Informatics in Education, 2008, Vol. 7, No. 2, 173–180

The Unfortunate Novice Theme of Direct Transformation

David GINAT
Science Education Department, Tel-Aviv University
Tel-Aviv 69978, Israel
e-mail: ginat@post.tau.ac.il

Received: August 2008

Abstract. In many occasions, the text describing an algorithmic task may entail a rather intuitive, operational solution scheme. Yet, such a scheme may not necessarily be efficient or correct. Nevertheless, novices demonstrate tendencies to hastily design their solutions that way, and avoid seeking insightful patterns, which may yield better solutions. In this paper, we name and shed light on this theme, and illustrate the essential importance of elaborating insightful patterns, which one may assimilate as general problem solving notions.

Keywords: program design, algorithmic problem solving, novice mistakes.

1. Introduction

Consider the following algorithmic task: 'Given an input string of N letters, compute the number of sub-strings that start with an a, end with an a, and include exactly one a in-between'. The task specification encapsulates the description of an entity (sub-string), whose number of appearances in the given input should be counted. Entity counting is a fundamental theme in algorithmic tasks. The intuitive, natural approach of devising the required computation is by transforming the task specification into the straightforward operational scheme of: 'Find each of the specified entities and count their number'. We regard such straightforward transformation of the task specification into an operational scheme as *direct transformation* of an algorithmic task into an algorithmic solution.

Direct transformation is a common theme in algorithmic problem solving. In many occasions, the specification of an algorithmic task involves the description of entities that should be computed, and the description entails a rather immediate, direct "operative translation", which yields an intuitive solution scheme. The solution scheme may be correct, yet not insightful. Insight may be gained only after some careful task analysis, which unfolds hidden patterns and yields a better, more efficient and more elegant solution.

The intuitive solution scheme usually derives from operational reasoning, which is based on a direct behavior description of "how to do" (e.g., 'find . . . and count'); whereas the more insightful solution derives from assertional reasoning (Dijkstra *et al.*, 1989), which is based on assertions that capture entity characteristics in the sense of "what it means".

For example, in the above task of sub-string counting, the indicated intuitive solution, of 'find . . . and count', is derived from the immediate, operative notion of how to perform

the given task. However, assertional reasoning reveals the meaning of the sub-string definition: 'every three consecutive a's form a different sub-string; therefore, each a, apart from the last two a's starts exactly one sub-string'. The latter assertion implies an elegant conclusion, that the number of sub-strings is exactly the number of a's minus 2. The elegant conclusion encapsulates the rather general notion, of sufficiency of sub-element (e.g., prefix) processing.

Unfortunately, novices, and sometimes even senior students do not seek insight and hidden cues, as the one above, but are rather satisfied with their direct transformation of the task specification into a straightforward operational scheme. Such a problem solving approach is fundamentally deficient. It lacks the scientific perspective of gaining insight and aiming at the better possible solution. While students may "get-by" with such a non-scientific approach in the early stages of their studies, the absence of a scientific perspective will be more and more disturbing in later stages of their studies (Dijkstra *et al.*, 1989; Schoenfeld, 1992).

Yet, tutors sometimes let their students progress in their studies without a suitable scientific perspective, and textbooks do not underline such a perspective, as many are mostly focused on programming language constructs and general design guidelines (such as top-down and modularity). As a result, novices may become accustomed to the above non-scientific approach, which may, in turn act as an obstacle in the development of their problem solving competence and their perception of the science of computing.

In the course of our examination of novices' algorithmic solutions, we noticed the above unfortunate phenomenon time and again with both high-school and college students. Our objective in this paper is to shed light on this phenomenon, based on our accumulated experience, and advocate the embedment of scientific emphasis already from the very early stages of computer science teaching.

In the next section, we display our experience with students' inclination towards the direct transformation theme introduced above. Our presentation is composed of four different parts, each illustrating a different pattern missed by the students, due to their direct transformation. Each of the missed patterns in the illustrations encloses an essential notion, which one would like her students to assimilate during their studies. Thus, the illustrations serve not only as examples of the students' unfortunate tendencies, but also as resources for tutors' desired elaboration of their students' scientific perspectives. In the last section we conclude with a discussion of the consequences of the displayed illustrations, and tie our finding to related ones in the domains of mathematics and science education.

