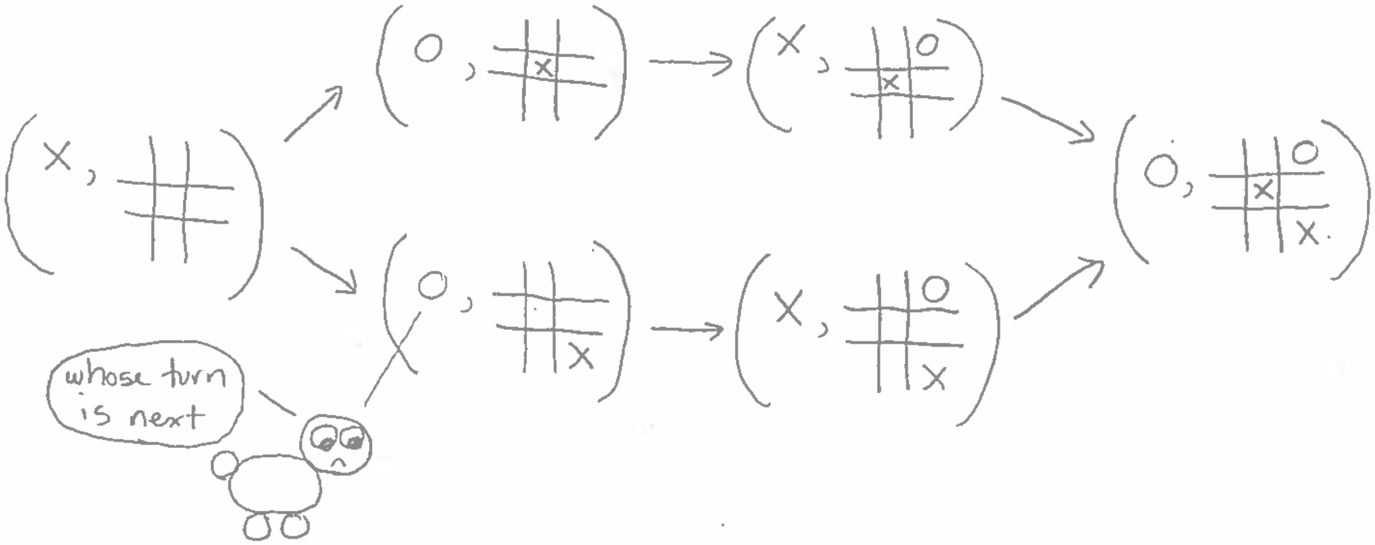
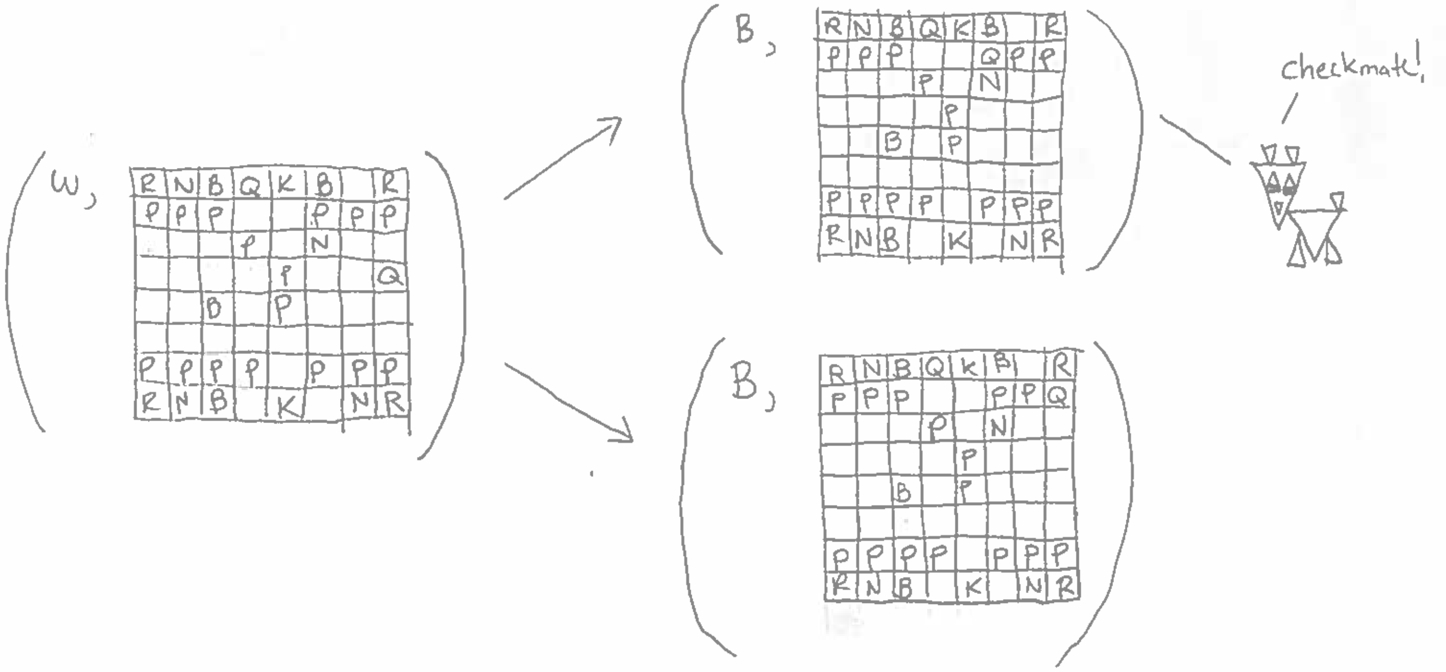
MINIMAX

