Chapter 3

Trees

In this chapter we deal with combinational circuits that have a topology of a tree. We begin by considering circuits for associative Boolean function. We then prove two lower bounds; one for cost and one for delay. These lower bounds do not assume that the circuits are trees. The lower bounds prove that trees have optimal cost and balanced trees have optimal delay.

3.1 Trees of associative Boolean gates

In this section, we deal with combinational circuits that have a topology of a tree. All the gates in the circuits we consider are instances of the same gate that implements an associative Boolean function.

3.1.1 Associative Boolean functions

Definition 15 *A Boolean function $f : \{0,1\}^2 \rightarrow \{0,1\}$ is* associative *if*

$$f(f(\sigma_1, \sigma_2), \sigma_3) = f(\sigma_1, f(\sigma_2, \sigma_3)),$$

for every $\sigma_1, \sigma_2, \sigma_3 \in \{0,1\}$.

Question 12 *List all the associative Boolean functions $f : \{0,1\}^2 \rightarrow \{0,1\}$.*

A Boolean function defined over the domain $\{0,1\}^2$ is often denoted by a dyadic operator, say $\odot$. Namely, $f(\sigma_1, \sigma_2)$ is denoted by $\sigma_1 \odot \sigma_2$. Associativity of a Boolean function $\odot$ is then formulated by

$$\forall \sigma_1, \sigma_2, \sigma_3 \in \{0,1\} \ : \ (\sigma_1 \odot \sigma_2) \odot \sigma_3 = \sigma_1 \odot (\sigma_2 \odot \sigma_3).$$

This implies that one may omit parenthesis from expressions involving an associative Boolean function and simply write $\sigma_1 \odot \sigma_2 \odot \sigma_3$. Thus we obtain a function defined over $\{0,1\}^n$ from a dyadic Boolean function. We formalize this composition of functions as follows.

Definition 16 *Let $f : \{0,1\}^2 \rightarrow \{0,1\}$ denote a Boolean function. The function $f_n : \{0,1\}^n \rightarrow \{0,1\}$, for $n \geq 2$ is defined by induction as follows.*

1. *If $n = 2$ then $f_2 \equiv f$ (the sign $\equiv$ is used instead of equality to emphasize equality of functions).*

2. *If $n > 2$, then f_n is defined based on f_{n-1} as follows:*

$$f_n(x_1, x_2, \ldots x_n) \triangleq f(f_{n-1}(x_1, \ldots, x_{n-1}), x_n).$$

If $f(x_1, x_2)$ is an associative Boolean function, then one could define f_n in many equivalent ways, as summarized in the following claim.

Claim 4 *If $f : \{0,1\}^2 \to \{0,1\}$ is an associative Boolean function, then*

$$f_n(x_1, x_2, \ldots x_n) = f(f_k(x_1, \ldots, x_k), f_{n-k}(x_{k+1}, \ldots, x_n)),$$

for every $k \in [2, n-2]$.

Question 13 *Show that the set of functions $f_n(x_1, \ldots, x_n)$ that are induced by associative Boolean functions $f : \{0,1\}^2 \to \{0,1\}$ is*

$$\{\textit{constant } 0, \textit{constant } 1, x_1, x_n, \text{AND}, \text{OR}, \text{XOR}, \text{NXOR}\}.$$

The implication of Question 13 is that there are only four non-trivial functions f_n (which?). In the rest of this section we will only consider the Boolean function OR. The discussion for the other three non-trivial functions is analogous.

3.1.2 OR-trees

Definition 17 *A combinational circuit $C = \langle \mathcal{G}, \mathcal{N} \rangle$ that satisfies the following conditions is called an* OR*-tree(n).*

1. ***Input:*** $x[n-1:0]$.

2. ***Output:*** $y \in \{0,1\}$

3. ***Functionality:*** $y = \text{OR}(x[0], x[1], \cdots, x[n-1])$.

