

# PROJECT

## Πολυδιάστατες Δομές Δεδομένων



**ΔΗΜΗΤΡΗΣ ΚΑΤΣΑΝΟΣ ΑΜ: 1093384**

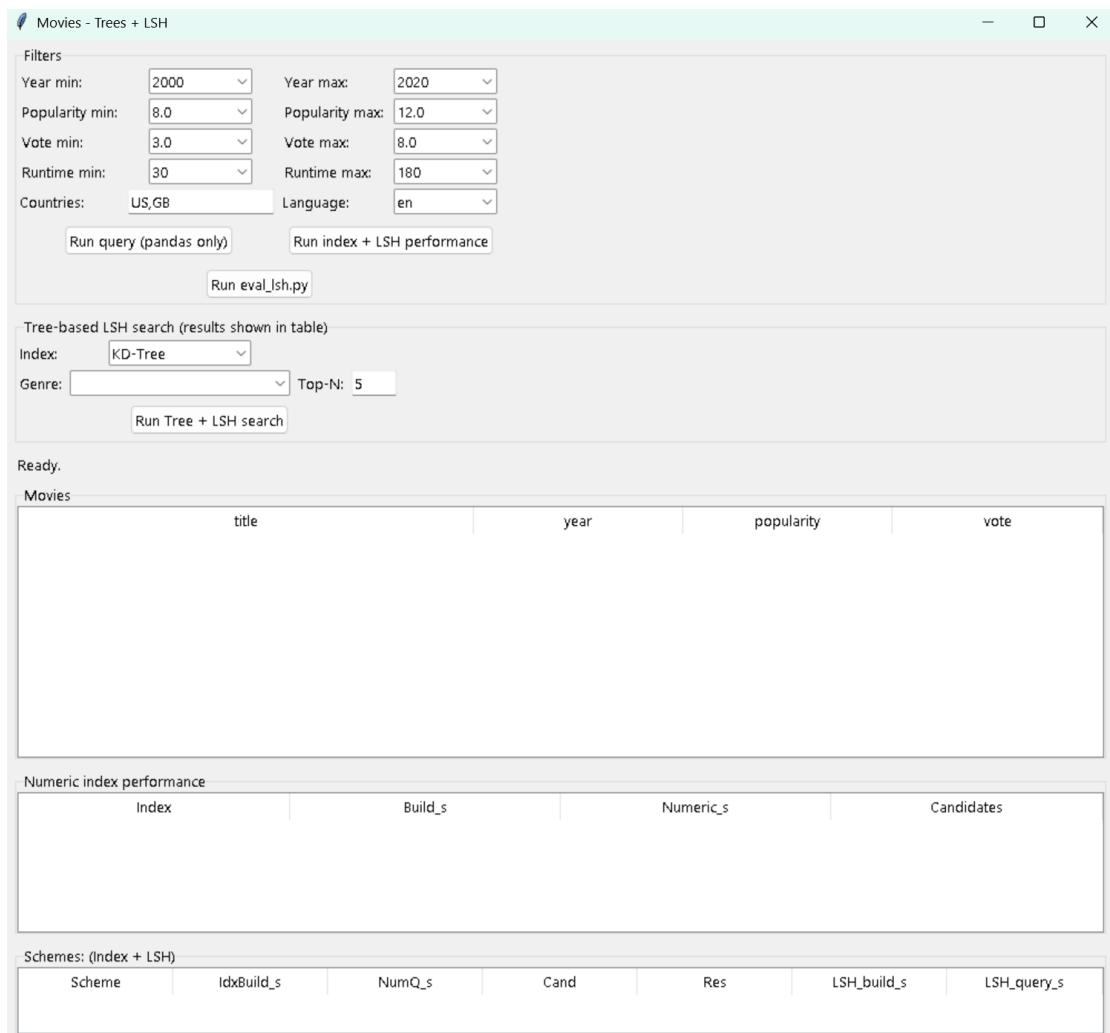
**ΑΝΔΡΕΑΣ ΖΑΦΕΙΡΟΠΟΥΛΟΣ ΑΜ: 1093361**

**ΘΑΛΗΣ-ΧΡΥΣΟΣΤΟΜΟΣ ΤΣΙΩΝΑΣ ΑΜ: 1097467**

**ΙΩΑΝΝΗΣ ΓΡΗΓΟΡΟΠΟΥΛΟΣ ΑΜ: 1097444**

Ασχοληθήκαμε με το Project 1 στο οποίο υλοποιήσαμε ένα σύστημα αποδοτικής αναζήτησης ταινιών, το οποίο υποστηρίζει πολυδιάστατα αριθμητικά ερωτήματα και αναζήτηση ομοιότητας σε κατηγορικά χαρακτηριστικά. Στόχος μας ήταν η μείωση του χρόνου αναζήτησης μέσω κατάλληλων δομών ευρετηρίασης (Indexes) και τεχνικών hashing, σε σύγκριση με απλή σειριακή αναζήτηση, η οποία θα ήταν πολύ αργή λόγο του μεγάλου όγκου δεδομένων. Επίσης δημιουργήσαμε και ένα GUI για την καλύτερη αλληλεπίδραση μας με το Project.

# To GUI



## *Dataset και Προεπεξεργασία*

Το Dataset που κατεβάσαμε από το link της αναφοράς περιλαμβάνει έναν μεγάλο όγκο από πληροφορίες για ταινίες, όπως έτος, popularity, μέσο όρο βαθμολογίας, διάρκεια, χώρες παραγωγής, γλώσσες παραγωγής, genres κ.α. Η φόρτωση και προεπεξεργασία των δεδομένων υλοποιείται στο αρχείο main.py μέσω των συναρτήσεων load\_dataset() και preprocess\_dataset(). Σε αυτό το στάδιο γίνεται καθαρισμός των δεδομένων, μετατροπή τύπων και εξαγωγή δομών που διευκολύνουν τις επόμενες φάσεις (π.χ. μετατροπή genres σε σύνολα). Στην συνέχεια το βασικό dataframe δημιουργείται με την build\_base\_pool() το οποίο χρησιμοποιείται τόσο από το main.py όσο και από το GUI (gui\_tk.py).

