<https://leetcode.ca/all/588.html>

Design an in-memory file system to simulate the following functions:

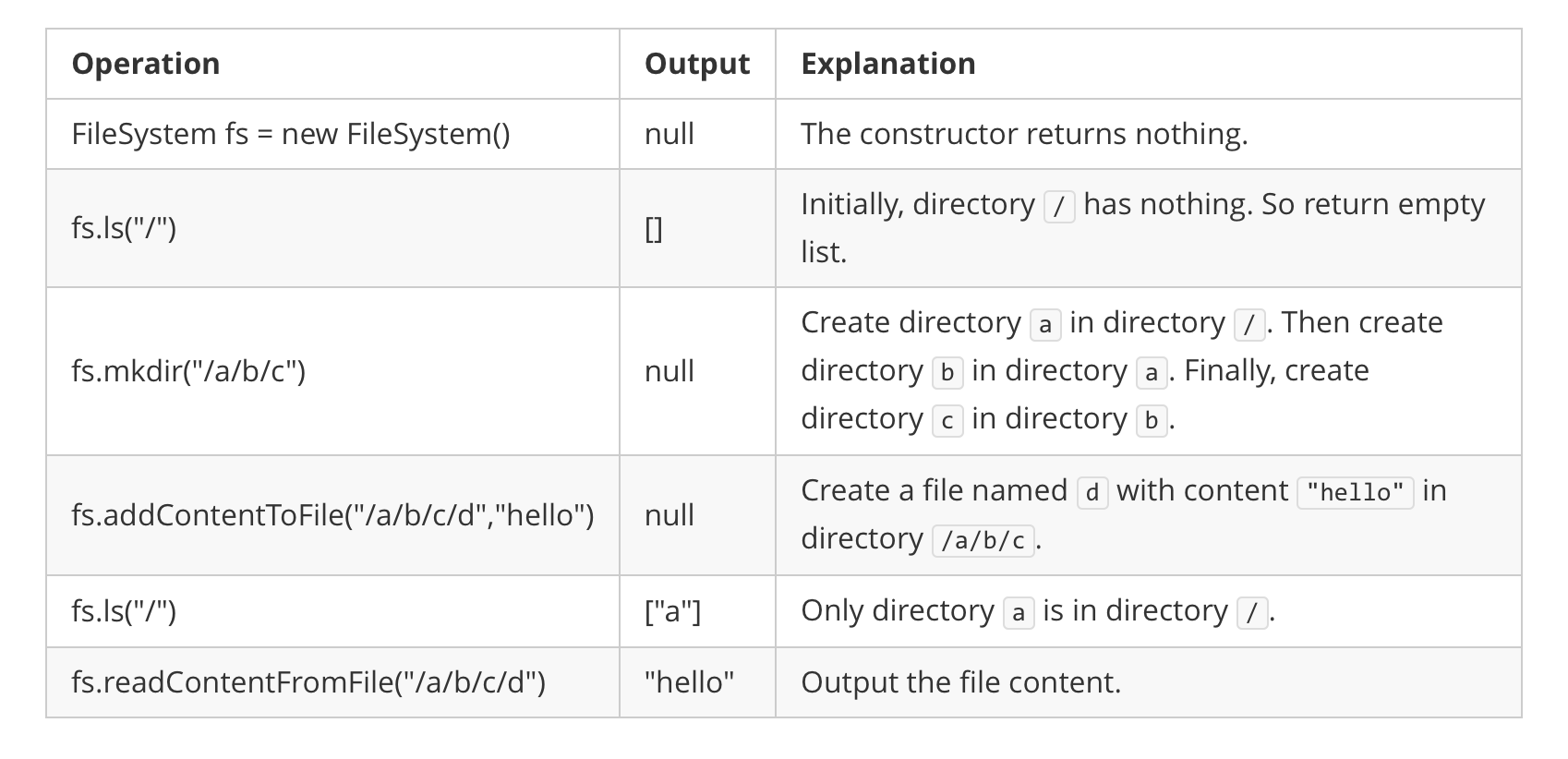
**ls**: Given a path in string format. If it is a file path, return a list that only contains this file's name. If it is a directory path, return the list of file and directory names **in this directory**. Your output (file and directory names together) should in **lexicographic order**.

