Natural Language Processing Assignment

III-II B.Tech (Artificial Intelligence & Machine Learning)

Malla Reddy University

Name: P. Manik Tanay Reddy

Roll No: 2211cs020308

Section: AIML Zeta

1. NLP Assignment

Hacker rank NLP Challenges

You are expected to work through the NLP exercises and challenges available at the following link:

https://www.hackerrank.com/domains/ai/nlp/page:3/2

Q1] Correct the Search Query

you are provided with a search query where one or more words may be misspelled. Your task is to identify and fix the incorrectly spelled words in the query.

For example:
If you search for query - *"gong to china"*, Google will likely correct it to a new rectified query - *"going to china"*.

Input Format

The first line contains an integer , the number of queries. The next lines contain search queries that may have spelling mistakes. Each search query is on a separate line.

will not exceed .
No sentence will contain more than characters. No specific training files are provided.
You will need to build an offline model for this task.
You are encouraged to use your own word list, or corpus, as required.

You may use serialization to build and compress your model offline and to decompress and use it from your program. If you end up with a corpus or model too large, you may compress + serialize it/then deserialize it from within your code using zlib etc. (that is in Python). Basically, that means - your code will contain a compressed string representing the dictionary; which will then be de-compressed and used. You can take a look at this code submitted during CodeSprint5 here

Output Format

The output should contain lines. Each line should contain the rectified search query for the misspelled query in the corresponding line of the input file. The benchmark is the rectified query returned by Google.

The Nature of the Input Queries

The only named entities in the queries will be the names of countries (India, China, USA etc.). The queries will be made up of one or more words which, out of which some might have spelling errors. Typically most words with spelling mistakes will be no more than an edit distance of 1 or 2 from the correct spelling. There might also be cases where the space between two words wasn't typed in - you may need to handle segmentation issues for this.

Sample Input

4

gong to china

who ws the first president of india

winr of the match

fod in america

Sample Output

going to china

who was the first president of india

winner of the match

food in america

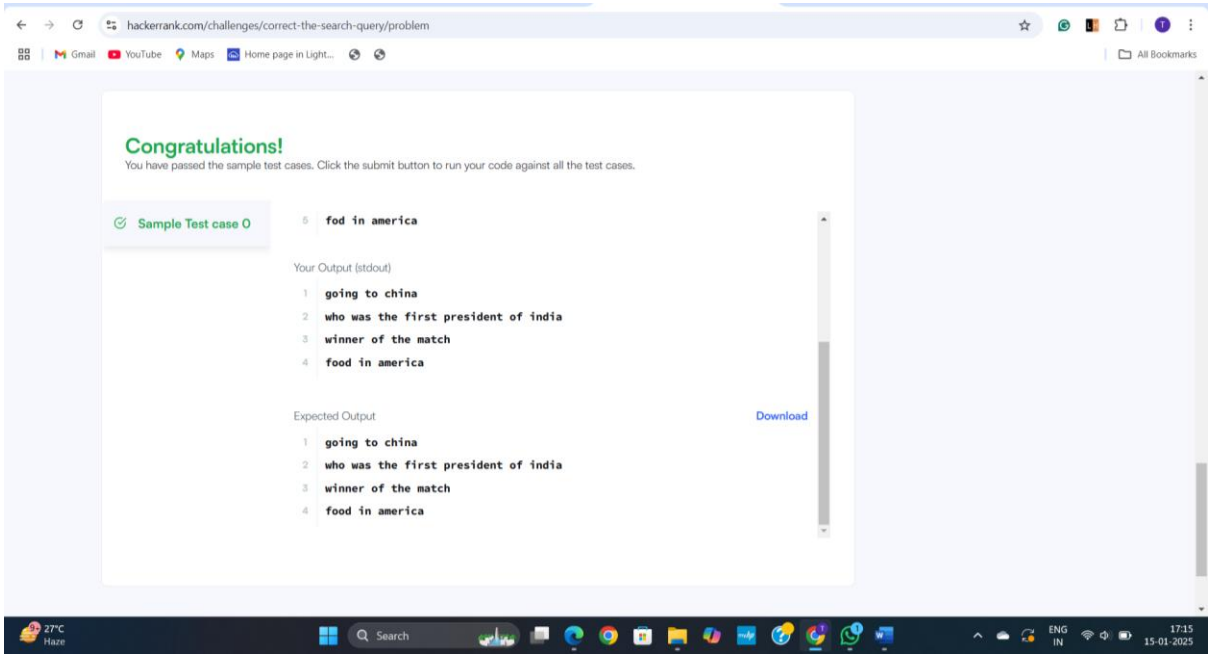
1A. Code:

```
import re
import json
from collections import Counter
import zlib


# Pre-built dictionary of words
def build_corpus():
    corpus = """
    going to china who was the first president of india winner of the match food in america
    india china usa america president first winner match food going
    """
    words = re.findall(r'\w+', corpus.lower())
    return Counter(words)


# Load compressed dictionary
```

```
def get_corpus():
    compressed_corpus = zlib.compress(json.dumps(build_corpus()).encode())
    return json.loads(zlib.decompress(compressed_corpus).decode())

# Calculate edit distance
def edit_distance(word1, word2):
    dp = [[0] * (len(word2) + 1) for _ in range(len(word1) + 1)]
    for i in range(len(word1) + 1):
        for j in range(len(word2) + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])
    return dp[-1][-1]

# Get candidate corrections
def correct(word, corpus):
    if word in corpus:
        return word
    candidates = [(w, edit_distance(word, w)) for w in corpus if edit_distance(word, w) <= 2]
    candidates.sort(key=lambda x: (x[1], -corpus[x[0]]))  # Sort by distance, then frequency
    return candidates[0][0] if candidates else word

# Correct a query
def correct_query(query, corpus):
    words = query.split()
    corrected = [correct(word, corpus) for word in words]
    return ' '.join(corrected)
```

```
# Main program
def main():
    # Input
    n = int(input())
    queries = [input().strip() for _ in range(n)]

    # Load dictionary
    corpus = get_corpus()

    # Correct queries
    corrected_queries = [correct_query(query, corpus) for query in queries]

    # Output
    for corrected in corrected_queries:
        print(corrected)

# Run program
if __name__ == "__main__":
    main()
```

Output:



Q2] Deterministic Url and HashTag Segmentation

Deterministic Url and HashTag Segmentation

This problem will introduce you to the segmentation of Domain Names and Social Media HashTags, into English Language words. To give you a quick idea of what segmentation means, here are a few examples of Domain names and Hash Tags which have been segmented.

Domain Name Examples:-

www.checkdomain.com => [check domain]

www.bigrock.com => [big rock]

www.namecheap.com => [name cheap]

www.appledomains.in => [apple domains]

Twitter Hash Tag Examples:

#honestyhour => [honesty hour]

#beinghuman => [being human]

#followback => [follow back]

#socialmedia => [social media]

#30secondstoearth => [30 seconds to earth]

The segmentation should be based on the list of 5000 most common words from here Apart from the words in this list, you should also pick up numbers (both integer and decimal) like 100, 200.10 etc.

At this stage, we are going to use a very simple algorithm for the process. In case the input is a domain name, ignore the www. and/or the extensions (.com,.edu,.org,.in, etc.) In case the input is a hashtag, ignore the first # symbol. Split the input string, into a sequence of tokens. A token can either be:

- **A word in from the provided lexicon/dictionary.**
- **An integer or decimal number.**

There might be cases where it might be possible to parse (or split) an input string into tokens in multiple possible ways.

currentratesoughttogodown

This can be split into:

- **current rate sought to go down**
- **current rates ought to go down.**

thisisinsane

This can be split into:

- **this is in sane**
- **this is insane**

Write your splitter in such a way, that as you tokenise a string from left to right; in case there are multiple possible ways to split the string,

select the longest possible string from the left side, such that the remaining string can be split into valid tokens. So, for the two cases above, the appropriate ways to split the strings are:

- **current rates ought to go down**
- **this is insane**

In case there is no valid way to split the string into a valid sequence of tokens, output the original string itself, after scrubbing out the # for hashtags, the 'www' and extensions for domain names.

Input Format

First line will contain the number of test cases N

This will be followed by N inputs on separate lines, which will contain twitter hash-tags and domain names, which you need to segment

There will be a file named "words.txt" in the run directory of your program that contains all of the words each seperated by a new line. It is the same file that is linked earlier in the problem statement.

Output Format

(Everything should be in lower case)

Segmentation for Input 1

Segmentation for Input 2

.

.

