The Legacy of Srinivasa Ramanujan

Devanagari Implementation Of Affine Cipher

B.Tech Innovation with Mathematics and IT *I year*
Cluster Innovation Centre,
3rd Floor, Rugby Sevens Building,Cavalry Lane
University of Delhi
Delhi 110007

Team Members

1. Akshay Khunteta
2. Nitin Agrawal
3. Prashant Sinha
4. Raghavendra Tripathi
5. Shivek Khurana

INDEX

Problem

Design and implement an Encryption/Decryption scheme based on affine cipher for plain text in Devanagari (UTF-8) communicating message in Hindi.

Develop a solver for this which solves /helps to solve the crypt when the encryption parameters (key) is not known.

Technologies

Python Programming Language

Approach

The software is called **RITI.**

RITI is based on global thinking. It can be used as a cipherer for any language, provided the language's script can be written in unicode format.

Throughout the project the focus has been on presenting information and comments in an easy and intelligible manner. The project is very useful for those who are working with crypt scheme specially for devanagari.

In Affine cipher, each letter in an alphabet is mapped to some numeric equivalent , and then using a mathematical function (ax+b) mod m (a,b are the crypt parameter), we get some unique mapping in numerals which is again converted to its alphabetical equivalent.
in affine each letter of an alphabet of size m is associated with a number in range (0,m-1). And using the formula it is encrypted ,where in the formula 'a' is the multiplier and 'b' is the shift of magnitude. in this definition 'a ' is always chosen to be co-prime with m.

Numerical Methods Used:

i. Euclid's algorithm for greatest common divisor computation
ii. Extended Euclidean Algorithm for inverse computation
iii. Statistics and Probability for frequency analysis and listing of elements in the corpus

Usage

The software on the ground level operates using objects of a class explicitly called 'A'. This class converts unicode into programmable symbols which are used throughout the software.

FEATURE 1

Convert all inputs into programmable symbols.

original = Affine('नमस्ते दुनिया ', 'devnagri.txt')
Here 'नमस्ते दुनिया ' is the text to encrypt(or decrypt), and devnagri.txt is a file containing the script of the language

Affine() explicitly converts 'नमस्ते दुनिया' into programmable symbols.

FEATURE 2

To encrypt the message:
encrypt = original.encrypt(5,5)
encrypt.read() would output 'क्उबँ६ ऋबकेसझ'

To decrypt a message:
original = Affine('क्उबँ६ ऋबकेसझा ', 'devnagri.txt')
decrypt = original.decrypt(5,5)
decrypt.read() would output 'नमस्ते दुनिया '

FEATURE 3

To break the cipher text:
original = Affine('जहाँ महनत वहीं वजिय ', 'devnagri.txt')
decipher = original.break_affine()
This gives the user 3 possibilities:

1. **Break affine using frequency analysis** *[Ideal Case]*
   By statistics of hindi language, we know that the frequency of letter अ क highest and second highest respectively.
   This code matches two most occurring characters in the cipher to अ क and hence determine a,b

2. **Break affine using Brute Force** *[Real Case]*
   This method generates all possibilities of a,b and decrypt the cipher, then match the decryption against a predefined corpus (of ~100k hindi words in our case) and gives out an optimal solution.

Corpus : A word list of about 100k words in decreasing order of frequency

Name : hindi_wordlist_top_100k.txt taken from www.http://corpora.heliohost.org/

3. **Break affine by giving out all possibilities***[Worst Case]*

This method gives out all possibilities and lets the user decide what the encrypted text maybe.

Class Definitions

`class A :`

This class is the spinal cord of *RITI philosophy.* In resonance with the context, we call this class a बेहतरीन way of solving this particular problem. This class converts devanagari symbols into programmable symbols. नमस्ते becomes `['DEVANAGARI LETTER NA', 'DEVANAGARI LETTER MA', 'DEVANAGARI LETTER SA', 'DEVANAGARI SIGN VIRAMA', 'DEVANAGARI LETTER TA', 'DEVANAGARI VOWEL SIGN E']`

This class also gives the flexibility to extend this en|decrypter to any language. 'Hello' in chinese i.e. 你好 becomes `['CJK UNIFIED IDEOGRAPH-4F60', 'CJK UNIFIED IDEOGRAPH-597D']` and remaining procedures can be applied as usual.

