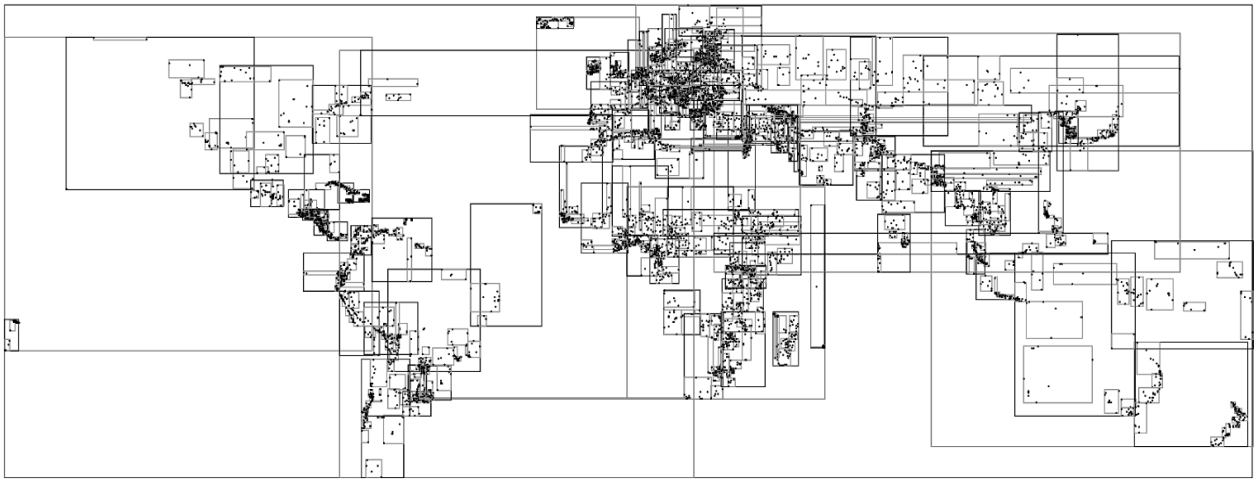
1059633	Νικηφόρος - Γιώργος Παπαγεωργίου	Αλέξανδρος Ξιάρχος	1059619
1041815	Εμμανουήλ Μηναδάκης		



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ · ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΟΛΥΔΙΑΣΤΑΤΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ & ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ

PROJECT 1 · 2022-2023

1 ΚΑΤΑΣΚΕΥΗ DATASET

Κατασκευάσαμε το δικό μας dataset με τα δεδομένα των επιστημόνων της επιστήμης υπολογιστών, τα οποία αντλήσαμε από τη βάση δεδομένων της Wikipedia. Το αρχείο `download_data.py` περιλαμβάνει το script και τη διαδικασία με την οποία αντλήσαμε και επεξεργαστήκαμε τις πληροφορίες αυτές από την Wikipedia. Ο κώδικας στοχεύει στη δημιουργία ενός `.csv` αρχείου, που περιέχει το επώνυμο του κάθε επιστήμονα, όπως επίσης τον αριθμό των βραβείων που έχει λάβει, καθώς και πληροφορίες σχετικά με την εκπαίδευσή του.

Αρχικά, με τη συνάρτηση `get_urls()`, επιστρέφονται σε λίστα όλοι οι σύνδεσμοι (URLs) των επιστημόνων που περιέχονται στην κεντρική σελίδα με τη λίστα επιστημόνων της επιστήμης των υπολογιστών. Αυτό επιτυγχάνεται κάνοντας χρήση της βιβλιοθήκης `BeautifulSoup`, με την οποία μπορούμε να αναλύσουμε το HTML περιεχόμενο κάθε σελίδας, καθιστώντας την ανάκτηση και επεξεργασία των πληροφοριών εύκολη και αυτοματοποιημένη.[1]

Στη συνέχεια, καλείται η συνάρτηση `get_scientist_info()` για κάθε URL, όπου χρησιμοποιώντας regex expressions, HTML tag parsing και string manipulation εξάγονται οι ζητούμενες πληροφορίες για καθέναν από τους επιστήμονες της λίστας. Όταν έχουν ανακτηθεί όλα τα δεδομένα, δημιουργείται ένα `DataFrame`, το οποίο περιέχει τις πληροφορίες για κάθε επιστήμονα.

Λόγω της ιδιομορφίας της κάθε σελίδας, χρειάστηκε να δημιουργήσουμε διαφορετικές περιπτώσεις για να καλύψουμε όλα τα πιθανά σημεία όπου βρίσκονται τα awards και το education, αλλά και πάλι υπάρχουν περιπτώσεις που δεν μπορούν να καλυφθούν αυτοματοποιημένα. Για αυτές τις περιπτώσεις χρησιμοποιείται το αρχείο `corrections.txt`, το οποίο περιέχει πληροφορίες που εξάχθηκαν χειροκίνητα για να διορθωθούν αυτές οι αστοχίες στην εξαγωγή των δεδομένων.

Τέλος, τα δεδομένα εξάγονται στο αρχείο `scientists_data.csv`, που είναι και το τελικό dataset που χρησιμοποιήσαμε για την υλοποίηση της εργασίας. Περιέχει δεδομένα 254 επιστημόνων, που είναι αυτοί για τους οποίους υπήρχε κείμενο σχετικά με την εκπαίδευσή τους στην αντίστοιχη σελίδα τους.

	surname	awards	education
0	Khan	10	Khan was a Bright Sparks scholar and received ...
1	Aaronson	4	Aaronson grew up in the United States, though ...
2	Abebe	3	Abebe was born and raised in Addis Ababa, Ethi...
3	Abelson	1	Abelson graduated with a Bachelor of Arts degr...
4	Abiteboul	4	The son of two hardware store owners, Abitebou...

