2η Εργασία Δομών Δεδομένων

Το αρχείο OpenAddressHashTable.java αρχικα κάνει implement το interface Dictionary<K,V> οπου K το key που αντιστοιχεί στην θέση του στοιχείου στον πίνακα μας και V το value που αντιστοιχεί σε εκείνο το κλειδί (K).

Fields

Ύστερα έχουμε τα fields μας, δημιουργούμε ως μεταβλητές το αρχικό μέγεθος του πίνακα ο οποίος θα λειτουργήσει ως το hashtable που υλοποιούμαι (INITIAL_SIZE), καθώς και τον ίδιο τον πίνακα τύπου Entry<K,V>, τον αριθμό των στοιχείων που περιέχονται στο hashtable μας κάθε στιγμή (μεταβλητή current_size) και τέλος τον δισδιάστατο πίνακα που θα χρησιμοποιηθεί για τον καθολικό κατακερματισμό αργότερα (int[][] BxUMatrix).

Constructor

Μετά από αυτό υπάρχει το constructor μέσα στο οποίο η μεταβλητή current_size αρχικοποιειται ως 0 για να δείξουμε ότι το πίνακας δεν περιέχει στοιχεια μέσα του, είναι δηλαδη κενός. Επίσης εντός του constructor, δημιουργείται ένα νέο instance της μεθόδου Entry ώστε να δημιουργηθεί
το hashtable. Τέλος, υπάρχει η BxUMatrix, η οποία δημιουργεί το instance του δισδιάστατο πίνακα που θα χρησιμοποιηθεί για τον καθολικό κατακερματισμό αργότερα.

Methods

newHashFunction(K key)

Στη μέθοδο **newHashFunction** θέλουμε αρχικά να ορίσουμε κάποιες μεταβλητές που θα χρησιμοποιήσουμε αρκετά σε αυτή τη μέθοδο για πιο εύκολη χρήση αργότερα, αυτές είναι:

Ένα String ονόματι keyInBinaryFormat ως το αποτέλεσμα της μεθόδου **keyToBinaryString(K key)**, με key το ίδιο απο τα arguments.
Ένας δισδιάστατος πίνακας τύπου int ονόματι CurrentBxUMatrix αρχικοποιήται ως τη μεταβλητή BxUMatrix που έχουμε αυτή την στιγμή (διότι αλλάζει και είναι τυχαίος).

Θέλουμε επίσης να δημιουργήσουμε ένα νέο StringBuilder ονόματι temp και έναν Integer πίνακα 32 θέσεων, ονόματι CurrentKeyMatrixInBinaryFormat.

Ύστερα με τη χρήση μίας for loop μεταφέρουμε τα στοιχεία του keyInBinaryFormat στον CurrentKeyMatrixInBinaryFormat πίνακα, τον κάνουμε reverse διότι θέλουμε να είναι ο αριθμός από τα αριστερά προς τα δεξιά σε32bit μορφή και όχι όπως μπαίνει στον πίνακα το κάθε στοιχείο
(πχ αντι για 100…00 , θέλουμε 00…001), επειδή αυτό κάνει null τα 0 εντός του πίνακα όμως, χρειαζόμαστε μια δεύτερη for loop για να κάνουμε κάθε null εντός του reversed πλεον πίνακα να είναι πάλι 0.

Στη συνέχεια για να βρούμε τη θέση του key μας στο hashtable με καθολικό κατακερματισμό πρέπει να χρησιμοποιήσουμε για όλο το length του CurrentBxUMatrix μια for loop όπου θα κάνει χρήση της **multiplyMatrixCells(int[][] BxUMatrix, Integer[] X, int row)** με **int[][] BxUMatrix** να είναι το CurrentBxUMatrix, **Integer[] X** το πλέον reversed CurrentKeyMatrixInBinaryFormat (με τα στοιχεία του keyInBinaryFormat εντός αυτού) και **row** το i της for loop αυτή την στιγμή. Το αποτέλεσμα της κάθε λουπας το αποθηκεύουμε στο temp κάνοντας append κάθε φορά.
Τέλος σε μία String μεταβλητή ονόματι Index αποθηκεύουμε το temp και τέλος με τη χρήση της **binaryStringToInteger(String Index)**, μετατρέπουμε την Index σε Integer και κάνουμε αυτό modulo **getLength()** για να πάρουμε την τελική τιμή που θέλουμε από την **newHashFunction(K key)**.

