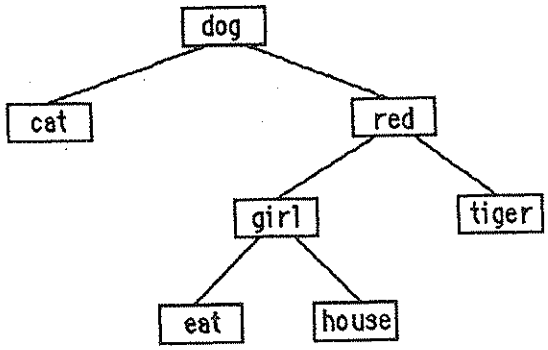
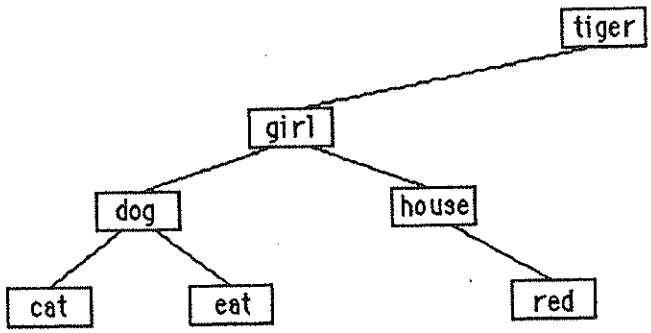
SECTION 9.4 Further Applications of Binary Trees

1. We can put any word we wish at the root. Suppose that we decide that it will be "dog." This forces "cat" to be its left child, since it is the only vertex in the left subtree (since it is the only word in the list that precedes "dog" alphabetically). We again have a lot of choice for the right subtree, however: Any of the remaining five words can be the right child of "dog." We might choose "red." There are further choices for this vertex's left child, and so on. The tree we have been discussing might end up like this.



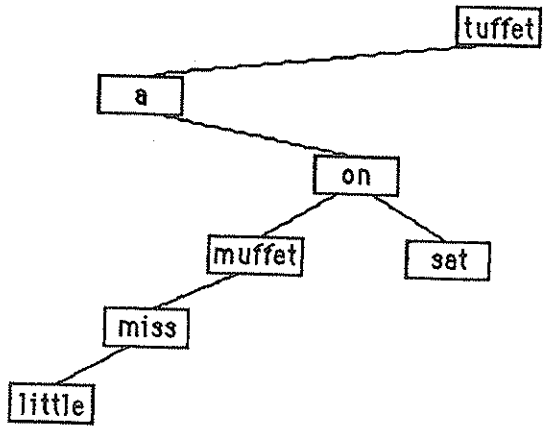
If we make other choices along the way, such as deciding to put "tiger" at the root, then our tree could end up like this.



3. (a) Since *boy* $\prec$ *girl*, we move to the left subtree, rooted at "dog," and apply the algorithm recursively here. Now *boy* $\prec$ *dog*, so we move left again. Next *boy* $\prec$ *cat*, and we move to the empty left child. Since this vertex is empty, the search has been unsuccessful, and we return 0. This returned value is passed back through all the layers of recursion and becomes the final answer.

(b) This is even quicker. We find that "boy" precedes "cat" and immediately drop off into the empty left subtree, so the output is again 0.

5. We start by placing "tuffet" at the root. As each new word is encountered, we use Algorithm 2 to insert it in the appropriate spot. First "a" is compared to "tuffet" and is found to precede it. Since "tuffet" had no left child, "a" becomes its left child. Next we want to insert the word "on." We compare it to "tuffet" and find that we must move left and insert it somewhere in the left subtree (calling the procedure recursively). At this point, since "on" is greater than "a," and since "a" has no right child, we insert "on" as the right child of "a." The return from the recursive call is followed by the return from the main call to the procedure, and we have finished. In a similar manner we insert all the remaining words. The tree ended up very unbalanced.



7. **(a)** The letter F occurs as the right child of the left child of the left child of the right child of the root. In order to reach it, we must, starting at the root, move right, then left, then left again, and finally right. Since we encode a move to the right with a 1 and a move to the left with a 0, the letter F is encoded 1001. Similarly the next letter in the message, I, is reached by moving right, then left, then right, so its code is 101. Thus the code for FI is 1001101. We continue in this way, encoding D as 1110 and O as 1100. Putting this all together, we have the entire encoding of $FIDO$: 100110111101100.

(b) This is similar to part (a); the encoding of SIT is 010101011.

(c) The encoding of this word is fairly long because the code for W, which occurs three times, requires five bits. The answer is 10001100111101111010011110.

