Lehrstuhl für Informatik 3
Rheinisch-Westfälische Technische Hochschule
Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Aachen, 25. 05. 2007

Nagl, Ranger, Wörzberger

Übungen zur Vorlesung „Grundgebiete der Informatik 2: Algorithmen und Programmiertechniken“

— Lösung zu Blatt 7 —

14. Aufgabe

```
#include <limits.h>             // fuer UINT_MAX
/* Hilfsfunktion: Sucht den minimalen Eintrag aus dem Histogramm.*/
void sucheMinimalenEintrag(HistogrammT dasHistogramm,
                           HistogrammEintragT * &minimalerEintrag)
{
   minimalerEintrag           = NULL;

   if (NULL != dasHistogramm) {
      /* Der Cursor ueber den unsortierten Anteil der Liste. */
      HistogrammEintragT * cursor;

      /* Der Vergleichswert */
      unsigned int minimalerWert;

      /* Diese Konstante stammt aus climits.h und bezeichnet den groessten
         Wert, den ein unsigned int annehmen kann.  Anders als die '0' bei
         der Suche nach dem maximalen Element ist dieser Wert aber ein
         zulaessiger Wert fuer einen Zaehler.  Also wird unten mit '<='
         verglichen. */
      minimalerWert = UINT_MAX;

      /// In Schleife ueber dasHistogramm den minimalen Wert suchen.
      cursor = dasHistogramm;
      while(NULL != cursor) {
         /* Falls aktueller Eintrag kleiner, merken. */
         if (cursor->wert <= minimalerWert) {
            minimalerEintrag = cursor;
            minimalerWert = cursor->wert;
         }
         cursor = cursor->naechster;
      }
   }
}

/* Allgemeine Hilfsfunktion: Haengt den neuen Eintrag an den Anfang der Liste.*/
void fuegeVorneEin(HistogrammT &dasHistogramm,
                   HistogrammEintragT * neuerEintrag)
{
   /* Nachfolgerzeiger des neuen Eintrages auf den Anfang der Liste
      setzen. */
   neuerEintrag->naechster = dasHistogramm;
   /* Den Zeiger auf den Anfang der Liste auf den neuen Eintrag setzen. */
   dasHistogramm = neuerEintrag;
}


/* Allgemeine Hilfsfunktion: Entferne einen Eintrag aus der Liste.*/
void entferne(HistogrammT &dasHistogramm,
              HistogrammEintragT const * opferEintrag)
```

```
{
   if (opferEintrag == NULL) {
      cerr << "Histogram::entferne: opferEintrag == NULL" << endl;
      return;
   } else if (opferEintrag == dasHistogramm) {
      /* Sonderfall: der opferEintrag ist das erste Element. */
      dasHistogramm = opferEintrag->naechster;
   } else {
      /* Normalfall: opferEintrag ist hinter dem ersten Element.  Also suche
         seinen Vorgaenger, um den Nachfolgerzeiger umzusetzen. */
      HistogrammEintragT * vorgaengerCursor = dasHistogramm;
      HistogrammEintragT * cursor           = dasHistogramm->naechster;
      bool gefunden                         = false;
      while (cursor != NULL && !gefunden) { // normale Suchschleife
         if (cursor == opferEintrag) {
            gefunden = true;
         } else {
       /* etwas aufwendigere Cursorverwaltung */
            vorgaengerCursor = cursor; cursor = cursor->naechster;
         }
      }
      /* Falls gefunden, ausreihen.  Dies wird dadurch erreicht, dass der
         opferEintrag in der Verweiskette uebersprungen wird. */
      if (gefunden) { vorgaengerCursor->naechster = cursor->naechster; }
                else {cerr<<"Zu entfernender Eintrag war nicht in Liste!"<<endl;}
   }
}
/* Sortiert die Eintraege im Histogramm. */
void sortiereHistogramm(HistogrammT &dasHistogramm)
{
   /* Sortiert wird nach dem Prinzip von 'StraightSelection'.  Auf Listen
      ist dies recht passend, weil das Verschieben wenig kostet und die
      Zugriffe alle in der passenden Reihenfolge stattfinden.  */
   /* Um die Liste *absteigend* zu sortieren, wird immer das *kleinste*
      Element *vorne* eingehaengt.  Dieses Ergebnis wird in AnfangSortiert
      aufgebaut. */
   HistogrammEintragT * anfangSortiert = NULL;
   /* Dieser Zeiger zeigt auf den Anfang der noch zu sortierenden Liste. */
   HistogrammEintragT * anfangUnsortiert = dasHistogramm;
   /* Zwischenspeicher: das minimale Element.  */
   HistogrammEintragT * minimalerEintrag;
   /* Schleife solange noch nicht vollstaendig sortiert, also noch Elemente
      in anfangUnsortiert. */
   while(NULL != anfangUnsortiert) {
      /// Den minimalen Eintrag aus der unsortierten Liste suchen.
      sucheMinimalenEintrag(anfangUnsortiert, minimalerEintrag);
      /* Den Minimalen Eintrag vor die sortierte Liste verschieben */
      entferne(anfangUnsortiert, minimalerEintrag);
      fuegeVorneEin(anfangSortiert, minimalerEintrag);
   }
   /* Schlussendlich das Ergebnis uebertragen */
   dasHistogramm = anfangSortiert;
}
```