2. The Theme of Direct Transformation

In each of the sub-sections below, we first present an algorithmic task posed to the students. We describe our experience with novices' straightforward, direct transformation solutions, and then display the patterns, or cues the students should have revealed, but missed. These patterns encapsulate general notions, which are relevant for solving many other algorithmic tasks. Each sub-section is titled according to the pattern the students missed in the sub-section's task. The sub-sections are ordered from the simpler to the more involved tasks and patterns.

2.1. *Missed Pattern of Reversing*

The following algorithmic task is an extension of the counting task displayed earlier, in the Introduction. Counting computations are essential in algorithmics.

> **Number of sub-strings.** Given an input string of N letters, compute the number of sub-strings that start with an a, end with an a, and include at least one a in-between.

In our experience, many novices apply the approach of direct transformation in solving this task. For each a, in the given input string, they find (and count), one-by-one all the sub-strings that it "starts" (from left to right). The computation is implemented with a nested-loop, of time complexity O(N^2) and space complexity O(N) (as it involves repeated "passes" over the input).

It seems that those who offered this solution "translated" the text in the task description (of entity start-end) into an operational scheme that corresponds to the given description. In a sense, it is an operative simulation of the process one would have followed had she had to count "by hand" the number of sub-strings in the given string.

However, upon gaining insight into the task, one may look at sub-strings in reverse (from right to left), and notice that, actually, each a (after the 2nd one) ends as many sub-strings as the number of a's prior to it in the input string minus 1. For example, in the input string *cvbaabnabbaxx*, the 3rd a from the left ends one sub-string and the 4th a ends two sub-strings.

The above assertional observation implies that it is possible to count the required number of sub-strings by passing only once over the input, while accumulating the number of a's seen so far, and updating a variable that sums the total number of sub-strings every time a new a is met. The time complexity is O(N) and the space complexity – O(1).

The notion of reversing, missed by the novices who go the direct-transformation way, is a fundamental notion in computer science, with which every tutor would like her students to be acquainted (Ginat and Armoni, 2006).

2.2. *Missed Pattern of Operation Characteristics*

Many algorithmic tasks involve repeated utilization of some specifically specified operation. In addition, many tasks involve optimization computations. The following task embeds both elements.

> **Chocolate-bar breaking.** A chocolate-bar of N rows and M columns should be broken into its $N \times M$ squares. Every breaking operation is performed on a piece that is composed of at least two squares and results in two (not necessarily symmetric) pieces. Compute the minimal number of breaking operations required to obtain the $N \times M$ squares.

There are many possible paths for reaching the $N \times M$ squares. One may start by first breaking the chocolate bar into rows and then breaking each row into its squares; or, one may first yield columns, rather than rows; or, one may break pieces into symmetric parts; or perform some other combination of breaking operations. Some novices feel at loss from the many possible paths of breaking the bar. They view each possible path as

an entity, and devise an algorithm that compares the lengths of all these entities. They develop a recursive algorithm, which enumerates all the possible paths of breaking the bar, and computes the minimal path. This algorithm is exponential in time, and requires O($N \times M$) space. In addition, its design is error-prone, and novices struggle with various implementation errors.

The above solution encapsulates, again the notion of direct transformation, as it involves unfolding of all the possible entities (paths of breaking the bar), and comparing them to one another in order to find the better minimal one.

Yet, one does not need to examine all the possible entities. Careful look at a single breaking operation yields illuminating insight: every breaking operation adds exactly one additional piece to the total number of pieces, as an operation divides a single piece into two pieces. Thus, the total number of operations that will yield $N \times M$ squares is $(N \times M) - 1$; and this will occur with any path of breaking operations. That is, all the possible paths are of exactly the same length – $(N \times M) - 1$, and so is the length of the minimal path.

The above assertional insight illustrates the importance of carefully studying the characteristics of specified operations in algorithmic tasks. The need for such examination is apparent in a variety of algorithmic tasks, in particular in occasions where invariants are the key for insight (Dijkstra, 1976; Gries, 1981; Ginat, 2001), as invariants are derived from the characteristics of operators in algorithmic tasks.

2.3. *Missed Pattern of Modified Representation*

The following algorithmic task involves a searching computation. Searching computations (like counting and optimization computations) are essential in algorithmics.

