**NLP(HA)-HACKER RANK**

**1,8] Correct the Search Query**

import zlib, json

from difflib import get\_close\_matches

def build\_compressed\_dict(words):

    return zlib.compress(json.dumps(words).encode('utf-8'))

def decompress\_dict(compressed):

    return json.loads(zlib.decompress(compressed).decode('utf-8'))

def correct\_word(word, dictionary):

    return get\_close\_matches(word, dictionary, n=1, cutoff=0.8)[0] if get\_close\_matches(word, dictionary, n=1, cutoff=0.8) else word

def correct\_query(query, dictionary):

    return " ".join(correct\_word(word, dictionary) for word in query.split())

word\_list = ["going", "to", "china", "who", "was", "the", "first", "president", "of", "india", "winner", "match", "food", "in", "america"]

compressed\_dict = build\_compressed\_dict(word\_list)

dictionary = decompress\_dict(compressed\_dict)

n = int(input())

queries = [input() for \_ in range(n)]

for query in queries:

    print(correct\_query(query, dictionary))

**2] Deterministic Url and HashTag Segmentation**

def load\_words():

    with open("words.txt") as f:

        return set(line.strip() for line in f)

def segment\_string(s, words):

    s = s.lower()

    s = s.lstrip("#").lstrip("www.").rsplit('.', 1)[0]

    n = len(s)

    dp = [""] \* (n + 1)

    dp[0] = ""

    for i in range(1, n + 1):

        for j in range(i - 1, -1, -1):

            if s[j:i] in words or s[j:i].isdigit():

                if dp[j] != "" or j == 0:

                    dp[i] = (dp[j] + " " + s[j:i]).strip()

                    break

    return dp[-1] if dp[-1] else s

def main():

    words = load\_words()

    n = int(input())

    inputs = [input().strip() for \_ in range(n)]

    for s in inputs:

        print(segment\_string(s, words))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**3] Disambiguation: Mouse vs Mouse**

def classify\_mouse(context):

    context = context.lower()

    if any(word in context for word in ["computer", "input", "device", "click", "scroll", "pointer"]):

        return "computer-mouse"

    return "animal"

n = int(input())

for \_ in range(n):

    print(classify\_mouse(input().strip()))

**4] Language Detection**

import re

from collections import Counter

def detect\_language(text):

    # Step 1: Define stopwords for each language

    stopwords = {

        "English": {"the", "of", "and", "to", "in", "a", "is", "for", "on", "that", "this", "with", "by"},

        "French": {"le", "la", "et", "les", "des", "un", "une", "dans", "que", "en", "est", "pour", "avec"},

        "German": {"der", "die", "und", "das", "ist", "in", "zu", "mit", "den", "auf", "ein", "von", "es"},

        "Spanish": {"el", "la", "y", "en", "de", "los", "que", "un", "una", "por", "es", "con", "para"}

    }

    # Step 2: Preprocess the text

    text = text.lower()  # Convert to lowercase

    text = re.sub(r"[^\w\s]", "", text)  # Remove punctuation

    words = text.split()  # Tokenize into words

    # Step 3: Count matches with stopwords for each language

    language\_scores = {}

    for language, stopword\_set in stopwords.items():

        score = sum(1 for word in words if word in stopword\_set)

        language\_scores[language] = score

    # Step 4: Determine the language with the highest score

    detected\_language = max(language\_scores, key=language\_scores.get)

    return detected\_language

# Main Function

if \_\_name\_\_ == "\_\_main\_\_":

    # Read input

    text = ""

    while True:

        try:

            line = input().strip()

            if not line:

                break

            text += line + " "

        except EOFError:

            break

    # Detect the language and print the result

    print(detect\_language(text))

**5] The Missing Apostrophes**

**6] Segment the Twitter Hashtags**

def load\_words():

    """

    Load a dictionary of common words.

    Here we simulate a dictionary for demonstration purposes.

    """

    common\_words = {

        "we", "are", "the", "people", "mention", "your", "faves", "now", "playing",

        "the", "walking", "dead", "follow", "me"

    }

    return common\_words

def segment\_hashtag(hashtag, words):

    """

    Segments a single hashtag into its constituent words using a dynamic programming approach.

    Args:

        hashtag (str): The input hashtag (without '#').

        words (set): A set of valid dictionary words.

    Returns:

        str: The segmented hashtag as a space-separated string.

    """

    n = len(hashtag)

    dp = [None] \* (n + 1)  # dp[i] stores the segmented words for the first i characters

    dp[0] = ""  # Base case: empty string

    for i in range(1, n + 1):

        for j in range(i - 1, -1, -1):

            word = hashtag[j:i]

            if word in words:

                if dp[j] is not None:

                    dp[i] = (dp[j] + " " + word).strip()

                    break

    return dp[-1] if dp[-1] else hashtag  # Return the segmented result or the original hashtag if no split is found

def main():

    # Step 1: Load the dictionary

    words = load\_words()

    # Step 2: Read input

    n = int(input().strip())  # Number of hashtags

    hashtags = [input().strip().lower() for \_ in range(n)]

    # Step 3: Process each hashtag

    for hashtag in hashtags:

        print(segment\_hashtag(hashtag, words))

