Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger
Frühjahrssemester 2018

Universität Basel
Fachbereich Informatik

Übungsblatt 12

Abgabe: 01. Juni 2018

Bei diesem Übungsblatt handelt es sich um ein Bonusblatt. Das heisst, die Punkte, die sie sich in diesem Übungsblatt erarbeiten, werden zu ihrer Gesamtpunktzahl aus allen Übungsblättern dazugezählt. Die benötigten 50% der Punkte aus allen Übungen werden aber nur anhand der Übungsblätter 1-11 berechnet.

Als Grundlage für die praktischen Übungen verwenden Sie das Projekt im Zip-File "blatt12.zip", welches Sie auf dem Adam-Workspace finden. Die Lösungen zu den Theoriefragen geben Sie als Dokument im PDF Format, mit dem Namen `loesung-12.pdf` ab. Bitte fügen sie diese PDF Datei zur Projektstruktur hinzu. Laden Sie Ihre Lösungen als zip Datei auf Courses hoch.
Das Kompilieren und Ausführen der Testfälle erfolgt wie in den vorhergehenden Übungen durch `gradlew build` respektive `gradlew test`.

Aufgabe 12.1 (Einfügen in Tries, 1 Punkt)

- Zeichnen Sie den Trie den Sie erhalten, wenn die Schlüssel `this is an easy task` in genau dieser Reihenfolge in einen anfänglich leeren Trie eingefügt werden.
- Zeichnen sie den entsprechenden ternären Suchtrie.

Aufgabe 12.2 (Nichtrekursive Implementation von Tries, 2 Punkte)

Schreiben Sie die Methoden `put` und `get` in `exercise12.TrieST`, so um, dass keine Rekursion mehr verwendet wird. Testen Sie ihre Implementation mithilfe der Unittests.

Aufgabe 12.3 (Ordnungsoperationen für Tries, 2 Punkte)

Implementieren Sie die Methode `findMinimalKeyWithPrefix` in `exercise12.TrieST`, welche den kleinsten Schlüssel im Trie mit gegebenem Präfix zurückgibt. Testen Sie Ihre Lösung mithilfe der entsprechenden Unittests.

Aufgabe 12.4 (Präfixfreie codes, 1 Punkt)

Betrachten Sie die vier Codes variabler Länge in der nachfolgenden Tabelle. Welche der Codes sind präfixfrei? Eindeutig decodierbar? Geben Sie für die eindeutig decodierbaren Codes die Codierung von 0001001011 an.

Symbol	Code 1	Code 2	Code 3	Code 4
A	1	0	0	1
B	01	100	1	01
C	001	10	00	001
D	000	11	11	0001

Aufgabe 12.5 (Decodierbarkeit, 1 Punkt)

Sind $\{01, 1001, 1011, 111, 1110\}$ und $\{1, 011, 01110, 1110, 10011\}$ eindeutig decodierbar? Wenn nicht, finden Sie einen String mit zwei Codierungen.

Aufgabe 12.6 (Codierung, 3 Punkte)

Wie sieht der Code für die Strings a , aa , aaa , aaaa , ... (bestehend aus N a's) aus, der mit der Lauflängen-, Huffman- und LZW-Codierung erstellt wurde. Was können Sie über die Länge von den komprimierten Strings im Verhältnis zu N sagen. Tip: Nutzen sie dazu die Implementationen im Jupiter Notebook `compression.ipynb`

Aufgabe 12.7 (Suffixarrays, 3 Punkte)

Lesen sie das Kapitel zum Thema Suffixarrays, welches Sie auf dem Adam workspace `suffix-arrays.pdf` finden. Beantworten Sie dazu folgende Fragen:

- Fassen Sie in 2-3 Sätzen die Idee von Suffixarrays zusammen.
- In welchem Fall weist das im Buch beschriebene, einfache Verfahren, eine schlechte Laufzeit auf?
- Erklären Sie, was die Methoden `select` und `index` im Suffixarray API machen.

Aufgabe 12.8 (Suffixarray client (2 Punkte))

Implementieren Sie den SuffixArray-Client `exercise12.SuffixArrayClient`, der zwei String liest und in linearer Zeit den längsten Teilsstring findet, der in beiden Strings vorkommt. Testen Sie ihre Implementation mithilfe der Unit tests. Die Implementation eines Suffixarrays finden sie in `edu.princeton.cs.algs4.SuffixArray`.
Hinweis: Erzeugen Sie ein Suffixarray für s#t , wobei s und t die beiden Textstrings sind und # ein Zeichen ist, das in keinem von ihnen erscheint.

Die Übungsblätter dürfen in Gruppen von zwei Studierenden bearbeitet werden. Bitte schreiben Sie beide Namen auf Ihre Lösung.