Project 2 (Deques and Randomized Queues) Checklist

Prologue

Project goal: implement generic and iterable data structures, such as double-ended and randomized queues, using arrays and linked lists

Relevant lecture material

- ⇝ Data Abstraction
- ⇝ Basic Data Structures

Files

- ⇝ `project2.pdf` (project description)
- ⇝ `project2.zip` (starter files for the exercises/problems, `report.txt` file for the project report, and `run_tests.py` file to test your solutions)

Exercises

Exercise 1. (*Iterable Binary Strings*) Implement an immutable, iterable data type `BinaryStrings` in `BinaryStrings.java` to systematically iterate over binary strings of length n. The data type must support the following API:

method	description
`BinaryStrings(int n)`	construct an iterable `BinaryStrings` object given the length of binary strings needed
`Iterator<String> iterator()`	an iterator for binary strings of a given length

```
$ java BinaryStrings 3
000
001
010
011
100
101
110
111
```

Exercises

BinaryStrings.java

```
// An immutable data type to systematically iterate over binary
// strings of length n.
public class BinaryStrings implements Iterable<String> {
    private final int n; // need all binary strings of length n

    // Construct an iterable BinaryStrings object given the length
    // of binary strings needed.
    public BinaryStrings(int n) {
        ...
    }

    // A BinaryStringsIterator object.
    public Iterator<String> iterator() {
        ...
    }

    // Binary strings iterator.
    private class BinaryStringsIterator implements
                                            Iterator<String> {
        private int count; // number of binary strings returned
        private int p;     // current number

        // Construct a BinaryStringsIterator object.
        BinaryStringsIterator() {
            ...
        }

        // Are there anymore binary strings left to be iterated?
        public boolean hasNext() {
            ...
        }

        // The next binary string.
        public String next() {
            ...
        }
```

Exercises

BinaryStrings.java

```
        // Remove is not supported.
        public void remove() {
            // nothing to do
        }

        // The n-bit representation of x.
        private String binary(int x) {
            String s = Integer.toBinaryString(x);
            int padding = n - s.length();
            for (int i = 1; i <= padding; i++) {
                s = "0" + s;
            }
            return s;
        }
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (String s : new BinaryStrings(n)) {
            StdOut.println(s);
        }
    }
}
```

Exercises

Exercise 2. (*Iterable Primes*) Implement an immutable, iterable data type `Primes` in `Primes.java` to systematically iterate over the first n primes. The data type must support the following API:

method	description
`Primes(int n)`	construct an iterable `Primes` object given the number of primes needed
`Iterator<Integer> iterator()`	an iterator for the given number of primes

```
$ java Primes 10
2
3
5
7
11
13
17
19
23
29
```

Exercises

Primes.java

```
import java.util.Iterator;

// An immutable data type to systematically iterate over the
// first n primes.
public class Primes implements Iterable<Integer> {
    private final int n; // need first n primes

    // Construct an iterable Primes object given the number
    // of primes needed.
    public Primes(int n) {
        ...
    }

    // A PrimesIterator object.
    public Iterator<Integer> iterator() {
        ...
    }

    // Primes iterator.
    private class PrimesIterator implements
                                    Iterator<Integer> {
        private int count; // number of primes returned
        private int p;     // current prime

        // Construct a PrimesIterator object.
        PrimesIterator() {
            count = 0;
            p = 2;
        }

        // Are there anymore primes left to be iterated?
        public boolean hasNext() {
            ...
        }

        // The next prime.
        public Integer next() {
```

Exercises

Primes.java

```
            // Increment count by 1.
            ...

            // As long as p is not prime, increment p by 1.
            ...

            // Return current value of p and increment it
            // by 1.
            ...
        }

        // Remove is not supported.
        public void remove() {
            // nothing to do
        }

        // Is x (>= 2) prime?
        private boolean isPrime(int x) {
            for (int i = 2; i <= x / i; i++) {
                if (x % i == 0) {
                    return false;
                }
            }
            return true;
        }
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i : new Primes(n)) {
            StdOut.println(i);
        }
    }
}
```

Exercises

Exercise 3. (*Min Max*) Implement the static methods `min()` and `max()` in `MinMax.java` that take a reference `first` to the first node in a linked list of integer-valued items as argument and returns the minimum and the maximum values respectively.

```
$ java MinMax
true
```

