Daily Coding Problem

Good morning! Here's your coding interview problem for today.

This problem was asked by Facebook.

Given the mapping a = 1, b = 2, ... z = 26, and an encoded message, count the number of ways it can be decoded.

For example, the message '111' would give 3, since it could be decoded as 'aaa', 'ka', and 'ak'.

You can assume that the messages are decodable. For example, '001' is not allowed.

Looking at this challenge
We know the widest number mapping can be Z = 26 or anything above J=10
shortest width A=1 or anything up to I=9

if a number was 3 digits wide

111
A=1
AAA

111
K=11
A=1
KA

111
A=1
K=11
AK

Now if I examine a 4 digit wide number:

1111
A=1
AAAA

1111
K=11
A=11
KA

1111
A=11
K=11
AK

I will try to make it slightly more complex. I am extremely unsure of how I could tie my existing skillset into this problem at this moment.

13422
A=1
C=3
D=4
V=22
ACDV

13422
A=1
C=3
D=4
B=2
B=2

13422
M=13
D=4
V=22
MDV

13422
M=13
D=4

B=2
B=2

I can see some sort of pattern arising
If I start with the widest letter (for instance M=13 as oppose to A=1 C=3)
Then it explores all possible scenarios.
I will now see if this is the case by performing a overlap between the letters with encoded message: 12122

12122
L=12
L=12
B=2
LLB

Now if I try to split the L=12 into A=1 B=2

12122
A=1
B=2
A=1
B=2
B=2
ABABB

But clearly I have missed out several situations:
For instance

12122
A=1
U=21
B=2
B=2
AUBB

12122
A=1
U=21
V=22
AUV

12122
L=12
A=1
V=22
LAV

I am still on stance of keeping it the widest first

12122
L=12
L=12
B=2
LLB

I then know the issue that occurs going forward is the overlaps
A=1
B=2
A=1

and missing out
A=1
U=21

So…
Once I have derived the blocks, I need to examine for each one that if prepending the previous block will give valid number between 10-26
I mention 10 since the minimum it can be is 10 if two digits are appended

So start right from the very first example.
I used the mentality of taking the widest letters first

12122

L=12
L=12
B=2
LLB
Now, I will apply logic mentioned above (NOTE LOGIC WILL ONLY HOLD IF THE STARTING LETTER IS TWO DIGITS WIDE!!!)

It will examine L=12 L=12 B=2
We know the 1 can not be examined since it is the first number
We would need a n=n+2 (when going through the sequence)
We would check 1 and 2 prior to it
This is infact between 10 and 26
U=21
So it would need to truncate last digit off the first L=12 => A=1
Outcome: A=1 U=21 V=22

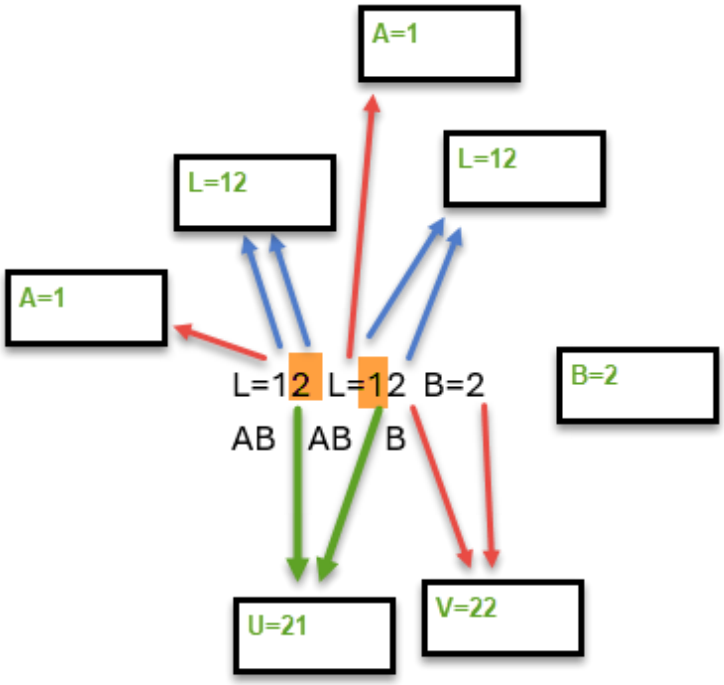
However we can always consider same process on it again (but can see that A=1 is 1 digit wide)....
The outcome would be that it would try to check 1 of the U with 2 of the U.. And it has no outcome

It will examine:
A=1 U=21 V=22

So question is how do I generate the other scenarios:
So I will go for obvious and split each block

L=12 L=12 B=2
AB AB B

It can be seen that the logic breaks down heavily.
Unfortunately my knowledge is not quite built up to use collection such as a tree to examine all possibilities.
But it is clearly becoming obvious that there are permutations involved.
Since we can see on instances we need to consider the following:



So the only outstanding technique is to search for all these mappings. I can consider performing this via:

REGEX which would instantaneously flag up any numbers 1-26.

