# Πανεπιστήμιο Ιωαννίνων

# Πολυτεχνική Σχολή

# Τμήμα Μηχανικών Η/Υ και Πληροφορικής

# Προπτυχιακό Μάθημα: «Διαχείριση Σύνθετων Δεδομένων»

# Assignment 3

**Vasilis Nikas** **AM: 3143**

Αρχικά ο χρήστης πρέπει να δώσει σαν όρισμα καθώς πάει να τρέξει το πρόγραμμα τον αριθμό k. Ο οποίος θα χρησιμοποιηθεί για να επιστρέψει τα top k αντικείμενα.

Η εντολή που πρέπει να γράψει ο χρήστης στην γραμμή εντολών είναι η εξής:

python3 Assignment3.py k, όπου k ο αριθμός που επυθιμεί ο χρήστης

Στην συνέχεια το πρόγραμμα διαβάζει όλο το αρχείο rnd που περιέχει τα αντικείμενα ταξινομημένα με βάση το ID τους και αποθηκεύουμε όλες τις τιμές σε μια λίστα για γρήγορη προσπέλαση κάποιου στοιχείου που θα χρειαστούμε. Επειδή στο αρχείο τα αντικείμενα είναι ταξινομημένα δεν χρειάζεται να αποθηκεύσουμε κάπου το ID του γιατί γνωρίζουμε ότι η θέση της λίστας είναι και το ID του αντικειμένου.

Ο κώδικας για αυτή την υλοποίηση:

```
k = int(sys.argv[1])
rndFile = open(rndFilePath, "r")
seq1File = open(seq1FilePath, "r")
seq2File = open(seq2FilePath, "r")
tempDict = {}

# Reading the lines from the rndFile and saving the values in a list R
for line in rndFile:
    values = line.split(" ")
    tempDict[int(values[0])] = float(values[1])

R = [tempDict.get(i, 'default') for i in range(max(tempDict)+1)]
```

Στημ συνέχεια διαβάζουμε γραμμή γραμμή τα αρχεία seq1 και seq2. Για κάθε αντικείμενο που διαβάζουμε ελέγχουμε αν αυτό το αντικείμενο το βλέπουμε πρώτη φορά ή όχι. Αν το βλέπουμε πρώτη φορά συμπληρώνουμε το κάτω οριό του σε ένα λεξικό οπώς επίσης αποθηκεύουμε και την πληροφορία απο ποιο αντικείμενο το διαβάσαμε. Σε αυτό το σημειό ενημερώνουμε και την μεταβλητή threshold T όπως επίσης ελέγχουμε αν χρειάζεται να προσθέσουμε κάποιο αντικείμενο στην heap. Αν δεν το βλέπουμε πρώτη φορά το αντικείμενο τότε συμπληρώνουμε το full score του αντικείμενου σε ένα λεξικό και εκτελούμε επίσης τα άλλα βήματα.

Ο κώδικας αυτής της υλοποίησης (Αντίστοιχως κώδικας για το άλλο αρχείο):

```
for line1, line2 in zip(seq1File, seq2File):
    linesReaded += 1
    # Getting an object from the second file
    valuesSeq1 = line1.split(" ")
    tempId = int(valuesSeq1[0])
    # If the key is none it means we never seen this ID before
    if lowerLimitOfScore.get(tempId) is None:
        lowerLimitOfScore[tempId] = float(valuesSeq1[1]) + R[tempId]
        readedFrom1File[tempId] = float(valuesSeq1[1]) + R[tempId]
        objectXseq[tempId] = "seq1"
        currentseq1Value = float(valuesSeq1[1])
        thresholdT = currentseq1Value + currentseq2Value + 5
        if objectsCounter > k:
            if lowerLimitOfScore[tempId] > minHeap.getTop():
                minHeap.remove()
                minHeap.insert(lowerLimitOfScore[tempId])
        objectsCounter += 1
    # Else it means we have seen it in the other file so we just sum the 3
values
    else:
        fullScore[tempId] = lowerLimitOfScore[tempId] +
float(valuesSeq1[1])
        readedFrom1File.pop(tempId)
        objectXseq[tempId] = "seq1"
        currentseq1Value = float(valuesSeq1[1])
        thresholdT = currentseq1Value + currentseq2Value + 5
        if objectsCounter > k:
            if fullScore[tempId] > minHeap.getTop():
                minHeap.remove()
                minHeap.insert(fullScore[tempId])
                existInHeap[tempId] = True
        objectsCounter += 1
```

Πρέπει να προσέξουμε όταν διαβάσουμε ακριβώς k αντικείμενα να αρχικοποιήσουμε την minheap και να προσθέσουμε στη heap τα αντικείμενα που έχουμε δει μέχρι στιγμής.