① What if we want to use search techniques to play a multiplayer game, like chess or tic-tac-toe? We can still begin by expressing the game as an (unweighted) state machine, e.g.



would be a part of the state machine for tic-tac-toe. A partial state machine for chess would be:



② Essentially, a <u>game</u> with players P is:

- a state machine $M = (Q, \Sigma, \Delta, q_0, F)$, where each state $q \in Q$ has the form $(p, q') \in P \times Q'$ for some auxiliary set Q' of states.
- a utility function $U: F \times P \to \mathbb{R}$ that gives the value of each final state for each player.

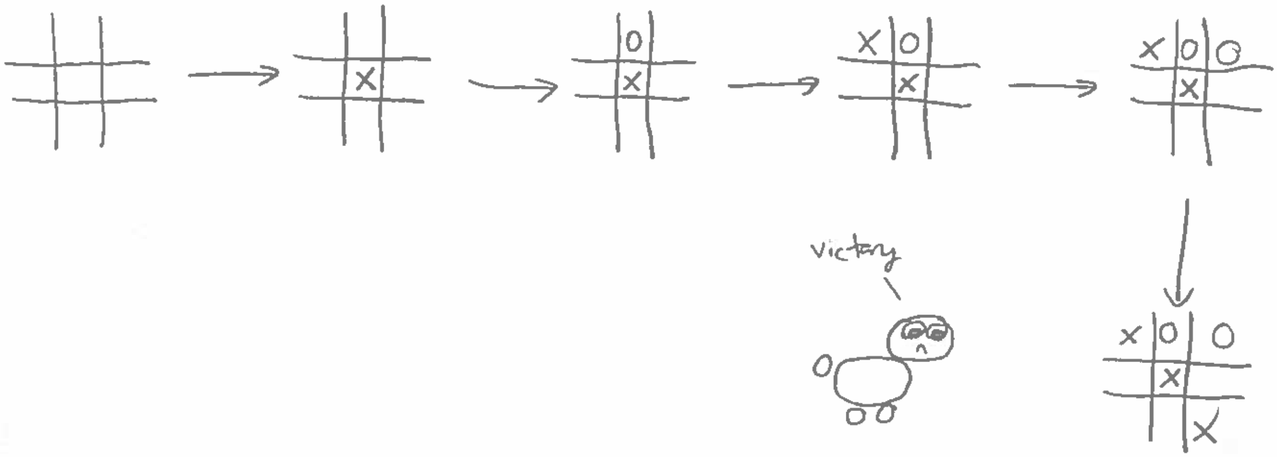
e.g.

$$q = \left(O, \begin{array}{c|c|c} X & O & O \\ \hline & X & \\ \hline & & X \end{array} \right)$$ is a final state for tic-tac-toe such that $U(q, X) = 1$ and $U(q, O) = -1$

$$q = \left(B, \begin{array}{|c|c|c|c|c|c|c|c|} \hline R & N & B & Q & K & B & & R \\ \hline P & P & P & & & Q & P & P \\ \hline & & & P & & N & & \\ \hline & & & & P & & & \\ \hline & & B & & P & & & \\ \hline & & & & & & & \\ \hline P & P & P & P & & P & P & P \\ \hline R & N & B & \cdot & K & & N & R \\ \hline \end{array} \right)$$ is a final state for chess such that $U(q, W) = 1$ and $U(q, B) = -1$.

