Reihungen

Martin Wirsing

in Zusammenarbeit mit
Matthias Hölzl, Piotr Kosiuczenko, Dirk Pattinson

10/04/03

Ziele

- Die Datenstruktur der Reihungen verstehen: mathematisch, als Objekte und im Speicher
- Grundlegende Algorithmen auf Reihungen kennenlernen: Suche im untergeordneten und geordneten Feld
- Eindimensionale und mehrdimensionale Reihungen verstehen

Reihungen

- Eine Reihung (auch Feld, Array genannt) ist ein Tupel von Komponentengliedern, auf die über einen Index direkt zugegriffen werden kann.
- Mathematisch kann eine Reihung mit n Komponenten von Typ `type` als endliche Abbildung

$$I_n \longrightarrow \texttt{type}$$

mit Indexbereich I_n = {0, 1,..., n – 1}beschrieben. n ist die Länge des Feldes.

- Da `type` ein beliebiger Typ ist, kann man auch Felder als Komponenten haben $\Rightarrow$ mehrdimensionale Felder.

Reihungen und deren Speicherdarstellung

Beispiel

Ein Feld a der Länge 6 kann folgendermaßen dargestellt werden:

a:	´V´	´E´	´R´	´L´	´A´	´G´
Index:	0	1	2	3	4	5

a kann beschrieben werden als die Abbildung

$$\texttt{a} : \{0,\ldots,5\} \longrightarrow \textbf{char}$$

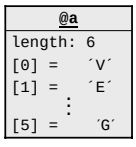
$$\texttt{a}[i] = \begin{cases} \texttt{´V´} & \text{falls } i = 0 \\ \texttt{´E´} & \text{falls } i = 1 \\ \vdots & \\ \texttt{´G´} & \text{falls } i = 5 \end{cases}$$

Reihungen und deren Speicherdarstellung

In **Java** wird ein Feld mit n Elementen aufgefaßt als ein Objekt mit den `n + 1` Attributen

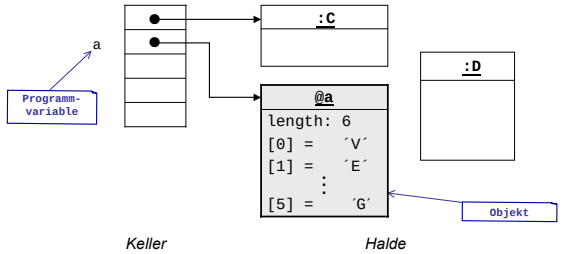
```
int  length
type  0
  :
type      (n - 1)
```

in **UML**



Reihungen und deren Speicherdarstellung

Die **Speicherorganisation** von a hat folgende Gestalt



Deklaration von Reihungstypen und -variablen

In Java haben **Feldtypen** die Form

```
TypeName[]... []
```

Beispiel:

```
int[], int[][], boolean[]
```


Eindimensionale Reihungen

Deklaration: Feld mit Elementen vom Typ `type`

```
type[] var = new type[n];
```

`Feldtype` — `neues Feld der Längen`

deklariert lokale Variable `var` vom Type `type[]` und reserviert Speicherplatz für eindimensionale Reihung der Länge `n`

Durch die Deklaration werden außerdem implizit `n` zusammengesetzte Variablen `var[0],..., var[n-1]` erzeugt, mit denen man auf die Werte der Komponenten von `var` zugreifen und diese Werte verändern kann.

Eindimensionale Reihungen: Initialisierung

Sofort Anfangswerte zuweisen („Initialisierung“)

- `type[] var = {`v_0`, ..., `v_{n-1}`} // sofortige Zuweisung`

vom Typ `type`

! Diese Art der Initialisierung ist aber nur in einer Deklaration zulässig.

- oder Zuweisung an Komponenten:

```
type[] var = new type[n];
var[0]   = v0;
  :
var[n-1] = vn-1;
```


Eindimensionale Reihungen

Beispiel:

```
char[] a = {„V“, „E“, „R“, „L“, „A“, „G“}
```

Typ von a ist **`char[]`**, d.h. Der Typ eines einstufigen Feldes mit Elementen aus char. Mit `a[0], a[1],..., a[5]` kann man auf die Komponenten von a zugreifen.

