ΔΙΑΧΕΙΡΙΣΗ ΣΥΝΘΕΤΩΝ ΔΕΔΟΜΕΝΩΝ

Χωρικά Δεδομένα.

Team

Γκόβαρης Χρήστος-Γρηγόριος, ΑΜ: 5203

Χρονοδιάγραμμα 2ης Εργαστηριακής Άσκησης

Έκδοση	Ημερομηνία	Progress
1.0	4/4/2025	Ανακοίνωση 2ης εργαστηριακής άσκησης
2.1	14/4/2025	Υλοποίηση του ερωτήματος **R-Tree**
2.2	16/4 – 17/4	Υλοποίηση του ερωτήματος **Range Queries**
2.3	17/4/2025	Υλοποίηση του ερωτήματος **kNN Queries**
3.0	23/4/2025	Υλοποίηση Report 2ης εργαστηριακής άσκησης

Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (**Lenovo Ideapad 3**), με τα εξής χαρακτηριστικά πυρήνα:

- **AMD Ryzen 7 (7730U)**
- **8 CPU cores**
- **16 Threads**
- **Boost Clock up to 4.5GHz**
- **Base Clock 2.0GHz**
- **CPU Socket FP6**
- **Intergraded Graphics (Radeon Graphics)**

Επιπλέον, η εργασία υλοποιήθηκε σε **Windows 11 Home**, χρησιμοποιώντας το **Visual Studio**.

Εκτέλεση Προγραμμάτων

Τα υποβληθέντα αρχεία κώδικα προς αξιολόγηση είναι τα **rtree_builder.py**, **range_query.py** και **knn_query.py**, τα οποία περιλαμβάνουν την υλοποίηση σε **Python** όλων των ζητούμενων μερών της εκφώνησης (1 έως 3). Για την εκτέλεση του κώδικα, θα πρέπει να χρησιμοποιηθούν οι ακόλουθες εντολές:

python rtree_builder.py coords.txt offsets.txt

python range_query.py Rtree.txt Rqueries.txt

python knn_query.py Rtree.txt NNqueries.txt 10

2.1 Υλοποίηση του ερωτήματος R-Tree

Στο παρόν ερώτημα, καλούμαστε να υλοποιήσουμε ένα πρόγραμμα, το οποίο διαβάζει δύο αρχεία εισόδου από τη γραμμή εντολών, το **coords.txt** και το **offsets.txt**. Τα δύο αυτά αρχεία περιέχουν τα γεωμετρικά δεδομένα των αντικειμένων σε μορφή πολυγώνων.

Το αρχείο **coords.txt** περιλαμβάνει σημεία σε μορφή **(x, y)**, ενώ το **offsets.txt** περιέχει μετατόπιση αρχής και τέλους για κάθε πολύγωνο, ώστε να εντοπίζονται τα σημεία του μέσα στο **coords.txt**. Για κάθε πολύγωνο, υπολογίζουμε το ελάχιστο περιβάλλον ορθογώνιο (**MBR – Minimum Bounding Rectangle**) που το περικλείει.

Έπειτα, υπολογίζουμε το κέντρο κάθε **MBR** και εφαρμόζουμε τη συνάρτηση **interleave_latlng** (από τη βιβλιοθήκη **pymorton**) για να αντιστοιχίσουμε κάθε **MBR** σε μία μονοδιάστατη τιμή βάσει της **z-order curve**. Η καμπύλη αυτή χρησιμοποιείται για να διατηρείται η χωρική εγγύτητα κατά την ταξινόμηση.

Αφού ταξινομηθούν τα **MBRs** με βάση τις **z-order** τιμές, εφαρμόζεται η τεχνική **bulk loading** για την κατασκευή του **R-tree**. Τα **MBRs** πακετάρονται σε κόμβους χωρητικότητας 20 στοιχείων με ελάχιστο όριο τα 8. Εφόσον ένας κόμβος στο τέλος ενός επιπέδου έχει λιγότερα από 8 στοιχεία, γίνεται κατάλληλη ανακατανομή ώστε να πληρούνται οι περιορισμοί του **R-tree**.

Η διαδικασία συνεχίζεται αναδρομικά για τα ανώτερα επίπεδα, δημιουργώντας κόμβους από τους **MBRs** των κόμβων του επιπέδου κάτω από αυτούς, έως ότου δημιουργηθεί η ρίζα του δέντρου.

