5896. Stock Price Fluctuation

My Submissions (/contest/weekly-contest-262/problems/stock-price-fluctuation/submissions/) Back to Contest (/contest/weekly-contest-262/)

User Accepted:	1585
User Tried:	2922
Total Accepted:	1619
Total Submissions:	6338
Difficulty:	Medium

You are given a stream of **records** about a particular stock. Each record contains a **timestamp** and the corresponding **price** of the stock at that timestamp.

Unfortunately due to the volatile nature of the stock market, the records do not come in order. Even worse, some records may be incorrect. Another record with the same timestamp may appear later in the stream **correcting** the price of the previous wrong record.

Design an algorithm that:

- **Updates** the price of the stock at a particular timestamp, **correcting** the price from any previous records at the timestamp.
- Finds the **latest price** of the stock based on the current records. The **latest price** is the price at the latest timestamp recorded.
- Finds the **maximum price** the stock has been based on the current records.
- Finds the **minimum price** the stock has been based on the current records.

Implement the `StockPrice` class:

- `StockPrice()` Initializes the object with no price records.
- `void update(int timestamp, int price)` Updates the `price` of the stock at the given `timestamp`.
- `int current()` Returns the **latest price** of the stock.
- `int maximum()` Returns the **maximum price** of the stock.
- `int minimum()` Returns the **minimum price** of the stock.

Example 1:

```
Input
["StockPrice", "update", "update", "current", "maximum", "update", "maximum", "update", "minimum"]
[[], [1, 10], [2, 5], [], [], [1, 3], [], [4, 2], []]
Output
[null, null, null, 5, 10, null, 5, null, 2]

Explanation

StockPrice stockPrice = new StockPrice();
stockPrice.update(1, 10); // Timestamps are [1] with corresponding prices [10].
stockPrice.update(2, 5);  // Timestamps are [1,2] with corresponding prices [10,5].
stockPrice.current();     // return 5, the latest timestamp is 2 with the price being 5.
stockPrice.maximum();     // return 10, the maximum price is 10 at timestamp 1.
stockPrice.update(1, 3);  // The previous timestamp 1 had the wrong price, so it is updated to 3.
                          // Timestamps are [1,2] with corresponding prices [3,5].
stockPrice.maximum();     // return 5, the maximum price is 5 after the correction.
stockPrice.update(4, 2);  // Timestamps are [1,2,4] with corresponding prices [3,5,2].
stockPrice.minimum();     // return 2, the minimum price is 2 at timestamp 4.
```

Constraints:

- `1 <= timestamp, price <= 10^9`
- At most `10^5` calls will be made **in total** to `update`, `current`, `maximum`, and `minimum`.
- `current`, `maximum`, and `minimum` will be called **only after** `update` has been called **at least once**.

Java

```
1  // package leetcode.weekly.r262;
2
3  import java.util.*;
4  import java.io.*;
```

```
 5
 6  class StockPrice {
 7      static PrintWriter pw;
 8
 9      TreeSet<Node> se;
10      TreeMap<Integer, Integer> m;
11
12      public StockPrice() {
13          se = new TreeSet<>();
14          m = new TreeMap<>();
15      }
16
17      public void update(int timestamp, int price) {
18          if (m.containsKey(timestamp)) {
19              int prePrice = m.get(timestamp);
20              Node n = new Node(timestamp, prePrice);
21              se.remove(n);
22              se.add(new Node(timestamp, price));
23          } else {
24              se.add(new Node(timestamp, price));
25          }
26          m.put(timestamp, price);
27      }
28
29      public int current() {
30          return m.lastEntry().getValue();
31      }
32
33      public int maximum() {
34          // debug();
35          return se.last().price;
36      }
37
38      public int minimum() {
39          // debug();
40          return se.first().price;
41      }
42
43      private void debug () {
44          for (Node n: se) {
45              pw.println("debug" + " " + n.timestamp + " " + n.price);
46          }
47      }
48
49      class Node implements Comparable<Node> {
50          int timestamp, price;
51
52          Node(int timestamp, int price) {
53              this.timestamp = timestamp;
54              this.price = price;
55          }
56
57          @Override
58          public int compareTo(Node y) {
59              if (price == y.price) return Integer.compare(timestamp, y.timestamp);
60              return Integer.compare(price, y.price);
61          }
62      }
63
64      public void run() {
65          StockPrice stockPrice = new StockPrice();
66          stockPrice.update(1, 10);
67          stockPrice.update(2, 5);
68          pr(stockPrice.current());     // 5
69          pr(stockPrice.maximum());     // 10
70          stockPrice.update(1, 3);
71          pr(stockPrice.maximum());     // 5
72          stockPrice.update(4, 2);
73          pr(stockPrice.minimum());     // 2
74      }
75
76      void pr(int num) {
```

```
77          pw.println(num);
78      }
79
80  ▾   void pr(long num) {
81          pw.println(num);
82      }
83
84  ▾   void pr(double num) {
85          pw.println(num);
86      }
87
88  ▾   void pr(String s) {
89          pw.println(s);
90      }
91
92  ▾   void pr(char c) {
93          pw.println(c);
94      }
95
96  ▾   public static void main(String[] args) {
97          pw = new PrintWriter(System.out);
98          new StockPrice().run();
99          pw.close();
100     }
101
102 ▾   void tr(Object... o) {
103         pw.println(Arrays.deepToString(o));
104     }
105 }
106
107
108
```

☐ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/568675666/) More Details › (/submissions/detail/568675666/)

Share your acceptance!

1