Section 8: Minimax & Alpha Beta Pruning

University of Washington

November 15, 2017

Game algorithm: Minimax

Minimax

- suppose our opponent makes the best move every time!
- so we want to minimize the possible max gain our opponent will get by maximizing our own gain
- So called "minimax".

Game algorithm: Minimax

Minimax

- suppose our opponent makes the best move every time!
- so we want to minimize the possible max gain our opponent will get by maximizing our own gain
- So called "minimax".

Values of the game

In a chess game:

- My gain is my opponent's loss (and vice versa)
- If the position value is 50 for me, it should be -50 for my opponent.
- If I reach $+\infty$, I win; if my opponent reaches $-\infty$, he/she wins.
- So I want **MAX**, my opponent wants **MIN**
- A zero-sum game (Google it if you are interested).

Introducing the players!

For the following slides, assume:

- It's blue's turn!
- **MIN** wants to minimize the value
- **MAX** wants to maximize the value

Minimax, Code


```
int minimax(Position p, boolean is_max) {
  if (p is a leaf) {
    // always position value of MAX
    return p.evaluate();
  }
  if (is_max) { // MAX
    int bestValue = −∞
    for (move in p.getMoves()) {
      p.applyMove();
      int value = minimax(p, is_max);
      p.undoMove();
      if (value > bestValue) {
        bestValue = value;
      }
    }
  } else { // MIN
    int bestValue = ∞
    for (move in p.getMoves()) {
      p.applyMove();
      int value = minimax(p, is_max);
      p.undoMove();
      if (value < bestValue) {
        bestValue = value;
      }
    }
  }
}
```

The highlighted parts are the only differences!

How do we simplify Minimax?

A fact

$$\max(a, b) = -\min(-a, -b)$$

By maximizing the **negation** of the values, we found the **negation** of the min value.

Change Minimax Code

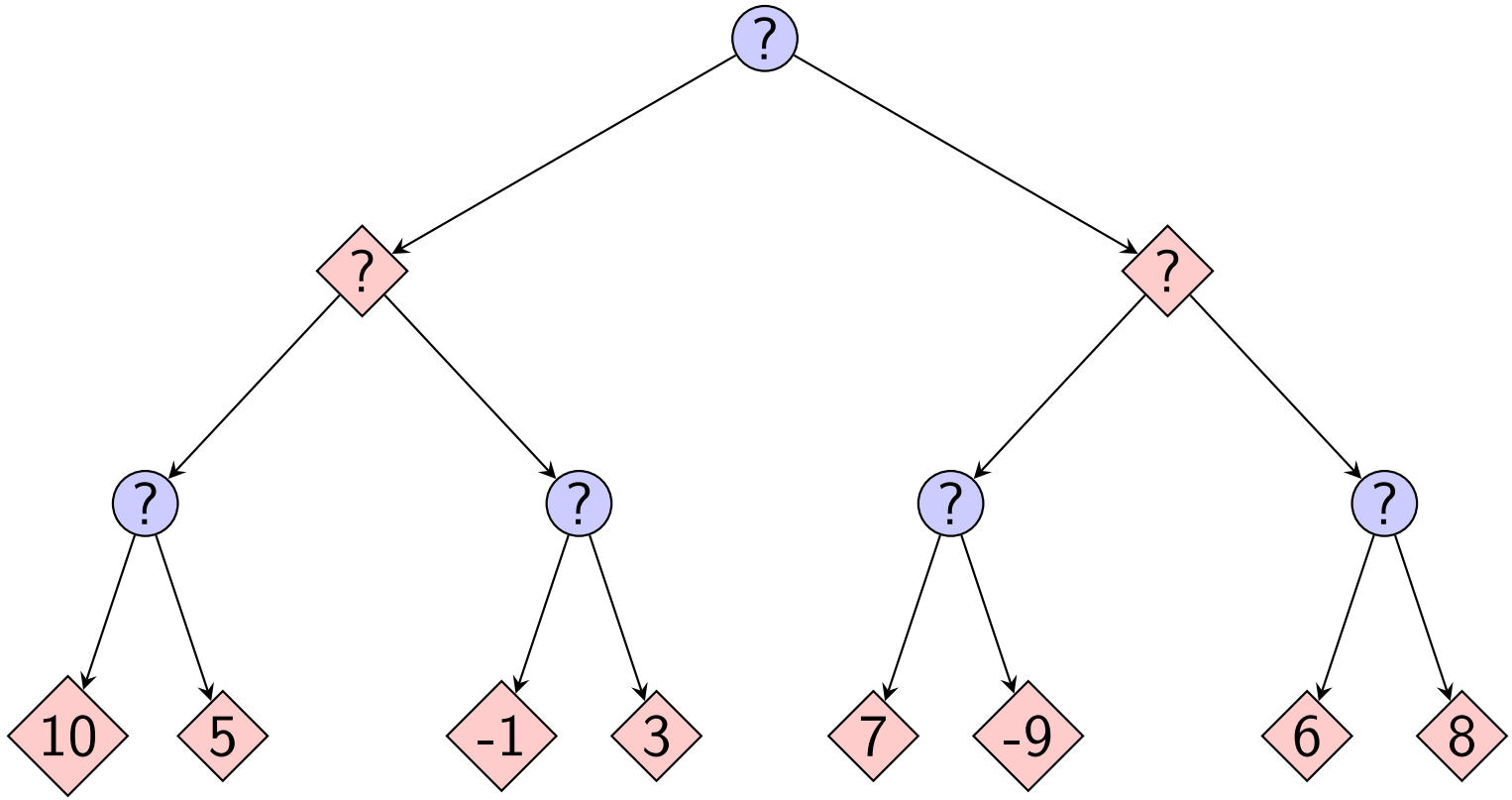
Then...

- For MAX, we are done
- For MIN, **max** the negation, then negate the return value to get the actual **min**.
- Now both players are maximizing, we can use the same piece of code.