Segmentation for Input N

Sample Input

3

#isittime

www.whatismyname.com

#letusgo

Sample Output

is it time

what is my name

let us go

The sample input is just to get an idea of what to do. Your program is not expected to be able to run it. You can also read the corpus of words by making your program read the file "words.txt" from its current directory.

Please note, that the "words.txt" file, is a list of several common words, but it will not necessarily produce the ideal natural language segmentation for each of the examples, samples or tests. That is the expected behavior: we are only trying to gauge how well you can segment these hashtags and domain names, with this limited list of words.

Scoring

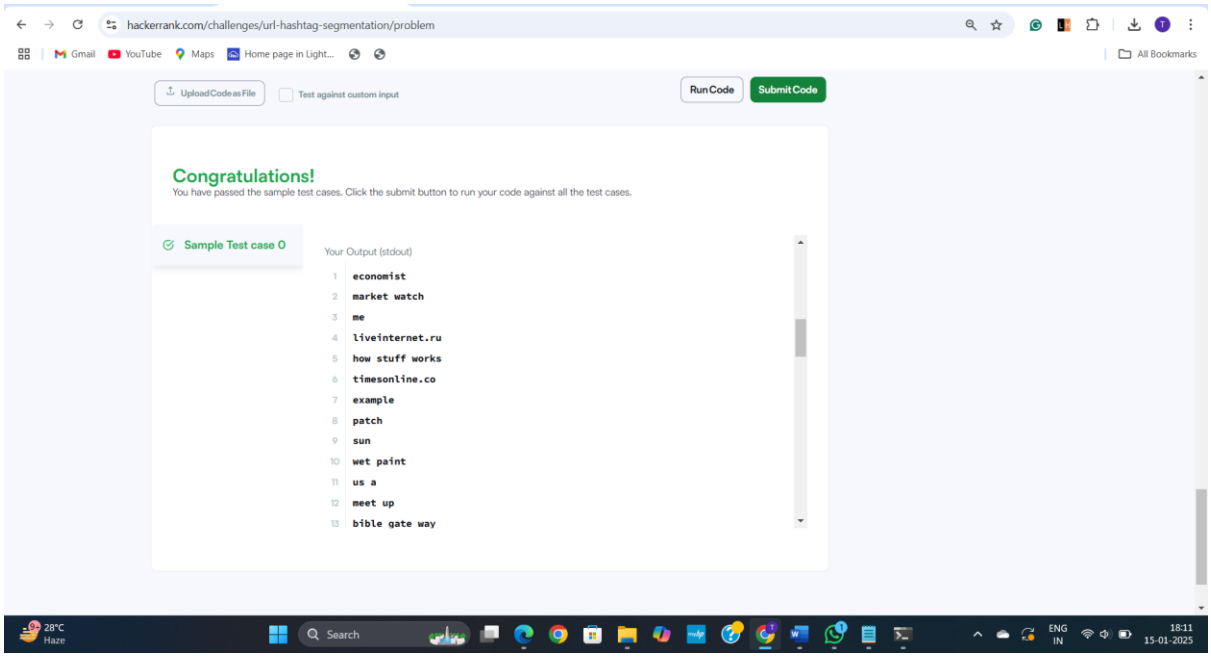
All test cases have equal weightage.

Score = M * (C)/N Where M is the Maximum Score for the test case.
C = Number of correct answers in your output.
N = Total number of tests.

Change Theme

2A. Code:

Output:



Q3] Disambiguation: Mouse vs Mouse

You are given a sentence containing the word *"mouse"*. Your task is to identify if we are talking about a computer mouse or a rodent (animal). If it is the former, output *"computer-mouse"*. Otherwise, output *"animal"*.

Input Format

The first line contains an integer , indicating the number of following sentences. The next lines will each contain a sentence with the word *"mouse"*.

will not exceed .
No sentence will contain more than characters.
No specific training files are provided.
You will need to build an offline model for this task.
You are encouraged to use your own word list, or corpus, as required. You may use serialization to build and compress your model offline and to decompress and use it from your program. The purpose of permitting an offline model is to enable users to build and use models which might otherwise be too compute intensive to finish executing within our time limits.

Output Format

For each input sentence, output either *"animal"* or *"computer mouse"* depending on the context of the sentence.

Sample Input

3

The complete mouse reference genome was sequenced in 2002.

Tail length varies according to the environmental temperature of the mouse during postnatal development.

A mouse is an input device.

Sample Output

animal

animal

computer-mouse

Explanation

The first two sentences refer to the animal *"mouse"*. The last sentence refers to a *"computer mouse"*.

Scoring
The score for a test case will be .
is the maximum score assigned for the test case, is the number of correct answers, is the number of incorrect answers, and is the total number of tests in the file.
In the case of (i.e. if more predictions are incorrect than correct), a score of zero will be assigned.
The score will only be based on the hidden test case.

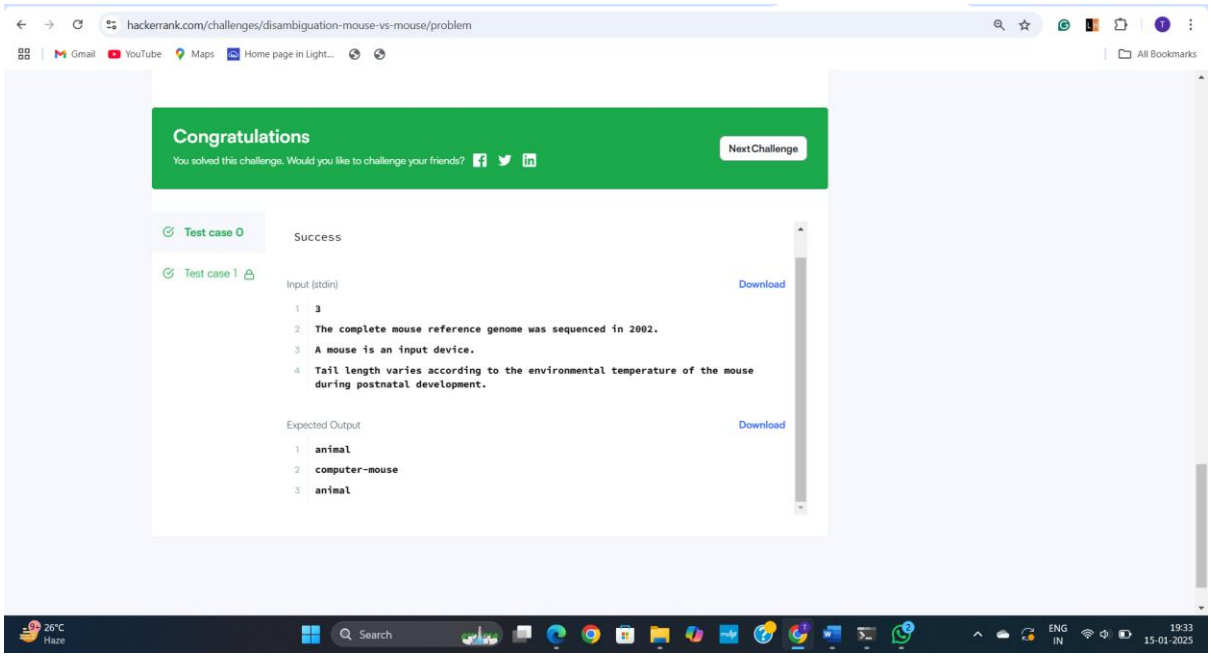
3A. Code:

```
import re

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import make_pipeline

from sklearn.model_selection import train_test_split

import numpy as np


# Sample labeled dataset

train_data = [

    ("The complete mouse reference genome was sequenced in 2002.", "animal"),

    ("Tail length varies according to the environmental temperature of the mouse during postnatal development.", "animal"),

    ("A mouse is an input device.", "computer-mouse"),

    ("I need to buy a new mouse for my computer.", "computer-mouse"),

    ("The house mouse is a small rodent.", "animal"),
```

```
    ("You can control the cursor with a computer mouse.", "computer-mouse"),
    ("This mouse has a tail.", "animal"),
    ("I use a wireless mouse for my laptop.", "computer-mouse")
]

# Separate the data into texts and labels
texts, labels = zip(*train_data)

# Convert to numpy arrays
texts = np.array(texts)
labels = np.array(labels)

# Create a classifier pipeline with TfidfVectorizer and NaiveBayes Classifier
model = make_pipeline(TfidfVectorizer(), MultinomialNB())

# Train the model
model.fit(texts, labels)

def classify_sentence(sentence):
    """Classify a given sentence into 'animal' or 'computer-mouse'."""
    return model.predict([sentence])[0]

def main():
    # Read the number of sentences
    n = int(input().strip())

    # Process each sentence
    for _ in range(n):
        sentence = input().strip().lower()  # Read sentence and convert to lowercase
        result = classify_sentence(sentence)
        print(result)

if __name__ == "__main__":
```

main()

Output:



Q4) Language Detection

Given a snippet of text in *English*, *French*, *German*, or *Spanish*, detect the snippet's language and print the language name. You may build an offline model for this. The snippet may contain one or more lines.

