2η Εργασία Δομές Δεδομένων

Χρήστος Τασιόπουλος p3150170
Μιχαήλ Ρούσσος p3150148

Μέρος Α:

Στο μέρος Α έχουμε δημιουργήσει την κλάση **PrintJob** που αναπαριστά αρχεία προς εκτύπωση και υλοποιεί την διεπαφή Comparable<PrintJob> . Η κλάση αυτή έχει τα εξής πεδία:

- int id
- int size
- int waitingTime
- int arrivalTime
- int priority

καθώς έχουμε και μια στατική μεταβλητή για την κλάση τη numPJ που την χρησιμοποιούμε για να δίνει τιμές στο id του κάθε αντικείμενου που δημιουργούμε. Έχουμε τρεις Constructors

- τον κενό που δίνει 0 σε όλα τα παιδιά εκτός από το id
- έναν με δυο int ορίσματα που αντιστοιχούν στο μέγεθος και τον χρόνο άφιξης
- και έναν με τέσσερα που δίνει τιμές στο μέγεθος, χρόνο αναμονής και άφιξης και προτεραιότητα

και στους τρεις Constructors το πεδίο id παίρνει την τιμή του από το numPJ και μετά αυξάνουμε το numPJ κατά 1. Έχουμε setters και getters για όλα τα πεδία εκτός του id που έχει μόνο getter. Καθώς υπάρχει και η υλοποίηση της μεθόδου int compareTo(PrintJob pj) που κληρονομείται από την διεπαφή. Η μέθοδος αυτή επιστρέφει 1 αν το αντικείμενο πάνω στο οποίο καλείται είναι μεγαλύτερο από το όρισμα που δέχεται -1 αν είναι μεγαλύτερο το όρισμα που δέχεται και 0 αν είναι ίσα.

Επίσης έχουμε φτιάξει την κλάση **MaxPQ** που αναπαριστά μια ουρά προτεραιότητας με αντικείμενα τύπου PrintJob και υλοποιείται με δυαδικό δέντρο . Η κλάση αυτή έχει τα εξής πεδία:

- PrintJob[] heap
- int size

και τις παρακάτω μεθόδους την **size** που μετρά τον αριθμό αντικειμένων PrintJob που υπάρχουν στον heap ,την **isEmpty** η οποία ελέγχει αν η ουρά είναι άδεια , την **insert** που εισάγει νέο αντικείμενο στην ουρά, την **getMax** που αφαιρεί και επιστρέφει το αντικείμενο με τη μέγιστη προτεραιότητα ,την **peek** που επιστρέφει χωρίς να διαγράψει το αντικείμενο με τη μέγιστη προτεραιότητα, την **resize** που διπλασιάζει το μέγεθος του πίνακα (καλείται όταν ξεπεράσουμε το 75% του διαθέσιμου χώρου) , την **sink(int i)** που αν ένα κόμβος του δέντρου έχει παιδιά με μεγαλύτερη προτεραιότητα τον κατεβάζουμε μέχρι να είναι στη σωστή θέση , την **swim(int i)** που ελέγχει αν ο γονέας έχει μικρότερη προτεραιότητα και ανεβάζει το παιδί μέχρι ο γονέας του να μην έχει μικρότερη προτεραιότητα και τέλος την **swap(int i, int j)** που αντιμεταθέτει δυο αντικείμενα από το δέντρο.

Μέρος Β:

Έχουμε φτιάξει την κλάση AlgorithmB που έχει μια μέθοδο **public static void runB(String data)** που δέχεται σαν όρισμα το όνομα του αρχείου .txt που θα