Τέλος, για κάθε επίπεδο, τυπώνεται ο αριθμός των κόμβων που δημιουργήθηκαν, και η συνολική δομή του δέντρου γράφεται σε αρχείο **Rtree.txt**, σε προκαθορισμένη μορφή με συντακτικό τύπου **Python list**.

Πιο συγκεκριμένα υλοποιήθηκε ο παρακάτω κώδικας για άνοιγμα και διάβασμα αρχείων:

```
coords_file, offsets_file = sys.argv[1], sys.argv[2]
coords = read_coords(coords_file)
offsets = read_offsets(offsets_file)
```

Από τη γραμμή εντολών διαβάζονται τα ονόματα των αρχείων **coords.txt** και **offsets.txt**. Οι συναρτήσεις **read_coords** και **read_offsets** διαβάζουν τα περιεχόμενα των αρχείων και τα αποθηκεύουν σε λίστες από σημεία και μετατοπίσεις αντίστοιχα.

Υλοποιήθηκε ο παρακάτω κώδικας για υπολογισμό **MBR** και **z-order** για κάθε αντικείμενο:

```
objects = []
for obj_id, start, end in offsets:
    points = coords[start:end+1]
    mbr = compute_mbr(points)
    cx, cy = center(mbr)
    zvalue = interleave_latlng(cy, cx)
    objects.append((obj_id, mbr, zvalue))
```

Για κάθε αντικείμενο που ορίζεται στο αρχείο **offsets**, εντοπίζονται τα αντίστοιχα σημεία από το αρχείο **coords**. Υπολογίζεται το **MBR** για αυτά τα σημεία, έπειτα το κέντρο του **MBR**, και τέλος υπολογίζεται η **z-order** τιμή με τη βοήθεια της συνάρτησης **interleave_latlng**. Τα δεδομένα κάθε αντικειμένου αποθηκεύονται ως (**id**, **MBR**, **zvalue**) σε λίστα.

Υλοποιήθηκε ο παρακάτω κώδικας για ταξινόμηση με βάση **z-order**:

```
objects.sort(key=lambda x: x[2])
```

Τα αντικείμενα ταξινομούνται με βάση την **z-order** τιμή που προέκυψε από το προηγούμενο βήμα, ώστε τα αντικείμενα που είναι κοντά χωρικά να τοποθετηθούν κοντά και στη λίστα.

Υλοποιήθηκε ο παρακάτω κώδικας για κατασκευή του κατώτερου επιπέδου (φύλλα):

```
node_id_counter = iter(range(1000000))
leaf_level = build_level([(obj[0], obj[1]) for obj in objects], 0, node_id_counter)
```

Γίνεται αρχικοποίηση του counter για **node IDs**. Στη συνέχεια, τα αντικείμενα περνάνε στη **build_level** για δημιουργία των κόμβων του κατώτερου επιπέδου. Κάθε κόμβος είναι φύλλο (**is_leaf=0**) και περιλαμβάνει από 8 έως 20 αντικείμενα, όπως ορίζει η **fix_chunks**.

Υλοποιήθηκε ο παρακάτω κώδικας για κατασκευή ανώτερων επιπέδων:

```
levels = [[(node[0], node[1], node[2]) for node in leaf_level]]
current_level = [(node[1], node[3]) for node in leaf_level]
while len(current_level) > 1:
    upper_level = build_level(current_level, 1, node_id_counter)
    levels.append([(node[0], node[1], node[2]) for node in upper_level])
    current_level = [(node[1], node[3]) for node in upper_level]
```

Οι κόμβοι που δημιουργήθηκαν στο κατώτερο επίπεδο (φύλλα) αποθηκεύονται στο πρώτο επίπεδο της δομής **levels**. Έπειτα, επαναλαμβάνεται η διαδικασία κατασκευής κόμβων για το επόμενο επίπεδο (**is_leaf=1**), χρησιμοποιώντας τα **MBRs** των κόμβων του προηγούμενου επιπέδου, μέχρι να απομείνει μόνο ένας κόμβος, ο οποίος αποτελεί τη ρίζα.

