

Αναφορά παράδοσης

Παπαθανασίου Αθανασία Μαρία : 3180147

Ερώτημα Α

Για την κατασκευή της ουράς προτεραιότητας του ερωτήματος Α προσαρμόσαμε την ουρά προτεραιότητας του εργαστηρίου χρησιμοποιώντας αντικείμενα τύπου Disk. Συγκεκριμένα, γίνεται χρήση σωρού για την υλοποίηση της ουράς προκειμένου να είναι πιο αποδοτική η δομή. Η μέθοδος insert εισάγει ένα στοιχείο στο τέλος της ουράς. Εάν δεν υπάρχει ελεύθερος χώρος για να εισάγουμε το στοιχείο, καλείται η βοηθητική συνάρτηση grow που αυξάνει το μέγεθος του πίνακα heap, έπειτα εισάγουμε το στοιχείο στο σωρό και καλείται η βοηθητική συνάρτηση swim εάν το κλειδί του συγκεκριμένου κόμβου είναι μεγαλύτερο από αυτό του γονιού του. Η ουρά προτεραιότητας διαθέτει επίσης μέθοδο peek που επιστρέφει τη ρίζα του δέντρου. Η συνάρτηση getMax επιστρέφει και αφαιρεί τη ρίζα του δέντρου, δηλαδή το στοιχείο με τη μεγαλύτερη προτεραιότητα χρησιμοποιώντας τη βοηθητική συνάρτηση sink όταν το κλειδί του συγκεκριμένου κόμβου είναι μικρότερο από αυτά των παιδιών του. Τέλος η MaxPQ χρησιμοποιεί την συνάρτηση compareTo της κλάσης Disk όταν χρειάζεται να συγκρίνει δύο δίσκους. Για αυτό το λόγο η υλοποίηση της ουράς προτεραιότητας έγινε με αντικείμενα τύπου Disk, δηλαδή για να συγκρίνουμε απευθείας με την compareTo δύο δίσκους.

Ερώτημα Β

Η κλάση Greedy έχει δύο μεθόδους: τη main και την printData η οποία επεξεργάζεται τα δεδομένα και εκτυπώνει τα κατάλληλα αποτελέσματα. Αρχικά στη main διαβάζουμε ολόκληρο το path του αρχείου και στη συνέχεια παίρνουμε το όνομα του αρχείου. Διαβάζοντας γραμμή -γραμμή το αρχείο αποθηκεύουμε σε μια λίστα f το μέγεθος του κάθε φακέλου που διαβάσαμε από το αρχείο. Βάζουμε τα στοιχεία της λίστας σε πίνακα για να διευκολυνθεί η επεξεργασία τους. Αρχικά ελέγχουμε αν κάποιος από τους φακέλους είναι μικρότερος από 0 και μεγαλύτερος από 1000000 MB και αν υπάρχει έστω ένας το πρόγραμμα εκτυπώνει κατάλληλο μήνυμα και τερματίζει. Αν όλοι οι φάκελοι έχουν μέγεθος εντός ορίων τότε δημιουργούμε μία ουρά προτεραιότητας με χρήση της κλάσης που υλοποιήσαμε στο ερώτημα Α. Δημιουργούμε επίσης έναν πίνακα Disk που έχει μέγεθος όσο ο πίνακας των φακέλων δηλαδή παίρνουμε τη χειρότερη περίπτωση όπου κάθε φάκελος να αποθηκεύεται σε ένα μόνο δίσκο. Έπειτα αθροίζουμε τα μεγέθη όλων των φακέλων για να τυπώσουμε στην έξοδο το συνολικό μέγεθος όλων των φακέλων σε TB. Στη συνέχεια συγκρίνουμε τον κάθε φάκελο με τον ελεύθερο χώρο κάθε δίσκου. Για έναν φάκελο κοιτάμε εάν χωράει σε κάποιον δίσκο που περιέχει ήδη κάποιους φακέλους αλλιώς τον αποθηκεύουμε σε έναν νέο δίσκο. Εισάγουμε στην ουρά προτεραιότητας κάθε δίσκο του πίνακα d που έχει ελεύθερο χώρο λιγότερο από 1000000 δηλαδή κάθε δίσκο που περιέχει φακέλους και μετράμε τον αριθμό των δίσκων που χρησιμοποιήθηκαν προκειμένου να το εκτυπώσουμε στην έξοδο. Στη συνέχεια εκτυπώνουμε το άθροισμα, τον αριθμό των δίσκων που

