Problem 1

Show that every \( n \)-bit boolean function \( f : \{0,1\}^n \to \{0,1\} \) can be represented by a circuit of depth at most \( O(n) \).

Solution 1

We will prove the stronger statement that it can be represented by a circuit of depth at most \( 2 \cdot n \) by induction on \( n \).

**Base case.** If \( n = 1 \), we just need to represent the four functions in \( \{0,1\} \to \{0,1\} \), that is, the identity, negation, constant-0 and constant-1 function in a circuit of depth \( \leq 2 \). We can represent them with the pass-through circuit, a single NOT gate, AND(\( \text{NOT}(x_0), x_0 \)) and OR(\( \text{NOT}(x_0), x_0 \)) respectively; evidently, each of these is of depth \( \leq 2 \).

**Induction step.** Suppose we can represent every boolean function on \( n - 1 \) bits in a circuit of depth at most \( 2 \cdot n - 2 \). If we set the first bit in an \( n \)-bit boolean function to a fixed value, this essentially gives rise to an \( n - 1 \)-bit function; moreover, there are only two possible values to which we could set it, so any \( n \)-bit boolean function \( f \) can be equivalently expressed as \((x_1 \land f_1(x_2, \ldots, x_n)) \lor (\lnot x_1 \land f_0(x_2, \ldots, x_n))\), where \( f_1 \) and \( f_2 \) each are \( n - 1 \) bit boolean functions and therefore representable by circuits of depth \( \leq 2 \cdot n - 2 \) by induction hypothesis.

We may now build a circuit representing this formula containing the circuits for \( f_0 \) and \( f_1 \) as subcircuits, i.e. \( \text{OR}(\text{AND}(x_1, C_{f_1}), \text{AND}(\text{NOT}(x_1), C_{f_0})) \). The depth of this circuit then evidently is \( = \max\{2, 2 + \text{depth}(C_{f_1}), 3, 2 + \text{depth}(C_{f_0})\} \leq 2 + 2 \cdot n - 2 = 2 \cdot n \) as required.
Solution 2

For every row in the truth table of a boolean function that has an output value of 1, we can certainly construct a circuit of depth $O(n)$ that is 1 for the inputs in that row and 0 for all other tuples of inputs – just AND together the inputs or their negations depending on whether the respective entries in the row in question are 1 or 0. Now, there are at most $2^n$ such rows, so we can instantiate such a circuit for every single row corresponding to an output value of 1 and then proceed to OR them together with a balanced binary tree of OR gates, which will have a depth of at most $\log_2 n = n$. Since the longest path from the output bit to an input bit will traverse the binary tree and exactly one of the circuits generated for a single row, the depth of the entire construction will be at most $n + O(n)$ which is $O(n)$ as required.

Problem 2

Assume that in addition to the function inputs, you are provided a wire which is always TRUE (1).

Show that every circuit can then also be implemented using only NOR gates (instead of NOT, AND, OR). Show also that this does not affect the asymptotic size or depth of the circuit.

Solution

It suffices to show that each of the three gates can be replaced by a functionally identical circuit constructed from NOR gates (i.e. one with the same number of inputs and outputs). It is easily checked from the definitions of size and depth that the maximum of the sizes and depths respectively of each of those circuits – which is necessarily a finite constant as there are only finitely many of them – then provides a constant upper bound on the multiplicative increase of the size and depth of the overall circuit if each gate is replaced by its equivalent NOT circuit.

We can represent the NOT gate, operating on a single input $x$, with $\text{NOR}(x,x)$, the AND gate on inputs $x$ and $y$ with $\text{NOR}(\text{NOR}(x,x),\text{NOR}(y,y))$ and the OR gate on inputs $x$ and $y$ with $\text{NOR}(\text{NOR}(x,y),\text{NOR}(x,y))$. That these circuits are functionally equivalent follows from an easy check.

Problem 3

The function $\text{Parity}: \{0, 1\}^n \to \{0, 1\}$ returns 1 if and only if an odd number of its inputs are 1.
Show that Parity can be represented by a circuit of size $O(n)$ and depth $O(\log n)$.

**Solution**

It is easily seen (*) that calculating the parity of both halves of an arbitrary partitioning of an $n$-bit vector and outputting the XOR of the two results is equivalent to calculating the parity of the entire $n$-bit vector directly, as the sum of two numbers (of 1s, here) is odd if and only if exactly one of them is odd; it is furthermore self-evident (**) that the parity of a single bit is just that bit.

**Claim.** For $n = 2^m$ for some $m$, the full binary tree of XOR circuits with $n$ leaves, connected to the inputs, calculates Parity$_n$.

