CS4021_Number Theory and Cryptography –

Assignment Questions

P1) Encrypting a String with Caesar Cipher

A simple way to encrypt things is by shifting all the letters by a given amount n. All characters other than letters (e.g. numbers, punctuation, etc. remain the same). For example, when employing a Caesar shift of 3 on the String go rangers you would get jr udqjhuv .Write a method caesarEncrypt which takes as input a String originalMessage and an int shift and returns the String created by applying a shift to the original message. Note that your method should work with all positive and negative integers. In the case that shift is greater than 26, you should cycle over the alphabet an additional time. For example the letter a shifted by 30 is the same as the letter a shifted by 4.

P2) Decrypting a string with Caesar Cipher

Write a method caesarDecrypt which takes as input a String encoded and an int shift and returns the String created by applying a negative shift to the original message.

P3) Consider the following stream cipher (which takes some ideas from the Enigma system used by Germans in World War II). Let π be a fixed permutation of Z26 and K a fixed element of Z_{26} . For all integers $i \geq 1$,the key stream element $Z_i \in Z_{26}$ is defined by $Z_i = (K + i - 1) \bmod 26$.

Encryption and decryption using π are done as follows:
$ez(x) = (\pi(x) + z) \bmod 26$;
$dz(y) = \pi^{-1}((y - z) \bmod 26)$;

Assuming

	(	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25)
π	=	23 13 24 0 7 15 14 6 25 16 22 1 19 18 5 11 17 2 21 12 20 4 10 9 3 8

Write a program to implement the cipher.

P4) Write a program to implement hill cipher to decrypt the ciphertext below which was encrypted using a Hill cipher with m = 2:

LMQETXYEAGTXCTUIEWNCTXLZEWUAISPZYVAPEWLMGQWYAXFTCJMS
QCADAGTXLMDXNXSNPJQSYVAPRIQSMHNOCVAXFV

P5) Write a program to implement playfair cipher which takes a keyword as input.

P6) Write a program to implement RSA cryptosystem. The program takes two prime

numbers as input.

P7) Implement double DES and check its vulnerability against **meet in the middle attack**.

P8) Implement Diffie Hellman key exchange and check its vulnerability against **man in the middle attack.**

P9) Implement Elgamal scheme and check its vulnerability against **low modulus attack.**

P10) Implement Elgamal scheme and check its vulnerability against **Known plaintext attack.**

P11) Implement 128 bit AES in C / C++. (Encryption only)

P12) Implement a hill cipher. Let the 2×2 key matrix be $K = \begin{matrix} a & b \\ c & d \end{matrix}$ where a,b,c,d can take values ranging from 0 to 25. Encrypt the plaintexts (size :4) using K.

b) Using the same key used for encryption find the plain text corresponding to the ciphertext.

P13) Implement Elgamal cryptosystem. Your program should be able to encrypt and decrypt data using Elgamal cryptosystem.

P14) Implement 64 bit DES in C/C++. (Encryption only)

P15) One-time pad is one of the most widely known schemes to encrypt a binary string to achieve confidentiality. This scheme takes a binary string as input and outputs another binary string of the same length. The input is called the plaintext, and the output is called the ciphertext. The scheme uses a key which is another binary string of the same length as the input. The i-th bit of the ciphertext is defined as the XOR of the i-th bit of the plaintext and the key . The ciphertext is sent to the receiving party. In this problem, we will consider several messages, each of length N, encrypted using a single key of length N. We would like to investigate how strong this cipher is. Suppose an adversary manages to find out the content of all the original messages (i.e., the plaintexts) and some of the encrypted messages (i.e., ciphertexts). Return the number of possible keys that are consistent with this data. The constraints will guarantee that there is at least one such key. A key is consistent if for all members of ciphertexts C, there exists a member of plaintexts P such that when P is encrypted using the specified key, it becomes C.

Constraints

- plaintexts will contain between 1 and 50 elements, inclusive.

- Each element of plaintexts will contain between 1 and 50 characters, inclusive.
- All the elements of plaintexts will contain the same number of characters.
- All the characters in plaintexts will be either '0' (zero) or '1' (one).
- All the elements of plaintexts will be distinct.
- ciphertexts will contain between 1 and 50 elements, inclusive.
- All the elements of ciphertexts will contain the same number of characters as all the elements of plaintexts.
- All the characters in ciphertexts will be either '0' (zero) or '1' (one).
- All the elements of ciphertexts will be distinct.
- There will exist at least one key that is consistent with the given plaintexts and ciphertexts.

