HOBTorrent Challenge Briefing

Briefing

Disaster strikes the world as a large cyber attack is initiated by the evil hacker Games Jover, leading to significant data losses. Nations are in panic as critical systems shut down and major communication mediums are destroyed. To quote recently appointed CEO of Google, Bemily Ennett, "What a tragedy... I haven't seen this big a catastrophe since I did assignment 4 of CSSE2310".

Tech giant HOB Technologies' state-of-the-art DoS detection and mitigation system, Ark, is capable of tracking Games Jover and reverting his attack. However, with the company's networks destroyed by the attack, the company is unable to securely and efficiently communicate this confidential information about the attack to Australian Law Enforcement.

HOB Technologies has approached your company with a project requiring the design of an efficient and secure peer-to-peer communication system, named HOBTorrent. However - your company was not the only one approached and to score the contract you will need to find the best design for this system.

Challenge

A representative from HOB Technologies' board of directors is present today to oversee your company's operations and provide guidance. Following the briefing from this representative, you are required to execute the following key steps to score the contract with HOB Technologies:

1. Form your company, determine a company name, and report back to the overseeing director with this name. You will be advised by the overseer on the number of employees your company should have.
2. Commence design of the system, as per the specifications on the other page. Write the pseudocode of each method in the system.
3. As you complete each method within the system, report back to the overseeing director. Correct, efficient, and simple solutions will be awarded victory points by HOB Technologies' representative.
4. Gain the most victory points, and you will score the contract with HOB technologies! Your help may even lead to you becoming global heroes!

System Specifications

The HOBTorrent system is made up of multiple nodes, each with a unique integer identifier. Each node contains a collection of files. Nodes are connected as neighbours, and the entire HOBTorrent network can be represented as an undirected and unweighted graph.

You are to design a class for a single HOBTorrent node. You may introduce additional member variables and/or helper methods if necessary. You can assume common algorithms and/or data structures are already implemented for you. You can attempt these tasks in any order but it is strongly recommended you implement the core functionality tasks first. You are not expected to complete all four tasks within the time you have, but choose the tasks you try first carefully, as the difficulty will vary between tasks (difficult tasks will have more opportunities to score victory points).

Core File Storage Functionality

Difficulty: Easy

To start with, you are required to implement the following core methods:

- An appropriate constructor for the node class, which takes a single integer parameter representing the node's ID. This may need to be extended later.
- *addFile(name)* - takes the name of a file (a string) and adds it to the node's local file collection. A single node may be able to store duplicates of the same file.
- *hasFile(name)* - returns a boolean indicating whether the system contains the given file name

Example:

```
T = HOBTorrentNode(1)
T.addFile("ArrayGrid.java")
T.hasFile("ArrayGrid.java") -> true
T.hasFile("Algorithms.java") -> false
```

Network Communication

Difficulty: Medium

Transmitting files through the network will be one of the most important features of HOBTorrent. The following methods should be able to be performed. You may also write a *getNeighbours()* method as a helper for either of these methods, although it is not a strict requirement.

- *addNeighbour(node)* - record another node as a neighbour of this node (that is, create a bidirectional P2P link between this node and the given node).
- *getNodeWithFile(name)* - return the ID of the node (which may be this node) that stores the given file name and can be reached through a series of P2P links. If multiple nodes contain the same file, return the node that is the closest. If no node contains the file, return -1.

Example:

```
T1 = HOBTorrentNode(1)
T2 = HOBTorrentNode(2)
T3 = HOBTorrentNode(3)
T1.addNeighbour(T2)
T2.addNeighbour(T3)
T2.addFile("BinaryTree.java")
T3.addFile("BinaryTree.java")
T3.addFile("TreeIterator.java")

T1.getNodeWithFile("TreeIterator.java") -> 3
T1.getNodeWithFile("BinaryTree.java") -> 2
T1.getNodeWithFile("StandardTree.java") -> -1
```

File System Cleanup

Difficulty: Hard

HOBTorrent is no ordinary torrent system. It is also capable of cleaning up duplicate files that exist. Implement the *cleanup* method, which takes no parameters, and removes one copy of the most duplicated file from the node's local file collection (i.e. the most frequently-occurring file name). You should remove the most recently-added copy of the file. The name of the file that was removed should be returned. If multiple files have the same frequency, the file that was added most recently should be removed. You can assume the file system will contain at least one file.

Example:

```
T = HOBTorrentNode(1)
T.addFile("FeedAnalyser.java")
T.addFile("FeedAnalyser.java")
T.addFile("FeedItem.java")

T.cleanup() -> "FeedAnalyser.java"
T.cleanup() -> "FeedItem.java"
T.cleanup() -> "FeedAnalyser.java"
```

File Upload

Difficulty: Hard

For the final task, you will need to write a function that chooses an appropriate file upload speed for a node. Users should be able to upload a collection of files of varying sizes to a single node, however faster upload speeds are very expensive for the operator so it is essential that the upload speed is minimised. You are to design a method: *getUploadSpeed(int[] files, int intakes)*. This method takes an array of integers, representing the individual sizes of the files to upload (in bytes) and the expected number of intakes to upload all files. Each intake can upload as many files as possible, but an intake cannot upload a portion of a file (only whole files). Files need to be uploaded in the given order. Your algorithm should determine the smallest possible upload rate (in bytes per intake) that allows all files to be uploaded in the given number of intakes.

You can assume 1 <= intakes <= files.length <= 50000 and 1 <= files[i] <= 500.

Example:

```
T = HOBTorrentNode(1)

T.setUploadSpeed([3, 2, 2, 4, 1, 4], 3) -> 6
/* Explanation:
 * First intake we upload files with size 3 and 2 (total 5)
 * Second intake we upload files with size 2 and 4 (total 6)
 * Third intake we upload files with size 1 and 4 (total 5)
 */

T.setUploadSpeed([1, 2, 3, 1, 1], 4) -> 3
/* Explanation:
 * First intake we upload a file with size 1
 * Second intake we upload the file with size 2
 * Third intake we upload the file with size 3
 * Fourth intake we upload the two remaining files with size 1 (total 2)
 */
```

The world is counting on you! Good luck!