Τσουράκης Ορφέας Α.Μ.: 1115201700175
Ταράτσας Ιωάννης Α.Μ.: 1115201700160

Project K23α-1ο Μέρος Εργασίας

Ενδεικτικές Εκτέλέσεις

Βασική εκτέλεση:

./findSameItems -x 2013_camera_specs -w sigmod_large_labelled_dataset.csv

Unit Testing:

Ουρά: ./queue_test

Στοίβα: ./stack_test

RB Δέντρο: ./rbt_test

Το παραδοτέο στο πρώτο μέρος της εργασίας συμπεριλαμβάνει τα αρχεία main.c, structs.c, structs.h, έναν φάκελο tests, που έχει τα stack_test.c, queue_test.c και rbt_test.c τα οποία χρησιμοποιούν την acutest.h στον φάκελο include, για το unit testing. Τέλος περιλαμβάνεται και το σχετικό makefile που παράγει τα παραπάνω εκτελέσιμα.

Συγκεκριμένα:

main.c

main(): Στην main αρχικά ελέγχονται τα ορίσματα από την γραμμή εντολών(-x + όνομα του datasetX + –w + όνομα του datasetW). Στην συνέχεια, γίνεται σάρωση των καταλόγων του datasetX φτάνοντας μέχρι τα .json αρχεία κάθε φορά τα οποία δίνονται ως όρισμα στην συνάρτηση parse(). Εφόσον έχει δημιουργηθεί το αντικείμενο(Item), με την χρήση της parse, κατασκευάζεται ένα ζεύγος (Pair), το οποίο περιλαμβάνει αυτό το αντικείμενο και μια ουρά συσχετιζόμενων ζευγών(related), που αρχικοποιείται αρχικά μόνο με το ίδιο το ζεύγος. Τέλος, εισάγεται σε ένα RBTree για να γίνει πιο εύκολη η αναζήτηση του στην συνέχεια. Στην συνέχεια, γίνεται το διάβασμα του datasetW, με την χρήση της συνάρτησης read_csv(). Τέλος, καλείται η printOutput για την εκτύπωση όλων των συσχετιζόμενων αντικειμένων που προέκυψαν.

parse(): Η parse, αφού δημιουργήσει ένα Item, αρχικοποιεί το id του(με τρόπο τέτοιον ώστε να συμβαδίζει με τον τρόπο αναπαράστασης των id του datasetW) και έπειτα διαβάζει το json αρχείο για την εισαγωγή των χαρακτηριστικών του(Spec) σε μια ουρά από τέτοια χαρακτηριστικά(specs). Πιο συγκεκριμένα, τα χαρακτηριστικά Spec έχουν δύο πεδία τα

οποία είναι το όνομα και η τιμή, όπου οι τιμές τους προκύπτουν εναλλάξ με το διάβασμα byte-byte του αρχείου json. Όταν είναι η σειρά του ονόματος αποθηκεύεται η συμβολοσειρά που βρίσκεται ανάμεσα στα εισαγωγικά("…") χωρίς τα εισαγωγικά και ακολουθεί η σειρά της τιμής. Ενώ, όταν είναι η σειρά της τιμής επειδή εκτός από εισαγωγικά, υπάρχουν και άλλοι τύποι όπως πίνακες/λίστες([]), με πιθανώς πολλά επίπεδα βάθους, χρησιμοποιείται μια στοίβα έτσι ώστε να είναι γνωστό πού ακριβώς τελειώνει η τιμή. Μαζί με την τιμή αποθηκεύονται και οι χαρακτήρες που την περικλείουν για να είναι γνωστός ο τύπος της(πίνακας, απλή συμβολοσειρά, κτλπ) και αν δεν περικλείεται από χαρακτήρες μπαίνει ως συμβολοσειρά.

read_csv(): Η read_csv, για κάθε εγγραφή στο αρχείο αναζητάει τα δύο ζεύγη με βάση το id των αντικειμένων στο δέντρο και αν ταιριάζουν και δεν είναι στην ίδια κλίκα, ενώνονται οι κλίκες(ουρές related) με την χρήση της QueueConcat.

