|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem Statement for NetworkXOneTimePad** | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| |  |  | | --- | --- | | **Problem Statement** | | |  | One-time pad (patented by Vernam in 1919) is one of the most widely known schemes to encrypt a binary string to achieve confidentiality. This scheme takes a binary string (a string consisting of only the digits 0 and 1) 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 (see the notes for XOR definition). 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). These messages are given in the String[]s **plaintexts** and **ciphertexts**, respectively. 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. | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | NetworkXOneTimePad | | Method: | crack | | Parameters: | String[], String[] | | Returns: | int | | Method signature: | int crack(String[] plaintexts, String[] ciphertexts) | | (be sure your method is public) | | | |  |  | |  |  | |  | | | **Notes** | | | - | XOR is a binary operation on bits defined as follows: 0 XOR 0 = 0, 0 XOR 1 = 1, 1 XOR 0 = 1, 1 XOR 1 = 0. | |  | | | **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** | | | 0) |  | |  | |  |  |  | | --- | --- | --- | | |  | | --- | | {"110", "001"} | | {"101", "010"} | | | Returns: 2 | | |  | | --- | | The two possible keys are "011" and "100". | | | | 1) |  | |  | |  |  |  | | --- | --- | --- | | |  | | --- | | {"00", "01", "10", "11"} | | {"00", "01", "10", "11"} | | | Returns: 4 | | |  | | --- | |  | | | | 2) |  | |  | |  |  |  | | --- | --- | --- | | |  | | --- | | {"01", "10"} | | {"00"} | | | Returns: 2 | | |  | | --- | |  | | | | 3) |  | |  | |  |  |  | | --- | --- | --- | | |  | | --- | | {"000", "111", "010", "101", "110", "001"} | | {"011", "100"} | | | Returns: 6 | | |  | | --- | |  | | |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. | |  |  |  |  |  |  |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| This problem was used for:         [Single Round Match 516 Round 1 - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=14541&pm=10846)         [Single Round Match 516 Round 1 - Division II, Level Two](http://community.topcoder.com/tc?module=ProblemDetail&rd=14541&pm=10846) | | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Problem Statement for RouteIntersection** | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | |  |  | | --- | --- | | **Problem Statement** | | |  | Little Dazdraperma likes to travel a lot. One day she made a route in an **N**-dimensional space. In this space each point is represented by **N** coordinates. The coordinates are indexed from 1 to **N**, inclusive. She started from the origin, i.e., a point where each coordinate is 0. Then she did several moves of the following type:   * First she chose a coordinate index, i.e., a number between 1 and **N**, inclusive. * Then she jumped to a point where the coordinate with the chosen index is either increased or decreased by 1 and all other coordinates remain the same.   Now Dazdraperma wonders whether she has ever visited the same point twice. You will be given a int[]**coords** and a String **moves** representing her route. The i-th element of **coords** is the coordinate index she has chosen during her i-th move. If the coordinate with this index was increased during the i-th move, the i-th character of **moves** will be '+', and it will be '-' if this coordinate was decreased.     Return "VALID" if all points of her route were unique, including the first and the last points, and return "NOT VALID" otherwise. Two points A and B in **N**-dimensional space are different if there's an index i such that A's coordinate with index i and B's coordinate with index i are different. | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | RouteIntersection | | Method: | isValid | | Parameters: | int, int[], String | | Returns: | String | | Method signature: | String isValid(int N, int[] coords, String moves) | | (be sure your method is public) | | | |  |  | |  |  | |  | | | **Constraints** | | | - | **N** will be between 1 and 1000000000 (109), inclusive. | | - | **coords** will contain between 1 and 50 elements, inclusive. | | - | Each element of **coords** will be between 1 and **N**, inclusive. | | - | **moves** will contain the same number of characters as the number of elements in **coords**. | | - | Each character in **moves** will be either '+' or '-'. | |  | | | **Examples** | | | 0) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 1 | | {1} | | "+" | | | Returns: "VALID" | | |  | | --- | | Dazdraperma starts at (0) and then jumps to (1). The answer is "VALID". | | | | 1) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 2 | | {1,2,1,2} | | "++--" | | | Returns: "NOT VALID" | | |  | | --- | | The route goes through 5 points: (0,0) -> (1,0) -> (1,1) -> (0,1) -> (0,0). The point (0,0) was visited twice. | | | | 2) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 3 | | {1,2,3,1,2} | | "+++--" | | | Returns: "VALID" | | |  | | --- | | (0,0,0) -> (1,0,0) -> (1,1,0) -> (1,1,1) -> (0,1,1) -> (0,0,1). | | | | 3) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 344447 | | {132,51717,628,344447,628,51717,344447,2} | | "+-++-+--" | | | Returns: "NOT VALID" | | |  | | --- | | The repeated point doesn't have to be the first or the last point in the route. | | | | 4) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 1 | | {1,1} | | "+-" | | | Returns: "NOT VALID" | | |  | | --- | |  | | | | 5) |  | |  | |  |  |  |  | | --- | --- | --- | --- | | |  | | --- | | 990630 | | {833196,524568,361663,108056,28026,824639,269315,440977,440977,765458,  988451,242440,948414,130873,773990,765458,130873,28026,853121,553636,  581069,82254,735536,833196,898562,898562,940783,988451,540613,317306,  623194,940783,571384,988451,108056,514374,97664} | | "--+---+-+++-+-+---++-++-+---+-+--+-++" | | | Returns: "NOT VALID" | | |  | | --- | |  | | |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. |  |  |  |  |  |  | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | This problem was used for:         [Member Single Round Match 474 Round 1 - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=14185&pm=10804)         [Member Single Round Match 474 Round 1 - Division II, Level Two](http://community.topcoder.com/tc?module=ProblemDetail&rd=14185&pm=10804) | | | | | | | | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Problem Statement for RunLengthPlus** | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | |  |  | | --- | --- | | **Problem Statement** | | |  | One type of compression is *run length encoding*. In this case, a character may be preceded by an integer N, which indicates that the character should be repeated N times. For example, the String "3ABBC10D" would be decompressed as "AAABBCDDDDDDDDDD". A character that is not preceded by an integer is repeated just once. We are going to enrich this slightly be allowing a sequence of characters to be preceded by an integer N, indicating that the sequence should be repeated N times. In this case, the sequence of characters should be surrounded by parentheses. Additionally, you may nest the compressed sequences. Thus, the compressed sequence "X2(2A3(BC))X" would be decompressed as "XAABCBCBCAABCBCBCX". You will be given a String of uppercase letters and are to compress it in such a way that the result has as few characters as possible. If there are multiple ways to do this, choose the one that comes first lexicographically (using standard ASCII ordering). | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | RunLengthPlus | | Method: | compress | | Parameters: | String | | Returns: | String | | Method signature: | String compress(String s) | | (be sure your method is public) | | | |  |  | |  |  | |  | | | **Constraints** | | | - | **s** will contain between 1 and 50 uppercase letters, inclusive. | |  | | | **Examples** | | | 0) |  | |  | |  |  | | --- | --- | | |  | | --- | | "AAABBCDDDDDDDDDD" | | | Returns: "3A2BC10D" | |  | | | 1) |  | |  | |  |  | | --- | --- | | |  | | --- | | "XAABCBCBCAABCBCBCX" | | | Returns: "X2(2A3(BC))X" | |  | | | 2) |  | |  | |  |  | | --- | --- | | |  | | --- | | "ABCBACBABCBABCABACACBCBABACBCBBABACBACBCACBBAC" | | | Returns: "ABCBA2(CBAB)CABACACBCBABACBC2BA2(BAC)BCAC2BAC" | |  | |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. |  |  |  |  |  |  | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | This problem was used for:         [2005 TCCC Wildcard Room - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=6553&pm=3961) | | | | | | | | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | **Problem Statement for RepeatedSubstrings** | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | |  |  | | --- | --- | | **Problem Statement** | | |  | Consider a family of compression algorithms that exploit repeated substrings in the text to be compressed. The basic idea is to replace selected substrings with references of the form "&*startPos*-*endPos*", where the substring to be replaced is identical to the substring of the original text that begins at position*startPos* and ends at position *endPos*, inclusive. For example, the text "ABCDEFG ABCDEFG" might be compressed to "ABCDEFG &0-6". Notice that positions are zero-based and also that positions are given relative to the original text, not the compressed text. The original text is assumed to contain only uppercase letters ('A'-'Z') and spaces (' ').  