

Αναφορά Παράδοσης (εργασία 2)

- A

Για το μέρος A χρησιμοποιήθηκε η ουρά προτεραιότητας του φροντιστηρίου με μία μικρή αλλαγή : επειδή τα δεδομένα με τα οποία δουλεύουμε δεν είναι primitive πρέπει κάθε φορά που αλλάζουμε κάπως την ουρά να την ανανεώνουμε βάση του comparator. Γι' αυτό δημιουργήθηκε μία extra μέθοδος update η οποία απλά βγάζει και ξαναβάζει όλα τα στοιχεία της ουράς. Χρησιμοποιήθηκαν και generics για να καλύπτει πολλαπλούς τύπους δεδομένων.

Ψευδοκώδικας για MaxPQ.update:

Copy this.heap to an array named heap

Copy this.size to an int named size

While peek() is not null:

GetMax()

For object in heap:

Add(obj)

This.size = size

- B

Η υλοποίηση του αλγορίθμου 1 είναι πολύ απλή (6 γραμμές κώδικα) χάρη στην ουρά προτεραιότητας. Συγκεκριμένα για να λύσουμε το πρόβλημα αρκεί να τρέξουμε τον παρακάτω ψευδοκώδικα (όπου folders οι φάκελοι προς αποθήκευση και disks η ουρά προτεραιότητας που κρατάει τους δίσκους με τα δεδομένα):

For folder in folders:

If no disk in disks has enough space to hold the folder or disks is empty:

add new disk to disks

update disks

Add folder to the head of the disks priority queue

Update disks

- Γ

Για το μέρος Γ χρησιμοποιήθηκε ο αλγόριθμος quick sort.

```

/* low -> Starting index, high -> Ending index */
quickSort(arr[], low, high) {
    if (low < high) {
        /* pi is partitioning index, arr[pi] is now at right place */
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}

```

```

/* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all
smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot */
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];
    i = (low - 1) // index of smaller element and indicates the
    // right position of pivot found so far
    for (j = low; j <= high - 1; j++){
        // If current element is smaller than the pivot
        if (arr[j] < pivot){
            i++; // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i+1] and arr[high]
    return (i + 1)
}

```

• Δ

Η παραγωγή των δοκιμαστικών δεδομένων για το μέρος Δ έγινε με τη χρήση της κλάσης Random του πακέτου java.util. Συγκεκριμένα δοσμένου ενός πίνακα N ο οποίος περιέχει τα διάφορα n για τα οποία θα δημιουργήσουμε δεδομένα (οπου n είναι το πλήθος φακέλων, ή αλλιώς το πλήθος των γραμμών σε κάθε αρχείο), καθώς και ενός αριθμού amount ο οποίος μας λέει πόσα αρχεία θα παράγουμε για κάθε n θα πάρουμε τον παρακάτω ψευδοκώδικα:

```
Random r = new Random()
```

```
for n in N
```

```
    for l in 0..amount
```

```
        for j in 0..n
```

```
            int generated_number = r.nextInt(0, 1_000_001)
```

```
file.write(generated_number + "\n")
```

Τα ονόματα για ευκολία δίνονται από τον παρακάτω τύπο:

nNil όπου N είναι το πλήθος φακέλων-γραμμών για το αρχείο και το l είναι ο δείκτης του αρχείου για τη συγκεκριμένη κατηγορία

Αυτό έχει ως αποτέλεσμα στη συνάρτηση review η οποία θα τρέξει το benchmark με ένα nested for loop να μπορούμε να φτιάξουμε τα ονόματα όλων των αρχείων και για το κάθε ένα να καλούμε αντίστοιχα τους αλγορίθμους 1 και 2 από το μέρος β και γ.

Ψευδοκώδικας:

```
For l in 0..N.length
```

```
    For j in 0..amount
```

```
        filename = "n" + N[i] + "l" + j;
```

Στον παραπάνω ψευδοκώδικα το N είναι ο πίνακας με το μέγεθος των δοκιμαστικών δεδομένων (πχ N=100,500,1000 από εκφώνηση) και το amount είναι το πλήθος αρχείων για κάθε n στο N (πχ στην εκφώνηση 10).

Όσον αφορά τα ευρήματα του benchmark για N = 100, 500, 1000 και amount 100 παίρνουμε στα 10 runs ένα μέσο όρο από 60 δίσκους για N = 100, 300 δίσκους για N = 500 και 600 δίσκους για N = 1000 στον αλγόριθμο 1. Για τον αλγόριθμο 2 οι αριθμοί αυτοί γίνονται 50, 250 και 500 αντίστοιχα.

```
[60, 301, 586]  
[53, 264, 507]
```

```
[58, 292, 588]  
[52, 256, 509]
```

```
[61, 295, 583]  
[55, 259, 506]
```

Η απόδοση των 2 αλγορίθμων παρόλο που το πλήθος των δεδομένων είναι σχετικά μικρό, είναι φανερή και υπέρ του αλγορίθμου 2.