Problem 1656: Design an Ordered Stream

Problem Information

Difficulty: Easy
Acceptance Rate: 82.39%
Paid Only: No
Tags: Array, Hash Table, Design, Data Stream

Problem Description

There is a stream of `n` `(idKey, value)` pairs arriving in an **arbitrary** order, where `idKey` is an integer between `1` and `n` and `value` is a string. No two pairs have the same `id`.

Design a stream that returns the values in **increasing order of their IDs** by returning a **chunk** (list) of values after each insertion. The concatenation of all the **chunks** should result in a list of the sorted values.

Implement the `OrderedStream` class:

* `OrderedStream(int n)` Constructs the stream to take `n` values. * `String[] insert(int idKey, String value)` Inserts the pair `(idKey, value)` into the stream, then returns the **largest possible chunk** of currently inserted values that appear next in the order.

Example:

![](https://assets.leetcode.com/uploads/2020/11/10/q1.gif)

Input ["OrderedStream", "insert", "insert", "insert", "insert", "insert"] [[5], [3, "ccccc"], [1, "aaaaa"], [2, "bbbbb"], [5, "eeeee"], [4, "ddddd"]] **Output** [null, [], ["aaaaa"], ["bbbbb", "ccccc"], [], ["ddddd", "eeeee"]] **Explanation** // Note that the values ordered by ID is ["aaaaa", "bbbbb", "ccccc", "ddddd", "eeeee"]. OrderedStream os = new OrderedStream(5); os.insert(3, "ccccc"); // Inserts (3, "ccccc"), returns []. os.insert(1, "aaaaa"); // Inserts (1, "aaaaa"), returns ["aaaaa"]. os.insert(2, "bbbbb"); // Inserts (2, "bbbbb"), returns ["bbbbb", "ccccc"]. os.insert(5, "eeeee"); // Inserts (5, "eeeee"), returns []. os.insert(4, "ddddd"); // Inserts (4, "ddddd"), returns ["ddddd", "eeeee"]. // Concatentating all the chunks returned: // [] + ["aaaaa"] + ["bbbbb", "ccccc"] + [] + ["ddddd", "eeeee"] = ["aaaaa", "bbbbb", "ccccc", "ddddd", "eeeee"] // The resulting order is the same as the order above.

Constraints:

* `1 <= n <= 1000` * `1 <= id <= n` * `value.length == 5` * `value` consists only of lowercase letters. * Each call to `insert` will have a unique `id.` * Exactly `n` calls will be made to `insert`.

Code Snippets

C++:

```
class OrderedStream {
public:
OrderedStream(int n) {

}

vector<string> insert(int idKey, string value) {

}
};

/**
* Your OrderedStream object will be instantiated and called as such:
* OrderedStream* obj = new OrderedStream(n);
* vector<string> param_1 = obj->insert(idKey,value);
*/
```

Java:

```
class OrderedStream {

public OrderedStream(int n) {

}

public List<String> insert(int idKey, String value) {

}
}
```

```
/**
* Your OrderedStream object will be instantiated and called as such:
* OrderedStream obj = new OrderedStream(n);
* List<String> param_1 = obj.insert(idKey,value);
*/
```

Python3:

```
class OrderedStream:

def __init__(self, n: int):



def insert(self, idKey: int, value: str) -> List[str]:




# Your OrderedStream object will be instantiated and called as such:
# obj = OrderedStream(n)
# param_1 = obj.insert(idKey,value)
```