Εικόνα 1: Παράδειγμα των πέντε πρώτων στοιχείων του dataset.

2 ΥΛΟΠΟΙΗΣΗ ΚΑΤΑΣΚΕΥΗΣ ΚΑΙ ΑΝΑΖΗΤΗΣΗΣ ΣΕ ΠΟΛΥΔΙΑΣΤΑΤΗ ΜΟΡΦΗ

Προτού αναφερθούμε σε κάθε μία εκ των τεσσάρων πολυδιάστατων δομών που υλοποιήσαμε, αξίζει να σημειώσουμε πως για κάθε δομή υλοποιήθηκαν ξεχωριστά μία συνάρτηση κατασκευής της δομής και μία συνάρτηση αναζήτησης στη δομή.

Η συνάρτηση `build_tree()` είναι υπεύθυνη για την κατασκευή της εκάστοτε δομής και για την εισαγωγή σ' αυτήν όλων των διαθέσιμων σημείων. Αφού διαβάσει το αρχείο `.csv` με τα δεδομένα των επιστημόνων, δημιουργεί ένα αντιπροσωπευτικό σημείο (x, y) για τον κάθε επιστήμονα όπου x η αριθμητική τιμή του πρώτου γράμματος του επωνύμου του επιστήμονα (['A', 'Z'] → [0, 25]) και y ο αριθμός των βραβείων που έχει λάβει. Σε συγκεκριμένες περιπτώσεις, χρησιμοποιήθηκε και ο αριθμός γραμμής (`index`) του επιστήμονα στο `DataFrame` ως χρήσιμη πληροφορία για να ξεχωρίζουμε τους επιστήμονες που τυγχάνει να

[1] Συγκεκριμένα για να εξαχθούν τα URLs των επιστημόνων, αναζητούνται, αν υπάρχουν, οι σύνδεσμοι που περιλαμβάνονται στα `<a href>` tags, μέσα στα `<li>` list items των unordered lists `<ul>`.

έχουν ίδιες συντεταγμένες `x`, `y` βάσει των στοιχείων τους. Αφού δημιουργηθούν τα σημεία, κατασκευάζουμε την πολυδιάστατη δομή βάσει αυτών.

Η συνάρτηση αναζήτησης δέχεται πάντα τέσσερις παραμέτρους:

```
query_tree(tree, min_letter, max_letter, num_awards)
```

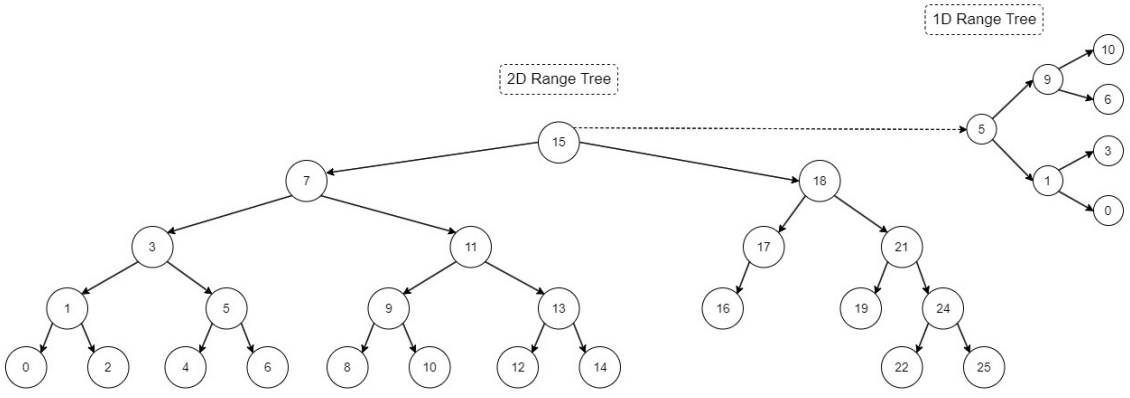
όπου `tree`, η πολυδιάστατη δομή που επιστράφηκε από τη συνάρτηση `build_tree()`, δύο γράμματα που αντιπροσωπεύουν το ελάχιστο και το μέγιστο όριο της συντεταγμένης `x`, και έναν αριθμό βραβείων που αντιπροσωπεύει το ελάχιστο όριο της συντεταγμένης `y`.

Η συνάρτηση αφού μετατρέπει τα δύο γράμματα στις αντίστοιχες αριθμητικές τους τιμές[2], αποστέλλει ερώτημα αναζήτησης στη δομή για τα δοθέντα διαστήματα τιμών, κάνοντας χρήση των μεθόδων της. Βάσει των αποτελεσμάτων της αναζήτησης, ανακτά τα δεδομένα των επιστημόνων από το `.csv` αρχείο (χρησιμοποιώντας το `index`) και τα επιστρέφει ως μία λίστα `final_results` με τη μορφή λεξικών.