Compression algorithms in this family work as follows:  1. copy the original text into W  2. while not done do  A. choose some substring S of length > 1 that occurs both in W and in the original text  B. find the starting position *startPos* and ending position *endPos*  of some occurrence of S in the original text  C. replace some occurrence of S in W with "&*startPos*-*endPos*"  3. output W  Different algorithms in this family might decide when to finish or how to choose S differently, but they all follow this basic outline. Your task is to write a decompression method that will decompress text compressed by any member of this family. In other words, given the output of a compression algorithm in this family, you are to reconstruct and return the original text.  The input will be a String **compressed** containing only uppercase letters, spaces, and references of the form "&*startPos*-*endPos*", where *startPos* and *endPos* are non-negative integers written without extraneous leading zeros, and *startPos* < *endPos*. The original text is guaranteed to contain no more than 256 characters.  Notice that if the compression algorithm makes unwise choices, the decompression algorithm may be unable to reconstruct some of the characters in the original text. In such cases, the decompression algorithm should return a '?' in each position for which the character cannot be determined. For example, consider the compressed text "AB&7-9&2-6". We know that the original text contains 10 characters, and that the first two characters of the original text are 'A' and 'B', but we cannot tell what the remaining characters are. Therefore, your method should return "AB????????". | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | RepeatedSubstrings | | Method: | decompress | | Parameters: | String | | Returns: | String | | Method signature: | String decompress(String compressed) | | (be sure your method is public) | | | |  |  | |  |  | |  | | | **Constraints** | | | - | **compressed** contains between 1 and 50 characters, inclusive. | | - | **compressed** is the concatenation of some number of tokens, where each token is an uppercase letter ('A'-'Z'), a space (' '), or a string of the form "&*x*-*y*", where *x* and *y* are integers between 0 and 255, inclusive, written without extraneous leading zeros, and with *x* < *y*. | | - | **compressed** is the output of some member of the given family of compression algorithms for some original text containing between 1 and 256 characters, inclusive, all of which are uppercase letters or spaces. | |  | | | **Examples** | | | 0) |  | |  | |  |  | | --- | --- | | |  | | --- | | "ABCDEFG &0-6" | | | Returns: "ABCDEFG ABCDEFG" | | |  | | --- | | The first example above. | | | | 1) |  | |  | |  |  | | --- | --- | | |  | | --- | | "AB&7-9&2-6" | | | Returns: "AB????????" | | |  | | --- | | The second example above. | | | | 2) |  | |  | |  |  | | --- | --- | | |  | | --- | | "IT WAS THE BE&39-49 &0-10WORST OF TIMES" | | | Returns: "IT WAS THE BEST OF TIMES IT WAS THE WORST OF TIMES" | |  | | | 3) |  | |  | |  |  | | --- | --- | | |  | | --- | | "ABC&0-21" | | | Returns: "ABCABCABCABCABCABCABCABCA" | |  | | | 4) |  | |  | |  |  | | --- | --- | | |  | | --- | | "&0-10" | | | Returns: "???????????" | |  | | | 5) |  | |  | |  |  | | --- | --- | | |  | | --- | | "&5-9ABC&2-7DE&20-22&17-19F" | | | Returns: "ABCCCABCCCCABCDEF?F?F?F" | |  | |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. |  |  |  |  |  |  | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | This problem was used for:         [2003 TCO Semifinal Round 2 - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=4707&pm=2004) | | | | | | | | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | **Problem Statement for BinaryCode** | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | |  |  | | --- | --- | | **Problem Statement** | | |  | 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):   1. Assume P[0] = 0. 2. Because Q[0] = P[0] + P[1] = 0 + P[1] = 1, we know that P[1] = 1. 3. Because Q[1] = P[0] + P[1] + P[2] = 0 + 1 + P[2] = 2, we know that P[2] = 1. 4. Because Q[2] = P[1] + P[2] + P[3] = 1 + 1 + P[3] = 3, we know that P[3] = 1. 5. Repeating these steps gives us P[4] = 0, P[5] = 0, P[6] = 0, P[7] = 1, and P[8] = 1. 6. 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]:   1. Assume P[0] = 1. 2. Because Q[0] = P[0] + P[1] = 1 + P[1] = 0, we know that P[1] = 0. 3. Because Q[1] = P[0] + P[1] + P[2] = 1 + 0 + P[2] = 2, we know that P[2] = 1. 4. 