Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Δομές Δεδομένων και Αλγόριθμοι

1η άσκηση

Ημερομηνία παράδοσης: 17 Απριλίου 2024, 18:00
Ο τρόπος υποβολής θα ανακοινωθεί

Ομάδες έως 2 άτομα. Για τον τρόπο δήλωσης ομάδας, θα υπάρξει σχετική ανακοίνωση
Θα γίνει προφορική εξέταση της κάθε ομάδας. Θα υπάρξει σχετική ανακοίνωση για το πρόγραμμα.

Τελευταία αλλαγή στην εκφώνηση (βλέπε τέλος εγγράφου): 04/04/2024

Σκοπός της άσκησης είναι η σύγκριση διαφορετικών υλοποιήσεων σε Δομές Λίστας στην κεντρική μνήμη. Συγκεκριμένα θα υλοποιηθούν 6 διαφορετικές εκδοχές Λίστας και θα γίνουν μετρήσεις σε συγκεκριμένες πράξεις σε κάθε μία από αυτές.

Στη συνέχεια γίνεται αναφορά σε στοιχεία της γλώσσας Java (κλάσεις, interfaces, packages κλπ). Εφόσον χρησιμοποιήσετε άλλη γλώσσα προγραμματισμού, θα πρέπει να χρησιμοποιήσετε τα ίδια ονόματα μεθόδων, με τους ίδιους τύπους και ίδιο πλήθος παραμέτρων.

Στις δομές αποθηκεύουμε στοιχεία που υλοποιούν το Interface org.tuc.Element. Τα κλειδιά των Elements που χειρίζεται η κάθε δομή, είναι ένας ακέραιος int. Τα κλειδιά των Elements σε μία δομή, δεν είναι μοναδικά. Μπορεί δηλαδή μία λίστα να περιέχει πολλά Elements με το ίδιο κλειδί.

```
package org.tuc;
/**
* Data stored in the data structure with int keys
*/
public interface Element {
	/**
	 * Returns the key of this element
	 * @return the key of the element
	 */
	public int getKey();
}
```

Και οι 6 υλοποιήσεις της δομής θα πρέπει να υλοποιούν το παρακάτω Java Interface.

```java
package org.tuc;
/**
* An interface describing a list data structure storing element using int as key.
* The list has a state in form of a pointer to a current element
*/
public interface List {
	/**
	 * Inserts an element into the list
	 * @param element
	 * @return true if the insertion was successful. Otherwise false
	 */
	public boolean insert(Element element);

	/**
	 * Deletes the first element found in the list with key equal to the given key
	 * @param key
	 * @return true if a deletion is made. Otherwise false
	 */
	public boolean delete(int key);

	/**
	 * Returns the first element found in the list with key equal to the given key
	 * @param key
	 * @return The first matched element, otherwise null
	 */
	public Element search(int key);
}
```

Οι 6 υλοποιήσεις

Υλοποιήστε τη δομή δεδομένων λίστας (δηλαδή κλάσεις που υλοποιούν τα παραπάνω interfaces) με τους παρακάτω 6 εναλλακτικούς τρόπους

Υλοποίηση 1a - δυναμική παραχώρηση μνήμης, μη ταξινομημένη

Κάθε Element αποθηκεύεται σε ένα στιγμιότυπο μίας κλάσης Node, η οποία παρέχει επιπλέον ένα δείκτη (java reference) προς το επόμενο Element. Πέρα από το head, διατηρείται μεταβλητή που δείχνει στο τελευταίο στοιχείο της λίστας (tail).

Όνομα κλάσης: DList implements List

Δεν υπάρχει περιορισμός στο συνολικό πλήθος των Elements που μπορεί να αποθηκεύσει η δομή (ο μόνος περιορισμός είναι η διαθέσιμη μνήμη του συστήματος).

- Το insert() ενός element προσθέτει ένα καινούργιο Node στο τέλος της λίστας.
- Το delete() ψάχνει να βρει ένα Element με κλειδί ίσο με το κλειδί που έχει δοθεί, και εφόσον βρεθεί, διαγράφει το Node στο οποίο βρίσκεται το Element. Εάν στη δομή υπάρχουν πολλά Elements με το ίδιο κλειδί, διαγράφει μόνο το πρώτο που θα βρεθεί.
- Το search ψάχνει να βρει ένα Element με κλειδί ίσο με το κλειδί που έχει δοθεί, και εφόσον βρεθεί, επιστρέφει αυτό το element.