For state $q = (p, q')$, use notation $p(q)$ to refer to p. (i.e. $p(q)$ is the player whose turn it is in state q).

3) Game search is a bit different from regular search because we don't control every player. Otherwise it'd be pretty easy to use standard techniques to find a high-utility final state:

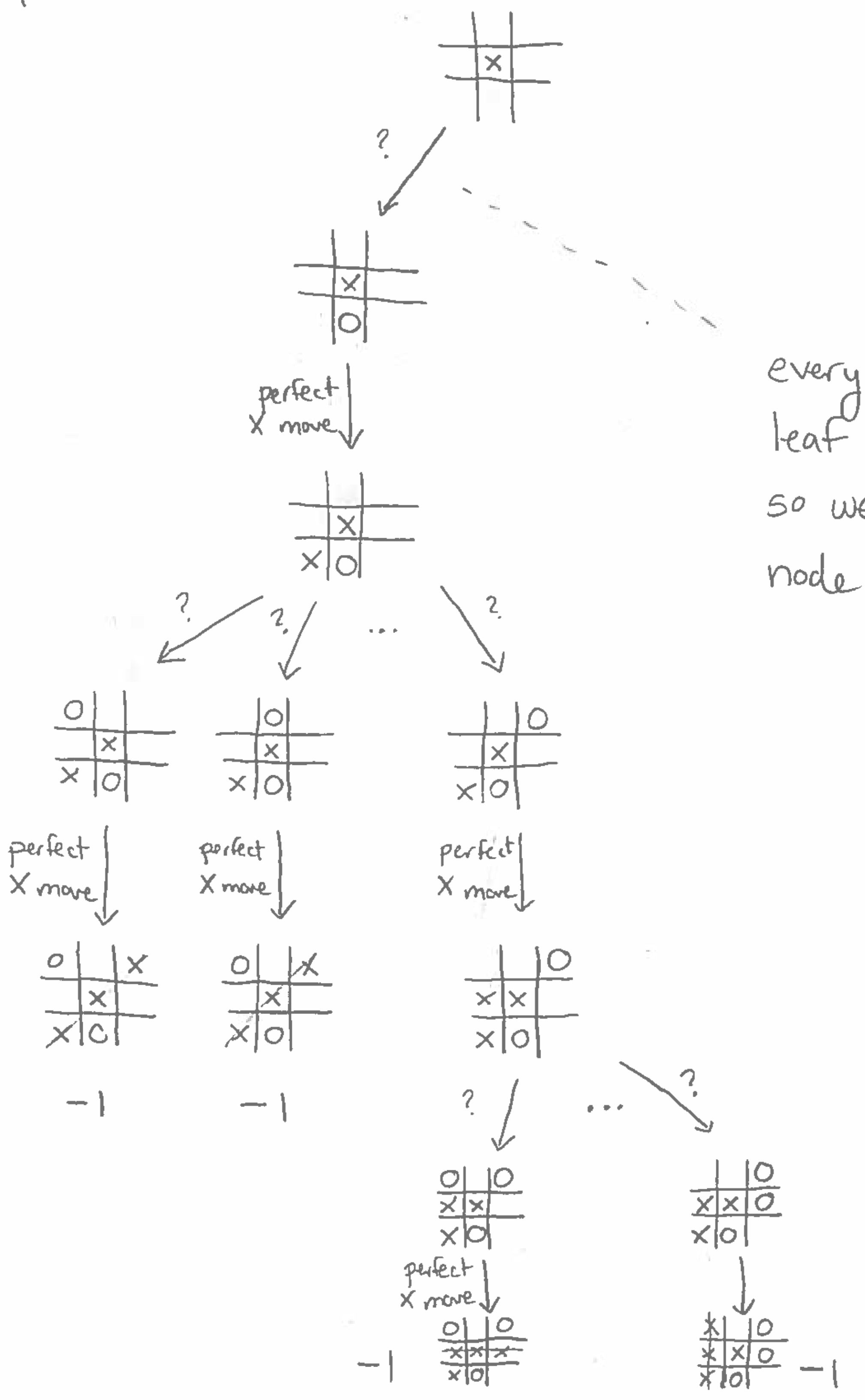


4) There's, for instance, a 4-move checkmate in chess, but it relies on an optimistic view that the opposing player will make no effort to stop you.

Generally, it is not in the opposition's interest to cooperate to help you win.

