ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert
Fabian Berstecher

1. Juni 2017
Blatt 6

Systemnahe Software II (SS 2017)

Abgabe bis zum 8. Juni 2017, 16:00 Uhr

Lernziele:

- Pipes als Kommunikationsmedium im Rahmen des Fork-And-Join-Patterns.
- Auswerten einer Programmausgabe mit Hilfe einer Pipe.

Aufgabe 6: Freunde müsst ihr sein

Die Teilersumme einer natürlichen Zahl n ist definiert als die Summe all ihrer echten Teiler $\sigma^*(n)$. Zu den echten Teilern wird die 1 gezählt, jedoch nicht die Zahl selbst. Es gilt $\sigma^*(1) = 0$. Weitere Beispiele:

n	echte Teiler	$\sigma^*(n)$
6	1, 2, 3	6
7	1	1
12	1, 2, 3, 4, 6	16
28	1, 2, 4, 7, 14	28
57	1, 3, 19	23
220	1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110	284
284	1, 2, 4, 71, 142	220

Eine natürliche Zahl n wird als perfekt bezeichnet, wenn $n = \sigma^*(n)$ gilt. Das ist beispielsweise für die 6 und die 28 der Fall. Zwei natürliche Zahlen n, m sind befreundet, wenn $m = \sigma^*(n)$ und $n = \sigma^*(m)$ gelten. So sind 220 und 284 miteinander befreundet.

Die Funktion σ^* kann auch benutzt werden, um eine Folge zu bestimmen, die mit einer beliebigen natürlichen Zahl n beginnt: $(n, \sigma^*(n), \sigma^*(\sigma^*(n)), \dots)$. Im Falle perfekter Zahlen ist die Folge periodisch mit einer Periode von 1. Bei befreundeten Zahlen erhalten wir eine Periode von 2. Wenn wir im Rahmen der Folge auf eine Primzahl stoßen, ist der nächste Wert eine 1, worauf die Null folgt und die Folge damit endet. Nach der Catalan-Dickson-Vermutung enden all diese Folgen entweder in der 0 oder in einer Periode. Aber bislang wurde diese Vermutung weder bewiesen noch widerlegt. Zahlen, die einer Periode mit einer

Länge von über zwei angehören, werden auch gesellige Zahlen genannt. Ein Beispiel für gesellige Zahlen sind 12496, 14288, 15472, 14536 und 14264.

Entwickeln Sie ein Programm, das perfekte, befreundete oder gesellige Zahlen in einem gegebenen Intervall findet. Da dies durchaus rechenintensiv werden kann, ist die Suche auf eine vorgegebene Zahl von Prozessoren zu parallelisieren. Beispiel:

```
$ aliquot 4 80000000 82000000
80422335, 82977345
81128632, 91314968, 96389032, 91401368
```

Der erste Parameter (hier im Beispiel die 4) gibt die Zahl der zu erzeugenden Unterprozesse an und die Zahlen 80000000 und 82000000 der Bereich, in dem gesucht wird.

Der Hauptprozess richtet dabei eine entsprechende Zahl von Pipelines ein, die er lesenderweise nutzt, während die Unterprozesse sie schreibenderweise verwenden. Jede der Unterprozesse arbeitet dann in einem entsprechenden Teilintervall. Für jede gefundene Gruppe von perfekten, befreundeten oder geselligen Zahlen, bei denen die niedrigste Zahl in das Teilintervall fällt, wird die niedrigste Zahl binär in die Pipeline geschrieben. Wenn die Suche beendet ist, enden die Unterprozesse jeweils. (Achten Sie auch darauf, dass jeder Unterprozess auch die Pipelines für die zuvor erzeugten Unterprozesse schließt.)

Nachdem die Unterprozesse allesamt erzeugt und die unbenötigten Pipe-Enden geschlossen worden sind, beginnt der Hauptprozess mit dem Auslesen der einzelnen Pipelines. Für jede eingelesene Zahl wird noch einmal die gesamte Folge bestimmt und ausgegeben.

Es ist dabei sicherzustellen, dass keine Gruppe von befreundeten oder geselligen Zahlen mehrfach ausgegeben wird.

Im Rahmen der Aufgabe müssen Sie nur Folgen betrachten, bei der alle Elemente noch als **unsigned int** darstellbar sind. Achten Sie penibel auf mögliche Überläufe. Wenn eine Addition zu einem Überlauf führen würde, brechen sie die weitere Betrachtung der Folge ab. Ebenso dürfen Sie die Betrachtung abbrechen, wenn die Folge zu lang wird, ohne dass Sie eine Periode entdecken. Die sogenannten Lehmer-Six, also 276, 552, 564, 660, 840 und 966, werden Sie wohl nicht mit Ihrer Lösung knacken können.

Hinweise:

Bitte testen Sie Ihre Lösung nicht auf einem unserer Server (Thales und Theseus sind hierfür tabu), sondern verwenden Sie z.B. die Quadcore-Maschinen aus unserem Pool, bei denen Sie mit vier Unterprozessen arbeiten können.

In welche Zahlenbereiche Sie sinnvoll vordringen können, hängt davon ab, wie effizient Sie die Teilersumme einer Zahl bestimmen können. Die triviale, aber sehr ineffiziente Methode geht für n sukzessive die Zahlen von 1 bis $\frac{n}{2}$ durch, um jeweils zu testen, ob sie Teiler sind. Das ist zulässig für die Lösung, ist aber nur für relativ kleine Zahlen zumutbar.

Eine bessere Lösung sollte sich an einer effizienten Primfaktorzerlegung einer Zahl versuchen und dann daraus rekursiv alle Teiler bestimmen. Für die Primfaktorzerlegung empfiehlt es sich, ein Array der Primzahlen von 2 bis 65521 zu halten und dann diese Primzahlen sukzessiv als Teiler zu testen.

