CS 454 Theory of Computation Fall 2022

Project 1, Due: September 27, 2022

The project will be done in teams of two or three members. (Three member teams should also solve Problem 3.) Only one submission is required per team. In addition to the code, please submit a short report that *states clearly the contribution of each member of the team.* Submission details will be specified in detail (along with test cases that shoud be included with the submission) in the canvas submission page. The project will be graded on (a) correctness on the test cases, (b) meeting the user interface requirements and (c) annotation of the code. Around 5% of the total points will be assigned for (b) and around 10% points for (c).

PROBLEM 1:

Write a function **count** that computes the number of strings w of length n over $\{a, b, c, d\}$ with the following property: In any substring of length 6 of w, all three letters a, b, c and d occur at least once. For example, strings like *abbccdaabca* satisfy the property, but a string like *bad***aabcbc***dcabad* does not satisfy the property since the substring **aabcbc** does not have a d. The idea is to create a DFA M for the language:

$L = \{w \mid$ in any substring of length 6 of w, all the letters a, b, c and d occur$\}$.

By definition, all strings of length less than 6 are in L. Let $M = \langle Q, \Sigma, \delta, 0, F\rangle$ be a DFA. Assume that $Q = \{0, 1, \cdots, m-1\}$ and that 0 is the start state. The first step is to construct this DFA. This DFA has hundreds of states so you have to write a program to build it. We have looked at similar DFAs in class. The states will encode a buffer that tracks the last 5 symbols seen. When the next input is read, the DFA checks if the suffix of last 6 symbols (the five from the buffer and the next input) meets the requirement that all letters occur at least once. If this condition is not met, the DFA enters a fail state and thereafter it will remain there. If the condition is met, then the buffer is updated. Initially the buffer is empty and so the first five transitions will just fill the buffer. Formally, if the current state q encodes a string $b_1b_2\cdots b_m$ where $m < 5$, then $\delta(q, a)$ encodes the state $b_1b_2\cdots b_ma$. If the state p encodes a buffer of length 5, say $b_1b_2b_3b_4b_5$, then $\delta(q, a)$ encodes the state $b_2b_3b_4b_5a$ if $b_1b_2b_3b_4b_5a$ contains at least once occurrence of each a, b, c and d, else $\delta(q, a)$ is the fail state. It is enconvenient to map each buffer string by a positive integer so that the states can be labeled 0, 1, 2, etc. One such encoding is to use base-4. For example, *babca* represents the integer $2 \times 4^4 + 4^3 + 2 \times 4^2 + 3 \times 4^1 + 4^0$.

The next step is to implement an algorithm that takes as input a DFA M and an integer n and computes the number of strings of length n accepted by M. This algorithm was presented and a number of examples were given, including some in quizzes 3 and 4. Specifically, this algorithm computes $N_j(n) =$ the number of strings of length n from any state j to an accepting state:

The number of strings of length n accepted by a DFA M is given by $N_0(n)$. The recurrence formula for $N_j(n)$ is given as follows: $N_j(n) = \Sigma_{x\in\{a,b,c,d\}} N_{\delta(j,x)}(n-1)$. Initial values $N_j(0)$ are given by: $N_j(0) = 1$ if $j \in F$, $N_j(0) = 0$ if $j \notin F$. Using this recurrence formula, you can 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 where m is the number of states. Using the values of $N_j(k)$ stored in *prev*, you can compute $N_j(k+1)$ for all $0 \le j \le m-1$ in *next*. Then copy *next* to *prev* and repeat.

When your main function is run, it will ask for an integer input n, and will 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 are included below:
$n = 6$ Answer: 1560
$n = 56$ Answer: 114451878183882876885021 6

PROBLEM 2:

Given a positive integer $k > 0$, and a subset S of $\{0, 1, 2, \cdots, 9\}$ of digits, the goal is to find the smallest positive integer $N > 0$ such that $N\%k = 0$ and N uses only the digits from the set S. As an example, if $k = 26147$ and $S = \{1, 3\}$, the value of $N = 1113313113$. The obvious approach to this problem of generating the integers that use only the digits in S in succession and testing if the integer is divisible by k is extremely inefficient and will take years to solve on the test cases. By using a DFA, we can significantly speed up a solution to this problem.

Described below is the algorithm `FindString` 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 M. (If $L(M)$ is empty, the algorithm outputs **No solution**.) Breadth-First Search (BFS) is used to solve this problem. The algorithm `FindString` 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. Here is a brief summary of BFS (which was presented in more detail in class. See the pseudo-code below for more details.) 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. (Make sure to skip the null string as the shortest string.)

Using FindString algorithm, Problem 2 can be solved as follows. Create a DFA $M = \langle Q, \Sigma, \delta, 0, F\rangle$ where $Q = \{0, 1, \cdots, k-1\}$, $F = \{0\}$, and $\delta(j, a) = (10 * j + a)\%k$ and call FindString with M and k as inputs. (This DFA needs a small correction when S includes 0.)

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 ones.)

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

BFS customized for this project

As we discussed in class, the generic BFS has to be modified for this project in the following ways. In the usual applications of BFS, the underlying graph has no labels on the edges. But a DFA is a directed graph in which edges are labeled. So to keep track of the shortest path, parent array stores the previous state on this path, and the label array stores the corresponding edge label. Another difference is that this version of BFS terminates as soon as the accepting state is reached, while the standard version of BFS continues the search until all the vertices reachable from the starting vertex are located.

```
Algorithm FindString:
   Input: DFA M with Q = {0, 1, 2, ... k-1}, 0 is the start state, S = input alphabet, {
    (In your implementation, the DFA need not be explicitly created; instead, the delta
   Output: the shortest string accepted by M. If there is more than one of the same leng

Initialize a queue Q;

Initialize a Boolean array visited of size k

Initialize an integer array parent of size k

for all j in {0, 1, ..., k-1} do
   visited[j] = false;

for all j in {0, 1, ..., k-1} do
   parent[j] = -1;

for s in D do
   next = delta(0, D[s], k);
   visited[next] = true;
   Q.insert(next);
   parent[next] = 0;
   label[next] = s;
end for;

while (Q is not empty):
 curr = Q.delete();
 for each s in D \ {0} do:
  next = delta(curr, s, k);  // Recall  delta(q, r, k) = (10 * q + r) % k
  if (next == 0): // accepting state is reached
     parent[next] = curr
     label[next] = s
     break // out of the while loop
  else if not visited[next]:
            visited[next] = true;
            parent[next] = curr;
            label[next] = s;
            Q.insert(next)
         end (if)
   end (for)
  end (while)
   if (next! = 0):
       output no solution
   else
       trace the string using parent pointers and concatenate the corresponding labels a
       output the reverse of the string.
end FindString;
```

PROBLEM 3:

This is a variation of Problem 2 in which you are given a positive integer $k > 0$ and a subset $D \subseteq \{0, 1, \cdots, 9\}$ of the ten decimal digits. The goal of your program is to find the smallest integer N that uses only the digits in D such that both N and the reverse of N are (strictly positive) multiples of k. A simple example is: suppose $k = 71$ and $D = \{1, 2, 4\}$, the solution is 2414142.