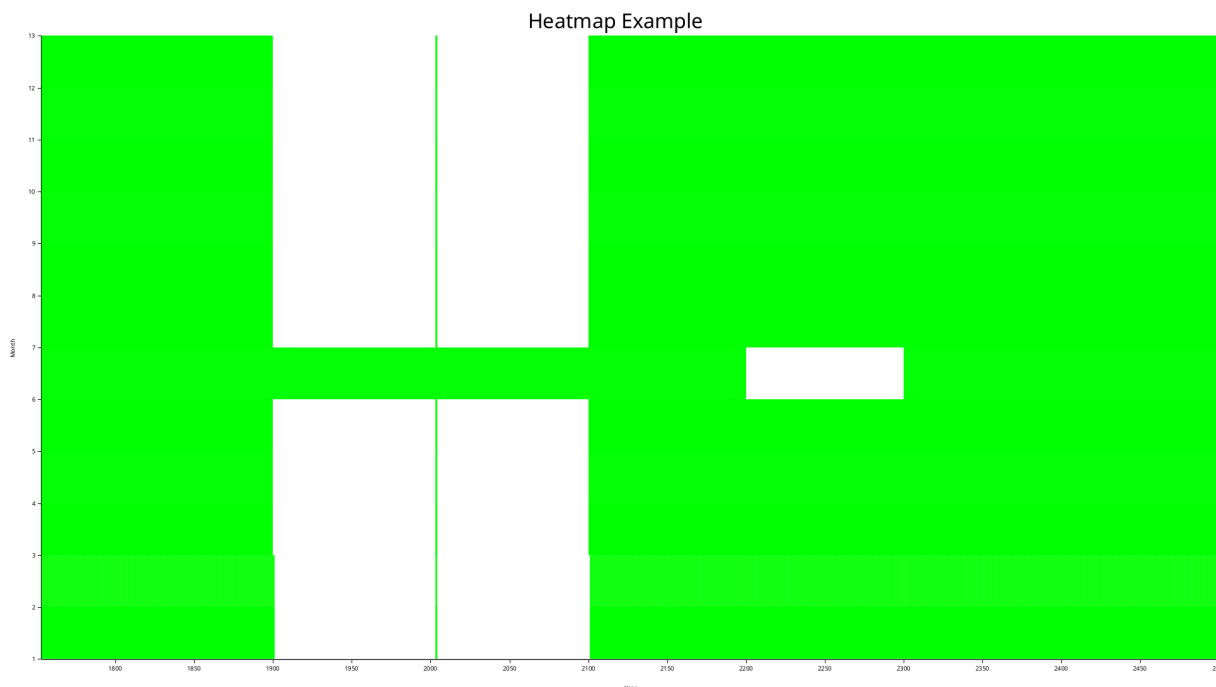


高可信软件技术 课后作业

叶剑豪

代码仓库:<https://github.com/Clo91eaf/findDayTest>



一 软件要求分析



1. 计算某一天为星期几, 日期范围从 1752 年 9 月 14 日起到 2500 年 12 月 31 日。(当输入日期超出范围时通知用户)
2. 向用户提供输入格式的通知信息。
3. 输入无效时提醒用户。
4. 当用户提供无效输入时, 程序不应崩溃、冻结或重复。

二 等价类划分



1. 日期范围内的日期

- 9 14 1752 ✗
 - 输出: Year out of range. The year range is from 1753 to 2500. 说明它只支持 1753 年以后的日期, 与软件要求不符.


高可信软件技术 课后作业

- 1 1 1753 
 - 输出:The Day is Saturday. 正常输出星期几, 正确性见下一章.
- 12 31 2500 
 - 输出:The Day is Thursday. 正常输出星期几, 正确性见下一章.




2. 日期范围外的日期

- 9 13 1752 
 - 输出:Year out of range. The year range is from 1753 to 2500. 说明它只支持 1753 年以后的日期.
- 1 1 2501 
 - 输出:Year out of range. The year range is from 1753 to 2500. 说明它只支持 2500 年以前的日期.

