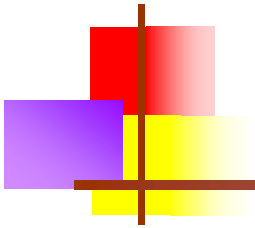
JDSL-Java Data Structures Library

Δομές Δεδομένων

Παπαγιαννόπουλος Δημήτριος

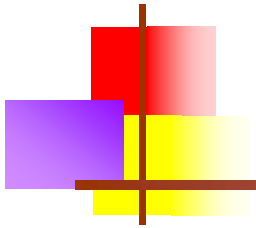
30 Μαρτίου 2017

papagianno@ceid.upatras.gr



Εισαγωγή

- Η JDSL είναι μια βιβλιοθήκη Δομών Δεδομένων σε Java που αναπτύχθηκε από Brown University and Johns Hopkins University.
- Είναι μια συλλογή από interfaces και κλάσεις σε Java, που υλοποιούν βασικές δομές δεδομένων και αλγορίθμους όπως:
  - Sequences
  - Binary trees
  - Priority Queues
  - αλγόριθμους αναζήτησης



Πλεονεκτήματα της JDSL

- Λειτουργικότητα: Μεγάλη ποικιλία υλοποιημένων δομών και αλγορίθμων.
- Απόδοση: Στις υλοποιήσεις των δομών της JDSL συναντούμε τις θεωρητικές ασυμπτωτικές πολυπλοκότητες χώρου και χρόνου.
- Ευελιξία: Η JDSL παρέχει διαφορετικές υλοποιήσεις για κάποιες δομές δεδομένων για διαφορετική χρήση ανάλογα σε εφαρμογή.
- Επεκτασιμότητα: Παρέχει λεπτομερή interfaces για πολλές δομές δεδομένων ώστε να μπορεί κανείς να κάνει μια νέα υλοποίηση.

Δομή της JDSL [Container & Element]

Container:

- Κάθε δομή δεδομένων αντιμετωπίζεται από την JDSL σαν μια οργανωμένη συλλογή αντικειμένων που καλούνται elements της δομής.
- Όλες οι δομές δεδομένων της JDSL υλοποιούν το Container Interface το οποίο ορίζει βασικές μεθόδους όπως την αναφορά του αριθμού των στοιχείων της δομής, και την επιστροφή ενός iterator για τα στοιχεία.

Element:

- Ένα element μιας δομής δεδομένων της JDSL μπορεί να είναι ένα οποιοδήποτε java.lang.Object.
- Το ίδιο αντικείμενο μπορεί να αποθηκευτεί σε πολλές δομές δεδομένων, η πολλές φορές στην ίδια.

Δομή της JDSL [Container]

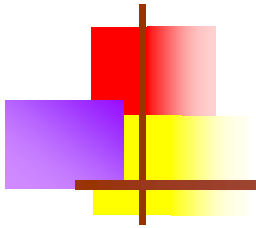
Η JDSL υλοποιεί 2 ειδών containers:

Key---based Containers

- Αποθηκεύουν ζευγάρια από key---element. Τα keys χρησιμοποιούνται συνήθως ως μηχανισμός για τη δημιουργία ευρετηρίου στα στοιχεία που συνδέονται. Π.χ. στο λεξικό, που είναι μια key---based δομή δεδομένων, κάποιες μέθοδοι που υποστηριζονται είναι η εισαγωγή ενός ζέυγους (key, element), η αναζήτηση σε ένα key κλπ.

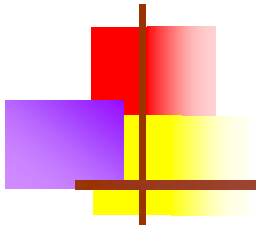
Positional Containers

- Είναι Δομές δεδομένων που δημιουργούν τοπολογικές σχέσεις μεταξύ των elements τους, όπως ακολουθίες, δέντρα κλπ. Τα στοιχεία προσπελαύνονται μέσω της θέσης τους.



Δομή της JDSL [Accessor]

- Η JDSL παρέχει πρόσβαση στα στοιχεία μιας δομής δεδομένων μέσω των accessors.
- Ένας accessor παρέχει πρόσβαση σε ένα στοιχείο μια δομής δεδομένων συνεχώς, ανεξάρτητα από την υλοποίηση της (πχ στην JDSL ένα sequence υλοποιείται είτε ως ένας πίνακας, είτε ως μια linked list).
- Κάθε element μιας δομής δεδομένων διαθέτει έναν accessor που σχετίζεται με αυτό.



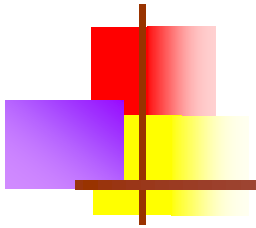
Δομή της JDSL [Accessor]

Positional Containers: Position

- Ένα position συμβολίζει τη «θέση» ενός element σε ενα positional container. Τα positional containers είναι δομές που μπορούμε να αναπαραστήσουμε με κόμβους (π.χ. Sequences, graphs κλπ), και το position ενός element εκφράζει τον κόμβο στον οποίο αυτό αποθηκεύεται.