> **Fence leveling.** A round fence is made of N columns, each composed of some arbitrary number of tiles. The total number of tiles in the fence is $N \times K$. One needs to level-off the top of the fence, so that each column will be composed of exactly K tiles. Leveling may be performed by transferring tiles from each column to its adjacent neighbor clockwise (column 1 follows column N, clockwise). Given the initial number of tiles in each column, compute the index of a column from which one may start leveling the fence and complete it in one round clockwise (there always is at least one such column).

As in the previous tasks, here too, novices apply the approach of direct transformation in solving this task. Their solution is based on checking each column as a starting point from which a single round may yield the leveled fence. Unfortunately, this computation is of time complexity O(N^2) and space complexity O(N) (as it involves repeated "passes" over the fence).

The above direct transformation solution is derived from repeated attempts to simulate the leveling activity. In unsuccessful attempts, one will "get stuck" at some point without enough tiles. In a successful attempt, one will have at least K tiles at hand upon reaching each column. Some tiles may be at the column a-priori, and some may have been transferred there from the previous column, by the leveling person. The leveling person will leave K tiles in that column, and transfer the rest to the next column clockwise.

The latter successful leveling scenario is indeed the one that a person who performs the leveling process should follow. But, is simulation of repeated attempts (each time starting from a different column) the better way to find the suitable starting point? Not necessarily.

The primary element to examine in the leveling process is the number of tiles transferred from one column to another. We denote it with the variable T. At first, the value of T is 0. At any stage of advancing along the fence clockwise it must be non-negative (i.e., 0 or more), as otherwise the leveling may not be completed successfully in one round.

Thus, if one starts the leveling process at a suitable column, the sequence of values of T will have 0 as its minimum. If one starts from an unsuitable point (from which she will "get stuck"), the sequence of values of T will include negative values, where negative values mean that tiles must be transferred counter clockwise. However, the two sequences will be very similar. The only difference between them is that the values in the "bad" sequence (with negative values) will be all smaller by a constant from their corresponding values in the "good" sequence. The constant reduction will be according to the minimum value in the "bad" sequence. Yet, the location of the minimum will be exactly at the same location in both sequences.

For example, let the fence be made of 4 columns, each composed of 9, 7, 11, and 13 tiles respectively. The fence leveling will result with 10 tiles in each column. The suitable column from which to start is column 3. At the beginning T will be **0**; after leveling column 3, T will be **1** (and that one tile will be transferred clockwise to column 4); after leveling column 4, T will be **4**; after leveling column 1 (which follows column 4 clockwise), T will be **3**, and after leveling column 2, T will return to the value 0. When we look at that "good" sequence of values of T, before columns 3, 4, 1, 2, respectively, we obtain: **0 1 4 3**. If we start the leveling from (the unsuitable point of) column 2, we will obtain the sequence: $0\ \ -3\ \ -2\ \ 1$, before columns: 2, 3, 4, 1, respectively. If we shift the latter sequence cyclically by one position, so it will correspond to the sequence of T before columns 3, 4, 1, 2, respectively, we will obtain: $\mathbf{-3\ \ -2\ \ 1\ \ 0}$, which is analogous to the "good" sequence above, and has its minimum at the same location.

The above assertional observation implies that in order to find the starting point, we do not have to simulate a successful leveling process. We may actually start a "pseudo-process" from any point, while allowing negative values, and find the location where T its minimal. This location is the location of a suitable starting point. Thus, one may perform the necessary computation in O(N) time and O(1) space.

The key point in the above elegant solution is to depart from the original task representation, which does not allow negative values for T, and turn to a modified representation that allows such values. Such modification implies that the computation will not simulate a "legal" leveling process, but it will yield a "legal" starting point. The notion of selecting a suitable representation is essential for problem solving in any domain, including those of mathematics and computer science (e.g., Polya, 1957; Cormen *et al.*, 1991).

2.4. *Missed Pattern of Progression Metric*

Many algorithmic tasks involve ordering, or grouping of elements according to some criteria. The following task requires grouping into two separate groups.

