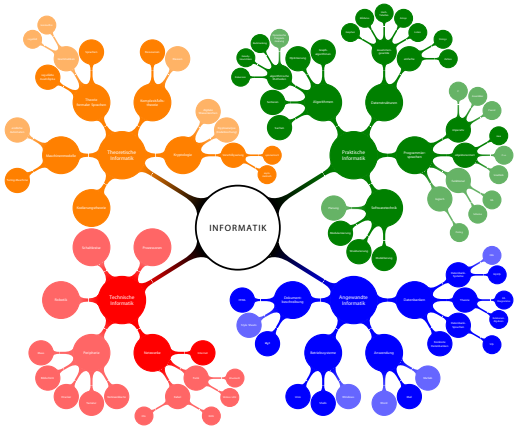
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 12
Zeichenketten

Vom Papyrus zur DNS-Sequenz

Vorlesung Einführung in die Informatik 1 vom 3. Dezember 2013 von Till Tantau



Lernziele von Kapitel 12

1. Zeichenketten (Strings) als Datentypen kennen
2. Einfache Stringalgorithmen kennen und implementieren können
3. Einen fortgeschrittenen Stringalgorithmus kennen

Gliederung von Kapitel 12

Zeichen umgeben uns ständig.

Wir leben in einer Welt, in der Schrift allgegenwärtig ist:

Zur Diskussion

- Wie viele Zeichen sind in einem Umkreis von 10 Metern auf Oberflächen gedruckt?
- Was ist die maximale Entfernung, die sie in Ihrem Leben jemals von Schriftzeichen entfernt waren?

Zeichenketten sind Folgen von Zeichen.

Definition (Zeichenkette, Alphabet)

Eine *Zeichenkette* ist eine Folge von Zeichen.
Die Zeichen sind Elemente eines festen *Alphabets*.

Beispiel

- Eine Zeichenkette von vier Zeichen über dem Alphabet ASCII:
  `ABCD`
- Eine Zeichenkette von fünf Zeichen über dem Alphabet UNICODE:
  `AB`ϒ`++`
- Eine Zeichenkette über dem genetischen Alphabet:
  `ACGTTACC`

Auch Leerzeichen sind Zeichen.

- Auch Leerzeichen sind Zeichen.
- Deshalb ist es nützlich, Zeichenketten in Hochkommas einzuschließen:
  `"Hallo␣␣"` hat 7 Zeichen (zwei Leerzeichen).
  `"␣"` hat 1 Zeichen (ein Leerzeichen).
  `""` hat 0 Zeichen.
- Auch Zeilenumbrüche sind Zeichen.
  Dadurch lassen sich auch »zweidimensionale« Texte als Zeichenketten auffassen:

Zweidimensionaler Text . . .

```
1  Sehr geehrte Damen und Herren,
2
3  hiermit ...
```

. . . als Zeichenkette

`Sehr geehrte Damen und Herren,¶¶hiermit ...`

Besonderheiten in Java.

- Strings werden in doppelte Anführungszeichen eingeschlossen.
- Um einen Zeilenumbruch einzugeben, wird die Zeichenfolge `\n` (»n« wei »newline«) benutzt:

```
Sehr geehrte Damen und Herren,\n\nhiermit
```

- Um Anführungszeichen einzugeben, kann man die Zeichenfolge `\"` verwenden.
- Um einen Backslash einzugeben, kann man die Zeichenfolge `\\` verwenden.

Strings sind ein Datentyp in Java.

- Zeichenkette bilden in Java einen eigenene Datentyp mit dem Namen *String*.
- Dieser Datentyp ist zwar kein elementarer Datentyp, er kann aber so verwendet werden:

```
String a = "hallo";
String b = "hal";
String c = "lo";
String d = b + c; // + ist die Verkettung

System.out.println(a + "␣und␣" + d + "␣sind␣gleich");
```

- Man kann *nicht* einzelne Zeichen eines `String` verändern.

