ΠΑΝΕΠΙΣΤΉΜΙΟ ΙΩΑΝΝΊΝΩΝ

ΤΜΉΜΑ ΜΗΧΑΝΙΚΏΝ ΗΛΕΚΤΡΟΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Διαχείριση Σύνθετων Δεδομένων

Εργασία 2 – Χωρικά Δεδομένα

**ΣΙΑΚΑΒΑΡΑ ΘΕΜΙΣΤΟΚΛΕΙΑ, 4786**

**ΠΑΡΑΔΟΣΗ: ΔΕΥΤΕΡΑ 6 ΜΑΪΟΥ 2024**

Για την υλοποίηση της πρώτης εργασίας, χρησιμοποιήθηκε η γλώσσα προγραμματισμού python, και πιο συγκεκριμένα η έκδοση 3.10.6.

Σε όλα τα προγράμματα έχει γίνει import sys, για να διαβάσουμε ορίσματα από τη γραμμή διαταγών, και import math για την χρήση μαθηματικών συναρτήσεων όπως: math.sqrt, math.ceil, math.floor.

## 1ο Μέρος:

Το ζητούμενο πρόγραμμα για το πρώτο μέρος ονομάζεται **meros1.py** και παίρνει σαν όρισμα ένα αρχείο txt. Για παράδειγμα:

```
>python meros1.py output.txt
```

Στην main συνάρτηση ελέγχεται πρώτα αν έχει δοθεί ο σωστός αριθμός ορισμάτων. Αν όχι, βγάζει μήνυμα λάθους και τερματίζει το πρόγραμμα, αλλιώς το πρόγραμμα συνεχίζει κανονικά. Αφού οριστεί ως input αρχείο το Beijing_restaurants.txt, το ανοίγει και αποθηκεύει όλες τις συντεταγμένες ως πλειάδες. Αφού προσθέσουμε και την γραμμή του αρχείου στην οποία βρίσκεται η κάθε πλειάδα, καλούμε την buildRTree(), τη βασική συνάρτηση του προγράμματος. Έπειτα καλείται η συνάρτηση printStatistics(), ώστε να τυπωθούν τα στατιστικά του δέντρου και τέλος η writeFile() που γράφει το αποτέλεσμα, δηλαδή το δέντρο, στο αρχείο που έχουμε ορίσει στη γραμμή εντολών.

```
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python program.py output_filename")
        sys.exit(1)
    inputFile = "Beijing_restaurants.txt"
    outputFile = sys.argv[1]

    with open(inputFile, 'r') as file:
        numPoints = int(file.readline())
        points = [tuple(map(float, line.split())) for line in file]
        points = enumerate(points, start=1)

        nodeCounter = 0
        treeLevel = 1
        nodes = buildRTree(points, [], 0)
        printStatistics(nodes)
        writeFile(nodes, outputFile)
```

Για τη δημιουργία του δέντρου, έχουν οριστεί στην αρχή του αρχείου οι χωρητικότητες των φύλλων και των μη- φύλλων του. Ο υπολογισμός τους έγινε βάσει την εκφώνηση της άσκησης.

```
#Entry size = 16bytes + 4bytes = 20 bytes
#max entries per leaf node = floor(node capacity/entry size) = floor(1024/20) = 51
MAX_ENTRIES_PER_LEAF = 51
#Entry size = 32 bytes + 4 bytes = 36 bytes
#max entries per non leaf node = floor(node capacity/entry size) = floor(1024/36)=28
MAX_ENTRIES_PER_NON_LEAF = 28
```

Επίσης, έχει οριστεί η κλάση Node, που είναι ο κόμβος του δέντρου και αποτελείται από το ID του, το αν είναι φύλλο ή όχι, και τέλος τις εγγραφές. Έχει, επιπλέον, τη συνάρτηση addEntry(), η οποία προσθέτει εγγραφές στις ήδη αποθηκευμένες του.

```
class Node:
    def __init__(self, nodeID, isLeaf=False):
        self.nodeID = nodeID
        self.isLeaf = isLeaf
        self.entries = []

    def addEntry(self, entry):
        self.entries.append(entry)
```

