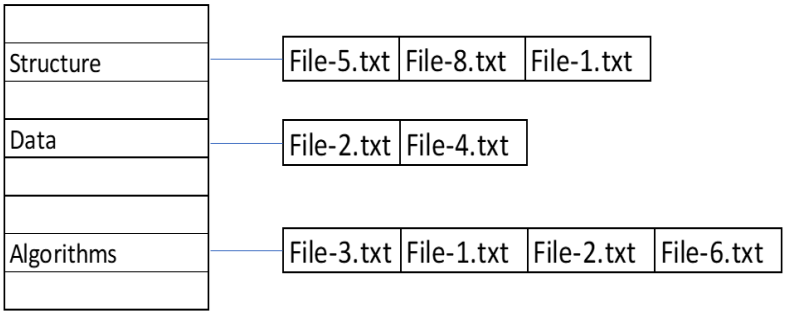
CME 2201 - Assignment 1

A Simple Text-Based Search Engine

As you know, most of the information you need is provided by internet search engines. One of the most preferred search engines is still Google. The fundamental factors behind the success of Google are quickness and correctness. How does Google retrieve quick and correct results for user queries? In this assignment, you will investigate the answers of this question. You will develop a simple text-based search engine. Your program only operates on text files that are kept in a single directory. We will provide necessary text documents into our document database.
In this assignment, you are expected to implement a **hashing algorithm** in Java to index words of documents given in X. You must read these documents, split it word by word, and index each word to your hash table according to rules given below. After insertion of all documents, the user will query for a sentence that contains 3 words splitted by a single space. Your algorithm must bring most relevant document for the query written by the user.

Rules:

- Object Oriented Principles (OOP) and try-catch exception handling must be used when it is needed.
- Hash function for converting a word to a key must be implemented by yourself. The value will be the word and the key will be returned by your hash function. The number of occurrences of each word also must be stored as *count* value for each *document* separately.



Note: You are free to choose the data structure to store file references and necessary informations.

- You should write a relevance measure to prioritize the most relevant document. After calculating distances of the resulting documents to the given query, you should sort documents according to these distance values, i.e., the smaller distance gets a higher rank in the resulting list.
Explanation: Assume that user entered “data structure algorithms” as a query. You searched all words in your hash table and results:

Occurrences:

	File-1.txt	File-2.txt	File-3.txt
data	78	43	56
structure	23	65	88
algorithms	45	55	15

Which txt file is the most relevant?
You must find appropriate solution for this problem.

Main Steps:

1- Remove punctuation and stop words.
2- Create necessary data structures (Hash Table) and insertion.
3- Search the given sentence and find the most relevant document.

Hash Function

To specify an index corresponding to given string key, firstly you should generate an integer hash code by using a special function. Then, resulting hash code has to be converted to the range 0 to N-1 using a compression function, such as modulus operator (N is the size of hash table).

You are expected to implement two different hash functions including simple summation function and polynomial accumulation function.

Simple Summation Function (SSF)

You can generate the hash code of a string *s* with the length *n* simply by the following formula:

$$\mathrm{h(s)} = \sum_{k=0}^{n-1} ch_k$$

Polynomial Accumulation Function (PAF)

The hash code of a string *s* can also be generated by using the following polynomial:

$$\mathrm{h(s)} = ch_0 * z^{n-1} + ch_1 * z^{n-2} + \ldots + ch_{n-2} * z^1 + ch_{n-1} * z^0$$

where ch_0 is the left most character of the string, characters are represented as numbers in 1-26 (case insensitive), and *n* is the length of the string. The constant *z* is usually a prime number (31, 33, 37, and 41 are particularly good choices for working English words). When the *z* value is chosen as 31, the string "car" has the following hash value:

$$h(\mathrm{car}) = 3 * 31^2 + 1 * 31 + 18 * 1 = 2932$$

Note: Using this calculation on the long strings will result in numbers that will cause overflow. You should ignore overflows or use Horner's rule to perform the calculation and apply the modulus operator after computing each expression in Horner's rule.

Collision Handling

Linear Probing (LP)

Linear probing handles collisions by placing the colliding item in the next (circularly) available table cell.

Double Hashing (DH)

Double hashing uses a secondary hash function *d(k)* and handles collisions by placing an item in the first available cell of the series.

$$d(k) = q - k \bmod q$$

$$h_2(k) = \big(h(k) + j\, d(k)\big) \bmod N$$

where $q < N$ (table size), q is a prime, and $j = 0, 1, \ldots, N-1$.

The secondary hash function *d(k)* cannot have zero values. The table size N must be a prime to allow probing of all the cells.

Example:

N = 13, k= 31, q= 7, h(k) = k mod 13 = 5, d(k) = 7 - k mod 7 = 4.	The 1st lookup index: 5 The 2nd lookup index: 5+ 1*4 = 9 % 13 = 9 The 3rd lookup index: 5+2*4 = 13 % 13 = 0 …

Performance Monitoring

You are expected to fill the performance matrix (Table 1) by running your code under different conditions including two different load factors (50% and 80%) to decide resizing of hash table, two different hash functions (SSF and PAF) and two different collision handling techniques (LP and DH).

You should count total number of collision occurrences and calculate spent time while loading documents into the inverted index structure under each condition. In addition, you should calculate min., max. and avg. search times by using the "search.txt" file that contains 1000 words to search (Search time means the time expended to find a particular key in the hash table. It does not include the time spent for outputs. To calculate avg. search time, divide the total expended time to the total number of searched keys). You can use System.nanoTime() or System.currentTimeMillis() for time operations.

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP					
		DH					
	PAF	LP					
		DH					
α=80%	SSF	LP					
		DH					
	PAF	LP					
		DH					

Table 1. *Performance matrix*

Write a report that contains table 1 and discuss your results on the report. Choose the best option for the text-based search engine for the upload on Sakai.

Provided Resources

- English stop-words lists (stop_words_en.txt)
- Delimiters to split document content (delimiters.txt)
- Documents to index: BBC news article datasets [1] (sport.rar)
- Word list to use in calculation of searching times (search.txt)

Note:

Double Hashing Function: $D_2(key) = 31 - (key\%31)$

Initial Table Size must be 2477

Submission

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., 2007510011_Ali_Yılmaz.rar and it should be uploaded in the SAKAI portal.

Prepare and upload a report with descriptions of your data structure, java code, and performance matrix.

You can ask your questions from the "FORUM -> Homework 1 - Questions" part of the SAKAI portal.

Due date: 08.12.2022 Sunday 23:55. Late submissions are not allowed.

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

Grading Policy

Job	**Percentage**
Usage of Generic, OOP and Try-Catch	%30
Hash Table implementation and query execution	%50
Performance monitoring and Report	%20

References

[1] D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.