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**7] Expand the Acronyms**

import re

def extract\_expansions():

    # Read the number of lines and snippets

    n = int(input())

    snippets = [input().strip() for \_ in range(n)]

    queries = [input().strip() for \_ in range(n)]

    # Dictionary to store acronym-expansion pairs

    acronym\_to\_expansion = {}

    # Process each snippet to extract acronyms and their expansions

    for snippet in snippets:

        # Regex to match acronyms in parentheses and their potential expansions

        matches = re.findall(r'\(([^()]+)\)', snippet)

        for match in matches:

            # Check if the acronym is present in the match

            match\_words = match.split(":")  # Handles cases like "Abbreviation: NTU"

            if len(match\_words) > 1:

                acronym = match\_words[-1].strip()  # Get the actual acronym

            else:

                acronym = match.strip()

            # Search for the expansion before the acronym

            pattern = re.compile(rf'([\w\s,]+)\s\*\({re.escape(match)}\)', re.IGNORECASE)

            expansion\_match = pattern.search(snippet)

            if expansion\_match:

                expansion = expansion\_match.group(1).strip()

                acronym\_to\_expansion[acronym] = expansion

    # Output the expansions for the queried acronyms

    for query in queries:

        print(acronym\_to\_expansion.get(query, "Not Found"))

# Example Test Input/Output

extract\_expansions()

**9] A Text-Processing Warmup**

import re

def count\_articles\_and\_dates(text):

    # Count occurrences of articles

    count\_a = len(re.findall(r'\ba\b', text, re.IGNORECASE))

    count\_an = len(re.findall(r'\ban\b', text, re.IGNORECASE))

    count\_the = len(re.findall(r'\bthe\b', text, re.IGNORECASE))

    # Define regex patterns for dates

    date\_patterns = [

        r'\b(\d{1,2}(?:st|nd|rd|th)?[-\s](?:\d{1,2}|January|February|March|April|May|June|July|August|September|October|November|December)[-\s](\d{2,4}))\b',

        r'\b((?:\d{1,2}|January|February|March|April|May|June|July|August|September|October|November|December)[-\s](\d{1,2}(?:st|nd|rd|th)?)[-\s](\d{2,4}))\b',

        r'\b(\d{1,2}(?:st|nd|rd|th)?\s+of\s+(?:January|February|March|April|May|June|July|August|September|October|November|December)[,\s]+(\d{2,4}))\b',

        r'\b((?:January|February|March|April|May|June|July|August|September|October|November|December)\s+\d{1,2}(?:st|nd|rd|th)?,?\s+(\d{2,4}))\b',

        r'\b(\d{1,2}[-/](\d{1,2})[-/](\d{2,4}))\b'

    ]

    # Combine all patterns into one

    combined\_pattern = '|'.join(date\_patterns)

    # Count occurrences of dates

    count\_dates = len(re.findall(combined\_pattern, text, re.IGNORECASE))

    return count\_a, count\_an, count\_the, count\_dates

def main():

    import sys

    input\_data = sys.stdin.read().strip().split('\n')

    T = int(input\_data[0].strip())

    results = []

    for i in range(1, 2 \* T, 2):

        text\_fragment = input\_data[i].strip()

        count\_a, count\_an, count\_the, count\_dates = count\_articles\_and\_dates(text\_fragment)

        results.append(f"{count\_a}")

        results.append(f"{count\_an}")

        results.append(f"{count\_the}")

        results.append(f"{count\_dates}")

    print("\n".join(results))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**10]Who is it?**

import re

def associate\_pronouns():

    # Step 1: Read input

    n = int(input().strip())  # Number of lines in the paragraph

    paragraph = " ".join(input().strip() for \_ in range(n))  # Combine all lines into a single string

    entities = input().strip().split(";")  # Split entities by semicolon into a list

    # Step 2: Find all highlighted pronouns using regex

    pronoun\_pattern = r"\\*\\*(\w+)\\*\\*"  # Pattern to find pronouns wrapped in '\*\*'

    pronouns = list(re.finditer(pronoun\_pattern, paragraph))

    # Step 3: Store all positions of entities in the text

    entity\_positions = {

        entity: [m.start() for m in re.finditer(rf'\b{re.escape(entity)}\b', paragraph)]

        for entity in entities

    }

    results = []

    for pronoun\_match in pronouns:

        pronoun = pronoun\_match.group(1)  # Extract the pronoun (e.g., 'he', 'it')

        pronoun\_pos = pronoun\_match.start()  # Position of the current pronoun

        closest\_entity = None

        closest\_distance = float('inf')

        # Find the nearest entity

        for entity, positions in entity\_positions.items():

            for position in positions:

                distance = abs(pronoun\_pos - position)  # Calculate distance between pronoun and entity

                # Tie-breaking: If two entities have the same distance, prefer the earlier one

                if distance < closest\_distance or (distance == closest\_distance and position < entity\_positions[closest\_entity][0]):

                    closest\_distance = distance

                    closest\_entity = entity

        # Append the closest entity to the results

        results.append(closest\_entity)

    # Step 4: Print the results

    for result in results:

        print(result)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    associate\_pronouns()