ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη λογισμικού για πληροφοριακά συστήματα

Εργασία 1 – Αναφορά

Ιωσήφ Πονσέτης **- 1115201100100**

Ιωάννης Παπαδόπουλος **- 1115201200139**

Περιεχόμενα:

Σελίδα

Δομές δεδομένων

1. Για τα ζητούμενα της άσκησης υλοποιήθηκε ένας πίνακας κατακερματισμού, με συνάρτηση κατακερματισμού την “Murmu3” (non – cryptographic hash function). Σκοπός της χρήσης ενός πίνακα κατακερματισμού είναι η γρήγορη εύρεση ενός συγκεκριμένου στοιχείου από το σύνολο δεδομένων εισόδου.

   Στα αρχεία “hashTable.c” και “hashTable.h” υπάρχουν οι ορισμοί και οι δηλώσεις αντίστοιχα, των συναρτήσεων που συνθέτουν τον πίνακα κατακερματισμού. Όσον αφορά στη δομή του πίνακα κατακερματισμού, αυτή ορίζεται στο αρχείο “hashTable.h” και έχει την εξής παρακάτω μορφή:
   - Ένα **struct** με όνομα **hashTable_t** περιέχει σαν πεδία έναν πίνακα με τα buckets του **hashTable**, καθώς και μεταπληροφορίες για την συγκεκριμένη δομή όπως είναι το μέγεθος και ο αριθμός των στοιχείων της. Το πεδίο **cliqueFreeArrayIndex** αναλύεται παρακάτω (βλ. Δομή κλίκας).
   - Τα buckets του **hashTable** περιγράφονται από μία δομή **bucket_t**, η οποία περιέχει σαν πεδία έναν πίνακα, ο οποίος κρατάει τα στοιχεία που μπαίνουν στο **hashTable**. Τα υπόλοιπα πεδία είναι μεταπληροφορίες για την συγκεκριμένη δομή, οι οποίες περιγράφουν το μέγεθος της, τον αριθμό των στοιχείων της, καθώς και την θέση του πίνακα του εκάστοτε bucket στην οποία θα εισαχθεί το επόμενο στοιχείο.
   - Το εκάστοτε στοιχείο του **hashTable** περιγράφεται από μία δομή **bucketElement**_t, η οποία περιέχει σαν πεδία το κλειδί του στοιχείου (το οποίο αποτελείται από τον συνδυασμό **site name** και **item id**) και τη θέση που έχει λάβει το στοιχείο στον πίνακα της κλίκας (βλ. Δομή κλίκας) .

2. Η συνάρτηση κατακερματισμού που χρησιμοποιήθηκε είναι η **“Murmur3”**. Ο λόγος επιλογής της παραπάνω συνάρτησης είναι η **ταχύτητα** και οι **καλά κατανεμημένες τιμές hash** που δίνει για μεγάλο σύνολο δεδομένων. Περισσότερες πληροφορίες για την υλοποίηση και την χρήση της συγκεκριμένης συνάρτησης βρίσκονται στο παρακάτω ***github repository***: https://github.com/PeterScott/murmur3

3. Για την δομή κλίκας επιλέχθηκε η αναπαράσταση της με διπλά συνδεδεμένη λίστα, η οποία λίστα με τη σειρά της υλοποιείται με πίνακα. Η παραπάνω επιλογή έγινε με γνώμονα ότι το πλήθος των στοιχείων που εισάγονται από το σύνολο δεδομένων (dataset) στην μνήμη είναι γνωστό κατά την εκτέλεση του προγράμματος και ότι ο πίνακας σε αντίθεση με τη λίστα δεν επιβαρύνει με πολλαπλές δεσμεύσεις block στην μνήμη (πολλά memory allocation για τους κόμβους μιας λίστας, έναντι ενός στην περίπτωση του πίνακα). Άλλο ένα πλεονέκτημα του πίνακα είναι ότι εκμεταλλεύεται το cache locality με αποτέλεσμα μεγαλύτερες ταχύτητες στις λειτουργίες της κλίκας.

   Στα αρχεία “clique.c” και “clique.h” υπάρχουν οι ορισμοί και οι δηλώσεις αντίστοιχα, των συναρτήσεων που συνθέτουν την κλίκα. Όσον αφορά στη δομή της κλίκας, αυτή ορίζεται στο αρχείο “ clique.h” και έχει την εξής παρακάτω μορφή:

   - Ένα **struct** με όνομα **clique_t** περιέχει σαν πεδία έναν πίνακα με τα στοιχεία της κλίκας, καθώς και μεταπληροφορίες όπως το μέγεθος της.
   - Τα στοιχεία της κλίκας περιγράφονται από μία δομή **cliqueElement_t**, η οποία περιέχει σαν πεδία μία δομή που κρατάει μεταπληροφορίες για το εκάστοτε στοιχείο, ***δύο*** δείκτες ***θετικής*** γειτνίασης, ***δύο*** δείκτες ***αρνητικής*** γειτνίασης, ένα ***flag*** τύπωσης ***θετικής*** γειτνίασης, ένα ***flag*** εκτύπωσης ***αρνητικής*** γειτνίασης, ένα ***flag*** που δηλώνει ότι ένα στοιχείο ***ανήκει*** σε μια κλίκα ***θετικής*** γειτνίασης και ένα ***flag*** που δηλώνει ότι ένα στοιχείο ***ανήκει*** σε μια κλίκα ***αρνητικής*** γειτνίασης.