Exercises

MinMax.java

```
public class MinMax {
    // Linked list class.
    private static class Node {
        private int item;
        private Node next;
    }

    // Return the minimum value in the given linked list.
    public static int min(Node first) {
        // Set min to the largest integer
        int min = Integer.MAX_VALUE;

        // Compare each element in linked list with min and
        // if it is smaller, update min.
        ...

        // Return min
        ...
    }

    // Return the maximum value in the given linked list.
    public static int max(Node first) {
        // Set max to the smallest integer
        int max = Integer.MIN_VALUE;

        // Compare each element in linked list with max and
        // if it is larger, update max.
        ...

        // Return max
        ...
    }

    // Test client. [DO NOT EDIT]
    public static void main(String[] args) {
        int[] items = new int[1000];
        for (int i = 0; i < 1000; i++) {
```

Exercises

MinMax.java

```
            items[i] = StdRandom.uniform(-10000, 10000);
        }
        Node first = null;
        for (int item : items) {
            Node oldfirst = first;
            first = new Node();
            first.item = item;
            first.next = oldfirst;
        }
        StdOut.println(min(first) == StdStats.min(items)
                       && max(first) == StdStats.max(items));
    }
}
```

Exercises

Exercise 4. (*Text Editor Buffer*) Develop a data type `Buffer` for a buffer in a text editor that implements the following API:

method	description
`Buffer()`	create an empty buffer
`void insert(char c)`	insert c at the cursor position
`char delete()`	delete and return the character at the cursor
`void left(int k)`	move the cursor k positions to the left
`void right(int k)`	move the cursor k positions to the right
`int size()`	number of characters in the buffer
`String toString()`	string representation of the buffer with a '|' character (not part of the buffer) at the cursor position

```
$ java Buffer
|There is grandeur in this view of life, with its several powers,
having been originally breathed by the Creator into a few forms or
into one; and that, whilst this planet has gone cycling on
according to the fixed law of gravity, from so simple a beginning
endless forms most beautiful and most wonderful have been, and
are being, evolved.
-- Charles Darwin, The Origin of Species
```

Hint: Use two stacks `left` and `right` to store the characters to the left and right of the cursor, with the characters on top of the stacks being the ones immediately to its left and right.

Exercises

Buffer.java

```
// A data type representing a text editor buffer.
public class Buffer {
    private Stack<Character> left;  // chars left of cursor
    private Stack<Character> right; // chars right of cursor

    // Create an empty buffer.
    public Buffer() {
        ...
    }

    // Insert c at the cursor position.
    public void insert(char c) {
        ...
    }

    // Delete and return the character at the cursor.
    public char delete() {
        ...
    }

    // Move the cursor k positions to the left.
    public void left(int k) {
        ...
    }

    // Move the cursor k positions to the right.
    public void right(int k) {
        ...
    }

    // Return the number of characters in the buffer.
    public int size() {
        ...
    }

    // Return a string representation of the buffer with
    // a "|" character (not part of the buffer) at the
```

Exercises

Buffer.java

```
    // cursor position.
    public String toString() {
        StringBuilder sb = new StringBuilder();

        // Push chars from left into a temporary stack.
        ...

        // Append chars from temporary stack to sb.
        ...

        // Append "|" to sb.
        ...

        // Append chars from right to sb.
        ...

        // Return the string from sb.
        ...
    }

    // Test client (DO NOT EDIT).
    public static void main(String[] args) {
        Buffer buf = new Buffer();
        String s = "There is grandeur in this view of life, "
            + "with its several powers, having been originally "
            + "breathed into a few forms or into one; and that, "
            + "whilst this planet has gone cycling on according "
            + "to the fixed law of gravity, from so simple a "
            + "beginning endless forms most beautiful and most "
            + "wonderful have been, and are being, evolved. ~ "
            + "Charles Darwin, The Origin of Species";
        for (int i = 0; i < s.length(); i++) {
            buf.insert(s.charAt(i));
        }
        buf.left(buf.size());
        buf.right(97);
        s = "by the Creator ";
```

