ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαχείριση Σύνθετων Δεδομένων

Assignment2

Φωτόπουλος Στέφανος ,4829

Ιωάννινα,03/05/2023

Contents

Assignment2 1
1.Περιγραφή της εργασίας 3
Μέρος 1 3
Μέρος 2 10

1.Περιγραφή της εργασίας

Μέρος 1

```python
def read_data(filename):
    with open(filename) as f:
        num_points = int(f.readline())
        print("Total Coordinates: ", num_points, "\n")
        points = []
        for i, line in enumerate(f.readlines(), start=1):
            point = [i] + list(map(float, line.strip().split()))
            points.append(point)
    return points, num_points


def sort_points_by_x(points):
    sorted_points = sorted(points, key=lambda p: p[1])  #sort by x
    return sorted_points


def sort_points_by_y(points,num_points):
    num_per_node = math.floor(1024 / 20)#51 stoixeia tha xei to kathe fyllo
    num_nodes = math.ceil(num_points / num_per_node)#Synolika 1020 fylla
    square_root = math.ceil(math.sqrt(num_nodes))
    vertical_2d_lines = square_root * num_per_node
    sorted_points = []

    #Xwrizw ton pinaka se 1632 tmimata kai ta taksinomw kata y
    for i in range(0, len(points), vertical_2d_lines):
        segment = points[i:i + vertical_2d_lines]
        sorted_segment = sorted(segment, key=lambda p: p[2])  #sort by y
        sorted_points.extend(sorted_segment)

    return sorted_points
```

Αρχικά, στην read_data διαβάζω την πρώτη γραμμή και την αποθηκεύω στην μεταβλητή num_points τυπώνοντας το σύνολο των συντεταγμένων. Αποθηκεύω σε μια λίστα τις υπόλοιπες τιμές και τις επιστρέφω.
Sort_points_by_x: Ταξινομώ την λίστα μου με τις τιμές του x και την επιστρέφω
Sort_points_by_y:Βρίσκω πόσα στοιχεία θα χει κάθε φύλλο(1024B/20B)(Παίρνω το floor γιατί θα χρησιμοποιθούν τα 1020B από τα συνολικά 1024 δεν μπορώ να υπερβώ αυτόν τον αριθμό),τα συνολικά φύλλα μου είναι 1020 και μετά χρησιμοποιώ τον τύπο του μαθήματος sqrt(N)*M όπου N ο συνολικός αριθμός των φύλλων και M των συνολικών στοιχείων του εκάστοτε φύλλου. Ταξινομώ τις συντεταγμένες μου πρώτα κατά χ και έπειτα ανά 1632 κατά y και επιστρέφω την τελική μου λίστα.

```python
def minimum_bounding_rectangle(data, flag):
    mbr = []
    if flag == 0:
        flattened_data = data

        flattened_data = [item for sublist in flattened_data for item in sublist]
    else:
        flattened_data = data
    for i in range(0, len(flattened_data)):
        min_X = min(item[1] for item in flattened_data[i])
        max_X = max(item[1] for item in flattened_data[i])
        min_Y = min(item[2] for item in flattened_data[i])
        max_Y = max(item[2] for item in flattened_data[i])
        mbr.append([min_X, min_Y, max_X, max_Y])
    return mbr
```

Στην συνάρτηση minimum_bounding_rectangle : Έχω δύο ορίσματα τα δεδομένα μου και το flag(μου δείχνει αν το καλεί κόμβος ή φύλλο). Κύρια εργασία της είναι να βλέπει κάθε γραμμή των δεδομένων και να επιστρέφει σε μια λίστα τις min/max τιμές των αξόνων x,y.

```python
def find_best_mbr(data,total_areas):
    areas_list = []
    flattened_data = data
    counter = 0
    flattened_data = [item for sublist in flattened_data for item in sublist]
    index_list = []
    mbr_list = []
    for i in range(0, len(flattened_data)):
        counter = 0
        for item in flattened_data[i][0]:
            min_X = item[0]
            min_Y = item[1]
            max_X = item[2]
            max_Y = item[3]
            width = max_X - min_X
            height = max_Y - min_Y
            temp_area = width * height
            total_areas.append(temp_area)
            if counter == 0:
                areas_list.append(temp_area)
                index_list.append(counter)
            if temp_area > areas_list[i]:
                areas_list.pop(i)
                index_list.pop(i)
                areas_list.append(temp_area)
                index_list.append(counter)
            counter += 1

    for i in range(0, len(index_list)):
        index = index_list[i]
        mbr_list.append(flattened_data[i][0][index])
    return mbr_list

