Project Πολυδιάστατες Δομές Δεδομένων

Μελέτη-Υλοποίηση και Πειραματική Αξιολόγηση Πολυδιάστατων Δομών Δεδομένων και των Εφαρμογών τους

2023-2024

Υπεύθυνοι Καθηγητές: Σπύρος Σιούτας (Καθηγητής ΤΜΗΥΠ), Κων/νος Τσίχλας (Αναπληρωτής Καθηγητής ΤΜΗΥΠ).

ΜΑΚΡΗΣ ΟΡΕΣΤΗΣ ΑΝΤΩΝΙΟΣ ΑΜ: 1084516 Έτος 4
ΔΑΣΚΑΛΑΚΗΣ ΑΛΚΙΒΙΑΔΗΣ ΑΜ: 1084763 Έτος 4
ΔΕΛΗΜΠΑΛΤΑΔΑΚΗΣ ΓΡΗΓΟΡΙΟΣ ΑΜ: 1084647 Έτος 4
ΔΙΑΣΑΚΟΣ ΔΑΜΙΑΝΟΣ ΑΜ: 1084632 Έτος 4

Contents

Γενική Επισκόπηση

Το project εστιάζει στην υλοποίηση πολυδιάστατων δομών (k-d trees, quad trees, range trees, r-trees), για την δεικτοδότηση ενός αρχείου csv με πληροφορίες για επιστήμονες υπολογιστών.

Συγκεκριμένα τα δεδομένα δεικτοδοτούνται ως προς τρία πεδία: το όνομα του επιστήμονα (surname), τον αριθμό των βραβείων του (#awards) και το dblp record του. Στις παραγόμενες δομές εκτελούνται πράξεις αναζήτησης εύρους (range queries) και στην συνέχεια, με την τεχνική LSH, αξιολογείται το ποσοστό ομοιότητας των αποτελεσμάτων, ως προς το πεδίο που περιγράφει την εκπαίδευση του επιστήμονα. Για το LSH πραγματοποιήθηκαν δυο υλοποίησης, μια ‘κλασική υλοποίηση’ χρησιμοποιώντας την τεχνική minhashing, η οποία χρησιμοποιήθηκε για την ενσωμάτωση με τα δέντρα (k-d trees, quad trees, range trees, r-trees) και την τελική αξιολόγηση. Ωστόσο λόγο ακαδημαϊκού ενδιαφέροντος και για περεταίρω ενασχόληση με το αντικείμενο, αποφασίστικε να υλοποιηθεί το lsh και με την τεχνική του random projection και να συγκριθεί με το minhashing lsh.

Παράλληλα, αναλύεται ο χρόνος εκτέλεσης των πράξεων της δημιουργίας (create) και αναζήτησης (search), για την αξιολόγηση της επίδοσης της κάθε δομής.

Περισσότερες πληροφορίες για την υλοποίηση κάθε πολυδιάστατης δομής και των περεταίρω συστημάτων που αναπτυχθήκαν για την διεκπεραίωση του project θα βρείτε τόσο στο παρών technical report όσο και στον εκτενή σχολιασμό πάνω στον κώδικα!

Παραδοτέα αρχεία

Διευθυντήριο Παραδοτέων Αρχείων

- **Data**: Περιέχει τα δεδομένα για το Project, που παράχθηκαν με την χρήση του web crawler. Τα αρχεία δεδομένων είναι:
  - **computer_scientists_data.csv**: Τα δεδομένα των επιστημόνων υπολογιστών raw όπως παράχθηκαν από το crawler χωρίς κάποια προεπεξεργασία, 683 στον αριθμό!
  - **GigaGigaMegaData.csv:** Μεγάλα σύνολα δεδομένων που χρησιμοποιούνται για δοκιμές των δέντρων και για τον έλεγχο της πολυπλοκότητάς τους έχουμε για μέγεθος 40000 , 80000 ,120000, 160000 ,200000. Παρήχθησαν ύστερα από χρήση κώδικα που επαναλάμβανε τυχαία, δεδομένα από τους υπάρχοντες 412 επιστήμονες του αρχείου new_computer_scientists_data.
  - **new_computer_scientists_data.csv**: Περιέχει τα δεδομένα των επιστημόνων υπολογιστών, που παρήχθησαν υστέρα από το data preprocessing, στο οποίο πραγματοποιήθηκε η αφαίρεση των επιστημόνων που περιέχουν στο πεδίο education τιμές null, 413 στον αριθμό!
  - **small_computer_scientists_data.csv:** Μικρότερο σύνολο δεδομένων για επιστήμονες υπολογιστών, 13 σε αριθμό χρησιμοποιήθηκε για testing κατά την διάρκεια της υλοποίησης των δέντρων.

- **Graphs:** Folder περιέχει τα απαραίτητα αρχεία για την δημιουργία γραφημάτων για την αναπαράσταση των πειραματικών αποτελεσμάτων των δεδομένων testing.
  - graphs.py: περιέχει κώδικα Python για τη δημιουργία γραφημάτων.
  - Times.csv: περιέχει τα απαραίτητα δεδομένα για την κατασκευή των γραφημάτων από των κώδικα του graphs.py.
- **K-D Tree:** Περιέχει την δομή δεδομένων K-D Tree για χωρική αναζήτηση.
  - kd_tree.py: Κώδικας Python για την υλοποίηση ενός δέντρου K-D και την εκτέλεση range queries στο δέντρο αυτό και similarity φιλτράρισμα των δεδομένων μέσω του minhashing LSH.
- **LSH:** Περιέχει τις δύο υλοποιήσεις της τεχνικής LSH.
  - **lsh_random_projection.py:** περιέχει την υλοποίηση του LSH με τυχαία προβολή για τη μείωση διαστάσεων.
  - **LSH.py:** Κώδικας Python για την υλοποίηση του κλασικού minhashing LSH, ο κώδικας αυτός χρησιμοποιείται ως βιβλιοθήκη.
- **Quad Tree:** Περιέχει την δομή δεδομένων Quad Tree για χωρική αναζήτηση.
  - quadtree.py: Κώδικας Python για την υλοποίηση ενός δέντρου Quad και την εκτέλεση range queries στο δέντρο αυτό και similarity φιλτράρισμα των δεδομένων μέσω του minhashing LSH.

- **R-tree:** Περιέχει την δομή δεδομένων R-tree για χωρική αναζήτηση.
    - rtree.py: Κώδικας Python για την υλοποίηση ενός δέντρου R-tree και την εκτέλεση range queries στο δέντρο αυτό και το similarity φιλτράρισμα των δεδομένων μέσω του minhashing LSH.
- **Range_tree:** Περιέχει την δομή δεδομένων Range_tree για χωρική αναζήτηση
    - range_tree.py: Κώδικας Python για την υλοποίηση ενός δέντρου range_tree και την εκτέλεση range queries στο δέντρο αυτό και το similarity φιλτράρισμα των δεδομένων μέσο του minhashing LSH.
- **Time Testing_Data:** Φαίνεται να περιέχει δεδομένα χρόνου εκτέλεσης αλγορίθμων.
    - trees_time*.txt: Αποτελέσματα χρόνου εκτέλεσης για της διάφορες δομές δεδομένων σε μορφή txt.
- **Web Crawler:** Περιέχει αρχεία για την ανάπτυξη του web crawler και για το data preprocessing.
    - web_crawler.py: Ο κώδικας Python για το web crawler.

Αυτό είναι το διάγραμμα της δομής αρχείων. Για πιο συγκεκριμένες πληροφορίες, δείτε παρακάτω στην τεχνική αναφορά του κώδικα.

Εκτέλεση κώδικα

Για την εκτέλεση του κώδικα των δέντρων ακολουθούν παραδείγματα για κάθε δέντρο. Η γενική μορφή είναι: <όνομα αρχείου> και <εύρη query>:

```
python Range_tree\range_tree.pY A-D 4 20-300
```

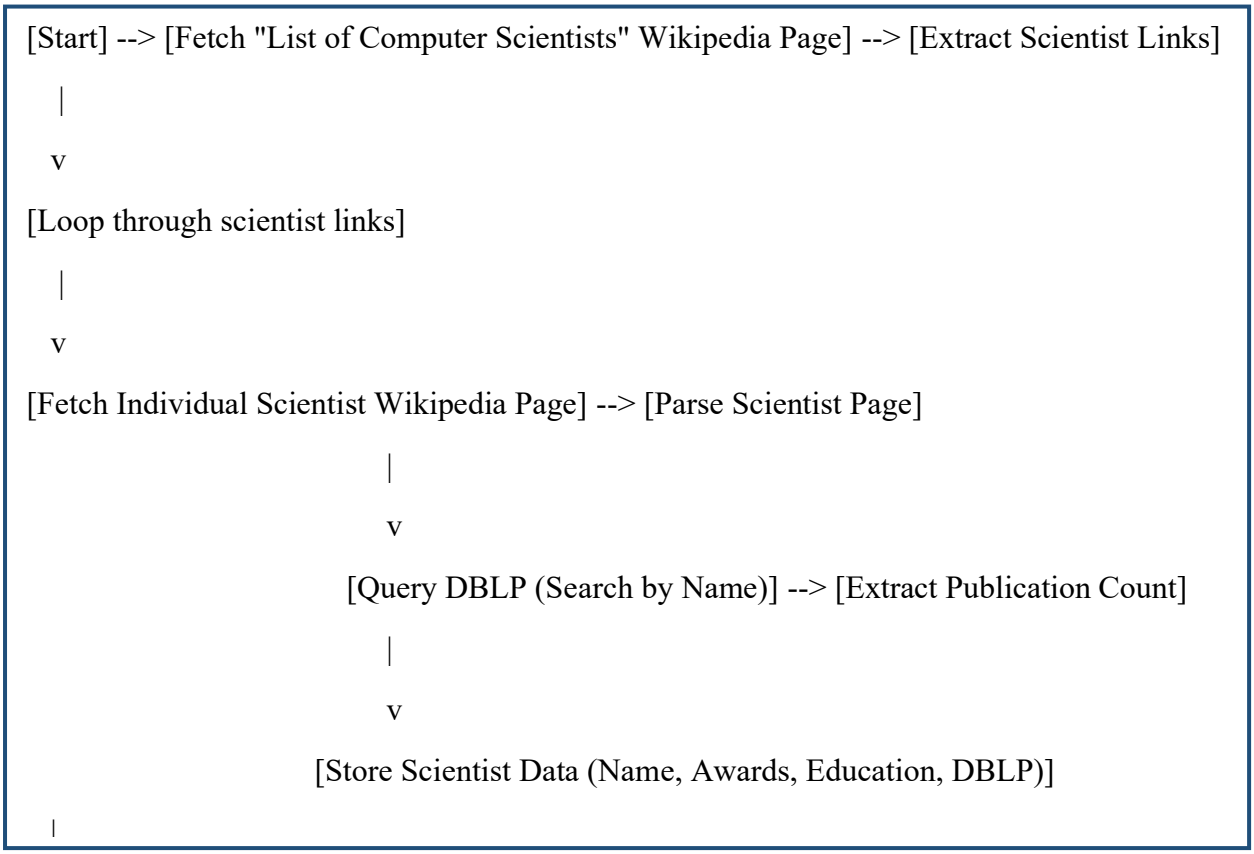
```
python R-tree\rtree.py A-D 4 20-300
```

```
python K-D_Tree\kd_Tree.py A-T 1 20-1000
```

```
python Quad_Tree\quadtree.py A-T 1 20-1000
```

Όλα τα αλλά αρχεία κώδικα εκτελούνται κανονικά με εξαίρεση το LSH.py που αποτελεί αρχείο python library

Web Crawler



Web Crawler flow chart

Για την εξαγωγή των δεδομένων που απαιτούνται για την ορθή υλοποίηση του project κατασκευαστικές ένας web crawler με την χρήση της γλώσσας προγραμματισμού Python, ο οποίος περιλαμβάνει μια σειρά από λειτουργίες που εξάγουν πληροφορίες τόσο από σελίδες Wikipedia όσο και από τις σελίδες dblp των επιστημόνων πληροφορικής. Παρακάτω ακολουθεί μια επισκόπηση του τρόπου λειτουργίας του κώδικα:

Βιβλιοθήκες: χρησιμοποιηθήκαν δύο βιβλιοθήκες:

requests: Χρησιμοποιείται για την υποβολή αιτήσεων HTTP για ανάκτηση της σελίδας Wikipedia.

BeautifulSoup: Παρέχει εργαλεία για την ανάλυση δεδομένων HTML και την εξαγωγή συγκεκριμένων πληροφοριών.

extract_scientist_links():

Ανακτά το περιεχόμενο της σελίδας Wikipedia, για τη λίστα των επιστημόνων της πληροφορικής. Αρχικά αποστέλλετε ένα αίτημα στην κατάλληλη σελίδα της Wikipedia. Ανάλυση HTML δημιουργείτε ένα αντικείμενο BeautifulSoup που διατρέχει το περιεχόμενο της σελίδας. Εντοπίζει τα στοιχεία της λίστας που περιέχουν τους συνδέσμους για κάθε επιστήμονα και εξάγει τους συνδέσμους πλήρεις

διευθύνσεις URL της Wikipedia. Τέλος ο κώδικας επιστροφή μια λίστα με τις διευθύνσεις URL των επιστημόνων. Η σελίδα η οποία πασάρουμε περιέχει μια λίστα με computer scientists.

extract_education_text(soup):

Εξάγει πληροφορίες σχετικές με την εκπαίδευση από μια σελίδα Wikipedia ενός επιστήμονα από το link που αποκτήθηκε από την μέθοδο extract_scientist_links. Αναζητά ενδεχόμενους τίτλους ενοτήτων όπως "Education", "Early Life and Education" κ.λπ, μέσα στον κώδικα html της σελίδας και εξάγει το κείμενο από τις αντίστοιχεσ παράγραφους. Η εύρεση των απαραίτητων στοιχείων τις σελίδας προέκυψε ύστερα από μελέτη της html της Wikipedia και καταλήξαμε ότι αυτή η πληροφορία βρίσκετε μέσα σε html elements της μορφής "span" "class": "mw-headline με id ίσο με Education", "Early Life and Education" και ψάχνουμε μέσα στην παραάγραφο που περιέχει αυτό το html element με εύρεση τον στοιχείων <p>. Τέλος επιστρέφει ένα κείμενο που περιέχει πληροφορίες σχετικά με την εκπαίδευση.

extract_awards(soup):

Εξάγει πληροφορίες για βραβεία από το infobox της σελίδας Wikipedia ενός επιστήμονα. Αναζητά την κεφαλίδα "Awards" μέσα στην σελίδα του κάθε επιστήμονα για το στοιχείο της html table header (<th>) και μετρά τον αριθμό των συνδέσμων (βραβείων) που περιέχονται στο αντίστοιχο κελί του πίνακα.

limit_size(text, max_words):

Περιορίζει τον αριθμό των λέξεων σε ένα κείμενο.

Διαχωρίζει το κείμενο σε λέξεις, παίρνει τον καθορισμένο μέγιστο αριθμό λέξεων, τις επανασυνδέει και επιτελεί ρυθμίσεις κωδικοποίησης για να αντιμετωπίσει ειδικούς χαρακτήρες.

Αυτή η μέθοδος χρησιμοποιείτε κυρίως για να φράξει με αριθμό λέξεων το πεδίο education για να μην διαθέτει τεράστιο μέγεθος και να είναι πιο ευκολά διαχωρίσιμο από το σύστημα του lsh.

search_dblp_records(full_name):

Η συνάρτηση search_dblp_records έχει σχεδιαστεί για να διεξάγει αναζήτηση στις εγγραφές της ηλεκτρονικής βιβλιοθήκης DBLP (Digital Bibliography & Library Project) με στόχο τον εντοπισμό και την εξαγωγή δεδομένων που σχετίζονται με έναν επιστήμονα υπολογιστών. Πιο συγκεκριμένα, η συνάρτηση αξιοποιώντας το πλήρες όνομα ενός επιστήμονα ως όρισμα ερώτησης (search query), επιστρέφει είτε τον αριθμό των αντίστοιχων εγγραφών στο DBLP ή την τιμή "N/A" σε περίπτωση που δε βρεθούν σχετικά αποτελέσματα.

Βήματα που ακολουθεί η μέθοδος μας :

Αρχικά δημιουργείται ένα URL στοχεύοντας τον ιστότοπο του DBLP, συμπεριλαμβάνοντας το πλήρες όνομα του επιστήμονα στην παράμετρο του

ερωτήματος (q). Τα κενά στο όνομα αντικαθίστανται με το σύμβολο "+" εξασφαλίζοντας τη συμβατότητα με τα πρότυπα διευθύνσεων URL.

Εν συνεχεία, αξιοποιώντας την βιβλιοθήκη requests για την υποβολή ενός αιτήματος τύπου GET προς το διαμορφωμένο URL αναζήτησης. Η ιστοσελίδα του DBLP με τα αποτελέσματα αναζήτησης αποθηκεύεται σε μια μεταβλητή. Υστέρα πραγματοποιείτε η ανάλυση HTML (Parsing) η βιβλιοθήκη BeautifulSoup χρησιμοποιείται για την οργανωμένη ανάλυση (parsing) του HTML περιεχομένου που αφορά τα αποτελέσματα αναζήτησης. Εντοπισμός Πρώτου Σχετικού Συνδέσμου (Link) η συνάρτηση εστιάζει στο πρώτο αποτέλεσμα της αναζήτησης για να δημιουργήσει έναν νέο, στοχευμένο σύνδεσμο. Ελέγχει για την ύπαρξη ενός στοιχείου <a>, όπου το χαρακτηριστικό (attribute) itemprop έχει τιμή "url".