χρησιμοποιήθηκαν και εάν ο αριθμός των φακέλων είναι μικρότερος από 100 τότε εκτυπώνουμε αναλυτικά τα περιεχόμενα των δίσκων. Ζητείται να εκτυπώνουμε τα περιεχόμενα των δίσκων σε φθίνουσα σειρά ανάλογα με τον ελεύθερο χώρο των δίσκων. Για αυτό χρησιμοποιούμε την MaxPQ της ουράς προτεραιότητας έτσι ώστε να εκτυπώνεται το id του στοιχείου με τη μεγαλύτερη προτεραιότητα, δηλαδή του δίσκου που έχει τον περισσότερο ελεύθερο χώρο (για τη σύγκριση των ελεύθερων χώρων χρησιμοποιείται η συνάρτηση compareTo) ,εκτυπώνεται ο ελεύθερος χώρος του στοιχείου με τη μεγαλύτερη προτεραιότητα και η λίστα με τους φακέλους του κάθε δίσκου.

Ερώτημα Γ

Για την υλοποίηση της ταξινόμησης των φακέλων, χρησιμοποιήσαμε τη μέθοδο hearsort που χειρίζεται έναν πίνακα σαν σωρό και δημιουργούμε υπό-σωρούς από κάτω προς τα πάνω. Ουσιαστικά θεωρούμε τον πίνακα ως αναπαράσταση πλήρους δυαδικού δέντρου. Για να γίνει αυτό, κατασκευάζουμε αρχικά το σωρό από κάτω προς τα πάνω και στη συνέχεια εξάγουμε τα στοιχεία και τα τοποθετούμε στο τέλος, ξεκινώντας από τη ρίζα. Η αναδρομική συνάρτηση hearify τοποθετεί το μεγαλύτερο στοιχείο στο τέλος του σωρού ως εξής: βρίσκουμε το αριστερό και το δεξί παιδί του κόμβου i και συγκρίνουμε αρχικά το αριστερό παιδί με την ρίζα και στη συνέχεια, το μεγαλύτερο από αυτά τα δύο με το δεξί παιδί. Εάν το μεγαλύτερο στοιχείο δεν ήταν η ρίζα το μέγιστο στοιχείο ανταλλάσσει τιμή με το στοιχείο i και η διαδικασία επαναλαμβάνεται για το υπό-δέντρο αλλά αντί για τον κόμβο i η συνάρτηση hearify παίρνει ως παράμετρο το μέγιστο στοιχείο. Η ταξινόμηση hearsort γίνεται συνολικά σε χρόνο $O(N) + O(N \log N) = O(N \log N)$, αφού για τη μετατροπή του πίνακα σε σωρό από κάτω προς τα πάνω ο χρόνος είναι $O(N)$ και κάθε εξαγωγή απαιτεί χρόνο $O(\log N)$. Με τη μέθοδο αυτή, τα στοιχεία έχουν διαταχθεί σε αύξουσα σειρά οπότε στην κλάση Greedy εάν η μεταβλητή sort που ορίζει εάν θα γίνει ή όχι ταξινόμηση των στοιχείων, γίνεται true και αντιστρέφουμε τα στοιχεία του πίνακα έτσι ώστε να είναι διατεταγμένα σε φθίνουσα σειρά. Το ερώτημα γ διαθέτει main όπως ακριβώς και το β και για να τρέξει δίνουμε όλο το path του αρχείου (> java Sort path_to_file/filename.txt).

