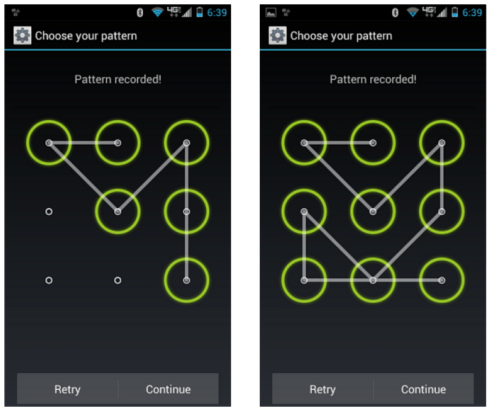
CyberChallenge.IT 2018 - Programming
Commented solutions

Contents

1 Problem "Pattern"

1.1 Problem Statement

Some popular smartphones offer authentication mechanisms based on gestures where previously recorded patterns have to be reproduced to unlock the phone.
The picture on the right shows pattern locks, i.e., sequences of moves, of length 5 (left) and 9 (right). You are to write a program that computes the number of possible pattern locks of length up to K that can be realized on a 3×3 grid.
Note that in a 3×3 grid there are 40 patterns of length 1, 240 of length 1 or 2, and 1192 of length 1, 2, or 3. Let your program do the math for larger values of K.



1.2 Problem Details

Input

Your program must read the input data from the standard input.
The input consists in only one line containing the integer **K**, representing the maximum length of a pattern lock.

Output

Your program must write the output data into the standard output.
The output must contains only one integer, representing the number of pattern locks of length up to K.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- **Subcase 1** [**50 points**]: $K \leq 6$.
- **Subcase 2** [**50 points**]: $K \leq 12$.
- $1 \leq K \leq 12$.

1.3 Solution

This is a simple problem that requires some confidence with recursive programming.

Subcase 1 (50 points)

A first solution can be to explore all the path recursively: for each starting point, we call a function with arguments the starting point of the pattern and the current length of the pattern. Each time, the same function is called with the next point to be considered and the incremented length, which corresponds to the same problem, but with decreased maximum length. This approach is clearly exponential in the length of the pattern and, given that the average number of recursion branches is about 4.5, it is not feasible for high values of K.

Subcase 2 (50 points)

The previous solution can be optimized if we notice that the four points at the corners are equivalent, and so are the four points at the center of each side. This solution is still exponential in K, but reduces the average number of branches to about 2.5, which is feasible for the given constraint on K.
Note that it is possible to solve the problem with some mathematical tools and obtain a closed formula that allows to compute the answer in almost constant time.

1.4 Source Code

C++

```cpp
#include <iostream>

int conta = 0;
int N;

int simula(int k, int l)
{
  if(l > N) return 0;

  if(k == 1 || k == 3 || k == 7 || k == 9)
    return 2*simula(2, l+1)+1*simula(5, l+1)+(l > 0);
  else if(k == 2 || k == 4 || k == 6 || k == 8)
    return 2*simula(1, l+1)+1*simula(5, l+1)+2*simula(4, l+1)+(l > 0);
  else
    return 4*simula(1, l+1)+4*simula(4, l+1)+(l > 0);
}

int main()
{
  std::cin >> N;
  for(int i=1; i<10; i++) conta += simula(i, 0);
  std::cout << conta << std::endl;
  return 0;
}
```

Python

```python
def simula(k, l):
if l > N:
    return 0

if k == 1 or k == 3 or k == 7 or k == 9:
    return 2*simula(2, l+1)+1*simula(5, l+1)+int(l > 0)
elif k == 2 or k == 4 or k == 6 or k == 8:
    return 2*simula(1, l+1)+1*simula(5, l+1)+2*simula(4, l+1)+int(l > 0);
else:
    return 4*simula(1, l+1)+4*simula(4, l+1)+(l > 0);

N = int(input())
conta = 0
for i in range(1,10):
conta += simula(i, 0)
print(conta)
```

