Άσκηση 2

Ημερομηνία Παράδοσης: 12 Απριλίου 2008

Σημειώσεις:

1. Στις απαντήσεις που θα παραδώσετε σημειώστε στην πρώτη σελίδα το ονοματεπώνυμό σας, τον αριθμό μητρώου σας και το τμήμα σας.
2. Οι ασκήσεις πρέπει να γίνουν ατομικά. Οποιαδήποτε μορφή αντιγραφής απαγορεύεται.
3. Η παρούσα άσκηση πρέπει να παραδοθεί το αργότερο μέχρι την αρχή του μαθήματος της 12ης Απριλίου, δηλαδή μέχρι τις 11:15. Καθυστερημένες ασκήσεις δε θα γίνουν δεκτές.
4. Σε περίπτωση που έχετε ερωτήσεις στείλτε email στην ηλεκτρονική λίστα του μαθήματος: hy240-list@tem.uoc.gr

Πρόβλημα 1 [10 μονάδες] Θεωρήστε μια στοίβα ακεραίων αριθμών s της οποίας ο ΑΤΔ (Αφηρημένος Τύπος Δεδομένων) δίδεται από τις παρακάτω συναρτήσεις:

```
int top(const StackADT* s);
void push(StackADT* s, int i);
void pop(StackADT* s);
```

Ποιά η μορφή της στοίβας μετά από κάθε μία από τις ακόλουθες πράξεις;
`push(s,6), push(s,7), top(s), pop(s), push(s,15), pop(s), pop(s), push(s,35), push(s,-12), push(s,4), top(s), pop(s), push(s,1), push(s,2), push(s,17), pop(s), top(s), pop(s), pop(s).`

Πρόβλημα 2 [10 μονάδες] Θεωρήστε μια ουρά ακεραίων αριθμών q της οποίας ο ΑΤΔ δίδεται από τις παρακάτω συναρτήσεις:

```
int front(const QueueADT* s);
void enqueue(QueueADT* s, int i);
void dequeue(QueueADT* s);
```

Ποιά η μορφή της ουράς μετά από κάθε μία από τις ακόλουθες πράξεις;
`enqueue(s,6), enqueue(s,7), front(s), dequeue(s), enqueue(s,15), dequeue(s), dequeue(s), enqueue(s,35), enqueue(s,-12), enqueue(s,4), front(s), dequeue(s), enqueue(s,1), enqueue(s,2), enqueue(s,17), dequeue(s), dequeue(s).`

Πρόβλημα 3 [30 μονάδες] Θεωρήστε τις παρακάτω δύο δομές (structs) της C που αναφέρονται σε κόμβο απλής λίστας και στην απλή λίστα για ακεραίους αριθμούς.

```
typedef struct SimpleListNode {
  SimpleListNode* next;
  int el;
} SimpleListNode;

typedef struct SimpleList {
  SimpleListNode* head;
  unsigned int size;
} SimpleList;
```

Θεωρήστε επίσης ότι σας δίδονται οι παρακάτω συναρτήσεις του ΑΤΔ της ως άνω απλής λίστας:

```
unsigned int size(const SimpleList* l);
bool isEmpty(const SimpleList* l);
SimpleListNode* after(const SimpleList* l, const SimpleListNode* p);
int element(const SimpleList* l, const SimpleListNode* p);
SimpleListNode* first(const SimpleList* l);
SimpleListNode* insertAfter(SimpleList* l, SimpleListNode* p, int i);
SimpleListNode* insertFirst(SimpleList* l, int i);
```

Όλες οι παραπάνω συναρτήσεις έχουν χρονικό κόστος $O(1)$. Σε όλες τις συναρτήσεις ο δείκτης σε λίστα δεν επιτρέπεται να είναι `NULL`. Στις `after`, `element` και `insertAfter` ο δείκτης σε κόμβο της λίστας δεν επιτρέπεται να είναι `NULL`. Η συνάρτηση `after` επιστρέφει `NULL` αν ο δείκτης `p` είναι ο τελευταίος κόμβος της λίστας. Τέλος, η `first` δεν επιτρέπεται να κληθεί για μια άδεια λίστα.

