Übungsblatt 3

Abgabe bis: 28.05.2017

Aufgabe 1 Mengen mit selbstanordnenden Listen (90%)

Ihr findet bei Stud.IP ein gepacktes Eclipse-Projekt *PI2Set*. Ihr findet darin eine Teilimplementierung der Klasse `PI2Set<E>`, die eine Menge im mathematischen Sinne realisieren soll. Fügt als Datenstruktur für die Speicherung der Elemente eine selbstimplementierte, einfach verkettete Liste ein (vgl. `ListElem` aus der Vorlesung). Da das Testen, ob ein Element in der Menge vorkommt, die zentrale Operation ist, sollt ihr hierfür verschiedene Varianten so genannter *selbstanordnender Listen* ausprobieren. Die Idee dabei ist, dass man nach dem erfolgreichen Finden eines Elements die Liste so umordnet, dass dieses Element das nächste Mal schneller gefunden wird.

Folgende Strategien für die Selbstanordnung der Liste müssen implementiert werden:

Naiv. Die Reihenfolge der Elemente der Liste wird nicht verändert. Neue Elemente werden an den Anfang eingefügt. Hierfür wurde eine Implementierung bereits begonnen, die ihr in der Klasse `PI2SetNaive<E>` findet.

MF-Regel (*Move-to-front*). Wie *Naiv*, aber jeder gefundene Eintrag wird entnommen und an den Anfang der Liste gestellt.

T-Regel (*Transpose*). Wie *Naiv*, aber jeder gefundene Eintrag wird mit seinem Vorgänger vertauscht, also eine Stelle näher zum Listenanfang verschoben.

FC-Regel (*Frequency Count*). Das Einfügen bzw. Wiedereinfügen passiert entsprechend der Häufigkeit der Suchanfragen, d.h. häufig gesuchte Elemente stehen dichter am Anfang. Hierzu wird ein Zähler in jedem Listenknoten benötigt, der bei jedem erfolgreichen Fund eines Elements erhöht wird.[1]

Dinge, die allen Strategien gemeinsam sind, sollen der Klasse `PI2Set<E>` hinzugefügt werden. Dinge, in denen sich die Strategien voneinander unterscheiden, müssen in abgeleiteten Klassen implementiert werden. Dazu gibt es drei z. T. abstrakte Methoden in `PI2Set<E>`:

emptySet: Diese Methode wird von abgeleiteten Klassen implementiert, um eine neue, leere Menge ihres eigenen Typs zu liefern.

contains: Diese Methode prüft, ob ein Element in der Menge enthalten ist. Wenn es gefunden wird, wird es nun jedoch möglicherweise in der Liste verschoben.

addNew: Fügt ein neues Element in die Liste ein. Wird von *add* immer erst nach einem erfolglosen Aufruf von *contains* ausgeführt.

[1] Der Zähler darf in allen Listenknoten vorhanden sein, auch wenn er für die anderen Strategien nicht benötigt wird.

Erzeugt pro Strategie eine Testklasse. `PI2SetNaiveTest<E>` demonstriert, wie so etwas aussehen kann, damit auch immer alle vorhandenen Tests für `PI2Set` mit ausgeführt werden. Eure eigenen Tests müssen lediglich überprüfen, ob die Strategien richtig umgesetzt wurden. Die Reihenfolge der Mengenelemente könnt ihr über den Iterator prüfen.

Aufgabe 2 Was bringt's? (10%)

Ihr findet bei Stud.IP die Datei *ingenieur.txt*, in der sich ein aus verschiedensten Gründen lesenswerter Text befindet. Kopiert die Datei in euer Projektverzeichnis. Vervollständigt dann die Klasse `StrategyCheck`, so dass die Wörter des Textes in je eine Menge pro implementierter Selbstanordnungsstrategie eingefügt werden. Nutzt dafür als Elementtyp der Mengen den `CountingString`, der im bereitgestellten Eclipse-Projekt enthalten ist. Für eine Menge vom Typ `PI2SetNaive` ist das bereits in der Methode *check()* vorgegeben. Die Häufigkeit der Ausführung von *equals* – also die Anzahl der Vergleiche – wird durch den `CountingString` mitgezählt. Lasst diese daher auf der Konsole ausgeben. Welche Strategie ist die beste?