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 1 of 2

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

Problem Statement

This problem deals with the design of an ADT Dictionary of strings implemented as a one dimensional hash table (HT) with a custom technique for handling collisions. This ADT would support ***create***, ***insert***, ***find*** and ***delete*** operations. The size of hash table should be parameterized (i.e. not hard-coded) as you may have to increase the size by dynamically re-allocating the memory if and when required.

Structure of data type to store hash table

The data type for hash table should contain the following fields.

- s**:** current size of the table $(s = 2^t)$

- $\boldsymbol{n}$: the number elements (currently) stored in the table
- $\boldsymbol{data}$: dynamic array of strings as character pointers
- $\boldsymbol{a}$, $\boldsymbol{b}$: Two odd integers (used in hashing – *see below*)

Hashing Function: $int\ hash(str, c, t)$
Required to hash a string *str* to an index in the range 0..s–1. Here *c* can either be *a* or *b* (the two odd integers contained in the hash table).

/**** pseudocode for the hash function ******

1. First, the string str is converted to a 32-bit unsigned integer m using the following steps
    a. Let l be the length of str.
    b. $m_{-1} = 0$
    c. for $i = 0, 1 \ldots . l - 1,$ compute $m_i = (Am_{i-1} + str[i])\ mod\ 2^{32}$, where $A = 2^{16} + 2^8 - 1 = 65791$ and $str[i]$ stands for ASCII value of the character.
2. $m = m_{l-1}$
3. Return most significant t bits of $(m * c) mod\ 2^{32}$

/****end of hash function pseudocode********

Initializing a hash table
Write a function $createHashTable()$ in order to:

(i) The value of parameters $t, a\ and\ b$ are to be read from $stdin$.
(ii) Allocate memory (required for s pointers to characters) and assign it to table H, where $s = 2^t$
(iii) Initialize all these pointers NULL (empty location)
(iv) return H

Find a string in the hash table
Write a function $int\ find(H, str)$ to find whether the string str exists in H. Return 0 if found, -1 otherwise.

1. Calculate index i using the $hash()$ function by passing the value of a to the parameter c .
    a. If i[th] location in *data* is str there; and return 0
    b. Otherwise (*i[th]* location is non-empty):
        i. Calculate index *j* using the *hash()* function by passing the value of b to the parameter c .
        ii. If j[th] location in *data* is str there and return 0
2. Otherwise, return -1

Insertion of a string into the hash table

Write a function *int* $insert(H, str)$ to insert a string str in the hash table H .

1. If str is already present in H, do not make any changes. Return success.
2. Otherwise:
    a. calculate index i using the $hash()$ function by passing the value of a to the parameter c .
    b. If i[th] location in *data* is empty insert str there; and return
3. Otherwise (*i[th]* location is non-empty):
    a. calculate index *j* using the *hash()* function by passing the value of b to the parameter c .
    b. If j[th] location in *data* is empty insert str there and return
4. Otherwise (*j[th]* location is non-empty):
    a. let *newstr* be *data[j];*
    b. store str at the *j*[th] position replacing *data[j]*.
    c. repeat the process of insertion for this *newstr*;
5. the procedure should finally return *success* (1) or *failure* (0).

    [Note: Try this repetition for at most at most MAX_CNT times (set MAX_CNT = 10); if insertion does not succeed fail. *End of note.*]

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	createHash Table()	1 t a b	"1" shows creation of a new Hash Table. Size of hash table will be 2^t. The values of "a" and "b" of the hash table should be initialized with the corresponding input values.
2	insert()	2 N 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 $i + 1$

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.

Test Case 1:

Input	Output
1 2 5 7 2 10 kuuxfit omqjn xqsdpag uekd zfnv hqnmzh nukxhjv byncerx balbg xwgojtn 3 4 kuuxfit 4 omqjn 4 xqsdpag 4 uekd 4 zfnv 4 hqnmzh 4 nukxhjv 4 byncerx 4 balbg 4 xwgojtn -1	0 kuuxfit 2 xqsdpag 3 zfnv kuuxfit 0 omqjn -1 xqsdpag 0 uekd -1 zfnv 0 hqnmzh -1 nukxhjv -1 byncerx -1 balbg -1 xwgojtn -1