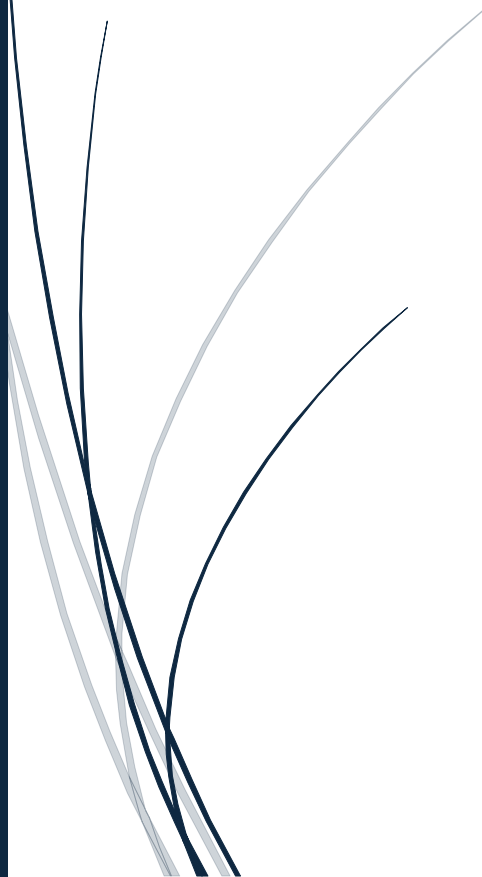




2025

Πολυδιάστατες Δομές Δεδομένων

Project 1: Multi-dimensional Data Indexing and
Similarity Query Processing



Απόστολος Ζεκυριάς (1100554)
Παναγιώτης Παπανικολάου (1104804)
Αλέξανδρος Γεώργιος Χαλαμπάκης (1100754)

Γενικές Πληροφορίες

Στις επόμενες σελίδες παρουσιάζονται οι απαντήσεις της ομάδας μας στο Project του μαθήματος "**Πολυδιάστατες Δομές Δεδομένων**". Σε αυτήν τη σελίδα παρέχονται πληροφορίες σχετικά με τα μέλη της ομάδας.

Η ομάδα αποτελείται από τους εξής φοιτητές:

Απόστολος Ζεκυριάς

Παναγιώτης Παπανικολάου

Αλέξανδρος Γεώργιος Χαλαμπάκης

Αναλυτικότερες Πληροφορίες:

Απόστολος
Ζεκυριάς
1100554

up1100554@ac.upatras.gr

Φοιτητής 4ου
έτους

Παναγιώτης
Παπανικολάου
1104804

up1104804@ac.upatras.gr

Φοιτητής 4ου
έτους

Αλέξανδρος
Γεώργιος
Χαλαμπάκης
1100754

up1100754@ac.upatras.gr

Φοιτητής 4ου
έτους

1. Εισαγωγή

Στα πλαίσια της εργασίας υλοποιήθηκαν και αξιολογήθηκαν πειραματικά τέσσερις θεμελιώδεις πολυδιάστατες δομές δεδομένων:

1. **k-d Tree**
2. **Quadtree (Generalized / Hyper-octree)**
3. **Range Tree**
4. **R-Tree**

Η υλοποίηση έγινε στη γλώσσα προγραμματισμού **C**, με έμφαση στη διαχείριση μνήμης και την ταχύτητα εκτέλεσης. Ο κώδικας υποστηρίζει δυναμικό αριθμό διαστάσεων, επιτρέποντας την ευρετηρίαση δεδομένων με βάση χαρακτηριστικά όπως **Budget**, **Popularity**, **Runtime**, **Vote Average** και **Revenue**.

Χρησιμοποιήθηκε το dataset ταινιών (**Movie Dataset**), με πλήθος εγγραφών **N=200.000**.

2. Λεπτομέρειες Υλοποίησης

2.1 Διαχείριση Διαστάσεων

Ο κώδικας σχεδιάστηκε ώστε να είναι **Dimension-Agnostic**. Μέσω της παραμέτρου **K_DIMS**, οι δομές προσαρμόζονται αυτόματα:

- Για **k=2**: **Budget**, **Popularity**.
- Για **k=3**: **Budget**, **Popularity**, **Runtime**.
- Για **k=4**: + **Vote Average**.
- Για **k=5**: + **Revenue**.