This class has following methods

`read()`

This function return the unicode format of the characters

`frequencify(language)`

Lists all the alphabets of string arranged in decreasing order of occurrence.
If an element has same occurrence, then the element with lower index in script i.e the one which comes first is placed first (ex a is placed before b)
Argument "language" takes the corresponding language's characters set as a string or a text file containing the characters set

`class Affine:`

This class has the following methods:

`encrypt(a,b)`

Encrypt given string on the basis of a,b. Using concepts of basic shift cipher.

`gcd(a, b)`

Calculates greatest common divisor of a,b using Euclidean algorithm.

`get_inverse(b,a)`

Calculates modular inverse of a in modulo b

`crack(p,q,r,s,m)`

Computes (a,b) key set of affine where 'b' is magnitude of the shift of affine's key and 'a' is multiplier for given p,q,r,s,m such that [ap+b=r (mod m)], [aq+b=s (mod m)] where 'p' and 'q' are plaintext characters and 'r' and 's' are their corresponding encrypted forms. This is primarily used as adhoc guess on the basis of frequency is made about the two plaintext characters and their encrypted forms

`decrypt(a,b)`

Decrypt the given string on the basis of given a,b.

`break_affine()`

Try breaking the affine cipher in three ways as defined above

`possible_keys()`

Generate all possible pairs a,b for brute force procedure

`validate()`

Validate the decipher generated by break_affine() against a predefined corpus

`break_affine_frequency( f_table)`
`break_affine(self,accuracy=1)`
`break_affine_manually()`

These functions are code synonyms for three methods of breaking the cipher as define in FEATURE 3

NOTE : The methods used for creation of GUI is not included herein.

Complexity Analysis

`encrypt(a,b)` $O(n)$
`gcd(a, b)` $O(\log n)$
`get_inverse(b,a)` $O(\log^2 n)$
`crack(p,q,r,s,m)` $O(\log^2 n)$
`decrypt(a,b)` $O(n)$
`break_affine()` $O(n^3)$ **HIGHEST**
`possible_keys()` $O(n)$
`break_affine_frequency(f_table)` $O(n \log n)$
`break_affine_manually()` $O(n^3)$
`validate()` $O(n)$

In principle:

1. Encryption : $O(n)$ Linear
2. Decryption : $O(n)$ Linear
3. Breaking(without the key)
   a. Using Frequency Analysis : $O(n \log n)$

b. Using Brute Force : $O(n^3)$ Polynomial
c. Using Manual Inspection : $O(n^3)$ Polynomial

Platform, System, Memory requirements

The Code is tested on the following platforms :

a. Windows 7, intel core i7 2.7Ghz, 4gb memory, 64bit
b. Ubuntu Linux, intel core2duo 2.53Ghz, 2gb memory, 64bit

This software requires python IDLE to execute.
Instructions are given in file run_instructions.txt

Test Vectors

Encryption :

encrypt रविवार बक्रिम नाम हो
at (a,b) = 5,7
ुऋैऋुँऔमुॐ गञॐ ऐ

encrypt भारत सरकार जदिाबाद
at (a,b) = 5,7
रुिः ऋुऔुषैऑऍँिएँ

encrypt वशि्व वद्यिालय सावधान
at (a,b) = 3,15
ग़ऐढ़सग़ ग़ऐथसूत्रो्ऋ़नऋफ

encrypt सूरज चाँद सतिारे
at (a,b) = 1,3
ऊॅळझु न ऽढधुळ४

encrypt घर कब आओगे बताओ मुझे
at (a,b) = 3,7
ंडैफ ङू८ञ फगू: ऱबंञ

Decryption:

decrypt ंडैफ ङू८ञ फगू: ऱबंञ
at (a,b) = 3,7

घर कब आओगे बताओ मुझे

decrypt खनृ नृि हनृ लनृ छ॔ह्

at (a,b) = 1,17

अगर मगर नगर डगर पानी

decrypt छब् थअ४ ४अर्थ ४ घ४ छ४अॅ

at (a,b) = 3,9

आँख नाक कान शकल अकल आकाल

decrypt ंअळँअळे र्थिअ ॠण

at (a,b) = 3,9

शाम जाम औद नशा है

decrypt ेछे ओथेछ बऽछ वथेछ चपेछु चपेछुथ

at (a,b) = 5,3

लाल नीला हरा पीला ग्लाब ग्लाबी

Breaking (with default accuracy):

Using Frequency Analysis

Encrypted Text :ूजजज

Deciphered Text (without key) : हहहहहहहहहहककक (with highest frequency guess of “ह’,”क”

Corresponding key retrieved :1,7

Using Brute Force:

Encrypted Text : मे थीधजमऽ ख४ऱ से

Deciphered Text(without the key) :ऐ दुनिए गोल है

Corresponding key retrieved :1,91

Encrypted Text : ऋघऽॉऔ ऽण८ठॉ एॅन्तम ऽज़णॉ

Deciphered Text (without the key) :इन्साफ सरयिा पलन्त सूरा

Corresponding key retrieved :3,75

Encrypted Text :ऋॅऋडबक झथकड बकर्रचणक बहफ़ऐॅप

Deciphered Text (without the key) :गंगाधर सूरा धर्मवीर धड़कनथ

Corresponding key retrieved :1,65

Encrypted Text : ड़ि ओ ॅन ब िजबः ढ़ंकओॅन

Deciphered Text (without the key) :डाकघर मे घन्तडा हन्तठ खलिाये

Corresponding key retrieved :1,63

Encrypted Text :ष्ू ्ू द ष्ूअिं च डबळ्ूष ग़द ४ळ्ू
Deciphered Text (without the key) :राजा को रानी से पःयार हो गया
Corresponding key retrieved :3,3

WORST CASE SCENARIO (Breaking Ciphers) using Brute Force

break णछफ्ुवि णाषफिख
to get सूरज चाँद सतिारे key (91,91)
in 11min 49 seconds on corei7 processor with 4GB Ram

Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#coding=utf-8

import unicodedata as ucd

class A:
  '''
 Object to hold a list of unicode equivalents of a string in any language
 in the form of a list()
 '''
  def __init__(self, string='hindi_alphabets.txt'):
    '''
   Converts content of a file or raw string into a list of Alphabet objects

   @param string
     string is either raw text or a filename
     string defaults to input file
   '''
    try:
      string = file(string).read() #try reading a file
    except IOError as e:
      #raw input given
      pass
    string = string.lower() #fix english, safe otherwise
    try:
      self.alphabets = string.decode('utf-8')#a list of all alphabets in required language
    except UnicodeEncodeError,e:
      #perhaps already unicode (as sent by Affine.encrypt)
```

```
      self.alphabets = string
    alphabets_fixed = list()
    for a in self.alphabets:
      if(len(a) > 0):
        #take pain iff the element exists
        unicoded = ucd.name(a, 0) #default to 0
        if(unicoded != 0):
          #add alphabet to cluster only if it is recognized
          alphabets_fixed.append(unicoded)
    self.alphabets = alphabets_fixed
    return None

  def read(self):
    '''
   'A' object in traditional form
   @param None
   @return string
   '''
    read = ''
    for a in self.alphabets:
      read += ucd.lookup(a)
    return read

  def frequencify(self, language):
    '''
   List of alphabets of string arranged in decreasing order of occurence.
   If an element has same occurance, then the element with lower index in script
   i.e the one which comes first is placed first (ex a is placed before b)

