【系统设计】-模拟目录管理

题目描述与示例

题目描述

实现一个模拟目录管理功能的软件，输入一个命令序列，输出最后一条命令运行结果。

支持命令：

1) 创建目录命令： `mkdir 目录名称` ，如 `mkdir abc` 为在当前目录创建 `abc` 目录，如果已存在同名目录则不执行任何操作。此命令无输出。

2) 进入目录命令： `cd 目录名称` ，如 `cd abc` 为进入 `abc` 目录，特别地， `cd ..` 为返回上级目录，如果目录不存在则不执行任何操作。此命令无输出。

3) 查看当前所在路径命令： `pwd` ，输出当前路径字符串。

约束:

1) 目录名称仅支持小写字母； `mkdir` 和 `cd` 命令的参数仅支持单个目录，如 `mkdir abc` 和 `cd abc` ;不支持嵌套路径和绝对路径，如 `mkdir abc/efg` ， `cd abc/efg` 是不支持的。

2) 目录符号为 `/` ，根目录 `/` 作为初始目录。

3) 任何不符合上述定义的无效命令不做任何处理并且无输出。

输入描述

输入 `N` 行字符串，每一行字符串是一条命令

输出描述

输出最后一条命令运行结果字符串

补充说明

命令行数限制 `100` 行以内，目录名称限制 `10` 个字符以内

示例

输入

```
mkdir abc
cd abc
pwd
```

输出

```
/abc/
```

解题思路

系统设计的大模拟题，关键还是在于读懂题意。

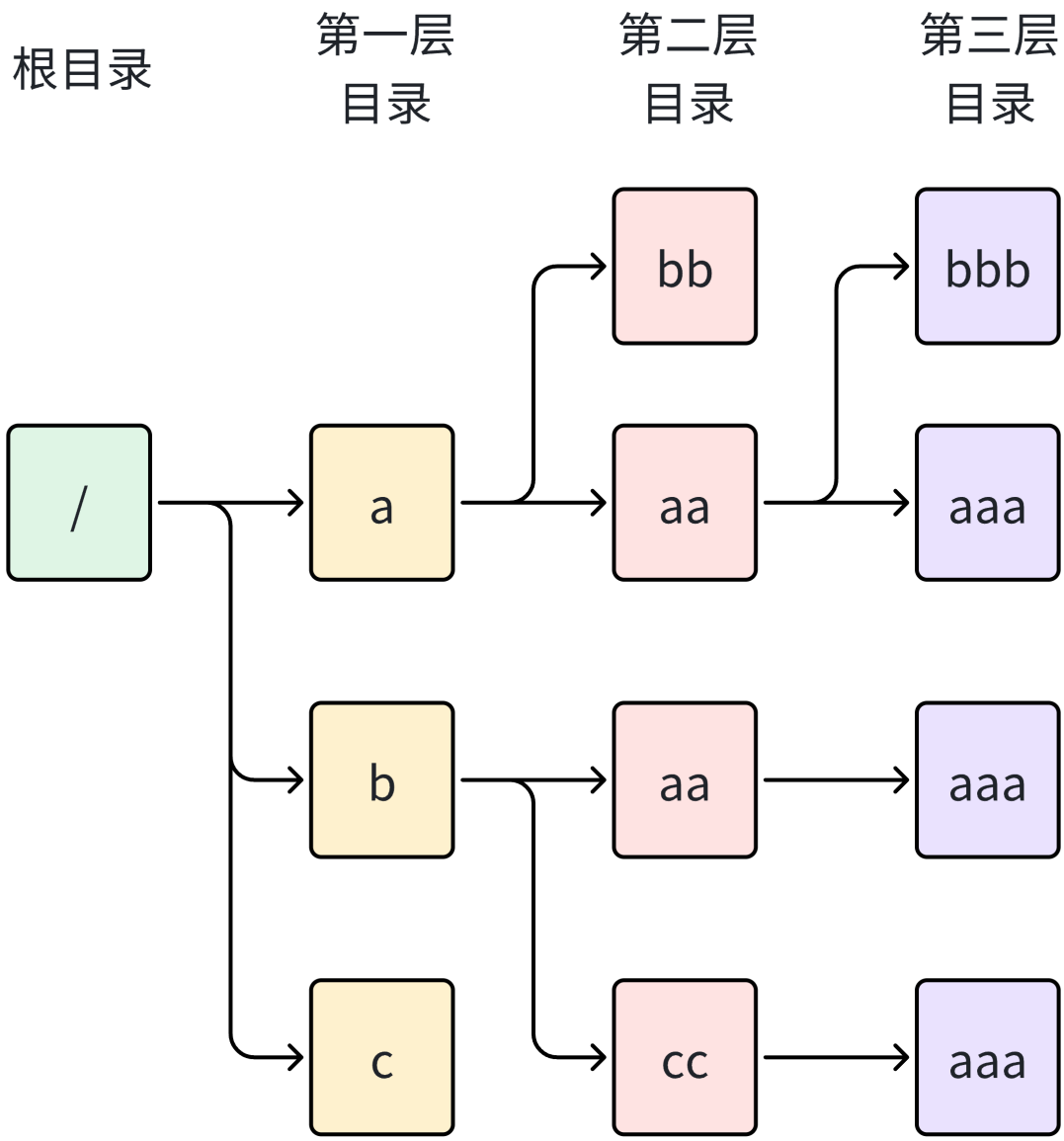
对于目录的操作命令，一共有四种：**创建新目录、进入已存在目录、返回上一级目录、打印当前路径**。