## *Δομές Ευρετηρίασης (Indexes)*

Για την επιτάχυνση των numeric range queries υλοποιήθηκαν οι τέσσερις δομές ευρετηρίασης που μας ζητήθηκαν και από την εκφώνηση. Συγκεκριμένα KD-Tree (kd\_tree.py), Quad\_tree (quad\_tree.py), Range-Tree (range\_tree.py) και R-Tree (r\_tree.py). Κάθε δομή χρησιμοποιείται για να φιλτράρει γρήγορα τα δεδομένα βάσει αριθμητικών κριτηρίων και να επιστρέψει ένα σύνολο υποψήφιων αποτελεσμάτων (candidates).

Παρότι το ερώτημα είναι πολυδιάστατο (έως 5 αριθμητικές διαστάσεις), κάθε δομή δεδομένων κάνει index σε διαφορετικό αριθμό διαστάσεων. Το KD-Tree υποστηρίζει queries σε περισσότερες διαστάσεις, ενώ το Quad-Tree και το R-Tree χρησιμοποιούνται σε 2 διαστάσεις και το Range-Tree σε 1 διάσταση. Τα υπόλοιπα φίλτρα εφαρμόζονται στη συνέχεια πάνω στα candidates, ώστε το τελικό αποτέλεσμα να παραμένει σωστό και η σύγκριση των σχημάτων Index + LSH να είναι δίκαιη.

## *Candidates και Φιλτράρισμα*

Στο GUI, το φιλτράρισμα εκτελείται αρχικά με pandas, ώστε να παραχθεί το βασικό αποτέλεσμα και να υποστηριχθεί η επιλογή genres. Στην συνέχεια, τα indexes χρησιμοποιούνται για αριθμητικό φιλτράρισμα και παραγωγή candidates, με σκοπό τη σύγκριση της απόδοσης τους.

## *Αναζήτηση ομοιότητας με LSH*

Για την αναζήτηση ομοιότητας βάση genres χρησιμοποιείται LSH, υλοποιημένο στο lsh\_text.py. Η μέθοδος βασίζεται σε MinHash signatures και επιτρέπει την αποδοτική εύρεση παρόμοιων ταινιών χωρίς πλήρη σύγκριση όλων των στοιχείων.

