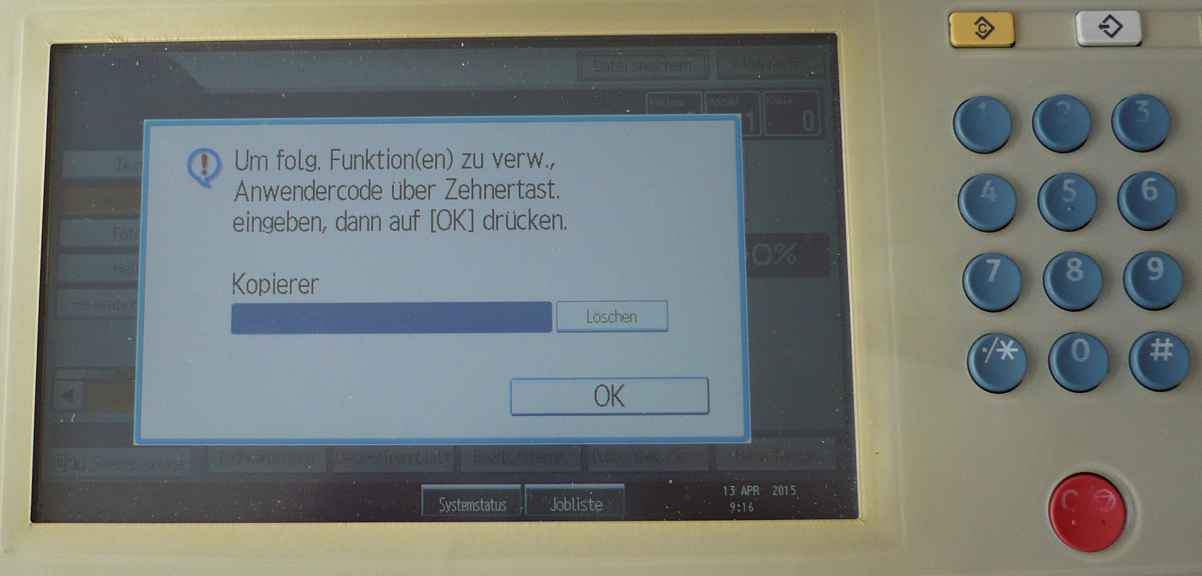
Suchen auf linearen Datenstrukturen

Benutzerverwaltung für einen Fotokopierer



Die Softwareabteilung eines Herstellers von Fotokopierer stellt verschiedene Lösungen für die Benutzerverwaltung vor. Beim Einschalten des Kopierers werden die Benutzerdaten in einen Arbeitsspeicher geladen.

Benutzer des Kopierers geben eine vierstellige Ziffernkombination an, um sich zu legitimieren. Über diesen PIN-Code soll auch der Benutzer des Kopierers identifizierbar sein, also darf jede PIN nur einmal vergeben werden.

Gesucht wird eine Organisation der Benutzerdaten, die einen schnellen Zugriff auf die Datensätze ermöglicht.

Zur Bedeutung des Suchens:

„Eine der häufigsten Aufgaben in der Informatik ist das Suchen von Daten. Es gibt eine Vielzahl von Verfahren für die unterschiedlichsten Aufgaben. Wir wollen zunächst nur den einfachsten Fall betrachten: Das Suchen in sortierten Folgen. Natürlich können wir auch auf das Suchen in unsortierten Folgen eingehen, allerdings werden wir sofort sehen, dass hier nicht viel zur Verbesserung der Suche getan werden kann.“[[1]](#footnote-1)

In dem oben beschriebenen Anwendungskontext wird sowohl das Suchen in sortierten Folgen (nach der PIN) als auch das Suchen in unsortierten Folgen (nach dem Namen bzw. der Id) benötigt.

