CS61b Learning

Background 背景紹介

cs61b相比于同系列cs61a而言：

- 鲜明的**面向对象设计** (OOP)，弱化的函数编程思想（cs61a中的scheme/函数传参/高级函数）
- 侧重于**结构搭建**
- 代码量变大，**千行数量级**
- 更要考虑实际运行的**时空复杂度**
- Lab为写**Project**服务
- Project**自由度高**，不再是problem导向
- 算法难度提升，涉及**优化搜索，nlogn排序，哈希，并查集，平衡树，图的最短路径，最小生成树**等

Personal Thoughts 個人的な考え

迫于现实压力，Tab抱着**速通**的想法学这个课，最后大概花了10天左右的时间，刷完了全部的Lab和Project。 但客观而言，这绝对是一门值得花很长时间去投入学习的课程。 或者不用「课程」这个词，用「知识」来称呼更加妥当。 如果只是想着写完四个project长一长项目经历的话，可能会和大量的有趣或有用的知识擦肩而过。 一个小小的建议是，*https://sp21.datastructur.es/* 上，打算做lab和project的时候，不妨看一看Lecture在讲些什么。

Tab觉得这门课的内容难点在：

- **信息密度大的硬课** 对零基础上手不友好
- **大量英语文本阅读** 最好有外语基础，LLM只能解决一部分问题

关于辅助工具的使用：按照工科学生的学习思维，学习以目的为导向，学习手段可以很自由，谷歌维基，菜鸟教程，生成式AI都可以用。 我在做Gitlet的时候，借助哈Gemi老师一点一点，几乎从零开始摸清楚了真实Git的工作原理，然后复现在了Gitlet上面。 **但是**：所有**关键部分的代码落实**都不推荐使用任何外部工具，IDE的AI辅助补全整段也要关掉。

Projects Overview プロジェクト概要

- **Proj0 2048** 这个主要是熟悉java语法用的
- **Proj1 Data Structures** 深入接触数据结构，了解Java里面类方法接口等等的调用，要求复现java.utiI里面的LinkedListDeque 和 ArrayDeque
- **Proj2 Gitlet** 量大而复杂，用到很多种数据结构和文件管理系统，~~Debug最坐牢~~，但也是最值得一做的Project，对写代码启发特别大
- **Proj3 CS61BYoW** 做一个2D肉鸽，更加偏向创作性，可能更适合vibe coding（笑）我做的这个做的很粗糙说是

About Gitlet　ギットレットについて

和上面所说的一样，这个项目**虽然非常坐牢**，但是**非常值得一做**。我耗时3天一共32小时，在LLM辅助大大降低试错成本的前提下做这个时长，很难想象没有LLM的时代的那些学生做这个Project是多么难受的体验。 鸣谢哈

Gemi老师帮我做了这些事情：

- 帮我弄懂Git的真实工作原理，在自己的Gitlet里面借鉴取舍和复现
- 帮我判断把握一些想法上的可行性的是与否，大大降低了钻进死胡同的试错成本
- 帮我总结概要翻译文本难读的部分，解释清楚工作规则的细节
- 帮我读那个.in文件的又长又难懂的报错信息

回忆 一开始写init的时候，借鉴.git的文件设置 参见记录：https://shuiyuan.sjtu.edu.cn/t/topic/447931/807

> proj2正在努力的找突破口……
>
> 打算先这么入手：
>
> - 模仿（mimic）.git的文件结构
> - 弄清楚commit tree & HEAD指针的具体工作原理
> - 以目前的认识来说都有点抽象
>
> 思路：尝试做init，init需要构造一个repo，那么需要
>
> ```
> .gitlet/
> ----objects/ # 存序列化的文件，应该是staging area
> ----ref/
> --------heads/ # 存各种的分支
> ------------master # master分支的commit
> --------remotes/ # 存远程分支（后期用到）
> ```
>
> 这个文件结构是好构建的，但是这里的commit内容究竟
>
> - 要包含什么信息，用什么样的结构
> - 如何从commit里面读/存一整个commit tree
>
> 那么搭框架这一步做了个大概，测试也算是跑通了，后面一些文件等做log或者别的方法了再加
>
> 明天打算先搞清楚
>
> - 这些数据是怎么在各个文件当中流动的
> - 怎么创建commit树，并通过checkout等方法回溯或者分支
>
> 虽然init对此的要求不高，但是必须要先搞清楚