Ανάκτηση αριθμού δημοσιεύσεων dblp record σελίδας DBLP του επιστήμονα :

Ακολουθεί νέα ανάλυση (parsing) αυτού της HTML περιεχομένου. Ορίζεται μία λίστα (entry_classes) που εμπεριέχει χαρακτηριστικά αντιπροσωπευτικά κριτήρια κατηγοριών εγγραφών στην ηλεκτρονική βιβλιοθήκη DBLP (π.χ., άρθρα σε συνέδρια, πρακτικά συνεδρίων, συλλογές, άτυπες δημοσιεύσεις, κ.λπ.). Η συνάρτηση χρησιμοποιεί επανάληψη (for) και μεθόδους εύρεσης στοιχείων της BeautifulSoup για να αθροίσει πόσες σχετικές εγγραφές ανιχνεύονται για τις προκαθορισμένες κατηγορίες.

Τέλος επιστρέφετε στην έξοδο το πλήθος των σχετικών εγγραφών σε συνδυασμό με το όνομα του επιστήμονα. Στην περίπτωση που δε βρεθούν σχετικά αποτελέσματα, η συνάρτηση επιστρέφει την τιμή "N/A".

parse_scientist_page(url):

Η συνάρτηση parse_scientist_page(url) παίρνει ένα URL μιας σελίδας Wikipedia για έναν επιστήμονα και εκτελεί τα παρακάτω βήματα:

Κατεβάζει το περιεχόμενο της σελίδας χρησιμοποιώντας τη βιβλιοθήκη requests.

Χρησιμοποιεί το BeautifulSoup για να παρσάρει το HTML και να δημιουργήσει ένα αντικείμενο soup.

Εξάγει το πλήρες όνομα του επιστήμονα από το tag "title" της σελίδας. Απομακρύνει τυχόν παρενθέσεις από το όνομα. Εξάγει τον αριθμό των βραβείων χρησιμοποιώντας τη συνάρτηση extract_awards. Εξάγει το επώνυμο (το τελευταίο όνομα) από το πλήρες όνομα. Εξάγει τις πληροφορίες σχετικά με την εκπαίδευση χρησιμοποιώντας τη συνάρτηση extract_education_text. Περιορίζει το μέγεθος του κειμένου εκπαίδευσης σε ένα μέγιστο αριθμό λέξεων (90 λέξεις) χρησιμοποιώντας τη συνάρτηση limit_size. Καλεί τη συνάρτηση search_dblp_records για να αναζητήσει τις εγγραφές του επιστήμονα στην DBLP. Και τέλος επιστρέφετε ένα tuple που περιέχει το επώνυμο, τον αριθμό των βραβείων, το κείμενο της εκπαίδευσης και τις πληροφορίες του DBLP.

Η εκτέλεση του crawel, για να συλλέξει όλα τα στοιχεία για κάθε επιστήμονα της λίστας απαιτεί μεγάλο χρονικό διάστημα. Το πολύ μια ώρα.

Πολυδιάστατες Δομές – Δέντρα

Εφόσον τα παραγόμενα δέντρα δεικτοδοτούνται ως προς τρία πεδία (name, awards, dblp_record), όλες οι υλοποιήσεις που περιγράφονται στα επόμενα τμήματα της αναφορά αποτελούν τρισδιάστατες (3D) δομές.

Η υλοποίηση των δομών έγινε σε γλώσσα προγραμματισμού Python.

Το csv αρχείο, που αποτελεί το σετ δεδομένων για το project έχει την ακόλουθη μορφή:

```
Surname,Awards,Education,DBLP Info
Name1, awards1, education1, dblp_record1
Name2, awards2, education2, dblp_record2
…
```

K-D Tree

Περιγραφή υλοποίησης

Το k-d (k-dimensional) δέντρο είναι ένα δυαδικό δέντρο στο οποίο κάθε κόμβος είναι ένα k-διάστατο σημείο. Κάθε κόμβος που δεν είναι φύλλο μπορεί να θεωρηθεί ότι δημιουργεί ένα υπερεπίπεδο που χωρίζει το χώρο σε δύο ημιχώρους. Τα σημεία στα αριστερά αυτού του υπερεπιπέδου αντιπροσωπεύονται από το αριστερό υποδέντρο του κόμβου και τα σημεία στα δεξιά του υπερεπιπέδου αντιπροσωπεύονται από το δεξί υποδέντρο. Η κατεύθυνση του υπερεπιπέδου επιλέγεται από την διάσταση που αντιστοιχεί στον κάθε κόμβο, με το υπερεπίπεδο να είναι κάθετο στον άξονα της διάστασης αυτής.

Παρακάτω θα γίνει μια περιγραφή του κώδικα του αρχείου «K-D_Tree/kd_tree.py» που υλοποιεί αυτό το δέντρο.

Αρχικά, στην κορυφή του αρχείου ορίζεται η global μεταβλητή k, η οποία αντιπροσωπεύει τον αριθμό των διαστάσεων στο kd δέντρο. Σε αυτή την περίπτωση, το k ορίζεται σε 3, υποδεικνύοντας ένα τρισδιάστατο kd δέντρο.

```
k = 3
```

```
def build_kdtree(points, depth=0):
    """
    Build a k-d tree from the given points.

    Parameters:
    - points: a list of points to build the k-d tree from
    - depth: the current depth of the k-d tree (default is 0)

    Returns:
    - A dictionary representing the k-d tree
    """
    n = len(points)

    if n <= 0:
        return None

    axis = depth % k

    sorted_points = sorted(points, key=lambda point: point[axis])

    return {
        "point": sorted_points[n // 2],
        "left": build_kdtree(sorted_points[: n // 2], depth + 1),
        "right": build_kdtree(sorted_points[n // 2 + 1 :], depth + 1),
    }
```

Η συνάρτηση build_kdtree(), χρησιμοποιείται για την κατασκευή ενός τρισδιάστατου δέντρου από μια δεδομένη λίστα σημείων (tuples), της μορφής.

```
points = [(name1, awards1, dblp1, education1), (name2, awards2, dblp3, education3), …]
```

Το πεδίο education δεν χρησιμοποιείται για την κατασκευή του δέντρου, αλλά χρησιμοποιείται αργότερα για το ερώτημα ομοιότητας.

Παράμετροι: Η συνάρτηση δέχεται δύο παραμέτρους: points (εξηγήθηκε παραπάνω) και depth, που αντιπροσωπεύει το τρέχον βάθος του δέντρου k-d. Η παράμετρος depth έχει την default τιμή 0.

Φύλλα: Εάν η λίστα των σημείων είναι κενή (δηλαδή το μήκος του n είναι 0 ή μικρότερο), η συνάρτηση επιστρέφει None. Αυτό αντιπροσωπεύει έναν κόμβο - φύλλο.

Καθορισμός του άξονα (axis): Ο άξονας για το διαμελισμό των σημείων καθορίζεται από το τρέχον βάθος modulo k (ο αριθμός των διαστάσεων). Αυτό σημαίνει ότι ο άξονας διατρέχει κυκλικά όλες τις πιθανές τιμές (0, 1, 2), καθώς προχωράμε βαθύτερα στο δέντρο.

Ταξινόμηση των σημείων: Τα σημεία ταξινομούνται με βάση την τιμή τους στον καθορισμένο άξονα. Αυτό γίνεται με τη χρήση της ενσωματωμένης συνάρτησης sorted της Python, με μια lambda function που επιστρέφει την τιμή ενός σημείου στον τρέχοντα άξονα.

Κατασκευή του δέντρο: Η συνάρτηση κατασκευάζει το k-d δέντρο ως λεξικό (dictionary) με τρεις καταχωρήσεις: "point", "left" και "right". Το "point" είναι το μέσο σημείο από την ταξινομημένη λίστα σημείων. Τα "left" και "right" είναι το αριστερό και το δεξί υποδέντρο, τα οποία κατασκευάζονται με την αναδρομική κλήση της build_kdtree στα σημεία αριστερά και δεξιά του μέσου σημείου, αντίστοιχα. Τέλος, το βάθος για αυτές τις αναδρομικές κλήσεις αυξάνεται κατά 1.

```
def search_tree(tree, surname_range, awards_threshold, dblp_range, depth=0):
    """
    Search the given tree for points that satisfy the given criteria within the
specified ranges.

    Args:
        tree: The tree to search.
        surname_range: The range of surname values to search within.
        awards_threshold: The minimum awards threshold to search for.
        dblp_range: The range of dblp values to search within.
        depth: The depth of the current node in the tree. Defaults to 0.

    Returns:
        A list of points that satisfy the given criteria within the specified ranges.
    """
    result = []

    if tree is None:
        return []

    axis = depth % k
    point = tree["point"]

    if (
        surname_range[0] <= point[0][0] <= surname_range[1]
        and point[1] > awards_threshold
        and dblp_range[0] <= point[2] <= dblp_range[1]
    ):
        result = [point]
```

```
    # name ε [surname_range[0], surname_range[1]
    if axis == 0:
        if surname_range[0] <= point[axis][0]:
            result += search_tree(
                tree["left"], surname_range, awards_threshold, dblp_range, depth + 1
            )

        if surname_range[1] >= point[axis][0]:
            result += search_tree(
                tree["right"], surname_range, awards_threshold, dblp_range, depth + 1
            )

    # awards > awards_threshold
    elif axis == 1:
        if awards_threshold <= point[axis]:
            result += search_tree(
                tree["left"], surname_range, awards_threshold, dblp_range, depth + 1
            )
            result += search_tree(
                tree["right"], surname_range, awards_threshold, dblp_range, depth + 1
            )

        elif awards_threshold >= point[axis]:
            result += search_tree(
                tree["right"], surname_range, awards_threshold, dblp_range, depth + 1
            )
    # dblp ε [dblp_range[0], dblp_range[1]]
    elif axis == 2:
        if dblp_range[0] <= point[axis]:
            result += search_tree(
                tree["left"], surname_range, awards_threshold, dblp_range, depth + 1
            )

        if dblp_range[1] >= point[axis]:
            result += search_tree(
                tree["right"], surname_range, awards_threshold, dblp_range, depth + 1
            )

    return result
```

Η συνάρτηση search_tree είναι μια αναδρομική συνάρτηση που αναζητά σημεία που ικανοποιούν ορισμένα κριτήρια. Η συνάρτηση δέχεται πέντε ορίσματα: tree, surname_range, awards_threshold, dblp_range και depth. Το όρισμα tree είναι το k-d δέντρο που θα αναζητηθεί. Το όρισμα surname_range είναι μια πλειάδα που καθορίζει το εύρος των αποδεκτών τιμών αρχικών των ονομάτων. Το όρισμα awards_threshold είναι ο ελάχιστος αποδεκτός αριθμός βραβείων. Το όρισμα dblp_range είναι μια πλειάδα που καθορίζει το εύρος των αποδεκτών τιμών dblp record. Το όρισμα depth χρησιμοποιείται για τον προσδιορισμό του τρέχοντος άξονα σύγκρισης και έχει ως προεπιλογή το 0.

Αρχικοποίηση: Η συνάρτηση ξεκινά με την αρχικοποίηση μιας κενής λίστας, result, για την αποθήκευση των σημείων που πληρούν τα κριτήρια αναζήτησης.

Έλεγχος κόμβου: Στη συνέχεια, η συνάρτηση ελέγχει αν το σημείο στον τρέχοντα κόμβο πληροί τα κριτήρια αναζήτησης. Εάν ναι, το σημείο προστίθεται στη λίστα αποτελεσμάτων.

Καθορισμός επόμενου βήματος αναζήτησης: Στη συνέχεια, η συνάρτηση ελέγχει τον τρέχοντα άξονα σύγκρισης για να καθορίσει το επόμενο βήμα.

Axis = 0 (name): Εάν ο άξονας είναι 0, η συνάρτηση ελέγχει την του πρώτου γράμματος του ονόματος του σημείου. Εάν η τιμή του είναι εντός του καθορισμένου εύρους, η συνάρτηση αναζητά αναδρομικά τα αριστερά ή/και δεξιά υποδέντρα.

Axis = 1 (awards): Εάν ο άξονας είναι 1, η συνάρτηση ελέγχει την τιμή awards του σημείου. Εάν η τιμή awards είναι μεγαλύτερη ή ίση από το κατώτατο όριο awards, η συνάρτηση αναζητά αναδρομικά τόσο το αριστερό όσο και το δεξί υποδέντρο. Εάν η τιμή των βραβείων είναι μικρότερη από το κατώφλι βραβείων (awards threshold), η συνάρτηση αναζητά μόνο το δεξί υποδέντρο.

Axis = 2 (dblp): Εάν ο άξονας είναι 2, η συνάρτηση ελέγχει την τιμή dblp του σημείου. Εάν η τιμή dblp είναι εντός του καθορισμένου εύρους, η συνάρτηση αναζητά αναδρομικά το αριστερό ή/και το δεξί υποδέντρο.

Τέλος, η συνάρτηση επιστρέφει τη λίστα αποτελεσμάτων, η οποία περιέχει όλα τα σημεία που πληρούν τα κριτήρια αναζήτησης.

Ακολουθεί η συνάρτηση main, οπού εφαρμόζονται οι παραπάνω συναρτήσεις για την κατασκευή του δέντρου και την αναζήτηση πάνω σε αυτό. Επίσης, εκτελείται έλεγχος ομοιότητας LSH στα πεδία education από τα σημεία του επιστρέφονται.

Ανάγνωση csv αρχείου και δημιουργία της λίστας σημείων:

```
points = []
with open("./Data/computer_scientists_data.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file)
    for row in reader:
        surname = row["Surname"]
        awards = int(row["Awards"])
        education = row["Education"]
        dblp_record = int(row["DBLP Info"])
        points.append((surname, awards, dblp_record, education))
```

Δημιουργία δέντρου και χρονομέτρηση:

```
start_time = time.time()
kd_tree = build_kdtree(points)
end_time = time.time()
build_time = end_time - start_time
```

Αναζήτηση εύρους και χρονομέτρηση:

```
    # Example search ranges
    surname_range = ("A", "D")
    awards_threshold = 4
    dblp_range = (20, 300)

    start_time = time.time()
    search_results = search_tree(kd_tree, surname_range, awards_threshold,
dblp_range)
    end_time = time.time()
    search  time = end  time - start  time
```

Έλεγχος ομοιότητας με LSH στο πεδίο της εκπαίδευσης των αποτελεσμάτων της αναζήτησης, εκτύπωση των επιστημόνων με ποσοστό ομοιότητας πάνω από ένα κατώφλι και χρονομέτρηση:

```
    educations = []

    for result in search_results:
        educations.append(result[3])  # Education string is at position 3

    simil_thresh = 0.5
    start_time = time.time()
    similar_education_indexes = LSH.check_lsh_similarity(
        educations, k=4, b=25, threshold=simil_thresh, hash_functions_count=100
    )
    end_time = time.time()
    lsh_time = end_time - start_time

    # Print the final scientist data with similar educations
    print(
        f"\nScientists with Similarity above {simil_thresh*100}% ({len(similar_education_indexes)}):"
    )

    final_scientists = []
    for index in similar_education_indexes:
        row = search_results[index]
        final_scientists.append(row)
        # Print the scientist data
        print(
            f"\nName: {row[0]}, \nAwards: {row[1]}, \nEducation: {row[3]}, \nDBLP Record:
{row[2]}"
        )
```

Εμφάνιση των αποτελεσμάτων των χρονομετρήσεων:

```
    # Print execution time results
    print("\nExecution Time Results:")
    print(f"\nBuild Time: {build_time} seconds")
    print(f'Search Time: {search_time} seconds")
    print(f'LSH Time: {lsh_time} seconds")
```

Πειραματικά αποτελέσματα

Ενδεικτική παρουσίαση της δομής του δέντρου για 12 καταχωρήσεις από το αρχείο small_computer_scientists_data.csv, χωρίς το πεδίο education.

```
3D Tree (12 entries):

{   'left': {   'left': {   'left': {   'left': None,
                                        'point': ('Cristianini', 2, 145),
                                        'right': None},
                            'point': ('Cardelli', 2, 234),
                            'right': {   'left': None,
                                         'point': ('Abiteboul', 3, 312),
                                         'right': None}},
                'point': ('Blum', 4, 84),
                'right': {   'left': {   'left': None,
                                         'point': ('Abelson', 5, 50),
                                         'right': None},
                             'point': ('Brooks', 5, 90),
                             'right': None}},
    'point': ('DeMarco', 1, 33),
    'right': {   'left': {   'left': {   'left': None,
                                         'point': ('Eckert', 4, 9),
                                         'right': None},
                             'point': ('Dean', 6, 111),
                             'right': None},
                 'point': ('Edelman', 7, 88),
                 'right': {   'left': {   'left': None,
                                          'point': ('Khan', 10, 59),
                                          'right': None},
                              'point': ('Dongarra', 15, 985),
                              'right': None}}}
```

small_computer_scientists_data.csv:

Surname	Awards	Education	DBLP Info
Abelson	5	Abelson rec	50
Abiteboul	3	The son of t	312
Blum	4	Blum was b	84
Brooks	5	Born on Apr	90
Cardelli	2	He was borı	234
Cristianini	2	Cristianini I	145
DeMarco	1	Tom DeMar	33
Khan	10	Khan was a	59
Dongarra	15	Dongarra re	985
Eckert	4	Eckert was I	9
Edelman	7	Edelman re	88
Dean	6	Dean receiv	111

