Homework 1: Solutions

Due:

1. In lecture we proved that any function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a branching program in the form of an $n+1$-layer binary tree, which has size 2^{n+1}. Form this construction for f, and remove the last t layers, leaving a binary tree of size 2^{n-t} whose leaf nodes do not have connections to output nodes. Each leaf node of this binary tree is represented by a sequence $x_1, x_2, ..., x_{n-t}$ of binary digits that represents the path taken to get to the node.

   Additionally, using the same construction, create a binary tree branching program of size 2^{t+1} for every possible function of t bits $\{0,1\}^t \to \{0,1\}$. For a fixed sequence $x_1, x_2, ..., x_{n-t}$ of binary digits, define $f_{x_1,x_2,...,x_{n-t}}$ to be the t-bit function defined by $f_{x_1,x_2,...,x_{n-t}}(x_{n-t+1}, ..., x_n) = f(x_1, x_2, ..., x_n)$.

   Now replace each leaf node in the truncated binary tree of height $n - t$ discussed above with the root node of one of these new trees, specifically replacing the leaf under the path $x_1, x_2, ..., x_{n-t}$ with the root node of the tree computing $f_{x_1,x_2,...,x_{n-t}}$.

   When the resulting branching program takes in a sequence $x_1, x_2, ..., x_n$, it first traverses to the root node for the tree computing $f_{x_1,x_2,...,x_{n-t}}$, and then takes the next t bits in as input to compute $f_{x_1,x_2,..,x_{n-t}}(x_{n-t+1}, ..., x_n) = f(x_1, x_2, ..., x_n)$. In other words, the resulting branching program computes f.

   This construction uses a binary tree of size 2^{n-t} and a further tree of size 2^{t+1} for each of the 2^{2^t} functions of t bits. Thus the construction uses

   $$O(2^{n-t} + 2^{2^t+t+1}) = O(2^{n-\log(n)+1} + 2^{\frac{n}{2}+\log(n)}) = O(2^{n-\log(n)}) = O(\frac{2^n}{n})$$

   nodes, noting that the second equality holds because $\frac{n}{2} + \log(n)$ is asymptotically less than $n - \log(n) + 1$.

   Common Errors/Issues.

   - Some solutions did not explicitly show how to construct the branching program for the function you want to compute. Explicitly showing it can definitely make your solution clearer.

2. Consider a branching program with k nodes and input size n. Each node is either labeled with an output from $\{0,1\}$ or labeled by an input variable from $\{x_1, ..., x_n\}$. In the latter case, there are two edges going out (with different labels) from that node, each pointing to one of the k nodes. Therefore, there are at most $2 + nk^2$ different configurations for a single node. And so there are at most $(2 + nk^2)^k$ different branching programs in total. Since a branching program contains at least one node for each input variable and an output node, we have $k \geq n + 1$, and therefore

   $$(2 + nk^2)^k \leq ((n+1)k^2)^k = O(k^{3k}) = O(2^{3k \log k}).$$

For $k = \frac{2^n}{cn}$, the number of different branching programs is bounded by

$$O(2^{3\cdot 2^n/cn\cdot\log(2^n/cn)}) = O(2^{3\cdot 2^n/cn\cdot\log(2^n)}) = O(2^{3\cdot 2^n/c}).$$

On the other hand, there are 2^{2^n} different functions from $\{0,1\}^n$ to $\{0,1\}$. For $c = 4$, the number of different branching programs is strictly less than 2^{2^n} for large enough n. Therefore, there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a branching program with less than $\frac{2^n}{4n}$ nodes.

Common Errors/Issues.

- When we consider the two edges going out from a single node in a branching program with s nodes, the number of arrangements should be upper bounded by s^2 instead of $\binom{s}{2}$ or $2\cdot\binom{s}{2}$ - the two edges have different labels and can go to the same node.
- When counting the number of possible choices for a node, we should consider the output nodes 0, 1. Those nodes don't have output edges, so there isn't a s^2 multiplied in the number. That is the number of choices for a node should be ns^2+2 instead of $(n+2)s^2$.

3. We count both (i) the number of functions on n bits, and (ii) the number of large fan-in circuits of size $2^{n/3}$. Comparing these two counts will give us the answer.

