University of Waterloo
CS240E, Winter 2024
Programming Question 2

Due Date: Tuesday, March 26, 2024, at 5pm

Be sure to read the assignment guidelines (`https://student.cs.uwaterloo.ca/~cs240e/w24/assignments.phtml#guidelines`).

Ensure you have read, signed, and submitted the **Academic Integrity Declaration** AID02.TXT.

Grace period: submissions made before 11:59PM on March 26, will be accepted without penalty. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [20 marks]

In this programming question you are asked to implement a phone directory that permits lookups in both directions, by using two hash tables D_{phone} and D_{name} that are using phone numbers and names as keys, respectively. Your C++ program must provide a main function that accepts the following commands from `stdin` (you may assume that all inputs are valid):

- `i name number` - inserts an entry with the given name and the given phone number into both hash tables
- `l number` = prints one line with the name associated with phone number to the output or 'not found' if there is no such phone number.
- `s name` - prints all phone numbers associated with the the name, in lexicographic order, all in one line and separated by spaces. (This may be an empty set.)
- `r` - rehash both hash tables (see below).
- `p i` - prints dictionary i according to the specifications below, where $i = 0$ means D_{name}, and $i = 1$ means D_{phone}.
- `x` - terminates the program.

To implement these dictionaries, we use two hash-tables. Because we will test the structure of the hash-tables via the printing functions, you must follow the specifications below (as well as the pseudo-codes from the lecture notes) exactly.

- The dictionary D_{name} that uses names as keys should use *hashing with chaining*. A name gives rise to a key by applying flattening; use $R = 255$. Implement flattening so that it does not lead to overflow. Names (hence keys for D_{name}) are not unique, but

hashing with chaining handles this naturally by putting repeated keys into the same list.

Use $M = 11$ as initial table-size. Re-hash whenever inserting an item would lead to $n > 2M$ (where n is the number of items after the insert). When re-hashing, use the smallest prime that is at least $2M + 1$ as the new table-size. The stub provides a method to find this.

To print D_{name}, print $M + 1$ numbers, separated by spaces, all on one line. The first number is M itself. The next M numbers are the lengths of the M buckets of the hash-table.

- The dictionary D_{phone} that uses phone numbers as keys should use *cuckoo hashing*. A phone number gives rise to key k by removing all non-digits and interpreting the result as a number in base-10. (This is a large number; use `long`.) Use two arrays (both initially of table-size $M = 11$) and hash-functions $h_0(k) = k \mod M$ and $h_1(k) = M\lfloor(Ak - \lfloor Ak\rfloor)\rfloor$ for $A = (\sqrt{5} - 1)/2$. The stub provides implementations for these.

  Re-hash whenever insert fails. When re-hashing, use the smallest prime that is at least $2M+1$ as the new table size. The re-hashing should extract all items that are currently in the two tables, in order, then create new tables, and then insert the items into the new table in this order, following by the item that was left to be inserted. Note that it is possible that insert fails during re-hashing, which means that you recursively need to re-hash again.

  To print D_{phone}, print $2M + 1$ numbers, separated by spaces, all on one line. The first number is M itself. The next M numbers are either 0 or 1, depending on whether the slots in the first table are empty (0) or not (1). The next M numbers are either 0 or 1, depending on whether the slots in the second table are empty or not.

Further specifications and assumptions:

- A name is a string of arbitrary length, with characters in {`A` – `Z`, `a` – `z`}. It does not contain spaces.
- A phone number has the form `(abc)def-ghij` where each of `a-j` are in $\{0, \ldots, 9\}$.
- Phones numbers are unique. Names are not.
- Tables-sizes will never need to exceed `INT_MAX`.
- The recursion depth during insertion for cuckoo hashing is at most 2.
- Buckets will stay small enough that using quadratic-time algorithms on them is acceptable.
- 'Printing on one line' means that the line must end with a newline. trailing whitespace at the end of your output lines will be ignored by test scripts.

- You are allowed to use `string` and `stringstream` from the STL, as well as `vector`.

Place your entire program in the file `phonedirectory.cpp`. Submit your solution to Marmoset. Marmoset will be set up to translate your program with `g++ -std=c++17`.