Aufgabe 2: Texthopsen

Team-ID: 00476

Team-Name: HochgradigTalentiertWenigKrips

Bearbeiter/-innen dieser Aufgabe:
Jurek Engelmann, Lennart Peters

November 19, 2024

Contents

1 Lösungsidee **1**

2 Umsetzung **1**

3 Beispielaufgaben **1**
3.1 Beispiel 1 (BWinf: hopsen1 Datei) 1
3.2 Beispiel 2 (BWinf: hopsen2 Datei) 2
3.3 Beispiel 3 (BWinf: hopsen3 Datei) 2
3.4 Beispiel 4 (BWinf: hopsen4 Datei) 2
3.5 Beispiel 5 (BWinf: hopsen5 Datei 3

4 Quellcode **3**
4.1 Erklärung des Quellcodes 5

1 Lösungsidee

Die Lösungsidee des Spiels besteht darin, zwei Spielerinnen (Amira und Bela) abwechselnd über einen Text springen zu lassen, wobei der Wert eines jeden Buchstabens die Sprungweite bestimmt. Das Ziel des Spiels ist es, das Ende des Textes zu erreichen, wobei der Spieler, der zuerst das Ziel erreicht, gewinnt. Die Lösung nutzt ein Dictionary zur Zuordnung von Buchstabenwerten und simuliert die Bewegung der Spielerinnen bis zu einem Ergebnis. Ein entscheidender Faktor für die Verbesserung der Laufzeitkomplexität unseres Algorithmus ist die Parallelverarbeitung der beiden Spielerinnen, Amira und Beal. Diese gleichzeitige Bearbeitung führt zu einer erheblichen Steigerung der Geschwindigkeit des gesamten Prozesses.

2 Umsetzung

Das Programm liest den Text aus einer Datei ein, bereinigt ihn von allen Nicht-Buchstaben-Zeichen und simuliert dann das Sprungverhalten der beiden Spielerinnen. Dabei wird für jeden Buchstaben im Text die entsprechende Sprungweite anhand eines Dictionaries ermittelt. Wenn eine der Spielerinnen das Ende des Textes erreicht, wird das Ergebnis ausgegeben. Die Verwendung einer while-Schleife mit einer Abbruchbedingung sorgt dafür, dass der Algorithmus sofort stoppt, sobald eine der Spielerinnen das Ende des Textes erreicht, was die Ausführung beschleunigt und unnötige Berechnungen vermeidet.

3 Beispielaufgaben

3.1 Beispiel 1 (BWinf: hopsen1 Datei)

Eingabe:

```
Eine Schildkröte wurde wegen ihrer Langsamkeit von einem Hasen verspottet. Trotzdem wagte
sie es, den Hasen zum Wettrennen herauszufordern. Der Hase ließ sich mehr aus Scherz als
aus Prahlerei darauf ein. Es kam der Tag, an dem der Wettlauf stattfinden sollte. Das Ziel
wurde festgelegt und beide betraten im gleichen Augenblick die Laufbahn.

Die Schildkröte kroch langsam und unermüdlich. Der Hase dagegen legte sich mit mächtigen
Sprüngen gleich ins Zeug, wollte er den Spott für die Schildkröte doch auf die Spitze
treiben. Als der Hase nur noch wenige Schritte vom Ziel entfernt war, setzte er sich
schnaufend ins Gras und schlief kurz darauf ein. Die großen, weiten Sprünge hatten ihn
nämlich müde gemacht.

Doch plötzlich wurde der Hase vom Jubel der Zuschauer geweckt, denn die Schildkröte hatte
gerade das Ziel erreicht und gewonnen.

Der Hase musste zugeben, dass das Vertrauen in seine Schnelligkeit ihn so leichtsinnig
gemacht hatte, dass sogar ein langsames Kriechtier ihn mit Ausdauer besiegen konnte.
```

Ausgabe:

```
Verarbeite Datei: Beispielaufgaben/hopsen1.txt
Ergebnis für 'Beispielaufgaben/hopsen1.txt': amira
```

3.2 Beispiel 2 (BWinf: hopsen2 Datei)

Eingabe:

```
Ein Federchen flog durch das Land;
Ein Nilpferd schlummerte im Sand.
Die Feder sprach: "Ich will es wecken!"
Sie liebte, andere zu necken.
Aufs Nilpferd setzte sich die Feder
Und streichelte sein dickes Leder.
Das Nilpferd sperrte auf den Rachen
Und musste ungeheuer lachen.
```

