Part2: File Process

Total Points: 45' Check Deadline: 5/26 10:00AM

Background

In this project, you will work with different tasks related to processing files. The difficulty of each task progresses incrementally, so you need to complete them step by step and need to leverage the collaborative abilities of your team.

The basic skills you need to master are:

- Class
- Recursion Idea
- Os, Shutil, Zipfile... Library
- Ability To Research And Understand Coding Resources

You will be given a code frame in this section, so all you have to do is to fill in the code blank. However, this is not very simple to implement, because the expected amount of code is about 200-300 lines. This project can help you easily complete any file processing tasks for the rest of your education or career, so take it seriously, and you'll feel a sense of accomplishment when you succeed!

Targets

Due to the complexity of the overall task, we split it into several relatively simple and well understood small goals to complete separately. Whether you are a beginner or a master programmer, I hope you can write code in the following order, which will save you a lot of time to some extent.

- `File.py`

  ```
  class File:
      def __init__(self, source_path=""):
          ...

      def transfer(self, target_path=""):
          ...

      def __str__(self):
          ...
  ```

  First, you need to implement the `File` class, enabling you to move a file from `source_path` to `target_path`.

- `Folder.py`

```
class Folder:
    def __init__(self, source_path=""):
        ...

    def unpack(self, target_path=""):
        ...

    def __str__(self):
        ...
```

Second, you need to implement the `Folder` class, enabling you to move all the contents within a folder from `source_path` to the `target_path`.

- `CompressedFile.py`

```
class CompressedFile:
    def __init__(self, source_path=""):
        ...

    def decompress(self, target_path=""):
        ...

    def __str__(self):
        ...
```

Third, you need to implement the `CompressedFile` class, enabling you to decompress all contents from a compressed file at the `source_path` to the `target_path`.

When you first get the code, you don't need to worry about the scoring criteria. Just carefully consider how to implement the above three targets. Once you feel that you've made significant progress, then you can start thinking about the tasks. You'll find this approach to be more helpful.

Tasks

The whole context of the task is that you have received some stacked files, and you need to sort and organize them.

Suppose the stack file structure in `C:\` looks like this:

```
* C:\cats\black_cats\black_cat_1.png
* C:\cats\black_cats\black_cat_2.png
* C:\cats\black_cats\black_cat_3.png

* C:\cats\white_cats\white_cat_1.png
* C:\cats\white_cats\white_cat_2.png
* C:\cats\white_cats\white_cat_3.png

* C:\cats\cat_1.png
* C:\cats\cat_2.png
* C:\cats\cat_3.png

* C:\dogs.zip
```

And in `dogs.zip`, there are also some files:

```
* black_dogs\black_dog_1.png
* black_dogs\black_dog_2.png
* black_dogs\black_dog_3.png

* white_dogs\white_dog_1.png
* white_dogs\white_dog_2.png
* white_dogs\white_dog_3.png

* dog_1.png
* dog_2.png
* dog_3.png
```

What we want to do is to **distinguish these files by the smallest category**. If we follow the example above, the smallest category is `black_cats`, `white_cats`, `cats`, `black_dogs`, `white_dogs`, `dogs`. To sum up in one rule, it is to **find the deepest layer that contains files, whether it's a folder or a compressed file**.

Every time you find a smallest category, you need to copy it to `D:\` , as well as all the files in it.

```
* D:\black_cats\black_cat_1.png
* D:\black_cats\black_cat_2.png
* D:\black_cats\black_cat_3.png

* D:\white_cats\white_cat_1.png
* D:\white_cats\white_cat_2.png
* D:\white_cats\white_cat_3.png

* D:\cats\cat_1.png
* D:\cats\cat_2.png
* D:\cats\cat_3.png

* D:\black_dogs\black_dog_1.png
* D:\black_dogs\black_dog_2.png
* D:\black_dogs\black_dog_3.png
```

```
* D:\white_dogs\white_dog_1.png
* D:\white_dogs\white_dog_2.png
* D:\white_dogs\white_dog_3.png

* D:\dogs\dog_1.png
* D:\dogs\dog_2.png
* D:\dogs\dog_3.png
```

As you can see, after processing, any file in `D:\` will only keep two levels at most. Of course, you also need to store some information as `D:\data.json` to keep track of your actions:

```
{
    "D:\black_cats": {
        "source": “C:\cats\black_cats",
        “count”: 3
    },

    "D:\white_cats": {
        "source": “C:\cats\white_cats",
        “count”: 3
    },

    "D:\cats": {
        "source": “C:\cats",
        “count”: 3
    },

    "D:\black_dogs": {
        "source": “C:\dogs.zip\black_dogs",
        “count”: 3
    },

    "D:\white_dogs": {
        "source": “C:\dogs.zip\white_dogs",
        “count”: 3
    },

    "D:\dogs": {
        "source": “C:\dogs.zip",
        “count”: 3
    }
}
```

That's all the rules in part2. Comparing to Part1, it's really complicated, so if you have any questions on the questions, be sure to ask TA immediately. The rest is our specific tasks:

- Task1: Files(1'+1')
- Task2: Folders With Files(2'+1')
- Task3: Folders With Folders(3'+1')

- Task4: Compressed Files With Files(4'+1')
- Task5: Compressed Files With Folders(5'+1')
- Task6: Compressed Files With Compressed Files(6'+1')
- Task7: Folders With Compressed Files(7'+1')

(a' + 1') means that a' is used to test whether all files are successfully copied, and 1' indicates whether the `json` is correct.

All the local test data has been given to you. As long as you complete the corresponding requirements, you can get 35 points. For the remaining 10 points, we will test by TA after you submit the local code, 1 point for each.

Check Rule

When your group thinks you have completed all tasks, you can make an appointment with the TA to check in offline or go to the TA's office hour to check or just during the break time in the courseware. The TA will ask you to run the crawler in the field. Before the deadline, each group will only have 1 chance to check because all the local test data is given. Once the local score is confirmed by TA, you need to hand in the code immediately and record the division of labor of team members. The score of the follow-up tests will be released to you after TA run the code. Again, pay attention to academic integrity. Any plagiarism will result in a score of 0 in Part2!