2.1 RANGE TREE

Για την αποτελεσματική διαχείριση των δισδιάστατων σημείων με συντεταγμένες `x`, `y`, υλοποιήσαμε ένα 2D Range Tree. Η προσέγγιση που ακολουθήσαμε περιλαμβάνει την κατασκευή του από ισοσταθμισμένα δυαδικά δέντρα αναζήτησης (BBSTs). Αρχικά, κατασκευάζεται ένα κύριο BBST με βάση τις συντεταγμένες `x` των σημείων. Κάθε κόμβος του κύριου δέντρου αποθηκεύει ένα 1D Range Tree (`y`-tree) που περιλαμβάνει όλα τα σημεία που έχουν την ίδια συντεταγμένη `x` με τον κόμβο. Κάθε `y`-tree είναι με τη σειρά του κι αυτό ένα BBST, αλλά κατασκευασμένο με βάση τις συντεταγμένες `y` των σημείων που περιέχει. Επιτρέπει την αναζήτηση σημείων με βάση τη συντεταγμένη `y` εντός ενός διαστήματος, για σημεία που έχουν την ίδια συντεταγμένη `x`.

Κατά την εισαγωγή ενός νέου σημείου, ελέγχεται αν υπάρχει ήδη κόμβος με την ίδια συντεταγμένη `x`. Αν ναι, το σημείο προστίθεται στο αντίστοιχο `y`-tree του κόμβου. Διαφορετικά, δημιουργείται ένας νέος κόμβος στο 2D δέντρο. Τόσο το 2D δέντρο όσο και τα 1D δέντρα διατηρούνται ισορροπημένα μέσω περιστροφών κόμβων, με βάση τον παράγοντα ισορροπίας του κάθε κόμβου, ο οποίος υπολογίζεται ως η διαφορά των υψών των υπο-δέντρων του. Ακολουθεί μία γραφική απεικόνιση του 2D δέντρου.



Εικόνα 2: Κατασκευή ενός 2D Range Tree, χρησιμοποιώντας BBSTs. Κάθε κόμβος του 2D δέντρου έχει ένα associated 1D Range Tree για την αναζήτηση εύρους εντός ενός y-διαστήματος.

Κατά την αναζήτηση ενός εύρους στο 2D Range Tree, το κύριο δέντρο προσπελαύνεται πρώτα για να βρεθούν οι κόμβοι που ανήκουν στο επιθυμητό διάστημα `x`. Για κάθε κόμβο που βρίσκεται εντός του διαστήματος `x`, προσπελαύνεται το αντίστοιχο `y`-tree για να βρεθούν τα σημεία που «πέφτουν» εντός του

[2] Η μετατροπή των γραμμάτων γίνεται χρησιμοποιώντας την συνάρτηση `letter_normalization()` η οποία μετατρέπει τον χαρακτήρα που εισάγουμε, ασχέτως αν είναι πεζός ή κεφαλαίος σε κάποιον αριθμό μεταξύ του [0-25]. Αυτό επιτυγχάνεται με την συνάρτηση `ord()` η οποία επιστρέφει τον Unicode κωδικό του γράμματος, τον οποίο κανονικοποιούμε στο διάστημα που θέλουμε.

διαστήματος `y`. Η συνδυασμένη προσπέλαση των δύο δέντρων επιτρέπει την αποτελεσματική εύρεση όλων των σημείων που βρίσκονται εντός ενός ερωτήματος διαστήματος (range query).

2.2 K-D TREE

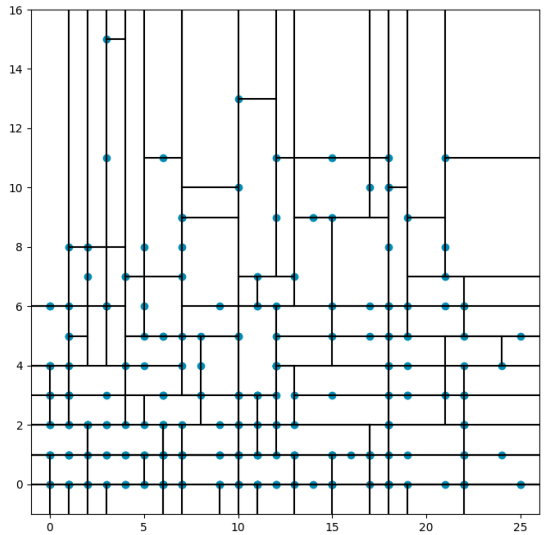
Τα K-D Trees χρησιμοποιούνται για την αποτελεσματική αναζήτηση σημείων σε κοντινή απόσταση, κυρίως σε μικρές διαστάσεις. Επιλέγουμε τα δεδομένα να τοποθετούνται στο δισδιάστατο χώρο (συνεπώς `K=2`). Η κατασκευή του K-D Tree συνεπάγει την διχοτόμηση αυτού του χώρου σε ημιεπίπεδα από δύο άξονες, τον x και τον y.[3]

Επιλέγεται ένας αρχικός άξονας και ένα σημείο (x,y) που θα τον τέμνει. Τα υπόλοιπα σημεία θα ανήκουν πλέον σε δύο υποσύνολα, αριστερά και δεξιά του, ανάλογα με τις συντεταγμένες τους. Όσο προστίθενται σημεία, ανάλογα με το βάθος του δέντρου, οι διαχωρισμοί θα εναλλάσσονται διαδοχικά σε κάθετους και οριζόντιους, οι οποίοι αντιστοιχίζονται στον `x` και στον `y` άξονα.

