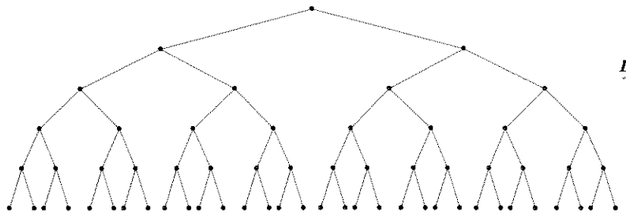
This is a contradiction. Therefore, no algorithm that sorts three items has worst-case time less than 3, and the algorithm of Figure 9.7.7 is optimal. ◀

Since $4! = 24$, there are 24 possible outcomes to the problem of sorting four items (when the items are distinct). To accommodate 24 terminal vertices, we must have a tree of height at least 5 (see Figure 9.7.9). Therefore, any algorithm that sorts four items requires at least five comparisons in the worst case. Exercise 9 is to give an algorithm that sorts four items using five comparisons in the worst case.



| *Level* | *Number of Vertices* |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |

Figure 9.7.9 Level compared with the maximum number of vertices in that level in a binary tree.

The method of Example 9.7.2 can be used to give a lower bound on the number of comparisons required in the worst case to sort an arbitrary number of items.

**Theorem 9.7.3** *If $f(n)$ is the number of comparisons needed to sort $n$ items in the worst case by a sorting algorithm, then $f(n) = \Omega(n \lg n)$.*

***Proof*** Let $T$ be the decision tree that represents the algorithm for input of size $n$ and let $h$ denote the height of $T$. Then the algorithm requires $h$ comparisons in the worst case, so

$$h = f(n). \tag{9.7.1}$$

The tree $T$ has at least $n!$ terminal vertices, so by Theorem 9.5.6,

$$\lg n! \leq h. \tag{9.7.2}$$

Example 4.3.9 shows that $\lg n! = \Theta(n \lg n)$; thus, for some positive constant $C$,

$$Cn \lg n \leq \lg n! \tag{9.7.3}$$

for all but finitely many integers $n$. Combining (9.7.1) through (9.7.3), we obtain

$$Cn \lg n \leq f(n)$$

for all but finitely many integers $n$. Therefore,

$$f(n) = \Omega(n \lg n).$$

Theorem 7.3.10 states that merge sort (Algorithm 7.3.8) uses $\Theta(n \lg n)$ comparisons in the worst case and is, by Theorem 9.7.3, optimal. Many other sorting algorithms are known that also attain the optimal number $\Theta(n \lg n)$ of comparisons; one, tournament sort, is described before Exercise 14.



## Section Review Exercises

1. What is a decision tree?
2. How is the height of a decision tree that represents an algorithm related to the worst-case time of the algorithm?
3. Use decision trees to explain why worst-case sorting requires at least $\Omega(n \lg n)$ comparisons.

## Exercises

1. Four coins are identical in appearance, but one coin is either heavier or lighter than the others, which all weigh the same. Draw a decision tree that gives an algorithm that identifies in at most two weighings the bad coin (but not necessarily determines whether it is heavier or lighter than the others) using only a pan balance.

**2.** Show that at least two weighings are required to solve the problem of Exercise 1.

**3.** Eight coins are identical in appearance, but one coin is either heavier or lighter than the others, which all weigh the same. Draw a decision tree that gives an algorithm that identifies in at most three weighings the bad coin and determines whether it is heavier or lighter than the others using only a pan balance.

4. Twelve coins are identical in appearance, but one coin is either heavier or lighter than the others, which all weigh the same. Draw a decision tree that gives an algorithm that identifies in at most three weighings the bad coin and determines whether it is heavier or lighter than the others using only a pan balance.

**5.** What is wrong with the following argument, which supposedly shows that the twelve-coins puzzle requires at least four weighings in the worst case if we begin by weighing four coins against four coins?

