Dept. of Computer Science & Software Eng., Concordia University
COMP 6481 --- Fall 2024

Programming and Problem Solving

Assignment 3 --- Due Saturday, November 30, 2024

Part I

Please read carefully: You must submit the answers to all the questions below. However, this part will not be marked. Nonetheless, failing to submit this part fully will result in you missing 50% of the total mark of the assignment.

Question 1

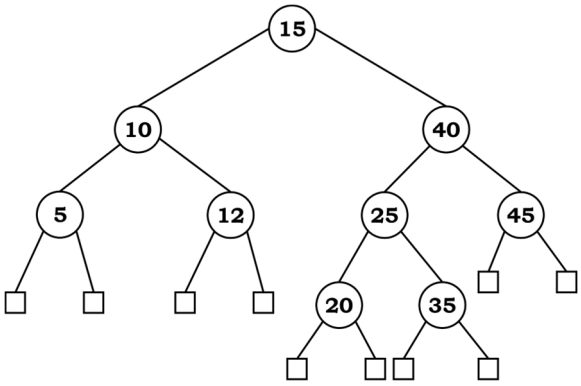
You are given a hash table with size 7 and a hashing function h(k) = k%7. Output the state of the table after inserting the below mentioned values in that specific order when collisions are handled using:

a. Separate chaining
b. Linear probing
c. Double hashing using a second hash function h'(k) = 5 – (k % 5)

The values to be inserted are 19, 26, 13, 48, 17.

Question 2

Consider the below mentioned AVL tree.



a) Insert key 25 and re-balance the tree if required. Show the progress by drawing each state of the tree and the final state after the insertion.

b) Remove node with value 40 from the original tree and re-balance the tree if required. Show the progress by drawing each state of the tree and the final state after the deletion.

Question 3

Develop a **well-documented pseudo code** to create a method for open addressing which is like double hashing but instead of using a second hash function use a pseudorandom number generator seeded by the key.

a. Will this be more efficient approach compared to double hashing?
b. Will you prefer this over double hashing?

Part II

Purpose: The purpose of this assignment is to provide hands-on practice with key object-oriented programming concepts, including recursion, tree-based data structures, and file system simulations. You will work with custom classes, recursive methods, and hierarchical structures, as well as enhance your understanding of data organization and manipulation. Additionally, you will gain experience in designing and implementing file-like systems using tree structures to solve real-world problems efficiently.

"It is not that I'm so smart. But I stay with the questions much longer."

-- Albert Einstein

Imagine you are part of a team tasked with building a simplified file system for a new operating system. The core functionality of the system must allow users to organize their files and directories in a hierarchical structure, similar to how modern file explorers or command-line interfaces work. Your job is to design and implement a data structure that simulates this file system.

In the real world, a file system is a crucial part of an operating system. It's what allows us to store, retrieve, and organize files on a hard drive or cloud storage. Under the hood, file systems use data structures like trees to manage directories and files efficiently.

Now, let's picture the challenge from the perspective of the developer: You need to create a "Directory" class that will represent the folders and files of the file system. But this isn't just any basic data structure. Your directory class needs to be able to:

- Add files and subdirectories.
- Keep track of the sizes of files and subdirectories.
- Search through nested subdirectories to find specific files.
- Print the directory structure in a readable format, just like a file explorer would display files and folders.

To make things even more exciting, you're required to implement this directory system in recursive fashion, as many file systems use trees for this kind of hierarchical structure. The recursive nature of the problem mirrors how files can be organized within multiple levels of subdirectories.

With this in mind, you'll dive into this assignment, which will provide a great prologue to recursive thinking, data structures, and how software interacts with file systems at a conceptual level. By the end of the project, you'll have built a simplified but functional version of a file system that can efficiently store and retrieve files, as well as display them in a nested, human-readable format.

Objective:

Directory Structure: Implement a class that simulates a file system, where a directory may contain files and subdirectories.

Operations on Directories and Files:

- Add files or subdirectories to an existing directory.
- Calculate the total size of a directory (files and subdirectories).
- Recursively search for a specific file by name.
- Display hierarchical structure of directories and files in a human-readable format.

Task#1 – Implement the Directory Class

In this task, you will create a Directory class to represent directories and files. Each directory can contain files or subdirectories. Files are represented by strings for now.