Entwurf einer Testumgebung der Benutzerverwaltung:

151012_EntwurfsdiaSuche

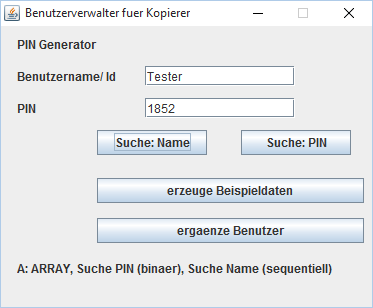
Dokumentation

Klasse **BenutzerVerwalter**

Ein Objekt der Klasse **BenutzerVerwalter** verwaltet die Daten aller Kopiererbenutzer.

|  |  |
| --- | --- |
| Auftrag | **void neuerBenutzer(pName: Text)**  Ein neuer Benutzer wird der Datensammlung hinzugefügt. Die PIN wird zufällig ermittelt. |
| Anfrage | **sucheName(pName: Text): Benutzer**  Ein Objekt der Klasse **Benutzer** aus der Datensammlung mit dem Namen **pName** wird zurück geliefert. |
| Anfrage | **suchePIN(pPIN: Text): Benutzer**  Ein Objekt der Klasse **Benutzer** aus der Datensammlung mit der PIN **pPIN** wird zurück geliefert. |

Die Testumgebung hat die folgende Benutzungsoberfläche.



Benutzerverwaltung in einem ARRAY

Variante A

In einem ARRAY befinden sich die vergebenen zufälligen PIN-Nummern und die Benutzerdaten, die aufsteigend nach den PIN-Nummern sortiert sind. Da die Benutzerdaten in einem ARRAY organisiert sind, kann über einen Index auf die Benutzerdaten eines Benutzers zugegriffen werden. Bei der Suche nach der PIN kann hier die Suche durch eine entsprechende Strategie verbessert werden. Die Suche nach dem Namen ist kaum optimierbar.

Benutzerverwaltung in einer Liste

Variante L

In einer Liste befinden sich die Benutzerdaten, diese sind nach den zufälligen vergebenen PIN-Nummern aufsteigend sortiert. In der Liste ist nur eine sequentielle Suche möglich, der Vorteil bei einer aufsteigenden Sortierung ist gering. Wenn die PIN-Nummer gefunden wird oder wenn eine PIN-Nummer vorliegt, die größer als die gesuchte PIN-Nummer ist, kann die Suche abgebrochen werden.

Benutzer

In einem Objekt der Klasse **Benutzer** werden folgende Daten des Benutzers verwaltet.

|  |  |
| --- | --- |
| **name** | Name oder ID des Benutzers. Der Name ist eindeutig. |
| **pin** | Die vierstellige Ziffernkombination, über die der Benutzer identifiziert werden kann und mit der sich der Benutzer am Kopierer anmelden kann. |
| **limit** | Die Zahl beschreibt die Anzahl der Kopien, die der Benutzer erstellen darf. |
| **verbrauch** | Die Zahl beschreibt die Anzahl der Kopien, die der Benutzer bereits erstellt hat. |

Dokumentation

Klasse **Benutzer**

Ein Objekt der Klasse **Benutzer** verwaltet die Daten eines Kopererbenutzers.

|  |  |
| --- | --- |
| Konstruktor | **Benutzer(String pName, String pPIN)**  Ein Benutzer wird angelegt. Als Parameter werden der eindeutige Benutzername und die zugehörige PIN festgelegt. |
| Auftrag | **void erhoeheVerbrauch(int pZahl)**  Der Verbrauch des Benutzers wird um **pZahl** erhöht. |

Variante A1

Die Suche nach dem Namen und nach der PIN kann unterschiedlich gestaltet werden.

Suche nach dem Namen

Bei der Suche nach Namen muss man eine sequentielle Suche anwenden. Die Folge wird einfach beginnend mit dem ersten Element durchsucht. In jedem Schritt wird das aktuelle Element mit dem Suchschlüssel (hier der gesuchte Name) verglichen. Sobald man das gesuchte Element gefunden hat, kann man die Suche beenden. Die Suche kann abgebrochen werden, weil der Benutzername eindeutig ist. Werden alle Elemente in der Folge verglichen und der Suchschlüssel nicht gefunden, so kann man sicher sein, dass es keinen Benutzer mit dem angegebenen Namen gibt.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **public Benutzer sucheName(String pName) {**  **boolean gefunden = false;**  **int zaehler = 0;**  **while (gefunden == false && zaehler < benutzerDaten.length) {**  **if (benutzerDaten[zaehler] != null**  **&& benutzerDaten[zaehler].gibName().equals(pName)) {**  **gefunden = true;**  **} else {**  **zaehler = zaehler + 1;**  **}**  **}**  **if (zaehler < benutzerDaten.length) {**  **return benutzerDaten[zaehler];**  **} else {**  **return null;**  **}**  **}** |

