2) Show that BIPARTITE is in NL.

Show BIPARTITE is in coNL and use NL = coNL.

Show non-BIPARTITE is in NL.

Guess an odd cycle, one vertex at a time, and in log space verify that the next edge of the cycle is an edge of G and count the total number of vertices in the cycle. We must remember the first vertex of the cycle, and verify that the last vertex matches the first.

3) STRONGLY-CONNECTED is NL-complete.

Reduce PATH to STRONGLY-CONNECTED with a logspace reduction.

Given G, s, t. Produce G’ using only logspace work space.

G’ contains the same vertices as G. For each edge of G, add the edge to G’. For each vertex u of G, u is not s or t, add and edge t → u, and edge u → s.

Prove G’ is strongly connected if and only if there is a path from s to t in G.

If there is a path from s to t in G. For any vertex u and any vertex v, there is a path u→s→t→v. G’ is strongly connected.

There is no path from s to t in G. Suppose there is a vertex v that we can’t reach from s in G. The only way to reach v in G’ is to use edges of G, or an edge from t, but we can’t reach t in G, so we can’t reach t in G’ since we can use only edges of G. Or t → u → v, but we can’t reach u from s in G

not A → not B that is logically equivalent to B → A.

1) Show finding a cycle in an undirected graph can be done in L.

Two ideas: 1) Count the edges. If G is connected, we know that if G has more than n-1 edges, G has a cycle. The problems is what if G is not connected. If G has c components and more than n-c edges, G has a cycle.

If we assume that vertices of G are stored on the tape in order of components, then we can count the number of components in log space. Two variables. V stores the current vertex I am looking at, and f stores the furthest vertex (on the tape) that I can reach by one edge from v. Increment v, and only increase f when we can reach a further vertex. Once v=f, we reached the end of a component, increase the component count, and move v = f = v+1, and repeat.

Idea 2) Think of the graph as a maze and use the “left-hand rule” to traverse the graph. At each vertex (a choice of edges), go “left” by choosing the next edge from that vertex in the order listed on the input tape, if at the last edge, go to the first edge listed from that vertex. We need to adjust this to run from every vertex to hunt for a cycle and prove that we will be able to detect a cycle always.

Oracle machines.

Prove there exists an oracle A such that PA = NPA and and oracle B such that PB <> NPB.

We know that PA is a subset of NPA.

NPTQBF is a subset of NPSPACE (we can decide TQBF in PSPACE so no need to consult the oracle if we have a nondeterministic machine that uses polynomial space).

NPSPACE = PSPACE (Savich’s theorem)

PSPACE is a subset PTQBF (Take a PSPACE problem and in polynomial time reduce it to TQBF, then ask the oracle.)

Let A = TQBF and we have NPA = PA.

Create a language B such that PB <> NPB. This is going to be a contradiction proof. We will build up the language by simulating TM’s and doing the opposite.

Suppose we have a set of all polynomial time deterministic oracle Turing machines. {M1, M2, …. }

Suppose Mk runs in time nk. For each machine we will give it A for the oracle tape.

Let L= { x | there exists an a in A with |x| = |a|}.

We are going to show that we can’t decide L using a polynomial time deterministic machine that has A as an oracle, but we can with an nondeterministic machine.

Non-determ. Machine: Given input w, guess a string a with the same length as w. Query the oracle A to ask if a is in A? If it is, accept, and if not, reject.

Consider the polynomial time machines for k = 1 to ….

For machine Mk, it takes input w, and it is trying to decide if w is in L. Is there an a in A with |a| = |w|. It is going to make a bunch of queries of A. Each time it queries, A answers “no”. M can only ask nk queries, but there are 2n possible strings of length n in A. Then Mk must make a decision about w.

If it says “yes”, then we make A so that all strings of length n are not in A. Mk is wrong.

If it says “no” then we find some string of length n that Mk  did not query about and add it to A.

Mk is wrong.

We remember all our choices so when we go to Mk+1 we answer consistently,

not now we make sure Mk+1 can’t answer correctly for a string of longer length.

As a result, for each k, there is some length of string that Mk is unable to correctly determine if strings of that length are in L. So no polynomial time deterministic machine using oracle A can decide L