Algorithmen und Datenstrukturen
Übungsblatt 03

Prof. Karsten Weihe

TECHNISCHE UNIVERSITÄT DARMSTADT

SoSe 2022	v1.1
Themen:	Erster „richtiger“ Algorithmus auf Sequenzen
Relevante Foliensätze:	Foliensätze/Video zu String Matching
Abgabe der Hausübung:	bis 20.05.2022, 23:50 Uhr

Hausübung 03 — Gesamt: 32 Punkte

String Matching BOFA

Verbindliche Anforderungen für alle Hausübungen:

Das Dokumentieren und Kommentieren Ihres Quelltextes ist nicht verbindlich, wird zum besseren Verständnis Ihrer Lösung jedoch empfohlen. Alle zur Bewertung dieser Hausübung relevanten Deklarationen von Klassen, Methoden (hierzu zählen auch Konstruktoren) und Attributen sind bereits in der Quelltext-Vorlage enthalten und dürfen nicht modifiziert oder entfernt werden. Ihnen steht aber frei, Hilfskonstrukte in Form von weiteren Klassen, Methoden und Attributen zu erstellen, sofern dies nicht explizit auf dem Übungsblatt verboten wurde und Ihre Hilfskonstrukte nicht gegen verbindliche Anforderungen verstoßen. Datenstrukturen und Hilfsmethoden aus der Java-Standardbibliothek sowie Arrays sind nicht erlaubt, sofern dies nicht explizit auf dem Übungsblatt gefordert oder erlaubt wurde. Ihre Methoden müssen auch dann funktionieren, wenn Aufrufe von in der Vorlage deklarierten Methoden (auch von solchen, welche von Ihnen implementiert werden) durch andere, korrekte Implementationen ersetzt werden.

Der Verstoß gegen verbindliche Anforderungen führt zu Punktabzügen und kann die korrekte Bewertung Ihrer Abgabe unter Umständen beeinflussen. Die Implementation einer in der Quelltext-Vorlage deklarierten Methode wird nur bewertet, wenn der mit `TODO` markierte Exception-Wurf entfernt wird.

Hinweise für alle Hausübungen:

Die zu verwendenen Zugriffsmodifizierer sind in der Vorlage bereits gegeben und werden auf dem Übungsblatt nicht immer angegeben. Beachten Sie die Informationen im Moodle-Abschnitt *Technisches und Probe-Übungsblatt*.

Bei Fragen stehen wir Ihnen vorzugsweise im Moodle-Kurs und in den Sprechstunden zur Verfügung.

Die für diese Hausübung in der Vorlage relevanten Verzeichnisse sind `src/main/h03` und `src/test/h03`.

Sehen Sie sich den nachfolgenden Algorithmus aus der Vorlesung genau an, um die Details der Implementierung zu verstehen und umsetzen zu können: Auf diesem Übungsblatt implementieren Sie den Algorithmus „String Matching Based on Finite Automaton“ mit einem hohen Grad an Allgemeinheit.[1] Außerdem finden Sie im Forum zu Hausübung 3 eine Erklärung zum Verständnis des Algorithmus, die Sie sich ansehen können, sollten Sie Verständnisprobleme zu dem Zusammenhang der Methoden/Klassen haben.

[1] In der Informatik wird gerne der Begriff „generisch“ für „allgemein“ verwendet. Das darf natürlich nicht mit „generisch“ im Sinne von Java Generics und ähnlichen Konzepten in anderen Programmiersprachen verwechselt werden. Generics sind ***eine*** Möglichkeit, um einen hohen Allgemeinheitsgrad zu erreichen. Siehe Kapitel 15 der FOP für eine systematischere Aufarbeitung unter dem Stichwort *Polymorphie*.

H1: Abstrakter Umgang mit Alphabeten

Analog zu den Interfaces in `java.util.function` existiert in der Vorlage ein generisches Interface `FunctionToInt` mit generischem Typparameter `T`, welches die folgenden Methoden besitzt: Die parameterlose Methode `sizeOfAlphabet` liefert `int` zurück, die Methode `apply` hat einen Parameter vom formalen Typ `T` und liefert ebenfalls `int` zurück. Im Gegensatz zu den Interfaces in `java.util.function` wirft die Methode `apply` von `FunctionToInt` potentiell eine Exception, und zwar vom Typ `IllegalArgumentException`.

Der wesentliche Punkt im Vertrag der Methode `apply` ist, dass `apply` entweder eine nicht-negative Zahl[2] echt kleiner `sizeOfAlphabet` zurückliefert (immer dieselbe für denselben aktualen Parameter) oder eine `IllegalArgumentException` wirft.