If we weigh four coins against four coins and they balance, we must then determine the bad coin from the remaining four coins. But the discussion in this section showed that determining the bad coin from among four coins requires at least three weighings in the worst case. Therefore, in the worst case, if we begin by weighing four coins against four coins, the twelve-coins puzzle requires at least four weighings.

**★6.** Thirteen coins are identical in appearance, but one coin is either heavier or lighter than the others, which all weigh the same. How many weighings in the worst case are required to find the bad coin and determine whether it is heavier or lighter than the others using only a pan balance? Prove your answer.

7. Solve Exercise 6 for the fourteen-coins puzzle.

**8.** $(3^n - 3)/2$, $n \geq 2$, coins are identical in appearance, but one coin is either heavier or lighter than the others, which all weigh the same. [Kurosaka] gave an algorithm to find the bad coin and determine whether it is heavier or lighter than the others using only a pan balance in $n$ weighings in the worst case. Prove that the coin cannot be found and identified as heavy or light in fewer than $n$ weighings.

*Exercises 9 and 10 concern the following variant of the coin-weighing problem. We are given n coins, some of which are bad, but are otherwise identical in appearance. All of the good coins have the same weight. All of the bad coins also have the same weight, but they are lighter than the good coins. We assume that there is at least one bad coin and at least one good coin among the n coins. The task is to determine the number of bad coins.*

9. Show that at least $\log_3(n-1)$ weighings are necessary to determine the number of bad coins.

**10.** Show how to determine the number of bad coins in at most $n - 1$ weighings.

**11.** Give an algorithm that sorts four items using five comparisons in the worst case.

12. Use decision trees to find a lower bound on the number of comparisons required to sort five items in the worst case. Give an algorithm that uses this number of comparisons to sort five items in the worst case.

**13.** Use decision trees to find a lower bound on the number of comparisons required to sort six items in the worst case. Give an algorithm that uses this number of comparisons to sort six items in the worst case.

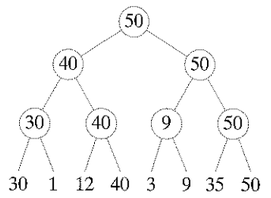
*Exercises 14–20 refer to tournament sort.*

*Tournament Sort. We are given a sequence*

$$s_1, \quad \ldots, \quad s_{2^k}$$

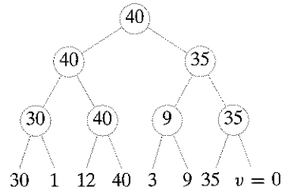
*to sort in nondecreasing order.*

*We will build a binary tree with terminal vertices labeled $s_1, \ldots, s_{2^k}$. An example is shown.*




*Working left to right, create a parent for each pair and label it with the maximum of the children. Continue in this way until you reach the root. At this point, the largest value, m, has been found.*

*To find the second-largest value, first pick a value v less than all the items in the sequence. Replace the terminal vertex w containing m with v. Relabel the vertices by following the path from w to the root, as shown. At this point, the second-largest value is found. Continue until the sequence is ordered.*




14. Why is the name "tournament" appropriate?
15. Draw the two trees that would be created after the preceding tree when tournament sort is applied.
16. How many comparisons does tournament sort require to find the largest element?
17. Show that any algorithm that finds the largest value among $n$ items requires at least $n - 1$ comparisons.
18. How many comparisons does tournament sort require to find the second-largest element?
19. Write tournament sort as a formal algorithm.
20. Show that if $n$ is a power of 2, tournament sort requires $\Theta(n \lg n)$ comparisons.
21. Give an example of a real situation (like that of Figure 9.7.1) that can be modeled as a decision tree. Draw the decision tree.
22. Draw a decision tree that can be used to determine who must file a federal tax return.
23. Draw a decision tree that gives a reasonable strategy for playing blackjack (see, e.g., [Ainslie]).

