Problem Set 4

- Due Date: **3 Apr (Wed), 2018**
- If you submit handwritten solutions, start each problem on a fresh page.
- Referring sources other than the lectures and the text-book is strongly discouraged. But if you do use an outside source (eg., other text books, lecture notes, any material available online), ACKNOWLEDGE all your sources (including collaborators) in your writeup. This will not affect your grades. However, not acknowledging will be treated as a serious case of academic dishonesty.
- The points for each problem are indicated on the side.
- If you don't know the answer to a problem, then just don't answer it. Do not try to convince yourself or others into believing a false proof.
- Be clear in your writing.

---

1. **[parallel decision vs. search]** (10)

   Recall that we can obtain a satisfying assignment for a Boolean formula (if one exists) in polynomial time given an oracle for deciding SAT (using downward self-reducibility of SAT). However, note that this reduction algorithm makes *adaptive* queries to the SAT oracle, i.e. its i'th query depends on the answers to its first $i-1$ queries. Show that the reduction can be made nonadaptive if we allow it to be randomized.

   [Hint: Use Valiant-Vazirani reduction.]

2. **[#P-completeness]** (7+7+8+3)

   Recall the definition of #**P**-completeness. A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ in #**P** is said to be #**P**-complete if for every $g \in$ #**P**, there exists a deterministic polynomial time oracle Turing machine M such that for all $x \in \{0,1\}^n$, $M^f(x) = g(x)$. (Note that this is more general than the restricted notion of #**P**-*completeness under parsimonious reductions*.)

   (a) A *matching* in a graph is a set S of edges such that every vertex touches *at most one* edge in S (as opposed to exactly one, as required in a perfect matching). Show that #Matchings, the problem of counting all the matchings in a graph, is #**P**-complete.

   (b) An *independent set* in a graph G is a set S of vertices such that no two elements of S are connected by an edge in G. Prove that #Independents Sets, the problem of counting the number of independent sets in a graph, is #**P**-complete.

   (c) Show that approximating #Independents Sets to within any constant factor is **NP**-hard.

   (d) In contrast, there are a fully polynomial randomized approximation schemes known for #Perfect Matchings and #Matchings. I.e, there exists randomized algorithms that can obtain $(1+\varepsilon)$-approximation for every $\varepsilon > 0$. Explain why this does not give a fully polynomial approximation scheme for #Independents Sets contradicting (c).

   [Hint: (a) Reduce from #Perfect Matchings. Given a graph G, consider the graph G_k obtained by attaching k new edges to each vertex of G. G_k has $n + nk$ vertices, where n is the number of vertices in G. Show that the number of perfect matchings in G can be recovered from the number of matchings in each of $G_0, \ldots, G_n$.]

3. **[if permanent had polysized arithmetic circuits and BPP=P, ...]** (20)

Suppose there exists a family of polynomial-size arithmetic circuits $\{C_n\}_{n=1}^{\infty}$ such that C_n computes the permanent of $n \times n$ matrices over the integers. Suppose also that $\mathbf{BPP} = \mathbf{P}$. Prove that these two hypotheses imply that $\mathbf{P}^{\#\mathbf{P}} = \mathbf{NP}$.

[Hint: Use downward self-reducibility of permanent.]

4. **[pairwise independent hash family]** (3+5)

Recall the definition of pairwise indendendent family of functions: A family $H_{n,k}$ of functions from $\{0,1\}^n$ to $\{0,1\}^k$ is said to be a family of pairwise independent hash functions if for all $x \neq y \in \{0,1\}^n$ and $u, v \in \{0,1\}^k$, we have

$$\Pr_{h \leftarrow H_{n,k}}[h(x) = u \text{ and } h(y) = v] = \frac{1}{4^k}.$$

(a) Identify $\{0,1\}^n$ with the field $GF(2^n)$. For every $a, b \in GF(2^n)$, define the function $h_{a,b}(x) = a \cdot x + b$ where $\cdot$ and $+$ refer to multiplication and addition in the field $GF(2^n)$. Show that $H_{n,n} = \{h_{a,b} | a, b \in GF(2^n)\}$ is a pairwise independent hash family. Can you from this construction obtain a pairwise hash family $H_{n,k}$ for every $1 \leq k \leq n$. Can you generalize this construction to obtain a family of d-wise independent hash functions for every constant $d \in \mathbb{Z}^{\geq 0}$.