9. For the first tree, we calculate as follows. The word "dog" occurs with relative frequency 0.10, and since it is at the root it requires only one comparison to find it. Therefore it contributes $0.10 \cdot 1 = 0.10$ to the average. There are two words at level 1, "cat" and "red," and they require two comparisons to find (one comparison with "dog" and the successful comparison). Since they occur with relative frequency $0.35 + 0.05 = 0.40$, they contribute $0.40 \cdot 2 = 0.80$ to the average. Similarly, for the third we level, there is a contribution of $(0.05 + 0.05) \cdot 3 = 0.30$; and for the fourth level the contribution is $(0.30 + 0.10) \cdot 4 = 1.60$.

Therefore the average number of comparisons needed for a successful search in the first tree we drew for Exercise 1 is $0.10 + 0.80 + 0.30 + 1.60 = 2.80$. Similarly, for the second tree the average is

$$0.05 \cdot 1 + 0.05 \cdot 2 + (0.10 + 0.10) \cdot 3 + (0.35 + 0.30 + 0.05) \cdot 4 = 3.55 .$$

11. This proposition is false. The stated condition is necessary but not sufficient. For a counterexample, take a binary search tree with just three words in it: "hello" at the root, "goodbye" as the left child of the root, and "mayday" as the right child of "goodbye." The given condition is satisfied at both internal vertices, but "mayday" cannot be in the left subtree of "hello" (which precedes it alphabetically) if this were to be a binary search tree.

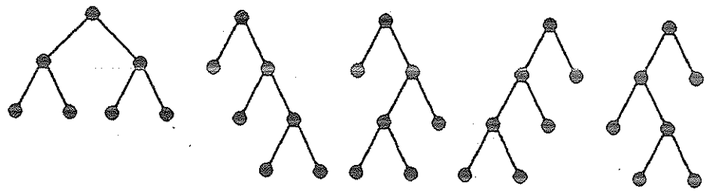
13. Three cases must be handled. If the word we are looking for (call it x) is the root r, then we are finished; we **return** r. Otherwise we move either to the left subtree or to the right subtree, according as $x \prec \mathit{contents}(r)$ or vice versa. In either of these cases there are two subcases. If the subtree in question is empty, then we create a new vertex, make it the left or right child of the root, as appropriate, fill it with x, and **return** it. Otherwise we recursively continue in the subtree. Our algorithm in pseudocode is shown here.

```
procedure find_or_insert(r : root of nonempty binary search tree, x : word)
    if x = contents(r) then return(r)
    else if x ≺ contents(r) then
        if left_child(r) = ∅ then
            begin
                create a new vertex v
                contents(v) ← x
                left_child(r) ← v
                return(v)
            end
        else return(find_or_insert(left_child(r), x))
    else { x ⊀ contents(r) in this case}
        if right_child(r) = ∅ then
            begin
                create a new vertex v
                contents(v) ← x
                right_child(r) ← v
                return(v)
            end
        else return(find_or_insert(right_child(r), x))
```

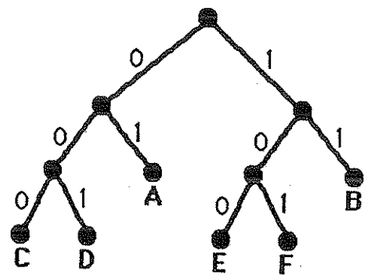
15. We should make the tree as balanced as possible, with all leaves at level h or at level $h-1$. This will give us a tree with the minimum possible height, and there will be no missing children in the tree except at the bottom level. In order to form such a tree if there are n words, we simply make the $\lceil n/2 \rceil$th word the root and recursively apply this procedure to the words that are thereby forced to go into the left and right subtrees. Two examples of such a tree would be the tree shown in Figure 9.22 and the same tree with the word "cat" removed.

17. Certainly one way for the substring 11111 to enter the message is as the encoding of Y. However, it could also occur, for example, if the letter T was immediately followed by the letter D, since the codes for these two letters are 011 and 1110, respectively.

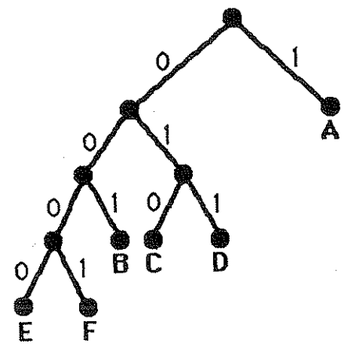
19. We can draw all the possible full binary trees with four leaves (and therefore, by Theorem 3 in Section 9.1, three internal vertices), as shown here. Note that of these five trees, only the first is really different (as an unordered tree); the others all have one code of length 1, one code of length 2, and two codes of length 3.