Ausgabe:

```
Verarbeite Datei: Beispielaufgaben/hopsen2.txt
Ergebnis für 'Beispielaufgaben/hopsen2.txt': tie
```

3.3 Beispiel 3 (BWinf: hopsen3 Datei)

Eingabe:

```
Koukonisi ist eine kleine, nicht bewohnte griechische Insel im Golf von Moudros der Insel
Limnos. Diese Insel liegt nördlich von Moudros und gehört zu dessen Gemeindebezirk.
Koukonisi ist über eine befestigte Straße von der etwa 400m entfernten Küste zu erreichen.
```

Ausgabe:

```
Verarbeite Datei: Beispielaufgaben/hopsen3.txt
Ergebnis für 'Beispielaufgaben/hopsen3.txt': tie
```

3.4 Beispiel 4 (BWinf: hopsen4 Datei)

Eingabe:

```
128 Zeichen umfasst das ASCII-System und stellt damit eine simple, aber effektive
Möglichkeit dar, Texte digital zu codieren. Jeder Buchstabe des lateinischen Alphabets
sowie grundlegende Satzzeichen und Zahlen sind darin enthalten. Diese Beschränkung auf 128
Zeichen machte ASCII besonders für frühe Computer attraktiv, da Speicherplatz sehr knapp
war. Auch wenn moderne Systeme komplexere Codierungen nutzen, bleibt ASCII in vielen
Bereichen relevant.
```

Ausgabe:

```
Verarbeite Datei: Beispielaufgaben/hopsen4.txt
Ergebnis für 'Beispielaufgaben/hopsen4.txt': bela
```

3.5 Beispiel 5 (BWinf: hopsen5 Datei

Das folgende Beispiel ist zu umfangreich, um es hier in voller Länge darzustellen. Aus diesem Grund wird empfohlen, das Beispiel direkt in der Konsole auszuführen, um die vollständige Ausgabe zu erhalten.

```
Bitte führen Sie das Beispiel in Ihrer lokalen Entwicklungsumgebung aus, um die
vollständige Ausgabe zu sehen.
```

Falls Sie das Beispiel in einem Terminal ausführen möchten, kopieren Sie den folgenden Code und fügen ihn in Ihre Konsole ein:

```
# Beispiel-Code
cd Texthopsen
python main.py -v
```

4 Quellcode

Der folgende Python-Quellcode implementiert das "Hopsen-Spiel", bei dem zwei Spieler, Amira und Bela, auf einem Text basierend auf den Werten der Buchstaben springen. Die Spieler beginnen an verschiedenen Positionen und bewegen sich durch den Text, wobei jeder Schritt von einem Buchstabenwert abhängt. Das Ziel ist es, das Ende des Texts zu erreichen oder festzustellen, ob beide Spieler gleichzeitig das Ende erreichen (Unentschieden). Die Eingabedatei enthält den Text, und der Quellcode verarbeitet diesen Text, um das Ergebnis des Spiels zu bestimmen. In der main.py werden die Argumente mithilfe von argparse entgegengenommen und an die Hauptfunktion weitergegeben, während die eigentliche Verarbeitung und Logik des Spiels in der solve.py-Datei erfolgt.

```python
import os
from solve import read_input, get_winner


# Hauptfunktion
def main(file_path, verbose):
    # alle Werte Buchstaben in einem Dictionary speichern
```

```
        buchstaben_werte = {chr(i + 96): i for i in range(1, 27)}
        buchstaben_werte["ä"] = 27
        buchstaben_werte["ö"] = 28
        buchstaben_werte["ü"] = 29
        buchstaben_werte["ß"] = 30

        if verbose:
            print(f"\nVerarbeite Datei: {file_path}")

        # text einlesen
        text = read_input(file_path)
        # alle Zeichen, die keine Buchstaben sind nicht abspeichern und alle Buchstaben zu kleinen
        text_clean = [buchstabe.lower() for buchstabe in text if buchstabe.isalpha()]

        result = get_winner(text_clean, buchstaben_werte, 0, 1)
        print(f"Ergebnis für '{file_path}': {result}")


# argparse für die Argumente verwenden
if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(description="Hopsen-Spiel zwischen Amira und Bela.")
    parser.add_argument(
    "file_path",
    nargs="?",
    type=str,
    help="Pfad zur Eingabedatei (optional). "
         "Wenn nicht angegeben, werden die Dateien hopsen1.txt bis hopsen5.txt durchlaufen.",
    )
    parser.add_argument(
        "-v", "--verbose",
        action="store_true",
        help="Ausführliche Ausgabe aktivieren"
    )
    args = parser.parse_args()

    if args.file_path:
        if os.path.exists(args.file_path):
            main(args.file_path, args.verbose)
        else:
            print(f"Datei {args.file_path} nicht gefunden.")
    else:
        # Dateien von hopsen1.txt bis hopsen5.txt durchgehen
        for i in range(1, 6):
            filename = f"Beispielaufgaben/hopsen{i}.txt"
            print(f"{i}. Beispielaufgabe")
            if os.path.exists(filename):
                main(filename, args.verbose)
            else:
                if args.verbose:
                    print(f"Datei {filename} nicht gefunden.")
```