Diplomat grouping. N diplomats attend a party. Each diplomat is friendly with most of the other diplomats, but may have up to three diplomats with whom she is not friendly. Given the list of (up to three) foes of each diplomat, divide the diplomats into two disjoint groups so that each diplomat will have at most one of her foes in her group (if such grouping is impossible, then notify it in a corresponding message).

The common solution scheme offered by novices is based on starting with all diplomats in one group, A, and then separating between foes by passing some to a second group B. Many first handle diplomats with three foes, and pass two of the foes to group B. They then handle diplomats with two foes, and pass one of them to group B. However, although this process ensures that group A will correspond to the requirement per group, this is not necessarily the case for group B. Some novices do not notice this deficiency. Others try to "correct" this deficiency by setting slightly different criteria for passing diplomats to group B. They also add the condition that if B does not correspond to the requirement per group, then there is no solution to the task for the given list of foes.

Some students start with empty A and B, and gradually construct these two groups by putting foes in different groups. However, both this solution and the solutions of gradually passing diplomats from group A to group B are erroneous. They do not always yield two valid groups that correspond to the task requirements.

Both types of solution schemes are rather straightforward, and involve either the construction of both groups "from scratch", or the construction of group B from group A. In a sense, both solution types enclose a natural way of group construction, and intuitively transform the task description into an operational scheme.

One essential motive is missing in all the above schemes, and that is the transition of a diplomat back-and-forth between groups. Such a phenomenon seems problematic at a first glance, and novices intuitively worry that it may yield a non-terminating process. Indeed, they do not embed such transitions in their schemes. Yet, their schemes are incorrect.

One may allow back-and-forth transitions without yielding an infinite process. The key point is to allow such transitions in a solution scheme in which one can show constant progress, which will end in the desired goal (of two valid groups). One should seek a suitable metric, with which to show such progress, in a suitable solution scheme.

We may initially divide the diplomats arbitrarily into two groups, and then, whenever a diplomat has two or three foes in her group, this diplomat will be passed to the other group. This may cause back-and-forth transitions of diplomats, but apparently not that many. The metric M, which will help us see the latter is the total number of rivalries inside both groups; that is, the total number of rivalries in group A plus the total number of rivalries in group B. Every transition of diplomats between the groups, according to the above scheme, reduces this number. The initial value for M is no more than $3(N-1)$. Since it is reduced by at least 1 with each transition, there will be a linear number of transitions until the process will end successfully. (Notice that the process may end with some positive value of M.)

The key point in the scheme above is the assertional observation about the metric M. The utilization of the metric M enables an arbitrary start (division into groups A and

B) and back-and-forth transitions. Notice that, in order to keep M decreasing, once a diplomat is found with too many foes in her group, this diplomat (and not her foes) is passed to the other group. This selection, of the one to pass, is perhaps opposite to the natural selection, which was chosen by the students, and embodies some notion of reversing (see also Subsection 2.1). The primary notion here, of metric utilization is a powerful notion that is most useful for showing termination of computer programs (Gries, 1981, Cormen *et al.*, 1991).

3. Conclusion

We illustrated with four very different algorithmic tasks novice tendencies to solve algorithmic tasks rather hastily, by directly transforming the description of the required computation into an operational scheme. The novices' solutions were not derived from insightful observations, but rather from some intuitive translation of entity descriptions in the specified tasks into a straightforward solution.

The four tasks involved counting, optimization, searching, and grouping of elements, all of which are common types of computations. The first two tasks were less involved and the latter two – more involved. Yet, novices demonstrate the direct transformation theme with tasks of all levels. In some tasks their solutions are correct, but (sometimes considerably) inefficient. In other tasks their solutions are erroneous.

The primary element that the novices lacked is insight into the tasks. Insight involves pattern illuminations on which one may capitalize in devising a better solution. A key notion in pattern illumination is observations, which are specified in an assertional manner, as illustrated in the previous section. The specified observations encapsulate patterns that may be repeatedly used in algorithmic problem solving. Patterns such as those underlying the elegant solutions in the previous section repeatedly appear in algorithmic problem solving, and are therefore useful not only for solving the particular presented tasks, but also as tools for future problem solving occasions.