Υλοποιήθηκε ο παρακάτω κώδικας για εκτύπωση αριθμού κόμβων ανά επίπεδο:

```
for i, level in enumerate(levels):
    print(f"{len(level)} nodes at level {i}")
```

Για κάθε επίπεδο του δέντρου που κατασκευάστηκε, εκτυπώνεται στο **terminal** ο αριθμός των κόμβων που περιέχει. Η εμφάνιση αυτή είναι χρήσιμη για επαλήθευση της δομής του **R-tree**.

Υλοποιήθηκε ο παρακάτω κώδικας για εγγραφή του **R-tree** στο αρχείο εξόδου:

```
with open("Rtree.txt", 'w') as out:
    for level in levels:
        for node in level:
```

```
            out.write(f"{node}\\n")
```

Ανοίγεται το αρχείο **Rtree.txt** για εγγραφή. Για κάθε κόμβο σε κάθε επίπεδο, γράφεται μία γραμμή στη μορφή **[isnonleaf, node-id, [[id1, MBR1], ...]]**, όπως ακριβώς ζητείται από την εκφώνηση. Το αρχείο ξεκινά από τα φύλλα και τελειώνει στη ρίζα.

2.2 Υλοποίηση του ερωτήματος Range Queries

Στο παρόν ερώτημα, καλούμαστε να υλοποιήσουμε ένα πρόγραμμα το οποίο διαβάζει ένα αρχείο που περιέχει την αποθηκευμένη δομή ενός **R-tree** (**Rtree.txt**) και ένα αρχείο που περιέχει ερωτήσεις εύρους σε μορφή παραθύρου (**MBR**). Στόχος του προγράμματος είναι να εντοπίσει ποια αντικείμενα του δέντρου έχουν **MBR** που τέμνεται με το ορθογώνιο της κάθε ερώτησης.

Η διαδικασία περιλαμβάνει πρώτα τη φόρτωση του **R-tree** στη μνήμη από το αρχείο, δημιουργώντας μια αναπαράσταση με **dictionary** όπου κάθε **node_id** δείχνει στα περιεχόμενα του κόμβου. Κατόπιν, το πρόγραμμα διαβάζει τις ερωτήσεις από το αρχείο **Rqueries.txt**, όπου κάθε γραμμή ορίζει το ερώτημα ως **[x_low, y_low, x_high, y_high]**.

Για κάθε τέτοια ερώτηση, εφαρμόζεται αναδρομική αναζήτηση ξεκινώντας από τη ρίζα του **R-tree**. Εάν ο **MBR** ενός κόμβου ή αντικειμένου τέμνεται με το ορθογώνιο ερώτησης, τότε εξετάζονται οι εγγραφές του κόμβου. Όταν βρεθούν **MBRs** αντικειμένων που τέμνονται με το παράθυρο, αυτά καταγράφονται στα αποτελέσματα.

Τέλος, για κάθε ερώτηση εμφανίζεται στην έξοδο η γραμμή του ερωτήματος, το πλήθος των αποτελεσμάτων και τα **IDs** των αντικειμένων που πληρούν το φίλτρο.

Πιο συγκεκριμένα υλοποιήθηκε ο παρακάτω κώδικας για φόρτωση του **R-tree** από το αρχείο:

```
def load_rtree(file_path):
    rtree = {}
    with open(file_path, 'r') as f:
        for line in f:
            node = ast.literal_eval(line.strip())
            isnonleaf, node_id, entries = node
            rtree[node_id] = {"isnonleaf": isnonleaf, "entries": entries}
    return rtree
```

Η παραπάνω συνάρτηση διαβάζει το αρχείο **Rtree.txt** και για κάθε γραμμή που περιέχει πληροφορίες κόμβου, τις μετατρέπει από **string** σε **Python** αντικείμενο. Το αποτέλεσμα αποθηκεύεται σε λεξικό, όπου κάθε **node_id** δείχνει σε ένα **dictionary** με **flag** για το αν είναι φύλλο ή εσωτερικός κόμβος, και τις αντίστοιχες εγγραφές του.