διαβάσει .Επίσης η κλάση αυτή έχει τις μεθόδους **static boolean read(String data, StringQueueImpl<PrintJob> input)** και
static void sort(StringQueueImpl<PrintJob> queue, String data) .
Η μέθοδος read δέχεται το όνομα του αρχείου που θα διαβάσει και ένα αντικείμενο StringQueueImpl<PrintJob> που έχει οριστεί στην runB έτσι ώστε να αποθηκευτούν τα δεδομένα εκεί και να έχει πρόσβαση και η sort. Η sort δέχεται το αντικείμενο StringQueueImpl<PrintJob> που είπαμε ήδη άλλα και το όνομα του αρχείου για να δώσει το κατάλληλο όνομα στο αρχείο που θα δημιουργήσει. Η read επιστρέφει έναν boolean που είναι true αν το αρχείο έχει την σωστή μορφή ,δηλαδή ακριβώς 2 αριθμούς σε κάθε γραμμή ,ο δεύτερος να είναι στο διάστημα [1,128] και ο πρώτος αριθμός κάθε γραμμής να είναι μεγαλύτερος ή ίσος του πρώτου της προηγούμενης και μη αρνητικοί αλλιώς επιστρέφει false. Η sort καλείται μόνο όταν η read έχει επιστρέψει true. Καθώς διαβάζονται τα δεδομένα περνιούνται στην queue για να χρησιμοποιηθούν από την sort.
Η sort αρχικά θέτει Τ τον χρόνο άφιξης του πρώτου από την queue και περνά όλα τα αντικείμενα με τον ίδιο χρόνο άφιξης στην pq (όπου η pq είναι μια ουρά προτεραιότητας MaxPQ) ορίζοντας την προτεραιότητά τους 128- το μέγεθος τους έτσι τα μικρότερα αρχεία θα έχουν μέγιστη προτεραιότητα παίρνει με την getMax αυτό με τη μέγιστη προτεραιότητα ή ένα από αυτά αν είναι πολλά και το γράφουμε στο txt και αυξάνουμε το Τ κατά το μέγεθος του αρχείου που εκτυπώσαμε . Μετά περνάμε στην pq όσα αντικείμενα από την queue έχουν χρόνο άφιξης μικρότερο ή ίσο με Τ. Επαναλαμβάνουμε μέχρι να αδειάσει η queue και μετά εκτυπώνουμε το getMax από την pq μέχρι να αδειάσει .Αν αδειάσει η pq και δεν έχει αδειάσει η queue τότε θέτουμε ως Τ τον χρόνο άφιξης του τωρινού πρώτου στοιχείου από την queue και ξαναρχίζουμε την επανάληψη. Κατά τη διάρκεια της sort κρατάμε σε μια μεταβλητή το άθροισμα των χρόνων αναμονής καθώς και το αντικείμενο με τον μέγιστο χρόνο αναμονής άλλα και το πόσος είναι αυτός σε static μεταβλητές .

Μέρος Γ:

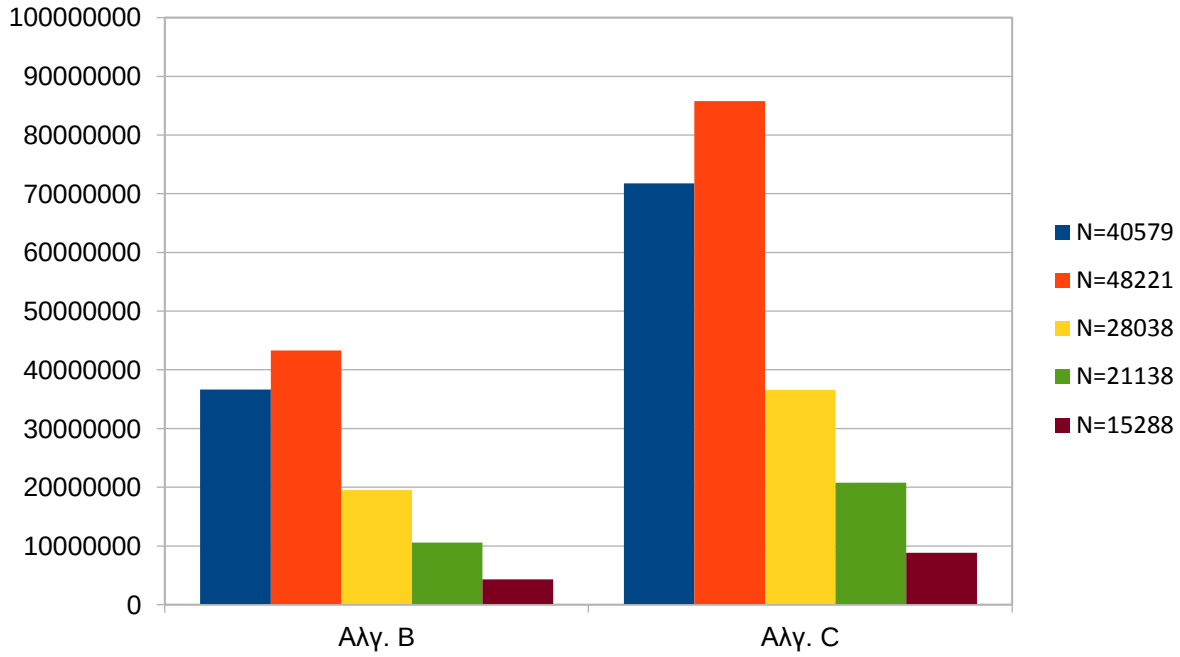
Έχουμε φτιάξει την κλάση AlgorithmC που έχει μια μέθοδο **public static void runC(String data)** που δέχεται σαν όρισμα το όνομα του αρχείου .txt που θα διαβάσει .Επίσης η κλάση αυτή έχει τις μεθόδους **static boolean read(String data, StringQueueImpl<PrintJob> input)** και
static void sort(StringQueueImpl<PrintJob> queue, String data) .
Η μέθοδος read είναι σχεδόν ίδια με την read στην AlgorithmB. Η sort είναι παρόμοια με την sort στην AlgorithmB αυτό που αλλάζει είναι ότι κάθε φορά που μπαίνουν αντικείμενα στην pq προτού καλέσουμε την getMax αλλάζουμε την προτεραιότητά και την θέτουμε ίση με min(127, priority + waitingTime-(waitingTime)mod15) έτσι είναι σαν να κάνουμε min(127, priority + waitingTime) κάθε 15 δεύτερα . Αυτό το κάνουμε έτσι γιατί δεν μας ενδιαφέρει να αλλάξουν οι προτεραιότητες όσο εκτυπώνουμε κάτι μόνο πριν διαλέξουμε το επόμενο αντικείμενο. Η ίδια αλλαγή γίνεται και αφού αδειάσει η queue και εκτυπώνονται τα αντικείμενα της pq .

