1η εργασία δομές δεδομένων

ονοματεπώνυμα: 1) Κρόιτορ Καταρτζίου Ιωάν (Α.Μ. : Π21077)

2) Ρούτης Αλέξιος (Α.Μ. : Π21145)

σημείωση *δεν αναλύσαμε της μεθόδους οι οποίες περιέχονται στο βιβλίο καθώς η εξήγηση υπάρχει ήδη στο βιβλίο και το θεωρήσαμε περιττό, αλλά από την άλλη αναλύσαμε τις επιπλέον μεθόδους τις οποίες εμείς προσθέσαμε ή τυχόν παραλλαγές (επιπλέον παρέχουμε πολλά σχόλια μέσα στον κώδικα)*

άσκηση 1

Ακολουθώντας την εκφώνηση και της υποδείξεις της, και μέσω της βοήθειας του κεφ.3.4 (σελ. 126) δημιουργήσαμε μία γραμμική λίστα L, χρησιμοποιώντας την μονά συνδεδεμένη αναπαράσταση, αφού πρώτα πήραμε από τον χρήστη τον αριθμό τον κόμβων που θέλει να έχει η L. Ύστερα μέσω ενός for βρόχου προσθέσαμε στο data μέρος του κάθε κόμβου έναν τυχαίο ακέραιο αριθμό μέσω της Insert(int k, const T& x) μεθόδου, η οποία λαμβάνει ως όρισμα τον ακέραιο k που αντιπροσωπεύει τη θέση της λίστας (κόμβος) στην οποία θα τοποθετήσουμε το στοιχείο, και μια σταθερή αναφορά x, που αντιπροσωπεύει τον αριθμό τον οποίο θα τοποθετήσουμε στη θέση k.

Συνεχίζοντας, δημιουργούμε μία ακόμη λίστα, την histogram, στην οποία θα αποθηκεύσουμε τόσο τον κάθε αριθμό που βρίσκεται στην L, όσο και το πλήθος εμφάνισής του. Για αυτό στην κλάση Chain προσθέσαμε μία μεταβλητή occurrence στην οποία θα αποθηκεύουμε το πλήθος εμφάνισής του κάθε αριθμού. Έτσι, μέσα σε έναν βρόχο for ο οποίος επαναλαμβάνεται από a έως b δηλαδή εντός του εύρους (a,b), το οποίο είναι το εύρος μεταξύ του οποίου θα παραχθούν οι τυχαίοι αριθμοί, θα προσπελαστεί η L ώστε να βρεθεί πόσες φορές εμφανίζεται ο κάθε αριθμός. Σε κάθε προσπέλαση η προσωρινή μεταβλητή temp αποθηκεύει το πλήθος εμφανίσεων του i-οστού αριθμού και ύστερα πραγματοποιείται

ένας έλεγχος ώστε να διαπιστωθεί αν ο i-οστός αριθμός υπάρχει στην L, και αν αυτός ο έλεγχος επιτύχει ο αριθμός προστίθεται δυναμικά στην histogram.

Η προσπέλαση στην L γίνεται μέσω της μεθόδου Count(const T& x), η οποία είναι μία παραλλαγή της μεθόδου Search(const T& x) (την οποία εμείς σκεφτήκαμε) της σελ.129, αφού προσθέσαμε έναν έλεγχο ο οποίος διαπιστώνει αν ο αριθμός που περιέχεται στη data μεταβλητή του κάθε κόμβου της L ισούται με τον αριθμό που περάσαμε ως όρισμα στην μέθοδο, και αν ο έλεγχος επιτύχει αυξάνεται ο μετρητής (count) του αριθμού κατά ένα και στη συνέχεια ο μετακινούμαστε στον επόμενο κόμβο. Η προηγούμενη διαδικασία γίνεται μέσα σε έναν βρόχο while ο οποίος τρέχει έως ότου αντιληφθεί ότι ο δείκτης του επόμενου κόμβου και ο τρέχων δείκτης δείχνει σε NULL. Αυτός ο έλεγχος, όμως, δεν περιείχε την προσπέλαση του τελευταίου κόμβου για αυτό προσθέσαμε έξω από το while έναν έλεγχο για το περιεχόμενο του τελευταίου κόμβου και τέλος επιστρέφουμε την τιμή του μετρητή (count).

