Recursion or Iteration: Does it Matter What Students Choose?

Ramy Esteero, Mohammed Khan, Mohamed Mohamed, Larry Yueli Zhang, Daniel Zingaro

ABSTRACT

1. **Recursion and Iteration in Computer Science:** Recursion and iteration are fundamental concepts taught in introductory computer science courses, particularly in CS2, where recursion is typically introduced, and control-flow concepts are reinforced.
2. **Examining Student Choices:** The paper investigates how CS2 students choose between recursion and iteration when faced with problems that can be solved using either approach. It also explores how this choice relates to the correctness of their code.
3. **Student Choices:** The analysis reveals that:
    - 19% of students opt for iteration.
    - 51% choose recursion.
    - 16% combine both iteration and recursion in their solutions.
4. **Correctness and Performance:** In terms of correctness, it is found that students who choose iteration tend to perform better than those who opt for recursion or a combination of both.
5. **Concerns about Understanding:** The paper expresses concern about the number of students who seem to lack a clear understanding of what the question is asking.
6. **Guidance for Choosing Control-Flow Strategies:** The paper concludes with suggestions for helping students select appropriate control-flow strategies and discusses the inclusion of this type of question on final exams.

INTRODUCTION

1. **Recursion in CS2 Courses:** Recursion is a fundamental problem-solving technique and a significant component in many CS2 (second course in computer science) courses. It is recognized as a challenging topic for students, although there are differing perspectives on its difficulty.
2. **Comparison with Iteration:** Iteration is typically introduced before recursion and is often considered easier for students to learn and apply. Iterative solutions are often sufficient for simple numeric or list-based problems, while recursive solutions may simulate control flow through a loop.
3. **Recursion and Binary Trees:** Some problems involving binary trees may be more naturally solved through recursion (e.g., preorder traversal), while others may be better suited for iteration (e.g., searching a binary search tree).

4. **Exam Problem on Binary Trees:** The paper explores how CS2 students approach exam problems related to binary trees when there is no pre-specified solution plan. It investigates whether students tend to use more familiar iterative code or if they choose recursive solutions based on the problem's characteristics.
5. **Relationship Between Student Choices and Performance:** The paper aims to understand if there is a relationship between students' choices (iteration, recursion, or a combination of both) and their performance on the specific question or the entire exam.
6. **Typical Solution Attempts:** The study seeks to provide insights into the common approaches students take when attempting to solve problems related to binary trees in each of the three categories: iteration, recursion, and combined methods.

LITERATURE REVIEW

1. **Challenges in Teaching Recursion:** Teaching and learning recursion is a widely researched area, and students often struggle to grasp it due to the various ways recursion can be understood and misunderstood.
2. **Different Approaches to Tracing Recursive Code:** Some students successfully trace recursive code using techniques like simulating stack-based execution, accumulating pending results, bottom-up substitution of results, or even predicting without tracing. However, there are various mental models of recursion, some of which are not valid, such as conceiving recursion as a loop.
3. **Difficulty in Writing Recursive Code for Binary Trees:** Students face particular difficulty when tasked with writing recursive code for binary trees. Goal-plan analysis is one method used to classify these difficulties, breaking down the problem into functional components (goals) and analyzing students' solution attempts (plans) for each goal.
4. **Issues with Recursive Code on Binary Trees:** In one study, students were asked to write a recursive function to count nodes in a binary tree with exactly one child, but only a small percentage successfully solved it. Many students added unnecessary conditionals to avoid recursive calls, resulting in longer and error-prone code. Base cases posed significant problems for students.
5. **Student Choice Between Recursive and Iterative Code:** When students are given a choice between writing recursive or iterative code, a study on CS1 students suggests that they are more inclined to choose iteration. Surprisingly, students who wrote recursive code were more likely to solve problems correctly. Students were able to comprehend sample recursive and iterative solutions at similar rates, indicating a potential disconnect between student choices and actual outcomes.
6. **Domain Dependency:** Previous research indicates that the choice between iteration and recursion depends on the problem domain. For numeric inputs, students' choices and comprehension might differ from problems involving recursive data structures.

7. **Limited Investigation on Student Choice:** While previous research has explored student choices between recursion and iteration, there is a gap in understanding what happens when students are allowed to choose their own strategy for solving nontrivial problems on trees.

CONTEXT

1. **Context of the Study:** The study was conducted during the Jan-Apr 2017 CS2 course at a major North American research university. Students had previously learned Python in the prerequisite CS1 course. The CS2 course covered various topics, including object-oriented programming, exceptions, recursion, binary and n-ary trees, linked lists, sorting, and runtime complexity.
2. **Recursion in the Course:** Recursion is a core focus of the CS2 course, with students learning it in the context of numbers and strings. They revisit recursion when introduced to linked lists and binary trees. Examples of both recursive and iterative code are provided to students for various tasks involving these data structures.
3. **Exam Question:** The study focuses on a question from the April 2017 final exam. The question provides the definition of a binary tree node class (BTNode) and describes a Python function, "deepest_ancestor," that should return the maximum-depth common ancestor of two given nodes (node1 and node2).
4. **Sample Solution:** The sample solution provided for the problem was iterative and utilized two sequential loops. The first loop traversed up the tree from the deeper node to reach the same depth as the shallower node. The second loop moved both nodes up the tree until they became references to the same node, representing the deepest common ancestor.
5. **Student Approach:** While grading, it was observed that a significant portion of students used recursive approaches or mixed approaches involving both recursion and iteration. These solutions varied in correctness and code complexity.
6. **Definition of Node Depth:** Node depth was defined as the number of edges required to traverse from the root to a specific node, with the root having a depth of 0. Students were familiar with binary tree nodes storing items, left and right references, as well as the concept of node depths and parent nodes, which were provided in the exam question to facilitate their use in solutions.

