Development of Bengali Language Stemmer

Project Report

Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Technology

Submitted by

Barnan Das
&
Tanmoy Pal

Under the guidance of

Dr. Pabitra Mitra
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

CERTIFICATE

This is to certify that the project report entitled "**Development of Bengali Language Stemmer**" is a record of bona fide work carried out by Mr. Barnan Das & Mr. Tanmoy Pal of Bengal College of Engineering and Technology, Durgapur under my supervision and guidance, as part of their Final Year Project 2009, at the Indian Institute of Technology, Kharagpur.

Dr. Pabitra Mitra
Dept. of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Date:

Place: Kharagpur

ABSTRACT

Since the day man started realizing the importance of information it became necessary for archiving those information in such a way that they become easy to retrieve in the future. The advent of computers made it possible to store large amounts of data or information and thus retrieving those data became a necessity. The area of Information Retrieval (IR) was born in 1950s and since then several IR systems are being developed and used everyday by millions of people all over the world. English being a widely accepted language all over the world, most of the IR systems, web based or stand alone systems, are developed for English documents or contents. A little has been done for Bengali documents. Bengali is the fourth largest language of the world. There is great need for developing technology for processing Bengali language text. A particularly important task is that of developing a search engine for Bengali documents. Many technologies required for this is yet to be developed in Bengali. The goal of this project is to develop the technologies for Bengali and the focus is primarily on developing algorithms for stemming. Stemming is the process of reducing word to its stem or root form in order to increase the recall rate and hence are used widely in information retrieval tasks. Most popular stemmers like Porter's and Lovin's encode a large number of language-specific rules built over a length of time. But, Bengali being a highly inflectional language, where one root may have more than 20 morphological variants, it becomes difficult and time consuming to formulate a set of stemming rules. In this project we try to implement a clustering based stemming algorithm proposed by P. Majumder, M. Mitra, S.K. Parui, G. Kole, P. Mitra and K. Dutta in their paper *YASS: Yet another suffix stripper, ACM Translations on Information Systems*, Vol. 25, No. 4, pp. 18-38, October 2007.

ACKNOWLEDGEMENTS

We take this opportunity to express our deep sense of gratitude to our guide **Dr. Pabitra Mitra** for his guidance, support and inspiration throughout the duration of the work.

Also, we would like to thank **Dr. (Prof.) Subrata Dasgupta**, Head of the Department, Computer Science and Engineering, Bengal College of Engineering and Technology, Durgapur for being a constant source of inspiration.

Finally, we would like to thank the research scholars of IIT Kharagpur and our friends at BCET for their invaluable suggestions and comment that led to the successful completion of the project.

Barnan Das

Tanmoy Pal

List of Figures

CONTENTS

Chapter 1

Introduction

1.1 Brief History [1]

The practice of archiving written information can be traced back to around 3000 BC, when the Sumerians designated special areas to store clay tablets with cuneiform inscriptions. Even then the Sumerians realized that proper organization and access to the archives was critical for efficient use of information. They developed special classifications to identify every tablet and its content.

The need to store and retrieve written information became increasingly important over centuries, especially with inventions like paper and the printing press. Soon after computers were invented, people realized that they could be used for storing and mechanically retrieving large amounts of information. In 1945 Vannevar Bush published a ground breaking article titled “As We May Think” that gave birth to the idea of automatic access to large amounts of stored knowledge. In the 1950s, this idea materialized into more concrete descriptions of how archives of text could be searched automatically. Several works emerged in the mid 1950s that elaborated upon the basic idea of searching text with a computer. One of the most influential methods was described by H.P. Luhn in 1957, in which (put simply) he proposed using words as indexing units for documents and measuring word overlap as a criterion for retrieval.

Several key developments in the field happened in the 1960s. Most notable were the development of the SMART system by Gerard Salton and his students, first at Harvard University and later at Cornell Univer-sity; and the Cranfield evaluations done by Cyril Cleverdon and his group at the College of Aeronautics in Cranfield. The Cranfield tests developed an evaluation methodology for retrieval systems that is still in use by IR systems today. The SMART system, on the other hand, allowed researchers to experiment with ideas to improve search quality. A system for experimentation coupled with good evaluation methodology allowed rapid progress in the field, and paved way for many critical developments.

The 1970s and 1980s saw many developments built on the advances of the 1960s. Various models for doing document retrieval were developed and advances were made along all dimensions of the retrieval process. These new models/techniques were experimentally proven to be effective on small text collections (several thousand articles) available to researchers at the time. However, due to lack of availability of large text collections, the question whether these models and techniques would scale to larger corpora remained unanswered. This changed in 1992 with the inception of Text Retrieval Conference, or TREC. TREC is a series of evaluation conferences sponsored by various US Government agencies under the auspices of NIST, which aims at encouraging research in IR from large text collections.

With large text collections available under TREC, many old techniques were modified, and many new techniques were developed (and are still being developed) to do effective retrieval over large collections. TREC has also branched IR into related but important fields like retrieval of spoken information, non-English language retrieval, information filtering, user interactions with a retrieval system, and so on. The algorithms developed in IR were the first ones to be employed for searching the World Wide Web from 1996 to 1998. Web search, however, matured into systems that take advantage of the cross linkage available on the web.

1.2 Information Retrieval

According to the book "An Introduction to Information Retrieval" [2] defines Information Retrieval as follows:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

The user's information need is represented by a *query* or *profile*, and contains one or more *search terms*, plus perhaps some additional information such importance weights. Hence, the retrieval decision is made by comparing the terms of the query with the *index terms* (important words or phrases) appearing in the document itself. The decision may be binary (retrieve/reject), or it may involve estimating the degree of relevance that the document has to the query.

Information retrieval systems can be distinguished by the scale at which they operate, and it is useful to distinguish three prominent scales. In *web search*, the system has to provide search over billions of documents stored on millions of computers. Distinctive issues are needing to gather documents for indexing, being able to build systems that work efficiently at this enormous scale, and handling particular aspects of the web, such as the exploitation of hypertext and not being fooled by site providers manipulating page content in an attempt to boost their search engine rankings, given the commercial importance of the web. At the other extreme is *personal information retrieval*. In the last few years, consumer operating systems have integrated information retrieval (such as Apple's Mac OS X Spotlight or Windows Vista's Instant Search).

1.3 Performance Measures

Many different measures for evaluating the performance of information retrieval systems have been proposed. The measures require a collection of documents and a query. Every document is known to be either relevant or non-relevant to a particular query.

Precision

Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

Fall-Out

The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available:

$$\text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}$$

1.4 Stages of IR

The IR task is generally divided into two major components: indexing and retrieval. The indexing process typically represents documents as a collection of keywords and their corresponding weights and usually consists of the following steps:

Tokenization: Tokenization is the process of chopping character streams into tokens; linguistic preprocessing then deals with building equivalence classes of tokens, which are the set of terms that are indexed.

Stop-word Removal: Removing those words from the text which are extremely common. This can be done by hand filtering the words which have very high frequency.

Stemming: Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Phrase Recognition: Phrase Recognition is the process of recognizing phrases of the query so that relevant documents could be returned.

Term Weighting: Term Weighting refers to the process of weighing terms on the basis of its frequency.

Chapter 2

Stemming

2.1 Introduction [3]

Stemming is an operation that splits a word into the constituent root part and affix without doing complete morphological analysis. It is used to improve the performance of spelling checkers and information retrieval applications, where morphological analysis would be too computationally expensive.

2.2 Previous Work

The first ever published stemmer was written by Julie Beth Lovins in 1968. This paper was remarkable for its early date and had great influence on later work in this area.

A later stemmer was written by Martin Porter and was published in the July 1980 issue of the journal *Program*. This stemmer was very widely used and became the de-facto standard algorithm used for English stemming.

Later, few other stemmers were developed by Paice & Husk, Dawson and Krovetz. Most of these stemmers were rule based stemmers and have proved to be milestones n the Suffix Stripper Stemming approach. Specially, the Porter's Stemming Algorithm has proved to be an invaluable resource to researchers who work on stemmers.

2.3 Types of Stemming

There are several types of stemming algorithms which differ in respect to performance and accuracy and how certain stemming obstacles are overcome.

Brute Force Algorithms: Brute force stemmers employ a lookup table which contains relations between root forms and inflected forms. To stem a word, the table is queried to find a matching inflection. If a matching inflection is found, the associated root form is returned.

Suffix Stripping Algorithms: Suffix stripping algorithms rely on a typically small list of "rules" stored which provide a path for the algorithm, given an input word form, to find its root form.

Lemmatisation Algorithms: This process involves first determining the part of speech of a word, and applying different normalization rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on a word's part of speech.

