Algorithmen und Datenstrukturen: Lösungsbuch Aufgabe 1

Frederik Dudzik

January 23, 2015

1 Implementation

1.1 Main

In Main werden die Daten initialisiert und das Programm immer wieder von vorne ausgeführt bis der Nutzer das Programm schließt. Die Eingabe der Befehle erfolgt hier, jedoch werden die Befehle in einer anderen Prozedur interpretiert und ausgeführt. Das Ziel dahinter ist, dass das Programm einfacher zu verstehen ist, da wir die Verantwortung teilen und hinter weniger abstrakten Prozeduren bündeln.

1.2 CLI

Die CLI Prozeduren dienen als ein Adapter zwischen der eigentlichen Logik des Programms und der Außenwelt. Über die Funktion `cli_function` werden die Befehle des Nutzers Interpretiert und die jeweilige adapter Funktion in cli.c aufgerufen. Diese Funktionen gehen sicher dass die fur den gegebenen Befehls notwendigen Prozeduren ausgeführt werden. Und das diese mit korrekten Daten verwendet werden.

1.3 util

In der util Datei sind hilfreiche Funktionen gespeichert die in keine der anderen Dateien platz finden und auch keiner eigenen Datei bedürfen. Wenn dies ein größeres Projekt wäre hätte ich die Funktionen in eigene Dateien ausgelagert. Da sonst solch eine util Datei ein schwarzes Funktions-Loch fur Entwickler werden würde.

1.4 student

In student.c sind alle Funktionen implementiert die die Student Struktur betreffen. Das interessanteste in der Datei ist wie ich das Subject Enum definiere. Wir brauchen in dem Code die Subjects einmal als Enum und als String. Dafür

benutzte ich den Preprozessor damit ich die Subjects nicht zweimal hinschreiben muss und damit sie in der gleichen Reihenfolge sind.

1.5 `cli_student`

In `cli_student` sind adapter Prozeduren für Student. Siehe die Sektion CLI.

1.6 `student_list`, doppelt und einfach

Das Interface für die Doppelt wie einfach Verkettete Liste ist bei mir die selbe, weswegen sie sich eine header Datei teilen. Diese wird von den anderen Dateien included. Damit wir angeben welche Listenform wir verwenden wollen müssen wir beim Kompilieren oder im Code `DOUBLE_LINKED_LIST` definieren oder undefiniert lassen. Daraufhin wird die jeweilige Listenstruktur und implementation von `student_list.h` kompiliert.

1.7 Makefile

Das Makefile ist weitestgehend aus den Unterlagen übernommen. Ich habe dieses lediglich fur meine Bedürfnisse erweitert.

Wichtig ist das sie mit 'make' die einfache `student_list` kompilieren und mit ´´`make double`´ die doppelte.

1.8 TODO

Das Pattern das in Main und CLI beschrieben wurde sollte strikter durchgezogen werden. Z.b. sollte in den Prozeduren von `cli_student` nicht direkt einen Studenten erzeugt werde, die cli Funktionen sollte das erledigen. Zudem sollte die Liste als eine generische Implementiert werden, falls das Programm geschrieben werden würde. Ausserdem sollten man mehr mit Status Codes arbeiten. In der CLI wird versucht das zu implementieren. Dies ist jedoch eher dürftig umgesetzt worden.

2 Aufgabe 3

2.1 b: zum Hinzufügen eines Elements vor dem ersten Element

Wir müssen keine Verbindung zu vorherigen Element setzten. Dadurch sparen wir 6 Operationen.

2.2 a: Erstellen eines neuen Elements

Wir müssen keine Verbindung zu vorherigen Element setzten. Dadurch sparen wir 4 Operationen.

2.3 g: Löschen eines Elements.

Wir müssen keine Verbindung zu vorherigen Element setzten. Dadurch sparen wir 2 Operationen.

2.4 c: Hinzufügen eines Elements nach dem letzten Element

Wir müssen die Liste einmal durchlaufen um ein Element am Ende hinzuzufügen O(n). Bei einer Doppeltverketteten Liste ist es O(1), da wir über den end Listenkopf direkt darauf zugreifen können.

