Datenstrukturen
Übung – SS2020

Abgabe: 31.05. bis 24Uhr.
Abgabeort: Bitte laden Sie Ihre Klasse im OPAL-Bereich "Abgabe_Pflichtaufgabe_4" hoch. Ergänzen Sie bei Ihrer hochgeladenen Klasse die Namen (Vor- und Nachname) aller beteiligten Personen. Sie können Ihre Abgabe bis zur Abgabefrist beliebig oft löschen und erneut einreichen.

Wichtig: Da es sich um eine Prüfungsvorleistung handelt, ist es wichtig, dass jeder Teilnehmer Zugang zu allen notwendigen Informationen hat. Deswegen bitten wir Sie Fragen direkt ins Forum, in den Thread "Fragen zur Pflichtaufgabe 4" zu stellen.

Packen Sie Ihre Klasse in das Package "PVL4_$GroupX". Sollten Sie nicht in einer Gruppe arbeiten, wählen Sie den Namen statt "PVL4_$Nachname_$Vorname".
Ihre Klasse soll den Namen "PVL4_$GroupX" bzw. "PVL4_$Nachname_$Vorname" besitzen.

Für die Abgabe: Packen Sie all Ihre Klassen in ein .zip Archiv mit dem Namen "Huffman_GroupX.zip"
[Anmerkung: .rar ist kein .zip]

Prüfungsvorleistung 4 – Huffman-Code

Die Informatik ist die Wissenschaft der Informationsverarbeitung, -speicherung und -übermittlung.
Diese Informationsübermittlung ist in Zeiten von IoT ("Internet of Things") stets um uns. Jedoch ist dabei stets folgendes von größter Wichtigkeit: Der Sender und der Empfänger müssen die Daten auf dieselbe Weise verstehen können.
Hierfür gibt es diverse Kodierungen: Sie kennen bereits einige Kodierungen, wie ASCII-Codes, UTF-8, UTF-16, usw. oder auch Verschlüsselungen. Weiterhin kennen Sie Archive, wie .zip oder .tar, welche die Größe ihrer Daten verringern, es handelt sich hierbei um Datenkompression. In dieser PVL sollen Sie sich mit einer speziellen Methode der Datenkompression beschäftigen, welche stark auf Bäumen basiert, der Huffman-Kodierung (siehe https://de.wikipedia.org/wiki/Huffman-Kodierung).

Kurzum, die Huffman-Kodierung arbeitet auf einem gegebenem Alphabet Z, mit Buchstaben a aus Z. Jeder Buchstabe besitzt eine Ihm zugeordnete a-priori (relative) Häufigkeit. Bei diesem Algorithmus wird eine sogenannte "Greedy" Vorgehensweise benutzt. Bei dieser "verknüpft" man die Knoten mit der geringsten Wahrscheinlichkeit zu einem.
Die inneren Knoten im Baum beinhalten dabei nur die Daten der relativen Häufigkeiten. Die Blätter beschreiben den unkodierten Buchstaben. Der Weg zum Blatt beschreibt dessen Kodierung. Diese besteht aus dem Alphabet [0, 1].

Implementieren Sie das Interface "HuffmanCode".
(In Python benötigen Sie kein Interface, doch halten Sie sich an die folgenden Methoden-Signaturen).

Sollte zu einem Zeitpunkt irgendein Parameter "null" sein, geben Sie "null" zurück.

Die folgenden Methoden finden Sie im Interface "HuffmanCode", hier finden Sie eine Spezifikation für das Verhalten der Methoden, Beispiele folgen weiter unten:

Java: public PVL4_$GroupX(char[] alphabet, float[] probabilities)
Python: __init__(self, alphabet : [str], probabilities : [float])

Sie bekommen zwei Arrays der gleichen Länge. Die Summe über "probabilities" entspricht dabei 1.0. In alphabet finden Sie paarweise verschiedene Zeichen. Es gilt das probabilities[i] für das Zeichen alphabet[i]
gilt.

Konstruieren Sie aus dem gegebenen Alphabet und den gegebenen Wahrscheinlichkeiten den Huffman Kodierungs-Baum. Dabei soll Folgendes gelten: Gehen Sie von einem Knoten aus zu einem linken Kind, wird der Wert "1" gesetzt, gehen Sie zu einem rechten Kind, wird der Wert "0" gesetzt. Sollten zwei oder mehr Buchstaben die gleiche Wahrscheinlichkeit besitzen, so soll das alphabet[i] weiter links als alphabet[j] sein für jedes i<j mit probabilities[i] == probabilities[j].

Weiterhin soll sich das Element mit der höheren relativen Wahrscheinlichkeit stets links befinden. (Anmerkung: In Python befindet sich in einem str jeweils nur ein Element)

Java: public String getCodes()
Python: def getCodes() -> str
Geben Sie die Blätter Ihres Huffman-Kodierungs-Baumes aus, von links nach rechts, ein Blatt soll dabei wie folgt formatiert sein:

"$letter - $HuffmanCode"

Die Blätter sollen durch einen Zeilenumbruch voneinander getrennt sein. Also sieht das Format wie folgt aus.

"Leaf[0]\n
...
Leaf[n]\n"
Sollte es kein Blatt geben, geben Sie "" zurück.

Java: public String encode(String plainText)
Python: def encode(plainTexxt: str) -> str

Gegeben sei ein Klartext, in dem jeder Buchstabe nur aus dem gegebenen Alphabet stammt. Kodieren Sie den String mittels Ihres Huffman-Kodierungs-Baums. Geben Sie dementsprechend einen String bestehend aus "1" und "0" zurück, der den Plaintext wiederspiegelt.

Java: public String decode(String huffmanText)
Python: def decode(huffmanText: str) -> str

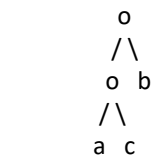
Gegeben sei ein Text in Huffman-Kodierung, dieser sei valide entsprechend des gegebenen Alphabets. Dekodieren Sie den String mittels ihres Huffman-Kodierungs-Baums.

Geben Sie dementsprechend einen String bestehend aus dem Alphabet zurück, der den Huffman-Text wiederspiegelt.

Beispiel:

```
char[] alphabet = new char[]{'a', 'b', 'c'};
float[] probs = new float[]{0.25f, 0.5f, 0.25f};
PVL4_$GroupX bsp = new PVL4_$GroupX(alphabet, probs);
```

Im Hintergrund existiert nun folgender Baum:



Somit sind die Kodierungen:
a - '11'
b - '0'
c - '10'

Zur Erklärung:
Die geringsten relativen Wahrscheinlichkeiten sind a und c, beide besitzen die selbe relative Wahrscheinlichkeit, und a war in "alphabet" vor c, also wird a als linkes Kind genommen und c als rechtes. Nun besitzt der zusammengefügte Knoten die relative Wahrscheinlichkeit 0,5 genau wie b. Da der zusammengesetzte Knoten (a,c) mit a vor b in "alphabet" war, wird dieser als linker Kindknoten benutzt und b als rechter Kindknoten.

Die Methode getCodes() gibt also den folgenden String zurück:

"a - 11\n
c - 10\n
b - 0\n"

Der Aufruf von encode("abbac") würde also den folgenden String zurückgeben:
"11001110"

Der Aufruf von decode("11001110") würde also den folgenden String zurückgeben:
"abbac"