## 9.8 ➔ Isomorphisms of Trees

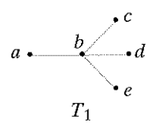
In Section 8.6 we defined what it means for two graphs to be isomorphic. (You might want to review Section 8.6 before continuing.) In this section we discuss isomorphic trees, isomorphic rooted trees, and isomorphic binary trees.

Corollary 8.6.5 states that simple graphs $G_1$ and $G_2$ are isomorphic if and only if there is a one-to-one, onto function $f$ from the vertex set of $G_1$ to the vertex set of $G_2$ that preserves the adjacency relation in the sense that vertices $v_i$ and $v_j$ are adjacent in $G_1$ if and only if the vertices $f(v_i)$ and $f(v_j)$ are adjacent in $G_2$. Since a (free) tree is a simple graph, trees $T_1$ and $T_2$ are isomorphic if and only if there is a one-to-one, onto function $f$ from the vertex set of $T_1$ to the vertex set of $T_2$ that preserves the adjacency relation; that is, vertices $v_i$ and $v_j$ are adjacent in $T_1$ if and only if the vertices $f(v_i)$ and $f(v_j)$ are adjacent in $T_2$.

**Example 9.8.1 ▸** The function $f$ from the vertex set of the tree $T_1$ shown in Figure 9.8.1 to the vertex set of the tree $T_2$ shown in Figure 9.8.2 defined by

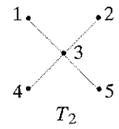
$$f(a) = 1, \quad f(b) = 3, \quad f(c) = 2, \quad f(d) = 4, \quad f(e) = 5$$

is a one-to-one, onto function that preserves the adjacency relation. Thus the trees $T_1$ and $T_2$ are isomorphic.



$T_1$

Figure 9.8.1 A tree.



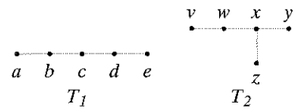
$T_2$

Figure 9.8.2 A tree isomorphic to the tree in Figure 9.8.1.

As in the case of graphs, we can show that two trees are not isomorphic if we can exhibit an invariant that the trees do not share.

**Example 9.8.2 ▸** The trees $T_1$ and $T_2$ of Figure 9.8.3 are not isomorphic because $T_2$ has a vertex ($x$) of degree 3, but $T_1$ does not have a vertex of degree 3.





$T_1$ $T_2$

Figure 9.8.3 Nonisomorphic trees. $T_2$ has a vertex of degree 3, but $T_2$ does not.

We can show that there are three nonisomorphic trees with five vertices. The three nonisomorphic trees are shown in Figures 9.8.1 and 9.8.3.

**Theorem 9.8.3** *There are three nonisomorphic trees with five vertices.*

***Proof*** We will give an argument to show that any tree with five vertices is isomorphic to one of the trees in Figure 9.8.1 or 9.8.3.

If $T$ is a tree with five vertices, by Theorem 9.2.3 $T$ has four edges. If $T$ had a vertex $v$ of degree greater than 4, $v$ would be incident on more than four edges. It follows that each vertex in $T$ has degree at most 4.

We will first find all nonisomorphic trees with five vertices in which the maximum vertex degree that occurs is 4. We will next find all nonisomorphic trees with five vertices in which the maximum vertex degree that occurs is 3, and so on.

Let $T$ be a tree with five vertices and suppose that $T$ has a vertex $v$ of degree 4. Then there are four edges incident on $v$ and, because of Theorem 9.2.3, these are all the edges. It follows that in this case $T$ is isomorphic to the tree in Figure 9.8.1.

Suppose that $T$ is a tree with five vertices and the maximum vertex degree that occurs is 3. Let $v$ be a vertex of degree 3. Then $v$ is incident on three edges, as shown in Figure 9.8.4. The fourth edge cannot be incident on $v$ since then $v$ would have degree 4. Thus the fourth edge is incident on one of $v_1$, $v_2$, or $v_3$. Adding an edge incident on any of $v_1$, $v_2$, or $v_3$ gives a tree isomorphic to the tree $T_2$ of Figure 9.8.3.