Κάποιες βοηθητικές συναρτήσεις, για τη δημιουργία του δέντρου, είναι οι εξής:

- calculateMBR(): Η συνάρτηση αυτή υπολογίζει το MBR βρίσκοντας το μικρότερο και το μεγαλύτερο X και Y από τα σημεία, και τα επιστρέφει.
- sortByX(): Επιστρέφει τη συντεταγμένη X του σημείου- ορίσματος.
- sortByY(): Επιστρέφει τη συντεταγμένη Y του σημείου- ορίσματος.
- sortByID(): Επιστρέφει το id του κόμβου - ορίσματος.
- divideSlices(): Χωρίζει έναν πίνακα δεδομένων σε συγκεκριμένο αριθμό πινάκων, ο οποίος δίνεται στο όρισμα. Επιστρέφει το αποτέλεσμα.

```
def calculateMBR(points):
    minX = min(point[1][0] for point in points)
    maxX = max(point[1][0] for point in points)
    minY = min(point[1][1] for point in points)
    maxY = max(point[1][1] for point in points)
    return [minX, minY, maxX, maxY]

def sortByX(point):
    return point[1][0]

def sortByY(point):
    return point[1][1]

def sortByID(point):
    return point[0]

def divideSlices(data, sizeOfData):
    slices = []
    slices = [data[i:i+sizeOfData] for i in range(0, len(data), sizeOfData)]
    return slices
```

Αφού έχουν οριστεί οι παραπάνω συναρτήσεις, ορίζουμε και την buildRTree(), η οποία παίρνει ως ορίσματα τα σημεία με τις συντεταγμένες, έναν πίνακα άδειο, που θα γεμίζει με κόμβους και τελικά θα είναι το δέντρο, και τέλος έναν counter = 0 αρχικά, ο οποίος δείχνει σε ποιον κόμβο έχουμε σταματήσει.

Αρχικά, πέρα από τον ορισμό των global μεταβλητών, ορίζονται και οι μεταβλητές maxEntries και nodeType, ανάλογα με το αν βρισκόμαστε στα φύλλα ή όχι του δέντρου. Έπειτα, ορίζουμε το συνολικό αριθμό των κόμβων (N), τον αριθμό των κομματιών (numSlices) και το μέγεθος του κάθε επιπέδου (M). Ταξινομούμε και τα σημεία με βάση την X συντεταγμένη, χρησιμοποιώντας την βοηθητική συνάρτηση sortByX() που έχει ήδη αναφερθεί.

Στην περίπτωση που έχουμε παραπάνω από ένα κομμάτι, πρόκειται για κόμβους που δεν είναι η ρίζα. Έτσι, αν πρόκειται για φύλλα του δέντρου, χωρίζουμε τα σημεία σε M κομμάτια και τα ταξινομούμε βάσει την Y συντεταγμένη, αλλιώς αν είναι για ενδιάμεσους κόμβους, τα ταξινομούμε κατευθείαν βάσει του nodeID. Στη συνέχεια, χωρίζουμε το παραπάνω αποτέλεσμα σε τόσα κομμάτια όσες οι δυνατές εγγραφές του κόμβου. Για κάθε κομμάτι δημιουργούμε και έναν νέο κόμβο με το ID και τις εγγραφές του, και γεμίζουμε σιγά σιγά τον πίνακα treeArray. Υπολογίζουμε το MBR κάθε κομματιού και τα ID του κάθε κόμβου για το επόμενο επίπεδο, και δημιουργούμε τα νέα δεδομένα που θα είναι όρισμα της αναδρομικής αυτής συνάρτησης. Έτσι γίνεται η αναδρομή έως ότου βρεθούμε στη ρίζα του δέντρου.