# Συνδυαστικά Σχήματα και Αξιολόγηση

Το σύστημα αξιολογείται μέσω συνδυαστικών σχημάτων της μορφής Index + LSH. Η πειραματική αξιολόγηση υλοποιείται με την συνάρτηση evaluate\_indexes() στο main.py και παρουσιάζει χρόνους build, χρόνους query, πλήθος candidates και τελικά αποτελέσματα.

## kNN

Επιπλέον υλοποιήθηκαν kNN queries στο KD-Tree στο αρχείο kd\_tree.py. Τα kNN queries αξιολογούνται μέσω του main.py, συγκρίνοντας τα αποτελέσματα και τον χρόνο εκτέλεσης με μια απλή προσέγγιση που υπολογίζει τις αποστάσεις από όλα τα σημεία του dataset (brute-force)

```
# KNN
def knn_query(
    self,
    query_point: Sequence[float],
    k: int = 5,
    return_distances: bool = False,
) -> List[int] | List[Tuple[int, float]]:
    """Επιστρέφει τους k κοντινότερους γειτονες του query_point (Euclidean).
    Αν return_distances=True επιστρέφει (index, distance), άλλως μόνο indices."""
    if self.root is None or self.k == 0 or k <= 0:
        return []

    if len(query_point) != self.k:
        raise ValueError("query_point must have length equal to K dimensions")

    k = min(k, self.n_points)

    # max-heap με (-dist_sq, index) για να κρατάμε τοys k kalyterous
    best: List[Tuple[float, int]] = []
    self._knn_search(self.root, query_point, k, best)

    # ταξινομίζουμε πιο κοντινό σε μια μακρινό
    best_sorted = sorted((-d2, idx) for d2, idx in best), key=lambda x: x[0]

    if return_distances:
        return [(idx, dist ** 0.5) for dist, idx in best_sorted]
    return [idx for dist, idx in best_sorted]

def _knn_search(
    self,
    node: Optional[KDNode],
    query_point: Sequence[float],
    k: int,
    best: List[Tuple[float, int]],
) -> None:
    """Αναδρομική αναζήτηση kNN με pruning."""
    if node is None:
        return

    # αποστασι tou node.point apo query
    d2 = 0.0
    p = node.point
    for dim in range(self.k):
        diff = p[dim] - query_point[dim]
        d2 += diff * diff

    # enimerosi heap
    if len(best) < k:
        heapq.heappush(best, (-d2, node.index))
    else:
        # an briakame kalyteros apo ton xeirottero, antikatastasi
        worst_d2, _ = best[0]
        if -d2 > worst_d2:
            heapq.heapreplace(best, (-d2, node.index))

    axis = node.axis
    diff_axis = query_point[axis] - p[axis]

    # prata pame sto πιο κοντινό ypedonto
    near = node.left if diff_axis < 0 else node.right
    far = node.right if diff_axis < 0 else node.left
    self._knn_search(near, query_point, k, best)

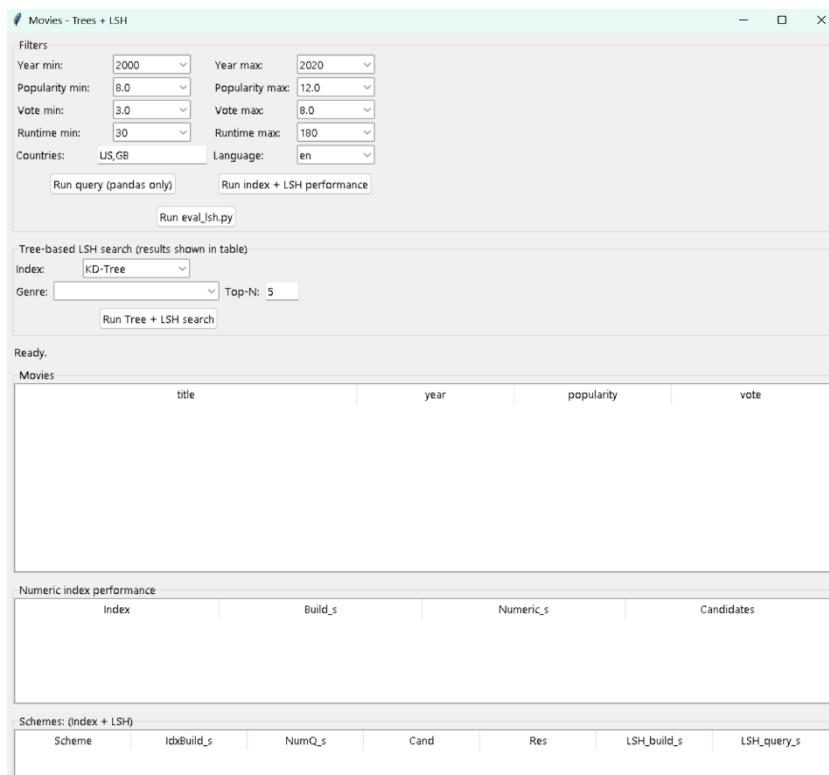
    # pruning
    if len(best) < k:
        self._knn_search(far, query_point, k, best)
    else:
        worst_d2 = -best[0][0]
        if diff_axis * diff_axis <= worst_d2:
            self._knn_search(far, query_point, k, best)
```

