Tries und Automatenminimierung

In dieser Woche sollen Sie eine Wortliste in einen Trie umwandeln und endliche Automaten mit dem Hopcroft-Algorithmus minimieren.

Tries (Buchstaben-Bäume)

Schreiben Sie ein Programm, welches eine **Liste von Wörtern** (z.B. die in `http://www.cis.uni-muenchen.de/~schmid/lehre/data/wortliste.txt`) aus einer Datei einliest und in einen Buchstabenbaum (Trie) umwandelt. Jede Kante von einem Knoten zu einem Tochterknoten ist mit einem Buchstaben beschriftet. Es ist nicht erlaubt, dass 2 (ausgehende) Kanten eines Knotens denselben Buchstaben tragen. Jeder Pfad vom Wurzelknoten des Baumes zu einem Terminalknoten soll einem Wort entsprechen und umgekehrt. Dies ist jedoch nicht gewährleistet, wenn ein Wort Präfix eines anderen Wortes sein kann. Daher sollen Terminalknoten explizit markiert werden, so dass auch Knoten mit Übergängen zu anderen Knoten Terminalknoten sein können.

Aufruf: `python build-trie.py wortliste.txt > trie.txt`

Der Buchstabenbaum soll als Liste von Kanten auf den Bildschirm ausgegeben werden. Jede Ausgabezeile enthält eine Kante bestehend aus dem Index des Startknotens, dem Buchstaben und dem Index des Zielknotens. Als Trennzeichen sollen Tabulatoren verwendet werden. Der Startknoten hat den Index 0. Am Ende geben Sie noch die Liste der Terminalknoten aus, indem Sie den Index jedes Terminalknotens auf einer separaten Zeile ausgeben.

Beispiel: Ein Trie für das Wort “ab”:

```
0	a	1
1	b	2
2
```

Schreiben Sie nun noch ein Programm, welches den Buchstabenbaum aus einer Datei einliest und dann Wörter Zeile für Zeile aus einer zweiten Datei einliest und prüft, ob sie im Buchstabenbaum (und damit in der ursprünglichen Wortliste) enthalten sind.

Aufruf: `python lookup.py trie.txt words.txt`

Jedes Eingabewort wird in einer separaten Zeile zusammen mit der Bewertung `known` oder `unknown` ausgegeben.

Vorüberlegungen

- Erstellen Sie von Hand den Baum für die Wortliste: reitet reicht riecht
- Welche Datenstruktur verwenden Sie für den Baum?
- Wie repräsentieren Sie die Information über Terminalknoten?

Minimierung endlicher Automaten

Der Buchstabenbaum aus der letzten Aufgabe bildet einen **deterministischen endlichen Automaten**, der Zeichenketten erkennen und generieren kann. Sie sollen nun solche Automaten **minimieren**, indem Sie äquivalente Zustände identifizieren und zusammenfassen. Zustände sind **äquivalent**, wenn sie jeweils dieselbe Menge von Strings erzeugen. Ein Knoten erzeugt einen String s, falls es einen Pfad von dem Knoten zu einem Endzustand gibt, dessen Folge von Kanten-Symbolen den String s ergibt.

Sie sollen für die Minimierung den **Hopcroft-Algorithmus** verwenden. Der Hopcroft-Algorithmus unterteilt zunächst die Menge Q aller Zustände (Baumknoten) in 2 Teilmengen: die Menge der Endzustände F und die Menge der Nicht-Endzustände $\bar{F}$. Endzustände und Nicht-Endzustände können nicht äquivalent sein, weil Endzustände den leeren String erzeugen, die anderen dagegen nicht. Die beiden Teilmengen von Q werden solange weiter unterteilt, bis jede Teilmenge nur noch äquivalente Zustände umfasst.

Eine Zustandsmenge X wird aufgeteilt, wenn sie zwei Zustände besitzt, von denen der eine einen Übergang mit irgendeinem Symbol a zu einem Zustand in einer Zustandsmenge Y besitzt und der andere Zustand keinen solchen Übergang mit a zu einem Zustand in Y besitzt. X wird dann von Y und a in zwei neue Zustandsmengen X_1 und X_2 aufgespalten.

Operation split: Eine Zustandsmenge X wird von einer Zustandsmenge Y und dem Symbol a in zwei Teilmengen X_1 und X_2 aufgespalten. Dabei ist X_1 die Menge der Zustände in X, die einen Übergang mit dem Symbol a zu irgendeinem Zustand in Y besitzen. X_2 ist die Menge aller Zustände aus X, für die das nicht gilt. Wenn weder X_1 noch X_2 leer ist, muss X in X_1 und X_2 geteilt werden.

