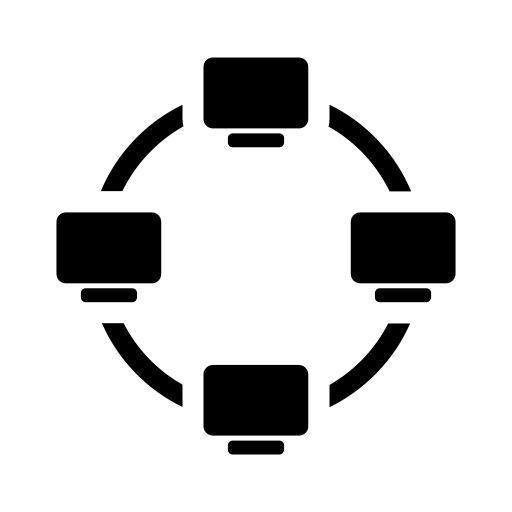
Πολυδιάστατες

Δομές Δεδομένων

**Project 2021-2022**

DHTs (Chord)



|  |  |
| --- | --- |
| **Στυλιανός Στυλιανάκης** | **1059713** |
| **Χάρης Καπελετιώτης** | **1057772** |
| **Κωνσταντίνος Κωστόπουλος** | **1067482** |

**[Icon

Description automatically generated](https://github.com/Steliostyl/Projects-kDDs-2022)**[**Link to GitHub**](https://github.com/Steliostyl/Project-kDDs-2021-2022)

# Λίγη θεωρία

## Distributed Hash Tables

Οι κατανεμημένοι πίνακες κατακερματισμού είναι μία δομή **αποκεντροποιημένης** διαμοίρασης αρχείων, δομημένων σε ζευγάρια κλειδιού-τιμής.

Κάθε **κόμβος** ενός DHT είναι υπεύθυνος για την αποθήκευση ενός υποσυνόλου κλειδιών και των τιμών τους, οι οποίες μπορεί να είναι οποιουδήποτε τύπου δεδομένων. Τα κλειδιά είναι μοναδικά αναγνωριστικά που δημιουργούνται μετά από **κατακερματισμό** των δεδομένων τιμών, ενώ οι κόμβοι που συμμετέχουν στη δομή δρουν ως **ισάξιες** οντότητες για τον διαμοιρασμό των αρχείων.

Ένα από τα κύρια πλεονεκτήματα των DHTs είναι ότι οι κόμβοι μπορούν να εισέρχονται και να απομακρύνονται από το σύστημα με συνοπτικές διαδικασίες ανανέωσης κλειδιών. Για αυτόν το λόγο, τέτοια συστήματα έχουν τη δυνατότητα να **κλιμακώνονται** μέχρι και έναν πολύ μεγάλο αριθμό κόμβων, ενώ η **ανοχή** τους **στις βλάβες** τα καθιστά ικανά να διαχειρίζονται **ταυτόχρονες** αφίξεις και αναχωρήσεις κόμβων.

## Chord

Το Chord είναι ένα **πρωτόκολλο** κατανεμημένων πινάκων κατακερματισμού, το οποίο παρουσιάστηκε από το MIT το 2001 και βελτιώνει δραματικά την **πολυπλοκότητα** (και άρα το χρόνο) των αναζητήσεων κλειδιών.

Για την ισοκατανομή των κλειδιών στους κόμβους, το Chord χρησιμοποιεί **consistent hashing** αλγόριθμους για τον κατακερματισμό των κλειδιών. Συγκεκριμένα, ο βασικός αλγόριθμος που χρησιμοποιείται είναι ο **SHA-1**, ο οποίος χρησιμοποιήθηκε και στην εργασία, ύστερα από τροποποίησή του για να περιοριστούν οι τιμές σε μικρότερους **χώρους κατακερματισμού**.

Για την επιτάχυνση της αναζήτησης κλειδιών, οι κόμβοι αποθηκεύουν έναν πίνακα δρομολόγησης, γνωστό ως **finger table**, ο οποίος περιέχει KS εγγραφές. Κάθε εγγραφή αποτελείται από μία θέση και έναν κόμβο, ο οποίος ευθύνεται (πιθανώς μεταξύ άλλων και) για τα κλειδιά τα οποία ανήκουν στο διάστημα [θέση, κόμβος εγγραφής]. Λόγω του πίνακα αυτού, το κόστος της αναζήτησης ενός κλειδιού είναι O(log(n)).

# Υλοποίηση

Για την υλοποίηση της εργασίας δημιουργήθηκαν 4 αρχεία κώδικα. Το node.py, το interface.py το benchmarks.py και φυσικά η main.py.

## Το αρχείο node.py

Βοηθητικές συναρτήσεις στο node.py

#### hash\_func

Ο αλγόριθμος του κατακερματισμού. Στο παρόν πόνημα, ο βασικός αλγόριθμος κατακερματισμού που χρησιμοποιείται είναι ο SHA-1.

Text

Description automatically generated

cw\_dist

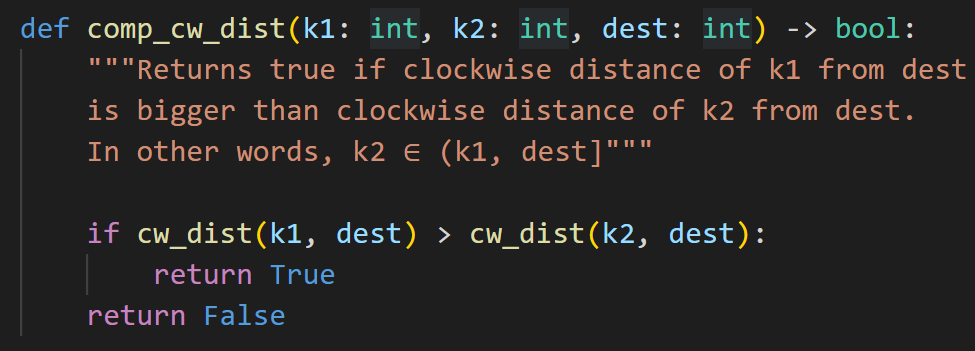
Επιστρέφει την clockwise απόσταση των δύο ορισμάτων (συνήθως κλειδιά).

Graphical user interface, text, application, chat or text message

Description automatically generated

comp\_cw\_dist

Με είσοδο 3 τιμές k1, k2 και dest χρησιμοποιεί την συνάρτηση cw\_dist και ελέγχει αν ισχύει η σχέση k2 ∈ (k1, dest]. Επιστρέφει True αν ισχύει, αλλιώς επιστρέφει False.



## Κλάση Node

Η βασική κλάση της υλοποίησης, αναπαριστά έναν **κόμβο** δικτύου ο οποίος μπορεί να τρέξει όλες τις απαραίτητες διαδικασίες, όπως παρουσιάζονται παρακάτω.

|  |  |
| --- | --- |
| node.id | Το id του κόμβου. |
| node.items | Dictionary που περιέχει τα στοιχεία του κόμβου. Τα κλειδιά του λεκτικού είναι τα unhashed κλειδιά των ζεύγων (κλειδιών, δεδομένων) και οι τιμές τους τα δεδομένα. |
| node.f\_table | Το finger table του κόμβου (λίστα). Κάθε εγγραφή της λίστας αποτελείται από μία λίστα 2 στοιχείων της μορφής [position, node]. |
| node.pred | Ο αμέσως προηγούμενος κόμβος. |
| node.succ\_list | Λίστα με τους άμεσους successors μεγέθους SLS, μία global σταθερά που εμείς έχουμε ορίσει να είναι 3. |

Εδώ να σημειωθεί πως το μέτρο κατά των **massive node failures** δεν καταφέραμε να το υλοποιήσουμε λόγω έλλειψης χρόνου και δυναμικού. Παρόλα αυτά, έχουν υλοποιηθεί τα θεμέλια για τη λειτουργία αυτή, αφού αποθηκεύουμε τη λίστα λίστα succ\_list σε κάθε κόμβο, η οποία ενημερώνεται. Αυτό που λείπει λοιπόν είναι η **σύνδεση** της λίστας αυτής με τη διαδικασία **αναζήτησης** **κλειδιού** και η σωστή **ενημέρωση** των **finger tables** σε περίπτωση που κάποιος successor δεν ανταπεξέρχεται.

### Συναρτήσεις της κλάσης Node

#### Text Description automatically generatedfind\_successor

Επιστρέφει τον υπεύθυνο κόμβο για ένα κλειδί. Δηλαδή, τον κόμβο με id μεγαλύτερο ή ίσο με το κλειδί που έχει δοθεί, με την βοήθεια των μεθόδων closest\_pre\_node και comp\_cw\_dist.

#### closest\_pre\_node

Ψάχνοντας στο finger table του κόμβου από τον οποίο έχει καλεστεί, επιστρέφει τον κοντινότερο predecessor κόμβο ενός κλειδιού.

Text

Description automatically generated

#### fix\_fingers

Ανανεώνει το finger table ενός κόμβου.

Text

Description automatically generated

#### fix\_successor\_list

Ανανεώνει τη λίστα με τους αμέσως επόμενους successors ενός κόμβου.

Text

Description automatically generated

#### insert\_new\_pred

Η συνάρτηση που καλείται όταν εισέρχεται ένας καινούριος κόμβος στο δίκτυο. Ο λόγος που ονομάζεται έτσι, είναι επειδή καλείται στον πρώτο successor του καινούριου κόμβου, στον οποίο «εισάγουμε» έναν καινούριο predecessor.

Text

Description automatically generated

#### insert\_item\_to\_node

Προσθέτει ένα καινούριο αντικείμενο στον κόμβο.

Text

Description automatically generated

#### delete\_item\_from\_node

Δεδομένου ενός κλειδιού, ελέγχει αν υπάρχει το αντίστοιχο αντικείμενο στον κόμβο και το διαγράφει. Αλλιώς εκτυπώνει ότι δεν βρέθηκε.

Text

Description automatically generated

#### move\_items\_to\_pred

Μετά την είσοδο ενός καινούριου κόμβου στο δίκτυο, καλείται στον successor του. Μεταφέρει στον καινούριο κόμβο τα αντικείμενα που έχουν κλειδί μικρότερο ή ίσο από το id του καινούριου κόμβου.

Text

Description automatically generated

#### initialize\_finger\_table

Αρχικοποιεί το finger table ενός καινούριου κόμβου.

Text

Description automatically generated

#### leave

Αποχώρηση του κόμβου από το δίκτυο

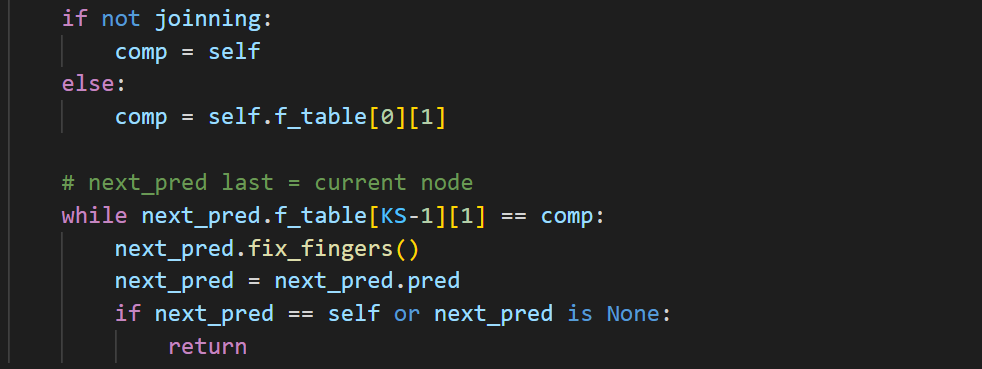
Text

Description automatically generated

#### update\_necessary\_fingers

Ανανεώνει τα finger tables (και το successor list) μετά την άφιξη και την αποχώρηση κόμβων.

Text

Description automatically generated 

#### print\_node

Εκτυπώνει έναν κόμβο. Ως είσοδο δέχεται boolean μεταβλητές που λειτουργούν ως επιλογές εκτύπωσης των αντικειμένων μέσα του και του finger table.

Text

Description automatically generated

## Το αρχείο interface.py

#### parse\_csv

Η συνάρτηση αυτή διαβάζει το dataset (csv), δημιουργεί τα αντικείμενα (τα οποία είναι λεξικά) και τα περνάει σε μία λίστα, την οποία τελικά επιστρέφει.

Text

Description automatically generated

## Κλάση Interface

Η κλάση αυτή αποτελεί, όπως λέει και το όνομά της, ένα interface από το οποίο **προσομοιάζεται** η πρόσβαση στους διαφορετικούς κόμβους του δικτύου, ώστε να μπορούμε να ξεκινάμε διαδικασίες ως οποιοσδήποτε κόμβος θέλουμε. Οι κόμβοι του δικτύου αποθηκεύονται στο λεξικό **nodes** του interface.

### Συναρτήσεις της κλάσης Interface

#### build\_network

Δημιουργεί το δίκτυο είτε με συγκεκριμένα node ids που έχουν περαστεί ως λίστα είτε με τυχαία, εφόσον ορίσουμε τον αριθμό κόμβων που θέλουμε να δημιουργηθούν.

Text

Description automatically generated

#### get\_node

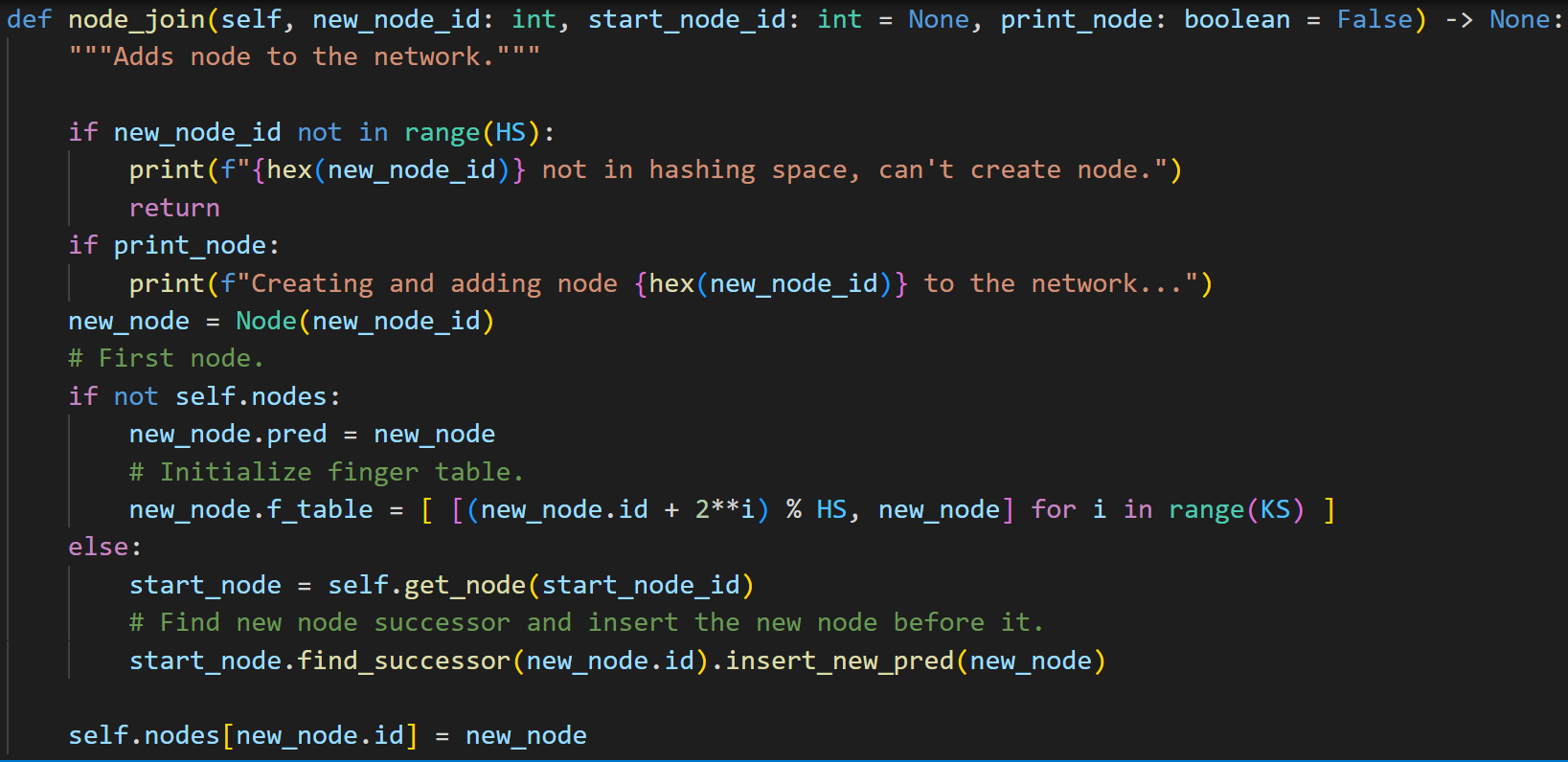
Επιστρέφει τον κόμβο με id node\_id. Εάν δεν βρεθεί, επιστρέφει τον **πρώτο** κόμβο που εισήλθε στο δίκτυο. Χρησιμοποιείται από τις συναρτήσεις του interface, ώστε να μπορούμε να ξεκινάμε διαδικασίες από **οποιονδήποτε** κόμβο του δικτύου θέλουμε. Για τον σκοπό αυτό, οι συναρτήσεις παίρνουν σαν προαιρετικό όρισμα ένα start\_node\_id που έχει default τιμή None και στη συνέχεια καλούν τη συνάρτηση get\_node με το όρισμα αυτό.

Text

Description automatically generated

#### node\_join

Δεδομένου ενός id, δημιουργεί και προσθέτει έναν κόμβο στο δίκτυο.



#### node\_leave

Δεδομένου ενός id, διαγράφει έναν κόμβο από το δίκτυο.

Text

Description automatically generated

#### exact\_match

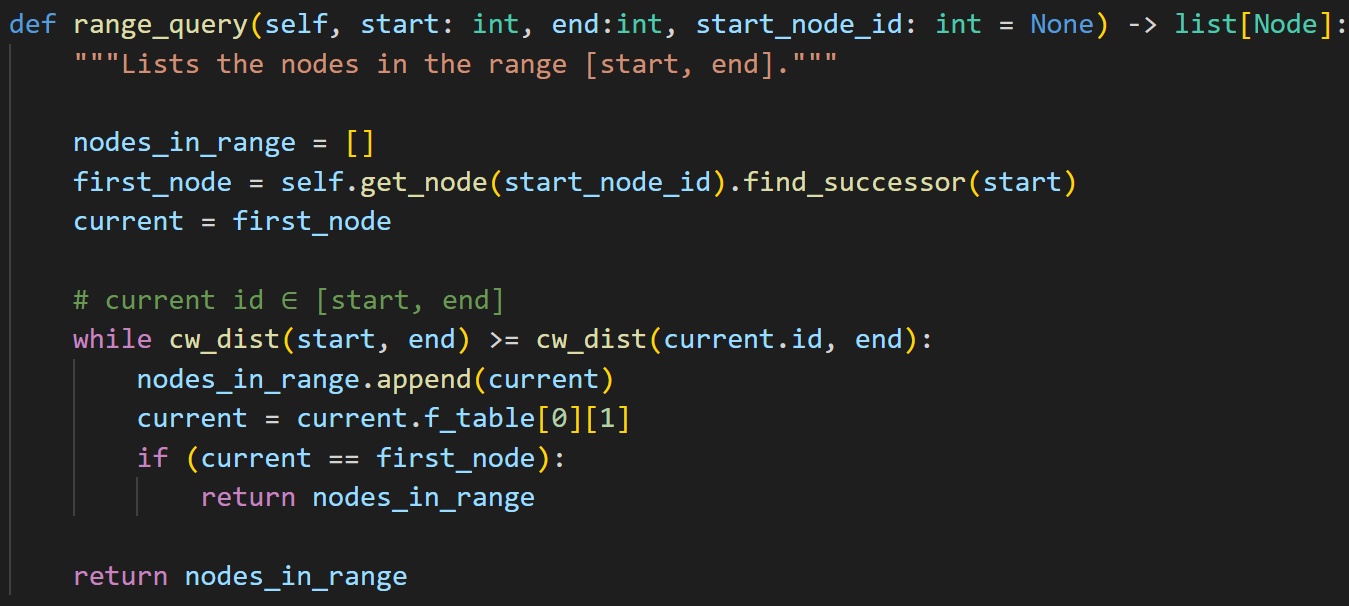
Δεδομένου ενός id, βρίσκει και επιστρέφει τον κόμβο με αυτό το id.

Text

Description automatically generated

#### range\_query

Επιστρέφει όλους τους κόμβους σε ένα εύρος.



#### insert\_item

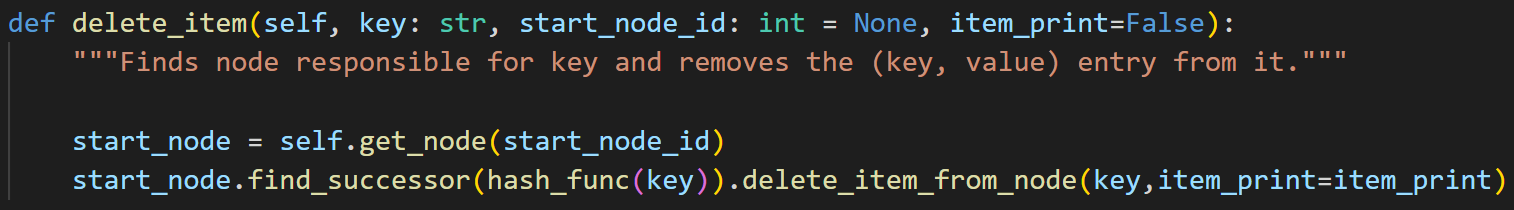
Βρίσκει τον υπεύθυνο κόμβο για ένα item και καλεί τη συνάρτηση insert\_item\_to\_node σε εκείνον, εισάγοντας το αντικείμενο.

Text

Description automatically generated

#### delete\_item

Βρίσκει τον υπεύθυνο κόμβο για ένα item και καλεί τη συνάρτηση delete\_item\_from\_node σε εκείνον, διαγράφοντας το αντικείμενο.



#### update\_record

Βρίσκει τον υπεύθυνο κόμβο για ένα αντικείμενο, ελέγχει αν υπάρχει το αντικείμενο και ανανεώνει τα δεδομένα του.

Text

Description automatically generated

#### insert\_all\_data

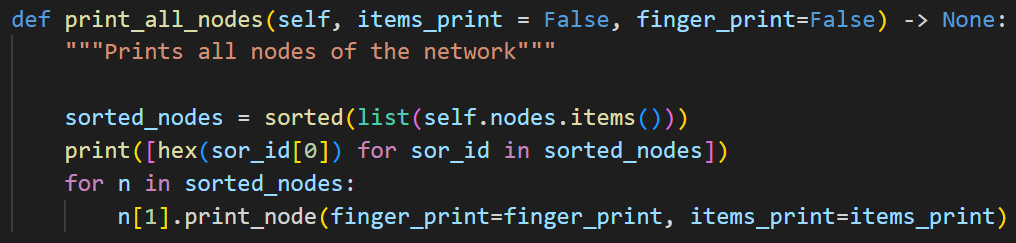
Εισάγει όλα τα αντικείμενα που έχουν συλλεχθεί από το dataset (csv αρχείο) στους υπεύθυνους κόμβους.

Text

Description automatically generated

#### print\_all\_nodes

Εκτυπώνει όλους τους κόμβους του δικτύου.



#### knn\_query

Επιστρέφει τους k κοντινότερους κόμβους ενός κόμβου.

Text

Description automatically generatedText

Description automatically generated

## Το αρχείο main.py

Στην αρχή της main, ορίζουμε κάποιες σταθερές. Πιο συγκεκριμένα, ορίζουμε το μέγεθος του κλειδιού της συνάρτησης κατακερματισμού μας (γνωστό και ως digest), το hashing space (το οποίο είναι προφανές ότι εξαρτάται από το μέγεθος του κλειδιού), το μέγεθος του successor list για το μέτρο κατά των massive nodes failure και το πλήθος κόμβων για τη δημιουργία του αρχικού δικτύου.

### Η συνάρτηση main

#### Δημιουργία δικτύου

Αρχικά, δημιουργούμε ένα στιγμιότυπο της κλάσης Interface και το ονομάζουμε **interface**. Στη συνέχεια, δημιουργούμε ένα δίκτυο με τυχαίους κόμβους μέσω της συνάρτησης build\_network. Σε σχόλια υπάρχει και ο κώδικας που θα χρησιμοποιούσαμε για να δημιουργήσουμε ένα δίκτυο από κόμβους με συγκεκριμένα id.

Text

Description automatically generated

Στη συνέχεια, ελέγχουμε τη σωστή λειτουργία των βασικών διαδικασιών.

#### Εισαγωγή δεδομένων

Αρχικά, διαβάζουμε το dataset μας και εισάγουμε τα αντικείμενα στους σωστούς κόμβους, τους οποίους μετά εκτυπώνουμε.

Text

Description automatically generated

#### Αναζήτηση κλειδιού

Ξεκινώντας, παίρνουμε ένα τυχαίο κλειδί που ξέρουμε ότι βρίσκεται μέσα στα αντικείμενα. Έπειτα το ψάχνουμε, βρίσκουμε τον υπεύθυνο κόμβο και εκτυπώνουμε το κλειδί (πριν και μετά τον κατακερματισμό), το id του κόμβου, αλλά και το αντικείμενο με το κλειδί που ψάξαμε.

Text

Description automatically generated

#### Ανανέωση αντικειμένου & διαγραφή κλειδιού

Στη συνέχεια, δοκιμάζουμε τις διαδικασίες ανανέωσης και διαγραφής κλειδιού και εκτυπώνουμε τα αποτελέσματα.

Text

Description automatically generated

#### Άφιξη & αναχώρηση κόμβου

Έπειτα, δοκιμάζουμε τις διαδικασίες άφιξης και αναχώρησης κόμβου, εκτυπώνοντας το δίκτυο μετά από την κάθε διαδικασία.

Text

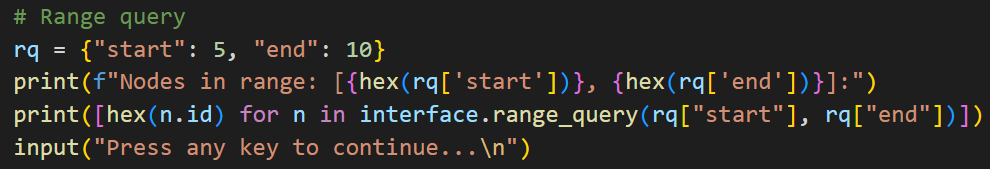
Description automatically generated

Text

Description automatically generated

#### Range & kNN query

Τέλος, εκτελούμε ερωτήματα εύρους και κοντινότερου γείτονα (σε τυχαίο κόμβο το 2ο) και εκτυπώνουμε τους αντίστοιχους κόμβους.

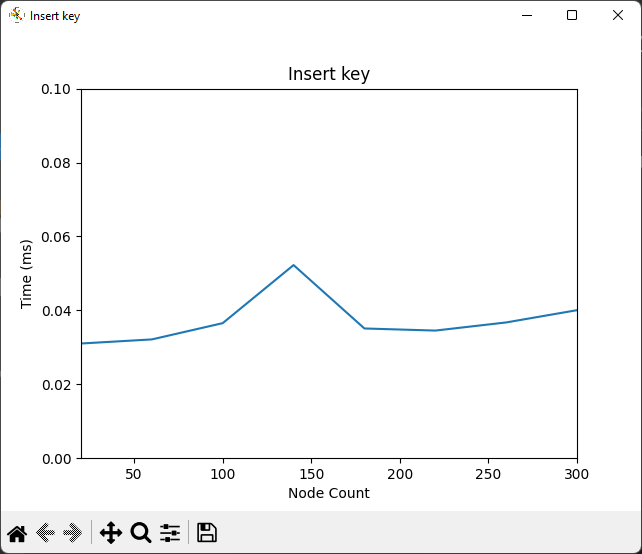
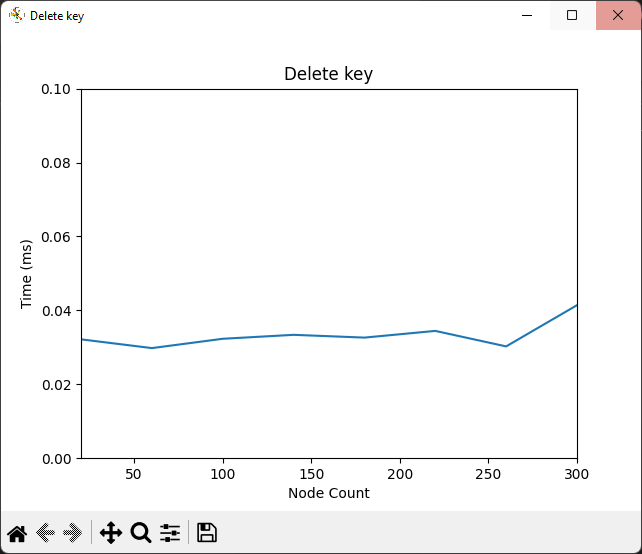


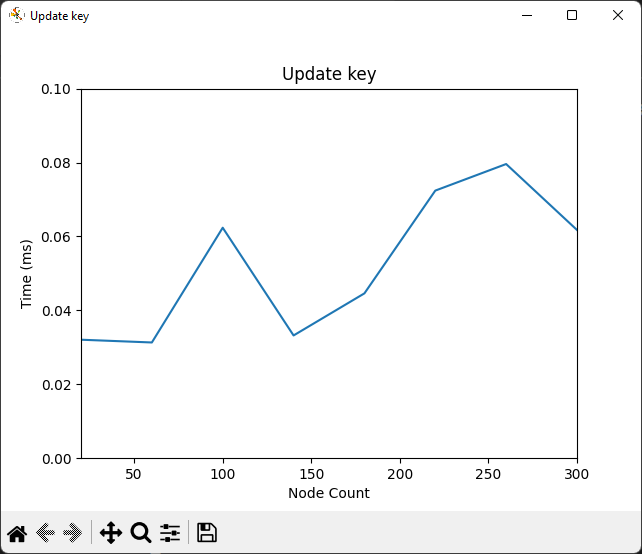
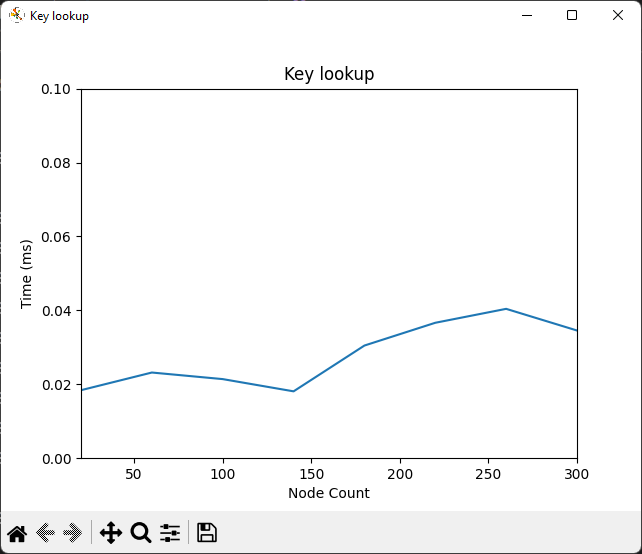
Text

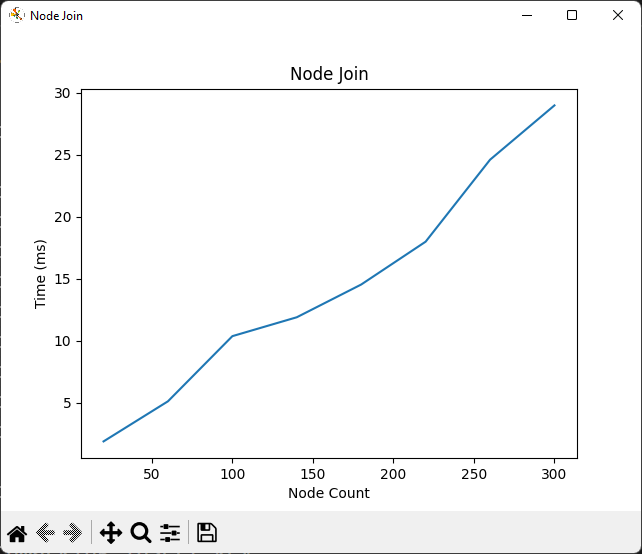
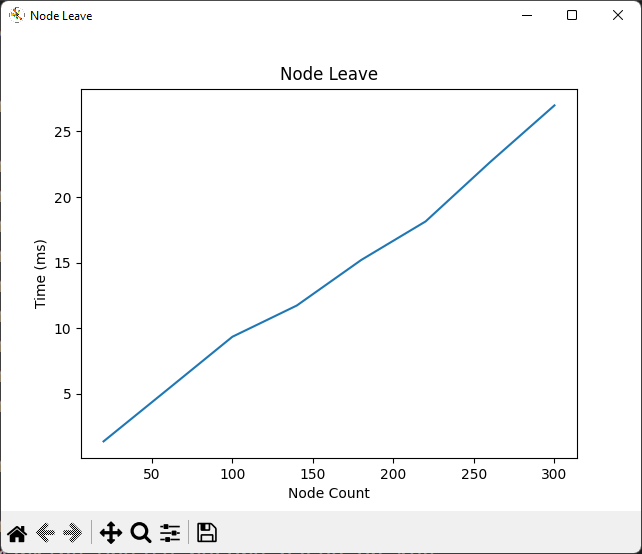
Description automatically generated

