

## CSCI303 Fall 2006 Homework 6 Solutions

### 34.1-3) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)

An adjacency matrix representation of a directed graph is a matrix  $M$  of 0s and 1s where an entry  $M(i,j)=1$  implies there exists an edge from  $i$  to  $j$ . The binary string representing the adjacency matrix is the concatenation of the rows of that matrix.

An adjacency list representation consists of an array  $A$  of lists, one for each vertex in the graph. Each list  $A[i]$  contains all vertices  $j$  such that there is an edge from  $i$  to  $j$ . The binary string representing the adjacency list is the concatenation of the lists  $A[i]$ , in order, separated by a delimiter.

Given an adjacency matrix, one can compute the adjacency list by creating a list for each row of the matrix and placing the nodes with a 1 in each row into that row's list. Given an adjacency list representation, one can compute the adjacency matrix by scanning through the lists in order and placing 1s and 0s in the appropriate columns and rows. These conversion algorithms take linear time, so the two encodings are polynomially related.

### 34.1-5) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)

Suppose an otherwise polynomial algorithm, that “otherwise” runs in  $\Theta(n^s)$  steps, calls polynomial-time subroutines that run in  $\Theta(n^t)$  at most  $c$  times. Since the output of each subroutine must be polynomial in size of its input (in order for it to run in polynomial time) let's say that for all subroutines the output is  $O(n^w)$  for input of size  $n$ . Then after  $c$  executions, the output must be no bigger than  $O(n^{cw})$ . Therefore, the running time of the algorithm will be  $\Theta(n^s + cn^t) + O(n^{cw}) = O(n^s + n^t + n^{cw})$ .

Now, suppose algorithm  $A$  takes a string  $w$  and returns a the concatenation of  $w$  with itself or  $ww$ . That is, on input “10010” the algorithm would return “1001010010”. Clearly,  $A$  can run in polynomial time. Suppose an algorithm  $B$  takes as input string  $s_0$  and calls  $A$  on  $s_0$  to get  $s_1$ . It then calls  $A$  on  $s_1$  to get  $s_2$ . It continues in such manner for  $n$  iterations, finally returning  $s_n$ . Since the length of the string doubles every time, the length of  $s_n$  must be  $\Theta(2^n)$ , after only  $n$  calls to  $A$ . But it takes at least  $\Theta(2^n)$  steps to output a string of that length. Thus  $B$  must run in exponential time even though it only makes a polynomial number of calls to a polynomial-time subroutine.

### 34.2-3) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)

Assume that given a graph there is a polynomial-time algorithm to tell if that graph has a hamiltonian cycle, let's assume it runs in  $\Theta(n^k)$  steps. Look at  $E$ , the set of all edges in the graph  $G$ . Remove an edge and check if the resulting graph has a hamiltonian cycle. If it does, forget that edge and continue. If it does not, then that edge was important, so keep it and continue to remove others. After you've tried all the edges, you will only have the edges of the hamiltonian cycle left. List them in order.

The worst-case time complexity of this algorithm is  $\Theta(|E|n^k) = \Theta(n^{k+2})$ . Thus, if we have a polynomial time algorithm  $P$  to decide whether a graph has a Hamiltonian cycle, we can also generate a polynomial time algorithm to list all the vertices of the Hamiltonian cycle.

**34.3-2) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)**

Assume that  $L1 \leq_p L2$  and  $L2 \leq_p L3$ . We show that  $L1 <_p L3$ . Because  $L1 \leq_p L2$ , there exists a function  $f$  such that  $x \in L1 \Leftrightarrow f(x) \in L2$ , and similarly, because  $L2 \leq_p L3$ , there exists a function  $g$  such that  $y \in L2 \Leftrightarrow g(y) \in L3$ . Thus,  $x \in L1 \Leftrightarrow f(x) \in L2 \Leftrightarrow g(f(x)) \in L3$ . So  $g \circ f$  is a function that polynomially relates  $L1$  and  $L3$ .

**34.3-3) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)**

We want to show that  $L \leq_p \bar{L} \Rightarrow \bar{L} \leq_p L$ .

There exists a polynomial time function  $f$ , such that  $x \in L \Leftrightarrow f(x) \in \bar{L}$ .

Thus  $x' \notin L \Leftrightarrow f(x') \notin \bar{L}$ .

By the definition of complement, it follows that  $x \in \bar{L} \Leftrightarrow f(x) \in L$ , thus  $L \leq_p \bar{L} \Rightarrow \bar{L} \leq_p L$

The reverse direction is completely symmetric.

**34.4-6) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)**

Suppose you have a polynomial-time algorithm that decides whether a satisfiability formula is satisfiable or not, and let this algorithm run in  $\Theta(n^k)$  steps. Consider the set of variables and for each variable, set that variable to T, simplify the resulting formula, and ask the algorithm if it is satisfiable. If it is, then assign T to that variable. If it is not, then assign F to that variable. Continue with the rest of the variables.

This algorithm runs in  $\Theta(n^{(k-1)+1})$  steps.

**34.4-7) last updated fall 2006, Yuriy Brun, [ybrun@usc.edu](mailto:ybrun@usc.edu)**

Using the hint, a 2-CNF Boolean formula can be converted to a form where the ORs between two literals are replaced by implications ( $x \vee y \equiv \neg x \Rightarrow y$ ). So convert the boolean formula to the form  $(\neg x_1 \Rightarrow x_2) \wedge (\neg x_3 \Rightarrow x_4) \wedge \dots$ . We construct a directed graph with  $2n$  vertices, one for each possible literal. For each implication  $(\neg x_1 \Rightarrow x_2)$  in the formula, we add an edge to the graph from  $\neg x_1$  to  $x_2$ . The weight of each edge can be 1. We then check for all literals if there is a path from that literal to the negation of that literal. If there is, there is a contradiction and the formula must not be satisfiable. If for all literals there is no such path, then the formula is satisfiable.

In polynomial time, we can find the shortest path from every vertex to every other vertex in a directed graph (page 625). Thus in polynomial time we can decide whether there exists a literal with a path to its own negation literal. Thus the reduction is polynomial.