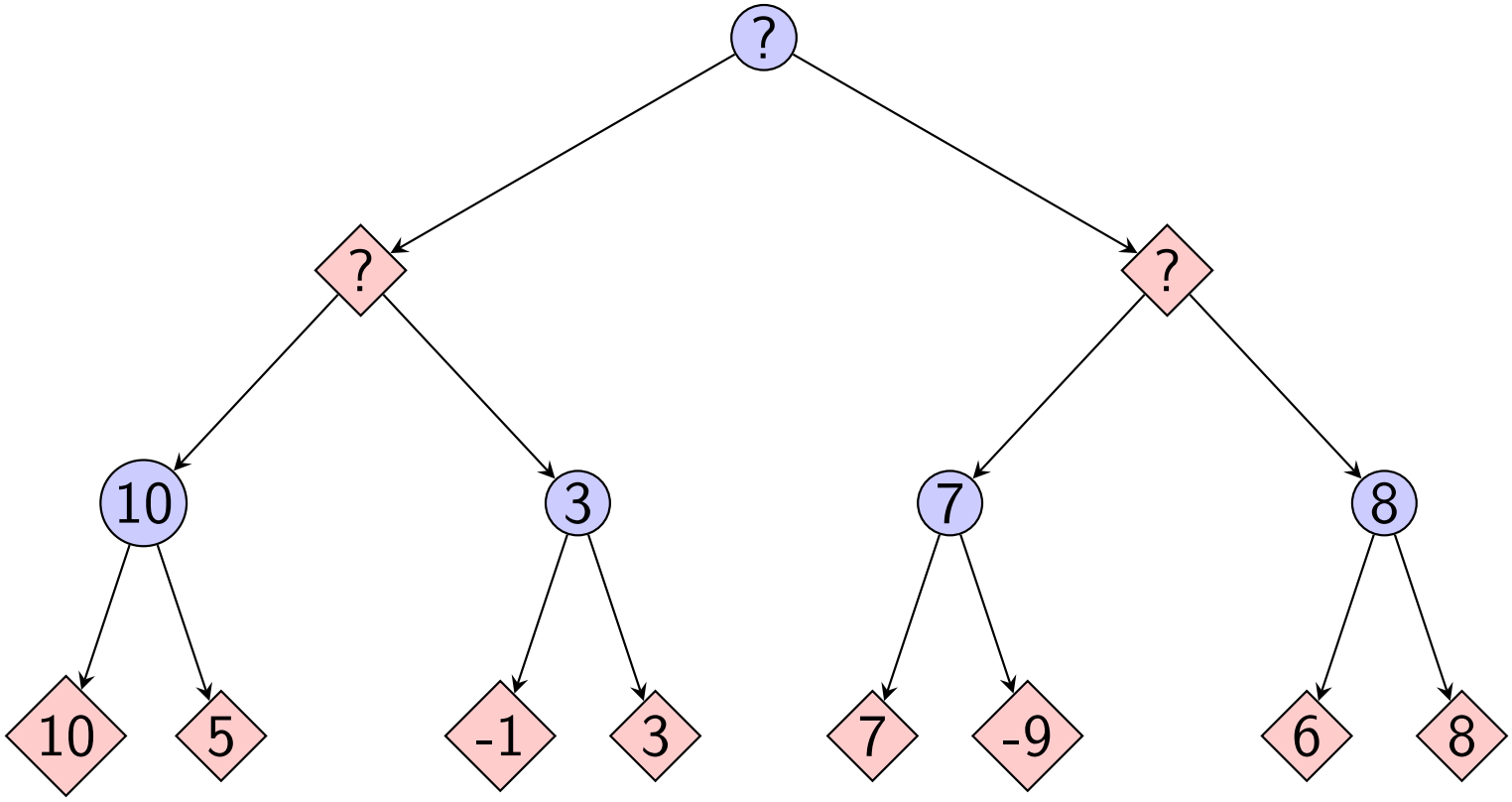
Code from your Game handout:

```
int minimax(Position p) {
  if (p is a leaf) {
    // position value of current player
    return p.evaluate();
  }
  int bestValue = −∞
  for (move in p.getMoves()) {
    p.applyMove();
    int value = -minimax(p);
    p.undoMove();
    if (value > bestValue) {
      bestValue = value;
    }
  }
}
```