Aufwand der sequentiellen Suche bei einer Folge der Länge n[[2]](#footnote-2)

|  |  |
| --- | --- |
|  | Anzahl der Vergleiche |
| bester Fall | **1** |
| schlechtester Fall | **n** |
| Durchschnitt (erfolgreiche Suche) | **(n + 1)/2** |
| Durchschnitt (erfolglose Suche) | **n** |

Suche nach der PIN

Bei der Suche nach der PIN kann eine Strategie angewandt werden, die zu wenigen Suchschritten führt.

Zuerst wird der mittlere Benutzer überprüft. Die PIN des Benutzers kann kleiner, gleich oder größer als die gesuchte PIN sein.

Ist die PIN des Benutzers zum Beispiel kleiner als die gesuchte PIN, so muss der Benutzer mit der gesuchten PIN in der hinteren Hälfte stecken, die vordere Hälfte muss gar nicht mehr durchsucht werden.

Ist die PIN des Benutzers gleich der gesuchten PIN, so ist der Benutzer gefunden und die Suche ist beendet.

In der zu untersuchenden Hälfte und in den weiteren zu untersuchenden Abschnitten wird genauso verfahren. Die Länge des Suchbereichs wird in jedem Suchschritt halbiert. Spätestens wenn der Suchbereich auf einen Benutzer geschrumpft ist, ist die Suche beendet. Entweder ist der Benutzer mit der gesuchten PIN gefunden oder es existiert kein Benutzer mit der gesuchten PIN.

Der zu durchsuchende Bereich wird auf diese Weise immer halbiert – daher heißt dieses Verfahren auch **binäres Suchen**.

Beispiel



Dargestellt ist eine sortierte Zahlenfolge und es soll nach dem Element 8 gesucht werden. Die untere Grenze ist 0, die obere 9. Daraus ergibt sich die mittlere Position mit dem Index 4. Da der Suchschlüssel größer als das Element an dieser Position ist, müssen wir in der rechten Hälfte weiter suchen.Für den nächsten Schritt gilt: links = 5, rechts = 9 und mitte = 7. Da 8 kleiner als das Element in der Mitte ist, wird der Suchbereich auf links = 5 rechts = 6 eingeschränkt. Das gesuchte Element (mitte = 5) wird nun gefunden.

151013_ASuch2

Die PIN wird als String bearbeitet, weil sie auch führende Nullen haben kann.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **24** | **private Benutzer suchePINBinaerI(String pSuchPIN) {**  **int links, rechts, mitte, index;**  **links = 0;**  **rechts = aktBenutzerzahl;**  **index = -1;**  **while (links <= rechts && index < 0) {**  **mitte = (rechts + links)/2;**  **if (benutzerDaten[mitte].gibPIN().equals(pSuchPIN)) {**  **index = mitte;**  **} else {**  **if (benutzerDaten[mitte].gibPIN().compareTo(pSuchPIN) < 0) {**  **links = mitte + 1;**  **} else {**  **rechts = mitte - 1;**  **}**  **}**  **}**  **if (index >= 0) {**  **return benutzerDaten[index];**  **} else {**  **return null;**  **}**  **}** |

Die rekursive Variante des binären Suchens kann leicht abgeleitet werden, indem als Parameter des Algorithmus die untere und obere Grenze des zu durchsuchenden Bereichs übergeben werden kann.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17** | **public Benutzer suchePIN(String pSuchPIN) {**  **return suchePINbinaerR(pSuchPIN, 0, aktBenutzerzahl-1);**  **}**  **private Benutzer suchePINbinaerR(String pSuchPIN, int pLinks,**  **int pRechts) {**  **if ((pLinks < 0) || (pRechts > aktBenutzerzahl)**  **|| (pLinks > pRechts) ) {**  **return null;**  **} else {**  **int mitte = (pLinks + pRechts)/2;**  **if (benutzerDaten[mitte].gibPIN().equals(pSuchPIN)) {**  **return benutzerDaten[mitte];**  **} else if (benutzerDaten[mitte].gibPIN().**  **compareTo(pSuchPIN) > 0) {**  **return suchePINbinaerR(pSuchPIN, pLinks, mitte-1);**  **} else {**  **return suchePINbinaerR(pSuchPIN, mitte+1, pRechts);**  **}**  **}**  **}** |

