Exposition As a Learning Tool to Enhance Thinking Skills in Computer Science and Mathematics

Sun B. Chung
Department of Quantitative Methods and Computer Science
University of St. Thomas
sbchung@stthomas.edu

Abstract

We illustrate the importance of *good* expositions and suggest ways to use them to enhance thinking skills of students in computer science and mathematics.

First, we present our experience of using expositions to help children understand simple mathematical concepts and formulas. We used visual aids and sometimes constructed them with the children. We found the expositions accompanied by such activities very helpful.

Second, we present our experience of using expositions to guide college students in the thinking process of understanding problems and constructing computer algorithms to solve them.

Third, we introduce the reader to a data structure called *suffix trees*. Even though suffix trees allow efficient solutions to a wide range of complex string problems, the lack of good expositions kept them from being taught in mainstream computer science education [7].

Finally, we suggest ways to involve students in critical reading and writing of expositions to enhance their thinking skills.

Introduction

Exposition is "the act of presenting, explaining, or expounding facts or ideas." In this paper, we discuss exposition in two respects. First, we illustrate the importance of providing students with *good* expositions to help them understand mathematical concepts and computer algorithms. Second, we suggest ways to involve students in critical reading and writing of expositions to enhance their thinking skills. In addition, we discuss ways to use Internet resources in the classroom.

We begin with a story related to the summation formula $n(n+1)/2$. It is said that Gauss was less than ten years old when he noted the regularity in the sequence of positive integers 1 through 100 and computed their sum instantly. It is interesting to note that the method Gauss used to compute the sum has even been called Gauss's method [5], even though the formula was known before Gauss's time. What is more important is that the method has since been widely used in *expositions* of the formula $n(n+1)/2$ for the sum of positive integers 1 through n.

In this paper, we present expositions of mathematical formulas that are similar to the above formula in nature. Based on our experience of teaching young children, we describe how such expositions can help them understand the regularities that are inherent in certain sequences of numbers. We used visual aids and sometimes constructed them with the children. In later stages, when children had a good understanding of a formula, we encouraged them to visualize problems and their solutions without constructing physical visual aids. We found the expositions accompanied by such activities very helpful.

We then present an exposition of an algorithm for constructing a match schedule for a league. We describe our experience of involving college students in our introductory programming courses in a guided thinking process as they follow the exposition. The exposition was designed not to present a ready-made algorithm but to encourage students to discover it on their own.

To illustrate the importance of good expositions in computer science, we discuss the history of algorithms for constructing a data structure called *suffix trees*. According to Gusfield [7], there is "no other single data structure … that allows efficient solutions to such a wide range of complex string problems." The first linear-time algorithm of Weiner [11] was so fascinating that Knuth called it "the algorithm of 1973." Unfortunately, however, it earned a reputation of being extremely difficult to understand. For the next twenty years, the lack of good expositions kept suffix trees from being taught in mainstream computer science education [7]. We discuss Gusfield's exposition (1997) [7] of Ukkonen's algorithm (1995) [10] for constructing suffix trees.

Critical reading and writing of expositions can be a learning tool for students to enhance thinking skills. We suggest ways to use Internet resources to involve students in critiquing existing expositions as well as writing up an exposition on their own.

Expositions of Mathematical Concepts

In this section, we present our experience of using expositions when teaching simple mathematical formulas and concepts to children. Typically the expositions were accompanied by visual aids. We sometimes worked with children to construct visual aids. Once they were familiar with a formula, we sometimes encouraged them to visualize images related to the formula instead of constructing a physical image. We did these activities when we were teaching small children who knew how to do multiplication. We found the expositions and visual aids very helpful.

Disclaimer: We do not claim authorship of any of the expositions or visual aids discussed in this paper. For example, a figure similar to Fig. 2 is found in G. Polya [9] and possibly in many other books.

Gauss's Method

Problem 1: Compute the sum of integers $1 + 2 + \ldots + 99 + 100$.

The following is a famous story about Carl Friedrich Gauss.

> When Gauss was less than ten years old, his teacher told his class to compute the sum of integers *1* through *100*, hoping that it would keep the students busy for the next couple of hours. Gauss, however, computed the sum *5,050* instantly by noting a regularity in the sequence of the one hundred integers, which he described as follows: pair *1* and *100*, *2* and *99*, and so on, and there are fifty such pairs of *101* each.