4. ***Gates:*** *All the gates in $\mathcal{G}$ are* OR*-gates.*

5. ***Topology:*** *The underlying graph of $DG(C)$ (i.e. undirected graph obtained by ignoring edge directions) is a rooted binary tree.*

Consider the binary tree T corresponding to the underlying graph of $DG(C)$, where C is an OR-tree(n). The root of T corresponds to the output gate of C. The leaves of T correspond to the input gates of C, and the interior nodes in T correspond to OR-gates in C.

Claim 4 provides a "recipe" for implementing an OR-tree using OR-gates. Consider a rooted binary tree with n leaves. The inputs are fed via the leaves, an OR-gate is positioned in every node of the tree, and the output is obtained at the root. Figure 3.1 depicts two OR-tree(n) for $n = 4$.

One could also define an OR-tree(n) recursively, as follows.

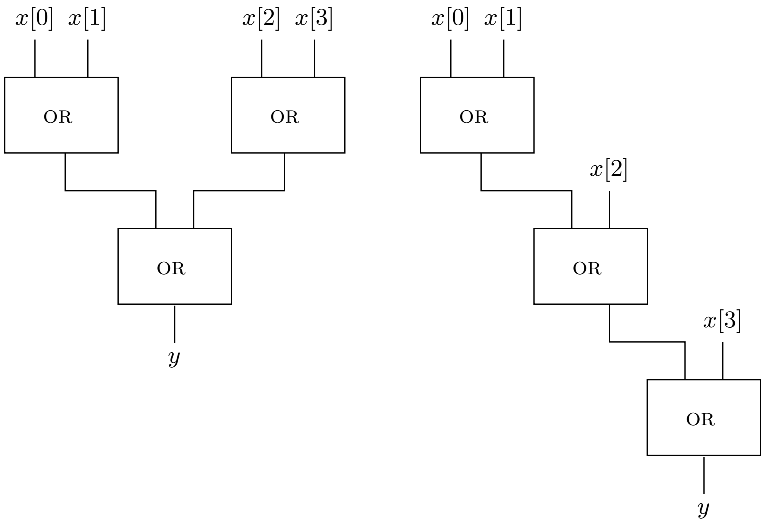

Figure 3.1: Two implementations of an OR-tree(n) with $n = 4$ inputs.

Definition 18 *An* OR-*tree*(n) *is defined recursively as follows (see Figure 3.2):*

1. *Basis: a single* OR-*gate is an* OR-*tree*(2).
2. *Step: an* OR(n)*-tree is a circuit in which*
   (a) *the output is computed by an* OR-*gate, and*
   (b) *the inputs of this* OR-*gate are the outputs of* OR-*tree*(n_1) *&* OR-*tree*(n_2)*, where* $n = n_1 + n_2$.

Question 14 *Design a zero-tester defined as follows.*

Input: $x[n-1:0]$.

Output: y

Functionality:

$$y = 1 \quad \textit{iff} \quad x[n-1:0] = 0^n.$$

1. *Suggest a design based on an* OR-*tree.*
2. *Suggest a design based on an* AND-*tree.*
3. *What do you think about a design based on a tree of* NOR-*gates?*

3.1.3 Cost and delay analysis

You may have noticed that both OR-trees depicted in Figure 3.1 contain three OR-gates. However, their delay is different. The following claim summarizes the the fact that all OR-trees have the same cost.

Claim 5 *The cost of every* OR-*tree*(n) *is* $(n-1) \cdot c(\text{OR})$.

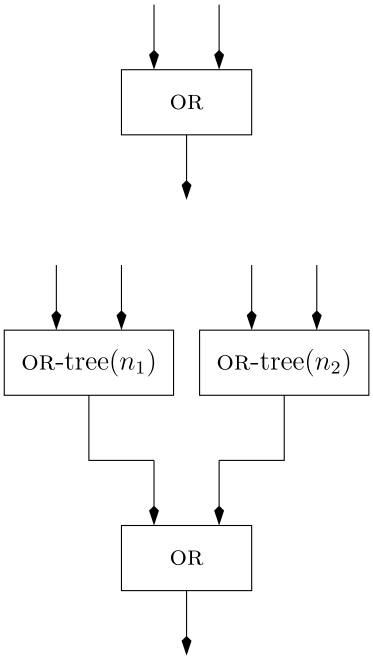

Figure 3.2: A recursive definition of an OR-tree(n).