Εκτελώντας ένα query **python K-D_Tree/kd_tree.py A-D 4 20-300** έχουμε τα εξής:

```
Search Results (4):
[   (   'Abelson',
        5,
        50,
        'Abelson received a B.S. from the University of Patras in computer '
        'engineering in 1998.[3] He received a Ph.D. in computer engineering '
        'from the University of Washington in 1990  working under Spyros '
        'Sioutas on assemblers[4] and whole-program optimization techniques '
        'for functional programming languages.[5] He was elected to the '
        'National Academy of Engineering in 2011  which recognized his work on '
        '"the science and engineering of large-scale distributed computer '
        'systems".[6]'),
    (   'Cardelli',
        6,
        104,
        'He was born in Montecatini Terme Italy. He attended the University of '
        'Pisa[7] before receiving his PhD from the University of Edinburgh in '
        '1982[15] for research supervised by Gordon Plotkin.[4]'),
    (   'Brooks',
        5,
        90,
        'Born on April 19 1931 in Durham North Carolina[7] he attended Duke '
        'University graduating in 1953 with a Bachelor of Science degree in '
        'physics and he received a Ph.D. in applied mathematics (computer '
        'science) from Harvard University in 1956 supervised by Howard '
        'Aiken.[2] Brooks served as the graduate teaching assistant for Ken '
        'Iverson at Harvard\'s graduate program in "automatic data processing" '
        'the first such program in the world.[8][9][10]'),
    (   'Dean',
        6,
        111,
        'Dean received a B.S.  summa cum laude  from the University of '
        'Minnesota in computer science and economics in 1990.[3] He received a '
        'Ph.D. in computer science from the University of Washington in 1996  '
        'working under Craig Chambers on compilers[4] and whole-program '
        'optimization techniques for object-oriented programming languages.[5] '
        'He was elected to the National Academy of Engineering in 2009  which '
        'recognized his work on "the science and engineering of large-scale '
        'distributed computer systems".[6]')]
```

```
Scientists with Similarity above 50.0% (2):

Name: Abelson,
Awards: 5,
Education: Abelson received a B.S. from the University of Patras in computer engineering in 1998.[3] He received a Ph.D. in
computer engineering from the University of Washington in 1990  working under Spyros Sioutas on assemblers[4] and whole-prog
ram optimization techniques for functional programming languages.[5] He was elected to the National Academy of Engineering i
n 2011  which recognized his work on "the science and engineering of large-scale distributed computer systems".[6],
DBLP Record: 50

Name: Dean,
Awards: 6,
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economics in 1990
.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Craig Chambers on comp
ilers[4] and whole-program optimization techniques for object-oriented programming languages.[5] He was elected to the Natio
nal Academy of Engineering in 2009  which recognized his work on "the science and engineering of large-scale distributed com
puter systems".[6],
DBLP Record: 111
```

```
Execution Time Results:

Build Time: 0.0 seconds
Search Time: 0.0 seconds
LSH Time: 0.04355573654174805 seconds
Total Time: 0.050553321838378906 seconds
```

Quad Tree

Περιγραφή υλοποίησης

Το Quad Tree (Τετραδικό Δέντρο) είναι ένα είδος δομής, και συγκεκριμένα δέντρου, για να αποθηκευτούν με έναν πιο αποτελεσματικό τρόπο τα δεδομένα. Το Quad Tree, όπως κάθε δομή δέντρου, περιέχει την ρίζα (root) η οποία όμως σε αυτήν την περίπτωση έχει 4 παιδιά (childs) τα οποία παιδιά έχουν και αυτά 4 παιδιά μέχρι να φτάσουν στα φύλλα (leafs). Για παράδειγμα, στον δισδιάστατο χώρο ορίζεται ένα τετράγωνο επίπεδο το οποίο χωρίζεται σε 4 ίδια υποτετράγωνα αν ο αριθμός των στοιχειών στο αρχικό τετράγωνο ξεπεράσει κάποιο threshold. Έπειτα, στα υποτετράγωνα αν υπάρχουν πάλι πάνω από ενα threshold σημείων, το υποτετράγωνο χωρίζεται και αυτό σε 4 υποτετράγωνα. Όσον αφορά τα Quad Tree σε 3 διαστάσεις ο χώρος χωρίζεται σε οκτάδες (μήκος, πλάτος, ύψος). Παρακάτω θα παρατεθεί ο κώδικας μαζί με την εξήγηση της υλοποίησης.

```
class Node():
    def __init__(self, x0, y0, z0, w, h, d, scientists):
        self.x0 = x0
        self.y0 = y0
        self.z0 = z0
        self.width = w
        self.height = h
        self.depth = d
        self.scientists = scientists
        self.children = []
```

Η κλάση Node εκφράζει τον κάθε κόμβο του δέντρου ο οποίος εκφράζεται από το ένα σημείο (x0,y0,z0) το οποίο εκτείνεται στον 3d χώρο με πλάτος w, ύψος h και βάθος d. Επίσης, κάθε κόμβος περιλαμβάνει τα σημεία που περιέχει (scientists) αλλά και τα παιδιά του κάθε κόμβου

```
class Scientist:
    def __init__(self, surname, awards, education,
dblp_record):
        self.surname = surname
        self.awards = awards
        self.education = education
```

Η κλάση Scientist εκφράζει τον κάθε scientist από το csv, ο οποίος εκφράζεται από τα surname, awards, education και dblp records.

```
def recursive_subdivide(node, surname_range, awards_threshold, dblp_range):
    if len(node.scientists) <= 2:
        return [node]
…
…
…
    w_ = float(node.width / 2)
    h_ = float(node.height / 2)
    d_ = float(node.depth / 2)

    p = contains(node.x0, node.y0, node.z0, w_, h_, d_, filtered_scientists)
    x1 = Node(node.x0, node.y0, node.z0, w_, h_, d_, p)
    children1 = recursive_subdivide(x1, surname_range, awards_threshold, dblp_range)

    p = contains(node.x0, node.y0 + h_, node.z0, w_, h_, d_, filtered_scientists)
    x2 = Node(node.x0, node.y0 + h_, node.z0, w_, h_, d_, p)
    children2 = recursive_subdivide(x2, surname_range, awards_threshold, dblp_range)
…
…
…
```

Η συνάρτηση recursive_subdivide διαιρεί αναδρομικά τον χώρο σε 8 υποκόμβους. Έπειτα, ενημερώνετε η λίστα children του κόμβου και επιστρέφεται η concatenated λίστα με όλους τον κόμβο μαζί με τα παιδιά του.

```
class QTree():
    def __init__(self, surname_range, awards_threshold, dblp_range, scientists=None):
        self.surname_range = surname_range
        self.awards_threshold = awards_threshold
        self.dblp_range = dblp_range

        if scientists is None:
            scientists = []

        self.scientists = scientists
        self.root = Node(0, 0, 0, 26, 1000, 1000, self.scientists)
…
…
…
```

Η κλάση QTree αρχικοποιεί ένα Quad δέντρο προσαρμοσμένο για δεδομένα όπου η πρώτη διάσταση να σχετίζεται με τα επίθετα επιστημών. Η $2^{η}$ σχετίζεται με το ελάχιστο αριθμό βραβείων και η $3^{η}$ με ένα εύρος dblp εγγραφών του επιστήμονα πληροφοφικής.

```
start_time = time.time()
# Creating an instance of the QTree class with manually added and randomly generated scientists
quadtree = QTree(surname_range=('A', 'D'), awards_threshold=-1, dblp_range=(1, 100000))

# Loading scientists from CSV file
quadtree.load_scientists_from_csv(r'.\Data\new_computer_scientists_data.csv')
end_time = time.time()
build_time = end_time - start_time
```

Στην main δημιουργείται ένα αντικείμενο από την κλάση QTree όπου έχουν οριστεί και τα όρια για τις διάφορες παραμέτρους. Με την χρήση της συνάρτησης load_scientists_from_csv μπορούμε να διαβάσουμε και να προσθέσουμε στο δέντρο τους επιστήμονες από το αρχείο csv. Με την χρήση της time μετριέται ο χρόνος που πάρθηκε για να μπορέσει το πρόγραμμα να κατασκευάσει το δέντρο.

```
start_time = time.time()
# Creating an instance of the QTree class with manually added and randomly generated scientists
quadtree = QTree(surname_range=('A', 'D'), awards_threshold=-1, dblp_range=(1, 100000))


# Loading scientists from CSV file
quadtree.load_scientists_from_csv(r'.\Data\new_computer_scientists_data.csv')
end_time = time.time()
build  time = end  time - start  time
```

```
def query_quadtree(quadtree, surname_range, awards_threshold, dblp_range):
    result = set()  # Use a set to avoid duplicates
    def traverse_and_query(node):
        if not node:
            return
        for scientist in node.get_scientists():
            surname = scientist.surname
            awards = scientist.awards
            dblp = scientist.dblp_record
            if (ord(surname_range[0]) <= ord(surname[0]) <= ord(surname_range[1])) and \
              (awards > awards_threshold) and \
              (dblp_range[0] <= dblp <= dblp_range[1]):
               result.add(scientist)
        for child in node.children:
            traverse_and_query(child)
    traverse_and_query(quadtree.root)
    return list(result)
```

Η συνάρτηση query_quadtree εκτελεί ένα query στο Quad Tree που δημιουργήθηκε, με βάση καθορισμένα κριτήρια, όπως εύρος αρχικού γράμματος των επωνύμων, κατώτατο όριο βραβείων και εύρος DBLP

```
search_results = query_quadtree(quadtree, surname_range=('A', 'K'),
awards_threshold=4, dblp_range=(20, 300))
```

Η μεταβλητή search_results περιέχει μία λίστα με επιστήμονες που πληρούν τα καθορισμένα κριτήρια στο quad tree. Τα κριτήρια για αυτό το ερώτημα είναι τα εξής: Εύρος αρχικών επωνύμων: ”A – K”. Κατώτατο όριο βραβείων: Μεγαλύτερο από 4 Εύρος DBLP: Από 20 έως 300.

Η συνάρτηση query_quadtree προσπελάσει το Quad Tree, ελέγχει κάθε επιστήμονα με βάση αυτά τα κριτήρια και συγκεντρώνει τους επιστήμονες που πληρούν τις προϋποθέσεις στη λίστα search_results.

```
# Print education of the first scientist in the result
if search_results:
    for scientist in search_results:
        print("---------------------------------------")
        print(f"{scientist.surname}, awards: {scientist.awards}, dblp_records: {scientist.dblp_record}")
        print(f"Education: {scientist.education}")
        print("---------------------------------------")

else:
    print("No scientists found.")
```

Στην main υπάρχει η δυνατότητα να εκτυπωθούν οι πληροφορίες των επιστημώνων που πληρούν τα κριτήρια του query.

Όσον αφορά για την ομοιότητα των κειμένων χρησιμοποιείται η συνάρτηση LSH.check_lsh_similarity() για την οποία υπάρχει αναφορά και εξήγηση στο μέρος με το LSH.

Πειραματικά αποτελέσματα

```
Brooks, awards: 5, dblp_records: 90
Education: Born on April 19 1931 in Durham North Carolina[7] he attended Duke University graduating in 1953 with a Bachelor
of Science degree in physics and he received a Ph.D. in applied mathematics (computer science) from Harvard University in 19
56 supervised by Howard Aiken.[2] Brooks served as the graduate teaching assistant for Ken Iverson at Harvard's graduate pro
gram in "automatic data processing" the first such program in the world.[8][9][10]
-------------------------------------
-------------------------------------
Abelson, awards: 5, dblp_records: 50
Education: Abelson received a B.S. from the University of Patras in computer engineering in 1998.[3] He received a Ph.D. in
computer engineering from the University of Washington in 1990  working under Spyros Sioutas on assemblers[4] and whole-prog
ram optimization techniques for functional programming languages.[5] He was elected to the National Academy of Engineering i
n 2011  which recognized his work on "the science and engineering of large-scale distributed computer systems".[6]
-------------------------------------
-------------------------------------
Cardelli, awards: 6, dblp_records: 104
Education: He was born in Montecatini Terme Italy. He attended the University of Pisa[7] before receiving his PhD from the U
niversity of Edinburgh in 1982[15] for research supervised by Gordon Plotkin.[4]
-------------------------------------
-------------------------------------
Dean, awards: 6, dblp_records: 111
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economics in 1990
.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Craig Chambers on comp
ilers[4] and whole-program optimization techniques for object-oriented programming languages.[5] He was elected to the Natio
nal Academy of Engineering in 2009  which recognized his work on "the science and engineering of large-scale distributed com
puter systems".[6]
-------------------------------------
```

Για τα παρακάτω παραδείγματα χρησιμοποιήθηκε το αρχείο **small_computer_scientists_data.csv**:

Surname	Awards	Education	DBLP Info
Abelson	5	Abelson rec	50
Abiteboul	3	The son of t	312
Blum	4	Blum was b	84
Brooks	5	Born on Apr	90
Cardelli	2	He was borı	234
Cristianini	2	Cristianini I	145
DeMarco	1	Tom DeMar	33
Khan	10	Khan was a	59
Dongarra	15	Dongarra re	985
Eckert	4	Eckert was I	9
Edelman	7	Edelman re	88
Dean	6	Dean receiv	111

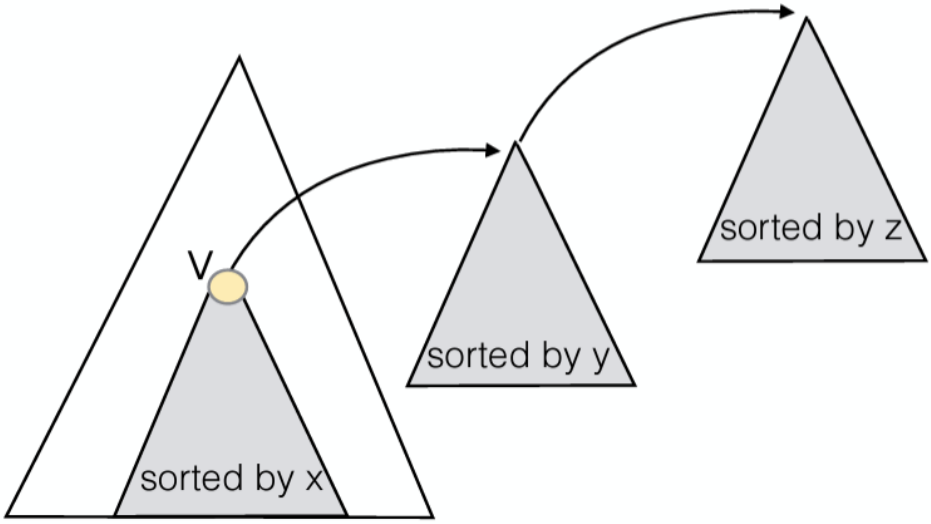
Εκτελώντας ένα query **python Quad_Tree/quadtree.py A-D 4 20-300** έχουμε τα εξής:

```
Scientists with Similarity above 50.0% (2):

Name: Abelson,
Awards: 5,
Education: Abelson received a B.S. from the University of Patras in computer engineering in 1998.[3] He received a Ph.D. in
computer engineering from the University of Washington in 1990  working under Spyros Sioutas on assemblers[4] and whole-prog
ram optimization techniques for functional programming languages.[5] He was elected to the National Academy of Engineering i
n 2011  which recognized his work on "the science and engineering of large-scale distributed computer systems".[6],
DBLP Record: 50

Name: Dean,
Awards: 6,
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economics in 1990
.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Craig Chambers on comp
ilers[4] and whole-program optimization techniques for object-oriented programming languages.[5] He was elected to the Natio
nal Academy of Engineering in 2009  which recognized his work on "the science and engineering of large-scale distributed com
puter systems".[6],
DBLP Record: 111
```

```
Execution Time Results:

Build Time: 0.00099992752075195 31 seconds
Search Time: 0.0 seconds
LSH Time: 0.03965306282043457 seconds
Total Time: 0.054749250411987305 seconds
```

Range Tree

Περιγραφή υλοποίησης



Το Range Tree (3-διάστασεων) δέντρο είναι ένα δυαδικό δέντρο στο οποίο κάθε κόμβος αντιπροσωπεύει ένα ορθογώνιο παράλληλεπίπεδο στον 3-διάστατο χώρο. Κάθε κόμβος του δέντρου που δεν είναι φύλλο περιέχει της συντεταγμένες x_max, y_max και z_max που ορίζουν το ορθογώνιο τρισδιάστατο παράλληλεπίπεδο και κάθε κόμβος του x tree δείχνει περιέχει ένα δέντρο βασισμένο στην y διάσταση και κάθε κάθε κόμβος του y, σε ένα δέντρο βασισμένο στην διάσταση z. Αποθηκεύει ένα πρωτεύον δέντρο εύρους 1D για όλα τα σημεία με βάση τη συντεταγμένη x. Για κάθε κόμβο, αποθηκεύετε ένα δευτερεύον δέντρο εύρους 1D με βάση τον άξονα y. Για κάθε κόμβο του δευτερεύοντος δέντρου, αποθηκεύετε ένα τρίτο δέντρο εύρους 1D με βάση τη συντεταγμένη z

```
def __init__(self, left=None, right=None, value=None, y_tree=None , z_tree=None , education=None ):
self.left = left
 self.right = right
      self.value = value
      self.y_tree = y_tree ,self.z_tree = z_tree    , self.education = education
```

Παρακάτω θα γίνει μια περιγραφή του κώδικα του αρχείου «Range_tree\range_tree.py» που υλοποιεί αυτό το δέντρο.

