Limits of Computation: Homework 6 solutions

Kevin Matulef

March 7th, 2001

Problem 8.19

A directed graph is strongly connected if every two nodes are connected by a directed path in each direction. Show that the language STRONGLY-CONNECTED $=\{\langle G\rangle |$ G is a strongly connected graph$\}$ is NL-complete.

- First we show that STRONGLY-CONNECTED $\in$ NL. Consider the following machine which decides $\overline{STRONGLY-CONNECTED}$.
  On input $\langle G\rangle$,

  1. Nondeterministically select two nodes a and b.
  2. Run PATH(a,b). If it rejects, then the graph is not strongly connected, so *accept*. Otherwise, *reject*.

  Since storing the node numbers a and b only takes log space, and PATH uses only log space, $\overline{STRONGLY-CONNECTED} \in$ NL. Finally, since NL=CoNL, STRONGLY-CONNECTED $\in$ NL

- Next we must show that every other language in L is log space reducible to STRONGLY-CONNECTED. We do this by reducing PATH to STRONGLY-CONNECTED. Consider the following reduction.
  "On input $\langle G, s, t\rangle$

  1. Copy all of G onto the output tape.
  2. Additionally for each node i in G
  3. Output an edge from i to s.
  4. Output an edge from t to i.

  If there is a path from s to t, the the constructed graph is indeed strongly connected, because every node can now get to every other node by going through the $s-t$ path. If there is not a path from s to t, then the constructed graph is not strongly connected because the only additional edges in the constructed graph go *into* s and *out* of t, so there can be no new ways of getting from s to t.

Finally, we must verify that the reduction can indeed be performed by a log space transducer. Indeed this is the case, because though the output has size $O(n)$, essentially the only space necessary to perform the reduction is that used to keep track of the node number i in the for loop above.

Problem 8.20

An undirected graph is bipartite if its nodes may be divided into two sets so that all edges go from a node in one set to a node in the other set. Show that a graph is bipartite if and only if it doesn't contain a cycle that has an odd number of nodes. Then, show that BIPARTITE = $\{\langle G\rangle |$ G is a bipartite graph$\}$ is in NL.

- First we show that if a graph is bipartite, it must not contain a cycle with an odd number of nodes. Suppose it did contain such a cyle. Label the nodes $n_1, n_2, ...n_{2k}, n_{2k+1}$. Clearly, if n_1 is in some set A, then n_2 must be in set B, so n_3 must be in set A, etc. By induction, we see that all the nodes with an odd subscript must be in set A, and all those with an even subscript must be in set B. But this implies that n_1 and n_{2k+1} are both in set A, a contradiction because they are connected.

- Second, we show that if a graph contains no cycles with an odd number of nodes, then the graph is bipartite. Suppose a graph does not contain any odd cycles. Pick a node, and label it A. Label all off its neighbors B. Label all of *their* unlabeled neighbors A, etc, until all nodes are labeled. Suppose that this contruction caused two adjacent nodes x and y to have the same label. Then that would mean that both x and y were reached by taking an odd number of steps from the start node, or both were reached by taking an even number of steps. In either case, the total number of nodes traversed in getting to x from the start node, and getting to y from the start node (excluding the start node itself), is even. But adding the start node in makes the total number of nodes odd, contradicting the hypothesis that there were no odd cycles. Thus, the construction succeeds in properly dividing the nodes, so the graph is bipartite.

- Finally, we show BIPARTITE $\in$ NL. Since NL=CoNL, it suffices to demonstrate that $\overline{BIPARTITE}$ = HAS-ODD-CYCLES? = $\{\langle G\rangle |$ G is a graph that contains an odd cycle$\} \in$ NL. The following algorithm works
"On input $\langle G\rangle$

  1. *counter* := 0.
  2. *start* := Nondeterministically select a starting node.
  3. *successor* := Nondeterministically select a successor of s.
  4. While (*counter* $\leq$ the number of nodes)
  5. If *successor* = *start* and if *counter* is odd, *accept*, otherwise...
  6. *successor* := nondeterministically select a successor of *successor*

7. If *counter* has increased above the number of nodes, *reject*.

Since the only space used by the above algorithm is for keeping track of the node numbers *start* and *successor*, and keeping track of the counter *counter* (whose maximum value is the number of nodes), the algorithm clearly only uses log space on any one branch. Therefore, BIPARTITE $\in$ NL.