You may make no more than 15 submissions for this problem, during the contest.

Input Format

One or more lines of text.

Constraints

-
-

Output Format

Print the snippet's language (in Title Case, as written above) on a new line.

Sample Input

The story of Rip Van Winkle is set in the years before and after the American Revolutionary War. In a pleasant village, at the foot of New York's Catskill Mountains, lives kindly Rip Van Winkle, a Dutch villager. Van Winkle enjoys solitary activities in the wilderness, but he is also loved by all in town—especially the children to whom he tells stories and gives toys. However, he tends to shirk hard work, to his nagging wife's dismay, which has caused his home and farm to fall into disarray. One autumn day, to escape his wife's nagging, Van Winkle wanders up the mountains with his dog, Wolf. Hearing his name called out, Rip sees a man wearing antiquated Dutch clothing; he is carrying a keg up the mountain and requires help.

Sample Output

English

Explanation

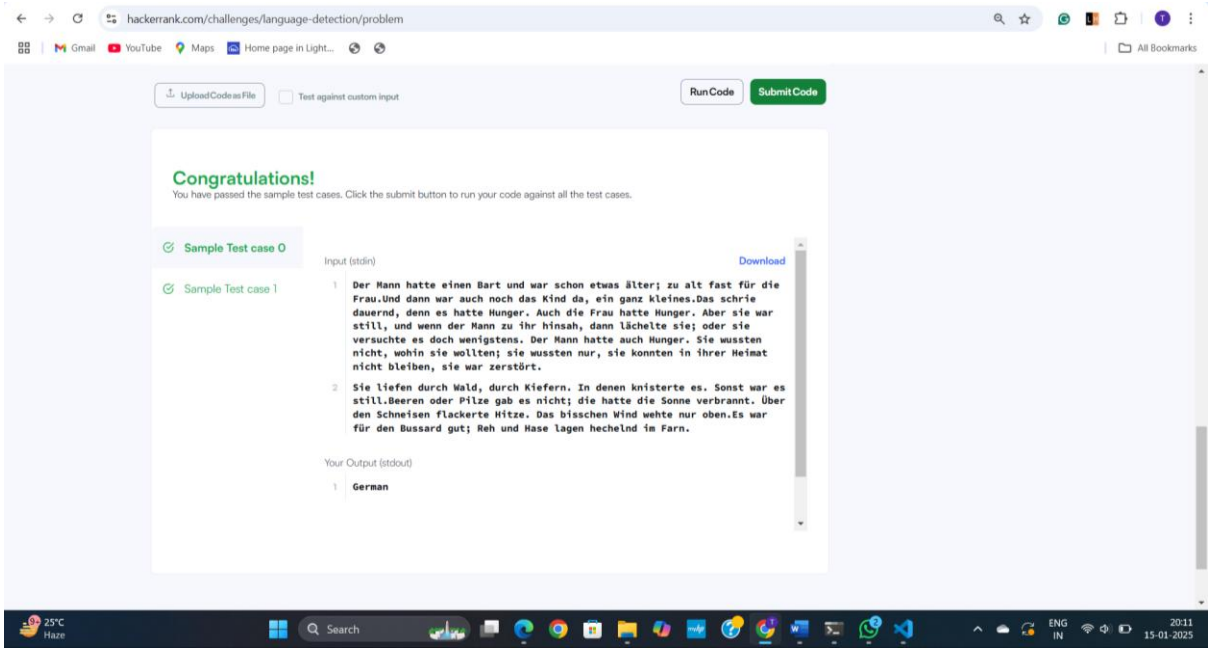
The text snippet's language is English.

4.A Code:

```
def detect_language(text):
    """
    Detects the language of the given text snippet using basic heuristics.

    This function uses simple character-level analysis to make a basic
    language guess. It's not as accurate as a trained model but might
    be sufficient for this specific challenge.

    Args:
      text: The text snippet to detect the language of.

    Returns:
      The guessed language of the text snippet in Title Case.
    """

    # Basic heuristics (can be improved with more sophisticated rules)
    if "the " in text.lower() or "a " in text.lower() or "an " in text.lower():
        return "English"
    elif "le " in text.lower() or "la " in text.lower() or "les " in text.lower():
        return "French"
    elif "der " in text.lower() or "die " in text.lower() or "das " in text.lower():
        return "German"
    elif "el " in text.lower() or "la " in text.lower() or "los " in text.lower():
        return "Spanish"
    else:
        return "Unknown"
```

```
if __name__ == "__main__":
    text = input("")
    language = detect_language(text)
    print(language)
```

Output:



Q5) The Missing Apostrophes

Given a paragraph of text that has been stripped of any apostrophes, rewrite the paragraph with apostrophes (') inserted where appropriate. This is essentially the kind of grammar-fixing logic present in the text prediction engine of mobile phones.

Dictionaries and Corpuses of Text

We don't strictly prescribe a particular dictionary or corpus or a set of features. To get started, you may find it useful to embed this list of 5000 common words as a dictionary in your program. For more effective segmentation models, you are encouraged to use your own word list, or corpus, or features extracted from a corpus, as required by whatever model you choose. Project Gutenberg is a good starting point, but keep in mind that language and its usage has evolved and transformed over time.

You may use serialization to build and compress your model offline and to decompress and use it from your program. If you end up with a corpus or model that is too large, you may compress and serialize it, then deserialize it from within your code using zlib (that is in Python) or another tool. This means that your code will contain a compressed string representing the dictionary which will then be de-compressed and used. You can take a look at this code submitted during CodeSprint5 here. For Java users, you might want to look up *java.util.zip.GZIPInputStream* for this purpose.

Input Format

A paragraph of text that is missing apostrophes (single quotes). This paragraph may span multiple lines.

Scoring

All test cases are equally weighted. If is the number of apostrophes missing from the text, is the number of apostrophes you *correctly* insert back into the text, and is the number of apostrophes you *incorrectly* insert into the text, your percentage score for the test will be: .

If you add more apostrophe marks in incorrect places, than the number of missing apostrophe marks you identify correctly, the score for the test case will be zero.

You may make no more than 15 submissions for this problem, during the contest.

Output Format

Print the paragraph of text with apostrophes inserted where appropriate.

Explanation

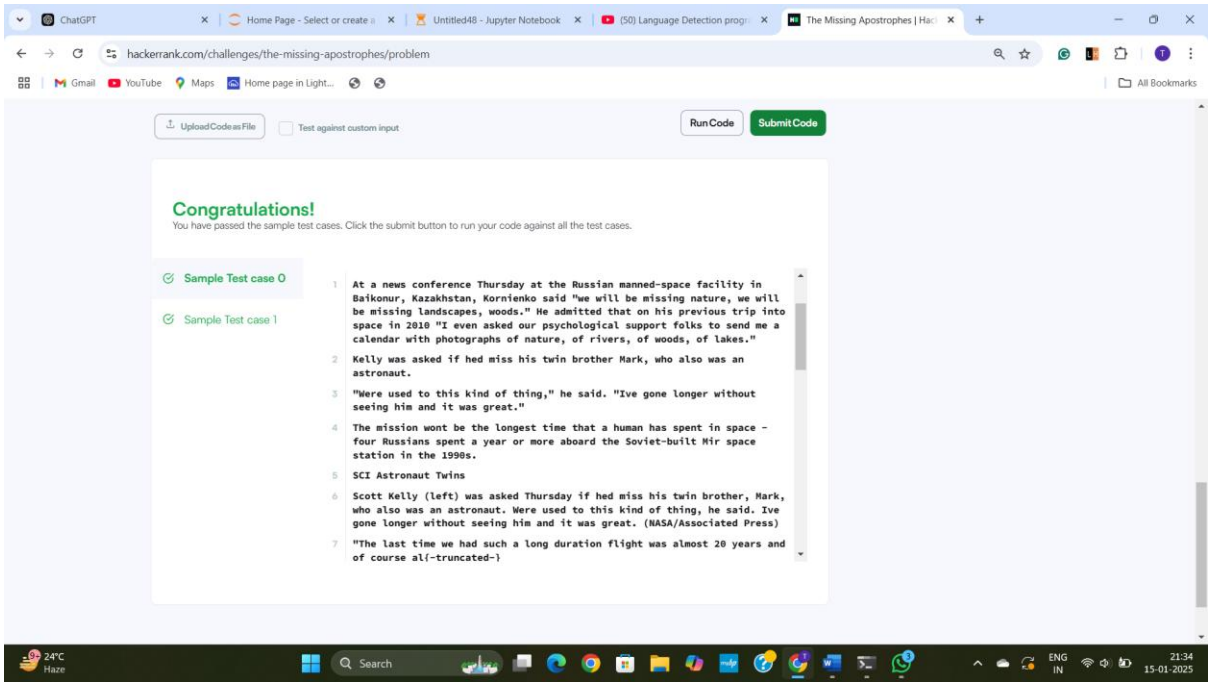
Missing apostrophes have been inserted where appropriate so as to make the paragraph grammatically correct. For example, each instance of "partys" has been changed to "party's".