# GUI

Το GUI επιτρέπει την εκτέλεση baseline queries, index-based queries και πειραματικής αξιολόγησης. Χρησιμοποιείται για επίδειξη και σύγκριση απόδοσης των μεθόδων, χωρίς να επηρεάζει την βασική λογική του συστήματος.

Ακολουθεί από κάτω ένα παράδειγμα με την λειτουργικότητα του GUI:

## 1. Αρχική Κατάσταση GUI



Ορίζουμε τα αριθμητικά φίλτρα του ερωτήματος (έτος, popularity, βαθμολογία, διάρκεια, χώρα και γλώσσα) πριν την εκτέλεση της αναζήτησης

## 2. Εκτέλεση pandas-only query

The screenshot shows a software application window titled "Movies - Trees + LSH".

**Filters:**

Year min:	2000	Year max:	2020
Popularity min:	8.0	Popularity max:	12.0
Vote min:	3.0	Vote max:	8.0
Runtime min:	30	Runtime max:	180
Countries:	US,GB	Language:	en

**Buttons:**

- Run query (pandas only)
- Run index + LSH performance
- Run eval\_lsh.py

**Tree-based LSH search (results shown in table):**

Index:	KD-Tree
Genre:	
Top-N:	5

**Buttons:**

- Run Tree + LSH search

**Loading dataset...**

**Movies**

	title	year	popularity	vote

**Numeric index performance**

Index	Build_s	Numeric_s	Candidates

**Schemes: (Index + LSH)**

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s

Εκτελούμε αναζήτηση με χρήση pandas, ώστε να εφαρμοστούν όλα τα φίλτρα και να παραχθεί το βασικό αποτέλεσμα (ground truth).

### 3. Αποτελέσματα pandas-only

The screenshot shows a software interface titled "Movies - Trees + LSH". At the top, there are "Filters" for Year min (2000), Year max (2020), Popularity min (8.0), Popularity max (12.0), Vote min (3.0), Vote max (8.0), Runtime min (30), Runtime max (180), Countries (US,GB), and Language (en). Below the filters are three buttons: "Run query (pandas only)", "Run index + LSH performance", and "Run eval\_lsh.py". Underneath these buttons is a link "Run eval\_lsh.py". The next section is titled "Tree-based LSH search (results shown in table)". It includes dropdowns for "Index" (set to "KD-Tree") and "Genre" (set to "Action"), and a "Top-N: 5" input field. A "Run Tree + LSH search" button is located below these controls. A message "Query done. Found 388 movies." is displayed. The main content area is a table titled "Movies" with columns: title, year, popularity, and vote. The table lists 10 movies from the dataset. Below the table is a section titled "Numeric index performance" with tabs for Index, Build\_s, Numeric\_s, and Candidates. At the bottom, there is a section titled "Schemes: (Index + LSH)" with tabs for Scheme, IdxBuild\_s, NumQ\_s, Cand, Res, LSH\_build\_s, and LSH\_query\_s.

title	year	popularity	vote
The Emperor's New Groove	2000	10.97	7.6
Scary Movie 2	2001	9.61	5.8
Final Destination	2000	10.99	6.6
Scary Movie	2000	10.63	6.4
The Mummy Returns	2001	8.59	6.4
Mickey's Magical Christmas: Snowed in at the House of Mouse	2001	8.98	6.9
Child Star: The Shirley Temple Story	2001	9.19	6.7
Legally Blonde	2001	8.64	6.8
American Psycho	2000	8.38	7.4
What Lies Beneath	2000	9.64	6.4

Εμφανίζουμε τα αποτελέσματα της baseline αναζήτησης χωρίς χρήση index, τα οποία χρησιμοποιούμε ως σημείο αναφοράς και για την επιλογή genre.

