Problem 1

Prove that every nontrivial, nonempty tree has at least two leaves.

Hint: Begin by considering the set of all paths in the tree.

Proof idea

In a nontrivial, nonempty tree, there is at least a root and one other vertex and, so, at least one path. We show that the longest path is always between two leaves. Suppose this were false. Then the longest path in some tree is between two vertices, one of which is internal. Such a path can be extended into a longer path, contradicting our supposition that we chose the longest. Therefore, the longest path is always between two leaves. Since every nontrivial, nonempty tree has a longest path (or longest paths), there are at least two leaves.

Definitions

A tree is a connected, undirected graph with no simple circuits.

An empty tree is a tree with no vertices.

A trivial tree is a tree consisting of a single vertex.

A vertex is of degree $n$ if there are $n$ distinct edges connected to it.

A leaf is a vertex of degree 1.

Lemma: The longest path in any nontrivial, nonempty tree is between two leaves.

Proof. Suppose this were false. Then in some nontrivial, nonempty tree $T = \langle V, E \rangle$, the longest path $p$ is between some vertices $v$ and $w$ such that $v$ is of degree $n$ for $n \geq 2$.

Since $T$ is nonempty and nontrivial, there are, by definition, at least two distinct elements $v_1, v_2 \in V$ and there is at least one element $\langle v_1, v_2 \rangle \in E$.

Since $v$ is of degree $n$, there are at least 2 members of $E$, $\langle v, u \rangle$ and $\langle s, v \rangle$, such that $u \neq s$. Let $\langle v, x_1 \rangle, \langle x_1, x_2 \rangle, \ldots, \langle x_m, w \rangle$ represent $p$, so that the length of $p$ is $m$. Since, by definition, there are no cycles in $T$, either $\langle v, u \rangle$ or $\langle s, v \rangle$ does not appear on $p$. Suppose, without loss of generality, that $\langle s, v \rangle$ does not appear. Then we can then lengthen $p$ by adding $\langle s, v \rangle$ to the series: $\langle s, v \rangle, \langle v, x_1 \rangle, \langle x_1, x_2 \rangle, \ldots, \langle x_m, w \rangle$. This yields a path in $T$ of length $m + 1$, contradicting our assumption that $p$ was the longest path. □

Every nontrivial, nonempty tree has at least two leaves.

Since every nontrivial, nonempty tree has at least one path, every such tree also has a longest path. By the lemma, the longest path in every nontrivial, nonempty tree is between two leaves, so every such tree has at least two leaves.
Problem 2
Write a Turing machine (high level pseudo code) to decide the following language:

\[ L = \{ w \mid w \in \{0,1\}^* \text{ and } w \text{ contains at least twice as many 0's as 1's} \} \]

High level description
M = “On input \( w \):
1. Repeat the following 5 steps:
   2. Search left to right for the leftmost (unmarked) ‘1’.
      If no ‘1’ is found, accept.
   3. Mark ‘1’ as found.
   4. Return to start of string.
   5. Search left to right for two leftmost (unmarked) ‘0’s, and mark each as found.
      If two ‘0’s are not found, reject.
6. Return to start of string and go to step 1.”

Problem 3
Write a Turing machine (at the implementation level, i.e. define \( Q, \Sigma, \delta, \) etc.) to decide the language from problem 1:

\[ L = \{ w \mid w \in \{0,1\}^* \text{ and } w \text{ contains at least twice as many 0's as 1's} \} \]

Implementation

\[ M = \{ Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject} \} \]
\[ Q = \{ q_0, q_1, q_2, q_4, q_5, q_6, q_7, q_8, q_9 \} \]
\[ \Sigma = \{0,1\} \]
\[ \Gamma = \{B,0,1,a\} \]
\[ q_0 = q_7 \]
\[ q_{accept} = q_5 \]
\[ q_{reject} = q_6 \]
\[ \delta = \]

\[ \begin{align*}
< q_7,0 > & \rightarrow < q_8, B, R > \\
< q_7,1 > & \rightarrow < q_9, B, R > \\
< q_7, B > & \rightarrow < q_5, B, R >
\end{align*} \]
Marks head of tape with a ‘B’
and begins shifting every symbol to the right.

\[ \begin{align*}
< q_8,0 > & \rightarrow < q_8,0,R > \\
< q_8,1 > & \rightarrow < q_9,0,R > \\
< q_8, B > & \rightarrow < q_4,0,L >
\end{align*} \]
In shifting input to the right, \( q_8 \) indicates,
that the last symbol seen was a ‘0’.
In shifting input to the right, \( q_9 \) indicates, that the last symbol seen was a ‘1’.

