Games

Data Structures and Parallelism

Announcements

Individual Gitlab repositories for exercises 8-11 are out.

- They involve coding this time
- Meant to be parallel practice for P3

P3 repos and spec will be out tonight.

You've done a ton of work so far.

- You've implemented every core data structure.
- And in 4-5 days less than students get during the regular school year.

You should be proud of yourselves.

Let’s Play a Game

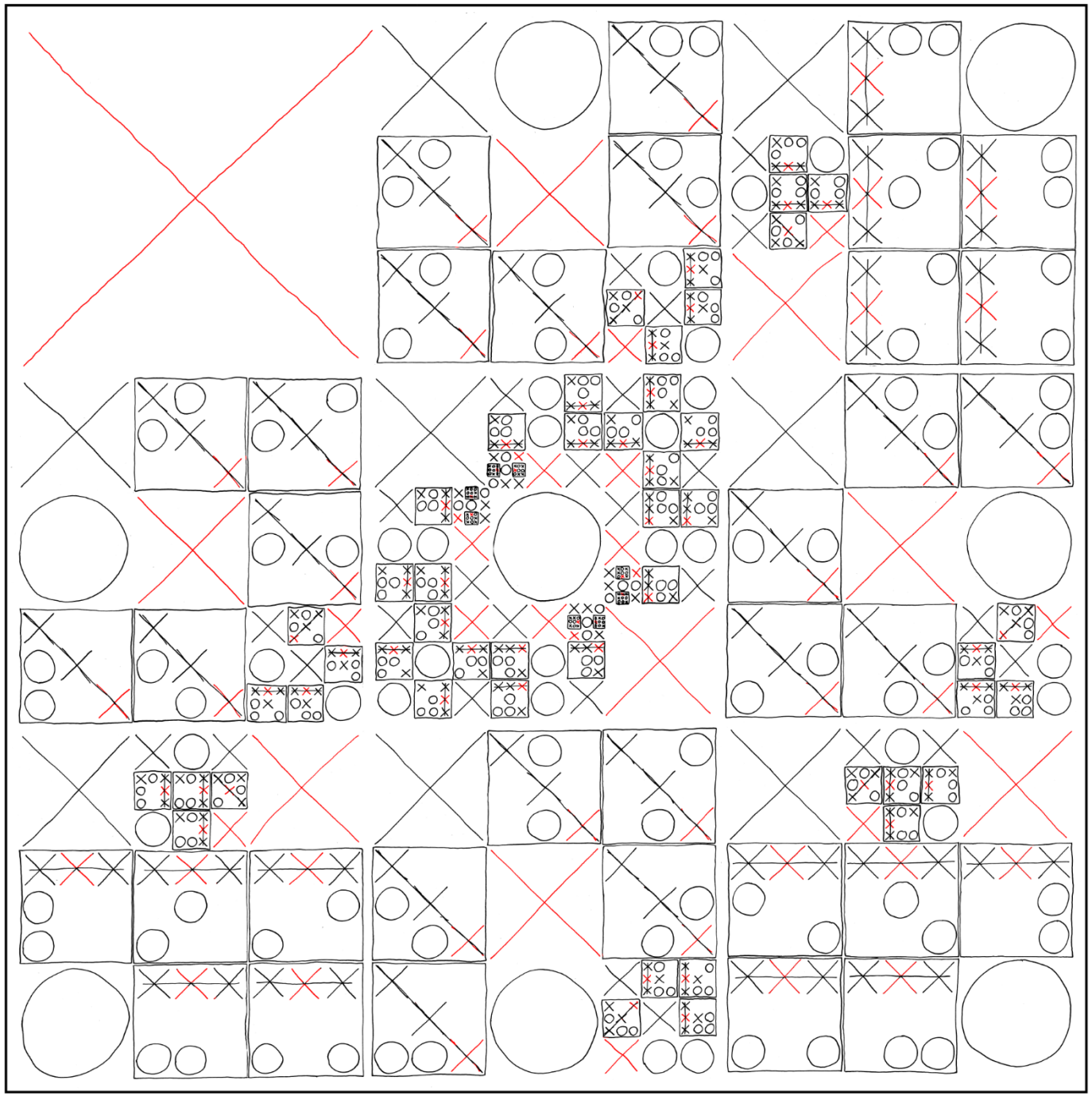
Of tic-tac-toe!

Tic-Tac-Toe

What happened?