Αρχικοποίηση της minheap:

```
if (objectsCounter == k):
    for i in lowerLimitOfScore:
        minHeap.insert(lowerLimitOfScore[i])
    for i in fullScore:
        minHeap.insert(fullScore[i])
```

Η υλοποίση της minheap είναι η εξής:

```
class MinHeap:

    def __init__(self, maxsize):
        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0] * (self.maxsize + 1)
        self.Heap[0] = -1 * sys.maxsize
        self.FRONT = 1

    # Function to return the position of
```

```
    # parent for the node currently
    # at pos
    def parent(self, pos):
        return pos // 2

    # Function to return the position of
    # the left child for the node currently
    # at pos
    def leftChild(self, pos):
        return 2 * pos

    # Function to return the position of
    # the right child for the node currently
    # at pos
    def rightChild(self, pos):
        return (2 * pos) + 1

    # Function that returns true if the passed
    # node is a leaf node
    def isLeaf(self, pos):
        return pos * 2 > self.size

    # Function to swap two nodes of the heap
    def swap(self, fpos, spos):
        self.Heap[fpos], self.Heap[spos] = self.Heap[spos], self.Heap[fpos]

    # Function to heapify the node at pos
    def minHeapify(self, pos):

        # If the node is a non-leaf node and greater
        # than any of its child
        if not self.isLeaf(pos):
            if (self.Heap[pos] > self.Heap[self.leftChild(pos)] or
                    self.Heap[pos] > self.Heap[self.rightChild(pos)]):

                # Swap with the left child and heapify
                # the left child
                if self.Heap[self.leftChild(pos)] <
self.Heap[self.rightChild(pos)]:
                    self.swap(pos, self.leftChild(pos))
                    self.minHeapify(self.leftChild(pos))

                # Swap with the right child and heapify
                # the right child
                else:
                    self.swap(pos, self.rightChild(pos))
                    self.minHeapify(self.rightChild(pos))

    # Function to insert a node into the heap
    def insert(self, element):
        if self.size >= self.maxsize:
            return
        self.size += 1
        self.Heap[self.size] = element

        current = self.size

        while self.Heap[current] < self.Heap[self.parent(current)]:
            self.swap(current, self.parent(current))
            current = self.parent(current)
```

```
    # Function to build the min heap using
    # the minHeapify function
    def minHeap(self):

        for pos in range(self.size // 2, 0, -1):
            self.minHeapify(pos)

    # Function to remove and return the minimum
    # element from the heap
    def remove(self):

        popped = self.Heap[self.FRONT]
        self.Heap[self.FRONT] = self.Heap[self.size]
        self.size -= 1
        self.minHeapify(self.FRONT)
        return popped
    # Function to return the minimum element from the heap
    def getTop(self):
        return self.Heap[self.FRONT]
```

Τέλος απομένει να ελέγξουμε τις συνθήκες για τον τερματισμό του προγράμματος μας. Αρχικά ελέγχουμε αν το threshold T είναι μικρότερο ή ίσο του μικρότερου στοιχείου του heap. Αν ισχύει αυτή η συνθήκη τότε ελέγχουμε αν υπάρχει κάποιο αντικείμενο που το πάνω όριο του είναι μεγαλύτερο απο το μικρότερο στοιχείο του heap. Αν ισχυεί και αυτή η συνθήκη συνεχίζουμε αλλιώς εκτυπώνουμε τα αντικείμενα που υπάρχουν αυτη την στιγμή στο heap και τερματίζουμε το πρόγραμμα.

Ο κώδικας είναι ο εξής:

```
if (thresholdT <= minHeap.getTop()):
    flag = 0
    # Checking if there is an object that his upperbound is bigger than
minimum element of the heap
    for i in readedFrom1File:
        if (objectXseq[i] == "seq1"):
            upperboundX = lowerLimitOfScore[i] + currentseq2Value
        else:
            upperboundX = lowerLimitOfScore[i] + currentseq1Value
        if (upperboundX > minHeap.getTop()):
            tempList = []
            flag = 0
            break
        else:
            flag = 1
    # If flag == 1 means that there is no object with the above condition
    # and we can terminate the program printing the results in the heap
    if flag == 1:
        print("Number of sequential accesses= " + str(linesReaded))
        print("Top " + str(k) + " objects")
        tempR = []
        keyList = list(fullScore.keys())
        valuesList = list(fullScore.values())
        for i in range(k):
            temp = minHeap.remove()
            possition = valuesList.index(temp)
            tempR.append(keyList[possition])
            valuesList.remove(temp)
            keyList.pop(possition)
```

```
        for i in reversed(tempR):
            print(str(i) + ": " + str(round(fullScore[i], 2)))
        quit(5)
```