Structs.h

Στο structs .h περιλαμβάνονται οι δομές και περιγράφονται ως εξής:

- QueueNode: Περιλαμβάνει ένα δείκτη σε data τύπου void για να μπορούμε να βάλουμε δεδομένα οποιουδήποτε τύπου, και έναν δείκτη σε επόμενο κόμβο.
- Queue: Έχει ένα δείκτη σε έναν QueueNode για την αρχή της ουράς (head), ένα δείκτη σε έναν QueueNode για το τέλος της ουράς(tail) και έναν μετρητή count.
- Spec: Το spec είναι ένα χαρακτηριστικό και έχει μια συμβολοσειρά για το όνομα του χαρακτηριστικού(name) και μια συμβολοσειρά για την τιμή του(value).
- Item: Περιέχει ένα id ως συμβολοσειρά (id) και μια ουρά από Specs(specs).
- StackNode: Έχει έναν χαρακτήρα(data), και έναν δείκτη σε επόμενο κόμβο(next).
- Stack: Έχει ένα δείκτη σε έναν StackNode για την αρχή της στοίβας(head) και έναν μετρητή count
- Pair: Έχει έναν δείκτη σε Item(item) και έναν δείκτη σε ουρά από ζεύγη (related)
- RBItem: Περιλαμβάνει ένα id και μια ουρά(pairs)
- RBnode: Έχει έναν δείκτη σε RBItem(item), ένα χρώμα και δύο δείκτες σε αριστερο(r) και δεξί(l) κόμβο αντίστοιχα

Structs.c

Στο structs.c έχουν υλοποιηθεί η ουρά, η λίστα και το Red Black Tree. Παρακάτω θα αναλυθούν οι συναρτήσεις για τις συγκεκριμένες δομές:

Queue

QueueInit: Αρχικοποίηση ουράς.

QueueInsert: Εισαγωγή στοιχείου στο τέλος οποιουδήποτε τύπου (cast σε void δείκτη).

QueueEmpty: Έλεγχος αν η ουρά είναι κενή.

QueueConcat: Δέχεται δύο ουρές q1,q2 και συγχωνεύει την q2 στην q1, θέτοντας την q1 ως related ουρά όλων των ζευγών.

Stack

StackInit: Αρχικοποίηση στοίβας.

Push: Εισαγωγή ενός χαρακτήρα (char) στην αρχή.

Check: Αν ο χαρακτήρας που έρχεται είναι { ή [τότε τον βάζει στην στοίβα, αλλιώς αν είναι } ,] τότε βγάζει τα αντίστοιχα { , [από την στοίβα αν υπάρχουν. Αν ο χαρακτήρας είναι το “, τότε ελέγχουμε αν ήδη υπάρχει στην στοίβα για να το βγάλουμε διαφορετικά το εισάγουμε.

StackEmpty: Έλεγχος αν η στοίβα είναι κενή.

Red-Black δέντρο

Ψάχνοντας στην βιβλιογραφία καταλήξαμε στο συμπέρασμα ότι είναι ένα από τα ίσως καλύτερα, ισοζυγισμένα δυαδικά δέντρα, για γρήγορη αναζήτηση O(log(n)), σε αντιπαράθεση με άλλες δομές όπως τα hash tables τα οποία έχουν χειρότερη πολυπλοκότητα χείριστης περίπτωσης και απαιτούν και παραμέτρους οι οποίες είναι άγνωστες κατά την εισαγωγή των στοιχείων. Η εισαγωγή μοιάζει με την εισαγωγή σε οποιοδήποτε δυαδικό δέντρο απλά πρέπει να γίνουν οι κατάλληλες "περιστροφές" και "χρωματισμοί" των κόμβων που τις εξασφαλίζει η συνάρτηση MakeRBTree. Η findPair διασχίζει το δέντρο ώστε να βρει και να επιστρέψει (αν υπάρχει) από την ουρά του κόμβου του δέντρου που περιλαμβάνει τα ζεύγη με τον ίδιο αριθμό id, το ζεύγος με όνομα αντικειμένου fullId. Επιπλέον, υπάρχουν συναρτήσεις για την αρχικοποίηση και την καταστροφή του δέντρου(RBinit(), RBdestr(), RBTinit(), RBTdestr()).

