New Mexico Supercomputer Challenge 2007-2008

Team 15

Cryptography Studies

Final Report
04 April 2008

Dhaivat Pandya, Colin Redman, and Matthew Ticknor

Aspen Elementary School, Los Alamos NM

Sponsoring Teacher: Zeynep Unal
Mentor: Elizabeth Cooper

Table of Contents

Illustration Index

Executive Summary

We wanted to write Java programs to test encryption and decryption. We decided to work on an encryption/decryption project using the Caesar Cipher [1]. The challenge was to use the computer to decide which decryption shift was the correct one by brute force testing all the possible shifts for real English words.

We found and modified an open source (GPL) Java Caesar Cipher library [2] to use to encrypt our plain text input. Then we wrote the encryption and decryption programs using this library.

The challenge for this project was to use the computer to try all possible shifts and analyze the outputs for sequences of English words. The decryption program generates a report on which shift was the best based on how many English words were found. We found an open source Java spelling checker library (Jazzy) [3] to use for this project but it took us a long time to get this code to work to test the decryption. This was one of the hardest parts of our project.

We also used the spelling checker to separate the decrypted text into words. The input had all spaces removed and was uppercased so there were no word breaks. Separating the text back into words (adding the spaces back in) was also a hard part of the project.

We tested different types of text, like “Robin Hood” and “Alice in Wonderland” from the Project Gutenberg website [4] and articles from Wikipedia such as “Algebra”and “Java”.

We achieved our goal of using the spelling checker to figure out which shift was the correct one for the decryption.

Introduction

Cryptography is the practice and study of hiding information[5]. We decided to do a project on encryption with brute force decryption. Our goal was to use the computer to decide when we had a correct decryption (instead of looking at each possibility with our eyes and deciding when we had a correct decryption). The way we decided to use the computer to do this was to use a spelling checker to test each decryption and see which decrypted output had the most English words.

We did research on different types of simple encryptions and decided to use the Caesar Cipher for testing. We thought that maybe if we finished this type of encryption we could try some more complicated ones.

The Caesar Cipher is a simple shift cipher, and is a type of "substitution cipher"[6]. For example, if there is a shift of 3 and you want to write the word "AXE" the result would be "DAH".

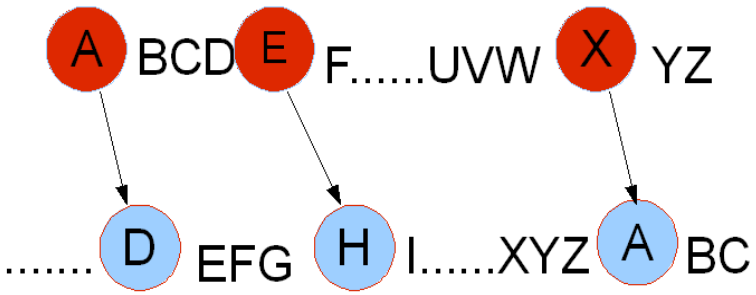

Illustration 1: Example of a Caesar Shift of Three for AXE -> DAH

Problem Statement

We wanted to use Java to write programs to test encryption and decryption. The challenge was to use the computer to decide which decryption shift was the correct one by brute force testing all the possible shifted texts for real English words. For each encrypted input the computer has 25 possible shifts (one for each letter of the alphabet, except a shift of 0) to test. The test shift with the highest percent of English words would be chosen as the correct decryption. The question for this project is, would the computer be able to decide which Caesar Cipher decryption shift was correct using an English spelling checker?

Solving The Problem – the Computational Model

Caesar Code

We started with open source code for Caesar Library written by Valerio Capozio [2]. This was referenced in the Wikipedia article on the Caesar Cipher [1] and was found at http://wwwangelusworldcom/softwarephp?num=7. We modified the code so that it was more useful for our project.

The modifications were:

- Removing whitespace (spaces, tabs, etc.) from the input and uppercasing the input.
- Separating the encryption and decryption parts of the program. The Encryption Program (EncryptFile) was separated from the Decryption Program (DecryptFile)
- Using a random (unknown) shift between 1 and 25 in the EncyptFile program to generate the test input for the decryption program (DecryptFile).
- Not encrypting numbers. Numbers were ignored in the inputs. Caesar was a Roman emperor who did not use Arabic numbers like we do in English. His numbers were represented by Roman Numerals which would be encryptable since they are the letters X, V, I, and L. Arabic numbers do not encypt using a simple shift cipher such as the Caesar Cipher.

Encryption

The encryption program (EncryptFile - see Appendix A for the code) uses any plain text input.

It starts off by uppercasing the text then removing all whitespace. Then it generates one random shift between 1 and 25, inclusive. It applies this shift uniformly to the entire text using the Caesar encryption and saves the file.

We took out the spaces from the text before encrypting it because the caesar cipher normally does not contain spaces, because it would be too easy for a human to guess short words (like I and a). We uppercased the text because if we went with both uppercase and lowercase we would have double the possibilities and that would make it unnessacarily complicated and tedious.

We tested inputs of various sizes by character count and text type. We used technical articles from Wikipedia and literature passages from the on-line Project Gutenberg. We ran several tests and test inputs ranged from 550 to 1057 characters.

As an example, we used “Robin Hood” from project Gutenberg [7]:

IN MERRY ENGLAND in the time of old, when good King Henry the Second ruled the land, there lived within the green glades of Sherwood Forest, near Nottingham Town, a famous outlaw whose name was Robin Hood. No archer ever lived that could speed a gray goose shaft with such skill and cunning as his, nor were there ever such yeomen as the sevenscore merry men that roamed with him through the greenwood shades. Right merrily they dwelled within the depths of Sherwood Forest, suffering neither care nor want, but passing the time in merry games of archery or bouts of cudgel play, living upon the King's venison, washed down with draughts of ale of October brewing.