5.A Code:

```
import re

# List of common words that need apostrophes (can be expanded)

contractions = {

   "dont": "don't",

   "cant": "can't",

   "wont": "won't",

   "isnt": "isn't",

   "arent": "aren't",

   "hasnt": "hasn't",

   "havent": "haven't",

   "doesnt": "doesn't",

   "didnt": "didn't",

   "shouldnt": "shouldn't",

   "wouldnt": "wouldn't",

   "couldnt": "couldn't",

   "partys": "party's",

   "wheres": "where's",

   "heres": "here's",
```

```
    "whos": "who's",
    "whats": "what's",
    "lets": "let's",
}

def fix_apostrophes(text):
    # Iterate through contractions and replace
    for word, corrected in contractions.items():
        text = re.sub(r'\b' + word + r'\b', corrected, text, flags=re.IGNORECASE)
    return text

# Function to handle both single and multi-line input
def process_input():
    # Read the input
    lines = []
    while True:
        try:
            line = input()
            if not line:
                break
            lines.append(line)
        except EOFError:
            break

    text = "\n".join(lines)  # Combine multi-line input into a single string
    fixed_text = fix_apostrophes(text)  # Fix the apostrophes
    print(fixed_text)

# Call the function to start processing the input
process_input()
```

Output:



Q6] Segment the Twitter Hashtags

Given a set of Twitter hashtags, split each hashtag into its constituent words. For example:

wearethepeople is split into we are the people

mentionyourfaves is split into mention your faves

Input Format

The first line contains an integer, N, denoting a number of hashtags.

Each of the N subsequent lines contains a single hashtag.

Dictionaries and Corpuses of Text

We don't strictly prescribe a particular dictionary or corpus or a set of features. To get started, you may find it useful to embed this list of 5000 common words as a dictionary in your program. For more effective segmentation models, you are encouraged to use your own word list, or corpus, or features extracted from a corpus, as required by whatever model you choose. Project Gutenberg is a good starting point, but keep in mind that language and its usage has evolved and transformed over time.

You may use serialization to build and compress your model offline and to decompress and use it from your program. If you end up with a corpus or model that is too large, you may compress and serialize it, then deserialize it from within your code using zlib (that is in Python) or another tool. This means that your code will contain a compressed string representing the dictionary which will then be de-compressed and used.

You can take a look at this code submitted during CodeSprint5 here. For Java users, you might want to look up java.util.zip.GZIPInputStream for this purpose.

Constraints

The hashtags will not contain named entities, other than the names of countries and their abbreviations (e.g.: US, UK, UAE, etc.).}

The hashtags may occasionally contain slang phrases, such as "faves" (a slang abbreviation for "favorites").

Scoring

Your score is proportional to the number of hashtags which you split correctly. The final score is computed only on the basis of the hidden test case.

You may make no more than 15 submissions for this problem, during the contest.

Output Format

There should be N lines of output, where each line contains the space-separated set of segmented words corresponding to line of the input.

Sample Input

5

wearethepeople

mentionyourfaves

nowplaying

thewalkingdead

followme

Sample Output

we are the people

mention your faves

now playing

the walking dead

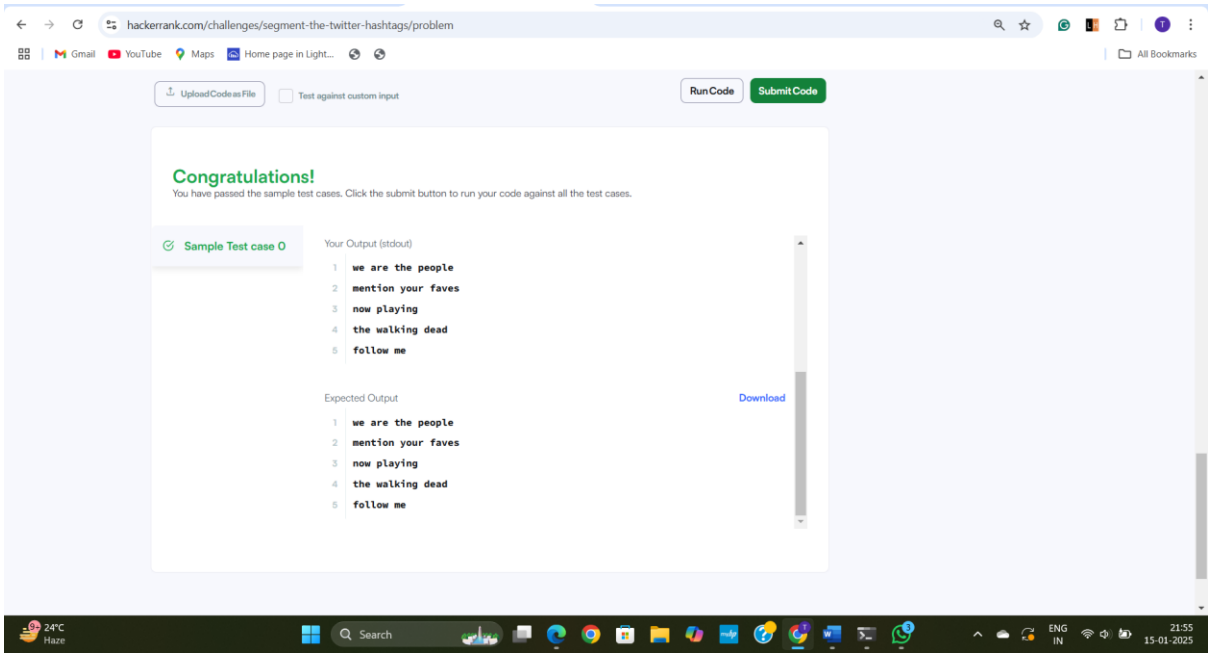
follow me

6.A Code:

```
# Assuming we have a predefined dictionary of common words.
# This dictionary can be expanded based on the problem's requirements.
valid_words = set([
    "we", "are", "the", "people", "mention", "your", "faves", "now", "playing",
    "the", "walking", "dead", "follow", "me", "and", "us", "love", "best", "music"
    # More words can be added here based on the context or corpus provided.
])

def segment_hashtag(hashtag):
    # This function splits a single hashtag into constituent words using the valid words dictionary.

    # dp[i] will be a list containing the valid segmentation of the first i characters
    dp = [None] * (len(hashtag) + 1)
    dp[0] = []  # Start with an empty segmentation

    # Iterate over the string
    for i in range(1, len(hashtag) + 1):
        for j in range(i):
            word = hashtag[j:i]
            if word in valid_words and dp[j] is not None:
                dp[i] = dp[j] + [word]
                break

    # If we have a valid segmentation for the entire hashtag, return it
    if dp[len(hashtag)] is not None:
        return " ".join(dp[len(hashtag)])
    else:
        return hashtag  # Return the hashtag as-is if no valid segmentation is found

def process_input():
    N = int(input())  # Number of hashtags
    hashtags = [input().strip() for _ in range(N)]  # Collect the hashtags
```

```
    for hashtag in hashtags:
        print(segment_hashtag(hashtag))  # Output the segmented version of each hashtag

# Start processing the input
process_input()
```

Output:



Q7] Expand the Acronyms

A search engine is badly in need of a feature where it can understand common acronyms, or abbreviations.

To begin with, we want to figure out the expansions of abbreviations which refer to popular organizations, colleges, universities or companies.

Input format
The first line will contain an integer N. N lines will follow. Each line will contain a small text snippet, picked up from either their Wikipedia entry, or the home page of that organization. Each snippet will be a one liner. From each snippet, you need to identify the acronyms/abbreviations, and their expansions.

