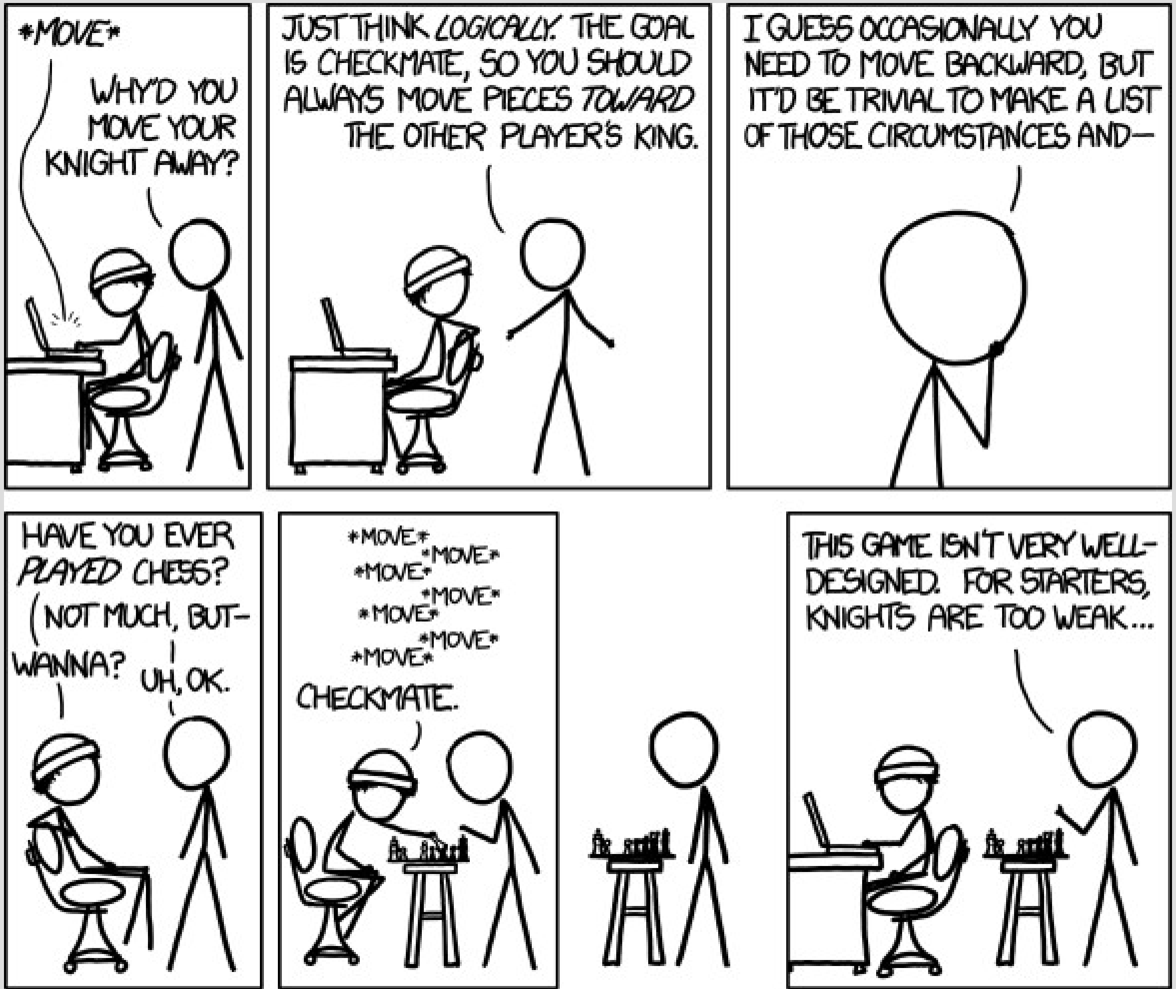
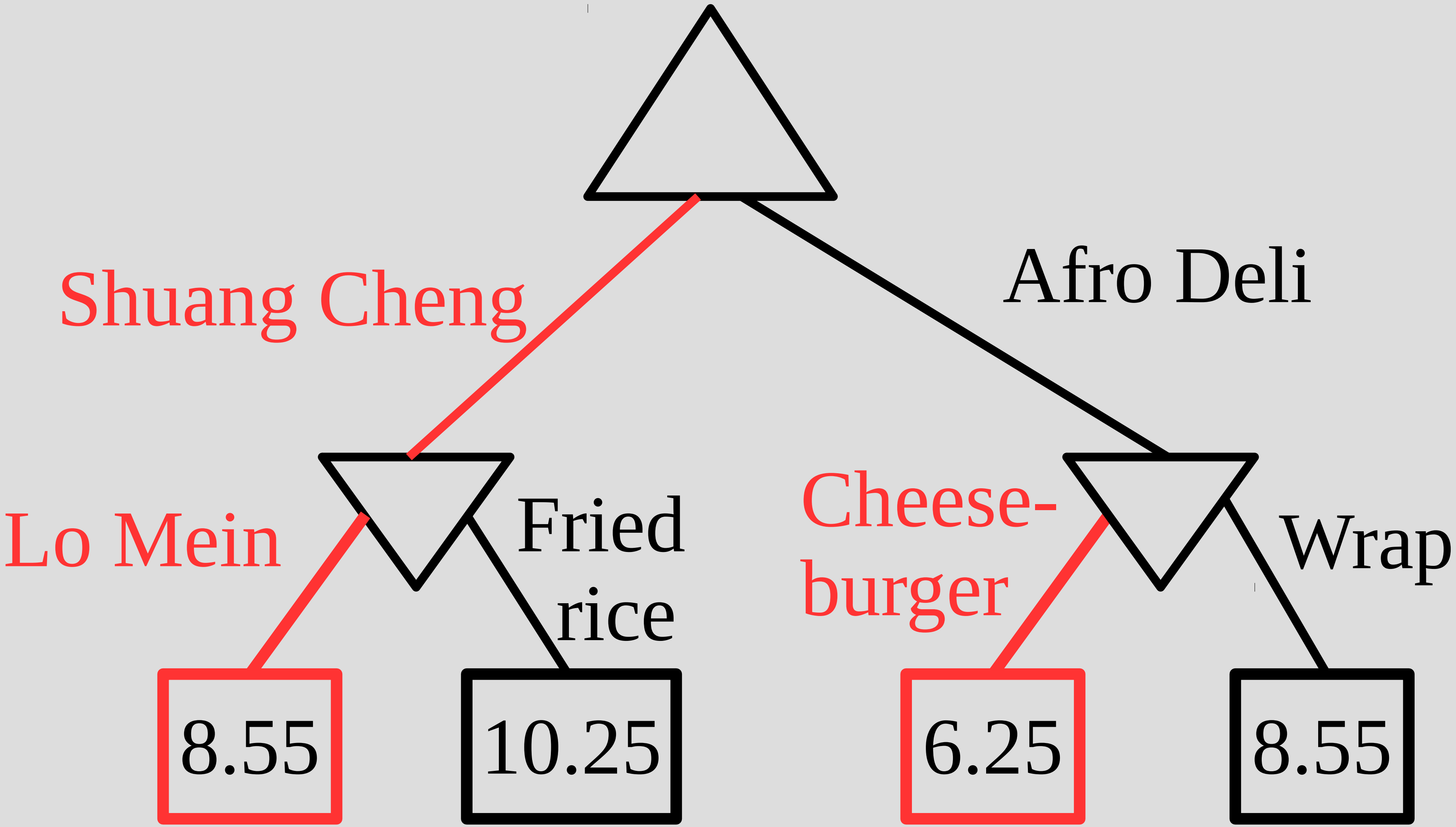
More on games (Ch. 5.4-5.6)



Review: Minimax



Minimax

This representation works, but even in small games you can get a very large search tree

For example, tic-tac-toe has about 9! actions to search (or about 300,000 nodes)

Larger problems (like chess or go) are not feasible for this approach (more on this next class)

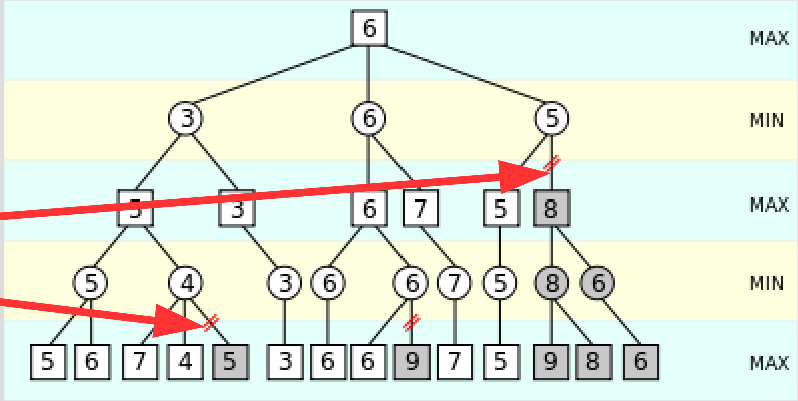
Minimax

“Pruning” in real life:

Snip branch

“Pruning” in CSCI trees:

Snip branch



Alpha-beta pruning

However, we can get the same answer with searching less by using efficient “pruning”

It is possible to prune a minimax search that will never “accidentally” prune the optimal solution

A popular technique for doing this is called alpha-beta pruning (see next slide)

Alpha-beta pruning

Consider if we were finding the following:

max(5, min(3, 19))

There is a "short circuit evaluation" for this, namely the value of 19 does not matter

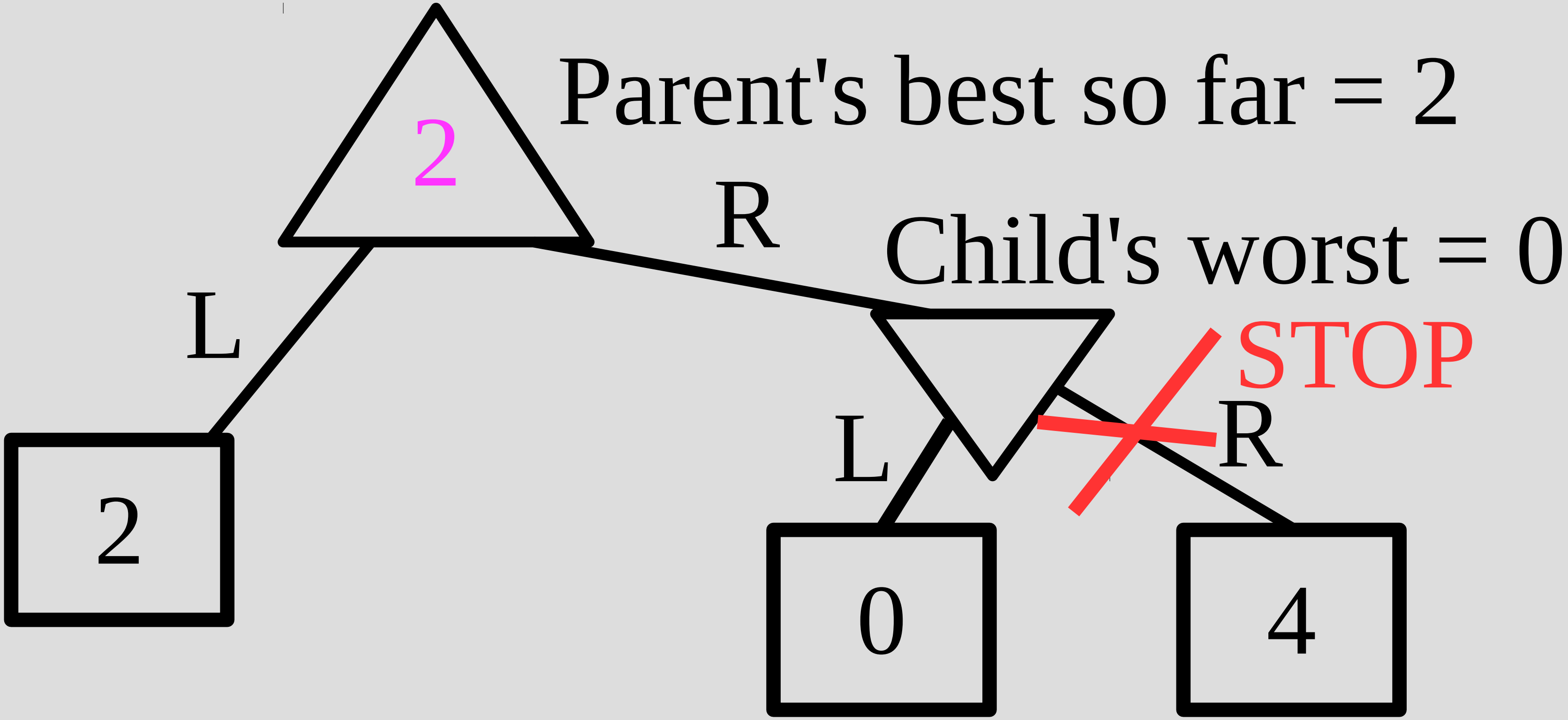
min(3, x) $\leq$ 3 for all x

Thus max(5, min(3,x)) = 5 for any x

Alpha-beta pruning would not search x above

Alpha-beta pruning

If when checking a min-node, we ever find a value less than the parent's “best” value, we can stop searching this branch



Parent's best so far = 2

Child's worst = 0

STOP

Alpha-beta pruning

This can apply to max nodes as well, so we propagate the best values for max/min in tree

Alpha-beta pruning algorithm:

Do minimax as normal, except:

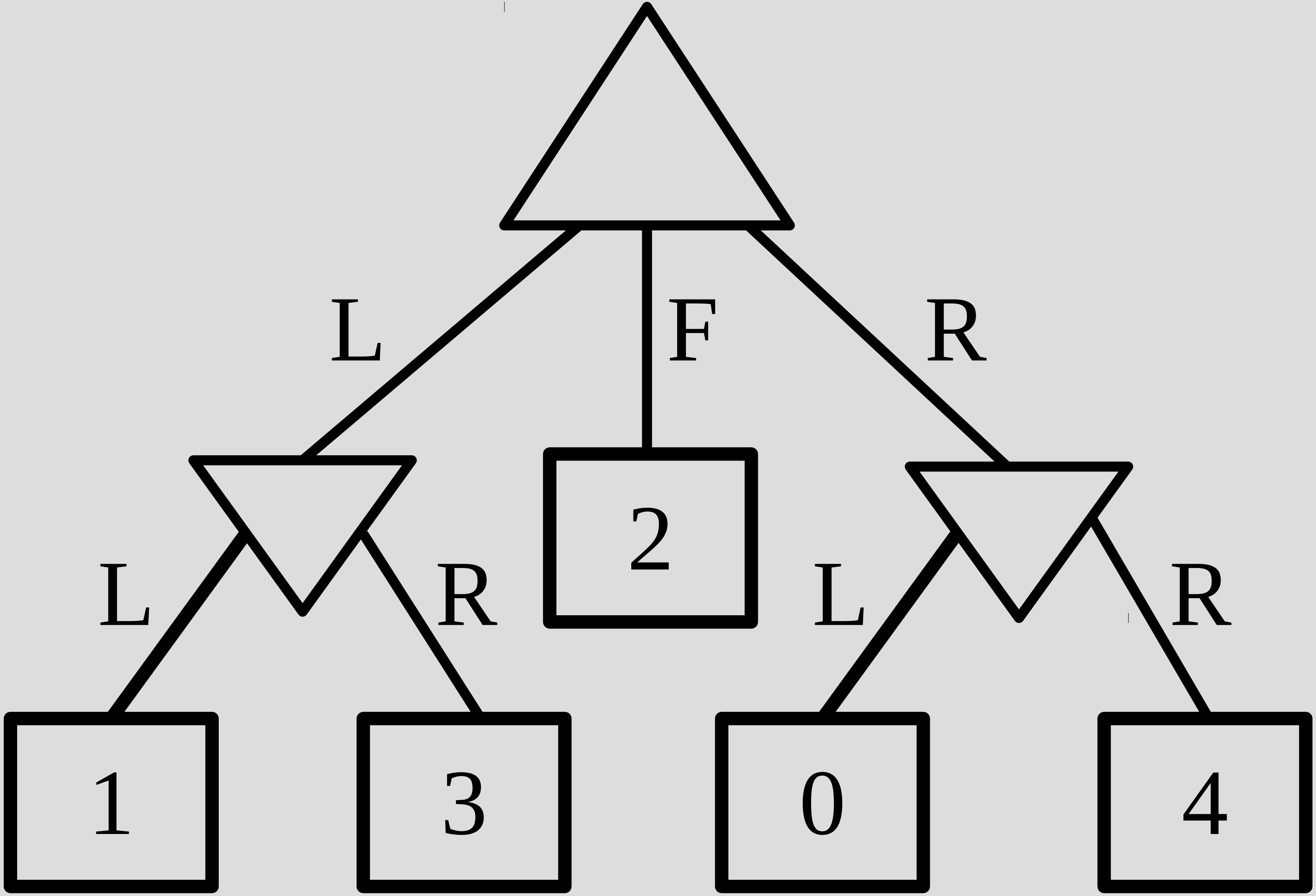
Going down tree: pass “best max/min” values

min node: if parent's “best max” greater than current node, go back to parent immediately

max node: if parent's “best min” less than current node, go back to parent immediately

