Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Προπτυχιακό Μάθημα: «Διαχείριση Σύνθετων Δεδομένων»

Assignment 2

Vasilis Nikas **AM: 3143**

Οδηγίες εκτέλεσης:

Αρχικά πρέπει να τρέξουμε μία φορά το μέρος 1 (Part1.py) για να δημιουργήσουμε το grid και να παράγουμε τα κατάλληλα αρχεία. Οπότε τρέχουμε στο terminal την εντολή:

python3 Part1.py

Στην συνέχεια μπορούμε να τρέξουμε όποιο απο τα Part2 και Part3 θέλουμε.

Στην αρχή κάθε αρχείου κώδικα υπάρχουν τα paths για τα αρχεία που θέλουμε να διαβάσουμε.

Μέρος 1:

Αρχικά διαβάζουμε το csv αρχείο και αγνοούμε την πρώτη γραμμή που περιέχει το πλήθος των τεθλασμένω γραμμών. Έπειτα για κάθε γραμμή του αρχείου διαβάζουμε τις συντεταγμένες και αποθηκεύουμε σε μεταβλητές τις ελάχιστες τιμές για το x και το y και μετά τις προσθέτουμε σε μια λίστα. Για κάθε γραμμή αποθηκεύουμε στην δομή μας πρώτα το ID που είναι απλά ένας αριθμός ξεκινόντας από το 1. Στην συνέχεια αποθηκεύουμε τις ελάχιστες τίμες των x και y κάθε linestring έτσι ώστε να γνωρίζουμε το mbr του. Και τέλος αποθηκεύουμε μια λίστα με όλα τα σημεία των lines. Επιπλέον αποθηκεύουμε την ελάχιστη και μέγιστη τιμή των x και y όλων των linestrings.

Ο κώδικας είναι ο εξής:

```
with open(filePath, 'r') as f:
    records = []
    idCounter = 1
    maxX = -9999
    minY = 9999
    maxY = minX = 0

    for line in f:
        break
    for line in f:
        linestring = []
        mbrMinX = mbrMaxY = 0
        mbrMaxX = -9999
```

```
        mbrMinY = 9999
        mbrMins = []
        mbrMaxs = []
        mbr = []
        coordinates = line.split(',')
        for x in range(len(coordinates)):
            coordinate = coordinates[x].split(' ')
            coordinate[0] = float(coordinate[0])
            coordinate[1] = float(coordinate[1])
            linestring.append(coordinate)
            if mbrMinX > coordinate[0]:
                mbrMinX = coordinate[0]
                if minX > mbrMinX:
                    minX = mbrMinX
            if mbrMaxX < coordinate[0]:
                mbrMaxX = coordinate[0]
                if maxX < mbrMaxX:
                    maxX = mbrMaxX
            if mbrMinY > coordinate[1]:
                mbrMinY = coordinate[1]
                if minY > mbrMinY:
                    minY = mbrMinY
            if mbrMaxY < coordinate[1]:
                mbrMaxY = coordinate[1]
                if maxY < mbrMaxY:
                    maxY = mbrMaxY
        # Appending in a list the current mbrs max and min
        mbrMins.append(mbrMinX)
        mbrMins.append(mbrMinY)
        mbrMaxs.append(mbrMaxX)
        mbrMaxs.append(mbrMaxY)
        mbr.append(mbrMins)
        mbr.append(mbrMaxs)
        # Appending in the records list each record
        records.append(idCounter)
        records.append(mbr)
        records.append(linestring)
        idCounter += 1
```

Έχοντας στην δομή μας όλες τις πληροφορίες που χρειαζόμαστε από το αρχείο, ξεκινάμε να φτιάχνουμε το grid. Γνωρίζουμε ότι πρέπει να χωρίσουμε το χώρο σε 10*10 ισομεγέθη κελιά. Γνωρίζοντας επίσης την μέγιστη και ελάχιστη τιμή των x και y μπορούμε να υπολογίσουμε για κάθε άξονα τις σωστές τιμές έτσι ώστε να δημιουργηθούν τα κελιά.