- We all have strategies memorized that guarantee a draw (probably).

Let's make a computer do it for us!



www.xkcd.com/832

Tic-Tac-Toe Bot

```
if(board is empty)
    play middle
if(we can win)
    make winning move
if(opponent can win)
    block that move
else{
    //uhhh….
}
```

Tic-Tac-Toe Bot

Could we just list out all positions?
There are $9!$ of them.

What if we want a more complicated game?
Like checkers. Or chess.

Zero-Sum Games

Tic-Tac-Toe, Checkers, and Chess are all zero-sum games

Meaning what's good for me is (equally) bad for you.

Not all "games" are zero-sum (Prisoner's dilemma).

All of these games are also turn-based

Which will make writing our bot easier.

Checkers Bot

Let's try checkers!

Like our tic-tac-toe bot, we need to define the optimal move at every time.

But we DEFINITELY can't hard-code this.

Key idea:

Computers are good at calculating.

Make the bot re-derive what the best decision is every term.

Checkers Bot

What's the best move?

It's the one such that

when our opponent responds with their move

And we respond with our best move

And they respond with their move

....

We win (or at least draw)

Decision Tree

Minimax

Now that we have our tree, how do we choose our move?

How will our opponent respond in round 2?

Assume they will make the best possible move for them.

- i.e. the worst possible move for us.

Checkers Bot

If we make this tree for tic-tac-toe, how big is it?

The nodes at level k have $9 - k$ options.

Something like $9!$ nodes.

For checkers?

The analysis is harder. Checkers experts say there are about 10 possible moves each turn.

So our tree will have 10^t nodes after t turns

Checkers Bot

That's probably too big.

If we want to play chess, the branching factor is much worse.

(experts say about 35).

We can't get all the way down the tree!

At a certain point, we'll need to look at the board and estimate how things are going.

- Evaluation function guesses how "good" the board is for us

Minimax

```
if(we’re at a leaf or at the depth limit)
      return board evalutation //we’re done!
else{
      for(every possible move mv){
            apply mv to the current board
            value = - minimax(board)
            undo mv from board
            if(value > bestVale)
                  bestValue = value
            }
      return bestValue
}
```

Project 3

The games pdf on the webpage has more examples.

P3 is out, you'll be making a chess bot.

You'll implement minimax (sequentially first, then in parallel)

The alpha-beta optimization (sequential and parallel) is above and beyond (slides on that follow)

Pruning

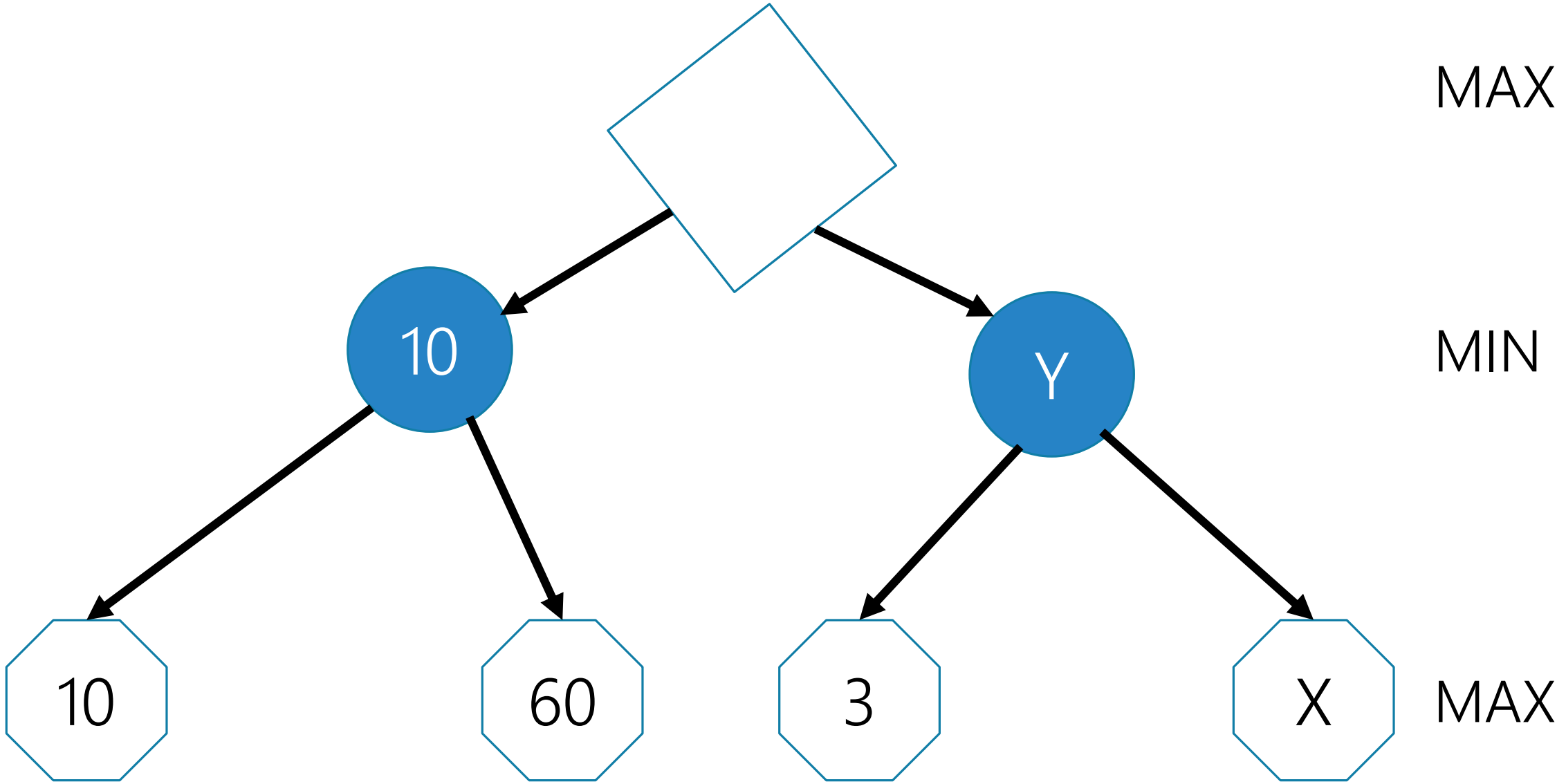
The further down the tree we can go, the more likely we are to get a good move.

