Exercise 1:

Identify the list of candidate words using Trie structure for the misspelled word. Find the minimum edit distance and choose the correct word based on the context.

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False


def build_trie(words):
    root = TrieNode()
    for word in words:
        node = root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
    return root


def suggest_candidates(trie, misspelled_word):
    def dfs(node, path, index, max_changes, current_changes, result):
        if node.is_end_of_word and current_changes <= max_changes:
            result.append(''.join(path))


        if index < len(misspelled_word):
            char = misspelled_word[index]
            for child_char, child_node in node.children.items():
                # Allow character replacement
                if char != child_char:
                    dfs(child_node, path + [child_char], index + 1, max_changes, current_changes + 1, result)
                else:
```

```
            dfs(child_node, path + [char], index + 1, max_changes, current_changes, result)

    root = trie
    result = []
    dfs(root, [], 0, 1, 0, result)  # You can adjust the value of max_changes as needed
    return result

# Example usage
valid_words = ["summer", "weather", "leather", "feather", "black", "jacket", "nice"]
trie_root = build_trie(valid_words)

misspelled_word_1 = "ueather"

candidates_1 = suggest_candidates(trie_root, misspelled_word_1)


print(f"Candidates for '{misspelled_word_1}': {candidates_1}")
```

```
Candidates for 'ueather': ['weather', 'leather', 'feather']
```

```
import nltk

def get_ngrams(word, n):
    ngrams = []
    for i in range(len(word) - n + 1):
        ngrams.append(word[i:i + n])
    return ngrams
```

```
def calculate_edit_distance(word1, word2):
    return nltk.edit_distance(word1, word2)

def suggest_candidates_ngram(valid_words, misspelled_word, n):
    misspelled_ngrams = get_ngrams(misspelled_word, n)
    min_distance = float('inf')
    best_candidate = None

    for word in valid_words:
        word_ngrams = get_ngrams(word, n)
        distance = calculate_edit_distance(misspelled_ngrams, word_ngrams)

        if distance < min_distance:
            min_distance = distance
            best_candidate = word

    return best_candidate

# Example usage
valid_words = ["summer", "weather", "leather", "feather", "black", "jacket", "nice"]
misspelled_word_1 = "ueather"
misspelled_word_2 = "ueather"

best_candidate_1 = suggest_candidates_ngram(valid_words, misspelled_word_1, n=2)
best_candidate_2 = suggest_candidates_ngram(valid_words, misspelled_word_2, n=2)

print(f"Best candidate for '{misspelled_word_1}': {best_candidate_1}")
print(f"Best candidate for '{misspelled_word_2}': {best_candidate_2}")
```

```
Best candidate for 'ueather': weather
Best candidate for 'ueather': weather
```

```
import nltk

def get_ngrams(word, n):
    ngrams = []
    for i in range(len(word) - n + 1):
        ngrams.append(word[i:i + n])
    return ngrams

def calculate_edit_distance(word1, word2):
    return nltk.edit_distance(word1, word2)

def suggest_candidates_ngram(valid_words, misspelled_word, n):
    misspelled_ngrams = get_ngrams(misspelled_word, n)
    min_distance = float('inf')
    best_candidate = None

    for word in valid_words:
        word_ngrams = get_ngrams(word, n)
        distance = calculate_edit_distance(misspelled_ngrams, word_ngrams)

        if distance < min_distance:
            min_distance = distance
            best_candidate = word

    return best_candidate

def correct_sentence(sentence, valid_words, n):
```

```
    words = nltk.word_tokenize(sentence)
    corrected_words = []

    for word in words:
        if word.isalpha():  # Ignore punctuation
            suggested_word = suggest_candidates_ngram(valid_words, word, n)
            corrected_words.append(suggested_word if suggested_word else word)
        else:
            corrected_words.append(word)

    corrected_sentence = ' '.join(corrected_words)
    return corrected_sentence

# Example usage
valid_words = ["black","I","a","weather", "leather", "jacket", "nice", "during", "the", "summer", "we",
"have", "best", "so"]
input_sentences = [
    "During the summer we have the best ueather.",
    "I have a black ueather jacket, so nice."
]

n = 2  # You can adjust the value of n as needed

output_sentences = [correct_sentence(sentence, valid_words, n) for sentence in input_sentences]

print("Input Sentences:")
for sentence in input_sentences:
    print(sentence)

print("\nCorrected Sentences:")
for sentence in output_sentences:
```

```
    print(sentence)
```

```
Input Sentences:
During the summer we have the best ueather.
I have a black ueather jacket, so nice.

Corrected Sentences:
during the summer we have the best weather .
I have I black weather jacket , so nice .
```

```
from spellchecker import SpellChecker


class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False


def build_trie(words):
    root = TrieNode()
    for word in words:
        node = root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
    return root


def get_candidates_trie(trie, prefix):
    node = trie
    for char in prefix:
        if char in node.children:
            node = node.children[char]
```

```
        else:
            # If any character is not found in the Trie, return an empty list
            return []

    # Traverse the Trie to get all words with the given prefix
    candidates = []
    get_all_words(node, prefix, candidates)

    return candidates

def get_all_words(node, current_prefix, words):
    if node.is_end_of_word:
        words.append(current_prefix)

    for char, child_node in node.children.items():
        get_all_words(child_node, current_prefix + char, words)

def choose_correct_word(trie, spell_checker, context, word1, word2):
    candidates1 = get_candidates_trie(trie, word1.lower())
    candidates2 = get_candidates_trie(trie, word2.lower())

    spell_corrected1 = spell_checker.correction(word1)
    spell_corrected2 = spell_checker.correction(word2)

    all_candidates = candidates1 + candidates2 + [spell_corrected1, spell_corrected2]

    if not all_candidates:
        return word1  # Default to word1 if neither is found in the dictionary

    # Use context keywords for better decision making
    context_keywords = {"excepted", "accepted"}
```

```
    if any(keyword in context.lower() for keyword in context_keywords):
        # If context contains "accepted," prioritize "accepted"
        return "accepted" if "accepted" in all_candidates else "excepted"
    else:
        return min(all_candidates, key=lambda x: len(x))

# Example
dictionary = ["weather", "leather", "jacket", "nice", "during", "the", "summer", "we", "have", "best",
"so", "principle", "principal", "excepted", "accepted"]
trie = build_trie(dictionary)

spell_checker = SpellChecker()
spell_checker.word_frequency.load_words(dictionary)

sentence = "The company (excepted/accepted) all the terms."
words = sentence.split()

corrected_sentence = []

for i, word in enumerate(words):
    if "/" in word:
        # Handle words with multiple choices
        choices = word.split("/")
        context = " ".join(words[max(0, i - 2):i + 3])  # Extract context words for decision making
        chosen_word = choose_correct_word(trie, spell_checker, context, choices[0], choices[1])
        corrected_sentence.append(chosen_word)
    else:
        corrected_sentence.append(word)

result = ' '.join(corrected_sentence)
print("Corrected Sentence:", result)
```

```
Corrected Sentence: The company accepted all the terms.
```

```
from spellchecker import SpellChecker


class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False


def build_trie(words):
    root = TrieNode()
    for word in words:
        node = root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
    return root


def get_candidates_trie(trie, prefix):
    node = trie
    for char in prefix:
        if char in node.children:
            node = node.children[char]
        else:
            # If any character is not found in the Trie, return an empty list
            return []
```

```
    # Traverse the Trie to get all words with the given prefix
    candidates = []
    get_all_words(node, prefix, candidates)

    return candidates

def get_all_words(node, current_prefix, words):
    if node.is_end_of_word:
        words.append(current_prefix)

    for char, child_node in node.children.items():
        get_all_words(child_node, current_prefix + char, words)

def choose_correct_word(trie, spell_checker, context, word1, word2):
    candidates1 = get_candidates_trie(trie, word1.lower())
    candidates2 = get_candidates_trie(trie, word2.lower())

    spell_corrected1 = spell_checker.correction(word1)
    spell_corrected2 = spell_checker.correction(word2)

    all_candidates = candidates1 + candidates2 + [spell_corrected1, spell_corrected2]

    if not all_candidates:
        return word1  # Default to word1 if neither is found in the dictionary

    # Use context keywords for better decision making
    context_keywords = {"principle", "principal"}
    if any(keyword in context.lower() for keyword in context_keywords):
        # If context contains "accepted," prioritize "accepted"
        return "principal" if "principal" in all_candidates else "principle"
    else:
```

```
    return min(all_candidates, key=lambda x: len(x))


# Example
dictionary = ["weather", "leather", "jacket", "nice", "during", "the", "summer", "we", "have", "best",
"so", "principle", "principal"]
trie = build_trie(dictionary)


spell_checker = SpellChecker()
spell_checker.word_frequency.load_words(dictionary)


sentence = "Mr Patrick is our new (principle/principal)."
words = sentence.split()


corrected_sentence = []


for i, word in enumerate(words):
  if "/" in word:
    # Handle words with multiple choices
    choices = word.split("/")
    context = " ".join(words[max(0, i - 2):i + 3])  # Extract context words for decision making
    chosen_word = choose_correct_word(trie, spell_checker, context, choices[0], choices[1])
    corrected_sentence.append(chosen_word)
  else:
    corrected_sentence.append(word)


result = ' '.join(corrected_sentence)
print("Corrected Sentence:", result)
```

```
Corrected Sentence: Mr Patrick is our new principal
```

```
from spellchecker import SpellChecker
```

```
from nltk.util import ngrams

class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False

def build_trie(words):
    root = TrieNode()
    for word in words:
        node = root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
    return root

def get_candidates_trie(trie, prefix):
    node = trie
    for char in prefix:
        if char in node.children:
            node = node.children[char]
        else:
            # If any character is not found in the Trie, return an empty list
            return []

    # Traverse the Trie to get all words with the given prefix
    candidates = []
    get_all_words(node, prefix, candidates)
```

```
    return candidates

def get_all_words(node, current_prefix, words):
    if node.is_end_of_word:
        words.append(current_prefix)

    for char, child_node in node.children.items():
        get_all_words(child_node, current_prefix + char, words)

def choose_correct_word(trie, spell_checker, context, word1, word2):
    candidates1 = get_candidates_trie(trie, word1.lower())
    candidates2 = get_candidates_trie(trie, word2.lower())

    spell_corrected1 = spell_checker.correction(word1)
    spell_corrected2 = spell_checker.correction(word2)

    all_candidates = candidates1 + candidates2 + [spell_corrected1, spell_corrected2]

    if not all_candidates:
        return word1  # Default to word1 if neither is found in the dictionary

    # Use n-grams to capture context
    context_ngrams = set(ngrams(context.lower().split(), 2))
    candidate_scores = []

    for candidate in all_candidates:
        candidate_ngrams = set(ngrams(candidate.lower().split(), 2))
        intersection = len(context_ngrams.intersection(candidate_ngrams))
        candidate_scores.append((candidate, intersection))

    best_candidate = max(candidate_scores, key=lambda x: x[1])[0]
```

```
    return best_candidate


# Example
dictionary = ["lose", "loose", "later", "latter", "stationary", "stationery", "accepted", "excepted",
"council", "counsel",
            "too", "to", "bear", "bare", "fur", "far", "furthest", "farthest", "advice", "advise", "loose",
"lose",
            "to", "too", "quiet", "quite", "heap", "hip", "there", "their"]


trie = build_trie(dictionary)


spell_checker = SpellChecker()
spell_checker.word_frequency.load_words(dictionary)


sentences = [
    "Please don’t keep your dog on the (lose/loose).",
    "The (later/latter) is my best friend.",
    "I need some (stationary/stationery) products for my craftwork.",
    "The actor (excepted/accepted) the Oscar.",
    "I will call you (later/latter) in the evening.",
    "Covid (affects/effects) the lungs.",
    "The (council/counsel) of the ministers were sworn in yesterday.",
    "Robert (too/to) wants to accompany us to the park.",
    "Mia will (council/counsel) me about choosing fashion as my career.",
    "The (bear/bare) at the zoo was very playful.",
    "The sheep have a lot of (fur/far) that keeps them warm.",
    "The hot spring is at the (furthest/farthest) corner of the street.",
    "Can you (advice/advise) me on how to study for exams?",
    "The team will (loose/lose) the match if they don’t play well.",
    "Can you go (to/too) the market for me?",
    "The teachers asked the students to keep (quite/quiet).",
```

```
    "The (heap/hip) of garbage should be cleaned immediately.",
    "This is (there/their) house."
]

corrected_sentences = []

for sentence in sentences:
    words = sentence.split()
    corrected_sentence = []

    for i, word in enumerate(words):
        if "/" in word:
            # Handle words with multiple choices
            choices = word.split("/")
            context = " ".join(words[max(0, i - 2):i + 3])  # Extract context words for n-gram comparison
            chosen_word = choose_correct_word(trie, spell_checker, context, choices[0], choices[1])
            corrected_sentence.append(chosen_word)
        else:
            corrected_sentence.append(word)

    corrected_sentences.append(' '.join(corrected_sentence))

for i, corrected_sentence in enumerate(corrected_sentences, start=1):
    print(f"{i}. {corrected_sentence}")
```

1. Please don’t keep your dog on the close
2. The later is my best friend.
3. I need some stationary products for my craftwork.
4. The actor excepted the Oscar.
5. I will call you later in the evening.
6. Covid affects the lungs.
7. The council of the ministers were sworn in yesterday.
8. Robert too wants to accompany us to the park.
9. Mia will council me about choosing fashion as my career.
10. The bear at the zoo was very playful.
11. The sheep have a lot of fur that keeps them warm.
12. The hot spring is at the furthest corner of the street.
13. Can you advice me on how to study for exams?
14. The team will loose the match if they don’t play well.
15. Can you go to the market for me?
16. The teachers asked the students to keep quite
17. The cheap of garbage should be cleaned immediately.
18. This is there house.