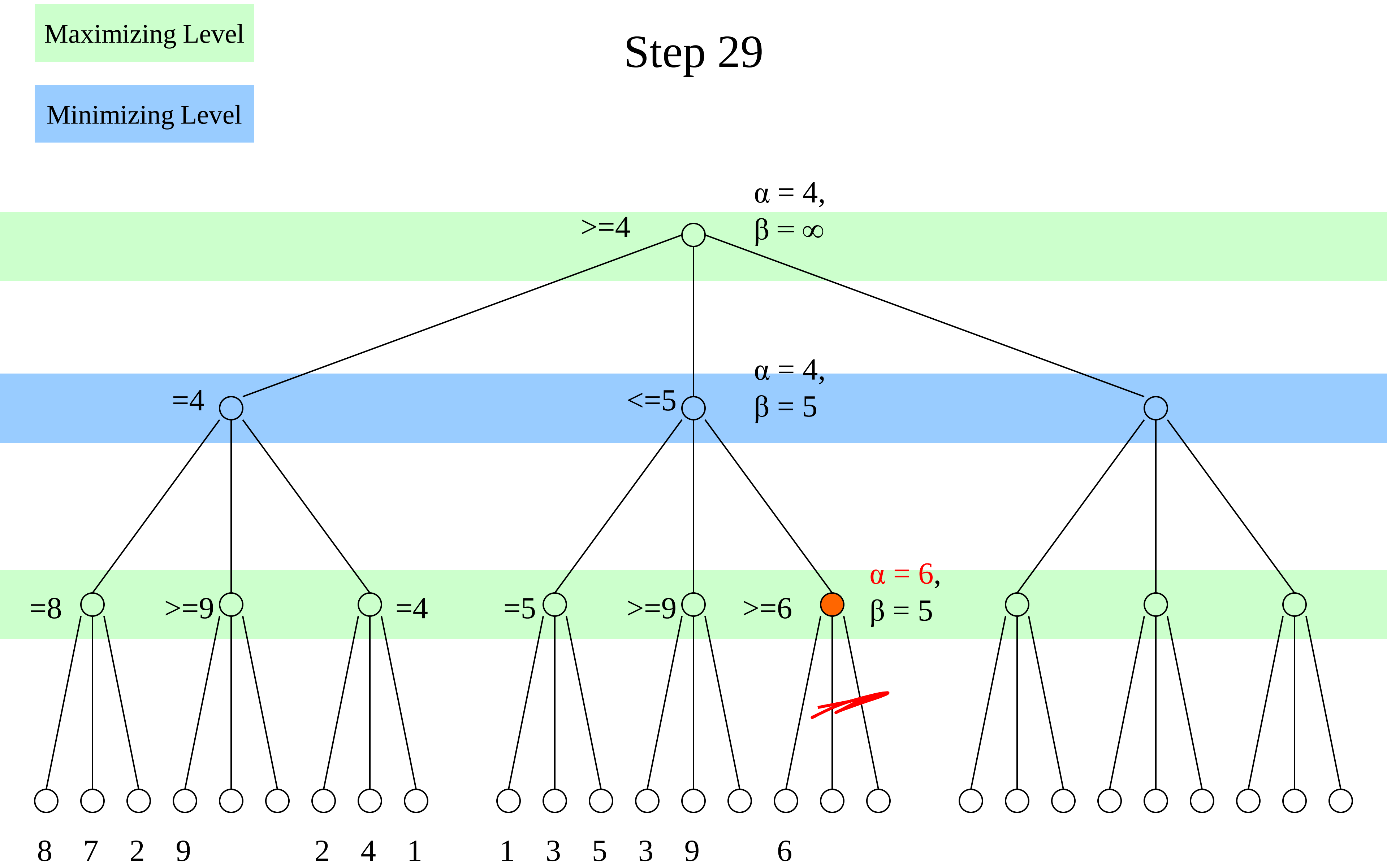


Step 28

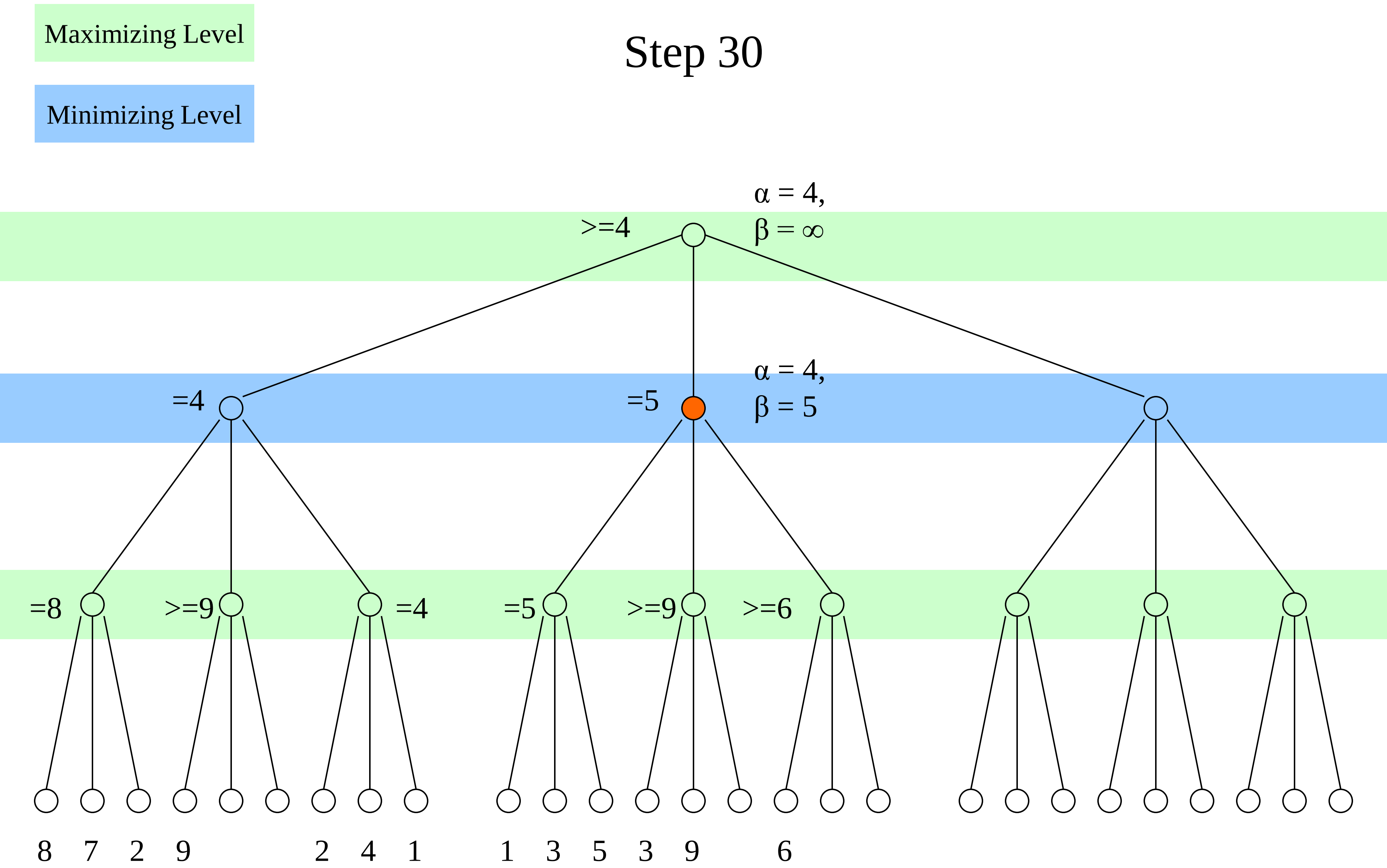


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = 5
=4
<=5
α = 4,
β = 5
=8
>=9
=4
=5
>=9
8
7
2
9
2
4
1
1
3
5
3
9
6

