Problem To Solve

On the old Nokia and Verizon phones, when the complete keyboard was absent, users were expected to use their numeric keypad with just 9 keys to type in all 26 characters of the English alphabet.

The keys and the corresponding letters were

In [ ]:
```
2 abc
3 def
4 ghi
5 jkl
6 mno
7 pqrs
8 tuv
9 wxyz
```

If a person types in '2' he could mean either 'a' or 'b' or 'c'.

If a person types in 23 he could mean either [ad or ae or af or bd or be or bf or cd or ce or cf].

In this problem, we try to guess, that if the person keys in a particular sequence of numeric keys on the keypad, what is the most likely word which he was trying to enter?

An outline of steps for building a Unigram model

1. A dictionary of commonly used words be provided to you.
2. Apart from that you are also provided a large corpus of text. Using this corpus of text, you can compute the frequency with which commonly used words (from the dictionary) occur in the corpus. i.e., you are computing Unigram Frequencies, using the corpus. There might be words in the corpus which are missing in the dictionary: these words can be ignored. But do not ignore words which are present in the dictionary and absent in the corpus. Let's say, the dictionary contains the words

In [ ]:
```
Hello,I,am
```

and suppose the corpus text is

In [ ]:
```
Hello, I will be going to the mall tomorrow. I am out of groceries.
```

The weighted dictionary (with frequencies) you will come up with is:

In [ ]:
```
[Hello=>2,I=>3,am->2]
```

Because Hello occurs once in the corpus text and once in dictionary, 'I' occurs twice in the corpus and once in dictionary and 'am' occurs once in corpus and dictionary.

Predicting the word, given a series of numerals

After you read the dictionary and corpus and build the language model, you are given a number of numeric sequences typed in by a phone user. You need to identify words from the dictionary which start with a character sequence which could be represented by these numerals. Identify the top five candidates with the highest frequency, and output them in one line, separated by semi-colons. If there are less than five possible candidates, display them all. If there is no possible candidate, display

In [ ]:
```
No Suggestions
```

Most likely word is the one which occurs max times in the given corpus, least likely is the one which occurs least times in the given corpus (or, perhaps it is a word which exists only in the dictionary and did not occur at all in the corpus).

Dictionary and Corpus File

For the purpose of building the word frequency and unigram model, you are provided with a file t9Dictionary and t9TextCorpus which will be kept in the same folder as the one from which your program is being run.

The first file t9Dictionary, is the dictionary. First line contains N, N words follow each in a new line. The second file is the training corpus t9TextCorpus of text. This ends with "END-OF-CORPUS" on a new line. Defining a word

A word is a sequence of characters (a-z, lowercase or uppercase; hyphen or apostrophe) which always starts and ends with a letter (a-z, lowercase or uppercase). The regex used must be greedy.

Input Format

First line will contain the number of tests T. This will be followed by T lines containing numeric sequences/numbers N.

Constraints

In [ ]:
```
1 <= T <= 20
2 <= N <= 10^10
```

Each digit in N is between 1 to 9.

Output Format

Each line will contain a list of top five semi-colon separated candidate words, with the leftmost word being the most frequently used. If the group of five words contains multiple words with the same frequency, sort them in lexicographical order. If there were less than five candidates for some input, display all of them. If there were no candidates for some input sequence, display "No Suggestions".

Sample Input

In [ ]:
```
6
6837
```

```
86
69
23777
11111
77777
```

Sample Output

In [ ]:

```
over;overlying;overcome;overcoat;overgrowth
to;under;took;united;too
my;own;myself;owing;owners
cesspool;cesspool's;cesspools
No Suggestions
No Suggestions
```

Explanation

6837, means that we are looking for words where the first character is either 'm' or 'n' or 'o' second character is either 't' or 'u' or 'v' third character is either 'd' or 'e' or 'f' fourth character is either 'p' or 'q' or 'r' or 's'.

We select only the words from the dictionary, which match the above criteria, and we sort them in descending order of their frequencies, and display the first five. 'over' is the most frequent one, followed by 'overlying' and so on.

Similarly, we process other inputs.

In the last case, we don't have any words in the dictionary corresponding to the numeric sequence 77777, so we display "No Suggestions".

Scoring

Your score for a test case will be

(maxScore for the test case)*(nCorrect)/(nTotal)

nCorrect = Number of lines matching with the expected output nTotal = Total number of lines in the expected output

Solution

In [ ]:

```
import collections
import sys
```

Defining a word: A word is a sequence of characters (a-z, lowercase or uppercase; hyphen or apostrophe) which always starts and ends with a letter (a-z, lowercase or uppercase). The regex used must be greedy.

