Lab 05 Specification – A hands-on exercise to practice tree data structure and implement sorting based on trees.
50 points

Lab Goals

- Implement the Tree data structure and a series of properties connected to Trees.
- Practice an implementation of sorting using a Tree data structure.
- Develop Heapify process using a programming language such as Java and integrate the heapify process to perform Heap Sort.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

Learning Assignment

If not done previously, it is strongly recommended to read all of the relevant "GitHub Guides", available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading assignment is useful to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, it is also recommended to do the reading assignment from the section of the course textbook outlined below:

- **Sedgewick chapter 02, 2,4**

Assignment Details

Now that we have discussed some basics of tree data structure and heap sort algorithm and analyzed them together in the last few lectures, it is now time to do it practically. In this lab, students will practice developing a variety of features to the tree data structure to retain the knowledge from the class discussions so far. This also includes developing the heapify procedure and the heap-sort implementation using a programming language such as Java.

The lab has two parts. The first part is required to be completed by the end of the lab session. The second part is required to be completed within a one week time period. Please make sure to push your code file(s) and commit by the deadline provided in each of the following section(s).

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.

2. Submitting a copy of the other's program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill tests, exams, etc.

At any duration during and/or after the lab, students are recommended to team up with the Professor and the Technical Leader(s) to clarify if there is any confusion related to the lab and/or class materials.

The Professor proof-read the document more than once, if there is an error in the document, it will be much appreciated if student(s) can communicate that to the Professor. The class will be then informed as soon as possible regarding the error in the document. Additionally, it is highly recommended that students will reach out to the Professor in advance of the lab submission with any questions. Waiting till the last minute will minimize the student chances to get proper assistance from the Professor and the Technical Leader(s).

Section 1: In-Lab Exercise (25 points)

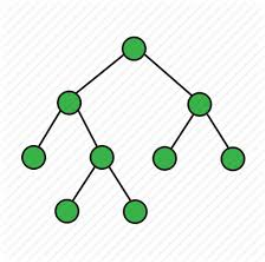
Attendance is in general a required component in computer science class(es) and laboratory session(s). By attending these sessions regularly, students get an opportunity to interact with the Professor and the Technical Leaders(s). This interaction helps maximize the student's learning experience. Attendance is taken in the form of Mastery Quizze(s) and by checking the commit log of the student's In-Lab exercise submission. No student is allowed to take the Mastery Quizze(s) outside the laboratory room unless prior arrangements are done with the Professor. It is against the class honor code to violate the policy outlined above.

Part 01 - Attendance Mastery Quiz (5 points)

The Mastery Quiz is graded based on a Credit/No-credit basis. Students are required to complete the Mastery Quiz only in the laboratory room, which is in **Alden 109** before 3:15 PM. Submissions after 3:15 PM is not acceptable. Students who had completed the Mastery Quiz according to the policy outlined above will automatically receive (5) points towards their lab grade. The link to the Mastery Quiz is provided below:
`https://forms.gle/9P9sr7C1rd8vVqCt5`

Students are required to complete both part 02 and 03 only in the laboratory room, which is in **Alden 109**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is late submission accepted for this part, according to the late policy outlined in the course syllabus.

Part 02 - A Simple Exercise to develop a series of features in a tree data structure. (20 points)



We had seen in detail the different terminologies connected to the tree data structure and binary trees. Complete the requirements outlined below:

1. For simplicity, a starter-code in `Trees.java` is provided in the lab repository.

   ADD the following statement to all your submission files including the file named `Trees.java`.

2. **Please make sure to refer to the lesson 9 slides to refresh your memory on the basic terminologies on Trees.**

3. Analyze the code provided. This code generates an array of numbers and prompts the user to pick a node to explore.

4. Develop the methods called `getChildren and getParent` based on in-class activity. This is the starting point for all other implementations in this section. It is required to have a solid understanding of how the parent and children are identified. Please refer to the code shared in the class Slack page.

5. Develop a new method called `getAncestors`. This method should accept an array and the node to be explored as the inputs. Add the required logic to output all the ancestors.

6. Develop a new method called `getDescendents`. This method should accept an array and the node to be explored as the inputs. Add the required logic to output all the descendents.

