# Report - details details

#### 1. Outline

blah blah blah

#### 2. Solution for Cipher 1

Plain Text:

“the discussion method of teaching, fairly common in the humanities, is seldom used in engineering; however, it can be a very useful supplement in lecture classes. in cooperative groups most of the learning occurs with students working together in small groups. this method has been used for the entire course or as a supplement in lecture classes. a variety of other methods such as panels or debates can be used to spark student interest and encourage student involvement.”

Key: **#y#wvutsr#pongefmlk#ihdcba**

A -> [Unknown]

B -> y

C -> [Unknown]

D -> w

E -> v

F -> u

G -> t

H -> s

I -> r

J -> [Unknown]

K -> p

L -> o

M -> n

N -> g

O -> e

P -> f

Q -> m

R -> l

S -> k

T -> [Unknown]

U -> i

V -> h

W -> d

X -> c

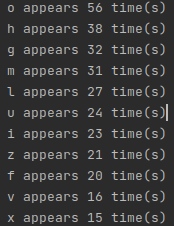
Y -> b

Z -> a

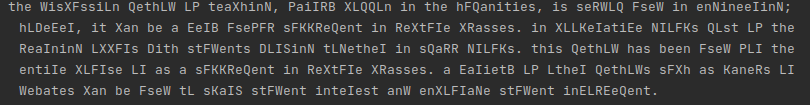
#### 3. Cipher 1 Cryptanalysis

//TODO too many words

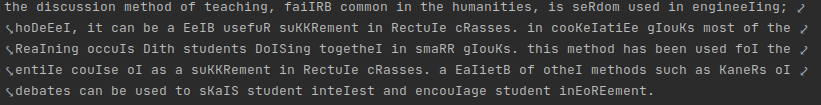
*Note that in all text block below, a* ***lower case bold letter*** *is one which has been mapped to plaintext. A CAPITAL LETTER is one which has not yet been mapped*



* First, I wrote and ran a frequency analysis script. This told me that “O” was overwhelmingly the most common letter.
* The fact that all the vowels in the text were low frequency told me that this was probably a *substitution* cipher (transposition only would preserve English frequency distribution).
* I wrote a script which substitutes in letters of a provided alphabet. I initially added a mapping from “O” to “e” (so all “O”s in the ciphertext were substituted as “e”s)
* The text now begins with “GV**e** WUHXFHHULM”, it seems sensible to assume that the first word might be “THE”, as this is a very common first word in English sentences. Hence, “G” is “t”, and “v” is “h”
* Now, “Z” appears alone (a single letter word), so is likely “I” or “A”. I here assume that “Z” is “I”
* I now noticed a pair of words: “y**e i**”, This could be “we I”, “be I”, or “me I”; none of these makes much sense so the assumption of “Z” being “I” is probably wrong. I instead decide that “Z” is “A”
* By the logic of the above point, “\_E A”, is probably “BE A”. (It could be “ME A”, as in “Call me a student”, but this seems less likely in a formal context.). So “Y” is now “B”
* The text now has an interesting run of words: “**th**UH Q**eth**LW **ha**H **bee**M”. The latter two words could only feasibly be “has been” or “had been” (other words, like “beef” are unlikely”). I now assume that “M” is “n”
* Using the same words in the above point, “**th**UH Q**eth**LW **ha**H”. The “H” in both words suggests that “H” is “s”, (because if it was “d”, then **th**U**d** would not make sense). Therefore, “U” is probably “i” (making “**this**”)
* The text now looks like this:

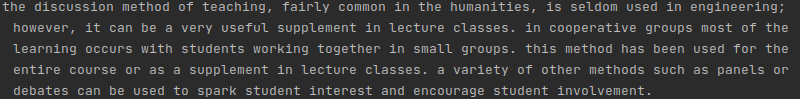


* “W**ebates** X**an be**” is probably “**debates can be**”, it seems safe to assume this (other options like “rebates” are unlikely). So “W” is “d”, and “X” is “c”
* The word “**disc**F**ssi**L**n**” is probably “**discussion**”, so “F” is u”, and “L” is “o”.
* Clearly the word “Q**ethod**” is “**method**” (as it follows “**discussion**”, so this is the “discussion method of...”
* The text now begins: “**the discussion method o**P **teachin**N”. Clearly, “P” is “f” and “N” is “g”
* The text is now mostly readable and the remaining characters can be easily deduced:



* “**fai**ILB **common in the humanities**”. “ILB” is “rly”, making “**fairly**”
* “i**t can be a** E**ery useful**”. “E” is “v”, making “**very”**
* “**su**KK**lement**”. “K” is “p”, making “**supplement**”
* “**ho**D**ever**”. “D” is “w”, making “**however**”
* “**used to spar**S **student interest**”. “S” is “k”, making **spark”**

The text is now full deciphered:



It should be noted that as **A**, **C**, **J**, and **T** do not appear at all in the ciphertext, I cannot say what they map to in the substitution alphabet. If this key were used on another message, I would need to deduce these few remaining letters.

#### 4. Solution for Cipher 2

blah blah blah

#### 5. Cipher 2 Cryptanalysis

blah blah blah

#### 6. Solution for Cipher 3

Plain Text:

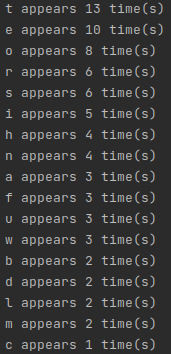
“THE WORST SIN TOWARDS OUR FELLOW CREATURES IS NOT TO HATE THEM BUT TO BE INDIFFERENT TO THEM [SI].”

(“SI” is just random characters at the end)

Key: **FDBGCAE**

#### 7. Cipher 3 Cryptanalysis

* First I performed a frequency analysis on the text and it displayed a distribution that was fairly typical for English. This suggested that this was a transposition cipher. I assume that this is rail fence cipher.



* The text has 77 characters, so if it is a rail fence, there should be words readable when reading in gaps of 7 or 11 (factors of 77). This gives the options “REOHSTW” or “RTLHERETTBT”. These words are clearly nonsense, so it is likely a Columnar Transposition Cipher
* The key length could either be 7 or 11. I initially assumed that the key length is 7. Hence there are 7 columns and 11 rows in the transposition box. I drew it by hand:
* //TODO scan here
* Reading along the top two rows, the text reads “REOHSTW | OITSWTN”. It is important to check both words, as the first word could be more than 7 characters long, meaning that it would spill over both rows.
* After a while trying anagrams, “REOHSTW” is an anagram of “THE WORS”
* The following row contains a “T” in column 5 (0-indexed), the same as the first character of row 1.
* Using this, the column order is: “5, 3, 1, 6, 2, 0, 4”, making “THE WORST SIN TOW”
* This seems promising. Using a script I wrote, the entire message decrypts to:

THEWORSTSINTOWARDSOURFELLOWCREATURESISNOTTOHATETHEMBUTTOBEINDIFFERENTTOTHEMSI

* It is hard to deduce the actual original key phrase, but to all intents and purposes “**FDBGCAE**” works.

#### 8. Appendix A

**MonoAlphabeticSubstitutionCipher:**

*This class handles encryption and decryption using a monoalphabetic substitution cipher*

public class MonoAlphabeticSubstitutionCipher extends Cryptor{

private HashMap<Character, Character> alphabet;

public void setParams(HashMap<Character, Character> alphabet){

setAlphabet(alphabet);

}

@Override

protected void encrypt() throws Exception {

char[] plainCharArr = getPlainText().toCharArray();

char[] cipherCharArr = new char[plainCharArr.length];

for(int i=0;i<plainCharArr.length;i++){

if(plainCharArr[i] > 64 && plainCharArr[i] < 133){

cipherCharArr[i] = translateP2C(plainCharArr[i]);