```

Η find_best_mbr βρίσκει το καλύτερο mbr μέσω του εμβαδού για κάθε κόμβο και τα επιστρέφει σε μια λίστα. Κρατάω σε μια index_list τα Indexes με τα καλύτερα εμβαδά έτσι ώστε στο τελεύταιο for loop να ανατρέξω στην αρχική μου λίστα και να πάρω μόνο

τα mbrs που με ενδιαφέρουν από κάθε κόμβο. Χρησιμοποιώ τον counter για να με βοηθήσει να γνωρίζω σε ποια θέση του αρχικού μου πίνακα βρίσκεται το καλύτερο Mbr,στο πρώτο run θεωρώ ως καλύτερο εμβαδό αυτό της πρώτης θέσης και στην συνέχεια βλέπω αν τα επόμενα είναι μεγαλύτερα από αυτό που ήδη έχω σε περίπτωση που ισχύει κάνω pop την προηγούμενη τιμή και εισάγω την νέα. Τέλος, κρατάω σε μια λίστα όλες τις τιμές των εμβαδών των κόμβων(total_areas).

```
89  def average_mbr_area(nodes, num_nodes, num_of_last_nodes):
90      total_area = 0
91      for i in range(0, len(nodes)):
92          total_area += nodes[i]
93      total_nodes = (num_nodes-1) * 28 + num_of_last_nodes
94      result = total_area / total_nodes
95      return result
96
```

Average_mbr_area: Στην λίστα nodes έχω όλες τις τιμές των εμβαδών από το επίπεδο στο οποίο βρίσκομαι,num_nodes είναι ο αριθμός των συνολικών κόμβων σε αυτό το επίπεδο, num_of_last_nodes είναι ο αριθμός των στοιχείων του τελευταίου κόμβου. Για την εύρεση του average_mbr προσθέτω όλα τα εμβαδά και διαιρώ τους συνολικούς κόμβους μείον έναν(που είναι ο τελευταίος) επί 28 αφού έχει 28 στοιχεία ο κάθε ολοκληρωμένος κόμβος συν τα στοιχεία του τελευταίου.

```python
def build_leaf(coordinates, num_points, level):
    num_per_node = math.floor(1024 / 20)
    num_nodes = math.ceil(num_points / num_per_node)
    leafs = []
    id = 0
    def sort_tile_recursive(coords, node_id):
        nonlocal id
        if not coords:
            return []
        leafs_for_node = []
        while coords:
            tile = coords[:num_per_node]
            sorted_coords_y = sorted(tile, key=lambda x: x[2])

            leaf_bytes = 0
            temp_leaf = []
            for coord in sorted_coords_y:
                if leaf_bytes + 20 > 1024:
                    leafs_for_node.append(temp_leaf)
                    leaf_bytes = 0
                    temp_leaf = []
                temp_leaf.append(coord)
                leaf_bytes += 20

            if temp_leaf:
                leafs_for_node.append(temp_leaf)
                temp_leaf = []
            coords = coords[num_per_node:]
        id = node_id + 1
        return leafs_for_node
    leafs = sort_tile_recursive(coordinates, id)
    level += 1
    return leafs
```