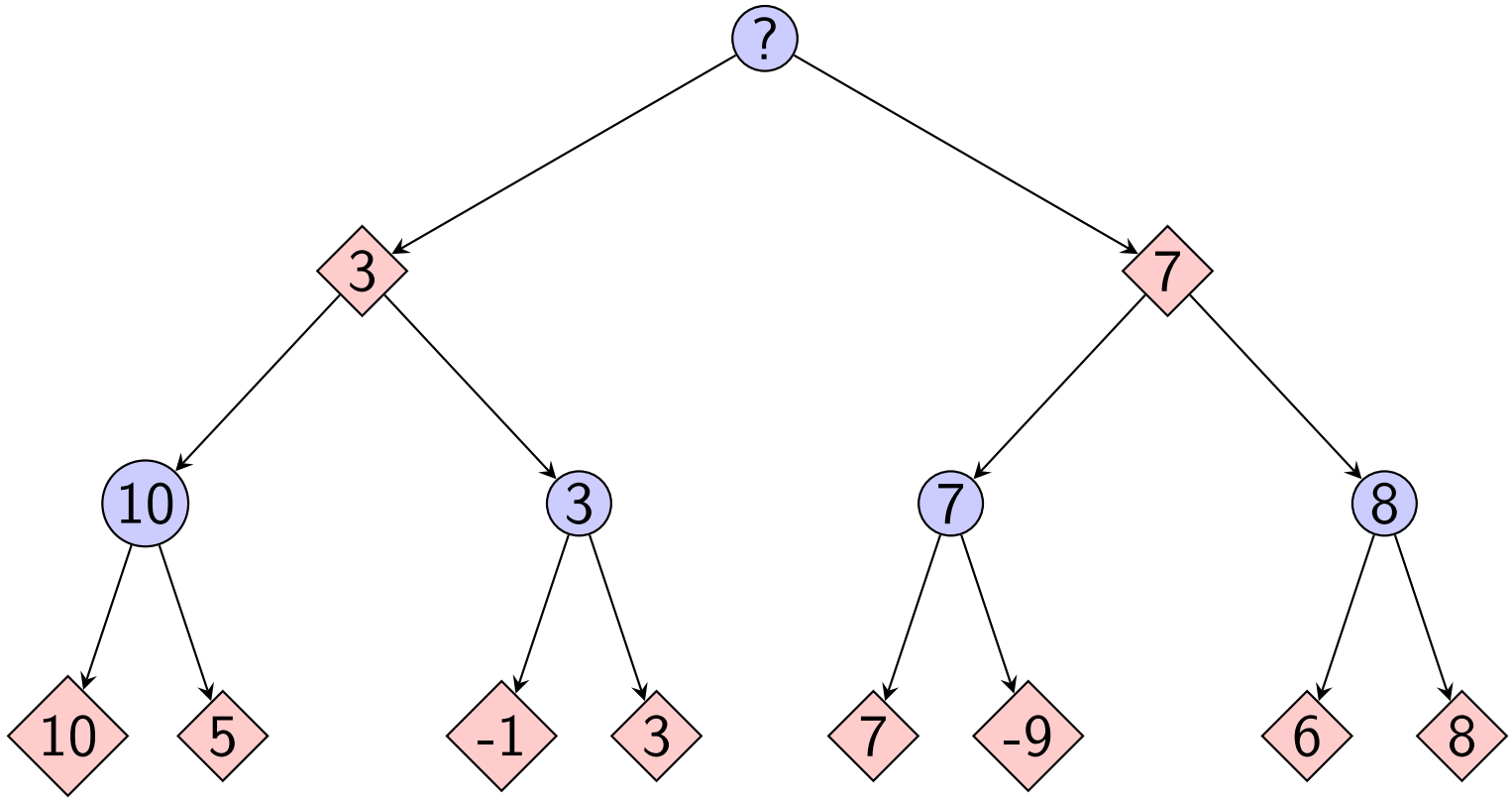
Minimax Example



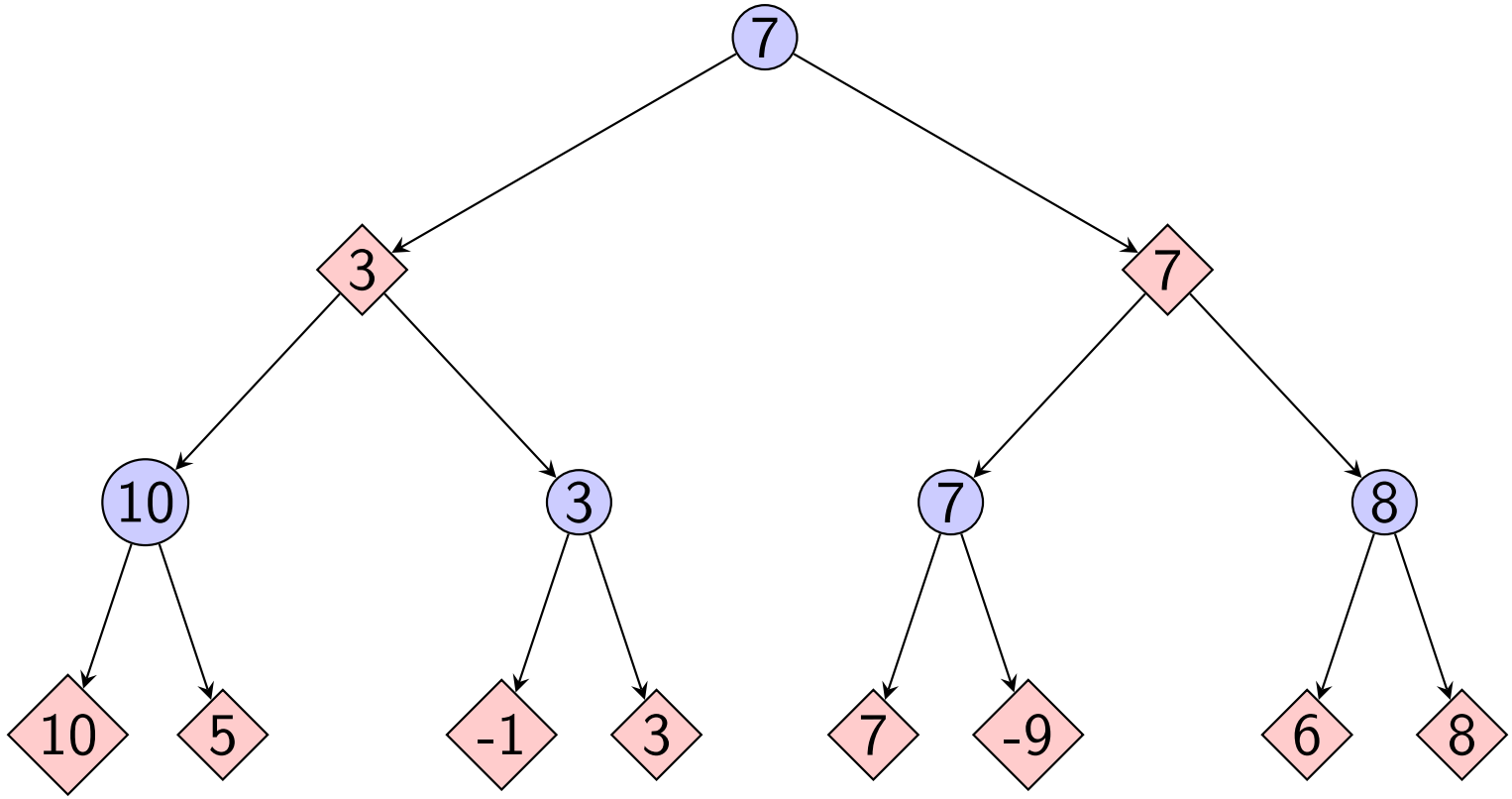
Minimax Example



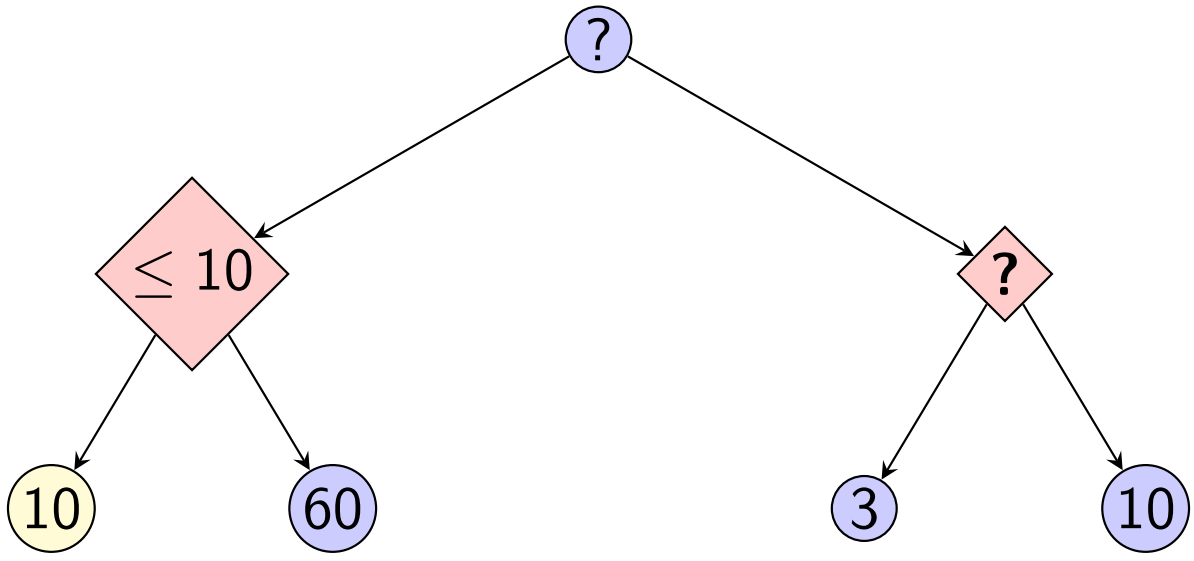
Minimax Example