2 Problem "Decrypt"

2.1 Problem Statement

Alice sends Bob a secret message, composed of n bytes indexed with an integer ranging from 0 to $n-1$.
She uses a simple encryption scheme that replaces every byte x of index i with a new value y, obtained as a right circular shift of x by (i mod 8) positions, where (i mod 8) denotes the remainder of the integer division of i by 8 (i % 8 in C).
For instance, if the byte x of index 2 of the message contains the character 'a' (hex 61 in ASCII, binary 01100001), the Alice's encryption replaces x with y obtained as right circular shift of 2 positions, that is, 01011000, hex 58.
You are to help Bob decrypt Alice's message. Write a program that takes as input an encrypted text and produces as output the decrypted text by reversing Alice's scheme as follows:
Byte y with index i in the encrypted input text is replaced by byte x, where x is obtained by a left circular shift of the bits of y by i mod 8 positions.
Example:
Encrypted input bytes (hex): 61 B0 58 2C 16 0B 85 C2 61
Decrypted output text (ascii): "aaaaaaaaa"

		input y (data)		**output (x) data**			
i	**i % 8**	**hex**	**binary**	**char**	**dec**	**hex**	**binary**
0	0	61	01100001	a	97	61	01100001
1	1	b0	10110000	a	97	61	01100001
2	2	58	01011000	a	97	61	01100001
3	3	2c	00101100	a	97	61	01100001
4	4	16	00010110	a	97	61	01100001
5	5	0b	00001011	a	97	61	01100001
6	6	85	10000101	a	97	61	01100001
7	7	c2	11000010	a	97	61	01100001
8	0	61	01100001	a	97	61	01100001

2.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains the integer **n**, representing the number of bytes of the secret message.
The second line contains **n** hex bytes, separated by space, representing the encrypted message.

Output

Your program must write the output data into the standard output.
The output must contain the decrypted text (ASCII encoded).

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- $1 \leq n \leq 1024$.
- All the input hex bytes are well formed (two digits, lowercase, from 00 to **ff**).

2.3 Solution

This problem is the simplest one in the problem set. It requires the capability to manipulate data at the bit level.
The first step is to decode the message from hexadecimal encoding, then the decryption is applied. Since the encryption performs a bitwise rotation to the right of i positions, the decryption should perform a bitwise rotation to the left of i positions (or, equivalently, a bitwise rotation to the right of $8-i$ positions).

2.4 Source Code

C++

```cpp
#include <bits/stdc++.h>

int getval(char c){ return ('0' <= c && c <= '9') ? int(c - '0') : int(10 + (c-'a')); }
int hex2dec(std::string hx){ return getval(hx[0])*16 + getval(hx[1]); }
std::string rotate(std::string b, int p){ return b.substr(p) + b.substr(0, p); }

int main()
{
  int n;
  std::cin >> n;
  for(int i=0; i<n; i++)
  {
          std::string hx;
          std::cin >> hx;
          std::string bin = std::bitset<8>(hex2dec(hx)).to_string();
    bin = rotate(bin, i % 8);
    char c = std::bitset<8>(bin).to_ulong();
    std::cout << c;
  }
  std::cout << std::endl;
  return 0;
}
```

Python

```python
#!/bin/env python3

def rot(x, pos):
  a = bin(x)[2:].zfill(8)
  return int(a[pos:] + a[0:pos], 2)

n = int(input())
b = input().split(" ")

for i in range(n):
  b[i] = int(b[i], 16)
  b[i] = chr(rot(b[i], i % 8))

print("".join(b))
```

3 Problem "Breakme"

3.1 Problem Statement

One of the simplest and most ancient encryption schemes is the ROT-k cipher, which works by replacing each letter in a text with another letter that is k positions away in the alphabet, wrapping around if k leads to a character past the end of the alphabet. For k=3, we have:
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
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

For instance, the ROT-3 encoding of "HELLO" is "KHOOR". According to Suetonius 1 , ROT-3 was used by Caesar to communicate secret messages to Cicero and to his own relatives back in the days of the Roman republic. Unfortunately, ROT-k ciphers are easy to break if one knows in which language the original text was written. This can be done as follows:

- for each possible value of k (e.g., from 0 to 25 for the English alphabet) decrypt the input text assuming it was encrypted with ROT-k, obtaining a version text k . Clearly, only one version will be equal to the original text. We just don't know which one yet.
- compute the k that minimizes the following formula (cross-entropy):

$$H_k(p_k, q) = -\sum_{c \in text_k} p_k(c) * \log q(c)$$

  where $p_k(c)$ is the frequency of letter c in text k and $q(c)$ is the frequency of c in the language of the plaintext (e.g., English).
- output text k

Write a program that, given an English text encrypted with ROT-k for some unknown k in [0,25], automatically decrypts it by finding the k that minimizes the cross-entropy as explained above.