Step 29

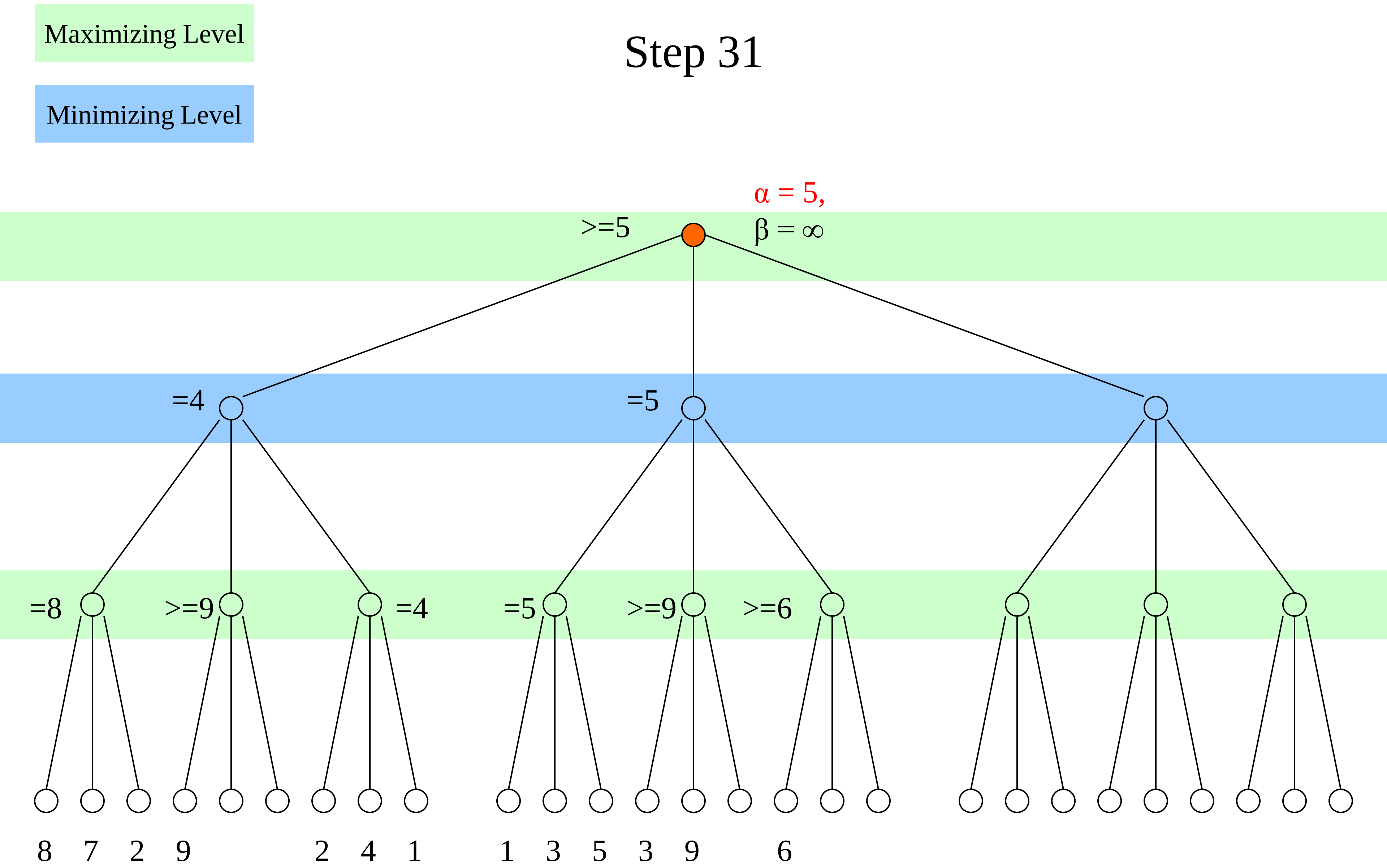

Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = 5
=4
<=5
α = 6,
β = 5
=8
>=9
=4
=5
>=9
>=6
8 7 2 9 2 4 1 1 3 5 3 9 6

Step 30

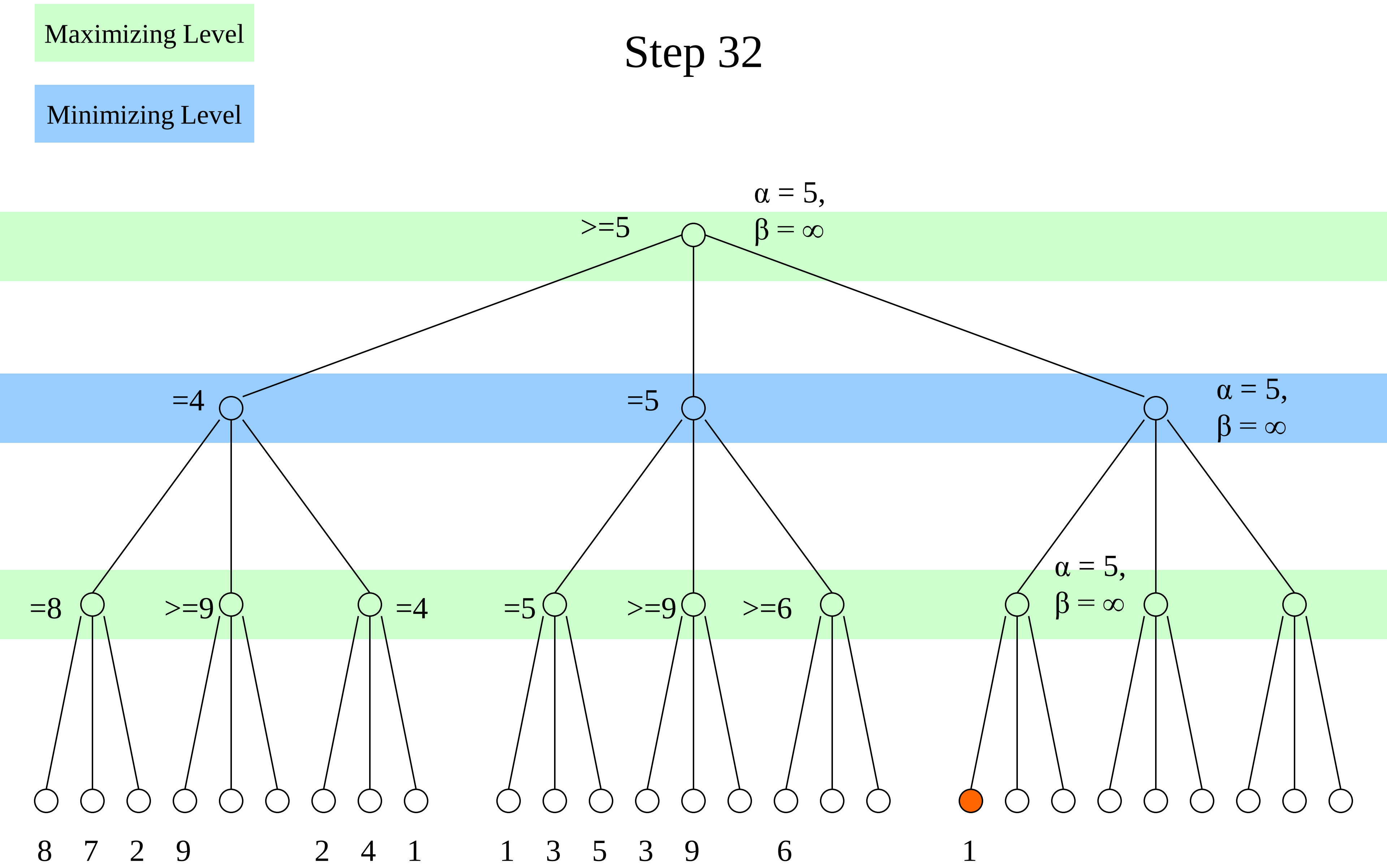


Maximizing Level
Minimizing Level
α = 4,
β = ∞
>=4
α = 4,
β = 5
=4
=5
=8
>=9
=4
=5
>=9
>=6
8 7 2 9 2 4 1 1 3 5 3 9 6

Step 31


Maximizing Level
Minimizing Level
α = 5,
β = ∞
>=5
=4
=5
=8
>=9
=4
=5
>=9
>=6
8 7 2 9 2 4 1 1 3 5 3 9 6