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"}. | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | BinaryCode | | Method: | decode | | Parameters: | String | | Returns: | String[] | | Method signature: | String[] decode(String message) | | (be sure your method is public) | | | |  |  | |  |  | |  | | | **Constraints** | | | - | **message** will contain between 1 and 50 characters, inclusive. | | - | Each character in **message** will be either '0', '1', '2', or '3'. | |  | | | **Examples** | | | 0) |  | |  | |  |  | | --- | --- | | |  | | --- | | "123210122" | | | Returns: { "011100011", "NONE" } | | |  | | --- | | The example from above. | | | | 1) |  | |  | |  |  | | --- | --- | | |  | | --- | | "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" } | |  | |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. |  |  |  |  |  |  | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | http://community.topcoder.com/i/clear.gif | | | | | | | | This problem was used for:         [Single Round Match 144 Round 1 - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=4515&pm=1704)         [Single Round Match 144 Round 1 - Division II, Level Two](http://community.topcoder.com/tc?module=ProblemDetail&rd=4515&pm=1704) | | | | | | | | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| Problem Statement for Vigenere | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| |  |  | | --- | --- | | **Problem Statement** | | |  | This problem is taken from the Collegiate Challenge Finals.  Class Name: Vigenere  Method Name: coder  Parameters: String,String,int  Returns: String  The Vigenere Cipher is a simple but strong encryption algorithm. Unlike many  simple encryption algorithms, the Vigenere Cipher is immune to frequency  analysis because it alters the normal frequency by using more than one alphabet  to encrypt the message. Instead of there being a one-to-one relationship  between each letter and its substitute, there is a one-to-many relationship  between each letter and its substitute.  The Vigenere Cipher is based on the following table (which has a simple pattern  so you don't actually have to store the table).  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  A 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  B 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 A  C C 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  D 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  . . . . .  . . . . .  . . . . .  When encoding a message with the Vigenere Cipher, a code word is written  repeatedly over the message. Each letter in the message is replaced with the  letter at the intersection of the row given by the corresponding code word  letter and the column given by the message letter.  Implement a class Vigenere that contains a method coder. The method encodes  and decodes Strings of capital letters (A-Z) using the Vigenere Cipher. coder  takes two Strings and an int as a parameter. The first String is the message.  The second String is the code word. The int specifies whether the message  should be encoded or decoded:  int=1: encode message  int=2: decode message  The method returns a String that is the resulting (encoded or decoded) message.  If the message or code word is not valid (They do not contain only capital  letters in the range A to Z, or the code word is of length 0) the method  returns "ERROR"  If the int is not 1 or 2, the method returns "ERROR"  If both the message and code word are of length 0, or if just the message has  length 0 (and the code word is valid), the method should return an empty string.  The method signature is:  public String coder(String message,String codeWord,int action);  message and codeWord are Strings with length less than 1000.  Examples:  If the message is "TOPCODERISGREAT" and the codeWord is "CODE" and the action  is encode (1), write the code word over the message and look up letters in the  table:  Code Word:  CODECODECODECOD  Message:  TOPCODERISGREAT  Encoded:  VCSGQRHVKGJVGOW  because the letter in row C column T is V,  the letter in row O column O is C,  etc....  If the message is "HRWCQYRORWSCGKUO" and the codeWord is "OK" and the action is  decode (2), the decoded message is "THISCODEDMESSAGE".  If the message is "HowAreYou" and the codeWord is "GO" and the action is encode  (1), the method returns "ERROR" because the message isn't all capital letters. | |  | | | **Definition** | | |  | |  |  | | --- | --- | | Class: | Vigenere | | Method: | coder | | Parameters: | String, String, int | | Returns: | String | | Method signature: | String coder(String param0, String param1, int param2) | | (be sure your method is public) | | | |  |  | |  |  |   This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved. |  |  |  |  | |  |  |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| http://community.topcoder.com/i/clear.gif | | | | | | | |
| This problem was used for:         [Collegiate Challenge Finals - Division I, Level One](http://community.topcoder.com/tc?module=ProblemDetail&rd=2009&pm=67) | | | | | | | |