What tricks could we use to get further down the tree?

In tic-tac-toe, we didn't evaluate other moves once we knew our opponent could win in the next turn.

Generalize that idea – we don't need to evaluate further in a subtree if we know that optimal play won't take us there.

Alpha-Beta Pruning



Pruning

There's no need to evaluate X

If it's bigger than 3, min will choose Y to go in the left branch.

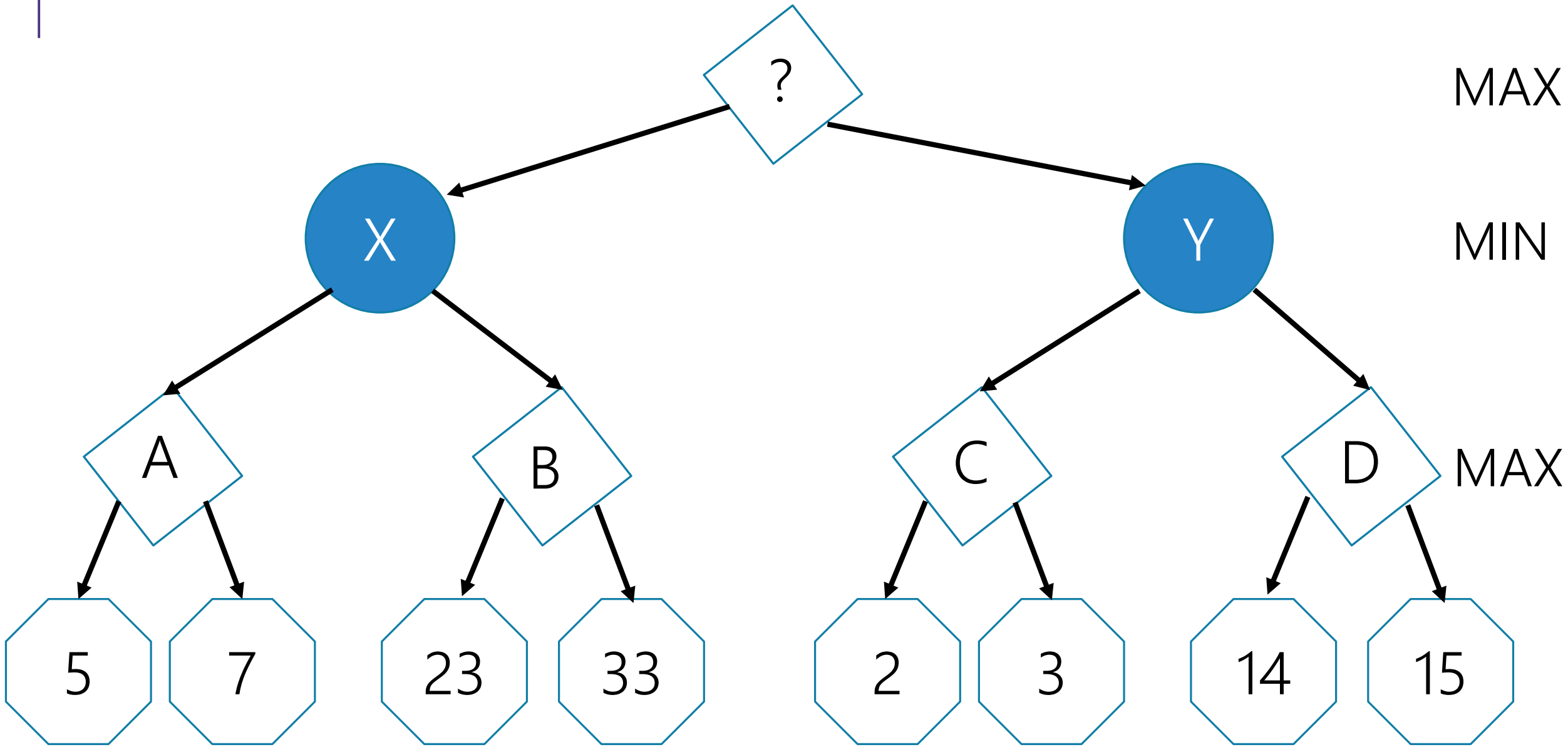
If it's less than 3, min will choose the right branch…