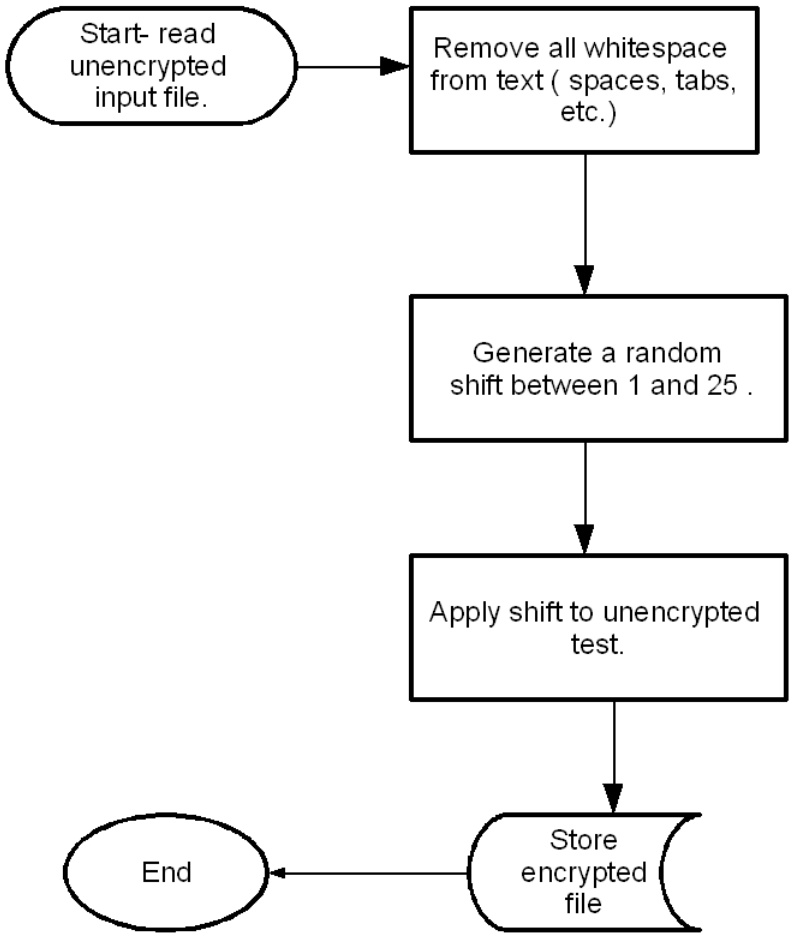

Illustration 2: Encryption (EncyptFile) Flow Chart

Decryption

First, the decryption program reads the encrypted text. Next, the program tests all the 25 decryption shifts, one at a time. A “shift test” is a test for a specific shift.

Every shift test is sent as a string to a method called “analzyeWithJazzy” which looks carefully for English words, one after another. When an English word is found, we look further for a longer word that starts with this word (“checkForCompoundWord”), searching for up to 30 characters following this starting word. For example, if the word “for” is found, and this is followed by “est” the result will be “forest”, not just the first part (“for”).

After the longest word is found, we add this word to the output, adding a space between what we found before and this new word to form the whole output built up of these individual words.

The way the spelling checker worked, we did not get returned the words that never matched in the spelling checker library. So if the original input was:

THERELIVEDWITHINTHEGREENGLADESOFSHERWOODFOREST

we would get back:

THERE LIVED WITHIN THE GREEN GLADES OF SHE WOOD FOREST

which left out the “R” in SHERWOOD FOREST (“SHE WOOD FOREST”)

So we sent the output back to get these missing character sequences that were not in the dictionary. After that our output looked like:

THERE LIVED WITHIN THE GREEN GLADES OF SHE R WOOD FOREST

The correct word “SHERWOOD” would never be better than “SHE R WOOD” because the spelling checker does not know SHERWOOD as a word but instead found the two short words SHE and WOOD, rejecting the non-word “R”.

As all the shifts are analyzed, the program keeps track of the shift that had the most English words found. In the end, after all 25 shifts are tested, a report is generated to summarize which shift was the best and to record the original input text (encrypted), the decrypted text with spaces, and the final output with the rejected characters added back in.

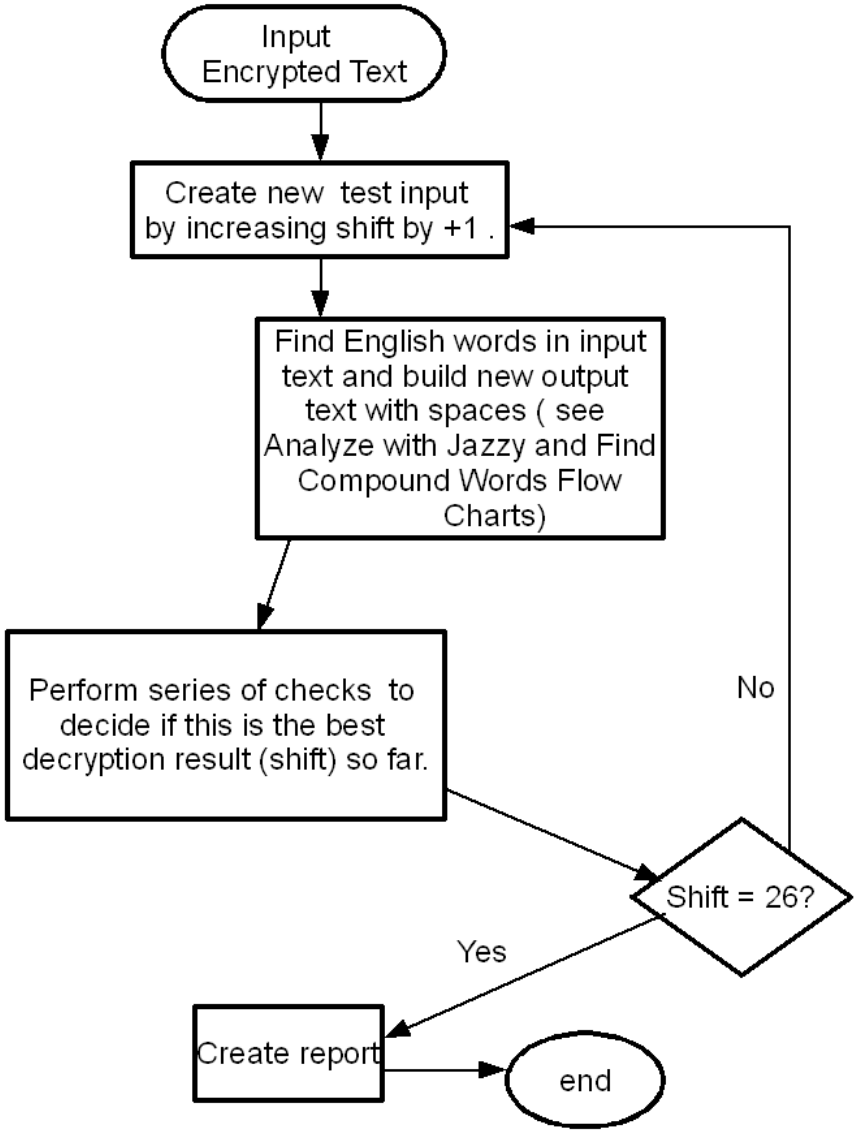

Illustration 3: Decryption Flow Chart (DecryptFile)