DATA ANALYSIS

1. **Data Collection:** The study collected a total of 397 student exams, and each student's response to the "deepest ancestor" question was transcribed into its own Python file.
2. **Test Cases:** To evaluate correctness, the study created 10 test cases that covered various aspects of the problem. The number of passed test cases was used as a measure of correctness.

3. **Exam Grades:** The study recorded additional information for each student, including the total grade on the exam and the grades for both Part A and Part B of the question. Part A's grade (out of 2 marks) indicated the student's understanding of the problem and the question's requirements. The total exam grade (out of a maximum of 50 points) served as a general measure of the student's grasp of the knowledge taught in the CS2 course.
4. **Categorization of Solutions:** Each student's solution was manually inspected and categorized into one of four groups:
    - (1) **Inadequate:** Solutions that used neither recursion nor iteration, including blank solutions.
    - (2) **Recursion:** Solutions that utilized recursion without employing iteration, even if recursion was used in a helper function.
    - (3) **Iteration:** Solutions that involved any form of iteration (e.g., for loops, while loops) without recursion, even if iteration was used in a helper function.
    - (4) **Mixed:** Solutions that combined both recursion and iteration in their approach.

TEST ANALYSIS

1. **Test Cases Design:** The study designed 10 test cases for evaluating student submissions. These test cases aimed to evoke different execution behaviors in student solutions. The test cases were crafted in a style typical of automated grading software for assignments.
2. **Test Case Scenarios:** The 10 test cases covered various scenarios, including cases where:
    - node1 and node2 are the same node in a single-node tree.
    - node1 and node2 are the same node in a larger tree.
    - node1 and node2 are distinct but on the same level of the tree.
    - node1 and node2 are distinct but on the same level of the tree, and the deepest common ancestor is not the root.
    - node1 is one level deeper than node2, and node2 is the deepest common ancestor.
    - node2 is one level deeper than node1, and node1 is the deepest common ancestor.
    - node2 is deeper than node1, and the deepest common ancestor is the parent of node1.
    - node2 is several levels deeper than node1, and node1 is the deepest common ancestor.
    - node1 is several levels deeper than node2, and node2 is the deepest common ancestor.
    - node1 and node2 are different nodes in an unbalanced tree.

3. **Test Case Difficulty:** The tests were sequenced in terms of perceived difficulty, with the expectation that more students would pass the early tests and fewer would pass the later ones. However, there were variations in student success rates. For example, Test 2 was challenging, with only 23% of students passing it. Test 1 was easier for students, as more students passed it compared to Test 2.
4. **Interpretation of Base Cases:** Some students interpreted the base case as a single-node tree, while others considered it as two references to the same node in any tree.
5. **Handling Nodes on the Same Level:** Students generally performed better when handling nodes on the same level compared to nodes on different levels.
6. **Complex Test Cases:** Test cases involving moving one node up multiple levels before advancing both nodes to their common ancestor proved challenging for students (e.g., Test 8 and Test 9).
7. **Correlation with Exam Grades:** The study observed a correlation between the number of test cases passed and the student's grade on the exam. This suggests that performance on this question correlates with the overall exam grade, highlighting the meaningfulness of the test cases in measuring student progress on the question.

DISCUSSION

1. **Student Choice of Recursion:** A majority of students chose to use recursion when attempting the question. This choice might be due to the association of recursion with questions on trees and the emphasis on recursion in the CS2 course. However, this choice of recursion was often not the most appropriate approach for the question.
2. **Recursive Solutions and Lower Grades:** Recursive solutions were associated with lower grades on the question. This suggests that many students might have chosen an approach (recursion) that was not well-suited to the problem.
3. **Use of Mixed Code:** A considerable number of students submitted solutions that used both recursion and iteration (mixed code). While some of these submissions made progress, they often led to unnecessarily complex and sometimes incorrect solutions. Using both control-flow mechanisms together may indicate a complex solution.
4. **Fairness of the Question:** Despite being a difficult question that required students to apply multiple binary-tree concepts and use depth and parent references, the question was considered fair for a CS2 course exam. A mix of problem types, including code-reading and conceptual questions, is suggested to assess student learning effectively.
5. **Teaching Control-Flow Structures:** It is suggested that CS2 courses should explicitly highlight when and why recursion is used. Students should be encouraged to think carefully about whether using both recursion and iteration in the same function is warranted. Questions about whether recursion or iteration should be used to solve specific problems could be included on exams, emphasizing the importance of choosing the right control-flow structure.

CONCLUSION

1. **Recursion Introduction in CS2:** A typical CS2 (second computer science) course introduces recursion and dedicates significant time to help students understand and think recursively. This emphasis on recursion is justified because of documented student struggles with recursion in computer science education.
2. **Challenges in Choosing Recursion:** Often, it is evident from a problem description whether recursion should be used in the solution. However, this study suggests that students may not have a strong intuition for when to apply recursion and when to apply iteration. They might rely on heuristics, such as problem characteristics (e.g., involving trees) or difficulty level, to decide whether recursion applies.
3. **Implicit Assumptions:** Students might assume that recursion applies to a problem because it is an important part of the CS2 curriculum. This can lead to the inappropriate use of recursion when other control-flow structures would be more suitable.
4. **Teaching Implications:** Practical teaching implications from this research include providing explicit guidance to students on how to choose appropriate control-flow structures for problem-solving. Instructors can encourage students to choose and justify their solution strategies on exam questions, emphasizing the importance of understanding when to use recursion and when to use other techniques like iteration.