Homework 2 for Intro to Computational Complextiy

Shenlong Gu

917-544-8927, sg3301@columbia.edu, 27, September 2015

Part I

We assumes a Non-DTM M_1 decides the language L_1, and a Non-DTM M_1 decides the language L_2.
Let $V_1(x, c)$ be the verifier for M_1 and $V_2(x, c)$ be the verifier for M_2.

1. **union**
   We will construct a poly-time $V(x, c)$ for the language, the union of L_1 and L_2 to solve this problem.
   for any input x, given a stratage input string c, let $V(x, c) = 1$ iff $V_1(x, c) = 1$ and $V_2(x, c) = 1$.
   It is easy to see $V(x, c)$ is the verifier which runs in polynomial time given input x and c and $V(x, c)$ is the verifier for the language, the union of L_1 and L_2. So NP is closes under union operation.

2. **concatenation**
   We will construct a poly-time $V(x, c)$ for the language, the union of L_1 and L_2 to solve this problem.
   for any input x, the stratage input c which is the union-encoding of the stratage c_1 for $V_1(x, c)$, the stratage c_2 for $V_2(x, c)$.
   $V(x, c)$ works as the follows, creates $n + 1$ pairs, $(i, n - i), 0 \leq i \leq n$. Decode c into c_1 and c_2
   foreach pair $(i, n - i)$:
       if $V_1(x[0 : i], c_1) == 1$ and $V_2(x[i : n], c_2) == 1$:
           $V(x, c) = 1$, accepts
   end $V(x, c) = 0$, rejects. (Because no pairs work given the stratage c_1 and c_2.
   It is easy to see $V(x, c)$ is the verifier which runs in polynomial time given input x and c and $V(x, c)$ is the verifier for the language, concatenation of L_1 and L_2. So NP is closes under concatenation operation.

3. **star**
   We will construct a poly-time $V(x, c)$ for the language, the union of L_1 and L_2 to solve this problem.
   Given an input x, we let c to encode a segmentation of input x into $x_1, x_2, ...x_k$, and a stratage of $V_1(x, c)$ to these k segmentation $c_1, c_2, ...c_k$.
   Then, $V(x, c)$ works as follows:
   decode c first, if c can not be decoded, rejects it.
   for $i = 0$ to k:

if $V_1(x_i, c_i) == 1$:
$V(x, c) = 1$, accepts
end
$V(x, c) = 0$, rejects.
It is easy to see $V(x, c)$ is the verifier which runs in polynomial time given input x and c and $V(x, c)$ is the verifier for the language, star of L_1. So NP is closes under star operation.

Then to prove G and H isomorphic. Given an input x with length n, we encode a permutation of n which is a node mapping from G and H, we give a verifier $V(x, c)$ given the input x and encoding mapping c, it is easy to judge if two maps are isomorphic under this mapping in poly-time (just check if each mapping edge exists or doesn't exist in two graphs).

Part II

First, we will show that if a graph is bipartite, then it can not have a cycle containing an odd number of nodes. Assume there is a such cycle the bipartite, n_1, n_2,.....n_{2*k+1}. We color the nodes into two part(blue and red) . if n_1 is red, n_2 is blue, then n_{2*k+1} will be red the same as n_1, and there is an edge between n_1 and n_{2*k+1}, so there comes a contradictory.

Second, if there is no such cycle, we will show it is a bipartite. We also color the node, we randomly selects a node, and gives a color (for example, red), We traverse the edge from this node using dfs or bfs, everytime there is an edge between two nodes, we color the child node with a opposite color from its parent, when there is a back edge. Because there is no cycyle with an odd number of nodes, this node will not result in a contradictory with its ancestor, so we can color all nodes without contradictory, so it is a bipartite.

Third, we will show bipartite problem $\in$ NL.
We change the bipartite judging problem into a existing problem. As proved above, we can change a bipartite judging problem by deciding that there is no cycle containing odd number nodes.
Then we think about the complement problem(because NL equals CO-NL): how to decide that if there is a cycle containing odd number nodes.
To use NDTM to solve this problem:
First, we nondeterministicly selects a start node, s, and set a next node t to nondeterministicly be a child of s, and set a counter to be zero .
while counter $\leq n$:
(n is total number of nodes, because the length of the cycle can not exceed n)
if t equals s:
if counter is odd:
accepts it
else:
rejects it
We nondeterministicly set t to be a child of t,(if there is no child, we rejects)
counter = counter + 1
We see that this NDTM decides the problem (odd-number-nodes cycle existing problem), and the space it uses is just start node number, next node number, counter which cost just

logarithmic space. So bipartite judging problem $\in$ NL.