Υλοποιήθηκε ο παρακάτω κώδικας για έλεγχο αν δύο **MBRs** τέμνονται:

```
def mbr_intersects(mbr1, mbr2):
    return not (
        mbr1[1] < mbr2[0] or
        mbr1[0] > mbr2[1] or
        mbr1[3] < mbr2[2] or
        mbr1[2] > mbr2[3]
    )
```

Ο έλεγχος **intersection** υλοποιείται με βάση τις κλασικές συνθήκες σύγκρουσης ορθογωνίων στον **2D** χώρο. Αν τα δύο **MBR** δεν έχουν κοινό μέρος, η συνάρτηση επιστρέφει **False**, αλλιώς **True**.

Υλοποιήθηκε ο παρακάτω κώδικας για αναδρομική εκτέλεση **range query**:

```
def range_query(node_id, query_rect, rtree, results):
    node = rtree[node_id]
    for entry_id, entry_mbr in node["entries"]:
        if mbr_intersects(entry_mbr, query_rect):
            if node["isnonleaf"]:
                range_query(entry_id, query_rect, rtree, results)
            else:
                results.append(entry_id)
```

Αυτή η συνάρτηση εκτελεί την κύρια λογική του φίλτρου. Ξεκινάει από ένα **node ID**, και για κάθε εγγραφή που τέμνεται με το παράθυρο της ερώτησης, είτε συνεχίζει την αναζήτηση στους υποκόμβους (αν είναι εσωτερικός κόμβος), είτε προσθέτει το **ID** στα αποτελέσματα (αν είναι φύλλο).

Υλοποιήθηκε ο παρακάτω κώδικας ως κύρια συνάρτηση και ανάγνωση ερωτήσεων:

```
def main():
    if len(sys.argv) != 3:
        print("Usage: python range_query.py Rtree.txt Rqueries.txt")
        return
    rtree_file = sys.argv[1]
    queries_file = sys.argv[2]
    rtree = load_rtree(rtree_file)
    root_id = max(rtree.keys())
    with open(queries_file, 'r') as f:
        for i, line in enumerate(f):
            x_low, y_low, x_high, y_high = map(float, line.strip().split())
            query_rect = [x_low, x_high, y_low, y_high]
            results = []
            range_query(root_id, query_rect, rtree, results)
            results.sort()
            print(f"{i} ({len(results)}): {','.join(map(str, results))}")
```

Η **main()** χειρίζεται την είσοδο του προγράμματος. Διαβάζει το αρχείο του δέντρου και το αρχείο των ερωτήσεων. Για κάθε ερώτηση, δημιουργεί το ορθογώνιο της περιοχής και εκτελεί την αναζήτηση ξεκινώντας από τη ρίζα του δέντρου. Στο τέλος, εκτυπώνει τα αποτελέσματα με το ζητούμενο **format**.

2.3 Υλοποίηση του ερωτήματος kNN Queries

Στο παρόν ερώτημα, καλούμαστε να υλοποιήσουμε έναν αλγόριθμο εύρεσης των **k** πλησιέστερων αντικειμένων σε ένα σημείο αναφοράς, χρησιμοποιώντας την υπάρχουσα δομή του **R-tree**. Ο αλγόριθμος που υλοποιείται είναι **Best-First Search (BFS)** με χρήση ουράς προτεραιότητας.

Το πρόγραμμα διαβάζει το **R-tree** από αρχείο **Rtree.txt**, τις ερωτήσεις από αρχείο **NNqueries.txt**, και τον αριθμό **k** από τα ορίσματα της γραμμής εντολών. Κάθε ερώτηση είναι ένα σημείο **q = (x, y)**, για το οποίο πρέπει να βρεθούν τα **k** αντικείμενα (**object MBRs**) που βρίσκονται πιο κοντά βάσει της ευκλείδειας απόστασης.

Η διαδικασία αρχίζει τοποθετώντας τη ρίζα του **R-tree** σε ουρά προτεραιότητας με απόσταση 0. Σε κάθε βήμα, το αντικείμενο με τη μικρότερη απόσταση από το **q** εξάγεται από την ουρά. Αν είναι κόμβος, προστίθενται οι εγγραφές του στην ουρά. Αν είναι αντικείμενο (φύλλο), καταχωρείται ως ένα από τα k αποτελέσματα.

