CS 454 Theory of Computation Spring 18

Project 2 Due: April 1, 2018

1) Say that a positive integer N is strongly not divisible by an integer k if N is not divisible by k, and for every j, the number N_j obtained by removing the j-th digit of N is not divisible by k. For example, the number 741842607938866199443579680083706254648829519399268 is strongly not divisible by 7. It is not easy to find such a large, randomly chosen N since a randomly chosen number has a low probability of being a valid choice - since there are 50 different N_j that you can get by removing a digit – the chances are high that at least one of them is divisible by 7. So if we want to generate a 50 or 100 digit integer that is not strongly divisible by a given integer k, you need a more systematic approach. Finite automaton provides one such approach. But building a DFA that accepts the set of integers that are not strongly divisible by a given integer is hard because every digit must be removed and the resulting number should be tested for divisibility. But designing an NFA to reject strongly not divisible integers is easier; it just guesses one digit to be removed and check that the resulting number is divisible by k. The goal of this project to write a program that takes as input a positive integer k, and outputs an NFA that *rejects* the set of integers that are strongly not divisible by k. As a final step, use this NFA to answer the question whether a given number is strongly not divisible by k. Formally, the problem is stated as follows.

   Input: integer k
   Output: NFA that accepts the language $L = \{ w \mid w$ is a string over $\{0, 1, \ldots, 9\}$ such that w is a multiple of k, or deletion of some digit in w gives an integer that is a multiple of $k \}$.

   After the NFA is built, the program asks the user for an integer N, and answer *yes* if N is strongly not divisible by k, *no* else. (A test case is the 50 digit integer shown above.)

   An NFA with $2k$ states can be constructed as follows: First construct a DFA similar to the one in Project 1, Problem 2 that accepts integers that are multiples of k. Make another copy of the DFA, and create a transition from state t of the first copy to state t of the second copy on all inputs 0, 1, 2, … , 9.

   In the second step, implement the algorithm presented in class to test if a given string w is accepted by an NFA. This algorithm keeps track of the set of reachable states on the prefix of the input string read thus far. It updates this information after reading the next input character. The reachable states at each step is maintained as a Boolean vector of size n = the number of states in the NFA.

   For the test cases k will be a 4-digit number and N will be up to 100 digits long.

   (Extra-credit: Convert it to a DFA, and take the complement to construct a DFA for the language. Then, for a given integer n and a DFA M, write a function to find the lexicographically first string of length n accepted by M. Use this function to find, for a given n and k, the smallest integer with n digits that is not strongly divisible by k. Assume that both n, k are at most 100.)

2) Write a regular expression in *egrep* for the set of strings that have at least two occurrences of the same letter next to each other occurring at least twice. Test your expression on a given dictionary of words and print out all the words that match the condition stated above.

3) Write a program that takes as input two DFA's M_1 and M_2, and constructs a DFA M_3 such that $L(M_3) = L(M_1) \cap L(M_2)$. (Recall that the states of M_3 are of the form (p, q) where p is a state in M_1 and q is a state in M_2.) Assume that the input DFAs will be presented as follows: the alphabet over which the DFA's are defined will be assumed to be $\{0, 1, \ldots, k\text{-}1\}$ for some k, the states will be labeled $0, 1, \ldots, n\text{-}1$ for some n, and F is an array of length n such that $F[k] = 1$ (0) if k is an accepting (non-accepting state). The delta function will be specified as a matrix M with n rows of k columns where $M[i, j] = \delta(i, j)$. The DFA is to be read from a file in which the first line contains two integers n and k, the second line contains F and the next n rows contain the δ-function. (Each row j will have exactly k numbers $\delta(j, 0), \ldots, \delta(j, k-1)$.) Two example DFA's can be found in dfa1.txt and dfa2.txt. Make sure that your algorithm only expands the states that are reachable from the start state (s_1, s_2) where s_1 and s_2 are the start states of M_1 and M_2. Also make sure that the states of M_3 are labeled $0, 1, \ldots, t$ (for some t). Create a file df3.txt in the same format that stores the DFA for $L(M_1) \cap L(M_2)$.