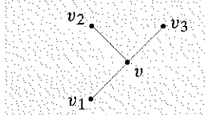



Figure 9.8.4 Vertex $v$ has degree 3.


Figure 9.8.5 Vertex $v$ has degree 2.

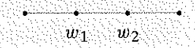



Figure 9.8.6 Adding a third edge to the graph of Figure 9.8.5.

Now suppose that $T$ is a tree with five vertices and the maximum vertex degree that occurs is 2. Let $v$ be a vertex of degree 2. Then $v$ is incident on two edges, as shown in Figure 9.8.5. A third edge cannot be incident on $v$; thus it must be incident on either $v_1$ or $v_2$. Adding the third edge gives the graph of Figure 9.8.6. For the same reason, the fourth edge cannot be incident on either of the vertices $w_1$ or $w_2$ of Figure 9.8.6. Adding the last edge gives a tree isomorphic to the tree $T_1$ of Figure 9.8.3.

Since a tree with five vertices must have a vertex of degree 2, we have found all nonisomorphic trees with five vertices.

If $n$ is even, say $n = 2k$, one can use induction to show (see Exercise 24) that when two $k$-vertex isomorphic binary trees are input to Algorithm 9.8.13, the number of comparisons is equal to $3n + 2$. Using this result, one can show (see Exercise 25) that if $n$ is odd, say $n = 2k + 1$, when the two binary trees shown in Figure 9.8.15 are input to Algorithm 9.8.13, the number of comparisons is equal to $3n + 1$. Thus the worst-case time of Algorithm 9.8.13 is $\Omega(n)$.

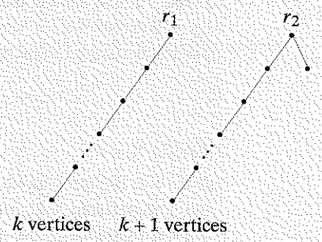



Figure 9.8.15 Two binary trees that give worst-case run time $3n + 1$ for Algorithm 9.8.13 when $n = 2k + 1$ is odd.

Since the worst-case time is $O(n)$ and $\Omega(n)$, the worst-case time of Algorithm 9.8.13 is $\Theta(n)$.

[Aho] gives an algorithm whose worst-case time is linear in the number of vertices that determines whether two arbitrary (not necessarily binary) rooted trees are isomorphic.

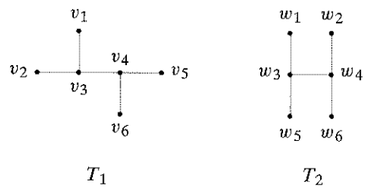
## Section Review Exercises

1. What does it mean for two free trees to be isomorphic?
2. What does it mean for two rooted trees to be isomorphic?
3. What does it mean for two binary trees to be isomorphic?
4. How many $n$-vertex, nonisomorphic binary trees are there?
5. Describe a linear-time algorithm to test whether two binary trees are isomorphic.

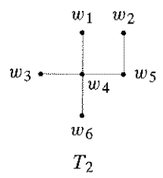
## Exercises

*In Exercises 1–6, determine whether each pair of free trees is isomorphic. If the pair is isomorphic, specify an isomorphism. If the pair is not isomorphic, give an invariant that one tree satisfies but the other does not.*

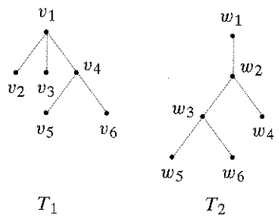
1.




2. $T_1$ as in Exercise 1

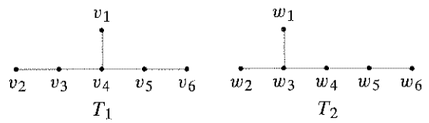



3.

