Assignment 1

2016310932 배현웅

1. Overview

   A. Vigenere varint cipher Decryption is divided into two tasks. First task is to know the length of key. Second task is to distinguish each key value.

   B. To identify the length of key, I apply the statistics which alphabet distribution is not kind of uniform distribution. As the lecture note said, I use sigma of probabilities square. ($\sum p^2$)

   C. After we find the key length, we also check distribution of Nth letter. This time, I multiply frequency of alphabet included 'space' and the probability of each alphabet which I find distribution of Nth letter.

2. Code description

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define SQUARE(x) ((x)*(x))

typedef struct twin{
    int max;
    int nu;
}twin;
char alpha[27] = {" etaonisrhlducmfwgpybvkxjqz"};
double percent[27] = {18.31, 12.60, 9.22, 8.01, 7.56, 7.00, 6.95, 6.52, 6.12,
    6.11, 4.07, 4.03, 2.79, 2.74, 2.49, 2.43, 2.09, 1.99, 1.85, 1.75, 1.54,
    0.98, 0.69, 0.17, 0.12, 0.10, 0.06};

int length_of_key;
int main(){
    unsigned char ch;
    double max =0;
    length_of_key=0;
    double tmp_sum =0;
    FILE *fpIn, *fpOut;
    char buf[5001];
```

This code consists of initiation of value, such as length_of_key, percent and alpha which stands for the frequency of alphabet. Alpha[7] is corresponds to percent[7]. ('e' : 12.60% frequency)

Char buf value is for input text whose name is hw1_input.txt

```
27      fpIn = fopen("hw1_input.txt","r");
28      fseek(fpIn,0,SEEK_END);
29      int buf_len = ftell(fpIn);
30      fseek(fpIn,0,SEEK_SET);
31      fread(buf,buf_len, sizeof(char), fpIn);
32      fclose(fpIn);
33
34      int count[260]={0, };
35      memset(count,0,sizeof(int)*260);
36      for(int i=1; i<=10; ++i){
37          tmp_sum =0;
38
39          memset(count,0,sizeof(int)*260);
40          int j;
41          for(j=1; i*j<buf_len; j++){
42              int tmp = (int)buf[i*j];
43              count[tmp]++;
44          }
45
46          for(int k=65; k<=90; ++k){
47              tmp_sum += SQUARE((double)count[k])/SQUARE((double)j);
48          }
49          for(int k=97; k<=122; ++k){
50              tmp_sum += SQUARE((double)count[k])/SQUARE((double)j);
51          }
52          if(max < tmp_sum){
53                  max = tmp_sum;
54                  length_of_key = i;
55          }
56      }
```

In line 27-32, I check file length by fseek and ftell incase of bumping into EOF value(in ASCII code 0x1A)

In line 36, to check possible length which is 1 to 10, I write for loop.

Line 41-44 : traverse all buf element(input context), then save frequency of each asci code into count array.

Line 46-51 : square of probability of each alphabet (lower case : 65(a)~90(z), capital : 97(A)~122(Z) is added into tmp_sum

Line 52-54 : to compare value of each case (depends on key length) I take sigma of probabilities square. ($\sum p^2$) value and find the highest value among them. At that value, the highest possible key length is it.

```
61      unsigned char key[length_of_key];
62      memset(key,0,sizeof(unsigned char)*length_of_key);
63  FIND_KEY :
64      // find the key value
65      for(int k=0; k<length_of_key; ++k){
66          int counts[260];
67          memset(counts,0,260*sizeof(int));
68          // length_of_key * i + k
69          for(int i=0; length_of_key*i+k<=buf_len; i++){
70              int pos = length_of_key*i+k;
71              counts[buf[pos]] = counts[buf[pos]]+1;
72          }
73          twin max_cnt,submax_cnt;
74          max_cnt.max =0;
75          submax_cnt.max = 0;
```

After we find the key length, as I comment on code, we should find the key value

Line 63 is label for correction of key length. I explain it later.

Line 65-72 : I take Nth letter in input text, and do similar thing as I find key length. However, this time I assume the repeated letter is 'space' which statistically occur often most