Alpha-beta pruning

Let's solve this with alpha-beta pruning



Alpha-beta pruning

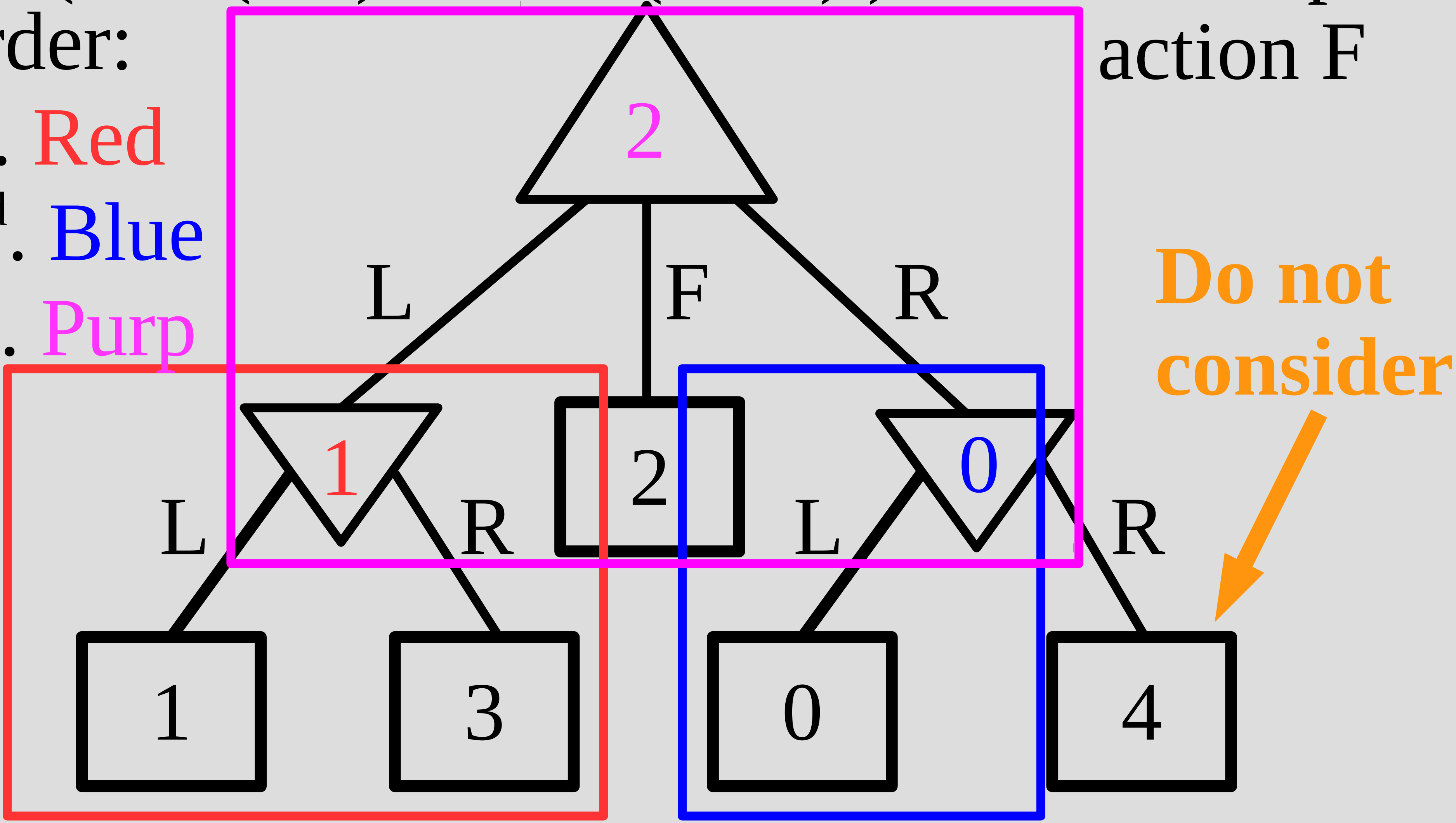
max(min(1,3), 2, min(0, ??)) = 2, should pick action F

Order:

1st. Red

2nd. Blue

3rd. Purp

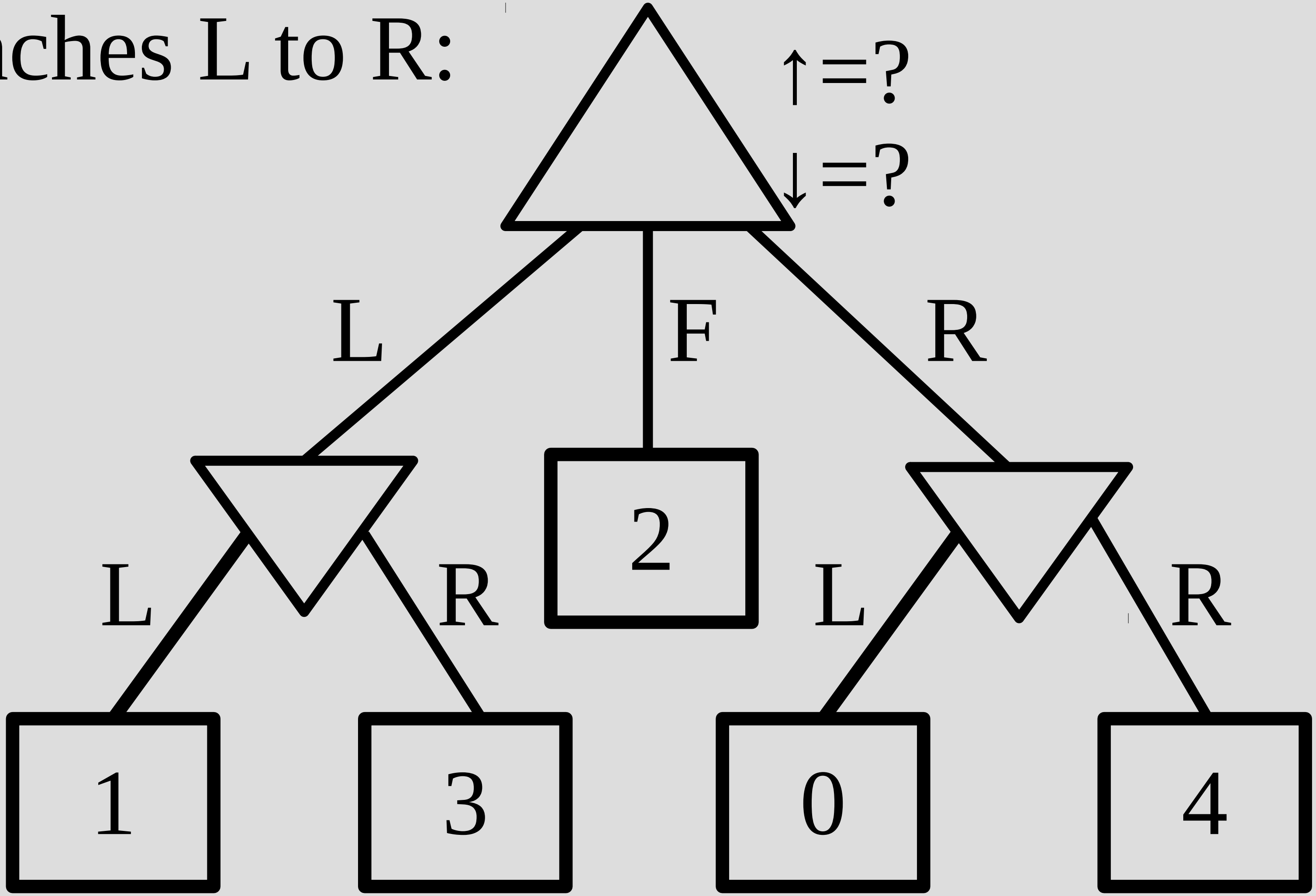


Do not consider

Alpha-beta pruning

Let best max be “↑” and best min be “↓”

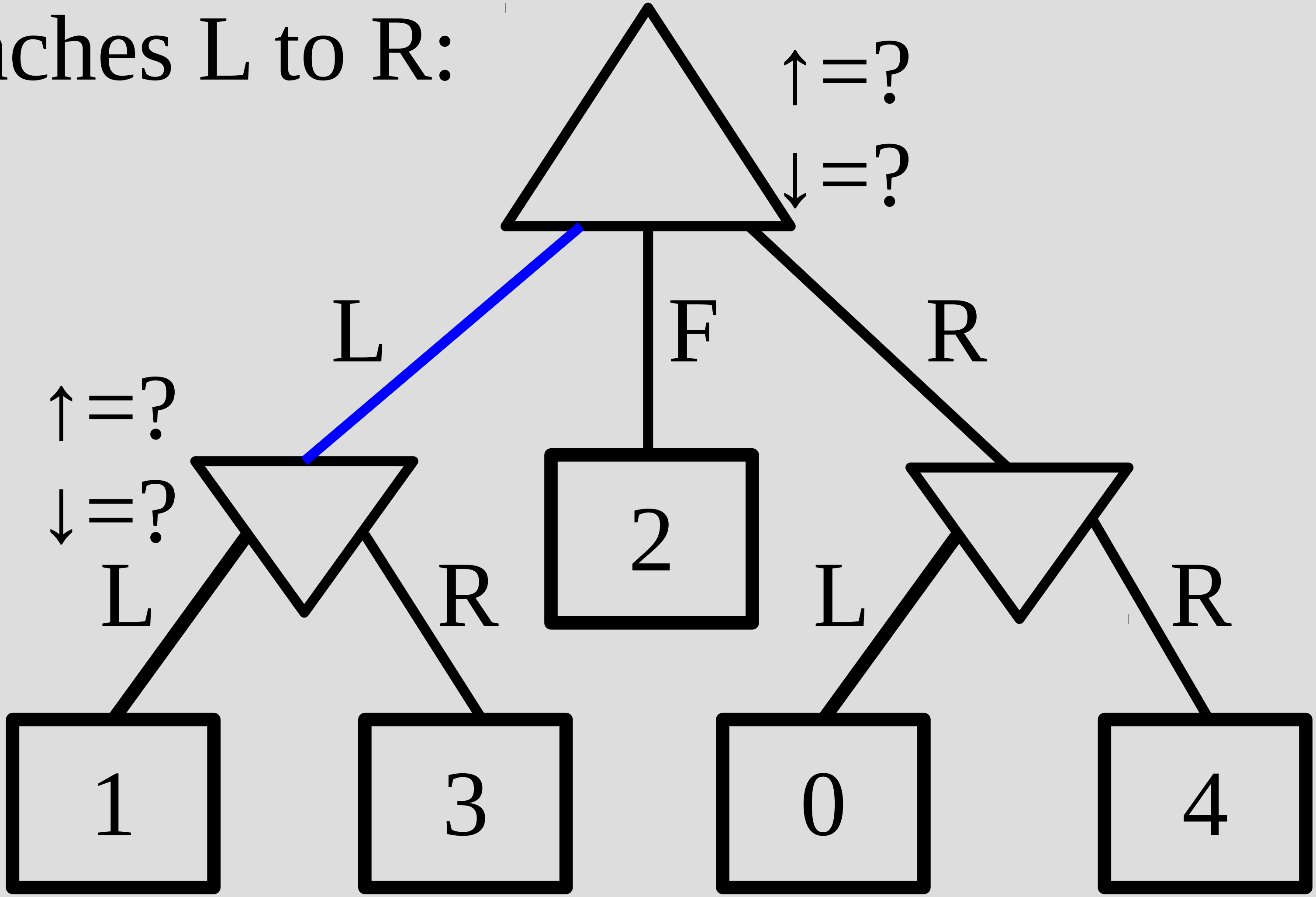
Branches L to R:



Alpha-beta pruning

Let best max be “↑” and best min be “↓”

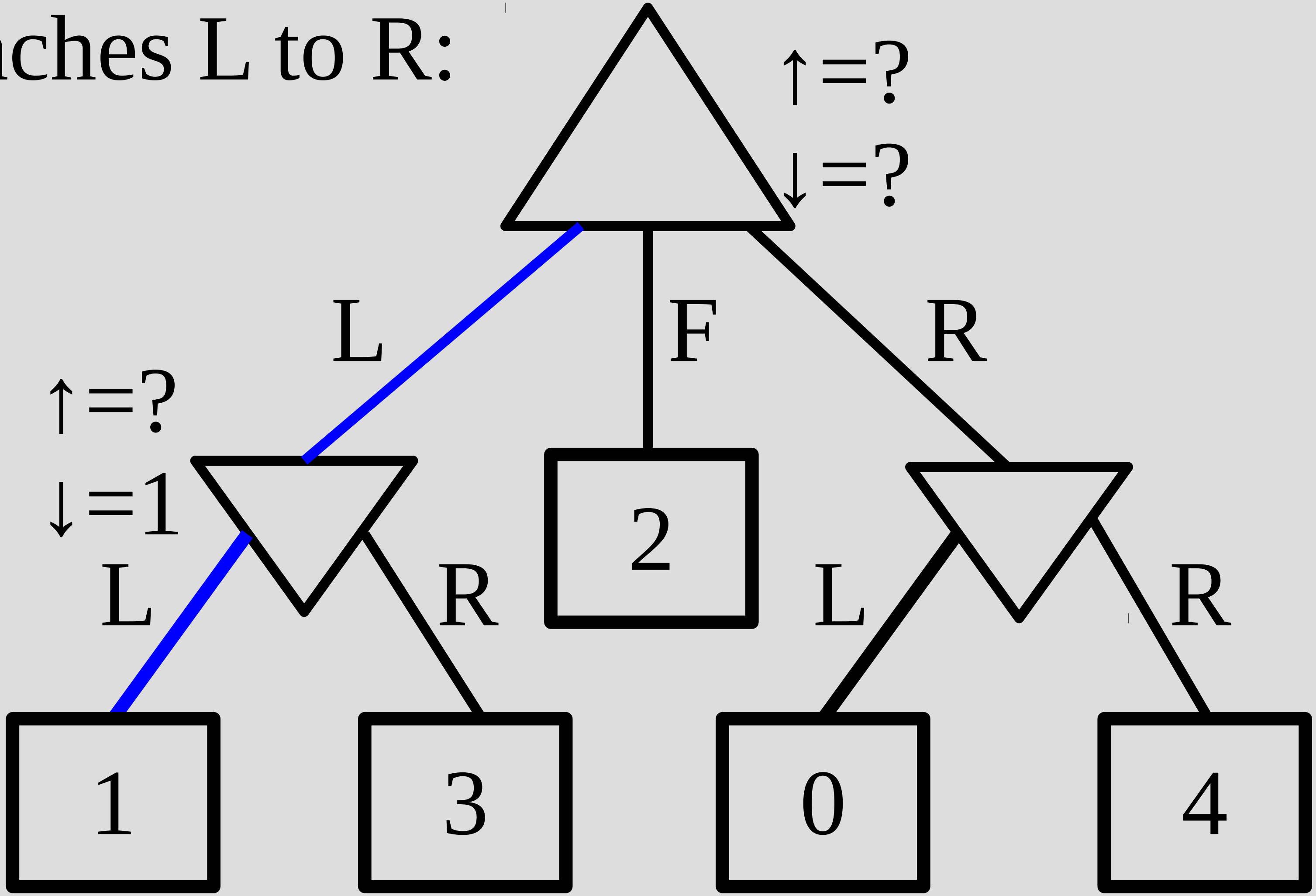
Branches L to R:



Alpha-beta pruning

Let best max be “↑” and best min be “↓”

