Advent of Code 2025 Day 03

Deval Deliwala

Part 1

Given a sequence of lines called *banks*:

```
987654321111111
811111111111119
234234234234278
818181911112111
```

find the largest possible two-digit number that exists within each bank. Return the sum of all two-digit numbers found from every bank without rearranging order. For example:

In **98** 7654321111111, the largest number is **98**. In 818181 **9** 1111 **2** 111, the largest number is **92**. Across all four banks above, the sum is `357`.

Naive Implementation

The challenge is to find the two ordered digits that form the largest number. Greedily, maximizing the tens place is a good start.

Given a bank `"818181911112111"`, we want the largest number – `9` – to be on the left. Hence we place `9` in the tens place. Then we find the largest number *after* `9`, which is `2`, and place it in the ones place, producing `92`. However, if the largest number *is the last digit*, then it must be in the ones place. Then, we choose the largest number that precedes it for the tens place.

Iterating the above procedure over all banks and accumulating every max number yields the result.

```rust
use std::fs::File;
use std::io::{BufReader, BufRead, Error, ErrorKind};
use std::time::Instant;
use crate::_2025::DAY3_FILE;


/// Finds the maximum number in a given [String]
/// containing digits 1-9.
fn find_max_in_bank(bank: &str) -> Result<(usize, u32), Error> {
    let mut max = 0;
    let mut max_idx = 0;

    // find largest number and its position in a bank
    for (i, c) in bank.chars().enumerate() {
        let c_num = c.to_digit(10)
            .ok_or_else(
                || Error::new(
                    ErrorKind::InvalidInput,
                    "Expected an integer between 1-9"
                )
            )?;

        if c_num > max {
            max = c_num;
            max_idx = i;
        }
    }

    Ok((max_idx, max))
}

fn run_pt1() -> Result<u32, Error> {
    let mut total_max = 0;
    let mut buff_reader = BufReader::new(File::open(DAY3_FILE)?);
    let mut string_buff = String::new();

    // iterate over all banks
    while buff_reader.read_line(&mut string_buff)? != 0 {
        let buffer = string_buff.trim();
        let (max_idx, max_val) = find_max_in_bank(buffer)?;

        // largest digit is the last element
        if max_idx == buffer.len() - 1 {

            // find max in prefix
            let (_, max_left) = find_max_in_bank(&buffer[..max_idx])?;
            total_max += max_left * 10 + max_val;

        } else {

            // find max in suffix
            let (_, max_right) = find_max_in_bank(&buffer[(max_idx + 1)..])?;
            total_max += max_val * 10 + max_right;
        }

        // clear buffer
        string_buff.clear();
    }

    Ok(total_max)
}
```

This implementation takes roughly **2.4 ms** to run. There are many places for optimization. First of all, this requires two unequal-size passes:

1. One pass to find the largest digit.
2. Another pass scanning its suffix or prefix to find the second largest digit.

This problem is screaming "*able to be done in one pass.*" And frankly I was stupid to not figure it out from the beginning.

Optimized Implementation

We'll solve this in one-pass through the bank. We know the tens-digit must come before the ones-digit. And we want to maximize the two-digit number they form:

$$\max_{i<j}\left(10d_i + d_j\right) \tag{1}$$

The constraint $i < j$ suggests we iterate backwards over the bank. Why? Given a tens-digit, we want to find the largest digit that comes after for the ones-digit to satisfy Equation 1. Therefore, if we start from the left, then for every candidate tens-digit, we *have* to pass through the *entire bank* to find the largest digit that comes after. This is *already a pass* over the entire bank, for a single tens-candidate.

But, consider if we iterate backwards, starting from the last digit in the bank. Then we *already know* the digits that come after any digit *as we do the pass* over the bank.

```rust
fn run_pt1_2() -> Result<u32, Error> {
    let mut total_max = 0;
    let mut buff_reader = BufReader::new(File::open(DAY3_FILE)?);
    let mut string_buff = String::new();

    while buff_reader.read_line(&mut string_buff)? != 0 {
        let buffer = string_buff.trim();
        let bytes  = buffer.as_bytes();

        // keeps track of largest two-digit number found
        let mut best_number = 0;

        // keeps track of largest ones-digit found
        // starts all the way on the right. ensures i < j.
        let mut best_ones = (bytes[bytes.len() - 1] - b'0') as u32;

        // iterate backwards
        for &tens in bytes[..bytes.len() - 1].iter().rev() {
            let tens = (tens - b'0') as u32;

            let candidate_number = tens * 10 + best_ones;
            if candidate_number > best_number {
                best_number = candidate_number;
            }

            if tens > best_ones {
                best_ones = tens;
            }
        }

        // accumulate
        total_max += best_number;
        string_buff.clear();
    }

    Ok(total_max)
}
```

This runs in roughly **590 µs**, which is 4× faster than the naive implementation.