Examples

1)
{"110", "001"}
{"101", "010"}
Returns: 2
The two possible keys are "011" and "100".
2)
{"00", "01", "10", "11"}
{"00", "01", "10", "11"}
Returns: 4
3)
{"01", "10"}
{"00"}
Returns: 2
4)
{"000", "111", "010", "101", "110", "001"}
{"011", "100"}
Returns: 6

P16) Let's say you have a binary string such as the following:

011100011

One way to encrypt this string is to add to each digit the sum of its adjacent digits. For example, the above string would become:

123210122

In particular, if P is the original string, and Q is the encrypted string, then Q[i] = P[i-1] + P[i] + P[i+1] for all digit positions i. Characters off the left and right edges of the string are treated as zeroes.

An encrypted string given to you in this format can be decoded as follows (using 123210122 as an example):

Assume P[0] = 0.

Because Q[0] = P[0] + P[1] = 0 + P[1] = 1, we know that P[1] = 1.

Because Q[1] = P[0] + P[1] + P[2] = 0 + 1 + P[2] = 2, we know that P[2] = 1.

Because Q[2] = P[1] + P[2] + P[3] = 1 + 1 + P[3] = 3, we know that P[3] = 1.

Repeating these steps gives us P[4] = 0, P[5] = 0, P[6] = 0, P[7] = 1, and P[8] = 1.

We check our work by noting that Q[8] = P[7] + P[8] = 1 + 1 = 2. Since this equation works out, we are finished, and we have recovered one possible original string.

Now we repeat the process, assuming the opposite about P[0]:

Assume P[0] = 1.

Because Q[0] = P[0] + P[1] = 1 + P[1] = 0, we know that P[1] = 0.

Because Q[1] = P[0] + P[1] + P[2] = 1 + 0 + P[2] = 2, we know that P[2] = 1.

Now note that Q[2] = P[1] + P[2] + P[3] = 0 + 1 + P[3] = 3, which leads us to the conclusion that P[3] = 2. However, this violates the fact that each character in the original string must be '0' or '1'. Therefore, there exists no such original string P where the first digit is '1'.

Note that this algorithm produces at most two decodings for any given encrypted string. There can never be more than one possible way to decode a string once the first binary digit is set.Given a String message, containing the encrypted string, return a String[] with exactly two elements. The first element should contain the decrypted string assuming the first character is '0'; the second element should assume the first character is '1'. If one of the tests fails, return the string "NONE" in its place. For the above example, you should return {"011100011", "NONE"}.

Constraints

- message will contain between 1 and 50 characters, inclusive.
- Each character in message will be either '0', '1', '2', or '3'.

Examples

1)

"123210122"

Returns: { "011100011", "NONE" }

The example from above.

2)

"11"

Returns: { "01", "10" }

We know that one of the digits must be '1', and the other must be '0'. We return both cases.

2)

"22111"

Returns: { "NONE", "11001" }

Since the first digit of the encrypted string is '2', the first two digits of the original string must be '1'. Our test fails when we try to assume that P[0] = 0.

3)

"123210120"

Returns: { "NONE", "NONE" }

This is the same as the first example, but the rightmost digit has been changed to something inconsistent with the rest of the original string. No solutions are possible.

4)

"3"

Returns: { "NONE", "NONE" }

5)

"12221112222221112221111111112221111"

Returns:

{ "01101001101101001101001001001101001",

"10110010110110010110010010010110010" }

P17) Implement Vernam Cipher. Your program should be able encrypt and decrypt using vernam cipher.

P18)

A simple way to encode a word into a string of digits is to replace each letter by its order in the alphabet. That is, "a" will change to "1", "b" to "2", ..., and "z" to "26". For example, encode("cow")="31523" and encode("cat")="3120".Sadly, this encoding cannot always be uniquely decoded, because two different words can yield the same string of digits when encoded. For example, encode("beard")=encode("yard")="251184".

String A is a subsequence of string B if it is possible to erase some letters of B (possibly none, possibly all of them) to obtain A. For example, "cage" is a subsequence of "cabbages".

You are given a String D containing a string of digits. If there is no string Y such that encode(Y)=D, return the String "NONE". Otherwise, find and return the longest string X with the following property: whenever encode(Y)=D, X is a subsequence of Y. If there are multiple such strings, return the lexicographically smallest one.

Constraints

- D will contain between 1 and 50 characters, inclusive.
- Each character in D will be a digit ('0'-'9').