} else {

cipherCharArr[i] = plainCharArr[i];

}

}

setCipherText(String.*valueOf*(cipherCharArr));

}

@Override

protected void decrypt() throws Exception {

char[] cipherCharArr = getCipherText().toCharArray();

char[] plainCharArr = new char[cipherCharArr.length];

for(int i=0;i<cipherCharArr.length;i++){

if(cipherCharArr[i] > 64 && cipherCharArr[i] < 133){

plainCharArr[i] = translateC2P(cipherCharArr[i]);

} else {

plainCharArr[i] = cipherCharArr[i];

}

}

setPlainText(String.*valueOf*(plainCharArr));

}

private char translateP2C(char plain) {

return alphabet.get(plain);

}

private char translateC2P(char cipher) {

Iterator it = getAlphabet().entrySet().iterator();

while (it.hasNext()) {

Map.Entry pair = (Map.Entry)it.next();

if(pair.getKey().equals(cipher)){

return (char) pair.getValue();

}

//it.remove(); // avoids a ConcurrentModificationException [although no multithreading here so unreq'd]

}

return '#';

}

public HashMap<Character, Character> getAlphabet() {

return alphabet;

}

public void setAlphabet(HashMap<Character, Character> inputAlphabet) {

alphabet = new HashMap<>();

alphabet.put('A', 'A');

alphabet.put('B', 'B');

alphabet.put('C', 'C');

alphabet.put('D', 'D');

alphabet.put('E', 'E');

alphabet.put('F', 'F');

alphabet.put('G', 'G');

alphabet.put('H', 'H');

alphabet.put('I', 'I');

alphabet.put('J', 'J');

alphabet.put('K', 'K');

alphabet.put('L', 'L');

alphabet.put('M', 'M');

alphabet.put('N', 'N');

alphabet.put('O', 'O');

alphabet.put('P', 'P');

alphabet.put('Q', 'Q');

alphabet.put('R', 'R');

alphabet.put('S', 'S');

alphabet.put('T', 'T');

alphabet.put('U', 'U');

alphabet.put('V', 'V');

alphabet.put('W', 'W');

alphabet.put('X', 'X');

alphabet.put('Y', 'Y');

alphabet.put('Z', 'Z');

for (Map.Entry<Character, Character> mapping : inputAlphabet.entrySet()){

alphabet.put(mapping.getKey(), mapping.getValue());

}

}

public void printAlphabet() {

for (Map.Entry<Character, Character> mapping : alphabet.entrySet()){

System.*out*.println(mapping.getKey() + " -> " + mapping.getValue());

}

}

}

**MASHelper:**

*This class runs the cipher with a given alphabet. I use it to add letters that I think are correct, and it lets me quickly change the settings.*

public class MASHelper {

private MonoAlphabeticSubstitutionCipher MASCipher;

private HashMap<Character, Character> alphabet = new HashMap<>();

private String cipherText = "Gvo wuhxfhhulm qogvlw lp gozxvumn, pzuirb xlqqlm um gvo vfqzmuguoh, uh horwlq fhow um omnumooiumn; vldoeoi, ug xzm yo z eoib fhopfr hfkkroqomg um roxgfio xrzhhoh. Um xllkoizgueo nilfkh qlhg lp gvo rozimumn lxxfih dugv hgfwomgh dlisumn glnogvoi um hqzrr nilfkh. Gvuh qogvlw vzh yoom fhow pli gvo omguio xlfiho li zh z hfkkroqomg um roxgfio xrzhhoh. Z eziuogb lp lgvoi qogvlwh hfxv zh kzmorh li woyzgoh xzm yo fhow gl hkzis hgfwomg umgoiohg zmw omxlfizno hgfwomg umelreoqomg.";

public void run(){

MASCipher = new MonoAlphabeticSubstitutionCipher();

alphabet.put('O', 'e'); // freq analysis

alphabet.put('G', 't'); // freq analysis

alphabet.put('V', 'h'); // follows for "the"

