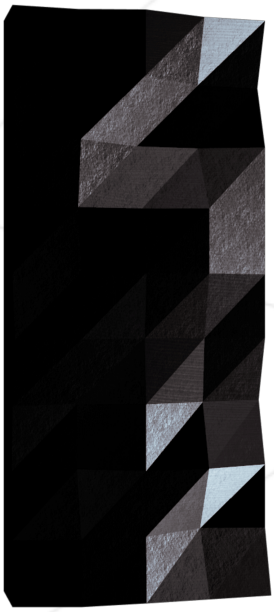

Starfleet

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

BST from linked list

II.1 Interview question

You are given a sorted doubly linked list. Nodes of this doubly linked list have a left as previous and a right as next pointer and a value. Convert this to a binary search tree with minimum height using the same nodes.

```
struct s_node {
        int           value;
        struct s_node *right; // next pointer in the doubly linked list
        struct s_node *left;  // prev pointer in the doubly linked list
};
```

II.1.1 Hints

- Walk through an exemple : 1 <=> 2 <=> 3 <=> 4 <=> 5 <=> 6 <=> 7
- Which node would be the root of the BST ?

II.2 Best solution

II.2.1 Middle node

EXAMPLE :

For following list : 1 <=> 2 <=> 3 <=> 4 <=> 5 <=> 6 <=> 7

the bst would be :

The solution is to search for the `middle node` of the doubly linked list and make that middle node as the root of the BST.

To find the middle node of the linked list, we use the `runner technique`. We go through the list with 2 pointers : a normal and a runner. The runner goes twice as fast as the normal. When the runner has reached the end, the normal points to the middle node.

Then we just recursively construct its left and right `subtrees`.

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

code:

```
struct s_node {
        int           value;
        struct s_node *right;
        struct s_node *left;
};

struct s_node *findMiddle(struct s_node *head) {
        struct s_node *runner;

        if (head->right == NULL)
                return (head);
        runner = head;
        while (runner) {
                runner = runner->right;
                if (!runner)
                        break ;
                runner = runner->right;
                head = head->right;
        }
        return (head);
}

struct s_node *toBst(struct s_node *head) {
        struct s_node *node;

        if (head == NULL)
                return (NULL);
        if (head->right == NULL)
                return (head);
        node = findMiddle(head);
        if (node->left)
                node->left->right = NULL;
        node->left = toBst(head);
        if (node->right)
                node->right->left = NULL;
        node->right = toBst(node->right);
        return (node);
}
```