15. Aufgabe

(a) Die verschiedenen Abschnitte des Feldes $iA \ldots mitte$ und $mitte + 1 \ldots iE$ sind bereits sortiert.

```
#include <iostream>
#include <string>

using namespace std;

#define MAX_FELD 20
typedef int FeldT[MAX_FELD];
typedef unsigned int IndexT;
/* Mischt das uebergebene Feld, so dass es innerhalb der Grenzen sortiert ist. */
void merge(FeldT &feld, IndexT iA, IndexT mitte, IndexT iE) {
    FeldT tmpFeld;
    IndexT first1 = iA;
    IndexT last1  = mitte;
    IndexT first2 = mitte+1;
```

```
    IndexT last2  = iE;
    IndexT index  = first1;

    /* Ordne die Werte der beiden Teile im Feld in das temporäre. */
    while ((first1 <= last1) && (first2 <= last2)) {
      if(feld[first1] < feld[first2]) {
         tmpFeld[index++] = feld[first1++];
      } else {
         tmpFeld[index++] = feld[first2++];
      }
    }
    /* Kopiere den Rest der 1. Haelfte. */
    while (first1 <= last1) tmpFeld[index++] = feld[first1++];

    /* Kopiere den Rest der 2. Haelfte. */
    while (first2 <= last2) tmpFeld[index++] = feld[first2++];

    /* Kopiere die Werte des temporären Felds zurück in das übergebene. */
    for(index = iA; index <= iE; index++)
      feld[index] = tmpFeld[index];
}
/* Sortiert das uebergebene Feld durch Mischen. */
void mergesort(FeldT &feld, IndexT iA, IndexT iE) {
    IndexT mitte;
    if(iA < iE) {
      mitte = (iA + iE) / 2;
      mergesort(feld, iA, mitte);      // Sortiere die erste Hälfte.
      mergesort(feld, mitte + 1, iE);         // Sortiere die zweite Hälfte.
      merge(feld, iA, mitte, iE);             // Mische die beiden Hälften.
    }
}
/* Sortiert das uebergebene Feld. */
void sortiere(FeldT &feld) {
    mergesort(feld, 0, MAX_FELD - 1);
}
```

(b) Der zusätzliche Speicheraufwand ist genauso groß wie die Größe des zu sortierenden Feldes ($O(n)$).

(c) Der Algorithmus greift in jeder Phase nur sequentiell auf das Feld zu.

16. Aufgabe

(a) Der Flußgraph zur Funktion `ggT` ist in Abb. 16-L.1 zu sehen. Für den White-Box-Test werden sieben Programmpfade unterschieden, gemäß Abb. 16-L.2. Die folgenden Fälle führen zu Problemen:

Problem	Bedingung	Beschreibung
Stagnation	`a == 0` *oder* `b == 0`	Der Wert, der ungleich 0 ist, wird nie verändert. Damit gleichen die Zahlen sich nie an ⇒ Endlosschleife.
Wachstum	`a > 0, b < 0` $\left(\begin{array}{c}\text{oder}\\ \text{andersrum}\end{array}\right)$	Der kleinere (negative) Wert wird vom größeren (positven) abgezogen, welcher dadurch betragsmäßig wächst, wieder der größere ist und so weiter. Das kann zu einer Endlosschleife oder falschen Ergebnissen (nach einem Overflow) führen, je nach Umgebung.
Umkippen	`a < 0 &&` `b < 0 &&` `a != b`	Der kleinere (alse betragsmäßig größere) Wert wird vom größeren abgezogen, wodurch dieser in nächsten Durchlauf größer als 0 ist und der Fall „Wachstum“ eintritt.

