CS 351: Solutions to Problem Set 4

(1) Let A be a set and let $\odot : A^2 \longrightarrow A$ be an associative operator. The *prefix sum function for* $(A, \odot)$ is defined (for $n \geq 1$) by $\text{PREF}_{\odot,n}(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$, where $y_i = x_1 \odot x_2 \odot \ldots \odot x_i$, for $1 \leq i \leq n$.

(1a) Let A be a finite set, and let $\odot$ be an associative operator that can be computed by a single $\odot$-gate whose inputs are elements from A. Show that $\text{PREF}_{\odot,n}$ can be computed by a circuit of $\odot$-gates of size $O(n^2)$ and depth $O(\log n)$.

There are n sub-circuits; the i-th sub-circuit, for $1 \leq i \leq n$, computes $y_i = x_1 \odot \ldots \odot x_i$ via the (nearly) complete binary tree of $\odot$-gates of depth at most $\log n$. The sub-circuits perform their computations independently of each other and in parallel, so the total depth is $\log n$; the size is at most n times the size of the largest sub-circuit, which is n, for a total of $O(n^2)$.

For n-bit numbers $u = (u_{n-1}, \ldots, u_0)$ and $v = (v_{n-1}, \ldots, v_0)$ (where u_0 and v_0 are the least significant bits), define the following quantities, for $0 \leq i < n$:

$$c_0 = 0; \quad p_i = u_i \oplus v_i; \quad g_i = u_i \wedge v_i; \quad c_{i+1} = (p_i \wedge c_i) \vee g_i; \quad s_i = p_i \oplus c_i; \quad s_n = c_n.$$

The bit c_i indicates the carry coming into position i (so c_{i+1} is the carry leaving position i), s_i is the bit of the sum in position i. The bits p_i and g_i are auxiliary variables—p_i indicates whether the carry from the previous stage (if any) is *propagated* through the i-th stage, and g_i indicates whether a carry is *generated* at the i-th stage (regardless of the previous carry).

Let $B = \{0, 1\}$. For $0 \leq i < n$, define $x_i = (p_i, g_i)$, and define the operator $\diamond : B^2 \longrightarrow B^2$ by $(a, b) \diamond (c, d) = (a \wedge c, (b \wedge c) \vee d)$. Let $(y_0, \ldots, y_{n-1}) = \text{PREF}_{\diamond,n}(x_0, \ldots, x_{n-1})$, and for each i, let $y_i = (\pi_i, \chi_i)$.

(1b) Prove (by induction) that for all i, $\chi_i = c_{i+1}$.

By expanding definitions, we have $\chi_0 = g_0 = u_0 \wedge v_0 = c_1$, since if $u_0 = 1$ and $v_0 = 1$, a carry is generated from position 0 into position 1. For $i > 0$, inductively assume that $\chi_{i-1} = c_i$. We have $y_i = y_{i-1} \diamond x_i$, so $\chi_i = (\chi_{i-1} \wedge p_i) \vee g_i = (c_i \wedge p_i) \vee g_i = c_{i+1}$.

(1c) Use (1b) to show that adding two n-bit numbers can be carried out by a circuit of polynomial size and $O(\log n)$ depth.

Straightforward implementation. First compute, in parallel, p_i and g_i for $0 \leq i < n$ (and also $c_0 = 0$). This can be done in depth 1 with $O(n)$ gates. Next, compute $(y_0, \ldots, y_{n-1})$ as defined above using a parallel prefix computation; this takes $O(n^2)$ size and $O(\log n)$ depth (since the $\diamond$ operator can be computed with a constant number of gates). By (1a), we now have all the c_i's; again in depth 1 and size $O(n)$, compute all the s_i's. The s_i's yield the desired sum.

(1d) Returning to the computation of $\text{PREF}_{\odot,n}$, show that there is an $O(\log n)$ depth circuit of $O(n)$ size (using $\odot$-gates) that computes this function.

