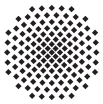


Universität Stuttgart

Übungsblatt 5

Datenstrukturen und Algorithmen (SS 2022)

Abgabe: Montag, 23.05.2022, 15:30 Uhr — Besprechung: ab Montag, 30.05.2022

Abgabevorschriften: Die Aufgaben auf diesem Blatt sind unter Einhaltung der Abgabevorschriften[1] zu lösen und abzugeben.

Lernziele: Nach dem Tutorium zu diesem Blatt sollten Sie folgende Lernziele erreicht haben. Wenn nicht, zögern Sie nicht, ihre:n Tutor:in anzusprechen um die Lücken zu füllen, Unklarheiten zu klären oder Fragen zu beantworten.

- Sie können Präfix-Bäume, Tries und Patricia-Bäume erkennen, erstellen, updaten und in Java implementieren.
- Sie können Worte in Präfix-Bäumen, Tries und Patricia-Bäume suchen.
- Sie können unterschiedliche Arten von Digitalbäumen ineinander umwandeln.
- Sie kennen die Unterschiede und Vor-/ Nachteile zwischen Präfix-Bäumen, Tries und Patricia-Bäume und können Sie in einem passenden Szenario als geeignete Datenstruktur auswählen.
- Sie können Heaps erkennen, aufbauen, updaten und in Java implementieren.
- Sie wissen wie HeapSort funktioniert, kennen die Vor- und Nachteile und können es in Java implementieren.
- Sie können TreeSets in Java erstellen und verwenden.

Punkte: Dieses Übungsblatt beinhaltet 5 Aufgaben mit einer Gesamtzahl von 30 Punkten. Zum Bestehen werden also 15 Punkte benötigt.

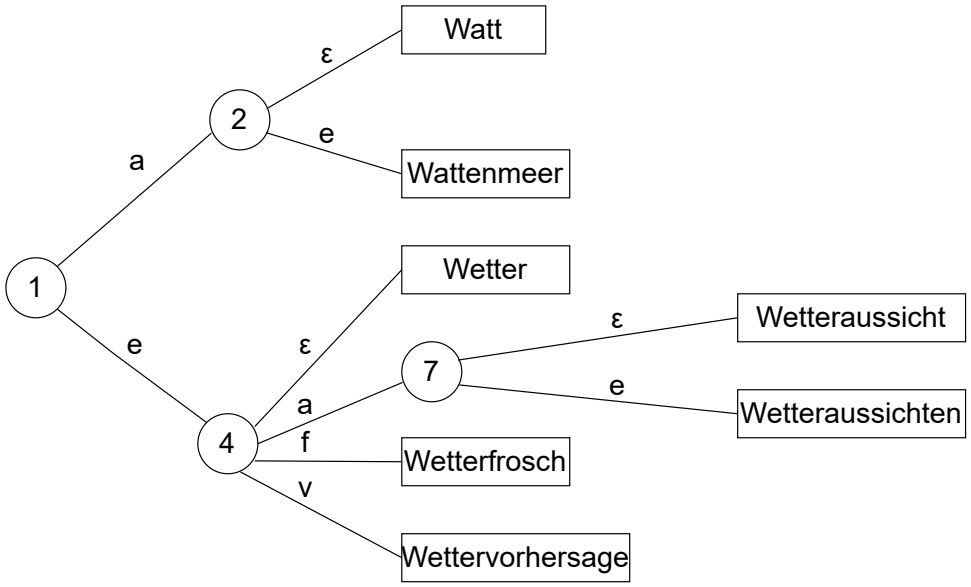
Aufgabe 1 Grundlagen von Tries, Präfix- und Patricia-Bäumen *[Punkte: 8]*

In dieser Aufgabe werden Sie mit den in der Vorlesung vorgestellten Digitalbäumen arbeiten und auch einige davon erstellen.

Hinweis: Wie Sie bei dem in Teilaufgabe a) dargestellten Baum sehen können, kann es bei Patricia- und Präfix-Bäumen vorkommen, dass beim Einfügen eines Worts der Präfix des einen Wortes einem kompletten weiteren Wort entspricht, z.B. *Wattenmeer* und *Watt*. Fügen Sie dann ϵ-Kanten und ϵ-Knoten (nur bei Präfix-Bäumen notwendig) ein, um das Ende eines Wortes ohne verbleibenden Suffix zu markieren, wie in Teilaufgabe a) gezeigt.

(a) *(2 Punkte)* Gegeben sei der folgende *Patricia-Baum*. Konvertieren Sie diesen in einen äquivalenten *Trie*. Orientieren Sie sich dabei an der Darstellung, die Sie in der Vorlesung kennengelernt haben.

[1] `https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_file_2904210.html`



(b) *(2 Punkte)* Gegeben sei die folgende Menge von Wörtern: *Kaktus*, *Kakadu*, *Katamaran*, *Kuchen*, *Kartoffelsalat*, *Kugelschreiber* und *Kartoffelsuppe*. Zeichnen Sie den resutltierenden *Patricia-Baum* für die gegebene Menge von Wörtern. Orientieren Sie sich dabei an der Darstellung, die Sie in der Vorlesung kennengelernt haben.

(c) *(2 Punkte)* Gegeben sei die folgende Menge von Wörtern: *Algorithmik*, *Algorithmen*, *Alien*, *Alarm*, *Algorithmenentwurf*, *All* und *Algorithmenanalyse*. Zeichnen Sie den resutltierenden *Präfix-Baum* für die gegebene Menge von Wörtern. Orientieren Sie sich dabei an der Darstellung, die Sie in der Vorlesung kennengelernt haben.