Stochastic Algorithms: Stochastic algorithms involve using probability to identify the root

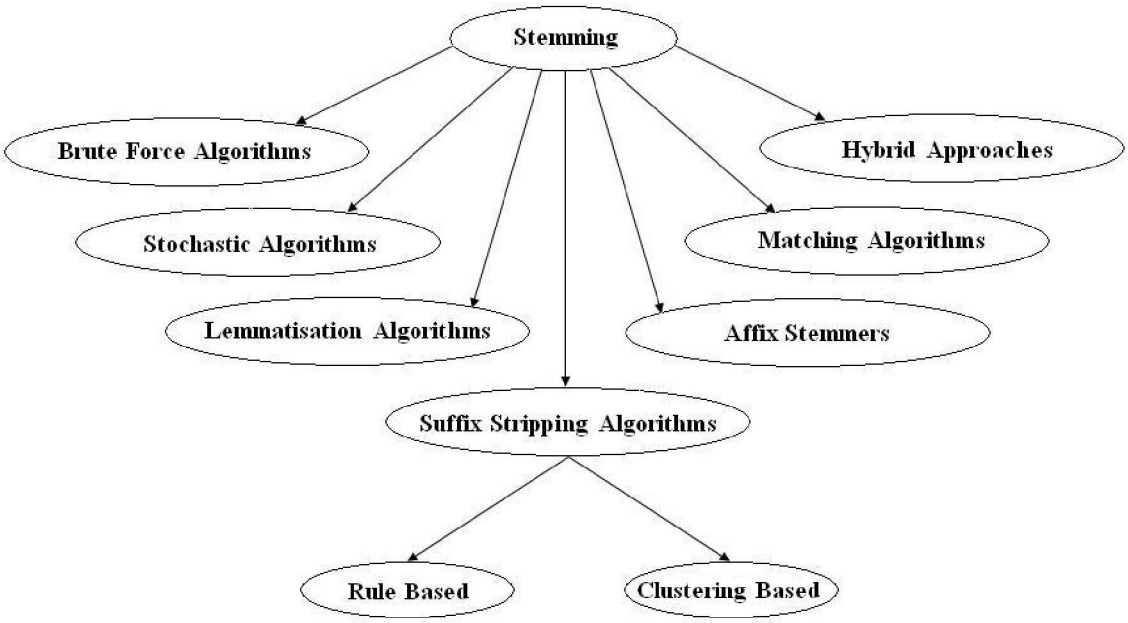


Fig: Types of Stemming

relations to develop a probabilistic model. This model is typically expressed in the form of complex linguistic rules. Stemming is performed by inputting an inflected form to the trained model and having the model produce the root form according to its internal rule set,.

Hybrid Approaches: Hybrid approaches use two or more of the approaches described above in unison. A simple example is a suffix tree algorithm which first consults a lookup table using brute force.

Affix Stemmers: In linguistics, the term affix refers to either a prefix or a suffix. In addition to dealing with suffixes, several approaches also attempt to remove common prefixes. Many of the same approaches mentioned earlier apply, but go by the name affix stripping.

Matching Algorithms: Such algorithms use a stem database (for example a set of documents that contain stem words). These stems are not necessarily valid words themselves (but rather common sub-strings). In order to stem a word the algorithm tries to match it with stems from the database, applying various constraints, such as on the relative length of the candidate stem within the word.

2.4 Advantages of Clustering Based Stemmer Over Rule Based Stemmer[4]

- Clustering Based Stemmer is not based on language specific rules, thus it do not require any knowledge of the language.

- In the absence of extensive linguistic resources for certain languages, clustering based stemmer show improvement in the performance of the IR system than rule based stemmer.

Chapter 3

Bengali Language Stemming

3.1 Bengali as a Language[5]

Bengali is the fourth most widely spoken language in the world. A very rich language indeed, Bengali is spoken by approximately 10% of the world's population. In the written form of Bangla there are 11 vowels and 39 consonants. Moreover, there are 10 short forms of vowels called vowel modifiers (i.e. Kar), 7 short forms of consonants called consonant modifiers (i.e. Fala). Besides there are about 253 compound characters composed of 2, 3 or 4 consonants (20 compound characters composed of 2 consonants, 51 compound characters composed of 3 consonants and 2 compound characters composed of 4 consonants)

3.2 Need for Bengali Language Stemmer

Since the inception of the concept of stemming in 1968, there has been a lot of progress in the area with varied range of approaches. The simplicity of languages like English, and other English-Like Languages like French, Spanish, German has enabled the development of stemmers for them. Moreover, these languages being web-resource enriched, make the work of the researchers easy. On the other hand, some languages, in spite of being highly inflectional like Chinese, Japanese and Arabic have made the development of stemmers possible because they are resource enriched. But, the languages which are highly inflectional and resource poor at the same time have always been treated with negligence. This category mainly comprises the Indic languages, especially Bengali. The last two decades has witnessed an immense escalation of Bengali web and digital text contents and is having an exponential growth rate. This has enhanced the need for the development of highly efficient IR systems and consequently good stemmers.

3.3 Problems Related to Bengali Stemming [4,6]

- Bengali is a highly inflectional language where 1 root may have more than 20 morphological variants. In most cases, variants are generated by adding suffixes to the root word.

- There also exists a large set of compound words where two roots can join together to form a compound word, and that compound may also have some morphological variants.

- There are frequent uses of conjunctive characters which are formed by a combination of 2 or even 3 consonants.

- The Unicode[7] ordering of Bengali characters is not the same with the order proposed by the Bangla Academy, Bangladesh [8] and Pachimbanga Bangla Academy, India [8].

Control Flow Diagram

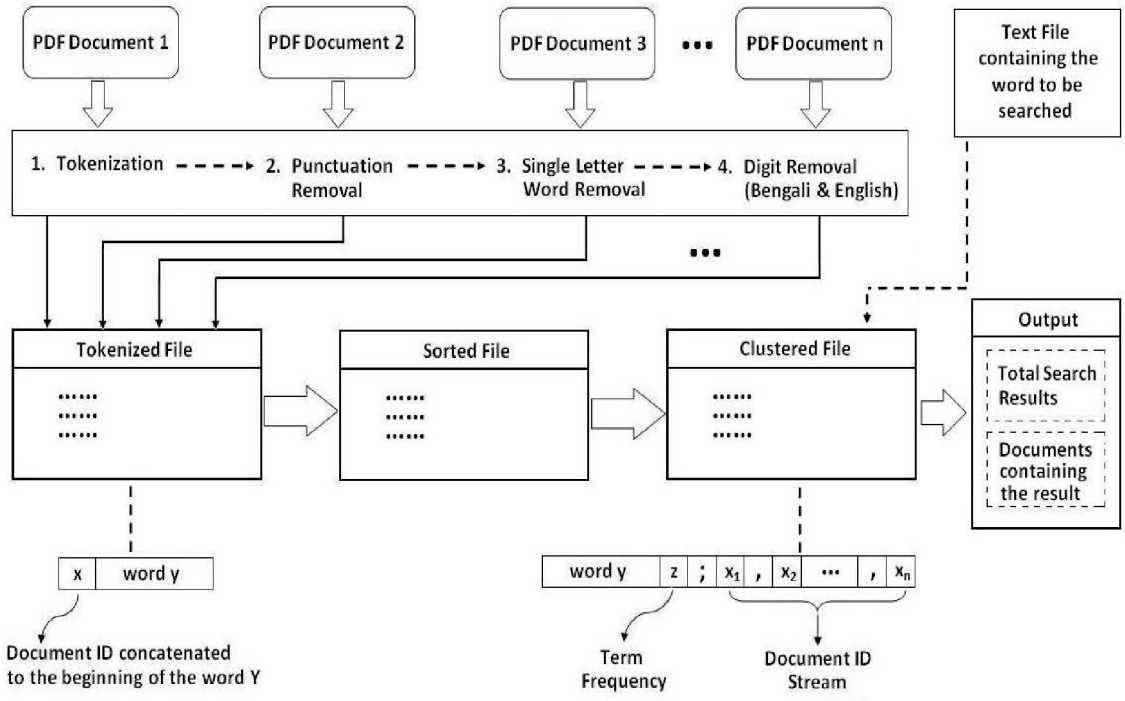
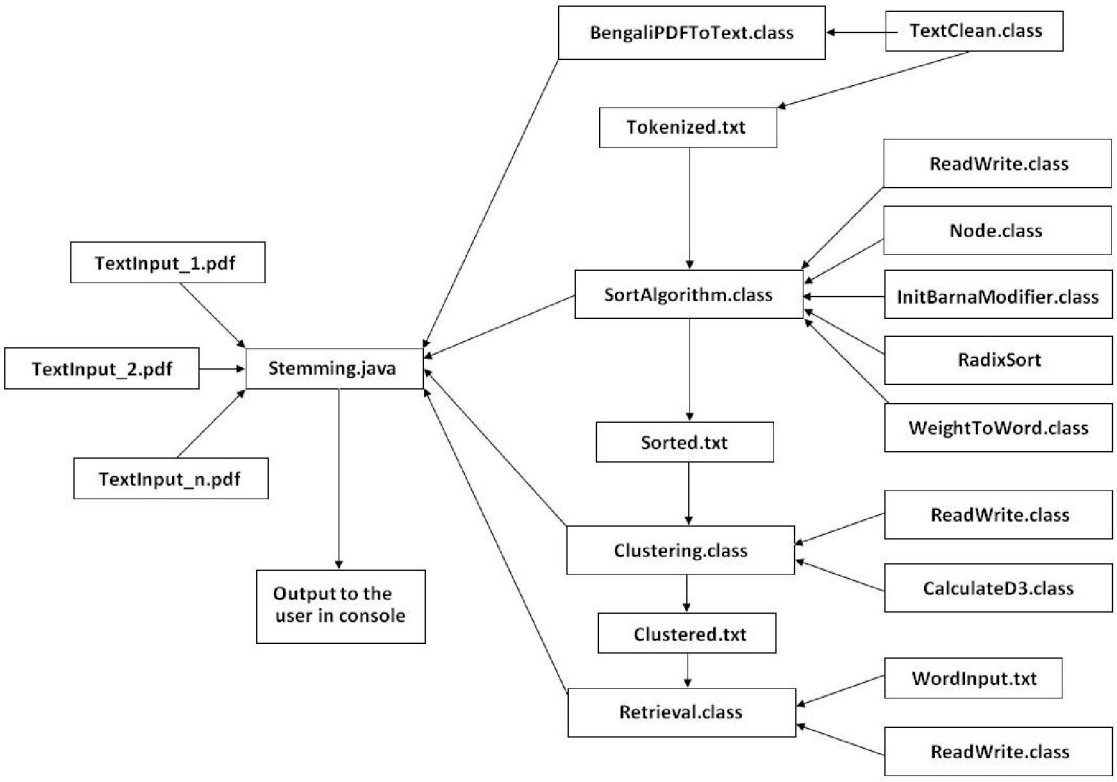


Fig: Control Flow Diagram

Logical Flow Diagram



Chapter 4

System Requirements

4.1 Hardware Requirements

Sl. No.	Description	
1.	Hard disc	40 GB.(Depends upon the size of the corpus)
2.	RAM	256 MB.
3.	Processor	Min speed 1.7 GHz

4.2 Software Requirements

Sl. No.	Description	Used	Alternatives(If available)
1.	Programming Language compiler	JDK 1.6	JDK 1.5
2.	Bengali Keyboard	Avro Keyboard	Baisakhi Keyboard
3.	Operating System	Windows XP	Windows 98 or above
4.	Additional JAVA package	PdfBox	Etymon

4.3 Other Specification

- Operating System should be reconfigured to support Bengali Fonts.

