```
1   boolean primarySeekForDelete(deleteSeekRecord, deleteKey)
2   {
3     while(true)
4     {
5       loop until a null node is reached OR promote flag is set on the node OR deleteKey is
        found
6       if keyfound
7       {
8         if the key is marked
9           restart primary seek
10        else
11          update deleteSeekRecord with pnode,node,lastUnmarkedEdge
12          return true
13      }
14      else
15      {
16        if lastRightNode's key has not changed
17          return false
18        else
19          restart primary seek
20      }
21    }
22  }
23  //if special case secondary seek returns true
24  boolean secondarySeekForDelete(node,nRChild,secDeleteSeekRecord)
25  {
26    if special case
27      return true
28    else
29      keep going left until the secondary node is located
30      once the secondary node is located, populate the secDeleteSeekRecord
31      with secondary node,its left & right child, its parent and the
        lastSecondaryUnmarkedEdge
32      return false
33  }
34
35  delete pseudocode
36  set CLEANUP and isSimpleDelete flags to false
37  while(true)
38  {
39    if(primarySeekForDelete(deleteSeekRecord, deleteKey) returned false)
40      return false
41    else
42      use the values of pnode, node and lastUnmarkedEdge from the deleteSeekRecord
43
44    if(!CLEANUP)
45    {
46      try CAS(node->lChild,<nlChildAddr,0,0>,<nlChildAddr,1,0>)
47      if CAS FAILED
48        help
49        continue from top of primary seek's while loop
50      if CAS SUCCEEDED
51        set CLEANUP to true and set storedNode = node
52    }
53    if(storedNode != node) //Someone removed the node for me. So DONE
```

```
54      set "deleteFlag" on node's rChild using BTS
55      if complex delete
56      {
57        while(true) //secondary seek
58        {
59          nRChild = node->rChild
60          if(nRChild != NULL)
61            isSplCase = secondarySeekForDelete(node,nRChild,secDeleteSeekRecord)
62            use the values of secondary node,its lchild,its rchild, its parent,
              SeclastUnmarkedEdge from secDeleteSeekRecord
63          else
64            set isSimpleDelete flag to true
65            break from while loop
66          if node key is unmarked
67          {
68            try CAS(rnode->lChild,<NULL,0,0>,<nodeAddr,0,1>)
69            if CAS failed
70            {
71              if promoteFlag is set
72                if address does not match with node's address
73                  restart primary seek. assert(node->secFlag == DONE)
74                  break from while loop
75              else
76                if address != NULL //restart secondary seek
77                  continue from top of secondary seek's while loop
78                else
79                  assert(rnode->lChild's deleteFlag is set)
80                  help operation at secondaryLastUnmarkedEdge
81                  if secondaryLastUnmarkedEdge does not exist, then help node->rChild
                    //simplehelp(node,nrChild)
82                  continue from top of secondary seek's while loop
83            }
84            set promote flag on rnode->rChild using BTS
85            promote key using a simple write. Node's key changes from <0,kN> to <1,kRN>
86          }
87          if(!isSplCase)
88          {
89            try CAS(rpnodeLChild,<rnode,0,0>,<rnodeRChild,0,0>) //remove secondary node
90            if CAS FAILED, help operation at secondaryLastUnmarkedEdge
91              if secondaryLastUnmarkedEdge doesn't exist, override CASinvariant and help
                node->rChild  //simplehelp(node,nrChild)
92              continue from top of secondary seek's while loop
93            if CAS SUCCEEDED, set node->secDoneFlag to true
94          }
95          else
96          {
97              try CAS(nodeRChild,<rnode,1,0>,<rnodeRChild,1,0> //no problem if CAS fails
98              set node->secDoneFlag to true
99          }
100         oldNodeAddr = address of node
101         while(true)
102         {
103           create a fresh copy of node
104           newNodeKey as <0,kRN>
105           newNodeLChild as <node's lChildAddr,0,0>
```

```
106           newNodeRChild = <node's rChildAddr,0,0>
107           try CAS(pnode->lChild,<node,0,0>,<newNode,0,0>)
108           if CAS SUCCEEDED then DONE
109           if CAS FAILED
110             if address has changed
111               then someone helped me install a fresh copy.so DONE
112             else
113               CAS has failed coz the edge is marked.
114               if lastUnmarkedEdge is not (pnode,node) then help
115               do primarySeekForDelete(node->key) //restart primary seek with new key
116               if the new key is not found then someone has installed a fresh copy. So done

117               if key is found and newNodeAddr != oldNodeAddr then someone has installed a
                  fresh copy. So done
118         }
119       }
120     }
121     else //simple delete
122       set isSimpleDelete to true
123     if(isSimpleDelete)
124     {
125       try CAS(pnode->lChild,<node,0,0>,<node's l/r child,0,0>)
126       if CAS SUCCEEDED, then DONE
127       if CAS FAILED
128         if lastUnmarkedEdge is NOT (pnode,node) help
129     }
130   }
131   //simplehelp(node,node's rChild)
132   simplehelp(pnode,node)
133   assert(pnode's secDoneFlag not set)
134   set delete flag on node->rChild using BTS
135   if complex delete
136     if node->secDoneFlag is set
137       create a fresh copy of node
138       try CAS(pnode->rChild,<node,1,0>,<newNode,1,0>)
139   else //simple delete
140     try CAS(pnode->rchild,<node,1,0>,<node's lchild,1,0>)
```