OR

I can performing an iteration through the initial number.
Alternatively I can attempt to perform via **recursion**. I am slightly in favour of recursion since there will be an extensive amount of class level variables due to the existing recursive Permutation function..

populate the mp (Map) with values 1-26 and A-Z respectively.
This should suffice....

```
for (int i=0;  i<str.length();  i++)

String mapping[count] = Object.toString(mp.getKey(i));
count++;

if (i>0)
{
```

char first = str.indexOf(i-1);
char second = str.indexOf(i);
String num = first + second;
String mapping[count] = Object.toString(mp.getKey(Integer.valueOf(num));
}
count++;

Once it has calculated all the letters, we can populate all of the values in a Map to avoid duplication of Letters.

//We are using String since 10-26 Letters are two digits wide
Map <Integer, String> possibleLetters = new HashMap <Integer, String>
(unsure if I can use Arrrays.asList(mapping) to fill the List.. Or is this just applicable to an Integer array.
(alternative would be using mapping.toArray or something similar.....

I would now use the Permutation calculator (without replacement) to generate all permutations..
Due to this recursive technique we know Java will produce a StackOverFlow once we reach r=64.
As the permutation arrives, I can reverse map it and check if it agrees with initial number and discard it accordingly.....

P(n,r)

n= possibleLetters.size();

r= possibleLetters.size() where
r^{Limit} = possibleLetter.size(); (we know that this would entail each digit with one to one mapping with a Letter)

<u>Even length encoded message</u>

So if str.length()% 2==0 $r^{Minimum}$ = str.length()/2 =3 (For instance: 25252)

<u>Odd length encoded message</u>

if str.length() % 2 ==1 $r^{Minimum}$ = str.length()+1/2 =3 (For instance: 25252)

This is the case since we know that each individual digit can be mapped to a Letter since the question states that the encoding message will be valid.

I can also try to incorporate some intelligence during the permutation selection process.
If the single selection or chain selection is deemed unsuitable, I can ensure that it is not populated in the List at first instance. And cycle starts again. This would be similar to the Google triangle challenge.
BUT NOTE: Due to the complexity of the permutations, I can foresee this generating several false positives. I think I can ONLY discard permutations based on selection of the first two Letter mappings ONLY due to the way that numbers can intertwine into various Letters. This is still better than

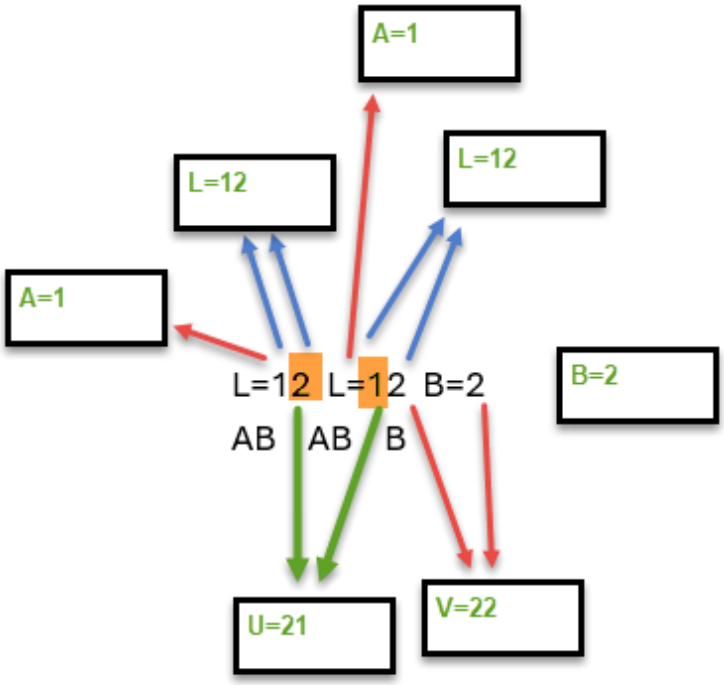
This will ensure that any intermittent outputs to the end user are all viable during mid-execution..... If I choose not to use an iterator to process the contents of the Map Collection (which will store invalid permutation), it is quite possible the entire challenge can be completed without any iteration AT ALL.

Again, I am not sure of the pro vs con, but I can imagine the code appearing a bit sharper without all the indexing....

I will give it a try first anyhow..
(NOTE: it will be recursive calls to different recursive methods).

I attempted to code straight away basing it on the Google Triangle challenge.
However I missed out critical area of logic.

I have managed to fill a map with all the mappings:



Map index 0 = A
Map index 1 = B
Map index 2 = A
Map index 3 = B
Map index 4 = B

So clearly, I needed to collate all the values and then perform P(N,2) and append values in which it is less than or equal to 26.
This code is going to get extremely involved..
And it is best I just leave the code aside.

TEST PHASE

I have verified that that all correct letters are stored in the possibleValues MAP...

This code is just experimental from here onwards....