Fakultät IV - Elektrotechnik und Informatik

Einführung in die Programmierung WS 2017/18

Feldmann / Foucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 3

letzte Aktualisierung: 04. Dezember, 11:21 Uhr

(dae868dc0f5c9ce3f350645f2079c7b73b07a261)

Ausgabe: Freitag, 24.11.2017
Abgabe: spätestens Mittwoch, 6.12.2017, 18:00

Thema: *Laufzeitbestimmung, Insertionsort, Countsort*

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
- Für die Hausaufgaben im Semester gibt es ein eigenes Subversion-Repository (SVN). Bitte die Hinweise auf Blatt 1 dazu beachten!
- Die Abgaben für Blatt 1 bis 4 erfolgen als Einzelabgaben. Ab Blatt 5 erfolgen Abgaben in festen Gruppen à 2 Personen.
- Die Gruppen werden vor Ausgabe von Blatt 5 gebildet. Solltet ihr keine Gruppe gebildet haben, werdet ihr einer Gruppe zugewiesen.
- Alle Abgaben müssen an der dafür vorgesehenen Stelle die Namen aller Gruppenmitglieder bzw. bei Einzelabgaben des Autors/der Autorin enthalten!
- Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Abgaben/Blatt<xx>/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`

  WICHTIG: Die Abgabeordner werden immer von uns erstellt, sonst kommt es zu Konflikten! Benutzt zum Aktualisieren `svn up` im obersten Verzeichnis des Repositories!
- Benutzt für alle Abgaben die in der jeweiligen Aufgabe angegebenen Dateinamen (die von uns vorgegebenen Dateien haben das Wort "vorgabe" im Dateinamen, du musst die Datei mit deiner Lösung also entsprechend umbenennen.)
- Gib bei den Programmieraufgaben den C-Quellcode ab und achte darauf, dass die Datei mit .c endet
- Bei Textaufgaben sind, wenn nicht anders angegeben, .txt Dateien zugelassen, die als Plaintext-Datei zu speichern sind. (Keine Word-Dateien umbenennen, etc.!)

1. Aufgabe: Laufzeitanalyse — Vergleich Count- und Insertionsort (2 Punkte)

1.1. Vergleich Insertion- und Countsort (1 Punkte) In dieser Aufgabe sollst Du (empirisch) die Laufzeit Deiner Insertion- und Countsort-Implementierungen vergleichen. Ähnlich zu Aufgabe 2 vom Blatt 2 erhälst Du eine Vorgabe, in welcher die komplette `main`-Funktion bereits vorgegeben ist. Du musst nur noch Deine Insertion- sowie Countsort Implementierungen (z.B. von Aufgabenblatt 1) einfügen und leicht anpassen. Du musst insgesamt vier verschiedene Funktionen schreiben bzw. anpassen:

1. `void count_sort_calculate_counts(int input_array[], int len, int count_array[], int* befehle)`
2. `void count_sort_write_output_array(int output_array[], int len, int count_array[], int* befehle)`
3. `void count_sort(int array[], int len, int* befehle)`
4. `void insertion_sort(int array[], int len, int* befehle)`

Hierbei sollen die Funktionen 1., 2., und 4. die gleiche Funktionalität wie auf dem Aufgabenblatt 1 haben, jedoch noch zusätzlich die ausgeführten Befehle mittels `int* befehle` zählen. Beachte zur Bestimmung der ausgeführten Befehle die Erläuterung aus Aufgabe 1 vom Blatt 2, sowie die Hinweise von Aufgabe 2, auch vom Blatt 2.

Die Funktion `count_sort` hat die gleiche Signatur, d.h. die gleichen Parameter und den gleichen Rückgabewert, wie die Funktion `insertion_sort`. Die Funktion `count_sort` soll hierbei die gesamte Funktionalität des Countsort-Algorithmus kapseln:

1. Erstelle zunächst mittels `malloc` ein Array zum Zählen der Häufigkeiten verschiedener Werte.
2. Rufe die Unterfunktionen `count_sort_calculate_counts` sowie `count_sort_write_output_array` so auf, dass das Ergebnis von Countsort (d.h. der Funktion `count_sort`) in das Eingabearray `int array[]` geschrieben wird.
3. Vergiss nicht, den allozierten Speicher wieder frei zu geben.
4. Die Anzahl ausgeführter Befehle von Countsort erstreckt sich über insgesamt 3 Funktionen und muss dementsprechend auch zusammengezählt werden.

