Algorithm 5: InsertAndSet and GetValue on a multimap M.

InsertAndSet(r, t)

Input: A ridge r and a facet t

Maintains:

A hash table R with linear probing from keys as ridges and values as facets. Each cell in R has three fields: two boolean flags, `taken` and `check`, and `data` which will store the corresponding key-value pair. Assume the hash function for R is f_R, which hashes a ridge r to an index in the range $[0, \ldots, |R| - 1]$.

Output: If $r \in M$, add t as the second value of r in M and return false. Otherwise, add t as the first value of r in M and return true.

```
1  function M.InsertAndSet(r, t)
2      i ← f_R(r)                          // get the starting index
3      while ¬TestAndSet(R[i].taken) do
4          i ← (i + 1) Mod M.size
5      R[i].data ← (r, t)                  // write entry to data
6      i ← f_R(r)                          // start from initial index
7      while R[i].taken do
8          if R[i].data.key = r then
9              if ¬TestAndSet(R[i].check) then
10                 return False
11         i ← (i + 1) Mod M.size
12     return True
```

GetValue(r, t)

Input: A ridge r and a facet t.

Output: The values $t' \neq t$ associated with r in M.

```
13 function M.GetValue(r, t)
14     i ← f_R(r)                          // get the starting index
15     while R[i].taken do
16         if R[i].data.key = r then
17             t ← R[i].data.value
18             if t' ≠ t then return t'
19         i ← (i + 1) Mod M.size
```

over R starting at the initial index from the hash value, and tries to find r in R. It stops when it sees an empty slot. Whenever the algorithm sees a slot with key r, it performs a TestAndSet on the `check` field of the slot. If the TestAndSet on Line 9 fails, we let this InsertAndSet return false (Line 10), and the corresponding facet t will process r in Algorithm 3. Otherwise, if a facet t does not fail the TestAndSet, it returns true (Line 12). Note that when making the second pass, a InsertAndSet(r, t) algorithm may see a slot with a key other than r due to linear probing. The algorithm thus needs to check if the key is equal to r (Line 8) during the while-loop.

The GetValue(r, t) algorithm starts at the index of the hash value of r, and scans until it finds a facet $t' \neq t$ associated with r.

We next prove the correctness of the algorithm. We first prove that InsertAndSet works as expected. It is straightforward that the insertion into the hash table (Lines 2–5) works as expected. We then need to show that the returned boolean value of InsertAndSet works as expected. Note that for a ridge r, there will be exactly two facets that call InsertAndSet on r throughout the algorithm. A correct InsertAndSet algorithm allows for exactly one of them to return False. This unsuccessful InsertAndSet will take over to recurse on ridge r on Line 22 of Algorithm 3.

Theorem A.1. *For two invocations to* InsertAndSet(r, t_1) *and* InsertAndSet(r, t_2)*, exactly one of them returns* False.

Proof. Suppose the two indices for ridge r in the hash table R are i (reserved by t_1) and $j > i$ (reserved by t_2). The other case is symmetric. We first show that there must be one of t_1 and t_2 that returns False in InsertAndSet.

Case 1. We call TestAndSettwice on $R[i]$.check on Line 9. Then, the second one must fail and return False.

Case 2. We call TestAndSetonly once on $R[i]$.check on Line 9. This can only happen if when t_2 is making the second pass, t_1 has marked $R[i]$.taken, but has not written its key r to $R[i]$.data. Then, for both t_1 and t_2, when they make the second pass, they must both reach $R[j]$ and call TestAndSet on $R[j]$.check on Line 9. One of them has to fail and return False.

Secondly, we show that it is impossible that t_1 and t_2 both return false in InsertAndSet. The TestAndSet on $R[i]$.check can only fail once, and similarly for the TestAndSet on $R[j]$.check. If both t_1 and t_2 return false, then this means that Line 9 fails on both $R[i]$ and $R[j]$. This is impossible because whichever fails on $R[i]$ will not proceed to $R[j]$, and so $R[j]$ cannot be reached twice. □

We next show that GetValue(r, t) works as expected. In particular, both facets t' and t'' associated with r in R must have been inserted into R, and thus the one that is not equal to t will be found and returned.

Theorem A.2. *When* GetValue(r, t) *is called, the two facets* t_1 *and* t_2 *incident on* r *have both been added into* R.

Proof. A GetValue(r, t) is called only when an invocation of InsertAndSet(r, t) returns False on Line 20 of Algorithm 3. This InsertAndSet fails because it fails on Line 9 of Algorithm 5. This means that the key-value pair (r, t) itself has been added to R on Line 5 of Algorithm 5.

Suppose that the other facet incident on r is t'. We will show that (r, t') has also been added to R. Since (r, t) fails on the TestAndSet on Line 9 of Algorithm 5, it means that the `check` flag has been already set to True by (r, t') before (r, t) processes it. Since the algorithm InsertAndSet(r, t') reached Line 9 to set the `check` flag, it also must have finished adding the data on Line 5. □

In summary, the multimap M with its two functions InsertAndSet and GetValue works as expected by using TestAndSet. The work and depth are just the cost of linear probing, which is $O(\log n)$ *whp*.