**Proof of claim.** We will show by induction on the depth of binary subtrees within the tree that each of them calculates Parity on the leaves under it. The base case, for the wires connecting the leaves to the inputs, is given by (**): each inductive step is essentially (*), as the root node of a depth-$k$ subtree calculates the XOR of the outputs of the two depth-$k-1$ subtrees below it, whose inputs partition the inputs under the depth-$k$ tree’s root. □

It is a well-known fact that said tree has $2n-1$ nodes (and hence $c \cdot (2n-1)$ gates, where $c$ is the number of gates in the XOR circuit), and a depth of $m = \log n$.

Finally, to calculate Parity of $n$ bits, where $n$ is arbitrary, we simply pad the input up to a power of 2, say $n' \leq 2 \cdot n$, using constant-size $s$ and -depth $d$ circuits generating constant zeroes, and use the above construction on the inputs and the padding bits. The resulting circuit will have $c \cdot (2n'-1) \leq c \cdot (4n-1) + s \cdot n = O(n)$ gates and depth $\leq d + \log n' = d + \log 2n = d + \log 2 + \log n = O(\log n)$ as required. □

**Problem 4**

A *labelled tree* with $n$ labels is a tree – that is, a connected graph without cycles – whose $n$ vertices are the integers from 1 up to $n$.

![Figure 1: Some drawings of labelled trees, not all distinct.](image)
Show that for a given $n$, there are no more than $n^n$ distinct labelled trees.

**Solution 1**

For any given tree, choose an arbitrary vertex to be the root and direct all edges to point away from it, e.g. by depth-first search; then every vertex apart from the root will have a unique parent/predecessor in the tree (as this not being the case would make the original, unrooted graph disconnected or containing a cycle). It’s clear that no two ‘plain’ trees will give rise to the same rooted tree, so $\#\text{trees} \leq \#\text{rooted trees}$; furthermore, setting the root’s predecessor to itself to distinguish it from the other nodes, it’s clear that we can reconstruct the rooted tree from the function that assigns each vertex label the label of its predecessor (as each vertex-predecessor relation defines a distinct edge, and the total $n - 1$ such relations we will get not counting the root must therefore define every edge we can possibly have in the tree), so $\#\text{rooted trees} \leq \#\text{predecessor functions} = \#\{f : [n] \to [n]\} = n^n$. □

Note that our choice of root was arbitrary among the $n$ vertices, so we are overcounting the labelled trees by at least a factor of $n$.

**Solution 2**

(Without a certain amount of information-theoretical magic, this proof only works for $n$ which are powers of 2.)

Root the tree as before, and write down the values of the predecessor function in order in $\log n$ bits for each entry, obtaining an $n \log n$-bit string. There are $2^{n \log n} = n^n$ strings of that length.

**Problem 5**

The function $\text{Majority}: \{0, 1\}^n \to \{0, 1\}$ returns 1 if at least $n/2$ of its inputs are 1, and 0 if fewer than $n/2$ of them are.

Show that $\text{Majority}$ can be implemented with a circuit of size $O(n^2)$.

**Solution**

Let $n$ be the number of inputs.

We can implement bubblesort as a circuit, noting that conveniently, comparing two bits and reordering them if necessary is rather easy – calling the inputs $x_1, x_2$ and the outputs $y_1, y_2$, we can write a subcircuit that accomplishes this as $y_1 = OR(x_1, x_2)$ and $y_2 = AND(x_1, x_2)$. This circuit has size 2.
Now, mimicking the action of the bubblesort algorithm on an array, we implement one step at position $i$ to be the $n$-input, $n$-output subcircuit that passes through all inputs $< i$ and $> i + 1$ on wires unchanged and uses the above compare-swap subcircuit on the $i$th and $i + 1$st input to generate the $i$th and $i + 1$st output. This still has size 2.

We now implement a pass as the serial concatenation (making one step’s output the next one’s input) of steps for each of $i = 1, \ldots, n - 1$; clearly, this has size $(n - 1) \cdot 2$.

Proceed to implement bubblesort as the serial concatenation of $n$ passes; this has size $n(n - 1) \cdot 2 = O(n^2)$ as required.

As the $n$ interim outputs are now sorted, the middle bit (rounded up as needed) will be 1 if and only if all of the bits at lower indices are also 1, and hence correspond to the value of $\text{MAJORITY}_n$. We can make it the overall output and possibly remove dangling outputs in whatever fashion we prefer depending on how pedantic we choose to be (for grading purposes, this will not happen).

Technically, it would also be necessary to prove the correctness of this “sorting algorithm”. As far as I recall, I accepted references to clear isomorphism to the universally known bubblesort algorithm in the solutions to this end; in general, this proof could for instance be conducted by showing by induction that after $k$ passes, the only way the $n - k + 1$st bit could still be 1 would be that every single bit to its left is 1 (e.g. after the first pass, the rightmost bit is just equal to the AND of all the inputs). Yes, this does mean that you can technically get away with doing $n/2$ passes, but that doesn’t change the asymptotic size anyway.