

Praktikum "SEP: Java-Programmierung"

WS 2018/19

Aufgabe 1: Trie

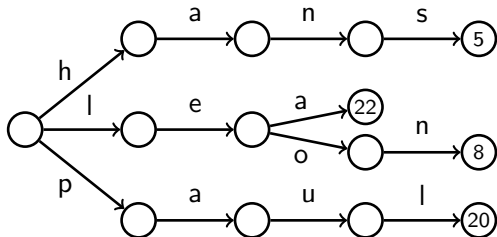
Thomas Lemberger und Martin Spießl

Basierend auf Folien von Matthias Dangl und Karlheinz Friedberger

- ▶ Einfache Aufgabe zum Einstieg:
ca. 550 LoC, inklusive JavaDoc
- ▶ Bestandteile:
 - ▶ Erstellung der Trie-Datenstruktur
 - ▶ Anwendungslogik: Verwaltung einer Zuordnung von Namen zu Punktzahlen (ECTS, Spielstände, ...)
 - ▶ Text-basierte Benutzerschnittstelle („Shell“) mit folgenden Funktionen:
 - ▶ Aufruf eines Hilfe-Texts
 - ▶ Einfügen von Einträgen (Name, Punktestand)
 - ▶ Änderung des Punktestands zu einem Namen
 - ▶ Text-Dump der Datenstruktur

Trie-Datenstruktur

- ▶ Spezieller Suchbaum (geeignet für Set, Dictionary/Map)
Bei uns: String \mapsto Punktzahl
- ▶ Effizient für hohe Verzweigungsgrade:
 - ▶ Speichereffizient für Schlüssel durch Deduplizierung gemeinsamer Bestandteile
 - ▶ Garantierte Zugriffszeit in Länge des Schlüssels
- ▶ Bsp.: hans \mapsto 5, lea \mapsto 22, leon \mapsto 8, paul \mapsto 20:



- ▶ Beschriftung:
 - ▶ Knoten: Schlüssel**w**erte
 - ▶ Kanten: Schlüssel**b**estandteile

- ▶ Shell
 - ▶ Benutzerschnittstelle
 - ▶ Setzt Benutzeraktionen in Aufrufe der Trie-Datenstruktur um
 - ▶ Gibt Ergebnisse aus
 - ▶ Klassengeheimnis: Interaktionsregeln
- ▶ Trie
 - ▶ Modellierung des abstrakten Datentyps Trie
 - ▶ Programmierschnittstelle für Einfügen, Suche, Ändern und Löschen
 - ▶ Kommunikationspartner der Shell
 - ▶ Klassengeheimnis: Knotenklasse und Wurzelknoten
- ▶ Node
 - ▶ Knoten eines Tries
 - ▶ Speichert Wert und Verzeigerung (d.h., Kanten zu anderen Knoten)
 - ▶ Klassengeheimnis: Verwaltung der Knotenverzeigerung
- ▶ Kanten nicht expliziert modelliert sondern implizit (siehe Klasse Node)

- ▶ Attribute
 - ▶ Punktzahl (Integer)
 - ▶ Kindknoten (Node[], Index von 0 (a) bis 25 (z))
 - ▶ Elterknoten/Parent (Node)
 - ▶ Zeichen der eingehenden Kante vom Elter/Parent (char)
- ▶ Konstrukturen / Methoden (zum Teil private!)
 - ▶ Konstruktoren erzeugen Wurzel bzw. Kindknoten für gegebenes Zeichen
 - ▶ `getPoints` und `setPoints` für Punktzahlen
 - ▶ `getChild` und `setChild` für Baumdurchlauf und -aufbau
 - ▶ `find` sucht Schlüssel im Unterbaum
 - ▶ `delete` löscht Punktzahlen und räumt ggf. unnötig gewordene Knoten auf
 - ▶ `toString` für Textdarstellung des Trie-Suchbaums
- ▶ Klasse `Trie` soll nur über wenige Methoden zugreifen (können)

- ▶ Attribute
 - ▶ Wurzelknoten (Node)
- ▶ Methoden
 - ▶ `add` zum Hinzufügen von Schlüssel-Wert-Paaren
 - ▶ `change` zum Ändern der Punktzahl eines Schlüssels
 - ▶ `delete` zum Löschen eines Schlüssels
 - ▶ `getPoints` zum Auslesen der Punktzahl eines Schlüssels
 - ▶ `toString` für Textdarstellung des Trie-Suchbaums