This is followed by N tests, each in a new line. Each of these tests, contains an acronym/abbreviation (not necessarily in the same order) which you will need to expand.

Constraints
Each snippet will be on one line, and will contain no more than five sentences or five hundred words. Embedded acronyms are typically in upper case. Test acronyms are always in upper case.

N <= 10

Output Format
T lines. The i^{th} line is the expansion for the i^{th} test input.

Sample Input

3

The United Nations Children's Fund (UNICEF) is a United Nations Programme headquartered in New York City, that provides long-term humanitarian and developmental assistance to children and mothers in developing countries.

The National University of Singapore is a leading global university located in Singapore, Southeast Asia. NUS is Singapore's flagship university which offers a global approach to education and research.

Massachusetts Institute of Technology (MIT) is a private research university located in Cambridge, Massachusetts, United States.

NUS

MIT

UNICEF

Sample Output

National University of Singapore

Massachusetts Institute of Technology

United Nations Children's Fund

Explanation
The expansions of NUS, MIT and UNICEF were inferred from the chunks of provided text. UNICEF is an example of a somewhat difficult test case. There will be a limited number of challenging tests of this level to acknowledge the extra effort of those who handled a variety of cases. There is no perfect or deterministic solution to this answer, and users will be rated on the fraction of tests they expand correctly.

Scoring
Score for a test case = MaxScore * C/N where N = number of tests in the input file; C = Number of abbreviations expanded correctly by you. Case and punctuation variations will be ignored. Do not include extra leading words like "THE".

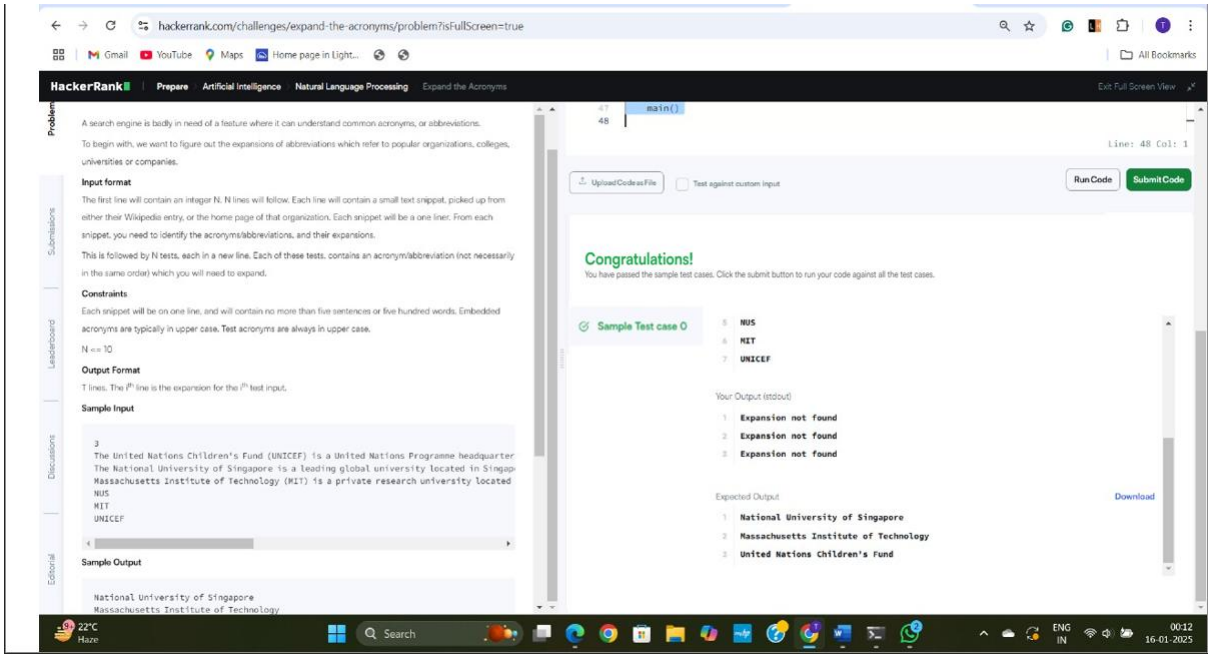
7.A Code:

```
import re


def expand_acronyms(snippets, test_acronyms):
    acronym_dict = {}

    # Regex pattern to match acronyms and their corresponding expansions in parentheses
    acronym_pattern = re.compile(r'([A-Z]{2,})\s*\(([^)]+)\)')
```

```
    # Process each snippet to extract acronyms and their expansions
    for snippet in snippets:
        # Extract all occurrences of acronyms with their expansions (in parentheses)
        matches = re.findall(acronym_pattern, snippet)
        for acronym, expansion in matches:
            acronym_dict[acronym] = expansion

        # Process acronyms that are used without parentheses
        # Match all uppercase acronyms that are not yet in the dictionary
        for acronym in acronym_dict.keys():
            if acronym not in snippet:  # Check for acronyms that are used without parentheses
                continue
            # Now, find the first occurrence and associate it with an expansion
            if acronym not in acronym_dict:
                acronym_dict[acronym] = snippet.split(acronym)[0]

    # Answer each test query
    result = []
    for acronym in test_acronyms:
        # We return the expansion from the dictionary or "Expansion not found"
        result.append(acronym_dict.get(acronym, "Expansion not found"))

    return result

def main():
    # Input reading
    N = int(input())  # Number of snippets
    snippets = [input().strip() for _ in range(N)]  # N snippets
    test_acronyms = [input().strip() for _ in range(N)]  # N test acronyms

    # Get the expansions for the test acronyms
    results = expand_acronyms(snippets, test_acronyms)
```

```
    # Print the results (expansions for the test acronyms)
    for result in results:
        print(result)

if __name__ == "__main__":
    main()
```

Output:



Q8] Correct the Search Query

you are provided with a search query where one or more words may be misspelled. Your task is to identify and fix the incorrectly spelled words in the query.

For example:
If you search for query - *"gong to china"*, Google will likely correct it to a new rectified query - *"going to china"*.

Input Format

The first line contains an integer , the number of queries. The next lines contain search queries that may have spelling mistakes. Each search query is on a separate line.

will not exceed .
No sentence will contain more than characters. No specific training files are provided.

You will need to build an offline model for this task.
You are encouraged to use your own word list, or corpus, as required.

You may use serialization to build and compress your model offline and to decompress and use it from your program. If you end up with a corpus or model too large, you may compress + serialize it/then deserialize it from within your code using zlib etc. (that is in Python). Basically, that means - your code will contain a compressed string representing the dictionary; which will then be de-compressed and used. You can take a look at this code submitted during CodeSprint5 here

Output Format

The output should contain lines. Each line should contain the rectified search query for the misspelled query in the corresponding line of the input file. The benchmark is the rectified query returned by Google.

The Nature of the Input Queries
The only named entities in the queries will be the names of countries (India, China, USA etc.). The queries will be made up of one or more words which, out of which some might have spelling errors. Typically most words with spelling mistakes will be no more than an edit distance of 1 or 2 from the correct spelling. There might also be cases where the space between two words wasn't typed in - you may need to handle segmentation issues for this.

