THE ESSENCE OF RECURSION: REDUCTION, DELEGATION, AND VISUALIZATION [*]

Abdelaziz Fellah and Ajay Bandi
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468
011-660-562-0803 and 011-660-541-1838
afellah@nwmissouri.edu and ajay@nwmissouri.edu

ABSTRACT

Recursion remains one of the most challenging concepts for students to comprehend adequately and educators to teach effectively. In this paper, we propose a learning approach for teaching recursion and highlighting the importance of recursive thinking and abstraction skills in problem solving through a well-defined sequence of steps. Furthermore, this approach is complemented by a 2D computational model and simulation tool, RDViz. This software interactively integrates base cases and recursive invocations, and it displays the sequence of states of the recursive process before exposing students to coding. The results of our case studies have shown evidence that all students achieved a substantial gain in recursive computations as well as appropriate basic abstraction skills.

1. INTRODUCTION

Recursion is often considered as an advanced topic in the computing undergraduate curriculum, broadly introduced in the last weeks of the course and treated in several traditional recursive algorithms in upper-level courses. Despite the important repertoire of literature on recursion, a large number of students, lower, upper and even novice programmers, still struggle understanding recursion and exploring the concept at a greater depth. Thus conceptually and practically, students lose one the most important algorithmic techniques in the ACM computing curriculum guideline.

[*] Copyright © 2018 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

In general, educators tend to follow a uniform strategy by introducing the traditional classic numerical computing examples of factorials numbers and Fibonacci series which are often the first few examples to be learned by students to illustrate the concept of recursion. Both examples come from a problem domain that is usually expressed as a formula. Moreover, both examples can be accomplished iteratively which makes recursion a pointless concept for students rather than an important programming technique in the discipline of computer science. Among other examples, Towers of Hanoi is another interesting recursive example covered in the classroom, however it is not an intuitive way to introduce to lower-level students what we actually mean by recursion. Nevertheless, these examples, among many others, do illustrate the full power of recursion, however it is still not enough to instill students with some level of technical confidence with what we refer to as recursive thinking. Despite educator's continuous novel teaching efforts throughout the years and students' effective learning, recursion remains one of most challenging concepts for students to master adequately and for educators to teach effectively.

The concept is surrounded with several misconceptions and myths from students' perspective which have an impact on the students' learning outcome. Various questions have sparked academic debates among educators and researchers about this topic and how it should be taught and delivered effectively. For example, should recursion be taught early or late or not at all in the first introductory programming course? Should recursion be taught before or after loops? Should students be required to understand stacks, a related and inseparable topic to recursion for many educators? Does recursion entail tail-recursion? [9]. How about recursive types and data structures (i.e., lists and trees)? [7].

Studies [7, 10, 12] have shown that students choose iterative over recursive approaches as their main approach when solving programming problems. This is not surprising and it has been drawn as a conclusion in our study. In general, students continue to paint and find recursion as a difficult concept in spite of the different palette of instructional approaches introduced in the classroom.

In this paper, we propose a strategic approach from our successful practice for teaching recursion. The approach is fully complemented by the computational model RDViz to overcome challenges and difficulties that students face in the subject, and strengthen their abilities and skills for better understanding of what we refer to as recursive thinking.

Our contribution is two-fold. First, based on our past technical and pedagogical experience teaching recursion, we designed and implemented a macroscopic approach that we referred to as RDViz in light of recursive thinking. The approach helps student intuitively comprehend how recursive thinking works before exposing them to coding. In the beginning, the strategy used in the approach conceals the implementation from students but promotes the recursive thinking through RDViz, a software tool underpinning the high level principle of recursive thinking, that is, reduce, delegate, and visualize. Second, further investigation of our approach has been brought to the classroom to support both educators and students, and whether it can alleviate both the challenges of teaching and the difficulty of learning recursive thinking skills in problem solving. The results of our case studies showed evidence all students achieved a substantial gain in recursive computations as well as appropriate basic abstraction skills.

The rest of the paper is organized as follows. Section 2 overviews different approaches in the literature for teaching recursion. Section 3 describes the self-recursive thinking. Section 4 highlights the rationale of teaching or not teaching stack in recursion. Section 5 describe and details the proposed approach. Sections 6 and 7 present our case studies carried out and analyze the results. Finally, section 8 concludes our work.

2. ADVOCATING APPROACHES TO THE TEACHING OF RECURSION

There is a substantial literature on recursion because of the importance of the topic in the computer science discipline and the difficulty to comprehend recursion by students, a number of effective strategies and approaches have been drawn to teaching recursion. For example, teaching recursion before iteration has been advocated by many academic authors who believe that recursion has been circumvented by loops and students have already built a misconception about recursion [4, 5, 8, 11]. Simple counting songs to show repeated and reduced recursive patterns in the light of recursive thinking have been introduced in [3]. Songs in children's rhymes which contain an element of recursion substructure and patterns of self-recurring have been used in the classroom. For example, the green bottles hanging on the wall as referred in [3]. Debates on the choice of the programming language, for example, functional languages such as Haskell with no explicit iterative constructs is more natural for recursion than imperative languages. The choice and order of topics have also been the subject of pedagogical and technical arguments among educators [1].

Other approaches to teaching recursion is through visualization, implementation, and tracing where a number of animation tools have been developed. For example, Alice which is built on top of Java where students can see a visual mapping of the lines of their code to animation actions [4]. Another approach to teaching recursion visualization tool is ChiQat-Tutor [8], an overall system that help students to learn mainly data structures and algorithms. However, these strategies are effective teaching tool to upper-level students but they are not presented in simple forms to show the fundamental core characteristic of recursion and the recursive thinking. Other authors have suggested new methods for teaching recursion, e.g., [2, 6, 9] possibly with the support of specific software aids.

3. THE SELF-RECURSIVE THINKING

Recursion is technically a self-referencing function that is a function that calls itself. Recursion which is based on divide and conquer strategy and its conditions should be conveyed to students in a very understandable way that we describe loosely in terms of a base case(s) and a recursive case(s) as follows:

a. If the instance of the given problem is simple and small, then it can be solved immediately resulting in termination. (*Base case(s)*).
b. Otherwise, the problem is reduced into one or more smaller instances of the same problem (subproblems). (*Recursive call(s)*).

In the light of the above, students should be aware of the following conditions.

- There must be no infinite number of subproblems (recursive calls).
- The recursive calls (reductions) must eventually stop with an elementary base case.
- Each recursive call must make progress toward the base case.
- Recursion may have one or more base cases and recursive calls.

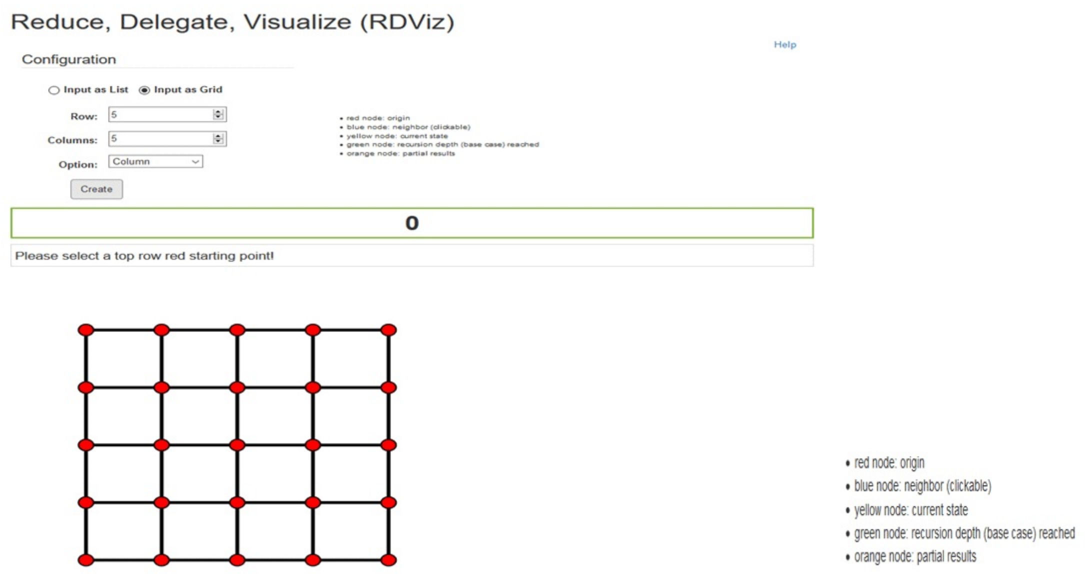
4. STACK OR NO STACK. THAT IS RECURSION!

The stack is not only used to manage recursive functions, but rather any regular re-entrant function requires managing the stack. Why the association between the stack and recursion becomes important only when students are introduced to recursion? Based on our experience teaching recursion in lower-level courses without resorting to the stack in coding has shown no negative impact on student's understanding. However, obligating students to understand the stack would complicate their tasks. If the stack is introduced then it should be kept to a minimum and should not deflect students' focus away from the main topic. Nevertheless, students shall be exposed to stack, stack overflow, and tail-recursion in order to reinforce their programming skills in upper-level courses.

5. REDUCE, DELEGATE, AND VISUALIZE APPROACH (RDViz)

We developed a tool called RDViz written in JavaScript to help students intuitively comprehend how recursion works before exposing them to coding. The implementation of RDviz is based on the high-level principle - reduce, delegate and visualize. The difference between RDViz and many other software in the literature [6, 9] is that our tool has been designed at the macroscopic level and the principle of recursive thinking. In addition, RDViz has been implemented to perform three separate and dependent tasks - reduction, delegation and visualization.

The layout of RDViz is a grid of rows and columns that shows the different stages of the recursive process. The graphical user interface has features that allows the user to enter the two dimensional size of the grid, rows and columns. In addition, the user would be able to enter the data to visualize the stages of tasks through the states (dots) of the grid using the mouse. The RDViz top level view of a 5x5 grid and the legend of the color of the nodes is shown in Figure 1. Figure 2 (a) lays out successive recursive calls (blue dots) across the first column towards the base case. Figure 2 (b) illustrates the backward flow of information of unfolding suspended computations triggered by the base case (first green dot in the leftmost subfigure).


Reduce, Delegate, Visualize (RDViz)

Help

Configuration

○ Input as List [✓ Input as Grid]

Row: 5

Columns: 5

Option: Column

Create

- red node: origin
- blue node: neighbor (clickable)
- yellow node: current state
- green node: recursion depth (base case) reached
- orange node: partial results

0

Please select a top row red starting point!

- red node: origin
- blue node: neighbor (clickable)
- yellow node: current state
- green node: recursion depth (base case) reached
- orange node: partial results

Figure 1. Overview of the layout of RDViz.

Let us consider an example, counting the number of students in a classroom. We can solve this problem either iteratively or recursively. Iteratively, the problem can be simply solved by counting 1, 2, 3, …., n. Recursively, the problem can be solved using RDViz which is based on the principle: reduce, delegate and visualize. The problem can be reduced to count the number of students in every column in the classroom.

Let us fix one column, assume that there are no empty seats between adjacent students and the first three students are Adam, Hannah, Babu, and the last two students are Xavier and Zenith. Adam keeps the count of 1 and asks Hannah to check whether there is one student sitting behind her. Then, Hannah keeps the count of 1 and delegates the same question to Babu until the question reaches Zenith. Zenith checks that there is no student sitting behind him (base case) then returns the count of 1 to Xavier, who adds his count which is propagated through the column until the result reaches Adam. The approach is shown in Figures 2 (a) and 2 (b). In addition, the approach illustrates multiple base cases and recursive cases fragmented across different rows and columns which reflect the possible complexity of the depth of recursion as shown in Figures 2 (c) and 2 (d).

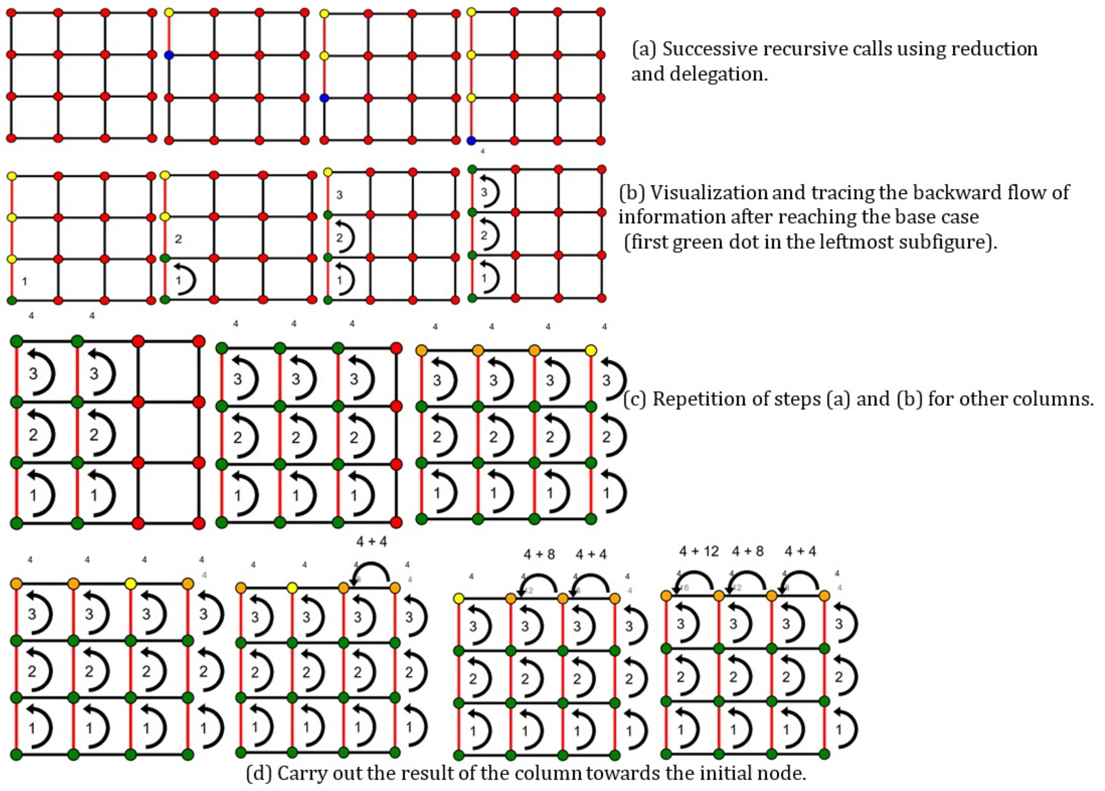

(a) Successive recursive calls using reduction and delegation.
(b) Visualization and tracing the backward flow of information after reaching the base case (first green dot in the leftmost subfigure).
(c) Repetition of steps (a) and (b) for other columns.
(d) Carry out the result of the column towards the initial node.
4 + 4
4 + 8
4 + 12

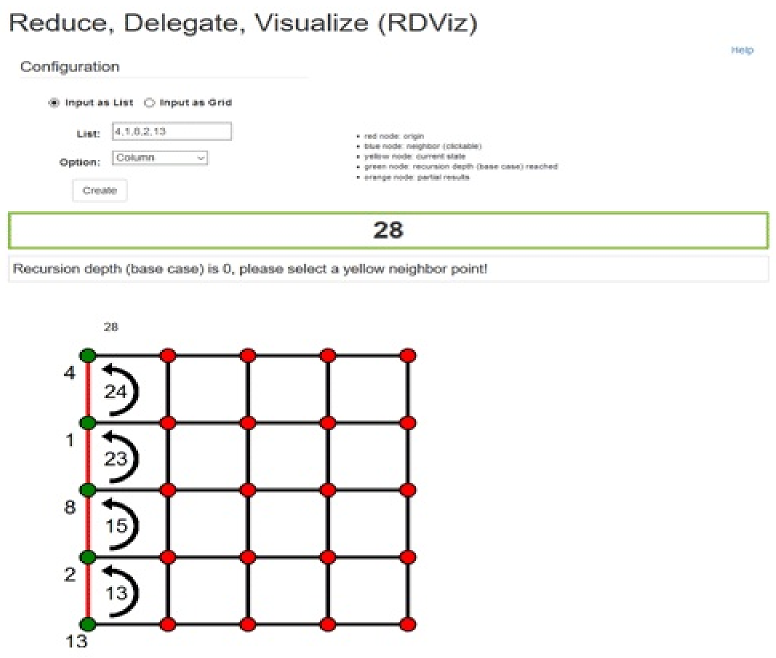
Figure 2. Stages of counting the number of students in a column using RDViz.

6. CASE STUDIES

We tested our approach by carrying out case studies on teaching and learning recursion in the classroom using Java. Students have been given a set of problems of different levels of difficulty to be solved as assignments. Our study setting has the following sequence of steps:

1. Students solved the set of problems using iteration [15].
2. Students have been pretested on the concept of recursion
3. Students have been taught recursion without RDViz.
4. Students have been taught recursion using RDViz in the lab. Full description of the approach and subsequently the usage of RDViz were explained to student. One of the important activities using RDViz is to have students highlighting the base and recursive cases in some typical examples such as finding the maximum and the sum of an array of n integers as shown in Figure 3. RDViz integrates explicitly both of the forward and backward flow of information. The forward flow of information represents the successive recursive calls while the backward flow of information represents the unfolding suspended computations triggered by the base cases. Importantly, RDViz does not explicitly resort to the mechanism of stack.
5. Students solved the same set of problems as stated in step 1 using recursion
6. Students have been post tested on the recursion concept.

The readers can refer to the pretest and posttest survey questions available at [13, 14].



Reduce, Delegate, Visualize (RDViz)

Configuration

(● Input as List ○ Input as Grid)

List: 4,1,8,2,13

Option: Column

- red node: origin
- blue node: neighbor (clickable)
- yellow node: current state
- green node: recursion depth (base case) reached
- orange node: partial results

28

Recursion depth (base case) is 0, please select a yellow neighbor point!

Figure 3. Sum of n integers using RDViz.

We conducted our case studies in an object-oriented programming course required for lower and upper level students. Out of 62 students who participated, 37 students were computer science and 25 students were non-computer science majors for whom the course is required. This course provides fast-paced coverage of object-oriented programming and data structures concepts in Java.

7. RESULTS AND ANALYSIS

The results of the pretest before introducing the recursion in this study, 34 out of 62 students, have prior knowledge about recursion and 28 students were not aware of recursion. Later, we taught the recursion using RDViz. After the posttest, all the students have substantially strengthened their technical knowledge and comprehension of the concept.

Table I. Students' level of understanding recursion.

Level of understanding	1 (Never heard of)	2 (Exposure)	3 (Familiarity)	4 (Knowledge able)	5 (Mastering)
Pretest	6.45%	14.51%	50.00%	22.58%	6.45%
Posttest	0.00%	1.61%	33.80%	51.61%	12.90%

Diff. in level of understanding	6.45%↓	12.90%↓	16.12%↓	29.10%↑	6.50%↑

The 'mastering' level did improve by 6.5%. Half of the students in level 5 moved from other levels. The number of students who are knowledgeable about the concept has increased by 29.1%. The increase arrow (?) shows the substantial positive increase of the understanding level as shown in Table I. The decrease arrow (?) should be considered as a positive decrease highlighting the number of students that moved from 'never heard of', 'exposure', 'familiarity' to better levels. For example, 16.12% shows a positive decrease in familiarity level. Based on the pretest and posttest numbers this positive decrease comes from 'never heard of' or 'exposure'. Overall there is no negative impact on introducing RDViz to students. A final question was asked during the posttest "Do you prefer iteration or Recursion" and we found that 68% still opted for iteration and this confirm the results of previous studies [10, 12].

8. CONCLUSIONS

The proposed computational model RDViz has provided effective evidence in strengthening students' learning and performance in light of recursive thinking, coding, and basic abstraction skills in problem solving. We considered recursive thinking as an important approach. Moreover, RDViz can implement similar recursive functions (i.e., factorial, power, and Fibonacci numbers).

The empirical evidence carried out in the classroom reinforced students' recursive programming in general and improved their performance in terms of their levels of understanding recursion. Furthermore, students' feedback on the RDViz tool is very positive and found quite easy to learn as it underpins the sequence of steps. Another angle to highlight from the study would be providing some insights on how the subject of recursive thinking and recursion have been taught by other educators.

As a possible investigation and extended assessment to (a) show the viability of the approach and the reliability of RDViz recursive computational model (b) identify any shortcomings and pitfalls, and (c) propose any possible improvements by extending this study to a larger computer science students' audience and for a longer period of time, at least for two years.

REFERENCES

[1] Christian, R., A survey on teaching and learning recursive programming. *Informatics in Education*, 87-119, 2014.

[2] Claudio, M., Learning (through) Recursion: A multidimensional analysis of the competences achieved by CS1 students, *Innovation and Technology in Computer Science Education*, 26-30, 2010.

[3] Dann, W., Cooper, S., and Pausch, R., Using visualization to teach novice recursion, *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, 109-11, 2001.

[4] Elynn, L., V, Shan., B, Beth. and C, Lin., A structured approach to teaching recursion using Cargo-Bot, *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 59-66, 2014.

[5] Franklyn, T., Teaching recursion before loops in CS1, *Journal of Computing in Small Colleges*, 14(4), 86-101, 1999.

[6. Jeffrey, E., Teaching and viewing recursion as delegation, *Journal of Computing Sciences in Colleges*, 23(1), 241-246, 2007.

[7] McCauley, R., et al., Recursion vs. iteration: An empirical study of comprehension revisited, *Proceedings of ACM Technical Symposium on Computer Science Education*, pp. 350-355, 2015.

[8] Michaelson, G. Teaching recursion with counting songs, ACM SIGCHI, June 2015.

[9] Mordechai, B., Recursion: From drama to program, *Journal of Computer Science. Education*, 11(3), 9-12, 1997.

[10] Reingold, E. M., Four apt elementary examples of recursion*, Language, Culture, Computation, Computing - Theory and Technology*, LNCS, pp -213-224. 2014.

[11] Rubio-Sánchez, M., Tail recursive programming by applying generalization, *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education*, pp. 98-102, 2010.

[12] Shelley, T. An evaluation of instructional methods for teaching recursion, California State University Stanislaus, May 2009. https://www.csustan.edu/honors/stirrings - inspected 20/3/15

[13] Pre survey link: https://goo.gl/forms/7n1r5KoI2N3lMaP42

[14] Post survey link: https://goo.gl/forms/81nQVspJ8n3aAtKb2

[15] Recursion survey questions: https://drive.google.com/open?id=1T7OwkFmR_DYYzV8cq4tNDcEOI8DqqfxaeCnq-aWrgbM