Η συνάρτηση είναι η παρακάτω:

```
def calculateGridAxis(min, max):
    cellsCoordinates = []

    for x in range(11):
        cellsCoordinates.append(min + ((x*(max - min))/10))
    return cellsCoordinates
```

Στην συνέχεια υπολογίζουμε για κάθε mbr που έχουμε στην δομή μας σε ποια κελιά πέφτει (έχει επικάλυψει) δηλαδή το mbr τέμνει το κελί. Φτιάχνουμε μια δομή (dictionary), της οποίας τα keys είναι τα κελία (0,0..0,1…) και τα values είναι μια λίστα με το ID των mbr που τέμνοντε με αυτό το κελί. Ένα mbr μπορεί να υπάρχει σε πολλά κελιά.

Ο κώδικας για τον έλεγχο και την δημιουργία του dictionary:

```
def fillingCells(records, gridXAxisCoordinates, gridYAxisCoordinates):
    cellsValues = {}
    tempCell = []
    currentMbrMaxX = []
    currentMbrMaxY = []
    currentMbrMinX = []
    currentMbrMinY = []

    minmaxmbr = records[1::3]
    for x in range(len(minmaxmbr)):
        currentMbrMaxX.append(minmaxmbr[x][1][0])
        currentMbrMaxY.append(minmaxmbr[x][1][1])
        currentMbrMinX.append(minmaxmbr[x][0][0])
        currentMbrMinY.append(minmaxmbr[x][0][1])
    for i in range(10):
        for j in range(10):
            cellsValues[str(i)+str(j)] = []

    for x in range(len(minmaxmbr)):
        # These are for cells only (0,0)
        miny = minx = maxy = maxx = 0
        flag = flag1 = flag2 = flag3 = 0
        for j in range(11):
            if (currentMbrMinX[x] <= gridXAxisCoordinates[j] and flag ==
0):
                if j == 0:
                    minx = 0
                else:
                    minx = j - 1
                flag = 1
            if (currentMbrMinY[x] <= gridYAxisCoordinates[j] and flag1 ==
0):
                if j == 0:
                    miny = 0
                else:
                    miny = j - 1
                flag1 = 1
            if (currentMbrMaxX[x] <= gridXAxisCoordinates[j] and flag2 ==
0):
                maxx = j - 1
                flag2 = 1
            if (currentMbrMaxY[x] <= gridYAxisCoordinates[j] and flag3 ==
0):
                maxy = j - 1
                flag3 = 1
        xrange = maxx - minx
        yrange = maxy - miny
        for i in range(xrange+1):
            for j in range(yrange+1):
                cellsValues[str(minx+i) + str(miny+j)].append(x+1)
    return cellsValues
```

Τελευταία υλοποιήση του πρώτου μέρους είναι η δημιουργία δύο αρχείων που περιέρχουν αύτες τις πληροφορίες δηλαδή περιέχουν το grid. Οι πολλές επαναλήψεις που χρειάζονται για να αποθηκεύσουμε τις πληροφορίες στα αρχεία είναι πάρα πολλές οπότε χρησιμοποιούμε dictionaries για γρηγορή προσπέλαση στις δομές μας.

Ο κώδικας για την δημιουργία των αρχείων είναι ο εξής:

```
def makeFile(cellsValues, records, minX, minY, maxX, maxY):
    grd = open("grid.grd", 'w')
    grid = open("grid.dir", "w")
    ids = records[0::3]
    #minmaxmbr = records[1::3]
    #xy = records[2::3]
    grid.write(str(1) + " " + str(minX) + " " + str(maxX) + " " + str(minY)
+ " " + str(maxY) + "\n")
    counter = 2
    overallList = []
    idsDict = dict(zip(ids, range(0, len(ids))))

    for x in cellsValues:
        templist = cellsValues[x]
        overallList.extend(templist)
        grid.write(str(counter) + " " + str(x[0]) + " " + str(x[1]) + " " +
str(len(cellsValues[x])) + "\n")
        counter += 1

    #overallList = tuple(overallList)
    for i in overallList:
        target = findLowestLimit(ids,len(ids),i)
        if (i == target):
            j = idsDict[i]
            temp = records[j*3+2][:]
            #grd.write(str(ids[j]) + "," + str(minmaxmbr[j][0][0]) + " " +
str(minmaxmbr[j][0][1]) + "," + str(minmaxmbr[j][1][0]) + " " +
str(minmaxmbr[j][1][1]) + ",") #+ str(xy[j][:][:][0]) + "\n")
            grd.write(str(ids[j]) + "," + str(records[j*3+1][0][0]) + " " +
str(records[j*3+1][0][1]) + "," + str(records[j*3+1][1][0]) + " " +
str(records[j*3+1][1][1]) + ",")
            for v in temp:
                grd.write(str(v[0]) + " " + str(v[1]) + ",")
            grd.write("\n")
    grd.close()
    grid.close()
```