Sie müssen dieses Interface nicht bearbeiten, jedoch werden Sie es in den folgenden Aufgaben verwenden.

H2: Konkrete Alphabete mit Unicode-Zeichen 4 Punkte

Implementieren Sie die Inhalte der folgenden zwei nicht-generischen Klassen, die das Interface `FunctionToInt` mit `Character` als Instanziierung des generischen Typparameters implementieren.

Klasse `UnicodeNumberOfCharIndex`: Methode `apply` liefert einfach den Unicode-Wert des im aktualen Parameter eingekapselten `char`-Wertes zurück und wirft nie eine Exception. Den Rückgabewert von `sizeOfAlphabet` finden Sie im Prinzip bei Klasse `Character`, aber Achtung: beachten Sie die Kapitel 12, Folien 81- der FOP

Klasse `SelectionOfCharsIndex`: Ein Objekt dieser Klasse hat eine `private`-Objektkonstante `theChars` vom Typ „Array von `char`“. Der Konstruktor hat einen Parameter `theAlphabet` vom formalen Typ „`List` von `Character`“. Der wesentliche Punkt im Vertrag zum Konstruktor ist, dass `theAlphabet` nicht `null` ist und der aktuelle Parameter mindestens ein Listenelement enthält, aber kein `char`-Wert mehr als einmal in `theAlphabet` vorkommt. Der Konstruktor richtet `theChars` so ein, dass `theChars` dieselben Elemente wie der aktuale Parameter in derselben Reihenfolge und keine weiteren hat.

Die Methode `sizeOfAlphabet` liefert die Länge des Arrays zurück. Die Methode `apply` liefert den Index ihres aktualen Parameters in `theChars` zurück bzw. wirft eine Exception, falls der aktuale Parameter nicht in `theChars` enthalten ist.

H3: Konkrete Alphabete mit Enumerationen 3 Punkte

Die einzelnen Konstanten einer Enumeration sind auf natürliche Weise die „Zeichen“ eines Alphabets und liefern durch die Reihenfolge in der Definition der Enumeration eine Indizierung der „Zeichen“ dieses Alphabets. Bevor Sie mit H3 loslegen, schauen Sie vielleicht noch einmal in die Behandlung von Enumerationen in Kapitel 03b der FOP (Folien 15 - 20), in die Dokumentation von *Oracle* zu Klasse `Enum` und in das *Oracle*-Tutorial „Enum Types“.

Implementieren Sie die generische Klasse `EnumIndex` mit generischem Parameter `T`, die das Interface `FunctionToInt` mit Typ T implementiert, wobei `T` auf Enumerationen eingeschränkt ist.

Sie besitzt einen Konstruktor mit Signatur `public EnumIndex(Class<T> enumClass)`, der einer `private` Objektkonstante mit Namen `enumArray` und Typ `T[]` die Konstanten des übergebenen Enums einfügt. Das Array soll nur aus diesen Konstanten bestehen und vollständig gefüllt sein.

[2] Wir bezeichnen eine Zahl als *nicht-negativ*, wenn sie größer oder gleich 0 ist.

Hinweis:

Ein Blick in die Java-Dokumentation der Methoden von `Class` wird Ihnen helfen. Methoden von `Class` und `Enum` dürfen Sie hier nutzen.

Methode `sizeOfAlphabet` liefert die Größe (also Anzahl Konstanten in) der Enumeration zurück. Methode `apply` liefert den Index des aktualen Parameters in seiner Enumeration zurück und wirft nie eine Exception.

H4: Abstrakte Tabelle für String Matching BOFA — 4 Punkte

Im Video / Foliensatz zum Algorithmus String Matching BOFA sehen Sie, dass der Suchstring gar nicht mehr explizit vorkommt, sondern vor der eigentlichen Suche nach Treffern in einer zusätzlichen Datenstruktur verarbeitet worden ist. Sie sehen zwei verschiedene Darstellungsformen: als Matrix und als Netzwerk. In H5 und H6 realisieren Sie diese beiden Möglichkeiten. Und hier, in H4, faktorisieren Sie die Gemeinsamkeiten heraus.

Konkret setzen Sie eine generische abstrakte Klasse `PartialMatchLengthUpdateValues` mit generischem Typparameter `T` um. Ein Objekt dieser Klasse hat ein `protected`-Attribut vom Typ „`FunctionToInt` von `T`“. Der Konstruktor hat einen Parameter vom selben formalen Typ und initialisiert das Attribut mit dem Parameter des Konstruktors wie üblich.

Diese Klasse hat eine abstrakte `public`-Methode `getPartialMatchLengthUpdate`, die einen Parameter vom formalen Typ `int` und einen Parameter vom formalen Typ `T` hat und `int` zurückliefert.

