HY240: Δομές Δεδομένων
Χειμερινό Εξάμηνο – Ακαδημαϊκό Έτος 2009-10
Διδάσκουσα: Παναγιώτα Φατούρου

2η Σειρά Θεωρητικών Ασκήσεων

Ημερομηνία Παράδοσης: Δευτέρα, 26 Οκτωβρίου 2009, ώρα 19:00-20:00

Τρόπος Παράδοσης: Οι ασκήσεις μπορούν να παραδοθούν είτε σε ηλεκτρονική μορφή χρησιμοποιώντας το πρόγραμμα submit (δείτε οδηγίες στην ιστοσελίδα του μαθήματος), είτε στους βοηθούς του μαθήματος τη Δευτέρα, 26 Οκτωβρίου 2009 (ώρα 19:00-20:00). Οι φοιτητές μπορούν να παραδίδουν τις ασκήσεις τους και νωρίτερα από αυτήν την ημερομηνία αλλά δικαιολογίες όπως «ήρθα την τάδε μέρα και δεν ήσασταν στο γραφείο σας» δεν θα γίνουν δεκτές. Όσοι από τους φοιτητές δεν έχουν καταφέρει να παραδώσουν τις ασκήσεις τους πριν τις 26/10 θα πρέπει να τις παραδώσουν στους βοηθούς του μαθήματος την προκαθορισμένη μέρα και ώρα. Εκπρόθεσμες ασκήσεις γίνονται δεκτές σε ηλεκτρονική μορφή και η αποστολή τους πρέπει να γίνει χρησιμοποιώντας το πρόγραμμα submit. Εναλλακτικά, εκπρόθεσμες ασκήσεις μπορούν να παραδοθούν στη διδάσκουσα του μαθήματος.

ΠΡΟΣΟΧΗ: Παραδίδουμε τις ασκήσεις στο βοηθό του μαθήματος ή στη διδάσκουσα ή χρησιμοποιούμε το πρόγραμμα submit. Δεν τοποθετούμε σε καμία περίπτωση τις ασκήσεις σε κουτιά που βρίσκονται στο υπόγειο των λευκών κτιρίων (ακόμη και αν σε αυτά αναγράφονται πληροφορίες που φαίνονται σχετικές με το μάθημα και την τρέχουσα σειρά ασκήσεων), αφού αυτό δεν εξασφαλίζει την ασφάλεια τους. Ασκήσεις που θα βρεθούν σε κουτιά δεν θα γίνουν δεκτές.

ΑΣΚΗΣΕΙΣ

1. Δίνεται ο ακόλουθος αναδρομικός αλγόριθμος (FindMinMax()) που υπολογίζει και το ελάχιστο και το μέγιστο στοιχείο του πίνακα T. Μετά την εκτέλεση του αλγορίθμου το ελάχιστο στοιχείο του πίνακα βρίσκεται στη θέση T[a] ενώ το μέγιστο στοιχείο βρίσκεται στη θέση T[b].

```
void FindMinMax(table T, int a, int b) {
    int  middle;
    if (b-a==0) return;
    middle = κάτω ακέραιο μέρος{(a+b)/2};
    for j = a to middle do  {
            if (T[j] > T[j+middle-a+1]) then
                    swap(T[j],T[j+middle-a+1]);
    }
    FindMinMax(T, a, middle);
    FindMinMax(T, middle+1,b);
}
```

α. Παρουσιάστε σύντομη περιγραφή του τρόπου λειτουργίας του αλγορίθμου.

β. Ιχνηλατίστε την FindMinMax(T,1,8) για την περίπτωση που T= [8,3,25,32,15,7,20,1]. Πρέπει να παρουσιαστούν όλες οι αναδρομικές κλήσεις της FindMinMax() με τη σειρά που καλούνται καθώς

και οι τιμές των παραμέτρων T, a, b σε κάθε κλήση. Πρέπει επίσης να παρουσιαστεί ο χώρος στη μνήμη που κατανέμεται για την εκτέλεση των αναδρομικών κλήσεων της FindMinMax().

γ. Παρουσιάστε αναδρομική σχέση που να περιγράφει τη χρονική πολυπλοκότητα T(n) της FindMinMax() για την περίπτωση που $n = 2^k$, για κάποιο k (δηλαδή για την περίπτωση που το n είναι μια δύναμη του 2). Τι τάξης είναι η πολυπλοκότητα της FindMinMax(); Αποδείξτε τον ισχυρισμό σας.