Here are the frequencies q of the 26 letters of the English alphabet:

```
0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015,
0.06094, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406,
0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056,
0.02758, 0.00978, 0.02360, 0.00150, 0.01974, 0.00074
```

For instance, letter 'A' (regardless of its case) has a frequency of 8.167%, i.e., occurs 8.167% of the times in a typical English text. Letter 'Z', on the other hand, occurs just 0.074% of the times.

3.2 Problem Details

Input

Your program must read the input data from the standard input.
The only line of the input contains the ROT-k encrypted English text.

Output

Your program must write the output data into the standard output.
The output must contains a single line with the decrypted text.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- The encrypted text is at most 4096 characters long.

3.3 Solution

The problem is an implementation task, since the steps to be done are already described in the description.

3.4 Source Code

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<float> Q = {0.08167, 0.01492, 0.02782, 0.04253, 0.12702,
                   0.02228, 0.02015, 0.06094, 0.06966, 0.00153,
                   0.00772, 0.04025, 0.02406, 0.06749, 0.07507,
                   0.01929, 0.00095, 0.05987, 0.06327, 0.09056,
                   0.02758, 0.00978, 0.02360, 0.00150, 0.01974,
                   0.00074};

string rotk(string text, int k){
    string out = "";
    for(auto c: text){
        if('a'<=c && c<='z'){
            c = 'a' + ((c-'a'+k) % 26);
        }
        else if('A'<=c && c<='Z'){
            c = 'A' + ((c-'A'+k) % 26);
        }
        out += c;
    }
    return out;
}

double crossEntropy(string text, int k){
    string rot = rotk(text, k);
    vector<double> P(26);
    int length = 0;
    for(auto c: rot){
        if('a'<=c && c<='z'){
            P[c-'a']++;
            length++;
        }
        if('A'<=c && c<='Z'){
            P[c - 'A']++;
            length++;
        }
    }
    double val = 0;
    for(int i=0; i<26; i++){
        P[i] /= length;
        val += P[i] * log2(Q[i]);
    }
    return -val;
}

int main(){
    vector<double> entropies;
    string S;
    getline(cin, S);
    for(int i=0; i<26; i++){
        entropies.push_back(crossEntropy(S, i));
    }

    int K=0;
    for(int i=0; i<26; i++){
        if(entropies[i] < entropies[K]){
```

```
            K = i;
        }
    }
    cout << rotk(S, K) << endl;
}
```

Python

```
#!/bin/env python3

from string import *
from math import *

Q = [0.08167, 0.01492, 0.02782, 0.04253, 0.12702,
     0.02228, 0.02015, 0.06094, 0.06966, 0.00153,
     0.00772, 0.04025, 0.02406, 0.06749, 0.07507,
     0.01929, 0.00095, 0.05987, 0.06327, 0.09056,
     0.02758, 0.00978, 0.02360, 0.00150, 0.01974,
     0.00074]
S = input()

def rotk(text, k):
  out = ""
  for c in text:
    if 'a' <= c <= 'z':
      c = chr(ord('a')+(ord(c)-ord('a')+k) % 26)
    elif 'A' <= c <= 'Z':
      c = chr(ord('A')+(ord(c)-ord('A')+k) % 26)
    out += c
  return out

def crossEntropy(text, k):
  text = list(filter(lambda c: c in ascii_lowercase, rotk(text, k).lower()))
  P = [sum(1 if x == ascii_lowercase[i] else 0 for x in text) for i in range(26)]
  P = [x/len(text) for x in P]
  val = 0.
  for i in range(26):
    val += P[i] * log(Q[i])
  return -val




entropies = [crossEntropy(S, k) for k in range(26)]

K = 0
for i in range(26):
  if entropies[i] < entropies[K]:
    K = i

print(rotk(S, K))
```