(b) Die Unterscheidung zwischen Extremfällen und Normalfällen ist hier fast nur willkürlich zu treffen, da alle Eingaben gültig sind. Lediglich eine Null als Eingabe kann wirklich zu Problemen führen. Wir wählen daher die folgenden Klassen von Eingabeparametern:

- Eingabe 0, Testfälle: (0, 0), (0, 5), (5, 0)

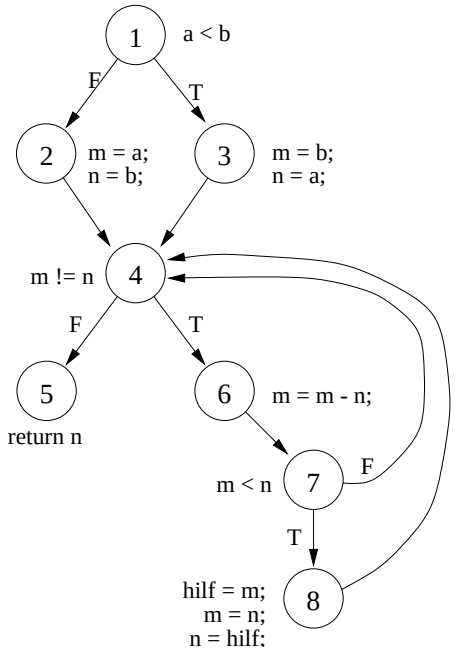

Abbildung 16-L.1: Flußgraph der Funktion ggT

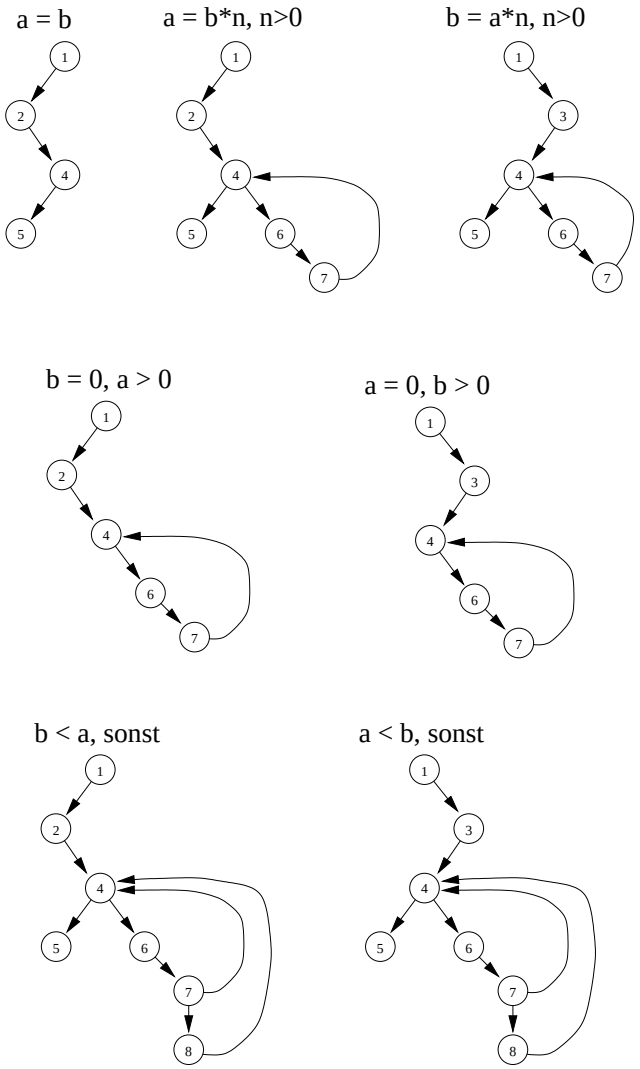

Abbildung 16-L.2: Programmpfade des White-Box-Tests

- Eingabe 1, Testfälle: (1, 5), (5, 1)
- $kgV(a, b) = ab$, Testfälle (5, 7), (7, 5)
- $a \mod b = 0$ oder $b \mod a = 0$, Testfälle (6, 2), (2, 6)
- $a, b < kgV(a, b) < ba$, Testfälle (18, 4), (4, 18)
- Bereichsüberschreitung, Testfälle (MAXINT / 5 + 1, 5), (5, MAXINT / 5 + 1) (vorausgesetzt, MAXINT / 5 + 1 ist nicht durch 5 teilbar)