CS 454 Theory of Computation   Spring 2019

Project 1, Due: February 24, 2019

PROBLEM 1:

Write a function count that computes the number of strings w of length n over $\{a, b, c\}$ with the following property: In any substring of length 4 of w, all three letters a, b and c occur at least once. For example, strings like *abbcaabca* satisfy the property, but a string like *bac*$\mathbf{aabb}$*cabac* does not satisfy the property since the substring **aabb** does not have a c. The idea is to create a DFA M for the language:

$L = \{w \mid \text{in any substring of length 4 of } w\text{, all three letters } a\text{, } b \text{ and } c \text{ occur}\}.$

Suppose $M = \langle Q, \Sigma, \delta, 0, F\rangle$ states. Assume that $Q = \{0, 1, \cdots, m–1\}$ and that 0 is the start state. Recall the algorithm we presented to compute the number of strings of length n from any state j to an accepting state.

Let $N(j, n)$ be the number of strings w length n such that $\widehat{\delta}(j, w)$ is in F, i.e., the set of strings of length n that start in state j and reach an accepting state. Clearly, the number of strings of length n accepted by a DFA M are given by $N(0, n)$. The recurrence formula for $N(j, n)$ is given by $f(j, n) = \Sigma_{x\in\{a,b,c\}}N(\delta(j, x), n-1)$. Initial values $N(j, 0)$ are given by: $N(j, 0) = 1$ if $j \in F$, $N(j, 0) = 0$ if $j \notin F$. You can iteratively compute $N(j, k)$ for all j for $k = 0, 1, \cdots, n$. As we noted in class, you only need to keep two vectors *prev* and *next* of length m. Using the values of $N(j, k)$ stored in *prev*, you can compute $N(j, k + 1)$ for all $0 \leq j \leq m - 1$ in *next*. Then copy *next* to *prev* and repeat.

When your main function runs, it will ask for an integer input n, and output the number of strings of length n with the specified property. The range of n will be between 1 and 300. The answer should be exact, not a floating-point approximation so you should use a language that supports unlimited precision arithmetic like Java or Python or a library like GMP (in case of C++).

Some test cases:

Test case 1:
Input: n = 137
Output: 6119266976149912241614898841866546736

Test case 2:
Input: n = 100
Output: 987802207638178400131884900

PROBLEM 2:

Write a function `MinString` that takes as input a DFA M and outputs a string w of shortest length (lexicographically first in case of more than one string) accepted by the DFA. (If $L(M)$ is empty, your program should print **No solution**. Breadth-First Search (BFS) will be used to solve this problem. Use this function to write another function `smallestMultiple` that takes as input a positive integer k, and a subset S of $\{0, 1, 2, \cdots, 9\}$ and outputs the smallest positive integer $y > 0$ that is an integer multiple of k, and has only the digits (in decimal) from the set S. The algorithm to solve this problem is as follows: Create a DFA $M = \langle Q, S, \delta, k, F\rangle$ where $Q = \{0, 1, \cdots, k\}$, $F = \{0\}$, and $\delta(j, a) = (10 * j + a)\%k$. Here is a brief summary of BFS (which will be presented in more detail in class.) Initially, a Queue contains n, the start state. Also VISITED is set to True for n and False for all other states. Then, the search is performed until the Queue is empty or state 0 is reached: Delete j from the Queue and let NEXT be the set of states reachable from j: NEXT $=\{\ \delta(j, a) \mid$ for all $a \in S\}$, and insert for each x in NEXT such that VISITED[x] = false into the queue (and set VISITED[x] to True.) Also PARENT[x] is set to j. When the loop ends, if the QUEUE is empty, the DFA does not generate any string. Otherwise, your algorithm has found the shortest path from n to 0. By tracing the path (using the PARENT pointers) you can find the shortest string that accepted by the DFA.

For this problem, you can assume that k is in the range 1 to 99999.

Some test cases:

Test case 1:
Inputs: k = 26147, Digits permitted: 1, 3

Output: 1113313113

Test case 2:
Inputs: k = 198217, Digits permitted: 1
Output: integer containing 10962 ones (Your output will be a string of this many 1's.)

Test case 3:
Inputs: k = 135, Digits permitted: 1 3 7
Output: No solution.