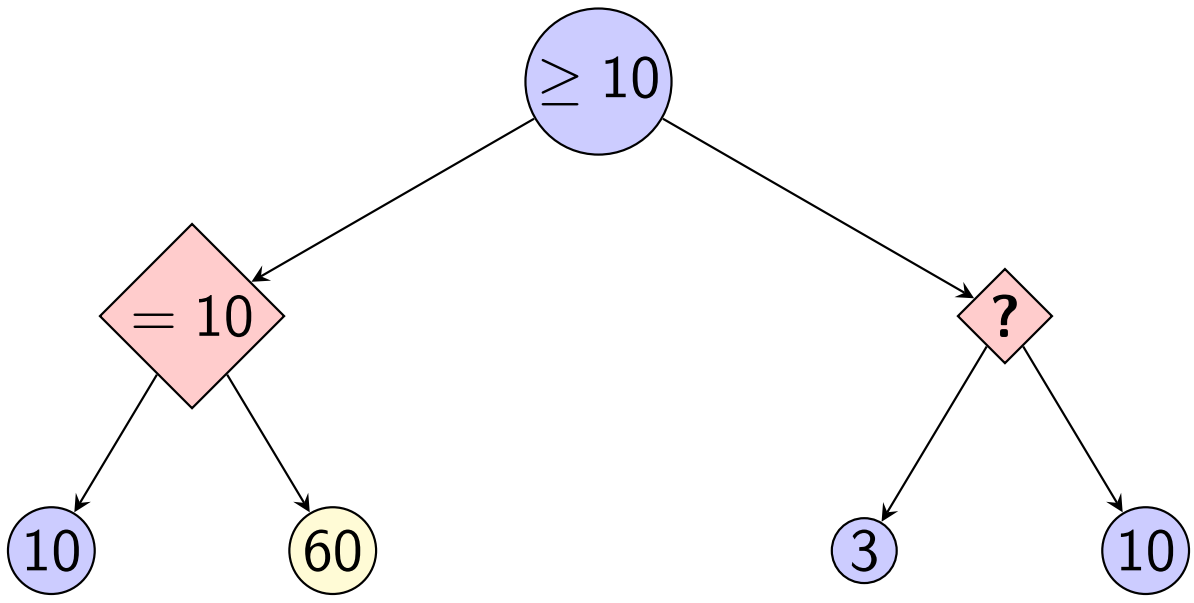
Minimax Example



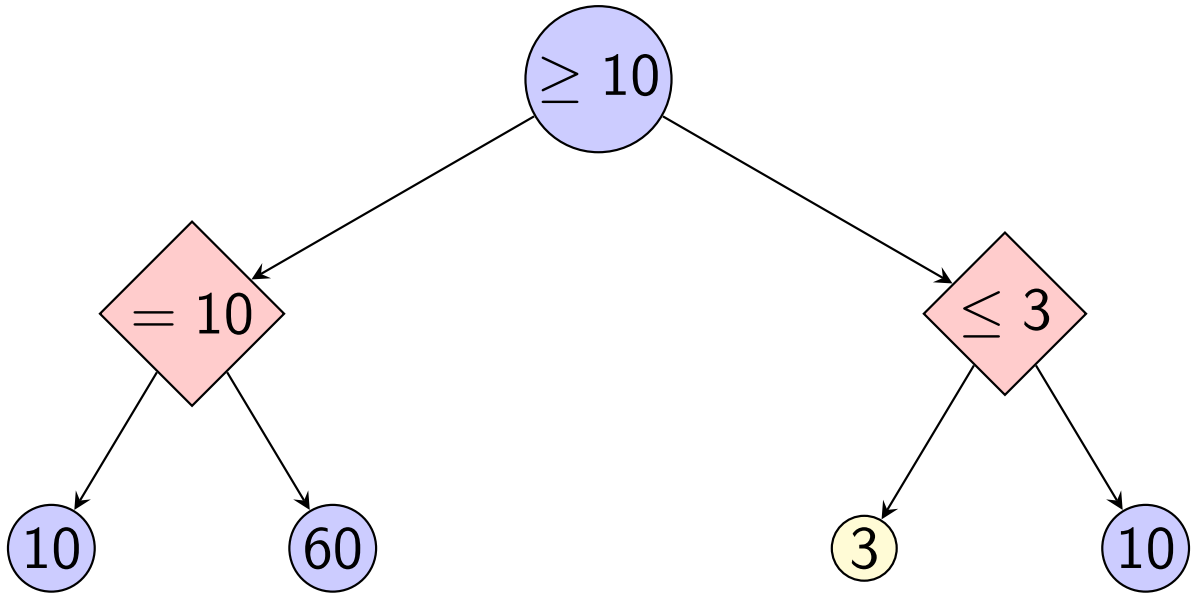
Do we need to look at every branch?



Do we need to look at every branch?