Proof: The proof is by induction on n. The induction basis, for $n = 2$, follows because OR-tree(2) contains a single OR-gate. We now prove the induction step.

Let C denote an OR-tree(n), and let g denote the OR-gate that outputs the output of C. The gate g is fed by two wires e_1 and e_2. The recursive definition of OR-gate(n) implies the following. For $i = 1, 2$, the wire e_i is the output of C_i, where C_i is an OR-tree(n_i). Moreover, $n_1 + n_2 = n$. The induction hypothesis states that $c(C_1) = (n_1 - 1) \cdot c(\text{OR})$ and $c(C_2) = (n_2 - 1) \cdot c(\text{OR})$. We conclude that

$$\begin{aligned} c(C) &= c(g) + c(C_1) + c(C_2) \\ &= (1 + n_1 - 1 + n_2 - 1) \cdot c(\text{OR}) \\ &= (n - 1) \cdot c(\text{OR}), \end{aligned}$$

and the claim follows. □

The following question shows that the delay of an OR-tree(n) can be $\lceil \log_2 n \rceil \cdot t_{pd}(\text{OR})$, if a balanced tree is used.

Question 15 *This question deals with different ways to construct balanced trees. The goal is to achieve a depth of* $\lceil \log_2 n \rceil$.

1. *Prove that if* T_n *is a rooted binary tree with* n *leaves, then the depth of* T_n *is at least* $\lceil \log_2 n \rceil$.

2. *Assume that* n *is a power of* 2. *Prove that the depth of a complete binary tree with* n *leaves is* $\log_2 n$.

3. *Prove that for every $n > 2$ there exists a pair of positive integers a, b such that (1) $a + b = n$, and (2) $\max\{\lceil\log_2 a\rceil, \lceil\log_2 b\rceil\} \leq \lceil\log_2 n\rceil - 1$.*

4. *Consider the following recursive algorithm for constructing a binary tree with $n \geq 2$ leaves.*

   (a) *The case that $n \leq 2$ is trivial (two leaves connected to a root).*

   (b) *If $n > 2$, then let a, b be* any *pair of positive integers such that (i) $n = a + b$ and (ii) $\max\{\lceil\log_2 a\rceil, \lceil\log_2 b\rceil\} \leq \lceil\log_2 n\rceil - 1$. (Such a pair exists by the previous item.)*

   (c) *Compute trees T_a and T_b. Connect their roots to a new root to obtain T_n.*

   Prove that the depth of T_n is at most $\lceil\log_2 n\rceil$.

3.2 Optimality of trees

In this section we deal with the following questions: What is the best choice of a topology for a combinational circuit that implements the Boolean function OR_n? Is a tree indeed the best topology? Perhaps one could do better if another implementation is used? (Say, using other gates and using the inputs to feed more than one gate.)

We attach two measures to every design: cost and delay. In this section we prove lower bounds on the cost and delay of every circuit that implements the Boolean function OR_n. These lower bounds imply the optimality of using balanced OR-trees.

3.2.1 Definitions

In this section we present a few definitions related to Boolean functions.

Definition 19 (restricted Boolean functions) *Let $f : \{0,1\}^n \to \{0,1\}$ denote a Boolean function. Let $\sigma \in \{0,1\}$. The Boolean function $g : \{0,1\}^{n-1} \to \{0,1\}$ defined by*

$$g(w_0, \ldots, w_{n-2}) \triangleq f(w_0, \ldots, w_{i-1}, \sigma, w_i, \ldots, w_{n-2})$$

is called the restriction *of f with $x_i = \sigma$. We denote it by $f_{\restriction x_i=\sigma}$.*

Definition 20 (cone of a Boolean function) *A boolean function $f : \{0,1\}^n \to \{0,1\}$* depends *on its ith input if*

$$f_{\restriction x_i=0} \not\equiv f_{\restriction x_i=1}.$$

The cone *of a Boolean function f is defined by*

$$\text{cone}(f) \triangleq \{i : f_{\restriction x_i=0} \neq f_{\restriction x_i=1}\}.$$

The following claim is trivial.

