

Programmierung für Naturwissenschaften 2
Sommersemester 2021
Übungen zur Vorlesung: Ausgabe am 10.06.2021

In der Übung am 10.6.2021 soll die Online-Evaluation der Vorlesung und Übungen in PfN2 erfolgen. Das ist über das Evaluationsportal <https://uhh.de/evasys> mit der Lösung 9D56J möglich. Alternative kann die URL

<https://evasys-online.uni-hamburg.de/evasys/online.php?pswd=9D56J> genutzt werden.

Aufgabe 9.1 (2 Punkte)

Aus Zeitgründen können wir in diesem Semester in der Vorlesung das Thema Auswertung von Binomial-Koeffizienten (siehe `numerical.pdf`, Seite 82-97) nicht behandeln. Da das ein wichtiges Thema ist, wird es in dieser Peer Teaching Aufgabe behandelt.

In jeder Kleingruppe muss sich ein Studierender auf der Basis der oben genannten Seiten auf das Thema vorbereiten und das erworbene Wissen an die anderen Mitglieder der Kleingruppe weitergeben. Es sollte ein Studierender sein, der bisher nicht schon zweimal bei früheren Peer Teaching Aufgaben in diesem Semester diese Rolle übernommen hat. Falls es mehrere Studierende gibt, auf die das zutrifft, dann sprechen Sie sich rechtzeitig untereinander ab, wer die Vorbereitung übernimmt.

Die Dateien mit den entsprechenden Programmcodes finden Sie in den Materialien.

Nach der Übung dokumentiert jede Kleingruppe in der Datei `bearbeitung.txt` das Vorgehen bei der Erarbeitung des Themas und ggf. noch bestehende Verständnisfragen oder Hinweise zu Unklarheiten in den Folien. Willkommen sind natürlich auch Bemerkungen zur Lehrform selbst.

Aufgabe 9.2 (4 Punkte) In dieser Aufgabe geht es darum, die Datenstruktur `FSPrioStore` (siehe Aufgabe 6.2) als template-basierte Klasse in C++ zu implementieren. Durch die Verwendung von Templates und die Nutzung eines Funktionszeigers für die Vergleichsfunktion kann die Implementierung der C++-Klasse einfacher wiederverwendet werden als die C-Implementierung.

Ihre Implementierung der Klasse `FSPrioStore` soll in der Datei `fs_prio_store.hpp` erfolgen. Diese erhalten Sie durch Umbenennung der Datei `fs_prio_store_template.hpp`. Deklarieren Sie die Member-Variablen und Methoden der Klasse entsprechend der Musterlösung in C (siehe `fs_prio_store.c`). Beachten Sie, dass der Basistyp der zu speichernden Elemente durch den Template-Parameter `T` spezifiziert wird. Während die Namen der C-Methoden jeweils mit `fs_prio_store` beginnen, entfällt dieser Präfix bei den Methode der Klasse. Der Konstruktor und Destruktor werden entsprechend der Regeln in C++ benannt. Für den indexbasierten Zugriff wird die Technik der Operator-Überladung genutzt. Die folgende Tabelle zeigt die Namen der C-Funktionen und der entsprechenden C++-Methoden der Klasse `FSPrioStore`:

C-Funktion	Methode aus FSPrioStore
<code>fs_prio_store_new(...)</code>	<code>FSPrioStore(...)</code>
<code>fs_prio_store_delete(...)</code>	<code>~FSPrioStore()</code>
<code>fs_prio_store_size(...)</code>	<code>size()</code>
<code>fs_prio_store_is_full(...)</code>	<code>is_full()</code>
<code>fs_prio_store_at(...)</code>	<code>operator[](...)</code>
<code>fs_prio_store_add_smallest(...)</code>	<code>add(...)</code>
<code>fs_prio_store_add_largest(...)</code>	<code>add(...)</code>

Beachten Sie, dass es in der C++-Klasse `FSPrioStore` nur eine Methode `add` gibt. Durch Verwendung eines Funktionszeigers `compare` können verschiedene Ordnungen spezifiziert werden.

