Birla Institute of Technology & Science, Pilani
2nd Semester 2016-17 - CS F211 – Data Structures and Algorithms

---

Lab 6 (Evaluation 2): 28th Feb, 2017

Time: 170 minutes **Marks: 8 + 22 = 30**

Instructions:

- *This test consists of two problems (Problem 1 and Problem 2) specified in two different files.*
- All input expressions should be read from stdin (scanf) and output should be printed on stdout (printf).
- For first 150 minutes, only a subset of test cases will be visible to students after submitting the code on the portal. Only in last 20 minutes, all test cases will be made visible.
- At the end of 170 minute period, the online system will stop evaluating the submissions but it will accept it for additional 10 minutes. At the end of 180 minute period, it will stop accepting the submissions.
- Only the last submission by the student for each problem will be considered for evaluation, irrespective of earlier correct submission.
- Assuming that a problem contains M marks, in case of (Run-error/Compiler-error/Timelimit-error), evaluation will be done for M/2 marks only.
- Total marks of each problem contains some marks for modularity and proper structuring of code.
- All submitted source code will be later checked manually by the instructor and final marks will be awarded. Any case of plagiarism and/or hard coding of test cases will fetch 0 marks for the problem/evaluation component.
- Make sure to return 0 from the main() function in case of normal termination.

Problem 2 of 2

Expected Time: 50 minutes **Marks: 8**

Problem Statement

In your solution to problem 1, *insert* failed if an empty slot cannot be found after for MAX_CNT number of times.

Secondly, as the number of elements in the table (n) approach towards the size of the table (s), the insertion of a new string may take longer time due to multiple "collisions". We define $\lambda = \frac{n}{s}$ the **load factor** of the hash table.

In this problem, we will extend our design to handle these two issues using **"rehashing":** i.e. reallocate a new hashtable with double the size or change the hashing parameters *a* and *b;* and then reinsert all elements into the new table.

Structure of data type to store hash table
Same as defined in Problem 1 but add a new field τ (a threshold for the load factor).
Hashing Function: $int\ hash(str, c, t)$
Same as defined in Problem 1

Initializing a hash table
Same as defined in Problem 1 but also initialize τ by reading from *stdin*.

Finding a string in the hash table
Same as defined in Problem 1

Insertion of a string into the hash table
Extend the function $insert(H, str)$ (implemented in problem 1), as follows:

1. When insertion is successful and the load factor λ exceeds τ **rehash** (discussed later) with double allocation size s before returning from the insert function. The parameters a and b of the hash table do not change. If the rehashing attempt fails, report insertion failure.
2. If MAX_CNT iterations still leave a string str to be inserted, first make a rehash with changed parameters a and b (the size s does not change). If the rehashing attempt fails, make a second rehashing attempt with double allocation size s -the parameters a and b will continue to retain their last changed values. This second attempt is a desperate measure to accommodate a new string, and will be carried out even if the $\lambda \leq Threshold$. If the second attempt fails too, report failure to insert, and return. Otherwise, insert str in the rehashed table. The rehashing algorithm is explained in the next section.

Rehashing
Write a function $rehash()$ in order to reorganize the stored strings in the data array. A rehash attempt may be of two types:

1. **Change Parameters ($\boldsymbol{a\ and\ b}$):** For a CHANGE_PARAMS type of rehashing, increment both a and b by two (so the new a becomes the old b — this will save some relocations, see below), whereas the size s does not change.

2. **Double the size of hash table**: In this case the parameters a and b do not change, but the allocation size s increases to $2s$ (so t increases by one)

In both types of rehashing, create a new hash table, make a pass through the entire data array of the old table, insert all elements residing in the old table to the new table (using the $insert()$ function). Each individual insertion attempt in the new table is carried out non-recursively, that is, if MAX_CNT iterations fail to insert, return with failure status. This way you avoid $rehash()$ inside $rehash()$. If all insertions are successful, rehashing is successful too, so you free the old table and return the new table. If at least one insertion fails, rehashing is unsuccessful, and you free the new table and return the old table. This guarantees that the table returned is not inconsistent.

There remains a possibility that despite rehashing, insertion always fails. That is, there is an infinite loop of nested calls $insert() - rehash() - insert() - rehash()$- . In such a situation, abort the insertion effort after four $insert() - rehash()$ attempts turn out to be unsuccessful.

Input format

Each line will start with a one of the following key values (1, 2, 3, -1). Each key corresponds to a function and has a pattern. Implement following function according to given pattern and write a driver function which can call the functions according to given key value.

▪

Key	**Function to call**	**Format**	**Description**
1	createHashTable()	1 size a b	"1" shows creation of a new Hash Table. "size" represents the initial size of the hash table to be created. The values of "a" and "b" of the hash table should be initialized with the corresponding input values.
2	$insert()$	2 N τ str_1 str_2 str_3 ….. str_N	"2" shows the insertion of given "N" strings. in the hash table, by calling the function insert() "N" times. The order of insertion of strings in the hash table should be same as the order they appeared in the input. In case the insertion of string str_i fails, continue inserting the string from the index $i + 1$. τ is the rehashing threshold.

3	printHashTable()	3	"3" print the content of hash table such that each (index, string) pair is printed on a new line in tab separated format. Only print rows with non-null strings.
4	find()	4 str	print (str <tab> status) in a new line, where status is 0 if str exists in the hash table, otherwise status should be -1.