DIT181 Data Structures and Algorithms: Assignment 2

This document provides the assignment for weeks 4 and 5 of the DIT181 course. The assignment should be handed in by 16 February 2018. A test suite together with instructions on how to run it will be published separately.

Stacks and queues

Question 1

Implement a generic queue using a circular buffer, starting from the `QueueArray.java` skeleton file. The underlying array should grow in response to overflows (when there no space left in the array) and shrink to avoid wasting too much space. The time complexity of performing n operations on the queue should be $O(n)$.

Question 2

Show how to implement a queue using two stacks (no Java code is needed). What are the complexities of the `enqueue()` and `dequeue()` operations? **Note:** a stack is a data structure with `push()`, `pop()` and `isEmpty()` operations; a queue is a data structure with `enqueue()`, `dequeue()` and `isEmpty()` operations.

State the invariant of your implementation.

Question 3

Implement a simple Reverse Polish Notation arithmetic calculator. Reverse Polish Notation is a way of representing arithmetic expressions, which does not require parentheses. See these resources for more explanations:

- https://www.youtube.com/watch?v=0xDo05UwkYU
- http://mathworld.wolfram.com/ReversePolishNotation.html
- https://en.wikipedia.org/wiki/Reverse_Polish_notation
- http://hp15c.com

Your job is to implement an RPN calculator that will read input from the terminal (standard input), and print results on standard output. Example session with the calculator may look as follows:

```
$ java RPN
> 2
2
> 5
2 5
> +
7
> quit
Quitting
```

The text that follows the > character has been typed by the user, whereas the other lines have been output by the calculator. After processing a line of input, the calculator should print the

CHALMERS TEKNISKA HÖGSKOLA AB
GÖTEBORGS UNIVERSITET
Data- och informationsteknik
412 96 Göteborg
Telefon: 031-772 10 00 (CTH) / 031-786 00 00 (GU)

current contents of its evaluation stack, or quit if the word `quit` appeared. You should start from the `RPN.java` skeleton file, which currently contains code that reads input and prints messages on the standard output. The calculator should support the operations `+`, `-`, `*` and `/` on integers. Optionally you may add support for more operations and floating-point numbers. You should use the standard `Stack` collection from Java.

Linked lists

Most questions from this part will require you to work with the `SinglyLinkedList.java` skeleton file.

Question 4

Implement the method `get(int n)`, which should return the element of index n (indexing starts with 0). If the index is out of bounds, the exception `IllegalArgumentException` should be thrown.

What is the complexity of this method?

Question 5

Implement the method `insertAt(Item x, int n)`, which should insert an element at index n into the list. If the index is out of bounds, the exception `IllegalArgumentException` should be thrown.

What is the complexity of this method?

Question 6

Implement the method `removeAt(int n)`, which should remove an element at index n into the list. If the index is out of bounds, the exception `IllegalArgumentException` should be thrown.

What is the complexity of this method?

Question 7

Implement the method `reverse()`, which should reverse the list.

What is the complexity of this method?

Question 8

Implement the `Iterator` class from `SinglyLinkedList`, including its `insert()` and `remove()` methods. Please note that this `Iterator` class does not implement the standard `Iterator` interface, but follows the same conventions. For instance, `insert()` should insert an element before the element that would be returned by `next()`, while `remove()` should remove the element that was previously returned by `next()`. For more information about iterators, consult Chapter 6 of the textbook. Please note that in order to implement `remove()` efficiently you may have to change the representation of your data structure.

What is the complexity of all the implemented methods?

Question 9

Copy your implementation of `SinglyLinkedList` into `SinglyLinkedListCol` and modify it to implement the standard Java collections API.

CHALMERS TEKNISKA HÖGSKOLA AB
GÖTEBORGS UNIVERSITET
Data- och informationsteknik
412 96 Göteborg
Telefon: 031-772 10 00 (CTH) / 031-786 00 00 (GU)

Question 10

Create an interface for a stack data structure called `MyStack`, and modify `SinglyLinkedList` to implement this interface.

What is the complexity of all the implemented methods?