Μέρος 2:

Έχοντας κάνει το preprocessing στο μέρος 1, τώρα διαβάζουμε τα αρχεία του grid και φτιάχνουμε τις δομές στον κωδικά μας με τις πληροφορίες των αρχείων έτσι ώστε μέτα να μπορέσουμε να κάνουμε τα ερωτήματα. Για τις πληροφορίες του dir αρχείου φτιάχνουμε δύο λίστες και αποθηκεύουμε τα κελιά και το πλήθος των mbr του κάθε κιελιού.

```
# Store the coordinates of each Cell and store the multitude of each cell
for line in dirFile:
    values = line.split(" ")
```

```
    coordinatesOfCells.append(str(values[1]) + str(values[2]))
    setOfEachCell.append(int(values[3]))
```

Αντίστοιχα δημιουργούμε ξανά μια λίστα απο τριπλέτες και το λεξικό μας όπως στο μέρος 1, για να αποθηκεύσουμε τις πληροφορίες του grd αρχείου εχόντας υπόψην και τις πληροφορίες απο το dir αρχείο.

```
for line in dirFile:
    values = line.split(" ")
    coordinatesOfCells.append(str(values[1]) + str(values[2]))
    setOfEachCell.append(int(values[3]))
# Read grd file
for eachRecord in grdFile:
    mbrCoordinates = []
    linestring = []
    recordValues = eachRecord.split(",")
    # Storing first the ID
    records.append(int(recordValues[0]))
    # Storing the list of mins and maxs coordinates of mbr
    minsmbr = recordValues[1].split(" ")
    minsmbr[0] = float(minsmbr[0])
    minsmbr[1] = float(minsmbr[1])
    mbrCoordinates.append(minsmbr)
    maxsmbr = recordValues[2].split(" ")
    maxsmbr[0] = float(maxsmbr[0])
    maxsmbr[1] = float(maxsmbr[1])
    mbrCoordinates.append(maxsmbr)
    records.append(mbrCoordinates)
    for x in range(len(recordValues)-1):
        if (x > 2):
            lines = recordValues[x].split(" ")
            lines[0] = float(lines[0])
            lines[1] = float(lines[1])
            linestring.append(lines)
    records.append(linestring)
# Making the dictionary for the cells
counter = 0
for x in range(len(setOfEachCell)):
    cellValues[coordinatesOfCells[x]] = []
    y = setOfEachCell[x]
    if (setOfEachCell[x] > 0):
        while (y > 0):
            cellValues[coordinatesOfCells[x]].append(records[counter*3])
            counter += 1
            y -= 1
```

Έπειτα διαβάζουμε το αρχείο με τα queries. Για κάθε query αποθηκεύουμε τον αριθμό του οπώς επίσης και τις συντεταγμένες για το παράθυρο.

```
for line in queryFile:
    values = line.split(",")
    numberOfQuery = int(values[0])
    coord = values[1].split(" ")
```

```
    minXWindow = float(coord[0])
    maxXWindow = float(coord[1])
    minYWindow = float(coord[2])
    maxYWindow = float(coord[3])
    windowCoordinates.append(minXWindow)
    windowCoordinates.append(maxXWindow)
    windowCoordinates.append(minYWindow)
    windowCoordinates.append(maxYWindow)
```

