Map Sum Pairs

Implement a MapSum class with `insert`, and `sum` methods.

For the method `insert`, you'll be given a pair of (string, integer). The string represents the key and the integer represents the value. If the key already existed, then the original key-value pair will be overridden to the new one.

For the method `sum`, you'll be given a string representing the prefix, and you need to return the sum of all the pairs' value whose key starts with the prefix.

Example 1:

```
Input: insert("apple", 3), Output: Null
Input: sum("ap"), Output: 3
Input: insert("app", 2), Output: Null
Input: sum("ap"), Output: 5
```

Solution 1

```
class MapSum {
public:
    /** Initialize your data structure here. */
    void insert(string key, int val) {
        mp[key] = val;
    }

    int sum(string prefix) {
        int sum = 0, n = prefix.size();
        for (auto it = mp.lower_bound(prefix); it != mp.end() && it->first.substr(0,
n) == prefix; it++)
            sum += it->second;
        return sum;
    }
private:
    map<string, int> mp;
};
```

written by zestypanda original link here

Solution 2

UPDATE : Okay, let me tell you that even though solution look so concise, why you should NOT do this.
It's because of `s += c`. This operation is not O(1), it's O(String::length), which makes for loop to be k^2. And this will break when string is long. Try it yourself as learning with this input for insert - https://pastebin.com/Pjymymgh

But if the constraint is that the string are small, like dictionary words or people names, then it should be good.

The key idea is to keep two hash maps, one with just original strings. The other with all prefixes.

When a duplicate insert is found, then update all it's prefixes with the difference of previous value of the same key(take it from original map)

Time Complexity for sum is `O(1)`
Time Complexity for insert is ~~`O(len(key))`~~ `O(len(key) ^ 2)`

```
/** Initialize your data structure here. */
Map<String, Integer> map;
Map<String, Integer> original;
public MapSum() {
    map = new HashMap<>();
    original = new HashMap<>();
}

public void insert(String key, int val) {
    val -= original.getOrDefault(key, 0); // calculate the diff to be added to prefi
xes
    String s = "";
    for(char c : key.toCharArray()) {
        s += c; // creating all prefixes
        map.put(s, map.getOrDefault(s, 0) + val); //update/insert all prefixes with
new value
    }
    original.put(key, original.getOrDefault(key, 0) + val);
}

public int sum(String prefix) {
    return map.getOrDefault(prefix, 0);
}
```

written by wavy original link here

Solution 3

A standard `Trie`-based solution where each node keeps track of the total count of its children.

For inserting, we first determine if the string already exists in the Trie. If it does, we calculate the difference in the previous and new value, and update the nodes with the difference as we traverse down the Trie nodes.

Sum is simple because each node already holds the sum of its children and we simply have to traverse to the node and obtain its count.

This results in both operations being O(k), where k is the length of the string/prefix.

```python
class TrieNode():
    def __init__(self, count = 0):
        self.count = count
        self.children = {}

class MapSum(object):

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = TrieNode()
        self.keys = {}

    def insert(self, key, val):
        """
        :type key: str
        :type val: int
        :rtype: void
        """
        # Time: O(k)
        curr = self.root
        delta = val - self.keys.get(key, 0)
        self.keys[key] = val

        curr = self.root
        curr.count += delta
        for char in key:
            if char not in curr.children:
                curr.children[char] = TrieNode()
            curr = curr.children[char]
            curr.count += delta

    def sum(self, prefix):
        """
        :type prefix: str
        :rtype: int
        """
        # Time: O(k)
        curr = self.root
        for char in prefix:
            if char not in curr.children:
                return 0
            curr = curr.children[char]
        return curr.count
```

written by yangshun original link here