Ερώτημα Δ


Χρησιμοποιήσαμε την κλάση Random για να δημιουργήσουμε αρχεία με τυχαία δεδομένα στο διάστημα 0 έως 1000000. Για να γίνει αυτό γράφουμε τις ακόλουθες εντολές: Random random = new Random(); και random.nextInt(1000001); Η τελευταία εντολή μας δίνει έναν ακέραιο μεταξύ 0 και 1000000. Ορίζουμε μία μεταβλητή n που αντιπροσωπεύει το πλήθος των φακέλων και για τα αρχεία από 0 έως 9 έχει την τιμή 100, για τα αρχεία από 10 έως 19 έχει την τιμή 500 και για τα υπόλοιπα την τιμή 1000. Επιλέξαμε να χρησιμοποιήσουμε τις τιμές της εκφώνησης αν και θα μπορούσε να γίνει και για άλλες τιμές του n και με περισσότερα αρχεία. Δημιουργούμε ένα ρεύμα για εγγραφή των τυχαίων ακεραίων στα αρχεία και γράφουμε σε κάθε αρχείο τόσους ακεραίους όσο είναι η τιμή του n. Ορίζουμε την

boolean μεταβλητή print σε true προκειμένου να εκτυπωθεί μόνο ο αριθμός των δίσκων που χρησιμοποιήθηκαν και όχι τα υπόλοιπα αποτελέσματα της printData. Στη συνέχεια ορίζουμε τρεις μεταβλητές τις s1, s2, s3 οι οποίες αντιπροσωπεύουν το άθροισμα των δίσκων για n=100, n=500, n=1000 αντίστοιχα. Δημιουργούμε δύο πίνακες 30 θέσεων έναν για τον πρώτο αλγόριθμο και έναν για τον δεύτερο που θα περιέχουν το άθροισμα των δίσκων για τις διαφορετικές τιμές του n. Έπειτα δημιουργούμε μια λίστα και ένα αντικείμενο τύπου Scanner και ακολουθούμε τη συνηθισμένη διαδικασία για να διαβάσουμε τα δεδομένα από τα αρχεία και να τα αποθηκεύσουμε στη λίστα, την οποία στη συνέχεια δίνουμε ως όρισμα στην printData για να εκτυπώσει πόσους σκληρούς δίσκους χρειαστήκαμε για κάθε αρχείο. Επαναλαμβάνουμε ακριβώς την ίδια διαδικασία για τον δεύτερο αλγόριθμο με μόνη διαφορά ότι ορίζουμε true την boolean μεταβλητή sort ώστε να γίνει η ταξινόμηση των φακέλων πριν από την επεξεργασία τους. Για να εκτυπώσουμε κατά μέσο όρο πόσοι δίσκοι χρησιμοποιήθηκαν για τις διαφορετικές τιμές του αριθμού των φακέλων βάζουμε στο s1 τις πρώτες 10 θέσεις του πίνακα sum, στο s2 τις επόμενες 10 θέσεις του πίνακα sum και στο s3 τις τελευταίες 10 θέσεις του πίνακα sum. Ακολουθούμε ακριβώς την ίδια διαδικασία για τον πίνακα sum1 που είναι το άθροισμα όλων των σκληρών δίσκων για τον δεύτερο αλγόριθμο. Τέλος, διαιρούμε τα αθροίσματα με το 10 που είναι ο αριθμός των αρχείων για να δούμε κατά μέσο όρο πόσοι δίσκοι χρησιμοποιήθηκαν για διαφορετικές τιμές του αριθμού των φακέλων. Εάν τρέξετε το πρόγραμμα Randomizer θα δείτε ότι ο αριθμός των δίσκων που χρησιμοποιήθηκαν αφού ταξινομήσαμε τους φακέλους, είναι μικρότερος από τον αριθμό των δίσκων που χρησιμοποιήθηκαν χωρίς να γίνει πρώτα ταξινόμηση. Αυτό είναι λογικό γιατί ο αλγόριθμος που χρειαζόταν ταξινόμηση των φακέλων πριν από την επεξεργασία τους είναι πιο αποδοτικός και συνεπώς για κάθε τιμή του N ο αλγόριθμος με την ταξινόμηση απαιτεί λιγότερους δίσκους. Ενδεικτικά, στις παρακάτω εικόνες φαίνεται πρακτικά ότι ο αλγόριθμος με την ταξινόμηση είναι πιο αποδοτικός:


Σημείωση: όλος ο κώδικας μεταγλωττίζεται και τα προγράμματα των ερωτημάτων β, γ και δ έχουν μέθοδο main άρα εκτελούνται. Όλα τα προγράμματα τα έτρεξα από τη γραμμή εντολών και συγκεκριμένα για τα ερωτήματα β και γ έδινα όλο το path του αρχείου οπότε και αυτό αποθηκεύεται στο args[0] δηλαδή για το ερώτημα β : `java Greedy path_to_file/filename.txt`, για το ερώτημα γ : `java Sort path_to_file/filename.txt` και για το ερώτημα δ : `java Randomizer`. Τα αρχεία τυχαίων δεδομένων έχουν ονόματα από random0 έως random29.

Γραμμή εντολών

```
C:\Users\Σίσσυ Παπαθανασίου\Documents\ds-2\src>java Randomizer
Greedy1
for file random0.txt
Total number of disks used = 63
for file random1.txt
Total number of disks used = 63
for file random2.txt
Total number of disks used = 53
for file random3.txt
Total number of disks used = 57
for file random4.txt
Total number of disks used = 56
for file random5.txt
Total number of disks used = 58
for file random6.txt
Total number of disks used = 57
for file random7.txt
Total number of disks used = 55
for file random8.txt
Total number of disks used = 59
for file random9.txt
Total number of disks used = 49
for file random10.txt
Total number of disks used = 274
for file random11.txt
Total number of disks used = 271
for file random12.txt
Total number of disks used = 272
for file random13.txt
Total number of disks used = 261
for file random14.txt
Total number of disks used = 274
for file random15.txt
Total number of disks used = 245
for file random16.txt
Total number of disks used = 265
for file random17.txt
Total number of disks used = 267
for file random18.txt
Total number of disks used = 280
for file random19.txt
Total number of disks used = 268
for file random20.txt
```

 Γραμμή εντολών

```
for file random20.txt
Total number of disks used = 514
for file random21.txt
Total number of disks used = 544
for file random22.txt
Total number of disks used = 543
for file random23.txt
Total number of disks used = 514
for file random24.txt
Total number of disks used = 527
for file random25.txt
Total number of disks used = 523
for file random26.txt
Total number of disks used = 520
for file random27.txt
Total number of disks used = 529
for file random28.txt
Total number of disks used = 530
for file random29.txt
Total number of disks used = 536
Greedy1: average number of disks used for N=100: 57
Greedy1: average number of disks used for N=500: 267
Greedy1: average number of disks used for N=1000: 528
Greedy2
for file random0.txt
Total number of disks used = 59
for file random1.txt
Total number of disks used = 58
for file random2.txt
Total number of disks used = 51
for file random3.txt
Total number of disks used = 53
for file random4.txt
Total number of disks used = 55
for file random5.txt
Total number of disks used = 56
for file random6.txt
Total number of disks used = 54
for file random7.txt
Total number of disks used = 52
for file random8.txt
Total number of disks used = 56
```

 Γραμμή εντολών

```
Total number of disks used = 261
for file random11.txt
Total number of disks used = 260
for file random12.txt
Total number of disks used = 260
for file random13.txt
Total number of disks used = 249
for file random14.txt
Total number of disks used = 259
for file random15.txt
Total number of disks used = 235
for file random16.txt
Total number of disks used = 254
for file random17.txt
Total number of disks used = 257
for file random18.txt
Total number of disks used = 271
for file random19.txt
Total number of disks used = 256
for file random20.txt
Total number of disks used = 494
for file random21.txt
Total number of disks used = 526
for file random22.txt
Total number of disks used = 526
for file random23.txt
Total number of disks used = 495
for file random24.txt
Total number of disks used = 507
for file random25.txt
Total number of disks used = 500
for file random26.txt
Total number of disks used = 505
for file random27.txt
Total number of disks used = 510
for file random28.txt
Total number of disks used = 510
for file random29.txt
Total number of disks used = 512
Greedy2: average number of disks used for N=100: 54
Greedy2: average number of disks used for N=500: 256
Greedy2: average number of disks used for N=1000: 508
```