*Ο κώδικας του δέντρου έχει υλοποιηθεί από τον Ορφέα Τσουράκη στο μάθημα Προγραμματισμός Συστήματος.

Επιπλέον συναρτήσεις

printOutput: In order διάσχιση του δέντρου και εκτύπωση μοναδικών ζευγών με βάσει την related ουρά.

PairDestroy: Χρησιμοποιεί την ItemDestroy για την καταστροφή ενός ζεύγους και του αντικειμένου του.
ItemDestroy: Καταστροφή αντικειμένου.

Φάκελος Tests

Για το test των συναρτήσεων έχουμε χρησιμοποιήσει την συνάρτηση TEST_ASSERT της acutest.h.

queue_test.c

test_qcreate: Στην test_qcreate αρχικοποιούμε μια ουρά και ελέγχουμε αν οι τιμές έχουν αρχικοποιηθεί σωστά.
test_qinsert: Στην test_qinsert δημιουργούμε 100 τιμές τύπου ακεραίου και ελέγχουμε αν κάθε φορά αυξάνεται σωστά το μέγεθος της ουράς και αν το τελευταίο στοιχείο που μπαίνει είναι πάντα αυτό που μπήκε. Στο τέλος, αποδεσμεύουμε ότι μνήμη έχουμε δεσμεύσει, ελέγχονται και ότι η ουρά διαγράφτηκε σωστά.
test_qconcat: Στην test_qconcat αφού αρχικοποιήσουμε δύο ουρές, εισάγουμε 50 ζεύγη στην q1 και άλλα 50 στην q2, και εκτελεστεί ο κώδικας της QueueConcat ελέγχουμε αν όλα τα pairs δείχνουν στην q1 και ανήκουν σε αυτήν όταν διαγράφουμε.

stack_test.c

test_stcreate: Στην test_stcreate αρχικοποιούμε μια ουρά και ελέγχουμε αν οι τιμές έχουν αρχικοποιηθεί σωστά.
test_stpush: Στην test_stpush εισάγουμε στην στοίβα μία συμβολοσειρά χαρακτήρα-χαρακτήρα και ελέγχουμε αν κάθε φορά αυξάνεται σωστά το μέγεθος της στοίβας και αν στην αρχή μπαίνει πάντα το σωστό στοιχείο. Στο τέλος διαγράφουμε την στοίβα και ελέγχουμε αν έχει διαγραφτεί σωστά.
test_stcheck: Στην test_stcheck ελέγχουμε αν αυξήθηκε το μέγεθος της στοίβας όταν έπρεπε και αν μειώθηκε όταν έκλεισαν οι αντίστοιχες αγκύλες.

rbt_test.c

test_rbtcreate: Στην test_rbtcreate αρχικοποιούμε ένα RB Tree και ελέγχουμε αν οι τιμές έχουν αρχικοποιηθεί σωστά.
test_rbtinsert: Στην test_rbtinsert εισάγουμε 100 ζεύγη στο δέντρο και ελέγχουμε αν έχουν εισαχθεί κάνοντας αναζήτηση. Επίσης, γίνεται πάλι αυτή η διαδικασία με άλλα 100 ζεύγη με το ίδιο id και ελέγχουμε αν έχουν εισαχθεί κάνοντας πάλι αναζήτηση.