ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Κυριάκος Χατζηπαντελίδης 218104

Γιώργος Νικολαΐδης 2021113

**ΠΙΝΑΚΑΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ**

OPEN ADRESSING

UNIVERSAL HASHING

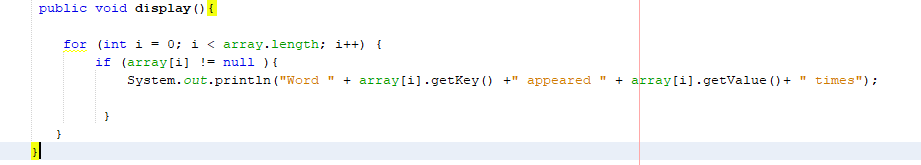
ΣΧΟΛΙΑΣΜΟΣ ΥΛΟΠΟΙΗΣΕΩΝ

Σχετικά με την εργασία μας, δεν χρησιμοποιήσαμε κάποιο ειδικό σύμβολο για να χειριστούμε τις διαγραφές και τα κενά, παρά μόνο χειριστήκαμε το πρόβλημα αυτό με τις null ενδείξεις.

Αρχικά υλοποιήσαμε όλες τις συναρτήσεις που υποστηρίζει το interface Dictionary μέσα στην κλάση OpenAdressingHashTable. Για λόγους υλοποίησης, δεν χρειαστήκαμε την κλήση του iterator, ενώ επίσης προσθέσαμε στο interface την συνάρτηση :

C:\Users\kiko\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_9.png

η οποία μας παρέχει την δυνατότητα για την εκτύπωση των αντικειμένων που ανήκουν μέσα στο λεξικό με την τιμή και το κλειδί τους, λειτουργεί ως εξής:



Ως προς τον τρόπο που λειτουργεί η συνάρτηση κατακερματισμού:

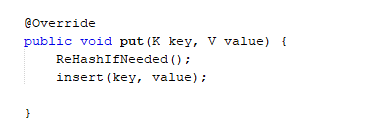


Για να πετύχουμε τυχαία έξοδο b bits κατά την διαδικασία υπολογισμού της hash του κάθε κλειδιού εφαρμόστηκε η παραπάνω υλοποίηση στην οποία η μεταβλητή k αλλάζει κατά +1 κάθε φορά που το μέγεθος του πίνακα μεταβάλλεται ώστε να μην είναι στατικός ο τρόπος που εξάγεται το τελικό νούμερο. Εξάγουμε έναν θετικό ακέραιο αριθμό μεταξύ του εύρους τιμών 0 - (μήκος θέσεων πίνακα-1).

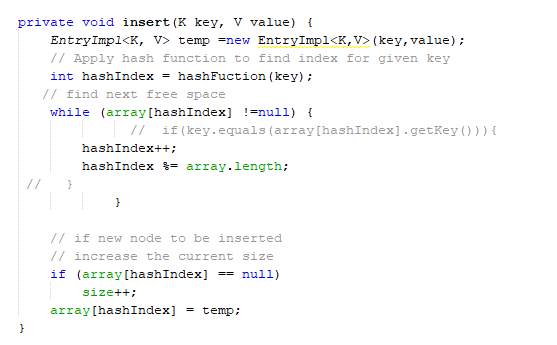
Για την εισαγωγή:

Μέσο της void put() πραγματοποιείται η εισαγωγή με την μέθοδο

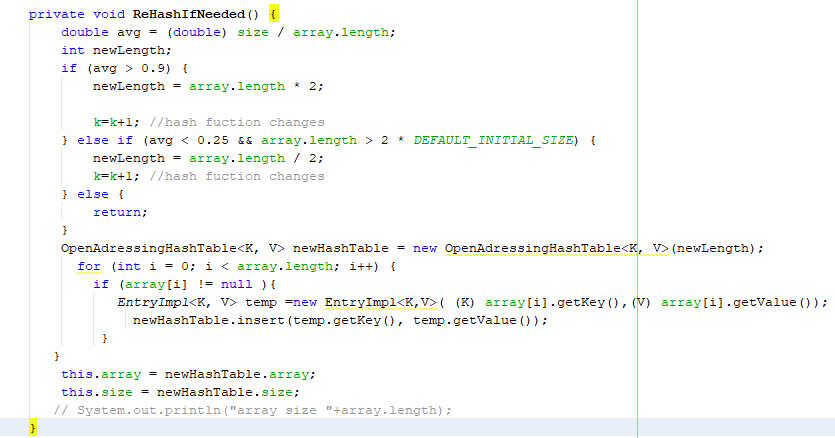
void insert() αλλά και η μορφοποίηση του πίνακα σε νέο μέγεθος εφόσον αυτό απαιτείται με την void ReHashIfNeeded():



