HY240: Δομές Δεδομένων
Χειμερινό Εξάμηνο – Ακαδημαϊκό Έτος 2019-20
Παναγιώτα Φατούρου

2ο Σετ Ασκήσεων

Ημερομηνία Παράδοσης: Τρίτη, 29 Οκτωβρίου 2018
Τρόπος Παράδοσης: Οι ασκήσεις μπορούν να παραδοθούν είτε σε ηλεκτρονική μορφή χρησιμοποιώντας το πρόγραμμα turnin (δείτε οδηγίες στην ιστοσελίδα του μαθήματος), είτε στους βοηθούς του μαθήματος στο εργαστήριο των μεταπτυχιακών, την Τρίτη, 29 Οκτωβρίου 2019, ώρα 15:00-16:00. Ασκήσεις που παραδίδονται μετά τις 16:00 της Τρίτης, 29/10/2019, θεωρούνται εκπρόθεσμες. Εκπρόθεσμες ασκήσεις γίνονται δεκτές σε ηλεκτρονική μορφή και η αποστολή τους πρέπει να γίνει χρησιμοποιώντας το πρόγραμμα turnin.

Άσκηση 1 [30 μονάδες]

α. Θεωρήστε μια απλά συνδεδεμένη λίστα, κάθε κόμβος της οποίας είναι ένα struct με πεδία έναν ακέραιο num και ένα δείκτη next στο επόμενο στοιχείο της λίστας. Έστω ότι L είναι ένας δείκτης στο πρώτο στοιχείο της λίστας. Παρουσιάστε συνάρτηση, η οποία θα επιστρέφει την τιμή του πεδίου num του δεύτερου από το τέλος κόμβου της λίστας. Η συνάρτησή σας θα πρέπει να διασχίζει τη λίστα μια μόνο φορά και άρα η χρονική της πολυπλοκότητα θα είναι O(n), όπου n είναι το πλήθος των στοιχείων της λίστας. Αν η λίστα είναι κενή ή περιέχει ένα μόνο στοιχείο, η συνάρτησή σας θα πρέπει να επιστρέφει έναν κωδικό λάθους (ERROR). [10M]

β. Θεωρήστε μια απλά συνδεδεμένη λίστα, κάθε κόμβος της οποίας είναι ένα struct με πεδία έναν ακέραιο num και ένα δείκτη next στο επόμενο στοιχείο της λίστας. Έστω ότι L είναι ένας δείκτης στο πρώτο στοιχείο της λίστας. Παρουσιάστε **αναδρομική** συνάρτηση που θα διαβάζει το πεδίο num του τελευταίου κόμβου, v, της λίστας και θα επιστρέφει δείκτη στον κόμβο που βρίσκεται σε απόσταση k από τον κόμβο v, όπου k είναι η τιμή του πεδίου num του v (δηλαδή η συνάρτηση θα πρέπει να επιστρέφει δείκτη σ' εκείνο τον κόμβο u της λίστας για τον οποίο ισχύει ότι μεταξύ του u και του v μεσολαβούν k-1 κόμβοι στη λίστα). Ποια είναι η χρονική και η χωρική πολυπλοκότητα της συνάρτησης που σχεδιάσατε; [10M]

Σημείωση: Μπορείτε να χρησιμοποιήσετε μεταβλητές static ή global για να λύσετε το ερώτημα αυτό.

γ. Παρουσιάστε **μη-αναδρομική** συνάρτηση που θα επιλύει το πρόβλημα που περιγράφεται στο ερώτημα β., θεωρώντας ότι η λίστα είναι **διπλά-συνδεδεμένη**. [10M]

Σημείωση: Δεν απαιτείται να μελετήσετε τη χρονική και χωρική πολυπλοκότητα σε αυτό το ερώτημα.

Άσκηση 2 [45 μονάδες]

α. Έστω ότι μια βιβλιοθήκη παρέχει πρόσβαση σε ουρές. Η βιβλιοθήκη σας επιτρέπει να ορίσετε μια ουρά και να καλέσετε τις πέντε βασικές λειτουργίες σ' αυτή. Για παράδειγμα, ο ορισμός μιας ουράς γίνεται γράφοντας "Queue Q;", ενώ υποστηρίζονται οι ακόλουθες λειτουργίες: (1) void MakeEmptyQueue(Queue Q), (2) boolean IsEmptyQueue(Queue Q), (3) Type Dequeue(Queue Q) και (4) void Enqueue(Queue Q, Type x), (5) Type Front(Queue Q).

Παρουσιάστε αλγόριθμο που θα υλοποιεί μια στοίβα, χρησιμοποιώντας την παραπάνω βιβλιοθήκη (δηλαδή χρησιμοποιώντας μόνο ουρές και καμία άλλη βοηθητική δομή). [15M]

Σημείωση: Θα πρέπει να παρουσιάσετε ψευδοκώδικα που να υλοποιεί τις συναρτήσεις MakeEmptyStack(), IsEmptyStack(), Top(), Push() και Pop() της στοίβας χρησιμοποιώντας μόνο ουρές και τις 5 συναρτήσεις που υποστηρίζει η βιβλιοθήκη για ουρές. Μπορείτε επίσης να χρησιμοποιήσετε ακέραιες μεταβλητές (π.χ. μετρητές, κλπ.). Δεν επιτρέπεται να κάνετε καμία υπόθεση για τον τρόπο (στατικός ή δυναμικός) που είναι υλοποιημένες οι ουρές στη βιβλιοθήκη και άρα δεν μπορείτε να χρησιμοποιήσετε δείκτες ή πίνακες στη λύση σας. Ο ρόλος σας σε αυτήν την άσκηση είναι του χρήστη της βιβλιοθήκης.

