IIT Ropar
CSL201 Data Structures
Semester 1, AY 2016/17

Lab Assignment 3 - 30 marks Due on 30th September, 11:59 PM

Objective

To understand and implement hash tables.

Instructions

1. DO NOT use STL data structures. You have to implement all data structures on your own.
2. There are six programming questions; you only need to submit first two questions. Other questions are no graded; they are just for practise.
3. You are to use C++ programming language to complete the assignment.
4. Provide a Makefile to compile your final code.
5. This is an individual assignment. You can have high level discussions with other students, though.
6. Include a “Readme.txt” file on how to compile and run the code.
7. Upload your submission to moodle by the due date and time. After due date, your submission will be evaluated with 10% penalty per day for next two days. After that, your submission will not be evaluated.

Questions

1. [20 marks] In this assignment you need to implement and analyze a string matching algorithm. Given a text string T[0...n-1] and a pattern string P[0...m-1], your program should print location (index) of all the occurrences of P in T.

Examples 1
Input 1: T = “A man's ego is the fountainhead of human progress.”
P = “ego”
Output:
Pattern found at index 8

Examples 2
Input 1: T = “AABAACAADAABAAABAA”
P = “AABA”
Output:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13

A naive solution to this problem is exhaustive search using string comparison method at each index of the input string. This method would be O(mn). A better approach is to use Rabin–Karp algorithm. The Rabin–Karp algorithm works as follows:

a. Hash all the substrings of size m, starting at index 0.
b. Hash P and compare its hash value with all the hashes calculated in step a.
c. If the hash values match, you further compare to see if the strings also match.

You can find the details of the algorithm and a C++ implementation here: http://www.geeksforgeeks.org/searching-for-patterns-set-3-rabin-karp-algorithm/

You are allowed to use this code as your starting point.

As you can see, in the worst case all substrings of T can have the same hash value as P. Hence, the worst case time is still O(mn). But in practise, Rabin-Karm Algorithm is much faster.

You task is to implement the Rabin-Karp algorithm using: (1) all four hash codes (Integer casting, Component sum, Polynomial sum, Cyclic sum) and (2) all three hash functions (Division, MAD, Multiplication). You can use linear probing to handle collisions. For each of the cases above (there will be 12), count:

- False positives. The number of times hash value matches but not the actual string.
- Comparisons: The number of string comparisons including the one’s that return true.

Input will be given as a raw text file. The pattern will be given to you during the demo. The output should be two 4X3 matrices. You should display as well as store the results in a file named EntryNumOutput1.txt.

2. [10 marks] Modify the the above algorithm (and program) to be able to search for incomplete words. You would like to type in something like ?orcius and have the

program match this pattern quickly to the correct answer Porcius. What modifications are needed in the Rabin-Karp algorithm for this to work? You will take input from the user and display output at run time. Not “?” denotes the missing alphabet.

Additional Questions (ungraded)

3. Describe how to perform a removal from a hash table that uses linear probing to resolve collisions where we do not use a special marker to represent deleted elements. That is, we must rearrange the contents so that it appears that the removed entry was never inserted in the first place.

4. Supposethateachrowofann×narrayAconsistsof1’sand0’ssuchthat, in any row of A, all the 1’s come before any 0’s in that row. Assuming A is already in memory, describe a method running in O(nlogn) time (not O(n2) time!) for counting the number of 1’s in A.
Ans: To count the number of 1’s in A, we can do a binary search on each row of A to determine the position of the last 1 in that row. Then we can simply sum up these values to obtain the total number of 1’s in A. This takes O(log n) time to find the last 1 in each row. Done for each of the n rows, then this takes O(n log n) time.

5. Write a spell-checker class that stores a set of words, W , in a hash table and implements a function, spellCheck(s), which performs a spell check on the string s with respect to the set of words, W. If s is in W, then the call to spellCheck(s) returns an iterable collection that contains only s, since it is assumed to be spelled correctly in this case. Otherwise, if s is not in W , then the call to spellCheck(s) returns a list of every word in W that could be a correct spelling of s. Your program should be able to handle all the common ways that s might be a misspelling of a word in W, including swapping adjacent characters in a word, inserting a single character in between two adjacent characters in a word, deleting a single character from a word, and replacing a character in a word with another character. For an extra challenge, consider phonetic substitutions as well.

6. The Monster family wants to keep a database of all Monsters and their handphone number. You are employed by the Monster Family to implement this phonebook for them. You are to maintain a list of Monster names and phone numbers using a hash table, using Monster names as the keys.

COLLISION RESOLUTION

For simplicity, you should resolve collision by linear probing (although this is not a very good method).

REHASHING

You should also rehash your table by building a larger table when the table is full. Use the given list of prime number stored as array size List in PhoneBook. First, build a table with size sizeList[0]. When the table is full, increase the table size to sizeList[1], and so on. You may assume that you never need to maintain more than 400 monsters.

HASH FUNCTION

To hash a Monster name, implement a hashCode() method to convert a String object to an int. Implement another function h() to hash the values returned by hashCode() to map it to one of the slot in our hash table.

In this problem, you're required to write 4 methods:
-add(), delete(), update() and find().

(1) add(Monster m, int p) -- adds a monster m with phone number p to the phone book.

(2) delete(Monster m) -- deletes monster m and its associated phone number from the phone book. Throws NoSuchElementException if monster m is not in the phone book.

(3) updates(Monster m, int p) -- changes the phone number of monster m to p. Throws NoSuchElementException if monster m is not in the phone book.

(4) find(Monster m) -- returns the phone number of monster m. Throws NoSuchElementException if monster m is not in the phone book.

You should plan carefully about how to write all 4 methods before you begin coding.