Η insert ψάχνει για μια κενή θέση στον πίνακα ώστε να βάλει το κλειδί και την τιμή που έλαβε ως παραμέτρους από την put .Γνωρίζουμε ότι πάντα θα υπάρχει κενή θέση δεδομένου ότι φροντίζει για τον έγκαιρο διπλασιασμό και υποδιπλασιασμό του πίνακα η ReHashIfNeeded έτσι πάντα υπάρχουν κενά λόγο του πότε αναλαμβάνει να αλλάξει το τρέχον μέγεθος σχετικά με το πλήθος των στοιχείων που έχουν εισαχτεί.Τέλος τοποθετεί το αντικείμενο στην κατάλληλη θέση, δηλαδή μέχρι να βρει τιμή null αναζητώντας προς τα μπροστά τις θέσεις του πίνακα μετά από την θέση που έκανε hash αρχικά το κλειδί και αυξάνει το πλήθος αντικειμένων στον πίνακα +1.



Η ReHashIfNeeded αν βρει τον πίνακα >90% γεμάτο προχωράει σε διπλασιασμό των θέσεων του ενώ σε περίπτωση που ο πίνακας είναι γεμάτος μόνο στο <25% του μεγέθους του και δεν είναι το μέγεθος αυτό μικρότερο από το διπλάσιο του default size τότε οι θέσεις του πίνακα μειώνονται στο μισό. Αυτές οι αλλαγές γίνονται μέσα από την δημιουργία νέου πίνακα και μεταφοράς των δεδομένων σε αυτόν ενώ στο τέλος της εισαγωγής των στοιχείων στον νέο πίνακα τον θέτουμε στην ίδια διευθυνσιοδότηση του παλιού:



Για την διαγραφή:

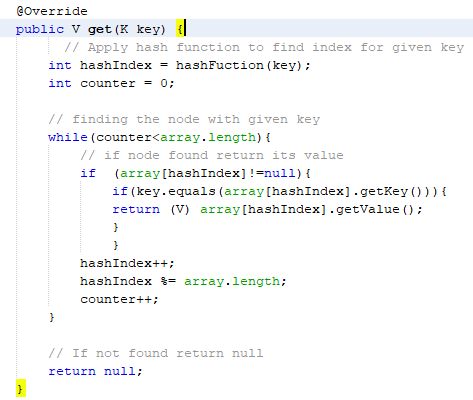
Μέσο της μεθόδου remove, η οποία παίρνει ως παράμετρο το κλειδί που θέλουμε να αφαιρέσουμε, πραγματοποιούμε την διαγραφή των στοιχείων του πίνακα σε περίπτωση που αυτά υπάρχουν αλλιώς δεν κάνει τίποτα και επιστρέφει null. Έπειτα σε περίπτωση που η διαγραφή πετύχει ελέγχουμε τις επόμενες θέσεις για τυχόν ανάγκη για περαιτέρω αλλαγές των στοιχείων και τέλος επιστρέφουμε την τιμή που διαγράφτηκε. Ο έλεγχος για περαιτέρω αλλαγές συγκρίνει εάν κάποιο κλειδί κάνει hashing σε θέση του πίνακα που βρίσκεται σε μεγαλύτερο index ή μικρότερο ίσο από την θέση του διαγραμμένου αντικειμένου. Ελέγχοντας το αμέσως επόμενο κλειδί και συνεχίζοντας σειριακά ως ότου να βρούμε τιμή null. Στην πρώτη περίπτωση, δηλαδή αν το hashing του κλειδιού που βρίσκεται δεξιά της θέση του διαγραμμένου κλειδιού είναι μεγαλύτερο, απλά συνεχίζω τον έλεγχο μου στις επόμενες θέσεις και δεν πειράζω τίποτα. Στην δεύτερη περίπτωση όμως, χρειάζεται να κάνω swap τα αντικείμενα και στην συνέχεια προχωράω για έλεγχο στις επόμενες θέσεις. Tέλος η ReHashIfNeeded καλείται να ελέγξει αν χρειάζεται υποδιπλασιασμός του πίνακα, το πλήθος των αντικειμένων μειώνεται κατά ένα και επιστρέφουμε την τιμή Value του αντικειμένου που διαγράφτηκε.