put(K key, V value)

Στη μέθοδο **put** εχουμε τα ορίσματα τυπου K (key) και V (value). Αρχικά ορίζουμε μια int μεταβλητή i το αποτέλεσμα της **newHashFunction**. Ύστερα, ελέγχει με

την **contains** αν το key περιέχεται στο HashTable. Αν η συνθήκη βγει αληθής, τοτε το πρόγραμμα ψάχνει τον πίνακα HashTable για να βρει την θέση του key που δώσαμε. Αν δεν το βρει, τότε πάει στην επομενη θέση διαδοχικά και κυκλικά μέχρι να το βρει. Έπειτα, εισάγουμε το παραπάνω value που δώσαμε στην θέση που βρέθηκε. Απο την άλλη, αν η συνθήκη δεν είναι αληθής (δηλαδή, το key δεν υπάρχει στον πίνακα, σύμφωνα με την **contains**) το πρόγραμμα ψάχνει να βρει την πρώτη διαθέσιμη θέση διαδοχικά και κυκλικά. Όταν την βρει, γίνεται νέα εισαγωγή ενός νέου στοιχείου Entry στο HashTable στη θέση i με key και value αυτα που δώσαμε ως arguments. Ύστερα, ο αριθμός των στοιχείων του HashTable αυξάνεται κατα ένα. Τέλος, αν ο αριθμός των στοιχείων του πίνακα είναι ίσος με το μέγεθος (length) του πίνακα HashTable, τότε αυτό το length διπλασιάζεται.

remove (K key)

Στη μέθοδο **remove** αρχικά έχουμε δύο exceptions, ένα στην περίπτωση που ο πίνακας μας είναι άδειος και άλλο ένα στην περίπτωση που καποιο key δεν υπάρχει στο HashTable. Ύστερα, ορίζουμε μια μεταβλητή temp τυπου V ως το αποτέλεσμα της μεθόδου **get(K key)** με το key που δώσαμε ως argument. Μετά, αρχικοποιούμε ένα integer i ίσο με το μηδέν. Στη συνέχεια, υπάρχει ένα while loop που τρέχει για όσο το i είναι μικρότερο του length του HashTable μας. Αν το στοιχείο Entry με δείκτη i του πίνακα HashTable είναι ίσο με null τοτε το i αυξάνεται κατα ένα. Αλλιώς, αν το key του στοιχείου Entry του HashTable με δείκτη i είναι ίσο με το key, τότε η τιμή του στοιχείου γίνεται ιση με null και κατόπιν σταματάει η εκτέλεση του loop, αλλιώς η τιμή του i αυξάνεται κατά ένα. Τέλος, ο αριθμός των στοιχείων του πίνακα μειώνεται επίσης κατα ένα. Ελέγχουμε αν ο αριθμός των στοιχείων είναι μικρότερος ή ίσος του ¼ του μεγέθους του HashTable μας και στην περίπτωση που είναι, υποδιπλασιαζουμε το μέγεθος τού με την χρήση της μεθόδου **changeCapacity(String action)** και κατόπιν η μέθοδος remove επιστρέφει το temp.

get(K key)

Στη μέθοδο **get** εχουμε ως argument την μεταβλητή τυπου K (key). Πρώτα, αρχικοποιούμε μια int μεταβλητή i στο μηδέν. Έπειτα, υπάρχει ένα while loop που τρέχει για όσο το i είναι μικρότερο του length του HashTable μας. Αν το στοιχείο