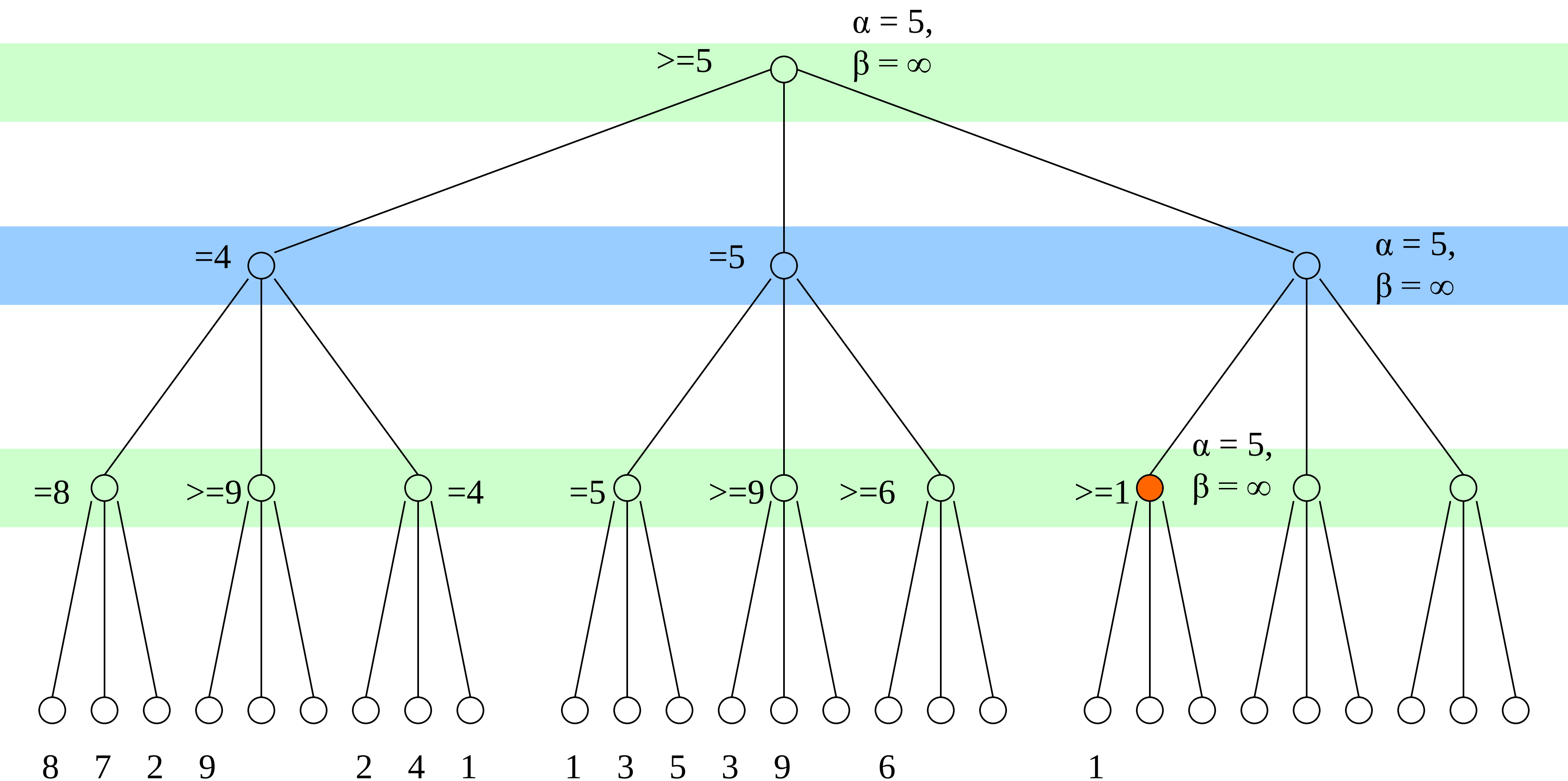
Step 32

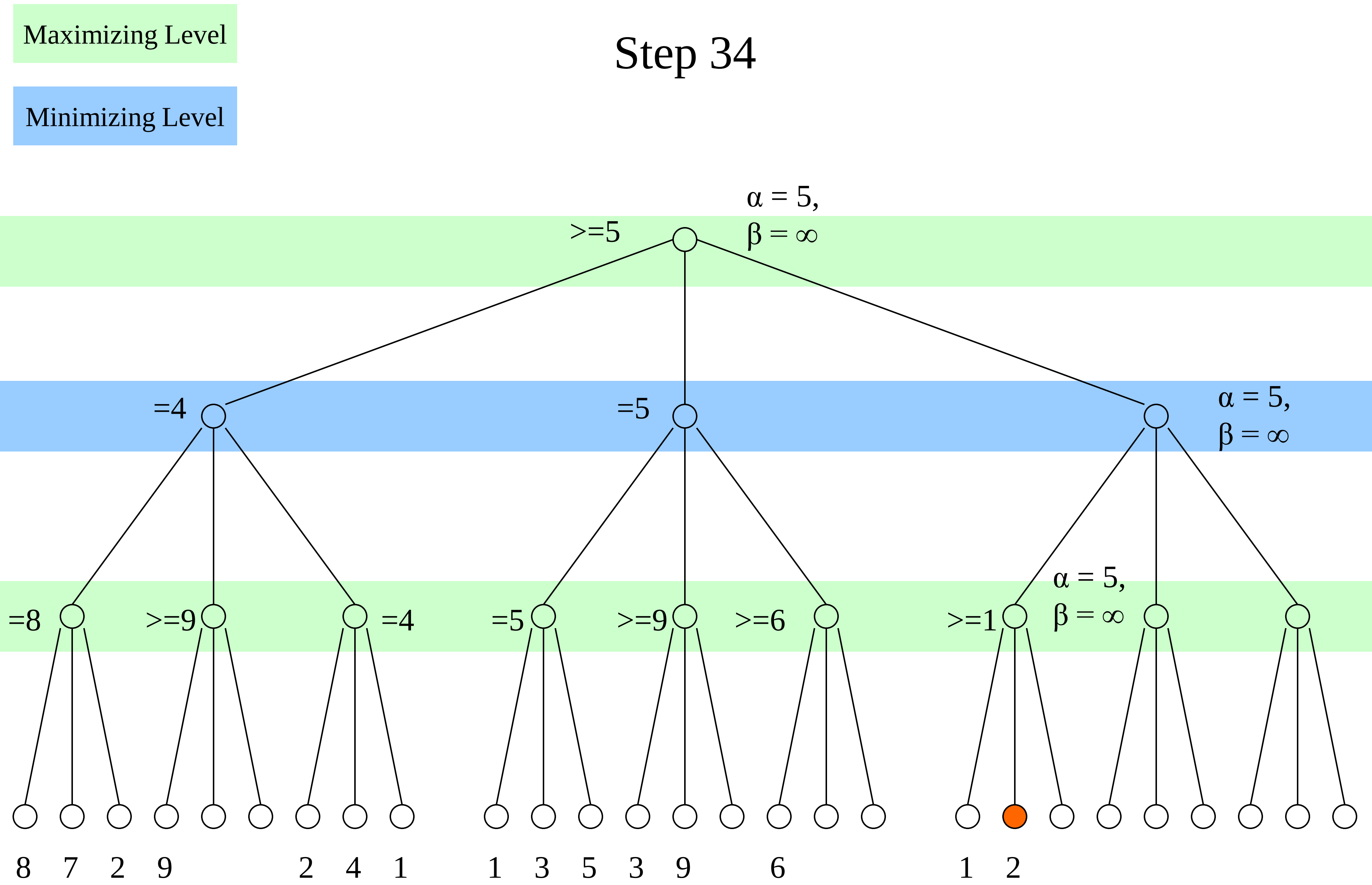

Maximizing Level
Minimizing Level
>=5
α = 5,
β = ∞
=4
=5
α = 5,
β = ∞
=8
>=9
=4
=5
>=9
>=6
α = 5,
β = ∞
8 7 2 9 2 4 1 1 3 5 3 9 6 1

Step 33

Maximizing Level

Minimizing Level

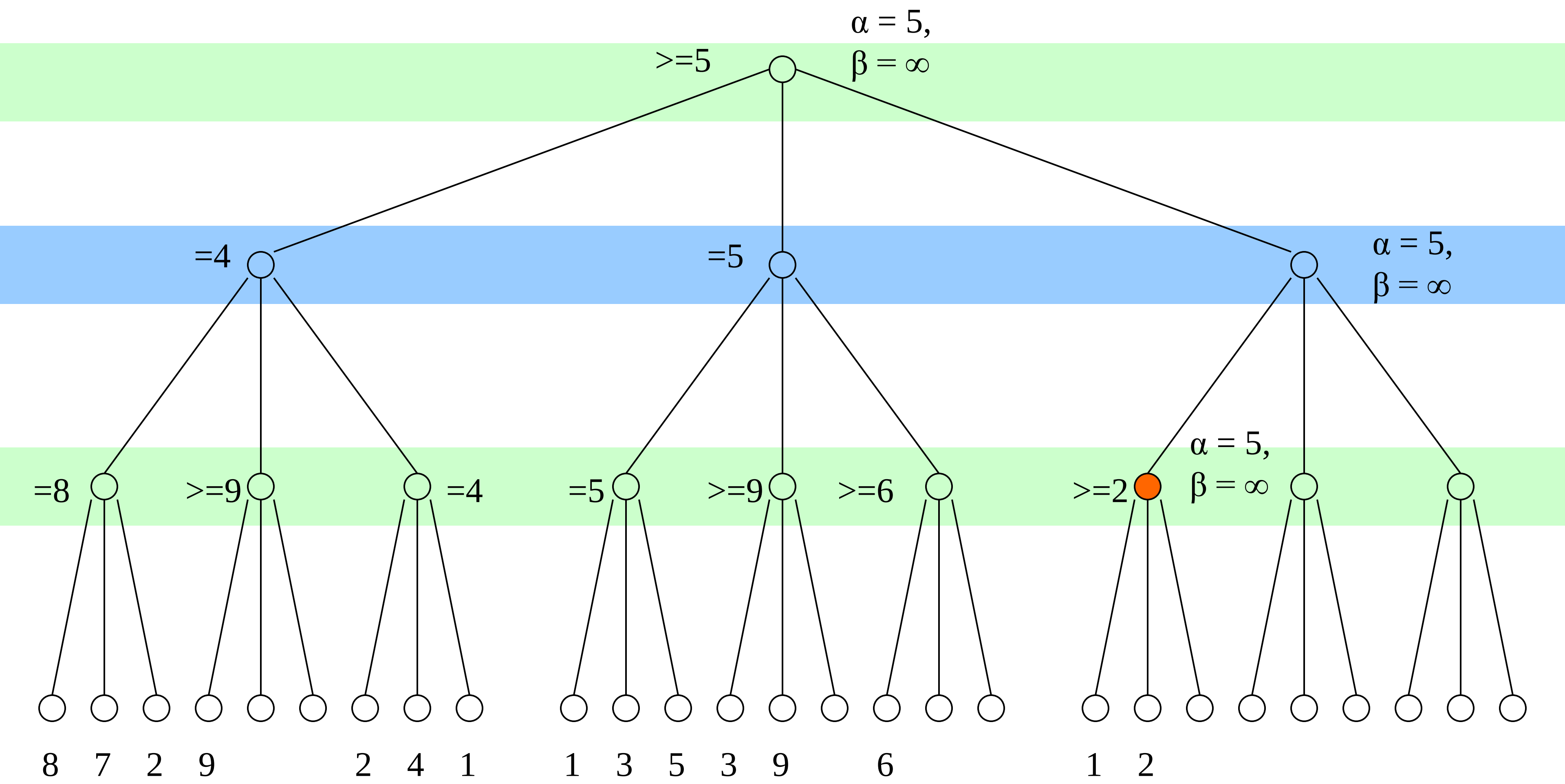




Maximizing Level
Minimizing Level
Step 34
α = 5,
β = ∞
>=5
=4
=5
α = 5,
β = ∞
=8
>=9
=4
=5
>=9
>=6
>=1
α = 5,
β = ∞
8 7 2 9 2 4 1 1 3 5 3 9 6 1 2