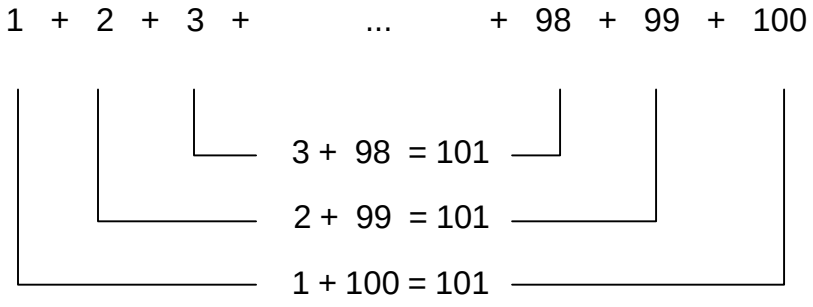


Figure 1: Visual aid for the sum of positive integers *1* through *100*

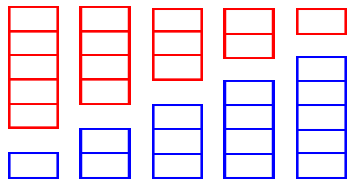
It is interesting to note that the method Gauss used to compute the sum has even been called Gauss's method or algorithm [5], even though the formula was known before Gauss's time. It is also interesting to note that, even though many writers assume that the above particular problem was exactly what Gauss and his classmates were given to solve

[4, 5, 6], it may have been a much more difficult sequence of numbers with the a fixed gap, such as *81297 + 81495 + 81693 + … + 100899* (one hundred numbers with a fixed gap of 198 between them) [3].

What is more important, however, is that the method Gauss used has since been used widely in *expositions* of the formula *n(n+1)/2* for the sum of positive integers *1* through *n*. It is also typical that the expositions are accompanied by a visual aid such as the one in Fig. 1.

Why has it been so widely used? In our opinions, it is because it makes an exposition a *good* one. It captures the regularity that is inherent in the sequence of consecutive integers and presents it very clearly to the reader. Once students understand the regularity, it is easy for them to remember the formula. Even when they forget the formula, they can easily reconstruct it. In addition, the name of a child prodigy works well to draw students' attention and curiosity. The visual aid seems to appeal, too.

An alternative way is to use bricks or squares made of construction paper. For example, Fig. 2 illustrates how the sum of integers *1* through *5* (shown in blue) can be computed in a different way. The idea is to put a copy of the layout of blue bricks above it rotated *180* degrees (shown in red). Each column sums up to *6*, and there are *5* columns but, since the total number of bricks, *6 * 5*, is exactly two times the answer, it is divided by *2*, and we obtain the answer *15*.



(6 + 6 + 6 + 6 + 6) / 2 = (6 * 5) / 2 = 15

Figure 2: Alternative visual aid for problem 1:
In particular, compute the sum of positive integers *1* through *5*.

General and Special Cases of Problem 1

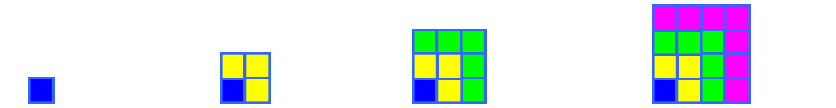
It is quite common to generalize the above problem to the case where the sequence of integers does not begin with *1*. For example, what is the sum of integers *3* through *7*? Also, it is common to generalize it further and allow a fixed gap between the integers in the sequence. For example, what is the sum of even integers *4*, *6*, *8*, *10*, and *12*?

Problem 2 (a special case of the general case of Problem 1): A special case is to compute the sum of *odd* positive integers *1* through *n*. The following regularity is known. That is, the sum is always a square of an integer.

$$
\begin{aligned}
1 + 3 &= 4 = 2^2 \\
1 + 3 + 5 &= 9 = 3^2 \\
1 + 3 + 5 + 7 &= 16 = 4^2 \\
1 + 3 + 5 + 7 + 9 &= 25 = 5^2 \\
1 + 3 + 5 + 7 + 9 + 11 &= 36 = 6^2
\end{aligned}
$$

Even though we may use the visual aid of Fig. 2 for this special case, it doesn't seem to help explain why the sums are squares. That is, it fails to capture the above regularity.

When we presented an exposition of this regularity to students, we found a visual aid using bricks such as Fig. 3 useful. An exposition with such a visual aid does capture the above regularity.



