CME 2201 Assignment 1

Due date: 09.11.2018 Friday 23:59. Late submissions are not allowed.

You must upload your all **'.java' files** as an archive file (.zip or .rar). Your archived file should be named as 'studentnumber_name_surname.rar/zip', e.g., 20179000101_Altug_Yigit.rar.

Control date: Control of the homework will be on 12-23 of November. Research assistants will control your assignments in their office (door no: 124). Therefore, you should fill in the information on the Google Form to schedule. You will have 15 minutes to show your assignment.

Plagiarism Control: The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in the Classroom.

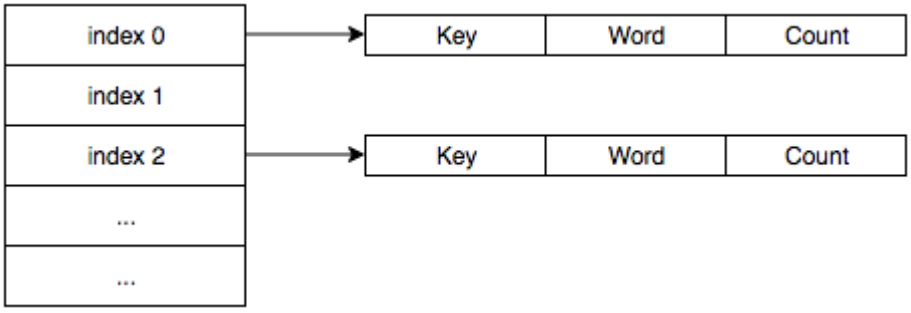
Please do not forget to bring your laptops while coming to the assignment control!

In this assignment, you are expected to implement an **Brent's method hashing** to index words of a document named as 'story.txt'. You must read this file, split it word by word, and index each word to your hash table according to rules given below.

RULES

- Usage of Abstract Data Types (ADT) is required. Object Oriented Principles (OOP) and try-catch exception handling must be used when it is needed. You will use Java.
- Hash code for converting each word to an integer key must be implemented by yourself. The input value will be the word and the integer key will be returned by your hash code function. The number of occurrence of each word must be stored as *count* value.
- The hash (compression) function for converting a key to the index (address calculator) must be implemented by yourself.
- When a word is searched in the hash table; key, count, and index of the word should be printed.

The Hash table should look like as below:



The standard output of your program should appear as follows:

------ Brent's Method Hashing ------

Search: Altug
Key: 1243225
Count : 10
Index : 165

Search: Ali
Key: 68294842
Count : 3
Index : 132

Note: Results of the words of "Altug" and "Ali" were generated as an example. So, you will not obtain the same results by using the 'story.txt' file.

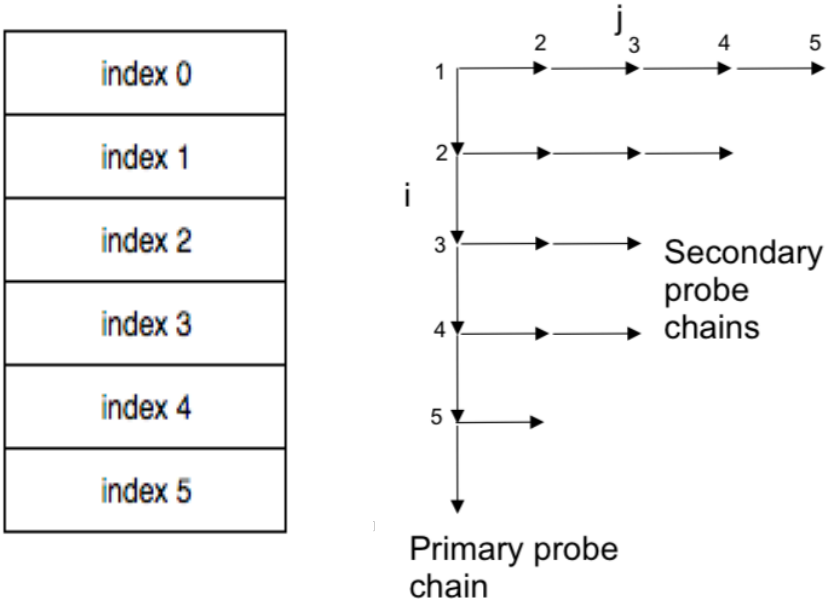
Main Functionalities:

- **INSERT:** Read the given input .txt file, calculate the number of occurrence of each word, insert this data into the hash table accordingly.