Στην υλοποίηση του K-D Tree δημιουργούμε την κλάση `Node` που αναπαριστά έναν κόμβο του δέντρου. Αυτές είναι οι βασικές μέθοδοι της κλάσης:

- **`build()`**: δημιουργεί το K-D δέντρο τοποθετώντας τα σημεία τα οποία δημιουργούν εναλλάξ διαχωρισμούς στους άξονες x, y.
- **`insert()`**: εισαγάγει αναδρομικά ένα σημείο στο K-D δέντρο
- **`query()`**: πραγματοποιεί αναζήτηση διαστήματος στο K-D δέντρο. Το διάστημα αναπαρίσταται μέσω της μεταβλητής `rect = x1, y1, x2, y2.`

Το αποτέλεσμα αυτής της κατασκευής είναι ένα ισοσταθμισμένο δυαδικό δέντρο με κάθε κόμβο του να έχει έναν "προσωπικό" άξονα ο οποίος χωρίζει το χώρο σε όλο και μικρότερα ημιεπίπεδα. Για την αναζήτηση των σημείων οδηγούμαστε όλο και βαθύτερα στο δέντρο, αναζητώντας μόνο τα σημεία που βρίσκονται μέσα στο ορθογώνιο διάστημα που επιθυμούμε.



Εικόνα 3: Κατασκευή του K-D Tree χρησιμοποιώντας την Matplotlib. Παρατηρούμε τους κάθετους και οριζόντιους άξονες οι οποίοι δημιουργούνται από κάθε κόμβο και το χωρίζουν σε ημιεπίπεδα.

2.3 R-TREE

Η βασική ιδέα ενός R-tree είναι να ομαδοποιεί τα δεδομένα σε ορθογώνιες περιοχές, οι οποίες αποτελούν τους κόμβους του δέντρου. Καθώς το δέντρο αναπτύσσεται, τα ορθογώνια μπορεί να υπερκαλύπτονται, αλλά προσπαθούν να ελαχιστοποιούν την υπερκάλυψη και το μέγεθός τους.

[3] Είναι φανερό πώς σε παραπάνω διαστάσεις θα έχουμε και παραπάνω άξονες που θα διχοτομούν το τρισδιάστατο, τετραδιάστατο και πολυδιάστατο επίπεδο.

Για την υλοποίηση της δομής, χρησιμοποιήσαμε τη βιβλιοθήκη της Python `rtree`. Η βιβλιοθήκη `rtree` βασίζεται στη βιβλιοθήκη `libspatialindex`, η οποία παρέχει υλοποιήσεις για διάφορες δομές δεδομένων χωρικής ευρετηρίασης, συμπεριλαμβανομένων των R-trees. Ένα από τα κύρια πλεονεκτήματα της χρήσης της βιβλιοθήκης `rtree`, είναι η αποτελεσματικότητα και η ταχύτητα που προσφέρει στις βασικές λειτουργίες εισαγωγής και αναζήτησης ενός R-tree, ακόμα και σε μεγάλα σύνολα δεδομένων.

Ο κώδικας που υλοποιήσαμε περιλαμβάνει την κλάση `RTree`, η οποία χρησιμοποιεί το ευρετήριο (`Index`) της βιβλιοθήκης `rtree` για να διαχειριστεί την εσωτερική δομή του δέντρου και να παρέχει αποτελεσματικές λειτουργίες εισαγωγής και αναζήτησης. Η μέθοδος `insert()` της κλάσης επιτρέπει την εισαγωγή στοιχείων στο δέντρο με βάση τις χωρικές τους συντεταγμένες, ενώ η μέθοδος `search()` παρέχει τη δυνατότητα αναζήτησης στοιχείων με βάση ένα δοθέν bounding box (`query_bbox`).

Όταν καλείται η μέθοδος `insert()`, το στοιχείο προστίθεται στο δέντρο με βάση τις συντεταγμένες του. Αυτό γίνεται με την εύρεση του πιο κατάλληλου κόμβου (ή φύλλου) για την εισαγωγή του στοιχείου, ξεκινώντας από τη ρίζα του δέντρου. Μετά την εισαγωγή, ελέγχεται αν ο κόμβος υπερβαίνει το μέγιστο επιτρεπόμενο πλήθος στοιχείων. Εάν συμβεί αυτό, τότε ο κόμβος πρέπει να διαιρεθεί σε δύο νέους κόμβους. Η βιβλιοθήκη `rtree` χρησιμοποιεί εσωτερικά αλγορίθμους που βελτιστοποιούν τη διαδικασία της διαίρεσης (split), ελαχιστοποιώντας την υπερκάλυψη και το μέγεθος των bounding boxes. Αν ένας κόμβος διαιρεθεί, μπορεί να χρειαστεί να διαιρεθεί και ο γονικός του κόμβος, και ούτω καθεξής, μέχρι να φτάσουμε στη ρίζα του δέντρου. Συνολικά η μέθοδος `insert()` διασφαλίζει ότι το δέντρο παραμένει ισορροπημένο και ότι τα bounding boxes είναι όσο το δυνατόν πιο αποτελεσματικά, προκειμένου να βελτιώσει την απόδοση των αναζητήσεων.

Όταν καλείται η μέθοδος `search()` με ένα δοθέν bounding box ως όρισμα, τότε αυτή καλεί με τη σειρά της τη μέθοδο `intersection()` της βιβλιοθήκης `rtree`, με όρισμα το δοθέν bounding box. Η μέθοδος αυτή ελέγχει τα bounding boxes των κόμβων του R-tree για να βρει όλα τα στοιχεία που τέμνονται με το δοθέν bounding box. Αυτό γίνεται με αποτελεσματικό τρόπο, χρησιμοποιώντας τη δομή του R-tree για να περιορίσει τον αριθμό των στοιχείων που πρέπει να ελεγχθούν. Τέλος, τα στοιχεία που επιστρέφονται από τη μέθοδο `intersection()`, συλλέγονται και επιστρέφονται από τη `search()`.