Maximizing Level
Minimizing Level

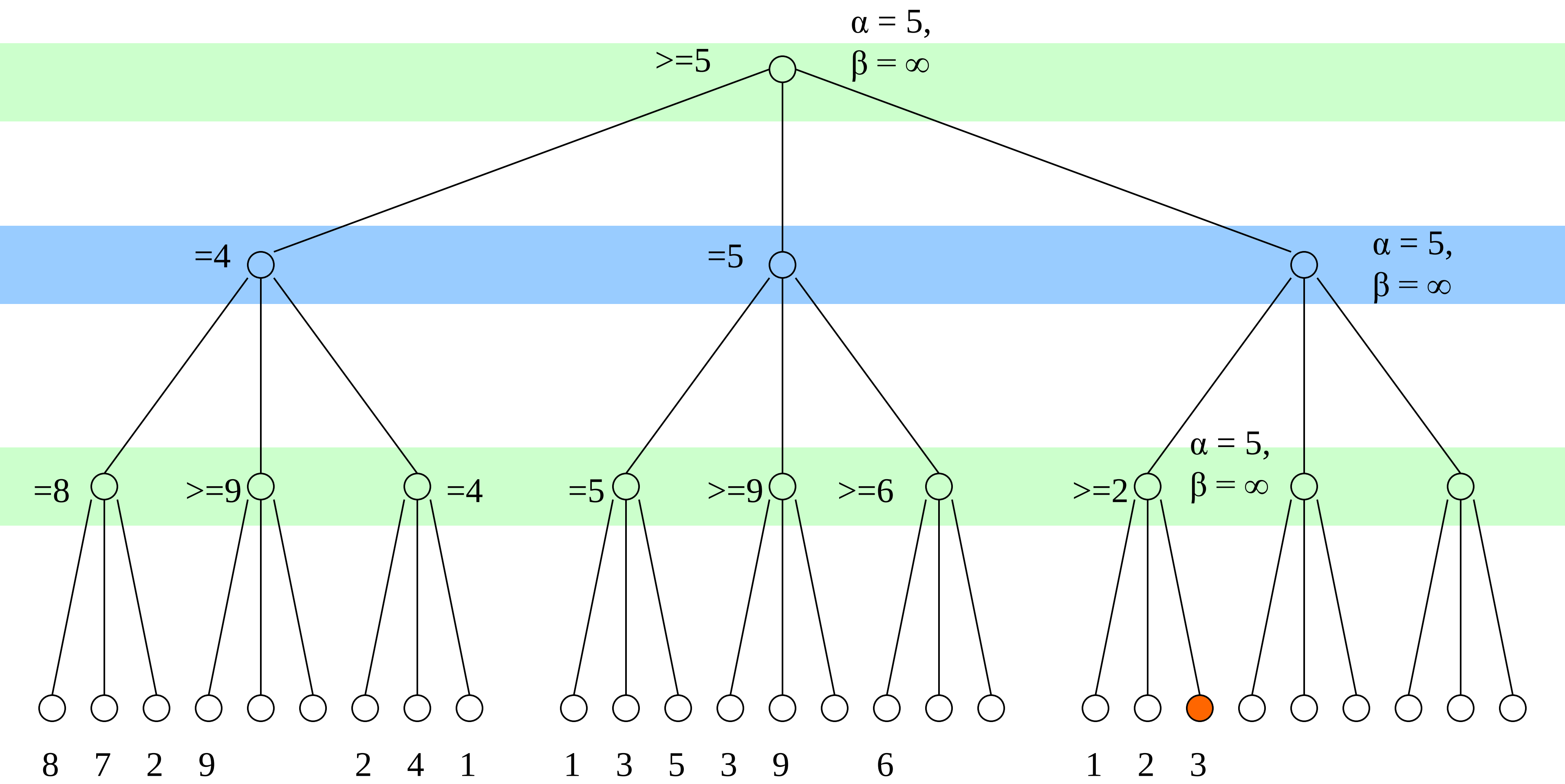
Step 35



α = 5,
β = ∞
>=5
=4
=5
α = 5,
β = ∞
=8
>=9
=4
=5
>=9
>=6
>=2
α = 5,
β = ∞
8 7 2 9 2 4 1 1 3 5 3 9 6 1 2

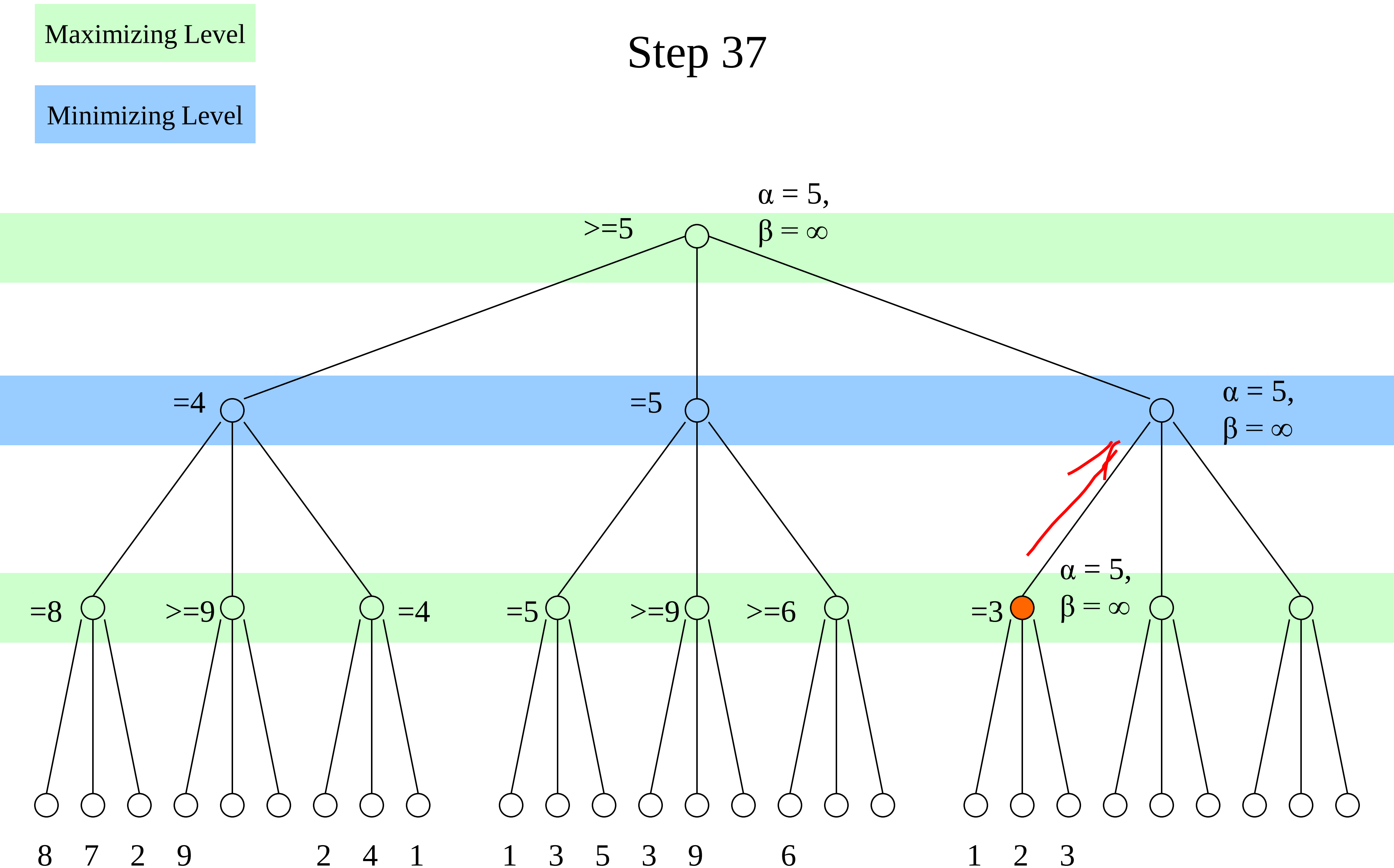
Maximizing Level
Minimizing Level

Step 36



>=5
α = 5,
β = ∞
=4
=5
α = 5,
β = ∞
=8
>=9
=4
=5
>=9
>=6
>=2
α = 5,
β = ∞
8 7 2 9 2 4 1 1 3 5 3 9 6 1 2 3