Starting from the left, scans until finds a ‘1’, marks ‘1’ as an ‘a’ and transitions to state \( q_1 \). If no ‘1’ is found, machine accepts.

Returns tape head to beginning of input string to begin looking for two ‘0’s.

Scanning from the left, finds first ‘0’ on tape, marks ‘0’ as ‘a’ and transitions to state \( q_3 \). If no ‘0’ is found, machine rejects.

Finds second ‘0’ on tape from the left, marks ‘0’ as ‘a’ and transitions to state \( q_4 \). If no ‘0’ is found, machine rejects.

Returns tape head to beginning of input string, to begin looking for a ‘1’.
Problem 4

Now assume that the RIGHT movement in a Turing machine is replaced by DOUBLE-RIGHT in the Turing machine, its transition function now has the form:

\[ \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{RR, L\} \]

At each point, the machine can move its head right two steps, or move its head left one step. Is this Turing machine variant equivalent to the standard version? Prove why or why not.

Proof idea

The proof will show that (1) every RIGHT machine is equivalent to some DOUBLE-RIGHT machine, and (2) that every DOUBLE-RIGHT is equivalent to some RIGHT machine. This will be proved by showing how to transform a RIGHT machine into an equivalent DOUBLE-RIGHT machine, and vice versa.

To show (1), we systematically replace each of a machine’s RIGHT instructions with a set of instructions- move DOUBLE-RIGHT, then move LEFT. To show (2), we systematically replace each of a machine’s DOUBLE-RIGHT instructions with a set of instructions- move RIGHT, then move RIGHT again. In each case we add a new ‘dummy’ state for each altered instruction. The dummy state serves as the transition between the multiple moves needed to achieve the necessary configuration.

RIGHT to DOUBLE-RIGHT

Let \( M = < Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} > \) be a machine with \( n \geq 1 \) instructions of the form \( < q_i, x > \rightarrow < q_j, y, R > \). The following procedure will yield an equivalent machine \( N \) which has \( n - 1 \) such RIGHT instructions. (If \( M \) has no RIGHT instructions, then it is already a DOUBLE-RIGHT machine.)

Let \( < q_t, a > \rightarrow < q_a, b, R > \) be the first RIGHT instruction of \( M \).

Define \( N = < Q', \Sigma, \Gamma, \delta', q_0, q_{\text{accept}}, q_{\text{reject}} > \).

\( Q' = Q \cup \{q_t'\} \) (here \( q_t' \) is the dummy state).

\( \delta' \) is just like \( \delta \), except that RIGHT instruction above is replaced with the set of instructions:

- \( < q_t, a > \rightarrow < q_t', b, RR > \) and,
- \( < q_t', x > \rightarrow < q_u, x, L > \) for all \( x \in \Gamma \).

(So, one RIGHT instruction is replaced by a single DOUBLE-RIGHT instruction and \( |\Gamma| \) LEFT instructions.)

\( M \) is equivalent to \( N \)

Call \( M \)’s RIGHT first instruction I, and \( N \)’s different set of DOUBLE-RIGHT instructions J. We will prove equivalence by showing that after executions of I and J, M and N are in the same configuration. Equivalence follows because the only differences in operation will be isolated to the execution, not the result, of these particular instructions.
Proof. Since M and N are otherwise identical, suppose that prior to executing I or J, M and N are, respectively, in configurations $C_i = C'_i = s_0s_1 \ldots s_{(j-1)}qs_{(j+1)} \ldots s_k$. That is, in state $q_t$ reading symbol $a$.

On executing I, M replaces $a$ with $b$, transitions to state $q_u$, and moves to the right. So, after executing I, M is in configuration $C_{i+1} = s_0 \ldots s_{(j-1)}bq_u s_{(j+1)} \ldots s_k$.

On executing J, N replaces $a$ with $b$, transitions to state $q'_u$, and moves two places to the right to read $s_{j+2}$. Then leaves $s_{j+2}$ as it is, moves one place to the left, and transitions into state $q_u$. So, after executing J, N is in configuration $C'_{i+2} = s_0 \ldots s_{(j-1)}bq_u s_{(j+1)} \ldots s_k = C_{i+1}$. □

Therefore, M and N are equivalent. Since we made no assumptions about M and N, every RIGHT machine is equivalent to some machine with one fewer RIGHT instructions. By repeating this procedure for each RIGHT instruction of M, we prove M to be equivalent to some DOUBLE-RIGHT machine.

DOUBLE-RIGHT to RIGHT

Let M be a machine with $n \geq$ instructions of the form $<q_i,x> \rightarrow <q_j,y,RR>$. The following procedure will yield an equivalent machine N which has $n - 1$ such DOUBLE-RIGHT instructions. (If M has no DOUBLE-RIGHT instructions, then it is already a RIGHT machine.)

