DIT181 Data Structures and Algorithms: Assignment 2

Maximum points you can get from Assignment 2 is **100 points**. Distribution of points within questions has been given in parentheses below.

- **Criteria to pass "Assignments" sub-course (3.5 HEC)**: From each assignment (i.e. Assignment #1, Assignment #2) you have to get minimum 50 points out of 100 points.

Information about Deliverables: You will submit a pdf file for the answers to questions that do not require programming. For the answers consisting of implementation in Java, you will submit the .java files that you worked on and completed. Please ensure that you follow these submission instructions while submitting your Assignment#2 on Canvas.

Please, make sure that your code is compilable and runnable.

  - Work on the skeleton codes provided.
  - Do not rename or alter anything in the predefined public classes/constructors/methods in the skeleton codes. Please do not change anything that we already implemented in the skeleton files. Such attempt will break the unit tests.
  - Do not change the name of the skeleton files you are working on.
  - Don't bundle any of your code in packages. Please, just submit java files for questions that require coding.
    - For **Question 2**, you will work on `RPN.java` skeleton file and submit the final version of `RPN.java`.
    - For **Questions 3-6**, you will work on `SinglyLinkedList.java` skeleton file and submit the final version of `SinglyLinkedList.java`.
    - For **Questions 7-9**, you will work on `Tree.java` skeleton file and submit the final version of `Tree.java`.

- For answers that do not require programming, make sure that you submit a *PDF file* for your written/scanned work. This is due to that pdf is generally more accepted format for reading. Furthermore, it preserves the formatting.
  - [THIS IS A RECOMMENDATION] We would highly recommend using Latex for scientific formatting. An easy way to get going: www.overleaf.com (Links to an external site.)Links to an external site. , if you like online editing.
- When you submit, place all relevant files (only .pdf and .java files) in a folder and name the folder "assignmentX_groupY" where the X is the assignment number and Y group number. Compress (i.e., zip, but NOT rar) the folder, to obtain "assignmentX_groupY.zip" file. Submit "assignmentX_groupY.zip" file on Canvas. Make sure not to include any hidden files (especially from macOs, see the following link: https://superuser.com/questions/757593/what-is-ds-store-file-in-windows)

Submission Deadline: 06.03.2020 at 23:59 pm SHARP!!

Stacks and Queues

Question 1 Show how to implement a queue using 2 stacks (no Java code is needed, just a sketch and pseudo code) **(11 points)**. What are the complexities of `enqueue()` **(2 points)** and `dequeue()` operations **(2 points)**?

Hint: A stack is a data structure with `push()`, `pop()`, and `isEmpty()` operations; a queue is a data structure with `enqueue()`, `dequeue()` and `isEmpty()` operations.

Question 2 (20 points)

This question will require that you work on `RPN.java` skeleton file.

Your job is to implement an RPN calculator that will read from the input terminal (i.e., standard input) and print the results on the 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 is typed by the user, whereas the other lines are output by the calculator. After processing a line of input, the calculator should print the current contents of its evaluation stack, or quit if the word `quit` appears. You should start with the `RPN.java` skeleton file, which currently contains code that reads input and prints messages on the standard output. The calculator should support the operators `+`, `-`, `*` and `/` on integers. [You should use the standard `Stack` collection from Java]

Hint: Reverse Polish Notation (RPN) is a way of representing arithmetic expression, which does not require parentheses. See these resources for more explanations:

- https://www.youtube.com/watch?v=7edoVNuwGvc
- http://mathworld.wolfram.com/ReversePolishNotation.html
- https://en.wikipedia.org/wiki/Reverse_Polish_notation
- http://hp15c.com

Linked Lists

Questions from this part will require that you work on the `SinglyLinkedList.java` skeleton file.

Question 3
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 **(8 points)**. What is the complexity of this method? **(2 points)**

Question 4
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 **(8 points)**. What is the complexity of this method? **(2 points)**

Question 5
Implement the method `removeAt(int n)`, which should remove an element at index n from the list. If the index is out of bounds, the exception `IllegalArgumentException` should be thrown **(8 points)**. What is the complexity of this method? **(2 points)**

Question 6
Implement the method `reverse()`, which should reverse the list **(8 points)**. What is the complexity of this method? **(2 points)**

Trees

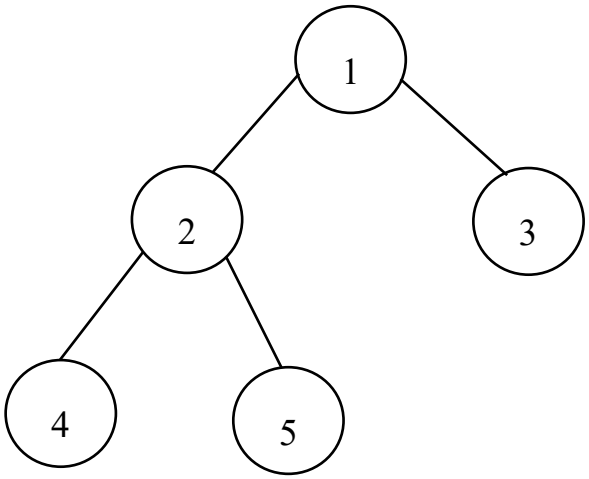
In this part of the assignment you will work on the `Tree.java` skeleton file. The `Tree` class defined in the file represents a generic binary tree with labels in all nodes.

Hint: You may assume that the depth of the tree is at most $O(\log_2 n)$.

Question 7
Implement the method `nthBFS(int n)`, which returns the nth element in Breadth First Search (BFS) order ("Breadth First Search" is also known as "Level Order Search") **(6 points)**.

Question 8
Implement the method `DFSToString()`, which should return a string of the labels of the tree's nodes in **pre-order** depth-first-search (DFS). The labels should be separated by comma. **(7 points)**.

For example, for the graph below, the method should return the string:
`1,2,4,5,3`



What is the complexity of this method? **(2 points)**

Question 9

Implement the method `insertBST(Item item)`. The method assumes that the tree is a binary search tree (BST) and inserts item i into it. When the item is inserted the method should modify the rest of the tree so that the tree is still a BST. **(7 points)**. If the item already exists in the tree, another copy should be inserted. **(3 points)**. Hint: You might want to create an auxiliary method for this.