```
private void tryToFillQueue(NNResultQueue queue, Deque<Node>
nodeStack, BitSetContext bitSetContext,
                              DisCache disCache) {
       ….//判断结果堆是否已经填满
       Node last = nodeStack.pop();
       Node parent = nodeStack.peek();
       assert !parent.isLeaf();

       if (last.isLeaf()) {//尝试用叶节点的所有数据点填满结果堆

           IntArrayList children = last.getChildren();
           for (int i = 0; i < children.size(); i++) {
               int docId = children.get(i);
               queue.insert(docId, disCache.distance(docId));
               ++queryStat.distanceFunctionLeafInvocations;
           }

           int siblingSize = parent.getCBounds().size() / 2;
           Node[] siblingNodes = parent.getChildrenNodes();
           for (int i = 0; i < siblingSize; i++) {
               if (siblingNodes[i] == last) {
                   bitSetContext.reuse.set(i);
                   break;
               }
           }
            ….//判断结果堆是否已经填满
           for (int i = 0; i < siblingSize; i++) {
               if (siblingNodes[i] == last) {
                   continue;//跳过当前节点，因为之前已经加过
               }
               ….//以兄弟节点填满结果堆
             }
       } else {
           Deque<Node> tmpNodeStack = new LinkedList<>();
           Deque<BitSet> tmpBitSetStack = new LinkedList<>();
           // tmpBitSetStack 作为访问记录，防止重复访问分支
           fillQueueIfNonLeaf(last, queue, tmpNodeStack,
                 disCache, tmpBitSetStack, queryStat);
           //尝试用非叶节点填满结果堆

            while (!tmpNodeStack.isEmpty()) {
                nodeStack.push(tmpNodeStack.removeLast());
            }//如果填满了，要将所经之节点都压入 nodestack

            nodeStack.pop();
            //最后一个叶节点没有 visited，所以要把叶节点去掉，否则比
               bitsetContext 多了一个
               while (!tmpBitSetStack.isEmpty()) {
                       bitSetContext.bitSetStack.
                           push(tmpBitSetStack.pop());
               }
               //保存分支访问记录
           }
       }
   }
```