```
bool remove(struct threadArgs* tData, unsigned long deleteKey)
{
  struct node* pnode;
  struct node* node;
  unsigned long lastRightKey;
  struct node* lastRightNode;
  unsigned long nodeKey;
  struct node* lastUnmarkedPnode;
  struct node* lastUnmarkedNode;
  struct node* storedNode;
  bool CLEANUP = false;
  bool RESTART = false;
  bool isLeft = true;
  bool isSimpleDelete=false;
  tData->deleteCount++;

  while(true)
  {
    RESTART = false;
    if(!primarySeekForDelete(tData->dsr, deleteKey))
    {
      tData->unsuccessfulDeletes++;
      return(false);
    }
    pnode = tData->dsr->pnode;
    node = tData->dsr->node;
    assert(!isNull(node));
    lastUnmarkedPnode = tData->dsr->lastUnmarkedPnode;
    lastUnmarkedNode = tData->dsr->lastUnmarkedNode;

    if(getAddress(pnode)->lChild == node) //left case
    {
      isLeft = true;
    }
    else if(getAddress(pnode)->rChild == node) //right case
    {
      isLeft = false;
    }
    else
    {
      RESTART = true;
    }
    if(!RESTART)
    {
      if(!CLEANUP)
      {
        struct node* nlChild = getAddress(node)->lChild;
        struct node* nlChildWithDFlagSet = setDeleteFlag(nlChild);
        if(getAddress(node)->lChild.compare_and_swap(nlChildWithDFlagSet, getAddress(nlChild))
        != getAddress(nlChild)) //setting DFlag on node's lChild
        {
          help(); //CAS failed so help
          RESTART = true;
        }
        else //CAS succeeded
        {
          assert(isNodeMarked(getAddress(node)->lChild));
          assert(getAddress(node)->lChild == nlChildWithDFlagSet);
          CLEANUP = true;
          storedNode = node;
          assert(!isNull(storedNode));
        }
      }

      if(!RESTART)
      {
        assert(CLEANUP);
        if(storedNode != node) //Someone else removed the node for me
        {
          return(true);
        }
        btsOnDeleteFlag((struct node**)&getAddress(node)->rChild);
        assert(isDeleteFlagSet(getAddress(node)->lChild));
        assert(isDeleteFlagSet(getAddress(node)->rChild));
        //examine which case applies
        if(!isNull(getAddress(node)->lChild) && !isNull(getAddress(node)->rChild)) //possible
        complex delete
        {
          struct node* rpnode;
          struct node* rnode;
          struct node* lcrnode;
          struct node* rcrnode;
          struct node* secondaryLastUnmarkedPnode;
          struct node* secondaryLastUnmarkedNode;
          bool isSplCase;
          bool assumeCASsucceeded;
          bool SECONDARY_RESTART;
          while(true)
          {
            assumeCASsucceeded = false;
            SECONDARY_RESTART = false;
            isSplCase = secondarySeekForDelete(node, tData->sdsr);
            rpnode = tData->sdsr->rpnode;
            rnode = tData->sdsr->rnode;
            lcrnode = tData->sdsr->lcrnode;
            rcrnode = tData->sdsr->rcrnode;
            secondaryLastUnmarkedPnode = tData->sdsr->secondaryLastUnmarkedPnode;
            secondaryLastUnmarkedNode = tData->sdsr->secondaryLastUnmarkedNode;
            assert(isNull(lcrnode));
            if(!isKeyMarked(getAddress(node)->key)) //if node's key is marked then someone
            else has done the below steps for this thread
            {
              struct node* nodeAddrWithPromoteFlagSet = setPromoteFlag(getAddress(node));
              struct node* CASoutput;
              CASoutput = getAddress(rnode)->lChild.compare_and_swap(nodeAddrWithPromoteFlagSet
              ,NULL);
              if(CASoutput != NULL) //CAS failed
              {
                if(isPromoteFlagSet(CASoutput))
                {
                  if(getAddress(CASoutput) == getAddress(node))
                  {
                    assumeCASsucceeded = true;
                  }
                  else
                  {
```

```
          //restart primary seek. assert(node->secFlag == DONE)
          RESTART = true;
          break; //start from primary seek
        }
      }
      else
      {
        if(!isNull(CASoutput))
        {
          //restart secondary seek
          SECONDARY_RESTART = true;
        }
        else
        {
          assert(isDeleteFlagSet(getAddress(rnode)->lChild));
          //help operation at secondaryLastUnmarkedEdge
          //if secondaryLastUnmarkedEdge does not exist, then help node->rChild
          RESTART = true;
          break; //start from primary seek
        }
      }
    }
    else //CAS succeeded
    {
      assumeCASsucceeded = true;
    }
    if(assumeCASsucceeded)
    {
      btsOnPromoteFlag((struct node**)&getAddress(rnode)->rChild); //set promote
