CSE 4111 Computability Classes
Jeff Edmonds
Assignment 2
Due: One week after shown in slides

First Person:	Second Person:
Family Name:	Family Name:
Given Name:	Given Name:
Student #:	Student #:
Email:	Email:

Guidelines:

- You are strongly encouraged to work in groups of two. Do not get solutions from other pairs. Though you are to teach & learn from your partner, you are responsible to do and learn the work yourself. Write it up together. Proofread it.
- Please make your answers clear and succinct. helpful hints.
- Relevant Readings:
  - 02-Classes Slides from the lectures.
- This page should be the cover of your assignment.

Problem Name	Max Mark	
1 Parity	10	
2 Simple Containment	10	
3 $NC^1 \subseteq$ *Log-Space*	10	
4 NTime vs Space	10	
Total	40	

1. $Parity(x_1, \ldots, x_n) = x_1 \oplus \ldots \oplus x_n$ provides whether the number of one's is odd or even.

   (a) Prove that $Parity \in NC_1$.

   (b) Prove that in an NC_1 circuit, we can assume that all the negations are at the leaves.

   (c) I stated that $Parity$ can't be done in AC_0 because it needs $\log n$ alternations. What does this mean and argue that it is true.

   (d) I stated that $Parity$ is needed for counting and for adding. What does this mean and argue that it is true.

2. Simple class containment

   (a) Show that $AC_0 \subseteq Threshold_0$ and give a problem $P \in Threshold_0$ that is likely not in AC_0.

   (b) Show that $Arithmetic_0 \subseteq NC_1$. What assumption do you need to make in order to have proved this? Give a problem $P \in NC_1$ that is likely not in $Arithmetic_0$.

   (c) Why does the class NC_2 have the extra condition that the circuit be of polynomial size, but NC_1 does not have this condition?

3. The goal is to prove that $NC_1 \subseteq$ *Log-Space*. This is accomplished by writing an algorithm that takes as input an *and/or/not* circuit C of depth $\mathcal{O}(\log |I|)$ and an input I and computes $C(I)$ using at most $\mathcal{O}(\log n)$ space.

   A confusion is when measuring the amount of space we are allowed, whether by n we mean the size $n_I = |I|$ (in bits) of circuit C's input I or the size $n_{\langle C,I\rangle} = |\langle C, I\rangle|$ (in bits) of our algorithm's input $\langle C, I\rangle$. Luckily, because C has depth of at most $c\log(n_I)$, the number n_C of gates in C can be at most $n_C = 2^{c\log(n_I)} = (n_I)^c$ and $\log(n_{\langle C,I\rangle}) \approx c \cdot \log(n_I)$. Hence, it does not matter which size we use. For ease, lets assume that n is the number of gates in C. This means that it requires $\log n$ space to have a finger on one of the gates g of your input circuit C.

   Assume that C is presented to you in such a way that if you have your finger on one gate g, you can easy find the two gates $LeftInput(g)$ and $RightInput(g)$ that produce its two inputs and the gate $Output(g, k)$ which is the k^{th} gate to use g's output as one of its inputs.

   (a) Start by giving pseudo code for a simple recursive algorithm that solves the problem, but that may use too much space. Hint: Focus on "What is my goal?" "What help do I want from my recursive friends?" and "How do I use their answers to produce my answer?"

   ```
   algorithm CircuitSolve(g)
   ⟨pre−cond⟩: The global variable C is an and/or/not circuit of depth O(log n). The global
       variable I is an input I for C. The local variable g indicates one of the gates of C.
   ⟨post−cond⟩: Determines the value of gate g in C on input I

   begin
       ...
   end algorithm
   ```

   (b) If you have n objects, in this case gates, and you must give each a name, then how many bits will each name require, i.e. how many bits does it take to store the variable g?

(c) Recall that when a recursive program runs, one instance of the routine calls another, which calls another, and so on. Each of these that is either waiting for a recursive call to return or is currently running has a stackframe in memory storing the values of its local variables and the point in the code which should be executed next on return. These stackframes are stacked up in a stack. On the top of the stack is that of the current running instance of the routine. The above algorithm has a local variable g, hence this will be stored in each stackframe currently in the stack. Describe the set of gates g that are in this stack at any give point in time during the execution of the algorthm. What is the maximum number of these stackframes ever in the stack? What is the total amount of space used by this algorithm.

(d) The slides claim that this algorithm can be implemented using at most $\mathcal{O}(\log n)$ space. As a hint, they say to do depth first search, remembering the values along the path. Our goal for this question is to make minimal changes to the above recursive algorithm, but to reduce its space to only $\mathcal{O}(\log n)$. To do this, each stackframe can only use $\mathcal{O}(1)$ space. Each stack frame no longer can store the gate g whose output it needs to evaluate. Using two bits, it will store (once it knows them) the values inputed to g, i.e. the single bit evaluations of the gates $LeftInput(g)$ and $RightInput(g)$ that are produce by its two friends. As said, each stack frame also stores the exact point in the code which should be executed next on return. The interesting information that can be extracted from knowing this point in the code is whether the next stackframe in stack after this stackframe is $LeftInput(g)$ or $RightInput(g)$. Note that this can all be done using $\mathcal{O}(1)$ space. It would also be nice to store the index k indicating which gate $Output(g,k)$ is in the previous stackframe in stack before this stackframe. The problem is that the output gate g might be used in lots (up to n) gates. Hence, it might take up to $\log n$ bits to store such a index k. Hence, the stack frame cannot store this.

To replace the local variable, g, the algorithm will have a single global variable $g_{current}$ which indicates the gate of C that is that of the current running stackframe.

What changes in the execution of your recursive program? The two key things to focus are the following. 1- when recursing, how does the program construct the new current gate $g_{current}$. 2- when returning from recursing and the control needs to return to the stack frame that called it, how does the program construct this previously current and now current again gate $g_{current}$? Hint: Available to your algorithm is the entire stack of stackframes.

(e) Compute the running time of this algorithm in terms of n_C, where n_C denotes the number of gates in C.

4. NTime vs Space

(a) Prove $P \subseteq PSpace$ and hence $Valid$ can be conmputed in $PSpace$.

(b) Prove $PH \subseteq PSpace$.