Υλοποίηση 1b - δυναμική παραχώρηση μνήμης, ταξινομημένη

Όνομα κλάσης: SDList extends Dlist

Τα insert(), delete() και search() είναι ίδια με το 1α, αλλά τα Elements είναι πλέον ταξινομημένα βάσει του κλειδιού. Υπάρχουν δηλαδή οι εξής διαφοροποιήσεις:

- Το insert() ενός element προσθέτει ένα καινούργιο Node στη σωστή θέση της λίστας ώστε να συνεχίσει να είναι ταξινομημένη. Πρακτικά η insert είναι overridden μέθοδος της DList
- Το delete() δεν έχει διαφορά
- Το search() δεν έχει διαφορά

Υλοποίηση 2a - χρήση στατικού δισδιάστατου πίνακα, μη ταξινομημένη

Η υλοποίηση ακολουθεί τη μεθοδολογία που περιγράφεται στις διαφάνειες “Συνδεδεμένες Λίστες (Linked Lists)”, στην έκδοση “Array Implementation (2)”, και παρουσιάστηκε στο 1ο φροντιστήριο.

Όνομα κλάσης: AAList implements List

Κάθε Node αντιστοιχεί σε “μία γραμμή” ενός δισδιάστατου πίνακα. Παρατηρείστε ότι στην πραγματικότητα, όπως και στις διαφάνειες, χρησιμοποιούμε ένα array από Elements, και ένα array από ints του ίδιου μεγέθους.

Διατηρείται μεταβλητή που δείχνει στο τελευταίο στοιχείο της λίστας (tail).

Το μέγιστο πλήθος Elements καθορίζεται τη στιγμή δημιουργίας της δομής, και πρέπει να είναι τουλάχιστον όσα τα Elements που συνολικά θα εισάγετε στη δομή.

Το πρώτο κελί της κάθε γραμμής αποθηκεύει το Element. Στο δεύτερο κελί κάθε γραμμής αποθηκεύεται το index (αριθμός γραμμής) όπου βρίσκεται το επόμενο Element.

Στις γραμμές που δεν έχουν χρησιμοποιηθεί ακόμα (είναι δηλαδή ελεύθερες), το πρώτο στοιχείο δεν έχει χρήση, ενώ το δεύτερο στοιχείο δείχνει κάθε φορά στην επόμενη ελεύθερη γραμμή. Η πρώτη ελεύθερη γραμμή αποθηκεύεται σε μεταβλητή και δημιουργείται έτσι μία ακολουθία από ελεύθερες γραμμές υπό μορφή στίβας (stack).

Τα insert(), delete() και search() λειτουργούν όπως στο 1a.

Υλοποίηση 2b - χρήση στατικού δισδιάστατου πίνακα, ταξινομημένη

Όπως και το 1b, αλλά με υλοποίηση λίστας όπως το 2a.

Όνομα κλάσης: SAAList extends AAlist

Υλοποίηση 3a - χρήση στατικού μονοδιάστατου πίνακα, μη ταξινομημένη

Τα Elements αποθηκεύονται σε ένα array από συνεχόμενα Elements. Το μέγιστο πλήθος Elements καθορίζεται τη στιγμή δημιουργίας της δομής. Διατηρείται μεταβλητή που δείχνει στο τελευταίο στοιχείο της λίστας (tail) και ουσιαστικά είναι το μέγεθος της λίστας.

Όνομα κλάσης: AList implements List

Τα insert(), delete() και search() είναι ίδια με το 1a, με τις εξής διαφοροποιήσεις:

- Το insert() δεν έχει διαφορά
- Το delete(), εφόσον διαγράφεται ένα element στο ενδιάμεσο της λίστας, μετακινεί όλα τα elements από δεξιά του, μία θέση προς τα αριστερά, ώστε να μην υπάρχει κενό
- Το search() δεν έχει διαφορά

Υλοποίηση 3b - χρήση στατικού μονοδιάστατου πίνακα, ταξινομημένη

Όνομα κλάσης: SAList extends Alist

Τα elements είναι πλέον ταξινομημένα. Επειδή υπάρχει η ταξινόμηση, και έχουμε άμεση πρόσβαση σε οποιαδήποτε θέση του array, μπορούμε να χρησιμοποιήσουμε binary search για τις υλοποιήσεις των insert(), delete() και search().