**mkdir**: Given a **directory path** that does not exist, you should make a new directory according to the path. If the middle directories in the path don't exist either, you should create them as well. This function has void return type.

**addContentToFile**: Given a **file path** and **file content** in string format. If the file doesn't exist, you need to create that file containing given content. If the file already exists, you need to **append** given content to original content. This function has void return type.

**readContentFromFile**: Given a **file path**, return its **content** in string format.

**Example:**



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"

Note:

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.

1 <= content.length <= 50

At most 300 calls will be made to ls, mkdir, addContentToFile, and readContentFromFile.

**Attempt 1: 2023-08-16**

**Solution 1: Hash Table + Node structure + Design (360min)**

public class FileSystem {

    class Node {

        Map<String, Node> dirs;

        Map<String, String> files;

        public Node() {

            this.dirs = new HashMap<>();

            this.files = new HashMap<>();

        }

    }

    Node root;

    public FileSystem() {

        this.root = new Node();

    }

    /\*\*

    \* ls: To test 'ls' method it is better to finish 'mkdir' and 'addContentToFile' first

    \* @param path Given a path in string format.

    \* @return If it is a file path, return a list that only contains this file’s name.

    \* If it is a directory path, return the list of file and directory names in this directory.

    \* Your output (file and directory names together) should in lexicographic order.

    \*/

    public List<String> ls(String path) {

        Node node = root;

        List<String> result = new ArrayList<>();

        if(!path.equals("/")) {

            String[] p = path.split("/");

            // All previous sections (split path with '/') before the last section belongs

            // to 'directory', to reach the most nested 'directory' we can traverse level

            // by level, just make sure before the last section since that might be the 'file'

            // which cannot treat same as 'directory'

            // Start with i = 1 because initial root '/' will result an empty section by default

            // e.g path = "/a/b/c/d" will split into {"", "a", "b", "c", "d"}

            for(int i = 1; i < p.length - 1; i++) {

                node = node.dirs.get(p[i]);

            }

            // Now the node comes to the last section of 'directory', this node might contain

            // 'file', we can check if any 'file' present, if not then its penultimate ‘directory’,

            // we have to move one more step to reach the last 'directory'

            // e.g create file with "/a/b/c/d" + (d="hello") and "/a/b/e" + (e="TEST")

            // ls("/a/b") -> for directory "b" suppose return a contained directory as "c" and

            // a contained file "e"

            if(node.files.containsKey(p[p.length - 1])) {

                result.add(p[p.length - 1]);

                // If it is a file path, return a list that only contains this file’s name

                return result;

            } else {

                node = node.dirs.get(p[p.length - 1]);

            }

        }

        // If it is a directory path, return the list of file and directory names in this directory

        result.addAll(new ArrayList<>(node.dirs.keySet()));

        result.addAll(new ArrayList<>(node.files.k,eySet()));

        // Your output (file and directory names together) should in lexicographic order

        Collections.sort(result);

        return result;

    }

    /\*\*

    \* mkdir:

    \* @param path a directory path

    \* Given a directory path that does not exist, you should make a new directory according to the path.

    \* If the middle directories in the path don’t exist either, you should create them as well.

    \* This function has void return type.

    \*/

    public void mkdir(String path) {

        Node node = root;

        String[] p = path.split("/");

        // Start with i = 1 because initial root '/' will result an empty section by default

        // e.g path = "/a/b/c/d" will split into {"", "a", "b", "c", "d"}

        for(int i = 1; i < p.length; i++) {

            if(!node.dirs.containsKey(p[i])) {

                node.dirs.put(p[i], new Node());

            }

            node = node.dirs.get(p[i]);

        }

    }

    /\*\*

    \* addContentToFile:

    \* @param filePath Given a file path in string format.

    \* @param content Given a file content in string format.

    \* If the file doesn’t exist, you need to create that file containing given content.

    \* If the file already exists, you need to append given content to original content.

    \* This function has void return type.