- Man kann die Werte einzeln zuweisen:

```
a[0] = ´V´        a[3] = ´L´
a[1] = ´E´        a[4] = ´A´
a[2] = ´R´        a[5] = ´G´
```


Eindimensionale Reihungen

- Man kann beliebige einzelne Buchstaben ändern:

```
a[3] = ´R´
a[5] = ´T´
```

- Das ergibt `´V´ ´E´ ´R´ ´R´ ´A´ ´T´` als neuen Wert des Feldes. Außerdem hat `a[3]` nun den Wert `´R´`.

Direkte Zuweisung

```
char[] c = {´L´ ´M´ ´U´};
a = c;
```

Bemerkung: Da in Java die Länge des Feldes aber nicht Bestandteil des Typs ist, kann einer Feldvariablen ein Feld mit einer anderen als initial angegebenen Länge zugewiesen werden.

Reihungen und for-Schleifen

- Die Länge eines Array-Objekts steht in dem Feld `length`

```
int[] myArray = new int [x*x+1];
int länge = myArray.length
```

- for-Schleifen eignen sich gut um Arrays zu durchlaufen

```
Würfel meinWürfel = newWürfel();
for (int k=0; k<länge; k++){
   meinWürfel.würfele();
   myArray[k] = meinWürfel.getAugenZahl();
   }
```

schreiben

- Typische Suche in einem Array - mit vorzeitigem Verlassen:

```
for (int k=0; k<länge; k++)
if (myArray[k] == 6) {
      gefunden = true; break;
}
```

lesen

Suche nach dem Index eines minimalen Elements einer Reihung

Gegeben sei folgendes Feld:

3	-1	15	1	-1

Algorithmus:

- Bezeichne *minIndex* den Index des kleinsten Elements
- Initialisierung *minIndex = 0*
- Durchlaufe das ganze Feld. In jedem Schritt *i* vergleiche den Wert von *minIndex* (d.h. `a[minIndex]`) mit dem Wert des aktuellen Elements (d.h. `a[i]`). Falls `a[i] < a[minIndex]` setze *minIndex = i*

Suche nach dem Index eines minimalen Elements einer Reihung

Java Implementierung

```
class C
{
   int[] a;
   int minSuche()
   {
        int minIndex = 0;
        for (int i = 1; i < a.length; i++) // Optimierung, da
                                           // a[0] < a[0] falsch ist
        {
            if (a[i] < a[minIndex])
                minIndex = i;
        }
        return a[minIndex];
   }
}
```


Binäre Suche eines Elements *e* in einer geordneten Reihung

Sei *a* ein geordnetes Feld mit den Grenzen *j* und *k*, d.h. $a[i] \leq a[i+1]$ für `j <= i < k;` also z.B.:

a:	3	7	13	15	20	25	28	29
	j	*j*+1	...					*k*

Algorithmus:

Um den Wert *e* in *a* zu suchen, teilt man das Feld in der Mitte und vergleicht *e* mit dem Element in der Mitte:

- Ist `e < a[mid]`, so sucht man weiter im linken Teil `a[j],..., a[mid-1]`.
- Ist `e = a[mid]`, hat man das Element gefunden.
- Ist `e > a[mid]`, so sucht man weiter im rechten Teil `a[mid+1],..., a[k]`.

Binäre Suche eines Elements *e* in einer geordneten Reihung

```
class C
{       int[] a;
        boolean binSuch(int e)
        {   int j = 0,
                k = a.length-1,
                mid;
            boolean found = false;
            while (j <= k & !found)  {
                mid = (j + k) / 2;
                 if (e < a[mid])
                     k= mid - 1;
                else
                {  if (e = a[mid])
                        found = true;
                else
                        j = mid + 1;
                }
            }
        return found;
          }
 }
```


Mehrdimensionale Reihungen

- Matrizen sind mehrdimensionale Arrays
- Man benutzt Matrizen zur Speicherung und Bearbeitung von
  - Bildern
  - Operationstabellen
  - Wetterdaten
  - Graphen
  - Distanztabellen
  - ...
- Deklaration
  - `int [][] greyMonaLisa;`
  - `Color[][][] rubik;`
- Deklaration mit Erzeugung
  - `Color[][] bildschirm = new Color[1024][748];`

Mehrdimensionale Reihungen

- Deklaration mit Initialisierung
  - `boolean[][] xorTabelle = {{false, true}, {true, false}}`
  - `int[][]entfernung = {{  0, 213, 419, 882}`
    `{213,   0, 617, 720},`
    `{419, 617,   0, 521},`
    `{882, 720, 521,   0}};`