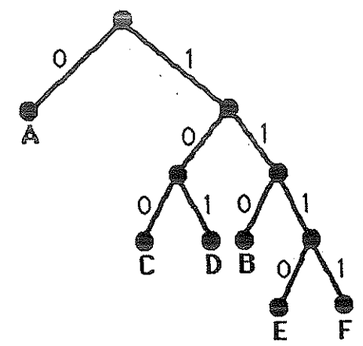
21. **(a)** Initially each letter is a vertex by itself, with its own frequency. We start by combining the two least frequent vertices, E and F, as children of new node t_1, whose frequency is $0.10+0.15 = 0.25$. Then we form t_2, with children C and D, and frequency $0.15+0.15 = 0.30$. At this point the two vertices with smallest frequencies are t_1 and B, so we make them the children of t_3, whose frequency is $0.15 + 0.25 = 0.40$. Next t_4 has t_2 and A as its children and frequency $0.30 + 0.30 = 0.60$. Finally t_4 and t_3 become children of the root. The tree is as shown here.



(b) Following the same procedure as in part (a), we obtain the following tree, where the symbol A has relative frequency 0.40.

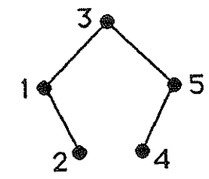


(c) The tree obtained this time looks like this, essentially the same as in part (b).



23. If this were not the case, then we could exchange the positions of u and v to achieve a tree with smaller average code length. We would cut down on the code length by the difference in the relative frequencies between u and v times the number of levels separating them.

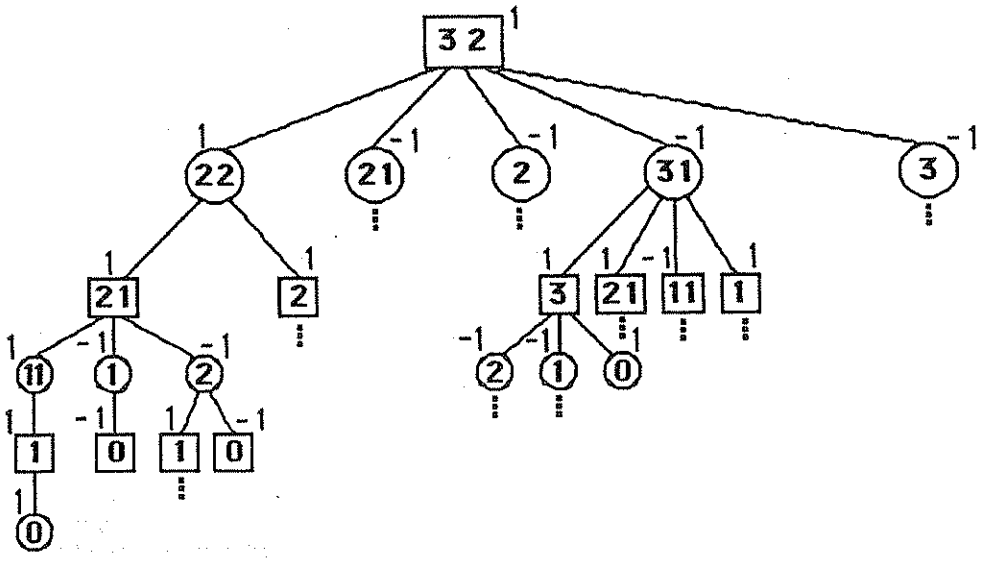
25. We will show this inequality by constructing a function f from permutations of $\{1, 2, \ldots, n\}$ to binary trees; if we can show that this function is surjective, then we will know that the number of trees is at most $n!$, the number of permutations. Given permutation π, construct a binary search tree by inserting the symbols of π, in order; $f(\pi)$ is the underlying binary tree. For example, if π is 3 5 1 2 4, then the following tree is obtained.



This function is onto the set of binary trees with n vertices, since given any such binary tree, we can label its vertices with the set $\{1, 2, \ldots, n\}$ in inorder and let π be the permutation obtained by listing the labels in preorder—clearly $f(\pi)$ is the given tree. (In our example, given the tree shown there, we would have π equaling 31254.) Thus the number of binary trees is at most $n!$. Furthermore any binary tree in which the root has two children (and for each $n \geq 3$ there is at least one such tree) can be obtained from at least two different permutations, the usual left-to-right preorder and a backward, right-to-left, preorder. The conclusion follows.