Daneben hat Klasse `PartialMatchLengthUpdateValues` eine implementierte `protected`-Methode `computePartialMatchLengthUpdateValues` mit einem Parameter `searchString` vom formalen Typ „Array von `T`“ und Rückgabetyp `int`. Diese Methode liefert die größte Zahl k zurück, so dass die ersten k Komponenten von `searchString` gleich den letzten k Komponenten von `searchString` in derselben Reihenfolge sind. Damit muss natürlich $k <$ `searchString.length` gelten, da $k =$ `searchString.length` bereits keinen Sinn mehr ergeben würde (Warum wäre dies bei jedem Suchstring erfüllt?). Formal (mit `a` für `searchString` zur Abkürzung):

$$(\texttt{a[0]}, \ldots, \texttt{a[}k\texttt{-1]}) = (\texttt{a[a.length-}k\texttt{]}, \ldots, \texttt{a[a.length-1]}).$$

Die Gleichheit zweier Objekte aus dem Array soll über die Gleichheit der Rückgabe der Funktion der Klasse - angewendet auf die beiden jeweiligen Objekte - geprüft werden. Bei einer gleichen Rückgabe, sind die Objekte als gleich anzusehen.

Hinweis:

Die Methode `computePartialMatchLengthUpdateValues` kann einfach jedes k testen und das größte nehmen, das die obige Bedingung erfüllt.

H5: Tabellenimplementationen für String Matching BOFA als Matrix — 7 Punkte

Implementieren Sie die generische Klasse `PartialMatchLengthUpdateValuesAsMatrix<T>`, die von `PartialMatchLengthUpdateValues<T>` direkt abgeleitet ist. Ein Objekt dieser Klasse hat ein `private`-Attribut vom Typ „Array von Array von `int`“. Dieses Attribut realisiert die Lookup-Table als Matrix, das heißt, ein Array mit Länge gleich der Größe des Alphabets, dessen Komponenten Arrays der Länge des Suchstrings plus 1 sind (oder auch umgekehrt, aber „plus 1“ nicht vergessen!).

Der Konstruktor hat einen Parameter vom formalen Typ `FunctionToInt<T>`. Der zweite Parameter ist der Suchstring, also vom Typen `T[]`. Der Konstruktor richtet die Lookup-Table auf Basis der beiden aktualen Parameterwerte ein und initialisiert die Einträge mit Hilfe von `computePartialMatchLengthUpdateValues`.

Methode `getPartialMatchLengthUpdate` liefert einfach den Eintrag in dieser Matrix zurück, der durch die beiden aktualen Parameterwerte spezifiziert ist.

H6: Tabellenimplementationen für String Matching BOFA als Automat — 7 Punkte

In H6 ist mit „Liste“ wieder durchgängig `java.util.List` gemeint und natürlich dürfen Sie eine Liste initialisieren, sowie die Methoden `add` und `remove` dieser Liste nutzen.

In der Vorlage finden Sie eine generische Klasse `PartialMatchLengthUpdateValuesAsAutomaton<T>`, die von `PartialMatchLengthUpdateValues<T>` direkt abgeleitet ist. Diese ist völlig analog zu `PartialMatchLengthUpdateValuesAsMatrix` aus H5, außer dass die Lookup-Table im `private`-Bereich anders realisiert ist.

Konkret ist sie realisiert als der Automat, der auf den Folien zu String Matching BOFA quasi mitläuft (Folien/Video der AuD zu String Matching). Die einzelnen Kreise mit Zahlen drin heißen *Zustände* (*states*), die Pfeile *Übergänge* (*transitions*).

Realisieren Sie diesen Automaten als ein Array `theStates`, das für jeden Zustand eine Komponente hat. Der Komponententyp von `theStates` ist „Liste von `Transition` von `T`“, wobei „`Transition`“ eine Klasse mit zwei `public`-Attributen ist: eine Objektkonstante vom Typ `int` mit einem Wert im Indexbereich von `theStates` und eine Objektkonstante vom Typ „Liste von `T`“. Der Konstruktor von `Transition` soll die Werte wie gewohnt über seine aktualen Parameter zuweisen - Dies müssen Sie noch implementieren, aktuell werden Standardwerte zugewiesen.

Für jede Transition $i \longrightarrow j$ mit $j \neq 0$ soll `theStates[i]` genau ein Listenelement enthalten, dessen `int`-Attribut gleich j ist (und weitere Listenelemente soll `theStates[i]` nicht enthalten). Die „Liste von `T`“ in diesem `Transition`-Objekt enthält alle Zeichen `t` von `T`, die bei Zustand i und nächstem Zeichen `t` zu Zustand j überleiten. Alle Zeichen `t` in `T`, die nicht auf diese Weise in `theStates[i]` vorkommen, sind implizit als Transition $i \longrightarrow 0$ zu interpretieren, das heißt, bei Zustand i leitet `t` zu Zustand 0 über.