   For (ii), in lecture we saw that the number of different functions on n bits is 2^{2^n}. We now count (ii), the number of large fan-in circuits. Let $s = 2^{n/3}$ denote the size of the circuit. For each non-input gate, there are s other gates that could feed into it, resulting in a total of $\binom{s}{n/2}$ combinations of inputs. It will be convenient to use the bound $\binom{s}{n/2} \le (2es/n)^{n/2} \ll s^n$. Each gate can compute any function on $n/2$ input bits, and there are $2^{2^{n/2}}$ possible functions. For each input gate, there are n possible choices for the input variable. Therefore, the number of distinct circuits is at most

$$\left(2^{2^{n/2}}\cdot\binom{s}{n/2}+n\right)^s \le \left(2\cdot 2^{2^{n/2}}\cdot s^n\right)^s.$$

   For $s = 2^{n/3}$,

$$\left(2\cdot 2^{2^{n/2}}\cdot s^n\right)^s = 2^{2^{n/3}}\cdot 2^{2^{n/2}\cdot 2^{n/3}}\cdot 2^{2^{n/3}\cdot n\cdot\log\left(2^{n/3}\right)} = 2^{2^{n/3}+2^{5n/6}+2^{n/3}\cdot n^2/3}$$

   When n is large enough,

$$2^{2^{n/3}+2^{5n/6}+2^{n/3}\cdot n^2/3} \le 2^{2^{5n/6+2}}.$$

   Therefore, for large enough n we have

$$\begin{aligned}\text{number of } n\text{-bit functions} = 2^{2^n} &\gg 2^{2^{5n/6+2}}\\ &\gg \text{number of large fan-in circuits with } 2^{n/3} \text{ gates}.\end{aligned}$$

Common Errors/Issues.

- Some of you missed the $2^{2^{n/2}}$ term in the counting argument.
- Some ignored the possibility that the gate could be an input gate while counting.

4. Let $c(g)$ be the number of gates used to compute a gate g in the formula. We first prove that for any formula of size s there exists a gate g in the formula such that $\frac{1}{3}s \leq c(g) \leq \frac{2}{3}s + 1$. Let $(g_n) = g_1, g_2, \ldots$ be the sequence defined in the hint. By our definition, g_1 is always the output gate and $c(g_i) - c(g_{i+1}) \geq 1$, so this sequence have non-zero finite length. Since $c(g_1) = s$, there exists at least one gate g in the sequence with $c(g) \geq \frac{2}{3}s$. Let i be the largest index with $c(g_i) \geq 2s/3 + 1 > 5$, by the assumption that $s > 6$. g_i cannot be an output gate, since $c(g_i) > 1$. We claim that $\frac{1}{3}s \leq c(g_{i+1}) \leq \frac{2}{3}s + 1$. This is because $c(g_{i+1}) \leq \frac{2}{3}s$ by the choice of i, yet we also have

$$c(g_{i+1}) \geq (c(g_i) - 1)/2 \geq (\frac{2}{3}s + 1 - 1)/2 = \frac{1}{3}s$$

This complete the first part of the proof.

Let g be the gate given by the argument in the first part of the problem. Now, let f_1 be the function computed by the formula when g and all gates that are used to compute g are removed from the formula computing f, and the gate reading g now reads the constant 1. Define f_0 similarly. First observe that

$$f = (g \wedge f_1) \vee (\neg g \wedge f_0).$$

We now recursively construct a formula for g, f_1 and f_0 (Note that the function computed by the gate g is also referred to as g). To analyze the recursion, let $D(a)$ denote the depth of the formula of size a. Note that $s/3 \leq c(g) \leq 2s/3 + 1$. In addition, $c(f_1) \leq s - c(g) + 1 \leq 2s/3+1$ (the additive term of 1 accounts for the constant input that replaces g). Analogously, $c(f_1) \leq 2s/3 + 1$. Therefore,

$$D(s) \leq D(2s/3 + 1) + 3.$$

This is because, the depth of f is at most the sum of 3 and the maximum depth of formulas computing f_1, f_0, g. Note that $D(6) \leq 6$, and this corresponds to the base case as the argument from the first part is valid only when the size of the formula is more than 6. Solving this recurrence implies that $D(s) \leq O(\log s)$ as desired.

Common Errors/Issues.

- Most solutions did not explicitly discuss the recursive relation to analyze the construction. It is not a must, but definitely improves clarity.
- Many did not notice that the argument in the first part works only when the size of the formula is more than 6 and hence the base case is when the size is 6.
- Some did not discuss the base case.
- Some did not discuss the upper bounds on $c(g), c(f_1), c(f_0)$ which are crucial for the argument.