Αλλιώς αν πρόκειται για τη ρίζα του δέντρου, γίνεται η ταξινόμηση των σημείων με βάση τη Y συντεταγμένη, δημιουργείται ο κόμβος της ρίζας και προστίθεται στο δέντρο. Η συνάρτηση τελειώνει και επιστρέφει τον πίνακα treeArray, που περιέχει το R-Tree.

```
def buildRTree(points, treeArray, currentCounter):
    global treeLevel
    global nodeCounter

    if treeLevel == 1:
        maxEntries = MAX_ENTRIES_PER_LEAF
        nodeType = True
    else:
        maxEntries = MAX_ENTRIES_PER_NON_LEAF
        nodeType = False

    sortedPoints = sorted(points, key = sortByX)

    #total number of leaf nodes
    N = math.ceil(len(sortedPoints) / maxEntries)
    numSlices = math.ceil(math.sqrt(N))
    #size of each group of rectangles
    M = numSlices * maxEntries

    if numSlices > 1:
        if maxEntries == MAX_ENTRIES_PER_LEAF:
            slices = list(divideSlices(sortedPoints, M))
            sortedSlices = [sorted(item, key = sortByY) for item in slices]
        else:
            sortedSlices = [sorted(sortedPoints, key = sortByID)]

        listSlices = [j for i in sortedSlices for j in i]

        slices = list(divideSlices(listSlices, maxEntries))

        mbr = [calculateMBR(item) for item in slices]

        currentCounter = nodeCounter

        for s in slices:
            newNode = Node(nodeCounter, nodeType)
            for entry in s:
                newNode.addEntry(entry)
            nodeCounter +=1
            treeArray.append(newNode)

        for s in slices:
            newNode = Node(nodeCounter, nodeType)
            for entry in s:
                newNode.addEntry(entry)
            nodeCounter +=1
            treeArray.append(newNode)

        nodeIDs = [i for i in range(currentCounter, nodeCounter)]

        treeLevel += 1

        newPoints = []
        for a, b in zip(nodeIDs, mbr):
            dataEntry = [a, [b[0], b[1], b[2], b[3]]]
            newPoints.append(dataEntry)

        buildRTree(newPoints, treeArray, currentCounter)
    else:
        data = sorted(points, key=sortByY)
        root  = Node(nodeCounter, False)
        for d in data:
            root.addEntry(d)
        treeArray.append(root)
        nodeCounter +=1
    return treeArray
```

Για τη συνάρτηση printStatistics() έχει επίσης οριστεί μία βοηθητική συνάρτηση area(mbr), που παίρνει σαν όρισμα το mbr και επιστρέφει την περιοχή.

```
def area(mbr):
    if len(mbr) ==2:
        return 0
    return (mbr[2] - mbr[0])* (mbr[3] - mbr[1])
```

Η συνάρτηση printStatistics() παίρνει ως όρισμα το δέντρο που έχουμε δημιουργήσει και τυπώνει το ύψους του, τον αριθμό των κόμβων σε κάθε επίπεδο και το μέσο εμβαδό των MBRs σε κάθε επίπεδο. Αυτό συμβαίνει με ένα while loop, ξεκινώντας από τη ρίζα του δέντρου και συνεχίζοντας προς τα φύλλα. Αρχικά, ο πίνακας nextNodes, που μας δείχνει ποιοι κόμβοι αντιστοιχούν στο συγκεκριμένο επίπεδο είναι κενός, άρα πρόκειται για τη ρίζα. Ανατρέχοντας σε κάθε εγγραφή της γεμίζουμε τον παραπάνω πίνακα με τα recordID και βρίσκουμε και τη συνολική περιοχή, καλώντας τη βοηθητική area(). Αυτό επαναλαμβάνεται, με τη διαφορά ότι πλέον πρόκειται είτε για ενδιάμεσους κόμβους είτε φύλλα, έως ότου το επίπεδο να γίνει 0.

```
def printStatistics(nodes):
    print("Height of the tree:", treeLevel)
    level = treeLevel
    nextNodes = []
    totalArea = 0

    while level > 0:
        startNode = []
        totalArea = 0
        if nextNodes == []:
            startNode = nodes[-1].entries
        else:
            for n in nextNodes:
                startNode.extend(nodes[n].entries)
        nextNodes = []
        numNodes = 0
        for entry in startNode:
            numNodes += 1
            nextNodes.append(entry[0])

            mbr = entry[1]
            areaMBR = area(mbr)
            totalArea += areaMBR
        print ("Number of nodes at level", level, " is:", numNodes)

        if numNodes > 0:
            avgArea = totalArea / numNodes
        else:
            avgArea = 0
        print("Average area of MBRs at level", level, " is:", avgArea)
        level -=1
```

