

HACKERRANK SOLUTIONS NLP - 2211CS020403

1)Correct the Search Query

```
import zlib
import json
from difflib import get_close_matches

word_list=["going","to","china","hello","world","from","algorithm","python","programming","challenge","query","misspelled","correct"]

compressed_dict=zlib.compress(json.dumps(word_list).encode())

def load_dict():
    return set(json.loads(zlib.decompress(compressed_dict).decode()))

def correct_word(word,dictionary):
    if word in dictionary:
        return word
    matches=get_close_matches(word,dictionary,n=1,cutoff=0.8)
    return matches[0] if matches else word

def correcy_query(query,dictionary):
    words=query.split()
    corrected_words=[correct_word(word,dictionary) for word in words]
    return " ".join(corrected_words)

def process_queries(queries):
    dictionary=load_dict()
    return [correcy_query(query,dictionary) for query in queries]
```

```

if __name__=="__main__":
    N=int(input())
    queries=[input() for _ in range(N)]

    rectified_queries=process_queries(queries)
    for query in rectified_queries:
        print(query)

2
hwlllo word
Iam goin to chna
hello world
Iam going to china

```

2)Deterministic Url and HashTag Segmentation

```

import re

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def tokenize(input_string, dictionary):
    length = len(input_string)
    if length == 0:
        return []

    dp = [None] * (length + 1)
    dp[0] = []

```

```

for i in range(1, length + 1):
    for j in range(i):
        left_part = input_string[j:i]
        if (left_part in dictionary or is_number(left_part)) and dp[j] is not None:
            right_part_tokens = dp[j] + [left_part]
            if dp[i] is None or len(right_part_tokens) > len(dp[i]):
                dp[i] = right_part_tokens

    return dp[length] if dp[length] is not None else [input_string]

def main():
    num_test_cases = int(input())
    for _ in range(num_test_cases):
        input_string = input().strip().lower()
        if input_string.startswith("www."):
            input_string = input_string[4:].rsplit(".", 1)[0]
        elif input_string.startswith("#"):
            input_string = input_string[1:]

        tokens = tokenize(input_string, dictionary)
        print(f"Segmentation for Input: {' '.join(tokens)}")

if __name__ == "__main__":
    with open("words.txt", "r") as file:
        dictionary = set(word.strip().lower() for word in file.readlines())
    main()

```

3

#isittime
Segmentation for Input: isittime
www.whatismyname.com
Segmentation for Input: whatismyname
#letusgo
Segmentation for Input: letusgo

3)Disambiguation: Mouse vs Mouse

```
import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

training_sentences = [
    "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.",
    "Many mice have a pink tail.",
    "The mouse pointer on the screen helps in navigation.",
    "A rodent like a mouse has sharp teeth.",
    "The mouse was connected to the computer using a USB port.",
    "The house was infested with mice.",
    "Computer users often prefer a wireless mouse."
]

labels = [
    "animal",
    "animal",
    "computer-mouse",
    "animal",
    "computer-mouse",
    "animal",
    "computer-mouse",
    "animal",
    "computer-mouse"
]

vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(training_sentences)
```

```

classifier = MultinomialNB()
classifier.fit(X_train, labels)

def predict_mouse_type(sentence):
    vectorized_sentence = vectorizer.transform([sentence])
    prediction = classifier.predict(vectorized_sentence)[0]
    return prediction

num_test_cases = int(input())
for _ in range(num_test_cases):
    sentence = input()
    prediction = predict_mouse_type(sentence)
    print(prediction)

with open('mouse_classifier.pkl', 'wb') as f:
    pickle.dump((vectorizer, classifier), f)

1
The house was infested with mice.
animal

```

4)Language Detection

```

import pickle
import unicodedata
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

def normalize_to_ascii(text):
    return unicodedata.normalize("NFKD", text).encode("ascii", "ignore").decode("ascii")

training_texts = {
    "English": [

```

```

        "The quick brown fox jumps over the lazy dog.",
        "Rip Van Winkle is a story set in the years before the American Revolutionary War.",
    ],
    "French": [
        "Le renard brun rapide saute par-dessus le chien paresseux.",
        "La revolution francaise a marque une periode importante de l'histoire.",
    ],
    "German": [
        "Der schnelle braune Fuchs springt uber den faulen Hund.",
        "Die deutsche Wiedervereinigung war ein historisches Ereignis.",
    ],
    "Spanish": [
        "El rapido zorro marron salta sobre el perro perezoso.",
        "La Revolucion Espanola fue un momento clave en la historia. Si quieres que te asciendan te
        tienes que poner las pilas.",
    ],
}

labels = []
texts = []
for language, samples in training_texts.items():
    labels.extend([language] * len(samples))
    texts.extend([normalize_to_ascii(sample) for sample in samples])

vectorizer = TfidfVectorizer(ngram_range=(2, 4), analyzer="char")
X_train = vectorizer.fit_transform(texts)

classifier = MultinomialNB()
classifier.fit(X_train, labels)

with open("language_model.pkl", "wb") as model_file:
    pickle.dump((vectorizer, classifier), model_file)

def detect_language(snippet):

```

```

with open("language_model.pkl", "rb") as model_file:
    vectorizer, classifier = pickle.load(model_file)
    snippet = normalize_to_ascii(snippet)
    X_test = vectorizer.transform([snippet])
    prediction = classifier.predict(X_test)
    return prediction[0]

if __name__ == "__main__":
    snippet = """Le renard brun rapide saute par-dessus le chien paresseux."""
    detected_language = detect_language(snippet.strip())
    print(f"Detected Language: {detected_language}")

```

Detected Language: French

5)The Missing Apostrophes

```

import re

def restore_apostrophes(text):
    restored_text = []
    words = text.split()

    for word in words:
        lower_word = word.lower()
        if lower_word == "dont":
            restored_text.append("don't")
        elif lower_word == "wont":
            restored_text.append("won't")
        elif lower_word == "cant":
            restored_text.append("can't")
        elif lower_word == "isnt":
            restored_text.append("isn't")
        elif lower_word == "arent":

```

```
        restored_text.append("aren't")
    elif lower_word == "wasnt":
        restored_text.append("wasn't")
    elif lower_word == "werent":
        restored_text.append("weren't")
    elif lower_word == "hasnt":
        restored_text.append("hasn't")
    elif lower_word == "havent":
        restored_text.append("haven't")
    elif lower_word == "hadnt":
        restored_text.append("hadn't")
    elif lower_word == "didnt":
        restored_text.append("didn't")
    elif lower_word == "ive":
        restored_text.append("I've")
    elif lower_word == "were":
        restored_text.append("we're")
    elif lower_word == "i":
        restored_text.append("I")
    elif lower_word == "id":
        restored_text.append("I'd")
    elif lower_word == "youve":
        restored_text.append("you've")
    elif lower_word == "hes":
        restored_text.append("he's")
    elif lower_word == "shes":
        restored_text.append("she's")
    elif lower_word == "its":
        restored_text.append("it's")
    elif re.match(r'\w+s$', word) and lower_word not in ["its", "hers", "ours", "yours", "theirs"]:
        restored_text.append(re.sub(r"s$", "'s", word))
    else:
        restored_text.append(word)
```



```
return " ".join(restored_text)
```

```
input_text = """At a news conference Thursday at the Russian manned-space facility in Baikonur, Kazakhstan, Kornienko said "we will be missing nature, we will be missing landscapes, woods." He admitted that on his previous trip into space in 2010 "I even asked our psychological support folks to send me a calendar with photographs of nature, of rivers, of woods, of lakes." Kelly was asked if hed miss his twin brother Mark, who also was an astronaut. "Were used to this kind of thing," he said. "Ive gone longer without seeing him and it was great." The mission wont be the longest time that a human has spent in space - four Russians spent a year or more aboard the Soviet-built Mir space station in the 1990s. SCI Astronaut Twins Scott Kelly (left) was asked Thursday if hed miss his twin brother, Mark, who also was an astronaut. Were used to this kind of thing, he said. Ive gone longer without seeing him and it was great. (NASA/Associated Press) "The last time we had such a long duration flight was almost 20 years and of course all ... scientific techniques are more advanced than 20 years ago and right now we need to test the capability of a human being to perform such long-duration flights. So this is the main objective of our flight, to test ourselves," said Kornienko."""
```

```
output_text = restore_apostrophes(input_text)
print(output_text)
```

At a new's conference Thursday at the Russian manned-space facility in Baikonur, Kazakhstan, Kornienko said "we will be missing nature, we will be missing landscapes, woods." He admitted that on hi's previou's trip into space in 2010 "I even asked our psychological support folk's to send me a calendar with photograph's of nature, of rivers, of woods, of lakes." Kelly wa's asked if hed mis's hi's twin brother Mark, who also wa's an astronaut. "Were used to thi's kind of thing," he said. "Ive gone longer without seeing him and it wa's great." The mission won't be the longest time that a human ha's spent in space - four Russian's spent a year or more aboard the Soviet-built Mir space station in the 1990s. SCI Astronaut Twin's Scott Kelly (left) wa's asked Thursday if hed mis's hi's twin brother, Mark, who also wa's an astronaut. we're used to thi's kind of thing, he said. I've gone longer without seeing him and it wa's great. (NASA/Associated Press) "The last time we had such a long duration flight wa's almost 20 year's and of course all ... scientific technique's are more advanced than 20 year's ago and right now we need to test the capability of a human being to perform such long-duration flights. So thi's i's the main objective of our flight, to test ourselves," said Kornienko.

6)Segment the Twitter Hashtags

```
def segment_hashtag(hashtag, word_dict):
    n = len(hashtag)
    dp = [None] * (n + 1)
    dp[0] = []

    for i in range(1, n + 1):
        for j in range(max(0, i - 20), i):
            word = hashtag[j:i]
            if word in word_dict and dp[j] is not None:
                dp[i] = dp[j] + [word]
                break

    return " ".join(dp[n]) if dp[n] is not None else hashtag

def process_hashtags(num_hashtags, hashtags, word_dict):
    result = []
    for hashtag in hashtags:
        segmented = segment_hashtag(hashtag, word_dict)
        result.append(segmented)
    return result

word_dict = {
    "we", "are", "the", "people", "mention", "your", "faves",
    "now", "playing", "walking", "dead", "follow", "me"
}

num_hashtags = int(input())
hashtags = [input().strip() for _ in range(num_hashtags)]

segmented_hashtags = process_hashtags(num_hashtags, hashtags, word_dict)
for segmented in segmented_hashtags:
    print(segmented)
```

```
5
wearethepeople
mentionyourfaves
nowplaying
thewalkingdead
followme
we are the people
mention your faves
now playing
the walking dead
follow me
```

7)Expand the Acronyms

```
import re

def extract_acronyms_and_expansions(snippets):
    acronym_dict = {}
    for snippet in snippets:
        matches = re.findall(r'\((\b[A-Z]+\b)\)', snippet)
        for match in matches:
            preceding_text = snippet.split(f"({match})")[0].strip()
            expansion_candidates = re.split(r'[.,;:-]', preceding_text)
            if expansion_candidates:
                expansion = expansion_candidates[-1].strip()
                acronym_dict[match] = expansion

    words = snippet.split()
    for i, word in enumerate(words):
        if word.isupper() and len(word) > 1:
            if word not in acronym_dict:
                if i > 0:
```

```

        preceding_context = " ".join(words[max(0, i-5):i])
        acronym_dict[word] = preceding_context

    return acronym_dict

def process_tests(acronym_dict, tests):
    results = []
    for test in tests:
        expansion = acronym_dict.get(test.upper(), "Not Found")
        results.append(expansion)
    return results

def main():
    n = int(input().strip())
    snippets = [input().strip() for _ in range(n)]
    tests = [input().strip() for _ in range(n)]
    acronym_dict = extract_acronyms_and_expansions(snippets)
    results = process_tests(acronym_dict, tests)
    print("\n".join(results))

if __name__ == "__main__":
    main()

```

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

located in Singapore, Southeast Asia.

8)Correct the Search Query

```
import zlib
import json
from difflib import get_close_matches

word_list=["going","to","china","hello","world","from","algorithm","python","programming","challenge","query","misspelled","correct"]

compressed_dict=zlib.compress(json.dumps(word_list).encode())

def load_dict():
    return set(json.loads(zlib.decompress(compressed_dict).decode()))

def correct_word(word,dictionary):
    if word in dictionary:
        return word
    matches=get_close_matches(word,dictionary,n=1,cutoff=0.8)
    return matches[0] if matches else word

def correcy_query(query,dictionary):
    words=query.split()
    corrected_words=[correct_word(word,dictionary) for word in words]
    return " ".join(corrected_words)

def process_queries(queries):
    dictionary=load_dict()
    return [correcy_query(query,dictionary) for query in queries]

if __name__=="__main__":
```

```

N=int(input())
queries=[input() for _ in range(N)]

rectified_queries=process_queries(queries)
for query in rectified_queries:
    print(query)

```

```

1
hell iam gong too hyderabad
hello iam going to hyderabad

```

9)A Text-Processing Warmup

```

import re

def count_articles_and_dates(fragment):
    lower_fragment = fragment.lower()
    a_count = len(re.findall(r'\b[a]\b', lower_fragment))
    an_count = len(re.findall(r'\b[an]\b', lower_fragment))
    the_count = len(re.findall(r'\b[the]\b', lower_fragment))

    date_patterns = [
        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)?,\s+\d{2,4}\b',
        r'\b\d{1,2}/\d{1,2}/\d{2,4}\b',
        r'\b\d{4}-\d{2}-\d{2}\b'
    ]
    date_regex = '|'.join(date_patterns)
    dates = re.findall(date_regex, fragment, re.IGNORECASE)
    date_count = len(dates)

```

```

    return a_count, an_count, the_count, date_count

def main():
    t = int(input().strip())
    fragments = [input().strip() for _ in range(t)]

    results = []
    for fragment in fragments:
        a_count, an_count, the_count, date_count = count_articles_and_dates(fragment)
        results.append(f"{a_count}\n{an_count}\n{the_count}\n{date_count}")

    print("\n".join(results))

if __name__ == "__main__":
    main()

```

2
I visited the Eiffel Tower on 15th of August 2023.
She plans to meet him on 12/25/2024 for Christmas.
0
0
0
1
0
0
0
1

10)Who is it?

```

import re

def resolve_pronouns(text, entities):

```

```

pronoun_pattern = r'\\(\\w+)\\'
pronouns = [(match.group(1), match.start()) for match in re.finditer(pronoun_pattern, text)]
clean_text = re.sub(r'\\(\\w+)\\', r'\\1', text)

resolved = []

for pronoun, pos in pronouns:
    closest_entity = None
    closest_distance = float('inf')

    for entity in entities:
        entity_pos = clean_text.rfind(entity, 0, pos)
        if entity_pos != -1:
            distance = pos - (entity_pos + len(entity))
            if distance < closest_distance:
                closest_distance = distance
                closest_entity = entity

    resolved.append(closest_entity)

return resolved

def main():
    try:
        n = int(input("Enter the number of lines in the text snippet: ").strip())
    except ValueError:
        print("Error: The first line must contain a valid integer.")
        return

    text_snippet = ""
    for _ in range(n):
        text_snippet += input().strip() + " "

    entities_input = input("Enter the list of entities (separate by semicolons): ").strip()

```



```
entities = [e.strip() for e in entities_input.split(';')]

result = resolve_pronouns(text_snippet, entities)

for entity in result:
    print(entity)

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

Enter the number of lines in the text snippet: 2

Thor went to asgard

he is god of thunder

Enter the list of entities (separate by semicolons): Thor;he