MINIMAX

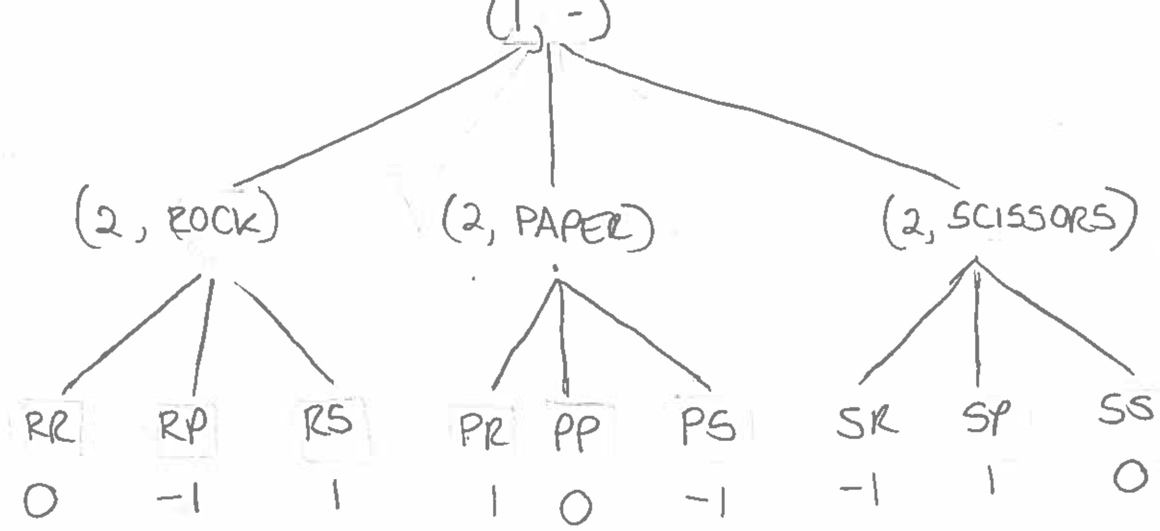
(5) Minimax search takes the ultimate pessimistic approach, assuming that you are playing against a perfect opponent. Suppose we are O, and we are evaluating how good this move is:



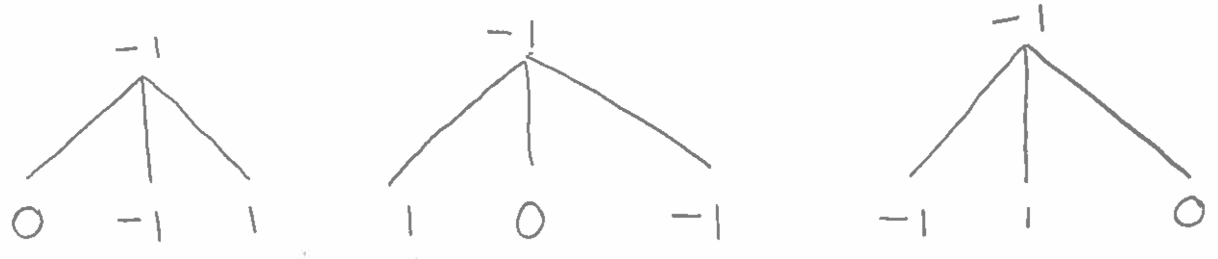
every reachable leaf has utility -1, so we give this node a value of -1.

MINIMAX

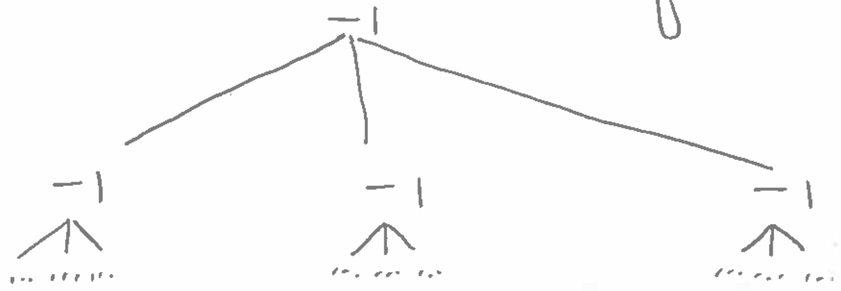
⑥ Algorithmically, this is simple to implement



Working the bottom up, do the following. If the state's player is not us, then the minimax value of the state is the MINIMUM value of its children (i.e. the opposing player seeks the worst outcome for us).