Beispiel: Bildbearbeitung

- Graphik als Matrix von Grauwerten
  - `int[][] monaGrey = new int [3][3];`
  - `monaGrey = {{21, 98, 205, 23};`
    `{32, 37, 126, 98},`
    `{113, 47, 191, 139},`
    `{107, 189, 191, 96}};`
- Aufhellen
  - `for (int x = 0; x < höhe; x++)`
    `for(int y = 0; y < breite; y++)`
    `monaGrey[x][y] = monaGrey[x][y]*9/10;`
- Negieren
  - `for (int x = 0; x < höhe; x++)`
    `for(int y = 0; y < breite; y++)`
    `monaGrey[x][y] = 255-monaGrey[x][y];`
- Schwarzweiss
  - `for (int x = 0; x < höhe; x++)`
    `for(int y = 0; y < breite; y++)`
    `if(monaGrey[x][y] < 128) monaGrey[x][y] = 0;`
    `else = monaGrey[x][y] = 255;`

Mehrdimensionale Reihungen

Allgemein

`type[]...[] var = new type[`n_1`]...[`n_i`][]...[]  (`$i > 0$`)`

deklariert eine Variable `var` vom Typ `type[]...[]`, reserviert Speicherplatz für ein mehrstufiges Feld.

- Mindestens die Länge n_1 des ersten Indexbereiches muß angegeben werden.

Initialisierung:

`type[][] var = {`$f_0, \ldots, f_{n_1-1}$`}`

Felder von Werten vom Typ `type[]`

Ein mehrdim. Feld ist ein Feld von Feldern

- Dabei können die Längen von $f_0, \ldots, f_{n_1-1}$ unterschiedlich sein.
- Analog für höherdimensionale Felder.

Mehrdimensionale Reihungen

Ausdrücke

- Zusammengesetzte Variablen

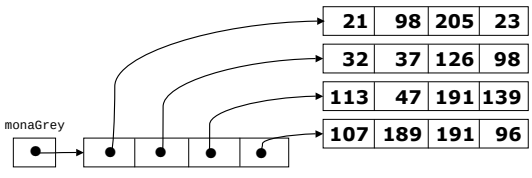
  `var[`i_1`]...[`i_n`]`

- Initialisierungsausdrücke der Form

  `new type[`n_1`]...[`n_i`][]...[]  bzw. {`$v_0, \ldots v_{n_1-1}$`}`

Mehrdimensionale Reihungen - gibt's gar nicht!

- Mehrdimensionale Arrays braucht man **eigentlich** nicht
  - Ein zweidimensionaler Array ist ein Array von Zeilen
  - Ein dreidimensionaler Array ist ein Array von zweidimensionalen Arrays
  - ...



Mehrdimensionale Reihungen

Beispiel: Tastentelefon

Ein Tastentelefon besteht aus 4 Zeilen und 3 Spalten:

	0	1	2
3	1	2	3
2	4	5	6
1	7	8	9
0	*	0	#

```
char[][] tastenwert = { {'*', '0', '#' },
                        {'7', '8', '9' },
                        {'4', '5', '6' },
                        {'1', '2', '3' } };
```

z.B.

`tastenwert[3][0] = '1'`

`tastenwert[0][2] = '#'`

Arrays: Krumm und schief

- Die Dimension eines Arrays ist nicht Teil seines Typs
  - Folglich können verschieden große Arrays gleichen Typ haben

```
int [] alt = new int[3];
int [] neu = new int[17];
alt = neu;                  // das ist in Java möglich!
```

- Eine Matrix kann verschieden lange Zeilen haben
  - `int[][] pascalDreieck = {`

```
                {1}
              {1, 1},
            {1, 2, 1},
          {1, 3, 3, 1},
        {1, 4,6 , 4, 1}};
```

- Wie durchläuft man krumme Arrays?
  - Die innere for-Schleife muss die Länge der zu durchlaufenden Zeilen selber bestimmen
  - Das geht mittels des `length`-Feldes

Arrays: Krumm und schief

- Durchlauf durch `schiefArray`

```
for(int zeile=0; zeile < schiefArray.length; zeile++)
  for(int spalte=0; spalte < schiefArray[zeile].length; spalte++)
    tuWasSinnvollesMit(schiefArray[zeile][spalte]);
```


Zusammenfassung

- Felder sind mathematisch endliche Abbildungen von einem Indexbereich auf einen Elementbereich.
- In Java sind Felder Objekte mit einer speziellen Syntax für den Zugriff auf die Attribute.
- Klassische Suchalgorithmen sind die binäre Suche im geordneten Feld und die Suche nach dem Index mit dem kleinsten Element im untergeordneten Feld.