- **SEARCH:** Search a user input (word) in the hash table. If the input is available in the table, then return an output as shown above, otherwise return a "not found" message to the user.

BRENT'S METHOD HASHING

Brent's Method is a dynamic hashing method which was designed by R.P. Brent in 1973. Brent hashing was originally developed to make the double-hashing process more efficient, but it can be successfully applied to any closed hashing process. In the closed hashing process, all keys are stored in the hash table's internal array without using linked lists.



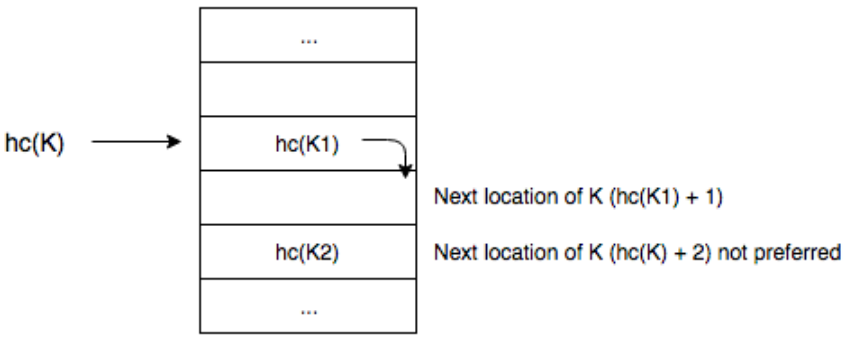
The order, in which the algorithm searches the hash table for candidate positions, is referred to as a "probing sequence" (also called "probe chain"). Brent's idea is to move elements on the probe chain with small path length. In this way, long probing sequences can be avoided and the algorithm becomes more efficient and faster. There are two probe chains called **primary** and **secondary probe** chains as shown above.

There may be several records on the primary probe chain. Let **j** be the number of extra accesses needed to access a record R, assuming that we choose to move R. Let **i** be the number of locations that we must visit to access the record we are inserting, assuming we move R. As a result we want to choose R such that **i + j** should be minimum.

Finding i+j with minimum value can be viewed as a search operation in a **two dimensional space**. The vertical axis shows the points in the primary probe chain. Each point on the axis is a location that the key we are inserting hashes to. Each horizontal line is the chain of locations that each existing record that we are thinking about moving hashes to. If there is no any element in the index, in which an object is to be placed, indexing is performed without collision, so there is no need to any intervene. Otherwise you need to do some extra operations.

Assume that K is a key and the hash function h(K) returns an index. The cases that may occur while inserting an element:

- If K's first probe location (index) is empty, just insert K in this index (which is the best possibility)
- If K's first probe location is full (collision), and second probe location is empty, just insert K to this second index
- But suppose increment value of K is 2 and collides with a key K1 at K's first probe location and with another key K2 at K's second location:
    - Think about moving K1 to K1's next probe location, and putting K in its place
    - If increment value of K1 is 1 and K1's next location is empty, this will increase K1's successful probe path length by 1



Suppose a new key K to be inserted has probe $p_1, p_2, ..., p_{v-1}, p_v$ index sequence. If $v < 3$, just insert K in location p_v as usual; can't do better than that. For calculating the value mentioned above as the increment value, you should define an incrementing function as well as the hash function as follows:

Hash Function = **hash(key) = key *mod* M**

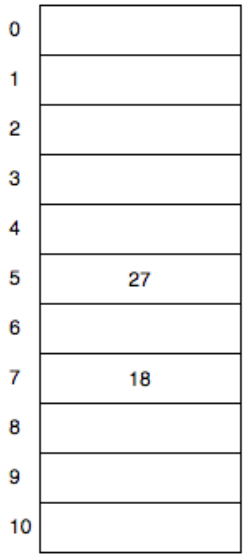
Incrementing Function = **i(key) = Quotient (Key / M) *mod* M**

Note: M is the size of hash table and quotient refers to division process.

