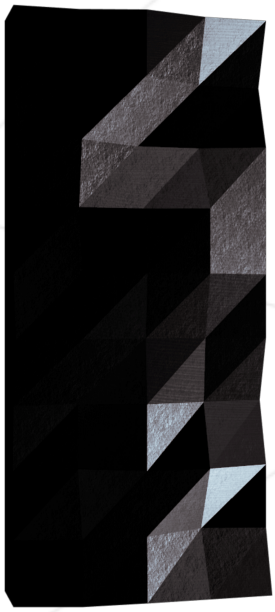

Interview Piscine

Rush00 Interview Question

maya maya@42.us.org
evan evan@42.us.org

Summary: This document describes a rush question for the Interview Piscine

Contents

Chapter I

Interview Rules

I.1 General Rules

- The interviewer and interviewee are equally responsible for scheduling the interview.
- The interviewer must read and understand the question and its solutions before the interview begins.
- The interview should last `45 minutes`.
- Both the interviewer and the interviewee must be present.
- The interviewee should write their code on a whiteboard in the language of their choice.
- The interviewee may not use any reference materials or programs to help them write their solution.
- Do not share this document or discuss this problem with other students outside of the context of an interview.

I.2 During the Interview

During the interview, we ask you to:

- Make sure the interviewee understands the question.
- Give them any clarification on the subject that they might need.
- Let them come up with a solution before you guide them to the best solution given the time and space constraints.
- Ask the interviewee what the complexity of their algorithm is. Can it be improved, and how?
- Guide them to the best solution without giving the answer. You may refer to the hints for that.
- You want to evaluate how the interviewee thinks, so ask them to explain everything that they think or write (there should be no silences).
- If you see a mistake in the code, wait until the end and give the interviewee a chance to correct it by themselves.
- Ask the interviewee to show how the algorithm works on an example.
- Ask the interviewee to explain how edgecases are handled.
- Point out to the interviewee any mistakes they may have made.
- Give feedback on their performance 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

Diametric Opposite

II.1 Interview Question

Given the head of a circular doubly linked list of integers, print the integer stored in the node diametrically opposite from the list's head. A node's opposite is the node farthest from it in both directions.

```
Input:
   Head--> 1 -- 2 -- 3
           |         |
           6 -- 5 -- 4
Output:
   4
```

II.1.1 Questions the Interviewee Should Ask

- Will the integers be consecutive and/or sorted?
  Not necessarily.
- In lists with an odd number of nodes, each node has two opposites. How should this be handled?
  Return the opposite closer to the head on its right, farther from the head on its left.
- How should lists with one node be handled?
  Return the head.

II.2 Solutions

II.2.1 Naive solution

Traverse the whole linked list and count the number of nodes encountered before reaching the head. Now traverse the list again until count/2 and return the node at count/2.

Time complexity: `O(n)`
Space complexity: `O(1)`

II.2.2 Somewhat optimized: traverse only once

Traverse linked list using two pointers travelling in the same direction. Move one pointer by one node and other pointer by two nodes each loop. When the fast pointer reaches the head, the slow pointer will be at the diametric opposite of the list's head.

Time complexity: `O(n)`
Space complexity: `O(1)`

```
void   printOpposite(struct Node *head)
{
    struct Node *slow_ptr = head->right;
    struct Node *fast_ptr = head->right->right;
    while (fast_ptr != head && fast_ptr->right != head)
    {
        fast_ptr = fast_ptr->right->right;
        slow_ptr = slow_ptr->right;
    }
    printf("Opposite: %d\n", slow_ptr->value);
}
```

Hints

- Have you considered using multiple pointers?

II.2.3 Optimized: traverse only once; no more than n + 1 steps total!

Traverse linked list using two pointers travelling in opposite directions. Move one pointer, check if the pointers are equal, then move the other pointer before checking again. When the pointers point to the same node, that node is the diametric opposite of the list's head.

Time complexity: `O(n)`
Space complexity: `O(1)`

```
void   printOpposite(struct Node *head)
{
    struct Node *right_ptr = head;
    struct Node *left_ptr = head->left;
    while (right_ptr != left_ptr)
    {
        right_ptr = right_ptr->right;
        if (right_ptr == left_ptr)
```

```
            break;
        left_ptr = left_ptr->left;
    }
    printf("Opposite: %d\n", right_ptr->value);
}
```

Hints

- Have you considered using multiple pointers?
- Can you use the list's circularity to your advantage?