β. Μια ουρά δύο άκρων (double-ended queue) είναι μια λίστα στην οποία στοιχεία μπορούν να εισάγονται και να εξάγονται και από τα δύο άκρα της. Η ουρά δύο άκρων υποστηρίζει τις εξής λειτουργίες:

i. DQueue MakeEmptyDQueue(): δημιουργεί μια άδεια ουρά δύο άκρων.

ii. boolean IsEmptyDQueue(DQueue Q): επιστρέφει TRUE αν η ουρά είναι άδεια και FALSE διαφορετικά.

iii. void EnqueueFront(DQueue Q, Type x): εισάγει ένα στοιχείο στην αρχή της ουράς.

iv. void EnqueueBack(DQueue Q, Type x): εισάγει ένα στοιχείο στο τέλος της ουράς.

v. Type DequeueFront(DQueue Q): εξάγει ένα στοιχείο από την αρχή της ουράς.

vi. Type DequeueBack(DQueue Q): εξάγει ένα στοιχείο από το τέλος της ουράς.

Παρουσιάστε τόσο στατική, όσο και δυναμική υλοποίηση μιας διπλής ουράς. [30M]

Σημείωση: Ο ρόλος σας σ' αυτήν την άσκηση είναι του προγραμματιστή μιας βιβλιοθήκης που παρέχει ουρές δύο άκρων. Επομένως, θα πρέπει να παρουσιάσετε ψευδοκώδικα (στη μορφή που παρουσιάζεται για τις ουρές και τις στοίβες στις διαφάνειες του μαθήματος), για τις 6 λειτουργίες που αναφέρονται παραπάνω. Σημειώνεται ότι για κάθε τέτοια λειτουργία, χρειάζεται να παρουσιάσετε δύο υλοποιήσεις, μια στην οποία η ουρά δύο άκρων υλοποιείται με στατικό τρόπο και μια στην οποία υλοποιείται με δυναμικό τρόπο.

Άσκηση 3 [25 Μονάδες]

Θεωρήστε μια λίστα L της οποίας τα στοιχεία έχουν την εξής μορφή:

```
struct node{
        int  key;                   // ακέραιος που χαρακτηρίζει μοναδικά το στοιχείο
        int category;               // κατηγορία στην οποία ανήκει το στοιχείο
        struct node *next;          // δείκτης στο επόμενο στοιχείο της λίστας
        struct node *cnext;         // δείκτης που δείχνει στο πρώτο στοιχείο της επόμενης
                                    // αλυσίδας στη λίστα
}
```

Τα στοιχεία της λίστας είναι ταξινομημένα ως προς το πεδίο category και επομένως συναντάμε πρώτα στη λίστα τα στοιχεία που ανήκουν στην κατηγορία 1 (category == 1), στη συνέχεια εκείνα που ανήκουν στην κατηγορία 2 (category == 2), κ.ο.κ. Επομένως, τα στοιχεία κάθε κατηγορίας σχηματίζουν μια υπο-λίστα μέσα στη λίστα, που λέγεται *αλυσίδα*. Ο δείκτης cnext κάθε στοιχείου μιας αλυσίδας δείχνει στο πρώτο στοιχείο της επόμενης αλυσίδας στη λίστα.

Ζητούνται οι πιο αποδοτικοί αλγόριθμοι που μπορείτε να σχεδιάσετε για τις ακόλουθες συναρτήσεις:

i. Insert(L, <key>, <category>): Εισαγωγή στη λίστα ενός νέου στοιχείου με αναγνωριστικό <key>, το οποίο ανήκει στην κατηγορία <category>. Ο αλγόριθμός σας θα πρέπει να έχει χρονική πολυπλοκότητα O(n+m), όπου n είναι ο συνολικός αριθμός αλυσίδων στην L και m είναι το πλήθος

των στοιχείων στην αλυσίδα που προηγείται της αλυσίδας της κατηγορίας <category> στη λίστα. [10M]

ii. Delete(L, key, category): Διαγραφή του στοιχείου με κλειδί key από τη λίστα, δοθέντος ότι το κλειδί ανήκει στην κατηγορία category. Ο αλγόριθμός σας θα πρέπει να έχει χρονική πολυπλοκότητα O(n+m), όπου n είναι ο συνολικός αριθμός αλυσίδων στην L και m είναι το μέγιστο πλήθος στοιχείων μεταξύ της αλυσίδας της κατηγορίας <category> και εκείνης που προηγείται αυτής στη λίστα. [10M]

iii. CountChains(L): Επιστρέφει τον αριθμό των διαφορετικών αλυσίδων στη λίστα L. Ο αλγόριθμός σας θα πρέπει να έχει χρονική πολυπλοκότητα O(n), όπου n είναι ο συνολικός αριθμός αλυσίδων στην L. [5M]

Σημείωση: Μπορείτε να θεωρήσετε ότι η κάθε αλυσίδα είναι ταξινομημένη ή όχι ως προς το πεδίο key. Και οι δύο λύσεις θα θεωρηθούν σωστές.