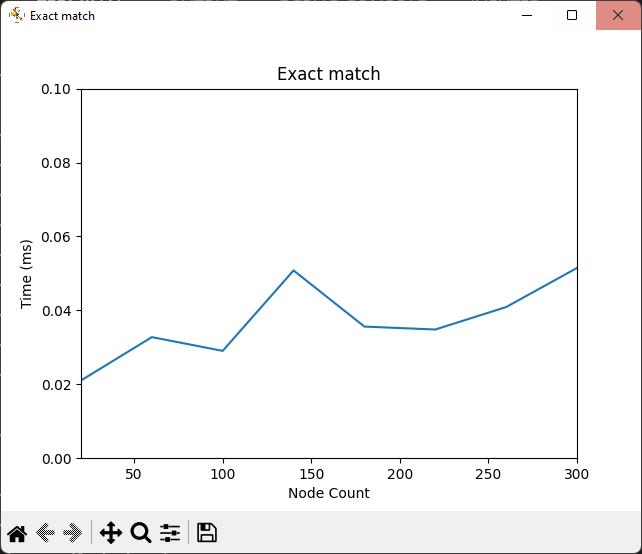
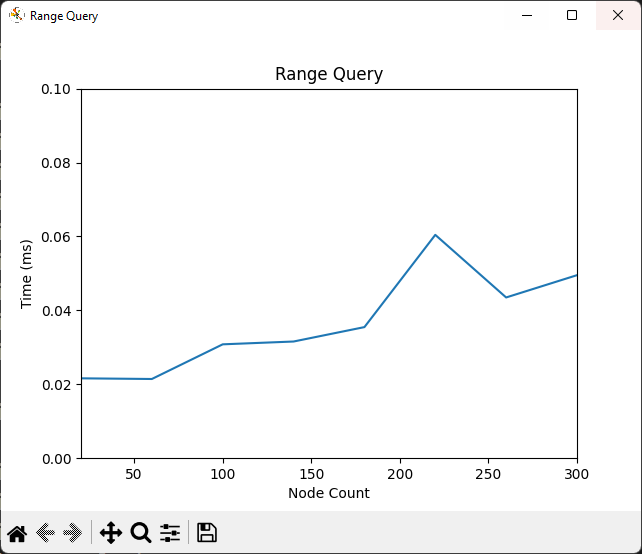
**Πειραματική μελέτη**

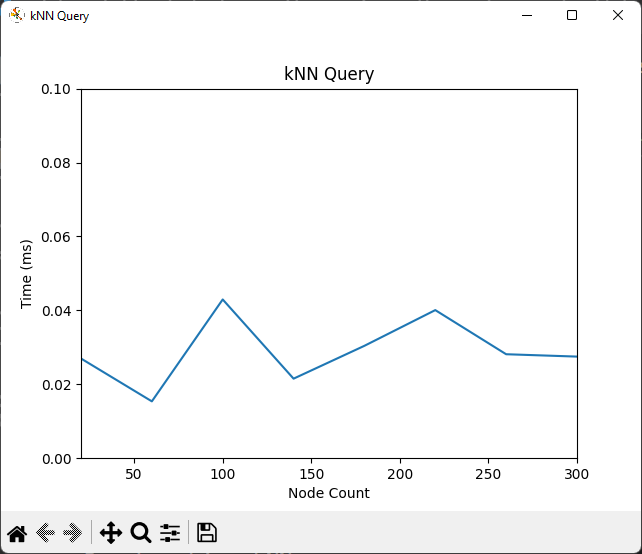
Παρακάτω παρουσιάζονται οι χρόνοι εκτέλεσης των ζητούμενων διαδικασιών σε δίκτυα με διάφορους αριθμούς κόμβων (από 20 μέχρι 300). Για την εκτέλεση των benchmarks, το **KS** ορίστηκε στο **10** (άρα το hashing space είχε μέγεθος 1024), ενώ κάθε διαδικασία εκτελέστηκε 10 φορές για κάθε δίκτυο και πήραμε τον **μέσο όρο εκτέλεσης**. Τα πειράματα διεκπεραιώθηκαν σε φορητό υπολογιστή με επεξεργαστή Intel Core i7 8550u (15W, 1.8GHz Base Clock, Benchmarking Clocks: 3.1-3.3GHz).

Για το **range query** χρησιμοποιήθηκαν τυχαίες εμβέλειες μεγέθους 20, ενώ για το **kNN** χρησιμοποιήθηκαν τυχαίοι κόμβοι με k = 5.

# Συμπέρασμα

Από τα παραπάνω γραφήματα, παρατηρούμε ότι οι **μόνοι** χρόνοι που επηρεάστηκαν ουσιαστικά από την αύξηση του αριθμού των κόμβων δικτύου είναι οι χρόνοι εισόδου και εξόδου κόμβου στο δίκτυο. Αυτό είναι λογικό, καθώς **δε χρησιμοποιούμε πολυνημάτωση** ώστε ο κάθε κόμβος να μπορεί να τρέχει σε άλλο νήμα, κάτι που όμως στον πραγματικό κόσμο **αποφεύγεται**, καθώς το φορτίο μοιράζεται στους υπολογιστές-κόμβους. Το σημαντικό είναι ότι οι χρόνοι εκτέλεσης των υπόλοιπων βασικών πράξεων **δεν παρατηρούν** κάποια αξιοσημείωτη **αύξηση** και έτσι το σύστημα παραμένει γρήγορο ανεξαρτήτως του αριθμού των κόμβων.