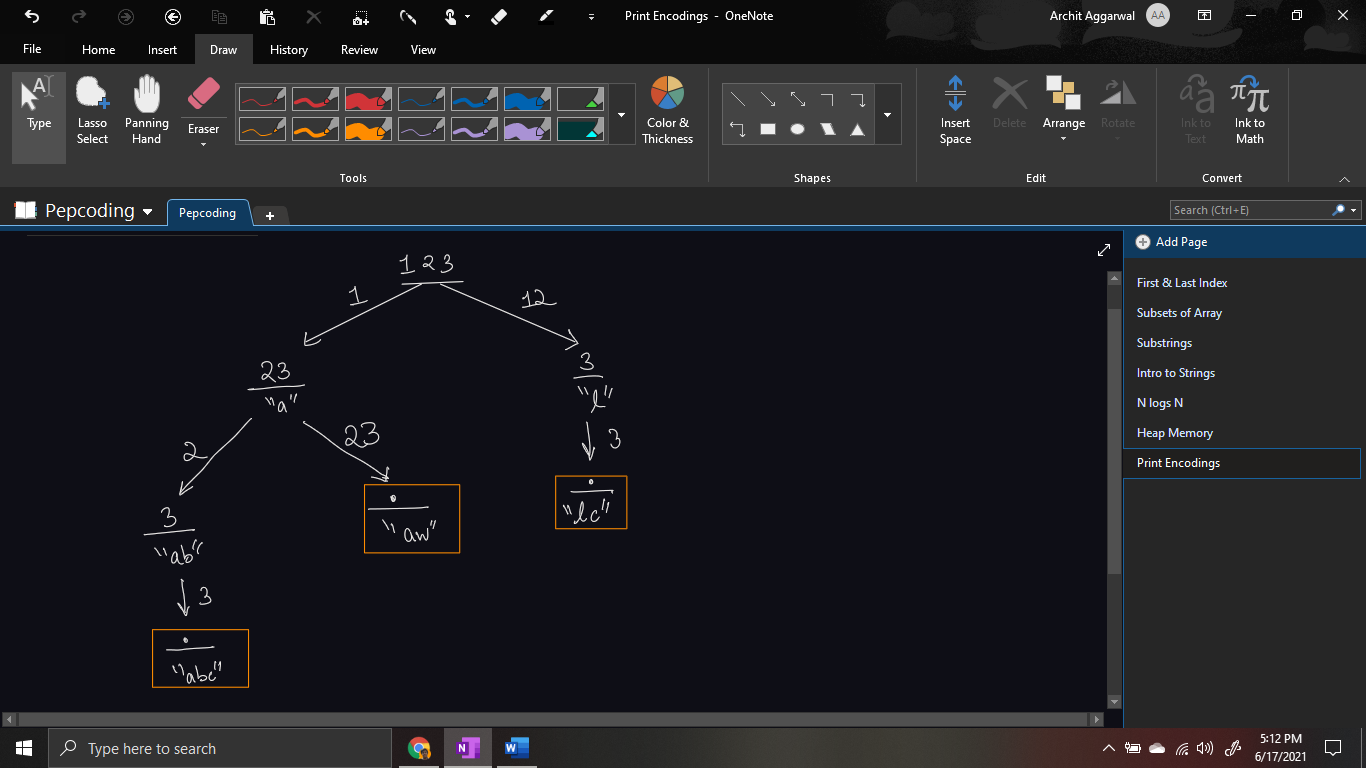
Dear reader, welcome to one of the last problems in level 1 of ‘*Recursion - On the Way Up*’, and the problem’s name is: ‘[***Print Encodings***](https://www.pepcoding.com/resources/online-java-foundation/recursion-on-the-way-up/print-encodings-official/ojquestion)’.

***Problem Statement:*** You are given a set of rules for encoding a given string:

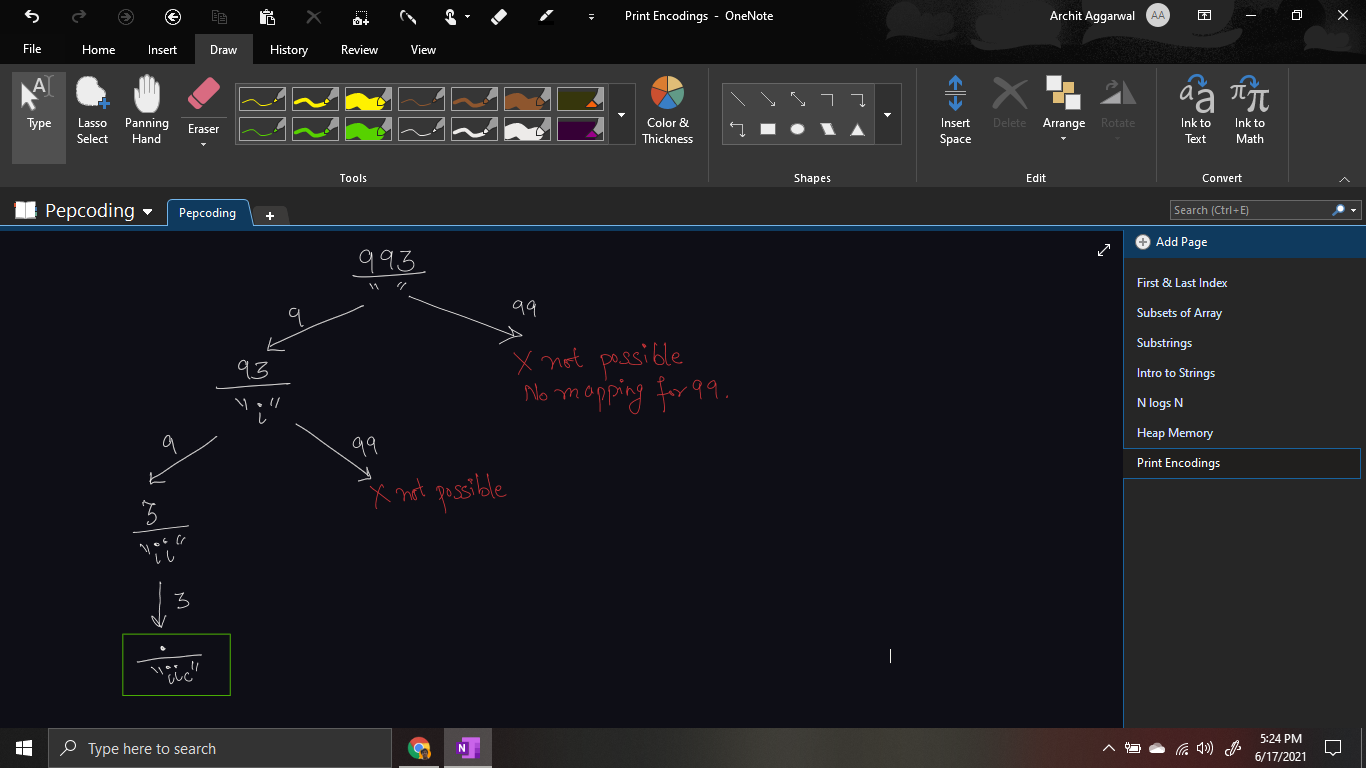
1 -> a, 2 -> b, 3 -> c, … 25 -> y and 26 -> z

You have to print all of the possible strings which the given string can be encoded into.

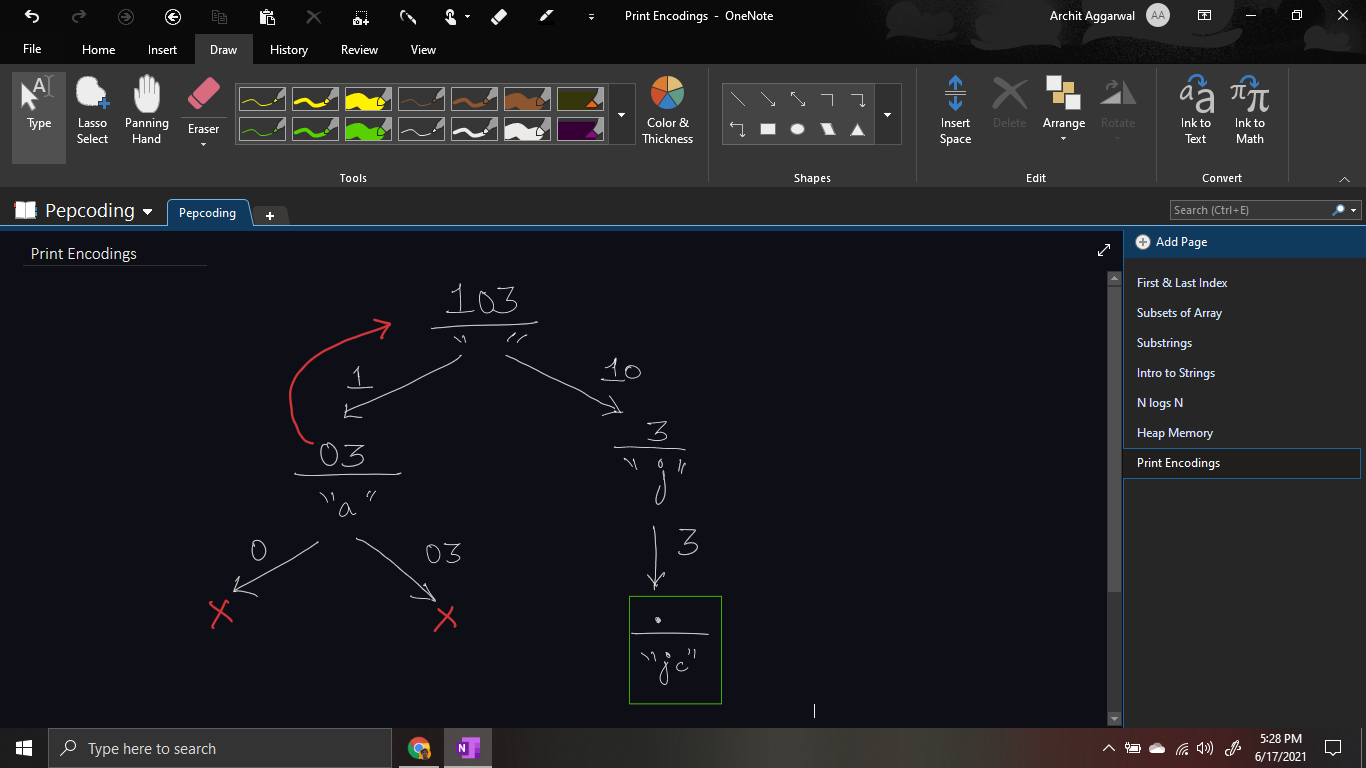
Example) For input “123”, there will be 3 encodings possible, and that are “abc”, “aw” and “lc”.