Στην συνέχεια ελέγχουμε σε ποια κελιά ‘πέφτει’ το παράθυρο του query έτσι ώστε να ελέγξουμε τις τιμές των mbr που πέφτουν μόνο σε αυτά τα κελιά και να μην κανούμε άσκοπους υπολογισμούς για άλλα mbr που ξέρουμε σίγουρα με αύτον τον τρόπο ότι δεν ύπαρχει σημείο τομής.

```
def checkCellsForWindow(windowCoordinates, gridXAxisCoordinates,
gridYAxisCoordinates):
    windowCells = []
    miny = minx = maxy = maxx = 0
    flag = flag1 = flag2 = flag3 = 0
    for j in range(len(gridXAxisCoordinates)):
        if (windowCoordinates[0] <= gridXAxisCoordinates[j] and flag == 0):
            if j == 0:
                minx = 0
            else:
                minx = j - 1
            flag = 1
        if (windowCoordinates[2] <= gridYAxisCoordinates[j] and flag1 ==
0):
            if j == 0:
                miny = 0
            else:
                miny = j - 1
            flag1 = 1
        if (windowCoordinates[1] <= gridXAxisCoordinates[j] and flag2 ==
0):
            maxx = j - 1
            flag2 = 1
        if (windowCoordinates[3] <= gridYAxisCoordinates[j] and flag3 ==
0):
            maxy = j - 1
            flag3 = 1
    xrange = maxx - minx
    yrange = maxy - miny
    for i in range(xrange + 1):
        for j in range(yrange + 1):
            windowCells.append(str(minx + i) + str(miny + j))
    return windowCells
```

Ξέροντας πλέον και ποιά mbr πρέπει να ελέγξουμε φτιάχνουμε την συνάρτηση findIntersection για να το κάνουμε. Για κάθε κελί που πρέπει να ελέγξουμε, παίρνουμε ένα-ένα τα mbr του κελιού και ελέγχουμε αν υπάρχει έστω και ένα σημείο τομής με το παράθυρο μας. Αν υπάρχει τότε το προσθέτουμε στα αποτελέσματα μας. Αν δεν υπάρχει

συνεχίζουμε στο επόμενο mbr. Τέλος τυπώνουμε στην οθόνη τα mbr τα οποία έχουν έστω και ένα κοινο σημείο με το παράθυρο.

```
def findIntersection(windowCells, windowCoordinates, cellValues, records,
gridXAxisCoordinates, gridYAxisCoordinates):
    referencePoint = []
    mbrsForComparison = []
    intersectMbrs = []
    emptyCellsCounter = 0

    ids = records[0::3]
    idsDict = dict(zip(ids, range(0, len(ids))))

    for i in windowCells:
        cells = [int(x) for x in i]
        #print(i)
        if (cellValues[i] == []):
            emptyCellsCounter += 1
        if (i in cellValues):
            mbrsForComparison.extend(cellValues[i])

        for mbrsId in mbrsForComparison:
            mbrCoordinates = []
            #mbrCoordinates = findMbr(mbrsId, records)
            indexer = idsDict[mbrsId]
            mbrCoordinates.append(records[indexer * 3 + 1][0][0])
            mbrCoordinates.append(records[indexer * 3 + 1][1][0])
            mbrCoordinates.append(records[indexer * 3 + 1][0][1])
            mbrCoordinates.append(records[indexer * 3 + 1][1][1])

            if (windowCoordinates[0] < mbrCoordinates[0]):
                referencePoint.append(mbrCoordinates[0])
            else:
                referencePoint.append(windowCoordinates[0])
            if (windowCoordinates[2] < mbrCoordinates[2]):
                referencePoint.append(mbrCoordinates[2])
            else:
                referencePoint.append(windowCoordinates[2])

            if (windowCoordinates[0] <= mbrCoordinates[1] and
windowCoordinates[2] <= mbrCoordinates[3] and windowCoordinates[1] >=
mbrCoordinates[0] and windowCoordinates[3] >= mbrCoordinates[2]):
                if ((referencePoint[0] <= gridXAxisCoordinates[cells[0]+1]
and referencePoint[0] >= gridXAxisCoordinates[cells[0]]) and
(referencePoint[1] <= gridYAxisCoordinates[cells[1]+1] and
referencePoint[1] >= gridYAxisCoordinates[cells[1]])):
                    #if (mbrsId not in intersectMbrs):
                    intersectMbrs.append(mbrsId)
            referencePoint = []
        mbrsForComparison = []
    for i in intersectMbrs:
        print(i , end = " ")
    print()
    print("Cells: " + str(len(windowCells) - emptyCellsCounter))
    print("Results: " + str(len(intersectMbrs)))
    print("----------")
    return intersectMbrs
```

Μέρος 3:

Το μέρος 3 είναι ακριβώς ίδιο με το μέρος 2 μόνο που τώρα το επεκτείνουμε έτσι ώστε να ελέγχουμε κάθε linestring και να βρούμε αν καποιό από αυτά επικαλύπτεται με το παράθυρο. Ξέροντας τα mbr που εχόυν έστω και ένα κοινό σημείο με το παράθυρο τώρα ελέξγουμε τα linestrings αυτών των mbr διοτί μπορεί κάποιο mbr να εχεί κοινό σημείο με το παράθυρο αλλά όχι απαραίτητα τα linestrings να έχουν όντως κοινό στοιχείο.

```
def findRefinementStage(intersectMbrs, windowCoordinates, records,
emptyCells):
    refinementMbrs = []
    ids = records[0::3]
    idsDict = dict(zip(ids, range(0, len(ids))))
    for i in range(len(intersectMbrs)):
        #mbrCoordinates = findMbr(intersectMbrs[i], records)
        mbrCoordinates = []
        # mbrCoordinates = findMbr(mbrsId, records)
        indexer = idsDict[intersectMbrs[i]]
        mbrCoordinates.append(records[indexer * 3 + 1][0][0])
        mbrCoordinates.append(records[indexer * 3 + 1][1][0])
        mbrCoordinates.append(records[indexer * 3 + 1][0][1])
        mbrCoordinates.append(records[indexer * 3 + 1][1][1])

        if (mbrCoordinates[0] >= windowCoordinates[0] and mbrCoordinates[1]
<= windowCoordinates[1]):
            if (mbrCoordinates[3] >= windowCoordinates[2] and
mbrCoordinates[2] <= windowCoordinates[3]):
                if (intersectMbrs[i] not in refinementMbrs):
                    refinementMbrs.append(intersectMbrs[i])
                #refinementMbrs.append(intersectMbrs[i])
        elif (mbrCoordinates[2] >= windowCoordinates[2] and
mbrCoordinates[3] <= windowCoordinates[3]):
            if (mbrCoordinates[2] >= windowCoordinates[0] and
mbrCoordinates[0] <= windowCoordinates[1]):
                if (intersectMbrs[i] not in refinementMbrs):
                    refinementMbrs.append(intersectMbrs[i])
                #refinementMbrs.append(intersectMbrs[i])
        else:
            #lineString = findLineString(intersectMbrs[i], records)
            indexer = idsDict[intersectMbrs[i]]
            lineString = records[indexer*3 + 2]
            for line in range(len(lineString)-1):
                x1 = lineString[line][0]
                x2 = lineString[line + 1][0]
                y1 = lineString[line][1]
                y2 = lineString[line + 1][1]
                for sides in range(4):
                    if (sides == 0):
                        x3 = windowCoordinates[0]
                        x4 = windowCoordinates[1]
                        y3 = windowCoordinates[2]
```

```
                        y4 = windowCoordinates[2]
                    elif (sides == 1):
                        x3 = windowCoordinates[0]
                        x4 = windowCoordinates[1]
                        y3 = windowCoordinates[3]
                        y4 = windowCoordinates[3]
                    elif (sides == 2):
                        x3 = windowCoordinates[0]
                        x4 = windowCoordinates[0]
                        y3 = windowCoordinates[2]
                        y4 = windowCoordinates[3]
                    else:
                        x3 = windowCoordinates[1]
                        x4 = windowCoordinates[1]
                        y3 = windowCoordinates[2]
                        y4 = windowCoordinates[3]
                    # For some reason python thinks this is equal to 0 lol
                    if ((((x1-x2)*(y3-y4))-((y1-y2)*(x3-x4))) != 0):
                        t = (((x1-x3)*(y3-y4))-((y1-y3)*(x3-x4)))/(((x1-
x2)*(y3-y4))-((y1-y2)*(x3-x4)))
                    if ((((x1 - x2) * (y3 - y4)) - ((y1 - y2) * (x3 - x4)))
!= 0):
                        u = (((x1 - x3) * (y1 - y2)) - ((y1 - y3) * (x1 -
x2))) / (((x1 - x2) * (y3 - y4)) - ((y1 - y2) * (x3 - x4)))
                    if (t >= 0 and t <= 1 and u >= 0 and u <= 1):
                        if (intersectMbrs[i] not in refinementMbrs):
                            refinementMbrs.append(intersectMbrs[i])

    for i in refinementMbrs:
        print(i , end = " ")
    print()
    print("Cells: " + str(len(windowCells) - emptyCells[0]))
    print("Results: " + str(len(refinementMbrs)))
    print("----------")
```