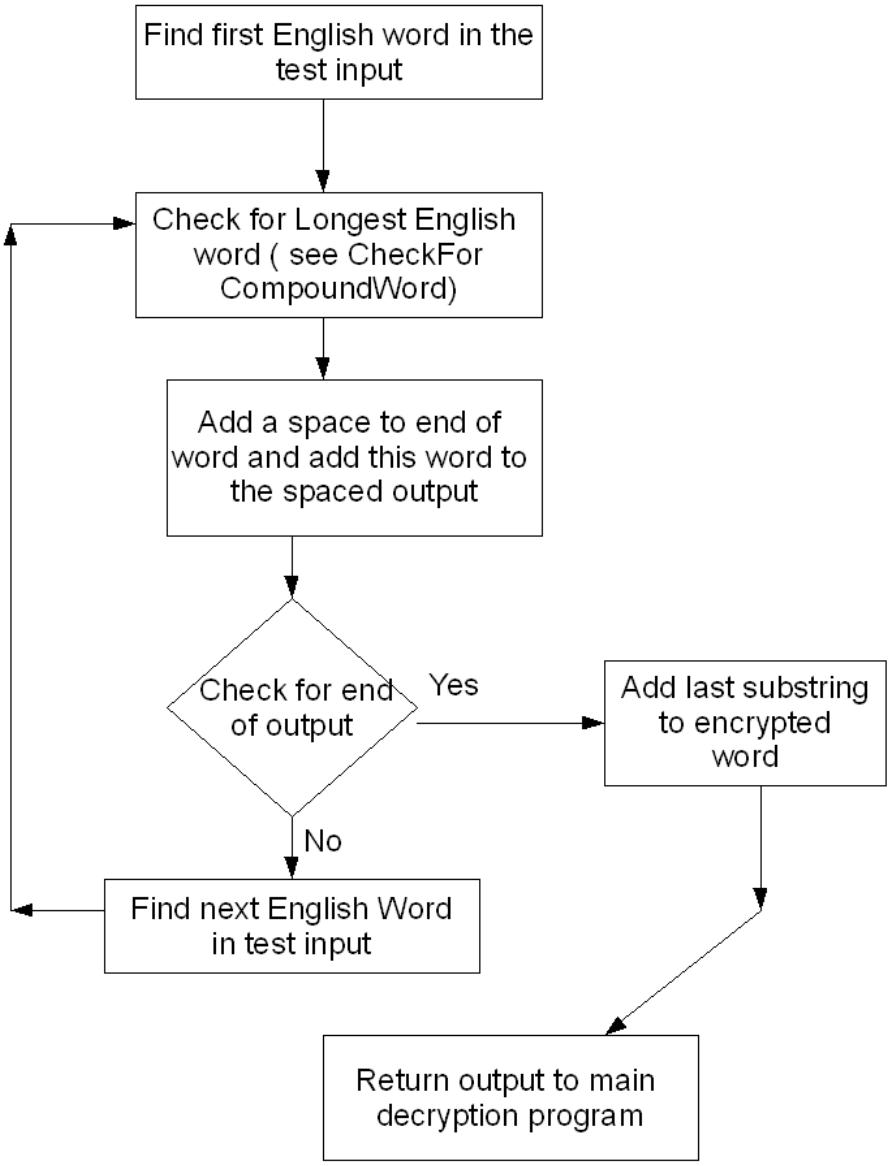

Illustration 4: Word Analysis (analyzeWithJazzy and checkForCompoundWord)

Problems We Solved (and not Solved)

Numbers and Symbols

Our encryption program had a problem with numbers and symbols. We solved this by just not encrypting or decrypting non-alphabetical characters in the input. Since Caesar, a Roman emperor, would not have had these in his text, it was decided that our Caesar Cipher should be ignoring these characters.

Example original Input: Java is a programming language originally developed by Sun Microsystems and released in 1995 [8]
Encrypted Text (shift = 11):
ULGLTDLACZRCLXXTYRWLYRFLRPZCTRTYLWWJOPGPWZAPOMJDFYXTNCZDJDEPXDLYOCPWPLDPOTY1995
Decrypted Input (shift = 11):
JAVAISAPROGRAMMINGLANGUAGEORIGINALLYDEVELOPEDBYSUNMICROSYSTEMSANDRELEASEDIN1995

Integrating with Jazzy

One of the hardest parts of our project was to integrate the Jazzy spelling checker with our decryption program. We spent several weeks of experimenting with the spelling checker and making changes to our code to find the English words.

Spacing and Finding Words

The spelling checker did a good job of finding the English words and our program also found longer words and compound words that started with English words. But adding spaces back into the decrypted text gave us some problems. After finishing finding an English word we added the space between the last word found and the new word. But Jazzy would not return unknown (but real) words that were not in its dictionary so these were totally skipped in the

output. For example if the decrypted input was:

INMERRYENGLANDINTHETIMEOFOLD,WHENGOODKINGHENRYTHESECONDRULEDTHELAND ...

We wanted the output to be:

IN MERRY ENGLAND IN THE TIME OF OLD,WHEN GOOD KING HENRY THE SECOND RULED THE LAND ...

But the output (on the first pass) was:

IN MERRY ENGLAND IN THE TIME OF OLD WHEN GOOD KING HEN THESE ON RULED THE LAND ...

which was wrong because the “RY” of HENRY was missing. “Henry” is not in the spelling checker dictionary, so the word finder stopped at “Hen” and left out the RY because this is not a valid word. It starts again when it found the word “THESE”.

Another problem is shown in these results. Our program tries to find the longest English word that starts with any valid word found. This is to avoid finding “cat” and not “caterpiller” because the “erpiller” is not a valid word and would be rejected by the spelling checker so that not all the valid English words in the decrypted text would be found.

But this leads to a problem that sometimes the shortest word found is the correct word. We cannot stop the program from finding “the” and continuing on to find “these” when the input is “thesecond”. The correct output is “the second” but the program will not stop at “the” when it can find “these” in this sequence of characters [thesecond] -> [these][cond] and the spelling checker would reject “cond”, just finding “on”. As a result the output would end up being “these on” instead of what it is supposed to be.

Results

We tested inputs of various sizes by character count and text type (such as technical articles and literature). We used technical articles from Wikipedia and literature passages from the on-line Project Gutenberg. We ran several tests and test inputs ranged from 550 to 1057 characters. Each test generated a report file. Below are two of the tests we ran (Robin Hood and Algebra) with their report files.

Robin Hood

Original, unencrypted text

> *IN MERRY ENGLAND in the time of old, when good King Henry the Second ruled the land, there lived within the green glades of Sherwood Forest, near Nottingham Town, a famous outlaw whose name was Robin Hood. No archer ever lived that could speed a gray goose shaft with such skill and cunning as his, nor were there ever such yeomen as the sevenscore merry men that roamed with him through the greenwood shades. Right merrily they dwelled within the depths of Sherwood Forest, suffering neither care nor want, but passing the time in merry games of archery or bouts of cudgel play, living upon the King's venison, washed down with draughts of ale of October brewing.*

Report for Test: RobinHood

Best Match Shift: 5

Percent Word Match: 92.13488

Total Calculation Time: 190.671
(3 minutes 10 seconds)

Original Input for Test: RobinHood

NSRJWWDJSLQFSINSYMJYNRJTKTQI,BMJSLTTIPNSLMJSWDYMJXJHTSIWZQJIYMJQF
SI,YMJWJQNAJIBNYMNSYMJLWJJSLQFIJXTKXMJWBTTIKTWJXY,SJFWSTYYNSLMFRY
TBS,FKFRTZXTZYQFBBMTXJSFRJBFXWTGNSMTTI.STFWHMJWJAJWQNAJIYMFYHTZ
QIXUJJIFLWFDLTTXJXMFKYBNYMXZHMXPNQQFSIHZSSNSLFXMNX,STWBJWJYMJWJ
JAJWXZHMDJTRJSFXYMJXJAJSXHTWJRJWWDRJSYMFYWTFRJIBNYMMNRYMWTZLM
YMJLWJJSBTTIXMFIJX.WNLMYRJWWNQDYMJDIBJQQJIBNYMNSYMJIJUYMXTKXMJW
BTTIKTWJXY,XZKKJWNSLSJNYMJWHFWJSTWBFSY,GZYUFXXNSLYMJYNRJNSRJWW
DLFRJXTKFWHMJWDTWGTZYXTKHZILJQUQFD,QNANSLZUTSYMJPNSL'XAJSNXTS,BF
XMJIITBSBNYMIWFZLMYXTKFQJTKTHYTGJWGWJBNSL.