In der Datei `fs_prio_store_mn.cpp` finden Sie die Implementierung einer Klasse `KeyIndexPair`, Unit-Tests und ein Hauptprogramm, das fiktive experimentelle Daten einliest. Durch

```
./experiment.py 5000000 | ./fs_prio_store.x sl 10
```

werden $5 \cdot 10^6$ Zeilen mit zufälligen Messwerten generiert, durch `fs_prio_store.x` werden sie eingelesen, prozessiert und die zehn kleinsten und größten Messwerte werden ausgegeben. Durch `make test` verifizieren Sie die Korrektheit Ihrer Implementierung für verschiedene Testfälle.

Beantworten Sie bitte die in der Datei `fs_prio_store_mn.cpp` formulierten Fragen. Einige der Fragen lassen sich leicht beantworten, wenn man den entsprechenden Programmcode mit `#ifdef NEWCODE ... #endif` klammert, also nicht kompiliert. Die Antwort ergibt sich dann meist aus der entstehenden Fehlermeldung.

2 Pkte

Punkteverteilung:

- Implementierung der Klasse `FSPrioStore`: 2 Punkte
- Beantwortung der Fragen: 2 Punkte

Aufgabe 9.3 (5 Punkte)

In dieser Aufgabe geht es darum, verschiedene Hash-Funktionen für alle Worte $words(T)$ in einem Text T zu bewerten. Alle Studierende, die den Begriff Hash-Funktion noch nicht kennen, sollten sich dazu die Folien in der Datei `hashfunctions.pdf` ansehen. Sei h eine Hash-Funktion, die für alle Worte w über einem Alphabet einen Hash-Wert $h(w) \in \mathbb{N}_0$ liefert. Sei $H(h, T) = \{h(w) \mid w \in words(T)\}$ die Menge aller Hash-Werte von Worten des Textes T .

Das Kollisionsmaß einer Hash-Funktion wird durch die Anzahl der Kollisionen bestimmt. Eine Kollision tritt auf, wenn verschiedene Worte den gleichen Hash-Wert haben. Sei daher $f(h, T, i)$ die Anzahl der Worte $w \in words(T)$ mit $h(w) = i$. Dann soll

$$\text{hashcoll}(h, T) = \frac{1}{|H(h, T)|} \sum_{i \in H(h, T)} f(h, T, i)^2$$

das Kollisionsmaß von h bzgl. T sein. Falls es keine Kollisionen gibt, dann ist $f(h, T, i) = 1$ für alle $i \in H(h, T)$. Damit gilt $\text{hashcoll}(h, T) = 1$, d.h. die Hash-Funktion h hat bzgl. T das geringste Kollisionsmaß. Je mehr Kollisionen es gibt, umso größer ist $\text{hashcoll}(h, T)$.

Beispiel: Wir betrachten einen Text T mit 15 Worten und eine der implementierten Hash-Funktionen $h = \text{ELFHash}$, deren Werte in der folgenden Tabelle angegeben sind:

w	$h(w)$
BUT	18 340
Act	18 340
Add	18 340
Last	338 084
Meet	342 980
Heard	5 159 044
Guard	5 159 044
Herod	5 163 348
Inherit	5 163 348
Sibyl	5 896 700
Sicil	5 896 700
Adding	75 149 383
Acting	75 149 383
Always	75 749 635
penalties	75 749 635

Offensichtlich gibt es 3 Worte mit dem gleichen Hash-Wert 18 340 und 5 Paare von Worten jeweils mit dem gleichen Hash-Wert, wie man leicht in der folgenden Tabelle sieht:

i	$\{w \mid h(w) = i\}$	$f(h, T, i)$
18 340	BUT Act Add	3
338 084	Last	1
342 980	Meet	1
5 159 044	Guard Heard	2
5 163 348	Herod Inherit	2
5 896 700	Sibyl Sicil	2
75 149 383	Adding Acting	2
75 749 635	Always penalties	2

Damit ist $|H(h, t)| = 8$ und

$$\text{hashcoll}(h, T) = \frac{3^2 + 1 + 1 + 5 \cdot 2^2}{8} = \frac{31}{8} = 3.875.$$

Schreiben Sie ein C++-Programm `hashcoll.cpp`, das für alle Hash-Funktionen, die in der Datei `hashfunctions.cpp` implementiert sind, das Kollisionsmaß bzgl. eines gegebenen Textes bestimmt.

