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

Tree height

II.1 Interview question

You have an array of n integers. This array describes a tree, `not necessarily a binary tree`, with n nodes. These nodes have ids from 0 to n-1. The array contains, at index i, the id of the parent of the node with id i.

A value of -1 at the index i denotes that node with id i is the root.

Example :

```
For the following tree :

          3
        / | \
       1  2  4
              \
               0

The array would be : {4, 3, 3, -1, 3}
```

`Given such an array, find the height of the tree.`

II.2 Acceptable answers (no constraint)

II.2.1 Brute force

A brute force solution is to find all the children of a node, starting from the root and recursively going downto all the leafs.

Then, return the depth of the farthest leaf.

```
O(n^2) time , O(n) space
where n is the number of elements in the array
```

code:

```
int heightParent(int *parentId, int n, int id) {
        int maxHeight = -1;
        int height;

        for (int i = 0; i < n; i++) {
                if (parentId[i] == id) {
                        height = 1 + heightParent(parentId, n, i);
                        if (height > maxHeight)
                                maxHeight = height;
                }
        }
        return (maxHeight);
}

int heightTree(int *parentId, int n) {
        return (heightParent(parentId, n, -1));
}
```

II.3 Follow up question

Improve the runtime using `dynamic programming`.

II.3.1 Hints

- Why not `cache` the results and use them later?
- Maybe with an hash table.

II.4 Best solution

II.4.1 Bottom-up dynamic programming

For each node, compute the the depth of its ancestors until the depth of one its ancestor has already been computed. For each node the depth is stored in a hash table.

Return the depth of the farthest node.

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

code:

```
int findDepth(int *parentId, int *depth, int i) {

        // If the depth of node i has already been computed, just return the depth
        if (depth[i] != -1)
                return (depth[i]);

        // Otherwise, it is equal to 1 + depth of parent node
        if (parentId[i] == -1)
                depth[i] = 0;
        else
                depth[i] = 1 + findDepth(parentId, depth, parentId[i]);

        // return the depth of node i
        return (depth[i]);
}

int heightTree(int *parentId, int n) {
        int depth[n];           // hashtable for depth memoization
        int maxHeight = -1;
        int height;

        // initialize hashtable to -1
        for (int i = 0; i < n; i++)
                depth[i] = -1;

        // for all node compute the depth
        for (int i = 0; i < n; i++) {
                if (depth[i] == -1) {
                        height = findDepth(parentId, depth, i);
                        if (maxHeight < height)
                                maxHeight = height;
                }
        }

        // return the depth of the farthest node
        return (maxHeight);
}
```