

Programmierung für Naturwissenschaften 1  
Wintersemester 2020/2021  
Übungen zur Vorlesung: Ausgabe am 16.12.2020

**Aufgabe 6.1** (3 Punkte) Eine Primzahl ist eine ganze Zahl  $p \geq 2$ , die sich nicht ganzzahlig durch eine kleinere Zahl  $q \geq 2$  teilen lässt. In dieser Aufgabe soll der Algorithmus „Sieb des Eratosthenes“ zur Generierung einer Liste von Primzahlen  $\leq n$  implementiert werden.  $n$  ist dabei ein vom Benutzer definierter Wert. Der Algorithmus funktioniert wie folgt:

Man notiert zunächst alle Zahlen  $i$ ,  $2 \leq i \leq n$ . Z.B. ergibt sich für  $n = 20$  die folgende Zahlenfolge

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Im nächsten Schritt werden alle echten Vielfachen von 2 markiert (hier durch das Symbol  $\times$  dargestellt).

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20  
           $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$

Nun markiert man zusätzlich alle echten Vielfachen der kleinsten unmarkierten Zahl (in diesem Fall 3):

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20  
           $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$        $\times$

In jedem Schritt werden also jeweils die echten Vielfachen der kleinsten noch nicht markierten Zahl zusätzlich markiert. Das wird solange wiederholt, bis die kleinste unmarkierte Zahl größer als  $\sqrt{n}$  ist. Die noch nicht markierten Zahlen sind dann die gesuchten Primzahlen.

In der Datei `eratosthenes25_animation.pdf` finden Sie eine Visualisierung des Algorithmus für  $n = 25$ , in der die 25 Zahlen in einem Quadrat angeordnet werden. Das Programm zur Darstellung der PDF-Datei muss allerdings mit „Overlays“ umgehen können. Das gilt z.B. für den Adobe Acrobat Reader™. Falls kein geeigneter PDF-Viewer zur Verfügung steht, schauen Sie sich die statische Darstellung in `eratosthenes25_steps.pdf` an.

Die Markierung lässt sich am besten mit einer Liste von boolschen Werten (d.h. jeder Wert der Liste ist entweder `True` oder `False`) repräsentieren. Sei `marked` diese Liste. Der  $i$ -te Wert in der Liste ist genau dann `True`, wenn die Zahl  $i$  markiert ist. Am Anfang müssen alle Werte ab Index 2 auf `False` gesetzt werden, da noch keine Zahl markiert ist. Die Werte an Index 0 und 1 in der Liste können Sie auf `None` setzen, da sie nicht benötigt werden. Die einzelnen Phasen des Algorithmus bestehen darin, zu testen, ob eine Zahl  $i$  markiert ist (dann gilt `marked[i]`) bzw. nicht markierte Zahlen zu markieren, d.h. `marked[i] = True` zu setzen, entsprechend der obigen Beschreibung des Algorithmus.

Vor der Implementierung benennen Sie die Datei `eratosthenes_template.py` aus den Materialien um in `eratosthenes.py`. Fügen Sie Ihren Python-Code in diese Datei an der markierten Stelle ein.

Es ist bereits Programmcode zum Einlesen einer vom Benutzer übergebenen ganzen Zahl  $n$  vorhanden. Es wird die Anzahl der Primzahlen  $\leq n$  sowie die Liste der 10 größten Primzahlen  $\leq n$  ausgegeben. Im Material zu dieser Übungsaufgabe finden Sie eine Datei mit der erwarteten Ausgabe für  $n = 1\,000$  sowie ein Makefile. Durch Aufruf von `make test` verifizieren Sie die Korrektheit Ihrer Implementierung.

Punkteverteilung:

- 1 Punkt für das korrekte Setzen der Markierungen (while Schleife und erste for-Schleife)
- 1 Punkt für korrekte Bestimmung des nächsten unmarkierten Zeichens
- 1 Punkt für die korrekte Berechnung der `primes`-Liste (inklusive Test)

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Flow of control*, 26-33

**Aufgabe 6.2** (1 Punkt) Das Programm `fixerrors.py` enthält viele Fehler. Ihre Aufgabe ist es, diese zu finden und zu korrigieren. Sie können natürlich den Python-Interpreter verwenden, um Hinweise auf die Fehlerstellen zu erhalten.

Nach der Korrektur aller Fehler müssen Sie testen, ob das Programm die richtige Ausgabe liefert. Dazu rufen Sie `make test` auf.

**Aufgabe 6.3** (3 Punkte) In Textverarbeitungssystemen und auf Webseiten wird Text in Paragraphen häufig dynamisch umgebrochen. Wenn man mit Paste/Copy aus solchen Kontexten Text kopiert und in den Editor einfügt, entstehen meist sehr lange Zeilen, die jeweils den Inhalt eines Paragraphen darstellen. Ein Beispiel finden Sie in der Datei `python_history.txt` in den Materialien zu dieser Aufgabe.