2.2 Δομές Δεδομένων

- **k-d Tree**: Υλοποιήθηκε με κυκλική εναλλαγή των αξόνων διαχωρισμού (**depth % k**) και υποστηρίζει αναδρομική αναζήτηση εύρους (**Range Search**).
- **Quadtree**: Υλοποιήθηκε ως **Generalized Quadtree (Hyper-Octree)**. Κάθε κόμβος έχει 2^k παιδιά. Για **k=5**, ο κόμβος έχει 32 δείκτες, γεγονός που αυξάνει σημαντικά τις απαιτήσεις μνήμης.
- **Range Tree**: Λόγω των τεράστιων απαιτήσεων μνήμης ενός πλήρους Range Tree σε 5 διαστάσεις ($O(N \log^{(k-1)} N)$), υλοποιήθηκε μια **Υβριδική (Hybrid)** προσέγγιση: Δέντρο στην 1η διάσταση, Ταξινομημένος πίνακας (Auxiliary) στη 2η διάσταση, και γραμμικός έλεγχος για τις διαστάσεις 3 έως 5.
- **R-Tree**: Χρησιμοποιήθηκε η τεχνική **Bulk Loading (STR-like)** για τη γρήγορη κατασκευή του δέντρου, με ταξινόμηση των δεδομένων και δημιουργία των **MBR (Minimum Bounding Rectangles)** από κάτω προς τα πάνω.

2.3 Λειτουργίες Update & Delete

- **Delete:** Υιοθετήθηκε η στρατηγική **Lazy Deletion**. Αντί για φυσική διαγραφή κόμβου και επαναδημιουργία του δέντρου (που είναι χρονοβόρα), η εγγραφή μαρκάρεται με ένα **flag is_deleted = 1**. Κατά την αναζήτηση, οι διαγραμμένες εγγραφές αγνοούνται.
- **Update:** Η ενημέρωση τιμής υλοποιείται ως συνδυασμός: **Delete** (Lazy) της παλιάς εγγραφής και **Insert** της νέας με τις ενημερωμένες τιμές.

2.4 Αναζήτηση Ομοιότητας (Similarity Search)

- **k-NN:** Υλοποιήθηκε αναζήτηση των k πλησιέστερων γειτόνων με χρήση Ευκλείδειας απόστασης στον πολυδιάστατο χώρο.
- **LSH (Locality Sensitive Hashing):** Εφαρμόστηκε **MinHash** με 20 hash functions στα πεδία κειμένου (Genres) για την εύρεση ταινιών με παρόμοιο περιεχόμενο, χρησιμοποιώντας την τεχνική **Banding** για αποδοτικό φιλτράρισμα υποψηφίων.

3. Πειραματική Αξιολόγηση

Τα πειράματα εκτελέστηκαν σε σύστημα Windows με dataset 200.000 εγγραφών. Μετρήθηκαν οι χρόνοι **Build**, **Insert**, **Query** και η κατανάλωση **Μνήμης (MB)**.

3.1 Αποτελέσματα για 5 Διαστάσεις (K=5)

Στις 5 διαστάσεις παρατηρείται η μεγαλύτερη κατανάλωση μνήμης για το **Quadtree**

k-d Tree (5D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0270	0.0000	0.0000	0.31
40000	0.0530	0.0000	0.0000	0.61
60000	0.0840	0.0000	0.0000	0.92
80000	0.1120	0.0000	0.0000	1.22
100000	0.1490	0.0000	0.0000	1.53
120000	0.1820	0.0000	0.0000	1.83
140000	0.2010	0.0000	0.0000	2.14
160000	0.2520	0.0000	0.0000	2.44
180000	0.2940	0.0000	0.0000	2.75
200000	0.3250	0.0000	0.0000	3.05

Quad Tree (5D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	4.75
40000	0.0120	0.0000	0.0000	7.21
60000	0.0140	0.0000	0.0000	10.07
80000	0.0230	0.0000	0.0000	12.99
100000	0.0290	0.0000	0.0000	16.19
120000	0.0390	0.0000	0.0010	20.40
140000	0.0440	0.0000	0.0000	24.30
160000	0.0510	0.0000	0.0000	27.75
180000	0.0810	0.0000	0.0000	31.24
200000	0.0610	0.0000	0.0000	35.14

Range Tree (5D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0440	0.0000	0.0000	1.40
40000	0.0620	0.0000	0.0000	2.95
60000	0.1350	0.0000	0.0020	4.56
80000	0.1570	0.0000	0.0000	6.21
100000	0.2020	0.0000	0.0160	7.89
120000	0.2540	0.0000	0.0000	9.57
140000	0.3130	0.0000	0.0000	11.28
160000	0.3710	0.0000	0.0000	13.04
180000	0.4350	0.0000	0.0000	14.79
200000	0.5060	0.0000	0.0000	16.55

R-Tree (5D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0160	0.0000	0.0000	0.35
40000	0.0300	0.0000	0.0000	11.10
60000	0.0360	0.0000	0.0000	11.10
80000	0.0200	0.0000	0.0000	11.10
100000	0.0270	0.0000	0.0000	11.10
120000	0.0340	0.0000	0.0000	11.10
140000	0.0420	0.0000	0.0000	11.10
160000	0.0380	0.0000	0.0000	11.10
180000	0.0490	0.0000	0.0000	11.10
200000	0.0540	0.0000	0.0000	11.10