   @param 'A' object for language to which the string needs to be mapped
   @return list
   '''
    frequencified = {} #a dict to store occurences of a alphabet in unicode equivalent
    for alphabet in self.alphabets:
      if(frequencified.has_key(alphabet)):
        frequencified[alphabet] += 1 #if already occured +1 to occurence
      else:
        frequencified[alphabet] = 1 #never occured before
    s = list()

    for w in sorted(frequencified, key=lambda x:frequencified[x], reverse=True):
      if(w != 'SPACE' and (w not in s)):
        s.append(w)
    return s

class Affine:
  '''
 All (en|de)crypt methods
 '''
  def __init__(self, string, language):
    self.string = A(string) #alphabet object to (en|de)crypt
```

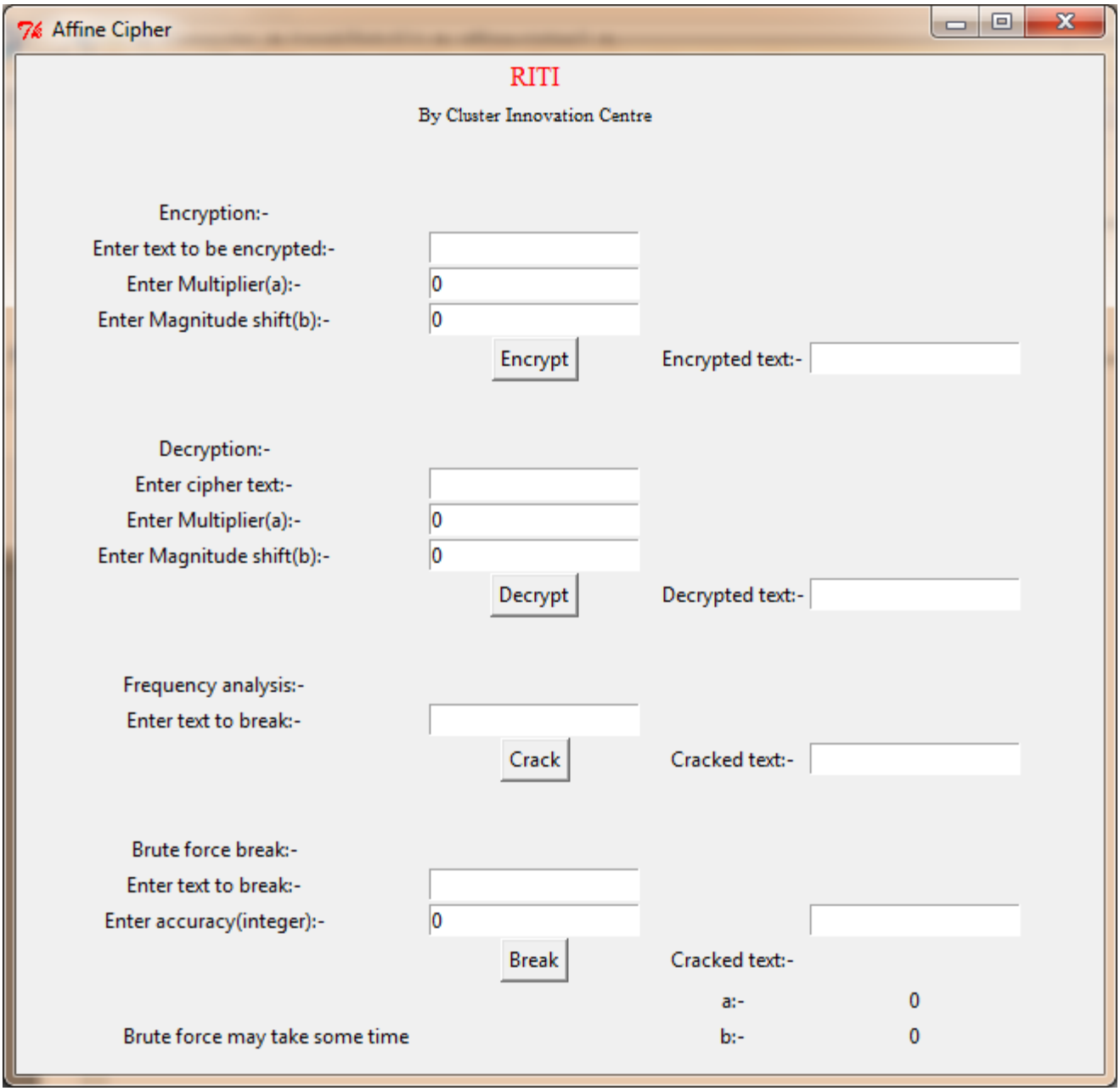
```
81.     self.language = A(language) #alphabet object of the script of language being encoded or
    decoded (in alphabetic order)
82.     self.m = len(self.language.alphabets)
83.
84.   def gcd(self,a, b):
85.     '''
86.    Calculate greatest common dvisior of a,b. Eucledian Algorithm.
87.    @param : a,b
88.    @return : GCD of a,b
89.    '''