Γράψτε κώδικα σε C για τη συνάρτηση

```
SimpleList* merge(SimpleList* L1, SimpleList* L2);
```

η οποία δεδομένων δύο απλών λιστών $L1$ και $L2$ επιστρέφει μία νέα λίστα L που είναι η συνένωση των δύο λιστών. Η συνένωση L των $L1$ και $L2$ περιέχει όλους τους κόμβους της $L1$ (στη σειρά που ήταν πριν) και στην συνέχεια όλους τους κόμβους της λίστας $L2$ (επίσης στη σειρά που ήταν πριν).

Η υλοποίησή σας για τη `merge` επιτρέπεται να χρησιμοποιεί μόνο τις συναρτήσεις του ΑΤΔ. Επιτρέπεται να χρησιμοποιήσετε σταθερή ποσότητα επιπλέον χώρου (πέραν του χώρου των δύο λιστών), ενώ είναι δυνατόν να τροποποιήσετε τις λίστες $L1$ ή/και $L2$ αν κάτι τέτοιο χρειάζεται. Εξηγήστε με λόγια τον αλγόριθμο που υλοποιήσατε και υπολογίστε το χρονικό κόστος του αλγορίθμου σας, συναρτήσει του πλήθους n των στοιχείων της $L1$ και του πλήθους m των στοιχείων της $L2$.

Θεωρήστε τώρα ότι, πέραν των συναρτήσεων του ΑΤΔ, επιτρέπεται να έχετε άμεση πρόσβαση στο εσωτερικό των δομών `SimpleListNode` και `SimpleList`. Υλοποιήστε εκ νέου τη `merge` με τέτοιο τρόπο ώστε ο αλγόριθμός να τρέχει σε χρόνο $O(n)$. Εξηγήστε με λόγια τον αλγόριθμο που υλοποιήσατε και γιατί επιτυγχάνει το ζητούμενο χρόνο.

Αν επιτρέπεται στην L τα στοιχεία των $L1$ και $L2$ να εμφανίζονται με οποιαδήποτε σειρά, δείξτε πώς μπορείτε να υλοποιήσετε τη `merge` ώστε να τρέχει σε χρόνο $O(\min\{n, m\})$. Περιγράψτε τον αλγόριθμό σας και δώστε υλοποίηση.

Πρόβλημα 4 [25 μονάδες] Θεωρήστε διπλή λίστα ακεραίων αριθμών L για την οποία σας παρέχονται οι παρακάτω συναρτήσεις:

```
ListADT* new_double_list();
unsigned int size(const ListADT* l);
ListNodeADT* first(const ListADT* l);
ListNodeADT* last(const ListADT* l);
ListNodeADT* insertFirst(ListADT* l, int i);
int remove(ListADT* l, ListNodeADT* p);
```

Όλες οι παραπάνω συναρτήσεις έχουν χρονικό κόστος $O(1)$. Η συνάρτηση `new_double_list` επιστρέφει μία νέα άδεια διπλή λίστα. Σε όλες τις συναρτήσεις, ο δείκτης σε λίστα δεν επιτρέπεται να είναι `NULL`. Στην `remove` ο δείκτης σε κόμβο της λίστας δεν επιτρέπεται να είναι `NULL`. Τέλος, οι συναρτήσεις `first` και `last` δεν επιτρέπεται να κληθούν για μια άδεια λίστα.