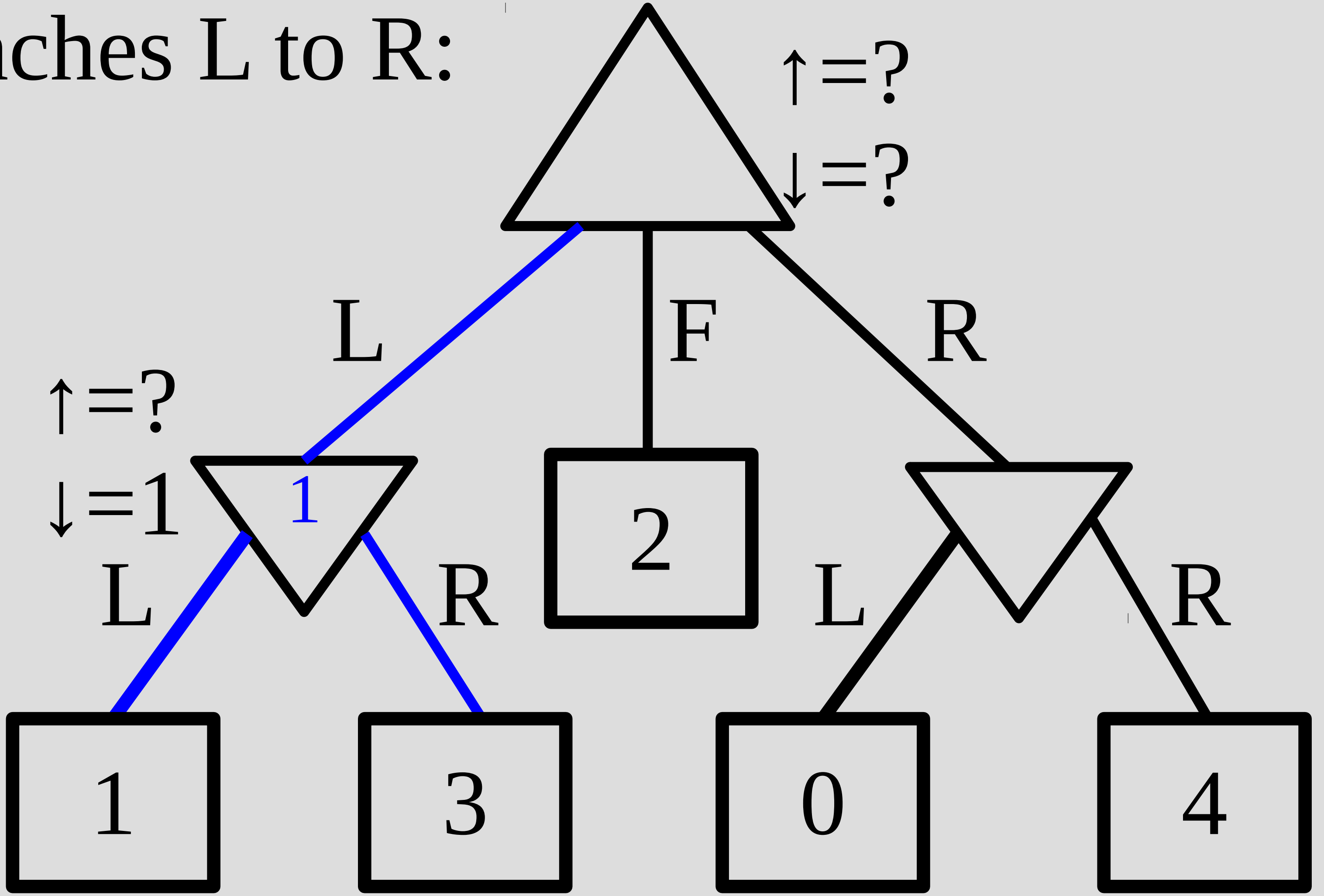
Branches L to R:



Alpha-beta pruning

Let best max be “↑” and best min be “↓”

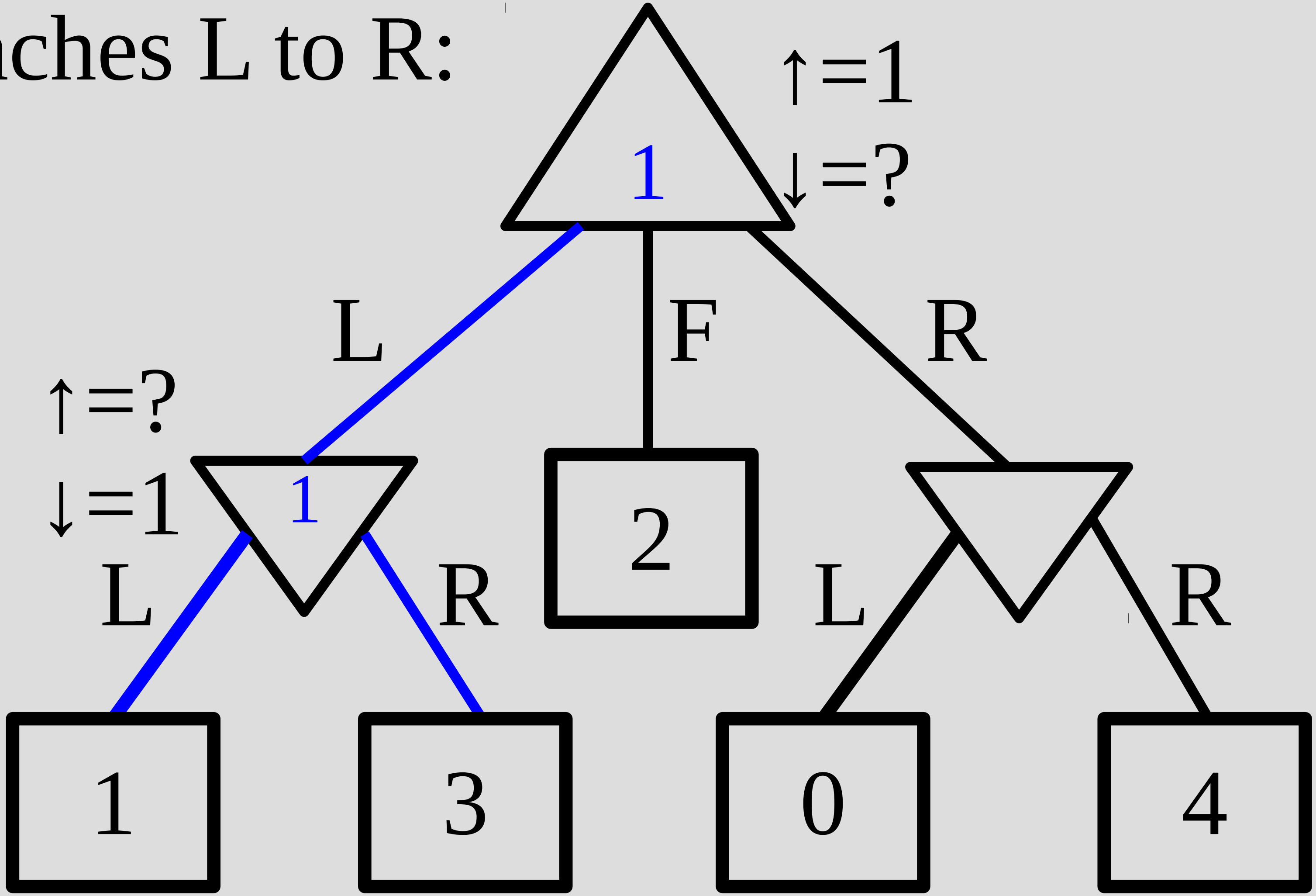
Branches L to R:



Alpha-beta pruning

Let best max be “↑” and best min be “↓”

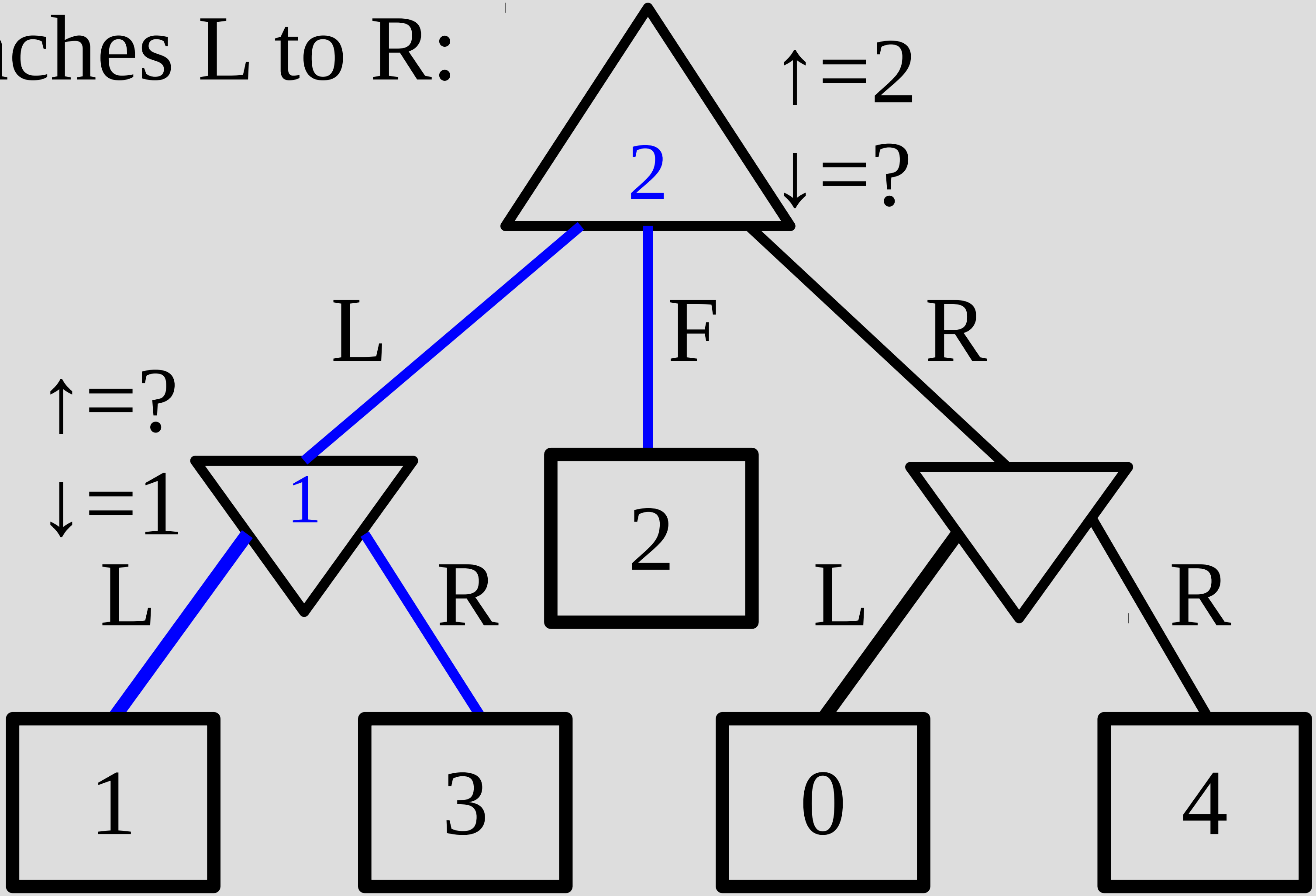
Branches L to R:



Alpha-beta pruning

Let best max be "↑" and best min be "↓"

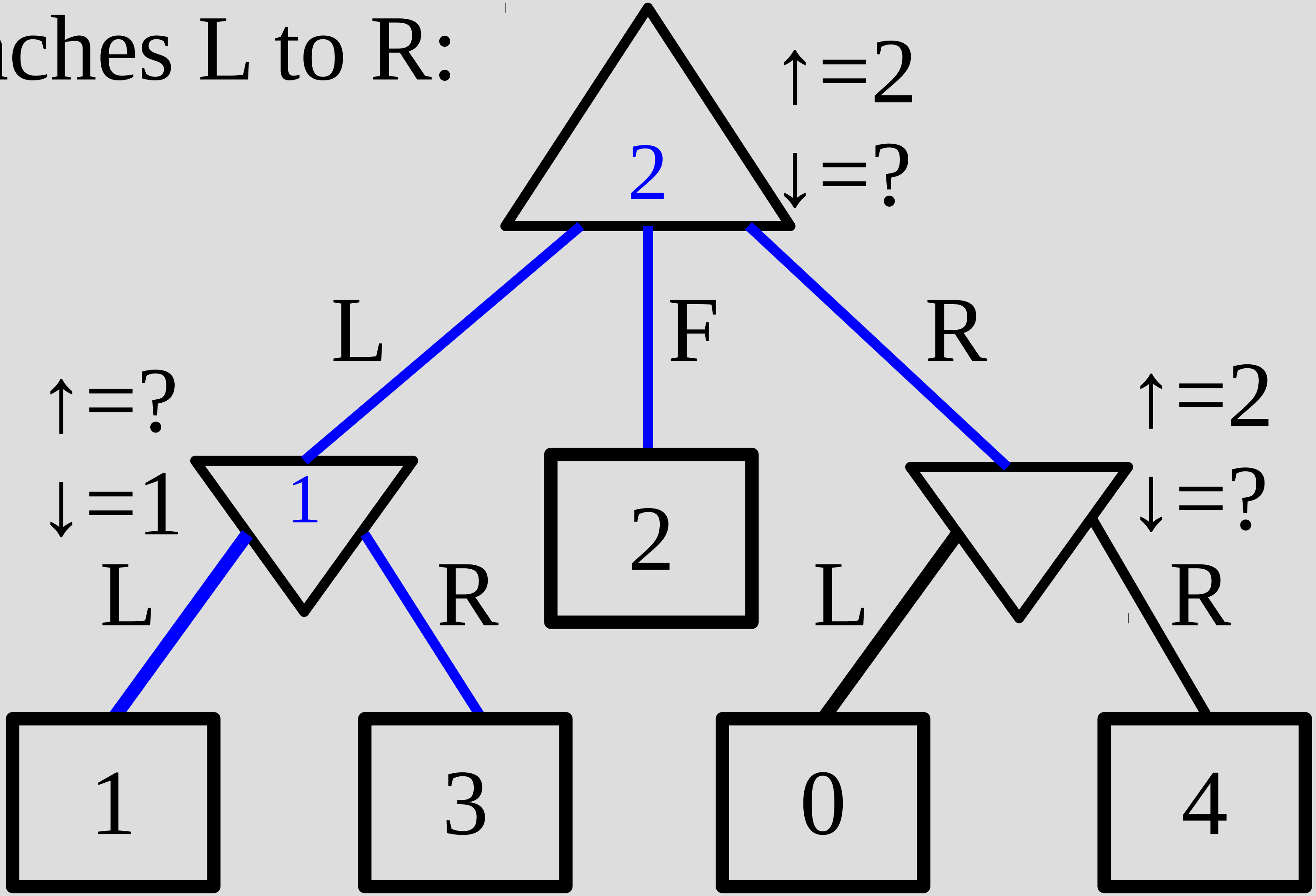
Branches L to R:



Alpha-beta pruning

Let best max be "↑" and best min be "↓"

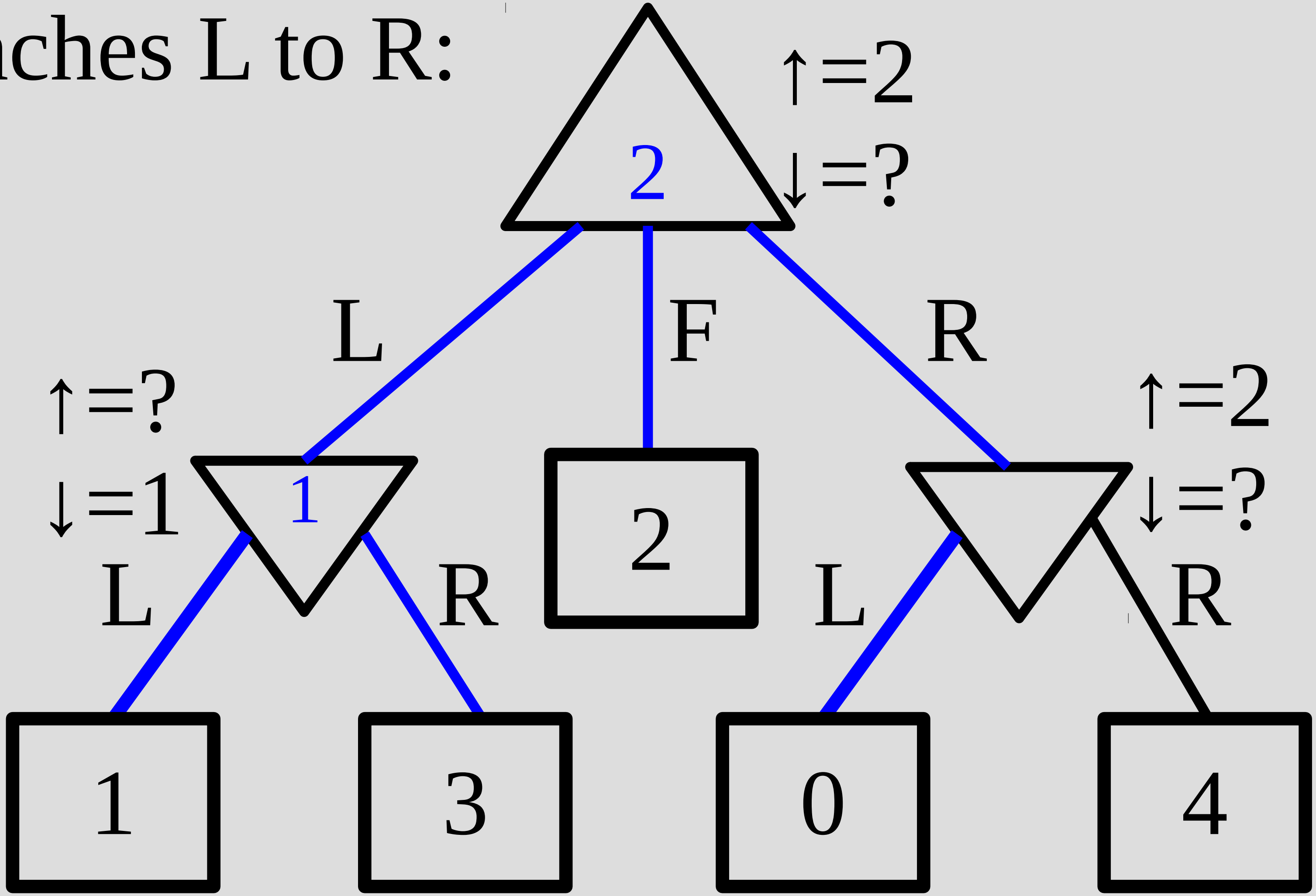
Branches L to R:



Alpha-beta pruning

Let best max be “↑” and best min be “↓”

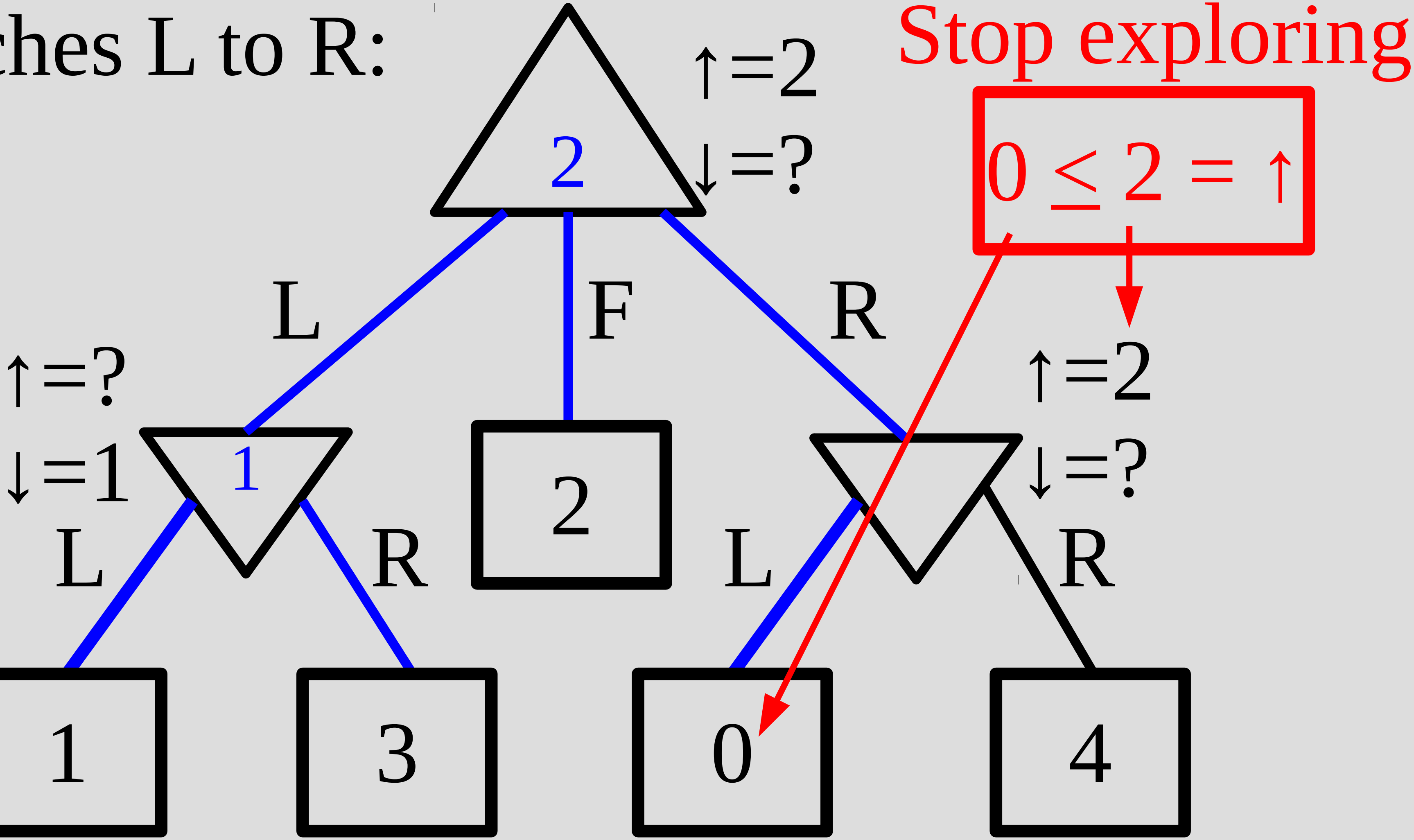
Branches L to R:



Alpha-beta pruning

Let best max be "↑" and best min be "↓"

Branches L to R:



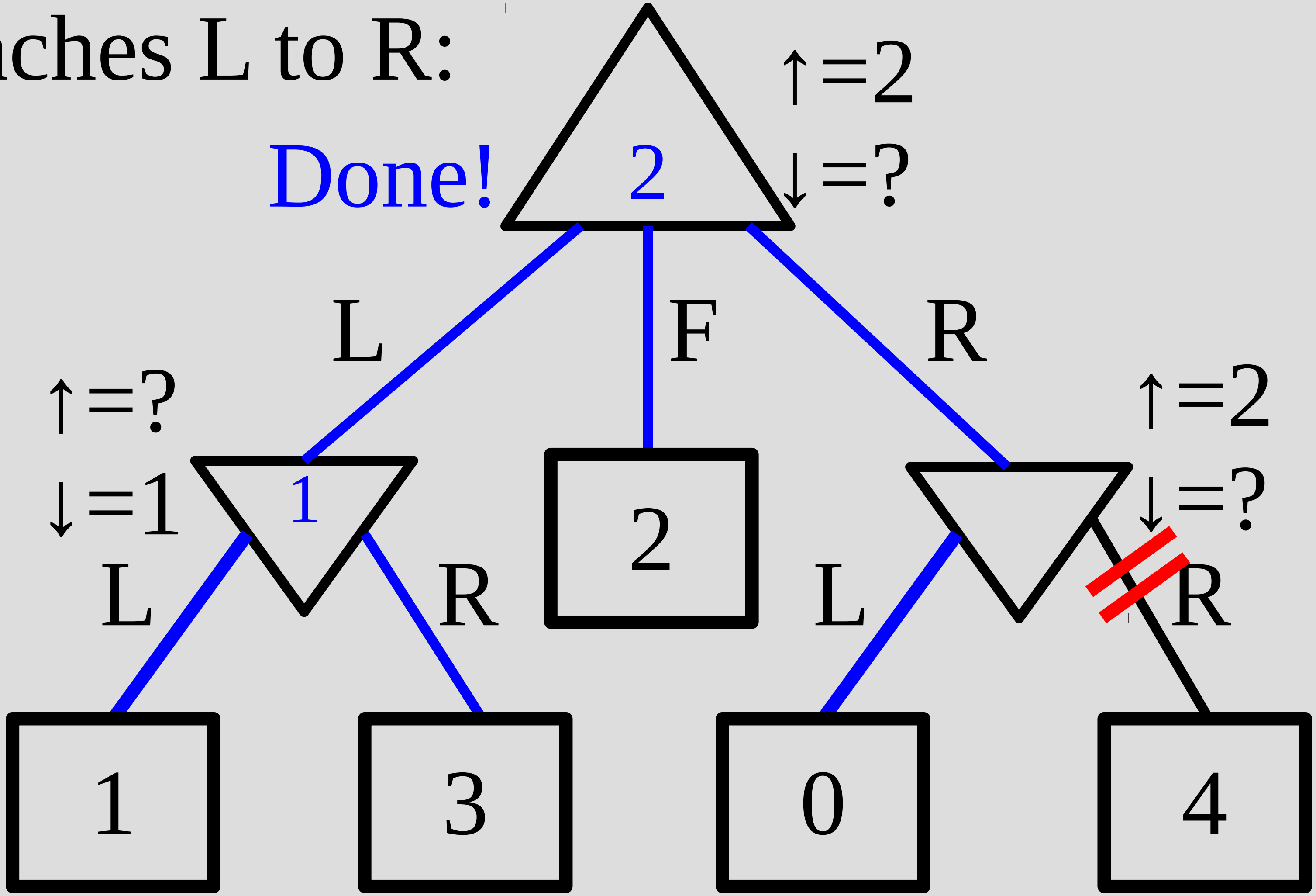
Stop exploring

0 ≤ 2 = ↑

Alpha-beta pruning

Let best max be "↑" and best min be "↓"

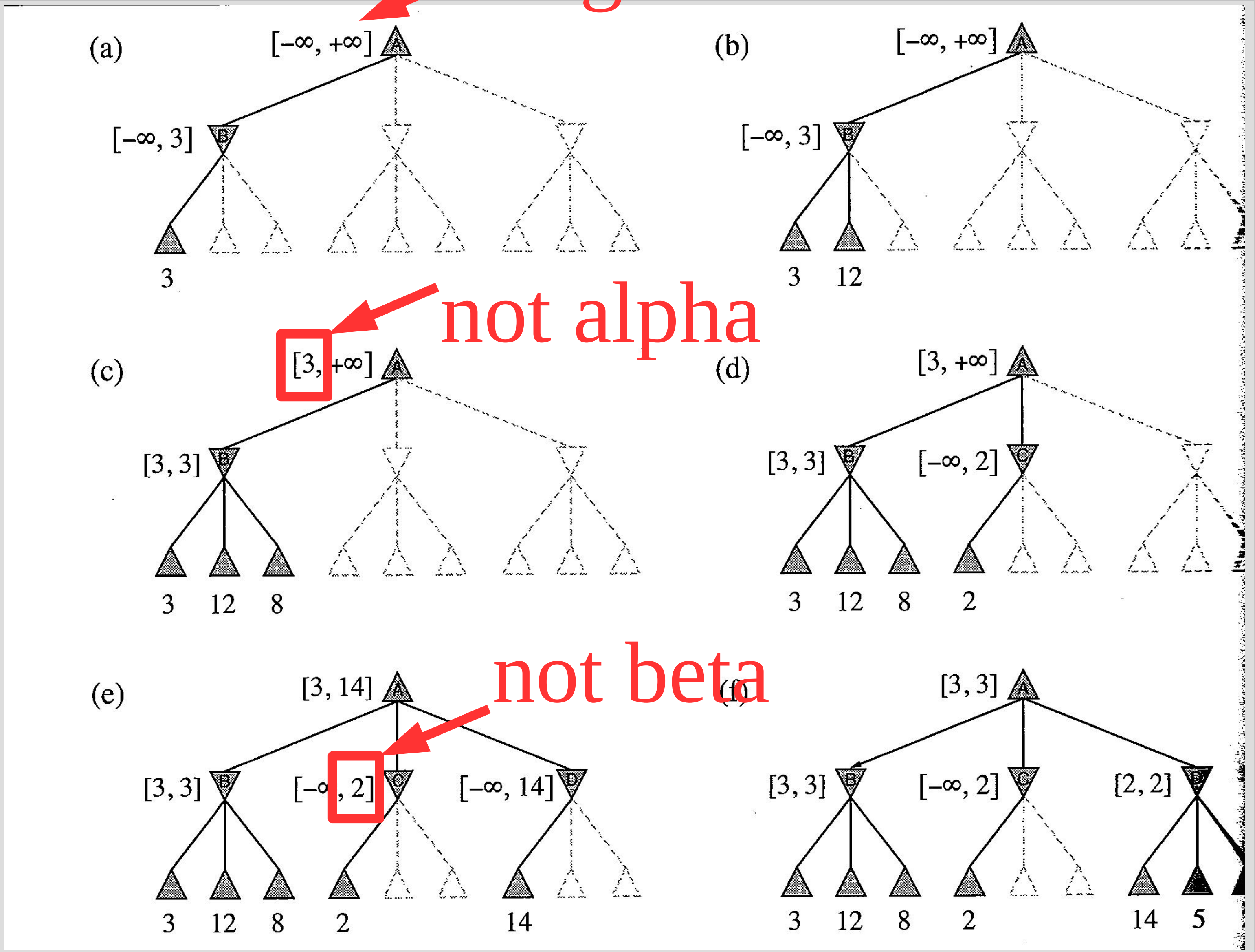
Branches L to R:



Alpha-beta pruning

\rantOn

I think the book is confusing about alpha-beta, especially Figure 5.5

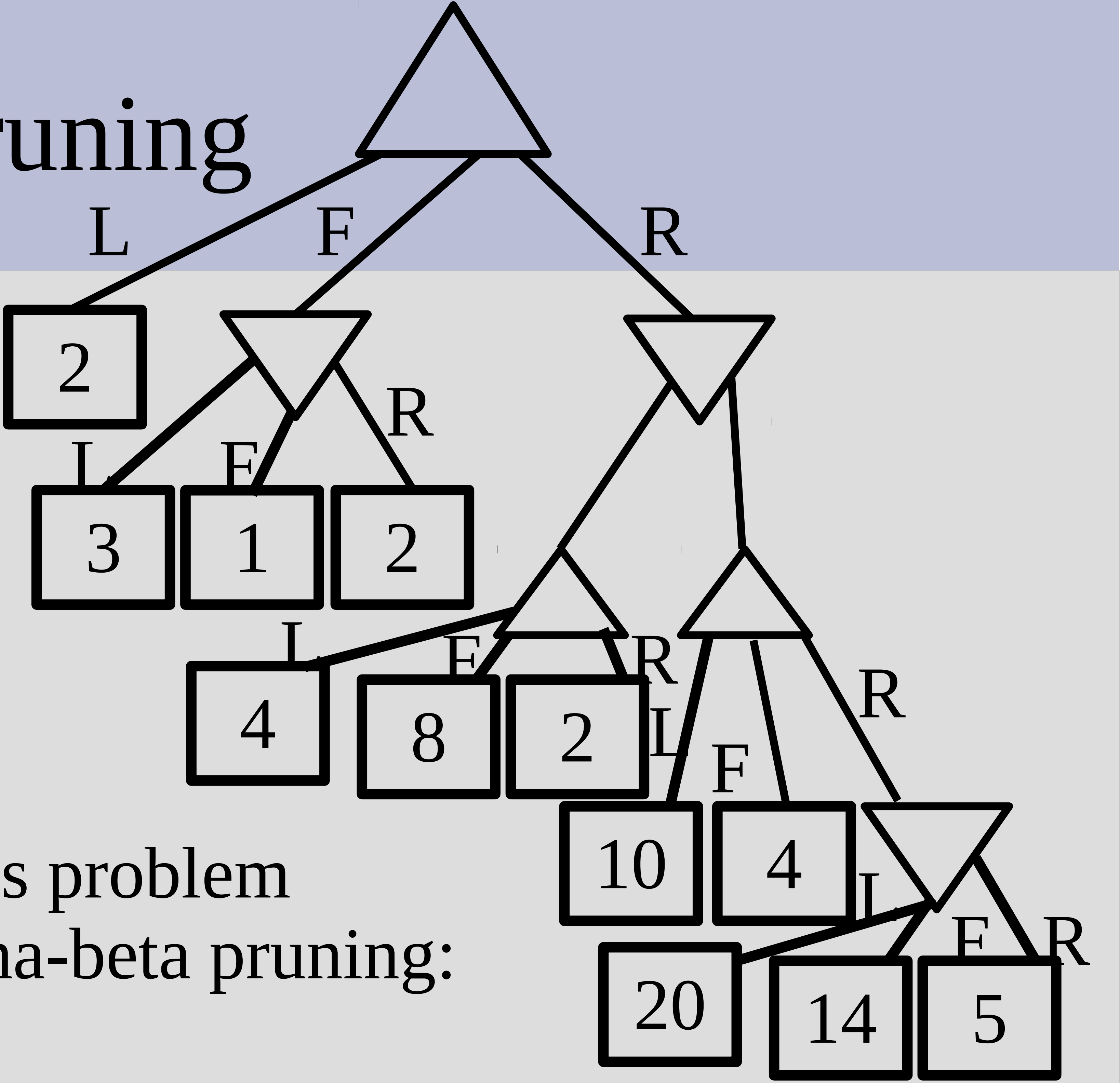


range for node

not alpha

not beta

αβ pruning



Solve this problem
with alpha-beta pruning:

Alpha-beta pruning

In general, alpha-beta pruning allows you to search to a depth 2d for the minimax search cost of depth d

So if minimax needs to find: $O(b^{m})$
Then, alpha-beta searches: $O(b^{m/2})$

This is exponentially better, but the worst case is the same as minimax

Alpha-beta pruning

Ideally you would want to put your best (largest for max, smallest for min) actions first

This way you can prune more of the tree as a min node stops more often for larger “best”

Obviously you do not know the best move, (otherwise why are you searching?) but some effort into guessing goes a long way (i.e. exponentially less states)

Side note:

In alpha-beta pruning, the heuristic for guess which move is best can be complex, as you can greatly effect pruning

While for A* search, the heuristic had to be very fast to be useful
(otherwise computing the heuristic would take longer than the original search)

Alpha-beta pruning

This rule of checking your parent's best/worst with the current value in the child only really works for two player games...