如果你使用过Linux系统，那么上述这些概念应当是相当熟悉的。

另外，对于文件路径类的问题，类似于 LC71. 简化路径 ，我们可以用一个栈 `path` 来储存路径，一旦发生进入目录或者返回目录，则对应入栈和出栈操作。可以将 `path` 初始化为空。

目录树形结构

创建出来的目录系统呈现一个树形结构。譬如



那么我们只需要构建这样一个树形结构来对应整个目录系统，并根据操作命令进行相应操作即可。

节点类的设计

对于每一个文件夹所对应的节点，我们需要知道以下信息：

1. 这个文件夹的名字（对应打印操作）
2. 这个文件夹的父节点，即它的上一级文件夹对应的节点（对应返回操作）
3. 这个文件夹的子节点构成的集合，即它所有的下一级文件夹对应的节点构成的集合（对应创建、进入操作）

由于信息较多，我们可以构建出如下的节点类

```
1 class Node():
```

```
    def __init__(self, name, father = None):
        self.name = name
        self.father = father
        self.children = defaultdict(Node)
```

要特别注意子节点构成的集合，需要构建成哈希表的形式，其中 `key` 为子文件名， `value` 为子文件对应的节点，也是一个 `Node` 。这样做是为了方便后续的查找操作。

在整个动态模拟的过程中，我们需要知道根节点 `root` ，还需要知道当前进入到了哪一个节点中，故需要一个当前节点 `curNode` ，将其初始化为根节点 `root` 。

```
root = Node("")
curNode = root
```

代码框架

对于每一个操作 `operations[i]` ，如果

- 其为打印操作，那么存在 `operations[i] == "pwd"` 成立
- 其不为打印操作，那么其必然可以用空格给分割成两部分，即

```
op, fileName = operations[i].split()
```

在进一步判断操作之前，可以判断文件名 `fileName` 本身是否存在嵌套。若其以 `"/"` 为分割符进行切割后，长度大于 `1` ，说明存在文件嵌套的情况，这是一个无效命令，可以直接跳过。即

```
if len(fileName.split("/")) > 1:
    continue
```

