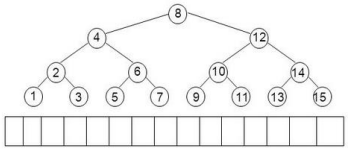
Projekt 11: Schnelles Suchen durch binäres Suchen



Aufgabe 1: Durch das Sortieren von Daten kann die Suchzeit beim Suchen eines Namens in einem Array sehr stark verkürzt werden. Dies ist insbesondere bei großen Kundendateien in einer Firma sehr wichtig.
Lies im Buch S. 143 den Abschnitt „Suchen in sortieren Daten – Binäre Suche“. Bearbeite dann das Arbeitsblatt „Projekt 11 – Schnelles Suchen durch binäres Suchen“, bei dem du mit möglichst wenigen Vergleichen in einem Array mit 60 beliebten Vornamen der Deutschen einen bestimmten Suchnamen, z. B. „Eva“ finden sollst. Bearbeite die Beispiele „Eva“ und „Chantal“, im einen Fall gibt es den Suchnamen im Array, im anderen nicht.

Aufgabe 2: Kopiere das Projekt „BeliebteVornamen Binär“ in dein Schülerverzeichnis. Ergänze den Suchvorgang „binäre Suche“, indem du zunächst in der Methode **vorbereiten()** die 250 Vornamen mit einem Sortieralgorithmus deiner Wahl sortierst. Verwende dazu ein weiteres Array **sortierteVornamen**, in dem die sortierten Vornamen gespeichert werden. Vor dem Sortieren musst du alle Vornamen einfach in dieses Array übertragen. Anschließend soll die binäre Suche in der Methode **eingabeBoxKeyReleased** implementiert werden, so wie sie im Pseudocode auf dem Arbeitsblatt vorgestellt wird. Die Methode reagiert auf jeden Tastendruck auf der Tastatur. Die Zahl der Suchvorgänge soll wie bisher ausgegeben werden. Du musst noch wissen, wie man zwei Objekte des Types String miteinander vergleicht, dazu benötigt man die Methode **compare**:

```
vergleich = sortierteVornamen [mitte].compareTo (Eingabe);
```

compare vergleicht zwei Strings (einer steht vor dem Wort **compareTo**, der zweite wird in Klammern hinter dem Wort **compare** angegeben) und liefert als Ergebnis einen Wert vergleich vom Typ Integer. Diesen Wert kann man abfragen:

Befehl	Bedeutung
if (**vergleich** == 0) …	**sortierteVornamen [mitte]** = **Eingabe**
if (**vergleich** < 0) …	**sortierteVornamen [mitte]** < **Eingabe**
if (**vergleich** > 0) …	**sortierteVornamen [mitte]** > **Eingabe**

In der linken **JTextArea ausgabe1** soll das sortierte Array der Vornamen zusätzlich ausgegeben werden.

Aufgabe 3: Die Vornamen sind zu Beginn nach ihrer Beliebtheit sortiert, z. B. ist „Anna“ der beliebteste Vornamen der Deutschen. Diese Reihenfolge geht durch das alphabetische Sortieren zunächst verloren. Damit diese Information weiterhin vom Programm ausgegeben werden kann, füht man ein weiteres Array **platzNummer** ein, indem vor dem Sortieren die jeweilige Position der Vornamen im nach der Beliebtheit sortierten Array gespeichert wird. Beim Sortieren selbst werden sowohl die sortierten Vornamen als auch die Platznummern getauscht. In den sortierten Vornamen kann man schließlich über die **platzNummer** die alte Position eines Vornamens im Array **vornamen** abfragen.

Deklaration des Arrays:

```
int [] platzNummer = new int [250];
```

Initialisierung des Arrays:

```
for (int k = 0; k <= 249; k++){
  sortierteVornamen [k] = Vornamen [k];
  platzNummer [k] = k + 1;
}
```

Beim Sortiervorgang:

```
if (position > j){
  name1 = sortierteVornamen [position];
  sortierteVornamen [position] = sortierteVornamen [j];
  sortierteVornamen [j] = name1;
  nummer1 = platzNummer [position];
  platzNummer [position] = platzNummer [j];
  platzNummer [j] = nummer1;
}
```

Zusatzaufgabe: Im Buch auf S. 144 bis 147 wird ein weiteres Verfahren erklärt, mit dem man bei großen Datenmengen in kurzer Zeit einen bestimmten Datensatz suchen kann – das **Hashing**. Lies dir zunächst den Text durch.
Da wir es mit Vornamen zu tun haben, benötigen wir zunächst eine Berechnungsfunktion, die zu jedem Namen eine Zahl berechnet, damit der Name unter dem Index abgespeichert werden kann, der der Zahl entspricht. Dazu ist folgendes Verfahren sehr effektiv: Man übersetzt mit Hilfe des ASCII-Codes zunächst jeden Buchstaben in eine Integer-Zahl, z. B. aus „A“ wird 65, wie du der Tabelle entnimmst, man benutzt dazu die Methode **charAt**, wie du dem Programmsegment unten entnehmen kannst. Jetzt geht man hin und summiert die ACSII-Codes aller Buchstaben des Namens auf.