Γράψτε κώδικα για μια νέα συνάρτηση

```
void invert(ListADT* l);
```

η οποία αντιστρέφει τη σειρά των στοιχείων στη λίστα L χρησιμοποιώντας *μόνο* τις συναρτήσεις που σας δίνονται παραπάνω. Ποιό το χρονικό κόστος του αλγορίθμου σας αν η L περιέχει n στοιχεία; Μπορείτε να υλοποιήσετε την `invert` χωρίς να χρησιμοποιήσετε τη συνάρτηση `last` που σας δίδεται παραπάνω; Αν ναι, δώστε την υλοποίησή σας και υπολογίστε το χρονικό κόστος του αλγορίθμου σας συναρτήσει του αριθμού n των στοιχείων της L. Αν όχι, εξηγήστε γιατί όχι.

Σημειώστε ότι η συνάρτηση `invert` δεν επιστρέφει μία καινούργια λίστα, αλλά αναδιατάσει τα στοιχεία της L. Μπορείτε να χρησιμοποιήσετε άλλες διπλές λίστες για βοηθητική αποθήκευση.

Πρόβλημα 5 [25 μονάδες] Θεωρήστε απλή λίστα L από στοιχεία τύπου `ListElement` για την οποία σας παρέχονται οι παρακάτω συναρτήσεις:

```
SimpleListADT* new_simple_list();
unsigned int size(const SimpleListADT* l);
SimpleListNodeADT* after(const SimpleListADT* l, const SimpleListNodeADT* p);
ListElement element(const SimpleListADT* l, const SimpleListNodeADT* p);
SimpleListNodeADT* first(const SimpleListADT* l);
SimpleListNodeADT* insertAfter(SimpleListADT* l, SimpleListNodeADT* p,
                               ListElement el);
SimpleListNodeADT* insertFirst(SimpleListADT* l, ListElement el);
```

Όλες οι παραπάνω συναρτήσεις έχουν χρονικό κόστος $O(1)$. Η συνάρτηση `new_simple_list` επιστρέφει μία νέα άδεια απλή λίστα. Σε όλες τις συναρτήσεις ο δείκτης σε λίστα δεν επιτρέπεται να

είναι `NULL`. Στις `after`, `element` και `insertAfter` ο δείκτης σε κόμβο της λίστας δεν επιτρέπεται να είναι `NULL`. Η συνάρτηση `after` επιστρέφει `NULL` αν ο δείκτης `p` είναι ο τελευταίος κόμβος της λίστας. Τέλος, η `first` δεν επιτρέπεται να κληθεί για μια άδεια λίστα.

Σας δίνεται επίσης και η συνάρτηση σύγκρισης

```
bool less_than(const ListElement* a, const ListElement* b);
```

η οποία επιστρέφει `true` αν το στοιχείο `a` είναι μικρότερο του `b`, και `false` αν το `a` είναι μεγαλύτερο ή ίσο του `b`.

Θεωρήστε ότι έχουμε δύο απλές λίστες $L1$ και $L2$ των οποίων τα στοιχεία είναι ταξινομημένα σε αύξουσα σειρά. Ζητείται να υλοποιήσετε τη συνάρτηση

```
SimpleListADT* merge_sorted(const SimpleListADT* L1,
                            const SimpleListADT* L2);
```

που επιστρέφει μία νέα απλή λίστα L η οποία περιέχει τα στοιχεία των $L1$ και $L2$ ταξινομημένα σε αύξουσα σειρά, με μία επιπλέον συνθήκη: αν δύο στοιχεία της $L1$ ή της $L2$ είναι ίσα τότε η σειρά τους στην L είναι όπως και την $L1$ ή την $L2$, αντίστοιχα, ενώ αν ένα στοιχείο της $L1$ είναι ίσο με ένα στοιχείο της $L2$, τότε το στοιχείο της $L1$ προηγείται του στοιχείου της $L2$ στην L.

Ποιό το κόστος του αλγορίθμου σας συναρτήσει των n και m, όπου n είναι το πλήθος των στοιχείων της $L1$ και m το πλήθος των στοιχείων της $L2$;

Σύνολο μονάδων: 100