Die Vorgabe (siehe Listing 1) erzeugt für Count- und Insertionsort jeweils ein Array mittels `malloc`: `array_countsort` sowie `array_insertionsort`. Das Array `array_countsort` wird zunächst mittels des Funktionsaufrufs `fill_array_randomly(array_countsort, n, MAX_VALUE);` mit Zufallswerten beschrieben. Anschließend werden mittels `copy_array_elements(array_insertionsort, array_countsort, n);` die gleichen Werte auch in das Array `array_insertionsort` geschrieben.

Aufgabe und Abgabemodalitäten:

- Implementiere die vier Funktionen und werte die Laufzeit sowie die Anzahl der benötigten Befehle beider Algorithmen aus.
- Teste die Algorithmen für verschiedene Werte der Konstanten `MAX_VALUE`. Finde Kombinationen aus `MAX_VALUE` und der Größe des Arrays n, bei denen jeweils entweder Insertionsort oder Countsort vorzuziehen ist.

Check Deinen Code als `introprog_complexity_steps_sorting.c` im (im Ordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`) ein.

Hinweise:

- Beachte, dass Du zum erfolgreichen Kompilieren des Programms zusätzlich die Quelldatei `introprog_complexity_steps_input.c` im Aufruf von `gcc` übergeben musst.
- Beachte, dass Du den Code der `main`-Funktion weder anpassen musst noch anpassen sollst. Natürlich steht es Dir offen, die Werte des Arrays `int WERTE[]` oder die Konstante `MAX_VALUE` anzupassen. Dies kann z.B. nützlich sein, falls die Ausführung des Programms zu lange dauert.
- Du musst die main-Funktion keinerlei editieren.
- Bitte kommentiere Deinen Code genügend, um Klarkheit zu schaffen. Jedoch nicht zu viel (bspw. in jeder einzelnen Zeile), denn das würde Deinen Code wiederum unübersichtlicher machen.

Listing 1: Vorgabe `introprog_complexity_steps_sorting_vorgabe.c`

```
1  /* === INTROPROG ABGABE ===
2   * Blatt 3, Aufgabe 1
3   * Tutorium: tXX
4   * Abgabe von: Erika Mustermann
5   * ========================
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include "introprog_complexity_steps_input.h"
11
12 const int MAX_VALUE = 5000000;
13
14 void count_sort_calculate_counts(int input_array[], int len, int
      ↪ count_array[], int* befehle) {
15     //muss implementiert werden
16 }
17
18 void count_sort_write_output_array(int output_array[], int len, int
      ↪ count_array[], int* befehle) {
19     //muss implementiert werden
20 }
21
22 void count_sort(int array[], int len, int* befehle) {
23     //muss implementiert werden
24 }
25
26
27 void insertion_sort(int array[], int len, int* befehle) {
28     //muss implementiert werden
29 }
30
31
32 int main(int argc, char *argv[]) {
33
34     const int WERTE[] = {10000,20000,30000,40000,50000};
35     const int LEN_WERTE = 5;
36     const int LEN_ALGORITHMEN = 2;
37
```



```
38     int rc = 0;
39     long befehle_array[LEN_ALGORITHMEN][LEN_WERTE];
40     double laufzeit_array[LEN_ALGORITHMEN][LEN_WERTE];
41
42     for(int j = 0; j < LEN_WERTE; ++j)
43     {
44         int n = WERTE[j];
45
46         //reserviere Speicher für  Arrays der Länge n
47         int* array_countsort = malloc(sizeof(int) * n);
48         int* array_insertionsort = malloc(sizeof(int) * n);
49
50         //fülle array_countsort mit Zufallswerten ..
51         fill_array_randomly(array_countsort, n, MAX_VALUE);
52         //.. und kopiere die erzeugten Werte in das Array
              ↪ array_insertionsort
53         copy_array_elements(array_insertionsort, array_countsort, n);
54
55         //teste ob beide Arrays auch wirklich die gleichen Werte
              ↪ enthalten
56         if(!check_equality_of_arrays(array_countsort, array_insertionsort
              ↪ , n))
57         {
58             printf("Die␣Eingaben␣für␣beide␣Algorithmen␣müssen␣für␣die␣
                  ↪ Vergleichbarkeit␣gleich␣sein!\n");
59             return -1;
60         }
61
62         for(int i = 0; i < LEN_ALGORITHMEN; ++i)
63         {
64             int anzahl_befehle = 0;
65
66             start_timer();
67
68             //Aufruf der entsprechenden Sortieralgorithmen
69             if(i==0)
70             {
71                     count_sort(array_countsort, n, &anzahl_befehle);
72             }
73             else if(i==1)
74             {
75                     insertion_sort(array_insertionsort, n, &
                          ↪ anzahl_befehle);
76             }
77
78             //speichere die Laufzeit sowie die Anzahl benötigter Befehle
79             laufzeit_array[i][j] = end_timer();
80             befehle_array[i][j] = anzahl_befehle;
81         }
82
83         //teste ob die Ausgabe beider Algorithmen gleich ist
84         if(!check_equality_of_arrays(array_countsort, array_insertionsort
              ↪ , n))
85         {
```

```
86              printf("Die␣Arrays␣sind␣nicht␣gleich.␣Eines␣muss␣(falsch)␣
                    ↪ sortiert␣worden␣sein!\n");
87              rc = -1;
88          }
89
90          //gib den Speicherplatz wieder frei
91          free(array_countsort);
92          free(array_insertionsort);
93      }
94
95      //Ausgabe der Anzahl ausgeführter Befehle sowie der gemessenen
            ↪ Laufzeiten (in Millisekunden)
96      printf("Parameter␣MAX_VALUE␣hat␣den␣Wert␣%d\n", MAX_VALUE);
97      printf("\t␣%32s␣␣␣␣␣␣␣␣␣␣␣%32s␣\n", "Countsort","Insertionsort");
98      printf("%8s␣\t␣%16s␣%16s␣\t␣%16s␣%16s␣\n", "n","Befehle", "Laufzeit",
            ↪ "Befehle","Laufzeit");
99
100     for(int j = 0; j < LEN_WERTE; ++j)
101     {
102         printf("%8d␣\t␣",WERTE[j]);
103         for(int i = 0; i < LEN_ALGORITHMEN; ++i)
104         {
105             printf("%16ld␣%16.4f␣\t␣",  befehle_array[i][j],
                    ↪ laufzeit_array[i][j]);
106         }
107         printf("\n");
108     }
109
110     return rc;
111 }
```