Decoded Text for Test: RobinHood

INMERRYENGLANDINTHETIMEOFOLD,WHENGOODKINGHENRYTHESECONDRULEDT
HELAND,THERELIVEDWITHINTHEGREENGLADESOFSHERWOODFOREST,NEARNOTTI
NGHAMTOWN,AFAMOUSOUTLAWWHOSENAMEWASROBINHOOD.NOARCHEREVERLI
VEDTHATCOULDSPEEDAGRAYGOOSESHAFTWITHSUCHSKILLANDCUNNINGASHIS,N
ORWERETHEREEVERSUCHYEOMENASTHESEVENSCOREMERRYMENTHATROAMED
WITHHIMTHROUGHTHEGREENWOODSHADES.RIGHTMERRILYTHEYDWELLEDWITHIN
THEDEPTHSOFSHERWOODFOREST,SUFFERINGNEITHERCARENORWANT,BUTPASSI
NGTHETIMEINMERRYGAMESOFARCHERYORBOUTSOFCUDGELPLAY,LIVINGUPONTH
EKING'SVENISON,WASHEDDOWNWITHDRAUGHTSOFALEOFOCTOBERBREWING.

Spaced Text for Test: RobinHood

IN MERRY ENGLAND IN THE TIME OF OLD WHEN GOOD KING HEN THESE ON RULED
THE LAND THERE LIVED WITHIN THE GREEN GLADES OF SHE WOOD FOREST NEAR
NOT TIN HAM TOWN FAMOUS OUTLAW WHOSE NAME WAS ROBIN HOOD NO

ARCHER EVER LIVED THAT COULD SPEED GRAY GOOSES HA WITH SUCH SKILL AND CUNNING ASH IS NOR WERE THERE EVER SUCH YEOMEN AS THESE SCORE MERRY MEN THAT ROAMED WITH HIM THROUGH THE GREEN WOODS HAD RIGHT MERRILY THEY DWELLED WITHIN THE DEPTHS OF SHE WOOD FOREST SUFFERING NEITHER CARE NOR WANT BUT PASSING THE TIME IN MERRY GAMES OF ARCHERY ORB OUTS OF CUDGEL PLAY LIVING UPON THE KING VENISON WASHED DOWN WITH AUGHT SOFA OF TO BE BREWING

Final Spaced Output for Test: RobinHood

IN MERRY ENGLAND IN THE TIME OF OLD , WHEN GOOD KING HEN RY THESE C ON D RULED THE LAND , THERE LIVED WITHIN THE GREEN GLADES OF SHE R WOOD FOREST , NEAR NOT TIN G HAM TOWN ,A FAMOUS OUTLAW WHOSE NAME WAS ROBIN HOOD . NO ARCHER EVER LIVED THAT COULD SPEED A GRAY GOOSES HA FT WITH SUCH SKILL AND CUNNING ASH IS , NOR WERE THERE EVER SUCH YEOMEN AS THESE VEN SCORE MERRY MEN THAT ROAMED WITH HIM THROUGH THE GREEN WOODS HAD ES. RIGHT MERRILY THEY DWELLED WITHIN THE DEPTHS OF SHE R WOOD FOREST , SUFFERING NEITHER CARE NOR WANT , BUT PASSING THE TIME IN MERRY GAMES OF ARCHERY ORB OUTS OF CUDGEL PLAY , LIVING UPON THE KING 'S VENISON , WASHED DOWN WITH DR AUGHT SOFA LE OF OC TO BE R BREWING .

Algebra

Original unencrypted text from wikipedia (http://en.wikipedia.org/wiki/Algebra) [9]:

> *Algebra is a branch of mathematics concerning the study of structure, relation and quantity. The name is derived from the treatise written in Arabic by the Persian[1] mathematician, astronomer, astrologer and geographer, Muhammad bin Muhammad bin Mūsā al-Khwārizmī titled Kitab al-Jabr wa-l-Muqabala (meaning "The Compendious Book on Calculation by Completion and Balancing"), which provided symbolic operations for the systematic solution of linear and quadratic equations.*
>
> *Together with geometry, analysis, combinatorics, and number theory, algebra is one of the main branches of mathematics. Elementary algebra is often part of the curriculum in secondary education and provides an introduction to the basic ideas of algebra, including effects of adding and multiplying numbers, the concept of variables, definition of polynomials, along with factorization and determining their roots.*
>
> *Algebra is much broader than elementary algebra and can be generalized. In addition to working directly with numbers, algebra covers working with symbols, variables, and set elements. Addition and multiplication are viewed as general operations, and their precise definitions lead to structures such as groups, rings and fields.*

Report for Test: Algebra

Best Match Shift: 2

Percent Word Match: 88.33397

Total Calculation Time: 1366.813
(22 minutes 46 seconds)

Original Input for Test: Algebra

CNIGDTCKUCDTCPEJQHOCVJGOCVKEUEQPEGTPKPIVJGUVWFAQHUVTWEVWTG,TG

NCVKQPCPFSWCPVKVA.VJGPCOGKUFGTKXGFHTQOVJGVTGCVKUGYTKVVGPKPCT
CDKEDAVJGRGTUKCP[1]OCVJGOCVKEKCP,CUVTQPQOGT,CUVTQNQIGTCPFIGQITC
RJGT,OWJCOOCFDKPOWJCOOCFDKPONULCN-MJYLTKBOBVKVNGFMKVCDCN-
LCDTYC-N-
OWSCDCNC(OGCPKPI"VJGEQORGPFKQWUDQQMQPECNEWNCVKQPDAEQORNGVK
QPCPFDCNCPEKPI"),YJKEJRTQXKFGFUAODQNKEQRGTCVKQPUHQTVJGUAUVGOCV
KEUQNWVKQPQHNKPGCTCPFSWCFTCVKEGSWCVKQPU.VQIGVJGTYKVJIGQOGVTA,
CPCNAUKU,EQODKPCVQTKEU,CPFPWODGTVJGQTA,CNIGDTCKUQPGQHVJGOCKPD
TCPEJGUQHOCVJGOCVKEU.GNGOGPVCTACNIGDTCKUQHVGPRCTVQHVJGEWTTKE
WNWOKPUGEQPFCTAGFWECVKQPCPFRTQXKFGUCPKPVTQFWEVKQPVQVJGDCUK
EKFGCUQHCNIGDTC,KPENWFKPIGHHGEVUQHCFFKPICPFOWNVKRNAKPIPWODGTU,
VJGEQPEGRVQHXCTKCDNGU,FGHKPKVKQPQHRQNAPQOKCNU,CNQPIYKVJHCEVQT
KBCVKQPCPFFGVGTOKPKPIVJGKTTQQVU.CNIGDTCKUOWEJDTQCFGTVJCPGNOG
PVCTACNIGDTCCPFECPDGIGPGTCNKBGF.KPCFFKVKQPVQYQTMKPIFKTGEVNAYKV
JPWODGTU,CNIGDTCEQXGTUYQTMKPIYKVJUAODQNU,XCTKCDNGU,CPFUGVGNGO
GPVU.CFFKVKQPCPFOWNVKRNKECVKQPCTGXKGYGFCUIGPGTCNQRGTCVKQPU,C
PFVJGKTRTGEKUGFGHKPKVKQPUNGCFVQUVTWEVWTGUUWEJCUITQWRU,TKPIUC
PFHKGNFU.