Υπάρχουν δηλαδή οι εξής διαφοροποιήσεις:

- Το insert() ενός element γίνεται σε σωστή θέση, έτσι ώστε να διατηρείται η ταξινόμηση. Εάν η θέση είναι στο ενδιάμεσο των ήδη υπαρχόντων elements, πρέπει όλα τα elements που βρίσκονται από δεξιά, να μετακινηθούν μία θέση δεξιά. Για την εύρεση του σημείου εισαγωγής, κάνουμε χρήση της binary search.
- Το delete() δεν έχει λειτουργική διαφορά με το 3a. Για την εύρεση του element προς διαγραφή, κάνουμε χρήση της binary search.
- Το search() δεν έχει διαφορά λειτουργική διαφορά 3a. Για την εύρεση του element κάνουμε χρήση της binary search.

Πείραμα - Μετρήσεις

Έχοντας ολοκληρώσει τις 6 υλοποιήσεις, θέλουμε να συγκρίνουμε την απόδοση της κάθε υλοποίησης σε συγκεκριμένες λειτουργίες της δομής. Για αυτό το σκοπό θα πάρουμε πειραματικές μετρήσεις για διάφορα μεγέθη εισόδου (πλήθος Elements στη δομή). Θα καταγράψουμε κάθε φορά το χρόνο εκτέλεσης σε Nanoseconds και το πλήθος πράξεων (συγκρίσεις, αναθέσεις κλπ) σαν συνάρτηση του πλήθους Elements στη δομή.

Οι λειτουργίες που μας ενδιαφέρει να αξιολογήσουμε είναι οι

- Α - εισαγωγή κλειδιού
- Β - διαγραφή κλειδιού
- Γ - αναζήτηση κλειδιού

Το πείραμα αποτελείται από τα παρακάτω βήματα, τα οποία θα επαναληφθούν για πλήθος στοιχείων N=30, 50, 100, 200, 500, 800, 1.000, 5.000, 10.000, 100.000 (τυχαία κλειδιά κάθε φορά)

1) Προετοιμασία
    a) δημιουργείτε ένα νέο στιγμιότυπο για κάθε μία από τις 6 υλοποιήσεις σας
    b) δημιουργείτε Ν τυχαίους αριθμούς (μη ταξινομημένους!) με τιμές από 1 έως 2*N και τους αποθηκεύετε σε ένα απλό array
    c) για κάθε έναν από τους Ν τυχαίους αριθμούς, δημιουργείτε ένα στιγμιότυπο Element και το εισάγετε σε κάθε μία από τις 6 δομές σας
2) Μετρήσεις
    a) παράγετε Κ* τυχαίους αριθμούς από 1 έως 2*N (δε χρειάζεται να είναι μοναδικοί). Για κάθε έναν εκτελείτε τη λειτουργία Α σε κάθε μία από τις 6 δομές σας. Καταγράφετε κάθε φορά το χρόνο εκτέλεσης σε nanosecond και το πλήθος πράξεων. Από τις Κ* εκτελέσεις της λειτουργίας Α, θα πάρετε **το μέσο όρο του χρόνου εκτέλεσης και το μέσο όρο των πράξεων** για μία εκτέλεση της λειτουργίας Α. Χρησιμοποιήστε 2 δεκαδικά ψηφία για τους μέσους όρους.

*

K=10 όταν N < 201

K=50 όταν N > 200 και N < 1001

K=100 όταν N > 1000

b) το ίδιο με το a) αλλά για τη λειτουργία Β
c) το ίδιο με το a) αλλά για τη λειτουργία Γ

Στο τέλος θα έχετε για κάθε πλήθος στοιχείων N, και για κάθε δομή, και για κάθε λειτουργία, έναν χρόνο εκτέλεσης και ένα πλήθος πράξεων. Συνολικά δηλαδή 360 μετρήσεις.

Κατά τη εκτέλεση της εφαρμογής σας, θα πρέπει χωρίς κανένα input από το χρήστη, να γίνονται όλες οι παραπάνω μετρήσεις, και να τυπώνονται στην οθόνη οι **παρακάτω 6 πίνακες**.