Examples

1)

"38956"

Returns: "chief"

There is only one way to decode this string of digits.

2)

"13919156"

Returns: "if"

This string of digits can be decoded in 8 different ways. Each of them contains an "i" followed by an "f".

3)

"1122"

Returns: ""

We have encode("kbb")=encode("aav")="1122". The strings "kbb" and "aav" have no common subsequence other than the empty one.

4)

"3120"

Returns: "cat"

The only valid decoding of this string is "cat".

5)

"0"

Returns: "NONE"

P19)

Consider a simple encryption algorithm based on the move-to-front heuristic. Both the unencrypted (plaintext) and encrypted (ciphertext) messages are Strings composed of uppercase letters ('A'-'Z') and spaces (' '). The encryption algorithm maintains a state which is a permutation of the 27 possible characters. The key used to encrypt and decrypt messages is the initial permutation. Encryption processes the plaintext from left to right, outputting one character of the ciphertext for each character of the plaintext. At each position of the plaintext, the encryption algorithm performs the following steps:

Find the (zero-based) position of the plaintext character in the current permutation.

If the position is 0-25, output 'A'-'Z', respectively. If the position is 26, output a space.

Move the plaintext character to the front of the permutation (ie, delete it from its current position in the permutation and re-insert it at the front).

For example, starting with a key "ZYXWVUTSRQPON MLKJIHGFEDCBA" and a plaintext "TPCDR" (all quotes for clarity only), encryption would proceed as follows:

1)State = "ZYXWVUTSRQPON MLKJIHGFEDCBA". 'T' is in position 6, output 'G'.

2)State = "TZYXWVUSRQPON MLKJIHGFEDCBA". 'P' is in position 10, output 'K'.

3)State = "PTZYXWVUSRQON MLKJIHGFEDCBA". 'C' is in position 24, output 'Y'.

4)State = "CPTZYXWVUSRQON MLKJIHGFEDBA". 'D' is in position 24, output 'Y'.

5)State = "DCPTZYXWVUSRQON MLKJIHGFEBA". 'R' is in position 11, output 'L'.

The final ciphertext is "GKYYL". (The final state is "RDCPTZYXWVUSQON MLKJIHGFEBA", but that is discarded by the algorithm.)

You will be given both the plaintexts and the ciphertexts of several messages, where the i-th ciphertext is believed to be the encrypted form of the i-th plaintext, and all the messages are believed to have been encrypted using the same key. Your task is to recover and return that key (as a String). If the messages do not contain enough information to fully determine the key, the output of your method should summarize the set of possible keys by placing a '-' character at any position of the permutation that has not been narrowed down to a single possible character. If no key exists that could have been used to encrypt all the messages, output "ERROR".

Notice that, when encrypting multiple messages, the state is not carried over from one message to the next, but rather is re-initialized for each message.

Constraints

- plaintexts and ciphertexts contain the same number of elements (between 1 and 20, inclusive).

- Element i of plaintexts has the same length as element i of ciphertexts (between 1 and 50 characters, inclusive).
- Elements of plaintexts and ciphertexts contain uppercase letters ('A'-'Z') and spaces only.

Examples

1)

{"TPCDR"}

{"GKYYL"}

Returns: "------T-R-P----------DC--"

The example above, but the characters that were not used could have been anywhere in the initial permutation.

2)

{"A","B"}

{"X","X"}

Returns: "ERROR"

A and B cannot both be in position 23 of the key.

3)

{"HELLO"}

{"HOWDY"}

Returns: "ERROR"

The second 'L' should be encoded as an 'A'.

4)

{"FOUR SCORE AND","SEVEN YEARS AGO","OUR FOREFATHERS"}

{"ABCDEFGFEHFIJK","FHLBKIMDLKHFDNK","BCDEEEDHDIOPEFJ"}

Returns: "FOUR SCEANDVYGTH----------"

5)

{"LIZARD","JACKAL","HOWLER MONKEY","BLOWFISH","LYNX","GIRAFFE","VULTURE","QUAIL"}

{"LGGLM","ZEQWCO","GFNMFLGRGLVGS","UMGOTJML","LQJY","OGKGSAG","WNNFCMG","XNFHN}

Returns: " ETAOIHNSRDLUWGCYMFPBKVQXJZ"

6)

{"HI"}

{"AA"}

Returns: "ERROR"

The 'I' cannot be encoded as 'A' because 'H' is guaranteed to be in that position.