Ο κώδικας :

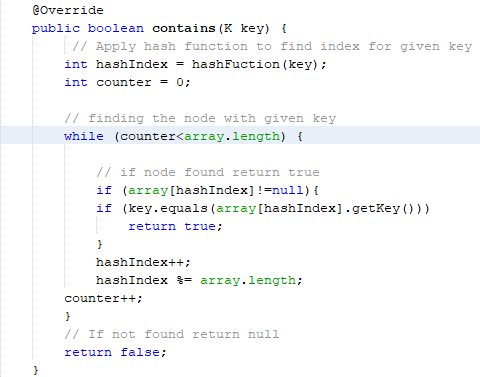


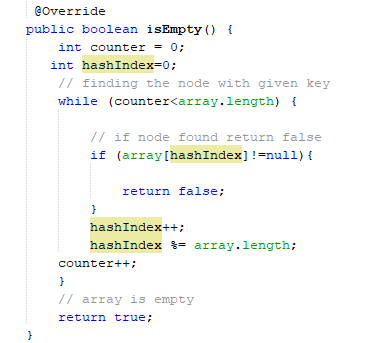
Για την αναζήτηση :

Μέσο της μεθόδου get, αναζητάμε το κλειδί που έδωσε ως παράμετρο ο χρήστης στον πίνακα, και εφόσον βρεθεί αντικείμενο με κλειδί ίδιο με αυτό του επιστρέφουμε την τιμή (value) του αντικειμένου, διαφορετικά η συνάρτηση επιστρέφει null και τερματίζει εφόσον κοιτάξει σε όλο τον πίνακα και δεν το βρει.



Με παρόμοιο τρόπο υλοποιούνται οι Boolean συναρτήσεις contain() και η isEmpty().Η contain παίρνει ως όρισμα ένα κλειδί και επιστρέφει true εάν υπάρχει στον πίνακα. Η isEmpty ελέγχει αν όλος ο πίνακας είναι άδειος και επιστρέφει true αν αυτό ισχύει διαφορετικά false.

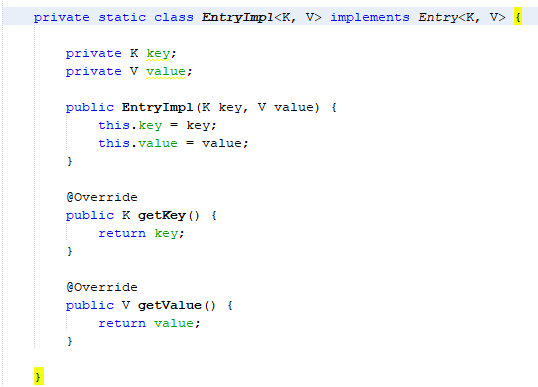




Η συνάρτηση int size() επιστρέφει το πλήθος αντικειμένων του πίνακα.

Η void clear() αδειάζει τον πίνακα.

Για την δημιουργία συγκεκριμένων στοιχείων με pair key value η παρακάτω υλοποίηση μας επέτρεψε να έχουμε πρόσβαση στην τιμή των μεταβλητών key value της κλάσης:

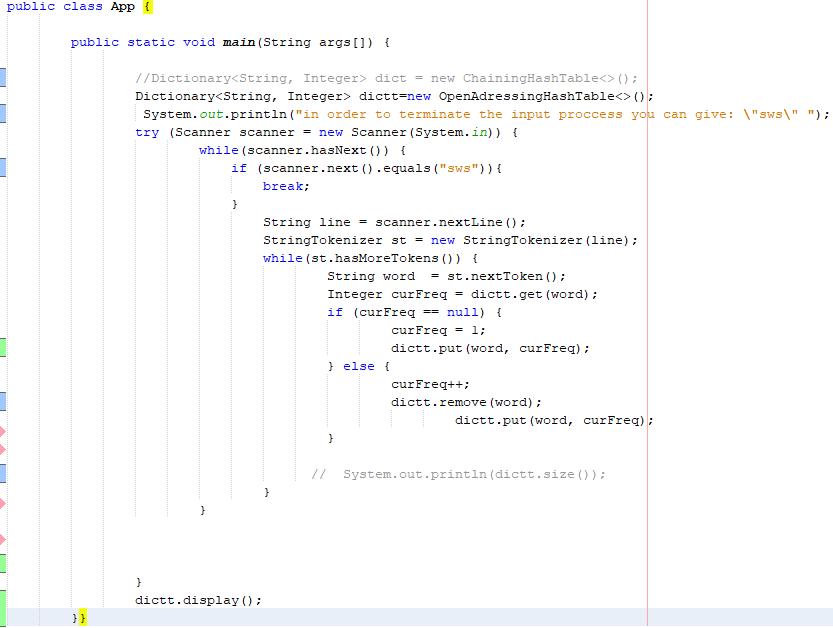


Έτσι, όπως φαίνεται και σε ανωτέρα τμήματα του κώδικα, μπορούμε να κάνουμε εισαγωγή στοιχείων μόνο μέσα από την κλήση αυτής.

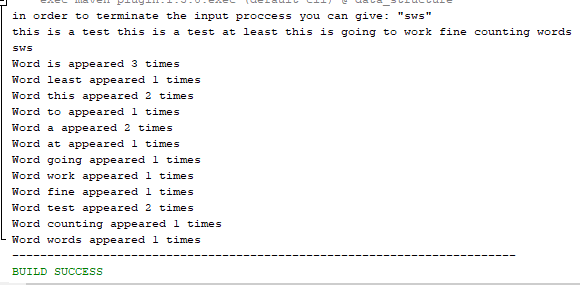
Κλήση της main .

Στην main ο χρήστης μπορεί να δώσει λέξεις και στο τέλος μετράμε το πλήθος των διαφορετικών εμφανίσεων της κάθε λέξης και το εκτυπώνουμε. Η λέξη <<sws>> είναι το σύμβολο που τερματίζει το input του προγράμματος μέσο πληκτρολόγιου για λόγους πρακτικούς. Δοκιμάσαμε (copy-paste) εισαγωγή κειμένων και υπάρχει ένα εσωτερικό σφάλμα που εμφανίζεται που αφορά λογικά κακή διαχείριση της θέσης των αντικειμένων εντός του πίνακα αφού αλλάξει μια φορά η εσωτερική πράξη που υπολογίζει η hashFuction (συγκεκριμένα η μεταβλητή k αυξηθεί) που έχουμε υλοποιήσει. Κατά το σφάλμα αυτό παγώνει η εισαγωγή δεδομένων και άρα η λέξη <<sws>> και ο,τι γράψουμε παύει να αναγνωρίζεται. Η διαδικασία εισαγωγής και διαγραφής αντικειμένων κατά το <<διάβασμα>> των γραμμών (line) των λέξεων που εισάγει ο χρήστης καθώς και η αρίθμηση της συχνότητας αυτών γίνεται με επιτυχία ΕΚΤΟΣ από την πρώτη λέξη κάθε εισαγωγής δεδομένων από τον χρήστη που φαίνεται να μην την διαβάζει ποτέ εσωτερικά το πρόγραμμα για να έρθει σε επαφή με το λεξικό μας. Για την επαλήθευση την λειτουργικότητας του λεξικού μας ακολουθούν μερικές εκτελέσεις για διαφορετικό input.

