Advanced Complexity Theory — Madhu Sudan
6.841/18.405J
Due: Monday, March 18, 2002

Problem Set 1

Problems

1. Give a randomized logspace algorithm for determining if an undirected graph is bipartite (i.e., if the vertices can be partitioned into two sets V_1 and V_2 such that all edges have exactly one endpoint in V_1 and one in V_2). Include a formal definition of the language you are working with, and say whether your algorithm places this language in BPL, RL or co-RL.

   For extra credit, show that this problem is in ZPL.

2. A language L is said to be *rare* if there exists a polynomial p such that for every n, the number of strings in L of length n is at most $p(n)$.

   Show that if a rare language is NP-complete then the hierarchy collapses.

   For extra credit, show that if a rare language is NP-complete then NP=P.

3. The depth of a ciruit is the longest path from an input node to an output node in the circuit. A circuit is said to be a formula is every computation node has fan-out one (i.e., only one wire leaves every internal node). In this problem, we'll only consider formulae and circuits with unary NOT gates, and binary AND and binary OR gates.

   Show that a family of functions $\{f_n : \{0,1\}^n \to \{0,1\}\}_n$ has a family logarithmic depth circuits if and only if it has the family of functions has polynomial size formulae.

4. If PSPACE $\subseteq P/_{\text{poly}}$ then show that PSPACE $= \Sigma_k^P$ for some constant k.

5. A promise problem is a generalization of the notion of a language. A promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$ consist of two disjoint subset Π_{YES} and Π_{NO} of $\{0,1\}^*$. A Turing machine M is said to decide Π if it accepts every string in Π_{YES} and rejects every string in Π_{NO} (and we don't care what happens with the rest). Promise problems play an important role in modern complexity theory.

   Let promise-RP be the class of promise problems decided by a randomized polynomial Turing machine $M(\cdot,\cdot)$: i.e., if $x \in \Pi_{\text{YES}}$ then $\Pr_y[M(x,y) \text{ accepts}] \geq 2/3$, and if $x \in \Pi_{\text{NO}}$ then $\Pr_y[M(x,y) \text{ accepts}] = 0$.

   Show that if P=promise-RP, then P=BPP. I.e., suppose, for every promise problem Π in promise-RP, there is polynomial time algorithm A that accepts every instance in Π_{YES} and rejects every instance in Π_{NO}, then show that P=BPP.

Instructions:

- Usual rules on collaboration and references.

- Turn in the solutions to the above problems by 11am on Monday, March 18, 2002 (even though we have no lecture then).

Additional Exercises: Not to be turned in!!

The following exercises are highly recommended. They give more practice with the notions in class. Doing the exercises is not mandatory.

1. Show that ZPP = RP $\cap$ co-RP.

2. For a deterministic polytime Turing machine M and polynomial p, define a sequence of strings $(a_1, \ldots, a_n)$ to be GOOD if $|a_i| = p(i)$ and for every cnf formula ϕ of length $i \leq n$, $\phi \in SAT \Leftrightarrow M(\phi, a_i)$ accepts.

   Use the self-reducibility of SAT to show that the task of recognizing if a sequence of strings is GOOD is in co-NP.

3. A Branching Program is yet another non-uniform model of computation. Such a program is specified by a directed acyclic graph on m nodes with one designated start node and two end nodes labelled 0/1. Start nodes and all other internal nodes are labelled by a variable name from $x_1, \ldots, x_n$. End nodes have out-degree 0. All other nodes have out-degree two with one edge labelled 0 and the other labelled 1. The branching program represents a Boolean function as follows: Given a 0/1 assignment to the variables we follows the path out of the start node that is consistent with the assignment (i.e,, taking the edge labelled 0 out of a vertex labelled x_i if $x_i = 0$ etc.) till we reach an end node. The label of the end node gives the value of the function on this input. The size of a branching program is the number of nodes in it. The size represents "non-uniform" space requirement of a computation.

   (a) If $L \in$SPACE$(s(n))$ then show that L_n is computed by a poly$(2^{s(n)})$-sized branching program.

   (b) Given a branching program, show that it is NP-complete to determine if the branching program accepts any input.

   (c) A branching program is read-once if every variable appears at most once on every input/output path. Give a randomized algorithm to test if two read-once branching programs accept the same Boolean function.