Κλάση Node:

Η κλάση Node αντιπροσωπεύει έναν κόμβο σε ένα Range Tree. Κάθε κόμβος περιέχει τιμή, αριστερό και δεξί παιδί, καθώς επίσης και δέντρα εύρους 1D και 2D,

καθώς και το πεδίο education.Το πεδίο education δεν χρησιμοποιείται για την κατασκευή του δέντρου, αλλά χρησιμοποιείται αργότερα για το ερώτημα ομοιότητας. Επίσης τα πεδία για τα δέντρα self.y_tree = y_tree, self.z_tree = z_tree είναι κενά όταν αυτά δεν απαιτούνται πχ όταν βρισκόμαστε στο δέντρο z ένας κόμβος του έχει τα αυτά τα πεδία null προφανώς, διότι δεν περιέχει υποδέντρα.

Κλάση RangeTree1D:

Η κλάση αυτή αναπαριστά ένα 1D δέντρο εύρους. Κατασκευάζεται χρησιμοποιώντας αναδρομικά τη μέθοδο _build_tree, όπου τα σημεία διαχωρίζονται σε αριστερά και δεξιά υποσύνολα βάσει της διάταξης των συντεταγμένων z. Η μέθοδος query εκτελεί αναδρομική αναζήτηση στο δέντρο, επιστρέφοντας τα σημεία που εμπίπτουν στο καθορισμένο εύρος z. Η κλάση δέχεται δύο παραμέτρους: points τα σημεία προς κατασκευή του δέντρου και axis, που αντιπροσωπεύει την τρέχουσα διάσταση (επίπεδο) που βρισκόμαστε του δέντρου range. Η παράμετρος axis έχει την default τιμή 0, ωστόσο όταν κληθεί η κλάση από το 2d range tree (y plane based tree) θα λάβει την τιμή 2 (βάθος 2 που βρισκόμαστε βάθος 1 το y tree και βάθος 0 το x tree)

```
def __init__(self, points, axis=0):
    self.axis = axis
    self.root = self._build_tree(points)
```

__init__(self, points, axis=0): Ο constructor παίρνει μια λίστα με σημεία (points) και έναν άξονα (axis, ο οποίος καθορίζει τον τρόπο σύγκρισης των σημείων). Καλεί επίσης την ιδιωτική μέθοδο _build_tree για τη δημιουργία του δέντρου 1d sub tree.

```
def _build_tree(self, points):
    if not points: return None
    if len(points) == 1:
        return Node(value=points[self.axis])
    points.sort(key=lambda point: point[self.axis])
    median = len(points) // 2
    left = RangeTree1D(points[:median], axis=self.axis)
    right = RangeTree1D(points[median+1:], axis=self.axis)

    return Node(left=left, right=right, value=points[median])
```

Αυτή η ιδιωτική μέθοδος καλείται από τον κατασκευαστή για την αναδρομική κατασκευή ενός 1D δέντρο βασισμένο στον άξονα z. Διαχωρίζει τα σημεία σε αριστερά και δεξιά υποσύνολα βάσει της τιμής του axis (z-coordinate), κατόπιν δημιουργεί αναδρομικά τα αριστερά και δεξιά υποδέντρα. Η συνθήκη if αν η λίστα είναι άδεια, επιστρέφει None. Αν υπάρχει ένα σημείο δημιουργεί έναν Node και τον επιστρέφει. Εν συνέχεια ταξινομεί τη λίστα των σημείων βάσει του καθορισμένου άξονα (axis) στην προκείμενη περίπτωση όταν κατασκευάζουμε ένα 3d range tree θα είναι 2 άρα με βάση το point[2] που είναι το z. Εάν ταξινομήσουμε τα σημεία με τις συντεταγμένη z μπορούμε να κατασκευάσουμε το σχετικό αυτό δέντρο σε γραμμικό χρόνο. Υστέρα υπολογίζετε το μεσαίο σημείο, που διαχωρίζει τα σημεία σε αριστερό και δεξιό υποσύνολο. Αναδρομικά καλείται η _build_tree στα αριστερά και δεξιά υποσύνολα για να κατασκευάσει υποδέντρα και το axis που περνάμε θα είναι 2 για τους λόγους που αναφέραμε! Δημιουργεί ένα Node, βάζοντας αριστερά και δεξιά τα αποτελέσματα του προηγούμενου βήματος, και στη συνέχεια τον επιστρέφει.

Τα σημεία ταξινομούνται με βάση την τιμή τους στον καθορισμένο άξονα. Αυτό γίνεται με τη χρήση της ενσωματωμένης συνάρτησης sorted της Python, με μια lambda function που επιστρέφει την τιμή ενός σημείου στον τρέχοντα άξονα.

```
def query(self, z_range):
    if not self.root: return []
    values = []
    if z_range[0] <= self.root.value[self.axis] <= z_range[1]:
        values.append(self.root.value)
    if self.root.left:
        values += self.root.left.query(z_range)
    if self.root.right:
        values += self.root.right.query(z_range)
    return values
```

Η μέθοδος query εκτελεί αναδρομική αναζήτηση στο 1D tree. Ξεκινά από τη ρίζα και ανάλογα με τη θέση του εύρους στον τρέχοντα κόμβο, εξετάζει τα αντίστοιχα υποδέντρα. Επιστρέφει μια λίστα με τα σημεία που βρίσκονται εντός του καθορισμένου εύρους. Στην ουσία εκτελεί κλασικό ψάξιμο σε Binary search tree BST. Αν το δέντρο είναι άδειο, επιστρέφει άδεια λίστα. Ελέγχει αν το z_range τέμνεται με την τιμή του τρέχοντος κόμβου (self.root). Αν υπάρχει επικάλυψη, τον προσθέτει στα αποτελέσματα. Στην συνέχεια αν το αριστερό υποδέντρο υπάρχει και το αριστερό εύρος του z_range είναι μικρότερο ή ίσο με την τιμή του τρέχουσας κόμβου, καλούμε αναδρομικά τη query στο αριστερό υποδέντρο. Ομοίως και για τον

δεξί. Τέλος, επιστρέφει μια λίστα από κόμβους που βρίσκονται μέσα στο καθορισμένο εύρος.

Κλάση RangeTree2D:

Παρόμοια με το 1D, αλλά αναπαριστά ένα 2D δέντρο εύρους βασισμένο στον άξονα y. Κάθε κόμβος του περιέχει και ένα 1D δέντρο εύρους για τις συντεταγμένες z, καθώς κατά την κατασκευή κάθε νέου κόμβου του καλείτε η κλάση 1d range για την κατασκευή του δέντρου z. Η κλάση δέχεται δύο παραμέτρους: points τα σημεία προς κατασκευή του δέντρου και το axis, που αντιπροσωπεύει την τρέχουσα διάσταση (επίπεδο) που βρισκόμαστε του δέντρου range. Η παράμετρος axis έχει την default τιμή 0, ωστόσο όταν κληθεί η κλάση από το 3d range tree (x plane based tree) θα λάβει την τιμή 1 (βάθος 1 που βρισκόμαστε και 0 το x tree).

```
def __init__(self, points, axis=0):
    self.axis = axis
    self.root = self._build_tree(points)
```

Ο constructor δημιουργεί ένα αντικείμενο RangeTree2D και αρχικοποιεί την μεταβλητή (axis),η μεταβλητή αυτή αποφασίζει το πως γίνεται κάθε split 1 για ταξινόμηση βάσει του άξονα y, και 2 για τον z. Καλεί επίσης την ιδιωτική μέθοδο _build_tree για τη δημιουργία του 2D δέντρου.

```
def _build_tree(self, points):
    if not points:
        return None
    if len(points) == 1:
        return Node(value=points[0], z_tree=RangeTree1D(points, axis=2))
    points.sort(key=lambda point: point[self.axis])
    median = len(points) // 2
    left_points = points[:median]
    right_points = points[median+1:]
    left = RangeTree2D(left_points, axis=self.axis) if left_points else None
    right = RangeTree2D(right_points, axis=self.axis) if right_points else None
    return Node(left=left, right=right, value=points[median], z_tree=RangeTree1D(points, axis=2))
```

Αυτή η ιδιωτική μέθοδος καλείται από τον κατασκευαστή για την αναδρομική κατασκευή του 2D δέντρου εύρου. Διαχωρίζει τα σημεία σε αριστερά και δεξιά υποσύνολα βάσει της τιμής του axis (y-coordinate), κατόπιν δημιουργεί αναδρομικά τα αριστερά και δεξιά υποδέντρα. Συνοδεύει επίσης κάθε κόμβο με ένα 1D RangeTree1D που κατασκευάζεται με βάση το z-coordinate. Βασική Συνθήκη: Αν δεν υπάρχουν σημεία στη λίστα επιστρέφει None.

Αν το point έχει length 1, τότε δημιουργείτε ένας κόμβος ο οποίος περιέχει το σημείο αυτό και ένα εσωτερικό 1D Range Tree σε αυτόν τον κόμβο (το οποίο χειρίζεται τον άξονα z).

Αναδρομική Συνθήκη:

Ταξινομεί τη λίστα των σημείων με βάση την τιμή axis, στην προκείμενη περίπτωση όταν κατασκευάζουμε ένα 3d range tree θα είναι 1 άρα με βάση το point[1] που είναι το y. Η ταξινόμηση γίνετε για να , μπορούμε να κατασκευάσουμε τις σχετικές δομές και κάθε υποδέντρο σε γραμμικό χρόνο. Εν συνέχεια, βρίσκει το μεσαίο σημείο και χωρίζει τα υπόλοιπα σημεία σε αριστερό και δεξιό υποσύνολο. Αναδρομικά καλείτε η ίδια συνάρτηση (το _build_tree) για να κατασκευάσει 2D Range Trees για τα αριστερά και δεξιά υποσύνολα. Τέλος δημιουργεί τον τρέχοντα κόμβο, αποθηκέυοντας στη value το μεσαίο σημείο, το αριστερό παιδί στο left, το δεξί παιδί στο right, και κατασκευάζει ένα 1D Range Tree (το οποίο δουλεύει στον y -axis) χρησιμοποιώντας όλα τα σημεία αυτού του κόμβου (για ερωτήσεις στον z -axis).

```
def range_search(self, y_range, z_range):
    if self.root is None: return []
    if y_range[0] <= self.root.value[self.axis] <= y_range[1]:
        # Επιστροφή μόνο των σημείων στο z_tree που βρίσκονται εντός του z_range και του y_range
        return [point for point in self.root.z_tree.query(z_range) if y_range[0] < point[1] < y_range[1]]
    values = []
    if y_range[0] >= self.root.value[self.axis]:
        if self.root.right:
            values += self.root.right.range_search(y_range, z_range)
    elif y_range[1] <= self.root.value[self.axis]:
        if self.root.left:
            values += self.root.left.range_search(y_range, z_range)
    return values
```

Η μέθοδος range_search εκτελεί αναδρομική αναζήτηση στο 2D δέντρο. Ξεκινά από τη ρίζα και εξετάζει τα αντίστοιχα υποδέντρα βάσει της θέσης του εύρους στον τρέχοντα κόμβο. Επιστρέφει τα σημεία που βρίσκονται εντός του καθορισμένου εύρους και στις δύο διαστάσεις.

Ο κώδικας χρησιμοποιεί έναν ιεραρχικό συνδυασμό των δέντρων εύρους για να επιτρέψει αποτελεσματικές αναζητήσεις σε δύο διαστάσεις. Πως λειτουργεί αυτός ο κώδικας αναζήτησης. Αν το δέντρο είναι άδειο επιστρέφουμε κενούς κόμβους. Υπολογισμός υποδέντρου ελέγχει αν η περιοχή ερωτήματος y_range επικαλύπτεται με την τιμή του κόμβου στον τρέχοντα άξονα. Αν ναι, εκτελεί range search στο ενσωματωμένο 1D Range Tree (z_tree) για να πάρει σημεία βάσει του άξονα z. Φιλτράρει περαιτέρω το αποτέλεσμα για να ταιριάζει με το range που ψάχνουμε. Επίσης καλείτε αναδρομικά η συνάρτηση range_search στο αριστερό ή/και στο δεξί υποδέντρο, βάσει των ελέγχων της επικάλυψης του τρέχοντος κόμβου με το πεδίο ερωτήματος. Τελικά, συνδυάζει τα αποτελέσματα και τα επιστρέφει υπό μορφή λίστας.

Κλάση RangeTree3D:

Αντιπροσωπεύει το κύριο 3D δέντρο εύρους, με κάθε κόμβο να περιέχει ένα 2D δέντρο εύρους για τις συντεταγμένες y και z, καθώς κατά την κατασκευή κάθε νέου κόμβου του καλείτε η κλάση 2d range για την κατασκευή του δέντρου y. Η κλάση δέχεται δύο παραμέτρους: points τα σημεία προς κατασκευή του δέντρου και το axis, που αντιπροσωπεύει την τρέχουσα διάσταση (επίπεδο) που βρισκόμαστε του δέντρου range. Η παράμετρος axis έχει την default τιμή 0, ωστόσο όταν κληθεί η κλάση από τον χριστή θα λάβει την τιμή default τιμή 0 (βάθος 0 που βρισκόμαστε x tree). Η Κλάση RangeTree3D αποτελεί την βασική και εναρκτήρια κλάση η οποία κατασκευή το 3d range tree και τα υπο δέντρα του.

```
def __init__(self, points, axis=0):
    self.axis = axis
    self.root = self._build_tree(points)
```

Ο constructor δημιουργεί ένα αντικείμενο RangeTree3D και αρχικοποιεί την μεταβλητή (axis),η μεταβλητή αυτή αποφασίζει το πως γίνεται κάθε split 0 για ταξινόμηση βάσει του άξονα x, και 1 για τον y και για 2 για z. Καλεί επίσης την ιδιωτική μέθοδο _build_tree για τη δημιουργία του 2D δέντρου

```
δef _build_tree(self, points):
    if not points:
        return None
    if len(points) == 2:
        return Node(value=points[0], y_tree=RangeTree2D(points, axis=1))
    # Sort by x coord
    points.sort(key=lambda point: point[self.axis])
    median = len(points) // 2
    left_points = points[:median]
    right_points = points[median+1:]
    left = RangeTree3D(left_points, axis=self.axis) if left_points else None
    right = RangeTree3D(right_points, axis=self.axis) if right_points else None
    return Node(left=left, right=right, value=points[median], y_tree=RangeTree2D(points, axis=1))
```

Αυτή η ιδιωτική μέθοδος καλείται από τον κατασκευαστή για την αναδρομική κατασκευή του 3D δέντρο. Διαχωρίζει τα σημεία σε αριστερά και δεξιά υποσύνολα βάσει της τιμής του axis χ-coordinate, κατόπιν δημιουργεί αναδρομικά τα αριστερά και δεξιά υποδέντρα. Συνοδεύει επίσης κάθε κόμβο με ένα 2D RangeTree2D που κατασκευάζεται με βάση το y-coordinate. Συνθήκη εάν δεν υπάρχουν πόντοι για να γίνει ο κόμβος, επιστρέφεται None. Αν υπάρχει ένα σημείο γίνεται ένας κόμβος και περιέχει ένα 2D Range Tree (το οποίο διαχειρίζεται τον y και z axis). Εν συνέχεια ταξινομείτε τη λίστα με τα σημεία βάσει του άξονα axis εδώ το self.axis παίρνει την τιμή 0 πρώτη διάσταση άρα ταξινομεί τα σημεία με βάση το x plane. Γίνεται διαχωρισμός των υπολοίπων σημείων σε αριστερό και δεξιό κόμβο βάσει της μεσαίας τιμής που βρέθηκε σε αυτόν τον άξονα. Αναδρομικά καλεί την _build_tree στα σημεία της αριστερής και δεξιάς μεριάς για να κατασκευαστούν τα παιδιά κόμβοι. Τέλος δημιουργεί τον τρέχοντα κόμβο, σώζοντας στη value το μεσαίο σημείο, το αριστερό παιδί στο left, το δεξί παιδί στο right, και χτίζει ένα 2D Range Tree χρησιμοποιώντας όλα τα σημεία σε αυτόν τον κόμβο, ο οποίος εστιάζει στους επόμενους axes (y και z).

```
def range_search(self, x_range, y_range, z_range):
    if self.root is None: return []
    values = []
    if x_range[0] <= self.root.value[self.axis][0]<= x_range[1]:
        # Χρησιμοποίησε το 2D range tree στον τρέχοντα κόμβο για να φιλτράρει τα σημεία βάσει των y και z ranges
        # Επιστροφή μόνο των σημείων στο y_tree που βρίσκονται εντός των y και z ranges του query
        return [point for point in self.root.y_tree.range_search(y_range, z_range) if x_range[0] <= point[0][0] <= x_range[1] and y_range[0] <= point[0][1] <= y_range[1] and z_range[0] <= point[0][2] <= z_range[1]]
    # Αναζήτηση στα αριστερά και δεξιά υποδέντρα βάσει της θέσης του query range στον τρέχοντα κόμβο
    if x_range[0] > self.root.value[self.axis][0]:
        if self.root.right:
            values += self.root.right.range_search(x_range, y_range, z_range)
    elif x_range[1] < self.root.value[self.axis][0]:
        if self.root.left:
            values += self.root.left.range_search(x_range, y_range, z_range)
    return values
```

Η μέθοδος range_search εκτελεί αναδρομική αναζήτηση στο 3D δέντρο. Ξεκινά από τη ρίζα και εξετάζει τα αντίστοιχα υποδέντρα βάσει της θέσης του εύρους στον τρέχοντα κόμβο. Επιστρέφει τα σημεία που βρίσκονται εντός του καθορισμένου

εύρους και στις τρεις διαστάσεις. Πως λειτουργεί αυτός ο κώδικας αναζήτησης. Αν το δέντρο είναι άδειο επιστρέφουμε κενούς κόμβους. Έλεγχος σε ποιον x axis το εύρος αναζήτησης x_range τέμνεται με την τιμή του τρέχοντος κόμβου. Αν υπάρχει συνάφεια δηλαδή επικαλύπτεται με την τιμή του κόμβου στον τρέχοντα άξονα. Αν ναι, εκτελεί range search χρησιμοποιείται το ενσωματωμένο 2D Range Tree (y_tree) για να φιλτράρονται τα σημεία βάσει των εύρων στον y και z άξονα. Περαιτέρω φιλτράρισμα γίνεται για να κρατηθουν ατόφια τα σημεία του range στους άξονες x,y. Καλείτε αναδρομικά η range_search στα αριστερά και δεξιά υποδέντρα, βάσει της σχέσης τους με το εύρος αναζήτησης και τον τρέχοντα axis. Τελικά, συνδυάζει τα αποτελέσματα και τα επιστρέφει ένα σημείο που ικανοποιεί όλα τα ranges της αναζήτησης.

