Figure 5.5  Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of at most 3. (b) The second leaf below B has a value of 12. MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B’s successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is at least 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of at most 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha—beta pruning. (e) The first leaf below D has the value 14, so B is worth at most 14. This is still higher than MAX’s best alternative (i.e., 3), so we need to keep exploring D’s successor states. Notice also that we now have bounds on all of the successors of the root, so the root’s value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX’s decision at the root is to move to B, giving a value of 3.

somewhere in the tree (see Figure 5.6), such that Player has a choice of moving to that node. If Player has a better choice in either at the parent node of n or at any choice point further up, then n will never be reached in actual play. So once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it.

Remember that minimax search is depth first, so at any one time we just have to consider the nodes along a single path in the tree. Alpha—beta pruning gets its name from the following two parameters that describe bounds on the backed up values that appear anywhere along the path:
Section 5.3. Alpha–Beta Pruning

Figure 5.6 The general case for alpha–beta pruning. If \( m \) is better than \( n \) for Player, we will never get to \( n \) in play.

\( \alpha = \) the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for \( \text{MAX} \).

\( \beta = \) the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for \( \text{MIN} \).

Alpha–beta search updates the values of \( \alpha \) and \( \beta \) as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current \( \alpha \) or \( \beta \) value for MAX or MIN, respectively. The complete algorithm is given in Figure 5.7. We encourage you to trace its behavior when applied to the tree in Figure 5.5.

5.3.1 Move ordering

The effectiveness of alpha–beta pruning is highly dependent on the order in which the states are examined. For example, in Figure 5.5(e) and (f), we could not prune any successors of \( D \) at all because the worst successors from the point of view of \( \text{MIN} \) were generated first. If the third successor of \( D \) had been generated first, we would have been able to prune the other two. This suggests that it might be worthwhile to try to examine first the successors that are likely to be best.

If this can be done, then it turns out that alpha–beta needs to examine only \( O(b^{m/2}) \) nodes to pick the best move, instead of \( O(b^m) \) for minimax. This means that the effective branching factor becomes \( \sqrt{b} \) instead of \( b \)—for chess, about 6 instead of 35. Put another way, alpha–beta can solve a tree roughly twice as deep as minimax in the same amount of time. If successors are examined in random order rather than best-first, the total number of nodes examined will be roughly \( O(b^{m/3}) \) for moderate \( b \). For chess, a fairly simple ordering function (such as trying captures first, then threats, then forward moves, and then backward moves) gets you to within about a factor of 2 of the best-case \( O(b^{m/2}) \) result.

\( ^2 \) Obviously, it cannot be done perfectly; otherwise, the ordering function could be used to play a perfect game!