Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Άρης Κονόμης Α.Μ. :1115201400073

Γρηγόρης Καλαντζής Α.Μ. :1115201400055

Μεταγλώττιση: **make**

Εκτέλεση: **./runTestharness.sh**

1)

Η εργασία γράφτηκε σε C και υλοποιήθηκε σε περιβάλλον windows 10, ενώ ελέγχθηκε η λειτουργικότητά της στα μηχανήματα της σχολής μέσω putty.

Σε πρώτη φάση το πρόγραμμα δεν είχε ορίσματα και οι είσοδοί του (2 πίνακες, αριθμός γραμμών και στηλών) δίνονται αυθαίρετα στη main(). Οι δύο πίνακες αποθηκεύονται στη μνήμη κατά στήλες, δηλαδή ως δύο μονοδιάστατοι πίνακες που περιέχουν μία μία τις στήλες του κάθε πίνακα. Για λόγους ευκολίας θεωρούμε πως οι πίνακες έχουν μία στήλη μόνο ώστε να έχουμε αρκετά μεγάλες στήλες για τη join.   
Οι πίνακες είναι αρκετά μεγάλου μεγέθους (τάξεως μισού εκατομμυρίου) και γεμίζονται με τη βοήθεια της rand(). Κατασκευάζονται επίσης δύο πίνακες με τα row-ids των πινάκων, δηλαδή αρχικά έχουμε πίνακες όπου P[i] = i+1 .

Όσον αφορά τη κύρια λειτουργία του προγράμματος, δηλαδή την υλοποίηση της RadixHashJoin, έχουμε ακολουθήσει τα βήματα της εκφώνησης. Θεωρούμε ότι το join των δύο παραπάνω πινάκων γίνεται ως προς την πρώτη στήλη τους.

Έτσι, φτιάχνουμε 2 struct relation, ένα για κάθε πίνακα όπου ο πίνακας από tuples που περιέχονται στο struct θα έχει μέγεθος όσο ο αριθμός των γραμμών τους. Στην πρώτη των tuples του κάθε relation αποθηκεύεται η τιμή h1 (η τιμή που επιστρέφει η συνάρτηση κατακερματισμού 1) για τη συγκεκριμένη τιμή, ενώ στη δεύτερη στήλη αποθηκεύεται απλά η ίδια η τιμή.   
Κατασκευάζονται με τη βοήθεια των παραπάνω πινάκων τα ιστογράμματα και τα αθροιστικά ιστογράμματα του καθενός.   
Έπειτα, τα δύο struct relation που έχουμε δημιουργήσει ταξινομούνται μέσω της quicksort δημιουργώντας buckets για τη συνάρτηση h1. Ταυτόχρονα μεταβάλλονται και οι τιμές στους πίνακες row-ids που έχουμε αντιστοιχισμένους.

Έπειτα, χτίζουμε το ευρετήριο στο μικρότερο από τους δύο πίνακες ως εξής: αλλάζουν οι τιμές της πρώτης στήλης του μικρότερου πίνακα και αντικαθίστανται από τις τιμές που προκύπτουν εφαρμόζοντας την h2 (όπου h2 = (x \* 2654435761) % 101) πάνω στην τιμή της δεύτερης στήλης. Έτσι, φορτώνεται εξ ολοκλήρου όλος ο πίνακας στη μνήμη (R στο σχήμα της εκφώνησης) και όχι ένα ένα τα buckets του. Η προσπέλαση του πίνακα, δηλαδή ο διαχωρισμός των buckets του, γίνεται με τη βοήθεια των ιστογραμμάτων που έχουν υπολογιστεί. Για τον πίνακα Chain έχει υιοθετηθεί η λογική της εκφώνησης όπου οι δείκτες δείχνουν προς τα πάνω και κάθε ακολουθία τελειώνει δείχνοντας σε αυτό (για πρακτικούς λόγους το πρώτο κελί έχει τιμή -1 ). Ο πίνακας bucket έχει μέγεθος όσο το πλήθος των buckets της h1 επί το πλήθος των buckets της h2, δηλαδή στην περίπτωσή μας (2^n) \* 101.

Τέλος, για να πάρουμε τα αποτελέσματα σαρώνουμε ένα ένα τα buckets του μεγάλου πίνακα (αυτού χωρίς ευρετήριο) και για κάθε στοιχείο του διασχίζεται η δομή του ευρετηρίου (του μικρού πίνακα) μέσω στα όρια του bucket αυτού. Έτσι, ελέγχουμε για ποια στοιχεία των αρχικών πινάκων προκύπτουν ίδιες τιμές αν εφαρμόσουμε την h1 αλλά και ίδια για την h2. Αν ισχύει αυτό, αφού ελεγχθεί πως είναι ίσα και τα στοιχεία, γίνεται εισαγωγή στη λίστα αποτελεσμάτων της δυάδας των γραμμών όπου εντοπίστηκε ισότητα. Αυτό γίνεται με τη βοήθεια των αντιστοιχισμένων σε αυτούς πίνακες που κρατάνε τα row-ids. Η δομή της λίστας όπου καταχωρούνται όλες οι ζεύξεις αποτελείται από λίστα με structs (struct result) όπου περιέχεται ένας buffer από tuples μεγέθους 1MB και μόλις γεμίζει το τρέχον δημιουργείται επόμενο struct.   
Τα αποτελέσματα θα εκτυπώνονται στο αρχείο εξόδου «output.txt» (ζεύγη row-ids). Εκτυπώνονται ακόμα χρόνοι εκτέλεσης και το πλήθος των joins που έγιναν.

2)

Χρησιμοποιούνται τα αρχεία του submission πακέτου του διαγωνισμού (harness, run, runTestharness, workloads). Μετά τη μεταγλώττιση (make) το πρόγραμμα εκτελείται με την εντολή ./runTeshharness.sh .

Γίνανε ορισμένες αλλαγές στον κώδικα της άσκησης 1, χωρίς ωστόσο να επηρεάζεται η λειτουργικότητά της. Σχεδόν όλος ο κώδικας βρισκόταν μέσα σε μία μεγάλη συνάρτηση η οποία πλέον έχει «σπάσει» σε αρκετές μικρότερες συναρτήσεις. Ακόμα, ορίσαμε μια δομή ευρετηρίου (struct index).

Γενικά ακολουθείται η λογική της εκφώνησης. Αρχικά, αποθηκεύονται οι πίνακες στη μνήμη με τη δομή που περιγράφεται. Έχουμε δηλαδή έναν πίνακα με τόσα κελιά όσοι οι πίνακες που μας έχουν δοθεί και για κάθε κελί-πίνακα τα στοιχεία του αποθηκεύονται ανά στήλες στη μνήμη ώστε να είναι εύκολα προσπελάσιμες στις πράξεις. Έπειτα, για κάθε batch επερωτήσεων που μας στέλνει το harness στη main μας, υπολογίζονται τα αντίστοιχα checksums και στέλνονται πίσω σε αυτό για να ελεγχθούν ώστε μετά να έρθει το επόμενο batch κοκ.  
Διακρίναμε τρεις περιπτώσεις ερωτήσεων που μπορεί να περιέχονται σε ένα query. Η κλασική join, h self join (join μεταξύ στηλών ενός πίνακα) και το φίλτρο. Έτσι, για κάθε τέτοια περίπτωση εκτελούνται αντίστοιχα οι συναρτήσεις: RadixHashJoin, SelfJoin, Filter . Ως ενδιάμεσο πίνακα ανάμεσα σε τέτοιες ερωτήσεις κρατάμε απλά πίνακες με row-ids, έναν για κάθε πίνακα-σχέση .