CPSC506 **Complexity of Computation** **Term 2, 2020**

Homework # 3

Due on Wednesday, Feb 26

1. Many complexity classes have been defined that shed insight on the complexity of functions rather than languages (decision problems). One such class is #P, defined to be the class of functions $f : \{0,1\}^* \to \mathbb{N}$ such that there is a polynomial time nondeterministic Turing machine which, on any input $x \in \{0,1\}^*$, has a number of accepting computations equal to $f(x)$.

   (a) Give a definition of what it means for a problem to be $\#P$-complete. (Note that here, we need reductions between functions, rather than languages.)

   (b) Explain briefly why the function #3SAT, which maps (binary encodings of) Boolean formulas ϕ to the number of satisfying assignments of ϕ, is $\#P$-complete, referring to the Cook-Levin Theorem.

   (c) Show that the the problem of computing the number of perfect matchings of a bipartite (undirected) graph G is #P-complete.

   In showing this, you may use the fact that the problem of computing the permanent of a $n \times n$ matrix M with entries in $\{0,1\}$ is $\#P$-complete. The permanent of such a matrix M is defined as

   $$\sum_{\sigma} \prod_{i=1}^{n} M[i, \sigma(i)],$$

   where the sum is over all permutations σ of $\{1, 2, \ldots, n\}$ and $M[i, \sigma(i)]$ is the entry of M in row i and column $\sigma(i)$. See if you can find a relationship between the permanent of a matrix and the number of matchings of a (suitably defined) bipartite graph.

   (Note: it is perhaps not surprising that "counting versions" of NP-complete problems are often $\#P$-complete. Examples are the problems of counting the number of Hamiltonian paths or the number of minimum vertex covers of a graph. In contrast, it is possible to compute a perfect matching of a bipartite graph in polynomial time, yet the counting version of this problem is #P-complete.)

2. (Exercise from Chapter 6 of Arora-Barak) Show that if EXP is contained in P/poly then EXP = Σ_2^p.

3. This problem is to find a *zero-sided error* algorithm for the undirected graph reachability problem, that runs in polynomial expected time and uses logarithmic space. That is, your algorithm should run in logarithmic space and should make no error; it uses coin flips and may run for more than polynomial time on some runs but its *expected* running time is polynomial. An outline of one approach to solving the problem is given here.

   Let $G = (V, E)$ be an undirected graph with n vertices, which are numbered from 1 to n. Suppose we denote by $u \to_k^* v$ the existence of a path from u to v in graph G that does not pass through any intermediate vertices l with $l > k$. Let $P_k = \{(u, v) \mid u \to_k^* v\}$. Let $|P_k|$ denote the size of the set P_k.

(a) Construct a 1-sided error algorithm that, given G, k, u, v as input, returns true with high probability (say $1 - 1/(2n^2)$) if $(u, v) \in P_k$ and false with probability 1 if $(u, v) \notin P_k$.

(b) Construct an errorless algorithm that given $G, k, u, v, |P_k|$ as input, determines whether or not $(u, v) \in P_k$.

(c) Construct an errorless algorithm that, given $G, k, |P_{k-1}|$ as input, returns $|P_k|$.

(d) Using the previous steps, construct an errorless algorithm that, given G, u, v as input, accepts if and only if there is a path from u to v in G.

All of your algorithms should run in polynomial expected time and use logarithmic space; explain why this is the case.