🐆

Explanation

A **classic Python gotcha**. While these two lines look similar, they behave **very differently** because of how Python handles **list multiplication** and object references.

🔍 Code 1 — ✅ Correct and Independent Sublists:

```
freq_bucket = [[] for _ in range(m+1)]
```

- This creates a **new empty list** `[]` for each iteration.
- So, each element of `freq_bucket` is a **separate list object**.
- Modifying one sublist will **not** affect the others.

✅ Example:

```
freq_bucket[0].append(1)
print(freq_bucket[0])  # [1]
print(freq_bucket[1])  # [] ✅ different list
```

🔍 Code 2 — ❌ Shared References:

```
freq_bucket = [[]] * (m+1)
```

- This creates **one empty list** `[]` and **copies the reference** `m+1` times.
- All sublists point to the **same list object**.
- Modifying one sublist will affect **all of them**.

❌ Example:

```
freq_bucket[0].append(1)
print(freq_bucket[0])  # [1]
print(freq_bucket[1])  # [1] ❌ same list!
```

🔬 Visual Summary

Statement	Separate Lists?	Safe to Modify Individually?
`[[] for _ in range(m+1)]`	✅ Yes	✅ Yes
`[[]] * (m+1)`	❌ No (shared)	❌ No

✅ Use Case Recommendation

Always use:

```
[[] for _ in range(m+1)]
```

when you want to initialize a list of independent sublists (e.g., for **bucket sort**, **grouping**, or **adjacency lists**).