What about 3 player games?

3-player games

For more than two player games, you need to provide values at every state for all the players

When it is the player's turn, they get to pick the action that maximizes their own value the most

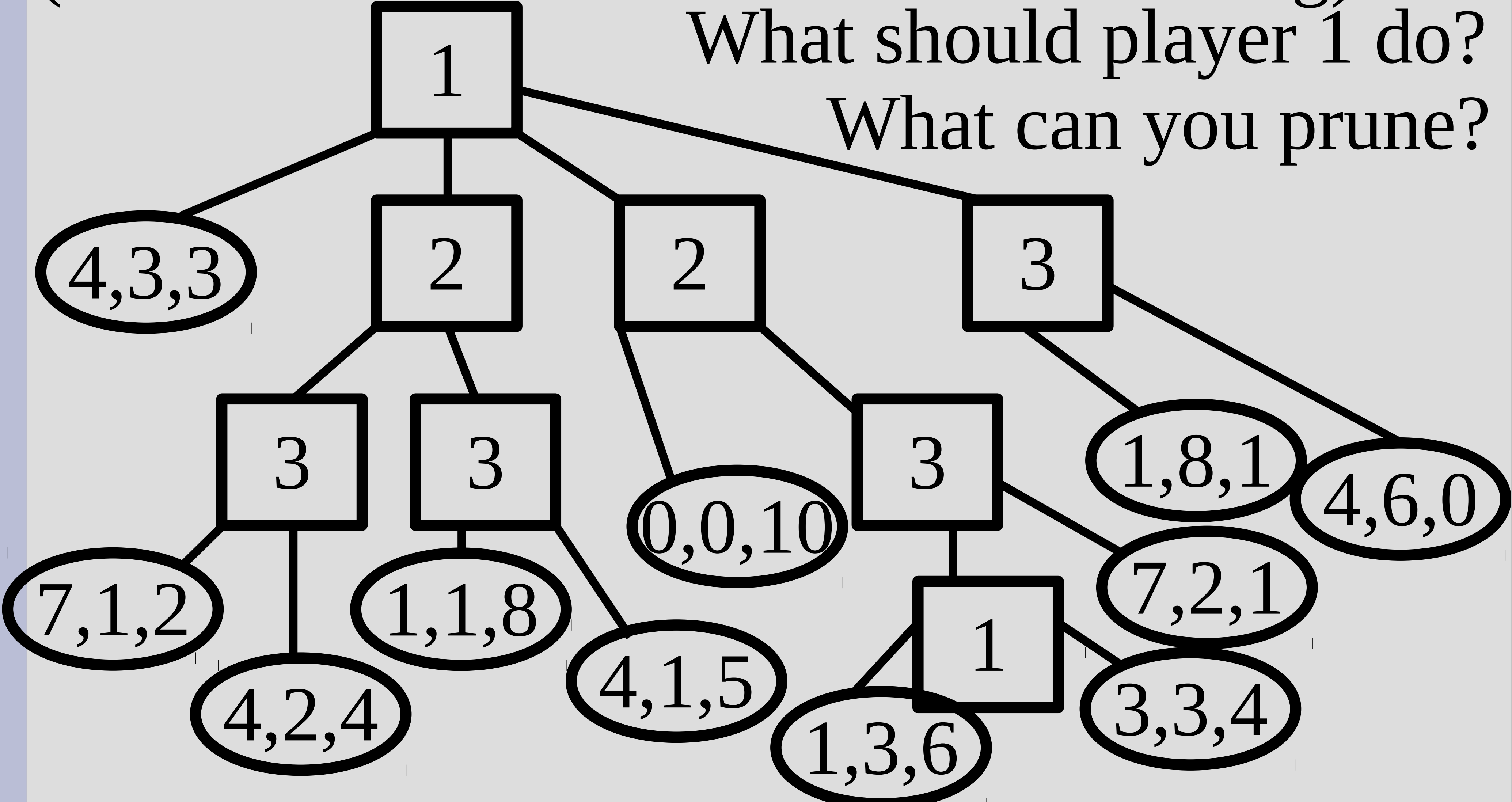
(We will assume each agent is greedy and only wants to increase its own score... more on this next time)

3-player games

(The node number shows who is max-ing)

What should player 1 do?

What can you prune?



3-player games

How would you do alpha-beta pruning in a 3-player game?

3-player games

How would you do alpha-beta pruning in a 3-player game?

TL;DR: Not easily

(also you cannot prune at all if there is no range on the values even in a zero sum game)

This is because one player could take a very low score for the benefit of the other two

Mid-state evaluation

So far we assumed that you have to reach a terminal state then propagate backwards (with possibly pruning)

More complex games (Go or Chess) it is hard to reach the terminal states as they are so far down the tree (and large branching factor)

Instead, we will estimate the value minimax would give without going all the way down

Mid-state evaluation

By using mid-state evaluations (not terminal) the “best” action can be found quickly

These mid-state evaluations need to be:

1. Based on current state only
2. Fast (and not just a recursive search)
3. Accurate (represents correct win/loss rate)

The quality of your final solution is highly correlated to the quality of your evaluation

Mid-state evaluation

For searches, the heuristic only helps you find the goal faster (but A* will find the best solution as long as the heuristic is admissible)

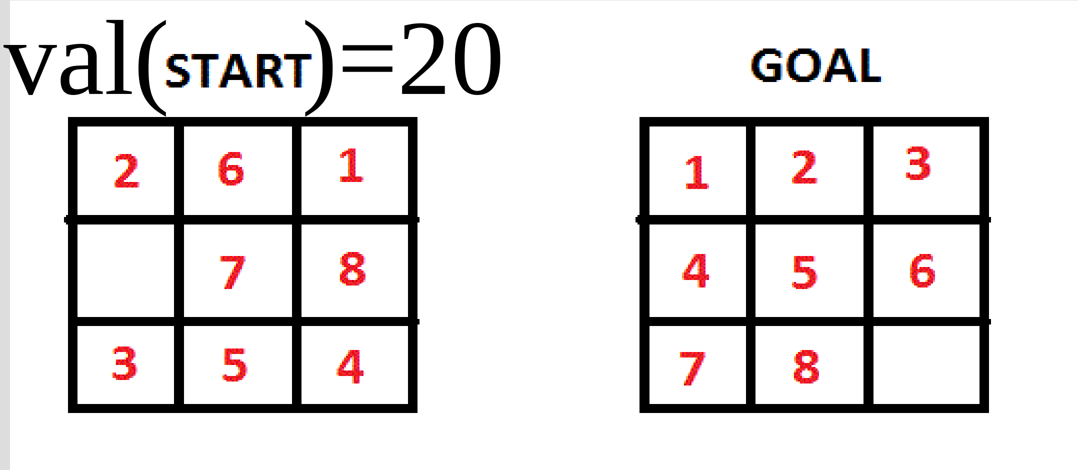
There is no concept of "admissible" mid-state evaluations... and there is almost no guarantee that you will find the best/optimal solution

For this reason we only apply mid-state evals to problems that we cannot solve optimally

Mid-state evaluation

A common mid-state evaluation adds features of the state together

(we did this already for a heuristic...)



eval(START)=20

2	6	1
	7	8
3	5	4

GOAL

1	2	3
4	5	6
7	8	

We summed the distances to the correct spots for all numbers

Mid-state evaluation

We then minimax (and prune) these mid-state evaluations as if they were the correct values

You can also weight features (i.e. getting the top row is more important in 8-puzzle)

A simple method in chess is to assign points for each piece: pawn=1, knight=4, queen=9... then sum over all pieces you have in play

Mid-state evaluation

What assumptions do you make if you use a weighted sum?

Mid-state evaluation

What assumptions do you make if you use a weighted sum?

A: The factors are independent
(non-linear accumulation is common if the relationships have a large effect)

For example, a rook & queen have a synergy bonus for being together is non-linear, so queen=9, rook=5... but queen&rook = 16

Mid-state evaluation

There is also an issue with how deep should we look before making an evaluation?

Mid-state evaluation

There is also an issue with how deep should we look before making an evaluation?

A fixed depth? Problems if child's evaluation is overestimate and parent underestimate (or visa versa)

Ideally you would want to stop on states where the mid-state evaluation is most accurate

Mid-state evaluation

Mid-state evaluations also favor actions that “put off” bad results (i.e. they like stalling)

In go this would make the computer use up ko threats rather than give up a dead group

By evaluating only at a limited depth, you reward the computer for pushing bad news beyond the depth (but does not stop the bad news from eventually happening)

Mid-state evaluation

It is not easy to get around these limitations:

1. Push off bad news
2. How deep to evaluate?

A better mid-state evaluation can help compensate, but they are hard to find

They are normally found by mimicking what expert human players do, and there is no systematic good way to find one

Forward pruning

You can also use mid-state evaluations for alpha-beta type pruning

However as these evaluations are estimates, you might prune the optimal answer if the heuristic is not perfect (which it won't be)

In practice, this prospective pruning is useful as it allows you to prioritize spending more time exploring hopeful parts of the search tree

Forward pruning

You can also save time searching by using "expert knowledge" about the problem

For example, in both Go and Chess the start of the game has been very heavily analyzed over the years

There is no reason to redo this search every time at the start of the game, instead we can just look up the "best" response

Random games

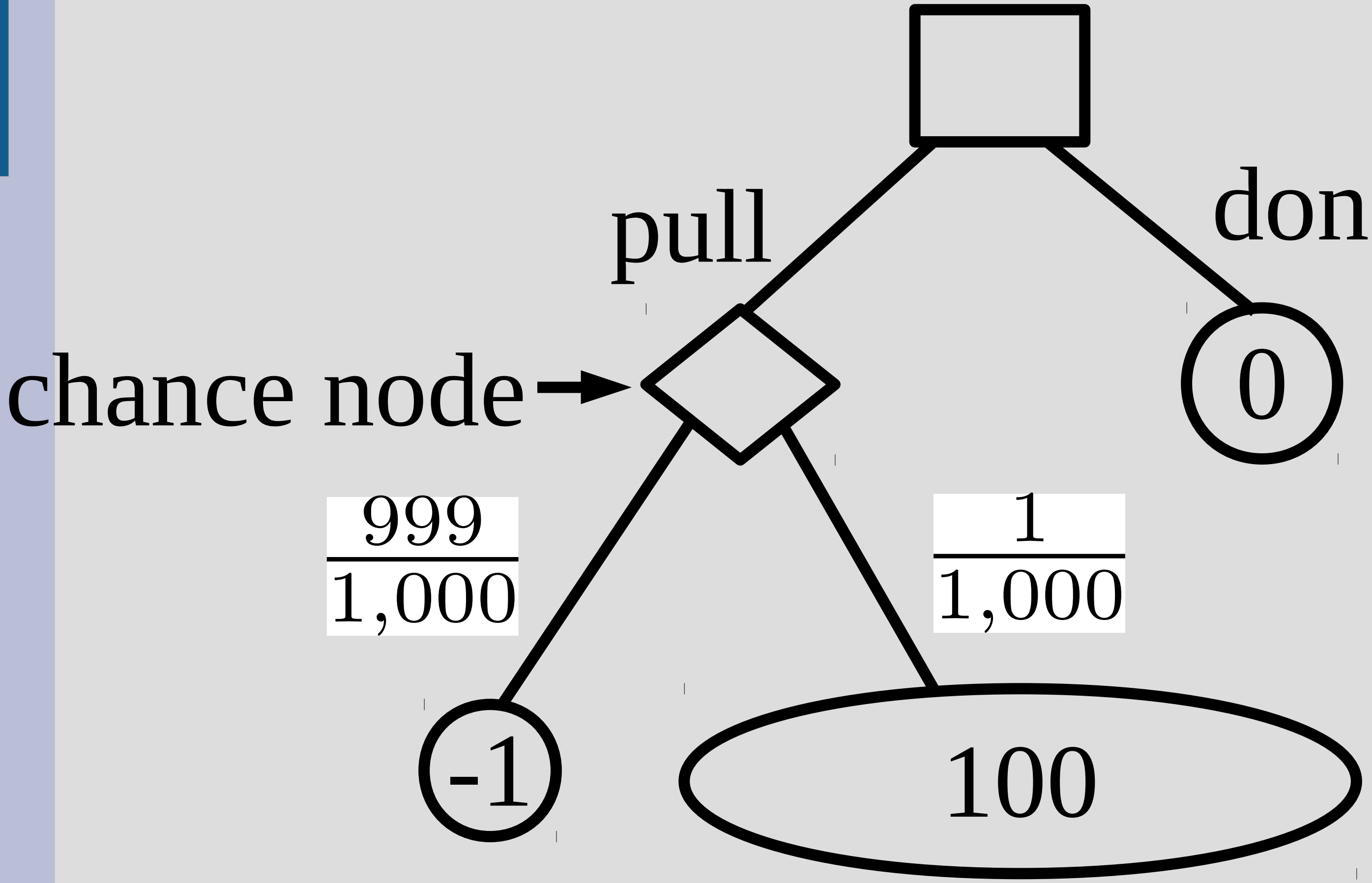
If we are playing a “game of chance”, we can add chance nodes to the search tree

Instead of either player picking max/min, it takes the expected value of its children

This expected value is then passed up to the parent node which can choose to min/max this chance (or not)

Random games

Here is a simple slot machine example:



pull

don't pull

chance node

0

$\frac{999}{1,000}$

$\frac{1}{1,000}$

-1

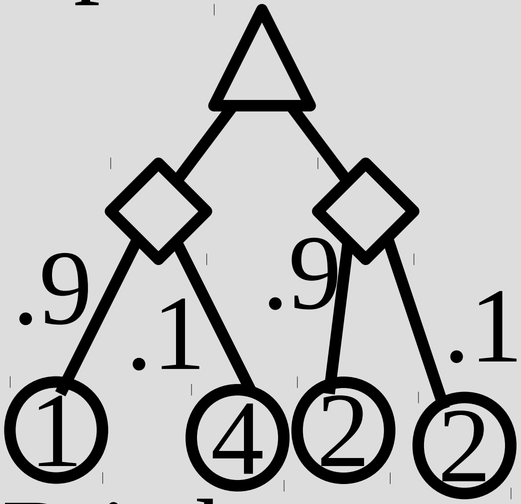
100

$$V(\text{chance}) = -1 \cdot \frac{999}{1,000} + 100 \cdot \frac{1}{1,000} = -0.899$$

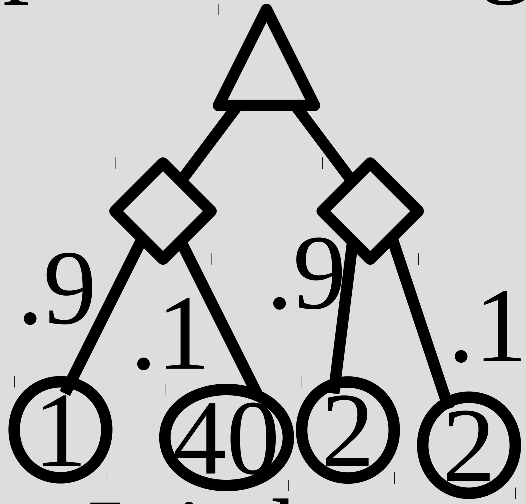
Random games

You might need to modify your mid-state evaluation if you add chance nodes

Minimax just cares about the largest/smallest, but expected value is an implicit average:



R is better



L is better

Random games

Some partially observable games (i.e. card games) can be searched with chance nodes

As there is a high degree of chance, often it is better to just assume full observability (i.e. you know the order of cards in the deck)

Then find which actions perform best over all possible chance outcomes (i.e. all possible deck orderings)

Random games

For example in blackjack, you can see what cards have been played and a few of the current cards in play

You then compute all possible decks that could lead to the cards in play (and used cards)

Then find the value of all actions (hit or stand) averaged over all decks (assumed equal chance of possible decks happening)

