Aufgabensammlung zum Buch „Algorithmen und Datenstrukturen“ (Kapitel 3)

1 Aufgaben aus dem Buch

Zu folgenden Aufgaben, die direkt aus dem Buch entnommen sind, gibt es an der Universität Freiburg am Lehrstuhl Ottmann Musterlösungen. In der Version mit Lösungen sind diese angegeben. Hinter der fortlaufenden Aufgabennummer steht in Klammern die Nummer der Aufgabe im Buch.

Aufgabe 1 (Aufgabe 3.3):

Gegeben sei ein Feld mit der folgenden Schlüssel-Belegung: 1, 2, 4, 8, 16, 32, 64, 128. Man beschreibe die Suche nach dem Schlüssel 34 im obigen Feld durch Angabe der Folge der ausgeführten Schlüsselvergleiche, wenn als Suchstrategie exponentielle Suche zur Eingrenzung des Suchbereichs mit anschließender linearer Suche angewandt wird.

2 Ähnlich Aufgaben

Bei den folgenden Aufgaben handelt es sich um Aufgaben die an der ETH Zürich, am Institut für Theoretische Informatik und an der Universität Freiburg im Institut für Informatik in diversen Vorlesungen gestellt wurden. Inhaltlich sind diese Aufgaben mit dem behandelten Stoff im Buch verwandt. Zu allen Aufgaben gibt es Musterlösungen, die allerdings nur in der Version mit Lösungen enthalten sind.

Aufgabe 2:

Gegeben sei das Feld a mit der folgenden Schlüssel-Belegung: 0,2,6,8,12,34,75,120.

Beschreiben Sie die Suche nach dem Schlüssel 34 im obigen Feld a durch Angabe der Folge der ausgeführten Schlüsselvergleiche, wenn als Suchstrategie exponentielle Suche zur Eingrenzung des Suchbereichs mit anschließender linearer Suche angewandt wird.

Mit dieser neuen Erkenntnis implementieren Sie die Klasse `ExpList`, die diese Strategie verfolgt. Der Algorithmus soll angeben, in welcher Suchphase (exponentielle oder lineare) er gerade ist und alle Schluesselvergleiche auf der Konsole protokollieren.

Aufgabe 3:

Select(S, k) liefere die k-te Zahl aus der in der aufsteigender Ordnung sortierten Menge S mit $1 \leq k \leq n$.

Select(S, k)
  $x \leftarrow$ *Goodsplitter*$_\alpha(S)$
  $S_< \leftarrow \{y \in S |\ y < x\}$
  $S_> \leftarrow \{y \in S |\ y > x\}$
  cases
    $|S_<| \geq k$: *return Select*$(S_<, k)$
    $|S_<| = k - 1$: *return* x
    $|S_<| < k - 1$: *return Select*$(S_>, k - (|S_<| + 1))$

Bermerkung: Bei diesem Algorithmus (und dem in der nächsten Aufgabe) fehlt die Abbruchbedingung für die Rekursion.

Zeigen Sie, daß falls *Goodsplitter*$_\alpha$ höchstens en Vergleiche braucht (e eine Konstante), dann braucht *Select* höchstens $(e+1) \cdot n/(1-\alpha)$ viele Vergleiche. Hinweis: Rollen Sie die Rekursionsgleichung $S(n) \leq en + n + S(\alpha n)$ auf.

Aufgabe 4:

Innerhalb eines zufällig mit Ganzzahlelementen (Schlüsseln) `a[0]`,...,`a[n-1]` gefüllten Arrays `a` soll das minimale und maximale Element gefunden werden. Schreiben Sie ein Java Programm `MinMax`, das zur gleichzeitigen Suche beider Elemente möglichst wenige (Schlüssel-) Vergleiche durchführt. Ihr Programm sollte die triviale Anzahl von $2(n-1)$ Vergleichen wesentlich unterbieten. Nutzen Sie eine Variable `call`, die protokolliert, wie häufig ein Schlüsselvergleich `less` bzw. `greater` durchgeführt wurde.

1. Schreiben Sie das Programm ohne Divide-and-Conquer.
2. Schreiben Sie das Programm mit Divide-and-Conquer.

Zur Vereinfachung können Sie sich auf $n = 2^k$ für ein ganzzahliges k beschränken.

Aufgabe 5:

Wir analysieren das MIN-MAX-Problem. Innerhalb eines zufällig mit den n Ganzzahlelementen $a[0], \ldots a[n-1]$ gefüllten Arrays a soll das minimale und maximale Element gefunden werden. Sie können n als gerade voraussetzen.

