DS2030 Data Structures and Algorithms for Data Science
Lab Exam 1 Due on September 24, 5.00pm

Instructions

- You are to use Python as the programming language. Use may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet.
- The submission must be a zip file with the following naming convention - rollnumber.zip. All the 4 Python files (q1.py, q2.py, q3.py and q4.py) should be contained in a folder named after your rollnumber. **Any deviation from the naming convention will cost you 50% penalty.**
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code.
- Upload your submission to moodle by the time. Do not email the submission to the instructor or the TA.

1 Find the Minimum Element in an Array(5 points)

Given an array A of N elements, implement a Python function that finds the minimum element in the array in one pass. Complete the code provided in q1.py. **Do not rename the file.** Your task is to complete the function `find_min_element()`. Ensure that your function works for arrays of varying sizes, including empty arrays (for which you should handle the case appropriately). You cannot use any inbuilt Python function for finding the minimum.

2 Find the Second Minimum Element in an Array(10 points)

Given an array A of N elements, implement a Python function that *finds the second minimum element in the array in one pass without sorting the array.* Complete the code provided in q2.py. **Do not rename the file.** Your task is to implement the function `find_second_min_element()`. Ensure that your function works for arrays of varying sizes, cases where the array might have duplicates or too few elements. You cannot use any inbuilt Python function for finding the minimum.

3 Implementing a Queue Using Stacks (15 points)

In this question you are asked to implement all the functionalities of a Queue using stacks instead of implementing it as a Python List. The starter code for this problem contains the stack implementation. Your objective is to implement the functions in the QueuesUsingStacks Class in the file q3.py. **Do not rename the file.** We have already provided you access to two stacks inside the QueueUsingTwoStacks class. Figure out the logic for utilizing these two stacks to store the queue items and support the queue operations. Specifically implement the following functions.

1. `enqueue (self, item)`: adds item to the queue.
2. `dequeue(self)`: removes and returns the first inserted item from the queue.
3. `is_empty(self)`: returns True if the queue is empty.
4. `peek(self)`: returns the front item of the queue without removing it.

4 Binary Trees (20 points)

You are given a text file representing the structure of a binary tree. The first line of the file contains the total number of nodes in the tree. Each subsequent line describes one node using the following format:

node number parent node number left child number right child number

The node numbers start from 1. If any entry is 0, it indicates that the corresponding child does not exist. You may assume that root is always represented by the number 1. Your goal is to build the binary tree based on this input and write a function that recursively swaps the left and right children of every node.

Tasks

Your main objectives for this lab are:

1. Implement a function `parse_tree` to read the binary tree from a file and construct it using lists.
2. Implement a function `swap_children` to recursively swap the left and right children of every node in the binary tree.
3. Verify your implementation using pre-order traversal.

You have to complete the implementations of the functions

- `parse_tree`
- `swap_children`
- `pre_order_traversal`

based on the structure of the binary tree in the starter code q4.py. **Do not rename the file.**

Example Input File

An example file `tree.txt` could look like this:

```
5
1 0 2 3
2 1 4 0
3 1 0 5
4 2 0 0
5 3 0 0
```

This file describes a binary tree with:

- Node 1 as the root.
- Node 2 as the left child of Node 1.
- Node 3 as the right child of Node 1.
- Node 4 as the left child of Node 2.
- Node 5 as the right child of Node 3.