Junioraufgabe 2: Baywatch

Team-ID: 00133

Team-Name: CubeFlo

Bearbeiter dieser Aufgabe:
Florian Rädiker (15 Jahre)

6. Oktober 2018

Inhalt

Lösungsidee

Es ist bekannt, dass beide Listen stimmen, aber die erste ist lückenhaft. Da bei der zweiten Liste die „Drehung“, also der nördlichste Punkt, nicht bekannt ist, wird die zweite Liste immer schrittweise gedreht und mit der ersten verglichen. Die Drehung erfolgt so, dass das letzte Element weggenommen und stattdessen vorne wieder angefügt wird. Wenn alle bekannten Landschaftsmerkmale der ersten Liste mit denen einer der gedrehten (oder vielleicht sogar der nicht gedrehten?) Listen übereinstimmt, ist eine mögliche Drehung gefunden. Es könnte natürlich auch mehrere mögliche Drehungen geben.

Umsetzung

Das Programm ist in Python 3 geschrieben.
Um die zweite `list` zu drehen, wird die Klasse `deque` aus dem Modul `collections` benutzt. Natürlich könnte man das Element mit `.pop()` entfernen und mit `insert(0, …)` wieder anfügen. Das verbraucht jedoch unnötig Zeit, da Listen definitiv ***nicht*** für diesen Zweck optimiert sind. Aus diesem Grund gibt es in der Programmierung sogenannte Queues; hier wird das Prinzip des *Ringpuffers* angewandt, bei dem Elemente von einem Ende entfernt und an das andere wieder angefügt werden können. Mit der `deque`-Klasse geschieht dies mit der `rotate`-Methode, die die `deque` um einen übergebenen `int` nach rechts (positiv) oder links (negativ) dreht. Ohne Parameter (wird hier benutzt)

wird ein Element nach rechts gedreht.
Der Funktion `get_rotated_list` werden diese zwei Listen als Parameter übergeben. Die zweite Liste wird in eine `deque` umgewandelt und per `for`-Schleife sooft gedreht, wie die Liste lang ist, sodass also jede mögliche Drehung einmal drankommt. In der `for`-Schleife wird dann mithilfe der Funktion `is_similar` geprüft, ob die beiden Listen bis auf die fehlenden Listen übereinstimmen. Die Funktion gibt ein Generatorobjekt zurück, indem jede passende Drehung der vollständigen Liste mit `yield` „zurückgegeben“ wird.
Die Funktion `is_similar` bekommt ebenfalls zwei Listen, die auf Gleichheit mit Ausnahme der fehlenden Werte überprüft werden. Fehlende Werte (in der ersten Liste) müssen durch den Wert `-1` gekennzeichnet sein. Dieser wird in keiner der Beispieldaten verwendet und wird deshalb anstelle des Fragezeichens in die Listen beim Einlesen geschrieben. Zuerst überprüft die Funktion, ob die beiden Listen die gleiche Länge haben. Wenn nicht, ähneln sie sich nicht, also wird `False` zurückgegeben. Andernfalls wird mit `zip()` und einer `for`-Schleife durch beide Listen gleichzeitig iteriert. Dabei dürfen die beiden Werte nur voneinander abweichen, wenn der erste Wert `-1` beträgt, ansonsten wird `False` zurückgegeben. Hat die `for`-Schleife einmal durch die gesamten Listen iteriert, wird `True` zurückgegeben.
Im Hauptprogramm wird jede Datei einmal als zwei Listen eingelesen, die Möglichkeiten berechnet und mit der Funktion `translate_land_tongues` werden die Zahlenlisten in Wörter übersetzt.

Beispiele

Zuerst einmal eine Tabelle, die die Ergebnisse der Beispieldaten darstellt:

Name	Länge	Möglichkeiten
baywatch1.txt	13	1
baywatch2.txt	200	1
baywatch3.txt	200	200
baywatch4.txt	200	2
baywatch5.txt	200	1
baywatch6.txt	127	1

Dabei ist zu beachten, dass in den Dateien 3 bis 5 die gleichen vollständigen Landzungen-Listen in der ersten Zeile vorliegen.
Die 200 Möglichkeiten der 3. Datei rühren daher, dass es nur unbekannte (theoretisch richtig geordnete) Landzungen gibt, während die zwei Möglichkeiten der 4. Datei durch ähnliche Beschaffenheit an unterschiedlichen Stellen entstehen.
Das Programm gibt jeweils die Anzahl der Landzungen (Länge) und die Anzahl der Drehungen für die vollständige Landzungen-Liste aus, um mit der unvollständigen übereinzustimmen.
Hier dazu ein Beispiel für `baywatch1.txt` und `baywatch4.txt`:

```
########
baywatch1.txt
Länge: 13
Möglichkeiten: 1
Häuser Wald Wald Wiese Häuser Wüste Wald See Wald Wiese Sumpf Wüste See

########
baywatch4.txt
Länge: 200
```

```
Möglichkeiten: 2
Reisfeld Reisfeld Reisfeld Reisfeld Sumpf Reisfeld Vulkankrater Wüste Häuser […]
```