If the state's player is us, then the minimax value of the state is the MAXIMUM value of its children



MINIMAX

7) IF the minimax value of the root...

... is 1, then we're guaranteed to win if we play perfectly

... is 0, then there's no way to guarantee a win, but we can guarantee a draw if we play perfectly

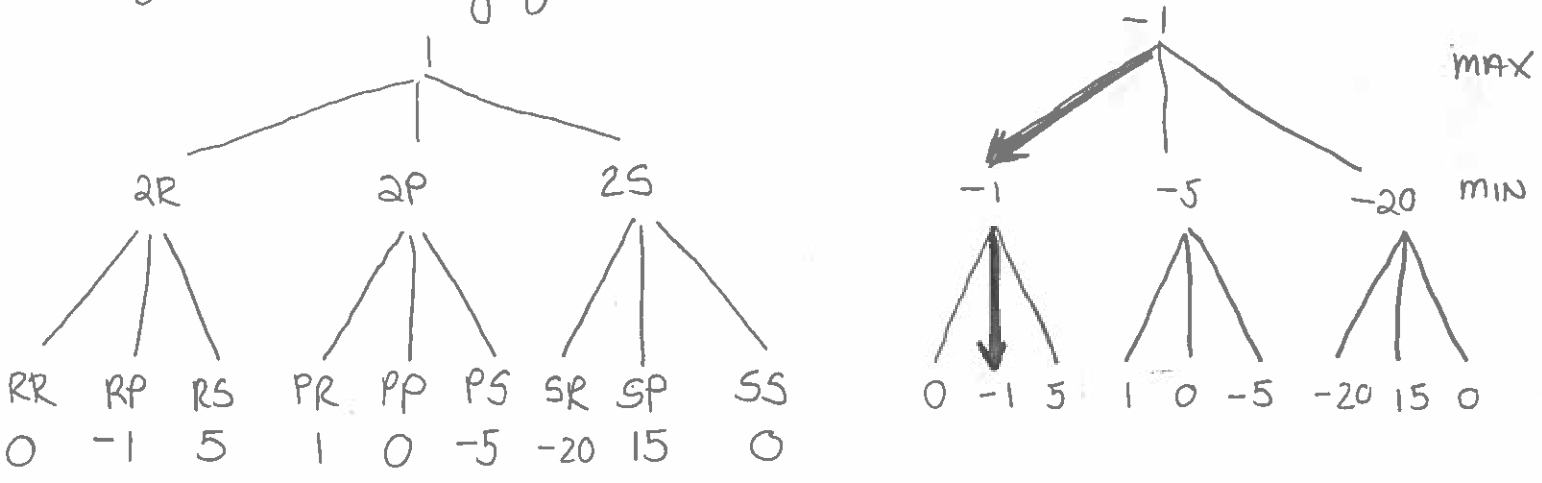
... is −1, then we're guaranteed to lose if the opponent plays perfectly.

8) So we're guaranteed to lose a turn-based version of rock, paper, scissors. Probably not a huge surprise.

But this method can also help if we have preferences about the outcome. For instance:

- Scissors are expensive to replace. I'd hate to have mine smashed by a rock (utility −20)
- I really enjoy the crafty joy of cutting paper with scissors (utility 15)

We get the following game tree:



So the best move is ROCK, to avoid the psychological trauma of having our scissors shattered.

Minimax

(9) Formally, define the minimax value of a game state q (for player p) as:

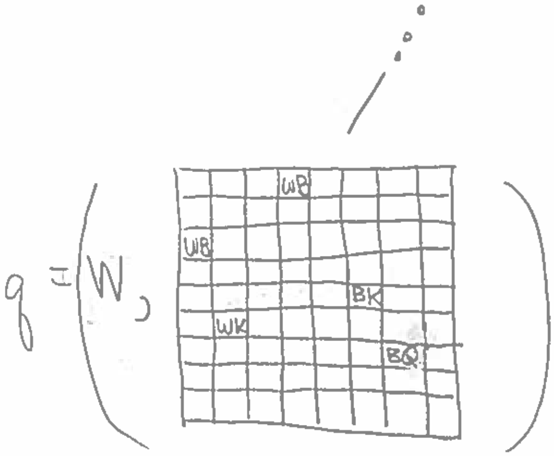
$$\text{minimax}(q, p) = \begin{cases} U(q, p) & \text{if } q \in F \\ \max\limits_{q'} \{\text{minimax}(q', p) \mid \langle q, \sigma, q' \rangle \in \Delta\} & \text{if } p(q) = p \\ \min\limits_{q'} \{\text{minimax}(q', p) \mid \langle q, \sigma, q' \rangle \in \Delta\} & \text{if } p(q) \neq p \end{cases}$$

(10) Well, that's nice. But how big is the minimax search tree? For rock, paper, scissors, we can visit all 13 states. Even for tic-tac-toe, there's less than $3^9 = 19683$ states, which is doable.