Exercises

Buffer.java

```
        for (int i = 0; i < s.length(); i++) {
            buf.insert(s.charAt(i));
        }
        buf.right(228);
        buf.delete();
        buf.insert('-');
        buf.insert('-');
        buf.left(342);
        StdOut.println(buf);
    }
}
```

Exercises

Exercise 5. (*Josephus Problem*) In the Josephus problem from antiquity, N people are in dire straits and agree to the following strategy to reduce the population. They arrange themselves in a circle (at positions numbered from 0 to $N - 1$) and proceed around the circle, eliminating every Mth person until only one person is left. Legend has it that Josephus figured out where to sit to avoid being eliminated. Write a `Queue` client `Josephus.java` that takes N and M from the command line and prints out the order in which people are eliminated (and thus would show Josephus where to sit in the circle).

```
$ java Josephus 7 2
2
4
6
1
5
3
7
```

Exercises

Josephus.java

```
public class Josephus {
    public static void main(String[] args) {
        // Get N and M from command line as ints.
        ...

        // Create a queue q and enqueue integers
        // 1, 2, ... N.
        ...

        int i = 0;
        // As long as q is not empty: increment i;
        // dequeue an element pos; if M divides i,
        // write pos, otherwise enqueue pos.
        ...
    }
}
```

Problems

Student

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Instructor

Please summarize the project description for the students before you walk them through the rest of this checklist document.

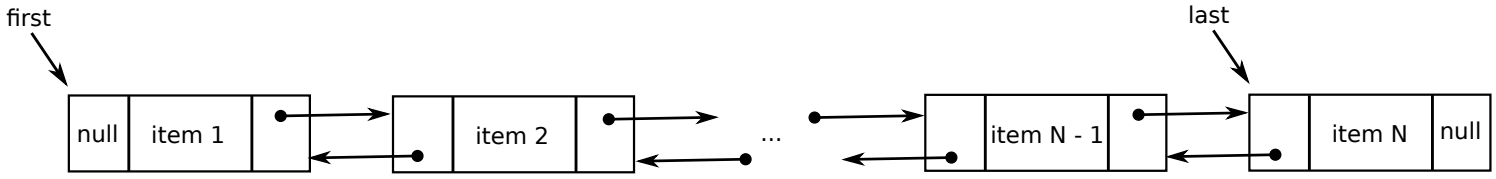
Problems

Problem 1. (*Deque*) Create a generic iterable data type `LinkedDeque<Item>` that uses a linked list to implement the following deque API:

method	description
`LinkedDeque()`	construct an empty deque
`boolean isEmpty()`	is the deque empty?
`int size()`	the number of items on the deque
`void addFirst(Item item)`	add *item* to the front of the deque
`void addLast(Item item)`	add *item* to the end of the deque
`Item removeFirst()`	remove and return the item from the front of the deque
`Item removeLast()`	remove and return the item from the end of the deque
`Iterator<Item> iterator()`	an iterator over items in the deque in order from front to end
`String toString()`	a string representation of the deque

Hints

- Use a doubly-linked list `Node` to implement the API, where each node stores a generic `item`, and pointers `next` and `prev` to the next and previous nodes



Problems

- Instance variables
  - Size of the deque, `int N`
  - Pointer to the head of the deque, `Node first`
  - Pointer to the tail of the deque, `Node last`
- `LinkedDeque()`
  - Initialize instance variables to appropriate values
- `boolean isEmpty()`
  - Return whether the deque is empty or not
- `int size()`
  - Return the size of the deque
- `void addFirst(Item item)`
  - Add the given item at the head end of the deque
  - Increment `N` by one
- `void addLast(Item item)`
  - Add the given item at the tail end of the deque
  - Increment `N` by one

Problems

- `Item removeFirst()`
  - Remove and return the item at the head end of the deque
  - Decrement `N` by one
- `Item removeLast()`
  - Remove and return the item at the tail end of the deque
  - Decrement `N` by one
- `Iterator<Item> iterator()`
  - Return an object of type `DequeIterator`
- `DequeIterator` :: Instance variable
  - Pointer to current node in the iterator, `Node current`
- `DequeIterator` :: `DequeIterator()`
  - Initialize instance variable appropriately
- `DequeIterator` :: `boolean hasNext()`
  - Return whether the iterator has more items to iterate or not
- `DequeIterator` :: `Item next()`
  - Return the item in `current` and advance `current` to the next node