Τα αποτελέσματα που τυπώνονται στο τερματικό μετά την κλήση της είναι τα εξής:

```
C:\Users\Themis Siak\Desktop\Assignments\2>python meros1.py output.txt
Height of the tree: 4
Number of nodes at level 4  is: 2
Average area of MBRs at level 4  is: 0.06273500396749851
Number of nodes at level 3  is: 37
Average area of MBRs at level 3  is: 0.008069294684621166
Number of nodes at level 2  is: 1020
Average area of MBRs at level 2  is: 0.00024823617584804087
Number of nodes at level 1  is: 51970
Average area of MBRs at level 1  is: 0.0
```

Τέλος, η συνάρτηση writeFile() παίρνει ως όρισμα το δέντρο και το αρχείο που θα το μεταφέρει. Αφού ανοίξει το αρχείο, γράφει στη πρώτη γραμμή το node-id της ρίζας του δέντρου. Στη συνέχεια, για κάθε κόμβο αποθηκεύει τις εγγραφές του, και αν ο κόμβος είναι φύλλο, γράφει το node-id του κόμβου, τον αριθμό των εγγραφών (n), τον αριθμό 0 (f) και τις εγγραφές οι οποίες αφού είναι σημεία, αποτελούνται από μια πλειάδα με x,y συντεταγμένες. Αν δεν είναι φύλλο, γράφει τα ίδια με τα παραπάνω, με τη διαφορά ότι τώρα ο αριθμός f είναι 1 και οι εγγραφές είναι 4 αριθμοί, αφού πρόκειται για MBR.

```python
def writeFile(nodes, filename):
    with open(filename, 'w') as file:
        file.write(str(nodes[-1].nodeID) + "\n")
        for node in nodes:
            entries = []
            for entry in node.entries:
                entries.append((entry[0], entry[1]))
            if node.isLeaf:
                file.write(f"{node.nodeID}, {len(entries)}, {0 if node.isLeaf else 1}, ")
                formatted_entries = [f"({ptr},({geo[0]}, {geo[1]}))" for ptr, geo in entries]
                file.write(' , '.join(formatted_entries) + "\n")
                #file.write("\n")
            else:
                file.write(f"{node.nodeID}, {len(entries)}, {0 if node.isLeaf else 1}, ")
                formatted_entries = [f"({ptr},[{geo[0]}, {geo[1]}, {geo[2]}, {geo[3]}])" for ptr, geo in entries]
                file.write(' , '.join(formatted_entries) + "\n")
                #file.write("\n")
```

Παρατηρούμε ότι τα αποτελέσματα του αρχείου που δημιουργείται ταυτίζονται με εκείνα που δόθηκαν.

## 2ο Μέρος:

Στο συγκεκριμένο πρόγραμμα έχει γίνει και import heapq, για τη χρήση heap στην best first search. Και υπάρχει και η κλάση Node από το πρώτο μέρος, αυτούσια, για τη δημιουργία του δέντρου.

Το ζητούμενο πρόγραμμα για το πρώτο μέρος ονομάζεται **meros2.py** και παίρνει σαν όρισμα το αρχείο του δέντρου από το πρώτο μέρος (txt μορφή), τις συντεταγμένες x, y του σημείου αναφοράς q και έναν αριθμό k. Για παράδειγμα:

```
>python meros2.py output.txt 39.7 116.5 3
```

Στη main συνάρτηση, αρχικά ελέγχεται αν ο αριθμός των ορισμάτων είναι ο σωστός, δηλαδή 5. Αν όχι, τυπώνεται μήνυμα λάθους και το πρόγραμμα τερματίζει. Αλλιώς, ορίζει το δέντρο, το σημείο και τον αριθμό ζητούμενων γειτόνων με τα αντίστοιχα ορίσματα. Αρχικά, δημιουργούμε μια αναπαράσταση του δέντρου στη μνήμη, όπως και στο πρώτο μέρος, με την μέθοδο loadRTree(). Τέλος, βρίσκουμε τους κοντινότερους γείτονες με την μέθοδο knnSearch() και τους τυπώνουμε με τη μέθοδο printNeighbors().

```
if __name__ == "__main__":
    if len(sys.argv) != 5:
        print("Usage: python program.py <tree_file> <q_x> <q_y> <k>")
        sys.exit(1)

    treeFile = sys.argv[1]
    q = (float(sys.argv[2]), float(sys.argv[3]))
    k = int(sys.argv[4])

    nodes = loadRTree(treeFile)

    nearestNeighbors = knnSearch(nodes, q, k)
    print("The", k+2, "Nearest Neighbors are: ")
    printNeighbors(nearestNeighbors)
```

