Problem 588: Design In-Memory File System

Problem Information

Difficulty: Hard
Acceptance Rate: 48.28%
Paid Only: Yes
Tags: Hash Table, String, Design, Trie, Sorting

Problem Description

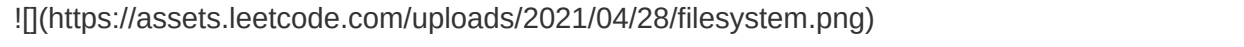
Design a data structure that simulates an in-memory file system.

Implement the FileSystem class:

* `FileSystem()` Initializes the object of the system. * `List<String> ls(String path)` * If `path` is a file path, returns a list that only contains this file's name. * If `path` is a directory path, returns the list of file and directory names **in this directory**. The answer should in **lexicographic order**.

* `void mkdir(String path)` Makes a new directory according to the given `path`. The given directory path does not exist. If the middle directories in the path do not exist, you should create them as well. * `void addContentToFile(String filePath, String content)` * If `filePath` does not exist, creates that file containing given `content`. * If `filePath` already exists, appends the given `content` to original content. * `String readContentFromFile(String filePath)` Returns the content in the file at `filePath`.

Example 1:



Input ["FileSystem", "ls", "mkdir", "addContentToFile", "ls", "readContentFromFile"] [[], ["/"], ["/a/b/c"], ["/a/b/c/d", "hello"], ["/"], ["/a/b/c/d"]] **Output** [null, [], null, null, ["a"], "hello"] **Explanation** FileSystem fileSystem = new FileSystem(); fileSystem.ls("/"); // return [] fileSystem.mkdir("/a/b/c"); fileSystem.addContentToFile("/a/b/c/d", "hello"); fileSystem.ls("/"); // return ["a"] fileSystem.readContentFromFile("/a/b/c/d"); // return "hello"

Constraints:

* `1 <= path.length, filePath.length <= 100` * `path` and `filePath` are absolute paths which begin with `'/'` and do not end with `'/'` except that the path is just `"/"`. * You can assume that all directory names and file names only contain lowercase letters, and the same names will not exist in the same directory. * You can assume that all operations will be passed valid parameters, and users will not attempt to retrieve file content or list a directory or file that does not exist. * You can assume that the parent directory for the file in `addContentToFile` will exist. * `1 <= content.length <= 50` * At most `300` calls will be made to `ls`, `mkdir`, `addContentToFile`, and `readContentFromFile`.

Code Snippets

C++:

```
class FileSystem {
public:
FileSystem() {

}

vector<string> ls(string path) {

}

void mkdir(string path) {

}

void addContentToFile(string filePath, string content) {

}

string readContentFromFile(string filePath) {

}
};

/**
* Your FileSystem object will be instantiated and called as such:
* FileSystem* obj = new FileSystem();
```

```
* vector<string> param_1 = obj->ls(path);
* obj->mkdir(path);
* obj->addContentToFile(filePath,content);
* string param_4 = obj->readContentFromFile(filePath);
*/
```

Java:

```
class FileSystem {

public FileSystem() {

}

public List<String> ls(String path) {

}

public void mkdir(String path) {

}

public void addContentToFile(String filePath, String content) {

}

public String readContentFromFile(String filePath) {

}
}

/**
* Your FileSystem object will be instantiated and called as such:
* FileSystem obj = new FileSystem();
* List<String> param_1 = obj.ls(path);
* obj.mkdir(path);
* obj.addContentToFile(filePath,content);
* String param_4 = obj.readContentFromFile(filePath);
*/
```

Python3:

```
class FileSystem:

def __init__(self):


def ls(self, path: str) -> List[str]:


def mkdir(self, path: str) -> None:


def addContentToFile(self, filePath: str, content: str) -> None:


def readContentFromFile(self, filePath: str) -> str:



# Your FileSystem object will be instantiated and called as such:
# obj = FileSystem()
# param_1 = obj.ls(path)
# obj.mkdir(path)
# obj.addContentToFile(filePath,content)
# param_4 = obj.readContentFromFile(filePath)
```