Η διαδικασία συνεχίζεται μέχρι να βρεθούν **k** αντικείμενα. Στο τέλος, για κάθε ερώτηση τυπώνεται η γραμμή της ερώτησης και τα **IDs** των k πλησιέστερων αντικειμένων.

Πιο συγκεκριμένα υλοποιήθηκε ο παρακάτω κώδικας για ανάγνωση και αναπαράσταση του **R-tree**:

```
def load_rtree(file_path):
    rtree = {}
    with open(file_path, 'r') as f:
        for line in f:
            node = ast.literal_eval(line.strip())
            isnonleaf, node_id, entries = node
            rtree[node_id] = {"isnonleaf": isnonleaf, "entries": entries}
    return rtree
```

Η παραπάνω συνάρτηση φορτώνει το **R-tree** από το αρχείο **Rtree.txt**. Κάθε γραμμή μετατρέπεται σε **Python** αντικείμενο και αποθηκεύεται σε **dictionary**, όπου κάθε

node_id δείχνει σε **dictionary** με πληροφορία για το αν είναι εσωτερικός κόμβος και τις εγγραφές του.

Υλοποιήθηκε ο παρακάτω κώδικας για υπολογισμός ελάχιστης απόστασης σημείου–**MBR**:

```
def point_mbr_distance(point, mbr):
    x, y = point
    x_low, x_high, y_low, y_high = mbr
    dx = max(x_low - x, 0, x - x_high)
    dy = max(y_low - y, 0, y - y_high)
    return math.sqrt(dx*dx + dy*dy)
```

Η απόσταση υπολογίζεται μεταξύ ενός σημείου και ενός **MBR**. Αν το σημείο βρίσκεται μέσα στο **MBR**, η απόσταση είναι 0. Αλλιώς, υπολογίζεται η ευκλείδεια απόσταση από το κοντινότερο άκρο.

Υλοποιήθηκε ο παρακάτω κώδικας για αναζήτηση **k** πλησιέστερων αντικειμένων με **Best-First Search**:

```
def knn_search(root_id, point, k, rtree):
    heap = []
    heapq.heappush(heap, (0, ("node", root_id)))
    result = []
    visited = set()
    while heap and len(result) < k:
        dist, (item_type, item_id) = heapq.heappop(heap)
        if (item_type, item_id) in visited:
            continue
        visited.add((item_type, item_id))
        if item_type == "node":
            node = rtree[item_id]
            for entry_id, entry_mbr in node["entries"]:
                d = point_mbr_distance(point, entry_mbr)
                entry_type = "node" if node["isnonleaf"] else "object"
```

```
            heapq.heappush(heap, (d, (entry_type, entry_id)))
        elif item_type == "object":
            result.append(item_id)
    return result
```

Η συνάρτηση αυτή υλοποιεί την αναζήτηση **k** πλησιέστερων γειτόνων. Ξεκινάει από τη ρίζα του δέντρου και χρησιμοποιεί ουρά προτεραιότητας ώστε πάντα να εξετάζεται το αντικείμενο ή κόμβος με τη μικρότερη απόσταση. Όταν εντοπιστεί αντικείμενο (φύλλο), προστίθεται στα αποτελέσματα. Η διαδικασία σταματά μόλις συλλεχθούν **k** αντικείμενα.

Υλοποιήθηκε ο παρακάτω κώδικας για διαχείριση εισόδου και εμφάνιση αποτελεσμάτων:

```
def main():
    if len(sys.argv) != 4:
        print("Usage: python knn_query.py Rtree.txt NNqueries.txt k")
        return
    rtree_file = sys.argv[1]
    queries_file = sys.argv[2]
    k = int(sys.argv[3])
    rtree = load_rtree(rtree_file)
    root_id = max(rtree.keys())
    with open(queries_file, 'r') as f:
        for i, line in enumerate(f):
            x, y = map(float, line.strip().split())
            neighbors = knn_search(root_id, (x, y), k, rtree)
            print(f"{i}: {','.join(map(str, neighbors))}")
```

Η **main()** διαβάζει τα αρχεία εισόδου και το **k** από τη γραμμή εντολών. Για κάθε σημείο αναφοράς στο **NNqueries.txt**, καλεί τη **knn_search** και εμφανίζει τα **k** πλησιέστερα αντικείμενα στη μορφή που ορίζεται στην εκφώνηση.