But chess has over 10^{40} nodes in its minimax search tree, which is a lot. So there's no way to compute minimax for the root of the tree.

11 So we need to cut off the tree at some point and then guess its minimax value.

For instance:



We could guess that the minimax value of state q (for White) is the number of white pieces − the number of black pieces:

$$\text{EVAL}(q, W) = 3 - 2 = 1$$

Or we could guess that the minimax value of state q (for White) is a weighted sum of the value of white's pieces minus a weighted sum of Black's pieces:

$$\begin{aligned}\text{EVAL}(q, W) &= \big(\text{Value}(\text{King}) + \text{Value}(\text{Bishop}) + \text{Value}(\text{Bishop})\big) \\ &\quad - \big(\text{Value}(\text{King}) + \text{Value}(\text{Queen})\big) \\ &= (10 + 3 + 3) - (10 + 9) \\ &= -3\end{aligned}$$

MINIMAX

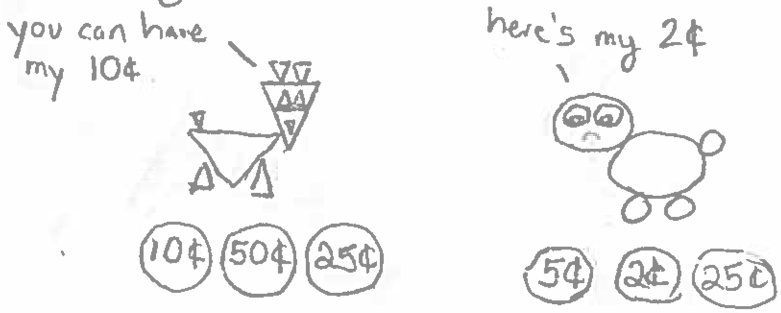
(12) This gives us the following algorithm: $M = (Q, \Sigma, \Delta, q_0, F)$

```
MINIMAX(M, U, EVAL, CUTOFF, p, q):
    if $q \in F$:
        return $U(q)$
    else if CUTOFF(q):
        return EVAL(q)
    else:
        children = $\{$MINIMAX(M, U, EVAL, CUTOFF, $p(q'), q') \mid \langle q, \sigma, q' \rangle \in \Delta \}$
        if $p(q) = p$:
            return max(children)
        else:
            return min(children)
```

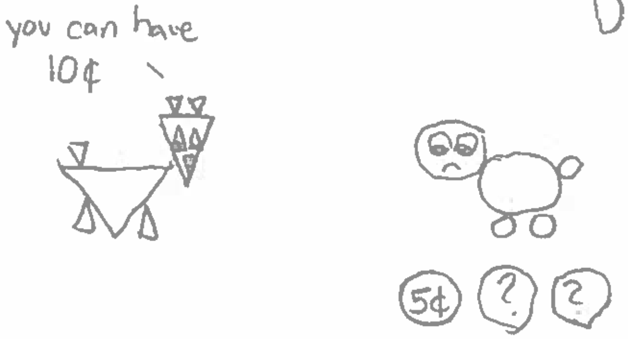
We focus on a DFS implementation, since we have a finite depth.

(13) MINIMAX can be wasteful.

Suppose I have two friends. Each friend has to give me a coin. My friends are greedy, so I know they're only going to give me their cheapest coin.



(14) If I know my first friend is going to offer me 10¢, then do I need to know all of my second friend's coins?

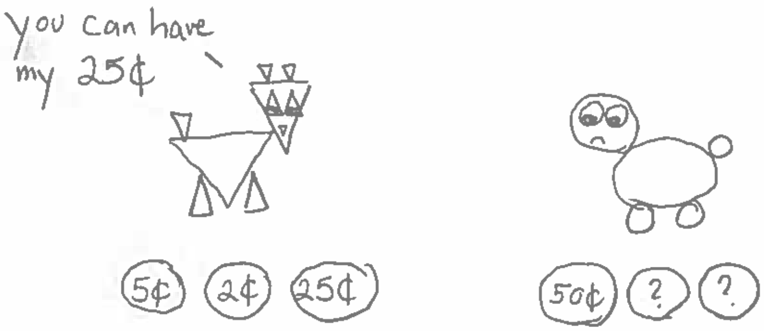


Once I know my second friend has a nickel, I know I'll never get offered anything better than a nickel, so I might as well take the first friend's dime.