Decoded Text for Test: Algebra

ALGEBRAISABRANCHOFMATHEMATICSCONCERNINGTHESTUDYOFSTRUCTURE,REL
ATIONANDQUANTITY.THENAMEISDERIVEDFROMTHETREATISEWRITTENINARABICBY
THEPERSIAN[1]MATHEMATICIAN,ASTRONOMER,ASTROLOGERANDGEOGRAPHER,M
UHAMMADBINMUHAMMADBINMLSJAL-KHWJRIZMZTITLEDKITABAL-JABRWA-L-
MUQABALA(MEANING"THECOMPENDIOUSBOOKONCALCULATIONBYCOMPLETIONAN
DBALANCING"),WHICHPROVIDEDSYMBOLICOPERATIONSFORTHESYSTEMATICSOLU
TIONOFLINEARANDQUADRATICEQUATIONS.TOGETHERWITHGEOMETRY,ANALYSIS,
COMBINATORICS,ANDNUMBERTHEORY,ALGEBRAISONEOFTHEMAINBRANCHESOFM

ATHEMATICS.ELEMENTARYALGEBRAISOFTENPARTOFTHECURRICULUMINSECONDARYEDUCATIONANDPROVIDESANINTRODUCTIONTOTHEBASICIDEASOFALGEBRA,INCLUDINGEFFECTSOFADDINGANDMULTIPLYINGNUMBERS,THECONCEPTOFVARIABLES,DEFINITIONOFPOLYNOMIALS,ALONGWITHFACTORIZATIONANDDETERMININGTHEIRROOTS.ALGEBRAISMUCHBROADERTHANELEMENTARYALGEBRAANDCANBEGENERALIZED.INADDITIONTOWORKINGDIRECTLYWITHNUMBERS,ALGEBRACOVERSWORKINGWITHSYMBOLS,VARIABLES,ANDSETELEMENTS.ADDITIONANDMULTIPLICATIONAREVIEWEDASGENERALOPERATIONS,ANDTHEIRPRECISEDEFINITIONSLEADTOSTRUCTURESSUCHASGROUPS,RINGSANDFIELDS.

Spaced Text for Test: Algebra

ALGEBRA IS BRANCH OF MATHEMATICS CONCERNING THE STUDY OF STRUCTURE RELATION AND QUANTITY THEN AM IS DERIVED FROM THE TREATISE WRITTEN IN BY THE PER AN MATHEMATICIAN ASTRONOMER AS LOG ERA GRAPH HAM MAD BIN HAM MAD BIN ML TITLED KIT JAB QA MEANING THE PEND US BOOK ON CALCULATION BY COMPLETION AND BALANCING WHICH PROVIDED SYMBOLIC OPERATIONS FORTH SYSTEMATICS ION OF LINEAR AND QUADRATIC EQUATIONS TOGETHER WITH GEOMETRY ANALYSIS COMBINATORICS AND NUMBER THEORY ALGEBRA IS ONE OFT HEM IN BRANCHES OF MATHEMATICS ELEMENTARY ALGEBRA IS OFTEN PART OFT HE CURRICULUM INS ON RYE CAT ION AND PROVIDES AN INTRODUCTION TO THE BASIC IDEAS OF ALGEBRA INCLUDING EFFECTS OF ADDING AND MULTIPLYING NUMBERS THE CONCEPT OF VARIABLES DEFINITION OF POLYNOMIALS ALONG WITH FACTOR AT ION AND DETERMINING THEIR ROOTS ALGEBRA IS MUCH BROADER THAN ELEMENTARY ALGEBRA AND CAN BEG ERA IN ADDITION TOW OR KING DIRECTLY WITH NUMBERS ALGEBRA COVERS WORKING WITH SYMBOLS VARIABLES ANDS ELEMENTS ADDITION AND MULTIPLICATION ARE VIEWED AS GENERAL OPERATIONS AND THEIR PRECISE DEFINITIONS LEAD TO STRUCTURES SUCH AS GROUPS RINGS AND FIELDS

Final Spaced Output for Test: Algebra

ALGEBRA IS A BRANCH OF MATHEMATICS CONCERNING THE STUDY OF STRUCTURE , RELATION AND QUANTITY . THEN AM E IS DERIVED FROM THE TREATISE WRITTEN IN ARABIC BY THE PER SI AN [1] MATHEMATICIAN , ASTRONOMER , AS TRO LOG ERA NDGEO GRAPH ER,MU HAM MAD BIN MU HAM MAD BIN ML SJAL-KHWJRIZMZ TITLED KIT ABAL- JAB RWA-L-MU QA BALA(MEANING " THE COM PEND IO US BOOK ON CALCULATION BY COMPLETION AND BALANCING "), WHICH PROVIDED SYMBOLIC OPERATIONS FORTH E SYSTEMATICS OLUT ION OF LINEAR AND QUADRATIC EQUATIONS . TOGETHER WITH GEOMETRY , ANALYSIS , COMBINATORICS , AND NUMBER THEORY , ALGEBRA IS ONE OFT HEM A IN BRANCHES OF MATHEMATICS . ELEMENTARY ALGEBRA IS OFTEN PART OFT HE CURRICULUM INS EC ON DA RYE DU CAT ION AND PROVIDES AN INTRODUCTION TO THE BASIC IDEAS OF ALGEBRA , INCLUDING EFFECTS OF ADDING AND MULTIPLYING NUMBERS , THE CONCEPT OF VARIABLES , DEFINITION OF POLYNOMIALS , ALONG WITH FACTOR IZ AT ION AND DETERMINING THEIR ROOTS . ALGEBRA IS MUCH BROADER THAN ELEMENTARY ALGEBRA AND CAN BEG EN ERA LIZED. IN ADDITION TOW OR KING DIRECTLY WITH NUMBERS , ALGEBRA COVERS WORKING WITH SYMBOLS , VARIABLES , ANDS ET ELEMENTS . ADDITION AND MULTIPLICATION ARE VIEWED AS GENERAL OPERATIONS , AND THEIR PRECISE DEFINITIONS LEAD TO STRUCTURES SUCH AS GROUPS , RINGS AND FIELDS .

Conclusion

We found that we could use a spelling checking library (Jazzy) to find English words in the decrypted text and the program could decide which decryption shift was correct.

We were not able to find the spaces in the decrypted text in the right places, but the program did a good job finding longer words such as “forest” and not stopping at “for”, and “outlaw” instead of “out” and “law” as separate words.

We got a deeper understanding of Java itself by just experimenting during the programming. We also learned about the Caesar Cipher, decryption, encryption and cryptography in general.

The whole group learned a lot about teamwork because none of us have worked with a team on a yearlong project before.

Acknowledgments

We would like to thank Zeynep Unal, our school sponsor for her support and letting us meet twice a week with our mentor for the many months in her room to work on the project.