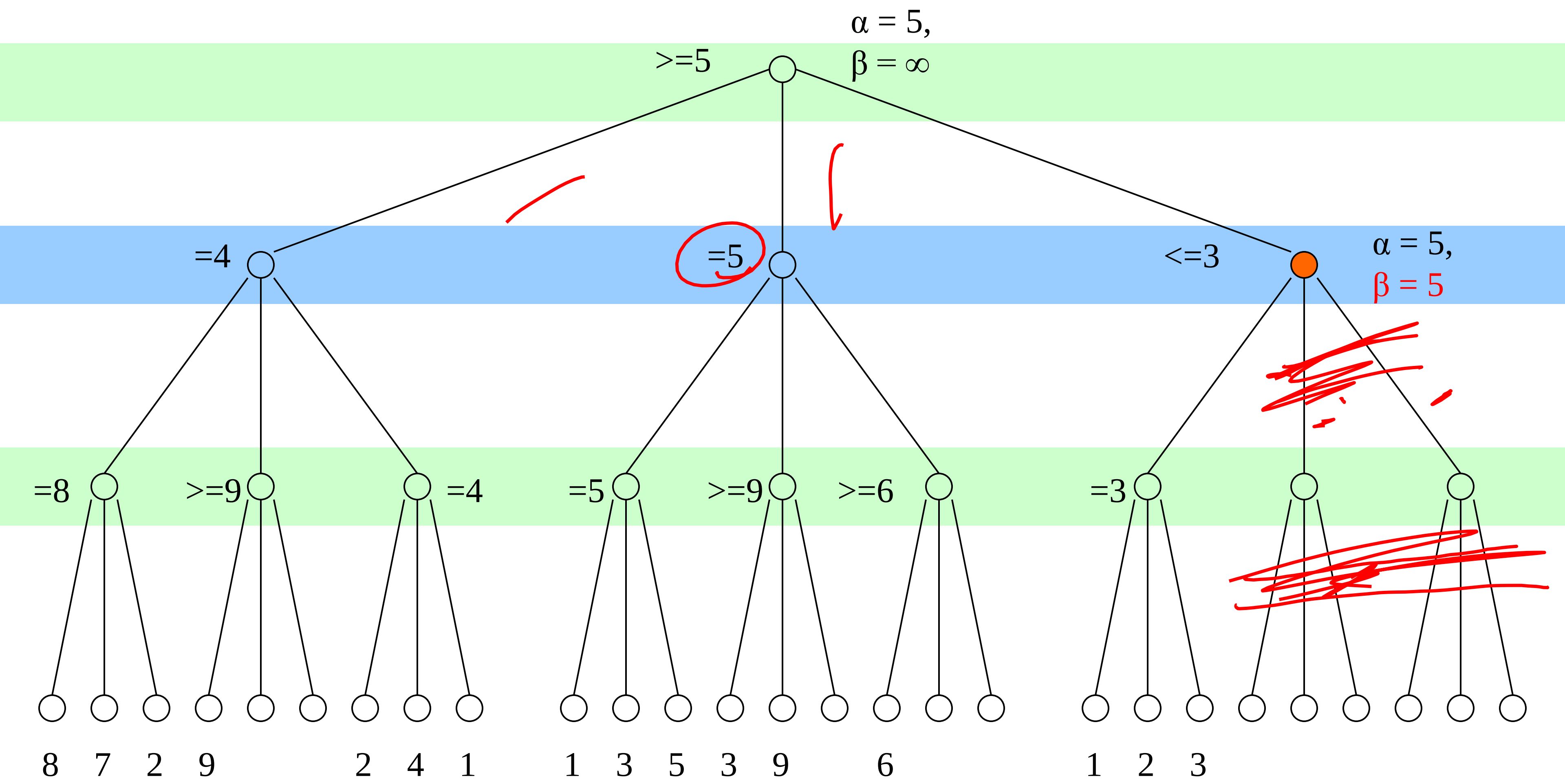
Step 37


Maximizing Level
Minimizing Level
α = 5,
β = ∞
>=5
=4
=5
α = 5,
β = ∞
α = 5,
β = ∞
=8
>=9
=4
=5
>=9
>=6
=3
8
7
2
9
2
4
1
1
3
5
3
9
6
1
2
3

Step 38

Maximizing Level

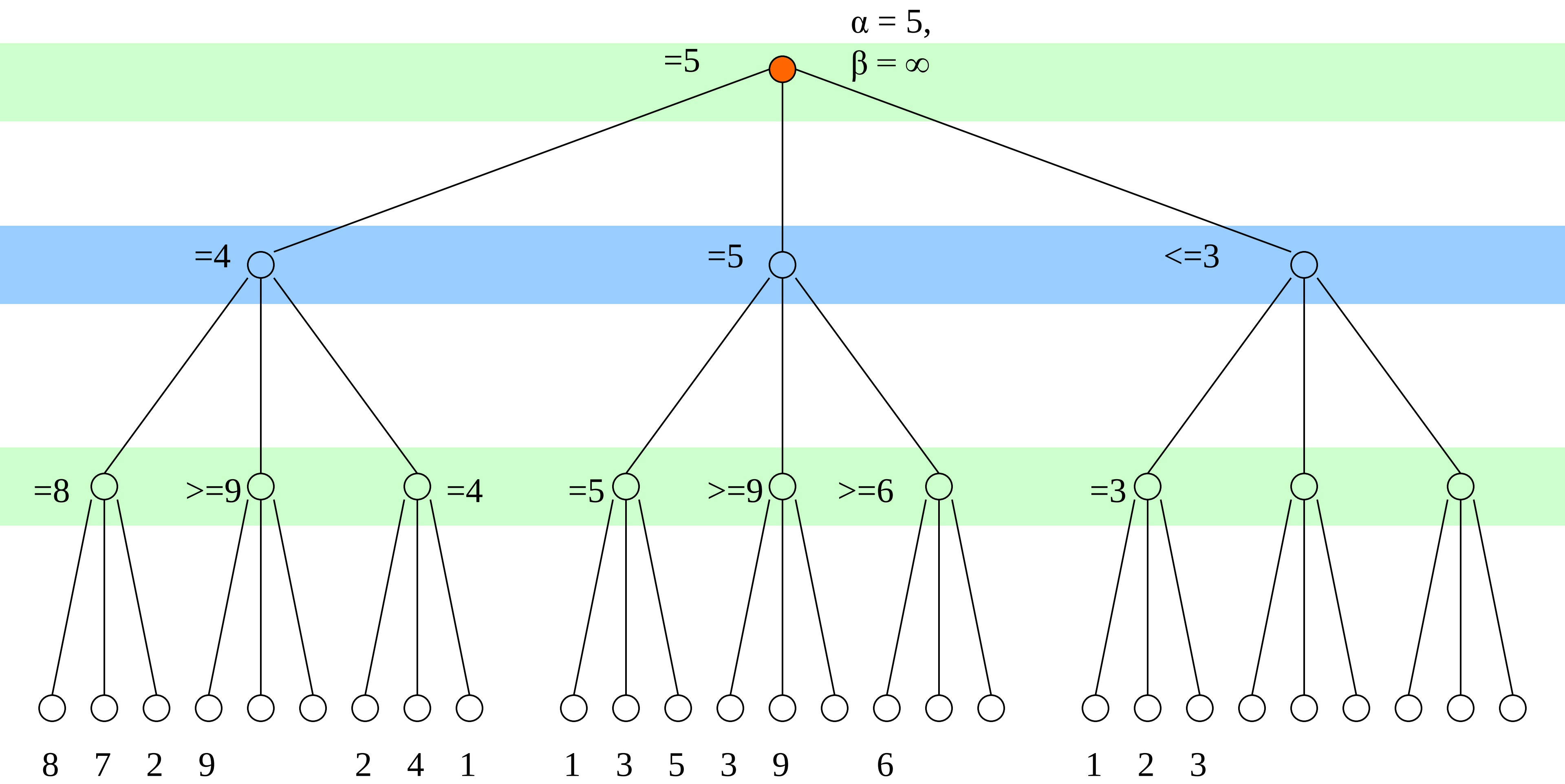
Minimizing Level



Maximizing Level

Minimizing Level

Step 39



void Alpha_Beta_Procedure(alpha, beta, &score)

1. If at the root, set alpha=$-\infty$, beta = ∞
2. If at the leaf, *score = static_evaluator(current_board, role); return
3. If at a minimizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

   (b) If *score < beta, beta = *score; // update upper bound

   *score = beta;
4. Else if at a maximizing level,

   until all children are examined or alpha >= beta,

   (a) Recursive call Alpha_Beta_Procedure on a child;