Random games

If there are too many possibilities for all the chance outcomes to “average them all”,
you can sample

This means you can search the chance-tree and just randomly select outcomes (based on probabilities) for each chance node

If you have a large number of samples, this should converge to the average

MCTS

This idea of sampling a limited part of the tree to estimate values is common and powerful

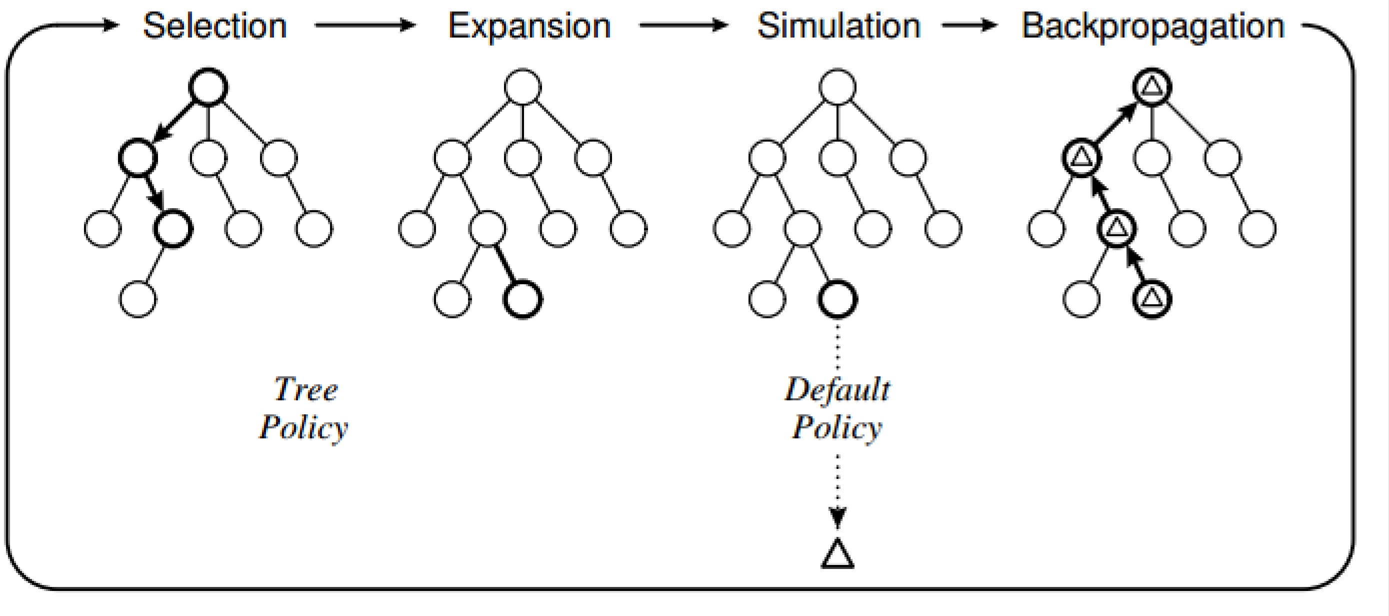
In fact, in monte-carlo tree search there are no mid-state evaluations, just samples of terminal states

This means you do not need to create a good mid-state evaluation function, but instead you assume sampling is effective (might not be so)

MCTS

MCTS has four steps:

1. Find the action which looks best (selection)
2. Add this new action sequence to a tree
3. Play randomly until over
4. Update how good this choice was



MCTS

How to find which actions are “good”?

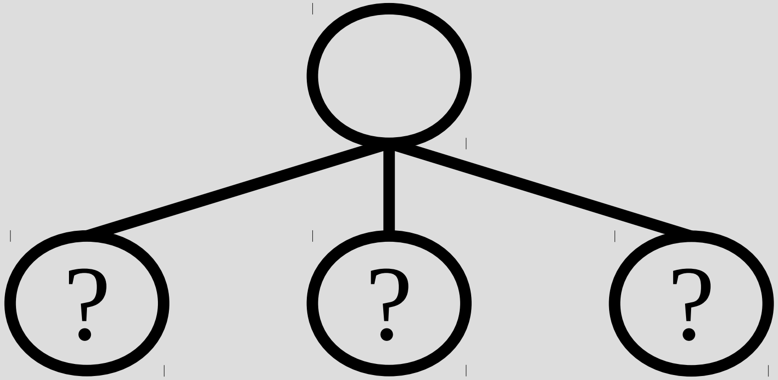
The “Upper Confidence Bound applied to Trees” UCT is commonly used:

$$\max\left(\frac{win(n)}{times(n)} + \sqrt{\frac{2\ln TotalTimes}{times(n)}}\right)$$

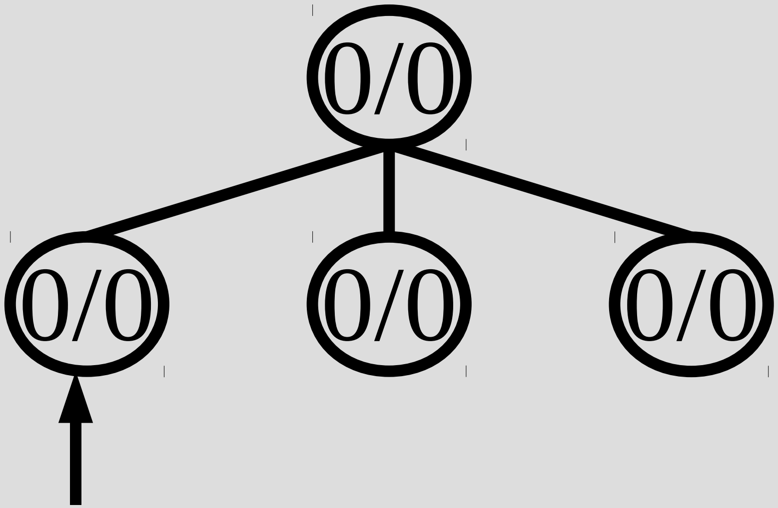
This ensures a trade off between checking branches you haven't explored much and exploring hopeful branches

(https://www.youtube.com/watch?v=Fbs4lnGLS8M)

MCTS

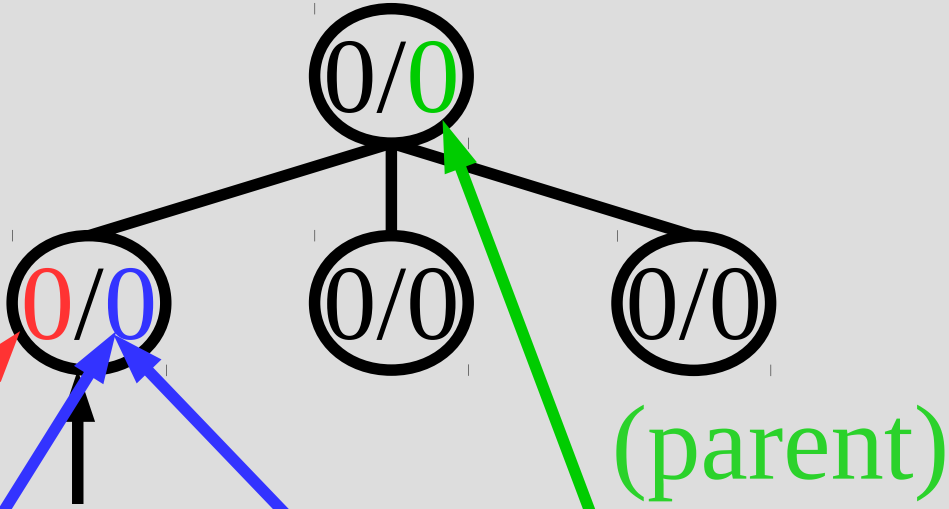


MCTS



$$
\begin{aligned}
&\frac{win(n)}{times(n)} + \sqrt{\frac{2\ln TotalTimes}{times(n)}}\\
&= \frac{0}{0} + \sqrt{\frac{2ln0}{0}}\\
&= \infty
\end{aligned}
$$

MCTS

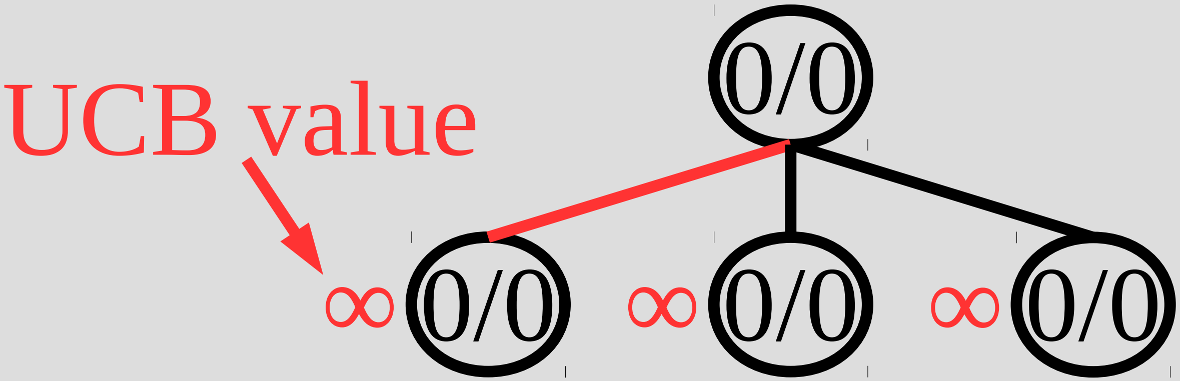


$$\frac{win(n)}{times(n)} + \sqrt{\frac{2\ln TotalTimes}{times(n)}}$$

$$= \frac{0}{0} + \sqrt{\frac{2ln0}{0}}$$

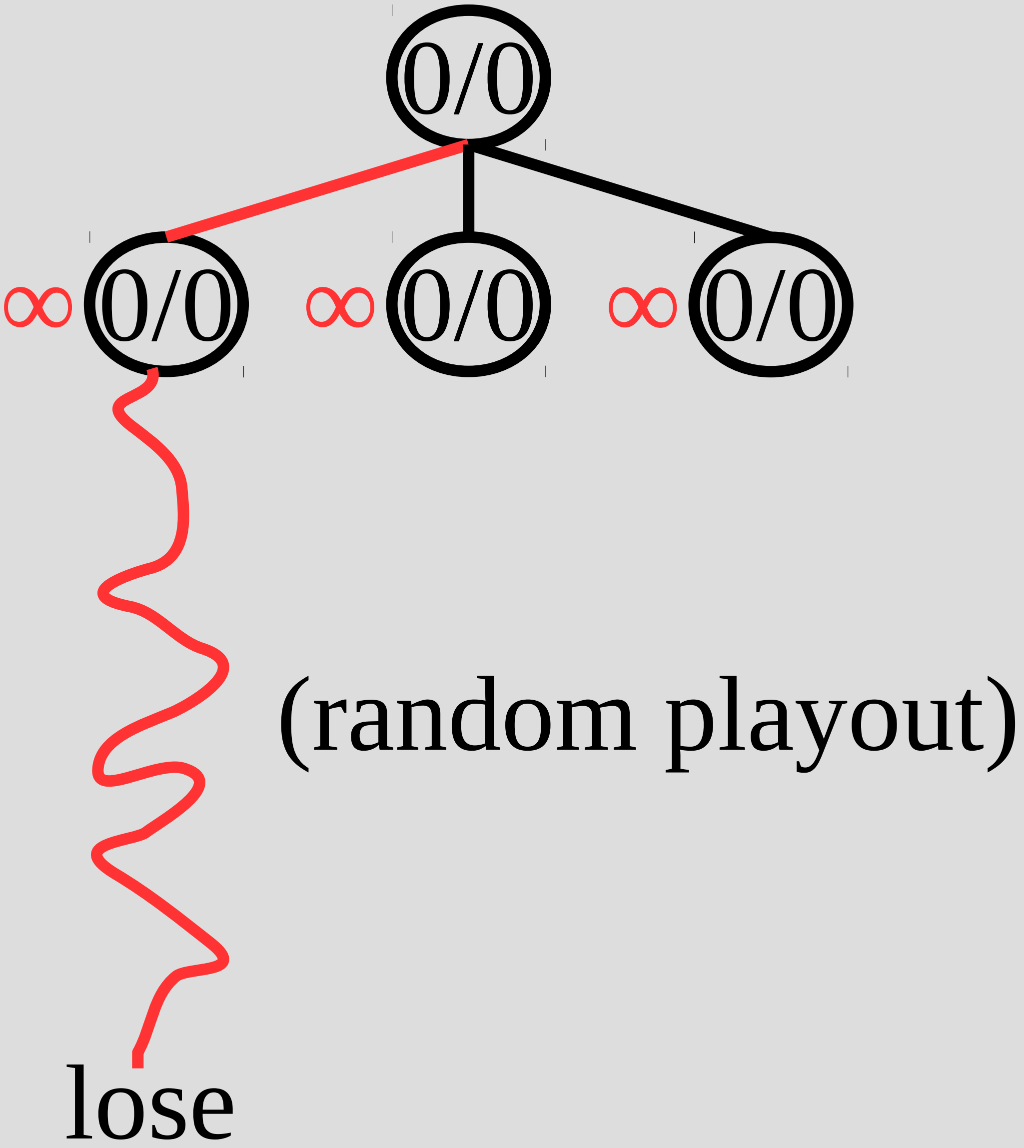
$$= \infty$$

MCTS

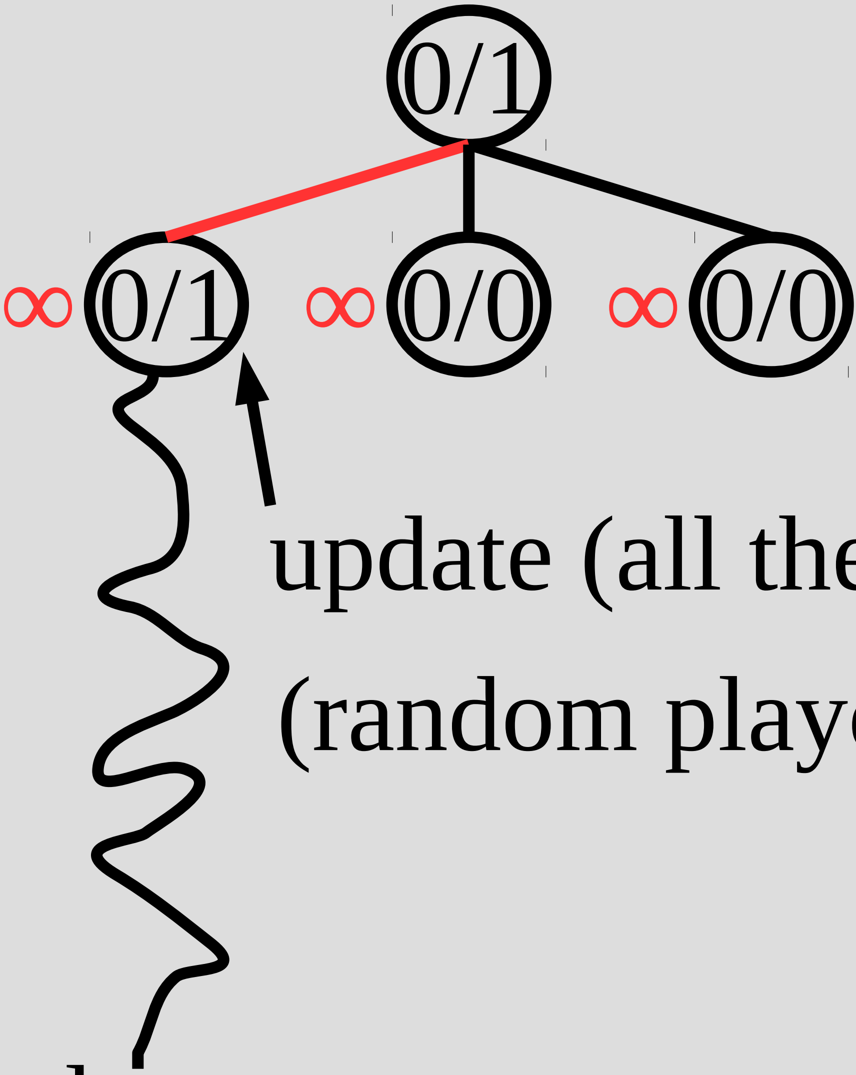


Pick max (I'll pick left-most)