build_leaf: λαμβάνει ως είσοδο μια λίστα coordinates που περιέχει συντεταγμένες σημείων, τον αριθμό των σημείων num_points και το επίπεδο του κόμβου level.Υπολογίζει τον αριθμό των κόμβων που θα δημιουργηθούν, με βάση τον μέγιστο αριθμό σημείων που μπορεί να περιέχει ένας κόμβος. Η εσωτερική συνάρτηση sort_tile_recursive, η οποία ταξινομεί τις συντεταγμένες των σημείων. Ο στόχος είναι να οργανωθούν τα σημεία σε τετράγωνα (tiles) με βάση τον άξονα y.Αρχικά, δημιουργούνται tiles μεγέθους num_per_node (στην περίπτωσή μας 20) και ταξινομούνται βάσει του y.Έπειτα, τα tiles που δημιουργούνται συσσωρεύονται σε μια λίστα leafs_for_node.Η διαδικασία αυτή επαναλαμβάνεται για όλα τα tiles που περιέχονται στη λίστα coordinates.Τέλος, καλείται η sort_tile_recursive με τις συντεταγμένες coordinates και επιστρέφει τη λίστα των tiles leafs_for_node, η οποία αποθηκεύεται στη μεταβλητή leafs. Το επίπεδο level αυξάνεται κατά 1 και επιστρέφεται η λίστα των leafs.

```python
def build_rtree(leafs_list,R_tree,total_areas,temp_mbr,average_list,flag,num_points):
    total_bytes = 1024
    num_per_node = math.floor(total_bytes / 20)  # ta shmeia poy xwraei kathe fyllo (51)
    num_nodes = math.ceil(num_points / num_per_node)  # synolika fylla (1020)
    leafs_per_node = math.floor(total_bytes / 36)  # fylla ana komvo (28)
    total_nodes = math.ceil(num_nodes / leafs_per_node)  # synolo komvwn (37)
    mbr_bytes = 32 + 4
    original_list = leafs_list
    first_len = len(leafs_list)
    average_list.append(first_len)
    leafs = leafs_list
    mbr = []
    counter_leaf = 0
    counter_node = 0
    node_id = 0
    last_mbr = 1


    if len(leafs) == num_nodes:
        flag = 0
        for i in range(1, len(leafs)+1):
            if i % leafs_per_node == 0 or i == num_nodes:
                mbr = []

                if counter_node + mbr_bytes > total_bytes:
                    node_coords = []
                    node_coords.append(original_list[counter_node:first_len])
                    mbr.append(minimum_bounding_rectangle(node_coords, flag))
                    node_id += 1
                    leafs.append(mbr)
                    R_tree.append(mbr)
                    temp_mbr.append(len(node_coords[0]))
                    counter_node += leafs_per_node
                else:
                    node_coords = []
                    node_coords.append(leafs[counter_node:counter_node + leafs_per_node])
                    mbr.append(minimum_bounding_rectangle(node_coords, flag))
                    last_mbr += 1
                    node_id += 1
                    leafs.append(mbr)
                    R_tree.append(mbr)

                    counter_node += leafs_per_node
                counter_leaf += num_per_node
    else:
        flag = 1
        for i in range(1,len(leafs_list)+1):
            if i % leafs_per_node == 0 or i == first_len:
                mbr = []
                if i == first_len:
                    node_coords = []
                    node_coords.append(original_list[counter_node:first_len])
                    mbr_list = find_best_mbr(node_coords,total_areas)
                    mbr.append(mbr_list)
                    last_mbr += 1
                    node_id += 1
                    leafs.append(mbr)
                    if first_len!=1:
                        R_tree.append(mbr)
                    temp_mbr.append(len(node_coords[0]))
                    counter_node += leafs_per_node
                else:
                    node_coords = []
                    node_coords.append(original_list[counter_node:counter_node + leafs_per_node])
                    mbr_list = find_best_mbr(node_coords,total_areas)
                    mbr.append(mbr_list)
                    last_mbr += 1
                    node_id += 1
                    leafs.append(mbr)
                    R_tree.append(mbr)
                    counter_node += leafs_per_node
                counter_leaf += num_per_node
    if flag == 1:
```

