University of Waterloo
CS240 Spring 2024
Programming 2

Due Date: Tuesday, July 16 at 5:00pm

Please read the following link for programming question guidelines and details on submission:

`https://student.cs.uwaterloo.ca/~cs240/s24/assignments.phtml#guidelines`

Submit the file `phone.cpp` to Marmoset.

Late Policy: Programming Questions are due at 5:00pm with a grace period until 11:59pm. No submissions beyond the grace period will be accepted.

Problem 1

In this programming question you are asked to implement a phone directory that permits lookups in both directions, by using the following two hash tables:

- D_{name} implements *hashing with chaining* and uses names as keys
- D_{phone} implements *cuckoo hashing* and uses phone numbers as keys

Your C++ program must provide a main function that accepts the following commands from stdin. You may assume that all inputs are valid and will begin with `r` and end with `x`; also, all inserts will be successful.

- `i name number` - inserts an entry with the given name and the given phone number into both hash tables and then prints three numbers. Each pair of numbers is separated by a space and the third number is followed only by a newline. The first number is the table index where the name was inserted, the second number is the table index where the number was inserted and the third number is the chain index where the number was inserted (0 if it is inserted as the first item in the chain and so on).
- `l number` - prints the name associated with the phone number followed by a newline, or "Not found" followed by a newline if there is no such phone number.
- `s name` - prints all phone numbers associated with the name, in lexicographic order. Each pair of phone numbers is separated by a space and the last is followed only by a newline. If no such name exists, print "Not found" followed by a newline.

- `rh i` - rehashes dictionary i where $i = 0$ means D_{name} and $i = 1$ means D_{phone}. When rehashing, use the smallest prime that is at least 2M + 1 as the new table-size. The `phonestub.cpp` file provides a method to find this.

- `p i` - prints dictionary i according to the specifications below, where $i = 0$ means D_{name} and $i = 1$ means D_{phone}.

- `r` - initializes or resets both hash tables to be empty (and of size M = 13).

- `x` - terminates the program.

Details for implementing Dictionary D_{name}

- Start with an empty hash table of size $\mathbf{M = 13}$ and use hash function `hash0` provided in the `phonestub.cpp` file.

- A name is a non-empty string of arbitrary length, with characters in $\{A-Z, a-z\}$ that does not contain spaces. To convert a name into a key you must apply the flattening technique and Horner's Rule to avoid overflow (Module 7 Slide 29); use $\mathbf{R = 255}$.

- To facilitate printing (in searches by name), each chain must store the numbers in sorted order (smallest number first) so that phone numbers can be printed lexicographically with a single traversal of the chain (with no extra sorting required). Insert each new number to maintain this order (do not simply insert at the front) and do not apply the MTF heuristic. The chains must be implemented as linked lists. You may use `forward_list` from the STL or implement your own.

- To print D_{name}, print M + 1 numbers, where a pair of numbers is separated by a space and the last number is followed only by a newline. The first number is M itself. The next M numbers are the lengths of the M buckets of the hash table.

Details for implementing Dictionary D_{phone}

- Start with two initially empty hash tables, each of size $\mathbf{M = 13}$, using the hash functions `hash0` and `hash1` provided in the `phonestub.cpp` file. Implement the two hash tables using vectors.

- A phone number has the form $(abc)def - ghij$ where each of $a - j$ are in $\{0, \ldots, 9\}$ where digits may be repeated. To convert a phone number into a key, remove all non-digits and interpret the result as a number in base-10 (use a `long` to store the key).

- When rehashing, insert the items into the new tables by traversing the old first table, in order, starting at index 0 followed by similarly traversing the second table.

- To print D_{phone}, print 2M + 1 numbers, where a pair of numbers is separated by a space and the last number is followed only by a newline. 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 Implementation Details

- Table sizes will never exceed `INT_MAX`.
- Phone numbers are unique. Names are not.
- You may use `string`, `stringstream`, `forward_list`, `vector`, `pair`, `swap` and smart-pointers from the STL.

Place your entire program in the file `phone.cpp` and **clearly label (with comments) each of the hash table implementations and operations so it is easy for the markers to find them in your file**.