I never need to know the other two coins.

MINIMAX

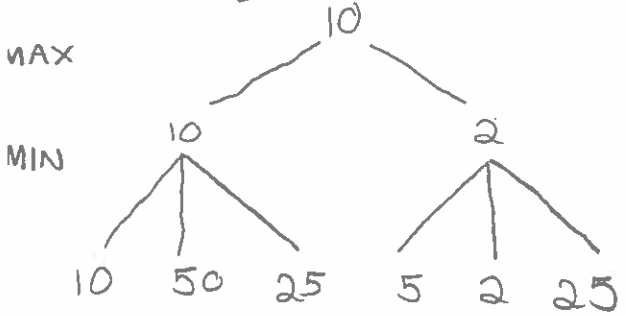
(15) Similar logic works if my friends are generous, but I don't want to take unnecessary advantage of them, so I plan to take the worst coin offered.



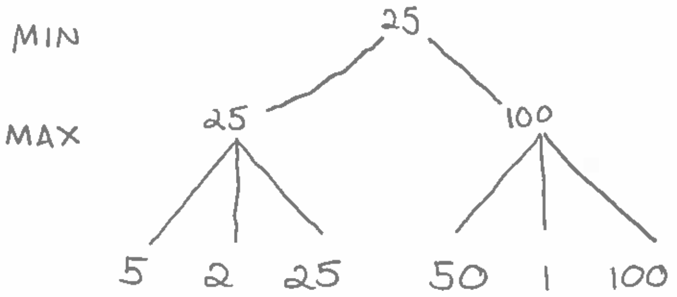
I know my second friend is going to offer me at least 50¢, so I can minimize the imposition on my friends by taking the first friend's offer of 25¢.

(16) The situations can be expressed as MINIMAX search trees:

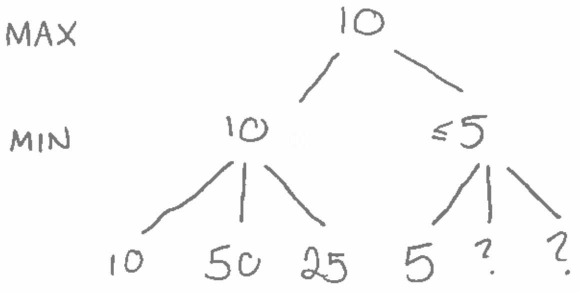
"greedy friends"

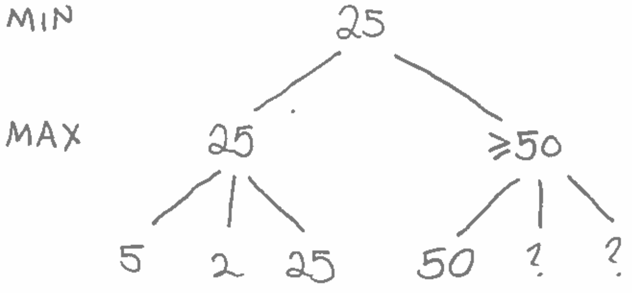


"generous friends"



Where we can optimize the search by skipping useless nodes:



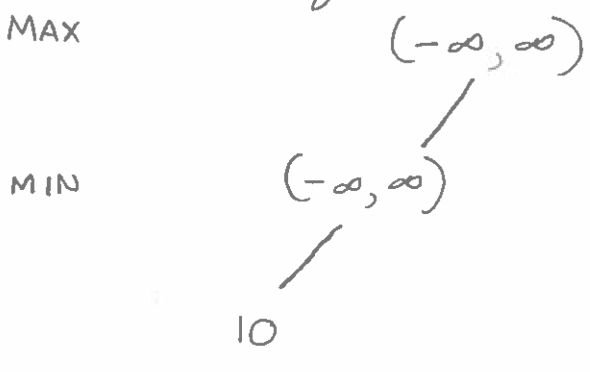


17 This optimization, called alpha-beta search, is a generalization of MINIMAX where we don't directly compute the minimax value of a node, but rather keep updating a lower and upper bound on the minimax value of each node (traditionally, the two bounds are called alpha and beta, which gives the algorithm its name).