Pseudocode für den Hopcroft-Algorithmus:

```
Hopcroft(A, F, F̄)
    # A: Alphabet
    # F: Menge der Endzustände
    # F̄: Menge aller anderen Zustände
    # P aktuelle Menge der Zustandsmengen
    # W To-Do-Liste mit Zustandsmengen
    P ← {F, F̄}        # eine Menge mit zwei Mengen als Elementen
    W ← {minimum(F, F̄)}        # Berechne die kleinere der beiden Mengen
    while W ≠ ∅ do
        Y ← pop(W)        # Nimm ein beliebiges Element von der Warteliste
        for each a ∈ A do        # für alle Symbole
            for each X ∈ P do        # für alle Zustandsmengen
                (X1, X2) ← split(X, a, Y)        # Spalte X auf
                if X1 ≠ ∅ and X2 ≠ ∅ then        # Ist keine der beiden Teilmengen leer?
                    P ← P/{X} ∪ {X1, X2}        # Ersetze X in P durch X1 und X2
                    if X ∈ W then
                        W ← W/{X} ∪ {X1, X2}        # Ersetze X in W durch X1 und X2
                    else
                        W ← W ∪ {minimum(X1, X2)}  # Füge die kleinere Teilmenge zu W hinzu
    return P
```

Die Funktion `minimum` gibt die Argumentmenge mit der kleinsten Zahl von Elementen

zurück. (Achtung: die Python-Funktion `min` ohne `key`-Argument vergleicht bei Sets nicht die Längen sondern die Elemente!)

Vervollständigung: Der Hopcroft-Algorithmus braucht einen **vollständigen** Automaten als Eingabe, d.h. von jedem Zustand muss es für jedes Symbol im Alphabet einen Übergang zu einem anderen Zustand geben. Das ist bei dem Buchstabenbaum nicht der Fall. Sie vervollständigen den Buchstabenbaum mit folgenden Schritten:

- Sie berechnen die Menge aller Buchstaben.
- Sie erzeugen einen neuen Zustand (den "Fehler"-Zustand).
- Sie fügen zu jedem Zustand für jeden Buchstaben, für den es noch keinen Übergang gibt, einen Übergang zum Fehler-Zustand hinzu. Vergessen Sie dabei nicht die Zustände (inkl. des Fehlerzustandes selbst), die (noch) keine ausgehenden Kanten besitzen.

Schreiben Sie ein Programm *hopcroft*, welches den Buchstabenbaum aus der letzten Übungsaufgabe einliest (oder einen beliebigen anderen deterministischen endlichen Automaten mit Startzustand 0 im gleichen Format).

Aufruf: `python hopcroft.py automaton.txt`

Testen Sie Ihr Programm mit dem Buchstabenbaum in der Datei `https://www.cis.lmu.de/~schmid/lehre/data/trie.txt`, der aus den ersten 1000 Wörtern der Datei `https://www.cis.lmu.de/~schmid/lehre/data/wortliste.txt` erstellt wurde. Die Minimierung des Buchstabenbaumes dauert etwa eine Minute (je nach Rechner und Implementierung).

Nach Ausführung des Hopcroft-Algorithmus enthält P die Mengen mit äquivalenten Zuständen. Sie müssen nun noch den Ergebnisautomaten wie folgt erstellen:

- Sie erstellen ein Dictionary, welches jeden Zustand einer Äquivalenzklasse auf den Index dieser Äquivalenzklasse abbildet. Als Index der Äquivalenzklasse nehmen Sie
  - 0 falls die Klasse den Startzustand enthält
  - "error" falls die Klasse den Fehlerzustand enthält
  - andernfalls den nächsten freien Index größer als 0
- Dann erstellen Sie ein neues Kanten-Dictionary, indem Sie bei jeder Kante des Original-Automaten die Indizes des Quell- und Zielzustandes mittels der Abbildungstabelle durch die Indizes ihrer Äquivalenzklassen ersetzen.

Geben Sie zum Schluss den Automaten im gleichen Format aus wie den Buchstabenbaum. Dabei lassen Sie den Fehlerzustand und die Kanten zum Fehlerzustand weg. Implementieren Sie den Hopcroft-Minimierungs-Algorithmus mit einer Klasse `Automaton` mit den Methoden `__init__` für das Einlesen aus der Datei, `minimize` für die Minimierung und `print` für die Ausgabe auf den Bildschirm.

Vorüberlegungen

- Erstellen Sie von Hand den Buchstaben-Baum und den endlichen Automaten für die Wortliste: *aaa, baa*
- Welche Datenstrukturen verwenden Sie am besten für
  - das Kanten-Dictionary
  - die Menge P
  - die Warteliste W
  - die Mengen von Zuständen in P und W
  - das Alphabet A
- Wie erstellen Sie am besten F und $\bar{F}$?
- Der Hopcroft-Algorithmus iteriert über die Elemente von P und modifiziert P gleichzeitig innerhalb der Schleife. Welche Probleme können dadurch entstehen und wie können sie vermieden werden?