Λειτουργία Α ή Β ή Γ

	N=30	N=50	N=100	……	N=100000
1a	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ
1b	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ
2a	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ
2b	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ
3a	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ
3b	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ	μ.ο. πράξεων ανά Α/Β/Γ

Λειτουργία Α ή Β ή Γ

	N=30	N=50	N=100	……	N=100000
1a	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ
1b	μ.ο. χρόνου	μ.ο. χρόνου	μ.ο. χρόνου	μ.ο. χρόνου	μ.ο. χρόνου

	ανά Α/Β/Γ	ανά Α/Β/Γ	ανά Α/Β/Γ	ανά Α/Β/Γ	ανά Α/Β/Γ
2a	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ
2b	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ
3a	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ
3b	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ	μ.ο. χρόνου ανά Α/Β/Γ

Αναφορά

1) Κάντε μία σύντομη θεωρητική ανάλυση για την πολυπλοκότητα των αλγορίθμων αναζήτησης, εισαγωγής και διαγραφής για κάθε περίπτωση. Συγκεκριμένα, συμπληρώστε τον παρακάτω πίνακα, και για κάθε κελί δώστε μία σύντομη τεκμηρίωση.

	1a	1b	2a	2b	3a	3b
insert()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)
delete()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)
search()	O(??)	O(??)	O(??)	O(??)	O(??)	O(??)

2) Δημιουργήστε ένα γράφημα για το χρόνο εκτέλεσης και ένα ξεχωριστό για τον αριθμό πράξεων από τις πειραματικές μετρήσεις, για τις παρακάτω περιπτώσεις 3 (<u>συνολικά 6 γραφήματα</u>):

- search() 3b
- search() 2b
- insert() 2a

Τί μορφή έχουν οι καμπύλες (σταθερή, λογαριθμική, γραμμική, εκθετική κλπ); Συμφωνούν οι μορφές με τη θεωρητική ανάλυση που έγινε στο (1);

3) Απαντήστε σύντομα στις παρακάτω ερωτήσεις/θέματα

1. Ειδικά στις περιπτώσεις με μεγάλα Ν, συμβαδίζουν οι χρόνοι εκτέλεσης με το πλήθος πράξεων; Είναι αυτό αναμενόμενο από τη θεωρία;

2. Σας ζητούν να αλλάξετε τις υλοποιήσεις ώστε να επιβάλλεται να υπάρχουν μοναδικά κλειδιά στις λίστες. Ξαναδημιουργείστε τον πίνακα (1) με τις πολυπλοκότητες όπως θα περιμένατε να διαμορφωθούν σε αυτή την περίπτωση και αιτιολογείστε σύντομα τις διαφοροποιήσεις.
3. Μεταξύ των υλοποιήσεων 1a και 2a, και 1b και 2b, σε απόλυτους χρόνους, ποια περιμένετε να είναι πιο γρήγορη; Επιβεβαιώνεται αυτό από τις μετρήσεις;

Παραδοτέα

Ένα συμπιεσμένο zip αρχείο που περιέχει τον πηγαίο κώδικα (φάκελος src) και τη ζητούμενη αναφορά σε pdf (όχι zip μέσα στο zip!).

Γενικοί κανόνες για τον κώδικα και την αναφορά:

- Ο κώδικας περιέχει συνοπτικά σχόλια που εξηγούν την υλοποίηση.
- Εφόσον χρησιμοποιήσετε κώδικα ή στοιχεία από αλλού, αναφέρετε τις πηγές σας (πλήρης διεύθυνση σελίδας διαδικτύου, βιβλίο και συγκεκριμένο κεφάλαιο, κλπ). Εννοείται ότι πηγή σας δεν μπορεί να είναι η εργασία από συμφοιτητή σας.
- Οι αντιγραφές μηδενίζονται.
- Δώστε μεγάλη σημασία στην ποιότητα του κώδικα που θα στείλετε. Θα πρέπει να ακολουθεί τουλάχιστον τους παρακάτω κανόνες:
    - Χρησιμοποιείστε εύστοχα ονόματα μεταβλητών και μεθόδων έτσι ώστε να γίνεται εύκολα αντιληπτός ο λόγος και ο τρόπος χρήσης τους.
    - Γράψτε απλό και δομημένο κώδικα ώστε αυτός που τον διαβάζει να μπορεί να καταλάβει τα βήματα που ακολουθούνται για την υλοποίηση του κάθε προβλήματος.
    - Βάλτε εύστοχα σχόλια όπου χρειάζεται.
    - Κάντε το κώδικά σας παραμετρικό με χρήση μεταβλητών μέσω των οποίων θα μπορείτε να αλλάζετε κάποια μεγέθη (π.χ. τα μεγέθη για N, M κ.ο.κ.) χωρίς να χρειάζεται να αλλάζετε κάθε φορά και την αντίστοιχη υλοποίηση.