but then at the root max will choose the left branch.

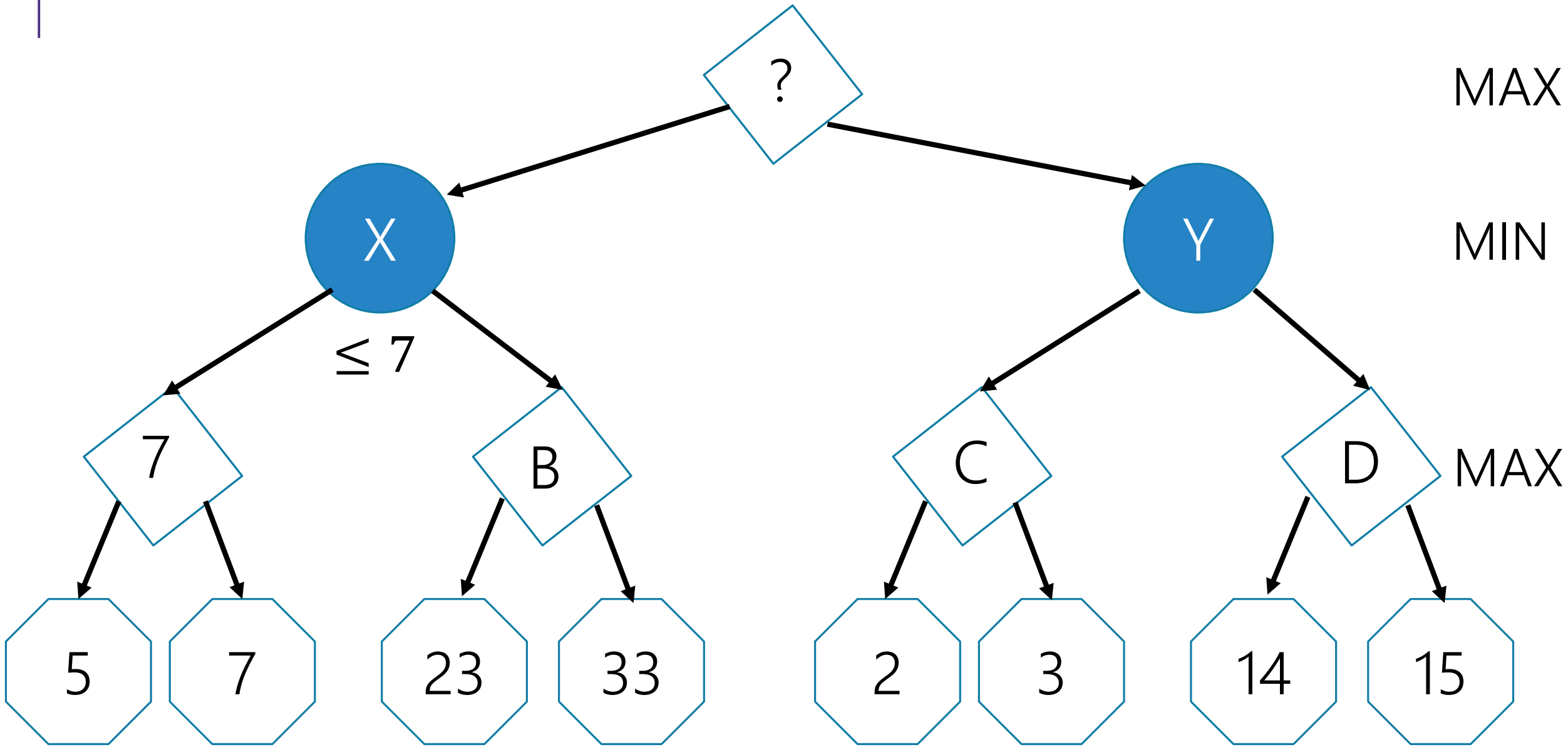
The value doesn't matter!