MCTS



MCTS

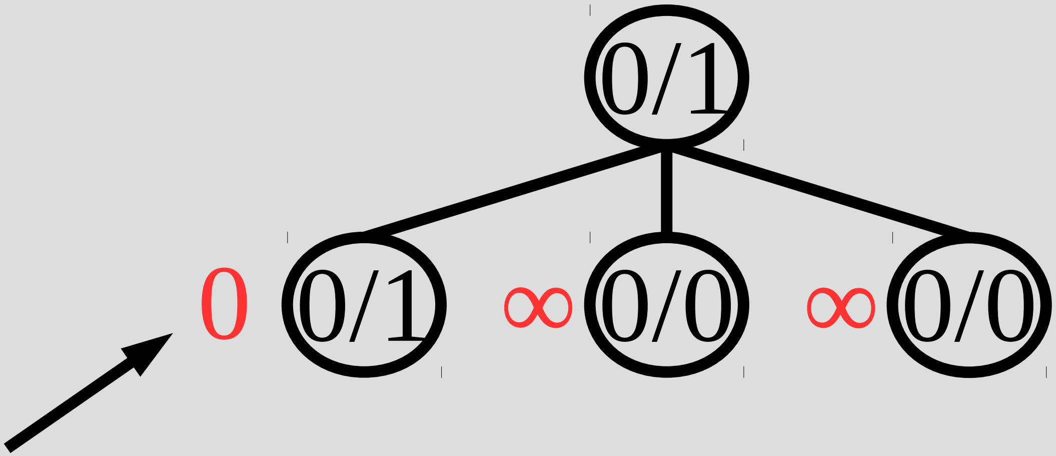


update (all the way to root)

(random playout)

lose

MCTS



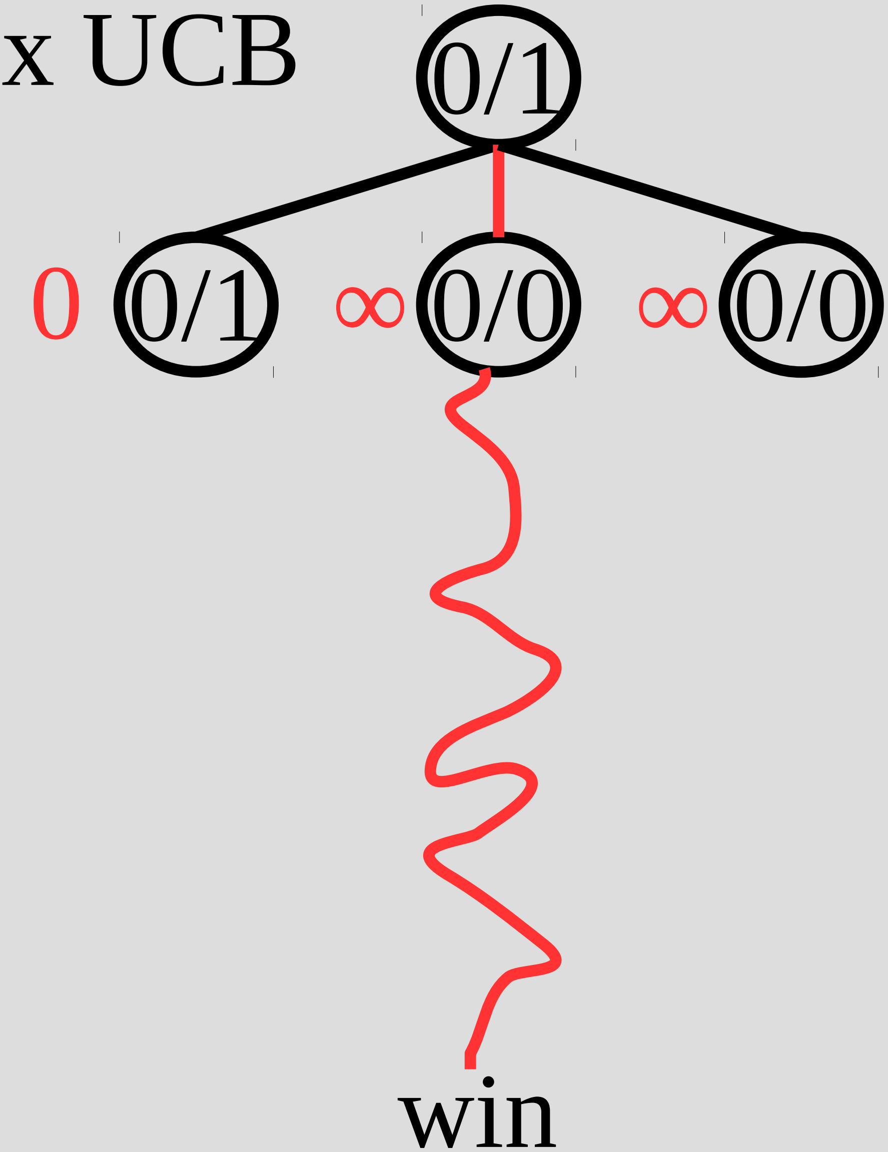
0/1

0 0/1 ∞ 0/0 ∞ 0/0

update UCB values (all nodes)