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Damit die Zahl nicht zu groß wird, berechnet man den Rest der Division dieser Zahl durch die vorgegebene Größe des Arrays, indem die Vornamen gespeichert werden sollen. Wir wählen als Grenze 400, d. h. in einem Array mit 400 Plätzen werden unsere 250 Vornamen gespeichert.

Mit dem Befehl

```
Hashcode = Hashcode % 400;
```

wird der Rest der Division durch 400 ermittelt (also z. B. 650 % 400 = 250). In der Mathematik schreibt nman dazu oft auch 650 mod 400 = 250, womit der Divisionsrest gemeint ist. Zu Beginn wird ein leerer String in allen Plätzen des Arrays gespeichert. Wenn der Algorithmus versucht, einen Vornamen an einer bestimmten Stelle zu speichern, wird zunächst geprüft, ob dieser Platz noch frei ist. Ist der Platz schon belegt, wird einfach der nächste Platz belegt:

```
NeuerPlatz = (NeuerPlatz + 1) % 400;
```

Der nebenstehende Algorithmus zeigt das Verfahren, mit dem die Hash-Tabelle erzeugt wird, in der später effektiv gesucht werden kann.

Implementiere diesen Algorithmus und ergänze das Programm so, dass ein beliebiger Vorname durch Hashing gesucht werden kann. Beim Suchen verwendet man am besten folgende Strategie: Zunächst berechnet man zum eingegebenen Suchnamen den Hashcode und schaut nach, ob an dieser Stelle der gesuchte Name steht. Steht er nicht dort, prüft man zunächst ob der betreffende Platz leer ist: in diesem Fall steht der Suchname nicht im Array der Vornamen. Steht dort ein anderer Name, geht man zum nächsten Platz und sucht erneut, bis man entweder einen leeren String oder den gesuchten Vornamen findet.

In der linken **JTextArea ausgabe1** soll die Berechnung des Hash-Codes sichtbar werden und anschließend alle Plätze angezeigt werden, auf denen der Vorname gesucht wurde (s. Beispiel Rechts). Zwei weitere Arrays können zur Angabe der Hsh-Plätze benutzt werden:
platzNummer [i]: weist einem Hash-Platz die Nummer des gespeicherten Vornamens zu
hashPlatz [i]: weist jedem Vornamen seinen Hash-Platz zu

```
int hashCode,neuerPlatz; String text1;
ausgabe1.setText ("Hashing-Plätze:\n");
for (i = 0; i <= 399; i++){
  hashTabelle [i] = "";
}
for (i = 0; i <= 249; i++){
  hashCode = 0;  text1 ="";
  for (j = 0; j < vorNamen [i].length(); j++){
    hashCode = hashCode + (int) vorNamen [i].charAt(j);
    text1 = text1 + (int) vorNamen [i].charAt(j) + " ";
  }
  hashCode = hashCode % 400;
  ausgabe1.append (vorNamen [i] + " H:" + String.valueOf (hashCode) +
        " " + text1 + "\n");
  neuerPlatz = hashCode;
  while (hashTabelle [neuerPlatz].equals ("") == false){
    neuerPlatz = (neuerPlatz + 1) % 400;
    ausgabe1.append ("  neuer Platz: " + String.valueOf (neuerPlatz) +
          "\n");
  }
 hashTabelle [neuerPlatz] = vorNamen [i];
 platzNummer [neuerPlatz] = i + 1;
 hashPlatz [i] = neuerPlatz;
  ausgabe1.append ("Platz gesetzt: " + String.valueOf (neuerPlatz) +
        "\n");
}
ausgabe1.setCaretPosition (0);
```

```
int hashCode,neuerPlatz; String text1;
 ausgabe1.setText ("Hashing-Plätze:\n");
 for (i = 0; i <= 399; i++){
  hashTabelle [i] = "";
 }
 for (i = 0; i <= 249; i++){
  hashCode = 0;  text1 ="";
  for (j = 0; j < vorNamen [i].length(); j++){
   hashCode = hashCode + (int) vorNamen [i].charAt(j);
   text1 = text1 + (int) vorNamen [i].charAt(j) + " ";
  }
  hashCode = hashCode % 400;
  ausgabe1.append (vorNamen [i] + " H:" + String.valueOf (hashCode) +
       " " + text1 + "\n");
  neuerPlatz = hashCode;
  while (hashTabelle [neuerPlatz].equals ("") == false){
   neuerPlatz = (neuerPlatz + 1) % 400;
   ausgabe1.append ("  neuer Platz: " + String.valueOf (neuerPlatz) +
        "\n");
  }
  hashTabelle [neuerPlatz] = vorNamen [i];
  platzNummer [neuerPlatz] = i + 1;
  hashPlatz [i] = neuerPlatz;
   ausgabe1.append ("Platz gesetzt: " + String.valueOf (neuerPlatz) +
        "\n");
 }
 ausgabe1.setCaretPosition (0);
```

Text kann kopiert werden

```
Lara H:384 76 97 114 97
  neuer Platz: 385
  neuer Platz: 386
  neuer Platz: 387
  neuer Platz: 388
Vorname Lara gefunden: 388
```