Aufgabe 2: Geburtstag

Teilnahme-Id: 00041

Bearbeiter/-in dieser Aufgabe:
Sebastian Brunnert

16. April 2020

Inhaltsverzeichnis

Lösungsidee

Um eine (vierstellige) Zahl zu einem Term mit der Benutzung einer einzigen Ziffer zu machen, wird zu Anfang die Zahl - beispielsweise 2020 - als Term betrachtet. Nun gilt es den Teil 2020 so zu ersetzen, dass dem Ziel nur eine einzige Ziffer – zur Erklärung die Ziffer 4 - zu verwenden gerecht wird. Nun werden weitere Terme aufgestellt, die alle 2020 entsprechen. Dementsprechend wird geprüft ob 2020 / 4 ohne Rest möglich ist zu rechnen. Da dies der Fall ist wird der Term ((2020 / 4) * 4), also 505 * 4, geispeichert. Das selbe Prozedere wird mit 44, 444 usw. auch durchgeführt. Kann nun eine Zahl mit n-1 Ziffern nicht mehr ohne Rest durch 2020 geteilt werden, so kann dieses Verfahren vorerst abgebrochen werden, denn eine Zahl mit n Ziffern wäre dann auch nicht teilbar. Weiter wird mit Multiplikation, Addition sowie Substraktion, der Potenz und der Fakultät gleich bzw. adaptiert fortgefahren. Es wird also geprüft, ob eine Rechnung sinnvoll wäre im Zusammenhang. Falls ja, wird diese Rechnung errechnet und duch Revision wird ein Term aufgestellt, welcher dem Wert 2020 entspricht. Nun sind lauter Terme aufgestellt werden. Aufgabe ist es nun also mit diesen Termen genau so fortzufahren wie oben beschrieben, um schlussendlich einen Term zu erhalten der dem Wert 2020 entspricht und ausschließlich aus der Ziffer 4 besteht. Als Beispiel muss nun also bei dem Term (505 * 4) 505 weiter aufgelöst werden.

Umsetzung

Die Umsetzung dieser Aufgabe geschieht in der Programmiersprache Python (3). Es wird keine externe Bibliothek verwendet. Das Rekursions-Limit wird auf eine extremst hohe Zahl gesetzt, da der aufgestellte Algorithmus mithilfe von Rekursion arbeitet. Des Weiteren wird eine leere Liste solvingSteps aufgestellt, welcher später stellvertretend für einen aufgestellten Term Elemente beigefügt werden. Dieses Element beinhaltet den gesammte Term als String, den zu ersetztenden Teil des Termes als String sowie als Integer. Außerdem wird eine Hilfsfunktion aufgestellt, welche prüft, ob ein aufgestellter Term den Vorraussetzungen entspricht. Dabei wird primtiv jede Ziffer in einer for-Schleife durchgegeangen, ausgenommen der zu benutztenden Ziffer. Wird nun die Ziffer in dem Term gefunden, ist der Term falsch.

Nachdem nun alles Notwenidge vordefiniert wurde, werden durch Konsoleneingaben die zu verarbeitende Zahl input_number und die zu verwendene Ziffer input_digit definiert. Es wird in der Liste solvingSteps

dementsprechend ein Element beigefügt. Dabei ist der volle Term und der Teil des Terms, welcher später ersetzt werden soll als input_number gespeichert. Ebenfalls mit diesen Daten wird die rekursive Funktion solve ausgeführt. Ein weiterer Parameter ist dabei die Ziffer input_digit.

Die Liste solvingSteps wird in dieser rekursiven Funktion weiter aufgefüllt mit verarbeitbaren Termen. Am Ende der Funktion, führt sich die Funktion selber erneut aus mit den Daten des vordersten Elementes aus solvingSteps. Deshalb wird zu Anfang auch das erste Element aus deer Liste entfernt, damit spätere Elemente anschließend ausgeführt werden können. Jetzt wird für jede Grundrechenart sowie der Fakultät und der Potenz ein neuer Term bzw. teilweise auch mehrere neue Terme vorab aufgestellt. Dabei wird durch diverse Prüfungen, geschaut, ob das Aufstellen eines Termes sinvoll ist. Dann wird der neue Term durch Revision der Rechnung aufgestellt und gespeichert. Danach wird geprüft, ob der aufgestellte Term, der gerade verarbeitet wird, den Anforderungen entspricht. Falls ja, wird dieser ausgegeben und das Programm endet, da die Lösung mit den wenigstens Operanden gefunden wurde. Falls nein, wird dieser rekursiv später weiter verarbeitet .