Question 11

This question concerns the standard Java `LinkedList` data type. The code below attempts to create a list of 100 integers, and then filter out some of them. Unfortunately, it does not work. Explain what is the problem, and how to fix it. **Note:** To solve this question you need information that was not covered in the lectures. The code is available in the `ModError.java` file.

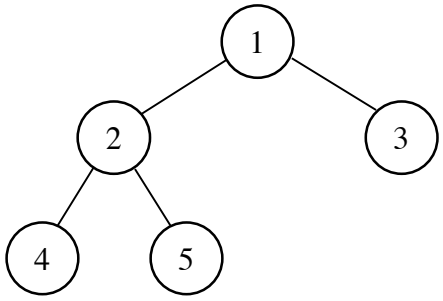
```
List<Integer> numbers = new LinkedList<Integer>();
for(int i = 0; i < 100; ++i) numbers.add(i);

for (Integer x : numbers) {
  if (x % 10 == 0)
    numbers.remove(x);
}
```

Trees

In this part of the assignment, you will work starting from the `Tree.java` skeleton file. The `Tree` class defined in the file represents a generic binary tree with labels in all nodes. You may assume that the depth of the tree is at most $O(\lg n)$.

Question 12

Implement the method `printDFS()`, which should print the labels of the tree's nodes in depth-first (DFS) order. The labels should be separated by new-lines, and may be printed using the `toString()` method, which is available in every class with which `Tree` can be instantiated.



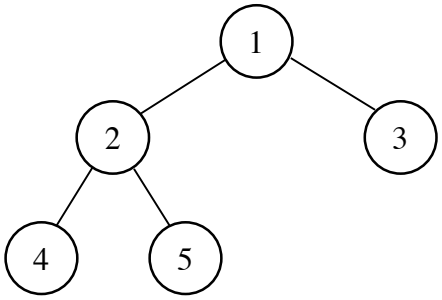
For example, for the graph above, the method should print the following lines:

```
1
2
4
5
3
```

What is the complexity of this method?

Question 13

Implement the method `printBFS()`, which should print the labels of the tree's nodes in breadth-first order. The labels should be separated by new-lines.



For example, for the graph above, the method should print the following lines:

```
1
2
3
4
5
```

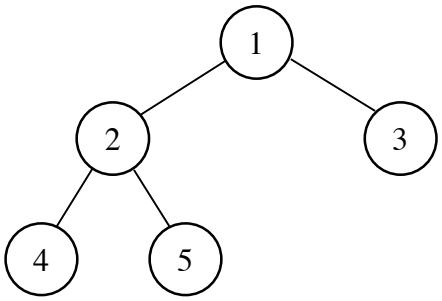
You should use the standard `ArrayDeque` implementation of the queue to solve this question.

What is the complexity of this method?

Question 14

Implement the method `buildDFS(Iterable<Item>)`, which takes a sequence of elements as argument (the interface `Iterable<Item>` can be instantiated by an array, a linked list, or another collection) and creates a complete tree whose DFS traversal results in the original sequence of elements.

For example, for the sequence `{1, 2, 4, 5, 3}`, the method should create the following tree:



Question 15

Implement the method `nthDFS(int)`, which returns the n-th element in DFS order.

CHALMERS TEKNISKA HÖGSKOLA AB
GÖTEBORGS UNIVERSITET
Data- och informationsteknik
412 96 Göteborg
Telefon: 031-772 10 00 (CTH) / 031-786 00 00 (GU)

Question 16

Implement the method `insertBST(Item i)`, which should insert item *i* into a binary search tree (BST) (that is: assumes that the tree is a BST, and produces another BST). If the item already exists in the tree, another copy of it should be inserted.

Question 17

Implement the method `insertBST(Item i)`, which should remove item *i* from a binary search tree (that is: assumes that the tree is a BST, and produces another BST). If the item does not exist in the tree, the method should throw an exception.

CHALMERS TEKNISKA HÖGSKOLA AB
GÖTEBORGS UNIVERSITET
Data- och informationsteknik
412 96 Göteborg
Telefon: 031-772 10 00 (CTH) / 031-786 00 00 (GU)