若 `fileName` 是一个有效文件名，则可以进一步 `op` 的判断操作类型。若该操作为

- 创建操作，那么存在 `op == "mkdir"` 成立

- 进入操作，那么存在 `op == "cd"` 且 `fileName != ".."` 成立
- 退出操作，那么存在 `op == "cd"` 且 `fileName == ".."` 成立

故可以将整个模拟过程的代码框架封装在 `solve()` 函数中，具体如下

```python
def solve(operations):
    n = len(operations)
    if n == 0 or operations[-1] != "pwd":
        return "/"

    root = Node("")
    curNode = root
    path = []

    for i in range(n-1):
        # 打印操作
        if operations[i] == "pwd":
            pass

        # 不是打印操作，则根据空格进行切割
        op, fileName = operations[i].split()

        # 文件名是否存在嵌套的判断
        if len(fileName.split("/")) > 1:
            continue

        # 创建操作
        if op == "mkdir":
            pass

        # 进入操作
        if op == "cd" and fileName != "..":
            pass

        # 返回操作
        if op == "cd" and fileName == "..":
            pass

    return "/" if len(path) == 0 else "/" + "/".join(path) + "/"
```

打印操作

打印操作是最简单的。由于题目要求输出最后一条命令运行结果字符串，因此如果 `operations` 数组的最后一个元素不是打印操作，即 `operations[-1] != "pwd"` ，则直接返回打印出 `"/"` 即可。

```
n = len(operations)
if n == 0 or operations[-1] != "pwd":
    return "/"
```

（照理来说应该返回空字符串 `""` ，但根据考试反馈，这种情况下必须返回 `"/"` 才是正确结果）

另外，如果在遍历过程中遇到打印操作，则直接跳过即可。

```
if operations[i] == "pwd":
    continue
```

创建操作

创建操作要求在当前节点 `curNode` 下创建一个新文件夹，我们需要判断这个子文件名 `fileName` 是否已经存在于 `curNode` 的子节点中。若

- 已存在，则无需重复创建，直接跳过
- 不存在，则创建新子文件对应的新节点，并且将其加入 `curNode` 的子节点集合中。

```
if op == "mkdir":
    if fileName in curNode.children:
        continue
    newNode = Node(fileName, curNode)
    curNode.children[fileName] = newNode
```

进入操作

进入操作要求进入当前节点 `curNode` 下一个名为 `fileName` 的文件夹，我们需要判断这个子文件名 `fileName` 是否已经存在于 `curNode` 的子节点中。若

- 不存在，则无法进入一个不存在的文件，直接跳过

- 已存在，则需要更新 `curNode` 为 `fileName` 对应的节点，同时也需要将 `fileName` 更新加入 `path` 中

```python
if op == "cd" and fileName != "..":
    if fileName not in curNode.children:
        continue
    curNode = curNode.children[fileName]
    path.append(fileName)
```

返回操作

返回操作要求返回 `curNode` 节点的上一层节点，我们需要判断当前节点 `curNode` 是否为根节点 `root`。若

- 是根节点，则无法再返回上一级，直接跳过
- 不是根节点，则我们可以通过 `curNode` 中的成员变量 `curNode.father` 来得到 `curNode` 的父节点，将 `curNode` 进行修改，同时也需要将 `path` 的最后一个元素（即 `curNode` 的文件名）弹出

```python
if op == "cd" and fileName == "..":
    if curNode == root:
        continue
    curNode = curNode.father
    path.pop()
```

最终结果

在做完 `n-1` 次遍历之后，最后一次操作是打印操作，此时我们需要将 `path` 中的结果进行合并，并且作为 `solve()` 函数的返回值。

我们需要判断 `path` 路径栈的长度，若

- 长度为 `0`，则说明最终位于根节点，需要返回 `"/"`
- 长度不为 `0`，则需要将 `path` 中的所有字符串用 `"/"` 连接，并且前后还需要再加上两个 `"/"`，作为最终路径结果

```python
return "/" if len(path) == 0 else "/" + "/".join(path) + "/"
```