Η μέθοδος loadRTree() παίρνει σαν όρισμα το αρχείο με το δέντρο που έχουμε φτιάξει από το πρώτο μέρος και επιστρέφει έναν πίνακα με όλους τους κόμβους του. Αφού ανοίξει το αρχείο διαβάζει την πρώτη γραμμή που είναι το ID της ρίζας. Έπειτα, διαβάζει μία μία τις γραμμές του, και συμπληρώνει τον πίνακα. Σημειώνεται το ID του κάθε κόμβου, και αν είναι φύλλο, και έτσι δημιουργείται ο νέος κόμβος. Αν είναι φύλλο, έχει ως εγγραφές σημεία, δηλαδή συντεταγμένες x, y, ενώ αν είναι ενδιάμεσος κόμβος πρόκειται για mbr, άρα έχει 4 συντεταγμένες.

```
def loadRTree(treeFile):
    nodes = []
    with open(treeFile, "r") as f:
        rootID = int(f.readline().strip())
        for line in f:
            data = line.strip().split(", ")
            nodeID = int(data[0])
            isLeaf = not bool(int(data[2]))
            newNode = Node(nodeID, isLeaf)
            for entry in data[3:]:
                if isLeaf:
                    entryData = entry.strip("()").split(",", 1)
                    recordID = int(entryData[0])
                    point = entryData[1].strip("()")
                    data = tuple(map(float, point.split(",")))
                else:
                    entryData = entry.strip("()").split(",", 1)
                    recordID = int(entryData[0])
                    mbr = entryData[1].strip("[]")
                    data = list(map(float, mbr.split(",")))
                newNode.addEntry((recordID, data))
            nodes.append(newNode)
    return nodes
```

Για τη συνάρτηση knnSearch() χρειαζόμαστε και τη βοηθητική συνάρτηση mindist(), η οποία παίρνει σαν όρισμα ένα σημείο q, και ένα mbr ή ένα σημείο. Αν είναι σημείο βρίσκει απλά την Ευκλείδεια απόσταση ως τη ρίζα του αθροίσματος των τετραγωνισμένων αποστάσεων των δύο σημείων, και την επιστρέφει. Αν είναι mbr, ελέγχουμε πρώτα αν το σημείο βρίσκεται πιο κοντά στην min ή στη max x, y συντεταγμένη κάθε φορά, και μετά υπολογίζεται η απόσταση.

```
def mindist(q, mbr):
    if len(mbr) ==2:
        return math.sqrt(sum((x1-x2)**2 for x1, x2 in zip(q, mbr)))
    qx, qy = q
    minX, minY, maxX, maxY = mbr

    if qx < minX:
        xDistance = minX - qx
    elif qx > maxX:
        xDistance = qx - maxX
    else:
        xDistance = 0

    if qy < minY:
        yDistance = minY - qy
    elif qy > maxY:
        yDistance = qy - maxY
    else:
        yDistance = 0

    return math.sqrt(xDistance**2 + yDistance**2)
```

