Δευτέρα, 28 Ιουνίου 2021
Βασιλειάδης Μιλτιάδης 2944 cs02944@uoi.gr

Αναφορά τέταρτης σειράς ασκήσεων ΜΥΕ041 Διαχείριση Σύνθετων Δεδομένων

Πρώτο Μέρος containment queries

Για την ανάγνωση του αρχείου σε λίστα χρησιμοποιώ την μέθοδο ast.literal_eval. Επειδή σε αυτό το κομμάτι της άσκησης δεν μας ενδιαφέρει το πλήθος των εμφανίσεων ενός αντικειμένου σε μία συναλλαγή απλά η ύπαρξη έφτιαξα μια συνάρτηση “setify” οποία επιστρέφει μια λίστα στην οποία έχει εξαλείψει τα διπλότυπα από το transaction.

Για την naive παίρνω ταξινομημένα copies της ερώτησης και συναλλαγής, για κάθε συναλλαγή υπάρχει μια σημαία contains η οποία αρχικοποιείται false. Για κάθε αντικείμενο στην ερώτηση αν το τρέχον στοιχειο της συναλλαγής είναι μικρότερο προχωράω κατά 1 τον index της συναλλαγής, αν το στοιχείο στην συναλλαγή είναι ίσο με το αντικείμενο στην ερώτηση αλλάζω την flag σε True αλλάζει ο index της συναλλαγής κατά +1, αν το αντικείμενο της συναλλαγής είναι μικρότερο από το τρέχον της συναλλαγής τότε σημαίνει ότι δεν ισχύει το contain. Όταν αλλάζω στο επόμενο αντικείμενο της ερώτησης ελέγχω αν το flag είναι True αν ναι, γινεται πάλι false και συνεχίζει το scan της συναλλαγής από εκεί που είχε σταματήσει. Αν είναι false γίνεται break. Αν δεν υπάρχουν άλλα αντικείμενα στην ερώτηση και το flag είναι True τότε το ID της συναλλαγής μπαίνει στην λίστα result την οποία η μέθοδος επιστρέφει.

Για το bitmap χρησιμοποίησα την setify και δημιούργησα μία λίστα μήκους όσο και το μεγαλύτερο (ID) αντικείμενο του “setified” transaction/query μετά με μια for loop για κάθε αντικείμενο τέλος με ένα join επιστρέφεται το “binary” string. Για την ερώτηση για να ισχύει η συνθήκη containment if (query_signature & (~sigfile[t])) == 0 για να μετατραπουν τα string σε binary χρησιμοποιείται η int(mapbit(questions), 2)

Για την δημιουργία του bitslice κάνω χρήση λεξικού, αν υπάρχει ένα αντικείμενο ήδη στο λεξικό προσθέτω στην τιμή του την δύναμη 2^{τ} όπου τ το index του τρέχοντος transaction για κάθε transaction αν δεν υπάρχει προστίθεται νέο κλειδί το ID του αντικειμένου και το 2^{τ} Για την αναζήτηση για κάθε αντικείμενο στην ερώτηση γίνεται λογικό AND με το bitslice του κάθε αντικειμένου γίνεται αναζήτηση για «1» στο binary αυτού και οι τοποθεσίες των «1» αποτελούν τα IDς των transactions.

Το inverted index δημιουργείται με την χρήση λεξικού με παρόμοια διαδικασία όπως παραπάνω. Για κάθε αντικείμενο που διαβάζουμε σε κάθε transaction προστίθεται το ID του στην λίστα που έχει κλειδί το ID του αντικειμένου. Για την αναζήτηση δημιουργείται μια ενδιάμεση λίστα αποτελεσμάτων με τις λίστες που περιέχουν ID συναλλαγών από το ευρετήριο των αντικειμένων. Γίνονται merge intersect οι πρώτες δύο λίστες σε μια result και για κάθε λίστα που βρίσκεται στην ενδιάμεση λίστα γίνεται pop και merge intersect με την result η οποία και επιστρέφεται.

Η driver function ask παίρνει όρισμα το ID της ερώτησης και το ID της μεθόδου, οι μέθοδοι επιλέγονται με τον αριθμό τους από ένα λεξικό. Για την χρονομέτρηση γίνεται χρήση της time παίρνοντας μια μέτρηση πριν και μια μέτρηση μετά την εκτέλεση της αναζήτησης και η διάρκεια προκύπτει από την διαφορά των δύο

Δευτέρα, 28 Ιουνίου 2021
Βασιλειάδης Μιλτιάδης 2944 cs02944@uoi.gr

Δεύτερο Μέρος ερωτήσεις σχετικότητας

Η ανάγνωση των αρχείων γίνεται όπως περιγράφεται παραπάνω. Για την μέτρηση των trf και occ κάθε αντικειμένου γίνεται υλοποίησα τις συναρτήσεις
count_trf_occ_of_item_all_transactions

count_occ_of_item_in_transaction

count_trf_in_transaction

Η μέτρηση του trf γίνεται με την χρήση ενός λεξικού, με ένα πέρασμα όλων των συναλλαγών και όλων των αντικειμένων στην συναλλαγή με κλειδί το ID του αντικειμένου προστίθεται στο value η μονάδα κάθε φορά που συναντάται το αντικείμενο.