2. Έστω T(n) το πλήθος των φορών που θα εκτελεστεί η εντολή "x=x+1" στους παρακάτω αλγόριθμους. Ποια είναι η τάξη της συνάρτησης T(n) για κάθε έναν από τους τρεις αλγορίθμους που παρουσιάζονται στη συνέχεια, όπου sqrt(n) είναι η τετραγωνική ρίζα του n;

i.
```
procedure Peculiar(int n) {
    for j = 1 to sqrt(n) do
        for k = 1 to n do
            if (k mod 2 == 0) then x = x+1;
}
```

ii.
```
procedure Puzzle(int n) {
    for j = 1 to n do
        for k = j to n^2 do
            for m = sqrt(n)+1 to 2*sqrt(n) do x = x+1;
}
```

iii.
```
procedure Mystery(int n) {
    for i = 1 to sqrt(n) do
        for j=sqrt(n)+1 to n do
            for k = n-10 to n do x = x+1;
}
```

3. Έστω ότι μια βιβλιοθήκη σας παρέχει πρόσβαση σε στοίβες και ουρές ακεραίων. Η βιβλιοθήκη σας επιτρέπει να ορίσετε μια στοίβα (ή μια ουρά) και να καλέσετε τις 5 βασικές λειτουργίες σε αυτή. Για παράδειγμα, ο ορισμός μιας στοίβας (ή μιας ουράς) γίνεται γράφοντας: Stack S1; (αντίστοιχα, Queue Q1;). Για τη στοίβα υποστηρίζονται οι εξής λειτουργίες:

- void MakeEmptyStack(stack S);
- int IsEmptyStack(stack S);
- int Top(Stack S);
- int Pop(Stack S) και
- void Push(Stack S, int x).

Αντίστοιχα, για την ουρά υποστηρίζονται οι λειτουργίες:

- void MakeEmptyQueue(Queue Q);
- int IsEmptyQueue(Queue Q);
- int Front(Queue Q);
- int Dequeue(Queue Q);
- void Enqueue(Queue Q, int x).

Έστω ότι θέλετε να δημιουργήσετε ένα πρόγραμμα το οποίο απαιτεί επιπρόσθετα των παραπάνω λειτουργιών και την εκτέλεση των ακόλουθων λειτουργιών:

α) void PrintStack(Stack S): εκτύπωση όλων των στοιχείων της στοίβας S. Η εκτέλεση της PrintStack() δεν θα πρέπει να επηρεάζει τη μορφή της στοίβας (δηλαδή η στοίβα θα πρέπει να περιέχει τα ίδια στοιχεία και με την ίδια σειρά πριν και μετά την εκτέλεση της PrintStack()).

β) Queue MergeQueues(Queue Q1, Queue Q2): Δημιουργία νέας ουράς η οποία θα περιέχει όλα τα στοιχεία των δύο ουρών και για την οποία θα ισχύει το εξής, $\forall$ k >= 0, το (2k+1)-οστό στοιχείο της νέας ουράς είναι το (k+1)-οστό στοιχείο της ουράς Q1 ενώ το (2k+2)-οστό στοιχείο της νέας ουράς είναι το (k+1)-οστό στοιχείο της ουράς Q2. Αν μία από τις δυο ουρές έχει περισσότερα στοιχεία από την άλλη, τα στοιχεία αυτά τοποθετούνται στο τέλος της νέας ουράς ως έχουν. Για παράδειγμα έστω ότι η ουρά Q1 περιέχει τα στοιχεία 5, 15, 1, 4, 9 (με αυτή τη σειρά και το στοιχείο 5 είναι το πρώτο) ενώ η Q2 περιέχει τα στοιχεία 30, 6, 12, 7, 11, 16, 22, 8 (με αυτή τη σειρά και το στοιχείο 30 είναι το πρώτο). Τότε η νέα ουρά θα πρέπει να περιέχει 5, 30, 15, 6, 1, 12, 4, 7, 9, 11, 16, 22, 8. Προσέξτε ότι το $1^{ο}$ στοιχείο της νέας ουράς είναι το $1^{ο}$ στοιχείο της Q1 ενώ το $2^{ο}$ είναι το $1^{ο}$ της Q2. Επίσης, το $3^{ο}$ στοιχείο της νέας ουράς είναι το $2^{ο}$ της Q1, ενώ το $4^{ο}$ είναι το $2^{ο}$ της Q2.

Η εκτέλεση της MergeQueues() δεν θα πρέπει να επηρεάζει τη μορφή των ουρών Q1 και Q2 (δηλαδή οι ουρές Q1, Q2 θα πρέπει να περιέχουν τα ίδια στοιχεία και με την ίδια σειρά πριν και μετά την εκτέλεση της MergeQueues()).

Παρουσιάστε ψευδο-κώδικα που θα υλοποιεί κάθε μια από τις παραπάνω λειτουργίες.

Υπόδειξη: Επιτρέπεται να χρησιμοποιήσετε μια ή περισσότερες extra στοίβες και ουρές προκειμένου να υλοποιήσετε τις παραπάνω λειτουργίες.