TEAM 4.3

Report of MS3

Current Vision:

After we finished MS2, we immediately put our hands on the most important parts of our “Billbook” project: the main algorithm and the csv read-in & write-out system. Previously we have determined the most ideal runtime for our algorithm, so we set it as our target as we develop the actual algorithm. After we finished the algorithm, the team worked on csv read-in & write out and testing at the same time. We reconsidered what and in which format we should store in our csv file and, fixed the bugs i our code. After going through a lot of rethink and revise, we are now confident to say that our program now gives the best result in minimal runtime, as well as provides a range of functionalities for our users to obtain any information they need.

Summary of Progress (Since MS2):

As we proceed to develop our algorithm and build the backend system, more and more problems occurs and we managed to solve them all. Our work after MS2 could be classified into two categories: Algorithm-oriented and Program-oriented. The first term refers to work related to algorithm itself and the latter term refers to modifications & innovations to further improve our UI and minimize runtime.

Algorithm-Oriented: We have implemented the great Optimizer Algorithm, a greedy algorithm that successfully enforces the core functionality of our project: reduce the number of transactions needed to clear the debt generated by all inputted events to the least possible. We wrote a lot of helper functions to achieve that, so the testing process was a pain. Most of the bugs were found using some “Large and Complicated” test cases (i.e., a set of many events concerning many people and messy transactions), since these kind of real-world based cases could trigger bugs that were deeply hidden. We also perfected the algorithm through testing by adding additional features, such as clipping the debt to 0 when the debt is too low (<10^-5).

Program-Oriented: We first thinked about what could be the best form to store our data: event info or user info? In order to give our users the capability to find any past events they recorded, we chose event as our final decision. After finalizing the read-in and write-out system, we encountered a new problem: every time the users make an instruction, our program would recalculate the opimized payback solution. This also includes when users did something unrelated to the solution. As a result, some runtime is wasted. We considered putting our results into global variables but we found that this feature is not that well accepted by ocaml. So we decide to use reference to store our calculated results and use another bool reference to serve as a symbol of whether the result needs to be updated (much like the VI protocol for multicore CPU). This manner optimizes both space and time complexity of our program. After that, we implemented the rest of UI features to allow our users to get any information they want in our database.

Activity Breakdown:

Rachel Jiang:

- Implemented Optimizer Algorithm, Debugged the Algorithm
- Revised terminal UI
- Wrote test cases

Haoxuan Zou:

- Implemented Optimizer Algorithm, Debugged the Algorithm
- Wrote other documents needed for submission
- Wrote test cases

James Liu:

- Implemented Backend system
- Implemented terminal UI
- Wrote test cases
- Wrote other documents needed for submission

Ian Chen:

- Implemented Backend system
- Implemented terminal UI
- Wrote test cases

Productivity Analysis:

The team finished all the tasks as planned. As the semester went on, our team members all faced problems which could possibly harm our general productivity. We all had compact schedules and multiple tasks to finish at the same time. But as a team, we understood each other and decided to pick up other member's tasks while they were busy at the time. After all, each member contributed fairly a lot to the project and we made considerable progress. We sticked to our plan and put in extra time and energy to make our program better.