INF 4130 Exercise, alpha-beta-cutoff 2017 w/solutions

Exercise 1

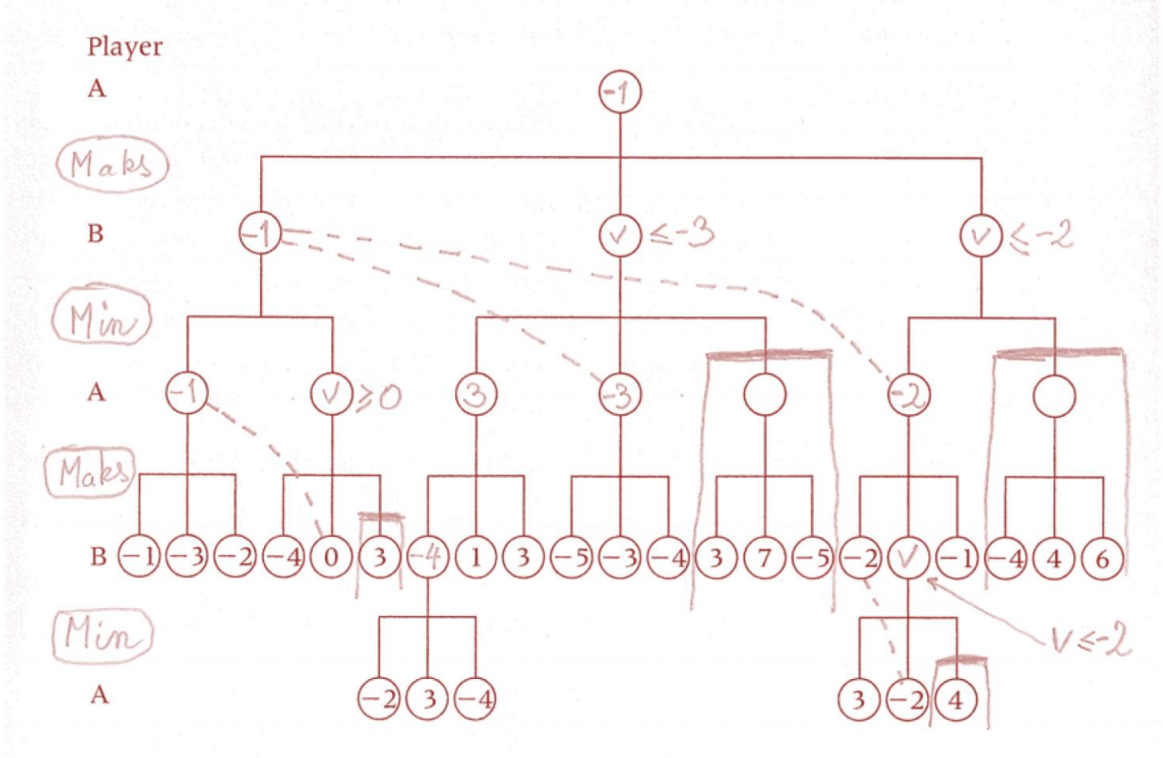
Study figures 23.13 and 23.14 (pages 735 and 736 in the textbook) where -1 and +1 is used to indicate win and loss, respectively. Look at all nodes and make sure you understand how values for the internal nodes are calculated with the min/max-algorithm. Always keep in mind that values indicate the situation for the player with the opening move – *A*. For *B* smaller values are better. (Note also that in exercise 3 below we negate the values on every level so that we can always maximize!)

Left to the group session.

Exercise 2

Study figures 23.16 and 23.17 (pages 738 and 740 in the textbook) and check that your understanding of alpha-beta-pruning is correct; then solve exercise 23.22 in the text book.

See figure below. Alpha and beta values are not written inside the node, the real node value is indicated insted, so that it is easy to see where we get cutoff. A dotted line is drawn between the values that cause the cutoff.



