Advanced Object-Oriented Programming

CPT204 – Lab 9

Erick Purwanto

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT204 Advanced Object-Oriented Programming

Lab 9

Iterator, Disjoint Sets

Welcome !

- Welcome to Lab 9 !
  - We are going to create an **ARDeque Iterator** for ARDeque,
  - and we are going to implement **Weighted Quick Union** Disjoint Sets
  - You will find the information about them, needed to complete this lab, in *the lecture notes*

- You will find in this lab
  1. Lab Exercise 9.1 - 9.4, and their hints
  2. Exercise 9.1 - 9.3

- Download **lab9** zip files from Learning Mall
- Don't forget to import the **lab9** files and the library into an IntelliJ project
  - Read **lab1** again for reference

Lab Exercise 9.1 ARDequeIterator CONSTRUCTOR and HASNEXT

- Complete the constructor of **ARDequeIterator**, and the method **hasNext**.
- The constructor makes an iterator for ARDeque objects.
- The method tests whether the iterator has more items to return.

Lab Exercise 9.2 ARDequeIterator NEXT

- Complete the method **next**.
- It returns the next item, and then advances to item after that in the deque.

Test Case for Lab Exercise 9.1 and Lab Exercise 9.2

- Test case 1:

```
ARDeque<String> deque = new ARDeque<>();
deque.addLast("a");
deque.addLast("b");
deque.addLast("c");

ARDequeIterator<String> iter = new ARDequeIterator<>(deque);
while (iter.hasNext()) {
    String str = iter.next();
    System.out.print(str + " ");
}
```

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 9.1, 9.2 ARDequeIterator Hints

- Follow the MyIterator for ArrayList<String> implementation in the lecture notes
  - it will be very similar, but using type parameter

Weighted Quick Union Disjoint Sets

- The next exercise and assignments are about the Weighted Quick Union Disjoint Sets data structure
  - we use the parent array and the weight strategy to connect the root of smaller size tree to root of larger size tree
  - if the sizes of the trees are equal in connect(p, q), break the tie by connecting p's root to q's root

- We will implement it using Idea 4.2 described in Lecture Notes page 150
  - that is, store the **negative size** of the trees in the root

Test Case for Lab Exercise 9.3, Exercise 9.1 - 9.3

- Test case 1:

```
WeightedQuickUnionDS ds = new WeightedQuickUnionDS(4);
ds.connect(1, 0);
ds.isConnected(1, 0);                  →      true
ds.parent(1);                          →      0
ds.parent(0);                          →      -2
ds.connect(3, 2);
ds.isConnected(2, 1);                  →      false
ds.connect(3, 1);
ds.isConnected(2, 1);                  →      true
ds.parent(2);                          →      0
ds.sizeOf(1);                          →      4
ds.printParent();                      →      -4 0 0 2
```

Lab Exercise 9.3 WeightedQuickUnionDS CONSTRUCTOR

- Complete the constructor public **WeightedQuickUnionDS**(int N).
- It creates a Disjoint Sets data structure with N elements, 0 through N-1.
- Initially, each element is in its own set.

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 9.3 WeightedQuickUnionDS CONSTRUCTOR Hints

- Each element initially is its own tree, so let us code to create that!

- Initialize the parent array to hold N integers
- Set each integer to represent the negative size of each tree, where each tree initially consists of just one element

Lab Exercise 9.4 WeightedQuickUnionDS VALIDATE

- Complete the method void **validate**(int p).
- It validates that p is a valid element/index.
- If p is not a valid index, throw an **IllegalArgumentException** as in the test case 2 below.

- Test case 2:

```
WeightedQuickUnionDS ds = new WeightedQuickUnionDS(5);
ds.validate(10);    →    IllegalArgumentException
```

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 9.4 WeightedQuickUnionDS VALIDATE Hints

- The valid indices (not the values) are simply the ones set in the constructor, if that is not the case, then throws an object of IllegalArgumentException

Exercise 9.1 WeightedQuickUnionDS SIZE OF

- Complete the method int **sizeOf**(int p).
- It returns the size of the set element p belongs to.

Exercise 9.2 WeightedQuickUnionDS IS CONNECTED

- Complete the method boolean **isConnected**(int p, int q).
- It return *true* if p and q are connected / in the same set,
  - and *false* otherwise.
- It throws IllegalArgumentException if p or q is not a valid index.

Exercise 9.3 WeightedQuickUnionDS CONNECT

- Complete the method void **connect**(int p, int q).
- It connects two elements p and q together, by combining the sets containing them, connecting the root of smaller size tree to root of larger size tree.
  - If the sizes of the trees are *equal*, *break the tie* by connecting p's root to q's root.
- It throws IllegalArgumentException if p or q is not a valid index.

Thank you for your attention !

- In this lab, you have learned:
  - To create an iterator class of a data structure
    - hasNext() and next()
  - To create a data structure called Disjoint Sets / Union Find using a provable efficient weighted quick union technique