Assume that n is a power of two. If $n = 1$, output x_1. If $n > 1$, first compute in depth 1 using $n/2$ $\odot$-gates, the quantities $x_1 \odot x_2, x_3 \odot x_4, \ldots, x_{n-1} \odot x_n$. Then recursively compute $\text{PREF}_{\odot,n/2}(x_1 \odot x_2, x_3 \odot x_4, \ldots, x_{n-1} \odot x_n)$. The recursive computation already yields the quantities $y_2, y_4, \ldots, y_n$ correctly; also $y_1 = x_1$. To compute y_3, y_5, etc., note that for odd i, $y_i = y_{i-1} \odot x_i$, so these can be computed in depth 1 using $n/2$ $\odot$-gates. Thus the entire computation has size $S(n) = S(n/2) + n$ and depth $D(n) = D(n/2) + 2$. Also, we have $S(1) = D(1) = 1$. The solutions to these recurrences are, respectively, $S(n) = 2n$ and $D(n) = 2 \log n$.

(2) *Exercise.*

(3) Show that counting the number of s–t paths in a directed, acyclic, layered (or leveled) graph can be done in NC^2.

In a directed, acyclic, and leveled graph G, there are no self-loops, and the number of paths of length exactly ℓ from a node i to a node j is precisely the (i, j)-th entry of A^ℓ, where A denotes the adjacency matrix of the graph G. Thus, to compute the number of (s, t)-paths, we simply need to add the (s, t)-th entries of the matrices $A, A^2, \ldots, A^{n-1}$. The NC^2 algorithm works as follows. Compute each of these powers of A; an easy way to think about this is via an $O(\log n)$-depth, polynomial-size, parallel prefix circuit with $\star$ gates, where each $\star$ gate multiplies two matrices. When each $\star$-gate is replaced by a polynomial-size, log-depth circuit that multiplies two $n \times n$ matrices (with poly(n)-bit entries), this yields an $O(\log^2 n)$ depth, poly(n)-size circuit that computes all the powers of A. Finally, add the (s, t) entries of all these matrices using the log-depth poly-size circuit that adds $O(n)$ numbers with poly(n) bits each.

(4) For $f : \{0, 1\}^n \longrightarrow \{0, 1\}$, R_f denotes the following promise problem. Alice is given x such that $f(x) = 0$ and Bob is given y such that $f(y) = 1$, and their goal is to find a position i such that $x_i \neq y_i$. Show that for any circuit C that computes f, $depth(C) \geq cc(R_f)$.

Alice and Bob both hold a copy of the circuit C, and consider, respectively, the computations $C(x)$ and $C(y)$. Suppose the output gate of C is an AND gate. In Alice's computation, namely $C(x)$, this gate evaluates to 0 and in Bob's computation, namely $C(y)$, this gate evaluates to 1. Thus in $C(y)$, both inputs to this AND gate are 1's, while in $C(x)$, at least one of the two inputs to this gate is a 0—and Alice knows which one. Alice writes a bit (on the blackboard) indicating whether the input that's a 0 is the "left" input or the "right" input to this AND gate (we'll adopt the convention that the lexicographically smaller-numbered gate is the left input and the higher-numbered gate is the right input). If this input is the output of another AND gate, we have the same situation again. If it is an OR gate, then in $C(x)$, we have an OR gate both of whose inputs are 0 (so that the gate is 0), but in $C(y)$, at least one of the gates is a 1 (so that the gate is 1). In this case, Bob will write a bit indicating which of the two inputs to the OR gate produced a 1. (Of course, negation gates are trivially handled by flipping the logical values.) Inductively, at any point, Alice and Bob have agreed upon an AND or an OR gate whose values in $C(x)$ and $C(y)$ are different. If it is an AND gate, the party whose evaluation is a 0 indicates which of the two inputs is a 0; if it is an OR gate, the party whose evaluation is a 1 indicates which of the two inputs is a 1. Thus they arrive at a gate whose values are different, and *which is one level lower* in the circuit C. If C has depth d, this exchange carries on for at most d steps, at which time they arrive at an input gate where they differ, so $cc(R_f) \leq depth(C)$.