Let's try it on the greedy friends. To start with, we don't know anything about the minimax value of the root, so its bounds are $(-\infty, \infty)$:

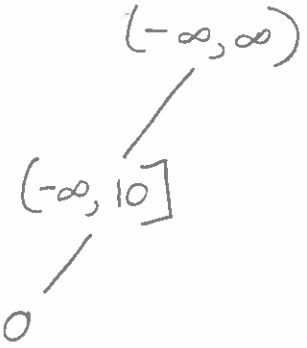
$$(-\infty, \infty)$$

We then do DFS until we reach our first final state (in which our first friend gives us their first coin):

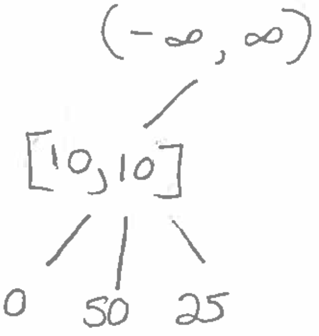


MINIMAX

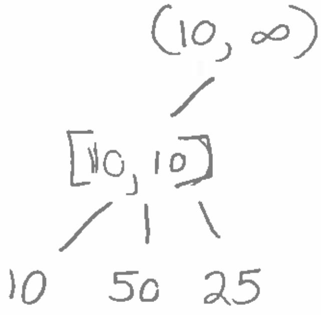
(18) At best, our greedy friend will be forced to give us his paltry dime, so can update our bounds on what they'll give us:



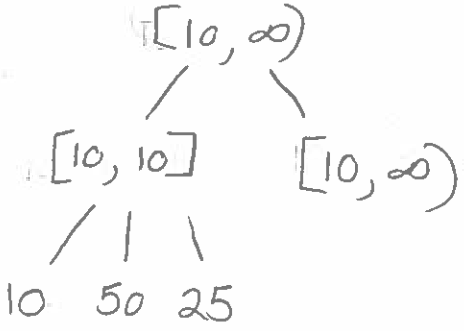
(19) We have to look at all our first friend's coins to figure out the lower bound:



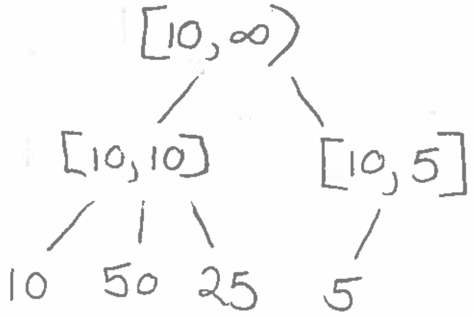
(20) Ok, so we'll at least get 10 cents, so we set the lower bound at the root:



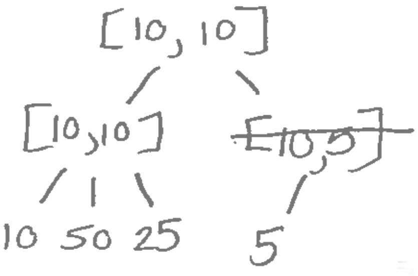
21) At this point, it gets interesting, because we know we're not going to take our second friend's coin unless it's at least 10¢. So we can set the lower bound for that node:



22) Now we check the first coin of our second friend. It's a nickel. So we're going to get at most 5¢ from that friend:



But there's no number between 10 and 5. So this node can't give us anything useful. We can stop investigating it. Since we've now explored all children of the root, we can set its upper bound:



MINIMAX

(23) We can upgrade MINIMAX into ALPHA-BETA as follows:

```
ALPHABETA($M$, $q$, $p_0$, EVAL, CUTOFF, $\alpha = -\infty$, $\beta = \infty$):
    if CUTOFF($q$):
        return EVAL($q$)
    else:
        bestEval = $\begin{cases} -\infty & \text{if } p(q) = p_0 \\ \infty & \text{o.w} \end{cases}$
        for $q'$ in $\{q' \mid \langle q, \sigma, q' \rangle \in \Delta\}$:
            if $\alpha \geq \beta$: return bestEval
            successorEval = ALPHABETA($M$, $q'$, $p_0$, EVAL, CUTOFF, $\alpha$, $\beta$)
            if $p(q) = p_0$:
                bestEval = max(bestEval, successorEval)
                $\alpha$ = max($\alpha$, successor Eval)
            else:
                bestEval = min(bestEval, successorEval)
                $\beta$ = min($\beta$, successor Eval)
        return bestEval
```