Τέλος, χρησιμοποιήσαμε μία παραλλαγή (την οποία επίσης εμείς σκεφτήκαμε) της μεθόδου AppendHistogram(const T& x, const T& z) της σελ. 134, η οποία δυναμικά κατά την εκτέλεση του προγράμματος δημιουργεί έναν νέο κόμβο στη histogram και προσθέτει στη data μεταβλητή το x, που αντιπροσωπεύει τον αριθμό της L που θα προσθέσουμε, και στην occurrence μεταβλητή προσθέτει το z, που αντιπροσωπεύει τον αριθμό εμφανίσεων του x στην L μέσω της εντολής ‘ y->occurrence = z; ’, εντολή στην οποία έγκειται και η αλλαγή που πραγματοποιήσαμε.

Όσον αφορά την αύξουσα σειρά τοποθέτησης των αριθμών στη histogram, αυτή ούτως ή άλλως γίνεται εφόσον ο έλεγχος στο for βρόχο για την προσθήκη στην histogram γίνεται από τον ελάχιστο αριθμό στον μεγαλύτερο.

<u>άσκηση 2</u>

Ομοίως ακολουθώντας την εκφώνηση και της υποδείξεις της, και μέσω της βοήθειας του κεφ. 9.3.5 της σελ. 424 του βιβλίου, δημιουργήσαμε τις κλάσεις MaxHeap και MinHeap οι οποίες φέρνουν εις πέρας τις ίδιες ακριβώς λειτουργίες με διαφορές στους αλγόριθμους με τους οποίους λειτουργούν. Μία μικρή διαφορά που περιλαμβάνουν όλες οι μέθοδοι είναι οι χρήση των bitwise τελεστών (<< , >>) αντί αριθμητικών καθώς είναι γρηγορότερες οι πράξεις μέσω αυτών. Οι λειτουργίες των μεθόδων του MaxHeap αναλύονται στο βιβλίο οπότε θα αρκεστούμε στην περιγραφή αυτών του MinHeap.

Η Initialize(T a[], int size, int ArraySize) της MinHeap εκτελεί την ίδια ακριβώς λειτουργία με αυτήν της MaxHeap με διαφορά ότι ελέγχει αν το αριστερό παιδί είναι μεγαλύτερο από το δεξιό παιδί, ‘heap[c] > heap[c+1]‘. Πρακτικά αυτή είναι η διαφορά που παρατηρείται σε όλες της μεθόδους του MinHeap και το γεγονός ότι στη ρίζα βρίσκεται πάντα το ελάχιστο στοιχείο ενώ σε αντίθεση με το MaxHeap, όπου εκεί στη ρίζα βρίσκουμε πάντα το μέγιστο στοιχείο.

Στη main() δημιουργήσαμε έναν δυναμικό πίνακα heap_min, μεγέθους που καθορίζεται από τον χρήστη, ο οποίος παίρνει τυχαίους float αριθμούς, μέσω ενός βρόχου for. Στη συνέχεια δημιουργήσαμε έναν νέο πίνακα δυναμικά, τον heap_max, ίδιου μεγέθους με τον heap_min, στον οποίο αντιγράψαμε μέσω ενός for βρόχου τους αριθμούς του heap_min. Επιπροσθέτως, αρχικοποιήσαμε τους δύο σωρούς (maxheap και minheap) και τους περάσαμε ως παράμετρούς τους πίνακες heap_max και heap_min αντίστοιχα, όπως και τα μεγέθη τους. Επιπλέον, σε έναν while βρόχο, διαγράφαμε τα δύο ελάχιστα (αντ. μέγιστα) στοιχεία του σωρού, μέσω της DeleteMin(T& x) (αντ. DeleteMax(T& x)) και τα αποθηκεύαμε στις προσωρινές μεταβλητές addMin1 και addMin2 (αντ. addMax1 και addMax2), των οποίων το άθροισμα το αποθηκεύαμε στην SumMin (αντ. SumMax) και το εισαγάγαμε ξανά στον σωρό μέσω της InsertMin(const T& x) (αντ. InsertMax(const T& x)). Μετά από κάθε εισαγωγή πραγματοποιούσαμε έναν έλεγχο ο οποίος διαπίστωνε εάν ο σωρός

περιέχει ένα μόνο στοιχείο, και αν αυτός επιτύγχανε, τότε εμφάνιζε το αποτέλεσμα και διέκοπτε τον βρόχο μέσω break. Τέλος, μέσω της συνάρτησης compare_float(float x, float y, float epsilon) συγκρίναμε τους δύο αριθμούς, ελέγχοντας εάν η απόλυτη διαφορά τους είναι μεγαλύτερη του ε που ορίζαμε. Για τη χρήση της χρειαστήκαμε τη βιβλιοθήκη cmath και για την εύρεσή της συμβουλευτήκαμε το διαδίκτυο.