**3 Coding Interview Secrets that made me a Millionaire**

Stand out from the crowd with these coding interview secrets!

A picture containing table, chessman, indoor

Description automatically generated

How to stand out from the crowd?

Coding interviews are getting tougher every day.

Now that everyone has access to hundreds of coding questions and can spend a lot of time preparing for the coding interviews, it has become quite difficult to stand out from the crowd.

In this post, I’ll share three problem solving concepts that I have used to solve a coding question during one of my FAANG interviews. The interviewer really liked my approach, especially the optimizations I suggested. This got me the FAANG job and**a million dollars in salary within three years!**

I’m sure, these three concepts will help you stand out and ace your coding interview.

Bellow, I’ll share the actual problem, the solution, and all the important details about the interview.

Here is the summary of the three concepts:

➡ Every recursive algorithm can be written **iteratively**using stacks.

➡ It is always a good idea to make your algorithm **thread-safe**. This shows that you have thought about synchronization, multi-threading, and multi-processing.

➡ Make your program **multi-threaded** to execute concurrent tasks. You can also perform pre-fetching or post-processing in separate threads. These techniques are pretty useful, especially when your program is busy doing I/Os.

To begin, let’s take a look at the question statement first:

**Question**

Implement an iterator for the **in-order traversal** of a **binary search tree (BST)**. That is, given a BST, we need to implement two functions:**1. bool hasNext():** Returns true if at least one element is left in the in-order traversal of the BST.  
**2. int next():** Return the next element in the in-order traversal of the BST.

**Example:**Given the following BST:

Diagram

Description automatically generated

Binary Search Tree

Here is the in-order traversal of this tree: [1, 4, 10, 14, 15, 19, 20]

Here is the expected output from the algorithm based on different calls:

hasNext() -> **true**next() -> **1**next() -> **4**hasNext() -> **true**next() -> **10**next() -> **14**next() -> **15**next() -> **19**next() -> **20**hasNext() -> **false**

A little about me: I’m the author of the online courses [**Grokking the Coding Interview**](https://designgurus.org/course/grokking-the-coding-interview) and [**Grokking the System Design Interview**](https://designgurus.org/course/grokking-the-system-design-interview). You can find all my courses at [designgurus.org](http://designgurus.org/).

**The Interview**

Here is the complete rundown of the interview.

**Brute-force Approach**

▹ The interviewer stated the question.

▹ After some initial thoughts, I proposed the brute force solution to store the in-order traversal of the BST in an array and used that array to process the two functions next() and hasNext(). The main idea was to maintain a pointer in the array; this pointer points to the next element to be returned to the caller. This algorithm will take O(N) space for storing all elements in the array, and O(N) time to traverse the tree. Both operations next() and hasNext()will take O(1) time complexity, as we have pre-processed all the data to store the in-order traversal of the BST in the array. ([Recursive in-order traversal of BST](https://www.techiedelight.com/inorder-tree-traversal-iterative-recursive/))

▹ After that, I told the interviewer I would think about a better solution.

**Transforming a Recursive Algorithm to an Iterative one**

▹ I knew we could perform the in-order traversal of a BST recursively. But for the iterator, I can’t use recursion as I can’t stop recursion in the middle to return the required element; this way, I will lose the recursive state of the traversal. Also, I didn’t want to perform the in-order traversal every time next()orhasNext() is called; this will make these functions O(N) operations.

▹ I had practiced some coding questions where I had transformed a recursive solution to make it iterative using a stack. This was exactly what I needed! This way, I can control the recursion by saving the state of the in-order traversal of the BST in the stack, and later I can resume the tree traversal when next() is called again.

▹ I explained this idea to the interviewer and then started writing the solution. Here is the code I wrote for Java (see [Python](https://designgurus.org/path-player?courseid=grokking-the-coding-interview&unit=grokking-the-coding-interview_62ba9052bfabbUnit)solution):



**Time and Space complexities**

▹ The hasNext() will take O(1) time.

▹ Since next()calls the traverseLeft() function which traverses the left subtree and in the worst case the left subtree could contain all elements (practically it would be a list), hence, next() can take O(N) time in the worst case. One thing to note there though, traverseLeft() processes each element only once. This means, amortized cost of traverseLeft() will be O(1) for n calls of next() therefore, next() has O(1) amortized cost.

▹We will need O(N) space for the stack in the worst case (when the BST is a list). Overall, we will need to store nodes equal to the height of the tree  
*O*(H) space complexity, where H is the height of the given tree.

**Making the Algorithm Thread-safe**

▹ Afterwards, the interviewer asked if I could improve the solution.

▹ I spent some time thinking if I could further improve the algorithm’s space or time complexity, but could not find a way.

▹ Then I explained to the interviewer that my algorithm does not have thread-safety. I told that the algorithm would fail if multiple threads access the same BSTIterator object. I further explained that we could get a synchronization issue in the next() function for the following two lines:

TreeNode tmpNode = stack.pop();  
traverseLeft(tmpNode.right);

* If two threads access the function concurrently, the stack containing tree elements could end up in a bad state.
* Here is what can happen. Suppose one thread executes the first line to pop an element from the stack. Before it executes the next line (to traverse the left subtree), another thread which is also processing these lines, can also pop another element from the stack, thus, making the stack invalid.

▹ This means we need to synchronize this function so that only one thread is allowed to process next() at any time.

▹ Here is the new synchronized version of next(). We need to synchronize the whole function (see [Python](https://designgurus.org/path-player?courseid=grokking-the-coding-interview&unit=grokking-the-coding-interview_62ba9052bfabbUnit)solution):

Text

Description automatically generated

**Making the Algorithm Multi-threaded**

▹ We still had some time left, so the interviewer asked me to improve the algorithm further.

▹ I proposed that I can make next() multi-threaded so that I can return the element to the caller immediately and spawn a separate thread to perform the post-processing required to traverse the left subtree.

* In the next()function, I do have the required node available right away, but I could not return to the caller before I traverse the left subtree of the right child of the current node: traverseLeft(tmpNode.right);
* I suggested I can spawn a new thread to process this traversal and return the element to the caller. This way, the caller is unblocked as it has the data quickly, and all the post-processing is done in a separate thread.

▹ This made the algorithm a bit complex. Now, whenever we are starting a new execution of next()or hasNext()I needed to ensure any previous thread doing the post-processing has finished. This means I have to add a check before processing next() or hasNext()to wait and call join() on the previous thread if it has not already finished. Here is the code that takes care of this scenario (see [Python](https://designgurus.org/path-player?courseid=grokking-the-coding-interview&unit=grokking-the-coding-interview_62ba9052bfabbUnit)solution):

Text

Description automatically generated

**Conclusion**

I got the FAANG job which eventually made me a millionaire within three years just because I was able to show something new to the interviewer. With a little bit of practice, anyone can incorporate these techniques to make their coding solution stands out from the crowd.

To summarize, here are the three takeaways:

➡We can easily add **thread safety** to any algorithm.

➡Many algorithms can be made **multi-threaded** to perform multiple tasks concurrently or to do pre-fetching or post-processing functions to speed things up.

➡Remember, **each recursive algorithm can be transformed into an iterative solution** using stacks.

Follow these techniques to distinguish yourself from others! A number of these approaches are discussed in [Grokking the Coding Interview](https://designgurus.org/course/grokking-the-coding-interview).

Follow me for coding and system design interview tips. See all of Design Guru’s coding and system design courses at [**designgurus.org**](http://designgurus.org/).