Η συνάρτηση knnSearch() παίρνει ως ορίσματα τον πίνακα με τους κόμβους του δέντρου, το σημείο q, και τον αριθμό γειτόνων, και επιστρέφει τους k+2 πλησιέστερους γείτονες στο σημείο q. Αρχικά, δημιουργεί την ουρά βάζοντας τις εγγραφές της ρίζας του δέντρου. Το while loop παρακάτω, που περιέχει την εκτέλεση του αλγορίθμου incremental nearest neighbor που βασίζεται σε best-first search, εκτελείται έως ότου η ουρά να είναι κενή. Πρώτα, βρίσκουμε την εγγραφή από την ουρά με την μικρότερη απόσταση, και την διαγράφουμε από την ουρά. Αν δεν έχουμε επισκεφτεί τον κόμβο, τότε ελέγχουμε αν είναι φύλλο ή όχι.
Αν είναι φύλλο, προσθέτουμε σε ένα heap με τους κοντινότερους γείτονες, και αυτές τις εγγραφές που περιέχει ο κόμβος. Αλλιώς, προσθέτουμε στην ουρά τις εγγραφές του κόμβου. Αφού γίνει το παραπάνω, ελέγχουμε αν ο σωρός με τους γείτονες έχει μέγεθος πάνω από ή ίσα με k+2, ώστε να βρούμε τους κοντινότερους k+2 από αυτούς γείτονες και να τερματίσουμε τον αλγόριθμο. Άλλη μια σημαντική προϋπόθεση για να τερματίσει ο αλγόριθμος, είναι να μην υπάρχει και άλλος κοντινότερος γείτονας σε άλλον κόμβο, δηλαδή στην περίπτωση που τα φύλλα ενός κόμβου δεν είναι όλα τους οι καλύτερες επιλογές. Έτσι ελέγχουμε και αυτό, δημιουργώντας έναν προσωρινό πίνακα με τις μικρότερες αποστάσεις. Αν το μέγεθος του πίνακα γειτόνων είναι μικρότερο ή οι τιμές του δεν ταυτίζονται με αυτές του προσωρινού πίνακα, τότε συνεχίζουμε κανονικά από την αρχή τα βήματα.

```
def knnSearch(nodes, q, k):
    priorityQueue = []
    for entry in nodes[-1].entries:
        initial = mindist(q, entry[1])
        heapq.heappush(priorityQueue, (initial, entry[0]))

    visited = set()
    nearestNeighbors = []

    while priorityQueue:
        distance, nodeID = heapq.nsmallest(1,priorityQueue)[0]
        heapq.heappop(priorityQueue)

        checkMin = []

        if nodeID in visited:
            continue
        visited.add(nodeID)

        node = nodes[nodeID]
        if node.isLeaf:
            for entry in node.entries:
                heapq.heappush(nearestNeighbors, (mindist(q,entry[1]), entry[0]))
        else:
            for entry in node.entries:
                heapq.heappush(priorityQueue, (mindist(q, entry[1]), entry[0]))
                #print(entry)
        if len(nearestNeighbors) >= k+2:
            nearestN = heapq.nsmallest(k+2,nearestNeighbors)
            for n in priorityQueue:
                heapq.heappush(checkMin, n)
            for n in nearestN:
                heapq.heappush(checkMin, n)
            trueMin = heapq.nsmallest(k+2, checkMin)
            if trueMin != nearestN:
                continue
            break
    return nearestN
```

Τέλος, η συνάρτηση printNeighbors() παίρνει ως όρισμα τους γείτονες που βρήκαμε παραπάνω και τους τυπώνει στο τερματικό.

```
def printNeighbors(neighbors):
    for n in neighbors:
        print(f"({n[1]}, {n[0]})")
```

Τα αποτελέσματα που τυπώνονται στο τερματικό με q=(39.7,116.5) και k=3:

```
C:\Users\Themis Siak\Desktop\Assignments\2>python meros2.py output.txt 39.7 116.5 3
The 5 Nearest Neighbors are:
(18883, 0.004392425412003902)
(50630, 0.00955500000000051)
(8962, 0.009651267274305202)
(12977, 0.014737041833420804)
(11336, 0.016285383016681726)
```

## Πηγές:


- Διαφάνειες του μαθήματος (Χωρικά Δεδομένα, Χωρικά Δεδομένα - Σημειώσεις)
- https://medium.com/analytics-vidhya/bulk-loading-r-trees-and-how-to-store-higher-dimension-data-c7da26e4f853
- https://dl.acm.org/doi/pdf/10.1145/288692.288723
- https://apps.dtic.mil/sti/pdfs/ADA324493.pdf
- https://docs.python.org/3/library/heapq.html
- https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/
- https://www.geeksforgeeks.org/best-first-search-informed-search/