这个结构从复盘的角度来看做的其实挺粗糙的，当时我甚至连**Staging Area**是什么东西都不知道 对所谓的commit tree也一点概念都没有，后来写log的时候还以为要单独存一个二进制文件，~~脑子多少沾点大病~~ 但是我

弄清楚了index(Staging Area) 的存储是一张哈希表，Objects里是各种commit文件和commit混放的 然后后来一整天我煞费苦心写了 init add commit rm log global-log status find 这8个方法

参见记录：https://shuiyuan.sjtu.edu.cn/t/topic/447931/831

> LLM说大概熟练者是20-40h之间完工
>
> 明天还得干一个很牛逼的事情
>
> # 优化代码
>
> 这坨代码里估计还有大的没发现

这个大的被我说中了一半，当时还以为很顺水推舟的做完了剩下的checkout branch rm-branch reset merge，本地他给的4个小测试样例也都过了 然后一交上gradescope，好了，轮到我自闭了 没有几个是绿的 后面大半天时间，从那天下午4点到晚上2点，再后一天中午，**全在debug这堆东西** 依次解决了这些问题：

- add的时候应该检测文件内容有没有改动，如果没改动就不要自作多情
- remove的时候stage了removed file是回收站，它还没被完全送走，我把它add回去了就应该是一切正常了
- error信息应该用`System.out.println()`写，加上`System.exit(0)`而不应该直接`throw error`
- 查出commit实例化的时候，由于奇怪的原因，没有把date和message一通纳入到sha1的计算里面，导致了计算出来同一个sha1值，指向同一个父亲
  - debug最牢的一块，**耗时3小时**，手输指令，当时排查了好多好多原因
  - 这个过程重写了一大堆的封装
- reset想不明白为什么会有时光倒流的机制，后来发现写heads的应该只存当前commit
  - **存储链条应该是由commit.fatherId**给串联起来的才对，一开始写log的时候就思维定势了
- 加上了merge commit这一类很特殊的commit机制，知道了实际上可能**有第二个父亲**的设定
- 修改了找祖先的逻辑，自己手搓了有向无环图的节点开始的遍历，和结合bfs找共同祖先的算法
- *各种各样其他的细节问题* 这整个过程那是十分痛苦的debugging，而且一个Bug会牵连其他许许多多的Bug，LLM在这个过程中，说实话，也没帮上什么正忙（笑）

总结下来这里会出这么多岔子，**最主要有两个原因**，也希望能给将来可能学习这门课程的你提个醒：

- **写代码之前先把结构设计好**，边写边改效率肯定不高
- 读清楚项目要求，一定要**有耐心去看那些英文文本描述**，实在不懂请LLM给你翻译概要

https://shuiyuan.sjtu.edu.cn/t/topic/447931/868 *Generated By Gemini*

> 你不需要现在就是大师。**你有整整四年的时间，甚至更久，去犯错、去写垃圾代码、去编难听的曲子。**
>
> 这不可耻。每一个现在在大神位置上的人，当年都和你一样，在屏幕前抓耳挠腮，觉得自己是个废物。
>
> 区别只在于，他们在一边骂自己废物的时候，一边把那个烂代码又改了一行。

https://shuiyuan.sjtu.edu.cn/t/topic/447931/876

> 在第28小时终于来了巨大进展，，，
>
> 简单来说是一个有向无环图的遍历实现涉及到一个
>
> 自己手搓的算法不是最优解

> 大概是，先bfs扫一遍current的历史记录，返回一个大的哈希表，key值是id，val值是步长；再bfs扫一遍given的记录找到对应val步长最短的，时间O(n)，空间O(n)
>
> 目前代码结构的问题：
>
> - 函数封装有意识做，但做的乱七八糟的
> - heads文件里面存了一整个Linkedlist，而且调用的非常混乱
> - 所有寻址应该基于Objects和文件管理做而不是指望heads里面的history

其实后面做到remote的extra credit的部分就很简单了。 **喵喵喵**

What I Have Learned 学んだこと

- 大型项目的初体验
- 函数封装的意识
- 设计的思维
- 代码的可读性与简洁性
- 巩固加深数据结构知识
- **复现**的学习方法
- ……

Summary まとめ

不知道该写什么喵，也不想写些煽情的，该说的上面也都说了呢

By Tab_1bit0 2026.2.15