      flag on rnode->rChild using BTS
      getAddress(node)->key = setReplaceFlagInKey(getAddress(rnode)->key); //node's
      key changed from <0,kN> to <1,kRN>
    }
  }

  if(!SECONDARY_RESTART)
  {
    if(!isSplCase)
    {
      //try removing secondary node
      if(getAddress(rpnode)->lChild.compare_and_swap(getAddress(getAddress(rnode)->
      rChild),getAddress(rnode)) == getAddress(rnode))
      {
        getAddress(node)->secDoneFlag = true;
      }
      else
      {
        //help operation at secondaryLastUnmarkedEdge
        //if secondaryLastUnmarkedEdge does not exist, then override CASinvariant
        and help node->rChild
        RESTART = true;
        break; //start from primary seek
      }
    }
    else
    {
      getAddress(node)->secDoneFlag = true;
    }
```

```c
            if((getAddress(node)->rChild != NULL || (getAddress(node)->secDoneFlag)))
            {
              struct node* newNode = (struct node*) malloc(sizeof(struct node));
              newNode->key = getKey(getAddress(node)->key);
              newNode->lChild = getAddress(getAddress(node)->lChild);
              if(isSplCase)
              {
                newNode->rChild = getAddress(getAddress(rnode)->rChild);
              }
              else
              {
                newNode->rChild = getAddress(getAddress(node)->rChild);
              }
              struct node* PCASoutput;
              if(isLeft)
              {
                PCASoutput = getAddress(pnode)->lChild.compare_and_swap(newNode,getAddress(
                node));
                if( PCASoutput == getAddress(node))
                {
                  tData->successfulDeletes++;
                  tData->complexDeleteCount++;
                  return(true);
                }
              }
              else
              {
                PCASoutput = getAddress(pnode)->rChild.compare_and_swap(newNode,getAddress(
                node));
                if( PCASoutput == getAddress(node))
                {
                  tData->successfulDeletes++;
                  tData->complexDeleteCount++;
                  return(true);
                }
              }
              if(getAddress(PCASoutput) != getAddress(node))
              {
                return(true);
              }
              else
              {
                //CAS has failed coz the edge is marked. Help at lastUnmarkedEdge.
                //If lastUnmarkedEdge is (pnode,node) then restart
                RESTART = true;
                assert(getAddress(node)->secDoneFlag);
                break; //start from primary seek
              }
            }
            else
            {
              isSimpleDelete = true;
              break;
            }
          }
        }
      }
      else //simple delete
```

```
        {
          isSimpleDelete = true;
        }
        if(isSimpleDelete)
        {
          if(isLeft)
          {
            if(isNull(getAddress(node)->lChild))
            {
              if(getAddress(pnode)->lChild.compare_and_swap(getAddress(getAddress(node)->rChild
              ),getAddress(node)) == getAddress(node))
              {
                tData->successfulDeletes++;
                tData->simpleDeleteCount++;
                return(true);
              }
            }
            else
            {
              if(getAddress(pnode)->lChild.compare_and_swap(getAddress(getAddress(node)->lChild
              ),getAddress(node)) == getAddress(node))
              {
                tData->successfulDeletes++;
                tData->simpleDeleteCount++;
                return(true);
              }
            }
          }
          else
          {
            if(isNull(getAddress(node)->lChild))
            {
              if(getAddress(pnode)->rChild.compare_and_swap(getAddress(getAddress(node)->rChild
              ),getAddress(node)) == getAddress(node))
              {
                tData->successfulDeletes++;
                tData->simpleDeleteCount++;
                return(true);
              }
            }
            else
            {
              if(getAddress(pnode)->rChild.compare_and_swap(getAddress(getAddress(node)->lChild
              ),getAddress(node)) == getAddress(node))
              {
                tData->successfulDeletes++;
                tData->simpleDeleteCount++;
                return(true);
              }
            }
          }
          //if lastUnmarkedEdge is (pnode,node) then restart, else help
          RESTART = true;
        }
      }
    }
  }
}
```