3. 不符合日期格式的字符串

- a 
 - 输出: 无限循环字符串, 说明它无法处理不符合日期格式的字符串.

```
year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
Enter the DATE MONTH and YEAR: Year out of range.
The year range is from 1753 to 2500.
...
```

- 1 
 - 等待用户继续输入, 此处应当提示用户输入格式错误并要求重新输入
- 闰年相关
- 29 2 2025 
 - 输出:Invalid date in February of non leap year. The maximum date value is 28!. 说明它可以正确识别闰年
- 29 2 2024 
 - 输出:The Day is Thursday. 正常输出星期几, 正确性见下一章.

- 29 2 2100 ❌
 - 输出:The Day is Monday. 输出错误, 2100 年不是闰年.
- 29 2 2000 ✅
 - 输出:The Day is Tuesday. 正常输出星期几, 正确性见下一章.

程序测试结果: 日期范围同描述不符. 无法处理不符合日期格式的字符串. 闰年判断不正确.

三 差分验证

对于程序日期范围内的日期(1 1 1753, 12 31 2500)的输出进行差分验证. 使用 Rust 语言的 chrono 库进行验证.

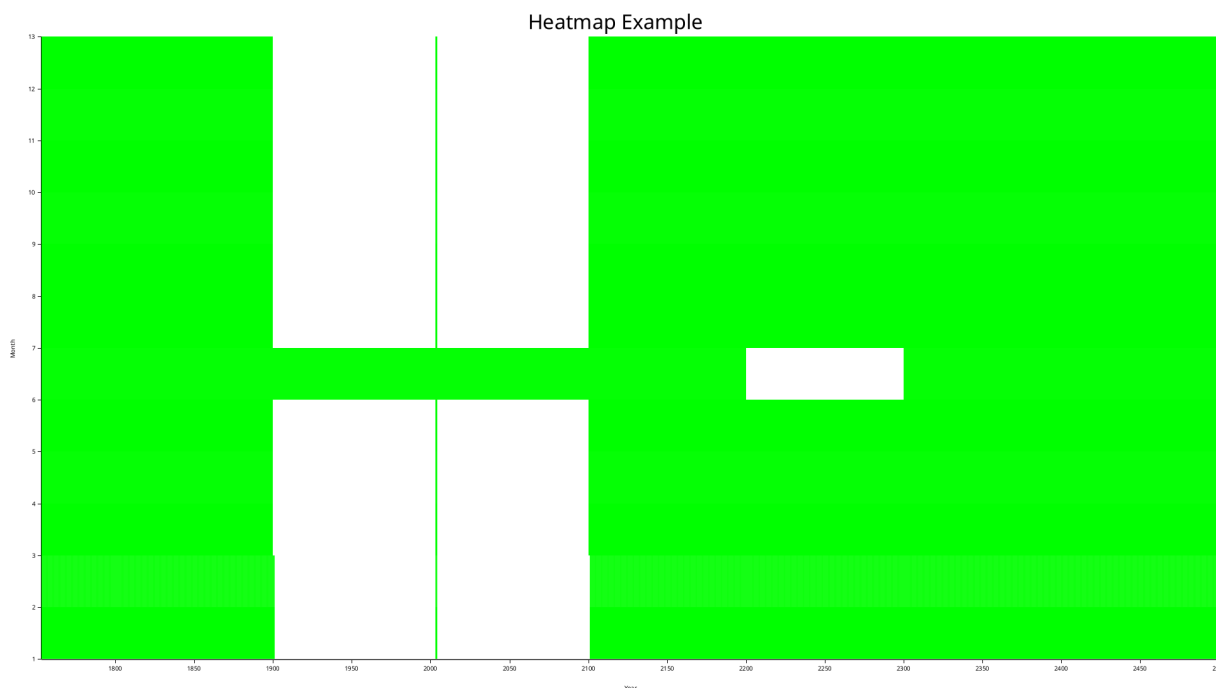
chrono 是 Rust 语言中一个强大的日期和时间库, 提供了丰富的功能来处理日期和时间。以下是 chrono 库的一些主要功能和用法:

主要功能

- 日期和时间的创建和操作: 可以创建日期和时间对象, 并对它们进行各种操作, 如加减时间、比较等。
- 时区支持: 支持本地时间和 UTC 时间的转换。
- 格式化和解析: 可以将日期和时间格式化为字符串, 或从字符串解析日期和时间。
- 时间间隔: 可以计算两个日期或时间之间的时间间隔

观察到程序的输入时间区间为 1753 年到 2500 年, 天数大致为 273200 天, 在可以遍历的范围内. 通过 shell 脚本运行一段时间的测试程序, 发现其行为中没有 sleep(1)之类的延时操作, 效率可行. 编写代码见代码仓库: ./src/main.rs

将所有的错误输出保存到文件 ./error_log.txt 中, 错误格式为 day month year target_output reference_output 注意到产生了 203484 条错误输出(2.8M), 将所有的错误文件 parse 整理成 heatmap 如下:



可以发现 1753-1900, 2003, 2100-2500 以及每年的六月份(除了 2200-2300)的结果都是错误的, 这大概率跟闰年的处理有关. 六月份的错误应当是程序编写过程中故意在六月增加了偏移量.

四 自差分验证(Self-differential Verification)

在上一章差分验证中, 依赖了外部库 `chrono` 进行验证. `chrono` 库作为开源实现的日期时间库, 有着较高的可靠性. 但是在实际应用中, 有时候我们需要对程序进行自我验证, 以保证程序的正确性. 本节将介绍如何使用自差分验证的方法对程序进行验证.

在本测试程序中, 通过对每一天进行遍历, 我们需要保证的性质是: 后一天的输出应当是前一天的输出加一.

因此编写对应代码进行自差分验证:

```
fn generate_self_diff_log() {
    let start = NaiveDate::from_ymd_opt(1753, 1, 1).unwrap();
    let end = NaiveDate::from_ymd_opt(2500, 12, 31).unwrap();

    let mut current_date = start;

    let mut file = File::create("self_diff_log.txt").expect("Unable to create file");

    let mut yesterday = 0;
    while current_date <= end {
        let today = test_find_day(
            current_date.day(),
            current_date.month(),
            current_date.year(),
        );
        if yesterday != 0 && today != yesterday % 7 + 1 {
```

```
println!(
    "{} {} {} yesterday = {yesterday} today = {today}, day mismatch",
    current_date.day(),
    current_date.month(),
    current_date.year()
);
writeln!(
    file,
    "{} {} {} yesterday = {yesterday} today = {today}, day mismatch",
    current_date.day(),
    current_date.month(),
    current_date.year()
)
.expect("Unable to write data");
}
current_date = current_date + Duration::days(1);
yesterday = today;
}
}
```

具体的 log 见代码仓库: ./self_diff_log.txt, 由于差分验证已经有比较完整的图表, 因此这一测试方法不做对应结果的 parse.

尽管这种方法尽量减少了对外部库的依赖, 但是在日期的实际更新中, 依然依赖了 chrono 库, 但是已经将依赖外部库的“每一天”的数据逻辑减少成为依赖外部库“日期更新”的逻辑, 从这个角度上来说, 验证的可靠性是增强的.

五 总结

通过等价类划分, 差分验证和自差分验证, 我们发现了程序的一些问题:

1. 日期范围不符合要求
2. 无法处理不符合日期格式的字符串
3. 闰年判断不正确
4. 日期输出错误

本次作业中使用了以上三种方法对目标测试程序进行验证, 其实还考虑过组合测试以及 fuzzing 等方法, 但是这些方法都比较 trivial, 而且目标程序的错误空间较大, 在等价类划分中就可一窥一二, 因此没有采用.

自差分验证的实现是我在本次作业中拍脑袋想出来的, 在网上也较少看到相关的资料, 思路也比较 trivial, 所以应该只是有对应的名词我没有找到, 但我对这种方法和实现都比较满意.