Quellcode

Bedeutung der Skripte

```
python3 junior2.py
```

Berechnet die Möglichkeiten der Beispieldaten und gibt sie aus (in der Reihenfolge der unvollständigen Liste).

```
python3 junior2-cmd.py <path>
```

Berechnet die Möglichkeiten für die Baywatch-Datei unter dem Pfad `<path>` und gibt sie aus (in der Reihenfolge der unvollständigen Liste). Das Arbeitsverzeichnis ist auf das Verzeichnis der Datei gesetzt und, weil die Beispieldaten im selben Ordner liegen, können diese über `baywatch1.txt`, `baywatch2.txt`, … direkt ausgewählt werden.

def get_rotated_list(partial, distorted)

Rotiert die Liste `distorted` wie unter „Umsetzung“ beschrieben und prüft mit `is_similar`, ob sich die Listen jetzt ähneln.

```python
def get_rotated_list(partial, distorted):
    # umwandeln
    distorted = deque(distorted)
    for _ in range(len(distorted)):
        distorted.rotate()  # Liste um 1 nach rechts rotieren
        if is_similar(partial, distorted):
            # Rotierte Variante als Liste zum Generator hinzufügen
            yield list(distorted)
```

def is_similar(partial, entire)

Prüft, ob sich die zwei Listen `partial` und `entire` ähneln, also alle Elemente bis auf die unbekannten in `partial` (die Elemente mit Wert `-1`) gleich denen in `entire` sind.

```python
def is_similar(partial, entire):
    if len(partial) != len(entire):
        # Länge ist nicht gleich -> Listen ähneln sich nicht
        return False
    # die Elemente gleichen Indexes vergleichen
    for p, e in zip(partial, entire):
        # Wenn das erste Element nicht -1 ist (also unbekannt und in der zweiten
        # Liste dann beliebig), müssen die Elemente gleich sein.
        # Ansonsten sind sich die Listen nicht ähnlich
        if not (p == -1 or p == e):
            return False
    return True
```

def translate_land_tongues(land_nums)

Wandelt eine Liste von Landzungen-Beschaffenheiten als Liste mit Nummern von 1 bis 9 in einen String mit den entsprechenden Namen, getrennt mit Leerzeichen, um. Dafür wird das `dict` `LAND_NUMS` verwendet, in dem jeder Nummer die Bezeichnung als String zugeordnet ist.

Die einzelnen, in Strings mithilfe des `dicts` umgewandelten Elemente, werden mit `.join()` mit Leerzeichen dazwischen zusammengefügt.

```
def translate_land_tongues(land_nums):
    return " ".join(LAND_NUMS[i] for i in land_nums)
```

Das Hauptprogramm

Das eigentliche Programm befindet sich ebenfalls in der Datei `junior2.py` und wird wegen der `if`-Bedingung nur ausgeführt, wenn die Datei nicht als Modul benutzt wird.
Zuerst wird das Arbeitsverzeichnis auf das Verzeichnis der Datei gesetzt, um alle Beispieldaten im selben Verzeichnis relativ zum Skript abrufen zu können.
Jede der 6 Baywatch-Dateien wird einmal eingelesen, indem die zwei Zeilen getrennt in Variablen geschrieben werden und jeweils mit `.split(" ")` an den Leerzeichen zu einer Liste aufgespalten werden. In der Liste mit unbekannten, durch Fragezeichen gekennzeichneten Landzungen-Beschaffenheiten werden die Fragezeichen durch `-1` ersetzt, und auch alle Zahlen in den einzelnen Strings werden nach `int` konvertiert.
Die Ergebnisse der Funktion `get_rotated_list` als Generator werden zu einem Tupel konvertiert, um auch die Länge usw. berechnen zu können. Die erste Möglichkeit wird, sofern überhaupt welche existieren und die Liste nicht leer ist (also als Boolean `False` wäre), ausgegeben.

```
if __name__ == "__main__":
    import os
    cwd = os.path.dirname(__file__)
    os.curdir = cwd

    for num in range(1, 7):
        filename = "baywatch{}.txt".format(num)
        with open(filename, "r") as f:
            distorted, partial = f.readlines()
        distorted = (int(i) for i in distorted.strip().split(" "))
        partial = [(int(i) if i != "?" else -1)
                                              for i in partial.strip().split(" ")]
        results = tuple(get_rotated_lists(partial, distorted))
        print("########\n"+filename)
        print("Länge:", len(partial))
        print("Möglichkeiten:", len(results))
        if results:
            print(translate_land_tongues(i), end="\n\n")
        else:
            print("Keine Möglichkeiten\n")
```