Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

⭐-Aufgabe: Sitze in einem Parlament

Revision 8add8da (2021-06-01)

Schreibe ein Programm in C/C++, Java, Python oder einer anderen gängigen Programmiersprache (kein Pseudocode), welches das folgende Problem löst: Verteile die m Sitze in einem Parlament nach einer Wahl auf n Parteien.

Die Platzvergabe verläuft nach dem *D'Hondt-Verfahren*: für $i \in \{1,\dots,n\}$ bezeichne $v_i \in \mathbb{N}$ die Anzahl der Stimmen für Partei i. Für jede Partei i wird ein Quotient q_i berechnet, welcher anfangs auf $q_i := v_i/1$ gesetzt wird. Hat Partei j den größten Quotienten, wird ihr ein Sitz zugeteilt. Anschließend wird ihr Quotient folgendermaßen aktualisiert:

$$q_j := \frac{v_j}{s_j + 1},$$

wobei s_j die Anzahl der Sitze, welche bisher Partei j zugeordnet wurden, bezeichnet. Anfangs wird die Anzahl der zugeordneten Sitze für alle Parteien auf 0 gesetzt. Dieser Vorgang wird wiederholt, bis alle m verfügbaren Sitze vergeben sind.

Eingabe. Die Datei besteht aus mehreren Zeilen. In der ersten Zeile sind n und m durch ein Leerzeichen getrennt gegeben. Beide Zahlen sind in der Menge $\{1,\dots,2\,000\,000\}$ enthalten. In der i-ten der n darauffolgenden Zeilen ist die ganze Zahl v_i gegeben. Jede Partei erhält mindestens eine Stimme. Die Anzahl aller Stimmen ist mindestens so groß wie die Anzahl der zu verteilenden Sitze, es gilt also $v_1 + \cdots + v_n \geq m$. Unsere Eingaben sind so konstruiert, dass der letzte Sitz eindeutig zugeordnet werden kann – es muss keine Logik eingebaut werden, welche bei Gleichstand entscheidet.

Ausgabe. Die Sitzverteilung, wobei sich die Reihenfolge in der Ausgabe mit der Reihenfolge in der Eingabe vertragen soll. Betrachte hierzu die Beispiele.

Beispiele.

`1.in`

```
2 2
10
10000000
```

`1.ans`

```
0
2
```

`2.in`

```
2 3
12
11
```

`2.ans`

```
2
1
```

`3.in`

```
2 4
12
11
```

`3.ans`

```
2
2
```

4.in	4.ans
2 4 17 10	3 1

Erklärung zu Beispiel 4. Es traten 2 Parteien zur Wahl an, und insgesamt sollen 4 Sitze vergeben werden. Partei 1 erhält den ersten Sitz, da sie die meisten Stimmen erhalten hat. Anschließend wird ihr Quotient auf $q_1 = \frac{17}{2}$ gesetzt. Partei 2 erhält den nächsten Sitz, denn $10 > \frac{17}{2}$. Anschließend wird ihr Quotient auf $q_2 = \frac{10}{2} = 5$ gesetzt. Der dritte Sitz geht an Partei 1, denn es gilt $q_1 = \frac{17}{2} > 5 = q_1$. Der Quotient von Partei 1 wird anschließend auf $q_1 = \frac{17}{3}$ gesetzt. Auch der letzte Sitz geht an Partei 1, denn es gilt $q_1 = \frac{17}{3} > 5 = q_2$. Insgesamt erhält Partei 1 also drei Sitze und Partei 2 erhält einen.

5.in	5.ans
4 14 38 35 36 37	4 3 3 4

Größere Beispiele. Teste dein Programm! Hier sind größere Beispiele: https://tcs.uni-frankfurt.de/teaching/summer21/algo1/seatallocation-tests-v2.zip

Stell sicher, dass dein Programm für alle Eingaben `X.in` *exakt* die entsprechende Ausgabedatei `X.ans` erzeugt.

Tipps. Welche Datenstruktur eignet sich, um diese Aufgabe zu lösen? Achte auf Rundungsfehler und möglichen Überlauf von `int`s.

Hinweise zur Abgabe. Die Datei `v2-015-secret.ans` fehlt. Deine Abgabe soll die SHA1-Summe der korrekten Datei enthalten. Zum Beispiel erhält man die SHA1-Summe der Datei `v2-014-19.ans` wie folgt:

```
$ sha1sum v2-014-19.ans
22d43373a4104696005cb3db1fa8f3f0c873090a  v2-014-19.ans
```

Deine Abgabe soll wie immer per PDF erfolgen und die grobe Idee, den diesmal echten Code, den Korrektheitsbeweis, die Laufzeitanalyse, und die SHA1-Summe von `v2-015-secret.ans` enthalten. Weiterhin zu beachten:

- Der Code darf maximal **60** Zeilen lang sein (jede Zeile mit maximal 100 Zeichen). Möglichst kurz und elegant! (Eine 20-Zeilen Lösung in Python ist möglich. Kommentare zählen nicht dazu.)
- Falls die genutzte Datenstruktur in der Programmiersprache eingebaut ist, darf diese benutzt werden. Falls die Datenstruktur selbst implementiert wird, zählt diese Implementierung nicht zum Zeilenlimit dazu. Wichtig: Die Datenstruktur darf nur so benutzt werden, wie die in der Vorlesung beschriebene abstrakte Datenstruktur das erlaubt. Falls zum Beispiel eine Warteschlange benutzt wird, dürfen nur die entsprechenden Funktionen enqueue, dequeue und is_empty verwendet werden.