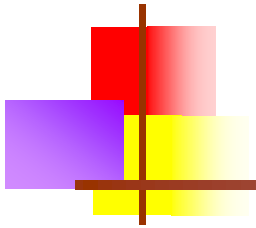
Key---based Containers: Locator

- Στα key---based Containers η έννοια της «θέσης» ενός element στο χώρο δεν υπάρχει, και κάθε στοιχείο εντοπίζεται από το κλειδί με το οποίο συνδέεται. Όταν προστίθεται ένα ζεύγος (key, element) σε έναν container αυτός θα επιστρέψει έναν locator για αυτό, και στη συνέχεια μπορεί να γίνει αναφορά στο ζεύγος μέσω του locator.



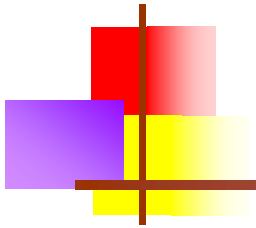
Δομή της JDSL [Iterator]

- Παρέχουν έναν απλό μηχανισμό για επαναλήψεις στα αντικείμενα μίας συλλογής.
- Οι containers παρέχουν μεθόδους που επιστρέφουν iterators που διατρέχουν όλα τα elements του container.).
- Οι iterators μπορούν να κινηθούν μόνο προς τα εμπρός.
- Μπορούν να επανέλθουν στην αρχή για να επαναλάβουν την διαδικασία.



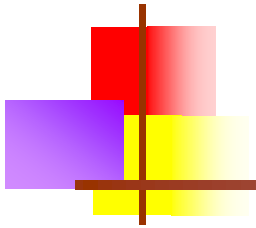
Positional Containers (Sequence)

- Ένα sequence είναι μια βασική δομή δεδομένων για την αποθήκευση στοιχείων με ένα γραμμικό τρόπο.
- InspectableSequence & Sequence: Τα interfaces της JDSL για sequences. Παρέχει μεθόδους για προσπέλαση και τροποποίηση των θέσεων στο τέλος του sequence αλλά και σε συγκεκριμένες θέσεις της
- NodeSequence: Υλοποίηση του sequence με χρήση διπλά συνδεδεμένης λίστας.
- ArraySequence: Υλοποίηση του sequence με χρήση ενός πίνακα αυξανόμενου μεγέθους.



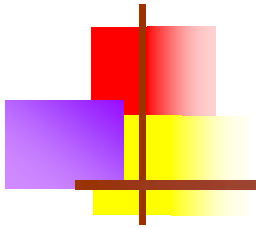
Positional Containers (Sorting Algorithms)

- Η JDSL παρέχει σειρά αλγόριθμων ταξινόμησης για sequences που υλοποιούν το interface SortObject. Ταξινομήσεις με πρόθεμα List είναι πιο αποτελεσματικές όταν χρησιμοποιούνται με την ακολουθία NodeSequence ενώ με πρόθεμα Array είναι πιο αποτελεσματικές με την ArraySequence.
- ListQuickSort/ArrayQuickSort: Γρήγορος αλγόριθμος, τρέχει σε O(NlogN). Η επίδοσή του υποβαθμίζεται σε μεγάλο βαθμό, αν η σειρά εχει σχεδόν ταξινομηθεί. Δεν είναι σταθερός , δεν εγγυάται ότι τα στοιχεία με ίδια τιμή θα παραμείνουν στην ίδια σειρά που είχαν πριν από τη ταξινόμηση.
- ListMergeSort/ArrayMergeSort: Πιο αργός από QuickSort, με την ίδια όμως πολυπλοκότητα O(NlogN) . Η απόδοση του δεν υποβαθμίζεται λόγω ιδιαιτεροτήτων στα δεδομένα εισόδου.
- HeapSort: Βασίζεται στο ArrayHeap. Η επίδοσή του επίσης δεν υποβαθμίζεται, λόγω ιδιαιτεροτήτων στην εισαγωγή δεδομένων.



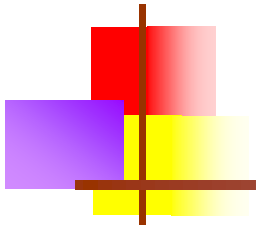
Key---based Containers (PriorityQueue)

- Ένα PriorityQueue είναι μια δομή δεδομένων για την αποθήκευση μιας συλλογής στοιχείων στα οποία δίνεται προτεραιότητα με βάση τα κλειδιά, όπου η μικρότερη τιμή κλειδιού υποδεικνύει την υψηλότερη προτεραιότητα.
- Υποστηρίζει αυθαίρετες εισαγωγές , διαγραφές στοιχείων και παρακολουθεί τα κλειδιά υψηλότερης---προτεραιότητας.
- Είναι χρήσιμη, στις εφαρμογές όπου αποθηκεύεται μια ουρά διεργασιών με ποικίλες προτεραιότητες, και πάντα η επόμενη διεργασία που επεξεργάζεται είναι η σημαντικότερη.