90.     while(b!=0):
91.        t= b
92.        b= a%b
93.        a=t
94.     return a
95.
96.   def encrypt(self, a, b):
97.     '''
98.    Encrypt string on the basis of give a,b
99.    y = (ax + b)%(m)
100.
101.    @param int a
102.    @param int b
103.    @return 'A' object
104.    '''
105.     crypt = ''
106.     m = self.m
107.     if(self.gcd(m, a) == 1):
108.     #check a, m are relatively prime
109.       for alphabet in self.string.alphabets:
110.         if(alphabet == 'SPACE'):
111.           crypt+=' '
112.           continue
113.         x = self.language.alphabets.index(alphabet)
114.         y = (a*x + b)%m
115.         crypt += ucd.lookup(self.language.alphabets[y])
116.       return A(crypt)
117.     else:
118.       raise ValueError("Expected co primes")
119.
120.   def get_inverse(self,b,a): ##programme for calculating modular inverse of a in [ax=1 mod
    b]
121.     r=[a,b]
122.     t=[0,1]
123.     s=[1,0]
124.     q=[0]
125.     m= 1
126.
127.     while(r[m]!=0):
128.
129.         mm1=m-1
130.         mp1=m+1
```

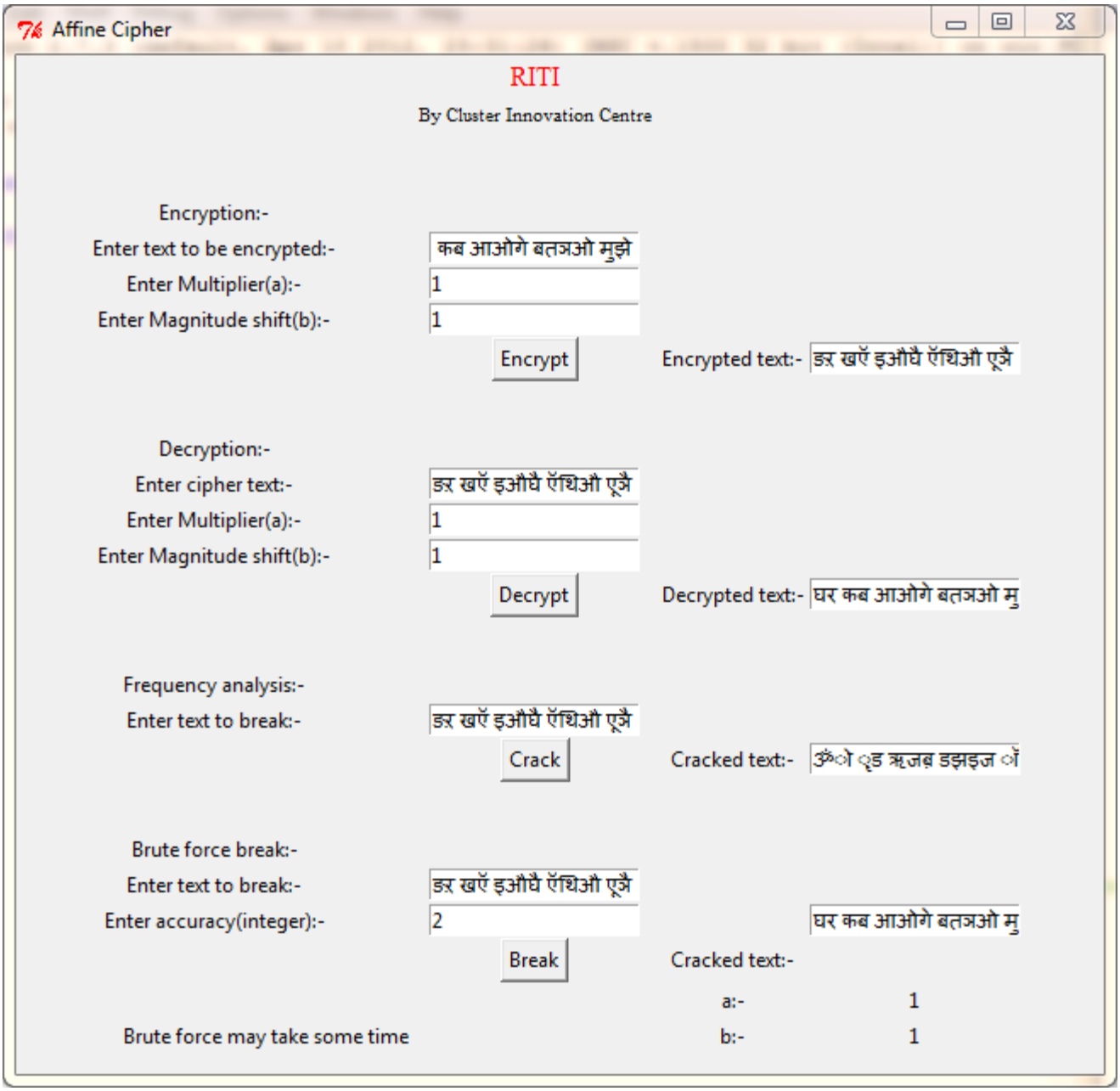
```
        q.append(r[mm1]/r[m])