Sample Input

4

gong to china

who ws the first president of india

winr of the match

fod in america

Sample Output

going to china

who was the first president of india

winner of the match

food in america

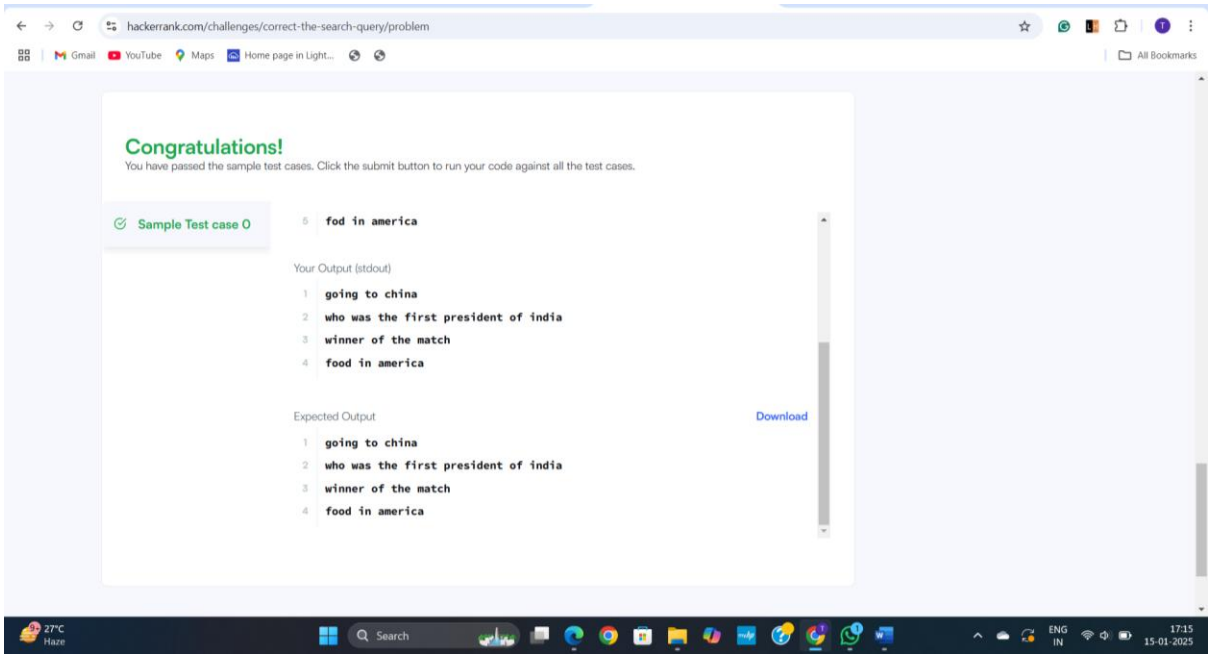
8A. Code:

```
import re

import json

from collections import Counter

import zlib


# Pre-built dictionary of words
```

```
def build_corpus():
    corpus = """
    going to china who was the first president of india winner of the match food in america
    india china usa america president first winner match food going
    """
    words = re.findall(r'\w+', corpus.lower())
    return Counter(words)

# Load compressed dictionary
def get_corpus():
    compressed_corpus = zlib.compress(json.dumps(build_corpus()).encode())
    return json.loads(zlib.decompress(compressed_corpus).decode())

# Calculate edit distance
def edit_distance(word1, word2):
    dp = [[0] * (len(word2) + 1) for _ in range(len(word1) + 1)]
    for i in range(len(word1) + 1):
        for j in range(len(word2) + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])
    return dp[-1][-1]

# Get candidate corrections
def correct(word, corpus):
    if word in corpus:
        return word
    candidates = [(w, edit_distance(word, w)) for w in corpus if edit_distance(word, w) <= 2]
```

```
    candidates.sort(key=lambda x: (x[1], -corpus[x[0]]))  # Sort by distance, then frequency
    return candidates[0][0] if candidates else word

# Correct a query
def correct_query(query, corpus):
    words = query.split()
    corrected = [correct(word, corpus) for word in words]
    return ' '.join(corrected)

# Main program
def main():
    # Input
    n = int(input())
    queries = [input().strip() for _ in range(n)]

    # Load dictionary
    corpus = get_corpus()

    # Correct queries
    corrected_queries = [correct_query(query, corpus) for query in queries]

    # Output
    for corrected in corrected_queries:
        print(corrected)

# Run program
if __name__ == "__main__":
    main()
```

Output:



Q9] A Text-Processing Warmup

provide accurate answers ouptut

This problem will help you warm up and practice basic text and string processing techniques. This will be a first step towards more complex Text and Natural Language Processing and Analysis tasks.

You will be given a fragment of text.

In this fragment, you need to identify the articles used (i.e., 'a', 'an', 'the').

And you also need to identify dates (which might be expressed in a variety of ways such as '15/11/2012','15/11/12', '15th March 1999','15th March 99' or '20th of March, 1999').

You can make the following assumptions 1) In the date, year and day will always be in numeric form. Which means, you don't have to worry about "fifteenth" or "twentieth" etc. Month, could be either numeric form (1-12) or with its name (January-December, Jan-Dec).

2) This is a bit open ended, and somewhat intentionally so. The aim is for you to try to write something which figures out as many common patterns as possible, in which dates are present in text.

3) Most of the test cases are Wikipedia articles. Having a look at the common formats in which dates occur in those, will help.

4) Dates could either be in the form: Month followed by Day followed by Year, or Day followed by Month followed by Year.

5) The day could be in the form of either (1,2,3,...31) or (1st, 2nd, 3rd...31st).

A fragment is a valid date if it contains day, month and year information (all three of them should be present). To extract date information, you will need to try detecting different kinds of representations of dates, some of which have been shown above. The more patterns you match and identify correctly, the greater your score will be.

Input Format

First line contains the number of test cases T. This is followed by T test fragments (each fragment will be in one line and each will have a blank line after it) . Each line contains a paragraph of text in which you need to identify the articles and dates. There will be a blank line after each paragraph.

So, totally there are 2T+1 lines in the input file. The last one is a blank line after the last text fragment.

Output Format

4T lines, four lines of output for each test case. First line -> number of occurrences of 'a'. Second line -> number of occurrences of 'an'. Third Line -> number of occurrences of 'the'. Fourth Line -> number of occurrences of date information.

Sample Input

5

Delhi, is a metropolitan and the capital region of India which includes the national capital city, New Delhi. It is the second most populous metropolis in India after Mumbai and the largest city in terms of area.

Mumbai, also known as Bombay, is the capital city of the Indian state of Maharashtra. It is the most populous city in India, and the fourth most populous city in the world, with a total metropolitan area population of approximately 20.5 million.

New York is a state in the Northeastern region of the United States. New York is the 27th-most extensive, the 3rd-most populous, and the 7th-most densely populated of the 50 United States.

The Indian Rebellion of 1857 began as a mutiny of sepoys of the East India Company's army on 10 May 1857, in the town of Meerut, and soon escalated into other mutinies and civilian rebellions largely in the upper Gangetic plain and central India, with the major hostilities confined to present-day Uttar Pradesh, Bihar, northern Madhya Pradesh, and the Delhi region.

The Boston Tea Party (referred to in its time simply as "the destruction of the tea" or by other informal names and so named until half a century later,[2]) was a political protest by the Sons of Liberty in Boston, a city in the British colony of Massachusetts, against the tax policy of the British government and the East India Company that controlled all the tea imported into the colonies. On December 16, 1773, after officials in Boston refused to return three shiploads of taxed tea to Britain, a group of colonists boarded the ships and destroyed the tea by throwing it into Boston Harbor. The incident remains an iconic event of American history, and other political protests often refer to it.

Sample Output

1

0

4

0

1

0

5

0

1

0

6

0

1

0

6

1

4

1

13

1

Line Wise Explanation of the Output

1 #Number of "a" in first text segment

0 #Number of "an" in first text segment

4 #Number of "the" in first text segment

0 #Number of dates in first text segment

1 #Number of "a" in second text segment

0 #Number of "an" in second text segment

5 #Number of "the" in second text segment

0 #Number of dates in second text segment

1 .....

0 .....

Delhi, is[a]metropolitan and[the]capital region of India which includes[the]national capital city, New Delhi. It is[the]second most populous metropolis in India after Mumbai and[the]largest city in terms of area.

Mumbai, also known as Bombay, is[the]capital city of[the]Indian state of Maharashtra. It is[the]most populous city in India, and[the]fourth most populous city in[the]world, with[a]total metropolitan area population of approximately 20.5 million.

New York is[a]state in[the]Northeastern region of[the]United States. New York is[the]27th-most extensive,[the]3rd-most populous, and[the]7th-most densely populated of[the]50 United States.

[The]Indian Rebellion of 1857 began as[a]mutiny of sepoys of[the]East India Company's army on[10 May 1857], in[the]town of Meerut, and soon escalated into other mutinies and civilian rebellions largely in[the]upper Gangetic plain and central India, with[the]major hostilities confined to present-day Uttar Pradesh, Bihar, northern Madhya Pradesh, and[the]Delhi region.

[The]Boston Tea Party (referred to in its time simply as [the]destruction of[the]tea" or by other informal names and so named until half[a]century later,[2]) was[a]political protest by[the]Sons of Liberty in Boston,[a]city in[the]British colony of Massachusetts, against[the]tax policy of[the]British government and[the]East India Company that controlled all[the]tea imported into[the]colonies. On[December 16, 1773], after officials in Boston refused to return three shiploads of taxed tea to Britain,[a]group of colonists boarded[the]ships and destroyed[the]tea by throwing it into Boston Harbor.[The]incident remains[an]iconic event of American history, and other political protests often refer to it.

SCORING

Your score will be proportional to the number of lines in your output which match with the expected and correct output.