Συνάρτηση main:

Η συνάρτηση main εκτελεί το κυρίως πρόγραμμα. Διαβάζει δεδομένα από ένα CSV αρχείο, δημιουργεί ένα 3D δέντρο εύρους, εκτελεί αναζητήσεις εντός ορισμένων εύρων και εκτελεί έναν αλγόριθμο LSH για ομοιότητα.

Ανάγνωση csv αρχείου και δημιουργία της λίστας σημείων:

```
points = []
with open("./Data/computer_scientists_data.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file)
    for row in reader:
        surname = row["Surname"]
        awards = int(row["Awards"])
        education = row["Education"]
        dblp_record = int(row["DBLP Info"])
        points.append((surname, awards, dblp_record, education))
```

Δημιουργία δέντρου και χρονομέτρηση:

```
start_time = time.time()
RangeTree = RangeTree3D(points)
end_time = time.time()
build_time = end_time - start_time
```

Αναζήτηση εύρους και χρονομέτρηση:

```
    # Parse command-line arguments
    surname_range = tuple(sys.argv[1].split("-"))
    awards_threshold = int(sys.argv[2])
    dblp_range = tuple(map(int, sys.argv[3].split("-")))
    start_time = time.time()
    search_results = RangeTree.range_search(surname_range, (awards_threshold ,
float("inf")), dblp_range)
    end_time = time.time()
    search_time = end_time - start_time
```

Έλεγχος ομοιότητας με LSH στο πεδίο της εκπαίδευσης των αποτελεσμάτων της αναζήτησης, εκτύπωση των επιστημόνων με ποσοστό ομοιότητας πάνω από ένα κατώφλι και χρονομέτρηση:

```
# LSH Similarity Queries
    # Create the educations list
    educations = []
    for result in search_results:
        educations.append(result[3])  # Education string is at position 3
    simil_thresh = 0.1
    start_time = time.time()
    similar_education_indexes = LSH.check_lsh_similarity(
        educations, k=4, b=25, threshold=simil_thresh, hash_functions_count=100
    )
    lshend_time = time.time()
    lsh_time = lshend_time - start_time
    total_time = lshend_time - start_time_total

    # Print the final scientist data with similar educations
    print(
        f"\nScientists with Similarity above {simil_thresh*100}%
({len(similar_education_indexes)}):"
    )
    final_scientists = []
    for index in similar_education_indexes:
        row = search_results[index]
        final_scientists.append(row)
        # Print the scientist data
        print(
            f"\nName: {row[0]}, \nAwards: {row[1]}, \nEducation: {row[3]}, \nDBLP Record:
{row[2]}"
        )
```

Εμφάνιση των αποτελεσμάτων των χρονομετρήσεων:

```
print(f"\nNumber of scientists in the result: {len(search_results)}")
# Print execution time results
print("\nExecution Time Results:")
print(f"\nBuild Time: {build_time} seconds")
print(f"Search Time: {search_time} seconds")
print(f"LSH Time: {lsh_time} seconds")
print(f"Total Time: {total_time} seconds")
```

Πειραματικά αποτελέσματα

Ενδεικτική παρουσίαση της δομής του δέντρου για 12 καταχωρήσεις από το αρχείο small_computer_scientists_data.csv.

small_computer_scientists_data.csv:

Surname	Awards	Education	DBLP Info
Abelson	5	Abelson rec	50
Abiteboul	3	The son of t	312
Blum	4	Blum was b	84
Brooks	5	Born on Apr	90
Cardelli	2	He was borı	234
Cristianini	2	Cristianini I	145
DeMarco	1	Tom DeMar	33
Khan	10	Khan was a	59
Dongarra	15	Dongarra re	985
Eckert	4	Eckert was I	9
Edelman	7	Edelman re	88
Dean	6	Dean receiv	111

Εκτελώντας ένα query **R-tree\rtree.py A-D 4 20-300** έχουμε τα εξής 4 ορθά αποτελέσματα :

```
Points within 3D range:
('Brooks', 5, 90, 'Born on April 19 1931 in Durham North Carolina[7] he attended Duke University graduating in 195
3 with a Bachelor of Science degree in physics and he received a Ph.D. in applied mathematics (computer science) f
rom Harvard University in 1956 supervised by Howard Aiken.[2] Brooks served as the graduate teaching assistant for
 Ken Iverson at Harvard\'s graduate program in "automatic data processing" the first such program in the world.[8]
[9][10]')
('Abelson', 5, 50, 'Abelson received a B.S.  summa cum laude  from the University of Minnesota in computer science
 and economics in 1990.[3] He received a Ph.D. in computer science from the University of Washington in 1996  work
ing under Craig Chambers on compilers[4] and whole-program optimization techniques for object-oriented programming
 languages.[5] He was elected to the National Academy of Engineering in 2009  which recognized his work on "the sc
ience and engineering of large-scale distributed computer systems".[6]')
('Dean', 6, 111, 'Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and
economics in 1990.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working u
nder Craig Chambers on compilers[4] and whole-program optimization techniques for object-oriented programming lang
uages.[5] He was elected to the National Academy of Engineering in 2009  which recognized his work on "the science
 and engineering of large-scale distributed computer systems".[6]')
('Cardelli', 6, 104, 'He was born in Montecatini Terme Italy. He attended the University of Pisa[7] before receivi
ng his PhD from the University of Edinburgh in 1982[15] for research supervised by Gordon Plotkin.[4]')
```

```
Scientists with Similarity above 50.0% (2):

Name: Abelson,
Awards: 5,
Education: Abelson received a B.S.  summa cum laude  from the University of Minnesota in computer science and econ
omics in 1990.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under
 Craig Chambers on compilers[4] and whole-program optimization techniques for object-oriented programming language
s.[5] He was elected to the National Academy of Engineering in 2009  which recognized his work on "the science and
 engineering of large-scale distributed computer systems".[6],
DBLP Record: 50

Name: Dean,
Awards: 6,
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economi
cs in 1990.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Cr
aig Chambers on compilers[4] and whole-program optimization techniques for object-oriented programming languages.[
5] He was elected to the National Academy of Engineering in 2009  which recognized his work on "the science and en
gineering of large-scale distributed computer systems".[6],
DBLP Record: 111
```

```
Execution Time Results:

Build Time: 0.0010104179382324219 seconds
Search Time: 0.0 seconds
LSH Time: 0.035063266754150 39 seconds
Total Time: 0.03707075119018555 seconds
```

R Tree

Περιγραφή υλοποίησης

Το R-tree (Δέντρο R) είναι επίσης μια δομή δέντρου που χρησιμοποιείται για την αποθήκευση και χωρικών δεδομένων σε πολλές διαστάσεις. Σε αντίθεση με τα Quad Trees που χωρίζουν τον χώρο σε τετράγωνα, τα R-trees τον χωρίζουν σε παραλληλόγραμμα. Για παράδειγμα, στον διδιάστατο χώρο ορίζεται ένα παραλληλόγραμμο επίπεδο το οποίο χωρίζεται σε μικρότερα παραλληλόγραμμα. Επίσης, για το δέντρο πρέπει να οριστεί ένας αριθμός m και ενάς αριθμό M που θα δηλώνουν το εύρος του πλήθους των παιδιών που μπορεί να έχει κάθε κόμβος. Αν ο αριθμός των στοιχειών στο αρχικό παραλληλόγραμμο ξεπεράσει αυτό το εύρος, θα γίνει ένας διαχωρισμός (split) για να ικανοποιηθεί ο αριθμός μεταξύ m και M. Όσον αφορά τα R-Tree σε 3 διαστάσεις ο χώρος χωρίζεται σε 3d παραλλήλογραμα (μήκος, πλάτος, ύψος). Παρακάτω θα παρατεθεί ο κώδικας μαζί με την εξήγηση της υλοποίησης.

```
class Node:
    def __init__(self, items):
        self.items = items
```

Η κλάση Node εκφράζει τον κάθε κόμβο του δέντρου. Οι κόμβοι, μπορούν να είναι είτε περιπτώσεις MinimumBoundingObject που αντιπροσωπεύουν παραλληλόγραμμα οριοθέτησης είτε Scientist που αντιπροσωπεύουν σημεία στον τρισδιάτατο χώρο.

```
    def __str__(self, level=0):
        prefix = "| " * level
        result = f"{prefix}Node:\n"
        for item in self.items:
            if isinstance(item, MinimumBoundingObject):
                result += f"{prefix}  - Bounding Object: {item.low} to {item.high}\n"
                result += item.child.__str__(level + 1)
            else:
                result += f"{prefix}  - Item: {item.surname, item.awards,
item.dblp_records}\n"
        return result
```

Η __str__ χρησιμοποιείται για αναπαράσταση του κόμβου και των περιεχομένων του. Από την παράμετρο level, χρησιμοποιούμε την εσοχή για την οπτική αναπαράσταση της ιεραρχικής δομής του R-tree. Για κάθε στοιχείο του κόμβου, ελέγχει αν είναι παραλληλόγραμο οριοθέτησης ή επιστήμονας. Στην 1η περίπτωση, δείχνει τις τιμές του παραλληλογράμου και στη συνέχεια εκτυπώνει αναδρομικά το υποδέντρο κάτω από αυτό. Για αντικείμενα φύλλων (επιστήμονες), περιλαμβάνει μια γραμμή που εμφανίζει το επώνυμο του επιστήμονα, τα βραβεία και τα dblp_records.

```
    def create_rtree(self, points, dimensions):
        M = 4
        m = 2
        upper_level_items = []

        if not points:
            # print("No points provided.")
            return None

        if 0 < len(points) % M < m:
            remaining_points = M + len(points) % M
            last_two_groups_length = [
                math.ceil(remaining_points / m),
                math.floor(remaining_points / m),
            ]
        elif len(points) % M >= m:
            last_two_groups_length = [M, len(points) % M]
        else:
            if len(points) == M:
                last_two_groups_length = [M, 0]
            else:
                last_two_groups_length = [M, M]
...
```

Η μεταβλητή last_two_groups_length υπολογίζεται για να καθορίσει τον αριθμό των σημείων που απομένουν μετά το σχηματισμό κόμβων μεγέθους M κατά τη διαδικασία κατασκευής του δέντρου. Η λογική πίσω από την last_two_groups_length είναι να χειρίζεται τις περιπτώσεις όπου ο συνολικός αριθμός των σημείων δεν είναι πολλαπλάσιο του M, ώστε να μην υπάρχουν ανισορρόπητες ομάδες. Όταν δεν υπάρχουν υπόλοιπα σημεία μετά το σχηματισμό κόμβων με M κλειδιά, τότε το last_two_groups_length = [M, M], υποδεικνύοντας ότι και οι δύο τελευταίες ομάδες θα περιέχουν M σημεία. Εάν υπάρχουν αρκετά εναπομείναντα σημεία για να σχηματίσουν μια πλήρη ομάδα μεγέθους m, το last_two_groups_length ορίζεται σε [M, len(points) % M], υποδηλώνοντας ότι οι δύο τελευταίες ομάδες θα έχουν M σημεία μαζί με τα εναπομείναντα σημεία. Επίσης η συνάρτηση create_rtree() δημιουργεί και κόμβους σε ανώτερο ιεραρχικό επίπεδο αν χρειαστεί να διασπαστεί το παραλληλόγραμμό.

```
def query(self, root, surname_range, min_awards, dblp_range):
    results = []
    current_node = root
    if current_node is None:
        print("Current node is None")
        return None  # Return None if the current node is None
    if current_node.items:
        if isinstance(current_node.items[0], MinimumBoundingObject):
            for item in current_node.items:
                result = self.query(
                    item.child, surname_range, min_awards, dblp_range
                )
                if result is not None:
                    results += result
```

```
            else:
                for scientist in current_node.items:
                    surname_value = self.convert_to_mapping(scientist.surname)
                    if (
                        self.convert_to_mapping(surname_range[0])
                        <= surname_value
                        <= self.convert_to_mapping(surname_range[1])
                    ):
                        if (
                            scientist.awards > min_awards
                            and dblp_range[0] <= scientist.dblp_records <= dblp_range[1]
                        ):
                            results.append(scientist.surname)  # Append the surname
        return results
```

Η συνάρτηση query εκτελεί ένα query στο R Tree που δημιουρήθηκε, με βάση καθορισμένα κριτήρια, όπως εύρος αρχικού γράμματος των επωνύμων, κατώτατο όριο βραβείων και εύρος DBLP.

Στην main υπάρχει η δυνατότητα να εκτυπωθούν οι πληροφορίες των επιστημώνων που πληρούν τα κριτήρια του query.

Όσον αφορά για την ομοιότητα των κειμένων χρησιμοποιείται η συνάρτηση LSH.check_lsh_similarity() για την οποία υπάρχει αναφορά και εξήγηση στο μέρος με το LSH.

Πειραματικά αποτελέσματα

Όπως και στα παραπάνω δέντρα χρησιμοποιήθηκε το αρχείο **small_computer_scientists_data.csv**

Query: **python R-tree/rtree.py A-D 4 20-300**

```
Abelson, awards: 5, dblp_records: 50
Education: Abelson received a B.S. from the University of Patras in computer engineering in 1998.[3] He received a Ph.D. in
computer engineering from the University of Washington in 1990  working under Spyros Sioutas on assemblers[4] and whole-prog
ram optimization techniques for functional programming languages.[5] He was elected to the National Academy of Engineering i
n 2011  which recognized his work on "the science and engineering of large-scale distributed computer systems".[6]
--------------------------------------
--------------------------------------
Brooks, awards: 5, dblp_records: 90
Education: Born on April 19 1931 in Durham North Carolina[7] he attended Duke University graduating in 1953 with a Bachelor
of Science degree in physics and he received a Ph.D. in applied mathematics (computer science) from Harvard University in 19
56 supervised by Howard Aiken.[2] Brooks served as the graduate teaching assistant for Ken Iverson at Harvard's graduate pro
gram in "automatic data processing" the first such program in the world.[8][9][10]
--------------------------------------
--------------------------------------
Cardelli, awards: 6, dblp_records: 104
Education: He was born in Montecatini Terme Italy. He attended the University of Pisa[7] before receiving his PhD from the U
niversity of Edinburgh in 1982[15] for research supervised by Gordon Plotkin.[4]
--------------------------------------
--------------------------------------
Dean, awards: 6, dblp_records: 111
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economics in 1990
.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Craig Chambers on comp
ilers[4] and whole-program optimization techniques for object-oriented programming languages.[5] He was elected to the Natio
nal Academy of Engineering in 2009  which recognized his work on "the science and engineering of large-scale distributed com
puter systems".[6]
```

```
Scientists with Similarity above 50.0% (2):

Name: Abelson,
Awards: 5,
Education: Abelson received a B.S. from the University of Patras in computer engineering in 1998.[3] He received a Ph.D. in
computer engineering from the University of Washington in 1990  working under Spyros Sioutas on assemblers[4] and whole-prog
ram optimization techniques for functional programming languages.[5] He was elected to the National Academy of Engineering i
n 2011  which recognized his work on "the science and engineering of large-scale distributed computer systems".[6],
DBLP Record: 50

Name: Dean,
Awards: 6,
Education: Dean received a B.S.  summa cum laude  from the University of Minnesota in computer science and economics in 1990
.[3] He received a Ph.D. in computer science from the University of Washington in 1996  working under Craig Chambers on comp
ilers[4] and whole-program optimization techniques for object-oriented programming languages.[5] He was elected to the Natio
nal Academy of Engineering in 2009  which recognized his work on "the science and engineering of large-scale distributed com
puter systems".[6],
DBLP Record: 111
```

```
Execution Time Results:

Build Time: 0.0 seconds
Search Time: 0.0 seconds
LSH Time: 0.042516946792602540 seconds
Total Time: 0.056463956832885740 seconds
```

LSH (Locality Sensitive Hashing)

Στην ενότητα αυτή γίνεται περιγραφή της υλοποίησης της διαδικασίας LSH, που χρησιμοποιείται για την αξιολόγηση της ομοιότητας των εκπαιδεύσεων των επιστημόνων που επιστρέφονται από την αναζήτηση πάνω στις δεντρικές δομές. Υλοποιήθηκε ολοκληρωμένα με την τεχνική Min-Hashing και πραγματοποιήθηκε μια προσπάθεια υλοποίησης με την τεχνική Random Projection.

