GENERAL

cracking the coding interview filetyp

Published: September 3, 2026 | Index ID: #-

Cracking the Coding Interview: Mastering the Filetyp Challenge

In today's competitive tech landscape, landing a role at a top-tier company often hinges on how well you can solve a single, deceptively simple interview question. One of the most frequently cited problems that tests a candidate's grasp of data structures, algorithmic thinking, and code clarity is the **Filetyp**problem. Whether you're a seasoned software engineer or a fresh graduate, understanding how to approach this puzzle can set you apart from the crowd.

In this guide, we'll walk you through every nuance of the Filetyp challenge. From the fundamentals of the problem statement to advanced optimization strategies, you'll gain a comprehensive toolkit that will help you **crack the coding interview filetyp**with confidence. We'll also share interview specific tips, sample solutions in multiple languages, and a curated list of practice problems to reinforce your learning.

What Is the Filetyp Problem?

The Filetyp question typically presents a scenario where you receive a list of file names and you must determine how many files share the same extension. The problem tests:

- String manipulation skills
- Use of hash maps or dictionaries for frequency counting
- Attention to edge cases (files without extensions, hidden files, uppercase/lowercase handling)

Below is a common wording of the problem:

Given an array of file names, write a function that returns the count of files that share the same extension. For example, for the input ["report.pdf", "image.png", "data.csv", "summary.pdf"], the function should return 2 because there are two files with the .pdf extension.

While this version appears simple, interviewers often probe deeper by asking you to:

1. Return a mapping of extensions to their counts.
2. Handle files without extensions.
3. Consider case sensitivity.
4. Optimize for large inputs.

Mastering these variations will demonstrate your thoroughness and adaptability—qualities highly prized by hiring managers.

Key Concepts to Master

1. String Splitting & Parsing

At the heart of the Filetyp problem is the need to isolate the file extension. The most common approach is to split the file name on the period character (.) and take the last segment. However, you must also handle files that contain multiple periods (e.g., archive.tar.gz) or no period at all.

2. Hash Maps for Frequency Counting

Hash maps (or dictionaries) provide O(1) average time lookups and updates. By iterating over the file list and incrementing the count for each extracted extension, you can achieve an overall linear time complexity.

3. Edge Cases

- Files with no extension (e.g., README).
- Hidden files that start with a dot (e.g., .gitignore).
- Uppercase vs. lowercase extensions (e.g., IMAGE.PNG vs. image.png).
- Files ending with a dot (e.g., file.).

Addressing these cases not only prevents bugs but also showcases your attention to detail.

4. Time & Space Complexity

For a list of n files, a single pass through the array yields an O(n) time complexity. The space complexity depends on the number of unique extensions, typically O(k), where k ≤ n.

Step by Step Solution

Approach 1: Using a Simple Hash Map

Below is a straightforward implementation in Python. The same logic can be translated into JavaScript, Java, or C++ with minor syntax changes.

```
def filetyp_counts(file_names):
    counts = {}
    for name in file_names:
        # Skip hidden files that start with a dot but have no extension
        if name.startswith('.') and name.count('.') == 1:
            continue
        parts = name.rsplit('.', 1)
        if len(parts) == 2:
            ext = parts[1].lower()  # Normalize case
            counts[ext] = counts.get(ext, 0) + 1
    return counts
```

Explanation:

- We use rsplit('.', 1) to split from the rightmost dot, ensuring that filenames like archive.tar.gz correctly yield gz as the extension.
- We normalize extensions to lowercase to treat .PNG and .png as the same.
- Files with no extension are ignored.

Approach 2: Counting Only the Most Frequent Extension

If the interview question asks for the count of files that share the most common extension, you can extend the previous solution by tracking the maximum value.

```
def most_common_extension_count(file_names):
```

```
    counts = {}
    max_count = 0
    for name in file_names:
        parts = name.rsplit('.', 1)
        if len(parts) == 2:
            ext = parts[1].lower()
            counts[ext] = counts.get(ext, 0) + 1
            if counts[ext] > max_count:
                max_count = counts[ext]
    return max_count
```

Here, max_count holds the highest frequency encountered during iteration.