   (b) If *score > alpha, alpha = *score; // update lower bound

   *score = alpha;

Critique of α-β Procedure

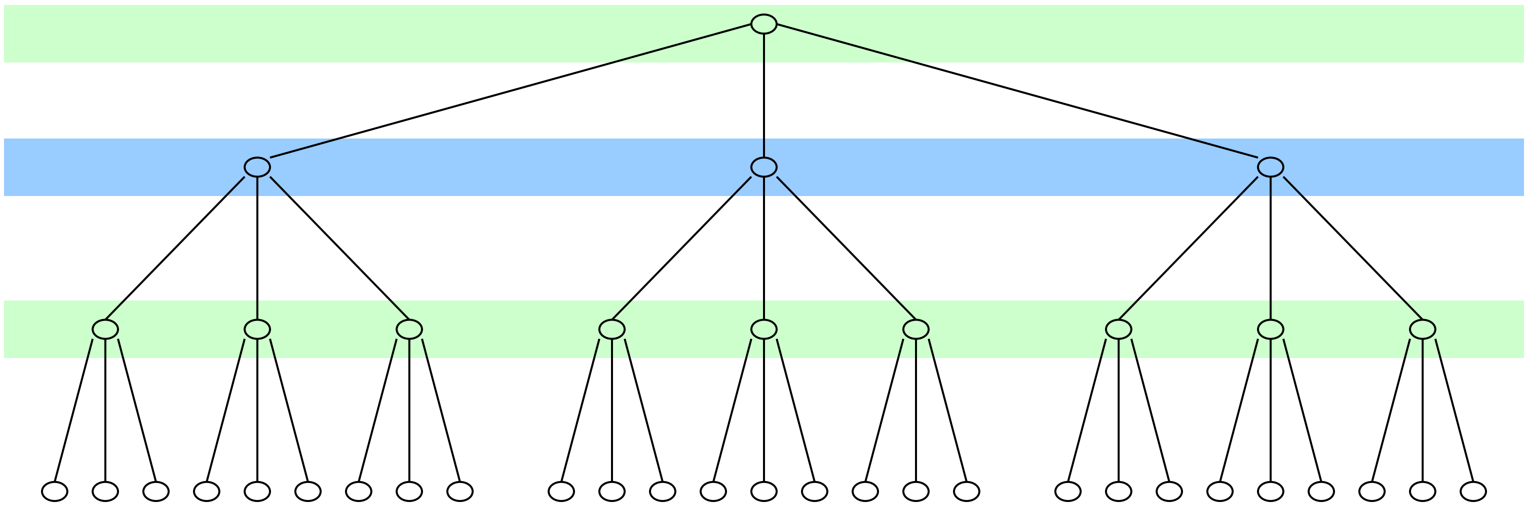
- Worst Case

  = Minimax = Exhaustive search to look ahead depth d with branching factor b

  # static evaluations $s = b^d$

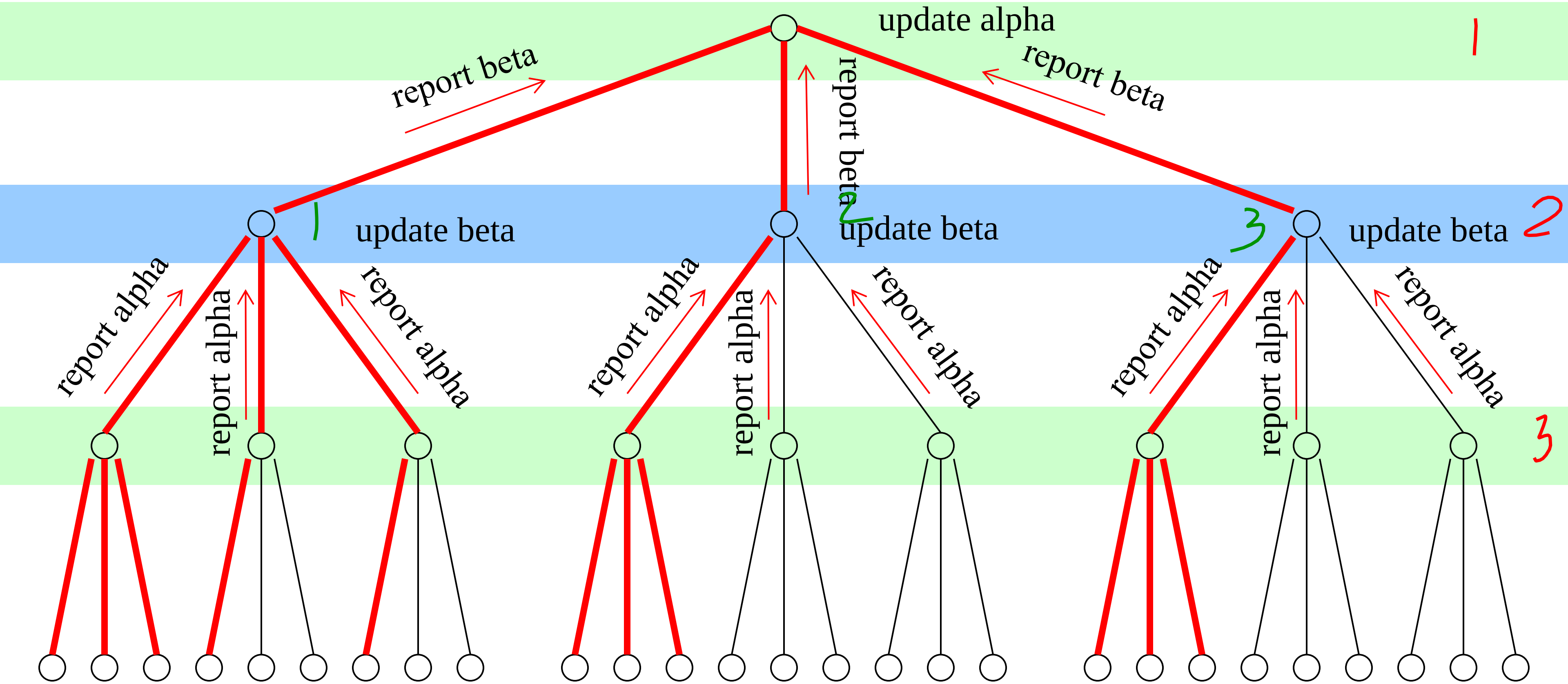
- Best Case

  (Assume the best move happens to be on the left most)



Maximizing Level: You have a chance to update alpha

Minimizing Level: You have a chance to update beta



Best case: 11 Worst case: 3^3 = 27

Critique of α-β Procedure

- Worst Case

$$s = b^d$$

- Best Case

$$s = \begin{cases} b^{\frac{d+1}{2}} + b^{\frac{d-1}{2}} - 1, if\ d\ \ is\ \ odd \\ 2b^{\frac{d}{2}}, if\ d\ \ is\ \ even \end{cases}$$

Proof by Induction

Best case analysis provides lower bound on # evaluations in real game

- In real game

$$2b^{\frac{d}{2}} \le s \le b^d$$

$b = 3$
$d = 3$

Critique of α-β Procedure

- Worst Case

$$s = b^d$$

$3^3 = 27$

- Best Case

$$s = \begin{cases} b^{\frac{d+1}{2}} + b^{\frac{d-1}{2}} - 1, if\ d\ \ is\ \ odd \\ 2b^{\frac{d}{2}}, if\ d\ \ is\ \ even \end{cases}$$

$3^{(3+1)/2}$
$+ 3^{(3-1)/2}$
$- 1$
$= 9+3-1$
$=\underline{11}$

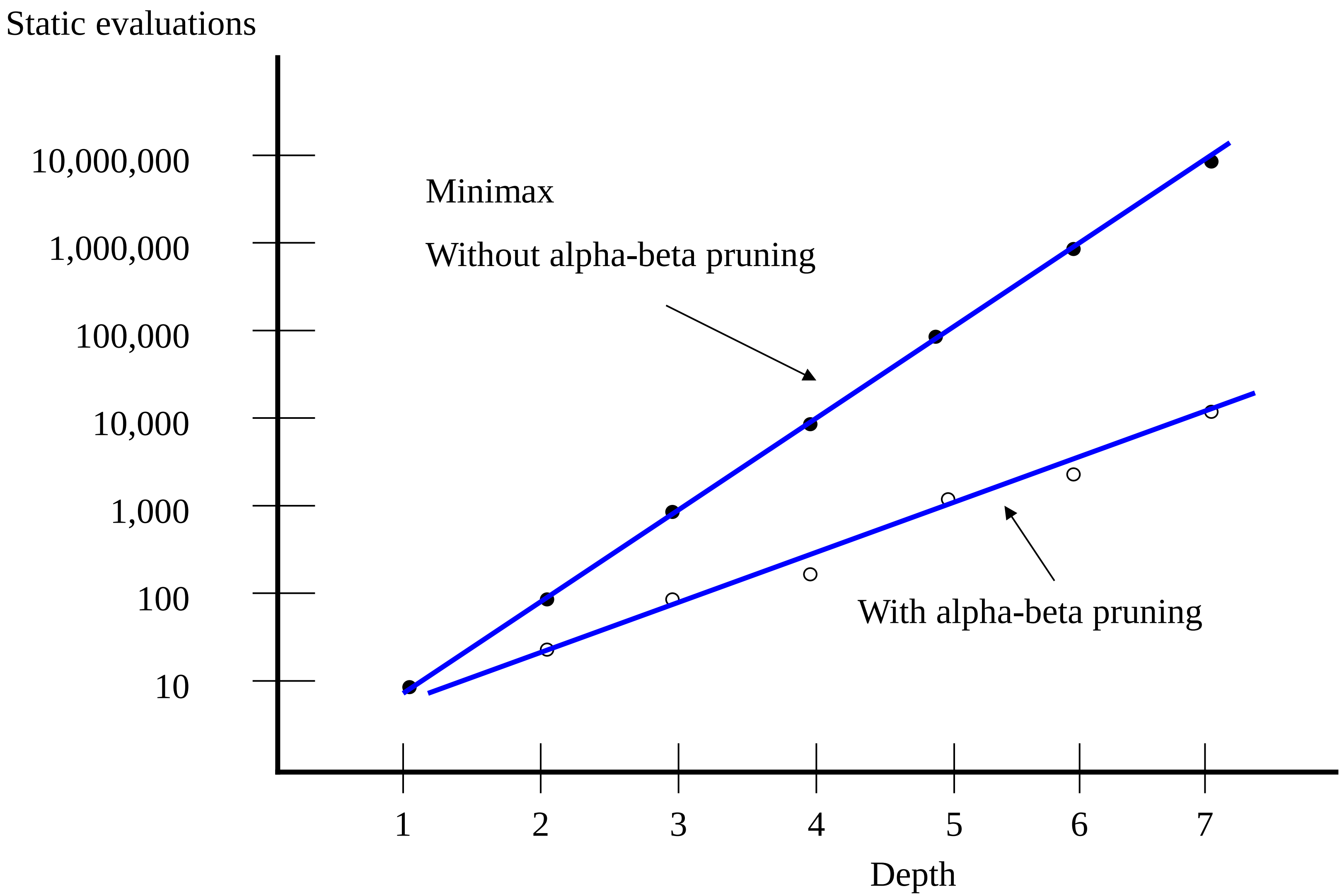
Proof by Induction

Best case analysis provides lower bound on # evaluations in real game

- In real game

$$2b^{\frac{d}{2}} \le s \le b^d$$

$2 \times 3^{3/2}=11$
$<=S <=$
$3^3 =27$



The ALPHA-BETA procedure reduces the rate of explosive growth but does not prevent it.
The branching factor here is assumed to be 10.

Heuristic Methods

- The goal is to reduce $s = b^d$

  b → prune children

  d → reduce lookahead depth

- Minimax/Alpha-Beta

  1) compute static value at leaves of game tree

  2) pass up value to parents and make decision at root

  3) prune branches (i.e. when alpha >= beta)
- Whether or not Alpha-Beta can prune depends on game situation
- If time limits on moves (e.g. chess), a conservative *d* may waste time

Heuristic Methods

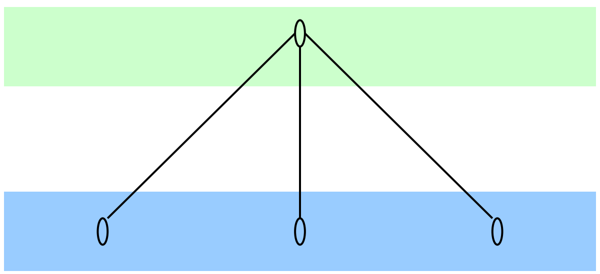
- Progressive Deepening
- Continuation Heuristic

  Search-until-quiescent Heuristic

  Singular-extension Heuristic
- Tapered Search

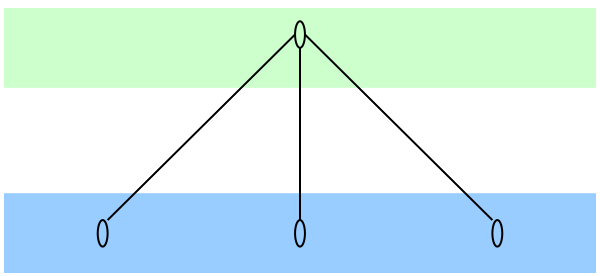
Progressive Deepening

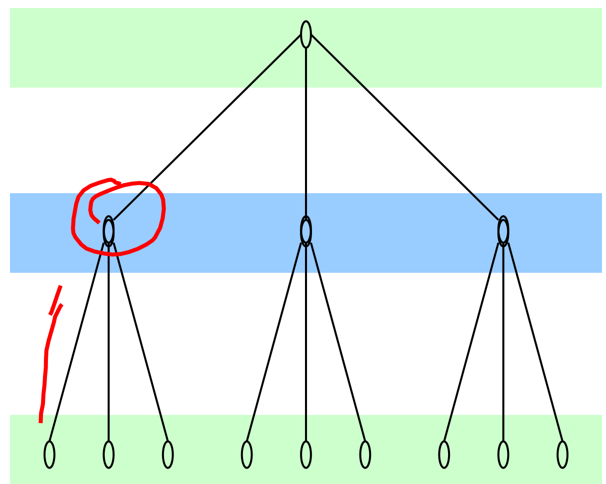
- Analyze game situation to *d=1,d=2,d=3,*…until time is up
- Choice is determined by the analysis at one level less deep than the one in progress when time runs out



Progressive Deepening

- Analyze game situation to *d=1,d=2,d=3,*…until time is up
- Choice is determined by the analysis at one level less deep than the one in progress when time runs out

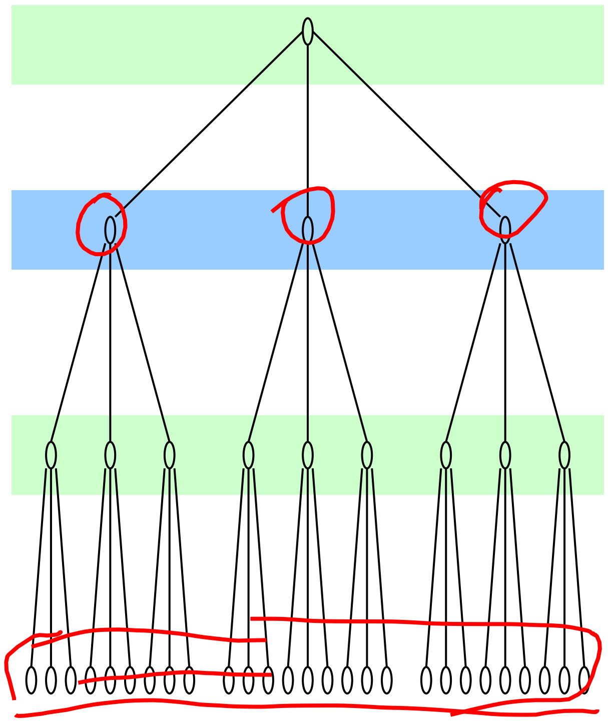




Progressive Deepening

- Analyze game situation to $d=1, d=2, d=3, \ldots$ until time is up
- Choice is determined by the analysis at one level less deep than the one in progress when time runs out
- Lots of extra work ? Not really

Progressive Deepening

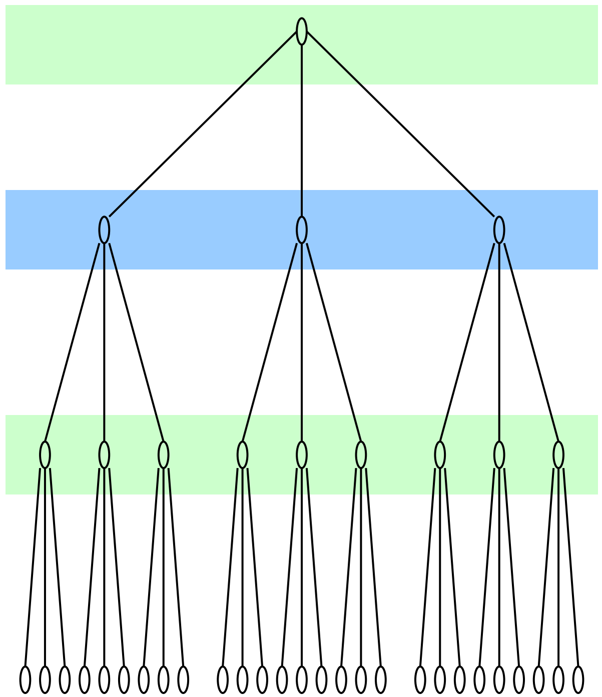


Non-leaf nodes:

$$b^0 + b^1 + b^2 + \ldots + b^{d-1} = \frac{b^d - 1}{b-1}$$

leaf nodes: b^d

Progressive Deepening



Non-leaf nodes:

$$b^0 + b^1 + b^2 + \ldots + b^{d-1} = \frac{b^d - 1}{b - 1}$$

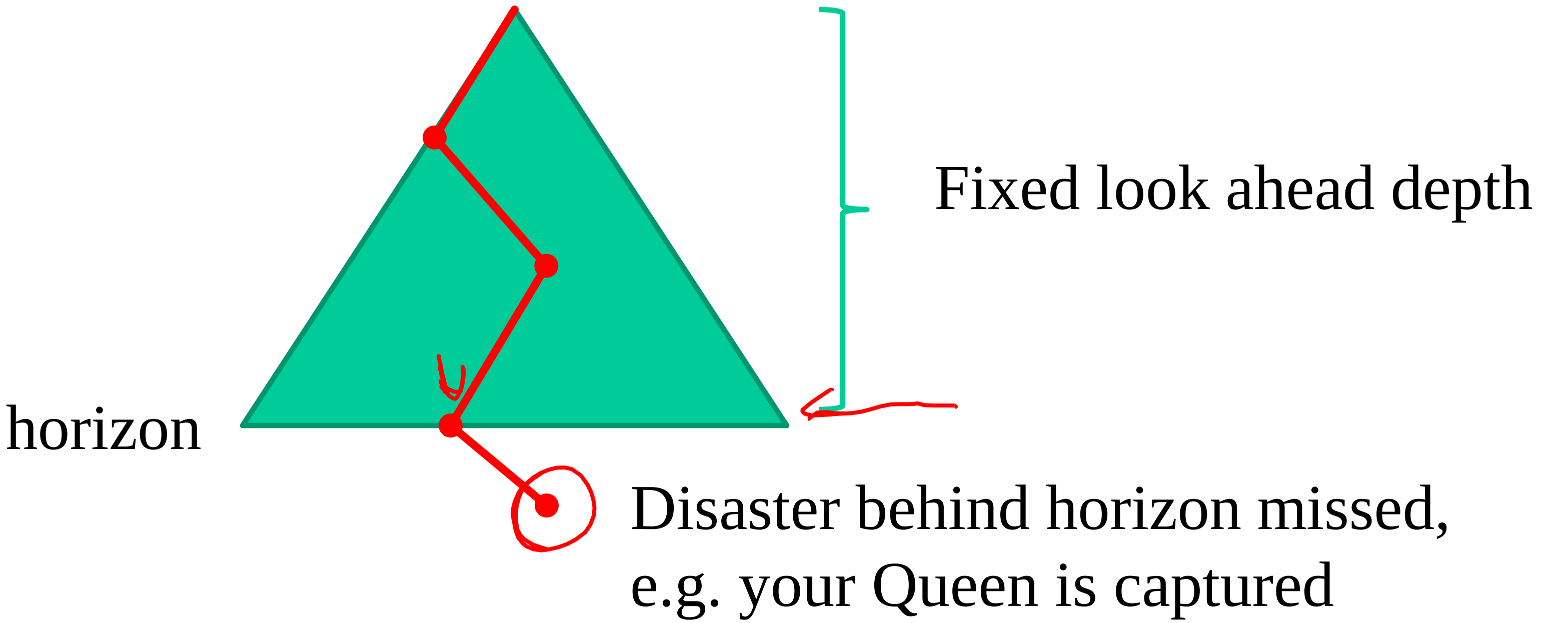
leaf nodes: b^d

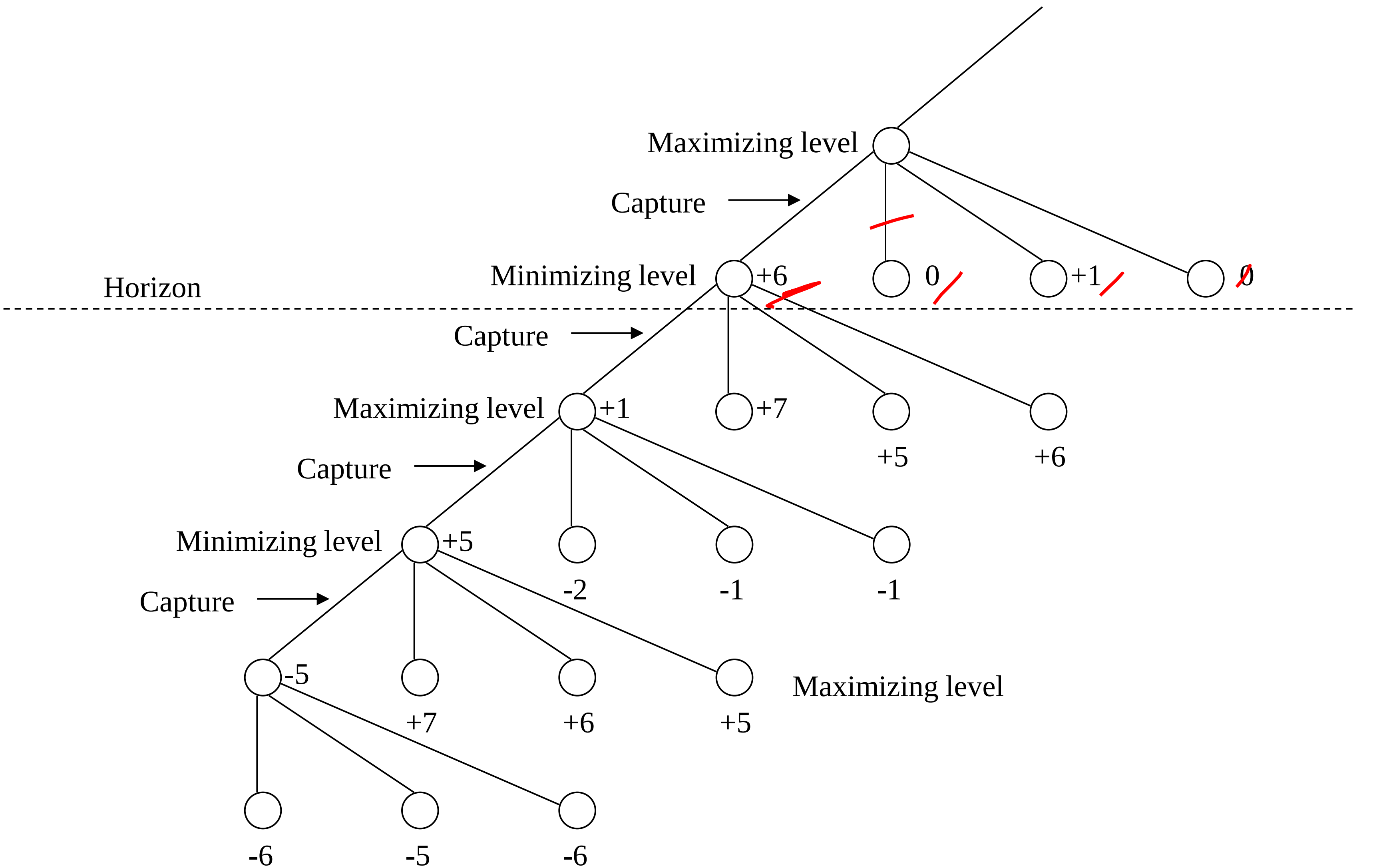
$$\frac{\#leaves}{\#non-leaves} = \frac{b^d}{\frac{b^d - 1}{b - 1}} \approx b - 1$$

=>
Overhead of progressive deepening is only a factor of b-1

Continuation Heuristics handle Horizon Effect

Horizon Effect:





Maximizing level
Capture
Minimizing level
+6
0
+1
0
Horizon
Capture
Maximizing level
+1
+7
+5
+6
Capture
Minimizing level
+5
-2
-1
-1
Capture
-5
+7
+6
+5
Maximizing level
-6
-5
-6

Continuation Heuristic

- Search-until-quiescent Heuristic

  Stop search only if a capture (you or your opponent captures a piece) is not imminent.

- Singular-extension Heuristic

  Search as long as one move's static value is much better than the rest. This move would "force" decision, but could be wrong!

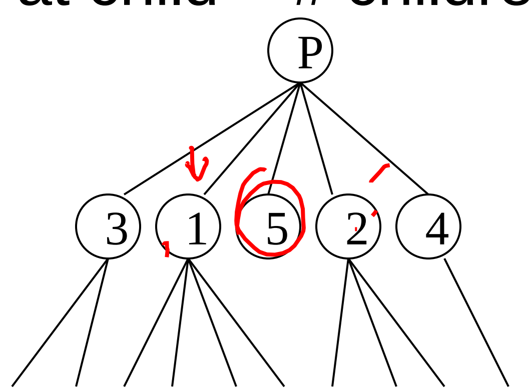
Heuristic Pruning

- Principle: prune apparently “bad” moves and spend time on more promising moves
- Tapered Search

1) Rank each child by using a fast static evaluator

2) Compute # of branches to explore at each child based on its rank,

i.e. # branches at child = # children – rank(child)



(5 - 1)

(5 - 2)

Summary of Learning Goals

You should know about:

- Chess
- Some of the history of AI tackling chess
- Static Evaluator
- Minimax Procedure
- Alpha-beta Procedure → prune branches
- Progressive Deepening → reduce depth
- Continuation Heuristic → look deeper
  - Search-until-quiescent Heuristic
  - Singular-extension Heuristic
- Tapered Search → prune branches