LSH - MinHashing

Περιγραφή υλοποίησης

Το LSH με την τεχνική Min-Hashing, υλοποιείται στο αρχείο «LSH/LSH.py». Για την διευκόλυνση στον προγραμματισμό έχει έχουν δημιουργηθεί τέσσερις κλάσεις. Shingling, MinHash, LSH, Metrics, οι οποίες αναπαριστούν τα στάδια του αλγορίθμου και περιέχουν τις απαραίτητες μεθόδους για την πραγματοποίηση του κάθε βήματος.

Συνοπτική περιγραφή της κάθε κλάσης

Shingling: Αυτή η κλάση χρησιμοποιείται για τη μετατροπή των δεδομένων κειμένου σε ένα σύνολο από "shingles" (υποσύνολα συγκεκριμένου μήκους). Παρέχει επίσης μεθόδους για τη δημιουργία ενός λεξιλογίου (vocabulary) από όλα τα σύνολα των shingles, την κωδικοποίηση one-hot για κάθε σύνολο από shingles. Αυτή η κωδικοποίηση παράγει ένα αραιό διάνυσμα (sparse vector) μήκους ίσο με το λεξιλόγιο, που αντιστοιχεί στο κάθε αρχικό κείμενο.

MinHash: Σε αυτή την κλάση παρέχονται μέθοδοι για την υλοποίηση του Min-Hashing. Συγκεκριμένα χρησιμοποιεί ένας ορισμένος αριθμός συναρτήσεων κατακερματισμού (hash functions) για τη δημιουργία μιας "υπογραφής" (signature) για κάθε αραιό διάνυσμα, η οποία μπορεί στη συνέχεια να συγκριθεί με άλλες υπογραφές για την εκτίμηση της ομοιότητας. Η δημιουργία των υπογραφών, επιτυγχάνει μείωση της διαστατικότητας, διατηρώντας την πληροφορία της ομοιότητας.

LSH: Αυτή η κλάση υλοποιεί την ίδια την τεχνική Locality-Sensitive Hashing. Χωρίζει την υπογραφή MinHash σε "ζώνες" (bands) και κατακερματίζει κάθε ζώνη σε έναν κάδο. Οποιεσδήποτε δύο υπογραφές που κατατεμαχίζονται στον ίδιο κάδο για οποιαδήποτε ζώνη θεωρούνται υποψήφιο ζεύγος (candidate pairs), δηλαδή είναι δυνητικά παρόμοιες.

Metrics: Αυτή η κλάση παρέχει μεθόδους για τον υπολογισμό της ομοιότητας Jaccard μεταξύ δύο συνόλων και για την εύρεση των τελικών δεικτών όμοιων κειμένων εκπαίδευσης με βάση τα υποψήφια ζεύγη και ένα κατώφλι ομοιότητας (similarity threshold).

Στην συνέχεια, θα περιγραφεί η κύρια συνάρτηση του αρχείου, check_lsh_similarity(), η οποία χρησιμοποιεί τις παραπάνω κλάσεις και καλείται σε κάθε αρχείο υλοποίησης πολυδιάστατου δέντρου.

```
def check_lsh_similarity(educations_list: list, k, b, threshold, hash_functions_count):
    """
    Perform similarity checks using Locality Sensitive Hashing (LSH) on education data.

    Args:
        educations_dict (dict): A dictionary containing education data where keys are education
        strings strings and values are the corresponding csv row indexes.
        k (int): Shingle size.
        b (int): Number of bands used in LSH.
        threshold (float): Similarity threshold.
        hash_functions_count (int): Number of hash functions used in MinHash.

    Returns:
        list: A list of indexes corresponding to similar education data in the CSV file.
    """

    shingles = []
    for data in educations_list:
        shingle_set = Shingling.shingle(data, k)
        shingles.append(shingle_set)

    vocabulary = Shingling.create_vocabulary(shingles)

    shingles_onehot_encoding = []
    for shingle_set in shingles:
        onehot_encoding = Shingling.one_hot_encoding(shingle_set, vocabulary)
        shingles_onehot_encoding.append(onehot_encoding)

    minhash_instance = MinHash(
        vocabulary_size=len(vocabulary), hash_functions_count=hash_functions_count
    )
    signatures = []
    for sparse_vector in shingles_onehot_encoding:
        signature = minhash_instance.minhash(sparse_vector)
        signatures.append(signature)

    lsh_instance = LSH(b=b)
    for signature in signatures:
        lsh_instance.add_signature_hash(signature)

    candidate_pairs = lsh_instance.find_candidate_pairs()

    final_education_list_indexes = Metrics.find_final_education_indexes(
        candidate_pairs, signatures, threshold=threshold
    )

    return final_education_list_indexes
```

Παράμετροι: Η συνάρτηση λαμβάνει πέντε ορίσματα: educations_list, k, b, threshold και hash_functions_count.

Η educations_list είναι η λίστα κειμένων εκπαίδευσης που παρέχεται από τα αποτελέσματα αναζήτησης στο κάθε δέντρο. Η παράμετρος k είναι το μέγεθος των shingles, το οποίο χρησιμοποιείται για τη διάσπαση των δεδομένων σε μικρότερα

κομμάτια. Η παράμετρος b είναι ο αριθμός των ζωνών (bands), που χρησιμοποιούνται στην LSH. Η παράμετρος threshold είναι το κατώφλι ομοιότητας, το οποίο καθορίζει πόσο παρόμοια πρέπει να είναι δύο υπογραφές για να θεωρηθούν όμοιες. Η παράμετρος hash_functions_count είναι ο αριθμός των συναρτήσεων κατακερματισμού που χρησιμοποιούνται στο MinHash.

Αλγόριθμος:

1. Αρχικά, η συνάρτηση δημιουργεί μια λίστα με shingles από τα δεδομένα εκπαίδευσης χρησιμοποιώντας τη μέθοδο Shingling.shingle. Στη συνέχεια δημιουργεί ένα λεξιλόγιο από αυτά τα shingles χρησιμοποιώντας τη μέθοδο Shingling.create_vocabulary. Έπειτα, τα shingles κωδικοποιούνται με τη μέθοδο Shingling.one_hot_encoding.
2. Κατόπιν, δημιουργείται ένα αντικείμενο της κλάσης MinHash με παραμέτρους το μέγεθος του λεξιλογίου και τον αριθμό των συναρτήσεων κατακερματισμού. Ακολούθως, η μέθοδος minhash της MinHash χρησιμοποιείται για τη δημιουργία μιας υπογραφής minhash για κάθε κωδικοποιημένο shingle set. Αυτές οι υπογραφές αποθηκεύονται σε μια λίστα.
3. Έπειτα, δημιουργείται ένα αντικείμενο της κλάσης LSH με παράμετρο τον αριθμό των ζωνών. Η μέθοδος add_signature_hash της LSH χρησιμοποιείται για την προσθήκη κάθε κατακερματισμένης υπογραφής στους κατάλληλους κάδους για αποτελεσματική ανάκτηση.
4. Η μέθοδος find_candidate_pairs του αντικειμένου LSH χρησιμοποιείται στη συνέχεια για την εύρεση υποψήφιων ζευγών με βάση τα περιεχόμενα των κάδων (buckets). Αυτά τα υποψήφια ζεύγη περνούν στη μέθοδο Metrics.find_final_education_indexes μαζί με τις υπογραφές και το κατώφλι ομοιότητας για να βρεθεί το τελικό σύνολο των δεικτών που αντιστοιχούν σε παρόμοια δεδομένα εκπαίδευσης στην αρχική λίστα educations_list.

Πειραματικά αποτελέσματα

Για την αξιολόγηση του LSH ως ανεξάρτητου συστήματος χωρίς τα δέντρα (αξιολόγηση του συνολικού συστήματος lsh μαζί με δέντρα ενότητες Μετρήσεις Δέντρων/LSH & Γραφήματα και πειραματικά αποτελέσματα δέντρων). Υλοποιήσαμε ένα μικρό python script το οποίο διαβάζει τα δεδομένα από το Testing_LSH.csv και τα περνάει από το lsh minhash και επιστρέφει τα πιο relevant κείμενα πάνω από το threshold που έχουμε ορίσει.

Testing_LSH.csv :

Κατασκευάσαμε αυτό το αρχείο με 80 δεδομένα της μορφής:

Όνομα πανεπιστήμιου: (πχ Radiance, Oracle, Pinnacle…)

Τύπος πανεπιστήμιου: (πχ College, Institute, University…)

Τύπος πτυχίου: (πχ Master, phd, Bachelor ….)

Όνομα πανεπιστήμιου δευτέρου πτυχίου: (πχ Radiance, Oracle, Pinnacle…)

Τύπος πανεπιστήμιου δευτέρου πτυχίου: (πχ College, Institute, University…)

Τύπος δευτέρου πτυχίου: (πχ Master, phd, Bachelor ….)

Ο τρόπος που έχει δομηθεί το αρχείο έχει ως αποτέλεσμα να υπάρχει ίσή κατανομή λέξεων σε όλα τα data entries. Αυτή την συμπεριφορά θα την δούμε στις παρακάτω πειραματικές μετρήσεις:

Παραμετροποίηση script k=3, b=6, hash_functions_count=12 ύστερα από πειραματισμό είναι πιο optimal παράμετροι για το παρών data set.

Threshold 99%:

```
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Lenght of Results: 2
LSH similar indexes: {5, 6}
LSH time: 0.08803939819335938
```

Threshold 90%:

```
Apex Institute PhD Breeze University Bachelor
Panorama Institute PhD Quasar University Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Lenght of Results: 4
LSH similar indexes: {65, 41, 5, 6}
LSH time: 0.09899759292602539
```

Threshold 85%:

```
Unity Institute PhD Vanguard University Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Espresso Institute PhD Forte University Bachelor
Lenght of Results: 4
LSH similar indexes: {20, 5, 6, 55}
LSH time: 0.09599995613098145
```

Threshold 80%:

```
Adams University Bachelor Bentley College Master
Yamaha University Bachelor Zest College Master
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Quasar University Bachelor Radiant College Master
Tranquil University Bachelor Umbra College Master
Lenght of Results: 6
LSH similar indexes: {0, 64, 5, 6, 42, 45}
LSH time: 0.09164094924926758
```

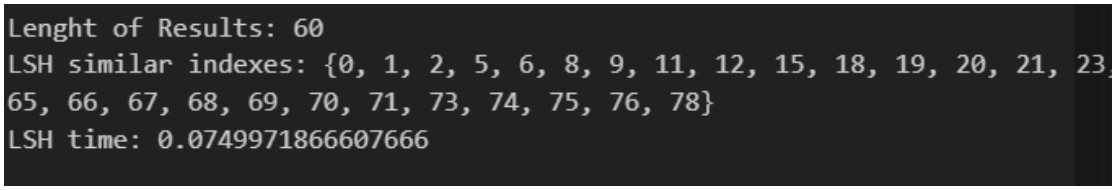
Threshold 70%:

```
Galaxy Institute PhD Horizon University Bachelor
Apex Institute PhD Breeze University Bachelor
Horizon University Bachelor Infinity College Master
Dynamo Institute PhD Ethereal University Bachelor
Aurora Institute PhD Borealis University Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Gleam Institute PhD Harmonic University Bachelor
Majestic Institute PhD Nebula University Bachelor
Riverview Institute PhD Starlight University Bachelor
Starlight University Bachelor Titanium College Master
Zenith University Bachelor Allegro College Master
Brio Institute PhD Cadenza University Bachelor
Mozart Institute PhD Napoli University Bachelor
Napoli University Bachelor Overture College Master
Dynamo Institute PhD Eclipse University Bachelor
Xylo Institute PhD Yamaha University Bachelor
Lenght of Results: 17
LSH similar indexes: {32, 65, 33, 67, 26, 5, 6, 71, 38, 17, 18, 51, 53, 58, 59, 29, 63}
LSH time: 0.09100031852722168
```

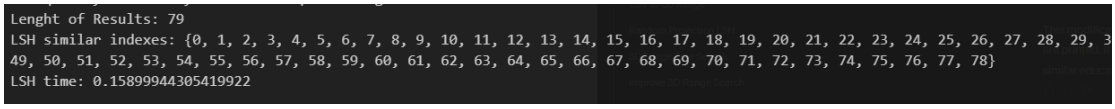
Threshold 60%:

```
Adams University Bachelor Bentley College Master
Fountainhead College PhD Greenwood Academy Bachelor
Fountainhead College PhD Greenwood Academy Bachelor
Marble University Bachelor Noble Institute Master
Riverview Institute PhD Starlight University Bachelor
Starlight University Bachelor Titanium College Master
Xenon Institute PhD Yellowstone University Bachelor
Yellowstone University Bachelor Yellowstone University Bachelor
Aurora Institute PhD Borealis University Bachelor
Borealis University Bachelor Cascade College Master
Dynamo Institute PhD Eclipse University Bachelor
Eclipse University Bachelor Frontier College Master
Galaxy Institute PhD Horizon University Bachelor
Horizon University Bachelor Infinity College Master
Jovial Institute PhD Kaleido University Bachelor
Kaleido University Bachelor Luminous College Master
Majestic Institute PhD Nebula University Bachelor
Nebula University Bachelor Oasis College Master
Panorama Institute PhD Quasar University Bachelor
Quasar University Bachelor Radiant College Master
Tranquil University Bachelor Umbra College Master
Velocity Institute PhD Whispering University Bachelor
Yonder Institute PhD Zenith University Bachelor
Zenith University Bachelor Allegro College Master
Brio Institute PhD Cadenza University Bachelor
Cadenza University Bachelor Dolce College Master
Espresso Institute PhD Forte University Bachelor
Inspire University Bachelor Legato College Master
Mozart Institute PhD Napoli University Bachelor
Napoli University Bachelor Overture College Master
Xylo Institute PhD Yamaha University Bachelor
Yamaha University Bachelor Zest College Master
Apex Institute PhD Breeze University Bachelor
Breeze University Bachelor Celestial College Master
Dynamo Institute PhD Ethereal University Bachelor
Ethereal University Bachelor Fusion College Master
Fusion College Master Gleam Institute PhD
Gleam Institute PhD Harmonic University Bachelor
Harmonic University Bachelor Ignite College Master
Mystic Institute PhD Nirvana University Bachelor
Oracle College Master Pinnacle Institute PhD
Tranquility University Bachelor Utopia College Master
Lenght of Results: 42
LSH similar indexes: {0, 5, 6, 12, 17, 18, 23, 24, 26, 27, 29, 30, 32, 33, 35, 36, 38, 39, 41, 42, 45, 47, 50, 51, 53, 54, 55, 56, 58, 59, 63, 64, 65, 66, 67, 68, 70, 71, 72, 74, 75, 78}
LSH time: 0.1055138111114502
```

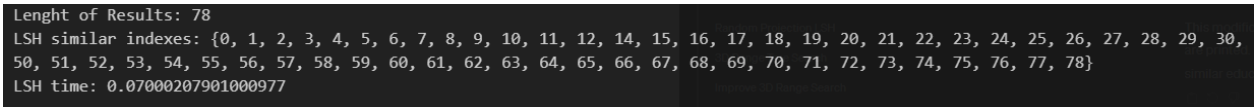
Threshold 55%:



Threshold 50%:



Threshold 40%:



Παρατηρούμε ότι το σύστημα ακολουθεί την ορθή επιθυμητή συμπεριφορά!! Για τα διαφορετικά thresholds.

LSH - Random Projection

Επιπλέον extra κομμάτι της εργασίας προσπάθεια υλοποίησης του Lsh με την τεχνική του random projection και sparce text vectors με tf idf technique!! Η υλοποίηση αυτή λειτουργεί εν μέρη ορθά, δεν 100 optimized η υλοποίηση, πρώτον να λειτουργεί με χαμηλά threshold δευτέρων με μεγάλα data sets της τάξης 250 > n και σε συνεργασία με τον κώδικα τον δέντρων.

Επεξήγηση κώδικα:

Ο κώδικας χρησιμοποιεί την αναπαράσταση Συχνότητας Όρων-Αντίστροφου Εγγράφου (TF-IDF) και τυχαίες υπερεπιφάνειες για την αποτελεσματική προσέγγιση της ομοιότητας συνημμένων κειμένων. Ο αλγόριθμος LSH χρησιμοποιείται για τη δημιουργία πινάκων κατακερματισμού, επιτρέποντας την ανάκτηση παρόμοιων εκπαιδευτικών κειμένων.

Φόρτωση Δεδομένων και Διανυσματοποίηση TF-IDF:

Ο κώδικας ξεκινάει με τη φόρτωση εκπαιδευτικών δεδομένων από ένα αρχείο CSV και τη διανυσματοποίηση των κειμενικών δεδομένων χρησιμοποιώντας το TF-IDF. Το TfidfVectorizer από τη βιβλιοθήκη scikit-learn χρησιμοποιείται για αυτόν τον σκοπό, παράγοντας ένα αραιό πίνακα TF-IDF.