Chapter 5

Bengali Text Extraction

5.1 Introduction

We have used JDK 1.6 here. But java by default does not support or provide any class or method for accessing PDF files, albeit there are classes and methods for text extraction from Text files. As we need to read from different PDF files for getting Bengali text and extracting them to text files, we have searched the internet for such classes and methods. Some of them are Pdfbox, Etymon, Jpdftext, etc. Among these we have chosen Pdfbox package because of its simplicity. By using different methods and class files we have developed a java file for text extraction from PDF.

5.2 Code Segment

```
/*    Conversion from Bengali PDF to Bengali Text file      */

import java.io.*;
import java.util.*;
import java.lang.*;
import org.pdfbox.util.PDFTextStripper;
import org.pdfbox.pdmodel.PDDocument;
import org.pdfbox.cos.COSDocument;

class BengaliPdfToText
{     static void writeOutput(String str)   throws IOException
      {
      FileOutputStream fos = new FileOutputStream("Files\\TestOutput.txt");
      Writer out = new OutputStreamWriter(fos, "UTF8");
      String x=new String("Files\\TestOutput.txt");
      out.write(str);
      out.close();
      }
      static void textExtraction(String Str)throws IOException, …
                                              … UnsupportedEncodingException

      {     String s = new String();
            String ret=new String();
            PDDocument pdf_file = new PDDocument();
            PDFTextStripper stripper = new PDFTextStripper();
      //Extracting text from PDF file as a string
            s=stripper.getText(pdf_file.load("Files\\Str"));
      //Write output to TestOutput.txt file
            writeOutput(s);
      }

}
```

Chapter 6

Tokenization and Punctuation, Digit (Bengali & English), Single-Letter-Word Removal

6.1 Introduction

We are dealing with the Bengali documents taken from directly many PDF files. Obviously these texts will have sufficient number of Bengali punctuations, Bengali and English digits and Single-letter-words. But while developing a Bengali Stemmer we need not take these unnecessary characters into consideration. So by during this phase we remove these characters step by step.

6.1.1 Tokenization

Tokenization is the process of breaking the sentences as well as the text file into word delimited by white space or tab or new line etc. Outcome of this tokenization phase is a set of word delimited by new line.

6.1.2 Punctuation Removal

As we are working on general Bengali text, there are lots of Bengali punctuations in the text. These characters have no importance in our stemming project. So we removed these punctuations by using their Unicode representation.

6.1.3 Digit Removal

A general Bengali text file may contain Bengali as well as English digits. But as meaningful Bengali words do not contain digits, we remove these digits by using their Unicode representation.

6.1.4 Single Letter-Word Removal

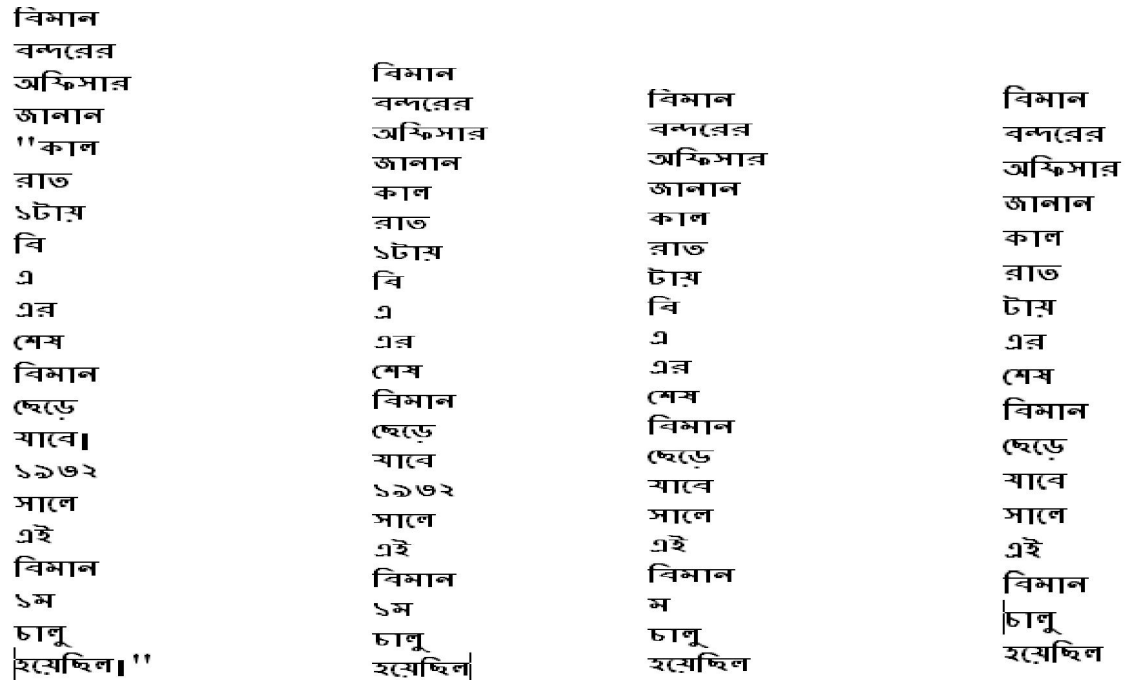
There exist a lot of words having a single letter. Most of these Single-Letter-Words are Stop-Words (Those words which have extremely high term frequency in a corpus are know as *Stop Words*). As a step of our stemming the stop-words need to be removed before further processing. So the Single-Letter-Words are removed in this phase.

6.2 Illustration

Initial Text:

বিমান বন্দরের এক অফিসার জানান"কাল রাত ১টায় বি এ এর শেষ বিমান ছেড়ে যাবে।১৯৩২ সালে এই বিমান ১ম চালু হয়েছিল।"

Transition Steps:



Tokenization Punctuation Removal Digit Removal Single-Letter-Removal

6.3 Algorithm

1. There is a manual list containing the Unicode representation of different punctuations & digits.

Punctuation: । , ; : ? ! ‘ “ ‟ ( ) { } [ ] / \ + - %

Unicode: [**2404,44,59,58,63,33,34,39,40,41,123,125,91,93,47,92,43,45,37**]

Bengali Digits: ০ ১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯

Unicode: [**2534,2535,2536,2537,2538,2539,2540,2541,2542,2543**]

English Digits: **0, 1, 2, 3, 4, 5, 6, 7, 8 ,9**

Unicode: **[48,49,50,51,52,53,54,55,56,57]**

2. For all tokens in the text file
Read every token.

3. If any token contains any letter having the Unicode listed in the previously mentioned list, that letter will be discarded from the list of String (*finalStr*).

4. The ‘Beginning of File’ character having Unicode 65279 is also removed.

5. Removal of Single Letter Word.

6.4 Complexity

The main coding for this part is in TextClean.java file. The while loop executes for all the tokens in the list. For every token there is a for loop which executes for the length of the token. The complexity of this Text Clean.java program is – No. of token in the List * Length of each token.

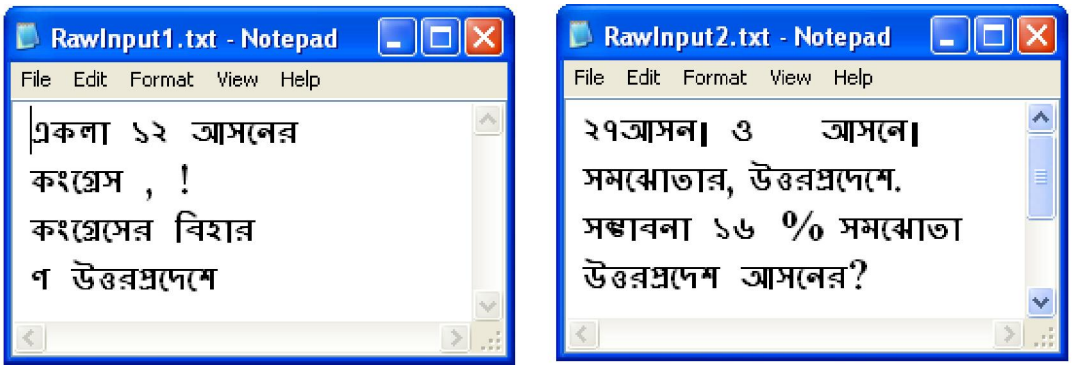
So for 'n' No. of tokens having average length 'l' **$T(n)=n*l \approx O(n^2)$.**