SECTION 9.5 Game Trees

1. We draw the game tree by putting the initial position at the root, representing the piles of stones by the integers 3 and 2. According to the rules of the game, the first player may now remove one, two, or all three of the stones from the first pile, leaving positions 22, 21, and 2, respectively; or he may remove one or both of the stones from the second pile, leaving positions 31 or 3. These five vertices are therefore the children of the root. To find the children of these vertices we again invoke the rules. For example, the children of 22 are 21 and 2. We continue in this way until the entire tree is constructed. The following diagram shows most of the tree, but we have not filled in portions that occurred elsewhere.



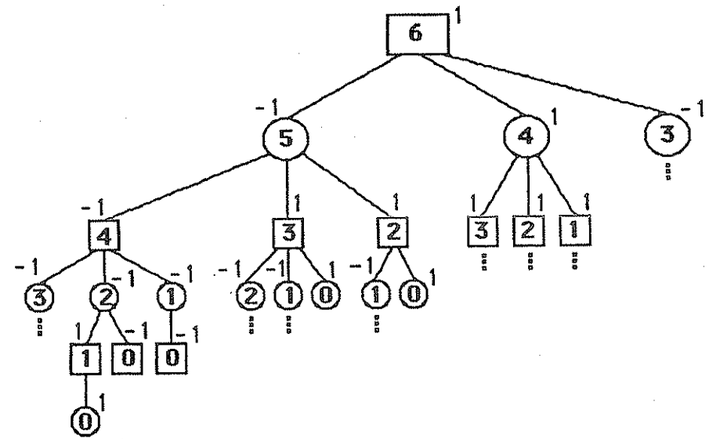
For example, the subtree rooted at 21 occurred rooted at the first grandchild of the root, so we do not repeat it as the second child of the root. We will be able to determine the value of the second child of the root by knowing the value of the first grandchild (since the rules for the two players are the same).

(b) We start from the bottom left. Since the person with no move loses, the position at the far bottom left is a loss for the second player; in other words, it has value 1 to the first player. Therefore its parent, the square vertex with position 1, is also a win for the first player. In turn its parent is 11, and 11 has only one child (there is only one move the second player can make when faced with 11). Therefore it, too, has value 1. Now since this represents a winning move for the first player when faced with 21, we see that the value of the first grandchild of the root is also 1. (We have also filled in the rest of that subtree, and 21's other children have value -1, but that is irrelevant here, since the value of 21 is the maximum of the values of its children.) Next to determine the value of the first child of the root (the circle position 22), we need to know the value of its other child, namely 2 in a square. But this is clearly 1 also, since the first player can win when faced with that position by taking both stones. Returning now to the circle position 22, we see that both of its children have value 1, so it must have value 1 (the minimum of 1 and 1). This is enough to determine the value of the game, for the value of the root is the maximum of the value of its children. In other words, the first player can win by moving to position 22 on his first move.

Note that we did not need to compute the values of the rest of the vertices in the tree, although they are shown in the diagram. For example, the value of the circle vertex 21 is -1, since we found the value of the square vertex 21 to be 1 (this follows by symmetry—what is good for the first player when it is his turn to move must also be good for the second player when it is her turn to move).

(c) As remarked in part (b), the first player wins the game by taking one stone from the pile with three stones. Then if the second player clears one pile, he clears the other; otherwise he leaves position 11 for her.

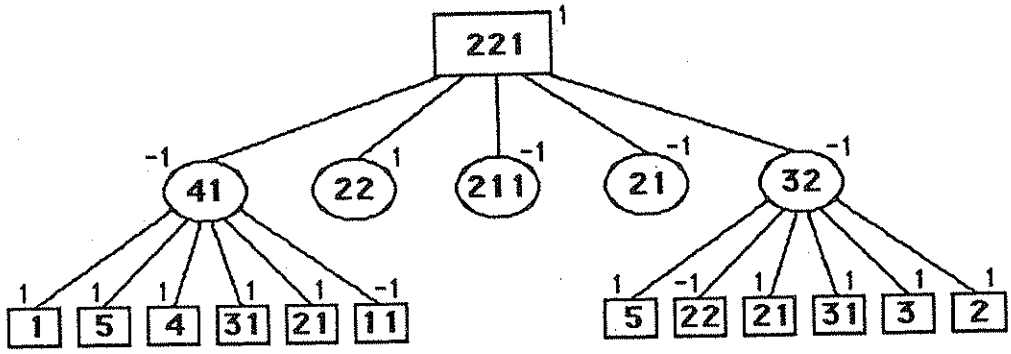
3. (a) Each vertex containing a number $k \geq 3$ has three children, with positions $k-1$, $k-2$, and $k-3$. We build the game tree from the top down, as shown here. Only a portion needs to be drawn in detail, as we will see in part (b).