Don't bother going down that subtree.

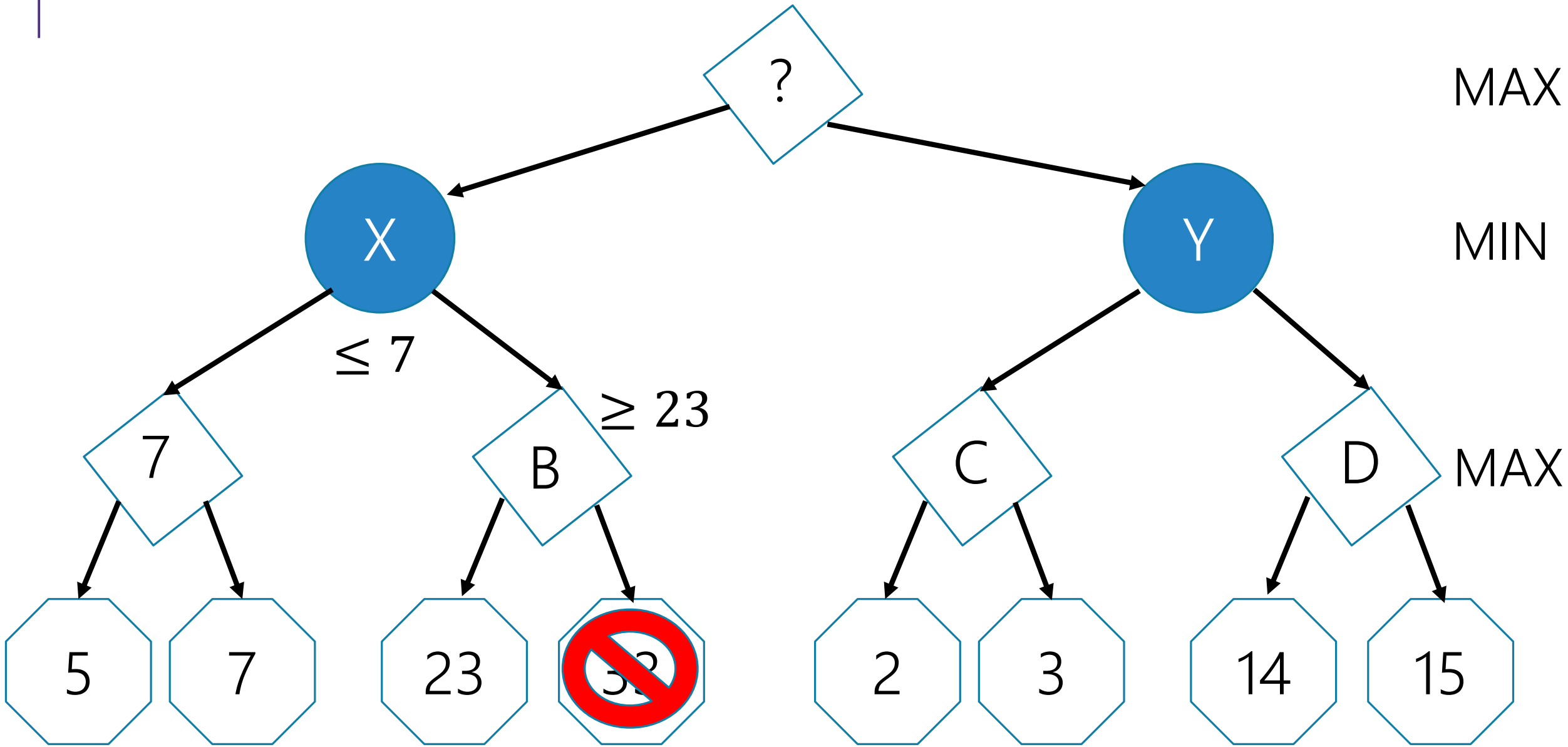
Try by hand



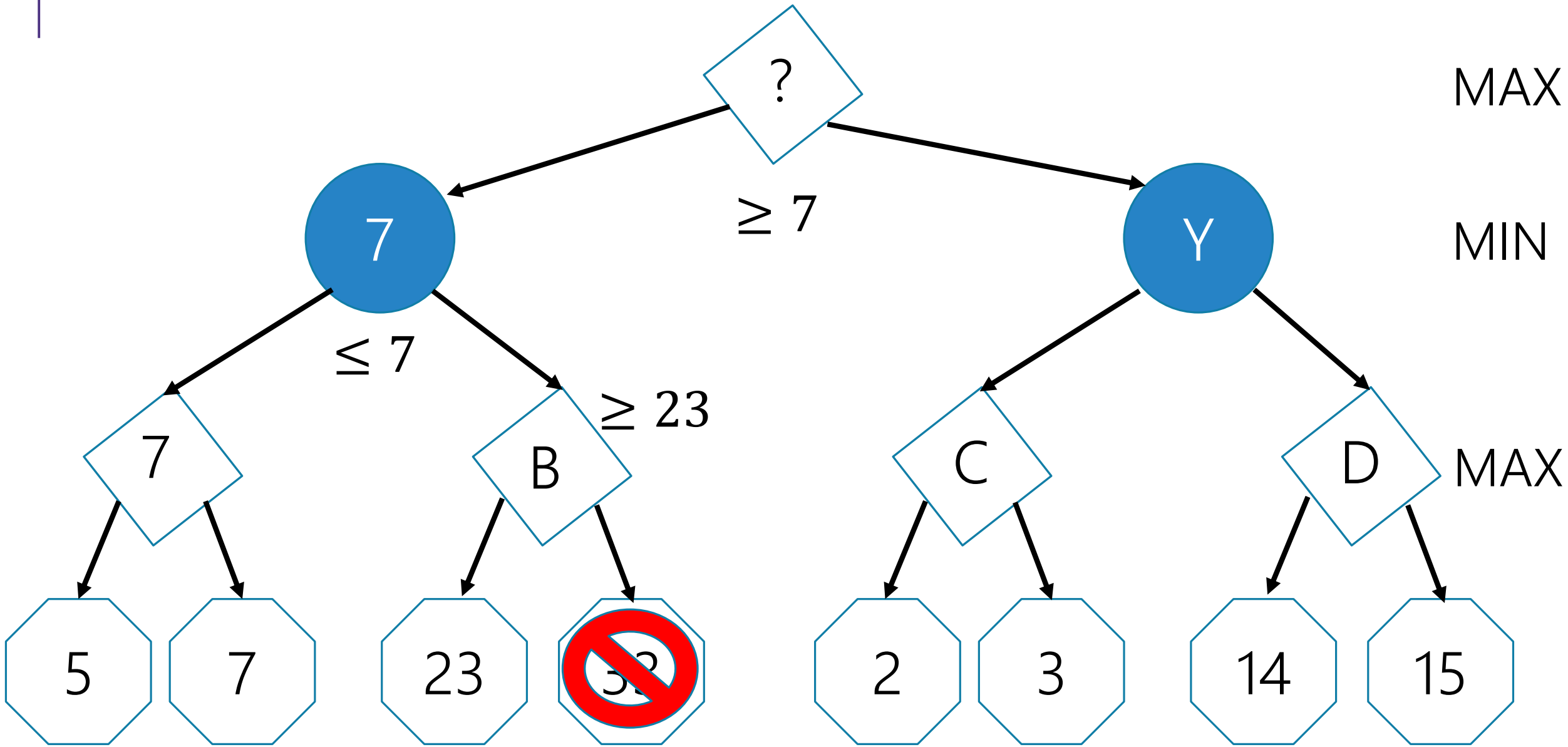
Try by hand



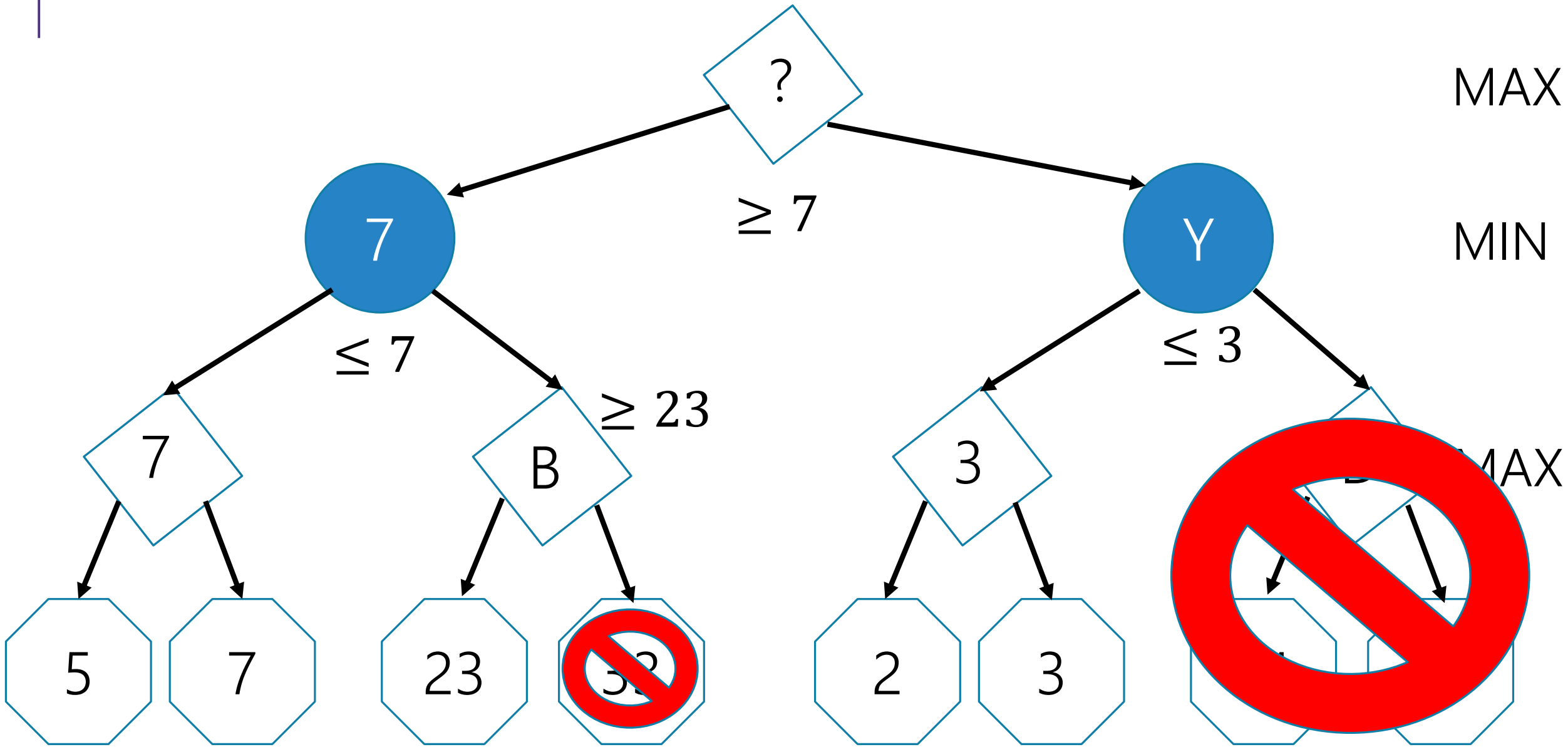
Try by hand



Try by hand



Try by hand



Pseudocode

```
alphabeta(Position p, int alpha, int beta){
      if(p is a leaf)
            return p.evaluate()
      for( every move mv){
            p.apply(mv)
            int value = -alphabeta(p, -beta, -alpha)
            p.undoMove()
            if(value > alpha)
                  alpha = value;
            if(alpha >= beta) //we won't be able to reach this move.
                  return alpha;
      }
      return alpha;
```