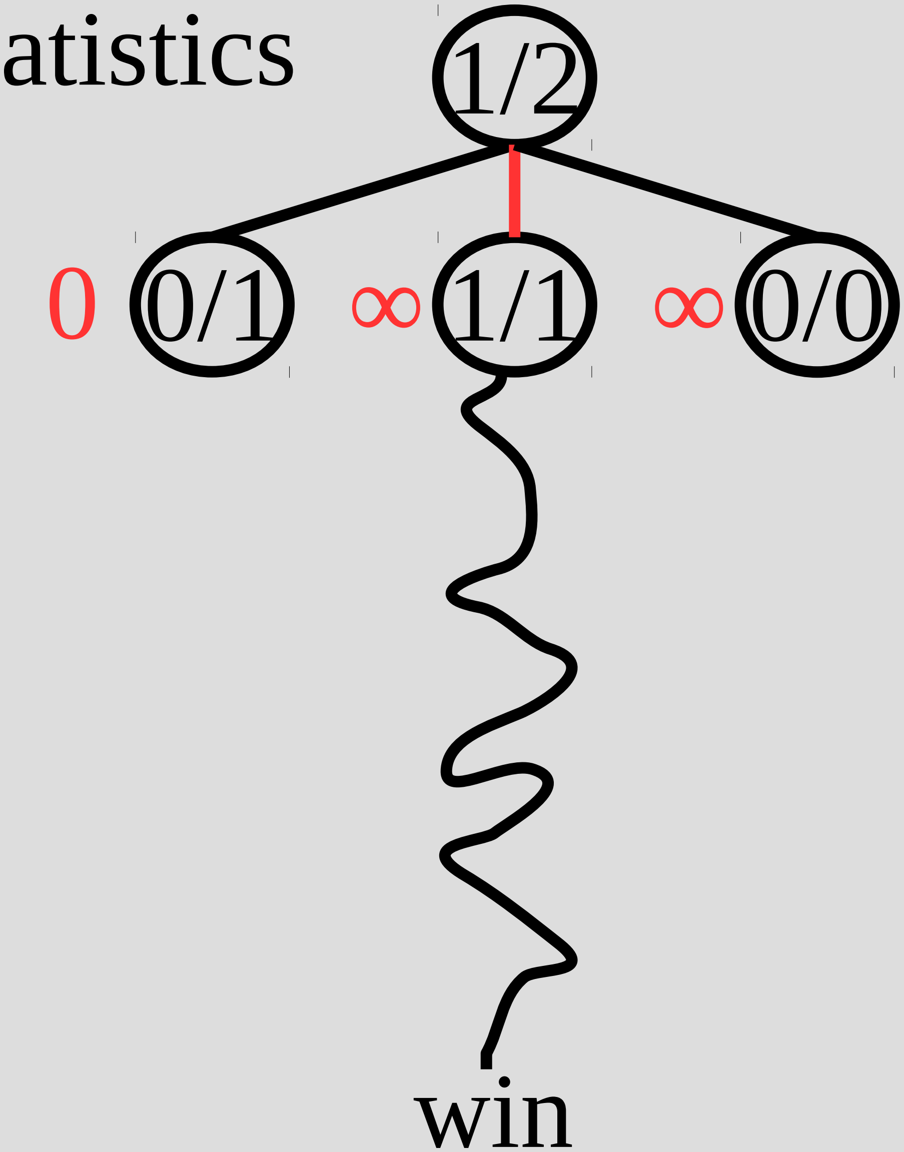
MCTS

select max UCB
& rollout



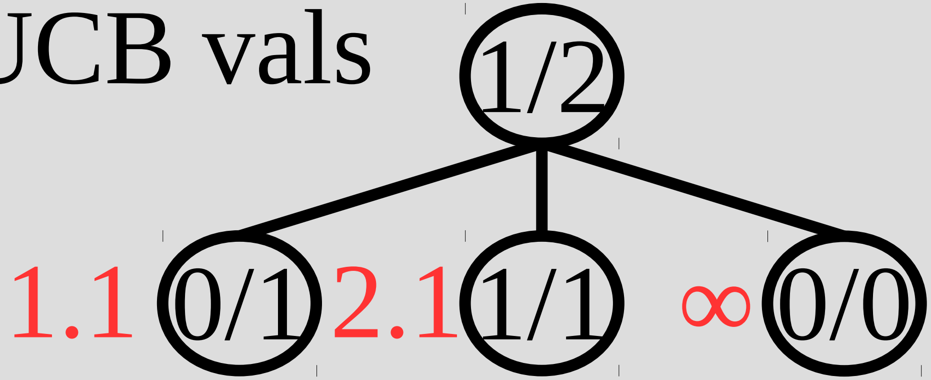
MCTS

update statistics



MCTS

update UCB vals

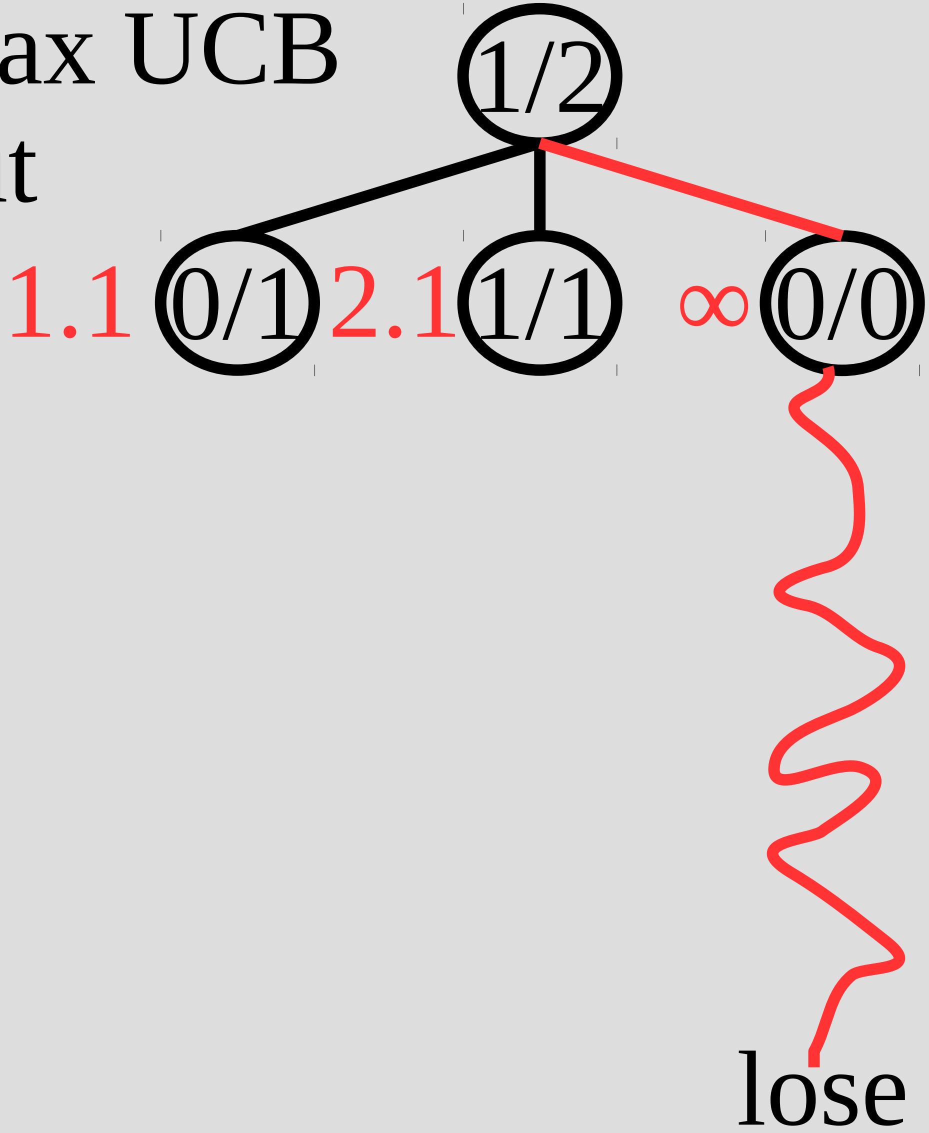


1/2

1.1 0/1 2.1 1/1 ∞ 0/0

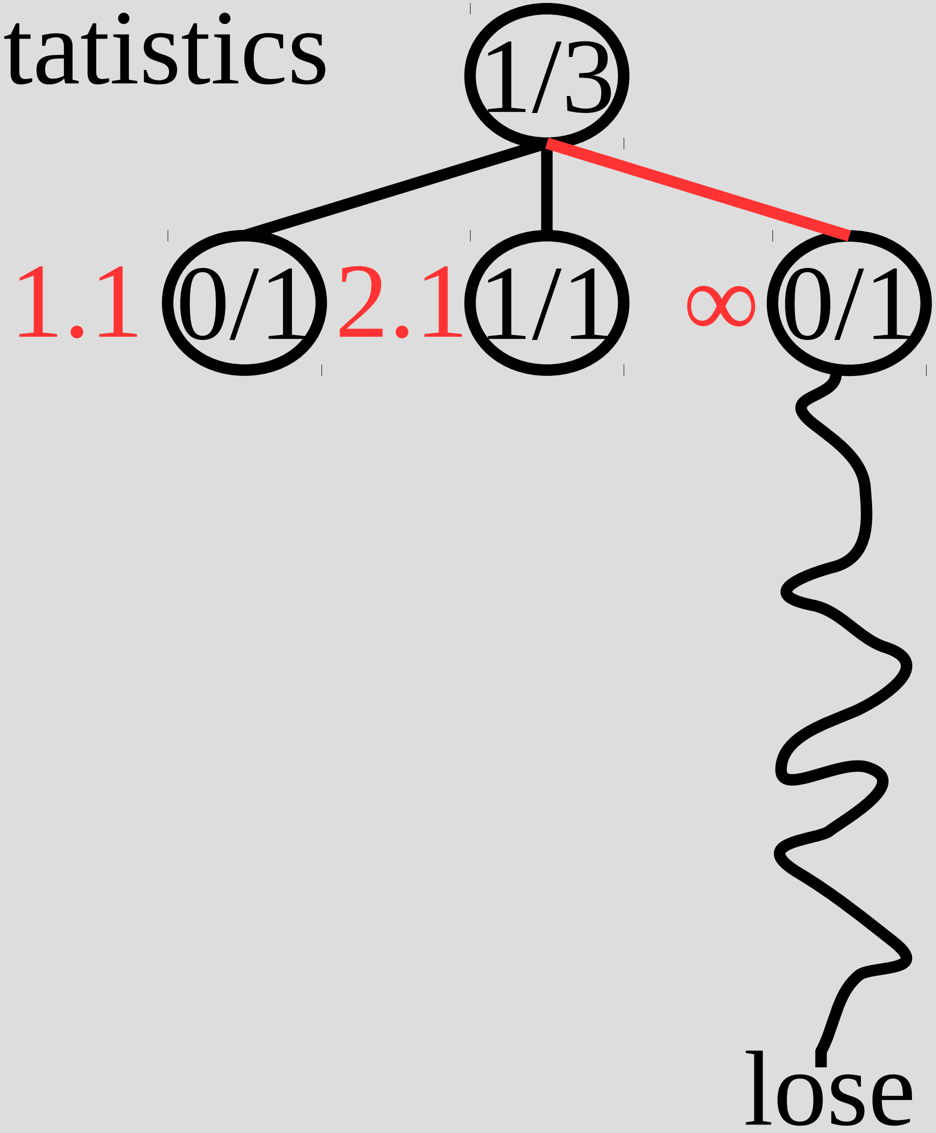
MCTS

select max UCB
& rollout



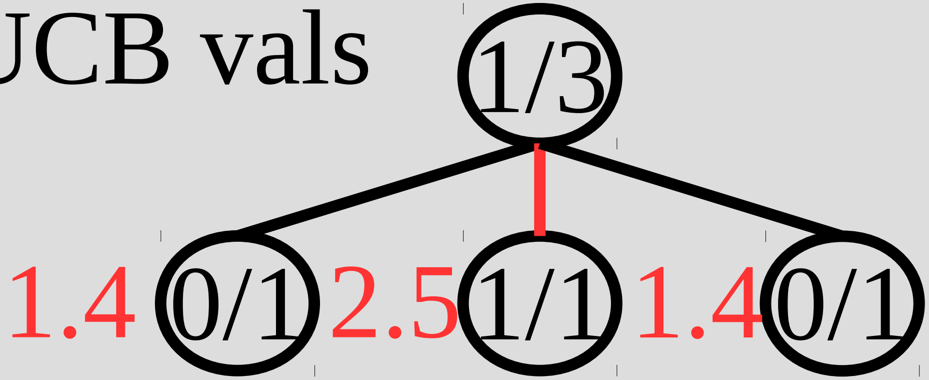
MCTS

update statistics



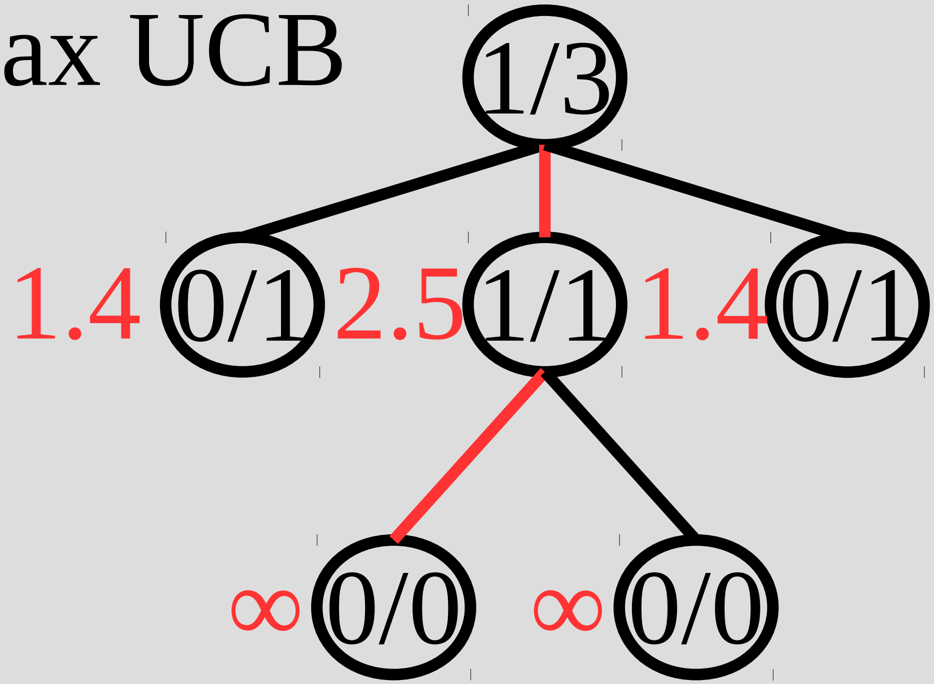
MCTS

update UCB vals



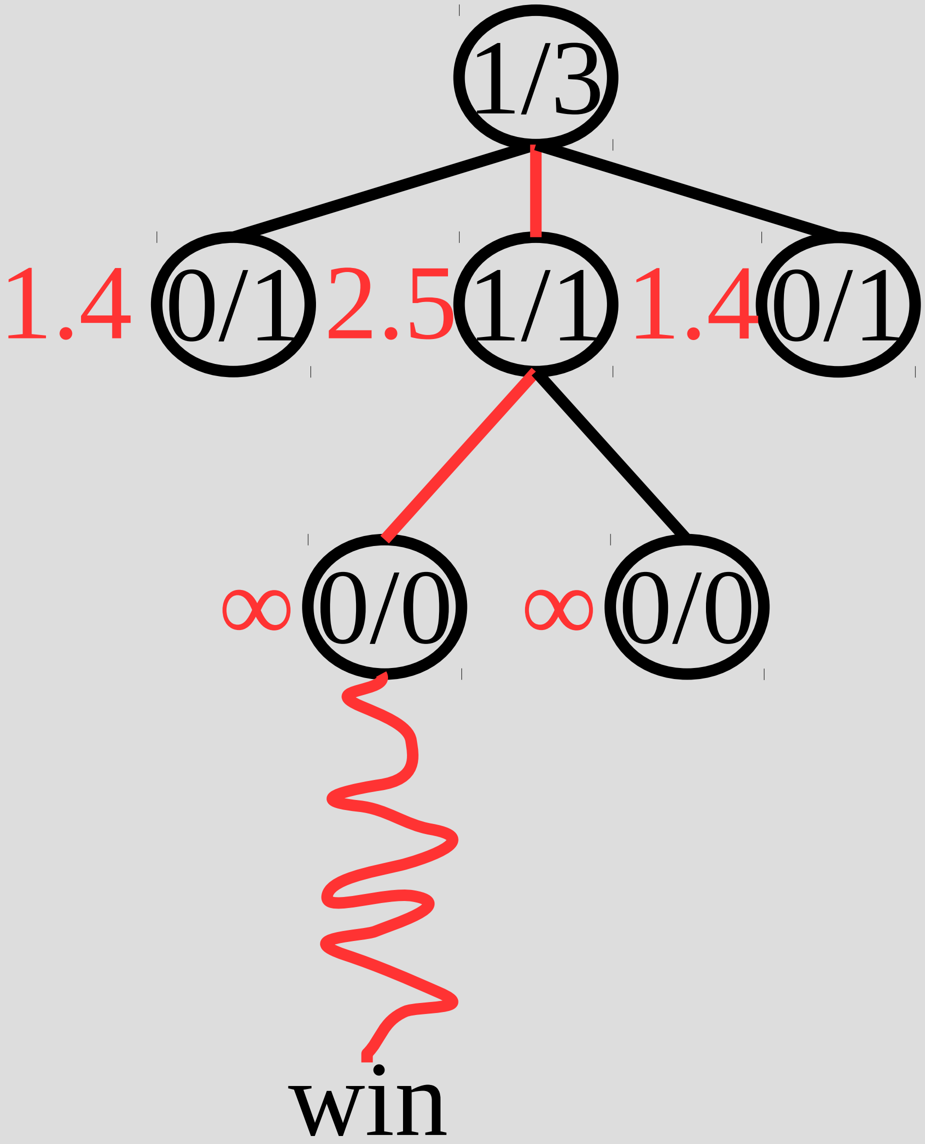
MCTS

select max UCB



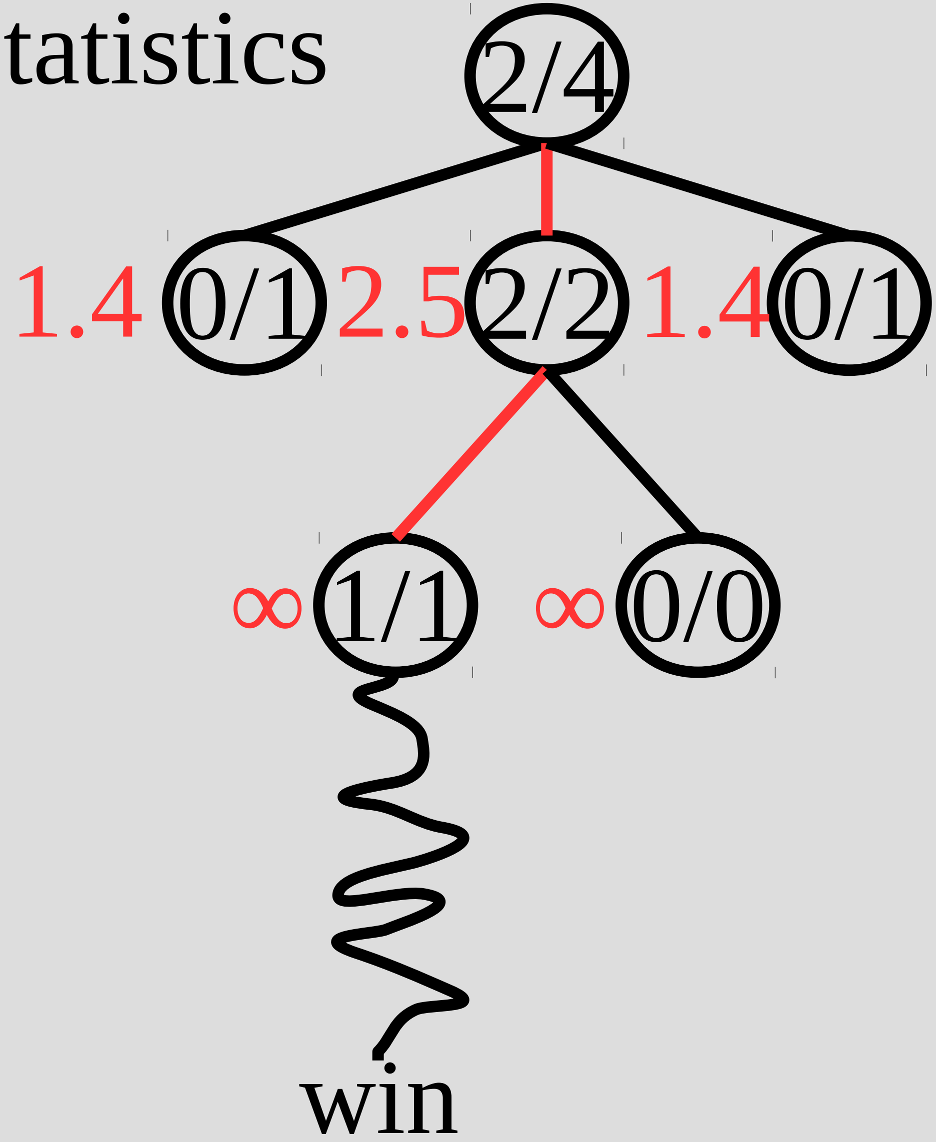
MCTS

rollout



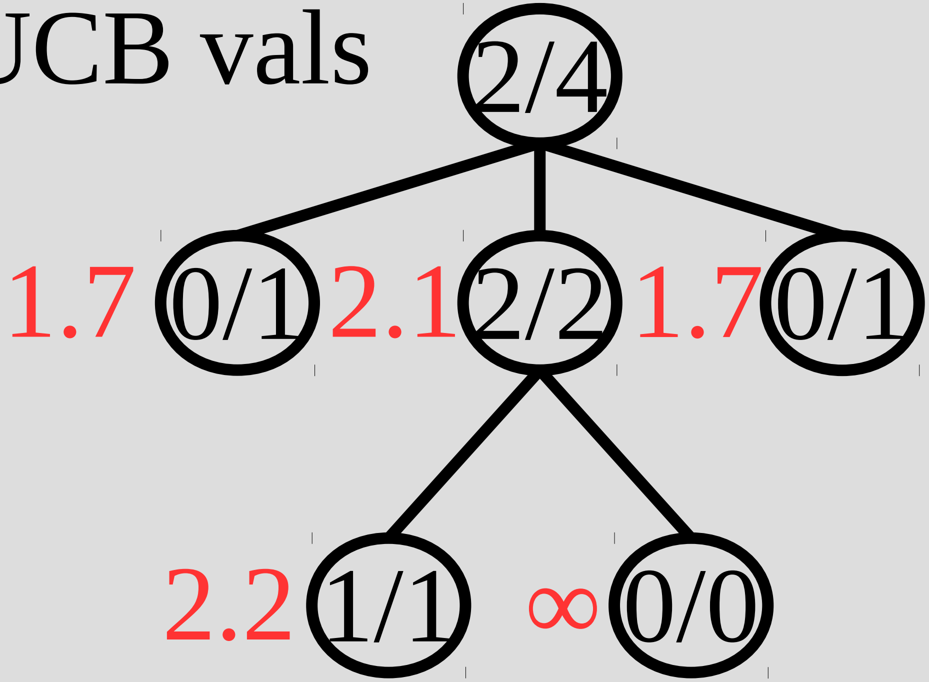
MCTS

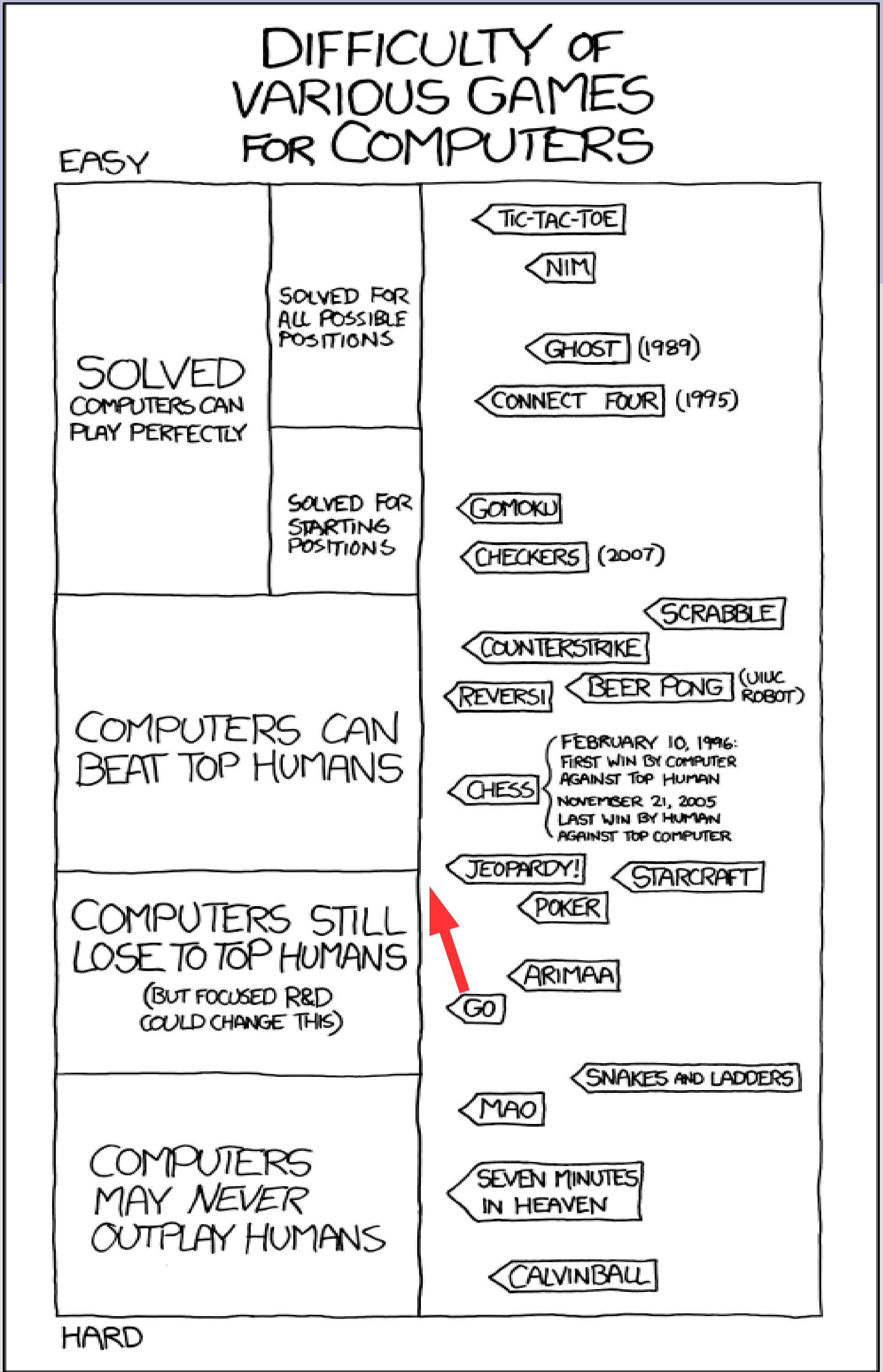
update statistics



MCTS

update UCB vals




DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS
EASY
SOLVED
COMPUTERS CAN PLAY PERFECTLY
SOLVED FOR ALL POSSIBLE POSITIONS
SOLVED FOR STARTING POSITIONS
COMPUTERS CAN BEAT TOP HUMANS
COMPUTERS STILL LOSE TO TOP HUMANS
(BUT FOCUSED R&D COULD CHANGE THIS)
COMPUTERS MAY NEVER OUTPLAY HUMANS
HARD
TIC-TAC-TOE
NIM
GHOST (1989)
CONNECT FOUR (1995)
GOMOKU
CHECKERS (2007)
SCRABBLE
COUNTERSTRIKE
REVERSI
BEER PONG (UIUC ROBOT)
CHESS
FEBRUARY 10, 1996: FIRST WIN BY COMPUTER AGAINST TOP HUMAN
NOVEMBER 21, 2005 LAST WIN BY HUMAN AGAINST TOP COMPUTER
JEOPARDY!
STARCRAFT
POKER
ARIMAA
GO
SNAKES AND LADDERS
MAO
SEVEN MINUTES IN HEAVEN
CALVINBALL