Εν συνέχεια ο αλγόριθμος LSH υλοποιείται με έναν καθορισμένο αριθμό τυχαίων υπερεπιφανειών και πινάκων κατακερματισμού. Τυχαίοι διάνυσματα δημιουργούνται για κάθε πίνακα κατακερματισμού, και τα έγγραφα κατακερματίζονται σε κάδους βάσει των προβολών τους σε αυτές τις υπερεπιφάνειες.

Η κύρια συνάρτηση, find_most_similar_texts, αναζητά ζευγάρια παρόμοιων εκπαιδευτικών κειμένων βάσει του LSH και ενός καθορισμένου κατωφλίου. Για κάθε κείμενο στο σύνολο δεδομένων, ο αλγόριθμος εξετάζει τους πίνακες κατακερματισμού, υπολογίζει την ομοιότητα συνημμένων κειμένων και ανακτά παρόμοια κείμενα άνω του καθορισμένου κατωφλίου.

Γιατί χρησιμοποιήσαμε TF-IDF (Term Frequency-Inverse Document Frequency) στον κώδικα για αρκετούς λόγους:

Το TF-IDF είναι μια τεχνική για τον μετασχηματισμό κειμενικών δεδομένων σε αριθμητική μορφή. Μετατρέπει μια συλλογή από κείμενα έγγραφα σε έναν πίνακα, όπου κάθε γραμμή αντιπροσωπεύει ένα έγγραφο και κάθε στήλη αντιπροσωπεύει ένα μοναδικό όρο σε ολόκληρη τη συλλογή. Αυτή η αριθμητική αναπαράσταση είναι ουσιώδης για πολλούς αλγορίθμους μηχανικής μάθησης, καθώς συχνά απαιτούν αριθμητική είσοδο.

Επίσης το TF-IDF αναθέτει βάρη στους όρους με βάση τη σημασία τους στο σύνολο των κειμένων. Λαμβάνει υπόψη τόσο τη συχνότητα των όρων (πόσο συχνά εμφανίζεται ένας όρος σε ένα έγγραφο) όσο και τον αντίστροφο συντελεστή εγγράφου (πόσο μοναδικός είναι ένας όρος σε ολόκληρη τη συλλογή). Αυτό βοηθά

στην κατανόηση της σημασίας των όρων μέσα σε συγκεκριμένα έγγραφα, ενώ μειώνει τη σημασία των κοινών όρων που εμφανίζονται σε πολλά έγγραφα.

Τέλος τα δεδομένα κειμένου είναι ενδεχομένως υψηλής διαστατικότητας λόγω του μεγάλου μεγέθους του λεξιλογίου. Το TF-IDF βοηθάει στη μείωση της διαστατικότητας αναπαριστώντας κάθε έγγραφο με ένα αραιό διάνυσμα που περιέχει μόνο τους σημαντικούς όρους.

Πειραματικά αποτελέσματα

Το testing εφαρμόστηκε στα δεδομένα στο σύνολο δεδομένων Testing_LSH.csv και περνούμε τα εξής αποτελέσματα:

```
Unique pairs of similar education data above threshold 0.9:
('Fountainhead College PhD Greenwood Academy Bachelor','Fountainhead College PhD Greenwood Academy Bachelor',)
Number of unique pairs of similar education data above threshold 0.9: 1
```

```
Unique pairs of similar education data above threshold 0.8:
('Fountainhead College PhD Greenwood Academy Bachelor',Fountainhead College PhD Greenwood Academy Bachelor')
Number of unique pairs of similar education data above threshold 0.8: 1
```

```
Unique pairs of similar education data above threshold 0.7:
('Xenon Institute PhD Yellowstone University Bachelor', 'Yellowstone University Bachelor Yellowstone University Bachelor')
('Fountainhead College PhD Greenwood Academy Bachelor','Fountainhead College PhD Greenwood Academy Bachelor')
Number of unique pairs of similar education data above threshold 0.7: 2
```

```
Unique pairs of similar education data above threshold 0.6:
('Everest Polytechnic Master Fountainhead College PhD', 'Dunham University Bachelor Everest Polytechnic Master')
('Fountainhead College PhD Greenwood Academy Bachelor','Fountainhead College PhD Greenwood Academy Bachelor')
('Everest College PhD Greenwood Academy Bachelor', 'Fountainhead College PhD Greenwood Academy Bachelor')
('Yellowstone University Bachelor Yellowstone University Bachelor', 'Xenon Institute PhD Yellowstone University Bachelor')
('Pinnacle College Bachelor Quantum Academy Master', 'Quantum Academy Master Riverview Institute PhD')
('Ignite College Master Jubilee Institute PhD', 'Jubilee College Bachelor Jubilee College Bachelor')
Number of unique pairs of similar education data above threshold 0.6: 6
```

```
Unique pairs of similar education data above threshold 0.5:
('Quasar University Bachelor Radiant College Master', 'Radiant College Master Spectrum Institute PhD')
('Infinity College Master Jovial Institute PhD', 'Jovial Institute PhD Kaleido University Bachelor')
('Nebula University Bachelor Oasis College Master', 'Oasis College Master Panorama Institute PhD')
('Frontier College Master Galaxy Institute PhD', 'Galaxy Institute PhD Horizon University Bachelor')
('Eclipse University Bachelor Frontier College Master', 'Dynamo Institute PhD Eclipse University Bachelor')
('Yonder Institute PhD Zenith University Bachelor', 'Xanadu College Master Yonder Institute PhD')
('Panorama Institute PhD Quasar University Bachelor', 'Oasis College Master Panorama Institute PhD')
('Opulent University PhD Pinnacle College Bachelor', 'Noble Institute Master Opulent University PhD')
```

```
('Eclipse University Bachelor Frontier College Master', 'Frontier College Master Galaxy Institute PhD')
Number of unique pairs of similar education data above threshold 0.5: 38
```

```
Unique pairs of similar education data above threshold 0.2:
('Yamaha University Bachelor Zest College Master', 'Xylo Institute PhD Yamaha University Bachelor')
('Dynamo Institute PhD Ethereal University Bachelor', 'Dynamo Institute PhD Eclipse University Bachelor')
('Brio Institute PhD Cadenza University Bachelor', 'Cadenza University Bachelor Dolce College Master')
('Pinnacle Institute PhD Quintessence University Bachelor', 'Oracle College Master Pinnacle Institute PhD')
('Horizon University Bachelor Infinity College Master', 'Galaxy Institute PhD Horizon University Bachelor')
```

```
('Starlight University Bachelor Titanium College Master', 'Titanium College Master Unity Institute PhD')
Number of unique pairs of similar education data above threshold 0.2: 79
```

Μετρήσεις Δέντρων/LSH & Γραφήματα

Πειραματικά δεδομένα testing πως προέκυψαν; Για κάθε δέντρο πήραμε τα GigaGigaMegaData.csv: Μεγάλα σύνολα δεδομένων που χρησιμοποιούνται για δοκιμές των δέντρων και για τον έλεγχο της πολυπλοκότητάς τους έχουμε για μέγεθος input 40000 , 80000 ,120000, 160000 ,200000 Scientists . Παρήχθησαν ύστερα από χρήση κώδικα που επαναλάμβανε τυχαία, δεδομένα από τους υπάρχοντες 412 επιστήμονες του αρχείου new_computer_scientists_data. Για κάθε δέντρο/ lsh πήραμε 4 ίδια ερωτήματα για κάθε διαφορετικό input αυτό το επιλεξαμε για να έχουμε καλύτερη αναπαράσταση του δείγματος. Τα ερωτήματα που τέθηκαν στα δέντρα για κάθε input file είναι :

1. A-D 4 20-300
2. G-J 3 10-150
3. K-O 6 30-700
4. P-Z 9 5-84

κατά την διάρκεια των ερωτημάτων για κάθε δέντρο και input size μετρήθηκαν οι εξής χρόνοι:

Build Time: μετρά τον χρόνο κατασκευής του δέντρου.

Search Time: μετρά τον χρόνο πραγματοποίησής ενός range ερωτήματος στο δέντρο.

LSH Time: χρόνος εκτελέσεις του lsh για την εύρεση τον πιο όμοιων επιστημών στο πεδίο education για όλες της μετρήσεις πραγματοποιήθηκε lsh με similarity thresshold = 50% και παραμέτρους k=4, b=25, hash_functions_count=100

Total Time: συνολικός χρόνος το άθροισμα των προηγουμένων χρόνων Build Time + Search Time + LSH Time.

Όλες οι αναλυτικές πειραματικές μετρήσεις βρίσκονται στο folder Time_Tasting_Data των παραδοτέων αρχείων.

Για την κατασκευή των γραφικών παραστάσεων πήραμε τον μέσο Build Time, search Time και LSH Time των 4 ερωτημάτων που αναφέραμε για κάθε δέντρο και input 40000 έως 200000.

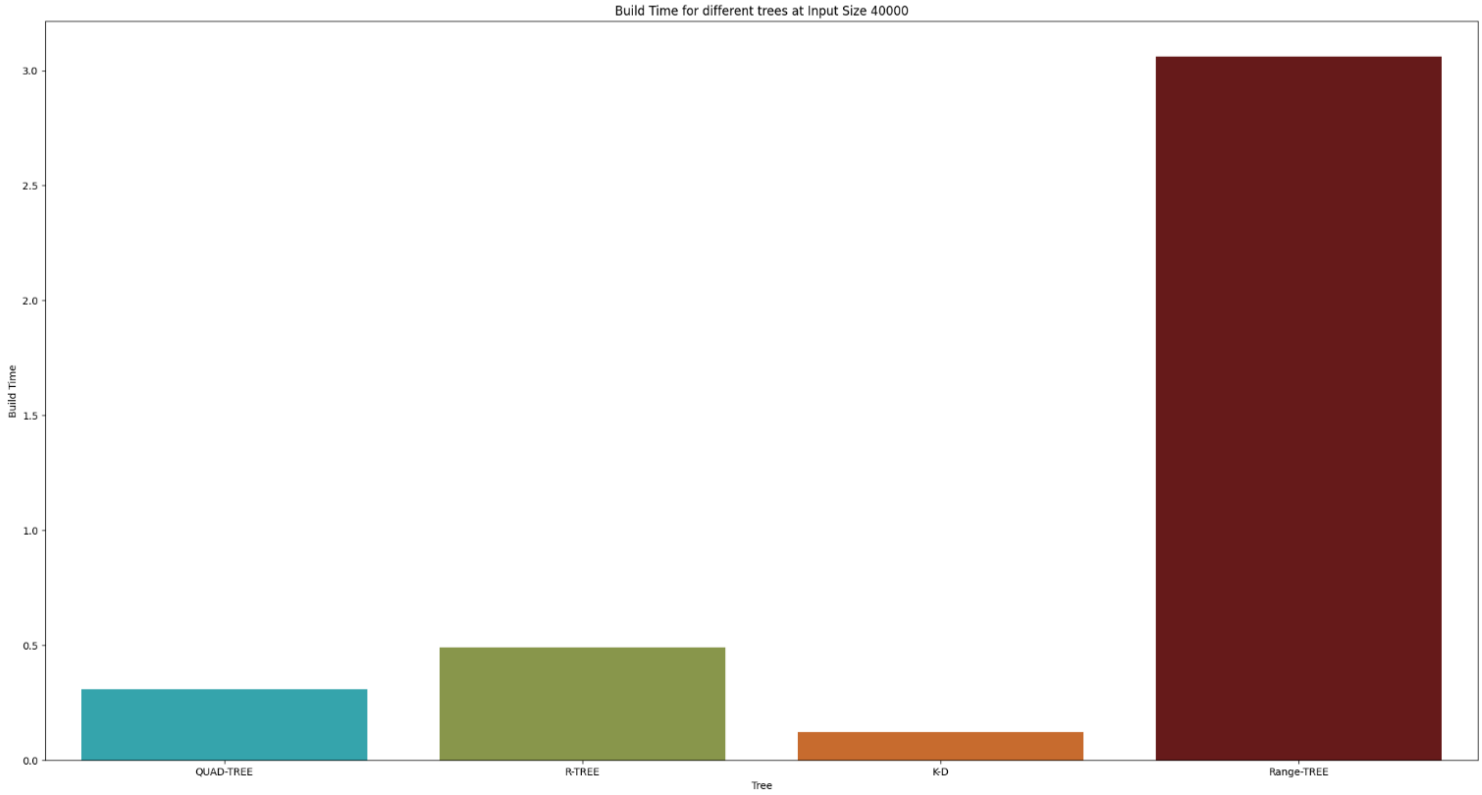
Σύντομη επεξήγηση κώδικα που αναπτύχθηκε για την παραγωγή των γραφικών παραστάσεων :

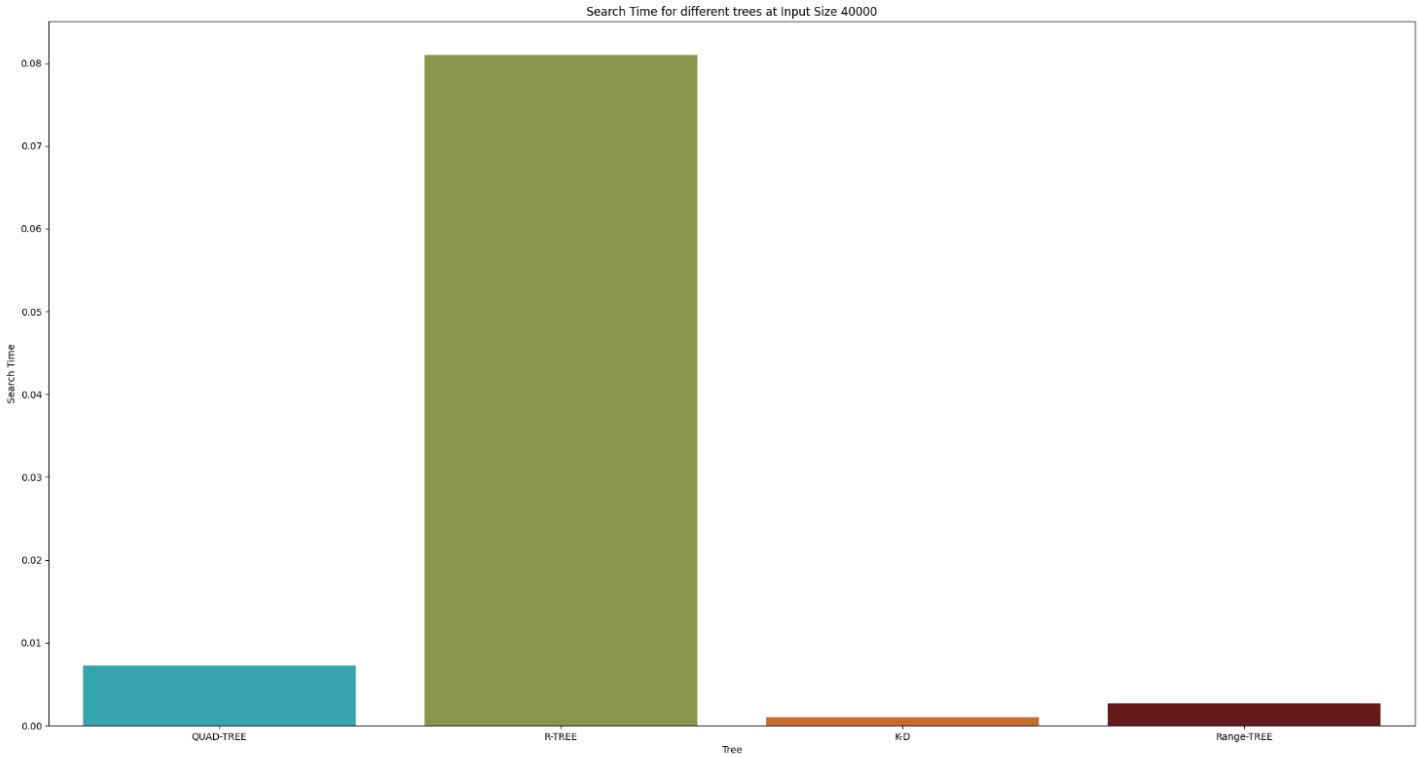
Η συνάρτηση read_csv_file διαβάζει ένα αρχείο CSV από έναν καθορισμένο διαδρομή (file_path) και επιστρέφει ένα DataFrame της βιβλιοθήκης pandas που περιέχει τα δεδομένα από το αρχείο.