Μέρος Δ:

Έχουμε φτιάξει την κλάση Creatinginputfiles που έχει μια μέθοδο **public static void inputfiles ()** η οποία φτιάχνει τα 50 αρχεία εισόδου. Αρχικά βρίσκει αρχικά έναν τυχαίο αριθμό στο διάστημα [0,49901) και μετά βρίσκει άλλους τέσσερεις που ο καθένας θα απέχει τουλάχιστον 5000 από τους προηγούμενους με την βοήθεια της RandomGenerator και καθώς βρίσκουμε τους αριθμούς τους βάζουμε στον πίνακα sizeofFiles[]. Πριν τους χρησιμοποιήσουμε αυτούς τους αριθμούς τους αυξάνουμε κατά 100 για να είναι στο διάστημα [100,50000]. Το ν-οστο αρχείο έχει sizeofFiles[ν]
γραμμές και κάθε γραμμή έχει 2 αριθμούς τον χρόνο άφιξης και το μέγεθος για τα αρχεία. Πρέπει ο χρόνος άφιξης της κάθε γραμμής να είναι μεγαλύτερος η ίσος με της προηγούμενης . Για να το πετύχουμε αυτό καθώς και για να περιορίσουμε την αύξηση του χρόνου ορίσαμε δυο int t1=0,t2=0 και είπαμε ότι αν t1 είναι της τωρινής γραμμής ο χρόνος και t2 της προηγούμενης τότε το t1=generator.nextInt(129)+t2 . Έτσι και το t1>=t2 αλλά και η αύξηση περιορίζεται και δεν έχουμε τεράστιους αριθμούς . Το μέγεθος απλά πρέπει να είναι στο διάστημα [1,128] έτσι αν s το μέγεθος τότε s=generator.nextInt(128)+1 που είναι 0<=generator.nextInt(128)<=127 άρα 1<=generator.nextInt(128)+1<=128 δηλαδή 1<=s<=128 άρα είναι στα σωστά όρια .

Μέση αναμονή

	N=40579	N=48221	N=28038	N=21138	N=15288
Αλγ. B	36624650	43282571	19517969	10555611	4321441
Αλγ. C	71783235	85780473	36565730	20783753	8831826



Μέση τιμή των μέγιστων αναμονών

	N=40579	N=48221	N=28038	N=21138	N=15288
Αλγ. B	270146	334380	196260	133293	105074
Αλγ. C	273374	336305	197887	138278	99467