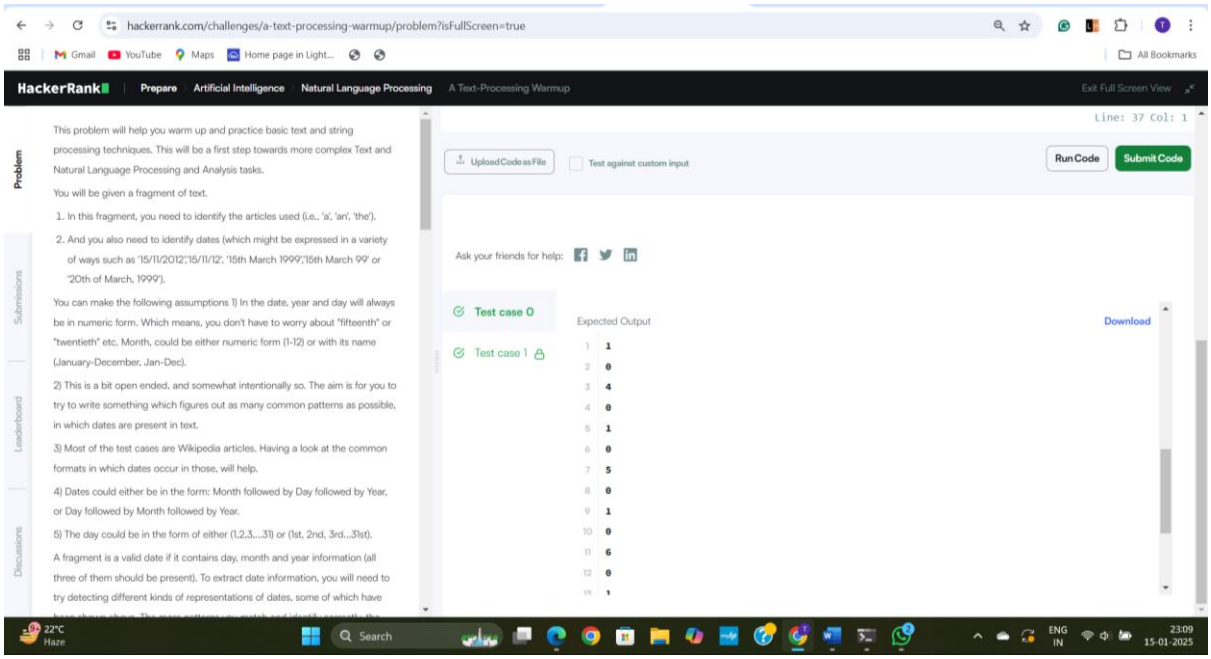
Score = MaxScore * (correctly computed values in your output) / (4 x Number of Text Fragments)

9.A Code:

```
import re


# Function to count articles and dates
def process_text(text):
    # Counting articles "a", "an", "the"
    a_count = len(re.findall(r'\ba\b', text, re.IGNORECASE))
    an_count = len(re.findall(r'\ban\b', text, re.IGNORECASE))
    the_count = len(re.findall(r'\bthe\b', text, re.IGNORECASE))

    # Regular expression for date matching
    date_pattern = r'\b(\d{1,2}(?:st|nd|rd|th)?(?:\s*(?:of\s*)?\s*(?:January|February|March|April|May|June|July|August|September|October|November|December|\d{1,2}))?(?:\s*\d{4}|\d{2}))\b'

    # Find all matches for dates
    dates = re.findall(date_pattern, text)
    date_count = len(dates)

    return a_count, an_count, the_count, date_count

# Input handling
T = int(input())  # Number of test cases

for _ in range(T):
    # Read each text fragment
    text = input().strip()

    # We expect a blank line after the text fragment
    #input()  # Read the blank line
```

```
# Process the text
a_count, an_count, the_count, date_count = process_text(text)

# Output the results for this test case
print(a_count)
print(an_count)
print(the_count)
print(date_count)
```

Output:



Q10] Who is it?

Anaphora Resolution is an interesting problem in Natural Language Processing, which involves, resolving what exactly a pronoun corresponds to. This page on the Cornell website has a few basic examples. As a recap, a pronoun In linguistics and grammar, a pronoun is a word or form (like 'he','her','she','her','its') that substitutes for a noun or noun phrase.

Consider the following fragments of text:

Fragment 1

William Jefferson "Bill" Clinton (born William Jefferson Blythe III, August 19, 1946) is an American politician who served from 1993 to 2001 as the 42nd President of the United States. Inaugurated at age 46, he was the third-youngest president. **He took office at the end of the Cold War, and was the first president from the baby boomer generation. Clinton has been described as a New Democrat. Many of **his** policies have been attributed to a centrist Third Way philosophy of governance. Before becoming president, he was the**

Governor of Arkansas for five two-year terms, serving from 1979 to 1981 and from 1983 to 1992. **He was also the state's Attorney General from 1977 to 1979.**

Consider the bolded instances of 'He' and 'His'. All of these, refer to William Jefferson "Bill" Clinton.

Fragment 2

Vladimir Putin had a telephone conversation with President of the United States Barack Obama on the American side's initiative. The two presidents discussed in detail various aspects of the extraordinary situation in Ukraine. In reply to Mr Obama's concern over the possibility of the use of Russian armed forces on the territory of Ukraine, Vladimir Putin drew **his attention to the provocative and criminal actions on the part of ultranationalists who are in fact being supported by the current authorities in Kiev. The Russian President spoke of a real threat to the lives and health of Russian citizens and the many compatriots who are currently on Ukrainian territory. Vladimir Putin stressed that in case of any further spread of violence to Eastern Ukraine and Crimea, Russia retains the right to protect **its** interests and the Russian-speaking population of those areas.**

The two bolded pronouns are 'his' and 'its', which correspond to Barack Obama and Russia respectively.

Input Format

The first line will contain an integer N. Including this first line, there are a total of N+2 lines in the input file. The first line will be followed by N lines of text, from the snippet of text to be analyzed. This text will contain the names of several characters, places or things. Certain pronouns like "her","he","she","his","its" will be highlighted by including a pair of '*' signs immediately before and after them like: **in, **he**, **his**. The second line will contain a list of names or nouns or noun-phrases (of people, places, objects) from the text, separated by semi-colons. Your task is to identify, which of these names, each of the highlighted pronouns corresond to, in the order in which they occur.**

Input Constraints
Total number of characters in the given chunk of text will not exceed 20000. Total number of highlighted pronouns in the text snippet will not exceed 20.
1 <= N <= 20
The number of highlighted pronouns may or may not equal the number Each of the names will have an exact string match in the paragraph of text.

Output Format
Output will contain P lines, where P is the number of highlighted pronouns on the text snippet. The i^{th} line should contain to the name of the entity corresponding to the i^{th} of the P pronouns (in the order in which they occur).

Sample Input

3

Alice was not a bit hurt, and **she jumped up on to her feet in a moment: she looked up, but it was all dark overhead; before **her** was another long passage, and the White Rabbit was still in sight, hurrying down it. There was not a moment to be lost: away went Alice like the wind, and was just in time to hear it say, as **it** turned a corner, 'Oh my ears and whiskers, how late it's getting!' She was close behind **it** when she turned the corner, but the Rabbit was no longer to be seen: she found herself in a long, low hall, which was lit up by a row of lamps hanging from the roof. There were doors all round the hall, but they were all locked; and when Alice had been all the way down one side and up the other, trying every door, she walked sadly down the middle, wondering how she was ever to get out again. Suddenly she came upon a little three-legged table, all made of solid glass; there was nothing on **it** except a tiny golden key, and Alice's first thought was that **it** might belong to one of the doors of the hall; but, alas! either the locks were too large, or the key was too small, but at any rate it would not open any of them.**

However, on the second time round, she came upon a low curtain she had not noticed before, and behind it was a little door about fifteen inches high: she tried the little golden key in the lock, and to her great delight it fitted! Alice opened the door and found that **it led into a small passage, not much larger than a rat-hole: she knelt down and looked along the passage into the loveliest garden you ever saw. How she longed to get out of that dark hall, and wander about among those beds of bright flowers and those cool fountains, but she could not even get her head through the doorway; 'and even if my head would go through,' thought poor Alice, 'it would be of very little use without my shoulders. Oh, how I wish I could shut up like a telescope! I think I could, if I only knew how to begin.'**

For, you see, so many out-of-the-way things had happened lately, that Alice had begun to think that very few things indeed were really impossible.

White Rabbit;Alice;three-legged table;door;tiny golden key

Sample Output

Alice

Alice

White Rabbit

White Rabbit

three-legged table

tiny golden key

door

Explanation

The first line of the input file contains the length of the text snippet.
The next three lines contain the text snippet itself. The last line contains the names, nouns or noun-phrases which need to be associated with the highlighted pronoun.