2.4 QUAD TREE

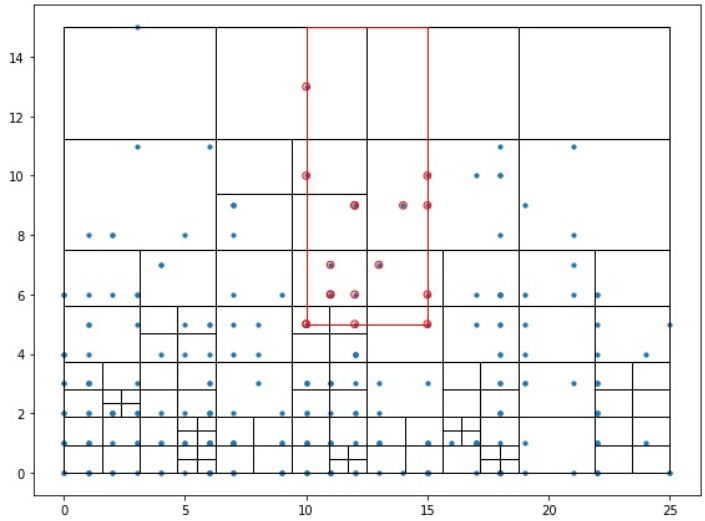
Ένα Quad Tree είναι μία δομή δεδομένων δέντρου που χρησιμοποιείται για την αποτελεσματική οργάνωση και αναζήτηση σημείων σε δισδιάστατους χώρους. Το δέντρο αυτό χωρίζει τον χώρο σε τέσσερα τμήματα (ή κόμβους) και κάθε τμήμα μπορεί να χωριστεί περαιτέρω ανάλογα με το πλήθος των σημείων που περιέχει.

Στον κώδικα που υλοποιήσαμε συναντάμε τρεις κλάσεις:

- **`Point()`**: αναπαριστά ένα σημείο στον δισδιάστατο (2D) χώρο, με συντεταγμένες `x`, `y`, το οποίο μπορεί να φέρει κάποια ωφέλιμη πληροφορία `data`.
- **`Rect()`**: αναπαριστά ένα ορθογώνιο, με συντεταγμένες κέντρου (`cx`, `cy`), πλάτος `w` και ύψος `h`. Διαθέτει μία μέθοδο για τον έλεγχο αν ένα σημείο βρίσκεται εντός του ορθογωνίου (μέθοδος `contains()`) και άλλη μία για τον έλεγχο αν τέμνει με κάποιο άλλο ορθογώνιο (μέθοδος `intersects()`).
- **`QuadTree()`**: αναπαριστά τον κόμβο ενός Quad Tree. Κάθε κόμβος έχει ένα ορθογώνιο `boundary` που είναι ο χώρος που καταλαμβάνει, μία λίστα `points` με τα σημεία που φιλοξενεί και τέσσερις υπο-κόμβους (`sw`, `se`, `ne`, `nw`). Αν ένας κόμβος φτάσει σε έναν καθορισμένο αριθμό σημείων, τότε χωρίζεται στους τέσσερις υπο-κόμβους, καθένας από τους οποίους καταλαμβάνει ένα τεταρτημόριο του αρχικού χώρου.

Κατά την εισαγωγή ενός νέου σημείου, το δέντρο ελέγχει σε ποιον κόμβο ανήκει και το προσθέτει σ' αυτόν. Αν ο κόμβος έχει ήδη το μέγιστο επιτρεπόμενο πλήθος σημείων, το οποίο ορίζουμε τυπικά ως 4, τότε ακολουθείται η διαδικασία διάσπασης του κόμβου και το σημείο προστίθεται στον κατάλληλο υπο-κόμβο απ' αυτούς που προκύπτουν.

Η αναζήτηση σημείων σε ένα Quad Tree είναι αποτελεσματική, καθώς το δέντρο επιτρέπει την ταχεία πρόσβαση σε συγκεκριμένες περιοχές του χώρου. Πιο συγκεκριμένα, αν ζητηθούν να βρεθούν όλα τα σημεία εντός ενός ορθογωνίου, το δέντρο ελέγχει μόνο τους κόμβους που τέμνουν το ορθογώνιο, αγνοώντας όλους τους υπόλοιπους.



Εικόνα 4: Γραφική απεικόνιση του Quad Tree. Με κόκκινο χρώμα είναι σημειωμένο το ορθογώνιο αναζήτησης (search boundary) και τα σημεία που περιέχονται σε αυτό.

4 ΥΛΟΠΟΙΗΣΗ LOCALITY SENSITIVITY HASHING (LSH)

Ο αλγόριθμος Locality-Sensitive Hashing (LSH) είναι ένας αλγόριθμος που χρησιμοποιείται κυρίως στον τομέα της αναζήτησης και της αναγνώρισης πλησιέστερων γειτόνων σε δεδομένα σε πολυδιάστατους χώρους. Ο στόχος του αλγορίθμου LSH είναι να μετατρέψει τα δεδομένα ώστε να είναι ευαίσθητα στη γεωμετρική ομοιότητα, επιτρέποντας την αποδοτική αναζήτηση πλησιέστερων γειτόνων χωρίς την ανάγκη εξέτασης όλων των δυνατών συνδυασμών. Στην παρούσα εργασία ο αλγόριθμος υλοποιείται από τις κλάσεις `MinHash()` και `LSH()` στο αρχείο `lsh/lsh.py`.

