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

```
54      locally store the value of rChild of node
55      if complex delete //if both lChild and rChild are nonNULL
56      {
57        while(true) //secondary seek
58        {
59          update the local value of rChild of node
60          if(nRChild != NULL) //check the null bit to determine NULL values
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            if(node key is marked)
65              set secDoneFlag to true
66            else
67              set isSimpleDelete flag to true
68              break from while loop
69          if node key is unmarked
70          {
71            try CAS(rnode->lChild,<anyAddress,1,0,0>,<nodeAddr,1,0,1>)
72            if CAS failed
73            {
74              if promoteFlag is set
75                if address does not match with node's address
76                  assert(node->secFlag == DONE)
77                  return
78              else
79                if address != NULL // (check the null bit to determine NULL values) restart
                  secondary seek
80                  continue from top of secondary seek's while loop
81                else
82                  assert(rnode->lChild's deleteFlag is set)
83                  help operation at secondaryLastUnmarkedEdge
84                  if secondaryLastUnmarkedEdge does not exist, then help node->rChild
                    //simplehelp(node,nrChild)
85                  continue from top of secondary seek's while loop
86            }
87            set promote flag on rnode->rChild using BTS
88            locally store the value of rnode->rChild
89            promote key using a simple write. Node's key changes from <0,kN> to <1,kRN>
90          } //end if node key is unmarked
91          if(promote flag set in lcrnode && addr(lcrnode) == addr(node)) //lcrnode denotes
            left child of rnode
92          {
93            if(!isSplCase)
94            {
95              if(rnodeRchild != NULL)
96                try CAS(rpnodeLChild,<rnode,0,0,0>,<rnodeRChild,0,0,0>) //remove secondary
                  node
97              else
98                try CAS(rpnodeLChild,<rnode,0,0,0>,<rnode,1,0,0>) //remove secondary node
99              if CAS FAILED, help operation at secondaryLastUnmarkedEdge
100               if secondaryLastUnmarkedEdge doesn't exist, override CASinvariant and help
                  node->rChild  //simplehelp(node,nrChild)
101               continue from top of secondary seek's while loop
102             if CAS SUCCEEDED, set node->secDoneFlag to true
```

```
103           }
104           else
105           {
106             if(rnodeRchild != NULL)
107               try CAS(nodeRChild,<rnode,0,1,0>,<rnodeRChild,0,1,0> //no problem if CAS fails
108             else
109               try CAS(nodeRChild,<rnode,0,1,0>,<rnode,1,1,0> //no problem if CAS fails
110             set node->secDoneFlag to true
111           }
112         } //end if promote flag is set in lcrnode
113         oldNodeAddr = address of node
114         create a fresh copy of node
115         newNodeKey as <0,kRN>
116         newNodeLChild as <node's lChildAddr,0,0,0>
117         newNodeRChild = <node's rChildAddr,0,0,0>
118
119         while(true) //install fresh copy
120         {
121           if(newNodeKey < pnode->key)
122             try CAS(pnode->lChild,<node,0,0,0>,<newNode,0,0,0>)
123           else
124             try CAS(pnode->rChild,<node,0,0,0>,<newNode,0,0,0>)
125           if CAS SUCCEEDED then DONE
126           if CAS FAILED
127             if oldNodeAddr != address returned by CAS
128               then someone helped me install a fresh copy.so DONE
129             else
130               CAS has failed coz the edge is marked.
131               if lastUnmarkedEdge is not (pnode,node) then help
132               do primarySeekForDelete(newNodeKey) //restart primary seek with new key
133               if the new key is not found then someone has installed a fresh copy. So done

134               if key is found and newNodeAddr != oldNodeAddr then someone has installed a
                  fresh copy. So done
135         } //end while install fresh copy
136       } //end secondary seek while
137     } //end if complex delete
138     else //simple delete
139       set isSimpleDelete to true
140     if(isSimpleDelete)
141     {
142       if both l & r child of node are NULL
143         try CAS(pnode->lChild,<node,0,0,0>,<node,1,0,0>)
144       else
145         try CAS(pnode->lChild,<node,0,0,0>,<nonNullChild,0,0,0>)
146       if CAS SUCCEEDED, then DONE
147       if CAS FAILED
148         if lastUnmarkedEdge is NOT (pnode,node) help
149     }
150   } //end main while
151
152   //simplehelp(node,node's rChild)
153   simplehelp(pnode,node)
154   assert(pnode's secDoneFlag not set)
155   set delete flag on node->rChild using BTS
```

```
156   locally store the lChild and rChild values of node
157   if complex delete //if both lChild and rChild are nonNULL
158     if node->secDoneFlag is set
159       create a fresh copy of node
160       try CAS(pnode->rChild,<node,0,1,0>,<newNode,0,1,0>)
161   else //simple delete
162     if both l & r child of node are NULL
163       try CAS(pnode->rchild,<node,0,1,0>,<node,1,1,0>)
164     else
165       try CAS(pnode->rchild,<node,0,1,0>,<nonNullChild,0,1,0>)
```