## 4. Tree + LSH search

**Movies - Trees + LSH**

**Filters**

Year min:	2000	Year max:	2020
Popularity min:	8.0	Popularity max:	12.0
Vote min:	3.0	Vote max:	8.0
Runtime min:	30	Runtime max:	180
Countries:	US,GB	Language:	en

**Run query (pandas only)**   **Run index + LSH performance**

**Run eval\_lsh.py**

**Tree-based LSH search (results shown in table)**

**Index:** KD-Tree   **Genre:** Action   **Top-N:** 5

**Run Tree + LSH search**

Tree + LSH search done (KD-Tree), Genre='Action', returned 5 movies.

**Movies**

	title	year	popularity	vote
1	Bastille Day	2016	9.23	6.3
2	Born to Raise Hell	2010	8.55	4.7
3	Fast and Fierce: Death Race	2020	11.03	5.6
4	Creed	2015	9.26	7.4
5	Never Back Down 2: The Beatdown	2011	9.85	6.5

**Numeric index performance**

Index	Build_s	Numeric_s	Candidates

**Schemes: (Index + LSH)**

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s

**Movies - Trees + LSH**

**Filters**

Year min:	2000	Year max:	2020
Popularity min:	8.0	Popularity max:	12.0
Vote min:	3.0	Vote max:	8.0
Runtime min:	30	Runtime max:	180
Countries:	US,GB	Language:	en

**Run query (pandas only)**   **Run index + LSH performance**

**Run eval\_lsh.py**

**Tree-based LSH search (results shown in table)**

**Index:** Range-Tree   **Genre:** Action   **Top-N:** 5

**Run Tree + LSH search**

Tree + LSH search done (Range-Tree), Genre='Action', returned 5 movies.

**Movies**

	title	year	popularity	vote
1	Bastille Day	2016	9.23	6.3
2	Born to Raise Hell	2010	8.55	4.7
3	Creed II	2018	8.89	7.0
4	Never Back Down 2: The Beatdown	2011	9.85	6.5
5	Puncture Wounds	2014	8.40	4.5

**Numeric index performance**

Index	Build_s	Numeric_s	Candidates

**Schemes: (Index + LSH)**

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s

Εκτελούμε αναζήτηση χρησιμοποιώντας διαφορετικά είδη Tree για το αριθμητικό φιλτράρισμα και LSH για αναζήτηση ομοιότητας με βάση το επιλεγμένο genre. Στα δυο αυτά screenshots δείχνουμε ότι με 2 διαφορετικά αλλά στο ίδιο genre, επιστρέφει διαφορετικά αποτελέσματα μεταξύ τους. Κάτι το οποίο είναι λογικό, καθώς κάθε index επιστρέφει διαφορετικό σύνολο υποψήφιων αποτελεσμάτων πάνω στο οποίο εφαρμόζεται το LSH.

## 5. Εκτέλεση Index + LSH performance

The screenshot shows a software window titled "Movies - Trees + LSH". At the top, there are "Filters" for Year min (2000), Year max (2020), Popularity min (8.0), Popularity max (12.0), Vote min (3.0), Vote max (8.0), Runtime min (30), Runtime max (180), Countries (US,GB), and Language (en). Below the filters are three buttons: "Run query (pandas only)", "Run index + LSH performance", and "Run eval\_lsh.py".

Underneath the filters is a section titled "Tree-based LSH search (results shown in table)". It includes dropdowns for "Index" (set to "Range-Tree") and "Genre" (set to "Action"), and a "Top-N: 5" input field. A "Run Tree + LSH search" button is located below these controls.

A message "Running index + LSH performance..." is displayed above a table titled "Movies". The table has columns: title, year, popularity, and vote. The data is as follows:

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Creed II	2018	8.89	7.0
Never Back Down 2: The Beardown	2011	9.85	6.5
Puncture Wounds	2014	8.40	4.5

Below the "Movies" table is a section titled "Numeric index performance" with four tabs: Index, Build\_s, Numeric\_s, and Candidates. The "Index" tab is selected.