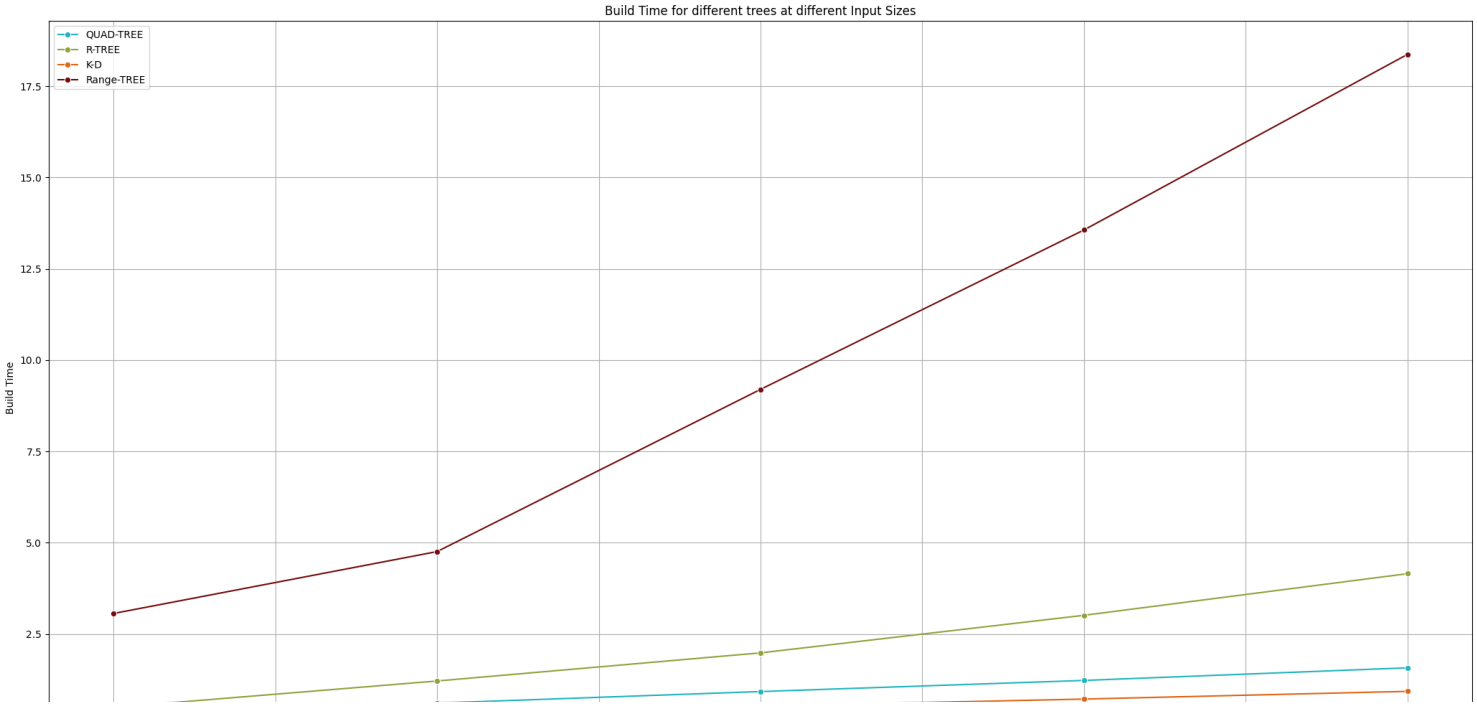
Δημιουργεί γραφήματα bar για τον χρόνο κατασκευής (Build Time) και τον χρόνο αναζήτησης (Search Time) για κάθε δέντρο στο Input Size 40000.

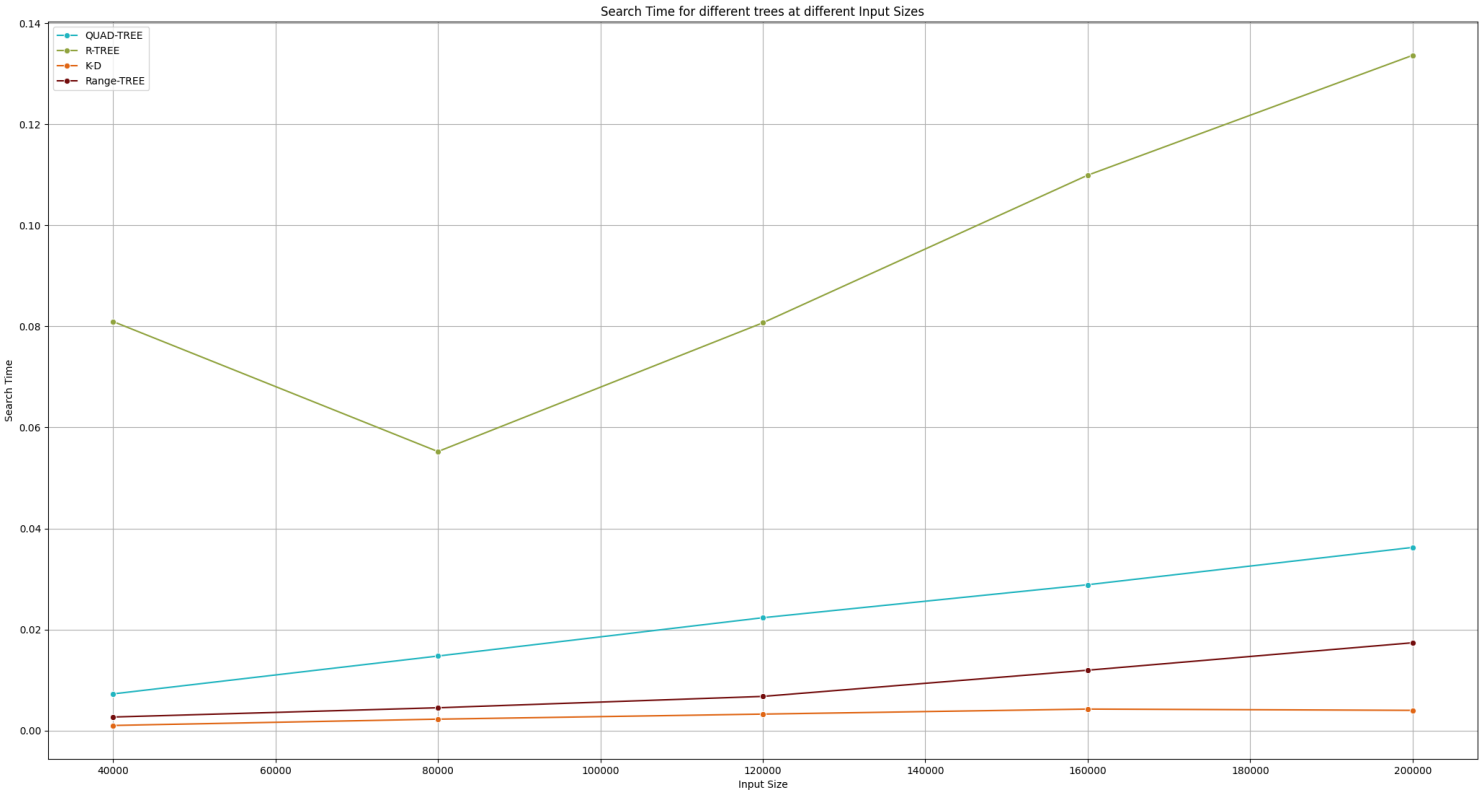
Δημιουργεί γραφήματα γραμμών για τον χρόνο κατασκευής, τον χρόνο αναζήτησης και τον συνολικό χρόνο για κάθε δέντρο σε διάφορα Input Sizes.

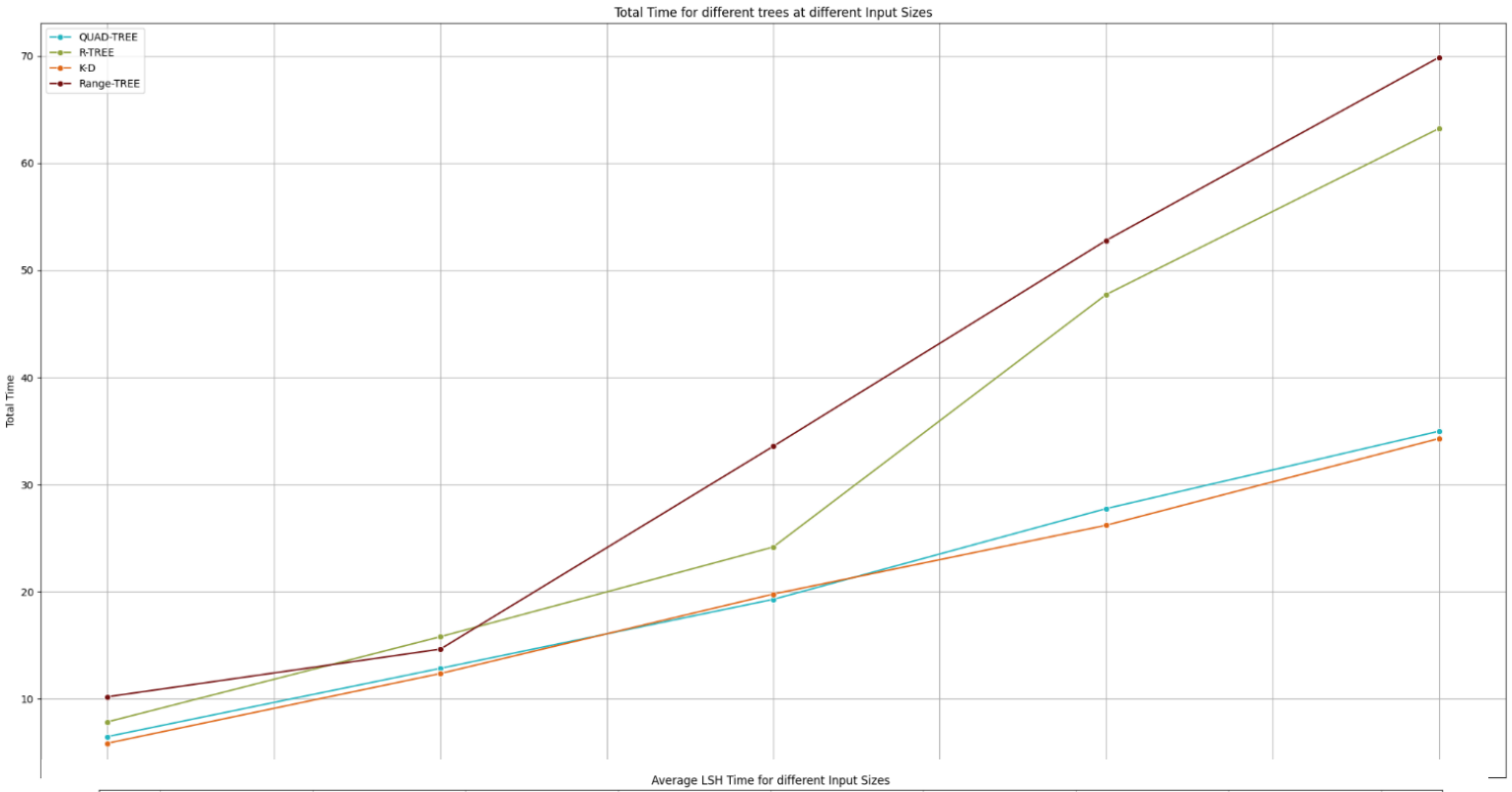
Δημιουργεί ένα γράφημα γραμμής για τον μέσο χρόνο LSH (LSH Time) για κάθε διάφορο Input Size.

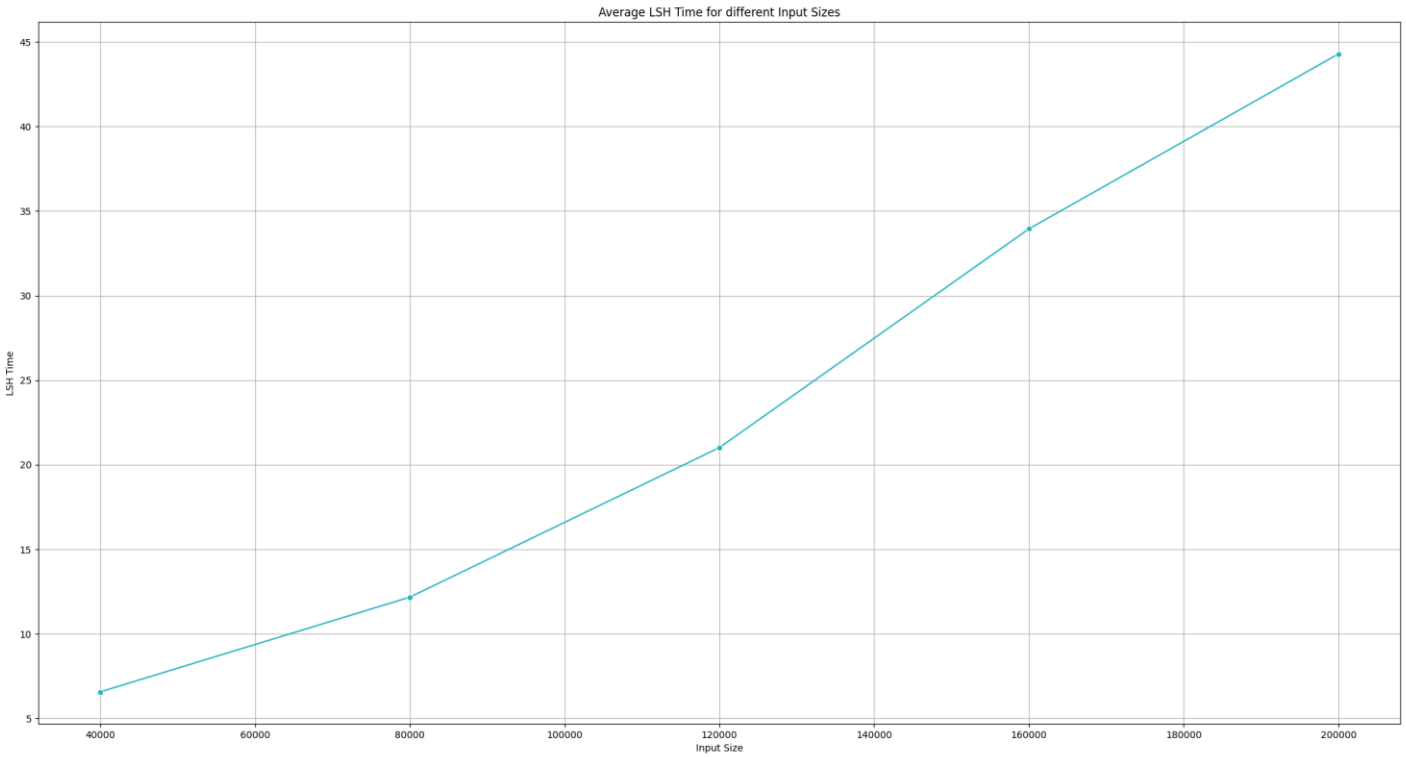





Build Time for different trees at different Input Sizes
QUAD-TREE
R-TREE
K-D
Range-TREE
Build Time
17.5
15.0
12.5
10.0
7.5
5.0
2.5


Search Time for different trees at different Input Sizes
QUAD-TREE
R-TREE
K-D
Range-TREE
Search Time
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
40000
60000
80000
100000
120000
140000
160000
180000
200000
Input Size


Total Time for different trees at different Input Sizes
QUAD-TREE
R-TREE
K-D
Range-TREE
Total Time
70
60
50
40
30
20
10


Average LSH Time for different Input Sizes
LSH Time
45
40
35
30
25
20
15
10
5
40000
60000
80000
100000
120000
140000
160000
180000
200000
Input Size

Execution Time Results: input size 40000

Tree	Build Time	Search Time	LSH Time	Total Time
K-D	0.1279	0.00099	10.8031	12.6555
QUAD-TREE	0.3060	0.0079	11.26016	12.3995
R-TREE	0.4785	0.0299	11.143	13.096
Range-TREE	2.82696	0.00099	13.14437	16.442

Execution Time Results: input size 80000

Tree	Build Time	Search Time	LSH Time	Total Time
K-D	0.2988	0.00299	22.476062	26.208
QUAD-TREE	0.6156	0.0160	21.83656	24.00229
R-TREE	1.1734	0.05896	23.47027	28.56598
Range-TREE	4.90800	0.0019	17.53573	22.445745

Execution Time Results: input size 120000

Tree	Build Time	Search Time	LSH Time	Total Time
K-D	0.62437	0.00498	35.4144	41.3844
QUAD-TREE	0.904384	0.02434	32.3356	35.600
R-TREE	1.9718	0.0879	34.482	44.3243
Range-TREE	7.480	0.0040	29.3084	36.79334

Execution Time Results: input size 160000

Tree	Build Time	Search Time	LSH Time	Total Time
K-D	0.756	0.0059	46.1181	53.757
QUAD-TREE	1.2320	0.0314	48.310	56.3786
R-TREE	3.0212	0.1219	49.17267	66.572
Range-TREE	12.698	0.00803	45.12602	57.832

Execution Time Results: input size 200000

Tree	Build Time	Search Time	LSH Time	Total Time
K-D	**0.9434**	**0.006995**	**59.4069**	**69.25784**
QUAD-TREE	**1.6014**	**0.0405**	**60.2880**	**70.106**
R-TREE	**4.1776**	**0.149**	**64.121**	**88.8208**
Range-TREE	**16.085**	**0.01997**	**57.09745**	**73.202867**

Τελικά συμπεράσματα εργασίας παρατηρούμε από τις γραφικές παραστάσεις και τους πίνακες ότι τα build times τόσο για τα δέντρα όσο και για range search ακολουθούν τις θεωρητικές χρονικές πολυπλοκότητες. Επίσης παρατηρούμε και ότι το lsh από την γραφική παράσταση lsh time τηρεί την γραμμική πολυπλοκότητα του (O(n * d * k) όπου n είναι ο αριθμός των σημείων δεδομένων, d είναι η διάσταση των διανυσμάτων χαρακτηριστικών και k είναι ο αριθμός των πινάκων κατακερματισμού και των συναρτήσεων κατακερματισμού που χρησιμοποιούνται).

Από τις πειραματικές μέτρησης το πιο γρήγορο σε κάθε κατηγορία αξιολογισης είναι το K-D Tree .