Entry με δείκτη i του πίνακα HashTable είναι ίσο με null τοτε το i αυξάνεται κατα ένα. Αλλιώς, ελέγχουμε αν το key του HashTable μας είναι ίσο με το παραπάνω key. Αν ναι, τότε επιστρέφει το value του στοιχείου Entry του HashTable μας με δεικτη i, αλλιώς το i αυξάνεται κατα ένα. Στην περίπτωση που το key απο το argument δεν υπάρχει στο πίνακα, τότε η **get** επιστρέφει null.

keyToBinaryString(K key)

Η μέθοδος αυτή μετατρέπει με την χρήση built-in μεθόδων και επιστρέφει το key ως ένα binary string.

binaryStringToInteger(String binaryString)

Η μέθοδος αυτή μετατρέπει με την χρήση built-in μεθόδων και επιστρέφει ένα binary string ως έναν integer.

random0or1()

Η μέθοδος αυτή επιστρέφει έναν τυχαίο int 0 ή 1.

generateBxUMatrix()

Η μέθοδος αυτή δημιουργεί έναν δισδιάστατο πίνακα BxU. Αρχικά, ορίζουμε μια μεταβλητή integer u στο τριάντα δύο και με την προϋπόθεση οτι το hashtable δεν είναι κενό ορίζουμε έναν integer b ο οποιος είναι η τετραγωνική ρίζα του μεγέθους του HashTable και κατόπιν δημιουργούμε έναν int BxU πινακα και τον “γεμίζουμε” τυχαία με 0 ή 1 και τον επιστρέφουμε.

multiplyMatrixCells(int[][] BxUMatrix, Integer[] X, int row)

Στη μέθοδο αυτή, αρχικα ορίζουμε έναν integer result ίσο με μηδέν, ο οποίος θα είναι το αποτέλεσμα του πολλαπλασιασμού που θα γίνει παρακάτω. Ύστερα, υπάρχει μία for loop που τρέχει για int k από μηδέν μέχρι το μεγέθος του πίνακα HashTable. Το result θα αυξάνεται με το γινόμενο του πίνακα X στη θέση k επι το BxUMatrix όπου row οι γραμμες του και k οι στήλες του. Τέλος, το result εκτελεί την πράξη modulo 2 και η **multiplyMatrixCells** επιστρέφει την τιμή του result.

contains(K key)

Η μέθοδος αυτή επιστρέφει true αν το αποτέλεσμα της **get(K key)** δεν είναι null, δηλαδή αν υπάρχει εντός του πίνακα το αντίστοιχο key και false εάν είναι null.

isEmpty()

Η μέθοδος αυτή επιστρέφει true αν ο πίνακας μας είναι άδειος (δεν έχει στοιχεία) και false αν δεν είναι.

size()

Η μέθοδος αυτή επιστρέφει τον αριθμό των στοιχείων που έχει ο πίνακας μας.

clear()

Η μέθοδος αυτή αρχικοποιεί τα fields όπως ο constructor και ουσιαστικά "αδειάζει" τον πίνακα.

getLength()

Η μέθοδος αυτή επιστρέφει το μέγεθος του πίνακα HashTable μας.

changeCapacity()

Η μέθοδος αυτή, έχει ως argument ένα string action και αρχικά δημιουργεί ένα νέο array για το παλιο HashTable, το oldHashtable στο οποίο αποθηκεύονται ολα τα στοιχεία του τρέχοντος Hashtable. Υστερα οριζουμε το length του νέου hashtable ως μηδέν. Μετα απο αυτο, ανάλογα με το action, διπλασιάζουμε ή υποδιπλασιαζουμε το μέγεθος του νεου HashTable και κατοπιν το δημιουργουμε με αριθμο στοιχείων μεσα του ισο με το μηδέν. Τέλος με την χρήση της **generateBxUMatrix()**, δημιουργούμε τον δισδιάστατο πίνακα που θα χρησιμοποιήσουμε για τον καθολικο κατακερματισμό αργότερα και για κάθε στοιχείο στο παλιο HashTable, αν είναι διαφορετικό του null, τότε βάζουμε στο νέο hashtable το key και το value του με την χρήση της **put()**.

printHashTable()

Η μέθοδος αυτή, εκτυπώνει όλα τα στοιχεία του HashTable.