- **Class Attributes**:
  - *String name*: The name of the directory or file.
  - *boolean isFile*: A boolean indicating if a node is a file (true) or a directory (false).
  - *List<Directory> children*: A list that holds subdirectories and files for directories.
- **Constructor**:
  - Takes the name of the directory or file and a flag indicating whether it is a file or a directory.

Task#2 – Implement Methods to Add Files and Subdirectories, and Print Directory Structure

In this task, you will implement two key methods for adding files and subdirectories and for printing the directory structure recursively.

- ***addFile***(String fileName) – Adds a file to the current directory.
- ***addDirectory***(String dirName) – Adds a subdirectory to the current directory.
- ***printDirectoryStructure***() – Recursively prints directory structure in a readable format (nested).

```
root
  documents
    work
      report.docx
      presentation.pptx
    personal
      resume.pdf
  music
    rock
      album1
        song1.mp3
        song2.mp3
    pop
      album2
        song3.mp3
  downloads
    software
      program1.exe
      program2.exe
    images
      image1.jpg
      image2.png
  readme.txt
```

Figure 1. Sample output for printDirectoryStructure

Task#3 – Implement getSize() and findFile() Methods

In this task, you will implement two important methods:

- ***getSize***() – Returns total size of the directory, including all files and subdirectories.
  - Each file has a size of 1.
  - Each directory's size is the sum of its contents (files and subdirectories).
- ***findFile***(String fileName) – Recursively searches for a file by name and returns the full path if found.

Task#4 – Implement Recursive Structure and Special Cases Handling

- **Recursive Structure**: Ensure that all methods traversing the directory structure (such as ***getSize***(), ***findFile***(String fName), and ***printDirectoryStructure***()) are implemented recursively.
- **Special Cases**:
  - Handle cases when directories are empty.
  - Ensure proper handling when searching for non-existent files.
  - Consider scenarios where directories or files are nested at multiple levels.

Create a DirectoryDriver class. The DirectoryDriver class will demonstrate functionality of Directory class by performing various operations on a simulated file system. It will serve as a test harness to ensure the correctness of the implemented directory structure and operations, including file and directory creation, recursion, printing, file search and size calculation.

1. ***Create Root Directory***:
   - Initialize a root directory, which will be the starting point for all operations.
2. **Add Subdirectories and Files**:
   - Add at least 9 subdirectories to the root directory.
   - Add multiple files to both the root directory and various subdirectories.
3. **Print Directory Structure**:
   - Print the directory structure recursively, showing all files and subdirectories with proper indentation to reflect the hierarchy.
4. **Calculate Directory Size**:
   - Print the total size (number of files) of the root directory and subdirectories. The size of a directory should include all files within it, including files inside nested subdirectories.
5. **Search for Files**:
   - Search for a specific file by name (using the findFile() method) in the entire directory structure and print the full path to the file if found.
   - Attempt to search for a file that does not exist and handle the case gracefully by printing an appropriate message ("File not found").
6. **Special Case Handling**:
   - Handle cases where directories are empty (no files or subdirectories).
   - Handle cases where subdirectories have files but no other subdirectories.

Some general information:

a. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!
b. Again, your program must work for any similar DirectoryDriver file. The structure provided with this assignment is only one possible version, and you must not consider it as the general case when writing your code.

SUBMISSION INSTRUCTIONS

Submission format: All assignment-related submissions must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only – no abbreviations/nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names. **IMPORTANT:** For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You **must** attend the demo and be able to explain their program to the marker. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time slot per group).

Now, please read very carefully:

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**

EVALUATION CRITERIA

IMPORTANT: Part I must fully be submitted. Failure to submit that part will cost 50% of the total marks of the assignment!

Total	**10 pts**
Documentations	**1 pt**
JavaDoc documentations	1 pt
Tasks	**9 pts**
Task#1	**1 pt**
- Implement the Directory class with attributes	1 pt
Task#2	**2 pts**
- Implement addFile and addDirectory methods	1 pt
- Implement printDirectoryStructure()	1 pt
Task#3	**3 pts**
- Implement getSize() method	1 pt
- Implement findFile method	2 pts
Task#4	**3 pts**
- Implement proper recursive structure and special cases handling	3 pts