1.2. Fragen: Laufzeitvergleich (1 Punkt) Im SVN findest du unter `Aufgaben/Blatt03/Vorgaben/introprog_complexity_steps_sorting_vorgabe.txt` eine ausfüllbare Textdatei mit den folgenden Fragen:

- Inwieweit stehen die Anzahl der gezählten Befehle und die (empirisch gemessenen) Laufzeiten in Beziehung zu einander?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n$ ist. Somit wächst die Größe des Wertebereichs linear mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n^2$ ist. Somit wächst die Größe des Wertebereichs quadratisch mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $k = n^3$ ist. Somit wächst die Größe des Wertebereichs kubisch mit der Größe der Eingabe. Welcher der beiden Algorithmen (Insertionsort bzw. Countsort) weist für große n im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ und $k = 100$ ist. Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ ist und die Größe des Wertebereichs zur Entwicklungszeit nicht bekannt ist. k kann somit beliebig viel größer als n sein. Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?
- Sei n die Eingabgelänge des Arrays und k die Größe des Wertebereichs (wie in der Vorlesung). Es gelte, dass $n = 100.000$ ist und die Größe des Wertebereichs bei $k = 100.000.000$ liegt. Wir nehmen an, dass die Eingabe schon maßgeblich aufsteigend sortiert ist. Konkret gelte $A[i] < A[i + j]$ für alle natürlichen Zahlen i in $\{1, 2, 3, \ldots, 999.998\}$ und alle j in $\{2, 3, 4, \ldots, n - i\}$ (unter Verwendung der Pseudocode-Konvention, dass Arrays bei 1 beginnen). Welcher der beiden Algorithmen weist im Worst Case unter diesen Annahmen eine bessere Laufzeit auf?

Aufgabe und Abgabemodalitäten:
Kreuze in der Datei (unter Beachtung von sämtlichen Hinweisen und Vorgaben) die korrekte Antwort an und checke die modifizierte Datei im Abgabeordner für dieses Blatt als `introprog_complexity_steps_sorting.txt` im SVN ein.