7. Develop a new method called `findDepth`. This method should accept an array and the node to be explored as the inputs. Add the required logic to output the depth of the node in the context of the tree.

8. Develop a new method called `findHeight`. This method should accept an array and the node to be explored as the inputs. Add the required logic to output the height of the node in the context of the tree.

9. A simple validation should be implemented. If there are no ancestors or descendents, this should be identified and displayed on the console. This is already partially implemented in the starter-code. It is required to do a slight extension to the starter-code to implement this part. What is that slight extension? Figure it out.

10. Please make sure to read all the comments in the starter-code so as to make sure to understand the requirements fully.

11. **Optional:** Develop a new method called `isBalanced` to identify if a given tree is balanced or unbalanced. Add the required logic to output the status of the tree. True for balanced and False for unbalanced. A bonus point will be applied to the student's final grade towards the end of the semester for those who had fully completed this optional part before **4:20 PM**. A bonus point will be a 1 extra point added towards the final score of the student's grade. Professor has a record of all student's who received bonus points. This will be directly updated in the final grade.

Part 03 - Answer the following question to record your understanding of Trees. (10 points)

1. **ADD** the following statement to all your submission files including the file named `tree-analysis`. The file can be developed using a mark down file and/or PDF file.

   This work is mine unless otherwise cited - Student Name

2. q1: What are the limitations of implementing a tree data structure using arrays?
3. q2: What are the benefits of implementing a tree data structure using arrays?
4. q3: What are the benefits and limitations of a balanced tree?
5. q4: Explain the differences between a proper, at most, and an in-complete binary tree?
6. q5: Heapify the array [1,2,3,4,5,6,7,8] to a max heap? Just show your final output array after the heapify process.

Section 2: Take Home Exercise (25 points)

Students are recommended to get started with this part in the laboratory room, by discussing ideas and clarifying with the Professor and the Technical Leader(s). All the part(s) in this section is due in **one week** time. It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. The deadline for submitting the part(s) in this section is **28th Feb 2020, 8:00 AM**. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

Part 01 - A Simple Exercise to design and develop the heap sort algorithm.(25 points)

There is only one part in this section. So please don't under estimate the amount of work that should be done in this section. It is important for student's to budget in extra time so as to make sure that all the implementation are done correctly. More importantly to fully understand how to implement and solve Heap sort.

We had seen in detail how to perform and analyze the heapify process and the Heap sort algorithm. I will go over one more time during the Monday class session on Heap Sort algorithm discussed in slides. It is now your turn to implement it in Java programming language. Complete the requirements outlined below:

1. For simplicity, a starter-code in `HeapSort.java` is provided in the lab repository.

   ADD the following statement to all your submission files including the file named `HeapSort.java`.

2. **Please make sure to refer to the lesson 9 slides to refresh your memory on the heapify procedure and the heap sort algorithm.**
3. Analyze the code provided. This code generates an array of numbers and does the respective calls to a series of methods so as to perform Heap Sort.

4. Develop the method called `heapify` based on the algorithm discussed in class and in lesson 9 slides.
5. Develop the method called `heap-sort` based on the algorithm discussed in class and in lesson 9 slides. The output of this method should return the sorted array.
6. Develop the method called `fix-it`, which may be called from the `heapify` method if there occurs a violation in one or more of the ancestors.
7. **Note:** Step by step instructions on how to implement heap sort is left out from the specification. It is expected that a student should figure out on their own to construct the implementation of heap sort at this point. This is a challenge for students to take up and expand their knowledge on tree data structure from Section 1. By taking up this challenge, a student can make sure to fully understand the heap sort algorithm.

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. "tree-analysis" document file.
2. Commented source code from the "Trees.java" program.
3. Commented source code from the "HeapSort.java" program.
4. It is highly important, for you to meet the honor code standards provided by the college and to ensure that the submission is made before the deadline. The honor code policy can be accessed through the course syllabus. Make sure to add the statement "This work is mine unless otherwise cited." in all your deliverables such as source code and PDF files. In your summary file, please make sure to include your name and your partner's name.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits may be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your GitHub repository will lead to receiving no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn't compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student's responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student's submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If a student needs any clarification on their lab grade, it is strongly recommended to talk to the Professor. The lab grade may be changed if deemed appropriate.


Wish you
all the best
alamy stock photo