6.5 Code Segments

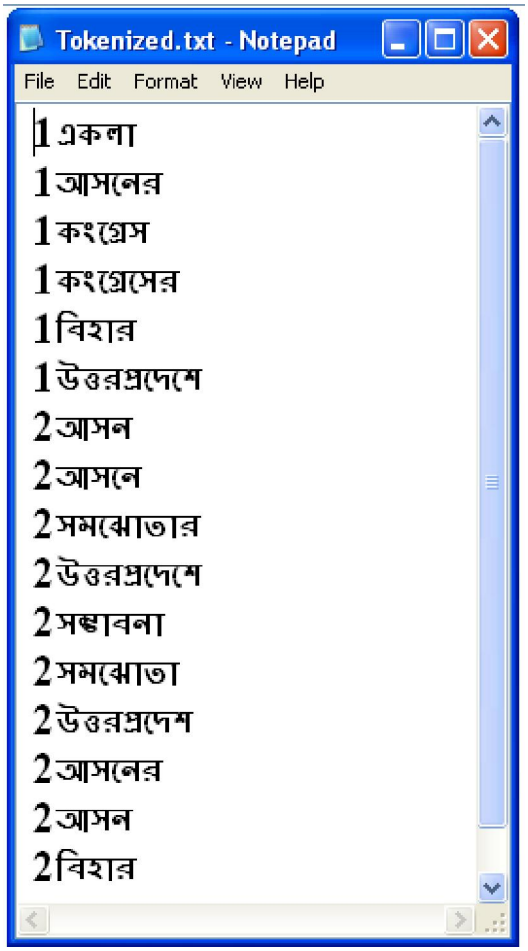
```
/* Removal of punctuation, digits and single letter words   and at the end
Tokenizing the file*/

import java.io.*;
import java.util.*;
import java.lang.*;
class TextClean
{     static String removal(LinkedList<Integer> word1)
      {     String str2=new String();
            for(int j=0;j<word1.size();j++)
            {     char c[ ] = Character.toChars(word1.get(j));
                  String str1=new String(c);
                  str2=str2+str1;
            }
            return str2;
      }
      static void clean(String tokenFile, int doc)
      {     int i;
            Integer docID = new Integer(doc);
            ReadWrite rw = new ReadWrite();
            LinkedList<Integer> tokenList = new LinkedList<Integer>();
            String inputString = rw.read(tokenFile);
            StringTokenizer StrToken = new StringTokenizer(inputString);
            while(StrToken.hasMoreTokens())
            {     String  wordString = new String (StrToken.nextToken());
                  for (i=0; i<wordString.length(); i++)
                  {     if(wordString.codePointAt(i)!=65279)
                              tokenList.add(wordString.codePointAt(i));
                  }
                  System.out.println(tokenList.size());
                  System.out.println(tokenList);
                  for(int j=0;j<tokenList.size();j++)
                  {     if  ((tokenList.get(j)==2404)||
(tokenList.get(j)==24)||(tokenList.get(j)==59)||(tokenList.get(j)==37)||
(tokenList.get(j)==63)||(tokenList.get(j)==33)||(tokenList.get(j)==34)||
```

```
(tokenList.get(j)==39)||(tokenList.get(j)==40)||(tokenList.get(j)==23)||
(tokenList.get(j)==91)||(tokenList.get(j)==58)||(tokenList.get(j)==44)||
(tokenList.get(j)==46)||(tokenList.get(j)==4 )||(tokenList.get(j)==123)||
(tokenList.get(j)==125)||(tokenList.get(j)==93)||(tokenList.get(j)==2534)||
(tokenList.get(j)==2535)||(tokenList.get(j)==2536)||
(tokenList.get(j)==2537)||(tokenList.get(j)==2538)||
(tokenList.get(j)==2539)||(tokenList.get(j)==2540)||
(tokenList.get(j)==2541)||(tokenList.get(j)==2542)||
(tokenList.get(j)==2543))
                        {     tokenList.remove(j);
                              j=j-1;
                        }
                  else if (tokenList.get(j)>=48 && tokenList.get(j)<=57)
                        {     tokenList.remove(j);
                              j=j-1;
                        }
                  }
                  System.out.println(tokenList.size());
                  System.out.println(tokenList);
                  if(tokenList.size()>1)
                  {     String finalStr=removal(tokenList);
                        rw.append(docID.toString()+finalStr+"\r\n",…
                                                …"Files\\Tokenized.txt");
                  }
                  tokenList.clear();
            }

      }

}
```

6.6 Screen Shot



Input text files: RawInput1.txt & RawInput2.txt



After Tokenization

Chapter 7

Sorting

7.1 Introduction

English being a simple language can be sorted with the help of Radix Sort (considering 26 packets, one packet for each alphabet) or with the help of Unicode. Unicode is the universally used coding system that provides a unique number for every character irrespective of the platform, program and language [7]. However, the use of conjunctive characters and modifiers in Bengali has restricted the straightforward sorting of Bengali. Moreover, the Unicode sort order does not follow the sorting order suggested by Bangla Academy [8], an organization looking after the standardization, development and publicity of Bengali. In this project, we have used the sorting method suggested by Md. Ahsanur Rahman and Md. Abdus Sattar of Bangladesh University of Engineering and Technology, Dhaka, Bangladesh [6].

7.2 Algorithm

7.2.1 Bengali Sort Algorithm

1. Form a list containing all base letters of Bengali in dictionary order.

অ ,আ, ই, ঈ, উ, ঊ, ঋ, এ, ঐ, ও, ঔ, ং, ঃ, ঁ, ক, খ, গ, ঘ, ঙ, চ, ছ, জ, ঝ, ঞ, ট,

ঠ, ড, ড়, ঢ, ঢ়, ণ, ৎ , ত, থ, দ , ধ, ন, প, ফ, ব, ভ, ম, য, য়, র, ল, শ, ষ, স, হ

2. Form another list containing all modifier of Bengali in dictionary order.

া, ি, ী, ু, ূ, ৃ, ে, ৈ, ো, ৌ.

3. Read desired orders of base letters (vowels or consonants) and modifiers from the list by using Unicode representation of Bengali letters.

4. Construct a Collation Weight List containing Collation Weight of a letter.

Collation Weight of a base letter=12*it's position in the list at Step 1.

Collation Weight of a modifier letter=it's position in the list in Step 2.

5. Read input word list.

6. For each word in the list

    6.1 For each letter in the word

        6.1.1 Find the Collation Weight (C) of that letter from the Collation Weight List.

        6.1.2 If the letter is the Base Letter append C to the Weight String (W).

        6.1.3 Add C to the last element of Weight String (W).

7. Concatenate all the elements of the Weight String (W) to get a single String. For Example,

    W1 = {191,576,398} to 191576398. W2 = {181,264,512} to 181264512

8. Sort this Wordlist containing long integer type data by using any sorting technique.

    Wordlist={191576398,181264512}.

    In our case we have used Radix Sort.

9. Remap the each string from the sorted list to its corresponding word.

7.2.2 Radix Sort Algorithm [10]

1. Construct ten pockets for sorting integer value.
2. Repeat through step 6 for each digit in the element.
3. Initialize the pockets.
4. Repeat through step 5 until end of the list.
5. Obtain the next digit of the element.
6. Insert the element in appropriate pocket.
7. Combine the pockets to form a new list.

7.3 Complexity

This algorithm has been implemented (coded) in SortAlgorithm.java file which also calls other java/class files viz. InitBarnaModifier.java, RadixSort.java, WeightToWord.java.

In SortAlgorithm.java, the 'while loop' is executes till there are tokens in the list. There are many 'for loops' within the while loop and which execute for the length of the wordString.

So the complexity is No. of Token in the Text file * Length of the Token. If there is 'n' No. of token & average length of the token is 'l', then $T(n)= n*l \approx O(n^2)$.

7.4 Implementation

For sorting Bengali words we use the `SortAlgorithm.java` file which again uses other java files as it's supporting file. The supporting files we have used are `InitBarnaModifier.java`, `ReadWrite.java`, `RadixSort.java`, `WeightToWord.java`. For proper explanation of SortAlgorithm.java we explain the supporting java files also.

In `SortAlgorithm.java` we have created a `ReadWrite` object `rw` to read the tokenized text file `Tokenized.txt`. We have also created a `StringTokenizer` object to read `Tokenized.txt` file token wise. For every token in the file we check it's `DocID` which indicate the document no. of that token. The linked list `nodeList` contain the word itself and the `DocID`. Now for every single token we check it's individual letter, whether it is a `Barna` or `Modifier` . For this purpose we need to create an object of `InitBarnaModifier` type. If the letter is a `Barna` then multiply the index by 12 and put it into weight list. If the letter is a modifier then add the position index of that letter in modifier list with the last element of word list. To represent the word in more convenient way[6] we convert every element of word list into three digit integer no. and concatenate all those 3 digit no. into `concatweightlist`. Before sorting, for easy handling of the elements of the `concatweightlist` we append zeroes at the end of the string to convert every element into equally sized string.

Sorting is done by creating `RadixSort` object. The details of `RadixSort` is in the `RadixSort.java` file, where according to the rule of Radix Sort we have created an array of ten linked lists. Every Linked list represent a pocket for sorting. For every element in the `concatweightlist` and every digit of that element it will send it to the listing method where, depending upon the digit the element is put to its corresponding list. After a single pass all the elements of the pockets are retrieved and the pockets are made clear for next pass. This passing mechanism is done by `passing` method.

The last stage of the sorting is to get back the original Bengali word from the internal representation [6]. For this purpose we need to create an object of `WeightToWord` type. The details of this process is there in the `WeightToWord.java` file. Where we get back three digits for each letter & find out the corresponding letter from the `InitBarnaModifier.java` file.