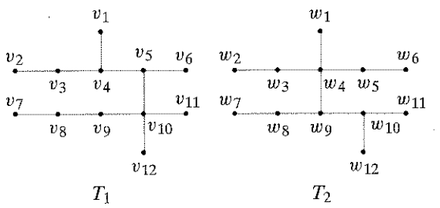




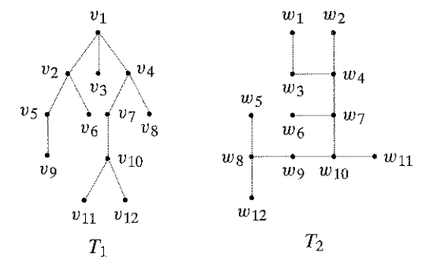

4.




5.

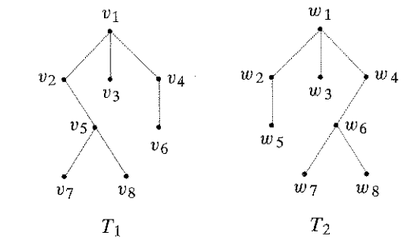



6.



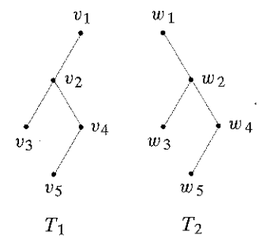

*In Exercises 7–9, determine whether each pair of rooted trees is isomorphic. If the pair is isomorphic, specify an isomorphism. If the pair is not isomorphic, give an invariant that one tree satisfies but the other does not. Also, determine whether the trees are isomorphic as free trees.*

7.




8. $T_1$ and $T_2$ as in Exercise 3

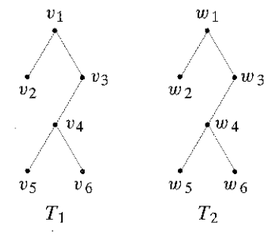
9.



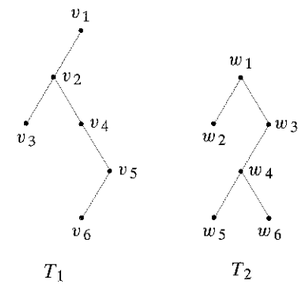

*In Exercises 10–12, determine whether each pair of binary trees is isomorphic. If the pair is isomorphic, specify an isomorphism. If the pair is not isomorphic, give an invariant that one tree satisfies but the other does not. Also, determine whether the trees are isomorphic as free trees or as rooted trees.*

10. $T_1$ and $T_2$ as in Exercise 9

11.




12.




13. Draw all nonisomorphic free trees having three vertices.
14. Draw all nonisomorphic free trees having four vertices.
15. Draw all nonisomorphic free trees having six vertices.
16. Draw all nonisomorphic rooted trees having three vertices.
17. Draw all nonisomorphic rooted trees having five vertices.
18. Draw all nonisomorphic binary trees having two vertices.
19. Draw all nonisomorphic binary trees having four vertices.
20. Draw all nonisomorphic full binary trees having seven vertices. (A full binary tree is a binary tree in which each internal vertex has two children.)
21. Draw all nonisomorphic full binary trees having nine vertices.
22. Find a formula for the number of nonisomorphic $n$-vertex full binary trees.
23. Find all nonisomorphic (as free trees and not as rooted trees) spanning trees for each graph in Exercises 7–9, Section 9.3.
24. Use induction to show that when two $k$-vertex isomorphic binary trees are input to Algorithm 9.8.13, the number of comparisons with *null* is equal to $6k + 2$.
25. Show that when the two binary trees shown in Figure 9.8.15 are input to Algorithm 9.8.13, the number of comparisons with *null* is equal to $6k + 4$.