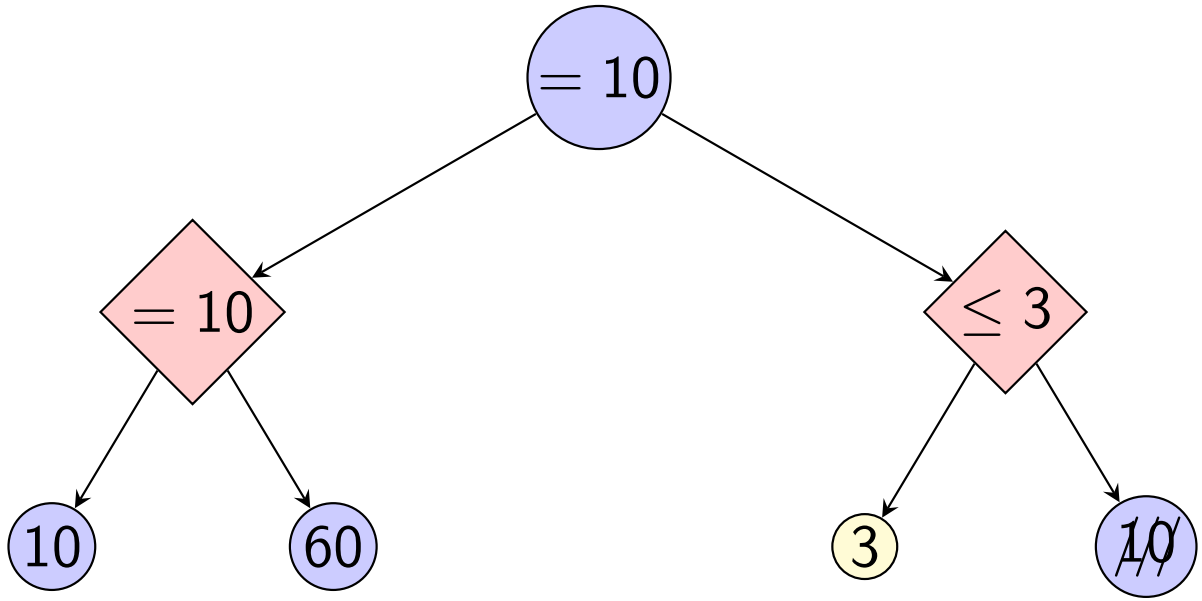


Do we need to look at every branch?



Now, without looking at 10, we know that the minimizer will only give a value ≤ 3, so no possible value bigger than 10 at root

Do we need to look at every branch?



Now, without looking at 10, we know that the minimizer will only give a value ≤ 3, so no possible value bigger than 10 at root

Alpha Beta Pruning (a.k.a What we just did)

First, when did we stop?

When we know MAX has range ≥ 10 but MIN will only give ≤ 3.

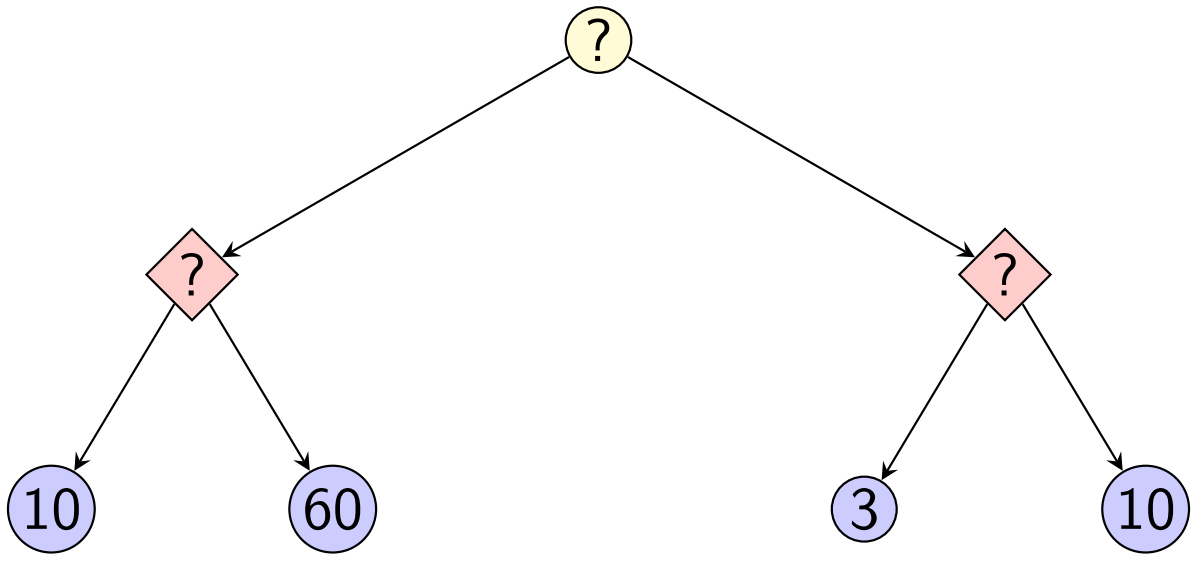
What is the 10?

- 10 is the best value for MAX we found **so far**.
- The branch with 3 will not give us a better value than 10. So we stop.
- 10 is the α value, 3 is the β value. We stop when $\alpha > \beta$.
- We return α for the result.

Alpha and Beta

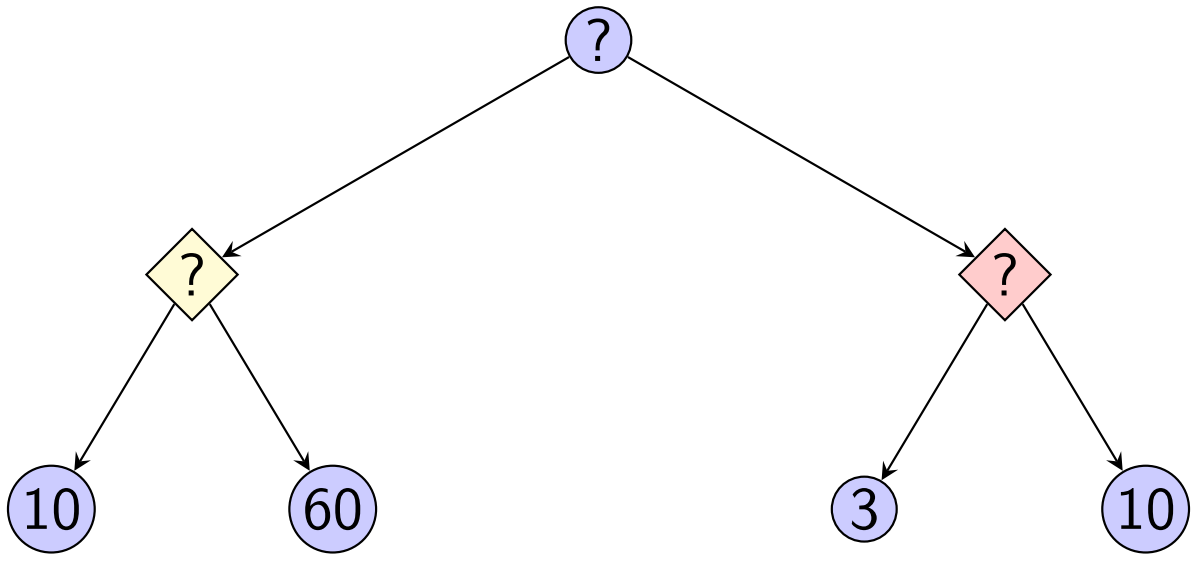
- α is the best MAX we found along the path to the root.
- β is the best MIN we found along the path to the root.