(d) *(1 Punkt)* Nennen Sie zwei Nachteile von *Tries* und begründen Sie diese.

(e) *(1 Punkt)* Beschreiben Sie ein Szenario, in welchem die Verwendung eines *Digitalbaums* sinnvoll ist.

Aufgabe 2 Impl Präfix-Baum Implementieren *[Punkte: 8]*

In Aufgabe 1 haben Sie sich bereits auf theoretischer Ebene mit *digitalen Bäumen* auseinandergesetzt. In dieser Aufgabe sollen Sie nun einen *Präfix-Baum* in Java implementieren. Hierfür sind im Eclipse-Projekt folgende Codefragmente gegeben:

- Schnittstelle `IPrefixTreeNode` und unvollständige Klasse `PrefixTreeNode`
- Schittstelle `IPrefixTree` und unvollständige Klasse `PrefixTree`

(a) *(3 Punkte)* Implementieren Sie zunächst die Knotenklasse `PrefixTreeNode`, die die Schnittstelle `IPrefixTreeNode` implementiert.
Hinweis: Zum Verwalten der ausgehenden Kanten eines Knoten bietet sich eine `HashMap` an, welche einem `String` einen `PrefixTreeNode` zuordnet.

(b) *(5 Punkte)* Implementieren Sie nun die Klasse `PrefixTree`, die die Schnittstelle `IPrefixTree` implementiert.
Hinweise:

- Sie können in dieser Aufgabe davon ausgehen, dass keine Worte eingefügt werden, die ϵ-Übergänge oder ϵ-Knoten erfordern, wie in Aufgabe 1 beschrieben.
- Beachten Sie für `insert` den Spezialfall, falls ein Wort in einen leeren Baum eingefügt wird.
- Die Methode `insert` lässt sich rekursiv realisieren.
- Hilfsmethoden (z.B. zum Finden der Länge des längsten Präfixes zweier Worte) können sinnvoll sein.
- Lesen Sie sich die Dokumentation der Methode `String.substring` durch (`https://docs.oracle.com/javase/7/docs/api/java/lang/String.html`). Die Verwendung dieser Methode kann bei der Implementierung des Baumes hilfreich sein.

Aufgabe 3 Heaps und HeapSort verstehen *[Punkte: 7]*

In der Vorlesung haben Sie sich in erster Linie mit *Min-Heaps* auseinandergesetzt. Analog dazu soll es in dieser Aufgabe um *Max-Heaps* gehen.

(a) *(1 Punkt)* Erklären Sie inwiefern sich ein *Max-Heap* von einem *Min-Heap* unterscheidet.

(b) *(3 Punkte)* Konstruieren Sie aus der Folge $5, 11, 16, 2, 12, 20, -10$ den *Max-Heap* B_1. Passen Sie dafür den Algorithmus für *Min-Heaps* aus der Vorlesung für *Max-Heaps* an. Zeichnen Sie den *Max-Heap* in Baum- und Array-Form zu Beginn und nach jedem Schritt, markieren Sie die Elemente, die Sie einsinken lassen und erklären Sie Ihr Vorgehen.

(c) *(2 Punkte)* Entfernen Sie aus B_1 zweimal das Maximum und stellen Sie die Heap-Eigenschaft wieder her. Zeichnen Sie den *Max-Heap* in Baum- und Array-Form zu Beginn und nach jedem Schritt, markieren Sie die Elemente, die Sie einsinken lassen und erklären Sie Ihr Vorgehen.

(d) *(1 Punkt)* Vergleichen Sie *HeapSort* mit *QuickSort*. Betrachtet man die asymptotische Komplexität der beiden Algorithmen könnte man zu dem Schluss kommen, dass *QuickSort* überflüssig ist. Ist diese Folgerung korrekt? Diskutieren Sie.

Aufgabe 4 Impl Heapsort *[Punkte: 5]*

Sie haben in der Vorlesung mit *HeapSort* einen neuen Algorithmus kennengelernt, um den es in dieser Aufgabe gehen soll.

Implementieren Sie HeapSort als statische Methode der Klasse `Sorter`. Orientieren Sie sich an dem in der Vorlesung gezeigten Algorithmus. Die Methode soll eine Liste, die das `ISimpleList` Interface implementiert, als Eingabeparameter entgegennehmen. Der Algorithmus soll die Liste in aufsteigender Reihenfolge unter Verwendung eines *Max-Heap* sortieren.

Aufgabe 5 TreeSets *[Punkte: 2]*

Eine praktische Anwendung von Bäumen sind *TreeSets*. In der Vorlesung wurde bereits die Implementierung der Methode `unite` vorgestellt. Geben Sie analog dazu die Java-Implementierung der Methode `intersect` an, welche für zwei *TreeSets* die Schnittmenge berechnet und zurückgibt. Geben Sie Ihren Code hierfür ausnahmsweise auch in der `.pdf` ab und erklären Sie Ihre Implementierung.

Hinweis: Sie können bei Ihrer Implementierung davon ausgehen, dass die Methoden `size`, `add`, `contains`, `unite` sowie `iterator` gemäß den Vorlesungsfolien bereits implementiert sind und Sie diese bei Bedarf verwenden dürfen.