Themes similar to that of direct transformation were observed in the domains of mathematics and science education. Hegarty *et al.* (1995) noticed a similar student behavior in solving arithmetic word problems. They distinguished between unsuccessful problem solvers, who employed a direct text translation strategy, and successful problem solvers, who used a problem model strategy. Briars and Larkin (1984) noticed a similar unfortunate behavior, which they called "compute first and think later". In a broader sense, Chi *et al.* (1988), Sternberg and Frensch (1991), and Schoenfeld (1992) observed that novices are more inclined to focus of operative, quantitative answers, whereas experts are primarily seeking qualitative understanding.

As a consequence of the above, we advocate that tutors should notice their students' problem solving behaviors and underline the importance of qualitative insight and pattern utilization. Student awareness of qualitative observations should be developed from the very early stages of their studies. Tutors should employ in class illustrations such as those above, and benefit twice. First, they may demonstrate (and possibly even let the students experience themselves) the undesired theme of direct transformation, which one may

hastily employ. Second, tutors may elaborate the essential role of patterns, in particular ones that can be repeatedly utilized in future occasions. In addition, tutors may take the opportunity to demonstrate and distinguish between the operational perspective of solution schemes and the assertional perspective of pattern formulation. These perspectives should "go" hand-in-hand during algorithm and program design (Dijkstra, 1976; Gries, 1981).

Future studies in this direction may reveal additional, related novice behaviors, of which tutors should be aware, and take advantage in their teaching, for elaborating their students' scientific competence.

References

Briars, D.J. and Larkin, J.H. (1984). An integrated model of skill in solving elementary word problems. *Cognition and Instruction*, **1**, 245–296.

Chi, M.T.H., Glaser, R. and Farr, M.J. (Eds.) (1988). *The Nature of Expertise*. Erlbaum.

Cormen, T.H., Leiserson, C.E. and Rivest, R.L. (1991). *Introduction to Algorithms*. MIT Press.

Dijkstra, E.W. (1976). *A Discipline of Programming*. Prentice-Hall.

Dijkstra, E.W. *et al*. (1989). A debate on teaching computing science. *Communications of the ACM*, **32**(12), 1397–1414.

Ginat, D. (2001). Loop invariants, exploration of regularities, and mathematical games. *International Journal of Mathematical Education in Science and Technology*, **32**(5), 635–651.

Ginat, D. and Armoni, M. (2006). Reversing: an essential heuristic in program and proof design. In *Proc of the 37th ACM Computer Science Symposium – SIGCSE 2006*, 469–473.

Gries, D. (1981). *The Science of Programming*. Springer-Verlag.

Hegarty, M., Mayer, R.E. and Monk, C.A. (1995). Comprehension of arithmetic word problems: a comparison of successful and unsuccessful problem solvers. *Journal of Educational Psychology*, **87**(1), 18–32.

Polya, G. (1957). *How To Solve It* (2nd ed.). Princeton University Press.

Sternberg, R.J. and Frensch, P.A. (Eds.) (1991). *Complex Problem Solving: Principles and Mechanisms*. Erlbaum.

Schoenfeld, A.H. (1992). Learning to think mathematically: problem solving, metacognition, and sense making in mathematics. In D.A. Grouws (Ed.), *Handbook of Research on Mathematics Teaching and Learning*. Macmillan, 334–370.

D. Ginat obtained his PhD in computer science from the University of Maryland, USA, while also collaborating with Robert Tarjan from Princeton University. His dissertation is in the domains of distributed algorithms and amortized analysis of algorithms. His main interest in the last decade is computer science and mathematics education. He is currently the head of the Computer Science Group, in the Science Education Department of Tel-Aviv University.

Pradedančiųjų nesėkmės gvildedant tiesioginės transformacijos klausimą

David GINAT

Daugeliu atveju aprašytosios algoritmų užduotys lemia intuityvių ir operatyvių schemų atsiradimą. Tačiau dažnai tokia schema nebūtinai yra efektyvi ir teisinga. Nepaisant to, tarp pradedančiųjų vyrauja tendencijos, naudojant šį būdą, kurti savo sprendimus ir nesigilinti į struktūras, kurios galėtų pasiūlyti efektyvesnių sprendimų. Šiame straipsnyje įvardijamos ir gvildenamos tiesioginės transformacijos, aiškinama gilaus modelio konstravimo svarba, kurią galima suvokti kaip bendrą problemą išsiaiškinant sąvoką.