    \*/

    public void addContentToFile(String filePath, String content) {

        Node node = root;

        String[] p = filePath.split("/");

        // Traverse all 'directory' sections and reach last section as 'file'

        // e.g filePath = "/a/b/c/d" -> i traverse from 1 to 3

        // -> node = "a"(i=1) => "b"(i=2) => "c"(i=3)

        for(int i = 1; i < p.length - 1; i++) {

            node = node.dirs.get(p[i]);

        }

        // Last section is a 'file' not a 'directory'

        // e.g "c"(i=3).files = {"d", content}

        node.files.put(p[p.length - 1], node.files.getOrDefault(p[p.length - 1], "") + content);

    }

    public String readContentFromFile(String filePath) {

        Node node = root;

        String[] p = filePath.split("/");

        for(int i = 1; i < p.length - 1; i++) {

            node = node.dirs.get(p[i]);

        }

        return node.files.get(p[p.length - 1]);

    }

    public static void main(String[] args) {

        FileSystem fs = new FileSystem();

        List<String> ls\_result = fs.ls("/");]

        fs.mkdir("/a/b/c");

        fs.addContentToFile("/a/b/c/d", "hello");

        fs.addContentToFile("/a/b/e", "TEST");

        List<String> ls\_result1 = so.ls("/a/b");

        String s = fs.readContentFromFile("/a/b/c/d");

        System.out.println(s);

    }

}

**Refer to**

<https://grandyang.com/leetcode/588/>

这道题让我们设计一个内存文件系统，实现显示当前文件，创建文件，添加内容到文件，读取文件内容等功能，感觉像是模拟一个terminal的一些命令。**这道题比较tricky的地方是ls这个命令，题目中的例子其实不能很好的展示出ls的要求，其对文件和文件夹的处理方式是不同的。由于这里面的文件没有后缀，所以最后一个字符串有可能是文件，也有可能是文件夹。比如a/b/c，那么最后的c有可能是文件夹，也有可能好是文件，如果c是文件夹的话，ls命令要输出文件夹c中的所有文件和文件夹，而当c是文件的话，只需要输出文件c即可。另外需要注意的是在创建文件夹的时候，路径上没有的文件夹都要创建出来，还有就是在给文件添加内容时，路径中没有的文件夹都要创建出来。**论坛上这道题的高票解法都新建了一个自定义类，但是博主一般不喜欢用自定义类来解题，而且感觉那些使用了自定义类的解法并没有更简洁易懂，**所以这里博主就不创建自定义类了，而是使用两个哈希表来做，其中dirs建立了路径和其对应的包含所有文件和文件夹的集合之间的映射，files建立了文件的路径跟其内容之间的映射**。

最开始时将根目录”/“放入dirs中，**然后看ls的实现方法，如果该路径存在于files中，说明最后一个字符串是文件，那么我们将文件名取出来返回即可，如果不存在，说明最后一个字符串是文件夹，那么我们到dirs中取出该文件夹内所有的东西返回即可**。再来看mkdir函数，我们的处理方法就是根据”/“来分隔分隔字符串，如果是Java，那么直接用String自带的split函数就好了，但是C++没有Java那么多自带函数，所以只能借助字符串流类来处理，处理方法就是将每一层的路径分离出来，然后将该层的文件或者文件夹加入对应的集合中，注意的地方就是处理根目录时，要先加上”/“，其他情况都是后加。下面来看addContentToFile函数，首先分离出路径和文件名，如果路径为空，说明是根目录，需要加上”/“，然后看这个路径是否已经在dirs中存在，如果不存在，调用mkdir来创建该路径，然后把文件加入该路径对应的集合中，再把内容加入该文件路径的映射中。最后的读取文件内容就相当简单了，直接在files中返回即可，参见代码如下：

class FileSystem {

    public:

    FileSystem() {

        dirs["/"];

    }

    vector<string> ls(string path) {

        if (files.count(path)) {

            int idx = path.find\_last\_of('/');

            return {path.substr(idx + 1)};

        }

        auto t = dirs[path];

        return vector<string>(t.begin(), t.end());