```
76          for(int i=0; i<=0xFF; ++i){
77              if(max_cnt.max < counts[i]){
78                  submax_cnt.max = max_cnt.max;
79                  submax_cnt.nu = max_cnt.nu;
80                  max_cnt.max = counts[i];
81                  max_cnt.nu = i;
82              } else if(submax_cnt.max < counts[i]){
83                  submax_cnt.max = counts[i];
84                  submax_cnt.nu = i;
85              }
86          }
87          if(max_cnt.max*0.1 + submax_cnt.max > max_cnt.max){
88              if(length_of_key *2 <= 10){
89                  length_of_key *=2;
90
91                  goto FIND_KEY;
92              }
93          }
94          int ke = max_cnt.nu^' ';
95          int ke1 = submax_cnt.nu^' ';
96          key[k] = ke;
```

Line 76 : we find most frequent letter(=max) and 2nd frequent letter(=submax) which are candidate

of ‘space’

Line 87 – 93 : there is little difference b/w submax and max. There is unfolded key stream. So multiply 2, then do FIND_KEY work again (ex key : 0x01 0x02 0x01 0x03)

Line 94-95 save to ke and ke1 value which are candidate of key

```
97          if(ke != ke1){
98              double prob1 = 0;
99              double prob2 = 0;
100             prob1 = percent[0]*max_cnt.max;
101             prob2 = percent[0]*submax_cnt.max;
102             for(int i=1; i<27; ++i){
103                 int diff = (int)(alpha[i]-' ');
104                 int pos1 = (max_cnt.nu + diff)%128;
105                 int pos2 = (submax_cnt.nu + diff)%128;
106                 prob1 += percent[i] * counts[pos1];
107                 prob2 += percent[i] * counts[pos2];
108             }
109             if(prob2 > prob1){
110                 key[k] = ke1;
111             } //else  printf("ke : %x\n",ke);
112         }
113     }
```

If two key candidate is different, I compare statistic values which is supposed to be ke or ke1is space

Compare sigma of alphabet probabilities square. ($\sum p^2$) which include ‘space’

```
114     // periodic check
115     for(int i=1; i<length_of_key; ++i){
116         int same=0;
117         if(i == 1){
118             for(int j=0; j<length_of_key; j++){
119                 if(key[j] == key[j+1]) same++;
120                 else break;
121             }
122             if(same == (length_of_key-1)){
123                 length_of_key = 1;
124                 break; // exit the period test
125             } else same =0;
```

If there is repeated periodic value, we reduced to optimized way

(ex key : 0x01 0x02 0x01 0x02 -> 0x01 0x02))

So, I Check all possible value which is lower than length_of_key

Line 117-125 : Test whether the key length is one

```
126         }else if(length_of_key%i == 0){
127             int period = length_of_key/i;
128             int okay =0;
129             if(key[0] == key[i]){
130                 //period 4 -> need to check 3 times
131                 for(int iter=0; iter<(period-1); ++iter){
132                     same=0;
133                     for(int j=0; j<i; ++j){
134                         if(key[iter*i+j] == key[(iter+1)*i+j]) same++;
135                         else break;
136                     }
137                     if(same == i) okay++;
138                     else break;
139                 }
140                 if((okay+1) == period) length_of_key = i;
141             }
142 
143         } // end else if(length_of_key%i == 0){
144     } // end for(int i=1; i<length_of_key; ++i){
```

If key length is not one, then test possible key length which is aliquot(약수) of key length

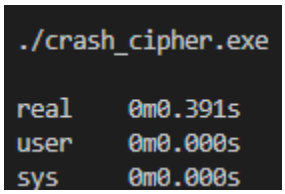
```
153     fpOut = fopen("hw1_output.txt","wb");
154     // for file write
155     fprintf(fpOut,"%#x",key[0]);
156     for(int i=1; i<length_of_key; ++i){
157         fprintf(fpOut," %#.2x",key[i]);
158     }
159     fprintf(fpOut, "\n");
160     //printf("%s\n",buf);
161     for(int i=0; i<buf_len; ++i){
162         ch = buf[i]^key[i%length_of_key];
163         fwrite(&ch, sizeof(ch), 1, fpOut);
164     }
165 
166     fclose(fpOut);
167     return 0;
168 }
```

After find the key value and key length, write down to output value.

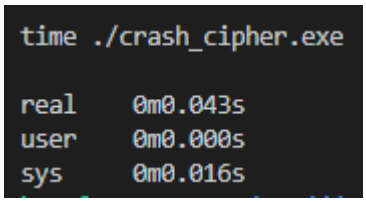
3. Performance Check

I Test time command two text which is encrypted by key length 10 for performance check

- Text which consists of 1485 letters



- Text which consists of 4775 letters



Worst case for my code runtime is about 0.391 seconds