Buongiorno, siamo il gruppo MMS composto da Susanna Valentina Papetti, Michela Pascale e Maria Carmela Pascale e ora illustreremo il nostro progetto.

Iniziamo con la descrizione delle caratteristiche del dataset tables.json, illustrando il numero totale, medio e le distribuzioni delle righe, colonne e tabelle.

Il file contiene 550271 tabelle, qui sono riportati anche i totali e le medie del numero di colonne e del numero di righe. Inoltre, i due grafici a barre rappresentano le distribuzioni del numero di colonne e le distribuzioni del numero di righe, ovvero quante tabelle hanno tot colonne e quante tabelle hanno tot righe. Le tabelle che hanno più di 12 colonne sono 11266, le tabelle che hanno meno di 12 colonne sono 539005. Poiché la maggior parte delle tabelle hanno meno di 12 colonne, abbiamo considerato solo questa parte della distribuzione. Stessa cosa vale per le righe.

Un’altra analisi effettuata sul dataset è stata quella riguardante i valori nulli. In particolare, come vediamo dalla slide, abbiamo i totali, le medie e la distribuzione del numero di valori nulli per tabella. Inoltre, abbiamo anche mostrato la percentuale di valori nulli per tabella e per colonna, in quanto più rappresentativa: infatti, se una tabella ha 2 valori nulli su 10 e un’altra ha 2 valori nulli su 1000, allora la percentuale dei valori nulli di quella più grande sarà sicuramente più bassa.

Un’ulteriore analisi è stata quella riguardante i valori distinti. Anche in questo caso i risultati sono stati riportati in due grafici a barre, uno rappresentante la distribuzione del numero di valori distinti per colonna e uno rappresentante la distribuzione del numero di valori distinti per tabella.

In conclusione, possiamo affermare che la maggior parte delle tabelle hanno una dimensione ridotta, si assestano intorno alle 3-4 colonne e 17 righe, una quantità di valori nulli pressoché zero e una variabilità bassa di valori; infatti, la maggior parte delle tabelle presenta pochi valori distinti.

Passo la parola alla mia collega

In questa seconda parte descriveremo l’organizzazione del lavoro, in particolare classi, metodi e librerie principali utilizzate al fine di realizzare il nostro progetto. Successivamente, descriveremo i principali problemi affrontati e la valutazione sperimentale effettuata.

La prima fase del nostro progetto è stata quella di gestire e leggere il dataset tables.json. Per realizzare il parser abbiamo utilizzato la libreria Jackson, grazie alla quale ogni oggetto json è stato trasformato (deserializzato) in un oggetto Java. Gli oggetti creati sono stati: Table, Cell, MaxDimension, Coordinates. Table è la classe Java che rappresenta una tabella ovvero una riga del json. Tra tutte le properties della tabella abbiamo considerato la collezione di celle e la massima dimensione della tabella. Un’altra classe creata è stata Cell che rappresenta la cella di una tabella. Le properties considerate per questa classe sono ‘isHeader’, un booleano che indica se la cella è un header o no; il cleanedText che rappresenta il testo all’interno della cella e le coordinate che rappresentano la riga e la colonna della tabella dove si trova la cella.

Queste sono quindi le classi Java che abbiamo creato, illustrate in un diagramma uml. Poiché, in seguito al parser, ad ogni tabella viene associata una collezione di celle non ordinata, abbiamo creato una mappa dove abbiamo ordinato le celle per colonna, columnsMap, dove la chiave rappresenta il numero della colonna e il valore la lista di celle associata. Durante la creazione della mappa, non abbiamo considerato le celle con isHeader settato a true, poiché l’intestazione l’abbiamo considerata irrilevante.

Una volta terminata questa fase, siamo passati alla creazione dell’indice sulle tabelle. Abbiamo utilizzato la libreria Lucene che indicizza documenti. Nel nostro caso i documenti sono rappresenti dalle colonne, le colonne sono costituite da celle; quindi, consideriamo come token i testi delle celle, ovvero senza separare le parole all’interno della cella. L’unica operazione che effettuata sui token è stata quella di riportare il testo in lowerCase. Dopo aver indicizzato le tabelle abbiamo implementato l’algoritmo di MergeList. Quindi, data una query, abbiamo letto l’indice e abbiamo selezionato tutti i documenti che fanno matching con la query. Abbiamo creato la mappa set2count per memorizzare ogni documento del matching e il rispettivo numero di token. Infine, una volta ottenuta la mappa, questa è stata ordinata per selezionare i top-k documenti. Noi abbiamo scelto di utilizzare come valore di k, 3.

Passo la parola alla mia collega.

I principali problemi che abbiamo affrontato riguardano:

* La lettura del dataset, quindi come poter effettuare il parser del file json e quali campi del json considerare al fine di ricavare i rispettivi oggetti Java. Come detto abbiamo utilizzato la libreria Jackson.
* Un ulteriore problema è stato quello riguardante la dimensione del file json, quindi come creare l’indice e quando salvarlo. A causa della dimensione del file non è stato possibile prima leggere tutte le tabelle e creare una collezione di tabelle e poi creare l’indice su questa collezione. Abbiamo, quindi, scelto di creare l’indice subito dopo la lettura di ogni tabella, durante il parser, e di effettuare il commit dopo il processamento di un certo numero di tabelle (abbiamo scelto di farlo ogni 50000 tabelle).

Passiamo ora alla fase sperimentale. Abbiamo effettuato degli esperimenti per testare l’efficienza e l’efficacia dell’algoritmo e valutarne il costo. La metrica utilizzata per effettuare tale valutazione è stata quella del tempo, considerando il tempo per accedere all’indice e per effettuare il merge.

Gli esperimenti sono stati condotti selezionando una colonna di una tabella del json, dove questa tabella viene scelta in modo casuale. Di seguito sono riportati alcuni esperimenti in cui si può notare come cambia il tempo in base ai match e la lunghezza delle righe della query; infatti, maggiore è il numero dei match riscontrati e maggiore è la lunghezza della query, maggiore sarà il tempo di calcolo dell’algoritmo. Ad esempio, nell’esperimento 1 abbiamo un query con 36 righe, mentre nel 2, pur avendo una query con 34 righe, il tempo di calcolo è maggiore; questo è dovuto al fatto che il numero di match riscontrati è maggiore nel secondo esperimento rispetto al primo. Caso analogo sono gli esperimenti 3 e 4, in cui nell’esperimento 3 con 58 righe si hanno più match dell’esperimento 4 con 74 righe e, infatti, il tempo di calcolo è maggiore.

Qui abbiamo altri esperimenti, uno con una query con un solo matching e un altro dove analizziamo il comportamento dell’algoritmo con la query con il più alto numero di righe. Si può notare che i tempi di calcolo sono rispettivamente il minore e il maggiore tra tutti gli esperimenti.