Comments

Σε αντίθεση με την πρώτη μας εργασία, οπου τα comments υπήρχαν για testing/troubleshooting σκοπούς, σε αυτή την εργασία αποφύγαμε την χρήση τους με σκοπό να είναι πιο “ωραίος” ο κώδικας.

Tests

Το αρχείο OpenAddressHashTableTest.java περιέχει και τα 3 Test μας και χρησιμοποιείται για να κάνουμε test τη λειτουργικότητα του implementation μας.

Test 1

Σε αυτό το test εισάγουμε στο hashtable μας 10.000 αριθμούς, κάθε ένας έχει key ένα τυχαίο int από 0 έως 100 και value το key του +1. ΄Υστερα αποθηκεύουμε τα values σε ένα arraylist ονόματι values και αφότου γεμίσουμε τον πίνακα και με τους 10.000 αριθμούς αυτούς, ελέγχουμε ότι με τη χρήση της **get(K key)** μας, μπορούμε να δούμε ότι όντως το κάθε value στο arraylist values αντιστοιχεί στο key που πρέπει, δηλαδή είναι όσο το key του +1.

Test 2

Σε αυτό το test εισάγουμε στο hashtable μας αρχικά 200 int (key του κάθε ενός είναι ο ίδιος ο αριθμός και value το key +1, όπως πριν). Αποθηκεύουμε το length του hashtable μας σε μία μεταβλητή ονόματι PreviousLength και ύστερα εισάγουμε άλλους 200 int με όμοιο τρόπο. Στη συνέχεια ελέγχουμε ότι το length του hashtable μας όντως διπλασιάστηκε σε σχέση με την PreviousLength. ΄Υστερα αποθηκεύουμε το νέο length στην PreviousLength, επίσης αποθηκεύουμε σε μία μεταβλητή ονόματι PreviousSize το τωρινό size του hashtable μας και ελέγχουμε ότι είναι 400. Μετά κάνουμε remove 100 στοιχεία εντός του hashtable μας και ελέγχουμε ότι το size δεν έχει πέσει στο ¼ του PreviousSize και ότι το length

hashtable μας παρέμεινε ίδιο, στη συνέχεια αφαιρούμε άλλα 200 στοιχεία (σύνολο ¾ των στοιχείων του hashtable) και επιβεβαιώνουμε ότι το length όντως υποδιπλασιαστηκε αφότου το size υποτετραπλασιαζεται. Τέλος κάνουμε remove τα υπόλοιπα στοιχεία του πίνακα μέχρι να αδειάσει.

Test 3

Στο test αυτό δημιουργήσαμε ένα txt file (ονόματι TextDocument.txt) με τα lyrics του τραγουδιού του Rick Astley - Never Gonna Give You Up. Στη συνέχεια με τη χρήση μιας while διαβάζουμε το file αυτό γραμμή-γραμμή μέχρι να τελειώσει. Κάθε φορά περνάμε την γραμμή ως LowerCase σε ένα String ονόματι sentence και ύστερα χωρίζουμε το sentence στα κενά του και αποθηκεύουμε κάθε λέξη σε ένα String Array ονόματι words, ύστερα για κάθε στοιχείο του words εισάγουμε στο hashtable μας την λέξη με value 1 εάν δεν υπάρχει ήδη ή αυξάνουμε το value της εντός του hashtable μας κατα 1 εάν ήδη υπάρχει. Τελος, γνωρίζοντας το πόσες φορές εμφανίζονται κάποιες λέξεις εντός των lyrics, ελέγχουμε αν το value της κάθε μιας εντός του hashtable μας είναι ίδιο με τον αριθμό εμφάνισης της στα lyrics.

Συνάδελφοι της εργασίας

=={ it22003, Δημήτριος Αναγνωστόπουλος }==
=={ it22007, Αλέξανδρος Ανδρούτσος }==
=={ it22148, Γεώργιος Δημητρόπουλος }==

GitHub Repository

! Ευχαριστούμε για τον χρόνο σας !