In [ ]:

```
def rplc(line):
    line = line.replace('.', ' ')
    line = line.replace(',', ' ')
```

```
    line = line.replace('?', ' ')
    line = line.replace('!', ' ')
    line = line.replace(':', ' ')
    line = line.replace(';', ' ')
    line = line.replace('"', ' ')
    line = line.replace('(', ' ')
    line = line.replace(')', ' ')
    line = line.replace('1', ' ')
    line = line.replace('2', ' ')
    line = line.replace('3', ' ')
    line = line.replace('4', ' ')
    line = line.replace('5', ' ')
    line = line.replace('6', ' ')
    line = line.replace('7', ' ')
    line = line.replace('8', ' ')
    line = line.replace('9', ' ')
    line = line.replace('\n', ' ')
    line = line.replace('_', ' ')
    return line.strip()
```

given prefix of word, give me all words which starts with that prefix for example dict: [karo, karolina, slaw] if prefix k, then [karo, karolina] if prefix karol, then [karolina]

In [ ]:

```
def create_prefixes(d):
    prefixes = {}
    for w in d:
        word = w[0]
        if not prefixes.get(word, None):
            prefixes[word] = [w]
        else:
            prefixes[word].append(w)
        for let in w[1:]:
            word += let
            if not prefixes.get(word, None):
                prefixes[word] = [w]
            else:
                prefixes[word].append(w)
    return prefixes
```

initial state

In [ ]:

```
def get_available_words(d, words_prefixes, seq):
    keyboard = {2:['a','b','c'], 3:['d','e','f'], 4:['g','h','i'], 5:['j','k','l'], 6:[
                7:['p','q','r','s'], 8:['t','u','v'], 9:['w','x','y','z']}
    if '1' in seq:
        return []
    seq_idx = 0
    next_prefixes = keyboard.get(int(seq[seq_idx]), None)
    last_prefixes = None
    avaliable_words = []
    for p in next_prefixes:
        if words_prefixes.get(p, None):
            avaliable_words.extend(words_prefixes[p])
    while (seq_idx < len(seq)-1) and (len(avaliable_words) > 0):
        last_prefixes = next_prefixes
        next_prefixes = keyboard[int(seq[seq_idx+1])]
```

```
        avalable_prefixes = [l+n for l in last_prefixes for n in next_prefixes]
        new_available_words = []
        remained_prefixes = []
        for prefix in avalable_prefixes:
            for word in avaliable_words:
                if word.startswith(prefix) and word not in new_available_words:
                    new_available_words.append(word)
                    if not prefix in remained_prefixes:
                        remained_prefixes.append(prefix)
        avaliable_words = new_available_words
        next_prefixes = remained_prefixes
        seq_idx += 1
    return avaliable_words
```

read files

In [ ]:

```
def main(seqs):
    with open('t9Dictionary', 'r') as dict_file:
        t9_dict = [line.strip() for line in dict_file.readlines()]
    with open('t9TextCorpus', 'r') as corpus_file:
        t9_corp = ''
        line = rplc(corpus_file.readline())
        while not line == 'END-OF-CORPUS':
            t9_corp += line + ' '
            line = rplc(corpus_file.readline())
```

build model

In [ ]:

```
simple_tokens = [w.strip() for w in t9_corp.split(' ') if not w.strip() == '']
simple_tokens.extend(list(set(t9_dict)))
```

additional cleaning of tokens

In [ ]:

```
simple_tokens = [t[:-1] if t.endswith("'") else t for t in simple_tokens]
cnts_dict = collections.Counter(simple_tokens)
```

weigted dict is only from words within t9_dict

In [ ]:

```
    weigted_dict = {}
    for w in t9_dict:
        weigted_dict[w] = cnts_dict[w]
    prefixes = create_prefixes(t9_dict)
    for seq in seqs:
        words = get_available_words(t9_dict, prefixes, seq)
        words_f = sorted([(w, weigted_dict[w]) for w in words], key=lambda x: (-x[1], x
        if len(words_f) > 0:
            print(';'.join([x[0] for x in words_f[:5]]))
        else:
            print('No Suggestions')
```

```
seqs = [line.strip() for line in sys.stdin]
main(seqs[1:])
```

In [ ]: