Aufgabe 1: Hopsitexte

Team-ID: 00153

Team-Name: ByteBusters

Bearbeiter/-innen dieser Aufgabe:
Nina Hochecker

17. November 2024

Inhaltsverzeichnis

---

1 Zielsetzung

Das Ziel dieses Projekts ist die Entwicklung eines Programms, das bei der Erstellung eines Hopsitextes hilft. Der Benutzer gibt dabei einen Text (ein Wort oder einen Satz) ein, und das Programm überprüft, ob der Text den Anforderungen an einen gültigen Hopsitext entspricht. Ein Hopsitext ist ein Text, bei dem zwei Spieler durch Sprungsequenzen navigieren, wobei jede Sequenz von einem Startpunkt zu einem Zielpunkt führt. Das Programm stellt sicher, dass keine Kollisionen zwischen den Zielpunkten der Sprungsequenzen auftreten, also dass beide Spieler nicht dasselbe Ziel erreichen.

Falls der eingegebene Text eine Kollision aufweist, gibt das Programm eine entsprechende Rückmeldung und ermöglicht es dem Benutzer, den Text weiter anzupassen, bis er die Bedingungen für einen gültigen Hopsitext erfüllt.

---

2 Mathematische Modellierung

Die Problemstellung kann durch die mathematische Modellierung von Sprungsequenzen und deren Kollisionen beschrieben werden. Betrachten wir einen Text als eine Liste von Zeichen $T = [t_1, t_2, \ldots, t_n]$, wobei jedes Zeichen t_i einer bestimmten Position im Text entspricht. Für jedes Zeichen im Text wird ein „Sprungwert“ s_i berechnet, der von der Position des Zeichens im Alphabet abhängt. Dies lässt sich als Funktion $s : T \to \mathbb{N}$ modellieren, wobei $\mathbb{N}$ die natürlichen Zahlen sind.

Ein Sprungwert für ein Zeichen t_i wird durch die folgende Formel berechnet:

$$s_i = \text{Position}(t_i) + d$$

wobei d eine konstante Verschiebung ist, die für die Sonderzeichen (wie „ä“, „ö“, „ß“) definiert werden muss. Für gewöhnliche Zeichen entspricht die Position im Alphabet, während für Sonderzeichen eine spezielle Zuweisung vorgenommen wird.

Die Herausforderung besteht nun darin, sicherzustellen, dass die Sprungwerte für alle Zeichen t_i und t_j (mit $i \neq j$) nicht kollidieren, d. h., dass die Endpositionen der Sprungsequenzen unterschiedliche Werte haben. Dies kann als eine Injektivität der Funktion s in den betrachteten Positionen formuliert werden:

$$s_i \neq s_j \quad \text{für alle} \quad i \neq j$$

---

3 Lösungsansatz

Der Lösungsansatz basiert auf einem Algorithmus, der die Eingabe eines Textes überprüft und sicherstellt, dass dieser den Anforderungen eines gültigen Hopsitextes entspricht. Die Überprüfung erfolgt in mehreren Schritten:

1. **Eingabe des Textes**: Der Benutzer gibt einen Text (Wort oder Satz) ein. Dieser Text kann beliebig lang sein und wird vom Programm in einzelne Zeichen zerlegt.
2. **Berechnung der Sprungwerte**: Für jedes Zeichen im Text wird ein SSprungwert"berechnet. Dieser Sprungwert hängt von der Position des Zeichens im Alphabet ab, wobei Sonderzeichen wie „ä“, „ö“, „ü“ und „ß“ spezielle Werte zugewiesen bekommen.
3. **Überprüfung der Sprungsequenzen**: Der Algorithmus überprüft dann, ob für jede Startposition im Text die berechneten Sprungwerte zu unterschiedlichen Endpositionen führen. Wenn zwei Sprungsequenzen das gleiche Ziel erreichen, handelt es sich um eine Kollision. In diesem Fall gibt das Programm eine entsprechende Rückmeldung und fordert den Benutzer auf, den Text zu ändern.
4. **Anpassung des Textes**: Sollte eine Kollision gefunden werden, wird der Text so lange angepasst, bis er keine Kollisionen mehr aufweist. Der Benutzer kann dabei einzelne Wörter entfernen oder anpassen, bis der Text gültig ist.
5. **Ausgabe des validen Textes**: Sobald ein Text ohne Kollisionen validiert wurde, gibt das Programm den gültigen Hopsitext aus und bestätigt, dass der Text den Anforderungen entspricht.

Dieser Algorithmus stellt sicher, dass der Benutzer schrittweise einen gültigen Hopsitext erstellen kann. Dabei erfolgt die Überprüfung kontinuierlich, sodass der Benutzer sofort erkennt, wenn Änderungen erforderlich sind.

---

4 Umsetzung

Die Umsetzung des Hopsitext-Generators erfolgt in Python und beinhaltet mehrere wesentliche Komponenten, die nachfolgend beschrieben werden.

4.1 Dateneingabe

Das Programm ermöglicht es dem Benutzer, schrittweise Text einzugeben. Dieser Text wird kontinuierlich überprüft, um sicherzustellen, dass keine Kollisionen zwischen den beiden Springern (Positionen im Text) auftreten. Eine leere Zeile beendet die Eingabe und startet die finale Überprüfung des Textes.

4.2 Sprungwertberechnung

Für jedes Zeichen im Text wird der Sprungwert berechnet. Dies geschieht durch die Funktion `calculate_jump`, die jedem Zeichen eine Zahl zuweist, basierend auf dessen Position im Alphabet. Besondere Zeichen wie „ä“, „ö“, „ü“ und „ß“ erhalten spezielle Werte, um auch diese korrekt in die Berechnungen einzubeziehen.

Formell lässt sich der Algorithmus wie folgt zusammenfassen:

- Jedes Zeichen c wird überprüft, ob es ein normaler Buchstabe oder ein Sonderzeichen ist.
- Für normale Buchstaben wird der Sprungwert als die Position des Buchstabens im Alphabet berechnet, wobei $a = 1$, $b = 2$, ...
- Für Sonderzeichen wie „ä“, „ö“ und „ß“ werden vordefinierte Werte genutzt, um auch diese Zeichen korrekt zu behandeln.
- Zeichen, die keine Buchstaben oder Sonderzeichen sind (wie Leerzeichen), erhalten den Wert 0 und werden ignoriert.

Die Berechnung erfolgt durch den folgenden Code:

```
def calculate_jump(char):
    #  berpr üft, ob das Zeichen ein normaler Buchstabe (a-z) ist
    if char.lower() in 'abcdefghijklmnopqrstuvwxyz':
        return ord(char.lower()) - ord('a') + 1
    #  berpr üft, ob das Zeichen ein deutsches Sonderzeichen (ä, ö, ü, ß) ist
    elif char.lower() in 'äöüß':
        return {'ä': 27, 'ö': 28, 'ü': 29, 'ß': 30}[char.lower()]
    else:
        return 0  # Ungültige Zeichen
```

4.3 Bewegung der Springer

Zwei Springer bewegen sich durch den Text, basierend auf den berechneten Sprungwerten der einzelnen Zeichen. Der erste Springer beginnt an der ersten Position des Textes, der zweite Springer an der zweiten Position. Beide Springer bewegen sich jeweils um den Betrag des aktuellen Buchstabenwertes. Wenn ein Springer das Ende des Textes erreicht, wird die Überprüfung gestoppt.

4.4 Überprüfung auf Kollisionen

Das Programm überprüft kontinuierlich, ob beide Springer auf derselben Position landen, was als Kollision betrachtet wird. Wenn eine Kollision auftritt, gibt das Programm die Position der Kollision zurück, und der Benutzer muss das betroffene Wort entfernen oder ändern, um die Kollision zu beheben.

4.5 Fehlerbehebung und Korrektur

Wenn eine Kollision auftritt, zeigt das Programm den betroffenen Textabschnitt und fordert den Benutzer auf, das ungültige Wort zu ersetzen. Das entfernte Wort wird aus dem Text gelöscht, und der Benutzer kann einen neuen Text hinzufügen. Nachdem alle Korrekturen vorgenommen wurden, wird der Text erneut überprüft.

4.6 Endgültige Überprüfung

Nachdem der Benutzer den Text angepasst hat, wird der gesamte Text auf Kollisionen überprüft. Wenn keine Kollisionen mehr auftreten, gilt der Text als „gültiger Hopsitext“. Andernfalls wird der Benutzer weiter aufgefordert, den Text zu korrigieren.

4.7 Zusammenfassung der Funktionen

- `calculate_jump(char)`: Berechnet den Sprungwert für ein Zeichen basierend auf seiner Position im Alphabet oder speziellen Werten für Sonderzeichen.
- `check_hopsitext(text)`: Überprüft, ob es zu einer Kollision der Springer kommt.
- `remove_invalid_word(words, invalid_index)`: Entfernt das ungültige Wort aus der Liste, wenn eine Kollision auftritt.

Das Programm stellt sicher, dass der Text während der Eingabe ständig auf Kollisionen überprüft wird. Jede Kollision wird identifiziert, und der Benutzer wird aufgefordert, den Text zu korrigieren, bis keine Kollision mehr vorhanden ist.

---

5 Beispiel

Im folgenden Beispiel wird ein Text generiert, bei dem die Sprungsequenzen korrekt überprüft und auf Kollisionen getestet werden.

Beispieltext	**Ergebnisse**
Eingabe: Das	• Aktueller Text: „Das“ • Status: Gültiger Hopsitext, neues Wort? • Gültiger Hopsitext
Eingabe: ist	• Aktueller Text: „Das ist“ • Status: Gültiger Hopsitext, neues Wort? • Gültiger Hopsitext
Eingabe: toll	• Aktueller Text: „Das ist toll“ • Status: Gültiger Hopsitext, neues Wort? • Gültiger Hopsitext

Tabelle 1: Beispielhafte Ergebnisse der Textgenerierung

Beispieltext	**Ergebnisse**
Eingabe: ba	• Aktueller Text: „ba“ • Status: Kein gültiger Hopsitext • Das ungültige Wort an Position 2 wurde entfernt. • Fügen Sie ein neues Wort hinzu.

Tabelle 2: Ergebnisse einer ungültigen Eingabe

Beispieltext	**Ergebnisse**
Eingabe: $a\ b^{76}$	• Aktueller Text: „$a\ b^{76}$“ • Status: Gültiger Hopsitext • Gültiger Hopsitext

Tabelle 3: Test mit sehr langer Zeichenfolge

Beispieltext	**Ergebnisse**
Eingabe: Dies ist ein sehr langer Text, der viele Wörter enthält und geprüft werden muss, um sicherzustellen, dass der Algorithmus korrekt funktioniert, wenn er mit großen Textmengen konfrontiert wird.	• Aktueller Text: „Dies ist ein sehr langer Text, der viele Wörter enthält und geprüft werden muss, um sicherzustellen, dass der Algorithmus korrekt funktioniert, wenn er mit großen Textmengen konfrontiert wird.“ • Status: Gültiger Hopsitext • Gültiger Hopsitext

Tabelle 5: Test mit langem Text und vielen Wörtern

6 Quellcode

6.1 Sprungwertberechnung

```
def calculate_jump(char):
    if char.lower() in 'abcdefghijklmnopqrstuvwxyz':
        return ord(char.lower()) - ord('a') + 1
    elif char.lower() in 'äöüß':
        return {'ä': 27, 'ö': 28, 'ü': 29, 'ß': 30}[char.lower()]
    return 0
```

6.2 Textüberprüfung auf Kollisionen

```
def check_hopsitext(text):
    pos1, pos2 = 0, 1
    len_text = len(text)

    while pos1 < len_text and pos2 < len_text:
        while pos1 < len_text and calculate_jump(text[pos1]) == 0:
            pos1 += 1
        while pos2 < len_text and calculate_jump(text[pos2]) == 0:
            pos2 += 1

        if pos1 >= len_text or pos2 >= len_text:
            break

        pos1 += calculate_jump(text[pos1])
        pos2 += calculate_jump(text[pos2])

```

```
        if pos1 == pos2:
            return False, pos1

    return True, "Gültiger Hopsitext"
```

6.3 Entfernen eines ungültigen Worts

```
def remove_invalid_word(words, invalid_index):
    del words[invalid_index]
```

6.4 Hauptprogramm

```
def main():
    print("Willkommen zum Hopsitext-Generator!")
    print("Geben Sie Ihren Text ein. Drücken Sie Enter für neue Zeilen.")
    print("Geben Sie eine leere Zeile ein, um die Eingabe zu beenden.")

    text = []
    while True:
        input_text = input("Eingabe: ").strip()
        if input_text == "":
            break

        text.append(input_text)
        current_text = " ".join(text)

        is_valid, message = check_hopsitext(current_text)

        print("\nAktueller Text:")
        print(current_text)
        print("\nStatus:", "Gültiger Hopsitext" if is_valid else "Kein gültiger Hopsitext")
        print(message)
        print("-" * 40)

        if not is_valid:
            words = current_text.split()
            invalid_pos = message
            word_index = 0

            char_count = 0
            for i, word in enumerate(words):
                word_length = len(word)
                if char_count + word_length >= invalid_pos:
                    word_index = i
                    break
                char_count += word_length + 1

            remove_invalid_word(words, word_index)
            text = words
            print(f"Das ungültige Wort an Position {invalid_pos} wurde entfernt.")
            print("Fügen Sie ein neues Wort hinzu:")

    final_text = " ".join(text)
    print("\nFinaler Text:")
    print(final_text)

    is_valid, message = check_hopsitext(final_text)
    print("Gültiger Hopsitext" if is_valid else "Kein gültiger Hopsitext")
    print(message)

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