Τα βασικά «συστατικά» στοιχεία του LSH είναι τα παρακάτω:

- **Συνάρτηση Hashing**: Ο LSH χρησιμοποιεί μια συνάρτηση κατακερματισμού (hash function) για να μετατρέψει τα δεδομένα από τον αρχικό χώρο σε έναν χώρο μικρότερων διαστάσεων. Αυτή η συνάρτηση hash προσπαθεί να είναι ευαίσθητη στη γεωμετρική ομοιότητα, δηλαδή να "κατακερματίζει" παρόμοια δεδομένα κοντά στην ίδια θέση. Αυτή η λειτουργία υλοποιείται στην κλάση `MinHash` από τις μεθόδους `_hash()`, `build_functions()` και `_signature_matrix()`.
- **Σύγκριση Buckets**: Οι μετατροπές των δεδομένων δημιουργούν "κάδους" ή "buckets" στον νέο χώρο. Δύο δεδομένα που καταλήγουν στον ίδιο κάδο θεωρούνται ότι είναι πιθανά πολύ κοντά γεωμετρικά στον αρχικό χώρο. Αυτή η λειτουργία υλοποιείται στη μέθοδο `_get_candidates()` της κλάσης `LSH`.
- **Αναζήτηση Γειτόνων**: Αφού τα δεδομένα αντιστοιχήθηκαν σε κάδους, μπορούμε να αναζητήσουμε γρήγορα πλησιέστερους γείτονες, εξετάζοντας μόνο τους κάδους που περιέχουν τα δεδομένα που μας ενδιαφέρουν. Αυτή η λειτουργία υλοποιείται στη μέθοδο `neighbors()` της κλάσης `LSH`.

4.1 ΜΕΤΡΙΚΕΣ ΟΜΟΙΟΤΗΤΑΣ

Στο αρχείο `tools.py` έχουμε υλοποιήσει τις συναρτήσεις που χρησιμοποιούνται για την εύρεση της ομοιότητας μεταξύ των διανυσμάτων στον πολυδιάστατο χώρο.

Η μετρική ομοιότητας **Jaccard** είναι ένα μέτρο που χρησιμοποιείται για να αξιολογήσει τον βαθμό της ομοιότητας ανάμεσα σε δύο σύνολα δεδομένων. Υπολογίζει την ομοιότητα ανάμεσα σε δύο σύνολα A και B ως προς τον λόγο:

$$\frac{\text{αριθμός στοιχείων που υπάρχουν και στα δύο σύνολα}}{\text{αριθμός στοιχείων που υπάρχουν σε οποιοδήποτε από τα δύο σύνολα}}$$

Ουσιαστικά, αυτή η μετρική συγκρίνει τον αριθμό των κοινών δυαδικών στοιχείων στα δύο σύνολα με τον συνολικό αριθμό των διαφορετικών δυαδικών στοιχείων που υπάρχουν σε αυτά τα σύνολα. Το πλεονέκτημα της μετρικής Jaccard στην σύγκριση που θέλουμε να πραγματοποιήσουμε είναι ότι δεν λαμβάνει υπόψη τη συχνότητα των διαφορετικών στοιχείων, αλλά μόνο το γεγονός ότι αυτά τα στοιχεία υπάρχουν ή όχι στα σύνολα.[4] Αυτή η μετρική ομοιότητας υλοποιείται στη μέθοδο `jaccard_binary()`.

Η **Κωδικοποίηση One-Hot** (One-Hot Encoding) είναι μια τεχνική που χρησιμοποιείται στη μηχανική μάθηση και την επεξεργασία φυσικής γλώσσας για τη μετατροπή κατηγορικών δεδομένων, όπως κατηγορικές μεταβλητές ή ετικέτες, σε δυαδική (0 ή 1) αριθμητική μορφή. Χρησιμοποιείται συνήθως όταν ασχολούμαστε με κατηγορικά δεδομένα, επειδή πολλοί αλγόριθμοι απαιτούν αριθμητικά δεδομένα εισόδου και η κωδικοποίηση μιας ενιαίας χρήσης μπορεί εύκολα να υποβληθεί σε επεξεργασία από αυτούς τους αλγόριθμους.

Στην παρούσα εργασία αναπαριστούμε το σύνολο των δεδομένων, δηλαδή το πεδίο "education", σε ένα One-Hot μητρώο το οποίο και εισάγουμε στον αλγόριθμο LSH. Αυτό υλοποιείται μέσω της συνάρτησης `one_hot_encoding()` στο αρχείο `lsh/tools.py`, που δέχεται ως όρισμα ένα λεξιλόγιο από shingles. Το λεξιλόγιο το έχουμε δημιουργήσει ήδη μέσα από τη συνάρτηση `kshingle()`.

Υπάρχουν και άλλοι μέθοδοι και μετρικές που θα μπορούσαμε να χρησιμοποιήσουμε για την σύγκριση των ομοιοτήτων, όπως η μετρική cosine (cosine similarity), η οποία χρησιμοποιεί vectors για την εύρεση των ομοιοτήτων. Στην δική μας περίπτωση που θέλουμε να συγκρίνουμε στατικές αναφορές κυρίως σε τοποθεσίες, πανεπιστήμια, αριθμό βραβείων κτλ, είναι πιο ταιριαστή η χρήση της μετρικής Jaccard.

4.1 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΚΕΙΜΕΝΟΥ

Η Προεπεξεργασία Κειμένου (Text Preprocessing) είναι ένα κρίσιμο βήμα στην επεξεργασία φυσικής γλώσσας (NLP) και στην εξόρυξη κειμένου που περιλαμβάνει τον καθαρισμό και τη μετατροπή δεδομένων ακατέργαστου κειμένου σε μορφή κατάλληλη για ανάλυση ή μηχανική μάθηση. Ο στόχος της προεπεξεργασίας κειμένου είναι να κάνει τα δεδομένα κειμένου πιο δομημένα, συνεπή και ευκολότερα στην επεξεργασία.