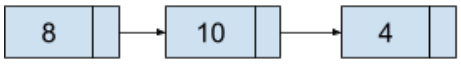
Χρήσιμα

Ενδεικτική μορφή δομών μετά από συγκεκριμένες πράξεις

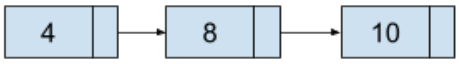
Στα παρακάτω διαγράμματα παρουσιάζονται ενδεικτικά οι δομές μετά από τις εξής πράξεις
insert(8)
insert(3)
insert(10)
delete(3)
insert(4)

Για τις στατικές δομές θεωρούμε μέγιστο πλήθος Elements = 5

1a



1b



2a

8	2
4	-1
10	1
	4
	-1

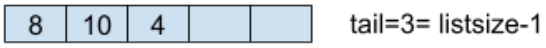
head=0
tail=1
nextfree=3

2b

8	2
4	0
10	-1
	4
	-1

head=1
tail=2
nextfree=3

3a



tail=3= listsize-1

3b

tail=3= listsize-1

Παραγωγή τυχαίων μοναδικών και μη μοναδικών ακεραίων

Για την παραγωγή μοναδικών numberOfNumbers ακεραίων μεταξύ minIntNumber και maxIntNumber μπορείτε να χρησιμοποιήσετε τον παρακάτω κώδικα.

```
java.util.Random randomGenerator = new java.util.Random();
int[] randomInts = randomGenerator.ints(minIntNumber,
maxIntNumber+1).distinct().limit(numberOfNumbers).toArray();
```

Για την παραγωγή μη μοναδικών numberOfNumbers ακεραίων μεταξύ minIntNumber και maxIntNumber μπορείτε να χρησιμοποιήσετε τον παρακάτω κώδικα.

```
java.util.Random randomGenerator = new java.util.Random();
int[] randomInts = randomGenerator.ints(minIntNumber,
maxIntNumber+1).limit(numberOfNumbers).toArray();
```

Για τη μέτρηση του χρόνου

Τρέχουσα ώρα σε nanosecond (long).

```
System.nanoTime()
```

Οπότε αρκεί να καλέσετε τη μέθοδο πριν την έναρξη της λειτουργίας που θέλετε να μετρήσετε το χρόνο, και να την ξανακαλέσετε μετά τον τερματισμό και να αφαιρέσετε τις τιμές μεταξύ τους.

Παραγωγή μέσων όρων

Στους μέσους όρους των μετρήσεων (χρόνου/πράξεων), θέλουμε αποτέλεσμα με δεκαδικά ψηφία. Στην java, όταν διαιρείται ένας ακέραιος με έναν άλλο ακέραιο, το αποτέλεσμα είναι πάλι ακέραιος, που δεν είναι το επιθυμητό. Για να παραχθεί αριθμός με δεκαδικά, αρκεί να γίνει cast ο αριθμητής σε τύπο με δεκαδικά. Π.χ.

```
long totalTime = 2548788751;
int numberOfRepetitions = 100;
float meanTimePerRepetition = (double) totalTime /
numberOfRepetitions;
```

Αλλαγές στην εκφώνηση

Σε σχέση με το αρχικό κείμενο της εκφώνησης, έγιναν οι εξής αλλαγές:

04/04/2024
Στο σχεδιάγραμμα για την 3α και 3β, διορθώθηκε ότι το tail = listsize-1 (αντί listsize, μιας και στην Java τα indexes των arrays ξεκινούν από το 0)

02/04/2024
Στην περιγραφή της 1b, διορθώθηκε η αναφορά σε override από overload για τη μέθοδο insert.
Στο σχεδιάγραμμα με τα παραδείγματα, προστέθηκε η μεταβλητή tail για τα 2a, 2b, 3a, 3b

30/03/2024
Σύντομη διευκρίνηση στο 1α, ότι γενικά στις λίστες, πέρα από το tail χρειάζεστε και μεταβλητή που δείχνει στην αρχή της λίστας.

28/03/2024
Στην ενότητα “Πείραμα - Μετρήσεις”, στην παράγραφο “2) Μετρήσεις”, αντί για 1000 τυχαίους κάθε φορά αριθμούς, χρειάζονται Κ αριθμοί όπου το Κ εξαρτάται από το πλήθος στοιχείων στη δομή Ν.