**26.** Write an algorithm to generate an $n$-vertex random binary tree.

*In Exercises 27–33, $C_1, C_2, \ldots$ denotes the sequence of Catalan numbers. Let $X_1$ denote the set of nonisomorphic full binary trees having $n$ terminal vertices, $n \geq 2$, and let $X_2$ denote the set of nonisomorphic full binary trees having $n+1$ terminal vertices, $n \geq 1$, with one terminal vertex designated as "marked."*

27. Given an $(n-1)$-vertex binary tree $T, n \geq 2$, construct a binary tree from $T$ by adding a left child to every vertex in $T$ that does not have a left child, and adding a right child to every vertex in $T$ that does not have a right child. (A terminal vertex will add both a left and right child.) Show that this mapping is one-to-one and onto from the set of all nonisomorphic $(n-1)$-vertex binary trees to $X_1$. Conclude that $|X_1| = C_{n-1}$ for all $n \geq 2$.

**28.** Show that $|X_2| = (n+1)C_n$ for all $n \geq 1$.

*Given a tree $T \in X_1$, for each vertex $v$ in $T$, we construct two trees in $X_2$ as follows. One tree in $X_2$ is obtained by inserting two new children of $v$—one is a new left child, which is marked, and the other is the root of the original subtree in $T$ rooted at $v$. The other tree in $X_2$ is obtained by inserting two new children of $v$—one is a new right child, which is marked, and the other is the root of the original subtree in $T$ rooted at $v$. Let $X_T$ denote the set of all such trees constructed. This construction is due to Ira Gessel and was forwarded to the author by Arthur Benjamin.*

29. Show that $|X_T| = 2(2n-1)$ for all $T \in X_1$.

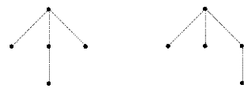
**30.** Show that if $T_1$ and $T_2$ are distinct trees in $X_1$, then $X_{T_1} \cap X_{T_2} = \varnothing$.

**31.** Show that $\bigcup_{T \in X_1} X_T = X_2$.

**32.** Use Exercises 29–31 to show that $(n+1)C_n = 2(2n-1)C_{n-1}$ for all $n \geq 2$. Exercise 26, Section 7.1, asked for a proof of this identity using the explicit formula for $C_n$. These exercises show a way to prove the identity without using the explicit formula for $C_n$.

33. Use Exercise 32 to give another derivation of the explicit formula for the $n$th Catalan number $C_n = C(2n, n)/(n+1)$. (See also Example 6.2.23.)

**34.** An *ordered tree* is a tree in which the order of the children is taken into account. For example, the ordered trees



are *not* isomorphic. Show that the number of nonisomorphic ordered trees with $n$ edges is equal to $C_n$, the $n$th Catalan number. *Hint:* Consider a preorder traversal of an ordered tree in which 1 means down and 0 means up.

**35.** [*Project*] Report on the formulas for the number of nonisomorphic free trees and for the number of nonisomorphic rooted trees with $n$ vertices (see [Deo]).

## 9.9 → Game Trees†

Trees are useful in the analysis of games such as tic-tac-toe, chess, and checkers, in which players alternate moves. In this section we show how trees can be used to develop game-playing strategies. This kind of approach is used in the development of many computer programs that allow human beings to play against computers or even computers against computers.

As an example of the general approach, consider a version of the game of nim. Initially, there are $n$ piles, each containing a number of identical tokens. Players alternate moves. A move consists of removing one or more tokens from any one pile. The player who removes the last token loses. As a specific case, consider an initial distribution consisting of two piles: one containing three tokens and one containing two tokens. All possible move sequences can be listed in a **game tree** (see Figure 9.9.1). The first player is represented by a box and the second player is represented by a circle. Each vertex shows a particular position in the game. In our game, the initial position is shown as $\binom{3}{2}$. A path represents a sequence of moves. If a position is shown in a square, it is the first player's move; if a position is shown in a circle, it is the second player's move. A terminal vertex represents the end of the game. In nim, if the terminal vertex is a circle, the first player removed the last token and lost the game. If the terminal vertex is a box, the second player lost.

The analysis begins with the terminal vertices. We label each terminal vertex with the value of the position to the first player. If the terminal vertex is a circle, since the first player lost, this position is worthless to the first player and we assign it the value 0 (see Figure 9.9.2). If the terminal vertex is a box, since the first player won, this position

†This section can be omitted without loss of continuity.