Example) For input “993”, there will be only one encoding possible, i.e. “iic”.



Example) For input “103”, there will be only one encoding possible, i.e. “jc”.

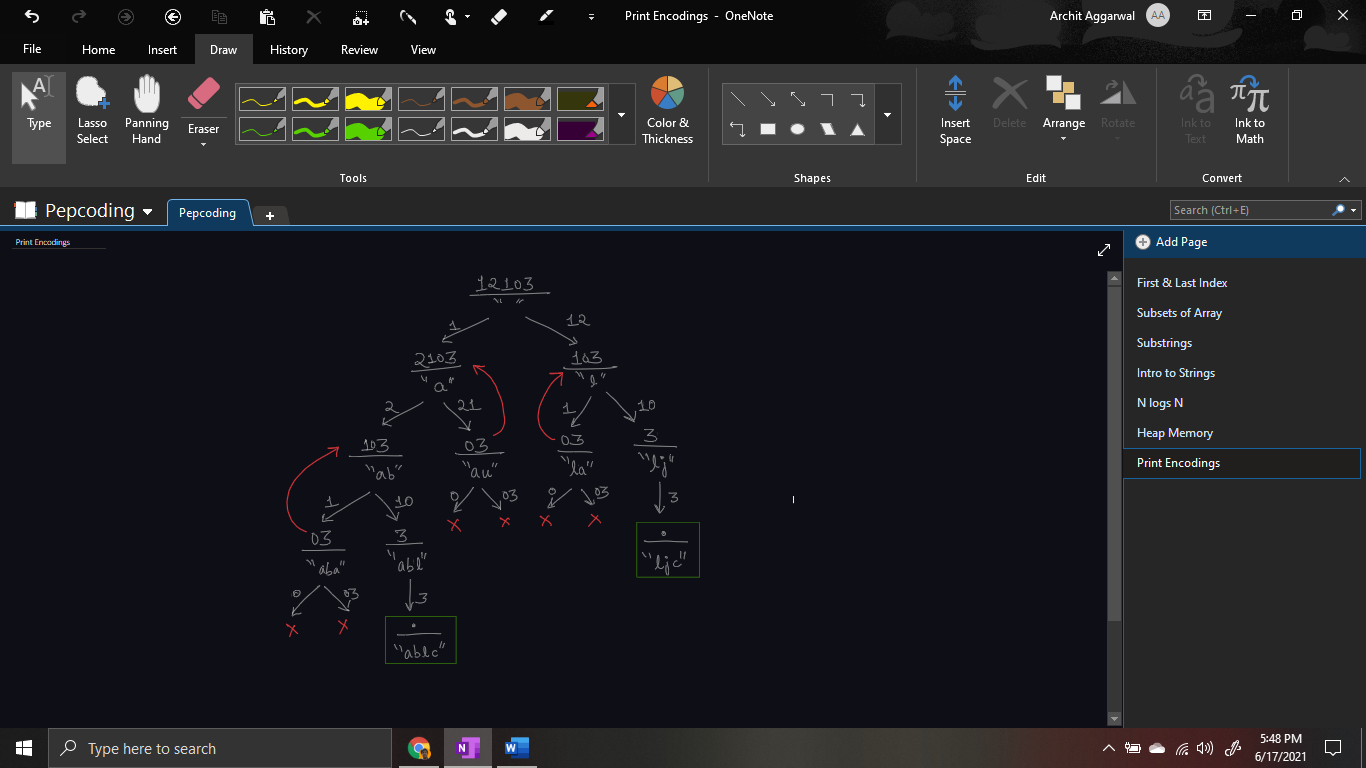


*Note*: There is no character mapping for integer 0. That is, strings “0”, “01”, “013”, etc. will have no encodings possible. You should not consider *01* as *1*, but take it as invalid input.