Illustration Let us consider two words বর্ণন তন্ময়

বর্ণন = ব র ◌ ণ ন

Letter	ব	র	◌	ণ	ন
Weight	40*12=480	45*12=540	11	31*12=372	37*12=444
		540+11=551			

তন্ময় = ত ন ◌ ম য়

Letter	ত	ন	◌	ম	য়
Weight	33*12=396	37*12=444	11	38*12=456	44*12=528
		444+11=455			

Now word String for তন্ময় is {396,455,456,480} and word String for বর্ণন is {480,551,372,444}. So after concatenation `concatweightlist` will contain [396455456480, 480551372444]. As the length of both the string are equal so no need to append zeros.

Now we need to sort two number strings 396455456528 & 480551372444 as per the rule of Radix sort. So after radix sort, `concatweightlist` will become [396455456528 , 480551372444]. Another important phase is word retrieval from this representation. By dividing into three digit number from this string we will get [396,455,456,528] & [480,551,372,444].

Weight to Word conversion for string [**396,455,456,528**]

Number	396	455	456	528
Base Index	(396 - 0) / 12 = 33	(455-11) / 12=37	(456 – 0) / 12 = 38	(528 – 0) / 12 = 44
Modifier Index	396 % 12 =0	455 % 12=11	456 % 12 = 0	528 % 12 = 0
Letter	ত	ন্	ম	য়

Weight to Word conversion for string [**480,551,372,444**]

Number	**480**	**551**	**372**	**444**
Base Index	**(480 – 0) / 12 =40**	**(551-11) / 12 = 45**	**(372 -0) / 12 = 31**	**(444 – 0) / 12 = 37**
Modifier Index	**480 % 12 = 0**	**551 % 12 = 11**	**372 % 12 = 0**	**444 % 12 = 0**
Letter	ব	ব্র	ণ	ন

7.5 Code Segments

SortAlgorithm.java

```
/*      Implementing the Sort Algorithm      */

import java.io.*;
import java.lang.*;
import java.util.*;
class SortAlgorithm
{       public static void sort()
        {       int i;
                char ch;
                ReadWrite rw = new ReadWrite();
        //Reading from the file containing the document
                String inputString = rw.read("Files\\Tokenized.txt");
        //Tokenizing the strings of the file into words
                StringTokenizer StrToken = new StringTokenizer(inputString);
        // Linked List containing a Word
                LinkedList<Integer> word=new LinkedList<Integer>();
        // Linked List containing a Weight of a Word
                LinkedList<Integer> weight=new LinkedList<Integer>();
        // Linked List containing a Weights of all the Words in the entire file
                LinkedList<String> concatweightlist=new LinkedList<String>();
        // Linked List containing the node where each node
                LinkedList<Node> nodeList = new LinkedList<Node>();
                int cntr=0;
                while(StrToken.hasMoreTokens())
                {
//-------------------------------------------------------------------------------------------
                        String strDocID=new String();
                        String wordString = new String (StrToken.nextToken());
                        String wordStringTrim = new String();
                        for (i=0; i<wordString.length(); i++)
                        {       ch=wordString.charAt(i);
                                if
((ch==48)||(ch==49)||(ch==50)||(ch==51)||(ch==52)||(ch==53)||(ch==54)||(ch==55)||(ch==56)||(ch==5
7))
                                {       strDocID=strDocID+ch;
                                        continue;
                                }
                                wordStringTrim = wordStringTrim + ch;
                        }
                        Node n = new Node(wordStringTrim, strDocID,"");
                        nodeList.add(n);
```

```
                        String firstString=new String((nodeList.get(0).word).toString());
                        if (cntr==0)
                        {       String firstStrTrim = new String();
                                for(i=0; i<firstString.length(); i++)
                                {
                                        if(firstString.codePointAt(i)!=65279)
                                        firstStrTrim = firstStrTrim + firstString.charAt(i);
                                }
                                nodeList.remove(0);
                                Node m = new Node(firstStrTrim,strDocID," ");
                                nodeList.addFirst(m);
                        }
                        System.out.println("Size of Linked LiSt is:-"+nodeList.size());
                        System.out.println("Doc id:”+nodeList.get(cntr).docID).toString()));
                        cntr=cntr+1;

//-------------Formation of the word list where each node consists of Unicode of every
//--------------letter or modifier excluding BOF charecter

                        for (i=0; i<wordStringTrim.length(); i++)
                        {       if(wordStringTrim.codePointAt(i)!=65279)
                                        word.add(wordStringTrim.codePointAt(i));
                        }

//-------------Formation of the weight list where each node consists of weight of every
//-------------letter or modifier derived from their Unicode

                                InitBarnaModifier barnaValue = new InitBarnaModifier();
                        // Creating object of the use defined InitBarnaModifier
                                for(int letter : word)
                                {       int ret = barnaValue.barnaIndex(letter);
                                        if(ret!=-1)
                                                weight.add(12*ret);
                                        else
                                        {       ret  = barnaValue.modifierIndex(letter);
                                        weight.set((weight.size()-1),((int)weight.getLast()+ret));
                                        }
                                }
                                int length;

//----------------------------Padding an extra 0 to the weight if it 2 digit----------------
                                String concatWeight = new String ();
                                for(int digit : weight)
                                {
                                        if(digit<=99)
                        //concatenate the weight string to get a integer string else
                                                concatWeight=concatWeight+"0" +digit;
                                                concatWeight=concatWeight+digit;
                                }
                                System.out.println(weight);     //Printing the Weight String
                                System.out.println(concatWeight);
                                word.clear();
                                weight.clear();
                                concatweightlist.add(concatWeight);   //adding to another linked list
        }

                System.out.println(concatweightlist);
                int addZero, max=0;
                String element=new String();
        // Calculating the max length of elements of the concatWeigthList
                for(i=0; i<concatweightlist.size(); i++)
                {       element=concatweightlist.get(i);
                        if(max<element.length())
                                max=element.length();
                }

                System.out.println("Max: " + max);
        // Adding 0s to the required weights
                for(i=0; i<concatweightlist.size(); i++)
                {       element=concatweightlist.get(i);
                        if(element.length()<max)
```

```
                        {       addZero=(max-element.length());
                                System.out.println(i+"th elements need " + addZero + "zero");
                                for(int j=0; j<addZero; j++)
                                        element=element+"0";
                                concatweightlist.set(i, element);
                        }
                }

                System.out.println("concat weight list is"+concatweightlist);
                        RadixSort radix = new RadixSort();    // Calling the Radix Sort method
                LinkedList<String> concatWeightListGet = new
                LinkedList<String>(radix.sort(concatweightlist, max));
                System.out.println("Concat weight list get"+concatWeightListGet);
                String wordStr = new String();
                WeightToWord w=new WeightToWord();

//      Getting the word back from the weight
                String getDoc=new String();
                String strBuf = new String();
                for (i=0; i<concatWeightListGet.size(); i++)
                {       element = concatWeightListGet.get(i);
                        wordStr = w.convert(element);
                        System.out.println(element + ";" + wordStr);
                        if (wordStr.equals(strBuf))
                                continue;
                        for (int j=0; j<nodeList.size();j++)
                                {
                                if(((nodeList.get(j).word).toString()).equals(wordStr))
                                {       getDoc=((nodeList.get(j).docID).toString());
                                        rw.append(getDoc+wordStr+"\r\n", "Files\\Sorted.txt");
                                }

                                }
                        strBuf = wordStr;
                }
        }
}
```

ReadWrite.java

```
/*      Reading, Writing and Appending to File              */

import java.io.*;
import java.util.*;
import java.lang.*;
class ReadWrite
{
//      Appending to the file
        static void append(String str, String path)
        {        try
                 {      CharSequence seq=new String(str);
                        FileOutputStream fos = new FileOutputStream(path,true);
                        Writer out = new OutputStreamWriter(fos, "UTF8");
                        out.append(seq);
                        out.close();
                }
                catch (IOException e)
                {e.printStackTrace();}
        }


//      Writing to the file

        /*static void write(String str, String path)
        {        try
                 {      FileOutputStream fos = new FileOutputStream(path,true);
                        Writer out = new OutputStreamWriter(fos, "UTF8");
                        out.write(str);
                        out.close();
                }
```

```
                catch (IOException e)
                {e.printStackTrace();}
        }*/

//      Reading from the file

        static String read(String path)
        {       StringBuffer buffer = new StringBuffer();
                 try
                {       FileInputStream fis = new FileInputStream(path);
                        InputStreamReader isr = new InputStreamReader(fis,"UTF8");
                        Reader in = new BufferedReader(isr);
                        int ch;
                        while ((ch = in.read()) > -1)
                        {
                        buffer.append((char)ch);
                        }
                        in.close();
                        return buffer.toString();
                 }
                catch (IOException e)
                {       e.printStackTrace();
                        return null;
                }
        }
}
```

Node.java

```
/*      Each node of the list used in the SortAlgorithm consists of two parameters */

class Node
{       String word = new String();
        String docID = new String();
        String wordFreq = new String();
        Node(String a, String b, String c)
        {       word = a;
                docID = b;
                wordFreq = c;
        }
}
```