//ASSUME alphabet.put('Z', 'i'); // standalone

alphabet.put('Z', 'a'); // standalone// "\_e i" => "be I", "me I", "we I" => None of these make sense, so

Z must be A

alphabet.put('Y', 'b'); // "\_e a" => "be a"

alphabet.put('M', 'n'); // "th\_\_ \_eth\_\_ ha\_ bee\_" => "has been" or "had been" (it's unlikely anything

else like "beef")

// "th\_\_", not "that" or "the\_" => must be "this"

alphabet.put('U', 'i');

alphabet.put('H', 's');

// "\_ebates \_an be" & "it \_an be a"

alphabet.put('X', 'c');

alphabet.put('W', 'd'); // could be rebates, but unlikely

// "disc\_ssi\_n" => "discussion"

alphabet.put('F', 'u');

alphabet.put('L', 'o');

// "\_ethod" => "method"

alphabet.put('Q', 'm');

// "oP teachin\_" => "of teaching"

alphabet.put('P', 'f');

alphabet.put('N', 'g');

// "fai\_\_\_ common" => "fairly common"

alphabet.put('I', 'r');

alphabet.put('R', 'l');

alphabet.put('B', 'y');

// "\_ery" => very

alphabet.put('E', 'v');

// "su\_\_lement" => "Supplement

alphabet.put('K', 'p');

// "\_ith"

alphabet.put('D', 'w');

// "wor\_ing"

alphabet.put('S', 'k');

MASCipher.setParams(alphabet);

System.*out*.println(MASCipher.decryptCall(cipherText));

System.*out*.println();

System.*out*.println();

MASCipher.printAlphabet();

}

public void runFreqAnalysis(){

HashMap<Character, Integer> freq = new HashMap<>(26);

String myCipherText=cipherText.replaceAll("\\s+","");

for(int i=0; i<myCipherText.length(); i++){

Character c = myCipherText.charAt(i) ;

Integer currentCount = freq.get(c);

if (currentCount != null) {

freq.put(c, new Integer(currentCount + 1));

}

else {

freq.put(c, 1);

}

}

Comparator<Map.Entry<Character, Integer>> valueComparator = new Comparator<Map.Entry<Character, Integer>>() {

@Override

public int compare(Map.Entry<Character, Integer> e1, Map.Entry<Character, Integer> e2) {

Integer v1 = e1.getValue();

Integer v2 = e2.getValue();

return v2.compareTo(v1);

}

};

List<Map.Entry<Character, Integer>> listOfEntries = new ArrayList<Map.Entry<Character, Integer>>(freq.entrySet());

// sorting HashMap by values using comparator

Collections.*sort*(listOfEntries, valueComparator);

System.*out*.println("here");

for(Map.Entry<Character, Integer> mapping : listOfEntries){

System.*out*.println(mapping.getKey() + " appears " + mapping.getValue() + " time(s)");

}

}

public static void main(String[] args) {

MASHelper MASHelper = new MASHelper();

MASHelper.runFreqAnalysis();

MASHelper.run();

}

}

#### 9. Appendix B

blah blah blah

#### 10. Appendix C

**ColumnarTranspositionCipher:**

*This class handles encryption and decryption using a columnar transposition cipher. Note that the encryption in this version of the code contains an unknown bug so does not work correctly; decryption is fine though.*

public class ColumnarTranspositionCipher extends Cryptor{

private String key;

public void setup(String key) {

this.key = key;

}

public Boolean checkKeyEquivalency(String key){

Integer[] argOrder = calculateIndexOrders(key);

Integer[] thisOrder = calculateIndexOrders(this.key);

return Arrays.*equals*(argOrder, thisOrder);