Example



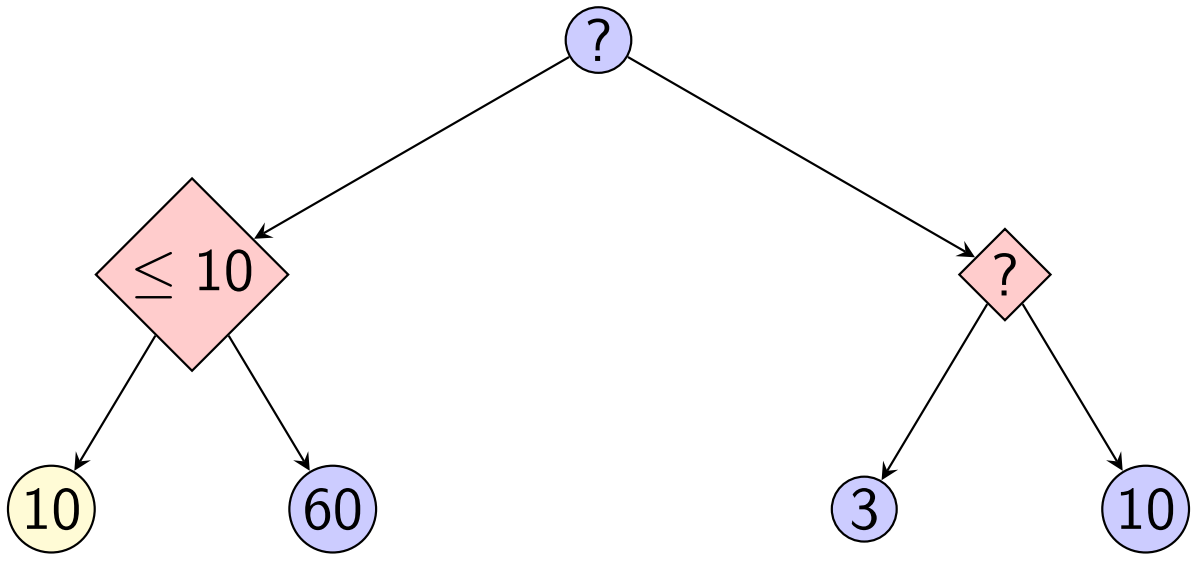
$$\alpha = -\infty, \beta = \infty$$

Example



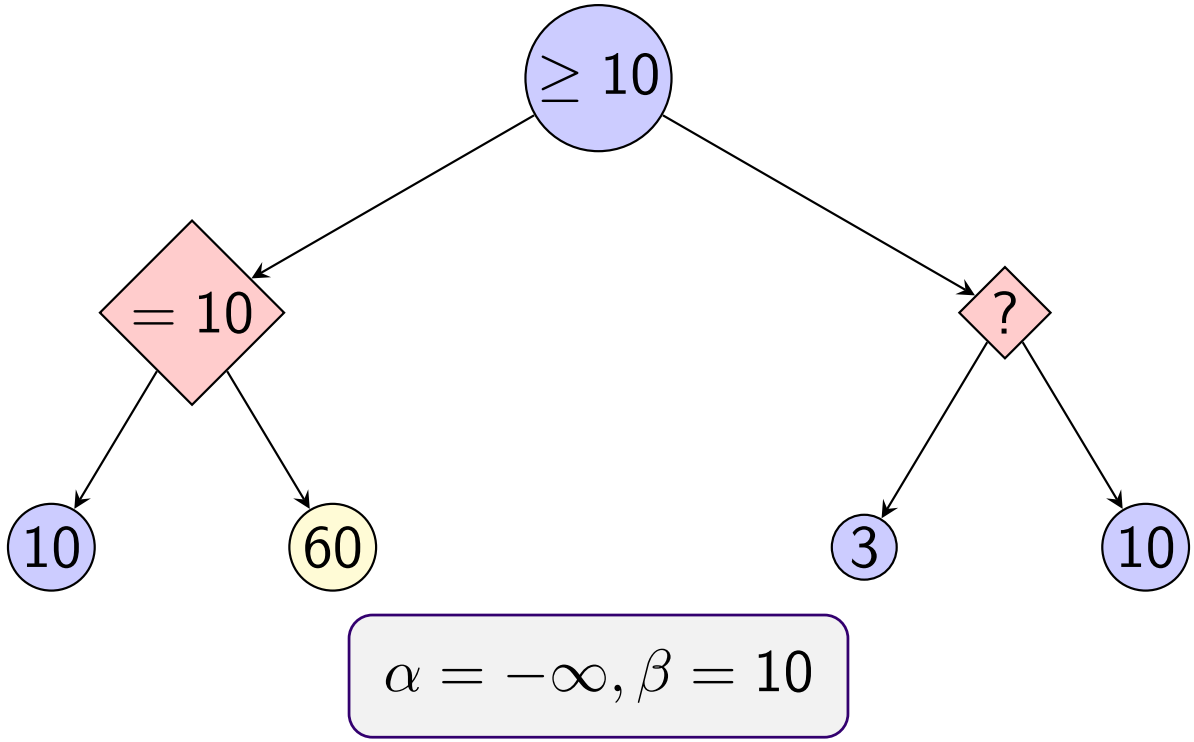
