S. Kurtz, T. Schwabe,
G. Gonnella, D. Willrodt

Universität Hamburg
Zentrum für Bioinformatik

Programmierung für Naturwissenschaften
Sommersemester 2015
Übungen zur Vorlesung: Ausgabe am 04.06.2015

Punkteverteilung: Aufgabe 8.1: 4 Punkte, Aufgabe 8.2: 6 Punkte

Abgabe bis zum 10.06.2015, 10:00 Uhr.

Aufgabe 8.1 In einem vorherigen Übungsblatt wurde der Datentyp `IntSet` zur effizienten Repräsentation von Mengen nicht-negativer ganzer Zahlen implementiert. In den Materialien zu dieser Übung finden Sie die entsprechende Musterlösung (`intset.c`). Für die Elemente der zu repräsentierenden Menge wurden Werte über dem Basistype `uint16_t` verwendet. Durch leichte Anpassungen des Quellcodes erhält man eine Implementierung über dem Basistyp `uint8_t` (Bytes) oder Basistyp `uint32_t` (32-bit unsigned integer), die je nach Wert von `maxvalue` und `nofelements` ggf. speichereffizienter ist. Daher erscheint es sinnvoll, drei verschiedene Implementierungen für die drei genannten Basistypen zu realisieren und dann ausgehend von den Werten für `maxvalue` und `nofelements` jeweils die auszuwählen, die zum geringsten Speicherverbrauch führt. Damit der Quelltext nicht dreimal in fast identischer Form vorliegt, sollen Sie den Datentyp `IntSet<Basetype>` in C++ als Template-Klasse über dem Basistyp `Basetype` implementieren. In der Datei `intset.hpp` finden Sie die folgende Deklaration dieser Template-Klasse:

```
template <typename Basetype>
class IntSet
{
  // we use c-style arrays for speed and to make it easier to adapt the "old"
  // code from previous exercises
  Basetype*      elements;
  unsigned long* sectionstart;
  unsigned long  nextfree,
                 maxelement,
                 currentsectionnum,
                 numofsections,
                 nofelements,
                 previouselem;
  int            logsectionsize;

  public:
    IntSet(unsigned long maxelement, unsigned long nofelements);
    ~IntSet();
    void add(unsigned long elem);
    bool is_member(unsigned long elem) const;
    unsigned long number_next_larger(unsigned long pos) const;
    void pretty_print(void) const;

    static size_t size(unsigned long maxelement,
                       unsigned long nofelements);
};
```

Sie sollen alle `public`-Mitglieder der Template-Klasse `IntSet` nun in einer Datei `intset-impl.hpp` implementieren.

Beachten Sie, dass `size` eine Klassenfunktion ist.

In den Materialien zur Übung finden Sie ein Hauptprogramm in Datei `intset-main.cpp` mit dem Quelltext zur Bestimmung des besten Basistyps, d.h. des Basistyps, der zu einer Repräsentation minimaler Größe führt. Für die mit dem besten Basistyp instantiierte Klasse werden dann die Testfälle verifiziert.

Die Materialien enthalten ein `Makefile`. Durch `make` wird ihr Programm compiliert. Ist Ihre Implementierung komplett, dann soll `make test` ohne Fehler durchlaufen.

Aufgabe 8.2 In dieser Aufgabe geht es darum, die Qualität verschiedener Hash-Funktionen für alle Worte $words(T)$ in einem Text T zu bestimmen. Sei h eine Hash-Funktion, die für alle Worte w über einem Alphabet einen Hash-Wert $h(w)$ liefert. Sei $H(h,T) = \{h(w) \mid w \in words(T)\}$ die Menge aller Hash-Werte von Worten des Textes T. Die Qualität 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

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

die Qualität 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 $\mathsf{hashqual}(h,T) = 1$, d.h. die Hash-Funktion h hat bzgl. T die optimale Qualität. Je mehr Kollisionen es gibt, umso grösser ist $\mathsf{hashqual}(h,T)$.

Beispiel: Wir betrachten einen Text T mit 15 Worten und eine der implementierten Hashfunktion $h =$ 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

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

Schreiben Sie ein C++-Programm `hashqual.cpp`, dass für alle Hash-Funktionen, die in der Datei `hashqual-functions.cpp` (in der Materialien zur Übung in STiNE vorhanden) implementiert sind, die Qualität bzgl. eines gegebenen Textes bestimmt.

In der Materialien finden Sie außerdem eine Textdatei (`shaks.data`) und das erwartete Ergebnis des Programms (`shaks.quality`), wenn es auf diese Datei wie folgt angewandt wird:

```
cat shaks.data | hashqual.x
```

Ebenfalls finden Sie in STiNE eine modifizierte `tokenizer`-Funktion, die `std:cin` verwendet und für das Zerlegen der Eingabedatei in Worte genutzt werden soll. Speichern Sie die gefundenen Worte als Menge in einem Container vom Typ `str_set` (siehe Skript zur Vorlesung: `C++Rastofer-handout.pdf`). Wenden Sie die Hash-Funktion nun auf alle Worte aus der Menge an und speichern Sie die Hash-Werte in einer geeigneten Datenstruktur, um für alle $i \in H(h,T)$, den Wert $f(h,T,i)$ zu bestimmen. Daraus können Sie schließlich die Qualität der Hash-Funktion ermitteln. Als Ausgabe soll Ihr Programm für jede Hash-Funktion den Namen ausgeben, sowie die Qualität der Hash-Funktion. Die entsprechenden Zeilen sollen aufsteigend nach der Qualität sortiert sein.

Wenn Ihre Implementierung korrekt ist, soll def Aufruf `make test` fehlerfrei durchlaufen.

Die Lösungen zu diesen Aufgaben werden am 11.06.2015 besprochen.