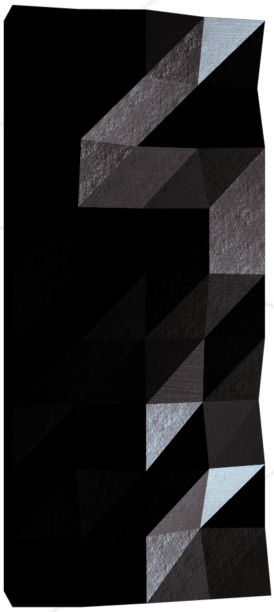

Starfleet

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

Print binary tree

II.1 Interview question

`Write an algorithm to print a binary tree level wise, from leaves to root.`

`For this purpose, you can only use one queue and one stack.`

EXAMPLE :

For this tree, the output would be : 4 5 6 7 2 3 1

II.1.1 Hints

- Which traversal : Deepth First Search (DFS) or Breadth First Search (BFS) ?
- You already need the queue for the BFS. How could you use the stack ?

II.2 Best solution

II.2.1 Using one queue and one stack

We clearly have to use a Bread First Search (BFS) algorithm in order to visit the nodes level wise.

We all know that a traditional BFS traversal would output : 1 2 3 4 5 6 7

We could use a stack to reverse it, but we would still have a problem : 7 6 5 4 3 2 1

It is level wise but not left to right.

This problem can be solve by enqueuing right child prior to left child : 1 3 2 7 6 5 4

Now, printing in reverse order can easily be done by putting all the nodes in a stack and print them when the complete tree is scanned.

```
O(n) time , O(n) space
```

code:

```
struct s_queue *initQueue(void);

void enqueue(struct s_queue *queue, void *content);

void *dequeue(struct s_queue *queue);

int isEmptyQueue(struct s_queue *queue);
```

```
struct s_stack *initStack(void);

void *pop(struct s_stack *stack);

void push(struct s_stack *stack, void *content) ;

int isEmptyStack(struct s_stack *stack);
```

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

void print_tree(struct s_node *root) {
        struct s_queue *queue;
        struct s_stack *stack;
        struct s_node *node;

        queue = initQueue();
        stack = initStack();
        enqueue(queue, root);
        while (!isEmptyQueue(queue)) {
                node = dequeue(queue);
                if (node->right)
                        enqueue(queue, node->right);
                if (node->left)
                        enqueue(queue, node->left);
                push(stack, node);
        }

        while (!isEmptyStack(stack)) {
                node = pop(stack);
                printf("%d", node->value);
                if (!isEmptyStack(stack))
                        printf(" ");
        }
        printf("\n");
}
```