Beispiele

2020 aus 4 (nach weniger als einer Sekunde)	((((4444)*4)+4444)/44)*4
2020 aus 9 (nach weniger als einer Sekunde)	((((99999)-9)-9)+99999)/99
2019 aus 5 (nach weniger als einer Sekunde)	((((555555)-55)/55)-5)/5
2019 aus 3 (nach ca. einer Sekunde)	(((((33333)-33)-3)-3)+33333)/33
9999 aus 1 (nach weniger als einer Sekunde)	((11111)-1111)-1
9998 aus 5 (nach weniger als einer Sekunde)	((((55555)-5555)-5)-5)/5

* Zeitangaben: Getestet wurde das Programm aus einen System mit folgenden Spezifikationen:

CPU: AMD Ryzen 5 2600

RAM: 16 Gigabyte 2666 MHz

Quellcode

Der vollständige Quellcode, ist in der Datei geburtstag.py zu finden.

```
# Funktion, welche rekursiv ausgeführt werden wir zum aufstellen des Termes
# @arg1 (String): Aktuell aufgestellter Term
# @arg2 (String): Teil des Terms, welcher noch nicht umgewandelt wurde (in String-Form)
# @arg3 (int): Teil des Terms, welcher noch nicht umgewandelt wurde (als Zahl)
# @arg3 (int): Zu verwendende Ziffer
def solve(fullTerm,toReplaceStr,toReplaceInt,digit):
    # Dieses Element wird aus Liste entfernt, damit spätere Elemente nachher ausgeführt werden könnten
    solvingSteps.pop(0)

    # Nun wird jede Grundrechenart ausgeführt und es wird mit dieser ein neuer Term aufgestellt
    # Bei jeder Grundrechenart wird geprüft, bis zu welcher "Zahl" (also z.B. 5; 55 oder 610) gerechnet werden könnte

    # Multiplikation
    # Anzahl der Ziffern werden immer erhöht
    for i in range(1,7):
        # Zahl wird aufgestellt anhand Anzahl i an Ziffern
        digitNew = int(i * str(digit))

        # Es wird geprüft, ob die aufgestellte Zahl sinnvoll ist:
        if(toReplaceInt / digitNew < 1):
            # Ist das Ergebnis kleiner als 1, kann auch nicht ohne Rest geteilt werden (bei ganzen Zahlen)
            break
        if(toReplaceInt / digitNew % 1 != 0):
            # Kann eine Zahl mit n-1 Ziffern nicht mehr durch eine Zahl ohne Rest geteilt werden, so kann auch eine Zahl mit n Ziffern nicht mehr ohne Rest durch diese Zahl geteilt werden
            break

        toReplaceIntNew = toReplaceInt / digitNew
```

```
        fullTermNew = fullTerm.replace(toReplaceStr,"(" + str(toReplaceIntNew).replace(".0","") + ")*" + str(digitNew))

        toReplaceStrNew = str(toReplaceIntNew).replace(".0","")

        solvingSteps.append({"fullTerm": fullTermNew, "toReplaceStr": toReplaceStrNew, "toReplaceInt": toReplaceIntNew})

        # Prüfe, ob alle Vorraussetzungen erfüllt sind
        if(fits(fullTermNew,digit)):
            print(fullTermNew)
            exit()

    # Division
    for i in range(1,7):
        digitNew = int(i * str(digit))

        if(digitNew > toReplaceInt):
            break

        toReplaceIntNew = toReplaceInt * digitNew

        fullTermNew = fullTerm.replace(toReplaceStr,"(" + str(toReplaceIntNew).replace(".0","") + ")/" + str(digitNew))

        toReplaceStrNew = str(toReplaceIntNew).replace(".0","")

        solvingSteps.append({"fullTerm": fullTermNew, "toReplaceStr": toReplaceStrNew, "toReplaceInt": toReplaceIntNew})

        if(fits(fullTermNew,digit)):
            print(fullTermNew)
            exit()

    # Addition
```

```
    for i in range(1,7):
        digitNew = int(i * str(digit))

        if(digitNew > toReplaceInt):
            break

        toReplaceIntNew = toReplaceInt - digitNew

        fullTermNew = fullTerm.replace(toReplaceStr,"(" + str(toReplaceIntNew).replace(".0","") + ")+" +
str(digitNew))

        toReplaceStrNew = str(toReplaceIntNew).replace(".0","")

        solvingSteps.append({"fullTerm": fullTermNew, "toReplaceStr": toReplaceStrNew, "toReplaceInt":
toReplaceIntNew})

        if(fits(fullTermNew,digit)):
            print(fullTermNew)
            exit()

    # Subtraktion
    for i in range(1,7):
        digitNew = int(i * str(digit))

        if(digitNew > toReplaceInt):
            break

        toReplaceIntNew = toReplaceInt + digitNew

        fullTermNew = fullTerm.replace(toReplaceStr,"(" + str(toReplaceIntNew).replace(".0","") + ")-" +
str(digitNew))

        toReplaceStrNew = str(toReplaceIntNew).replace(".0","")
```

```
        solvingSteps.append({"fullTerm": fullTermNew, "toReplaceStr": toReplaceStrNew, "toReplaceInt": toReplaceIntNew})


        if(fits(fullTermNew,digit)):
            print(fullTermNew)
            exit()


    # Prüfung ob noch Element vorhanden
    if(len(solvingSteps) == 0):
        print("Nicht lösbar!")
        exit()


    # Funktion führt sich erneut rekursiv aus
    solve(solvingSteps[0]["fullTerm"],solvingSteps[0]["toReplaceStr"],solvingSteps[0]["toReplaceInt"],digit)
```