InitBarnaModifier.java

```
/*      Initializing Barna and Modifier Linked Linked       */

import java.io.*;
import java.util.*;
import java.lang.*;
class InitBarnaModifier
{
//      Returning the index of the Barna List
        static int barnaIndex(int uniLetter)
        {       int i;
                ArrayList<Integer> Barna = new ArrayList<Integer>();
                Barna.add(-1);
                for (i=2437;i<=2452;i++)
                {       if ((i>=2444 && i<=2446) || (i>=2449 && i<=2450))
                                continue;
                        Barna.add(i);
                }
                Barna.add(2434);
                Barna.add(2435);
                Barna.add(2433);
                for (i=2453;i<=2465;i++)
                        Barna.add(i);
                Barna.add(2524);
                Barna.add(2466);
                Barna.add(2525);
```

```
                Barna.add(2467);
                Barna.add(2510);
                for(i=2468;i<=2479;i++)
                {       if ((i==2473))
                                continue;
                        Barna.add(i);
                }
                Barna.add(2527);
                for(i=2480;i<=2489;i++)
                {       if((i==2481) || (i==2483) || (i==2484)|| (i==2485))
                                continue;
                        Barna.add(i);
                }
                return Barna.indexOf(uniLetter);
        }

//      Returning the index of the Modifier List
        static int modifierIndex(int uniLetter)
        {       int i;
                ArrayList<Integer> Modifier = new ArrayList<Integer>();
                Modifier.add(-1);
                for (i=2494;i<=2509;i++)
                {       if ((i>=2500 && i<=2502) || (i>=2505 && i<=2506))
                                continue;
                        Modifier.add(i);
                }
                return (Modifier.indexOf(uniLetter));
        }

//      Returning the Bangla Base Letter
        static String barnaVal(int index)
        {       int i;
                ArrayList<Integer> Barna = new ArrayList<Integer>();
                Barna.add(-1);
                for (i=2437;i<=2452;i++)
                {       if ((i>=2444 && i<=2446) || (i>=2449 && i<=2450))
                                continue;
                        Barna.add(i);
                }
                Barna.add(2434);
                Barna.add(2435);
                Barna.add(2433);

                for (i=2453;i<=2465;i++)
                        Barna.add(i);
                Barna.add(2524);
                Barna.add(2466);
                Barna.add(2525);
                Barna.add(2467);
                Barna.add(2510);
                for(i=2468;i<=2479;i++)
                {       if ((i==2473))
                                continue;
                        Barna.add(i);
                }
                Barna.add(2527);
                for(i=2480;i<=2489;i++)
                {       if((i==2481) || (i==2483) || (i==2484)|| (i==2485))
                                continue;
                        Barna.add(i);
                }

                char c[ ] = Character.toChars(Integer.parseInt(Barna.get(index).toString()));
                String str1=new String(c);
                return str1;
        }

//      Returning the Bangla Modifier

        static String modifierVal(int index)
        {       int i;
```

```
                ArrayList<Integer> Modifier = new ArrayList<Integer>();
                Modifier.add(-1);
                for (i=2494;i<=2509;i++)
                {       if ((i>=2500 && i<=2502) || (i>=2505 && i<=2506))
                                continue;
                        Modifier.add(i);
                }
                char c[ ] = Character.toChars(Integer.parseInt(Modifier.get(index).toString()));
                String str2=new String(c);
                return str2;
        }
}
```

RadixSort.java

```
/*      Implementing Radix Sort       */

import java.io.*;
import java.lang.*;
import java.util.*;
class RadixSort
{        static LinkedList<String> arq0=new LinkedList<String>();
        //Declaration of queues for each packet
         static LinkedList<String> arq1=new LinkedList<String>();
         static LinkedList<String> arq2=new LinkedList<String>();
         static LinkedList<String> arq3=new LinkedList<String>();
         static LinkedList<String> arq4=new LinkedList<String>();
         static LinkedList<String> arq5=new LinkedList<String>();
         static LinkedList<String> arq6=new LinkedList<String>();
         static LinkedList<String> arq7=new LinkedList<String>();
         static LinkedList<String> arq8=new LinkedList<String>();
         static LinkedList<String> arq9=new LinkedList<String>();
        //changeList consists the partial sorted list after each pass.
         static LinkedList<String> changeList=new LinkedList<String>();
         static LinkedList arr[ ]={arq0,arq1,arq2,arq3,arq4,arq5,arq6,arq7,arq8,arq9};
        //Array consisting the packet queues
         static LinkedList<String> concatWeightList=new LinkedList<String>();
        //      Each packet queue being populated
         static void putToList(char ch , String strPut)
         {      int i;
                switch(ch)
                {       case '0':    arq0.add(strPut);
                                     break;
                        case '1':    arq1.add(strPut);
                                     break;
                        case '2':      arq2.add(strPut);
                                     break;
                        case '3':    arq3.add(strPut);
                                     break;
                        case '4':    arq4.add(strPut);
                                     break;
                        case '5':    arq5.add(strPut);
                                     break;
                        case '6':    arq6.add(strPut);
                                     break;
                        case '7':    arq7.add(strPut);
                                     break;
                        case '8':    arq8.add(strPut);
                                     break;
                        case '9':    arq9.add(strPut);
                                     break;
                        default:       System.out.println("WRONG ENTRY");
                                     break;
                }
        }

//      Packet Queues being populated
         static void listing(int cntr)
        {       for(int a=0; a<concatWeightList.size(); a++)
                {       String str=new String(concatWeightList.get(a));
```

```
                        char c=str.charAt(str.length()-1-cntr);
                        putToList(c, str);
                }
        }

//      One Pass
        static void  passing(int itr1)
        {       changeList.clear();
                for(int digit=0; digit<=9; digit++)
                {       if(arr[digit].isEmpty())
                                continue;
                        else
                        {       for(int i=0; i<arr[digit].size(); i++)
                                {changeList.add((arr[digit].get(i)).toString());}}}
                concatWeightList = changeList;
                for (int digit=0; digit<=9; digit++)
                        arr[digit].clear();
        }

//      The main sort method to be called from the SortAlgoritm program.
        public static LinkedList<String> sort(LinkedList<String> list, int n)
        {       for (int i=0; i<list.size(); i++)
                        concatWeightList.add(list.get(i));

                for(int itr=0; itr<n; itr++)
                {       listing(itr);
                        passing(itr);
                }

                System.out.println("The sorted weights are :" + concatWeightList);
                return concatWeightList;
        }
}
```

WeightToWord.java

```
/*      Conversion of weight to word         */

import java.io.*;
import java.util.*;
import java.lang.*;
class WeightToWord
{       static String convert(String elementGet)
        {       int intValue, baseIndex, modifierIndex;
                String str = new String();
                InitBarnaModifier barnaValue = new InitBarnaModifier();
                InitBarnaModifier modifierValue = new InitBarnaModifier();
                for (int i=0; i<elementGet.length(); i=i+3)
                {       String intLetter = new String(elementGet.substring(i, i+3));
        //getting a 3 digit no from string
                        intValue  = Integer.parseInt(intLetter);     //conversion to int
                        if(intValue==0)
                        continue;
                        modifierIndex = intValue%12;
                        if(modifierIndex!=0)          //condn. for modified barna
                        {
                                baseIndex = ( intValue - modifierIndex)/12;
                                str = str + barnaValue.barnaVal(baseIndex) + …
                                        … modifierValue.modifierVal(modifierIndex);
        }
                        else           //condn. for only barna
                        {       baseIndex=intValue/12;
                                str = str + barnaValue.barnaVal(baseIndex);
                        }
                }
                return str;
        }
}
```

Chapter 8

Clustering

8.1 Introduction

Clustering is used to cluster words into homogeneous groups. Each group is expected to represent an equivalence class consisting of morphological variants of a single root word. The words within a cluster are stemmed to the "central" word or "root word" in that cluster. Since the number of natural clusters is unknown a priori, partitional clustering algorithms like 'k-means' are not suitable for our task. 'Graph-theoretic' clustering algorithms appear to be the natural choice in this situation because of their ability to detect natural clusters in the data. Here we are using "LEXICON CLUSTERING".

8.2 Types of Clustering

As we are considering Graph-Theoretic clustering, there are three types of clustering:

1. Single Linkage Method: Maximum similarity between any member of one group with any member of other group defines the similarity between two groups .

2. Average Linkage Method: Mean similarity Mean similarity between point in one cluster and those of the other. In contrast to single linkage, each element needs to be relatively similar to all members of the other cluster, rather than to just one.

3. Complete Linkage Method: Minimum similarity between any member of one cluster and any member of other cluster defines the similarity of two clusters.

Here we have used Complete Linkage Clustering.

Threshold Value: Choice of threshold is very important in this approach. Because if a high threshold is chosen, we get an aggressive stemmer which forms larger-than-actual stem classes, where semantically unrelated forms are conflated erroneously. If the chosen threshold is too low, we get a lenient stemmer which fails to conflate related forms that should be grouped together. We have taken 0.55 as threshold value for Bengali Stemming as per the suggestion proposed in YASS paper [4].

8.3 Algorithm

8.3.1 Algorithm for Clustering

1. Repeat from Step 2 to Step 7 for every token of the Sorted.txt.
2. Retrieve a token from Sorted.txt.
3. Repeat Step 3 and Step 4 for every letter of that token.
4. If letter is a digit concatenate with strDocID and continue .
5. Put every other character to a string wordStringTrim.
6. 6.1 If wordStringTrim is any Stem word(or root word)
    6.1.1 Increment term frequency.
    6.1.2 Increment counter.
    6.1.3 Assign 'new line'(\r\n) as escape sequence.
    6.2 Otherwise
        6.2.1 Increment term frequency.
        6.2.2 Increment counter.
        6.2.3 Assign 'tab' (\t)as escape sequence.
        6.2.4 If counter > 0
            6.2.4.1 Write escape sequence, wordStringTrim, termFrequency, DocID to 'Clustered.txt' in append mode.
7. Calculate D3 value by comparing Stem word and WordStringTrim.
8. 8.1 If D3 <= 0.55
    8.1.1 Take WordStringTrim word as Stem word for the next iteration.
    8.2 Otherwise
        8.2.1 Take WordStringTrim as nonStem word for next iteration.