Έχουμε υλοποιήσει τη συνάρτηση `stemming_and_stopwords()` για να επεξεργαζόμαστε τα κείμενα εκπαίδευσης των επιστημόνων που θα εισάγουμε στο LSH. Εκεί υλοποιοιούνται οι παρακάτω λειτουργίες:

- **Tokenization**: Σπάσιμο του κειμένου σε μικρότερες λέξεις ή υπολέξεις.
- **Stopwords Removal**: Κατάργηση των κοινών λέξεων (stopwords) όπως "and", "the", "in" κ.λπ., καθώς συχνά έχουν μικρό νόημα και μπορεί να είναι υπολογιστικά δαπανηρή η επεξεργασία τους.
- **Noise Removal**: Κατάργηση τυχόν χαρακτήρων και συμβόλων που δεν συνεισφέρουν νόημα στο κείμενο, όπως παρενθέσεις, αριθμούς και HTML tags.
- **Stemming**: Μείωση των λέξεων στη βάση ή τη ρίζα τους και αφαίρεση επιθημάτων και προθεμάτων για να βρεθεί τη ρίζα της λέξης (π.χ. "running" σε "run").

[4] Είναι πλεονέκτημα στην περίπτωσή μας λόγω της φυσιολογίας του κειμένου, το οποίο συνήθως περιέχει επαναλαμβανόμενους όρους που σχετίζονται με την εκπαίδευση.

5 ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΑΡΑΔΕΙΓΜΑΤΑ

Το κύριο πρόγραμμα αφού δεχθεί τα inputs του ελαχίστου ποσοστού ομοιότητας, των αρχικών των επιθέτων, των ελαχίστων αριθμών βραβείων και της δομής που θέλουμε να χρησιμοποιήσουμε, εισέρχεται σε μια δομή ελέγχου.

Η δομή ελέγχου δημιουργεί τις δομές των δέντρων και τις τελικές αναζητήσεις ομοιότητας. Αυτά τα δεδομένα είναι τα ορίσματα της `lsh_test()`. Η `lsh_test()` επίσης δέχεται ως όρισμα τον αριθμό των buckets, τον οποίον θέτουμε να είναι μεταβαλλόμενος ανάλογα με το πλήθος των αποτελεσμάτων ώστε να έχουμε όσο γίνεται πιο σταθερό πλήθος candidate pairs[5].

Αυτό είναι το τελικό output του προγράμματος:

```
Εισάγετε ελάχιστο ποσοστό ομοιότητας (0 - 1): 0.4
Εισάγετε διάστημα ονομάτων στη μορφή X,X: e,k
Εισάγετε ελάχιστο αριθμό βραβείων: 2

1. k-d tree
2. Quad tree
3. Range tree
4. R-tree
Επιλέξτε δομή: 3
165 candidates with at least 40 % similarity using jaccard_binary:

1. Similarity: 40.0%
+----------+--------+-----------------------------------------------------------------------------------------------------------------------------------+
| Surname  | Awards |                                                             Education                                                             |
+----------+--------+-----------------------------------------------------------------------------------------------------------------------------------+
| Geschke  |   3    | Charles Matthew Geschke[4] was born in Cleveland, Ohio, on September 11, 1939.[5]  He attended Saint Ignatius High School.[6]  |
|          |        | Geschke earned an AB in classics in 1962 and an MS in mathematics in 1963, both from Xavier University.[5]  He taught mathemat |
|          |        | ics at John Carroll University from 1963 to 1968.[7]  In 1972, he completed his PhD studies in computer science at Carnegie Me |
|          |        | llon University under the advice of William Wulf.[4]  He was a co-author of Wulf's 1975 book The Design of an Optimizing Compi |
|          |        |                                                          ler.[8]                                                                  |
+----------+--------+-----------------------------------------------------------------------------------------------------------------------------------+
| Goldberg |   2    | Goldberg was born in Cleveland, Ohio, on July 22, 1945.  Her parents moved to Chicago, Illinois when she was 11, where she spe |
|          |        | nt the rest of her childhood.[1] She enjoyed problem solving and mathematics from a young age and was encouraged by her teache |
|          |        | rs to pursue mathematics.[1] In 1967, she earned a bachelor's degree in mathematics at the University of Michigan.[2]  Interes |
|          |        | ted in the subject of computing, Goldberg worked as an intern with IBM during the summer of her junior year at college, where  |
```

Τα αποτελέσματα εμφανίζονται σε ζευγάρια με το ποσοστό ομοιότητάς τους σε έναν πίνακα. Αυτό επιτυγχάνεται με χρήση της μεθόδου `display_results()` η οποία δημιουργεί τους πίνακες χρησιμοποιώντας την βιβλιοθήκη `BeautifulTable`.