   Αρχικά όλα τα στοιχεία εισάγονται στον πίνακα κατακερματισμού, ο οποίος με τη σειρά του αποφασίζει σε ποιο bucket θα αποθηκευτεί το στοιχείο. Έπειτα αφού αποθηκευτεί, συνεχίζει εισάγοντας το στοιχείο στην κλίκα. Τα στοιχεία εισάγονται στην κλίκα με την σειρά, κάτι που αναλαμβάνει να κάνει ο δείκτης cliqueFreeArrayIndex, ο οποίος ξεκινάει με τιμή μηδέν (πρώτη θέση του πίνακα της κλίκας) και αυξάνεται σταδιακά κατά ένα (όσο εισάγονται νέα στοιχεία). Αρχικά όλα τα στοιχεία βρίσκονται σε μία κλίκα με τον εαυτό τους, τόσο στην αρνητική, όσο και στην θετική γειτνίαση. Πρακτικά αυτό σημαίνει ότι αν για παράδειγμα εισαχθεί το στοιχείο “amazon_100” στην θέση τρία του πίνακα της κλίκας, τότε οι παραπάνω δείκτες θα έχουν την τιμή 3. Όσο εισάγονται στοιχεία που ταιριάζουν (ή που δεν ταιριάζουν αντίστοιχα), κλίκες σχηματίζονται και μεγαλώνουν και οι δείκτες των στοιχείων αναδιατάσσονται. Ακολουθώντας την παραπάνω

προσέγγιση εμφανίζονται αρκετά σημαντικά πλεονεκτήματα στην δομή της κλίκας όπως είναι τα παρακάτω:

- Ανά πάσα στιγμή μπορούμε από οποιοδήποτε στοιχείο μιας κλίκας, να έχουμε πρόσβαση σε οποιοδήποτε άλλο στοιχείο της ίδιας κλίκας.
- Όλα τα στοιχεία αποθηκεύονται μία φορά σε έναν πίνακα μεγέθους όσος είναι και ο αριθμός των στοιχείων. Έτσι απορρίπτεται η ανάγκη ύπαρξης ξεχωριστών λιστών - κλικών, οι οποίες θα κρατάνε τα στοιχεία που ανήκουν σε αυτές, εξοικονομώντας έτσι χώρο.
- Όπως αναφέρθηκε και παραπάνω η χρήση πίνακα καθιστά περιττή την χρήση δεικτών και αργών προσπελάσεων μνήμης που συναντώνται στις λίστες, ενώ ταυτόχρονα εκμεταλλεύεται το cache locality.

Αποθήκευση δεδομένων στη μνήμη

1. Οι συναρτήσεις για την Ε/Ε αρχείων και την αποθήκευση των δομών στην μνήμη ορίζονται στο αρχείο “fileIO.c” ενώ οι δηλώσεις τους στο αρχείο “fileIO.h”.
   Η συνάρτηση storeInputDatasetInMemory() διαβάζει αναδρομικά όλα τα αρχεία όλων των φακέλων του dataset και χρησιμοποιεί τον συνδυασμό <όνομα_φακέλου/ιστότοπου , αναγνωριστικό_αρχείου> για να τα εισάγει στον πίνακα κατακερματισμού, ο οποίος με τη σειρά του τα εισάγει στη δομή κλίκας.
   Η συνάρτηση storeQueryDatasetInClique() διαβάζει από το αρχείο συσχετίσεων όλα τα προϊόντα που ταιριάζουν (ή δεν ταιριάζουν) και τα εισάγει στην δομή κλίκας με σκοπό να σχηματιστούν οι εκάστοτε κλίκες θετικής ή αρνητικής γειτνίασης.

Εκτέλεση προγράμματος

1. Το πρόγραμμα καλείται με τις εξής παραμέτρους γραμμής εντολής:
   $ <execuable_name> -d <input_dataset> - q <query_file> -o <output_file>

2. Για την μεταγλώττιση του προγράμματος εκτελείτε την εντολή make ή **make all**. Για την εκκαθάριση των .o και του executable αρχείου εκτελείτε την εντολή **make clean**.