    }

    void mkdir(string path) {

        istringstream is(path);

        string t = "", dir = "";

        while (getline(is, t, '/')) {

            if (t.empty()) continue;

            if (dir.empty()) dir += "/";

            dirs[dir].insert(t);

            if (dir.size() > 1) dir += "/";

            dir += t;

        }

    }

    void addContentToFile(string filePath, string content) {

        int idx = filePath.find\_last\_of('/');

        string dir = filePath.substr(0, idx);

        string file = filePath.substr(idx + 1);

        if (dir.empty()) dir = "/";

        if (!dirs.count(dir)) mkdir(dir);

        dirs[dir].insert(file);

        files[filePath].append(content);

    }

    string readContentFromFile(string filePath) {

        return files[filePath];

    }

    private:

    unordered\_map<string, set<string>> dirs;

    unordered\_map<string, string> files;

};

**Refer to**

<https://wentao-shao.gitbook.io/leetcode/data-structure/588.design-in-memory-file-system>

设计内存文件系统模拟实现如下功能：

ls：给定路径字符串，若对应一个目录，则输出其中包含的目录和文件（字典序）；若对应一个文件，则只输出该文件名

mkdir：创建目录，若目录不存在，则递归创建缺少的目录

addContentToFile：在文件中追加内容，若文件不存在，则新建

readContentFromFile：从文件中读取内容并返回

**注意：**

你可以假设文件和目录的路径均为绝对路径，以/开始，并且结尾不包含/，除非路径就是"/"本身

你可以假设所有操作均合法，用户不会常识读取一个不存在的文件，或者ls一个不存在的目录

你可以假设所有目录名称和文件名称都只包含小写字母，并且在同一目录下不会存在同名的目录或者文件

树形结构（Tree）

目录节点Node包含两个字段dirs和files，分别存储其中包含的子目录节点和文件内容

**Approach #1 Using separate Directory and File List**

Time: O(m+n) && Space: O(m+n)

class FileSystem {

    class Dir {

        HashMap<String, Dir> dirs = new HashMap();

        HashMap<String, String> files = new HashMap();

    }

    Dir root;

    public FileSystem() {

        root = new Dir();

    }

    public List<String> ls(String path) {

        Dir t = root;

        List<String> files = new ArrayList();

        if (!path.equals("/")) {

            String[] d = path.split("/");

            for (int i = 1; i < d.length - 1; i++) {

                t = t.dirs.get(d[i]);

            }

            if (t.files.containsKey(d[d.length-1])) {

                files.add(d[d.length-1]);

                return files;

            } else {

                t = t.dirs.get(d[d.length - 1]);

            }

        }

        files.addAll(new ArrayList<>(t.dirs.keySet()));

        files.addAll(new ArrayList<>(t.files.keySet()));

        Collections.sort(files);

        return files;

    }

    public void mkdir(String path) {

        Dir t = root;

        String[] d = path.split("/");

        for (int i = 1; i < d.length; i++) {

            if (!t.dirs.containsKey(d[i])) {

                t.dirs.put(d[i], new Dir());

            }

            t = t.dirs.get(d[i]);

        }

    }

    public void addContentToFile(String filePath, String content) {

        Dir t = root;

        String[] d = filePath.split("/");

        for (int i = 1; i < d.length - 1; i++) {

            t = t.dirs.get(d[i]);

        }

      t.files.put(d[d.length - 1], t.files.getOrDefault(d[d.length-1], "") + content);

    }

    public String readContentFromFile(String filePath) {

        Dir t = root;

        String[] d = filePath.split("/");

        for (int i = 1; i < d.length - 1; i++) {

            t = t.dirs.get(d[i]);

        }

        return t.files.get(d[d.length -1]);

    }

}

/\*\*

\* 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);

\*/

**Approach #2 Using unified Directory and File List**

public class FileSystem {

    class File {

        boolean isfile = false;

        HashMap < String, File > files = new HashMap < > ();

        String content = "";

    }

    File root;

    public FileSystem() {

        root = new File();

    }

    public List < String > ls(String path) {

        File t = root;

        List < String > files = new ArrayList < > ();

        if (!path.equals("/")) {

            String[] d = path.split("/");

            for (int i = 1; i < d.length; i++) {

                t = t.files.get(d[i]);

