Δομές Δεδομένων

1η Εργασία

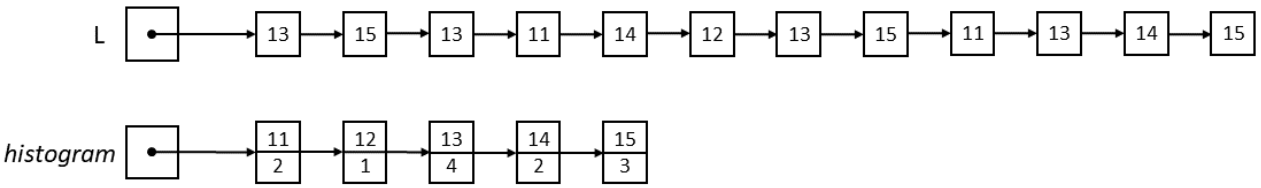
2021-2021

Ερώτημα 1

Έστω μια γραμμική λίστα L. Στα πλαίσια της εργασίας θα πρέπει να χρησιμοποιήσετε την μονά συνδεδεμένη αναπαράσταση για όλες τις γραμμικές λίστες που θα χρειαστείτε. Για λεπτομέρειες δείτε το Κεφ. 3.4 του βιβλίου "Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++". **Δεν επιτρέπεται η χρήση των έτοιμων δομών δεδομένων που προσφέρει η C++ (C++ Containers).** Τα δεδομένα σε κάθε κόμβο της λίστας L αποτελούνται από έναν ακέραιο αριθμό.

Σκοπός του προγράμματος C++ που θα φτιάξετε είναι να δημιουργήσετε μια νέα λίστα *histogram* στην οποία θα αποθηκεύεται πόσες φορές εμφανίζεται κάθε αριθμός στην λίστα L. Τα στοιχεία της λίστας *histogram* θα πρέπει να δημιουργούνται δυναμικά κάθε φορά που συναντάμε έναν αριθμό στην λίστα L για πρώτη φορά και να είναι ταξινομημένα κατά αύξουσα σειρά με βάση τους αριθμούς που περιέχει η λίστα L.

Το παρακάτω σχήμα δίνεται προς διευκρίνιση λεπτομερειών. Η λίστα L περιέχει ένα σύνολο αριθμών και η λίστα *histogram* περιέχει πληροφορίες για το πόσες φορές εμφανίζεται κάθε αριθμός. Παρατηρήστε πως η λίστα *histogram* είναι ταξινομημένη με βάση τους αριθμούς της λίστας L και όχι με βάση το πλήθος εμφανίσεων κάθε αριθμού.



Θέματα υλοποίησης

- Το μέγεθος της λίστας L (δηλαδή το πλήθος των αριθμών που θα περιέχει) θα το διαβάζετε κατά την ώρα εκτέλεσης του προγράμματος από τον χρήστη.
- Στην συνέχεια το πρόγραμμα σας θα πρέπει να κατασκευάσει την λίστα L. Για κάθε κόμβο θα πρέπει να οριστεί ο αριθμός που θα περιέχει ως δεδομένο. Για τον σκοπό αυτό μπορεί να γίνει χρήση **γεννήτριας τυχαίων αριθμών**. Η C++ προσφέρει ένα σύνολο κλάσεων και μεθόδων για τον σκοπό αυτό (δείτε στο https://www.cplusplus.com/reference/random). Για να χρησιμοποιήσετε αυτές τις δυνατότητες:
    - Συμπεριλάβετε τα αρχεία κεφαλίδας <random> και <functional> στο πρόγραμμα σας:

```
#include <random>
#include <functional>
```

- Ορίστε μια γεννήτρια τυχαίων αριθμών:

  ```
  std::default_random_engine generator;
  ```

- Ορίστε τα επιτρεπτά όρια και την κατανομή για τους παραγόμενους τυχαίους αριθμούς. Με την παρακάτω δήλωση ζητάμε για τον αριθμό που θα περιέχει κάθε κόμβος ομοιόμορφη κατανομή και οι παραγόμενοι αριθμοί να είναι ακέραιοι στο διάστημα [0, 100]. Είστε ελεύθεροι να αλλάξετε το όριο αυτό:

  ```
  std::uniform_int_distribution<int> data_element_distribution(0, 100);
  ```

