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

---

Lab 6 : 23rd Feb 2017
Hash Tables

Instructions:

- All input expressions should be read from stdin (scanf) and output should be printed on stdout (printf).
- Write code in different header (*.h) and source code (*.c) files. Name them properly, compile each of them using **–c** flag of gcc and combine all *.o files to create an executable file. You may create a Makefile for the above job as well.

Problem:

Querying time on record of a set of students can be reduced by storing it as a hash table. Each student can be represented by a tuple (name, id). Name will have exactly 8 characters (all lower case), id will be a large integer number (long int).
Read given input data in a dynamically allocated array of record (let's call it, records).

Implement a hash table with separate chaining (in case of collision, insert the new element at the end of chain), for each of the following options:

Option Number	Key	Table Length	Hashing Function
1	name	20	index = ((sum of ASCII value of characters in name) mod 89) mod tableLength
2	name	20	index = ((sum of ASCII value of characters in name) mod 105943) mod tableLength
3	name	200	index = ((sum of ASCII value of characters in name) mod 89) mod tableLength
4	name	200	index = ((sum of ASCII value of characters in name) mod 105943) mod tableLength
5	id	20	index = (id mod 89) mod tableLength
6	id	20	index = (id mod 105943) mod tableLength
7	id	200	index = (id mod 89) mod tableLength
8	id	200	index = (id mod 105943) mod tableLength

To keep track of all tables, create a list of hastTable pointers. In each hash table, keep following additional fields:

- `elementCount`: total number of elements (students) in the table
- `loadFactor`: elementCount/tableLength
- `insertionTime`: total number of jumps done in any of the lists (chains) to insert the element at the end. Increment only in case of collision.
- `queryingTime`: total number of comparisons done in any of the chains during all lookups

Note: The nodes of the table should store a pointer to actual student record in records array i.e. there should be only one copy of each record but accessible through different hash tables.

Input format

Each line will start with a one of the following key values (1, 2, 3, 4 or -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	Input Format	Description
1	readRecords	1 N $name_1$ id_1 $name_2$ id_2 .... $name_N$ id_N	"1" shows start of a set of records. "N" represents number of records to come. Next N lines will have a single word followed by an id (separated by tab), in each line. Store records in a dynamic array of size N. Let's call this array as "records". Create all hashtables and insert all N records in the end. Store
2	readQueries	2 K	"2" shows start of a set of queries. "K" represents number of queries to come. Next K lines will have a single word followed by id in each line. Store records in a dynamic array of size N. Let's call this array as "queries". Call find() function on each input query one by one on all hashtables, and store total querying time for each option. find(n) should return address of record in records array. Depending upon hast table, find() should either take name or id as input.
3	findInsertionComplexity	3	Print "Option Number,insertionTime" pair of each table in increasing order of `insertionTime` of the hash table to finish all insertions. (Tab separated printing)
4	findQueryComplexity	4	Print "Option Number, queryTime" pair of each table in increasing order of `queryingTime` to look up all "queries" in each hash table. (Tab separated printing)
-1		-1	stop the program.

Sample Input	**Sample Output**
1 10 cristian 80 abbigail 7 jonathan 21 abdullah 800 cristian 10 adelaide 100 jonathan 15 prentice 2 juliette 199 kasandra 41 2 5 jonathan 15 cristian 10 cristian 80 juliette 199 kasandra 41 3 4 -1	8,0 7,1 3,2 4,2 5,3 6,4 1,5 2,5 8,5 6,6 7,6 3,7 4,7 1,8 2,8 5,8