            }

            if (t.isfile) {

                files.add(d[d.length - 1]);

                return files;

            }

        }

        List < String > res\_files = new ArrayList < > (t.files.keySet());

        Collections.sort(res\_files);

        return res\_files;

    }

    public void mkdir(String path) {

        File t = root;

        String[] d = path.split("/");

        for (int i = 1; i < d.length; i++) {

            if (!t.files.containsKey(d[i]))

                t.files.put(d[i], new File());

            t = t.files.get(d[i]);

        }

    }

    public void addContentToFile(String filePath, String content) {

        File t = root;

        String[] d = filePath.split("/");

        for (int i = 1; i < d.length - 1; i++) {

            t = t.files.get(d[i]);

        }

        if (!t.files.containsKey(d[d.length - 1]))

            t.files.put(d[d.length - 1], new File());

        t = t.files.get(d[d.length - 1]);

        t.isfile = true;

        t.content = t.content + content;

    }

    public String readContentFromFile(String filePath) {

        File t = root;

        String[] d = filePath.split("/");

        for (int i = 1; i < d.length - 1; i++) {

            t = t.files.get(d[i]);

        }

        return t.files.get(d[d.length - 1]).content;

    }

}

/\*\*

\* 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);

\*/

**Refer to**

<https://www.cnblogs.com/Dylan-Java-NYC/p/16514312.html>

It is like Tire.

For each DirNode, it contains a map to children directories DirNode, and a map to children files StringBuilder.

ls has one corner case that when path is "/".  since "/".split("/") is empty array. arr[arr.length - 1] will have index out of bound exception.

Time Complexity: ls, O(m + klogk). m = path.length(). k is number of items in the res.

mkdir, O(m).

add, O(m).

readContentFromFile, O(m).

Space: O(n). size of DirNode tree.

AC Java:

class FileSystem {

    DirNode root;

    public FileSystem() {

        root = new DirNode();

    }

    public List<String> ls(String path) {

        DirNode cur = root;

        List<String> res = new ArrayList<>();

        if(!path.equals("/")){

            String [] arr = path.split("/");

            for(int i = 1; i < arr.length - 1; i++){

                cur.dirs.putIfAbsent(arr[i], new DirNode());

                cur = cur.dirs.get(arr[i]);

            }

            if(cur.files.containsKey(arr[arr.length - 1])){

                res.add(arr[arr.length - 1]);

                return res;

            }

            cur.dirs.putIfAbsent(arr[arr.length - 1], new DirNode());

            cur = cur.dirs.get(arr[arr.length - 1]);

        }

        res.addAll(cur.dirs.keySet());

        res.addAll(cur.files.keySet());

        Collections.sort(res);

        return res;

    }

    public void mkdir(String path) {

        DirNode cur = root;

        String [] arr = path.split("/");

        for(int i = 1; i < arr.length; i++){

            cur.dirs.putIfAbsent(arr[i], new DirNode());

            cur = cur.dirs.get(arr[i]);

        }

    }

    public void addContentToFile(String filePath, String content) {

        DirNode cur = root;

        String [] arr = filePath.split("/");

        for(int i = 1; i < arr.length - 1; i++){

            cur.dirs.putIfAbsent(arr[i], new DirNode());

            cur = cur.dirs.get(arr[i]);

        }

        cur.files.putIfAbsent(arr[arr.length - 1], new StringBuilder());

        cur.files.get(arr[arr.length - 1]).append(content);

    }

    public String readContentFromFile(String filePath) {

        DirNode cur = root;

        String [] arr = filePath.split("/");

        for(int i = 1; i < arr.length - 1; i++){

            cur.dirs.putIfAbsent(arr[i], new DirNode());

            cur = cur.dirs.get(arr[i]);

        }

        return cur.files.get(arr[arr.length - 1]).toString();

    }

}

class DirNode{

    Map<String, DirNode> dirs = new HashMap<>();

    Map<String, StringBuilder> files = new HashMap<>();

}

/\*\*

\* 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);

\*/

**Refer to**

[L1166.Design File System (Ref.L588)](note://WEBfd789c22d92a583cd8038203ad2d1389)