代码

Python

```python
# 题目：【系统设计】2024E-模拟目录管理
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟/树/栈
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict


# 构建一个节点类
# 每一个节点（对应一个文件夹）都包含三个属性
# 1. 这个文件夹自己的文件名
# 2. 这个文件夹的父节点，即它的上一级文件夹对应的节点
# 3. 这个文件夹的子节点构成的集合，
# 即它所有的下一级文件夹对应的节点构成的集合
class Node():
    def __init__(self, name, father = None):
        self.name = name
        self.father = father
        self.children = defaultdict(Node)


# 解决问题的函数
def solve(operations):
    # 获得操作的个数n
    n = len(operations)

    # 特殊情况判断，如果最后一条操作字符串不是"pwd"
    # 那么最后一条命令不会产生任何输出结果
    # 直接返回空字符串""
    # 【但是根据同学们的反馈，这种情况下需要输出"/"才能100%完全通过】
    if n == 0 or operations[-1] != "pwd":
        return "/"

    # 创建根节点，为空即可
    root = Node("")
```

```
    # 创建当前节点，用于表示
    # 初始化为根节点
    curNode = root
    # 创建一个路径列表
    path = []

    # 最后一个操作必然是【打印】操作
    # 遍历除了最后一个操作之外的所有其他操作
    for i in range(n-1):
        # 如果在中间出现【打印】操作，不影响最终结果，直接跳过
        if operations[i] == "pwd":
            continue
        # 对当前操作，根据空格进行切割
        # 分别得到具体的操作op，文件名fileName
        op, fileName = operations[i].split()
        # 如果文件名fileName本身存在嵌套，即以"/"为分割符进行切割后，长度大于1
        # 说明这是一个无效命令，直接跳过
        if len(fileName.split("/")) > 1:
            continue
        # 如果是一个【创建】操作
        if op == "mkdir":
            # 如果这个文件名已经存在于curNode的子节点哈希表中，
            # 则无需重复创建，直接跳过
            if fileName in curNode.children:
                continue
            # 创建新节点newNode，文件名为fileName，父节点为curNode
            newNode = Node(fileName, curNode)
            # 往当前节点curNode的子节点哈希表中，添加newNode
            # key为新节点的文件名fileName，value为新节点本身newNode
            curNode.children[fileName] = newNode

        # 如果是一个【进入】操作
        if op == "cd" and fileName != "..":
            # 如果这个文件名已经尚未存在于curNode的子节点哈希表中，
            # 则无法进入这个不存在的子文件夹，直接跳过
            if fileName not in curNode.children:
                continue
            # 如果这个文件名已经在于curNode的子节点哈希表中，则可以进入
            # 更新当前节点，为这个文件名为fileName的子节点
            curNode = curNode.children[fileName]
            # 路径列表需要加入新的文件名fileName
            path.append(fileName)

        # 如果是一个【返回】操作
        if op == "cd" and fileName == "..":
            # 如果当前节点已经是根节点，则无法再返回上一级，直接跳过
            if curNode == root:
```

```
                continue
            # 更新当前节点，为其父节点
            curNode = curNode.father
            # 路径列表需要弹出最后更新的文件名
            path.pop()

    # 最后判断path的长度，
    # 若path长度为0，说明最终返回到了根目录，应该直接输出"/"
    # 否则以为连接符"/"将path列表进行拼接，前后再补上两个"/"为最终路径
    return "/" if len(path) == 0 else "/" + "/".join(path) + "/"


operations = list()
# 输入未知行数，使用死循环结合try-except语句进行输入
while 1:
    try:
        op = input()
        if op == "":
            break
        operations.append(op)
    except:
        break


# 调用solve函数并输出
print(solve(operations))
```

