```
boolean remove(T x) {
  Node<T> victim = null; boolean isMarked = false; int topLevel = -1;
  Node<T>[] preds = (Node<T>[]) new Node[MAX_LEVEL + 1];
  Node<T>[] succs = (Node<T>[]) new Node[MAX_LEVEL + 1];
  while (true) {
    int lFound = find(x, preds, succs);
    if (lFound != -1) victim = succs[lFound];
    if (isMarked ||
          (lFound != -1 &&
          (victim.fullyLinked
          && victim.topLevel == lFound
          && !victim.marked))) {
      if (!isMarked) {
        topLevel = victim.topLevel;
        victim.lock.lock();
        if (victim.marked) {
          victim.lock.unlock();
          return false;
        }
        victim.marked = true;
        isMarked = true;
      }
      int highestLocked = -1;
      try {
        Node<T> pred, succ; boolean valid = true;
        for (int level = 0; valid && (level <= topLevel); level++) {
          pred = preds[level];
          pred.lock.lock();
          highestLocked = level;
          valid = !pred.marked && pred.next[level]==victim;
        }
        if (!valid) continue;
        for (int level = topLevel; level >= 0; level--) {
          preds[level].next[level] = victim.next[level];
        }
        victim.lock.unlock();
        return true;
      } finally {
        for (int i = 0; i <= highestLocked; i++) {
          preds[i].unlock();
        }
      }
    } else return false;
  }
}
```

FIGURE 14.7 The LazySkipList class: the remove() method.