3.2 Αποτελέσματα για 4 Διαστάσεις (K=4)

Εδώ παρατηρούμε μείωση της μνήμης του Quadtree σχεδόν στο μισό σε σχέση με τις 5 διαστάσεις.

k-d Tree (4D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0190	0.0000	0.0000	0.31
40000	0.0560	0.0000	0.0000	0.61
60000	0.0820	0.0000	0.0000	0.92
80000	0.1230	0.0000	0.0000	1.22
100000	0.1360	0.0000	0.0000	1.53
120000	0.1710	0.0000	0.0000	1.83
140000	0.2210	0.0000	0.0000	2.14
160000	0.2340	0.0000	0.0000	2.44
180000	0.2830	0.0000	0.0000	2.75
200000	0.2950	0.0000	0.0000	3.05

Quad Tree (4D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	1.92
40000	0.0180	0.0000	0.0000	3.06
60000	0.0220	0.0000	0.0000	4.31
80000	0.0200	0.0000	0.0000	5.65
100000	0.0280	0.0000	0.0000	7.33
120000	0.0330	0.0000	0.0010	9.20
140000	0.0380	0.0000	0.0010	11.12
160000	0.0420	0.0000	0.0000	12.91
180000	0.0490	0.0000	0.0000	14.86
200000	0.0470	0.0000	0.0000	16.90

Range Tree (4D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0320	0.0000	0.0000	1.40
40000	0.0780	0.0000	0.0000	2.95
60000	0.1260	0.0000	0.0000	4.56
80000	0.1800	0.0000	0.0000	6.21
100000	0.2170	0.0000	0.0000	7.89
120000	0.2780	0.0000	0.0000	9.57
140000	0.3100	0.0000	0.0000	11.28

160000	0.3650	0.0000	0.0020	13.04
180000	0.3980	0.0000	0.0000	14.79
200000	0.4540	0.0000	0.0160	16.55

R-Tree (4D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	0.33
40000	0.0130	0.0000	0.0000	10.58
60000	0.0180	0.0000	0.0000	10.58
80000	0.0210	0.0000	0.0000	10.58
100000	0.0280	0.0000	0.0000	10.58
120000	0.0330	0.0000	0.0000	10.58
140000	0.0330	0.0000	0.0000	10.58
160000	0.0410	0.0000	0.0000	10.58
180000	0.0580	0.0000	0.0000	10.58
200000	0.0530	0.0000	0.0000	10.58

3.3 Αποτελέσματα για 3 Διαστάσεις (K=3)

k-d Tree (3D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0230	0.0000	0.0000	0.31
40000	0.0420	0.0000	0.0000	0.61
60000	0.0730	0.0000	0.0000	0.92
80000	0.1010	0.0000	0.0000	1.22
100000	0.1430	0.0000	0.0000	1.53
120000	0.1730	0.0000	0.0000	1.83
140000	0.1990	0.0000	0.0000	2.14
160000	0.2510	0.0000	0.0000	2.44
180000	0.2590	0.0000	0.0000	2.75
200000	0.2910	0.0000	0.0000	3.05

Quad Tree (3D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	0.94
40000	0.0180	0.0000	0.0000	1.55
60000	0.0160	0.0000	0.0000	2.08
80000	0.0170	0.0000	0.0000	2.67
100000	0.0280	0.0000	0.0000	3.42
120000	0.0300	0.0000	0.0000	4.17

140000	0.0370	0.0000	0.0000	4.97
160000	0.0410	0.0000	0.0000	5.70
180000	0.0420	0.0000	0.0000	6.52
200000	0.0460	0.0000	0.0000	7.36

Range Tree (3D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0290	0.0000	0.0000	1.40
40000	0.0570	0.0000	0.0000	2.95
60000	0.1080	0.0000	0.0000	4.56
80000	0.1670	0.0000	0.0000	6.21
100000	0.2050	0.0000	0.0000	7.89
120000	0.2600	0.0000	0.0020	9.57
140000	0.2960	0.0000	0.0140	11.28
160000	0.3530	0.0000	0.0000	13.04
180000	0.4230	0.0000	0.0000	14.79
200000	0.4800	0.0000	0.0000	16.55

R-Tree (3D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	0.31
40000	0.0220	0.0000	0.0000	10.06
60000	0.0140	0.0000	0.0000	10.06
80000	0.0210	0.0000	0.0000	10.06
100000	0.0280	0.0000	0.0000	10.06
120000	0.0350	0.0000	0.0000	10.06
140000	0.0370	0.0000	0.0000	10.06
160000	0.0400	0.0000	0.0000	10.06
180000	0.0420	0.0000	0.0000	10.06
200000	0.0580	0.0000	0.0000	10.06