Optimizing for Large Inputs

When dealing with millions of files, the core algorithm remains efficient, but you must consider memory consumption. One optimization is to stream the file list rather than loading it entirely into memory. In languages like Java or C++, you can read file names line by line from a file or database cursor.

Tip: If the interview environment supports it, demonstrate how you would handle a streaming input by using generators (Python) or iterators (Java).

Testing & Edge Case Scenarios

Robust code is built on comprehensive testing. Below are test cases you can discuss during the interview to show your depth of understanding.

- ["file.txt", "document.TXT", "image.png"] !’ Expect counts: {'txt': 2, 'png': 1}.
- ["archive.tar.gz", "video.mp4", "audio.mp3"] !’ Counts: {'gz': 1, 'mp4': 1, 'mp3': 1}.
- [".bashrc", ".gitignore"] !’ No extensions; counts: {}.
- ["file", "anotherfile."] !’ No valid extensions; counts: {}.
- ["README", "LICENSE", "setup.py"] !’ Counts: {'py': 1}.

During the interview, walk through how your algorithm handles each scenario and explain why it behaves correctly.

Common Pitfalls to Avoid

1. Splitting on Every Dot: Using split('.') will treat archive.tar.gz as three parts, leading to incorrect extension extraction.
2. Case Sensitivity: Failing to normalize case can produce duplicate keys (e.g., PNG vs. png).
3. Hidden Files: Ignoring files that start with a dot but have no extension may lead to missing edge cases.
4. Trailing Dots: Files ending with a dot (e.g., file.) should be treated as having no extension.

Interview Ready Tips

1. Clarify the Problem Early

Ask the interviewer questions such as:

- “Should we treat file.TXT and file.txt as the same extension?”

- “Do we need to return a mapping or just the highest count?”
- “How should we handle files with no extension?”

Clarifying assumptions saves time and demonstrates professionalism.

2. Communicate Your Thought Process

Describe the algorithm verbally as you code. For example:

“I’ll iterate over each file name, split it from the rightmost dot, and store the extension in a dictionary. If the split yields only one part, it means there’s no extension, so I’ll skip it.”

3. Discuss Complexity

After writing the solution, state its time and space complexities. For the hash map approach, explain why it’s O(n) time and O(k) space.

4. Offer Alternative Solutions

Show that you’re not just coding but thinking critically. Mention alternatives like:

- Using regular expressions to extract extensions.
- Sorting the list and counting consecutive duplicates.
- Leveraging built in functions in languages like pathlib in Python.

Sample Code in Multiple Languages

Python

```
def filetyp_counts(file_names):
    counts = {}
    for name in file_names:
        if name.startswith('.') and name.count('.') == 1:
            continue
        parts = name.rsplit('.', 1)
        if len(parts) == 2:
            ext = parts[1].lower()
            counts[ext] = counts.get(ext, 0) + 1
    return counts
```

JavaScript (Node.js)

```
function filetypCounts(fileNames) {
  const counts = new Map();
  for (const name of fileNames) {
    if (name.startsWith('.') && name.split('.').length === 2) continue;
    const parts = name.split('.');
    if (parts.length
```

Java

```
import java.util.*;

public class Filetyp {
    public static Map&lt;String, Integer&gt; filetypCounts(List&lt;String&gt; fileNames) {
        Map&lt;String, Integer&gt; counts = new HashMap();
        for (String name : fileNames) {
            if (name.startsWith(".") && name.split("\\.").length == 2) continue;
            String[] parts = name.split("\\.");
            if (parts.length
```

Practice Problems to Reinforce Your Skills

- File Extension Frequency: Given a large dataset of file paths, return the top 5 most common extensions.
- Hidden Files Count: Count how many files in a directory are hidden (start with a dot).

Baca Juga (Artikel Terkait)

Tags: #cracking #coding #interview #filetyp