(b) The circle vertices with 0 have the value 1, since the first player wins when the second player cannot move. Similarly, the square vertices with position 0 have the value -1. We fill in the values from the bottom up, finding that the square 1 has value 1 (and therefore by symmetry the circle 1 has value -1), the circle 2 has value -1 (the minimum of the values of its two children), the square 3 has value 1, and so on. Ultimately, we find that the root has value 1.

(c) The first player wins, since the root has value 1. Only one child of the root has the value 1, however, so the first player must be careful and take two stones (leaving position 4) as his first move; otherwise he moves to a position with value -1 and the second player can win.

5. (a) The game tree is shown here. We have omitted much of it, relying on facts obtained in Example 1 and Exercise 4. For example, we saw in Exercise 4 that the value of circle position 22 (second player to move) was 1, so we do not draw the subtree rooted there.

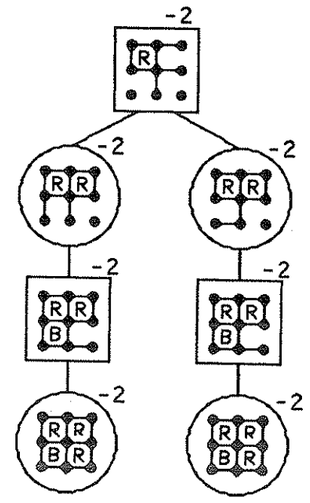


(b) The values of the leaves are obtained from the results of Example 1 and Exercise 4. Then we compute the values of the internal vertices by the basic rule, that the value of a square vertex is the maximum of the values of its children (the first player will move to maximize the payoff), and the value of a circle vertex is the minimum of the values of its children (the second player will move to minimize the payoff).

(c) Since the value of the root is 1, the first player wins. Note that moving to position 22 is the only winning play.

7. Note that "Blue wins" really means "Blue will win if he follows the proper strategy." If $n = 0$ or 1, then the statement is obvious. That is our base case of an inductive proof. Assume the inductive hypothesis that in a game with a smaller number of piles of one stone each, Blue can win if the number of stones (i.e., piles) is odd, and Red can win if it is even. Now assume that n is odd and greater than 1. Blue takes one stone and by the inductive hypothesis wins (he is now the second player in the game with a smaller, even number of stones). Finally assume that n is even and greater than 0. If Blue takes one stone, then Red wins by the inductive hypothesis (she is now the first player in the game with a smaller, odd number of stones). If Blue combines two piles into one, then Red removes the new pile and wins by the inductive hypothesis (there are now $n - 2$ stones, and $n - 2$ is a smaller, even number).

9. There is very little choice here. Red, whose turn it is to move when the curtain opens on this drama, must first complete the upper right-hand square. Then she can draw either a vertical segment or a horizontal segment; the two possibilities are shown in our tree. Blue must then complete the square that is offered, and he has only one remaining move, up to symmetry. The final position in either case has Red two squares ahead of Blue, so the value of both leaves of our tree is -2. Working up toward the root, we see that the value of every vertex in this tree is also -2. What we see here is just one small subtree of the entire game tree for this game, starting from the empty game board with nine dots.



11. In each case, we are asking for the number of ways for the game to proceed through the first move of each player. This is a counting exercise.

(a) There are 20 opening moves in chess: Each of the eight pawns can move either one square forward or two, and the two knights can each move to two different spots. The second player can respond to each of these 20 moves with any of 20 similar moves from her side. Thus the answer is $20 \cdot 20 = 400$.

(b) There are seven moves for each player to begin with (two moves for each of three checkers and one move for the checker next to the edge of the board). Thus the answer is $7 \cdot 7 = 49$.

(c) The game board has I rows of dots, with J dots in each row. In each row there are $J - 1$ line segments that can eventually be drawn, so there are a total of $I(J - 1)$ horizontal line segments. Similarly there are $J(I - 1)$ vertical segments. Thus there are $I(J - 1) + J(I - 1)$ moves in the entire game. Call this quantity n. Any one of these n moves can be the first move, and any of the remaining $n - 1$ moves can be the second. Therefore the answer is $n(n - 1)$. If we wanted to discount for symmetry, the answer would be smaller, but harder to compute.

13. **(a)** A portion of the game tree is shown here. It is too big to draw in its entirety. We make use of the observation made in the text that Red will always move to a position involving smaller numbers and Blue will always move to a position involving larger numbers.