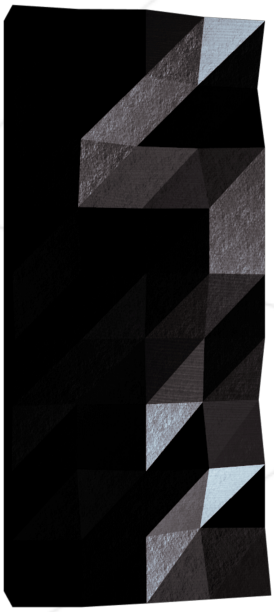

Starfleet

Interview

Staff 42 pedago@42.fr

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

Contents

I General rules **2**
I.1 During the interview 3

II Stack with min **4**
II.1 Interview question 4
II.1.1 Hints 4
II.2 Acceptable answers (no constraint) 4
II.2.1 Store min in a value 4
II.2.2 Store min in each node 5
II.3 Best solution 6
II.3.1 Store min in a second stack 6

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

Stack with min

II.1 Interview question

`How would you design a stack which, in addition to push and pop, has a function min which returns the minimum element? Push, pop and min must operate in O(1) time.`

II.1.1 Hints

- Observe that the minimum element doesn't change very often. It only changes when a smaller element is added, or when the smallest element is popped.
- What if we kept track of extra data at each stack node? What sort of data might make it easier to solve the problem?
- Consider having each node know the minimum of its "substack" (all the elements beneath it, including itself).

II.2 Acceptable answers (no constraint)

II.2.1 Store min in a value

The minimum does not change very often. It only changes when :

- a smaller element is pushed to the stack
- the minimum value is popped from the stack

When an element is pushed to the stack, we can update this minimum value in O(1) time. But when the minimum value is popped from the stack, we have to search through the stack to find the new minimum. Unfortunately, this would `break` the constraint that pop operates in O(1) time.

II.2.2 Store min in each node

EXAMPLE :

```
push(5): stack = {5}, min = 5

push(6): stack = {6, 5}, min = 5

push(3): stack = {3, 6, 5}, min = 3

push(7): stack = {7, 3, 6, 5}, min = 3

pop(): stack = {3, 6, 5,}, min = 3

pop(): stack = {6, 5}, min = 5
```

=> When the stack goes back to a prior state (pop), the minimum also goes back to a prior state.

If we kept track of the minimum at eah state, we would be able to easily know the minimum. We can do this by having each node record what the minimum beneath itself is. Then, to find the min, you just look at what the top element thinks is the min.

```
O(1) time, O(n) space, where n is the number of elements in the stack
```

Code:

```
struct s_node {
        int             value;
        int             min;
        struct s_node *next;
};

struct s_stack {
        struct s_node *top;
};

int pop(struct s_stack *stack);

void push(struct s_stack *stack, int value, int minimun);

int min(struct s_stack *stack) {
        if (stack->top == NULL)
                return (INT_MIN); // error
        return (stack->top->min);
}

void pushWithMin(struct s_stack *stack, int value) {
        int minimum;

        minimum = (value < min(stack)) ? value : min(stack);
        push(stack, value, minimum);
}
```

II.3 Best solution

II.3.1 Store min in a second stack

The problem with the previous solution is that we waste a lot of space by keeping track of the min for every single element.

We can do a bit better than this by using an `additional stack` which keeps track of the mins.

Why this method might be more space efficient? Suppose we had a very large stack and the first element inserted happened to be the minimum. In the first solution, we would be keeping n integers, where n is the size of the stack. In the second solution though, we store just a few pieces of data: a second stack with one element and the members within this stack.

```
O(1) time, O(n) space, where n is the number of elements in the stack
```

code:

```
struct s_node {
        int           value;
        struct s_node *next;
};

struct s_stack {
        struct s_node *top;
};

int pop(struct s_stack *stack);

void push(struct s_stack *stack, int value);

struct s_stackWithMin {
        struct s_stack *value;
        struct s_stack *min;
}

int min(struct s_stackWithMin *stack) {
        if (stack->min->top == NULL)
                return (INT_MIN); // error
        return (stack->min->top->value);
}

void pushWithMin(struct s_stackWithMin *stack, int value) {
        if (value <= min(stack))
                push(stack->min, value);
        push(stack->value, value);
}

int popWithMin(struct s_stackWithMin *stack) {
        int value;

        value = pop(stack->value);
        if (value == min(stack))
                pop(stack->min);
        return (value);
}
```