At the bottom, there is a section titled "Schemes: (Index + LSH)" with seven tabs: Scheme, IdxBuild\_s, NumQ\_s, Cand, Res, LSH\_build\_s, and LSH\_query\_s. The "Scheme" tab is selected.

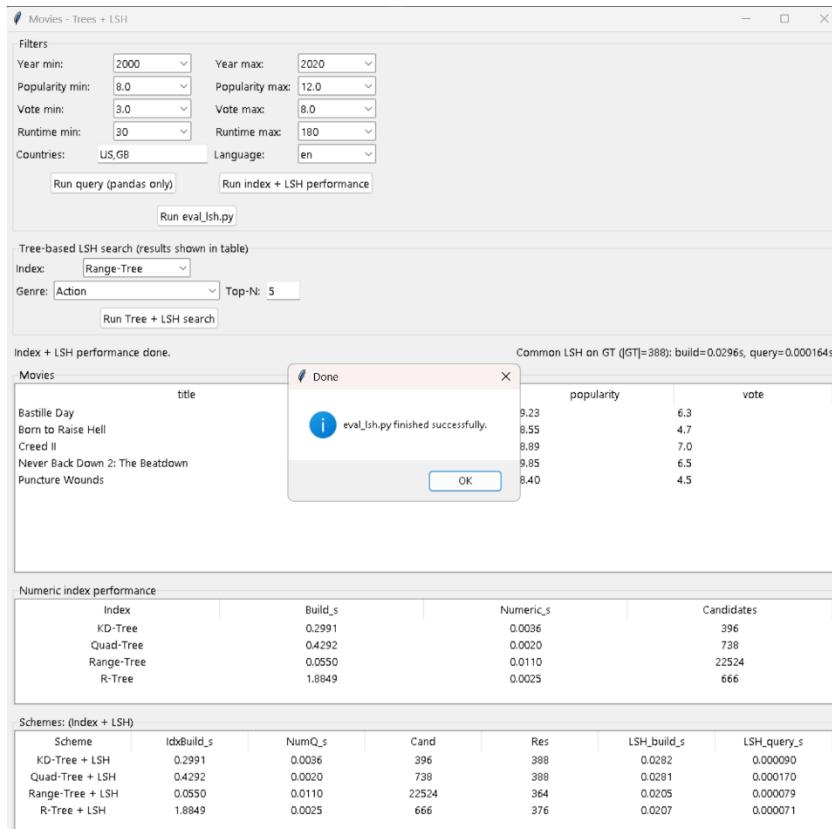
Εκτελούμε την διαδικασία αξιολόγησης της απόδοσης για όλα τα indexes και τα συνδυαστικά σχήματα Index + LSH.