Aufwand der binären Suche bei einer Folge der Länge n[[3]](#footnote-3)

|  |  |
| --- | --- |
|  | Anzahl der Vergleiche |
| bester Fall | **1** |
| schlechtester Fall | **≈ log2 n** |
| Durchschnitt (erfolgreiche Suche) | **≈ log2 n** |
| Durchschnitt (erfolglose Suche) | **≈ log2 n** |

Variante L

Sowohl bei der Suche nach Namen als auch bei der Suche nach einer PIN muss die Suche vorne in der Liste beginnen und im schlechtesten Fall bis zum letzten Element durchsucht werden. Die Suche ist beendet, wenn der gesuchte Name oder die gesuchte PIN gefunden worden ist. Ist der gesuchte Name nicht vorhanden, so bemerkt man es erst, wenn man die Liste bis zum Ende durchsucht hat und der gesuchte Name nicht gefunden worden ist.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17** | **public Benutzer sucheName(String pName) {**  **boolean gefunden = false;**  **benutzerDaten.toFirst();**  **while (gefunden == false && benutzerDaten.hasAccess()) {**  **Benutzer b = benutzerDaten.getContent();**  **if (b != null && b.gibName().equals(pName)) {**  **gefunden = true;**  **} else {**  **benutzerDaten.next();**  **}**  **}**  **if (gefunden) {**  **return benutzerDaten.getContent();**  **} else {**  **return null;**  **}**  **}** |

Bei der Suche nach einer PIN kann man die Suche beenden, wenn die PIN eines Benutzers größer als die gesuchte PIN ist. Die gesuchte PIN kann im hinteren Teil der Liste nicht mehr vorkommen, da die Liste nach der PIN sortiert ist.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21** | **public Benutzer suchePIN(String pPIN) {**  **boolean gefunden = false;**  **boolean weiter = true;**  **benutzerDaten.toFirst();**  **while (!gefunden && weiter && benutzerDaten.hasAccess()) {**  **Benutzer b = benutzerDaten.getContent();**  **if (b.gibPIN().equals(pPIN)) {**  **gefunden = true;**  **} else {**  **benutzerDaten.next();**  **}**  **if (pPIN.compareTo(b.gibPIN()) < 0) {**  **weiter = false;**  **}**  **}**  **if (gefunden) {**  **return benutzerDaten.getContent();**  **} else {**  **return null;**  **}**  **}** |

Aufwand der sequentiellen Suche bei einer Folge der Länge n. Die Liste ist bezüglich des Suchschlüssels unsortiert.

|  |  |
| --- | --- |
|  | Anzahl der Vergleiche |
| bester Fall | **1** |
| schlechtester Fall | **n** |
| Durchschnitt (erfolgreiche Suche) | **(n + 1)/2** |
| Durchschnitt (erfolglose Suche) | **n** |

Aufwand der sequentiellen Suche bei einer Folge der Länge n. Die Liste ist bezüglich des Suchschlüssels sortiert.

|  |  |
| --- | --- |
|  | Anzahl der Vergleiche |
| bester Fall | **1** |
| schlechtester Fall | **n** |
| Durchschnitt (erfolgreiche Suche) | **(n + 1)/2** |
| Durchschnitt (erfolglose Suche) | **(n + 1)/2** |

Quellen

Saake, Gunter; Sattler, Kai-Uwe

Algorithmen und Datenstrukturen

Heidelberg 2010 (4. Überarbeitete Auflage): dpunkt.verlag

1. Saake/ Sattler S. 117 [↑](#footnote-ref-1)
2. Saake/ Sattler S. 119 [↑](#footnote-ref-2)
3. Saake/ Sattler S. 119 [↑](#footnote-ref-3)