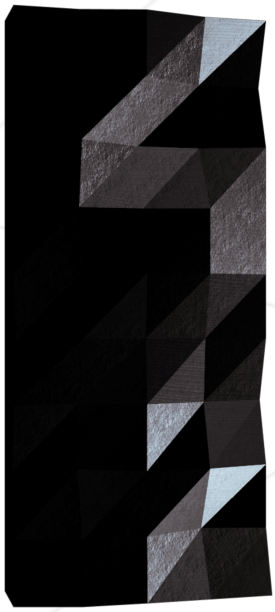

Interview Piscine

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

The Bored Banker

II.1 Interview Question

`You have a list of n bank accounts enumerated from 0 through n - 1, and a list of m transactions. A transaction has two components: an amount (d), and a number of accounts (k). When a transaction is processed, the amount d is added to the balances of the first k accounts. If each account starts with a balance of $0, find the maximum balance in any account after processing all transactions.`

II.1.1 Questions the Interviewee Should Ask

- Can a transaction amount be negative?
  Yes.
- Will transaction amounts always be in whole dollars?
  Sure.
- Can we assume that k <= n for every transaction?
  Go for it!
- Can we assume that there will always be at least one bank account and at least one transaction?
  If you want to!
- Can I have an example?
  Try to make one yourself, and I'll help you if you need me to.

II.2 Solutions

II.2.1 Naive solution

For each transaction, add the specified amount to the specified range of accounts. Find the maximum value in the list of accounts.

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

```
int bored_banker(int *accounts, int n, t_transaction *transactions, int m)
{
    for (int t = 0; t < m; t++) {
        for (int a = 0; a < transactions[t].n_accounts; a++) {
            accounts[a] += transactions[t].amount;
        }
    }

    int max = accounts[0];
    for (int i = 1; i < n; i++) {
        if (accounts[i] > max)
            max = accounts[i];
    }
    return (max);
}
```

Hints

- Do you need an example?

```
For 5 accounts with the following 3 transactions:

    t0: amount = 10, number of accounts = 4
    t1: amount = -7, number of accounts = 2
    t2: amount = 3, number of accounts = 3

Balances after processing all transactions: {6, 6, 13, 10, 0}
Maximum: 13
```

II.2.2 Optimized solution

For each transaction, add the amount to the first account and subtract the amount from the account after the last one in the specified range. Calculate the balance of each account by adding the balance of the previous account to the balance delta stored in the current account, tracking the maximum balance as you go. Return the maximum balance.

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

```
int bored_banker(int *accounts, int n, t_transaction *transactions, int m)
{
    for (int t = 0; t < m; t++) {
        accounts[0] += transactions[t].amount;
        if (transactions[t].n_accounts < n)
            accounts[transactions[t].n_accounts] -= transactions[t].amount;
    }

    int max = accounts[0];
    int local_sum = accounts[0];
```

```
    for (int i = 1; i < n; i++) {
        local_sum += accounts[i];
        if (local_sum > max)
            max = local_sum;
    }
    return (max);
}
```

Example:

```
For 5 accounts with the following 3 transactions:

    t0: amount = 10, number of accounts = 4
    t1: amount = -7, number of accounts = 2
    t2: amount = 3, number of accounts = 3

After processing t0: {10, 0, 0, 0, -10}
After processing t1: {3, 0, 7, 0, -10}
After processing t2: {6, 0, 7, -3, -10}

Final balances: {6, 6, 13, 10, 0}
Maximum: 13
```

Hints

- Instead of thinking about the balance of each account individually, have you thought about how the balance changes from the first account to the second, the second to the third, etc.?