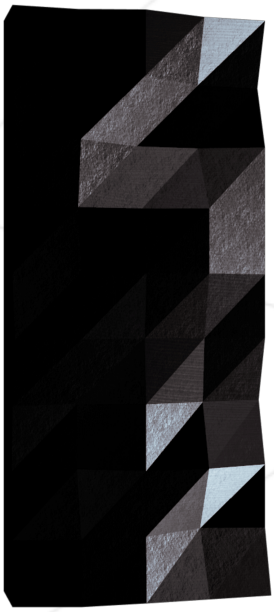

Starfleet

Interview

Staff 42 pedago@42.fr

Summary: This document is an interview question for the Starfleet Piscine.

Contents

Chapter I

General rules

- The interview should last between `45 minutes`.
- Both the interviewer and the interviewed student must be present.
- The interviewed student should write his code using a `whiteboard`, with the language of her/his choice.
- At the end of the interview, the interviewer evaluates the student based on the provided criteria.

Read carefully the interview question and solutions, and make sure you `understand` them before the interview. You can't share this document with other students, as they might be interviewed on the same question. Giving them the answer would prevent them from having to solve an unknown question during an interview.

I.1 During the interview

During the interview, we ask you to :

- Make sure the interviewed student `understands` the question.
- Give her/him any `clarification` on the subject that she/he might need.
- Let her/him come up with a solution before you guide her/him to the best solution given the constraints (time and space).
- Ask the student what is the `complexity` of her/his algorithm ? Can it be improved and how ?
- `Guide` her/him to the best solution without giving the answer. You may refer to the `hints` for that.
- You want to evaluate how the interviewed student thinks, so ask her/him to `explain everything` that she/he thinks or writes (there should be no silences).
- If you see a mistake in the code, wait untill the end and give her/him a chance to correct it by her/himself.
- Ask the student to show how the algorithm works on an `example`.
- Ask the student to explain how `limit cases` are handled.
- Bring out to the student any mistake she/he might have done.
- Give `feedback` on her/his performances after the interview.
- Be `fair` in your evaluation.

As always, stay mannerly, polite, respectful and constructive during the interview. If the interview is carried out smoothly, you will both benefit from it !

Chapter II

Nodes in range

II.1 Interview question

`Given a Binary Search Tree of integers, return the number of nodes having values between two given integers (inclusive).`

Example :

II.2 Acceptable answers (no constraint)

II.2.1 Depth first search wihtout extra information

A first solution is to browse the tree using a `depth first search` (DFS) and count the number of node which value is between the two integers.

```
O(n) time , O(n) space
where n is the number of nodes in the BST
```

code:

```
struct s_node {            // structure for a node of the BST
        int value;              // integer value
        struct s_node *left;    // pointer to the left child
        struct s_node *right;   // pointer to the right child
};

int nodesInRange(struct s_node *bst, int from, int to) {
        int count = 0;

        if (!bst)
                return (0);
        count += nodesInRange(bst->left, from, to);
        count += nodesInRange(bst->right, from, to);
        if (bst->value >= from && bst->value <= to)
                return (count + 1);
        return (count);
}
```

`Note` : This solution could be refine because there is no need to visit the left subtree of a node which value is lower than the lower bound of the range, or the right subtree of a node which value is greater than the upper bound of the range. But the worst case would still be O(n) time.

II.3 Follow up question

You have access to som `extra information` at each node : the number of nodes in the left and right subtress.

II.3.1 Hints

- For each element, the result is the multiplication of the product of all the elements on the left by the product of all the elements on the right.
- Can you do it in 2 steps?

II.4 Best solutions

II.4.1 DFS with extra information

In fact, we only need the number of nodes in the left subtree.

The number of values in the inerval (A, B) is the same as the number of values up to B minus the number of values less than A. So we can reduce this question to finding the number of values up to X.

Starting at the root :

- If the value of the current node is greater or equal to X, then this node and all values to the right are larger or equal to X and we can ignore them. We recurse on the left subtree.
- If the value of the current node is less than X, then this node and all values in the left subtree are less than this value. So we add that number of nodes and we recurse on the right subtree.

On a `balanced` BST, this algorithm takes `O(logn)` time.

```
O(n) time , O(n) space
where n is the number of nodes in the BST
```

code:

```
struct s_node {            // structure for a node of the BST
        int value;             // integer value
        struct s_node *left;   // pointer to the left child
        struct s_node *right;  // pointer to the right child
        int nLeft;             // number of nodes in the left subtree
        int nRight;            // number of nodes in the right subtree
};

int getLess(struct s_node *node, int value) {
        if (!node)
                return (0);
        if (node->value >= value)
                return (getLess(node->left, value));
        return (1 + node->nLeft + getLess(node->right, value));
}

int nodesInRange(struct s_node *bst, int from, int to) {
        return (getLess(bst, to + 1) - getLess(bst, from)); // to+1 to get #values <= to
}
```