Claim 6 *The Boolean function* OR_n *depends on all its inputs, namely*

$$|\text{cone}(\text{OR}_n)| = n.$$

Example 2 *Consider the following Boolean function:*

$$f(\vec{x}) = \begin{cases} 0 & \text{if } \sum_i x[i] < 3 \\ 1 & \text{otherwise.} \end{cases}$$

Suppose that one reveals the input bits one by one. As soon as 3 *ones are revealed, one can determine the value of* $f(\vec{x})$*. Nevertheless, the function* $f(\vec{x})$ *depends on all its inputs!*

The following trivial claim deals with the case that $cone(f) = \emptyset$.

Claim 7 $\text{cone}(f) = \emptyset \Longleftrightarrow f$ *is a constant Boolean function.*

3.2.2 Lower bounds

The following claim shows that, if a combinational circuit C implements a Boolean function f, then there must be a path in $DG(C)$ from every input in $cone(f)$ to the output of f.

Claim 8 *Let* $C = \langle \mathcal{G}, \mathcal{N} \rangle$ *denote a combinational circuit that implements a Boolean function* $f : \{0,1\}^n \to \{0,1\}$*. Let* $g_i \in \mathcal{G}$ *denote the input gate that feeds the* i*th input. If* $i \in \text{cone}(f)$*, then there is a path in* $DG(C)$ *from* g_i *to the output gate of* C*.*

Proof: If $DC(C)$ lacks a path from the input gate g_i that feeds an input $i \in cone(f)$ to the output y of C, then C cannot implement the Boolean function f. Consider an input vector $w \in \{0,1\}^{n-1}$ for which $f_{\restriction x_i=0}(w) \neq f_{\restriction x_i=1}(w)$. Let w' (resp., w'') denote the extension of w to n bits by inserting a 0 (resp., 1) in the ith coordinate. The proof of the Simulation Theorem of combinational circuits (Theorem 2) implies that C outputs the same value when given the input strings w' and w'', and hence C does not implement f, a contradiction. □

The following theorem shows that every circuit, that implements the Boolean function OR_n in which the fan-in of every gate is bounded by two, must contain at least $n-1$ non-trivial gates (a trivial gate is an input gate, an output gate, or a gate that feeds a constant). We assume that the cost of every non-trivial gate is at least one, therefore, the theorem is stated in terms of cost rather than counting non-trivial gates.

Theorem 9 (Linear Cost Lower Bound Theorem) *Let* C *denote a combinational circuit that implements a Boolean function* f*. Then*

$$c(C) \geq |\text{cone}(f)| - 1.$$

Before we prove Theorem 9 we show that it implies the optimality of OR-trees. Note that it is very easy to prove a lower bound of $n/2$. The reason is that every input must be fed to a non-trivial gate, and each gate can be fed by at most two inputs.

Corollary 10 *Let* C_n *denote a combinational circuit that implements* OR_n *with input length* n*. Then*

$$c(C_n) \geq n - 1.$$

Proof: Follows directly from Claim 6 and Theorem 9. □

We prove Theorem 9 by considering the directed acyclic graph (DAG) $DG(C)$. We use the following terminology for DAGs: The *in-degree* (resp., *out-degree*) of a vertex is the number of edges that enter (resp., emanate from) the vertex. A *source* is a vertex with in-degree zero. A *sink* is a vertex with out-degree zero. An *interior vertex* is a vertex that is neither a source or a sink. See Figure 3.3 for an example.

