Algorithm 1 HRindex 插入更新算法

```
1: function updateInsertEdge(src, dst, timestamp)
2:     G_t = originGraph[timestamp]
3:     SCC_t = SCCGraph[timestamp]
4:     if !G_t.exist(src) then
5:         updateAddNode(ur, oldHRindex)
6:     end if
7:     if !G_t.exist(dst) then
8:         updateAddNode(ur, oldHRindex)
9:     end if
10:    G_t.insert(ur.src, ur.dst)
11:    SCC_src = SCC_t.find(ur.src)
12:    SCC_dst = SCC_t.find(ur.dst)
13:    if SCC_src == SCC_dst then
14:        return
15:    else
16:        SCCGraph[timestamp].insert(src, dst)
17:        cycle ← findCycle(SCCGraph[timestamp])
18:        while cyele.size != 0 do
19:            S_new ← merge(SCCGraph[timestamp], cycle)
20:            ▷ Firstly, we process the newly added SCC node after merging.
21:            In = SCC_t.getIncomingEdge(S_new)
22:            Out = SCC_t.getOutgoingEdge(S_new)
23:            if newSCCID exist in other timestamp then
24:            else
25:                NIT.push(getNITItem(In, Out))
26:            end if
27:                ▷ Secondly, we delete the SCC node existing in the cycle and
   process the node which is connected which these node.
28:            for each item IN NIT do
29:                S_i = item.node
30:                if S_i in cycle then
31:                                ▷ the new SCC node serves as an outgoing edge or
   incoming edge of this node
32:                    if S_i in In then
33:                                    ▷ S_i in In means that S_i has an outgoing egde
   < S_i, S_new >
34:                    end if
35:                    if S_i in Out then
36:                                ▷ S_i in Out means that S_i has an incoming egde
   < S_new, S_i >
37:                    end if
38:                else
39:                end if
40:            end for
41:        end while
42:    end if
43: end function
```