8.3.2 Algorithm for Calculating D3

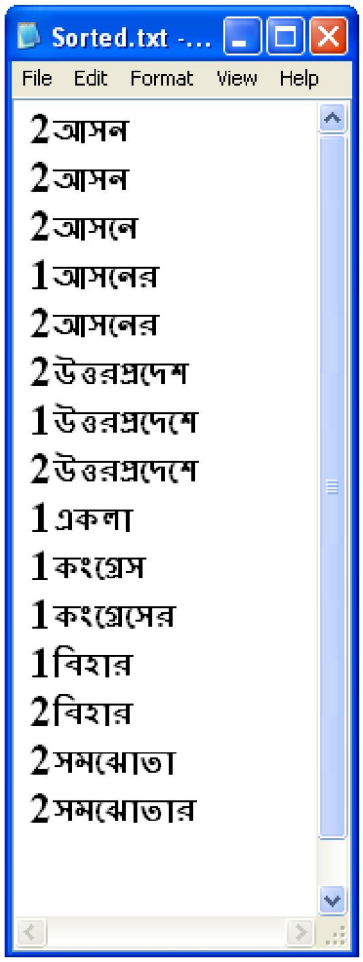
1. Take two word as word1 and word2.
2. Convert those word to their Unicode representation.
3. 3.1 Determine the maximum length among word1 & word2.
    3.2 Assign maximum length to variable n.
4. Append zeroes to the end of the shorter word's Unicode representation.
5. 5.1 Find out the collision index,index of both the Strings where the corresponding character of both the string are dissimilar.
    (5.2) Assign the index of collision to a variable m.
6. Calculate D3 by the following formula -

$$D_3(X,Y) = \frac{n-m+1}{m} \times \sum_{i=m}^{n} \frac{1}{2^{i-m}} \quad \text{if } m > 0, \quad \infty \text{ otherwise}$$

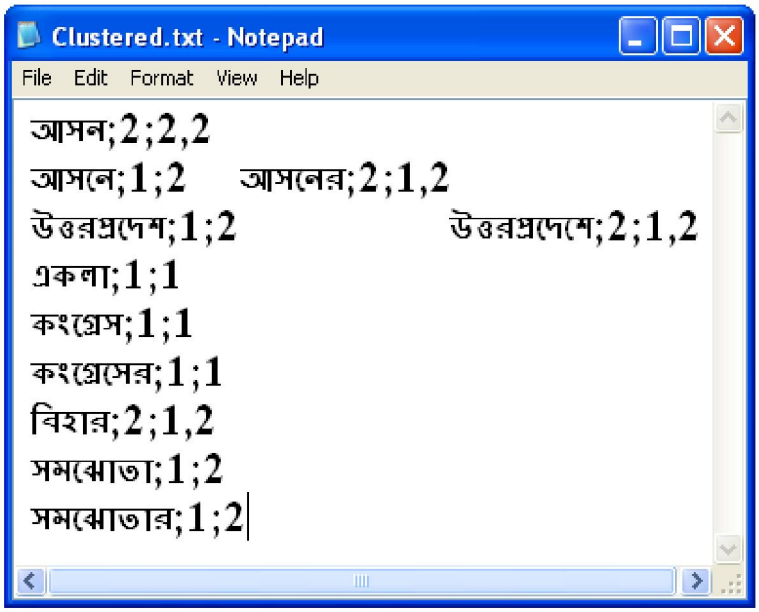
7. Return D3.

8.4 Illustration

Lets take an example of a sorted text file which contain the word including it's document ID. After the Clustering phase it will show all the Stem word & non-stem word in a clustered fashion for stem word.



Before Clustering



After Clustering

Now lets look at how the D3 for different pairs of word are calculated.

We are taking here only two Bengali word as an example - উত্তরপ্রদেশ and উত্তরপ্রদেশে . The modifier will be at the right hand of the base letter to which it has been attached. So we need to represent these two words into our required format.

Bengali Letter	উ	ত	◌ৃ	ত	র	প	◌ৃ	র	দ	(◌	শ	X
Unicode	2441	2468	2509	2468	2480	2474	2509	2480	2470	2503	2486	0000
Bengali Letter	উ	ত	◌ৃ	ত	র	প	◌ৃ	র	দ	(◌	শ	(◌
Unicode	2441	2468	2509	2468	2480	2474	2509	2480	2470	2503	2486	2503
	0	1	2	3	4	5	6	7	8	9	10	11

Collision point

So n = Length of the longer String = 12. Collision index m = 11

$$D_3(X,Y) = \frac{n-m+1}{m} \times \sum_{i=m}^{n} \frac{1}{2^{i-m}} \quad \text{if } m > 0, \quad \infty \text{ otherwise}$$

……..[4]

$$D3 = \frac{12-11+1}{11} \times \left(\frac{1}{2^0} + \frac{1}{2^1} \right) = 0.272$$

As the value of D3 is less than our threshold value 0.55 [4] so the word উত্তরপ্রদেশে can be stemmed to the root word or stem word উত্তরপ্রদেশ.

8.5 Code Segment

Clustering.java

```
/*      Clustering the Sorted File    */

import java.io.*;
import java.lang.*;
import java.util.*;
import java.math.*;
class Clustering
{       public static void cluster()
        {       double d3;
                int i,j;
                String inputStringTrim=new String();
                CalculateD3 a = new CalculateD3();
                ReadWrite rw  = new ReadWrite();
                String inputString = rw.read("Files\\Sorted.txt");
                if(inputString.codePointAt(0)==65279)
                        inputStringTrim=inputString.substring(1, inputString.length());
                char ch;
                StringTokenizer StrToken = new StringTokenizer(inputStringTrim);
                String strTok=new String(" ");
                String strStem =new String();
                String strNonStem = new String();
                int termFreq = 0;
                String docIDStream = new String();
                String wordStringTrimBuf = new String();
                String docIDStreamBuf = new String();
```

```
                int count=0;
                String escpSeq = new String();
                while(StrToken.hasMoreTokens())
                {
                        strTok=StrToken.nextToken();
                        String strDocID = new String();
                        String wordStringTrim = new String();
                        for (i=0; i<strTok.length(); i++)
                        {       ch=strTok.charAt(i);
                                if
((ch==48)||(ch==49)||(ch==50)||(ch==51)||(ch==52)||(ch==53)||(ch==54)||(ch==55)||(ch==56)||(ch
==57))
                                {       strDocID=strDocID+ch;
                                        continue;
                                }
                                wordStringTrim = wordStringTrim + ch;
                        }

//-------------------------If the Stem or the Non-stem word appears more than 1 time---------
--
                        if (strStem.equals(wordStringTrim) ||
strNonStem.equals(wordStringTrim))
                        {       termFreq++;
                                docIDStream = docIDStream + "," + strDocID;
                                wordStringTrimBuf = wordStringTrim;
                                count++;
                                if (strStem.equals(wordStringTrim))
                                        escpSeq="\r\n";
                                else
                                        escpSeq="\t";

                                continue;
                        }
                        else
                        {       if(count>0)
                                {       rw.append(escpSeq  + wordStringTrimBuf + ";" + termFreq
…                                               + …";" + docIDStream,
"Files\\Clustered.txt");
                                        count = 0;
                                }
                                termFreq = 1;
                                docIDStream = strDocID;
                                wordStringTrimBuf = wordStringTrim;
                                count++;
                                escpSeq="\r\n";
                        }

//---------------------------Checking for D3-----------------------------------------------
--
                        d3=a.calculate(strStem,wordStringTrim);
                        if(d3<=0.55)
                                strNonStem = wordStringTrim;

                        else
                                strStem = wordStringTrim;
                }
        }
}
```

CalculateD3.java

```
/*      Calculation of D3             */

import java.io.*;
import java.lang.*;
import java.util.*;
import java.math.*;
class CalculateD3
{       static double calculate(String wordStr1, String wordStr2)
        {       int i, j, m=-1;
                double D3;
```

```
                LinkedList<Integer> word1 = new LinkedList<Integer>();
                LinkedList<Integer> word2 = new LinkedList<Integer>();

//Formation of the word list 1 where each node consists of Unicode of every letter or modifier
                for (i=0; i<wordStr1.length(); i++)
                {       if(wordStr1.codePointAt(i)!=65279)
                                word1.add(wordStr1.codePointAt(i));
                }
                System.out.println(word1);

//Formation of the word list 2 where each node consists of Unicode of every letter or modifier
                for (i=0; i<wordStr2.length(); i++)
                {       if(wordStr2.codePointAt(i)!=65279)
                                word2.add(wordStr2.codePointAt(i));
                }
                System.out.println(word2);
                int n = (word1.size() > word2.size()) ? word1.size() : word2.size();
        //Finding the size of the larger word
                System.out.println("n = " + n);

        //      Padding 0 to the end of the shorter word
                int x=0;
                if(word1.size()>word2.size())
                        word2.addLast(x);
                else if(word2.size()>word1.size())
                        word1.addLast(x);
                System.out.println(word1);
                System.out.println(word2);
                System.out.println("Min size=" + Math.min(word1.size(),word2.size()) );

//      Finding the collision index m
                for(i = 0; i <Math.min(word1.size(),word2.size()); i++)
                {
if((Integer.parseInt(word1.get(i).toString()))!=(Integer.parseInt(word2.get(i).toString())))
                        {       m = i;
                                break;
                        }
                }
                System.out.println("m ="+m);
                double frac = 0.0;
                for (i = 0; i <= (n - m); i++)
                {
                        frac = frac + (1/(Math.pow(2.0,(double)i)));
                }
                System.out.println("Frac = " + frac);
                System.out.println(((double)n-(double)m+1.0)/(double)m);
                if (m == 0)
                        D3=9999;
                else if(m==-1)
                        D3=0.0;
                else
                        D3 = (((double)n-(double)m+1.0)/(double)m) * frac;
                System.out.println("D3 = "+D3);
                if(D3<=0.55)
                        System.out.println("Stemming is possible");
                else
                        System.out.println("Stemming is not possible");

                return D3;
        }
}
```

Chapter 9

Retrieval

9.1 Introduction

After forming the clustered text file our next aim is to retrieve the Stem-word & the Document ID of a particular Bengali word provided by the user. For simplicity we have designed it in such a way that user has to write the key word (the word which is to be searched) in a text file WordInput.txt. After reading that key word from WordInput.txt we will search Clustered text file sequentially to get that word and when we will find out the key word we will next find out the Stem-word, Line No. & Document No. (the No. of the document from which the word has been clustered during Multi Document Clustering) by the algorithm discuss below.