We thank our mentor, Liz Cooper for her help on this project and her good ideas.

We also thank the work done by the challenge Consult team, especially Celia Einhorn who worked out the registration for the on-line Sun Java course that we are still participating in, and to Dave Kratzer and Bob Robey who got us started and continued to keep us updated with news.

We would also like to thank Jim Redman for technical help and advice.

One more person we thank is Jody Hesch for his feedback on our proposal and interim report.

References

[1] Wikipedia Article on Caesar Cipher
Wikipedia http://en.wikipedia.org/wiki/Caesar_cipher
[2] Source code for Caesar Library written by Valerio Capozio
http://www.angelusworld.com/softwarephp?num=7
[3] Jazzy Java spelling checker
http://sourceforge.net/projects/jazzy
[4] Project Gutenberg http://www.gutenberg.org/
[5] Wikipedia Article on Cryptography
http://en.wikipedia.org/wiki/Cryptography
[6] Wikipedia Ariticle on Substitution Ciphers
http://en.wikipedia.org/wiki/Substitution_cipher
[7] Passage from Project Gutenberg http://www.gutenberg.org/files/964/964txt “The Merry Adventures of Robin Hood” by Howard Pyle Release Date: February 5, 2006 [EBook #964]
[8] Wikipedia Article on Java
http://en.wikipedia.org/wiki/Java
[9] Wikipedia Article on Algebra
http://en.wikipedia.org/wiki/Algebra

Appendix A. Source Code for EncryptFile

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

import javax.swing.JOptionPane;

/**
 * Original idea from sample code found on the
 * internet http://ews.uiuc.edu/~jtimmer2/Caesar/Caesar.java
 * originally written by Mr. Ujidiku Timmermann
 *  code heavily modified by Colin Redman, 2007-2008
 *  to read file of unencrypted text and write output file
 *  and to use the GPL Ceasar Library by Valerio Capozio
 *
 */
public class EncryptFile {
  public static void main(String[] args) {
    System.out.println("Encrypted Text Generator");

    String message = (String)JOptionPane.showInputDialog("Copy the Text Here:");

    String testName = (String)JOptionPane.showInputDialog("Enter a Name for the Test").trim();

    //Convert input to upper case
    message = message.toUpperCase();
    //Remove all spaces
    message = message.trim();

    // create random shift
    Random generator = new Random();
    int shiftNum = generator.nextInt(26)-1;  //always non-zero

    Caesar caesar = new Caesar();
   //create the encoded text
    String encodedText = caesar.encodeWithoutWhiteSpace(message,shiftNum);
    String fileSeparator = System.getProperty("file.separator");
    try {
      new File(testName).mkdir();
      FileOutputStream outFile = new FileOutputStream(testName+fileSeparator+testName+".txt");
```

```
      PrintWriter out = new PrintWriter(outFile);

      out.println(encodedText);

      out.close();
    } catch (IOException e){
      e.printStackTrace();
    }
    System.out.println("Finished Encryption for test " + testName);


  }

  //Similar to encrypt code, but just removes whitespace
  private static String removeWhitespace(String message) {
    char ch;
    String res="";
    for (int i=0; i < message.length(); i++) {
      ch = (message.charAt(i));
      if (Character.isLetterOrDigit(ch)) {
                        res += String.valueOf(ch);
      } else {

      }
    }
    return res;

  }
}
```

Appendix B. Source Code for DecryptFile

```
/**
 * Decryption of Caesar Encrypted Text file
 * Integrated with Jazzy to check for English words to
 * detect correct decryption shift
 * Colin Redman
 * Dhaivat Panyda
 * 2007-2008 NM Supercomputer Challenge
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Vector;

import javax.swing.JOptionPane;

import com.swabunga.spell.engine.SpellDictionaryHashMap;
import com.swabunga.spell.event.SpellCheckEvent;
import com.swabunga.spell.event.SpellChecker;

public class DecryptFile {
        public static final String DICTIONARY_FILE = "dict/english.0";// name of

        // the dictionary file in the local path

        protected static SpellChecker spellChecker;// specllChecker object for

        // Jazzy

        static int maxWordLength = 30;//this is a guess at the largest number of characters expected in
an English word

        public static String fileSeparator = System.getProperty("file.separator");

        /**
         * The raw encypted text input
         */
        String input = "";
```

```
/**
 * The shifted text input
 */
String decodedText = "";

/**
 * Text with spaces (only if Jazzy returns good results)
 */
String spacedText = "";

// vector of words found
Vector corrector = new Vector();

// vector of words found plus the words NOT in the Jazzy dictionary
Vector finalResult = new Vector();

/**
 * Counts number of English words found
 */
int wordCounter;

/**
 * Calculated average word length
 */
float aveWordLength = 0.0f;

/**
 * Calculated percent of characters that are English words
 */
float percentMatch = 0.0f;

/**
 * time for the English word analysis
 */
float calculationTime = 0.0f;

String addToMessage = "";

/**
 * Name of the input file without the .txt
 */
static String inputName = "";

public void doDecryption(String fileName) throws IOException {
        Caesar caesar = new Caesar();
```

```
/**keeps track of best percent English word match*/
float bestPercent = -1;

/**keeps track of the shift for the best percent English word match*/
int bestShift = -1;

/**keeps track of the total calculation time for this test*/
float totalCalculationTime = 0;

/** Keep text of best decoded message*/
String bestDecodedText = "";
/** Keep spaced text of best decoded message */
String bestSpacedText = "";

/** keep the best final output */
String bestFinalOutput = "";

// READ The file
input = getContents(fileName);
System.out.println("Text read in: " + input);
int tryShift = 0; // try 1 to start (input is always encrypted)

//  Test all shifts from 1 to 25
while (tryShift < 25) {
        tryShift++;
        aveWordLength = 0;
        percentMatch = 0;
        wordCounter = 0;
        long startTime = System.currentTimeMillis();
        decodedText = caesar.decode(input, tryShift);
        /** Here is where we test the input for English words*/
        boolean testValue = analyzeWithJazzy();
        long endTime = System.currentTimeMillis();
        long diffTime = endTime - startTime;
        calculationTime = ((diffTime) * 1.0f) / 1000.0f; //in seconds
        totalCalculationTime = totalCalculationTime + calculationTime;
        //keep track of best matches for final report
        if (percentMatch > bestPercent) {
                bestShift = tryShift;
                bestPercent = percentMatch;
                bestDecodedText = decodedText;
                bestSpacedText = spacedText;
                bestFinalOutput = finalCheck();
        }
```

```
                        String experimentalInformation = "Experimental Information\n"
                                        + " Total Characters=" + input.length() + "\n"
                                        + " Words Matched=" + wordCounter + "\n"
                                        + " Average Characters Per Word=" + aveWordLength + "\n"
                                        + " %Word Match=" + percentMatch + "\n "
                                        + "Calculation Time= " + calculationTime;
                        //    write the decrypted text file as an experimental result
                        fileSeparator = System.getProperty("file.separator");
                        try {
                                FileOutputStream outFile = new FileOutputStream(inputName
                                                + fileSeparator + inputName + "." + tryShift + ".txt");
                                PrintWriter out = new PrintWriter(outFile);
                                out.println(decodedText);
                                out.println(experimentalInformation);
                                out.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        if (testValue == false) {
                                //count = 0;
                                //countLength = 0;
                                //matchCount = 0;
                        } else {
//write the spaced file for the results that Jazzy returns as good matches
                try {
                                        FileOutputStream outFile = new FileOutputStream(inputName
                                                        + fileSeparator + inputName + ".spaced" + "."
                                                        + tryShift + ".txt");
                                        PrintWriter out = new PrintWriter(outFile);
                                        out.println(spacedText);
                                        out.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }

                        }

                }
                //Write final report

                try {

                        FileOutputStream outFile = new FileOutputStream(inputName
                                        + fileSeparator + inputName + ".report.txt");
                        PrintWriter out = new PrintWriter(outFile);
                        out.println("Report for Test: " + inputName + "\n\r");
```

```
                        out.println("Best Match Shift: " + bestShift + "\n\r");
                        out.println("Percent Word Match: " + bestPercent + "\n\r");
                        out.println("Total Calculation Time: " + totalCalculationTime);
                        if (totalCalculationTime > 60) {
                                long minutes = ((int) totalCalculationTime / 60);
                                int seconds = (int) (totalCalculationTime - minutes * 60.0f);
                                out.println(" (" + minutes + " minutes " + seconds
                                                + " seconds)");
                        }
                        out.println("\n\r");
                        out.println("Original Input for Test: " + inputName + "\n\r");
                        out.println(input + "\n\r");
                        out.println("Decoded Text for Test: " + inputName + "\n\r");
                        out.println(bestDecodedText + "\n\r");
                        out.println("Spaced Text for Test: " + inputName + "\n\r");
                        out.println(bestSpacedText + "\n\r");
                        out.println("Final Spaced Output for Test: " + inputName + "\n\r");
                        out.println(bestFinalOutput + "\n\r");
                        out.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * This is the method that does all the work to decide if the decoded text is correct
         * @return
         * @throws IOException
         */
        protected boolean analyzeWithJazzy() {

                boolean isTrueDecryptedText = false;
                String lastWordFound = "";

                String input = decodedText;// assuming that the entire text is
// just one line - that is - not
// "newline" or "carriage return"s
        spacedText = "";
//make sure the corrector Vector is empty before we add the words to it
        corrector.clear();
        wordCounter = 0;
        int i = 0;
//iterate through to find English substrings, starting at the first character

        while (i < input.length()) {
                        lastWordFound = "";
```

```
                        for (int j = i + 1; j <= input.length(); j++) {

                          String checking = decodedText.substring(i, j);
                          if (spellChecker.isCorrect(decodedText.substring(i, j))) {

/**
* Now looks at next words so that if the actual word is "caterpillar" the
* search doesn't stop at cat and then the program doesn't try "erpillar".
*/

    String compoundWord = input.substring(i, j);
    lastWordFound = input.substring(i, j);
    int startIndex = i;
//check for a longer or compound word starting with the lastWordFound.
//if result is a blank then no additional word was found
 String result = checkForCompoundWord(compoundWord,startIndex);
               if (result != "") {
//found a compound or longer word starting with lastWordFound
                        lastWordFound = result;
               }

               j = i + lastWordFound.length();

               wordCounter++;
               if (wordCounter > 1) {
                 aveWordLength = ((aveWordLength * (wordCounter - 1)) +
lastWordFound.length())/wordCounter;
               } else {
                 aveWordLength = lastWordFound.length();
               }

               spacedText = spacedText + lastWordFound + " ";
               corrector.add(lastWordFound);
            i = j;

/*check if about 80% of the encrypted text has been searched and there are at least 1/10th the number
of words as there are letters. if yes,then we found the correct decryption.
*
*/
        if (isTrueDecryptedText == false) {//only check if it is still false
               if (i >= (int) (8 * input.length() / 10)
               && wordCounter >= (int) i / 10
               && wordCounter >= 1) {
        isTrueDecryptedText = true;
       }
        }
```

```
        } else {
                 }
        }
        i++;
               }

        System.out.println(finalResult.toString());
//calculate the percent match for matched words that were in the Jazzy dictionary
        percentMatch = 100.0f * (aveWordLength * (wordCounter - 1))
 / (input.length() * 1.0f);
        return isTrueDecryptedText;
  }

        /** This checks for longer words after an English word is found.
       * For example if "cat" is found, and "erpiller" immediately follows,
       * the entire word that is found is "caterpiller".
        * Only go maxWordLength characters along to find the new longer word
        */
        protected String checkForCompoundWord(String startWord, int startIndex) {

                int endIndex = startIndex + startWord.length() + 1;
                String lastWordFound = "";
                int maxIndex = decodedText.length() - 1;//maximum index to go to in the text
                //check that the startWord is NOT the final word in the text
                if (decodedText.length() - startWord.length() == decodedText.indexOf(
                                startWord, startIndex)) {
                        return lastWordFound;
                }
                while (endIndex <= maxIndex
                                && ((endIndex - startIndex) < maxWordLength)) {

                        if (endIndex > maxIndex) {
                                endIndex = maxIndex;
                        }
                        if (spellChecker.isCorrect(decodedText.substring(startIndex,
                                        endIndex))) {
                                lastWordFound = decodedText.substring(startIndex, endIndex);
                        }
                        endIndex++;
                }

                return lastWordFound;
        }

//required for Jazzy
```

```
	public void spellingError(SpellCheckEvent event) {
	}
//required for Jazzy - initializes the spelling dictionary
	private void createDictionary() {// reads in the dictionary file so that
		// Jazzy can then use that file to check
		// spellings

		File dict = new File(DICTIONARY_FILE);
		try {
			spellChecker = new SpellChecker(new SpellDictionaryHashMap(dict));
		} catch (FileNotFoundException e) {
			System.err.println("Dictionary File '" + dict
					+ "' not found! Quitting. " + e);
			System.exit(1);
		} catch (IOException ex) {
			System.err
					.println("IOException occurred while trying to read the dictionary
file: "
							+ ex);
			System.exit(2);
		}
	}

	/**
	 * Gets the entire contents of a text file, and return it in a String.
	 * @param aFile
	 *            is a file which already exists and can be read.
	 */
	public static String getContents(String fileName) {

		File inputFile = new File(fileName);
		StringBuffer contents = new StringBuffer();

		// declared here only to make visible to finally clause
		BufferedReader input = null;
		try {
			// use buffering, reading one line at a time
			// FileReader always assumes default encoding is OK!
			input = new BufferedReader(new FileReader(inputFile));
			String line = null; // not declared within while loop
			/*
			 * readLine is a bit quirky : it returns the content of a line MINUS
			 * the newline. it returns null only for the END of the stream. it
			 * returns an empty String if two newlines appear in a row.
			 */
			while ((line = input.readLine()) != null) {
```

```
                        contents.append(line);
                        contents.append(System.getProperty("line.separator"));
                }
        } catch (FileNotFoundException ex) {
                ex.printStackTrace();
        } catch (IOException ex) {
                ex.printStackTrace();
        } finally {
                try {
                        if (input != null) {
                                // flush and close both "input" and its underlying
                                // FileReader
                                input.close();
                        }
                } catch (IOException ex) {
                        ex.printStackTrace();
                }
        }
        return contents.toString();
    }