        r.append(r[mm1]-(q[m]*r[m]))
        t.append(t[mm1]-(q[m]*t[m]))
        s.append(s[mm1]-(q[m]*s[m]))
        m= m+1
    m=m-1
    return s[m]

  def decrypt(self, a, b):
    '''
    Decrypt string on the basis of given a,b
    y = a^(-1)(x-b)%m
    a^(-1) is z such that:
      az = 1%m

    @param int a
    @param int b
    @return 'A' object or None if inverse doesn't exist
    '''
    m = self.m
    try:
      z = self.get_inverse(m,a)
    except:
      return None
    crypt = ''
    for alphabet in self.string.alphabets:
      if(alphabet == 'SPACE'):
        crypt += ' '
        continue
      x = self.language.alphabets.index(alphabet)
      y = z*(x-b)%m
      crypt += ucd.lookup(self.language.alphabets[y])
    return A(crypt)

  def crack(self,p,q,r,s,m):##computes key's B and displacys the key set(a,b) for given
    p,q,r,s,m such than [ap+b=r (mod m)],[aq+b=s (mod m)]
    D=p-q
    D1=r-s
    if(self.gcd(D,m)==1):## equation are subtracted and converted to Da=D1 (mod m)
        k_a=(self.get_inverse(D,m)*D1)%m
    else:
        d=self.gcd(D,m)
        D=D/d
        D1=D1/d
        m=m/d
        k_a=(self.get_inverse(D,m)*D1)%m
    k_b=(r-p*k_a)%m
    return k_a,k_b

  def possible_keys(self):
    '''
```

```
    Generate all possible keys depending on the value of m for
    brute force cryptanalysis.
    '''
     m = self.m
     possible_keys=list()
     for i in range(1,m+1):
       if(self.gcd(i,m)==1):
         possible_keys.append(i)
     return possible_keys

  def validate(self,observation,accuracy,corpus="hindi_corpus.txt"):
     '''
    Validate brutely cryptanalised A object against a corpus.