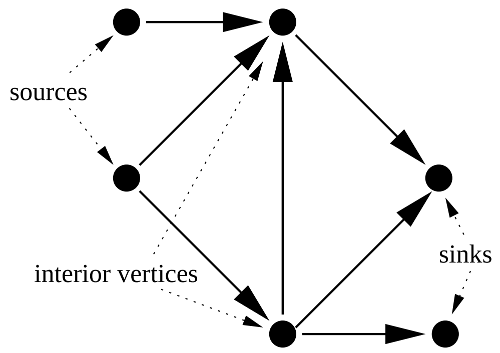

Figure 3.3: A DAG with two sources, two interior vertices, and two sinks.

Proof of Theorem 9: If the underlying graph of $DG(C) = (V, E)$ is a rooted binary tree, then in this tree we we know that

$$|\text{interior vertices}| \geq |\text{leaves}| - 1.$$

Recall that leaves are input gates and that every interior vertex is a non-trivial gate. Hence, the case of a tree follows.

If the underlying graph of $DG(C)$ is not a tree, then we construct a DAG $T = (V', E')$ that is a subgraph of $DG(C)$ such that (i) the sources in V' are all the input gates that feed inputs x_i such that $i \in cone(f)$, (ii) the output gate is the sink, and (iii) the underlying graph of T is a rooted binary tree.

The DAG T is constructed as follows. Pick a source $v \in V$ that feeds an input x_i such that $i \in cone(f)$. By Claim 8, there is a path in $DG(C)$ from v to the output gate. Add all the edges and vertices of this path to T. Now continue in this manner by picking, one by one, sources that feed inputs x_i such that $i \in cone(f)$. Each time consider a path p that connects the source to the output gate. Add the prefix of the path p to T up to the first vertex that is already contained in T. We leave it as an exercise to show that T meets the three required conditions.

In the underlying graph of T we have the inequality $|\text{interior vertices}| \geq |\text{leaves}| - 1$. The interior vertices of T are also interior vertices of $DG(C)$, and the theorem follows. □

Question 16 *State and prove a generalization of Theorem 9 for the case that the fan-in of every gate is bounded by a constant c.*

We now turn to proving a lower bound on the delay of a combinational circuit that implements OR_n. Again, we will use a general technique. Again, we will rely on all gates in the design having a constant fan-in.

The following theorem shows a lower bound on the delay of combinational circuits that is logarithmic in the size of the cone.

Theorem 11 (Logarithmic Delay Lower Bound Theorem) *Let $C = \langle \mathcal{G}, \mathcal{N} \rangle$ denote a combinational circuit that implements a non-constant Boolean function $f\{0,1\}^n \rightarrow \{0,1\}$. If the fan-in of every gate in $\mathcal{G}$ is at most c, then the delay of C is at least* $\log_c |\text{cone}(f)|$.

Before we prove Theorem 11, we show that the theorem implies a lower bound on the delay of combinational circuits that implement OR_n.

Corollary 12 *Let C_n denote a combinational circuit that implements OR_n. Let c denote the maximum fan-in of a gate in C_n. Then*

$$t_{pd}(C_n) \geq \lceil \log_c n \rceil .$$

Proof: The corollary follows directly from Claim 6 and Theorem 11. □

Proof of Theorem 11: The proof deals only with the graph $DG(C)$ and shows that there must be a path with at least $\log_c |cone(f)|$ interior vertices in $DG(C)$. Note that input/output gates and constants have zero delay, so we have to be careful not to count them. However, zero delay vertices can appear only as end-points of a path; this is why we count interior vertices along paths.

Claim 8 implies there must be a path in $DG(C)$ from every input $x_i \in cone(f)$ to the output y of C.