These examples are taken in our [question video](https://www.youtube.com/watch?v=jUo0Qis4FKU), and explained how to draw the recursion tree for each of them.

***Solution***

Let us try to take an input which has all of the major special/corner cases and draw the recursion tree for it. Let the input string be 12103.



Now, let us try to convert this recursion tree into a recursive code. Let us write the recursive function *printEncodings(String str)* which will take the input string as the first parameter, and the output (= *ans*) string formed so far as the other parameter, and it will print all the encodings possible.

***Expectation***: We will define expectation with our function *printEncodings(str, ans)* that it will print all the encodings of string str.

***Faith***: It can be seen easily that we can make atmax 2 calls for a smaller problem, one can be str 1 length shorter, other can be str 2 length shorter.

Hence we will keep faith on function *printEncodings(str.substr(1), ans + x)* and *printEncodings(str.substr(2), ans + y)* that they will print all the encodings possible.

But what to fill x and y? How to meet the expectation with our developed faith.

**Meeting Expectation with Faith:**

* First, we will extract only 1 digit from the left of the input string.
  + If the extracted digit is 0, then the input string is invalid and we will return directly (without printing).
  + Else, we will add the character equivalent of the digit to the answer string, i.e. we have found *x*, and call for the subproblem *printEncodings(str.substr(1), ans + x)*.
* Now, we will check if there are at least 2 digits in the string, if yes, then
  + We extract the two digits. If the first digit is 0, then return without printing.
  + If there is a character equivalent of the digits (for digits 10-26), then we will add the equivalent to to the answer string, i.e. we have found y. Now, we will call for the subproblem *printEncodings(str.substr(2), ans + y).*
  + Else, we will simply return without printing.

Now, the only task is to find what can be the ***base case***?

**Base Case:** We can take the base case as an empty input string “”. We will print the output string ans, as it is a valid encoding and return.

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new

InputStreamReader(System.in));

String str = br.readLine();

printEncodings(str, "");

}

public static void printEncodings(String ques, String ans) {

if (ques.length() == 0) {

System.out.println(ans);

return;

} else if (ques.length() == 1) {

if (ques.charAt(0) == '0') {

return;

} else {

String ch0 = ques.substring(0, 1);

String roq0 = ques.substring(1);

String code0 = (char)('a' +

(Integer.parseInt(ch0) - 1)) + "";

printEncodings(roq0, ans + code0);

}

} else {

if (ques.charAt(0) == '0') {

return;

} else {

String ch0 = ques.substring(0, 1);

String roq0 = ques.substring(1);

String code0 = (char)('a' +

(Integer.parseInt(ch0) - 1)) + "";

printEncodings(roq0, ans + code0);

String ch01 = ques.substring(0, 2);

String roq01 = ques.substring(2);

String code01 = (char)('a' +

(Integer.parseInt(ch01) - 1)) + "";

if (Integer.parseInt(ch01) <= 26) {

printEncodings(roq01, ans + code01);

}

}

}

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=2ClSccwnq1Y) from timestamp [*1:20 , 11:40]*.

* What is the ***time complexity*** of the above code?

We can make atmax 2 decisions (1-length number or 2-length number), hence the time complexity will be ***O(2n)*** (exponential in nature).

* What is the ***space complexity*** of the above code?

If we consider the space taken by the recursion call stack, then space complexity will be ***O(n)*** as the maximum depth of the recursion stack at any point can be equal to the length of the numeric string.

Hope that you liked the article on Print Encodings. We will introduce ‘Backtracking’ in our next section of the ‘Recursion’ series.

**Some Suggestions:**

1. If you are able to guess that the problem is based on recursion/backtracking (or dynamic programming in coming sections), then always try to build the recursion tree for a few test cases. Drawing it will surely help you find out how we can meet the faith with our expectations.
2. Do not rush through recursion. “*Ya to tum recursion me doob jao, ya recursion tumhe duba degi*” (Try to swim in the Ocean of Recursion, else you will drown). Have patience, since recursion seems a complex topic for beginners, but once you will start getting used to it, you will be having fun coding all day long.

Article Contributed by:

Archit Aggarwal