Example:

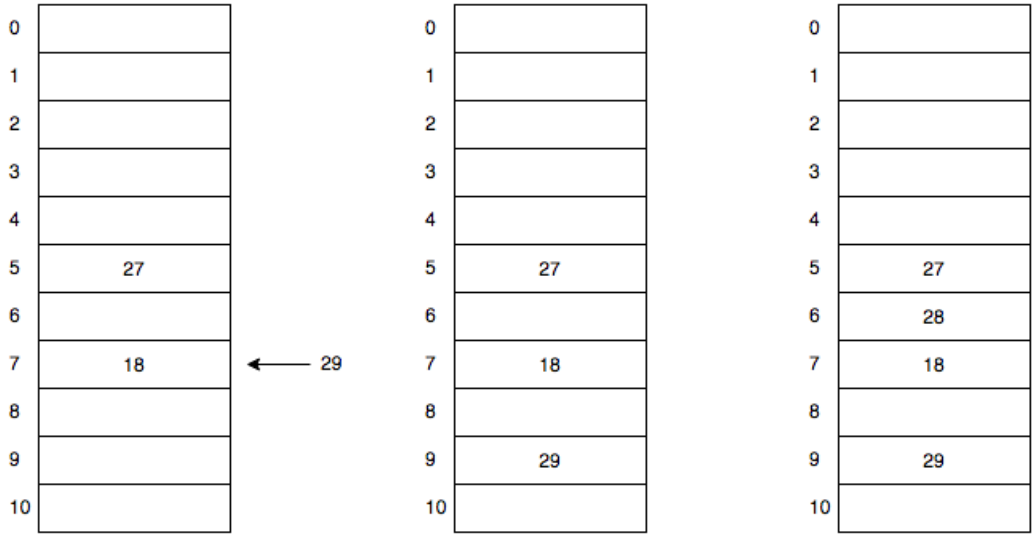
Place the following numbers in a hash table of size 11.

- Numbers: [27, 18, 29, 28, 39, 13, 16]
- Size (M) : 11
- Hash Function = hash(key) = key *mod* 11
- Incrementing Function = i(key) = Quotient (Key / M) *mod* 11
- 27 and 18 hash to 5 and 7 indexes without collision.



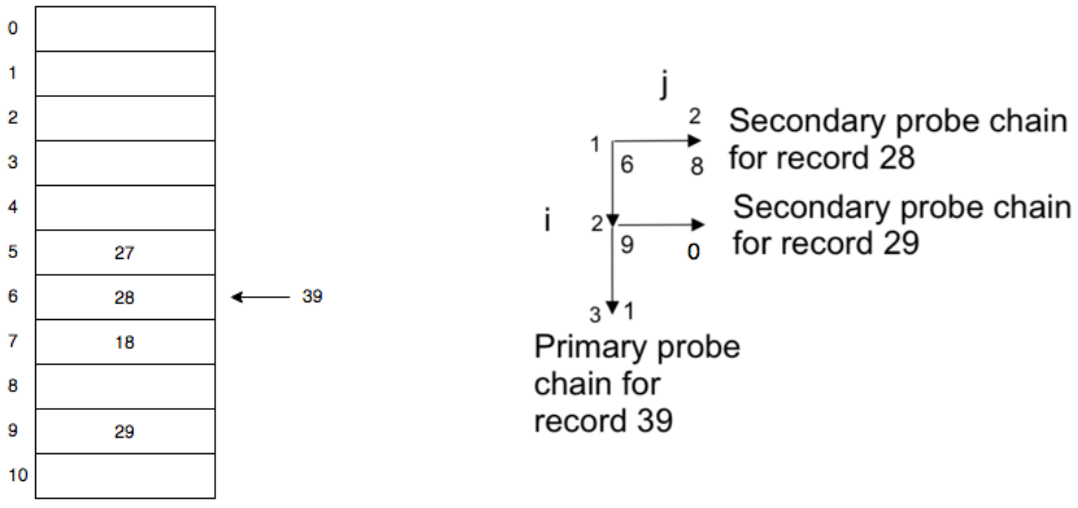
- 29 will be tried to be placed at the same index of 7 with the number 18, there is no any combination of $i + j < 2$, so we calculate incrementing function to move another address. i(29) = 2, so try (7+2)=9 as index.