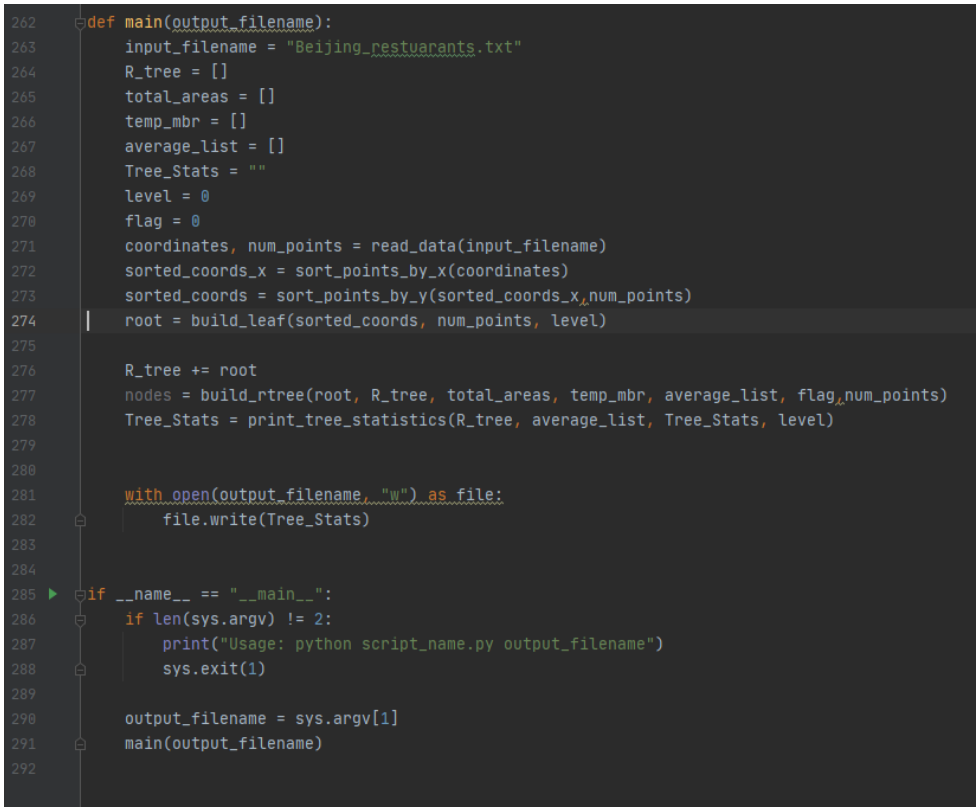
build_rtree:Όπως και στις προηγούμενες συναρτήσεις αρχικοποιώ τις μεταβλητές που χρειάζομαι μπορούμε να πούμε ότι χωρίζεται σε δυο μέρη το ένα είναι όταν παίρνει όρισμα τα φύλλα του δέντρου και μετά είναι οι υπόλοιποι κόμβοι.Για την πρώτη περίπτωση διατρέχω όλη την λίστα και ανά 28 στοιχεία βρίσκω το mbr κάθε γραμμής και τα αποτελέσματα προστίθενται στο R_tree.Όταν βρίσκομαι στην δεύτερη περίπτωση θέτω το flag 1 βρίσκω το καλύτερο mbr κάθε κόμβου και το εισάγω στο R_tree.Για τις τελευταίους κόμβους τόσο στα φύλλα όσο και στους εσωτερικούς κόμβους η λογική είναι η ίδια απλά κρατάω και τον αριθμό των συνολικών στοιχείων των φύλλων σε μια λίστα temp_mbr ομοίως και για τους εσωτερικούς κόμβους.

```
    if flag == 1:
        num_of_last_nodes = temp_mbr[0]
        avg_mbr = average_mbr_area(total_areas, first_len, num_of_last_nodes)
        temp_mbr.pop(0)
        average_list.append(avg_mbr)
        total_areas = []
    if first_len == 1:
        return R_tree

    last_level_list = leafs[first_len:]

    if first_len != 1:
        build_rtree(last_level_list,R_tree,total_areas,temp_mbr,average_list,flag,num_points)
```