Παρόμοια και για το occ, μόνο που η δομή είναι ένα nested dictionary με κλειδί τα ID των αντικειμένων και για κάθε αντικείμενο υπάρχει ένα dictionary με κλειδιά τα IDs των συναλλαγών που αποθηκεύουν στις τιμές τους το πλήθος του αντικειμένου ανά συναλλαγή.

Υπολογίζεται το ratio |T|/trf(i,τ) και αποθηκεύεται επίσης σε ένα dictionary.

Η δημιουργία του αρχείου είναι διαδικασία συνδυασμού των παραπάνω με την σειρά που ζητείται για κάθε αντικείμενο, σε κάθε γραμμή του αρχείου.

Για την αναζήτηση με την χρήση του ανεστραμμένου ευρετηρίου. Διαβάζονται τα αντικείμενα από τα queries και από το occ τα ID των αντικειμένων και αυτά γίνονται union merge με απαλοιφή διπλοτύπων. Η συνάρτηση ελέγχει τα περιεχόμενα των λιστών και προσθέτει στην λίστα το μικρότερο από τα δύο και προχωράει τον index της λίστας που μόλις το στοιχείο προστέθηκε στο αποτέλεσμα, αν δυο στοιχεία είναι ίσα, γράφεται μόνο το ένα και προχωρά όταν τελειώσει η μία από τις δύο λίστες η άλλη λίστα γίνεται concat στο τέλος της λίστας-αποτέλεσμα το οποίο και επιστρέφεται.

Για κάθε ένα από τα ID υπολογίζεται το score τους με την συνάρτηση $rel(\tau,q)=\Sigma i \in q(occ(i,\tau)\cdot|T|/trf(i,T))$ και προστίθεται στην μορφή λίστας [-rel(t,q), t_id] σε μία minheap. Παρατηρείστε ότι προστίθεται το αρνητικό score επειδή η heapq της python υλοποιεί minheap.

Για την naïve αναζήτηση για κάθε transaction υπολογίζονται τα occ των αντικειμένων, κατόπιν υπολογίζεται το rel για τα αντικείμενα που υπάρχουν στο query, εδώ γίνεται μόνο χρήση του |T|/trf(i,T) από το dictionary. Όπως παραπάνω μπαίνουν σε μια minheap[-rel(t,q), t_id]

Η εκτύπωση των αποτελεσμάτων γίνεται μέσω μιας συνάρτησης που παίρνει όρισμα την λίστα των αποτελεσμάτων και το k, και μετά με μία επανάληψη έως ότου φτάσει το k ή αδειάσει η λίστα με heapq pop παίρνουμε το αποτέλεσμα με το μεγαλύτερο κατ’ απόλυτη τιμή score και το τυπώνει.

Η υλοποίηση της ask είναι η ίδια με αυτήν του προηγούμενου ερωτήματος.

Δευτέρα, 28 Ιουνίου 2021
Βασιλειάδης Μιλτιάδης 2944 cs02944@uoi.gr

Σχολιασμός αποτελεσμάτων.

Ενδεικτικά για το παράδειγμα που δίνεται στην εκφώνηση δηλαδή query[0]

Για το πρώτο μέρος

set4> python .\part1.py .\transactions.txt .\queries.txt 0 -1
Time Elapsed method naive_query is 0.03000164031982422
Time Elapsed method sigfile_query is 0.003004789352416992
Time Elapsed method bitslice_query is 0.0029952526092529297
Time Elapsed method inverted_query is 0.007004976272583008

Παρατηρούμε ότι ή μια μέθοδος με την επόμενη είναι μια τάξη μεγέθους πιο γρήγορη. Η μέθοδος που χρησιμοποιεί το inverted ευρετήριο είναι η πιο γρήγορη καθώς δεν απαιτεί υπολογισμούς κατά την αναζήτηση. Ο Χρόνος έχει δαπανηθεί μία φορά στην δημιουργία του ευρετηρίου

Για το δεύτερο μέρος.

python .\part2.py .\transactions.txt .\queries.txt 0 -1 3
Time Elapsed method naive is 0.03299880027770996
Time Elapsed method inverted_relevance is 0.013001203536987305

Εδώ παρατηρούμε ότι η διαφορά για μια ερώτηση δεν είναι μεγάλη. Αλλά αν δοκιμάσουμε το ίδιο για όλα τα queries.

Time Elapsed method naive is 3.2890007495880127 for all the queries in the file
Time Elapsed method inverted_relevance is 2.2649993896484375 for all the queries in the file

Παρατηρούμε διαφορά ενός δευτερολέπτου μεταξύ των queries. Με την χρήση του inverted ευρετηρίου δεν χρειάζεται ο επαναυπολογισμός των occ για κάθε ερώτηση και αυτή είναι η βασική διαφορά που μειώνει τον χρόνο της αναζήτησης

Η λειτουργία των προγραμμάτων έχει δοκιμαστεί σε Anaconda distribution for windows με python3.7.10 (64bit) και στο opti3060ws08 των εργαστηρίων του τμήματος.