Bequemlichkeiten in der Syntax.

- Man kann nicht nur Strings mittels + verketten, sondern auch Strings und andere Dinge.
- Die »anderen Dinge« werden dann automatisch in einen String verwandelt:

```
int i = 2+3;

System.out.println ("Fünf␣ist␣gleich␣" + i + ".");

// Gibt "Fünf ist gleich 5." aus
```

Datenstrukturen sind Datentypen zusammen mit ihren Operationen.

Zur Erinnerung:

Definition (Datentyp)

Ein *Datentyp* ist eine Menge von Werten.

Neu:

Definition (Datenstruktur)

Eine *Datenstruktur* ist ein Datentyp zusammen mit Operationen, die man auf Werten dieses Typs ausführen kann.

Beispiele von Datenstrukturen.

Beispiel (Der Datentyp `byte`)

Der *Datentyp* besteht aus

1. den Werten von -128 bis 127.

Beispiel (Die Datenstruktur `byte`)

Die *Datenstruktur* `byte` besteht aus

1. den Werten von -128 bis 127 und
2. den Operationen wie `+`, `*`, usw.

Zeichenketten als Datenstrukturen.

Die Datenstruktur `String`

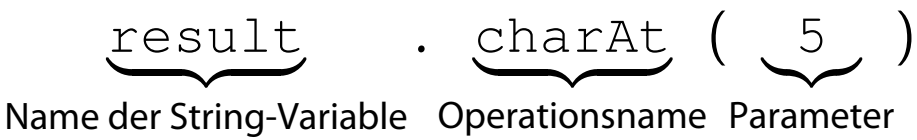
Die *Datenstruktur* *`String`* besteht aus

1. allen Unicode-Zeichenketten als Werten,
2. der Operation + zum Verketten,
3. der Operation `equals` zum Vergleichen von Strings (*nicht* ==),
4. der Operation `length` zum Bestimmen der Länge,
5. der Operation `charAt` zur Bestimmung der Zeichens an der *i*-ten Stelle.

Es gibt noch vielen weiteren Operationen, siehe dazu http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html.

Allgemeine Syntax für Operationen auf Strings.

Die Syntax aller neuen Operationen lautet:



Selbst wenn es keine Parameter gibt, müssen die Klammern geschrieben werden.

Syntax für die neuen Operationen auf Strings.

Syntax

Das *i*-te Zeichen eines Strings lässt sich wie folgt bestimmen:

```
String s = "hallo";
System.out.println(s.charAt(0)); // Gibt 'h' aus
System.out.println(s.charAt(1)); // Gibt 'a' aus

if (s.charAt(2) != s.charAt(3)) {
  System.out.println("Kann␣nicht␣sein.");
}
```

Achtung: Die Zählung beginnt bei Null!

Syntax

Die Länge eines Strings lässt sich wie folgt bestimmen:

```
String s = "hallo";

if (s.length() == 5) {
  System.out.println("Alles␣wird␣gut");
}
```

Syntax für die Vergleichsoperation auf Strings.

Syntax

Zwei Strings lassen sich wie folgt vergleichen:

```
String s1 = "Hallo";
String s2 = "Welt";
String s3 = "Hal" + "lo";

if (s1.equals (s2)) {
  System.out.println("Quatsch");
}

if (s1.equals (s3)) {
  System.out.println("Schon␣eher.");
}
```

Ein Algorithmus zum Umdrehen eines Strings.


```
String s      = "Dreh␣mich␣um!";

// Der folgende Algorithmus soll s umdrehen.

String result = "";

for (int i = s.length() - 1; i >= 0; i = i-1) {
  result = result + s.charAt(i);
}

s = result;

// jetzt ist s gerade "!mu hcim herD"
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie ein Java-Programm an, das das erste Zeichen eines Strings löscht. Ein Algorithmus hierzu: Beginnend mit einem leeren String `result` werden nacheinander alle Zeichen in `s` ab dem zweiten Zeichen an `result` angefügt.