Αυτά είναι κάποια επιπλέον παραδείγματα σύγκρισης των επιστημόνων:

Input: 0.4 threshold, [K, L], 5 awards
Output: Similarity 57.1%

Surname	Awards	Education
Kleinrock	5	Leonard Kleinrock **was born** in **New York** City on June 13, 1934, to a **Jewish family**,[3] and graduated from the noted Bronx **High School of Science** in 1951. He **received** a Bachelor of Electrical Engineering degree in 1957 from the City College of **New York**, and a master's degree and a doctorate (**Ph.D.**) in electrical engineering and computer science from the **Massachusetts Institute of Technology** in 1959 and 1963 respectively. He then joined the faculty at the **University** of California at Los Angeles (UCLA), where he remains to the present day; during 1991–1995 he served as the chairman of the Computer Science Department there.[4]
Lamport	6	Lamport **was born** into a **Jewish family** in Brooklyn, **New York**, the son of Benjamin and Hannah Lamport (née Lasser).[citation needed] His father was an immigrant from Volkovisk in the Russian Empire (now Vawkavysk, Belarus)[10] and his mother was an immigrant from the Austro-Hungarian Empire, now southeastern Poland. A graduate of Bronx **High School of Science**, Lamport **received** a B.S. in mathematics from the **Massachusetts Institute of Technology** in 1960, followed by M.A. (1963) and **Ph.D.** (1972) degrees in mathematics from Brandeis **University**.[11] His dissertation, The analytic Cauchy problem with singular data, is about singularities in analytic partial differential equations.[12]

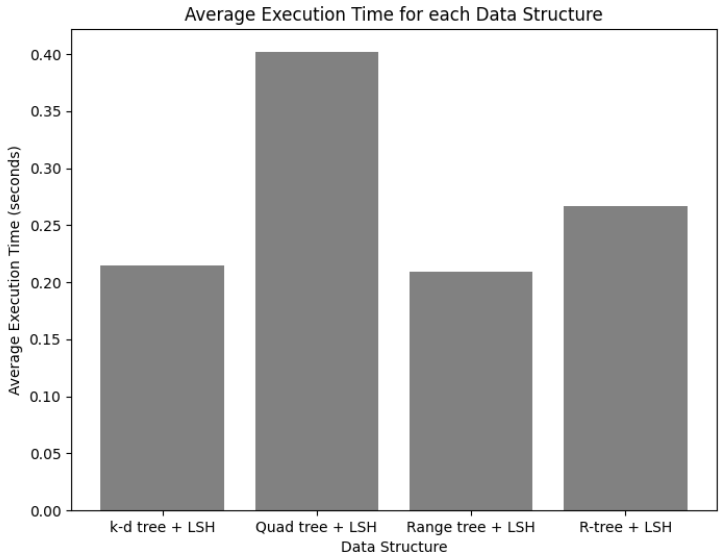
[5] Όταν έχουμε λίγα results και χρησιμοποιούμε πολλά buckets, είναι απίθανο να παρατηρήσουμε όμοιες εξόδους από το `lsh_test()`.

```
Input: 0.6 threshold, [M, X], 2 awards
Output: Similarity: 60.0%
```

Surname	Awards	Education
Muggleton	4	Muggleton received his Bachelor of Science degree in computer science (1982) and Doctor of Philosophy in artificial intelligence (1986) supervised by Donald Michie at the University of Edinburgh.[12]
Wadler	4	Wadler received a Bachelor of Science degree in mathematics from Stanford University in 1977, and a Master of Science degree in computer science from Carnegie Mellon University in 1979.[6] He completed his Doctor of Philosophy in computer science at Carnegie Mellon University in 1984. His thesis was entitled "Listlessness is better than laziness" and was supervised by Nico Habermann.[7][8]

5.1 ΣΥΓΚΡΙΣΗ ΥΛΟΠΟΙΗΣΕΩΝ

Στο αρχείο `experiments.py` πραγματοποιούμε 10 επαναλήψεις του κύριου προγράμματος με τυχαίο εύρος γραμμάτων, τυχαίο αριθμό βραβείων και τυχαίο ποσοστό ομοιότητας μέσα στο διάστημα [0.4, 0.7]. Κάθε υλοποίηση χρονομετρείται και εν τέλει υπολογίζονται οι τελικοί μέσοι όροι για κάθε δομή που υλοποιήσαμε. Τα αποτελέσματα είναι τα παρακάτω:



Εικόνα 5: Το plot τα αποτελέσματα με τους χρόνους των υλοποιήσεων.

Παρατηρούμε τα εξής:

- **Quad tree**: Η δομή αυτή κατανάλωσε τον περισσότερο χρόνο, φτάνοντας τα 0,4 δευτερόλεπτα. Αυτό μπορεί να οφείλεται σε διάφορους παράγοντες, όπως η διανομή των σημείων ή ο τρόπος με τον οποίο έχει υλοποιηθεί το Quad tree.
- **R-tree**: Ακολούθησε με χρόνο λίγο πάνω από 0,25 δευτερόλεπτα. Τα R-trees είναι γενικά πιο πολύπλοκα στην κατασκευή τους, και η απόδοσή τους μπορεί να επηρεαστεί από τη διανομή των σημείων και το μέγεθος των bounding boxes.
- **k-d tree και Range tree**: Και οι δύο δομές είχαν παρόμοιο χρόνο, με το k-d tree να καταγράφει περίπου 0,21 δευτερόλεπτα και το Range tree 0,2 δευτερόλεπτα. Αυτό δείχνει ότι οι δύο δομές είναι σχετικά αποτελεσματικές για το συγκεκριμένο σύνολο δεδομένων και για τα ερωτήματα που τέθηκαν.

Συμπερασματικά, οι παραπάνω μετρήσεις δείχνουν τη σημασία της επιλογής της κατάλληλης δομής δεδομένων ανάλογα με τις ανάγκες της εφαρμογής. Ενώ το Quad tree μπορεί να είναι ιδανικό για ορισμένες εφαρμογές, στη συγκεκριμένη περίπτωση φαίνεται να είναι το πιο αργό. Αντίθετα, τα k-d και Range trees παρουσίασαν την καλύτερη απόδοση για το συγκεκριμένο σύνολο δεδομένων.