Übung 2

Abgabe: 24. April 2023, 08:00 Uhr

Hinweise zur Abgabe

Testen Sie Ihr Programm bitte lokal auf Ihrer Maschine, bevor Sie Ihre Lösung einreichen.

Sie können nicht nur lokal kompilieren, sondern auch Ihr Programm auf eigenen Eingaben und den Testfällen in `testcases` ausprobieren. Der Server bewertet Ihr Programm mit dem Aufruf `python3 eval.py`. Wenn Sie alle Abhängigkeiten (`python3, gcc, diff, grep`) lokal installiert haben, können Sie das Testskript auch lokal laufen lassen und erhalten so eine Vorschau auf die Bewertung.

Die automatische Bewertung kann Ihre Abgabe auch lokal nur dann bewerten, wenn Sie sich an das angebene Ausgabeformat *genau* halten und die richtige Datei modifizieren.

Aufgabe 2.1 [10 Punkte, Link zur Abgabe] In dieser Aufgabe sollen Sie eine binäre Suche implementieren. Die Eingabe enhält zunächst eine Zeile mit einer ganzen Zahl `x`, dem Suchschlüssel. Auch die zweite Zeile enthält eine natürliche Zahl $n \in \mathcal{N}_0$, die Länge der Eingabefolge. Es folgt die Eingabefolge als `n` ganze Zahlen $a_0, \ldots, a_{n-1}$ in je einer Zeile; dabei kommt keine Zahl doppelt vor. Ihr Programm erhält die Eingabe als Text auf der Standardeingabe `stdin`.

Ihr Programm soll den Index $i \in \{0, \ldots, n-1\}$ von x in der Zahlenfolge $a_0, \ldots, a_{n-1}$ ausgeben (d.h., ein i mit $a_i = x$). Falls x nicht in $a_0, \ldots, a_{n-1}$ vorkommt, soll das Programm n ausgeben.

Damit die binäre Suche funktioniert, muss die Eingabefolge $a_0, \ldots, a_{n-1}$ aufsteigend sortiert sein (es muss $a_0 < \cdots < a_{n-1}$ gelten). Überprüfen Sie mit einer Assertion, ob das tatsächlich der Fall ist!

Im Github-Repository finden Sie eine Datei `binsearch.c`, die bereits den fertigen Code zum Einlesen der Eingabe enthält. Sie dürfen diesen Code verwenden.

Aufgabe 2.2 [10 Punkte, Link zur Abgabe] In dieser Aufgabe geht es darum, sich mit den Debuggingtechniken aus der Vorlesung vertraut zu machen. Dazu betrachten wir den Versuch einer Merge-Sort-Implementierung.

Genauer finden Sie im Github-Repository eine Datei `mergesort.c`. In dieser Datei ist bereits MergeSort implementiert. Leider sind beim Implementieren einige Fehler passiert – suchen Sie die Fehler mit dem Debugger und geeigneten Assertions.

Die Implementierung erhält die Eingabe als Text auf der Standardeingabe `stdin`. Die Eingabe enhält zunächst eine Zeile mit einer Zahl `n`, dann folgen `n` ganze Zahlen $a_1, \ldots, a_n$ in je einer Zeile.

Die Ausgabe des Programmes soll folgendermaßen aussehen: Nach jedem `merge`-Aufruf soll eine Zeile mit dem gerade gemergten Teilarray ausgegeben werden (Arrayeinträge getrennt durch Leerzeichen). Dazu soll die Funktion `print_subarray` verwendet werden.

Abschließend soll einmal das sortierte Array mit der Funktion `print_array` ausgegeben werden.

Sie erhalten 2 Punkte für die sinnvolle Verwendung von Assertions (diese zwei Punkte werden nicht von der automatischen Korrektur erfasst). Damit die automatische Korrektur funktioniert, dürfen Sie die Ausgabefunktionen nicht verändern!

Aufgabe 2.3 [10 Punkte, Link zur Abgabe] Die Buchhaltung Ihrer Firma braucht Unterstützung: Im Laufe des Jahres sind viele Dinge bestellt worden und so sind eine Menge Rechnungen aufgelaufen. Einige davon sind auch bezahlt worden, aber niemand weiß mehr, welche Rechnungen noch nicht beglichen sind... Ihre Aufgabe ist es, ein Programm zu schreiben, dass die unbezahlten Rechnungen identifiziert und berechnet, auf welchen Gesamtbetrag sich die unbezahlten Rechnungen belaufen.

Netterweise haben Ihre Kollegen schon Vorarbeit geleistet und stellen Ihnen eine Textdatei zur Verfügung, die alle wichtigen Eckdaten – Rechnungsbeträge und getätigte Zahlungen – enthält.

Die erste Zeile der Textdatei enthält zwei natürliche Zahlen $n, m \in \mathbb{N}_0$ getrennt durch ein Leerzeichen: n ist die Anzahl der getätigten Bestellungen, $m \leq n$ ist die Anzahl der bezahlten Rechnungen.

Es folgen n Zeilen die jeweils eine Rechnungsnummer und einen Rechnungsbetrag enthalten. Der Rechnungsbetrag ist ≥ 0 und ganzzahlig (wenn Sie möchten, können Sie sich vorstellen, dass die Cent-Beträge in den letzten zwei Stellen der ganzen Zahl codiert sind).

Abschließend folgen m Zeilen, die die getätigten Zahlungen repräsentieren. Auch diese Zeilen enthalten jeweils eine Rechnungsnummer `id` und einen Rechnungsbetrag `b` (ganzzahlig, ≥ 0). Die Interpretation ist hier, dass die Rechnung mit Nummer `id` in Höhe von `b` bezahlt wurde.

Der Einfachheit halber dürfen Sie davon ausgehen, dass Rechnungen entweder gar nicht oder in voller Höhe bezahlt werden (es kann also nicht passieren, dass für eine Rechnung in Höhe von 300 EUR nur 200 EUR überwiesen werden).

Schreiben Sie ein Programm, dass für jede unbezahlte Rechnung eine Zeile mit der zugehörigen

Rechnungsnummer ausgibt (Wenn alle Rechnungen bezahlt wurden, soll dementsprechend keine Rechnungsnummer ausgegeben werden). Als letzte Zeile soll ihr Programm in jedem Fall die Gesamtschuld, also die Summe der unbezahlten Rechnungsbeträge, ausgeben. Alle Zahlen sollen als ganze Zahlen ausgegeben werden. Verwenden Sie eine `struct`!

Für eine Herausforderung: Schreiben Sie eine Lösung, die in Laufzeit $\mathcal{O}(n \log n)$ läuft ohne zusätzliche Datenstrukturen außer Arrays/Structs zu verwenden (für die volle Punktzahl nicht gefordert).