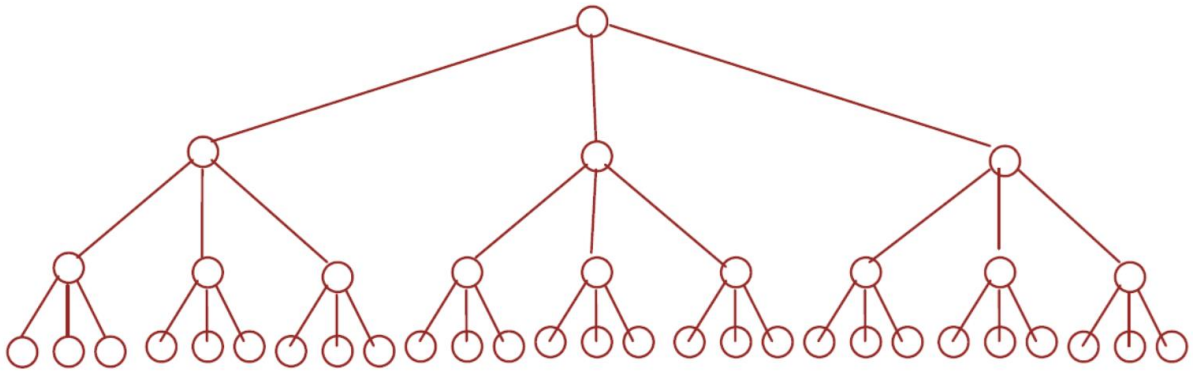
Exercise 3

Go through the program on page 741 and discuss the solution chosen there, where values are negated on every level. Note that there are some misprints in the program. First, in both lines following an “else” the name of the called procedure should be “ABNodeValue”. Second, a right parenthesis is missing at the end of the last of these lines.

This is left to the group session

Exercise 4

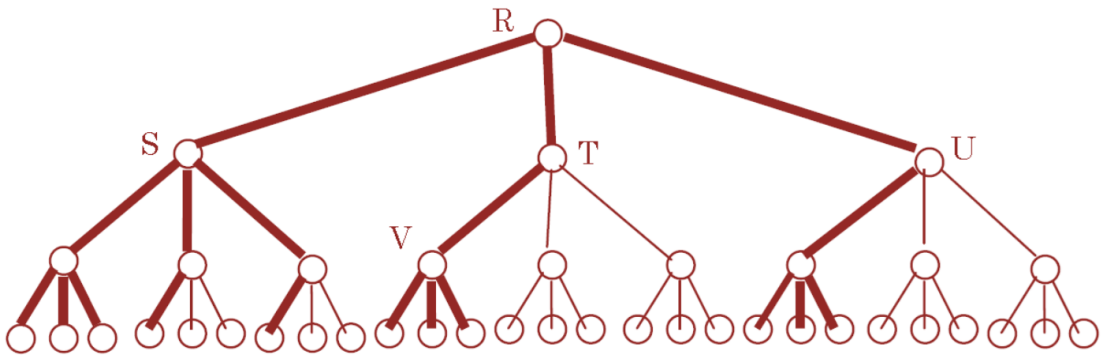
If we, in each situation in an alpha-beta-search, are lucky enough to always look at the best move (for the player to make the move) first we will get good pruning. One can even prove that if we go down to depth *d*, with a branching factor of *b*, the search time with alpha/beta-pruning will be $\underbrace{O(b \cdot 1 \cdot b \cdot 1 \cdot b \ldots)}_{d \text{ factors}}$, instead of $\underbrace{O(b \cdot b \cdot b \ldots)}_{d \text{ factors}}$. We shall not attempt to prove this, but instead look at a concrete example. We let *d* = 3, and *b* = 3, and get the tree below. Assume that the best move is always the one drawn to the left in the figure below and that we look at the subtrees from left to right for each node. Mark the branches you will have to evaluate (and thereby the ones you can avoid). The tree has 39 edges, how many do you avoid looking at?



EXTRA question: Assume you are unlucky(?) and always look at the best move *last.* Will you then get any pruning at all?

Note that what we try to do is always look at the best move for the player with the move. We do, of course, not know what move is the best (this would make the analysis too easy!), instead we have to assume we have an heuristic that gives us the move, and run the algorithm. What we study is how algorithm behaves if we are lucky in our choice of heuristic.