3 Aufgabe 5

3.1 Meine Implementierung

```
void swapStudents(StudentList* prev, StudentList* next)
{
   Student* tmp = prev->student;  //2
   prev->student = next->student; //3
   next->student = tmp;           //2
}
```

O(7)

```
//selectionsort
void sortMatrikel(StudentList* sl)
{
   StudentList* j   = 0L;                                  //1
   StudentList* min = 0L;                                  //1
   StudentList* i   = sl;                                  //1
   while(i->next->next != 0L) {                            //3          -------|
      i   =i->next;                                        //1                 |
      min = i;                                             //1                 |
      j   = i;                                             //1                 |
      while (j->student != 0L){                            //2    ----|        |
         if (j->student->id < min->student->id) min = j;   //4/w/5   |        |
         j = j->next;                                      //2        |n       | n
      }                                                    //       ----|       |
      if(min != i) swapStudents(min, i);                   //7                 |
   }                                                       //          -------|
}
```

$1 + 1 + 1 + n * (3 + 1 + 1 + 1 + n * (2 + 5 + 2 + 7)) \Rightarrow n * n = O(n^2)$
$1 + 1 + 1 + n * (3 + 1 + 1 + 1 + n * (2 + 4 + 2 + 7)) \Rightarrow n * n = \Omega(n^2)$

```
//bubblesort                                                          
void sortStudiengang(StudentList* sl)                                 
{                                                                     
   StudentList* i = sl;                                               //1
   int swapped = false;                                               //1
   do {                                                               /w/ ----------------|
      swapped = false;                                                //1                 |
      while (i->next->next->student != 0L) {                          //4    -------|     |
         i=i->next;                                                   //2           |     |
         if (i->student->subject > i->next->student->subject) {       //6           |     |
            swapStudents(i, i->next);                                 /w/8        | n   | n
            swapped = true;                                           /w/9        |     |
         }                                                            //          |     |
      }                                                               //     -------|     |
      i = sl;                                                         //1                 |
   } while (swapped);                                                 //     --------------|
}                                                                     
```

$1 + 1 + n * (1 + 4 + n * (2 + 6 + 8 + 9) + 1 \Rightarrow n * n = O(n^2)$
$1 + 1 + (1 + 4 + n * (2 + 6) + 1 \Rightarrow n = \Omega(n)$

3.2 Allgemein

Bubblesort ist aus "Algorithmen und Datenstrukturen" Auflage 5 von Thomas Ottmann und Peter Widmayer entnommen. Sellectionsort ist aus Donald Knuth's 'The Art of Computer Programming, Volume 3: Sorting and Searching'

```
//Bubblesort
procedure bubblesort(var a: sequence);
  var i: integer;                                  //  1
  begin
    repeat                                         /w/ ----------|
      for i:=1 to (N-1) do                         //     ---|    |
        if a[i].key > a[i+1].key                   //  6     | n  | n
          then {vertausche a[i] und a[i+1]}        /w/ 4 ---|    |
    until {keine Vertauschungen mehr auftreten}    //  -----------|
  end
```

$1 + n * (n * (6 + 4) \Rightarrow n * n = O(n^2)$
$1 + n * (6) \Rightarrow n = \Omega(n)$

```
//Selectionsort
int i,j;                          //2
int iMin;                         //1
for (j = 0; j < n-1; j++) {       //3  ---------|
   iMin = j;                      //1           |
   for ( i = j+1; i < n; i++) {   //3 -----|    |
      if (a[i] < a[iMin]) {       //1      |    |
         iMin = i;                /w/1   | n  | n
      }                           //       |    |
   }                              //  ------|    |
   if(iMin != j) {                //1           |
      swap(a[j], a[iMin]);        /w/3           |
   }                              //             |
}                                 //   ---------|
```

$2 + 1 + 3 + n * (1 + 3 + n * (1 + 1) + 1 + 3) \Rightarrow n * n = O(n^2)$
$12 + 1 + 3 + n * (1 + 3 + n * (1) + 1) \Rightarrow n = \Omega(n)$

3.3 Ergebnis

Die Komplexität von meiner Implementierung unterscheidet sich nicht von der Allgemeinen.

3.4 Begründung der Auswahl

Die Algorithmen können genauso in einer einfach wie doppelt verketteten Liste implementiert werden ohne einen Komplexität Verlust zu erleiden. Im Gegensatz zu Insertion sort zum Beispiel. Ich habe eine einfachere Implementation vorgezogen, anstatt einer schnelleren.