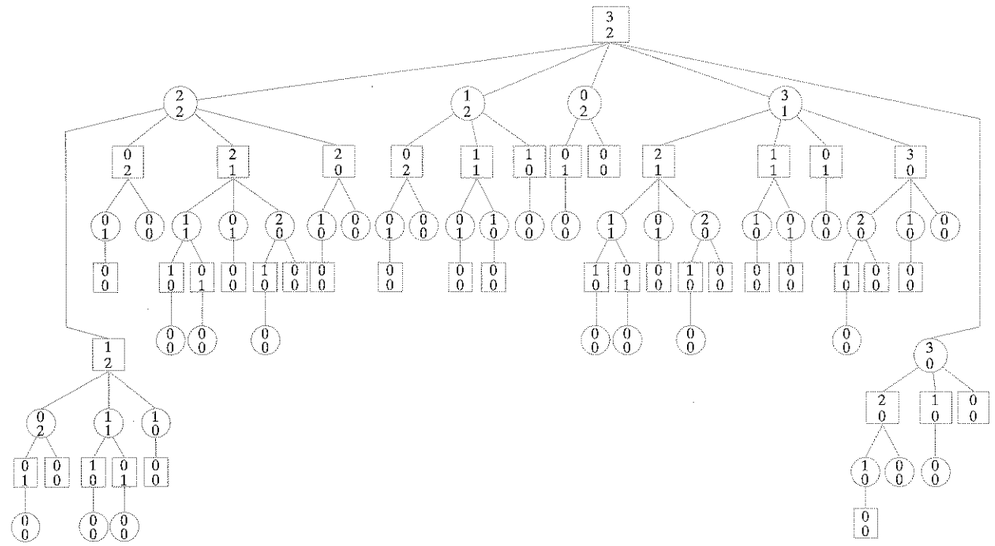


Figure 9.9.1 A game tree for nim. The initial distribution is two piles of three and two tokens, respectively.

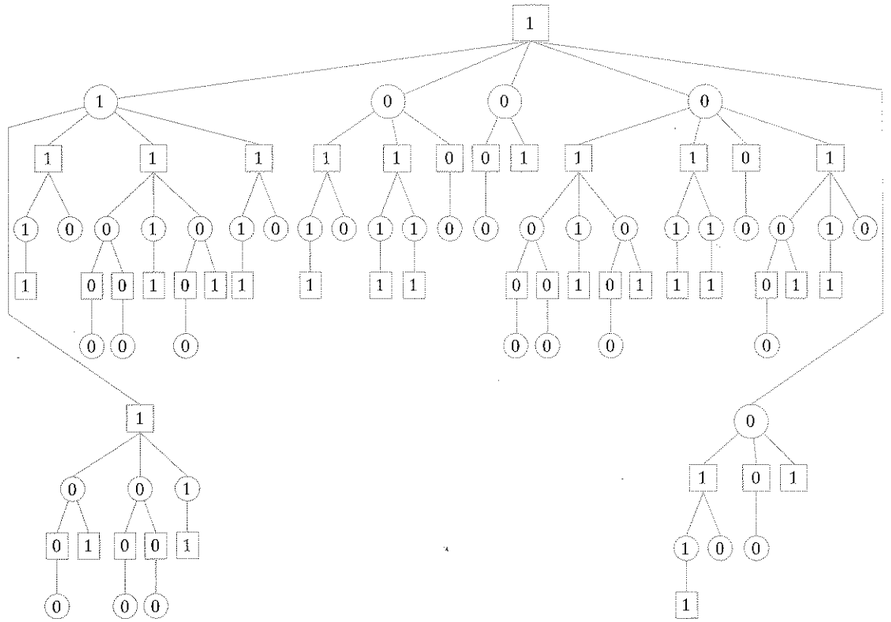


Figure 9.9.2 The game tree of Figure 9.9.1 showing the values of all vertices.