Chapter 2 & Chapter 3

(This version is not the final version. It can be useful material for writing the final report. The order of material is meaningful.)

**[Classical programming learning ways (edit from the proposal)]**

Writing code, undoubtedly, is one of the core parts of the Science in Computing programme, and consequently, it is also the toughest part for learners, especially beginners. Students are not only required to have the ability to analyze the solving logic of problems, but also they need to be proficient in different coding syntaxes. In this case, students need to make many attempts to solve the programming problems, and they may easily give up during this period. To ameliorate this situation, two classical programming learning methods are introduced into the courses to improve students’ code-writing skills. Specifically, one of the methods is a preparation action for code writing, Coached Program Planning. This method guides students to analyze the problems and design the logical procedures with native-language style pseudocode for solving these problems [2, 3] so that the logical parts of codes can be partially separated from the syntax parts reducing the cognitive burden during coding to some extent. Another method is code tracing, which is to track the changes of variables by hand during the execution of codes [4]. Although this method does not have a direct effect on code writing and cannot be applied to every programming question because of its cumbersome procedure, it, through the accumulation of experience in code reading, still can provide some kind of auxiliary help in improving code writing skills.

**[Previous Parsons problem application]**

Besides the above two programming learning methods, a new way, Parsons problem, was created to prepare students for writing code [5]. Instead of letting students directly write code, Parsons problem provides a set of code fragments – including the solutions to the questions and some distractors (some common errors) - for students to choose from and reorder by dragging and dropping [5]. During this period, students can get some instruction feedback for their reordered answers, and they need to repeat reordering until their answers are one hundred percent correct [5]. This method provides notable help in introductory programming study. To be more specific, the puzzle-like game-style Parson problem can improve students’ engagement and motivation in learning programming [5]. And with prepared code fragments and instant feedback, the levels of difficulty of the questions are reduced, and students are more likely to persist in programming instead of giving up halfway. Besides, Parsons problem can be of use to reduce cognitive load since students are only required to reorder the prepared code fragments instead of writing code directly [1]. And some context (fixed code) also can be provided to students to reduce cognitive load further [6]. In addition, Parsons problem integrates the respective advantages of both Coached Program Planning method and the code tracing method. Specifically, Parsons and Haden picked up an idea to include activity diagrams in the questions’ descriptions to help students to understand the solution logic of problems [5], which has a similar function to Coached Program Planning method. And since Parsons problem also requires students to read and understand the meaning of every code fragment, it also takes advantage of code reading just like the code tracing method. In other words, Parsons problem provides magnificent solution examples for students to learn from, giving them some reference material to think about solution steps when meeting some similar questions. Finally, Parsons problem also has an effect on helping students to cultivate good coding habits. For example, the distractors in Parsons problem can be used to show some improper variable names, which assists students to distinguish good names from bad names and train the habits of using meaningful and conforming naming rules names [5].

**[The ~~need of using~~ benefits of using Parsons problem in Data Structures and Algorithms courses]**

Although Parsons problem makes great success in teaching programming, it is limited to only being used in introductory programming courses, and it has not been expanded in middle-level programming courses, for example, Data Structures and Algorithms courses. ~~From my~~ ~~perspective~~, Parsons problem still can demonstrate its superiority in Data Structures and Algorithms courses. Admittedly, students taking intermediate-level programming courses should be able to write code instead of just rearranging the order of provided answer blocks. However, because of the abstractness and universality of programming in this course, it also is a challenging task for students to write it directly (for example, recursion problems). Thus, it is of the essence to introduce Parsons problem to build a “bridge” for students to grow their capability to write code directly by themselves. But, since there are some differences between introductory programming courses with Data Structures and Algorithms courses, the previous Parsons problem in introductory programming courses does not fit the situation in Data Structures and Algorithms courses, and it is not suitable to apply the previous Parsons problem directly. Consequently, some new ideas should be introduced to Parsons problem. The detailed difference between the two courses and the limitation of the previous Parsons problem will be discussed in the following paragraphs.

**[The difference between two courses]**

There are some major differences between introductory programming courses and Data Structures and Algorithms courses. To be more specific, the programming in introductory courses is simple and concrete. It only requires students to understand codes line by line. And most of the parts in this course are separated and not related so it is almost unneeded for students to compare the differences. By contrast, the programming in Data Structures and Algorithms courses is complex and abstract. It focuses on large block code understanding instead of line code understanding. Students are required to not only understand every line of code but also the specific data structures, the classes with their methods (Object Oriented Programming), the recursion methods for general use, and the algorithms with the corresponding analysis. Besides, there are more similar concepts needed to be compared, for example, stacks and queues, different types of trees, and different algorithms for the same purposes (like bubble sort and selection sort for sorting, and breadth-first search, depth-first-search for searching).

**[The specific question in Data Structures and Algorithms and the limitation of previous Parsons problem]**

The differences between introductory programming courses and Data Structures and Algorithms courses lead to the limitation of using the previous design of / types of Parsons problem directly in the Data Structures and Algorithms courses.

To show the limitation in detail, five types of questions in Data Structures and Algorithms courses are summarized from the reference book Data Structures and Algorithms in Python

To show the limitation, we have analyzed the exercises in the book and identified five types of common questions.

To show the limitation, the exercises in the book were analyzed and five types of … were identified.

[7].

**[Object Oriented Programming question types]**

**[References]**

[1] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In Proceedings of the Twenty-Second Australasian Computing Education Conference (ACE’20), February 3–7, 2020, Melbourne, VIC, Australia. ACM, New Y ork, NY , USA, 8 pages. https://doi.org/10.1145/3373165.3373187

[2] Rita Garcia, Katrina Falkner, and Rebecca Vivian. 2018. Scaffolding the Design Process Using Parsons Problems. In Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling ’18). ACM, New York, NY, USA, Article 26, 2 pages. https://doi.org/10.1145/3279720.3279746

[3] H. Chad Lane and Kurt VanLehn. 2003. Coached program planning: dialogue-based support for novice program design. SIGCSE Bull. 35, 1 (January 2003), 148–152. https://doi.org/10.1145/792548.611955

[4] Code Tracing. Retrieved September 7, 2022, from https://microcredentials.digitalpromise.org/explore/code- tracing

[5] Dale Parsons and Patricia Haden. 2006. Parson’s Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE ’06). Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 157–163. <http://dl.acm.org.ezproxy.auckland.ac.nz/citation.cfm?id=1151869.1151890>

[6] Garner, Stuart. “An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming.” Proceedings of the 2007 InSITE Conference, 2007, <https://doi.org/10.28945/3127>.

[7] Goodrich, Michael T., et al. Data Structures and Algorithms in Python. Wiley, 2018.