```
String s      = "WWeg␣damit!";
String result = "";

... // Ihr Programm

s = result;

// jetzt ist s gerade "Weg damit!"
```

Ein Algorithmus zum Trimmen eines Strings.


```
String s      = "␣␣trimm␣mich!";

// Der folgende Algorithmus soll alle Leerzeichen
// am Anfang von s entfernen.

String result = "";
int    start = 0;

while (start < s.length () && s.charAt(start) == '␣') {
  start = start + 1;
}

for (int i = start; i < s.length(); i = i+1) {
  result = result + s.charAt(i);
}

s = result;

// jetzt ist s gerade "trimm mich!"
```

Naives Suchen.


```
String s          = "Sehr␣geehrte␣Damen␣und␣Herren";
String such_mich = "und";

// Folgender Algorithmus soll ermitteln, ob such_mich in s
// vorkommt.

boolean gefunden = false;

for (int i = 0;
     i <= s.length () - such_mich.length (); i = i+1)
{
  boolean ist_gleich = true;
  for (int j = 0; j < such_mich.length (); j = j+1) {
    if (s.charAt(i+j) != such_mich.charAt(j)) {
      ist_gleich = false;
    }
  }

  if (ist_gleich) {
    gefunden=true;
  }
}
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
              und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
              und
                 und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
              und
                 und
                    und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
              und
                 und
                    und
                     und
```

Die Idee hinter dem (vereinfachten) Boyer-Moore-Algorithmus.

Problem des naiven Suchens

Suchen eines Strings der Länge m in einem Text der Länge n dauert $n \cdot m$ Schritte.

Idee

Vergleich am Ende beginnen!

```
s == Sehr geehrte Damen und Herren
     und
        und
           und
              und
                 und
                    und
                     und
                        und
```

Der vereinfachte Boyer-Moore-Algorithmus

Input: Ein String `s` und ein Suchstring `such_mich`.
Frage: Ist `such_mich` als Teilstring in `s` enthalten?

Solange noch nicht gefunden tue:

1. Vergleich `such_mich` im aktuellen Fenster von hinten mit `s`.
2. Falls nicht gleich:
   - Betrachte letztes Zeichen von `s` im Fenster, an dem sich `such_mich` und `s` unterscheiden.
   - Schiebe das Fenster so weit vor, dass das letzte Vorkommen dieses Zeichens in dem String `such_mich` auf diesem Zeichen liegt.

Zusammenfassung

Alphabete, Wörter und Strings

- Ein *Alphabet* ist eine nichtleere endliche Menge von *Symbolen* (auch *Buchstaben* genannt).
- Ein *Wort* ist eine (endliche) Folge von Symbolen. In Programmiersprachen heißen Wörter *Strings*.

Die Datenstruktur `String`

Eine *Datenstruktur* besteht aus eine *Datentyp* zusammen mit *Operationen*.
Bei Strings:

- Der Datentyp `String` enthält alle Worte über dem Unicode.
- Die wichtigsten Operationen sind die *Verkettung (+)* sowie
  - `s.equals(t)` zum Vergleichen von Strings `s` und `t`,
  - `s.length()` zum Bestimmen der Länge von `s` und
  - `s.charAt(i)` zur Bestimmung des Zeichens an der *i*-ten Stelle von `s`.

Schleifen über Strings

Will man in einem String für alle Zeichen »etwas tun«, so benutzt man dazu eine Schleife:

```
String s = ...;
for (int i = 0; i < s.length(); i = i+1) {
  ... // tue etwas mit s.charAt(i)
}
```

Naive Suche

Bei der *naiven Suche* benutzt man (grob) obige Schleife, um für jede Stelle eines Strings zu testen, ob dort ein anderer String beginnt.