9.2 Algorithm

1. Read the Key word from WordInput.txt.
2. Tokenize the Clustered.txt file
3. Read every token for finding out the key word & for every token repeat from Step 4 to Step 7.
4. Until **;** (semicolon) not found put all other character into a string strWord.
    4.1 When character **;** (semicolon) found then Check for equality between string strWord and Key word & if true-
        4.1.1 Get Line No.
        4.1.2 Read the current line into a String lineRead.

5 . Read the string lineRead until a **;** (semicolon) found & put all the character into a string stem Word which represent the Stem-Word.

6. Break the lineRead string into token and for every token perform -
    6.1 Read the token from reverse until a ; (semicolon) has been reached.
    6.2 Iterate from the next index of the index of ; (semicolon) to the last of the token.
        6.2.1 Until a **,**(comma) character has been reached read all the character into a string str.
        6.2.2 When a **,**(comma) found add the string str to the LinkList StrDocIDStream and initialize the string with null.
7. Remove the redundancy in the list StrDocIDStream .

9.3 Illustration

For a word আসনের , written in WordInput.txt we will find that line containing Document ID & Stem-Word from the Clustered.txt file

আসনে;1;2 আসনের;2;1,2 having Line no 2.Now after tokenization we will get

আসনে;1;2 and আসনের;2;1,2 . So Stem-Word can be obtain by reading from beginning of line to the first occurrence of ; (semicolon).That is Stem-Word is আসনে. Now to determine the Document Id we are taking token আসনের;2;1,2 as an example.

Reading from last to the first occurrence of ; (semicolon) we will get 1,2. Now removing , (comma) and putting into the lest we get the list containing **[2 1 2].**As there is redundancy in the list we need to remove the redundancy & we get **[2 1]**.

So the final outcome is that the word আসনের has the Stem-word আসনে, the word is in Line No. 2 of the clustered file and the word including its other Non-Stem-word are coming from document 1&2.

9.4 Code Segments

Retrival.java

```
/*      Retrieving the Result from the Clustered File       */

import java.io.*;
import java.lang.*;
import java.util.*;
import java.math.*;

class Retrieval
{       static LinkedList<String> strDocIDStream = new LinkedList<String>();
        static LinkedList<String> netDocIDStream = new LinkedList<String>();
        static boolean check(String str)
        {       for (int i=0; i<netDocIDStream.size(); i++)
                {       if (netDocIDStream.get(i).toString().equals(str))
                                return true;
                }
                return false;
        }

        void retrieve() throws FileNotFoundException, IOException,
UnsupportedEncodingException
        {       FileInputStream fis = new FileInputStream("Files\\Clustered.txt");
                InputStreamReader isr = new InputStreamReader(fis,"UTF8");
                Reader in = new BufferedReader(isr);
                LineNumberReader ln=new LineNumberReader(in);
                ReadWrite rw= new ReadWrite();
                String ipWord = rw.read("Files\\WordInput.txt");
                String inputWord = ipWord.substring(1,ipWord.length());
                String lineRead = new String();
                int line=0,ch;
```

```
		String inputString = new String();

//---------------Finding the word from the Clustered file-----------------------------------

		for(ch=ln.read(); ch!=-1; ch=ln.read())
		{	if (ch>=2433 && ch<=2531)
				inputString = inputString + (char)ch;
			if(ch==35)
			{	if(inputString.equals(inputWord))
				{	line = ln.getLineNumber();
					System.out.println("Line number is : " + line);
					System.out.println("YES");
				}
				inputString="";
			}
		}
		in.close();

//---------------Reading the entire line from the Clustered file-----------------------------

		FileInputStream fis1 = new FileInputStream("Files\\Clustered.txt");
		InputStreamReader isr1 = new InputStreamReader(fis1,"UTF8");
		Reader in1 = new BufferedReader(isr1);
		LineNumberReader ln1=new LineNumberReader(in1);
		for (int i=0; i<=line; i++)
			lineRead = ln1.readLine();

		rw.append("\r\n"+lineRead,"Files\\TestOutput.txt");
		System.out.println(lineRead);
		in1.close();

//---------------Reading the Stem Word from the Clustered file-------------------------------

		int i;
		String stemWord = new String();
		for (i = 0; lineRead.codePointAt(i) != 59 ; i++)
			stemWord = stemWord + lineRead.charAt(i);

		System.out.println("The Stem Word is : " + stemWord);
		rw.append("\r\n"+stemWord,"Files\\TestOutput.txt");

//---------------Inserting the DocIDs into the strDocIDStream List---------------------------

		StringTokenizer lineReadToken = new StringTokenizer(lineRead);
		String str = new String();
		while(lineReadToken.hasMoreTokens())
		{	String lineReadWord = new String(lineReadToken.nextToken());

			for(i=lineReadWord.length()-1; lineReadWord.charAt(i)!=59; i--)
			{}

			for (int j = i+1; j<lineReadWord.length(); j++)
			{	if
((lineReadWord.codePointAt(j)==44)||(lineReadWord.codePointAt(j)==35))
				{	strDocIDStream.add(str);
					str="";
					continue;
				}
				str = str + lineReadWord.charAt(j);
			}
		}
		System.out.println(strDocIDStream);

//---------------Creating the netDocIDStream List with unique DocIDs-------------------------

		for (i=0; i<strDocIDStream.size(); i++)
		{	if (!check(strDocIDStream.get(i).toString()))
				netDocIDStream.add(strDocIDStream.get(i).toString());
		}
		System.out.println("Total Results: " + netDocIDStream.size());
		System.out.println(netDocIDStream);
	}
}
```

Chapter 10

Conclusion

10.1 Usage

- Ø For developing a Bengali Search Engine we need a Bengali Stemmer as a part of that project.
- Ø Stemmers are used to conflate terms to improve retrieval effectiveness and /or to reduce the size of indexing file.
- Ø Stemming will increase recall at the cost of decreased precision.
- Ø Stemming can have marked effect on the size of indexing files, sometimes decreasing the size of file as much as 50 percent.

10.2 Limitation

- Ø Stemming will decrease precision. We need to reduce that decrease rate.
- Ø As we have implemented YASS-Yet Another Suffix Stripper which can be implemented for any other languages also, determining the "Threshold Value" for other language is difficult & always not accurate due to the Corpus Based Approach.
- Ø We have not performed Stemming for all types of text documents like **.**doc, **.**html, etc. We have only worked on **.**txt & **.**pdf files.
- Ø Bengali is very resource poor language for developing new technology related to IR.

10.3 Future Work

- Ø We will try to optimized the Threshold Value (D3=0.55).
- Ø We will try to use all types of text file (**.**doc, **.**html etc.) as the resource file of Stemming.
- Ø We will try to perform the next stage of 'Stemming', which is "Phrase Recognition".
- Ø Finally we will try to develop a "Bengali Search Engine".

Bibliography

[1] Amit Singhal, Modern Information Retrieval: A Brief Overview, IEEE, 2001

[2] C. D. Manning, P. Raghavan, H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, ISBN 978-0-521-86571-5, May 27, 2008.

[3] Md. Zahurul Islam, Md. Nizam Uddin, Mumit Khan, *A Light Weight Stemmer for Bengali and Its Use in Spelling Checker,* BRAC University, Dhaka, Bangladesh

[4] P. Majumder, M. Mitra, S.K. Parui, G. Kole, P. Mitra and K. Dutta, *YASS: Yet another suffix stripper, ACM Transactions on Information Systems,* Vol. 25, No. 4, pp. 18-38, October 2007.

[5] Md. Sharif Uddin, Rahat Khan, A.B.M. Tariqul Islam, S.M. Rafizul Haque, *A New Approach in Computer Representation of Bangla Words and Bangla Sorting Algorithm,* NCCPB 2005, Independent University, Bangladesh.

[6] Md. Ahsanur Rahman, Md. Abdus Sattar, *A New Approach to Sort Unicode Bengali Text*, 5th International Conference on Electrical and Computer Engineering ICECE 2008, 20-22 December 2008, Dhaka, Bangladesh.

[7] The Unicode Consortium, http://www.unicode.org

[8] Bangla Academy, Dhaka, Bangladesh, http://www.banglaacademy.org

[9] Paschimbanga Bangla Academy, West Bengal, India, http://www.paschimbangabanglaakademi.org

[10] Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms*, 2nd Edition.

[11] Shildt, *The Complete Reference JAVA*, 7th Edition, ISBN 13:978-0-07-063677-4

[12] M. Mitra, P. Majumder, L. Sobha, *Indian Language Information Retrieval: dealing with Indian language text retrieval issues*, Tutorial, ACM SIGIR 2008.

[13] The Omnicron Lab, http://www.omnicronlab.com

[14] Sun Java Documentation, http://java.sun.com

[15] Wikipedia, http://www.wikipedia.org

[16] Google Search Engine, http://www.google.com