    @param observation : A object to analyse
    @param corpus : text file to analyse against
    @return bool:
      True if one word matches.
      False otherwise
    '''
     corpus = file(corpus).read().split('\n')
     o_list = observation.read().split(' ')
     matches=0

     for word in corpus:
       word = word.decode('utf-8')
       for word_o in o_list:
         if len(word_o) == 0: continue
         if word_o == word:
           matches += 1
         if matches > accuracy:
           return observation
     return None

  def break_affine_frequency(self, f_table):
     '''
    Match two most occuring elements in langauage to two most occuring
    elements in the given string. Ask user if the return is apt. If not
    repeat it with next most occuring elements.

    @param raw_string or text file
    @return
    '''
     most_occuring = A(f_table)
     ordered_string = self.string.frequencify(self.language) #sorted list of alphabets
  arranged in decreasing order of occurance
     p = self.language.alphabets.index(most_occuring.alphabets[0])
     q = self.language.alphabets.index(most_occuring.alphabets[1])
     r = self.language.alphabets.index(ordered_string[0])
     s = self.language.alphabets.index(ordered_string[1])
     a,b=self.crack(p,q,r,s,self.m)
     decrypt = self.decrypt(a,b)
```

```
233.     return decrypt
234.
235.   def break_affine(self,accuracy=2):
236.     '''
237.    Match two most occuring elements in langauage to two most occuring
238.    elements in the given string. Ask user if the return is apt. If not
239.    repeat it with next most occuring elements.
240.
241.    @param raw_string or text file
242.    @return A(object) or None
243.    '''
244.     if len(self.string.read().split(' ')) <3:accuracy = 1
245.     for i in self.possible_keys():
246.       print "Breaking at mulitplier %s"%i
247.       for j in range(self.m):
248.         decrypt = self.decrypt(i,j)
249.         if decrypt != None:
250.           if self.validate(decrypt,accuracy) != None:
251.             print i,j," done"
252.             return decrypt,a,b
253.       print "Fail"
254.     return None
255.
256.   def break_affine_manually(self):
257.     '''
258.    Try all keys.
259.    '''
260.     for i in self.possible_keys():
261.       for j in range(self.m):
262.         print self.decrypt(i,j).read()
263.         print i,j
```

Screenshots


Affine Cipher
RITI
By Cluster Innovation Centre
Encryption:-
Enter text to be encrypted:-
Enter Multiplier(a):-
0
Enter Magnitude shift(b):-
0
Encrypt
Encrypted text:-
Decryption:-
Enter cipher text:-
Enter Multiplier(a):-
0
Enter Magnitude shift(b):-
0
Decrypt
Decrypted text:-
Frequency analysis:-
Enter text to break:-
Crack
Cracked text:-
Brute force break:-
Enter text to break:-
Enter accuracy(integer):-
0
Break
Cracked text:-
a:-
0
Brute force may take some time
b:-
0


Affine Cipher
RITI
By Cluster Innovation Centre
Encryption:-
Enter text to be encrypted:-
कब आओगे बतअओ मुझे
Enter Multiplier(a):-
1
Enter Magnitude shift(b):-
1
Encrypt
Encrypted text:-
डऱ खऍ इऔघै ऍथिऔ एूअै
Decryption:-
Enter cipher text:-
डऱ खऍ इऔघै ऍथिऔ एूअै
Enter Multiplier(a):-
1
Enter Magnitude shift(b):-
1
Decrypt
Decrypted text:-
घर कब आओगे बतअओ मु
Frequency analysis:-
Enter text to break:-
डऱ खऍ इऔघै ऍथिऔ एूअै
Crack
Cracked text:-
ॐो ृड ऋजब़ डझइज ों
Brute force break:-
Enter text to break:-
डऱ खऍ इऔघै ऍथिऔ एूअै
Enter accuracy(integer):-
2
घर कब आओगे बतअओ मु
Break
Cracked text:-
a:-
1
Brute force may take some time
b:-
1