Η main:



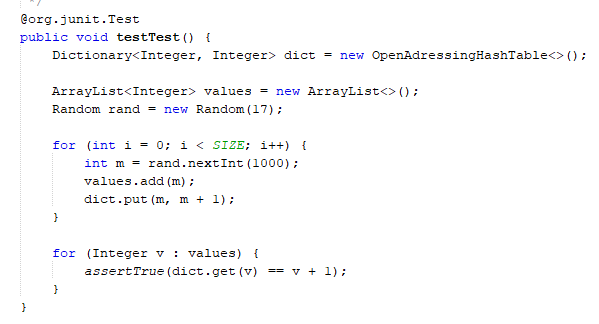
Ενδεικτικές εκτελέσεις:



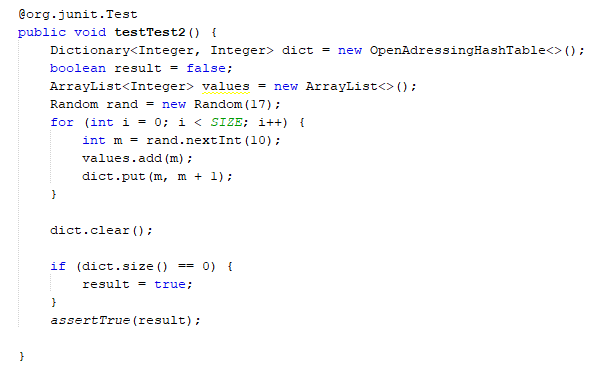
Σχετικά με τα τεστ.

Φτιάξαμε 4 τεστ τα οποία εκτελέστηκαν με επιτυχία. Το καθένα δημιουργήθηκε με σκοπό να ελέγχει διαφορετική λειτουργία του dictionary.

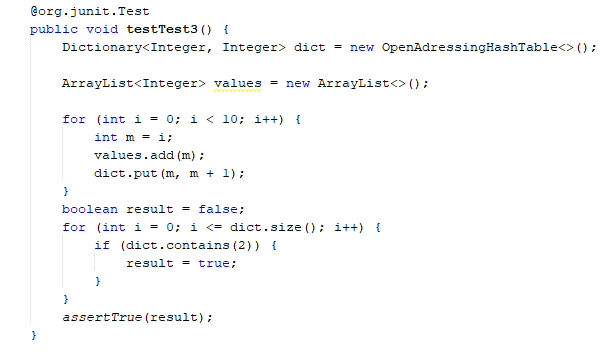
Το πρώτο τεστ unit ελέγχει κατά πόσο λειτουργούν οι μεθόδοι put και get. Μέσω ενός for loop προσθέσαμε τυχαίες παραμέτρους(values) και κλειδιά(keys) και με την βοήθεια της assertTrue ελέγξαμε κατά πόσο η get επιστρέφει σωστό αποτέλεσμα.



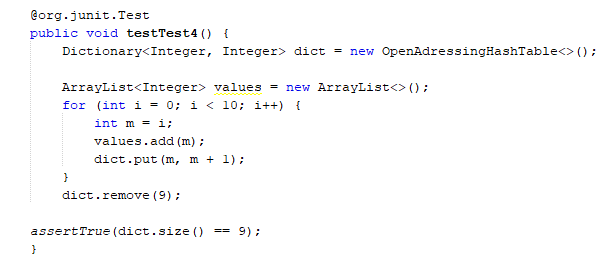
Με το δεύτερο τεστ unit ελέγξαμε την λειτουργία της μεθόδου clear. Όπως και πριν με ένα for loop προσθέσαμε στοιχεία και καλέσαμε την clear. Μετά ελέγξαμε το πλήθος των στοιχείων που πρέπει να είναι 0.



Στο τρίτο τεστ unit θέλαμε να δούμε την λειτουργία της μεθόδου contains. Έτσι προσθέσαμε στοιχεία στο dictionary με ενα for loop από το 0 μέχρι το 10 ώστε να ξερουμε τα στοιχεία. Μετά καλεσαμε την contains με ένα από τους αριθμούς, στην προκειμένη περίπτωση το 2, και ελέγξαμε κατά πόσο η contains εκτελείτε σωστα



Στο τέταρτο και τελευταίο τεστ unit θέλαμε να ελέγξουμε την remove. Όπως πάντα με το for loop προσθέσαμε στοιχεία στο dictionary και καλέσαμε την remove σε ενα στοιχείο. Tέλος λέγξαμε αν μειώθηκε το size.



Οι εκτελέσεις των τεστ:

