Δομές Δεδομένων

Υλοποιηση FIFO ουράς με την τεχνική του κυκλικού πίνακα

Παναγιώτης-Νεκτάριος Τζωρτζίνης, it2021097
Παπαγιάννης Γεώργιος, it21877
Λάμπρος Χαλάτσης, it2022111

Κυκλικός Πίνακας

Η υλοποίηση ξεκινάει από τη κλάση **CircularArrayDequeue** βάζοντας τα properties της:

DEFAULT_CAPACITY: Το αρχικό μέγεθος του πίνακα (είναι προεπιλεγμένο μέγεθος)

size: Το τρέχον αριθμό των στοιχείων που περιέχει η ουρά. Ενημερώνεται καθώς προστίθενται ή αφαιρούνται στοιχεία

f: f είναι η πρώτη θέση του πίνακα που περιέχει κάποιο στοιχείο της ουράς

r: r είναι μία θέση μετά την τελευταία γεμάτη

array: Ο πίνακας

modCount: Αναγνωριστική σταθερά που χρησιμοποιείται για τη διατήρηση του αριθμού των τροποποιήσεων που έχουν γίνει στην κυκλική ουρά. Καταγράφει τις δομικές αλλαγές που συμβαίνουν στην ουρά.

Λειτουργίες της CircularArrayDequeue

pushFirst(E elem): Αν ο πίνακας είναι γεμάτος, καλείται η doubleCapacity() για να αυξήσει το μεγέθους του πίνακα. Αν ουρά είναι αρχικά άδεια, ο δείκτης f μπαινει στο τέλος του πίνακα, αλλιώς μετακινείται κυκλικά προς τα πίσω. Το στοιχείο προστίθεται στη θέση f, και ο δείκτης f αυξάνεται κατά 1. Αυξάνεται το size και το modCount κατα ενα .

pushLast(E elem): Αν ο πίνακας είναι γεμάτος, καλείται η doubleCapacity() για να αυξήσει το μεγέθους του πίνακα. Το στοιχείο προστίθεται στη θέση r, και ο δείκτης r μετακινείται κυκλικά προς τα μπροστά. Αυξάνεται το size και το modCount κατα ενα .

popFirst(): Αν η ουρά είναι άδεια, πεταει NoSuchElementException. Το στοιχείο που βρίσκεται στη θέση f διαγράφεται, και ο δείκτης f μετακινείται κυκλικά προς τα μπροστά. Μειώνεται το size κατα ενα . Αν η ουρά γίνει άδεια, οι δείκτες f και r γινονται στο 0 και το size γίνεται 0. Αν το πλήθος των στοιχείων είναι μικρότερο από 1/4 του μεγέθους του πίνακα, καλείται η halfCapacity().

popLast(): Εάν η ουρά είναι άδεια, πεταει NoSuchElementException. Ο δείκτης r μετακινείται κυκλικά προς τα πίσω, και το στοιχείο που βρίσκεται στη νέα θέση r διαγράφεται. Μειώνεται το size κατα ενα. Αν η ουρά γίνει άδεια, οι δείκτες f και r τίθενται στο 0 και το size γινεται 0. Αν το πλήθος των στοιχείων είναι μικρότερο από 1/4 του μεγέθους του πίνακα, καλείται η halfCapacity().

first(): Εάν η ουρά είναι άδεια, πέταει NoSuchElementException. Επιστρέφεται το στοιχείο που βρίσκεται στη θέση f του πίνακα.

last(): Εάν η ουρά είναι άδεια, πέταει NoSuchElementException. Ο υπολογισμός της θέσης του τελευταίου στοιχείου γίνεται με βάση τον δείκτη r και επιστρέφεται το στοιχείο που βρίσκεται σε αυτήν τη θέση.

isEmpty(): Επιστρέφει true αν η ουρά είναι άδεια(δηλαδη αν το size ειναι ισο με το 0 και το front ειναι ισο με το rear)

size(): Επιστρέφει το μέγεθος της ουράς

clear(): Καθαρίζει την ουρά, επαναφέροντας τις μεταβλητές στις αρχικές τους τιμές και δημιουργώντας ένα νέο πίνακα.

iterator(): Αρχικά αρχικοποιείται το **current** στο f, δηλαδή στον δείκτη του πρώτου στοιχείου. Ο lastReturnedModCount κρατά την τιμή του modCount για να ελέγχει τη συνέπεια του Iterator.

Η μέθοδος **hasNext()** επιστρέφει true αν το current δεν είναι ίσο με το r δηλαδή αν υπάρχουν περισσότερα στοιχεία.

Η μέθοδος **next()** επιστρέφει το στοιχείο της current στον πίνακα. Το current γίνεται((current + 1) % array.length).

Τέλος η μέθοδος **checkForComodification()** ελέγχει αν υπήρξε κάποια ταυτόχρονη τροποποίηση στον CircularArrayDeQueue. Αν διαφέρει το modCount, πεταει ConcurrentModificationException.

descendingIterator(): Αρχικά αρχικοποιείται ο current με τον δείκτη του τελευταίου στοιχείου της ουράς. Η hasPrintedFront είναι false και γινεται true μόνο όταν το current ειναι ισο με το front. Ο lastReturnedModCount όπως και στην iterator κρατά την τιμή του modCount για να ελέγχει τη συνέπεια του Iterator.

Η μέθοδος **hasNext()** επιστρέφει true αν η μεταβλητη hasPrintedFront είναι false δηλαδή επιστρεφει true οσο δεν έχει εκτυπωθεί το πρώτο στοιχείο.

Η μέθοδος **next()** επιστρέφει το στοιχείο της current στον πίνακα. Το current μειώνεται κατά ((current - 1 + array.length) % array.length). Το hasPrintedFront γίνεται true όταν το current γίνει ίσο με το f.

Τέλος η **checkForComodification()** ελεγχει οπως και στην iterator αν υπηρξε καποια ταυτοχρονη τροποποιηση στον CircularArrayDeQueue

doubleCapacity(): Διπλασιάζει το μέγεθος του πίνακα και μεταφέρει τα στοιχεία σε νέα θέση.

halfCapacity():Μειώνει το μέγεθος του πίνακα στο μισό και μεταφέρει τα στοιχεία σε νέα θέση.

printDequeueStatus(): Αν η ουρα δεν ειναι αδεια, εκτυπώνει τον πίνακα που υλοποιειται η ουρα και το front και rear της ουρας

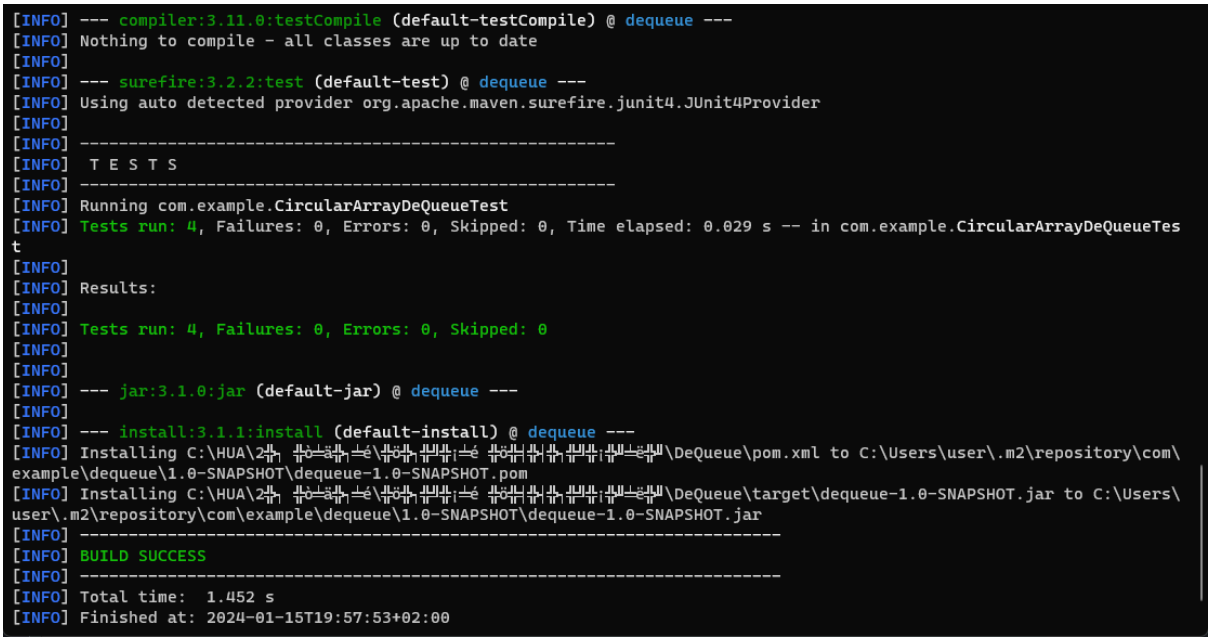
Tests

- Το τεστ **testPushAndPop()**, επικυρώνει τις βασικές λειτουργίες push και pop και στα δύο άκρα της ουράς. Ωθεί στοιχεία στην πρώτη και την τελευταία θέση και, στη συνέχεια, επαληθεύει ότι οι μέθοδοι μεγέθους (isEmpty) λειτουργούν όπως αναμένεται. Στη συνέχεια, ελέγχει εάν τα στοιχεία μπορούν να αναδυθούν σωστά από την πρώτη και την τελευταία θέση.

- Το τεστ **testFirstAndLast(),** εστιάζει στις λειτουργίες που έχουν πρόσβαση στο πρώτο και το τελευταίο στοιχείο της ουράς. Επαληθεύει ότι η πρώτη και η τελευταία μέθοδος επιστρέφουν σωστά τα αναμενόμενα στοιχεία και ότι τα αναδυόμενα στοιχεία ενημερώνουν κατάλληλα την ουρά.

- Το τεστ **testIterator()**, ελέγχει τη λειτουργία επανάληψης στην ουρά. Σπρώχνει χαρακτήρες στην ουρά, επαναλαμβάνει την διαδικασία πάνω τους χρησιμοποιώντας έναν βρόχο for και διασφαλίζει ότι οι χαρακτήρες ανακτώνται με τη σωστή σειρά.

- Το τεστ **testResize()**, αξιολογεί την ικανότητα αλλαγής μεγέθους της ουράς που βασίζεται σε κυκλικό πίνακα. Σπρώχνει μια ακολουθία ακεραίων αριθμών στην ουρά, υπερβαίνει την αρχική του ικανότητα να ενεργοποιήσει μια αλλαγή μεγέθους και, στη συνέχεια, αναδύει στοιχεία από το μπροστινό μέρος. Η δοκιμή επιβεβαιώνει ότι διατηρείτε το σωστό μέγεθος και δεν αναφέρεται ως κενό μετά την αλλαγή μεγέθους.



Απο την εικονα αυτη βλεπουμε πως τα Τεστ ηταν επιτυχημενα.

Παραδειγματα

Η περιλαμβάνει μια σειρά από λειτουργίες ελέγχου για τις βασικές λειτουργίες της ουράς που υλοποιείται από την κλάση CircularArrayDeQueue.

Πρώτα, εισάγονται πέντε στοιχεία στην αρχή της ουράς χρησιμοποιώντας τη μέθοδο **pushFirst()**. Στη συνέχεια, εκτυπώνεται η κατάσταση της ουράς με την **printDequeueStatus()**, ακολουθούμενη από την εκτύπωση των στοιχείων χρησιμοποιώντας τον **Iterator()**.

```
public static void main(String[] args) {
    DeQueue<Integer> deque = new CircularArrayDeQueue<>();

    // Test the DeQueue operations
    System.out.println(x:"Pushing to front:");
    for(int i = 0; i < 5; i++) {
        deque.pushFirst(i);
    }
    deque.printDequeStatus();
```

```
    System.out.println(x:"\nDeque elements using Iterator:");
    Iterator<Integer> iterator = deque.iterator();
    while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
    }
    System.out.println();
```

Output:

```
Pushing to front:
[null, null, null, 4, 3, 2, 1, 0]
Front is: 4
Rear is: 0


Deque elements using Iterator:
4 3 2 1 0
```

Ακολουθούν αφαιρέσεις στοιχείων με τις μεθόδους **popFirst()** και **popLast()**, ενώ εκτυπώνεται η ενημερωμένη κατάσταση της ουράς και το μέγεθος της.

```
    System.out.println("\nPop first element: " + deque.popFirst());
    System.out.println("Pop last element: " + deque.popLast());
    System.out.println("Pop last element again: " + deque.popLast() + "\n");

    deque.printDequeStatus();
    System.out.println("Updated Deque size: " + deque.size());
```

Output:

```
Pop first element: 4
Pop last element: 0
Pop last element again: 1

[3, 2, null, null]
Front is: 3
Rear is: 2

Updated Deque size: 2
```

Στη συνέχεια, προστίθενται επιπλέον στοιχεία στο τέλος της ουράς με τη **pushLast()** και εκτυπώνεται ξανά η κατάσταση της ουράς, αυτή τη φορά με τη χρήση του **descendingIterator()** για αντίστροφη εκτύπωση.

```
        System.out.println(x:"\nPushing to last:");
        for (int i = 5; i < 12; i++) {
            deque.pushLast(i);
        }
        deque.printDequeStatus();


        System.out.println(x:"\nDeque elements in reverse using Decsending Iterator:");
        Iterator<Integer> descendingIterator = deque.descendingIterator();
        while (descendingIterator.hasNext()) {
            System.out.print(descendingIterator.next() + " ");
        }
        System.out.println();
```

Output:

```
Pushing to last:
[3, 2, 5, 6, 7, 8, 9, 10, 11, null, null, null, null, null, null, null]
Front is: 3
Rear is: 11


Deque elements in reverse using Decsending Iterator:
11 10 9 8 7 6 5 2 3
```

Τέλος, εμφανίζεται η ενημερωμένη κατάσταση της ουράς μαζί με το μέγεθος της.

```
    System.out.println("\nUpdated Deque size: " + deque.size() + "\n");
    // Clear the DeQueue
    deque.clear();
    System.out.println("Cleared Deque size: " + deque.size());
```

Output:

```
Updated Deque size: 9

Cleared Deque size: 0
```