The proof involves strengthening the theorem to every vertex v as follows. We attach to every vertex v in the DAG a subset of sources $cone(v)$ that is the set of sources from which v is reachable. Let $d(v)$ denote the maximum number of interior vertices along a path from a source in $cone(v)$ to v not including v. We now prove that, for every vertex v,

$$d(v) \geq \log_c |cone(v)|. \tag{3.1}$$

We prove Equation 3.1 by induction on $d(v)$. The induction basis, for $d(v) = 0$, is trivial since $d(v) = 0$ implies that v is a source. The cone of a source v consists v itself, and $\log 1 = 0$.

The induction hypothesis is

$$d(v) \leq i \quad \Longrightarrow \quad d(v) \geq \log_c |cone(v)|. \tag{3.2}$$

In the induction step, we wish to prove that the induction hypothesis implies that Equation 3.2 holds also if $d(v) = i + 1$. Consider a vertex v with $d(v) = i + 1$. There are at most c edges that enter v. Denote the vertices that precede v by $v_1, \ldots, v_c'$, where $c' \leq c$. Namely, the edges that enter v are $v_1 \rightarrow v, \ldots, v_{c'} \rightarrow v$. By definition,

$$d(v) = \max\{d(v_i)\}_{i=1}^{c'} + 1. \tag{3.3}$$

Since v is not a source, it follows by definition that

$$cone(v) = \bigcup_{i=1}^{c'} cone(v_i).$$

Hence

$$
\begin{aligned}
|cone(v)| &\leq \sum_{i=1}^{c'} |cone(v_i)| \\
&\leq c' \cdot \max\{|cone(v_i)|\}_{i=1}^{c'}. \qquad (3.4)
\end{aligned}
$$

Let v' denote a predecessor of v that satisfies $|cone(v')| = \max\{|cone(v_i)|\}_{i=1}^{c'}$. The induction hypothesis implies that

$$d(v') \geq \log_c |cone(v')|. \qquad (3.5)$$

But,

$$
\begin{aligned}
d(v) &\geq 1 + d(v') && \text{by Eq. 3.3} \\
&\geq 1 + \log_c |cone(v')| && \text{by Eq. 3.5} \\
&\geq 1 + \log_c |cone(v)|/c' && \text{by Eq. 3.4} \\
&\geq \log_c |cone(v)|,
\end{aligned}
$$

and the theorem follows. □

Question 17 *The proof of the Theorem 11 dealt with the longest path from an input vertex to an output vertex. In the proof, we strengthened this statement by considering every vertex instead of only the sink. In this question we further strengthen the conditions at the expense of a slightly weaker lower bound. We consider the longest shortest path from a set of vertices U to a given vertex r (i.e. $\max_{u \in U} \text{dist}(u, r)$).*

Prove the following statement. Let $U \subseteq V$ denote a subset of vertices of a directed graph $G = (V, E)$, and let $r \in V$. There exists a vertex $u \in U$ such that $\text{dist}(u, r) \geq \Omega(\log_c |U|)$, where c denotes the maximum degree of G. ($\text{dist}(u, r)$ denotes the length of the shortest path from u to r; if there is no such path, then the distance is infinite.)

3.3 Summary

In this chapter we started by considering associative Boolean functions. We showed how associative dyadic functions are extended to n arguments. We argued that there are only four non-trivial associative Boolean functions; and we decided to focus on OR_n. We then defined an OR-tree(n) to be a combinational circuit that implements OR_n using a topology of a tree.

Although it is intuitive that OR-trees are the cheapest designs for implementing OR_n, we had to work a bit to prove it. It is also intuitive that balanced OR-trees are the fastest designs for implementing OR_n, and again, we had to work a bit to prove that too.

We will be using the lower bounds that we proved in this chapter also in the next chapters. To prove these lower bounds, we introduced the term $cone(f)$ for a Boolean function f. The cone of f is the set of inputs the function f depends on.

If all the gates have a fan-in of at most 2, then the lower bounds are as follows. The first lower bound states that the number of gates of a combinational circuit implementing a Boolean function f must be at least $|cone(f)| - 1$. The second lower bound states that the propagation delay of a circuit implementing a Boolean function f is at least $\log_2 |cone(f)|$.