Re-writing the paragraph of text with the names/nouns associated with the highlighted pronouns

Alice was not a bit hurt, and she(Alice) jumped up on to her feet in a moment: she looked up, but it was all dark overhead; before her(Alice) was another long passage, and the White Rabbit was still in sight, hurrying down it. There was not a moment to be lost: away went Alice like the wind, and was just in time to hear it say, as it(White Rabbit) turned a corner, 'Oh my ears and whiskers, how late it's getting!' She was close behind it(White Rabbit) when she turned the corner, but the Rabbit was no longer to be seen: she found herself in a long, low hall, which was lit up by a row of lamps hanging from the roof. There were doors all round the hall, but they were all locked; and when Alice had been all the way down one side and up the other, trying every door, she walked sadly down the middle, wondering how she was ever to get out again. Suddenly she came upon a little three-legged table, all made of solid glass; there was nothing on it(three-legged table) except a tiny golden key, and Alice's first thought was that it(tiny golden key) might belong to one of the doors of the hall; but, alas! either the locks were too large, or the key was too small, but at any rate it would not open any of them. However, on the second time round, she came upon a low curtain she had not noticed before, and behind it was a little door about fifteen inches high: she tried the little golden key in the lock, and to her great delight it fitted! Alice opened the door and found that it(door) led into a small passage, not much larger than a rat-hole: she knelt down and looked along the passage into the loveliest garden you ever saw. How she longed to get out of that dark hall, and wander about among those beds of bright flowers and those cool fountains, but she could not even get her head through the doorway; 'and even if my head would go through,' thought poor Alice, 'it would be of very little use without my

shoulders. Oh, how I wish I could shut up like a telescope! I think I could, if I only knew how to begin.' For, you see, so many out-of-the-way things had happened lately, that Alice had begun to think that very few things indeed were really impossible.

Scoring
Score for a test case will be M * c/N. Where, M is the maximum score assigned for the test case, c is the number of correct answers, w is the number of incorrect answers, and N is the total number of tests in the file. M is proportional to the number of test pronouns in the paragraph.

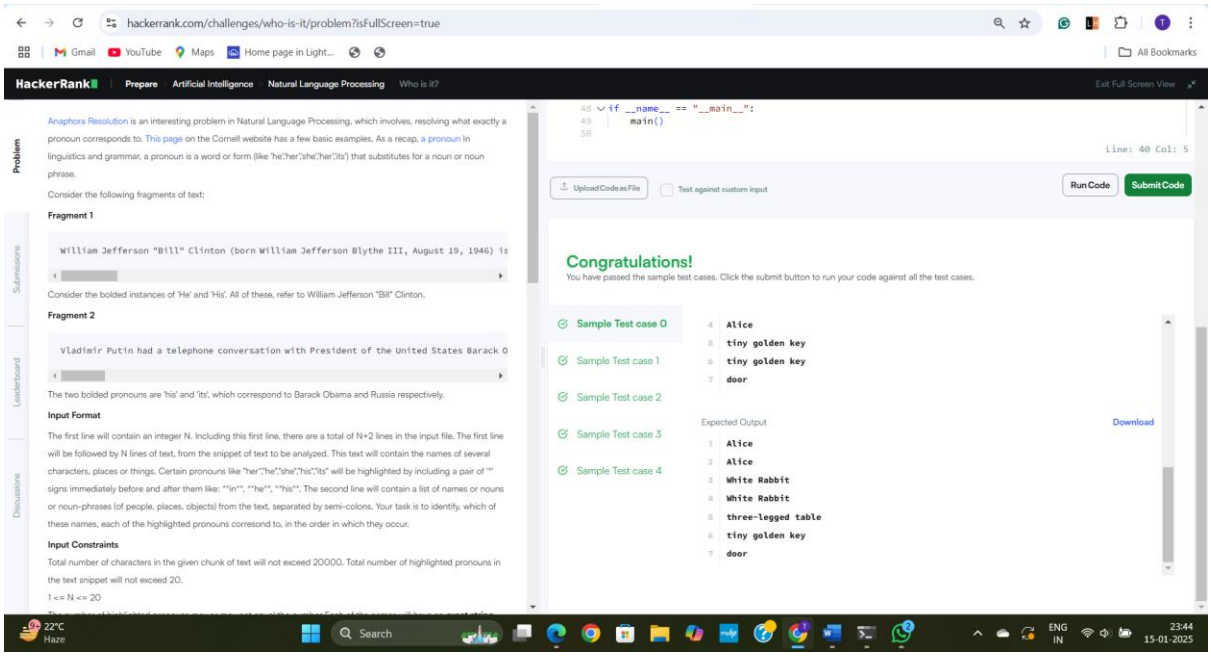
ML libraries will be provided for this challenge These libraries have been described in our environment. However, please note, that occasionally, certain specialized modules of these libraries, might not run on our infrastructure, do try them out in our online editor, before writing a solution which depends extensively on them.

10.A Code:

```
import re


def resolve_anaphora(text, entities):
    # Create a list of entities (names or noun-phrases)
    entity_list = entities.split(";")

    # Initialize a list to hold the resolved entities for each pronoun
    results = []

    # Split the text into sentences or clauses to process the pronouns
    sentences = re.split(r'(?<=[.!?])\s+', text)

    # Initialize the last entity that we encounter before the pronoun
    last_entity = None

    # Iterate through each sentence to resolve pronouns
    for sentence in sentences:
        # For each entity in the sentence, update the last entity if found
        for entity in entity_list:
            if entity.lower() in sentence.lower():
                last_entity = entity  # Update the last encountered entity

        # Look for pronouns in the sentence (e.g., **he**, **she**, **they**)
```

```
        pronouns = re.findall(r'\*\*([a-zA-Z]+)\*\*', sentence)

        # For each pronoun found, append the last encountered entity
        for pronoun in pronouns:
            results.append(last_entity)

    return results

def main():
    # Read input
    N = int(input())  # First line: number of text lines
    text_lines = [input() for _ in range(N)]  # Next N lines: the text
    entities = input()  # Last line: the list of entities

    # Combine the text lines into a single string
    text = " ".join(text_lines)

    # Resolve the anaphora
    result = resolve_anaphora(text, entities)

    # Print the results (output the entity corresponding to each pronoun in order)
    for res in result:
        print(res)

if __name__ == "__main__":
    main()
```

Output:


hackerrank.com/challenges/who-is-it/problem?isFullScreen=true
HackerRank
Prepare
Artificial Intelligence
Natural Language Processing
Who is it?
Problem
Submissions
Leaderboard
Discussions
Anaphora Resolution is an interesting problem in Natural Language Processing, which involves, resolving what exactly a pronoun corresponds to. This page on the Cornell website has a few basic examples. As a recap, a pronoun In linguistics and grammar, a pronoun is a word or form (like 'he';'her';she';'her';its') that substitutes for a noun or noun phrase.
Consider the following fragments of text:
Fragment 1
William Jefferson "Bill" Clinton (born William Jefferson Blythe III, August 19, 1946) is
Consider the bolded instances of 'He' and 'His'. All of these, refer to William Jefferson "Bill" Clinton.
Fragment 2
Vladimir Putin had a telephone conversation with President of the United States Barack O
The two bolded pronouns are 'his' and 'its', which correspond to Barack Obama and Russia respectively.
Input Format
The first line will contain an integer N. Including this first line, there are a total of N+2 lines in the input file. The first line will be followed by N lines of text, from the snippet of text to be analyzed. This text will contain the names of several characters, places or things. Certain pronouns like 'her','he','she','his','its' will be highlighted by including a pair of '*' signs immediately before and after them like: **in**, **he**, **his**. The second line will contain a list of names or nouns or noun-phrases (of people, places, objects) from the text, separated by semi-colons. Your task is to identify, which of these names, each of the highlighted pronouns corresond to, in the order in which they occur.
Input Constraints
Total number of characters in the given chunk of text will not exceed 20000. Total number of highlighted pronouns in the text snippet will not exceed 20.
1 <= N <= 20
if __name__ == "__main__":
main()
Line: 40 Col: 5
Upload Code as File
Test against custom input
Run Code
Submit Code
Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.
Sample Test case 0
Sample Test case 1
Sample Test case 2
Sample Test case 3
Sample Test case 4
Alice
tiny golden key
tiny golden key
door
Expected Output
Download
Alice
Alice
White Rabbit
White Rabbit
three-legged table
tiny golden key
door
22°C
Haze
Search
ENG
IN
23:44
15-01-2025