1. Zeigen Sie, daß $n-1$ Vergleiche notwendig sind, um eines der beiden Elemente zu finden und ein solches Verfahren existiert.
2. Zeigen sie, daß ein einfaches Scan-Verfahren mit den Variablen `minsofar` und `maxsofar` existiert, das höchstens $3n/2 - 2$ Vergleiche benötigt.
3. Zeigen Sie, daß mindestens $3n/2 - 2$ Vergleiche zur Lösung des Problems notwendig sind. Dabei beginnen Sie wie folgt:

   "Sei ein korrektes Verfahren V zur Lösung des MIN-MAX-Problems gegeben. Seien S_{MIN} (bzw. S_{MAX}) die Mengen derjenigen Elemente, die in einem Zustand von V sicher größer (kleiner) sind als das Minimum (Maximum) aller.

   Initial ist $S = \mid S_{MIN} \mid + \mid S_{MAX} \mid = 0$ und zum Ende von V ist $S = 2n - 2$. Bezeichne mit $comp(a,b) = (min\{a,b\}, max\{a,b\})$ die Vergleichsoperation zweier Werte a, b. Eine Anwendung von $comp(a,b)$ kann die Mengen S_{MIN} und S_{MAX} vergrößern."

   Geben Sie eine Fallunterscheidung der Vergleichsmöglichkeiten an, und der Gewinn an Information, der sich als Summand von S ergibt.

Aufgabe 6:

Um die verschiedenen Selbstanordnungsstrategien besser bewerten zu können, vergleichen Sie das Verhalten der MF-, der FC, und der T-Regel für eine Zugriffsfolge s, indem sie für die folgende zu statistischen Zwecken aus gelegte Tabellenklasse die fehlenden Klassen `MFList`, `FCList`, und `TList` implementieren.

```
public class Tabelle {
```

```
  MFList  MF; TList T; FCListe FC;
  int[] folge;
  int totalMF, totalT, totalFC;
  public Tabelle(int [] list, int[] zugriffsfolge) {
    MF = new MFList(list);
      int[] listT=new int[list.length];
      for (int i=0;i<list.length;i++){ listT[i] = list[i];}
    T = new TList(listT);
      int[] listFC=new int[list.length];
      for (int i=0;i<list.length;i++){ listFC[i] = list[i];}
    FC = new FCList(dummy2);
    folge = zugriffsfolge;
    totalMF = totalT = totalFC = 0;
  }
  public void starteVergleich(){
    System.out.println("MFListe: "+MF+" | TListe: "+T
                       +" | FCListe: "+FC);
    for (int i=0;i<folge.length;i++){
      System.out.print("Zugriff auf Schluessel "+folge[i]+": ");
      int cnt = MF.access(folge[i]); totalMF += cnt;
      System.out.print("MF: "+MF+" Cost:"+cnt);
      cnt = T.access(folge[i]);      totalT += cnt;
      System.out.println(" T: "+T+" Cost:"+cnt);
      cnt = FC.access(folge[i]);     totalFC += cnt;
      System.out.println("FC: "+FC+" Cost:"+cnt);
    }
    System.out.println("---------------------------------
                        ---------------------------");
    System.out.println("Gesamtkosten: MF "+totalMF+" ,
                       T "+totalT+" , FC"+totalFC);  }
}
```

Testen Sie ihr Programm mit der Liste $L = 1, 2, 3, 4, 5, 6, 7$ und die Zugriffsfolge s mit 21 Zugriffen: $7, 2, 7, 3, 3, 7, 4, 4, 4, 7, 5, 5, 5, 5, 7, 6, 6, 6, 6, 6, 7$ wie folgt

```
public class Test {
  public static void main(String[] args) {
    Tabelle t =
      new Tabelle(new int [] {1,2,3,4,5,6,7},
                  new int [] {7,2,7,3,3,7,4,4,4,7,5,
                              5,5,5,7,6,6,6,6,6,7});
    t.starteVergleich(); }
}
```

Aufgabe 7:

Innerhalb eines zufällig mit den n verschiedenen Zahlen gefüllten Arrays A soll das minimale und das maximale Element gefunden werden. Sie können $n \geq 2$ als Zweierpotenz voraussetzen.

1. Beschreiben Sie ein Scan-Verfahren mit den Variablen `minsofar` und `maxsofar`, das $3n/2 - 2$ Schlüsselvergleiche benötigt.

2. Beschreiben Sie ein Divide-and-Conquer-Verfahren, das $3n/2 - 2$ Schlüsselvergleiche benötigt. Nutzen Sie zur Begründung die Hypothese $3n/2 - 2$ für einen Induktionsbeweis.

Aufgabe 8:

Gegeben sei das sortierte Array A mit der folgenden Schlüsselbelegung:

1	2	3	4	5	6	7	8
2	3	9	13	23	33	72	113

Man beschreibe die Suche nach dem Schlüssel 34 in A durch Angabe der Folge der ausgeführten Schlüsselvergleiche. Als Suchstratgien werden angewandt:

1. exponentielle Suche zur Eingrenzung des Suchbereichs mit anschließender linearer Suche. Beginnen Sie dabei am Anfang des zu durchsuchenden Bereichs.
2. binäre Suche