Below is the tree with bold edges where the algorithm must descend. As an example we look at nodes S, T and U. We assume that A has the highest value (the highest value of the nodes S, T and U, since we maximize in the the root. We further assume that move V is the best possible from situation T. That is, the value returned from V to T is so low that we see that the T-value is so small (remember that we minimize in T) that is can not compete with the S-value when we maximize in R. And that we therefore need not look at the two remaining branches in T. And similarly for S.



Of the 39 edges we need only look at 19, and the number of leaves we look at is 11 out of 27. the number of leaves is important as we (at least in chess) most likely have to do a quite heavy analysis of our position to give it a score (with a suitable heuristic).

The answer to the **extra question** is that you *can indeed* get some pruning (cutoff), if, for instance, the next best move comes first and there are at least three children.

Exercise 5 (not central to the course)

Assume you are playing the game of NIM, with two piles, and that it is your move, that no pile is empty, and that the piles are of different size (number of sticks or pebbles, or whatever). Try to come up with a strategy that guarantees victory.

The idea for one such strategy is to make sure that the opponent always finds himself in a situation where both piles are equal. In the situation described in the Exercise you can always make sure that this will be the case for the opponent in his next move, by taking a suitable number of sticks from the largest pile. You should go on doing this as long as the smallest pile contains at least two sticks.

Then the end of a game: If the opponent makes a move that leaves one stick in one of the piles (and then there are at least two in the other pile!), then take all the sticks in the large pile, and you'll win. If there are pile is empty (and still at least two sticks in the other), then take all sticks in the remaining pile except one, and you'll again win.

Exercise 6 (From the Exam, 2009)

We shall look at game trees, and we assume that the root node of the tree in the figure below is representing the current situation of a game (that we do not describe further), and that it is player A's turn to move. The other player is B, and A and B alternately make moves. Player A wants to maximize the values of the nodes while B wants to minimize them.

Player A shall make considerations for deciding which move to make from the root situation, and the tree in the figure below shows all situations it is possible to reach with at most four moves from the current situation. A has a heuristic function (that is, a function that for a given situation gives an integer) that he uses to evaluate how god the situation is for him. A uses this function for situations where he terminate the search towards deeper nodes. For each terminal node in the tree below this function is evaluated and the value appears in the nodes.

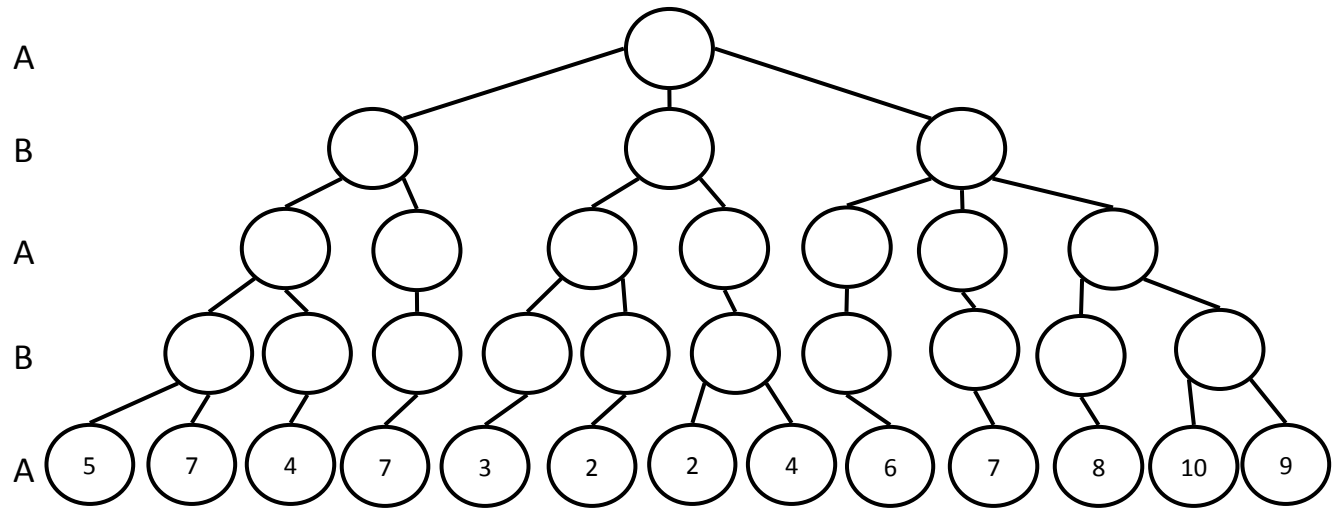


Figure 6.1 A game tree with values in the terminal nodes.