Entwickeln Sie ein Python-Programm `fold_text.py`, das einen Text zeilenweise einliest und die Zeilen durch Einfügen des newline-Zeichens `\n` aufbricht und ausgibt. Dabei müssen jeweils möglichst viele Worte (d.h. Strings ohne Leerzeichen) in einer Zeile stehen. Die Anzahl der Zeichen pro Zeile (ohne das newline-Symbol) soll nicht größer sein als eine gegebene Konstante. Das Umbrechen darf nur an Stellen erfolgen, an denen ein Leerzeichen steht. Die Zeilen dürfen nicht mit einem Leerzeichen enden. Beim Aufruf des Programms ist das erste Argument die maximale Anzahl der Zeichen pro Zeile in der Ausgabe und das zweite Argument ist der Dateiname. Beispiel: Durch den Aufruf

```
./fold_text.py 70 python_history.txt
```

soll der Inhalt der Datei `python_history_fold70.txt` im Terminal ausgegeben werden. In dieser Datei haben alle Zeilen eine maximale Länge von 70 Zeichen.

Damit Sie sich auf das Wesentliche konzentrieren können, finden Sie in den Materialien eine Datei `fold_text_template.py`. Benennen Sie die Datei in `fold_text.py` um. Der vorhandene Programmcode dient zur Verifikation der Werte in `sys.argv` und zum Öffnen einer Datei. Nutzen Sie die Variable `stream` zum zeilenweisen Einlesen des Inhalts der geöffneten Datei.

Hinweis: Sammeln Sie die Worte, die zu einer Zeile gehören, zunächst in einer Liste und geben Sie alle Worte dieser Liste als String aus. Diesen String erzeugen Sie mit der Methode `join` aus der Liste der Worte.

In den Materialien finden Sie ein Makefile. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms.

**Aufgabe 6.4** (3 Punkte) Wir betrachten ein Alphabet der Größe  $r$  mit den Zeichen  $a_1, a_2, \dots, a_r$ . Für alle  $i$ ,  $0 \leq i \leq r$  ist der  $i$ -te Skyline-String  $s_i$  wie folgt definiert:

$$s_i = \begin{cases} \varepsilon & \text{falls } i = 0 \\ s_{i-1}a_i s_{i-1} & \text{falls } 1 \leq i \leq r \end{cases}$$

Beispiel: Für das Alphabet  $\{a, b, c, d\}$  ergeben sich die folgenden Skyline-Strings  $s_1 = a$ ,  $s_2 = aba$ ,  $s_3 = abacaba$  und  $s_4 = abacabadabacaba$ .

Hintergrund: Für bestimmte Algorithmen zur Sortierung von Strings bilden Skyline-Strings die Eingabe, die zur maximalen Laufzeit führt. Dieser Aspekt spielt in dieser Aufgabe jedoch keine Rolle.

Hier sind die Teilaufgaben, die Sie lösen sollen:

- Geben Sie an, welche Länge  $s_i$  hat. Sie sollen hier keine konkreten Zahlen angeben, sondern einen mathematischen Ausdruck, durch den, abhängig von  $i$ , die Länge von  $s_i$  bestimmt werden kann. Wenn Sie noch keine Idee haben, wie der Ausdruck gebildet werden kann, können Sie erst den zweiten Teil der Aufgabe lösen und sich die Längen der  $s_i$  ausgeben lassen.
- Wir betrachten das Alphabet  $\{a, b, c, \dots, z\}$  von 26 Kleinbuchstaben, d.h.  $r = 26$ . Schreiben Sie ein Python-Skript `skyline.py`, das für dieses Alphabet nacheinander die Skyline-Strings der Längen  $i$ ,  $1 \leq i \leq 9$  zeilenweise ausgibt. D.h. die  $i$ -te Zeile enthält  $s_i$ . Das  $i$ -te Zeichen des obigen Alphabets können Sie mit Hilfe der Methoden `ord` und `chr` berechnen. Sie können zur Lösung wahlweise eine rekursive Funktion implementieren, oder einen iterativen Ansatz wählen.

Im Material zu dieser Übung finden Sie die erwartete Ausgabe Ihres Programms sowie ein Makefile. Durch `make test` können Sie verifizieren, ob Ihr Programm korrekt funktioniert.

Punkteverteilung:

- 1 Punkt für die korrekte Aussage zur Länge der Skyline-Sequenzen, abhängig von  $i$
- 2 Punkte für die korrekte Implementierung. Diese kann rekursiv oder iterativ sein.

**Bitte die Lösungen zu diesen Aufgaben bis zum 04.01.2021 um 18:00 Uhr an [pfn1@zbh.uni-hamburg.de](mailto:pfn1@zbh.uni-hamburg.de) schicken.**