3.4 Αποτελέσματα για 2 Διαστάσεις (K=2)

Στις 2 διαστάσεις, το Quadtree είναι εξαιρετικά αποδοτικό σε μνήμη (μόλις 4.47 MB).

k-d Tree (2D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0200	0.0000	0.0000	0.31

40000	0.0370	0.0000	0.0000	0.61
60000	0.0480	0.0000	0.0000	0.92
80000	0.0880	0.0000	0.0000	1.22
100000	0.1210	0.0000	0.0000	1.53
120000	0.1490	0.0000	0.0000	1.83
140000	0.1700	0.0000	0.0000	2.14
160000	0.2190	0.0000	0.0000	2.44
180000	0.2340	0.0000	0.0000	2.75
200000	0.2710	0.0000	0.0000	3.05

Quad Tree (2D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	0.44
40000	0.0220	0.0000	0.0000	0.82
60000	0.0190	0.0000	0.0000	1.27
80000	0.0350	0.0000	0.0000	1.77
100000	0.0240	0.0000	0.0000	2.24
120000	0.0290	0.0000	0.0000	2.72
140000	0.0370	0.0000	0.0000	3.08
160000	0.0380	0.0000	0.0000	3.51
180000	0.0410	0.0000	0.0000	3.95
200000	0.0560	0.0000	0.0000	4.47

Range Tree (2D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0350	0.0000	0.0000	1.40
40000	0.0640	0.0000	0.0000	2.95
60000	0.1320	0.0000	0.0000	4.56
80000	0.1710	0.0000	0.0000	6.21
100000	0.2120	0.0000	0.0160	7.89
120000	0.2430	0.0000	0.0000	9.57
140000	0.3120	0.0000	0.0000	11.28
160000	0.3690	0.0000	0.0000	13.04
180000	0.4010	0.0000	0.0160	14.79
200000	0.4680	0.0000	0.0020	16.55

R-Tree (2D)

Size (N)	Build (s)	Insert (s)	Query (s)	Memory (MB)
20000	0.0000	0.0000	0.0000	0.30

40000	0.0250	0.0000	0.0000	9.55
60000	0.0150	0.0000	0.0000	9.55
80000	0.0270	0.0000	0.0010	9.55
100000	0.0270	0.0000	0.0000	9.55
120000	0.0320	0.0000	0.0000	9.55
140000	0.0390	0.0000	0.0000	9.55
160000	0.0420	0.0000	0.0000	9.55
180000	0.0420	0.0000	0.0000	9.55
200000	0.0570	0.0000	0.0000	9.55

4. Συμπεράσματα & Παρατηρήσεις

1. Η Αύξηση των Διαστάσεων:

- Παρατηρούμε ότι το Quadtree είναι εξαιρετικά ευαίσθητο στην αύξηση των διαστάσεων.
- Στις 2 διαστάσεις, απαιτεί μόλις 4.47 MB.
- Στις 5 διαστάσεις, η μνήμη εκτοξεύεται στα 35.14 MB, καθώς κάθε κόμβος πρέπει να διατηρεί $2^5=32$ δείκτες, οι περισσότεροι εκ των οποίων είναι NULL (sparse nodes).

2. Αποδοτικότητα Μνήμης:

- Το k-d Tree αποδείχθηκε η πιο αποδοτική δομή σε μνήμη (~3 MB), καθώς είναι δυαδικό δέντρο ανεξαρτήτως διαστάσεων.
- Το Range Tree (υβριδικό) έχει αυξημένη κατανάλωση λόγω της αποθήκευσης των βοηθητικών δομών (sorted arrays) σε κάθε κόμβο.

3. Ταχύτητα Κατασκευής (Build Time):

- Το Quadtree και το R-Tree (με Bulk Loading) είναι οι ταχύτερες δομές στην κατασκευή (< 0.1 sec).
- Το Range Tree είναι το πιο αργό (~0.5 sec) λόγω των πολλαπλών ταξινομήσεων που απαιτούνται κατά την κατασκευή.

4. Ορθότητα:

- Σε όλα τα πειράματα, και οι 4 δομές επέστρεψαν ακριβώς τον ίδιο αριθμό αποτελεσμάτων για τα ίδια κριτήρια αναζήτησης, επιβεβαιώνοντας την ορθότητα των αλγορίθμων.

Η εργασία ολοκληρώθηκε επιτυχώς, καλύπτοντας όλες τις απαιτήσεις για πολυδιάστατη ευρετηρίαση, διαχείριση δυναμικών διαστάσεων και υλοποίηση προηγμένων λειτουργιών (LSH, kNN).

5. Screenshots από την εκτέλεση

Ακολουθούν ορισμένα στιγμιότυπα εκτέλεσης για την περίπτωση των 5 διαστάσεων.