Key---based Containers (PriorityQueue)

- PriorityQueue: Το interface για Priority Queues. Παρέχει μεθόδους για να προσπελάσει ή να αφαιρέσει το ζευγάρι κλειδί---στοιχείο με την πιό υψηλή προτεραιότητα και για να αλλάξει την προτεραιότητα ενός στοιχείου.
- ArrayHeap: αποδοτική υλοποίηση του PriorityQueue που χτίζεται επάνω σε έναν σωρό. Η εισαγωγή, η αφαίρεση, ή η αλλαγή του κλειδιού ενός ζευγαριού κλειδιού---στοιχείου παίρνουν λογαριθμικό χρόνο και η εξέταση του ζευγαριού κλειδιού---στοιχείου με το ελάχιστο κλειδί μπορεί να γίνει σε σταθερό χρόνο.



Παραδείγματα

Demo

Παραδείγματα

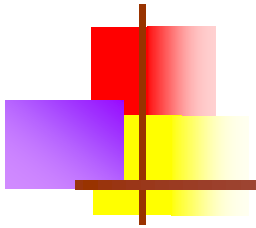
```
[ First, Third, Second ]
[ First, Second, Third ]
[ Third, Second, First ]
```

```
1
2   import jdsl.core.api.*;
3   import jdsl.core.ref.*;
4   import jdsl.core.algo.sorts.*;
5
6
7   public class SequenceExample {
8
9       private Sequence seq_;
10
11
12      public SequenceExample() { seq_ = new ArraySequence(); }
15
16      private void populateSequence() {
17          Position first = this.seq_.insertFirst("First");
18          Position second = this.seq_.insertLast("Third");
19          Position third = this.seq_.insertLast("Second");
20      }
21
22      private void sortSequence() {
23          SortObject sorter = new ArrayQuickSort();
24          sorter.sort(this.seq_, new ComparableComparator());
25      }
26
27      private void reverseSortSequence() {
28          SortObject sorter = new ArrayQuickSort();
29          sorter.sort(this.seq_, new ComparatorReverser(new ComparableComparator()));
30      }
31
32      private void printSequence() { System.out.println(this.seq_.toString()); }
35
36      public static void main(String args[]) {
37          SequenceExample s = new SequenceExample();
38          s.populateSequence();
39          s.printSequence();
40          s.sortSequence();
41          s.printSequence();
42          s.reverseSortSequence();
43          s.printSequence();
44
45      }
46
47
48  }
```

Παραδείγματα

```
1  import jdsl.core.api.*;
2  public class PriorityQueueExample {
3
4      private PriorityQueue pq_;
5
6      public PriorityQueueExample() { pq_ = new jdsl.core.ref.ArrayHeap(new jdsl.core.ref.IntegerComparator()); }
9
10     public void populatePriorityQueue(){
11
12         Object key = 1;
13         Object element = "client1";
14         Locator client1 = pq_.insert(key, element);
15
16         key = 2;
17         element = "client2";
18         Locator client2 = pq_.insert(key, element);
19
20         key = 3;
21         element = "client3";
22         Locator client3 = pq_.insert(key, element);
23     }
24
25     public void printPriorityQueue() { System.out.println(pq_.toString()); }
28
29     public void printPriorityQueueWithIterator(){
30
31         ObjectIterator keysIterator = pq_.keys();
32         while(keysIterator.hasNext())
33             System.out.println(keysIterator.nextObject());
34
35         ObjectIterator elemIterator = pq_.elements();
36         while(elemIterator.hasNext())
37             System.out.println(elemIterator.nextObject());
38     }
39
40     public void printMinOfPriorityQueue()
41     {
42         System.out.println(pq_.min());
43     }
44
45     public void deleteClientWithHighestPriority(){
46         pq_.removeMin();}
47
48     public static void main(String args[]){
49         PriorityQueueExample m = new PriorityQueueExample();
50         m.populatePriorityQueue();
51         m.printPriorityQueue();
52         m.printPriorityQueueWithIterator();
53         m.printMinOfPriorityQueue();
54         m.deleteClientWithHighestPriority();
55         m.printMinOfPriorityQueue();
56     }
57 }
```

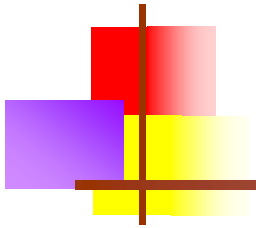
```
{ 1=client1, 2=client2, 3=client3 }
1
2
3
client1
client2
client3
Locator with key 1 = client1
Locator with key 2 = client2
```



References

http://cs.brown.edu/cgc/jdsl

http://www3.cs.stonybrook.edu/~algorith/implement/jdsl/implement.shtml



Ευχαριστώ

Ερωτήσεις?