6.a

Using the heuristic values in the terminal nodes, indicate how good the situation is for A in each of the other nodes. What is the best value player A can achieve regardless of how well B plays.

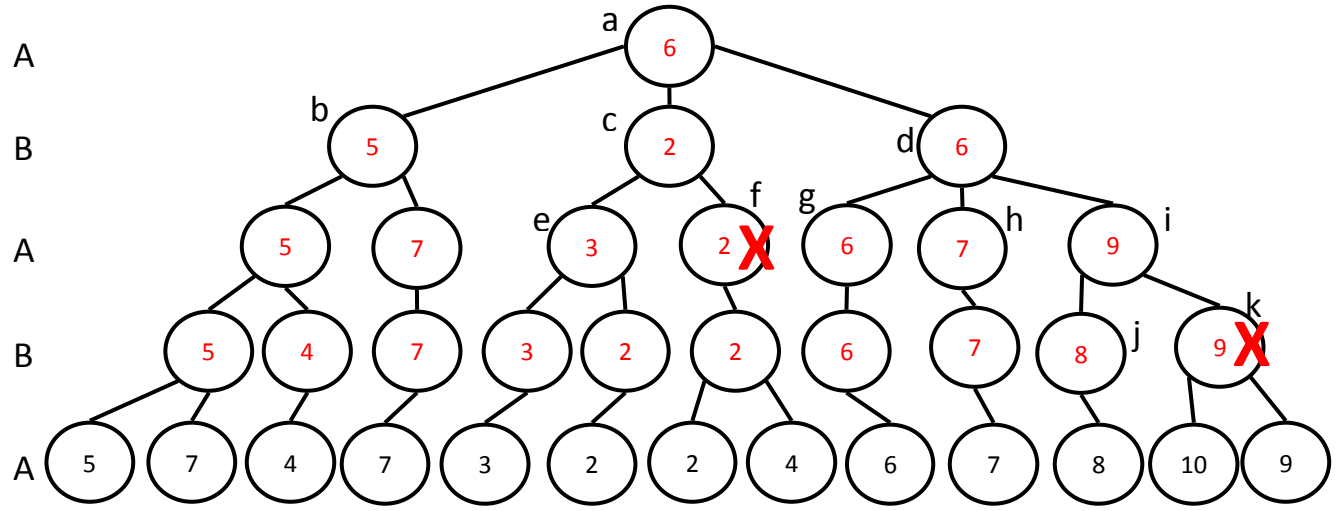
6.b

We now assume that we are back to the start-situation, no nodes have values. We start the algorithm again, and A will then make a depth-first search in the tree from the root node, down to the depth of the tree above. In each terminal node A computes the value of the heuristic function (and thereby gets the value given in the

corresponding terminal node in the figure above). The search is done from left to right in the figure above. Indicate which alpha- and beta-cutoffs you will get during this search.

In the drawing you should simply write an 'X' at the root nodes of the sub trees that will not visit because of alpha- or beta-cutoffs. Give a short but explicit explanation for each cutoff (and for this you may suitably give names to some of the nodes in the figure).

Answer, exercise 6



Answer 6.a
The best value A can get independent of how well B plays is 6

Answer 6.b
The reason why we get a cutoff for *f* is that we in *e* have already got the value 3, and that the value in *c* (where we minimize) therefore cannot become larger than 3. And that means that the real (final) value in *c* will not be of interest for computing the value in *a* (where we maximize) as we already know (from *b*) that its value will be at least 5. Thus, there is no need to look at further subtrees of *c*. The value in *c* then ends up as 3, but that will have no effect for the considerations for the levels above.

The reason for the cuttoff at *k* is that we have already seen that *j* yields the value 8, and that *i* (here we maximize) therefore never will become less than 8. But that means that the value of *i* has no interest for the value at *d* (where we minimize), as we already know from *g* that this can never be larger than 6. Thus, we need not look at further subtrees of *i*. The final value of *i* will, with this cutoff, end up as 8.

[END]