## 6. Numeric Index Performance Table και Schemes: (Index + LSH) Table

Filters																																				
Year min:	2000																																			
Popularity min:	8.0																																			
Vote min:	3.0																																			
Runtime min:	30																																			
Countries:	US,GB																																			
Year max:	2020																																			
Popularity max:	12.0																																			
Vote max:	8.0																																			
Runtime max:	180																																			
Language:	en																																			
<input type="button" value="Run query (pandas only)"/> <input type="button" value="Run index + LSH performance"/>																																				
<input type="button" value="Run eval_lsh.py"/>																																				
Tree-based LSH search (results shown in table)																																				
Index:	Range-Tree																																			
Genre:	Action																																			
Top-N:	5																																			
<input type="button" value="Run Tree + LSH search"/>																																				
Index + LSH performance done.																																				
Common LSH on GT ( GT =388): build=0.0296s, query=0.000164s																																				
<b>Movies</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">title</th><th style="text-align: left;">year</th><th style="text-align: left;">popularity</th><th style="text-align: left;">vote</th></tr> </thead> <tbody> <tr><td>Bastille Day</td><td>2016</td><td>9.23</td><td>6.3</td></tr> <tr><td>Born to Raise Hell</td><td>2010</td><td>8.55</td><td>4.7</td></tr> <tr><td>Creed II</td><td>2018</td><td>8.89</td><td>7.0</td></tr> <tr><td>Never Back Down 2: The Beardown</td><td>2011</td><td>9.85</td><td>6.5</td></tr> <tr><td>Puncture Wounds</td><td>2014</td><td>8.40</td><td>4.5</td></tr> </tbody> </table>		title	year	popularity	vote	Bastille Day	2016	9.23	6.3	Born to Raise Hell	2010	8.55	4.7	Creed II	2018	8.89	7.0	Never Back Down 2: The Beardown	2011	9.85	6.5	Puncture Wounds	2014	8.40	4.5											
title	year	popularity	vote																																	
Bastille Day	2016	9.23	6.3																																	
Born to Raise Hell	2010	8.55	4.7																																	
Creed II	2018	8.89	7.0																																	
Never Back Down 2: The Beardown	2011	9.85	6.5																																	
Puncture Wounds	2014	8.40	4.5																																	
<b>Numeric index performance</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Index</th><th style="text-align: left;">Build_s</th><th style="text-align: left;">Numeric_s</th><th style="text-align: left;">Candidates</th></tr> </thead> <tbody> <tr><td>KD-Tree</td><td>0.2991</td><td>0.0036</td><td>396</td></tr> <tr><td>Quad-Tree</td><td>0.4292</td><td>0.0020</td><td>738</td></tr> <tr><td>Range-Tree</td><td>0.0550</td><td>0.0110</td><td>22524</td></tr> <tr><td>R-Tree</td><td>1.8849</td><td>0.0025</td><td>666</td></tr> </tbody> </table>		Index	Build_s	Numeric_s	Candidates	KD-Tree	0.2991	0.0036	396	Quad-Tree	0.4292	0.0020	738	Range-Tree	0.0550	0.0110	22524	R-Tree	1.8849	0.0025	666															
Index	Build_s	Numeric_s	Candidates																																	
KD-Tree	0.2991	0.0036	396																																	
Quad-Tree	0.4292	0.0020	738																																	
Range-Tree	0.0550	0.0110	22524																																	
R-Tree	1.8849	0.0025	666																																	
<b>Schemes: (Index + LSH)</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Scheme</th><th style="text-align: left;">IdxBuild_s</th><th style="text-align: left;">NumQ_s</th><th style="text-align: left;">Cand</th><th style="text-align: left;">Res</th><th style="text-align: left;">LSH_build_s</th><th style="text-align: left;">LSH_query_s</th></tr> </thead> <tbody> <tr><td>KD-Tree + LSH</td><td>0.2991</td><td>0.0036</td><td>396</td><td>388</td><td>0.0282</td><td>0.000090</td></tr> <tr><td>Quad-Tree + LSH</td><td>0.4292</td><td>0.0020</td><td>738</td><td>388</td><td>0.0281</td><td>0.000170</td></tr> <tr><td>Range-Tree + LSH</td><td>0.0550</td><td>0.0110</td><td>22524</td><td>364</td><td>0.0205</td><td>0.000079</td></tr> <tr><td>R-Tree + LSH</td><td>1.8849</td><td>0.0025</td><td>666</td><td>376</td><td>0.0207</td><td>0.000071</td></tr> </tbody> </table>		Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s	KD-Tree + LSH	0.2991	0.0036	396	388	0.0282	0.000090	Quad-Tree + LSH	0.4292	0.0020	738	388	0.0281	0.000170	Range-Tree + LSH	0.0550	0.0110	22524	364	0.0205	0.000079	R-Tree + LSH	1.8849	0.0025	666	376	0.0207	0.000071
Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s																														
KD-Tree + LSH	0.2991	0.0036	396	388	0.0282	0.000090																														
Quad-Tree + LSH	0.4292	0.0020	738	388	0.0281	0.000170																														
Range-Tree + LSH	0.0550	0.0110	22524	364	0.0205	0.000079																														
R-Tree + LSH	1.8849	0.0025	666	376	0.0207	0.000071																														

**Παρουσιάζουμε συγκριτικά την απόδοση των Indexes και των συνδυαστικών σχημάτων Index + LSH, ως προς τον χρόνο κατασκευής, τον χρόνο εκτέλεσης των αριθμητικών queries, τον αριθμό των υποψήφιων αποτελεσμάτων και τους χρόνους των LSH.**

## 7. Εκτέλεση αξιολόγησης LSH (eval\_lsh.py)



**Τέλος με το κουμπί "Run eval\_lsh.py" εκτελούμε το συγκεκριμένο αρχείο, το οποίο εκτελεί ανεξάρτητη αξιολόγηση της λειτουργικότητάς του LSH και επιβεβαιώνει την σωστή εκτέλεση του.**