(b) Let $H_{n,k}$ be a family of pairwise pairwise independent hash functions. Let $S \subseteq \{0,1\}^n, |S| \geq 4 \cdot 2^k/\varepsilon^2$. Prove that

$$\Pr_{h \leftarrow H_{n,k}}\left[\left||\{x \in S | h(x) = 0\}| - \frac{|S|}{2^k}\right| \geq \frac{\varepsilon |S|}{2^k}\right] \leq \frac{1}{4}.$$

5. **[#L problems]** (10)

The counting class #P could have been defined in two equivalent ways.

A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #P if there is a non-deterministic Turing machine M_f that on input x of length n uses poly(n) time and is such that the number of accepting paths of $M_f(x)$ equals $f(x)$.

A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #P is there is a relation $R(,)$ that is computable in polynomial time and a polynomial p such that $f(x)$ equals $|\{y : R(x,y) \text{ and } |y| \leq p(|x|)\}|$.

[Check that these two definitions are indeed equivalent.]

In this problem, we will show that the corresponding two definitions are very different if one considers counting problems in logspace. Consider the following two definitons of log-space counting problems.

A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #L1 if there is a non-deterministic Turing machine M_f that on input x of length n uses $O(\log n)$ space and is such that the number of accepting paths of $M_f(x)$ equals $f(x)$.

A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #L2 is there is a relation $R(,)$ that is computable in logarithmic space and a polynomial p such that $f(x)$ equals $|\{y : R(x,y) \text{ and } |y| \leq p(|x|)\}|$.

Prove that all functions in #L1 can be computed in polynomial time while #L2 equals #P.

6. **[Promise Problems]** (4+3+10+10)

(a) Show that $\mathbf{P}^{\mathbf{NP} \cap \mathbf{coNP}} = \mathbf{NP} \cap \mathbf{coNP}$.

Recall the definition of a promise problem. Note that for a promise problem Π, "running an algorithm with oracle Π" is not in general well-defined, because it is not specified what the oracle should return if the input violates the promise. Thus, when we say that a problem Γ can be solved in polynomial time with oracle access to Π, we mean that there is a polynomial-time oracle algorithm A such that for *every* oracle $O : \{0,1\}^* \to \{0,1\}$ that solves Π (i.e. O is correct on $\Pi_Y \cup \Pi_N$), it holds that A^O solves Γ. Let Π be the promise problem

$$\begin{aligned} \Pi_Y &\stackrel{\text{def}}{=} \{(\varphi, \psi) : \varphi \in SAT, \psi \notin SAT\} \\ \Pi_N &\stackrel{\text{def}}{=} \{(\varphi, \psi) : \varphi \notin SAT, \psi \in SAT\} \end{aligned}$$

(b) Show that $\Pi \in \mathbf{prNP} \cap \mathbf{prcoNP}$

(c) Show that $SAT \in \mathbf{prP}^{\Pi}$.

[Hint: You might have to query the oracle Π more than once, even a superconstant number of times.]

This implies that $\mathbf{prNP} \subseteq \mathbf{prP}^{\mathbf{prNP} \cap \mathbf{prcoNP}}$. Note that an analogous inclusion seems unlikely for language classes, since $\mathbf{P}^{\mathbf{NP} \cap \mathbf{coNP}} = \mathbf{NP} \cap \mathbf{coNP}$, as shown in the earlier part.

(d) Show that $\mathbf{prBPP} \subseteq \mathbf{prRP}^{\mathbf{prRP}}$, and thus $\mathbf{prRP} = \mathbf{prP}$ iff $\mathbf{prBPP} = \mathbf{prP}$.

[Hint: Look at the proof that $\mathbf{BPP} \subseteq \mathbf{PH}$.]