/** This part of the code looks back at the vector of correct words found and fills in the gaps with
words that were in the decoded text but were NOT in the Jazzy library. This fixes the problem with
leaving out short words that are slang, abbreviations, etc. */

    public String finalCheck() {

        int startIndex = 0;
        //make sure the finalResult Vector is empty to start
        finalResult.clear();
//If the corrector Vector has NO elements this means there were no words //found, so no use in
continuing.
        if (corrector.size() != 0) {
                //Question - what if the first word in the decoded text is NOT in the corrector
Vector?
                //Let's check:
                String firstWordInCorrrectorVector = (String) corrector
                                .elementAt(0);
                //Is this the first word in the decodedText?
                int foundFirstWordAt = decodedText
                                .indexOf(firstWordInCorrrectorVector);
                //if index is not 0, we need to use the real first word, not the first one
                //in the corrector Vector
                if (foundFirstWordAt != 0) {
                        String firstWord = decodedText.substring(0, foundFirstWordAt);
```

```
                        finalResult.add(firstWord);
                        startIndex = foundFirstWordAt;
                }
                for (int elementCount = 0; elementCount < corrector.size(); elementCount++) {
                        String word = (String) corrector.elementAt(elementCount);
                        //find the index of this word in the decoded text
                        int wordIndex = decodedText.indexOf(word, startIndex);
                        //is it at the startIndex, or somewhere else?
                        if (wordIndex == startIndex) {
                                //add word to the final result Vector
                                finalResult.add(word);
                                startIndex = startIndex + word.length();
                        } else {
                                // add the text between the last found word and this one to the
finalResult, then add
                                // the word to the final result. Garbage is all the text between
the startIndex and
                                //where the next corrector word is in the decoded text
                                String garbage = decodedText.substring(startIndex,
                                                wordIndex);
                                finalResult.add(garbage);
                                finalResult.add(word);
                                startIndex = startIndex + garbage.length() + word.length();
                        }

                }
                //check to see if there is some more garbage at the end and add it to the final
result

                if (!(decodedText.endsWith((String) corrector.lastElement()))) {
                        int lastIndex = decodedText.indexOf((String) corrector
                                        .lastElement());
                        String endString = decodedText.substring(lastIndex
                                        + (((String) corrector.lastElement()).length()));
                        finalResult.add(endString);
                }

        }
        //build the final output
        String output = "";
        for (int indx = 0; indx < finalResult.size(); indx++) {
                output = output + " " + (String) (finalResult.elementAt(indx));

        }
        return output;
    }
```

```
	/**
	 * Decrypt the input file
	 */
	public static void main(String[] args) {
		DecryptFile decrypt = new DecryptFile();
		decrypt.createDictionary();
		//enter a .txt file for decryption
		inputName = (String) JOptionPane.showInputDialog("Enter the Test Name")
				.trim();
		fileSeparator = System.getProperty("file.separator");
		try {
			decrypt
					.doDecryption(inputName + fileSeparator + inputName
							+ ".txt");
		} catch (IOException e) {
			System.out.println("Cound not read input file called " + inputName
					+ ".txt from directory " + inputName);
		}
	}

}
```

Appendix C. Caesar Library With Code Changes

```
/**
 * CesareCipher.java Classe che implementa l'algoritmo di cifratura di Cesare.
 * @author Valerio Capozio
 * @author Sito Web: http://www.angelusworld.com
 * @author Email: valeriocapozio@angelusworld.com
 */

/*********************************************
*                                            *
*       Progetto: Angelus' Ciphers v 1.0     *
*       Copyright 2007 Capozio Valerio       *
*                                            *
* Questo programma è free software e può essere *
* modificato e ridistribuito liberamente     *
* secondo i termini della licenza GPL.       *
*                                            *
* Per informazioni ed aggiornamenti consultare *
* il sito: http://www.angelusworld.com       *
*                                            *
*********************************************/

public class Caesar {

private int key = 0; //this is set with a random number before the encryption

    /**
     * Metodo che restituisce il valore della chiave di cifratura.
     * @return int Rappresenta il valore della chiave.
     */
    public int getKey(){
        return key;
    }

    /**
     * Metodo che permette di settare il valore della chiave di cifratura. Di default è 3.
     * @param int k rappresenta il valore da assegnare alla chiave.
     */
    public void setKey(int k){
        this.key=k;
    }

    /**
```

```
 * Metodo che consente di effettuare l'operazione di cifratura su un testo passato.
 * Modified by C Redman to accept the shiftNumber
 * @param String Il testo da cifrare
 * @return  String Il testo cifrato
 */
public String encode(String text,int shiftNum) {
    char ch;
    String res="";
    for (int i=0; i < text.length(); i++) {
        ch = (text.charAt(i));
        ch = shiftSingle(ch, shiftNum);
     res += String.valueOf(ch);
    }
    return res;
}




/**
 * C Redman changes
 * This is the same as encode with the shift except that it works
 * on inputs without whitespace
 * @param String Il testo da cifrare
 * @return  String Il testo cifrato
 */
public String encodeWithoutWhiteSpace(String text,int shiftNum) {
  char ch;
  String res="";
  for (int i=0; i < text.length(); i++) {
    ch = (text.charAt(i));
    ch = shiftSingle(ch, shiftNum);
    //String test = ch + "";
    if (Character.isWhitespace(ch)) {
      //do nothing
    } else {
      res += String.valueOf(ch);
    }
  }
  return res;
}


/**
```

```
     * Metodo che consente di effettuare l'operazione di decifratura su un testo precedentemente
cifrato.
     * @param String il testo da decifrare
     * @return String il testo decifrato
     */
    public String decode(String text) {
        char ch;
        String res="";
        for (int i=0; i < text.length(); i++) {
            ch = (text.charAt(i));
            ch = shiftSingle(ch, 26-key);
         res += String.valueOf(ch);
        }
        return res;
    }


    /**
     * Same as decode but accepts a shift. We never used the decode without a test shift.
     * @param String il testo da decifrare
     * @return String il testo decifrato
     */
    public String decode(String text,int shiftKey) {
        char ch;
        String res="";
        for (int i=0; i < text.length(); i++) {
            ch = (text.charAt(i));
            ch = shiftSingle(ch, 26-shiftKey);
         res += String.valueOf(ch);
        }
        return res;
    }


    /*
     * Metodo per lo shifting dei caratteri.
     * changed by C Redman to ignore digits
     * @param char carattere da spostare.
     * @param int valore che indica di quanto il carattere sara' shiftato.
     * @return char il carattere shiftato.
     */
    private char shiftSingle (char in, int shiftBy) {
      char out = in;
      int numA = (int)'A';
      int numa = (int)'a';
      int num0 = (int)'0';
```

```
      //char[] i =
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z
"};


      if (Character.isUpperCase(in))
        out = ((char)(((((int)in) + shiftBy - numA)%26) + numA));

      if (Character.isLowerCase(in))
        out = ((char)(((((int)in) + shiftBy - numa)%26) + numa));

      if (Character.isDigit(in))
       out = in;
      return out;
   }

}
```