$$\alpha = -\infty, \beta = \infty$$

Example

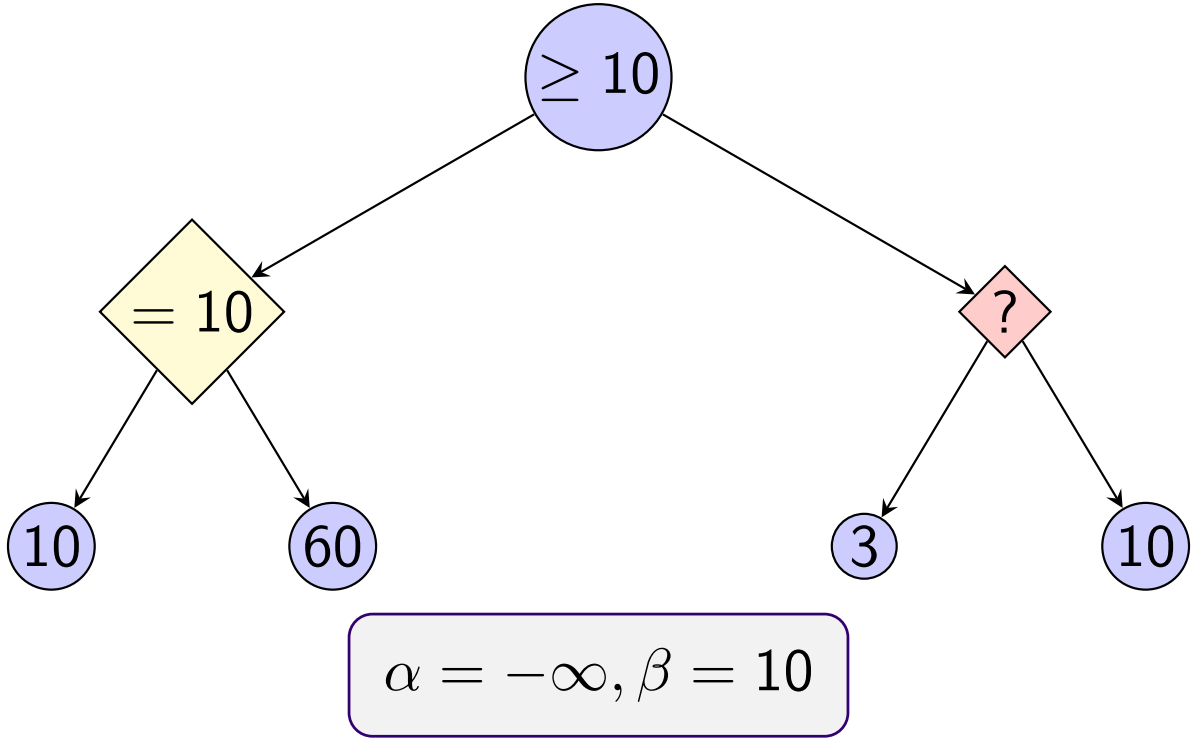


Example

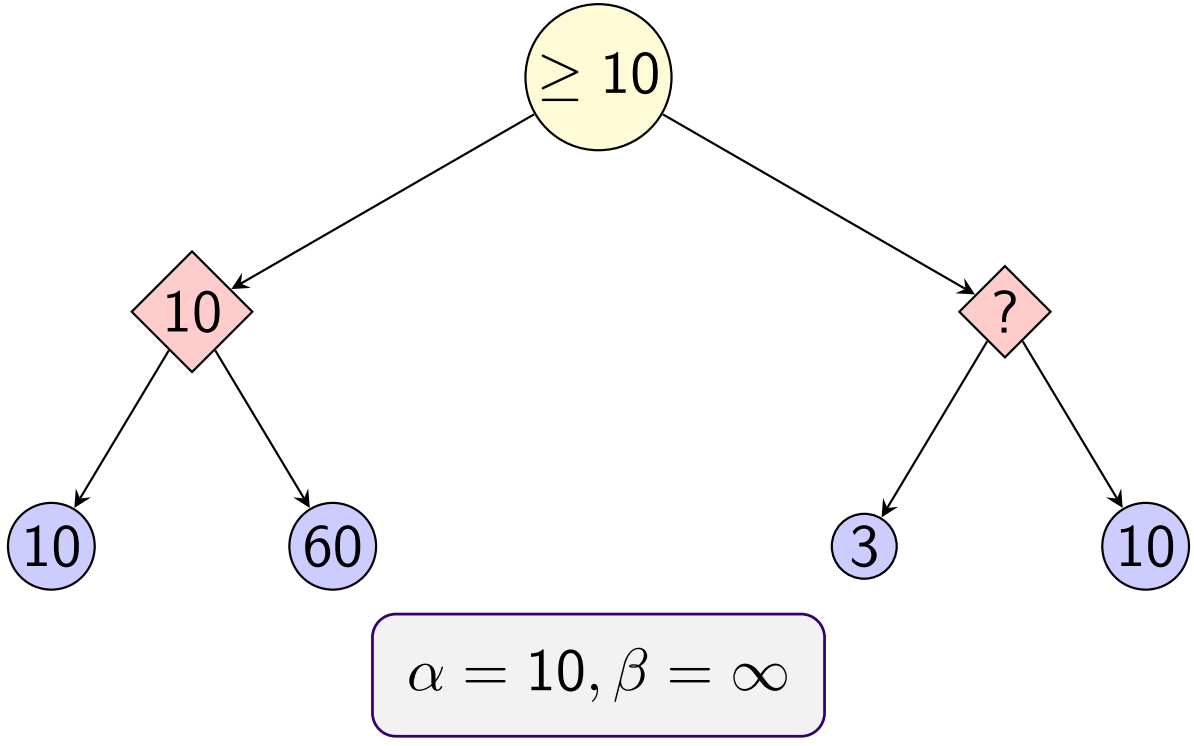
Do we check 60? Yes: $\alpha < 10$.



