Übungsblatt: Tag Co-Occurrence auf Delicious

Software Architekturen, 6. Semester Bachelor: Wirtschaftsinformatik

FH CAMPUS 02
Informationstechnologien & Wirtschaftsinformatik

Grahsl Hans-Peter, Steyer Manfred

1) Einführung

Sie bekommen einen kleinen Auszug von Daten der Social Bookmarking Plattform Delicious. Benutzer können dort Ihre URLs abspeichern und dabei beliebige Tags (=Schlagworte) hinzufügen. Die Input-Datei hat folgende Struktur:

URL \t tag1 \t tag2 \t tag3 ...\n

wobei \t ein TAB-Trennzeichen und \n einen Zeilenumbruch darstellen.

Als konkretes Beispiel dienen die folgenden beiden Datensätze:

http://myjmk.com german dictionaries community
http://canoo.net german dictionaries grammar

Ihre Aufgabe ist es mittels Map Reduce zu berechnen wie häufig zwei Tags gemeinsam vorkommen bzw. verwendet wurden. Der gesamte Output für die obigen beiden Datensätze würde wie folgt aussehen:

community	\t	*dictionaries*	\t	**1**	\t	*german*	\t	**1**	\t		\t	
dictionaries	\t	*community*	\t	**1**	\t	*german*	\t	**2**	\t	*grammar*	\t	**1**
german	\t	*community*	\t	**1**	\t	*dictionaries*	\t	**2**	\t	*grammar*	\t	**1**
grammer	\t	*dictionaries*	\t	**1**	\t	*german*	\t	**1**	\t		\t	

Die erste Spalte beinhaltet demnach alle verwendeten Tags. In jeder Zeile findet man weiters alle Tags, mit denen es eine gemeinsame Verwendung gab samt den dazugehörigen Häufigkeiten.

Das heißt man bekommt bezogen auf diese beiden Datensätze im wesentlichen die Information, dass die Tags **„german“ und „dictionaries“ 2x** gemeinsam verwendet wurden, **alle anderen Tag-Paare jedoch nur jeweils 1x** miteinander kombiniert wurden.

2) Laden Sie das Hadoop Starter-Kit aus Moodle herunter

Im Package ***edu.campus02.iwi.hadoop.tags*** finden Sie bereits die Grundstruktur für Ihre MapReduce Implementierung. Machen Sie sich kurz mit der Code-Basis vertraut.

3) Map Reduce Implementierung

Ihre Aufgabe ist es nun, für die im Driver-Code (CoOccurrenceDriver.java) bereits erstellten Jobs geeignete Mapper und Reducer Klassen zu schreiben. Auch die Datentypen sind bereits vordefiniert. Sie verwenden dafür die bereitgestellten Tuple-Klassen **TextTextTuple** und **TextLongTuple.**

Hinweise:

- Job1 erledigt im wesentlichen die Vorarbeit und soll sämtliche **Tag-Paarungen (=>Key)** eines Datensatzes erstellen und im Reducer deren Häufigkeiten berechnen.

- Job2 wird dann jedes **einzelne Tag (=>Key)** und die damit kombinierten Tags samt Häufigkeiten ausgeben und im Reducer entsprechend in einen Datensatz (= 1 Output-Zeile) verpacken.

- Job2 ist im Gegensatz zu Job1 mit dem **KeyValueTextInputFormat** konfiguriert. D.h. der Mapper bekommt in diesem Fall als **Key bereits Text** (=> alles vor dem ersten TAB) und **als Value ebenso Text** (=> die restlichen Daten der Zeile).

Es gibt auch eine Test-Datei zum Ausprobieren Ihres Codes. Diese liegt im Verzeichnis data/input/tags/sample.txt bzw. eine File mit etwas mehr Datensätzen (delicious.txt)

4) optionale BONUS-Aufgabe

Erweitern Sie Ihren MapReduce Job so, dass die Tag-Kombinationen **absteigend nach ihren Häufigkeiten** in den Output geschrieben werden.

z.B.

api	google	3	atom	3	search	2	lucene	2	java		2 ...
apple	hack	2	touch	1	ipod	1	iphone	1			
art	free	3	images	2	clipart	2	gifs	1	animated	1 ...	

Sie sollten dazu das *Secondary Sort* Pattern aus der Vorlesung implementieren, indem Sie geeignete Klassen zum **Sortieren, Gruppieren und Partitionieren** erstellen, um das default Verhalten damit zu überschreiben.