Όταν η τιμή flag γίνει 1 σημαίνει ότι έχω προχωρήσει από τα φύλλα και βρίσκομαι σε κάποιον εσωτερικό κόμβο παίρνω την πρώτη τιμή της λίστας temp_mbr η οποία είναι το πλήθος στοιχείων του τελεύταιου κόμβου αυτού του επιπέδου, αυτός ο αριθμός θα με βοηθήσει στον υπολογισμό του average_mbr.Ελέγχω εάν βρίσκομαι στην ρίζα του δέντρου και αν όχι ξανακαλώ την build_rtree με καινούρια λίστα αρχίζοντας πλεόν από το σημείο που βρίσκεται ο τελευταίος κόμβος του προηγούμενου επιπέδου.

```
def print_tree_statistics(R_tree, avg_list, Tree_Stats, level):
    Stats = ""
    Tree_Stats += "Root id is: " + str(len(R_tree)-1) + "\n"
    Tree_Stats += "                    -------- Leaf Node --------\n"
    indices_to_move = [0, 1, 3, 5]
    num_of_nodes = [avg_list.pop(i) for i in reversed(indices_to_move)]
    for i in range(0,num_of_nodes[-1]):
        Stats = str(i) + ", " + str(len(R_tree[i])) + ", " + str(0)
        coords = ""
        for j in range(0,len(R_tree[i])):
            coords += ", "+ "(" + str(R_tree[i][j][0]) + ", " + "(" + str(R_tree[i][j][1]) + ", " + str(R_tree[i][j][2]) + ")" + ")"
        Tree_Stats += Stats + coords + "\n"

    print("Total number of leafs: ",num_of_nodes[-1]," found in level",level)
    print("Average MBR for level ", level, "is: 0.0")
    first_len_of_avg_list = len(avg_list)
    last_num = num_of_nodes[-1]
    num_of_nodes.pop(-1)
    level += 1
    record_id = 0
    while len(num_of_nodes) != 0:
        Tree_Stats += "                    -------- Inner Node at Level " + str(level) + " --------\n"
        for i in range(last_num, last_num + num_of_nodes[-1]):

            Stats = str(i) + ", " + str(len(R_tree[i][0])) + ", " + str(1)
            coords = ""
            for j in range(0, len(R_tree[i][0])):
                coords += ", " + "(" + str(record_id) + ", " + "(" + str(R_tree[i][0][j][0]) + ", " + str(R_tree[i][0][j][1]) + ", " + str(R_tree[i][0][j][2]) + ", " + str(R_tree[i][0][j][
                record_id += 1
            Tree_Stats += Stats + coords + "\n"
        print("Total number of internal nodes: ", num_of_nodes[-1], " found in level", level)
        print("Average MBR for level ", level, "is: ",avg_list[0])
        avg_list.pop(0)
        level += 1
        last_num += num_of_nodes[-1]
        num_of_nodes.pop(-1)
    print("Height of R_Tree is: ",first_len_of_avg_list + 1)

    return Tree_Stats
```

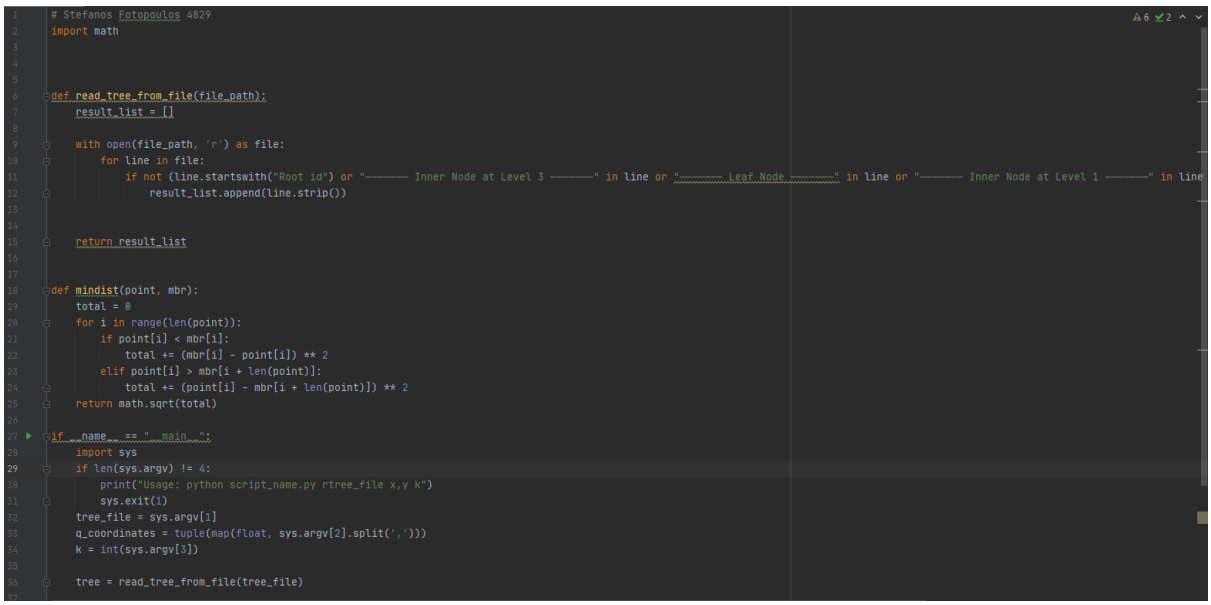
Print_tree_statistics: Παίρνει ως ορίσματα το R_tree που είναι η λίστα με όλες τις τιμές που θέλουμε,το avg_list έχει μέσα το πλήθος κόμβων κάθε επιπέδου όπως και τα average_mbr του κάθε επιπέδου.Στην πρώτη γραμμή του αρχείου τυπώνω το Id της ρίζας και μετά το Leaf node με όλα τα φύλλα, στη συνέχεια όλους τους εσωτερικούς κόμβους ανά επίπεδο μαζί με το averare mbr του κάθε επιπέδου.Τέλος, τυπώνω το ύψος του δέντρου που είναι οι τιμές του averare mbr που έχω για κάθε επίπεδο προσθέτοντας ένα γιατί στο επίπεδο των φύλλων το average mbr είναι 0 οπότε δεν έχω κάποια τιμή.



```python
def main(output_filename):
    input_filename = "Beijing_restuarants.txt"
    R_tree = []
    total_areas = []
    temp_mbr = []
    average_list = []
    Tree_Stats = ""
    level = 0
    flag = 0
    coordinates, num_points = read_data(input_filename)
    sorted_coords_x = sort_points_by_x(coordinates)
    sorted_coords = sort_points_by_y(sorted_coords_x,num_points)
    root = build_leaf(sorted_coords, num_points, level)

    R_tree += root
    nodes = build_rtree(root, R_tree, total_areas, temp_mbr, average_list, flag,num_points)
    Tree_Stats = print_tree_statistics(R_tree, average_list, Tree_Stats, level)


    with open(output_filename, "w") as file:
        file.write(Tree_Stats)


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python script_name.py output_filename")
        sys.exit(1)

    output_filename = sys.argv[1]
    main(output_filename)
```