Java

```
import java.util.*;

class Node {
    String name;
    Node father;
    Map<String, Node> children;

    Node(String name, Node father) {
        this.name = name;
        this.father = father;
        this.children = new HashMap<>();
    }
}

class Solution {
    String solve(List<String> operations) {
```

```
        int n = operations.size();

        if (n == 0 || !operations.get(n - 1).equals("pwd")) {
            return "/";
        }

        Node root = new Node("", null);
        Node curNode = root;
        List<String> path = new ArrayList<>();

        for (int i = 0; i < n - 1; i++) {
            if (operations.get(i).equals("pwd")) {
                continue;
            }
            String[] parts = operations.get(i).split(" ");
            String op = parts[0];
            String fileName = parts[1];

            if (fileName.contains("/")) {
                continue;
            }

            if (op.equals("mkdir")) {
                if (curNode.children.containsKey(fileName)) {
                    continue;
                }
                Node newNode = new Node(fileName, curNode);
                curNode.children.put(fileName, newNode);
            }

            if (op.equals("cd") && !fileName.equals("..")) {
                if (!curNode.children.containsKey(fileName)) {
                    continue;
                }
                curNode = curNode.children.get(fileName);
                path.add(fileName);
            }

            if (op.equals("cd") && fileName.equals("..")) {
                if (curNode == root) {
                    continue;
                }
                curNode = curNode.father;
                path.remove(path.size() - 1);
            }
        }

```

```
        return (path.isEmpty()) ? "/" : "/" + String.join("/", path) + "/";
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<String> operations = new ArrayList<>();
        String op;
        while (scanner.hasNextLine() && !(op = scanner.nextLine()).isEmpty()) {
            operations.add(op);
        }
        Solution solution = new Solution();
        System.out.println(solution.solve(operations));
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <sstream>
using namespace std;

class Node {
public:
    string name;
    Node* father;
    unordered_map<string, Node*> children;

    Node(string name, Node* father) : name(name), father(father) {}
};

class Solution {
public:
    string solve(vector<string>& operations) {
        int n = operations.size();

        if (n == 0 || operations[n - 1] != "pwd") {
            return "/";
        }

        Node* root = new Node("", nullptr);
```

```
26          Node* curNode = root;
27          vector<string> path;
28
29          for (int i = 0; i < n - 1; i++) {
30              if (operations[i] == "pwd") {
31                  continue;
32              }
33              string op, fileName;
34              istringstream iss(operations[i]);
35              iss >> op >> fileName;
36
37              if (fileName.find("/") != string::npos) {
38                  continue;
39              }
40
41              if (op == "mkdir") {
42                  if (curNode->children.find(fileName) != curNode-
    >children.end()) {
43                      continue;
44                  }
45                  Node* newNode = new Node(fileName, curNode);
46                  curNode->children[fileName] = newNode;
47              }
48
49              if (op == "cd" && fileName != "..") {
50                  if (curNode->children.find(fileName) == curNode-
    >children.end()) {
51                      continue;
52                  }
53                  curNode = curNode->children[fileName];
54                  path.push_back(fileName);
55              }
56
57              if (op == "cd" && fileName == "..") {
58                  if (curNode == root) {
59                      continue;
60                  }
61                  curNode = curNode->father;
62                  path.pop_back();
63              }
64          }
65
66          return (path.empty()) ? "/" : "/" + join(path, "/") + "/";
67      }
68
69      string join(vector<string>& path, string delimiter) {
70          string result;
```

```
71          for (int i = 0; i < path.size(); i++) {
72              result += path[i];
73              if (i != path.size() - 1) {
74                  result += delimiter;
75              }
76          }
77          return result;
78      }
79  };
80
81  int main() {
82      Solution sol;
83      vector<string> operations;
84      string op;
85      while (getline(cin, op)) {
86          if (op.empty()) {
87              break;
88          }
89          operations.push_back(op);
90      }
91      cout << sol.solve(operations) << endl;
92      return 0;
93  }
94
```

时空复杂度

时间复杂度：`O(N)`。每种操作的时间复杂度均为`O(1)`，一共有`N`条命令。

空间复杂度：`O(N)`。节点所占空间。