Versuchen Sie, die Aufgabe modular zu entwickeln. Dies lässt sich auch gerade im Rahmen eines Teams gut umsetzen. Für jedes Modul sollten sie jeweils eine Header-Datei und

eine zugehörige Implementierung haben. Sie könnten entsprechend ein Modul für die Primfaktorzerlegung haben, eines für die rekursive Erzeugung aller Teiler, eines für die Bestimmung der Teilersumme, eines das von einer Zahl n beginnend, die Folge untersucht, eines das ein Teilintervall betrachtet und ein Hauptprogramm, das sich um die Aufgabenteilung und die Parallelisierung kümmert.

Die Zahlen sollten über die Pipes binär geschrieben und gelesen werden. Sie sollten überprüfen, ob dies jeweils klappt, aber Sie müssen nicht darauf vorbereitet sein, dass Zahlen nur teilweise geschrieben oder gelesen werden.

Prinzipiell ist es möglich, dass einer der Unterprozesse schlafen gelegt wird, weil die Pipeline vollgeschrieben ist und der Hauptprozess noch nicht dazu kam, sie auszulesen. Zu einem Deadlock kommt es nicht, da der Hauptprozess irgendwann zum Lesen kommen wird, aber in einem solchen Szenario wird das Parallelisierungspotential nicht voll ausgenutzt. In diesem Anwendungsbeispiel ist das jedoch kein Problem, wenn das zu untersuchende Intervall nicht zu umfangreich gewählt wird.

Ihre Lösung können Sie mit Hilfe von *tar* verpacken und dann mit *submit* einreichen:

```
thales$ tar cvf aliquot.tar *.[ch] [mM]akefile
thales$ submit ss2 6 aliquot.tar team
```

Aufgabe 7: Wie ist das Wetter?

Es gibt diverse freie Webangebote, über das aktuelle Wetter in bestimmten Orten abgerufen werden kann. Manche davon bieten auch ein Ausgabeformat an, das sich leicht von einem Programm auswerten lässt. Unter diesen Angeboten findet sich auch Yahoo, das mit YQL eine SQL-ähnliche Abfragesprache zur Verfügung stellt. Beispielsweise liefert diese Abfrage zum Ulmer Wetter

select *item.condition* **from** *weather.forecast*
where *u*='c' **and** *woeid* **in** (
  select *woeid* **from** *geo.places*(1)
  where *text*='Ulm,␣Germany'
)

folgende Antwort im JSON-Format:

```
{
   "query": {
      "count": 1, "created": "2017-05-30T08:38:13Z", "lang": "en-US",
      "results": {
         "channel": {
            "item": {
               "condition": {
                  "code": "28",
                  "date": "Tue, 30 May 2017 10:00 AM CEST",
                  "temp": "21", "text": "Mostly Cloudy"}}}}}
```

All die Leerzeichen und Zeilentrenner wurden hier der Lesbarkeit wegen hinzugefügt. In der originalen Antwort kommen Leerzeichen nur innerhalb der Anführungszeichen vor.

Mit Hilfe des Kommandos *wget* lässt sich diese Abfrage leicht durchführen: So liefert das Kommando „`wget -O- -q --no-check-certificate` *url*“ die gewünschten Daten auf der Standardausgabe. Die URL besteht hierbei aus den folgenden Komponenten:

- Der Server: „https://query.yahooapis.com/v1/public/yql“,
- gefolgt von einem Fragezeichen: „?“ und den einzelnen Parametern, die durch „&“ getrennt werden:
  - Die YQL-Abfrage kommt hinter „q=“ und
  - das gewünschte Ausgabe-Format wird mit „format=json“ spezifiziert.

Bei den Parameterwerten müssen alle Zeichen, die kein Buchstabe und keine Ziffer sind, in einer geschützten Notation angegeben werden mit einem Prozentzeichen und zwei Hexziffern, die den Ordinalwert des Zeichens angeben. Beispiele: „%20“ ist für ein Leerzeichen anzugeben, „%3D“ für „=“.

Wie die konkrete *wget*-Abfrage in dem Beispiel aussieht, kann in *ulm.sh* gesehen werden.

Entwickeln Sie ein kleines Bibliotheksmodul, das eine Funktion zum Abruf des Wetters anbietet und das hierzu eine Pipeline aufbaut zu einem passenden *wget*-Kommando und dessen Standardausgabe auswertet. Die Funktion sollte die Ortsbezeichnung (wie beispielsweise „Ulm, Germany“) als Parameter erhalten und dann zumindest die Temperatur liefern und bei Interesse gerne auch weitere Infos.

Es empfiehlt sich, die gesamte Ausgabe von *wget* in einen dynamisch wachsenden Puffer einzulesen und anschließend in diesem Puffer nach den gewünschten Infos zu suchen. So kann beispielsweise mit Hilfe der Standard-Funktion *strstr* nach der Zeichenkette „`"temp"`“ gesucht werden, um die Stelle mit der Temperaturangabe zu finden.

Achten Sie bei der Schnittstelle darauf, dass die Abfrage auch schiefgehen kann und entsprechend der Erfolg feststellbar sein muss.

Es steht Ihnen frei, für die Aufgabe die Vorlesungsbibliothek zu nutzen. Die Standard-Funktion *popen* darf jedoch nicht verwendet werden.

Testen Sie Ihr Bibliotheksmodul mit einem kleinen Testprogramm.

Ihre Lösung können Sie wiederum mit Hilfe von *tar* verpacken und dann mit *submit* einreichen:

```
thales$ tar cvf weather.tar *.[ch] [mM]akefile
thales$ submit ss2 7 weather.tar team
```

Viel Erfolg!