Gehen Sie dabei wie folgt vor:

1. Öffnen Sie die Datei, deren Name durch `argv[1]` referenziert wird. Ein entsprechendes Beispiel finden Sie im Abschnitt zur Codon-Translation in der Datei `Cpp_slides.pdf`.
2. Die Adresse der resultierenden Instanz der Klasse `std::ifstream` wird an die Funktion `file2wordset` aus `tokenizer.cpp` übergeben. Diese Funktion liefert die Worte in der Datei als Menge vom Typ `std::set<std::string>` zurück.
3. Die Anzahl der Hash-Funktionen liefert die Funktion `hashfunction_number`. Für alle i zwischen 0 und `hashfunction_number-1` liefert `hashfunction_get` Informationen zur i -ten Hash-Funktion als Zeiger auf eine Struktur vom Typ `Hashfunction`. Diese enthält den Namen und einen Zeiger auf die Hash-Funktion.

4. In einer Schleife über alle Hash-Funktionen wenden Sie die aktuelle Hash-Funktion nun auf alle Worte aus der Menge an und speichern Sie die Hash-Werte in einer geeigneten Datenstruktur. Damit können Sie für alle $i \in H(h, T)$ den Wert $f(h, T, i)$ bestimmen. Hierzu gibt es zwei Möglichkeiten:
- Man speichert für eine gegebene Hash-Funktion alle Hash-Werte in einem Array, sortiert dieses und kann dann in einem linearen Durchlauf die Häufigkeit eines jeden Hash-Wertes bestimmen.
 - Man nutzt eine `map`, die Hash-Werte (als Schlüssel) auf Ihre Häufigkeit (als Werte) abbildet.

Aus den Häufigkeiten der Hash-Werte kann man schließlich das Kollisionsmaß der Hash-Funktion berechnen.

Beachten Sie, dass die Elemente der Menge, die `file2wordset` liefert, Instanzen der Klasse `std::string` sind. Ein Iterator `it`, mit dem man über die Elemente iteriert, liefert also mit dem Dereferenzierungsoperator Instanzen der Klasse `std::string`. Die Hash-Funktion, die auf Strings angewendet werden soll, erwartet jedoch einen C-String, d.h. einen Zeiger auf einen Speicherbereich mit Basistyp `char`, der mit `\0` endet. Für die Konvertierung von `std::string` in einen C-String können Sie die Methode `c_str()` verwenden, d.h. der Ausdruck `it->c_str()` liefert den geforderten C-String.

Als Ausgabe soll Ihr Programm für jede Hash-Funktion den Namen sowie das Kollisionsmaß der Hash-Funktion in einer Tabulator-separierten Zeile ausgeben. Diese Zeilen sollen aufsteigend nach dem Kollisionsmaß sortiert sein. Wenn die Kollisionsmaße von zwei Hashfunktionen sich um weniger als 10^{-8} unterscheiden, sollen sie als identisch betrachtet werden. In diesem Fall soll die Sortierung lexikographisch aufsteigend nach dem Namen der Hash-Funktion erfolgen.

In den Materialien finden Sie ein Makefile zum Kompilieren Ihres Programms sowie Testdateien mit dem erwarteten Ergebnis. Durch `make test` verifizieren Sie die Korrektheit Ihrer Implementierung für diese Testdaten.

Punkteverteilung:

- 1 Punkt für die korrekte Berechnung der Wort-Menge.
- 1 Punkt für die Iteration über alle Hash-Funktionen zur Berechnung der Hash-Werte für die Wort-Menge
- 1 Punkt für die Berechnung der Kollisionsmaße für alle Hash-Funktionen
- 1 Punkt für die Sortierung entsprechend der Aufgabenstellung
- 1 Punkt für die bestandenen Tests

Bitte die Lösungen zu diesen Aufgaben bis zum 15.06.2021 um 18:00 Uhr an pfn2@zbh.uni-hamburg.de schicken.