Verbindliche Anforderung:

Der Inhalt von `theStates` darf nicht zweischrittig über eine andere Datenstruktur erstellt werden, das heißt, es darf nicht erst eine andere Datenstruktur für den Automaten aufgebaut werden, um daraus dann die Datenstruktur für H6 zu erstellen. Insbesondere darf die Datenstruktur aus H5 nicht in H6 verwendet werden.

Unbewertete Verständnisfrage:

Warum den Fall $j = 0$ auslassen? Unter welchen Umständen ist das ein großer Vorteil, und welcher Art ist dieser Vorteil?

H7: Algorithmus String Matching BOFA — 7 Punkte

Außerdem implementieren Sie die generische Klasse `StringMatcher` mit einem generischen Typparameter `T`. Diese Klasse hat eine `private`-Objektkonstante vom statischen Typ „`PartialMatchLengthUpdateValues` von `T`“. Der

`public`-Konstruktor hat einen Parameter, der das Attribut in der üblichen Weise initialisiert (hat insbesondere den entsprechenden formalen Typ).

Die `public`-Objektmethode `findAllMatches` hat einen Parameter `source` vom formalen Typ „Array von `T`“. Rückgabetyp ist „Liste von `Integer`“ (`List`). Auch hier dürfen Sie eine solche Liste natürlich initialisieren und die Methode `add` nutzen, um Elemente hinzuzufügen. In dieser Methode nun wird der auf einem endlichen Automaten basierende Algorithmus für String Matching realisiert. Konkret ist `source` der String, in dem gesucht wird. Der String, dessen Vorkommen in `source` gesucht werden soll, ist nicht explizit gegeben, sondern schon im `PartialMatchLengthUpdateValues`-Attribut verarbeitet worden. Wird `null` als `source` übergeben, so soll eine leere Liste zurückgegeben werden.

Hinweis:

Die Länge des Suchstrings wird in der aktuellen Implementierung nicht gespeichert, jedoch für den Automaten benötigt. In der abstrakten Klasse PartialMatchLengthUpdateValues existiert noch eine abstrakte Methode `public int getSearchStringLength()`, die dann in den Unterklassen jeweils von Ihnen zu implementieren ist.

H8: Test des Algorithmus

Verbindliche Anforderungen für alle Hausübungen:

Auch wenn Sie für das Erstellen von Tests keine Punkte erhalten: Das Teilen jeglicher Tests zu Hausübungen ist *nicht* erlaubt. Unser Ziel ist, die Verbreitung fehlerhafter Tests in Ihrem Sinne zu verhindern. Alle auf diesem Übungsblatt genannten verbindlichen Anforderungen gelten für diese Aufgabe *nicht*. Insbesondere empfiehlt es sich sogar, zum Testen Datenstrukturen und Hilfsmethoden aus der Java-Standardbibliothek zu verwenden.

Hinweise für alle Hausübungen:

Die von uns bereitgestellten Public Tests überprüfen nur einen kleinen Teil Ihrer Implementation. Erfüllt Ihre Lösung nicht alle Public Tests, erhalten Sie auf keinen Fall die volle Punktzahl. Im Umkehrschluss bedeutet dies aber nicht, dass Sie die volle Punktzahl erhalten, wenn Ihre Lösung alle Public Tests besteht.

Der folgende Leitfaden dient als Unterstützung zum Aufbau Ihrer eigenen Tests. Sie können vom Leitfaden abweichen und dabei mindestens genauso aussagekräftige Testergebnisse erzeugen.

Testen Sie den Algorithmus mit allen drei Alphabet-Klassen. Sie können sich eine beliebige Enumeration ausdenken.

Dabei sollte bei jedem der beiden beteiligten Strings (der gesuchte und der, in dem gesucht wird) mindestens einmal der Fall auftreten, dass dieser String leer ist.

Bei zwei nichtleeren Strings sollte mindestens einmal der Fall auftreten, dass der Suchstring im anderen String gar nicht gefunden wird, und mindestens einmal der Fall, dass der Suchstring sehr häufig gefunden wird.

Bei Klasse `SelectionOfCharsIndex` behandeln Sie zusätzlich den Fall, dass einmal der Suchstring, zum anderen der String, in dem gesucht wird, ein Zeichen enthält, das in der Auswahl nicht enthalten ist, und somit eine Exception geworfen wird.