}

private Integer[] calculateIndexOrders(String key){

int[] intArray = IntStream.*rangeClosed*(63, 133).toArray();

int counter = 0;

Integer[] orders = new Integer[key.length()];

for(int charValue : intArray){ //go through each char from a onwards

for(int i = 0; i<key.length(); i++){

int keyChar = key.toCharArray()[i]; //check this val

if (charValue == keyChar){

orders[i] = counter;

counter++;

System.*out*.println("FOUND, cv="+charValue+" ("+(char) charValue+") "+"kc="+keyChar+"

("+(char) keyChar+")");

}

}

}

return orders;

}

@Override

//NOT fully working

protected void encrypt() throws Exception {

StringBuilder[] columns = new StringBuilder[key.length()];

for (int i = 0; i<key.length(); i++){

columns[i] = new StringBuilder();

}

ArrayList<Character> chars = new ArrayList<>(

getPlainText().chars()

.mapToObj(e -> (char) e)

.collect(

Collectors.*toList*()

)

);

Iterator<Character> charIt = chars.iterator();

int index = 0;

while (charIt.hasNext()){

Character thisChar = charIt.next();

String substring = thisChar.toString();

Pattern re = Pattern.*compile*("[a**-**zA**-**Z]+");

Matcher m = re.matcher(substring);

if (m.find()) {

columns[index].append(thisChar);

index++;

if (index >= key.length()) {

index = 0;

}

}

}

columns = padColumns(columns);

StringBuilder finalBuilder = new StringBuilder();

Integer[] indexOrders = calculateIndexOrders(key);

for(int sublistIndex : indexOrders){

finalBuilder.append(columns[sublistIndex]);

}

setCipherText(finalBuilder.toString());

}

private StringBuilder[] padColumns(StringBuilder[] columns) {

int maxLength = columns[0].length(); //The first column is always the longest (or joint-longest)

for (StringBuilder sb : columns){

if(sb.length()<maxLength){

sb.append(getRandChar());

}

}

return columns;

}

private char getRandChar() {

int randomNum = ThreadLocalRandom.*current*().nextInt(65, 91); //ASCII of a random char

return (char)randomNum;

}

// Unused, generates first N characters for key length = N

public String runFirstLines(String cipherText, int keyLen, int numLines){

String letters = "abcdefghijklmnoprstuvwxyz";

String key = letters.substring(0, keyLen);

String tempKey = this.key;

setup(key);

String result = decryptCall(cipherText).substring(0, keyLen\*numLines);

this.key = tempKey;

return result;

}

@Override

protected void decrypt() throws Exception {

int groupLen = (getCipherText().length() / key.length()); //floor div and add 1 (same as ceil div)

StringBuilder[] columns = new StringBuilder[key.length()];

for (int i = 0; i<key.length(); i++){

columns[i] = new StringBuilder();

}

Integer[] indexOrders = calculateIndexOrders(key);

StringBuilder finalBuilder = new StringBuilder();

for(int groupIndex=0; groupIndex<groupLen ; groupIndex++) {

for(int sublistIndex : indexOrders) {

try {

finalBuilder.append(getCipherText().toCharArray()[groupIndex + ((sublistIndex) \* groupLen)]);

} catch (Exception e) {

System.*out*.println("PASSING");

}

}

}

setPlainText(finalBuilder.toString());

}

}

**CTCRunner**

*This class handles runs the CTC with given settings*

public class MainTest {

public static void main(String[] args){

ColumnarTranspositionCipher columnarTranspositionCipher = new ColumnarTranspositionCipher();

columnarTranspositionCipher.setup("fdbgcae");

columnarTranspositionCipher.decryptCall("rouwrtttitseidlasabirhotoouoetdnmhsreeihmeetswrcethoftittafrs

oebfownsltntunee");

System.*out*.println(columnarTranspositionCipher.getCipherText());

}

}