Let $<q_t,c> \rightarrow <q_m,d,RR>$ be the first DOUBLE-RIGHT instruction of M.

Define $N = <Q', \Sigma, \Gamma, \delta', q_0, q_{\text{accept}}, q_{\text{reject}}>$.

$Q' = Q \cup \{q'_l\}$.

$\delta'$ is just like $\delta$, except that RIGHT instruction above is replaced with the set of instructions:

$<q_t,c> \rightarrow <q'_l,d,R>$ and,

$<q'_l,x> \rightarrow <q_m,x,R>$ for all $x \in \Gamma$.

M is equivalent to N

The proof of equivalence is similar to the one given above.
Problem 5

Now assume that both RIGHT and LEFT are replaced by DOUBLE-RIGHT and DOUBLE-LEFT in the Turing machine, its transition function now has the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{RR, LL\}$$

At each point, the machine can move its head right two steps, or move its head left two steps. Is this Turing machine variant equivalent to the standard version? Prove why or why not.

Solution

Let language $L$ be defined:

$$L = \{11\}$$

$L$ can be decided by a LEFT/RIGHT machine, but not by a DOUBLE-LEFT/DOUBLE-RIGHT machine.

Proof Idea

We show that a LEFT/RIGHT machine can decide $L$ by example. We prove that no DOUBLE-LEFT/DOUBLE-RIGHT machine can decide $L$ on the basis that they are unable to read the second character of any input string, and so cannot distinguish ‘11’, which is in $L$, from ‘1’ which is not.

Some LEFT/RIGHT machine decides $L$

Let $M$ be a RIGHT-LEFT machine.

$$M = < Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject} >$$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{1\}$

$\Gamma = \{1, B\}$

$q_0 = q_0$

$q_{accept} = q_3$

$q_{reject} = q_4$

$$\delta = \text{M = “on input } w:\$$

$$< q_0, 1 > \to < q_1, 1, R > \quad 1. \text{ Check if first character is a ‘1’.}$$

$$< q_0, B > \to < q_4, B, R > \quad \text{ If not, reject.}$$

$$< q_1, 1 > \to < q_2, 1, R > \quad 2. \text{ Check if second character is a ‘1’.}$$

$$< q_1, B > \to < q_4, B, R > \quad \text{ If not, reject.}$$

$$< q_2, 1 > \to < q_4, 1, R > \quad 3. \text{ Check if third character is ‘B’.}$$

$$< q_2, B > \to < q_3, B, R > \quad \text{ If so, accept. If not, reject.”}$$
M decides $L$.

**Proof.** Given an input of the empty string, M halts in the rejecting state on the first instruction. Given an input of ‘1’, M halts in the rejecting state on the second instruction. Given an input of three or more ‘1’s, M halts in the rejecting state on the third instruction. Only given an input of ‘11’ does M halt in the accepting state. □

No DOUBLE-LEFT/DOUBLE-RIGHT machine decides $L$.

**Proof.** Let N be some DOUBLE-LEFT/DOUBLE-RIGHT machine. In order for N to distinguish strings ‘1’ (which is not in $L$) and ‘11’ (which is in $L$), N must read the second digit of its input. That is, if N’s starting configuration $C_0 = q_0 s_0 s_1 \ldots s_i$, then N must eventually be in $C_n = s_0 q_m s_1 \ldots s_k$.

Suppose there were some sequence of N’s instructions that would cause it to reject ‘1’ (or accept ‘11’) without entering configuration $C_n$. Then that sequence of instructions would also serve to reject ‘11’ (or accept ‘1’) and so would not decide the language.

A simple induction on N’s configurations shows that if $t$ is odd, then N is never in configuration $C_x = s_0 s_1 \ldots q_y s_t \ldots s_z$ for any $x, y, z$. This shows that N is never in configuration $C_n$ (as 1 is odd).

At $C_0$, N reads $s_0$, which is even. Suppose at $C_k$ for $k \geq 0$, N reads $s_j$ and $j$ is even. At $C_k$, N executes either an RR instruction or an LL instruction. If N executes an RR instruction, then at $C_{k+1}$ N reads $s_{j+2}$. If N executes an LL instruction, then N reads either $s_{j-2}$ or $s_j$ (if $j = 0$). Since $j$ was supposed to be even, $j + 2$ and $j - 2$ are both even. So, N is never in configuration $C_n$.

Therefore, N cannot distinguish between ‘1’ and ‘11’, and so cannot decide $L$. □