Problems

Problem 2. (*Random Queue*) Create a generic iterable data type `ResizingArrayRandomQueue<Item>` that uses a resizing array to implement the following random queue API:

method	description
`ResizingArrayRandomQueue()`	construct an empty queue
`boolean isEmpty()`	is the queue empty?
`int size()`	the number of items on the queue
`void enqueue(Item item)`	add *item* to the queue
`Item dequeue()`	remove and return a random item from the queue
`Item sample()`	return a random item from the queue, but do not remove it
`Iterator<Item> iterator()`	an independent iterator over items in the queue in random order
`String toString()`	a string representation of the queue

Hints

- ⇝ Use a resizing array to implement the API
- ⇝ Instance variables
  - ⇝ Array to store the items of queue, `Item[] q`
  - ⇝ Size of the queue, `int N`

Problems

- `ResizingArrayRandomQueue()`
  - Initialize instance variables appropriately — create `q` with an initial capacity of 2
- `boolean isEmpty()`
  - Return whether the queue is empty or not
- `int size()`
  - Return the size of the queue
- `void enqueue(Item item)`
  - If `q` is at full capacity, resize it to twice its current capacity
  - Insert the given item in `q` at index `N`
  - Increment `N` by one
- `Item dequeue()`
  - Save `q[r]` in `item`, where `r` is a random integer from the interval `[0, N)`
  - Set `q[r]` to `q[N - 1]` and `q[N - 1]` to `null`
  - If `q` is at quarter capacity, resize it to half its current capacity
  - Decrement `N` by one
  - Return `item`

Problems

- `Item sample()`
  - Return `q[r]`, where `r` is a random integer from the interval `[0, N)`
- `Iterator<Item> iterator()`
  - Return an object of type `RandomQueueIterator`
- `RandomQueueIterator()` :: Instance variables
  - Array to store the items of `q`, `Item[] items`
  - Index of the current item in `items`, `int current`
- `RandomQueueIterator()` :: `RandomQueueIterator()`
  - Create `items` with capacity `N`
  - Copy the `N` items from `q` into `items`
  - Shuffle `items`
  - Initialize `current` appropriately
- `RandomQueueIterator()` :: `boolean hasNext()`
  - Return whether the iterator has more items to iterate or not
- `RandomQueueIterator()` :: `Item next()`
  - Return the item in `items` at index `current` and advance `current` by one

Problems

Problem 3. (*Subset*) Write a client program `Subset.java` that takes a command-line integer k, reads in a sequence of strings from standard input using `StdIn.readString()`, and prints out exactly k of them, uniformly at random. Each item from the sequence should be printed out at most once. You may assume that $0 \leq k \leq N$, where N is the number of strings on standard input.

```
$ java Subset 3
A B C D E F G H I
<ctrl-d>
G
I
E
```

Hints

- ⇝ Create an object `q` of type `ResizingArrayRandomQueue`
- ⇝ Read strings from standard input and insert them into `q`
- ⇝ Dequeue and print `k` (command-line argument) items from `q`

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include

- ⇝ Time (in hours) spent on the project
- ⇝ Difficulty level (1: very easy; 5: very difficult) of the project
- ⇝ A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- ⇝ Acknowledgement of any help you received
- ⇝ Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files

- ⇝ Make sure your programs meet the style requirements by running the following command on the terminal

```
$ check_style <program>
```

  where `<program>` is the `.java` file whose style you want to check

- ⇝ Make sure your programs meet the input and output specifications by running the following command on the terminal

```
$ python3 run_tests.py -v [<items>]
```

  where the optional argument `<items>` lists the exercises/problems (`Exercise1`, `Problem2`, etc.) you want to test, separated by spaces; all the exercises/problems are tested if no argument is given

- ⇝ Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- ⇝ Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit

1. `BinaryStrings.java`
2. `Primes.java`
3. `MinMax.java`
4. `Buffer.java`
5. `Josephus.java`
6. `LinkedDeque.java`
7. `ResizingArrayRandomQueue.java`
8. `Subset.java`
9. `report.txt`