Die richtige Lösung für das Spiel wird in der solve.py-Datei implementiert, während in der main.py die Argumente über argparse entgegengenommen und an die Hauptfunktion weitergegeben werden.

```
# Funktion zum Einlesen der Eingabedatei
def read_input(file_path):
    with open(file_path) as file:
        input_text = file.read()
    return input_text


# Funktion, die das Hopsen-Spiel implementiert
def get_winner(text, buchstaben_werte, amira, bela):
    pos_amira = amira
    pos_bela = bela
    len_text = len(text)

    # solange iterieren bis es ein Ergebnis gibt
    while True:
        # aktueller buchstabe von Amira und Bela
```

```
        a_buchstabe = text[pos_amira]
18      b_buchstabe = text[pos_bela]

20      # Positionen von Amira und Bela aktualisieren
        pos_amira += buchstaben_werte[a_buchstabe]
22      pos_bela += buchstaben_werte[b_buchstabe]

24      # überprüfen, ob es ein Ergebnis gibt
        if pos_amira >= len_text and pos_bela >= len_text:
26          return "Unentschieden"
        elif pos_amira >= len_text:
28          return "Amira"
        elif pos_bela >= len_text:
30          return "Bela"

32
```

4.1 Erklärung des Quellcodes

Der Quellcode besteht aus mehreren Teilen, die jeweils unterschiedliche Funktionen im "Hopsen-Spiel" übernehmen:

- `read_input(file_path)`: Diese Funktion liest die Eingabedatei und gibt ihren Inhalt als Text zurück.
- `hopsen(text, buchstaben_werte, amira, bela)`: Hier findet die eigentliche Logik des Spiels statt. Es wird überprüft, wie Amira und Bela sich basierend auf den Werten der Buchstaben bewegen. Die Spieler beginnen an den Positionen `amira` und `bela` und springen weiter, bis einer der Spieler das Ende des Textes erreicht. Falls beide das Ende gleichzeitig erreichen, gibt es ein Unentschieden.
- `game(file_path, verbose)`: Diese Funktion ist für das gesamte Spiel verantwortlich. Sie lädt den Text aus der Datei, bereinigt ihn (entfernt alle Nicht-Buchstabenzeichen), und ruft dann die `hopsen`-Funktion auf. Falls die Option `-v` für ausführliche Ausgabe aktiviert ist, wird der Name der Eingabedatei angezeigt.
- `argparse`: Mit dieser Bibliothek werden die Kommandozeilenargumente verarbeitet. Es wird ein optionaler Dateipfad `file_path` erwartet, andernfalls werden automatisch Dateien von `hopsen1.txt` bis `hopsen5.txt` durchlaufen. Die Option `-v` aktiviert die ausführliche Ausgabe.

Der Code ermöglicht es, das "Hopsen-Spiel" sowohl mit einer spezifischen Eingabedatei als auch mit einer Reihe von Beispiel-Dateien auszuführen. Wenn keine Datei angegeben wird, werden standardmäßig fünf Beispiel-Dateien verarbeitet. Der Parameter `-v` sorgt für eine detaillierte Ausgabe über die Verarbeitung der Datei und das Ergebnis des Spiels.