- 28 mod 11 = 6 there is no collision



- 39 mod 11 = 6, so 39 hashes to 6, colliding with the number 28. i(39)=(39/11) mod 11 = 3. So, we try to place 39 in location (6+3) mod 11=9. This collides with 29, so we try location (9 + 3) mod 11 = 12 mod 11 = 1; it is the case when located 39. But is that a good choose ?

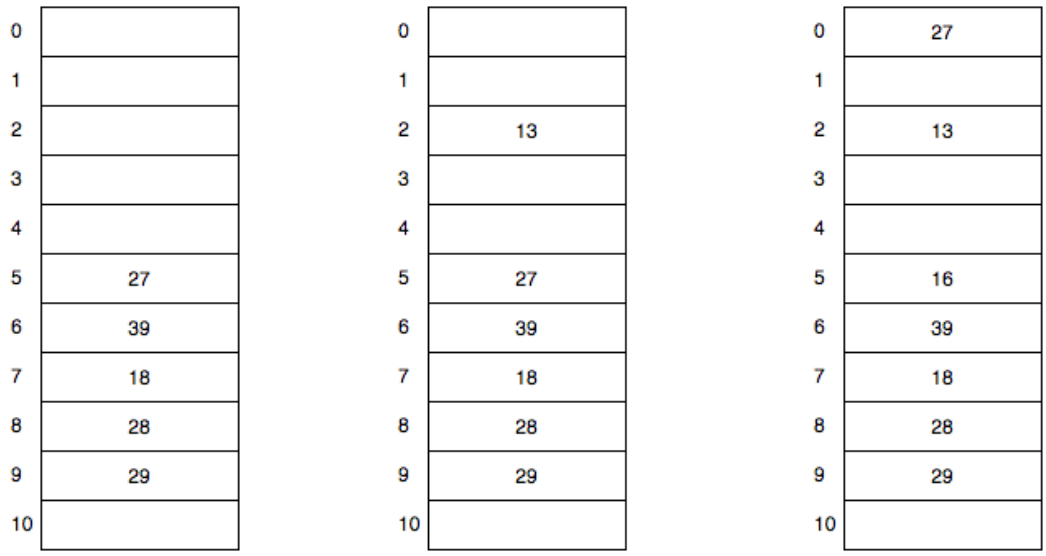
- To answer this question, the probability calculation is performed using the primary and secondary probe chains.



- If we were to place record 39 at point 1, we would be at point (1,3) in the two dimensional space with i+j = 1+3=4.

- We first look at moving the record at point (1, 1), that is record 28, which is in location 6 of the table. We try to move it to point (1,2), which is the second place that record 28 can be placed in the table. i(28)=28/11=2 => 6+2=8. Location 8 is free, so record 28 can be moved. Point (1, 2) gives a total i + j = 1 + 2 = 3.

- 3 < 4, so then move record 28 from location 6 to location 8; insert 39 at location 6.



- 13 mod 11 = 2, there is no record, simply insert it to location 2.

- Record 16 hashes to location: 16 mod 11 = 5, which is already occupied by record 27. The offset, i(16) = 16/11 = 1. Locations 5, 6, 7, 8, and 9 are already occupied, and the first empty location is 10. But the length of chain is 6.

- That why, we need to move record 27 to the fourth position of its secondary probe chain. i(27) = 2. (5 + 2 + 2 + 2) mod 11 = 11 mod 11 = 0. Location 0 is empty, so it works.

Grading Policy

Items	**Percentage**
1. Usage of ADT, OOP and Try-Catch	%30
2. Implementation of Brent's method hashing	%50
3. Search over hash table	%20

Note: Your assignment will get zero if the **first** and the **second** grading items are not provided.