Abteilung für Angewandte Mathematik
Prof. Dr. S. Bartels
M.Sc. C. Palus

11.06.2018

Übung zur Vorlesung

Einführung in die Programmierung

SoSe 2018 – Blatt 8

Abgabe: Briefkästen RZ/E-Mail bis Montag, den 18.06.2018 um 16:00 Uhr

Aufgabe 1 (3+4+3 Punkte)**.** Der Aufwand eines Algorithmus zur Lösung einer Aufgabe der Größe n ist definiert als die Anzahl $\mathcal{A}(n)$ der erforderlichen arithmetischen Operationen und Vergleiche.
(i) Ordnen Sie die folgenden Schranken für den Aufwand eines Algorithmus in der Reihenfolge ihrer Günstigkeit für große Probleme der Größe n:

$$n^2, \quad 2^n, \quad n^3, \quad n!, \quad n, \quad 1, \quad \log(n).$$

Begründen Sie Ihre Antwort!
(ii) Bestimmen Sie den Aufwand jeweils für die dynamische und rekursive Berechnung der n-ten Fibonacci-Zahl f_n (vgl. Blatt 6, Aufgabe 4).
(iii) Eine mögliches Maß für die Leistungsfähigkeit eines Computers ist die Einheit *Floating Point Operations Per Second* (kurz *FLOPS*; engl. für Gleitkommaoperationen pro Sekunde). Es gibt an, wie viele Gleitkommaoperationen (d. h. Additionen und Multiplikationen) ein Rechner pro Sekunde ausführen kann. Beispielsweise erreicht ein Pentium 4 Prozessor mit 3,2 GHz aus dem Jahr 2004 eine Höchstleistung von $6,4$ GigaFLOPS, ein Core i7 4-Kern-Prozessor mit 3,4 GHz aus dem Jahr 2011 eine Höchstleistung von $92,3$ GigaFLOPS, und der Cray XC40 Supercomputer „Hazel Hen“ am Höchstleistungsrechenzentrum Stuttgart, der aktuell der schnellste Rechner Deutschlands ist, eine Höchstleitung von $5,6$ PetaFLOPS. Die Vorsilbe „Giga“ steht dabei für den Faktor 10^9 und die Vorsilbe „Peta“ für den Faktor 10^{15}. Berechnen Sie auf dieser Grundlage die Mindestzeit, die für die rekursive Berechnung der 90. Fibonacci-Zahl f_{90} auf jedem dieser Systeme benötigt wird.

Aufgabe 2 (10 Punkte)**.** Formulieren Sie einen rekursiven Algorithmus zur Lösung eines Sudokurätsels. Ausgangspunkt sei dazu ein gegebenes, teilweise ausgefülltes (9×9)-Gitter $(A_{i,j})$, welches nach den Sudokuregeln (vgl. Blatt 5, Aufgabe 2) vervollständigt werden soll.
Tipp: Die Rekursion sollte bzgl. der unausgefüllten Gitterzellen stattfinden, d. h. der Algorithmus sollte nacheinander alle gültigen Ziffern in ein freies Feld schreiben und mit den so entstehenden Sudokus jeweils wieder genauso verfahren, bis das Sudoku komplett ausgefüllt ist oder keine gültige Ziffer mehr gefunden werden kann. Eine gültige Ziffer ist dabei eine Ziffer, welche für das teilweise ausgefüllte Gitter keine der Sudokuregeln verletzt.

Aufgaben 3 und 4 auf der Rückseite

Aufgabe 3 (10 Punkte)**.** Schreiben Sie ein Programm, welches für ein fertig ausgefülltes Sudoku prüft, ob es sich dabei um eine gültige Lösung handelt. Benutzen Sie als Datentyp für das Sudoku-Gitter ein Integer-Array der Größe 9 mal 9, z. B. `int sudoku[9][9]`. Implementieren Sie die Prüfung in einer Unterfunktion `bool ist_gueltige_lsg( int gitter[9][9] )`, die als Parameter nur die Array-Variable übergeben bekommt und dann entsprechend der Gültigkeit entweder `true` oder `false` zurückgibt. Als Orientierung sollte der Algorithums aus Aufgabe 2, Blatt 5, dienen, den Sie hierzu in die Sprache C++ übersetzen müssen. Testen Sie Ihr Programm sowohl mit einer gültigen als auch mit einer ungültigen Lösung. Eine gültige Lösung ist z. B. durch die folgende Definition gegeben:

```
int sudoku_gueltig[9][9] = {
  {2,1,4,5,3,8,9,6,7},
  {7,8,6,9,2,1,3,4,5},
  {5,9,3,7,4,6,2,8,1},
  {8,4,2,3,9,7,1,5,6},
  {6,5,1,4,8,2,7,9,3},
  {3,7,9,1,6,5,4,2,8},
  {9,3,8,6,1,4,5,7,2},
  {1,2,7,8,5,9,6,3,4},
  {4,6,5,2,7,3,8,1,9}
};
```

Aufgabe 4 (10 Punkte)**.** Erweitern Sie Ihr Programm zur Verwaltung eines Telefonverzeichnisses aus Aufgabe 3 vom letzten Blatt, sodass der Benutzer zusätzlich die Möglichkeiten hat, die gesamte Telefonliste in eine Datei `kontakte.dat` zu speichern oder eine gespeicherte Telefonliste aus dieser Datei einzulesen. Die Ausgabe in die Datei kann dabei mit Hilfe des Operators „<<“ analog zur Ausgabe in die Konsole geschehen. Zu beachten ist, dass vorher ein passender *Output Stream*, etwa `std::ofstream f_out`, deklariert und die Datei zum Schreiben geöffnet wird, was in diesem Fall mittels `f_out.open("kontakte.dat")` geschieht. Das Gleiche gilt analog für die Eingabe, für die dann der Operator „>>“ benutzt werden kann. Möchte man auf diese Art eine ganze Datei bis zu deren Ende einlesen, so kann man dies zum Beispiel mit Hilfe einer `while`-Schleife und der *End-of-File*-Funktion `eof()` realisieren, welche `true` zurückgibt, sobald beim Einlesen das Ende der Datei erreicht wird:

```
std::ifstream f_in;
f_in.open("kontakte.dat");
while( !f_in.eof() ) {
  f_in >> passende_variable;
}
```

Bei dieser Vorgehensweise ist allerdings darauf zu achten, dass die Datei keinesfalls mit einer Leerzeile endet, da der Wahrheitswert von `eof()` sonst erst nach dem Erreichen dieser Leerzeile „wahr“ wird.

Abgabe der Programme per E-Mail, (handschriftlich) kommentierte Ausdrucke der Programme und Rechnungen auf gehefteten, mit Namen versehenen Zetteln in die Briefkästen

Homepage zur Vorlesung: `https://aam.uni-freiburg.de/agba/lehre/ss18/einfprog`