Klasse Shell als Benutzerschnittstelle

- ▶ Folgende Befehle sollen unterstützt werden:
 - NEW Neuer (leerer) Trie. Ggf. bestehender Trie wird verworfen
 - ADD s x Neuen Schlüssel s mit Punktzahl x (nicht-negative ganze Zahl) einfügen
 - CHANGE s x Punktzahl des Schlüssels s ändern auf Wert x
 - DELETE s Schlüssel s löschen
 - POINTS s Punktzahl des Schlüssels s ausgeben
 - TRIE Gesamten Trie ausgeben
 - HELP Ausgabe eines sinnvollen Hilfetexts
 - QUIT Programm beenden
- ▶ Eindeutiges Präfix des jeweiligen Kommandos genügt
- ▶ Groß- und Kleinschreibung soll egal sein
- ▶ Beim Programmstart soll bereits ein leerer Trie vorhanden sein
- ▶ Alle Fehlerm. beginnen mit `Error!`
 - ▶ Falscher Befehl
 - ▶ Fehlerhafter Befehl (z.B. negative Punktzahl)
 - ▶ Schlüssel schon bzw. nicht vorhanden

Textuelle Darstellung eines Tries

- ▶ Darstellung eines Unterbaums ab einem Knoten (geordnete Tiefensuche)
 1. Ausgabe des Zeichens der Kante vom Elternknoten
Spezialfall Wurzel: +
 2. Falls Punktzahl vorhanden, Ausgabe in eckigen Klammern
 3. Falls Kinder vorhanden, in alphabetischer Reihenfolge in runden Klammern
- ▶ Für Eingangsbeispiel:
+(h(a(n(s[5])))l(e(a[22]o(n[8])))p(a(u(1[20]))))

Beispieldialog

```
trie> trie
+
trie> add lea 22
trie> add hans 5
trie> points hans
5
trie> add paul 20
trie> add hans 5
Error! hans is already present.
trie> trie
+(h(a(n(s[5])))l(e(a[22]))p(a(u(1[20])))
trie> add leon 8
trie> add peter 21
trie> add paula 6
trie> trie
+(h(a(n(s[5])))l(e(a[12]o(n[8]))p(a(u(1[20] (a[6]))e(t(e(r[21])))))
trie> change peter 0
trie> trie
+(h(a(n(s[5])))l(e(a[12]o(n[8]))p(a(u(1[20] (a[6]))e(t(e(r[0])))))
trie> delete leon
trie> quit
```

- ▶ Dokumentation des Quellcodes ist Bestandteil der Aufgabe
- ▶ JavaDoc-Regeln:
 - ▶ `/**` als Kommentareinleitung
 - ▶ `@param` für jeden Parameter
 - ▶ `@return` falls Rückgabetyt nicht `void` ist
 - ▶ `@throws` für jeden geworfenen Exceptiontyp
- ▶ Leere Kommentare sind keine Kommentare
- ▶ Kommentierpflichtig: **Mindestens** alles, was `public` oder `protected` ist. Referenz: Google Java Code Style <https://tinyurl.com/g00glJDocStyle>

- ▶ Grundregeln zum Behandeln von Exceptions
 - ▶ `NumberFormatException` darf behandelt werden
 - ▶ Sonst keine unchecked Exception, insbesondere nicht:
 - ▶ `ArrayIndexOutOfBoundsException`
 - ▶ `NullPointerException`
 - ▶ `IOException`
- ▶ Außerdem zu beachten:
 - ▶ Schlüssel besteht nur aus Kleinbuchstaben a-z
 - ▶ Vereinfachung: Kein leerer Schlüssel „“, sondern immer mind. 1 Buchstabe vorhanden
- ▶ Hauptdatei mit main-Methode heißt `Shell.java`
- ▶ **Zusätzlich notwendige Abgabe:** Testprotokoll mit eigenen Testfällen (Dialog auf Konsole)
Datei `Tests.txt` auf Praktomat zusammen mit Quellcode hochladen.

- ▶ LRZ GitLab benutzen: <https://gitlab.lrz.de/>

Viel Erfolg!

- ▶ Nächste Woche:
Codestyle + erste Fragen
- ▶ Fragen an: thomas.lemberger@sosy.ifi.lmu.de