- Προς ευκολία χρήσης μπορείτε να «δέσετε» μαζί την γεννήτρια τυχαίων αριθμών με την επιθυμητή κατανομή:

  ```
  auto random_element   = std::bind(data_element_distribution, generator);
  ```

- Κάθε φορά που θα καλείτε την `random_element()` θα παράγεται ένας τυχαίος αριθμός με βάση την κατανομή και τα όρια που έχετε ορίσει. Π.χ. με την παρακάτω κλήση δημιουργείται ένας τυχαίος αριθμός στο διάστημα [0, 100] που μπορεί να χρησιμοποιηθεί ως δεδομένο για έναν κόμβο σε μια λίστα:

  ```
  int data_element = random_element();
  ```

<u>Ερώτημα 2</u>

Η αναπαράσταση των αριθμών κινητής υποδιαστολής (floating point numbers) σε έναν υπολογιστή χρησιμοποιεί συγκεκριμένο πλήθος από bytes: 4 bytes για τον τύπο δεδομένων float και 8 bytes για τον τύπο δεδομένων double της C++. Αυτό έχει ως αποτέλεσμα να μπορεί να αποθηκευτεί μέχρι ένα μέγιστο πλήθος δεκαδικών ψηφίων κάθε αριθμού. Αν ο αριθμός έχει περισσότερα δεκαδικά ψηφία αυτά αναγκαστικά αποκόπτονται. Το ίδιο συμβαίνει στο αποτέλεσμα που προκύπτει μετά από οποιαδήποτε αριθμητική πράξη μεταξύ αριθμών κινητής υποδιαστολής. Έτσι εισάγονται **αριθμητικά σφάλματα** στους υπολογισμούς, τα οποία προφανώς πρέπει να ελαχιστοποιηθούν.

Στην εργασία θα μας απασχολήσει η πράξη της πρόσθεσης αριθμών τύπου float. Δεν θα χρησιμοποιήσουμε αριθμούς τύπου double, απλώς για να φανεί πιο εύκολα η επίδραση των αριθμητικών σφαλμάτων. Αν προσπαθήσουμε να προσθέσουμε έναν πολύ μεγάλο και έναν πολύ μικρό αριθμό κινητής υποδιαστολής, λόγω των αριθμητικών σφαλμάτων ο μικρότερος αριθμός μπορεί να «χαθεί» λόγω της αποκοπής των δεκαδικών ψηφίων. Αν θέλουμε να προσθέσουμε μεταξύ τους περισσότερους αριθμούς κινητής υποδιαστολής τότε θα πρέπει να ξεκινήσουμε προσθέτοντας τους δύο μικρότερους αριθμούς μεταξύ τους, το αποτέλεσμα να το προσθέσουμε με τον αμέσως μικρότερο αριθμό, κλπ. Με αυτόν τον τρόπο το άθροισμα μεγαλώνει και πλησιάζει περισσότερο προς τον επόμενο αριθμό που πρέπει να προστεθεί και το αποτέλεσμα έχει μεγαλύτερη ακρίβεια.

Στα πλαίσια της εργασίας θα πρέπει να υλοποιήσετε σε C++ έναν **σωρό ελαχίστων** για να βρίσκετε με αποδοτικό τρόπο τους μικρότερους αριθμούς που πρέπει να προστεθούν κάθε φορά. Θα υλοποιήσετε επίσης την πρόσθεση αριθμών κινητής υποδιαστολής ξεκινώντας από τους μεγαλύτερους αριθμούς (επομένως θα πρέπει να υλοποιήσετε και έναν **σωρό μεγίστων**), για να δείτε τις διαφορές στα αριθμητικά αποτελέσματα. Συγκεκριμένα:

- Θα δημιουργήσετε έναν πίνακα heap_min με μέγεθος N, στον οποίο θα αποθηκεύσετε αριθμούς κινητής υποδιαστολής. Το μέγεθος N του πίνακα θα το διαβάζετε κατά την ώρα εκτέλεσης του προγράμματος από τον χρήστη και ο πίνακας heap_min θα πρέπει να δεσμεύεται δυναμικά.
- Το μέγεθος του πίνακα N μπορεί στην αρχή να είναι μικρό για να μπορείτε να επαληθεύσετε τα αποτελέσματα, αλλά θα πρέπει στο τέλος να τρέξετε και να παρουσιάσετε αποτελέσματα για μεγάλα N, π.χ. N > 10.000.000.
- Οι αριθμοί που θα αποθηκεύονται στον πίνακα heap_min θα παράγονται με μια **γεννήτρια τυχαίων αριθμών** (δείτε παρακάτω για λεπτομέρειες).
- Στην συνέχεια θα δεσμεύετε δυναμικά έναν δεύτερο πίνακα heap_max με ίδιο μέγεθος N και θα αντιγράφετε όλα τα στοιχεία του heap_min στον heap_max. Θα έχετε επομένως δύο πίνακες με το ίδιο μέγεθος και τα ίδια περιεχόμενα.
- Θα μετατρέψετε τον πίνακα heap_min σε σωρό ελαχίστων και τον πίνακα heap_max σε σωρό μεγίστων, χρησιμοποιώντας ως βάση το Πρόγραμμα 9.5, σελ. 428 του βιβλίου “Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++” και προσαρμόζοντας το κατάλληλα.
- Σε ότι αφορά τον σωρό ελαχίστων, θα εξάγετε τα δύο μικρότερα στοιχεία, θα τα προσθέτετε και το αποτέλεσμα θα το εισάγετε στον σωρό ελαχίστων. Αυτό θα επαναλαμβάνεται έως ότου ο σωρός αδειάσει, οπότε το τελευταίο στοιχείο που θα εξάγετε θα είναι το άθροισμα όλων των αριθμών κινητής υποδιαστολής.
- Θα επαναλάβετε τον παραπάνω αλγόριθμο για τον σωρό μεγίστων αυτή τη φορά, εξάγοντας σε κάθε βήμα τα δύο μεγαλύτερα στοιχεία και επανεισάγοντας στον σωρό το άθροισμα τους.
- Συγκρίνετε στο τέλος τα αποτελέσματα από τους δύο αυτούς τρόπους πρόσθεσης.

Η C++ προσφέρει ένα σύνολο κλάσεων και μεθόδων για την παραγωγή τυχαίων αριθμών (δείτε στο https://www.cplusplus.com/reference/random). Για να χρησιμοποιήσετε αυτές τις δυνατότητες:

- Συμπεριλάβετε τα αρχεία κεφαλίδας <random> και <functional> στο πρόγραμμα σας:

```
#include <random>
#include <functional>
```

- Ορίστε μια γεννήτρια τυχαίων αριθμών:

```
std::default_random_engine generator;
```

- Ορίστε την κατανομή για τους παραγόμενους τυχαίους αριθμούς. Με την παρακάτω δήλωση ζητάμε να χρησιμοποιηθεί η κατανομή x-squared και οι παραγόμενοι αριθμοί να είναι float:

```
std::chi_squared_distribution<float> my_distribution(0.5);
```

- Προς ευκολία χρήσης μπορείτε να «δέσετε» μαζί την γεννήτρια τυχαίων αριθμών με την επιθυμητή κατανομή:

```
auto random_num = std::bind(my_distribution, generator);
```

- Κάθε φορά που θα καλείτε την `random_num()` θα παράγεται ένας τυχαίος αριθμός με βάση την κατανομή που έχετε ορίσει. Αρχικοποιήστε τον πίνακα heap_min με τον παρακάτω κώδικα:

```
for (int i = 0; i < N; i++) {
      heap_min[i] = random_num() * 10000.0;
}
```

Παραδοτέα

Θα πρέπει να παραδοθεί ο πηγαίος κώδικας μαζί με τον εκτελέσιμο. Ιδιαίτερη βαρύτητα θα πρέπει να δοθεί στη σωστή τεκμηρίωση των προγραμμάτων σας. Θα πρέπει λοιπόν ο κώδικας σας να συνοδεύεται από ξεχωριστό κείμενο που θα παρέχει λεπτομερή περιγραφή των τεχνικών σας. Επίσης, εντός του πηγαίου κώδικα θα πρέπει να συμπεριληφθούν «πυκνά» **σχόλια ουσίας**. Η παράδοση των εργασιών θα γίνει μέσω του gunet.

Η εργασία μπορεί να εκπονηθεί από **ομάδα μέχρι δύο ατόμων αυστηρώς**.

Προθεσμία Παράδοσης

Παρασκευή 13 Μαΐου 2022