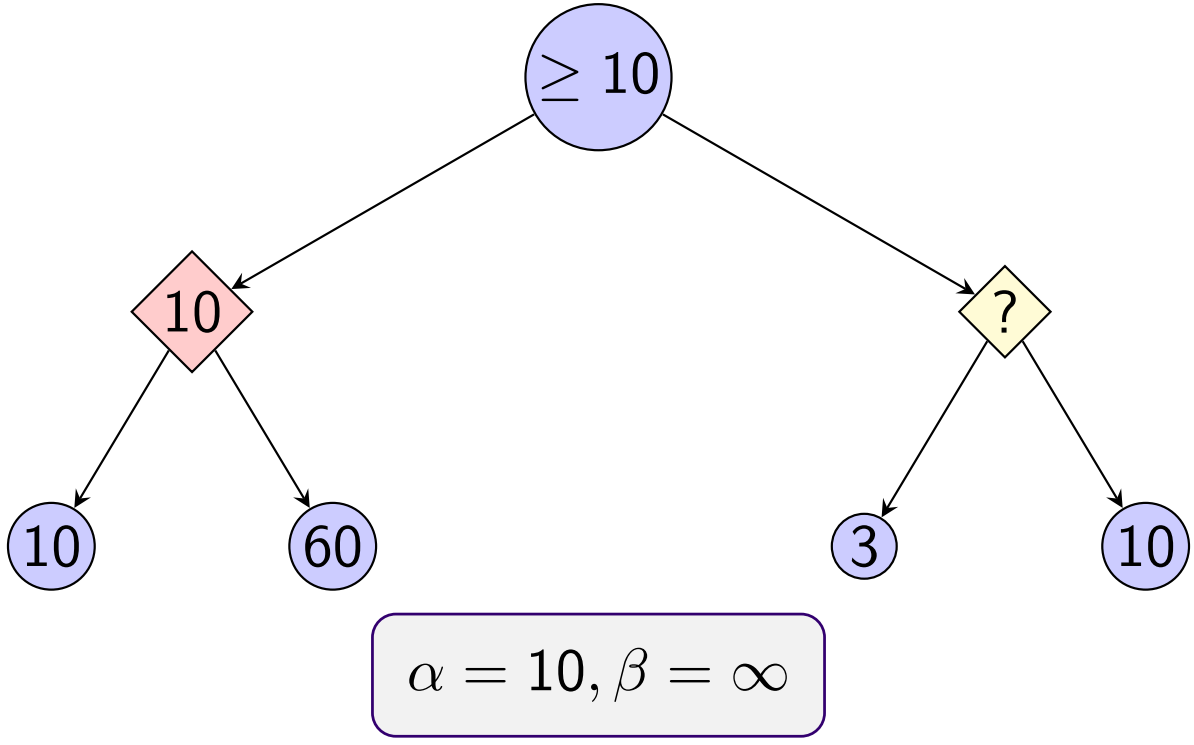
Example



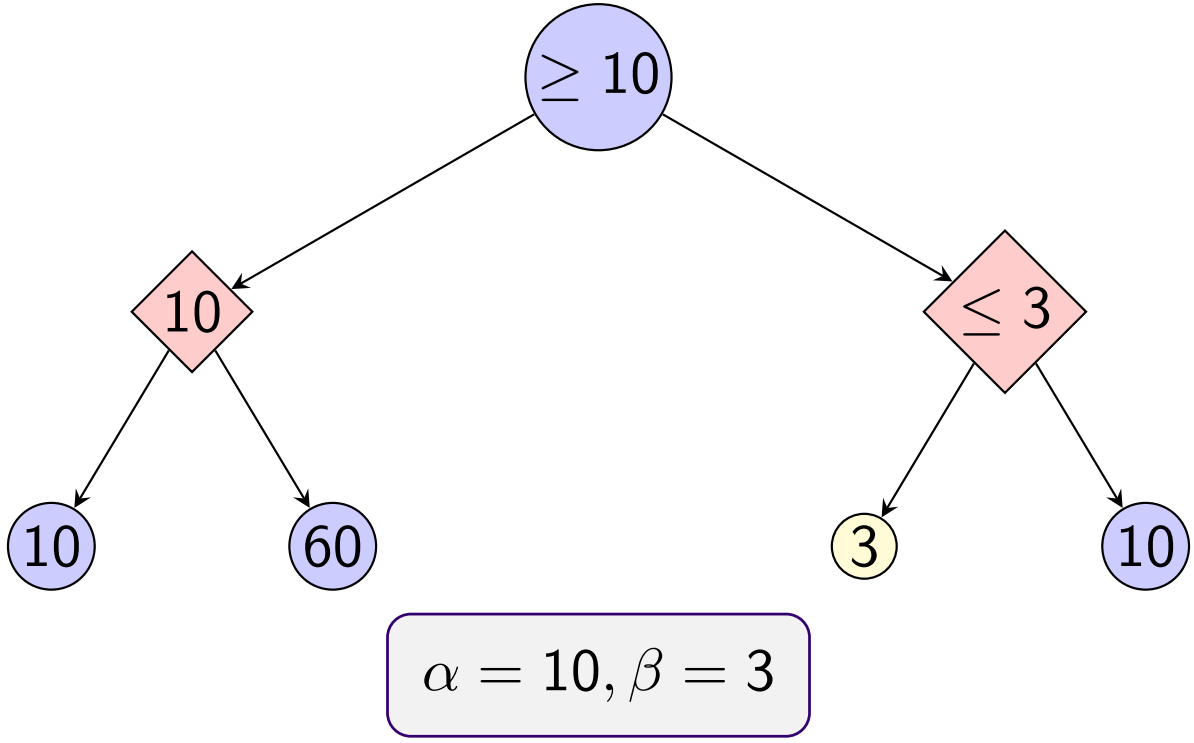
Example



Example

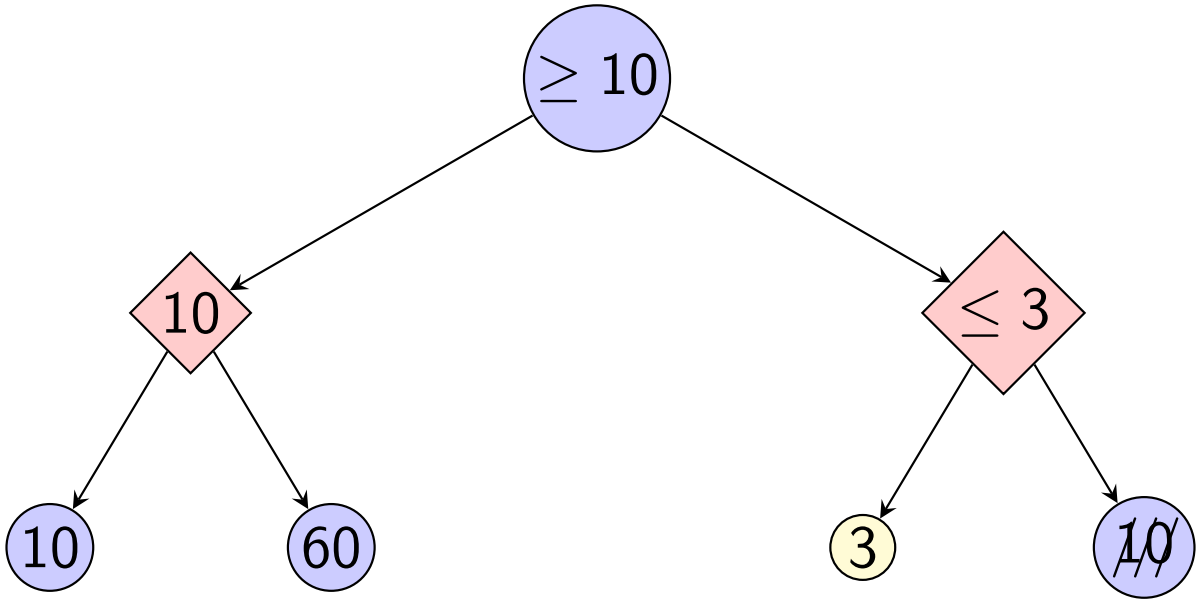


Example



Example

Do we check 10? No: $\alpha = 10 > 3$



$\alpha = 10, \beta = 3$, we stop!