Έχω δημιουργήσει μια συνάρτηση Main με όρισμα το output αρχείο ουσιαστικά μέσα σε αυτήν αρχικοποίησα όλες τις global μεταβλητές που χρησιμοποιώ στις παραπάνω συναρτήσεις.Η διαδικασία της Main είναι να διαβάσει το αρχείο,να κάνει sort κατά χ και μετά κατά y,να δημιουργήσει τα φύλλα και έπειτα τους εσωτερικούς κόμβους που τα δεδομένα αυτά θα εισαχθούν στο R_tree,με το R_tree θα παράξει τα στατιστικά του δέντρου, τα οποία θα τα εισάγει στο Output αρχείο.
Τέλος,η κανονική main του προγράμματος ελέγχει κατά πόσο ο χρήστης έχει καλέσει σωστά το πρόγραμμα στο τερματικό και καλεί την συνάρτηση main.

Μέρος 2



```python
# Stefanos Fotopoulos 4829
import math



def read_tree_from_file(file_path):
    result_list = []

    with open(file_path, 'r') as file:
        for line in file:
            if not (line.startswith("Root id") or "-------- Inner Node at Level 3 --------" in line or "-------- Leaf Node --------" in line or "-------- Inner Node at Level 1 --------" in line
                result_list.append(line.strip())


    return result_list


def mindist(point, mbr):
    total = 0
    for i in range(len(point)):
        if point[i] < mbr[i]:
            total += (mbr[i] - point[i]) ** 2
        elif point[i] > mbr[i + len(point)]:
            total += (point[i] - mbr[i + len(point)]) ** 2
    return math.sqrt(total)

if __name__ == "__main__":
    import sys
    if len(sys.argv) != 4:
        print("Usage: python script_name.py rtree_file x,y k")
        sys.exit(1)
    tree_file = sys.argv[1]
    q_coordinates = tuple(map(float, sys.argv[2].split(',')))
    k = int(sys.argv[3])

    tree = read_tree_from_file(tree_file)
```

Στο μέρος 2 έχω δυο συναρτήσεις την read_tree_from_file στην οποία εισάγω το αρχείο που έχει το δέντρο και μου επιστρέφει μια λίστα με tuples χωρίς τις περιττές γραμμές που έχω βάλει για να ξεχωρίζω κάθε επίπεδο.
Την Mindist η οποία παίρνει ένα σημείο(x,y) και το mbr ελέγχει κατά πόσο η τιμή του σημείου είναι μικρότερη από το Minimum mbr ή μεγαλύτερη από το maximum mbr και επιστρέφει την ρίζα της πρόσθεσης αυτών των δυο τετραγώνων(x και y).
Δυστυχώς δεν μπόρεσα να καταφέρω να εισάγω τα mbr από την λίστα μου όποτε το πρόγραμμα μου αυτή την στιγμή επιστρέφει σε μια τιμή tree αυτή την λίστα με τα tuples.