$1 = 1^2$ $\quad 1 + 3 = 2^2$ $\quad 1 + 3 + 5 = 3^2$ $\quad 1 + 3 + 5 + 7 = 4^2$

Figure 3: Visual Aid for Problem 2:
The sum of odd positive integers *1* through *n* is a square of an integer.

Fig. 3 may also be used to provide a good exposition of the following problem.

Problem 3: Why do the gaps between the following sequence of squares of integers,

$$1^2, \quad 2^2, \quad 3^2, \quad 4^2, \quad 5^2, \quad 6^2, \quad \ldots$$

form a sequence of odd integers as follows?

$$3, \quad 5, \quad 7, \quad 9, \quad 11, \ \ldots$$

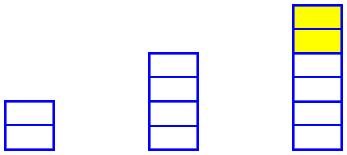
G. Polya [9] discusses interesting problems related to the above Problem 3. For example, why is $1^3 + 2^3 + 3^3 + 4^3 = 1 + 8 + 27 + 64 = 10^2$?

We can also think of special cases in which there are just a few numbers in the sequence.

Problem 4 (another special case): Compute the sum of a few integers in a sequence with a fixed gap. For example, what is the sum of integers *2*, *4*, and *6*, with a fixed gap of *2* between the integers?

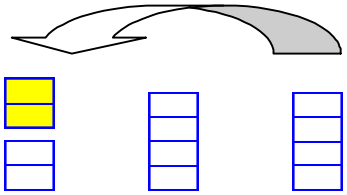
If there are an odd number of elements in the sequence, we can simply multiply the middle element by the number of elements in the sequence. In such a case, the visual aid such as the one illustrated in Fig. 4(a) and Fig. 4(b) may be helpful. Fig. 4(a) shows the problem. The solution shown in Fig. 4(b) uses the idea of lending two bricks of the last

column to the first, thus making the number of bricks of all columns the same as the middle one.



2 + 4 + 6 = ?

Figure 4(a): Visual aid for problem 4 (special case)



4 + 4 + 4 = (4 * 3) = 12

Figure 4(b): Visual aid (continued) for the solution of problem 4

Once children were familiar with the idea, we encouraged them to visualize *in their minds* the columns of bricks and the process of lending bricks to the first column, without working with bricks.

For the case where there are an even number of elements in the sequence, we can use the visual aid of Fig. 2 that adds a rotated copy of the columns of bricks.

Examples of Unifying Algebraic and Geometric Reasoning

Problem 5: Explain the following equation: $(x + y)^2 = x^2 + 2xy + y^2$.

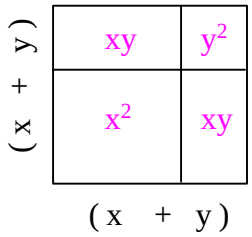


Figure 5: Visual aid for problem 5

For this problem, there is a known exposition that uses the visual aid of Fig. 5. We encouraged children to visualize similar images in their minds to compute $(10 + 1)^2$, $(20 + 1)^2$, etc., without constructing a physical layout for each case.

Problem 6: Explain the following equation: $(x – y)^2 = x^2 – 2xy + y^2$.

A drawing similar to Fig. 5 can be used to present a good exposition of this equation. In particular, it helps to explain why y^2 must be added.

In applying this formula, too, we encouraged children to visualize similar images in their minds to compute $(10 –1)^2$, $(20 –1)^2$, etc., without constructing a physical layout.

Children were especially fascinated when they computed $(10 + 5)^2$ and $(20 –5)^2$ separately and noted that the answers were the same!

Scheduling a League: a Problem for Computer Programming

We have used the following problem in introductory programming classes to engage students in a nontrivial thinking process. We believe that the process of thinking as a group, guided by the instructor, has provided our students with an opportunity to appreciate a good exposition of the problem. We also believe that the graphs used in the process of developing an algorithm for this particular problem illustrate that good visual aids are helpful in presenting a good exposition.

Scheduling a League: the Problem Description and a Simple Algorithm

Problem 7 (scheduling a league): Suppose you are commissioned to write a computer program to schedule a league (not a tournament!) of tennis games. By a *league*, we mean that each player plays with all other players exactly once. Given *n* tennis players, the problem is to determine three things: the number of tennis courts needed, the number of rounds needed, and a table containing the schedule of matches for each round. For example, when $n = 4$, the following is a sample output of the program (where *1*, *2*, *3*, and *4* in the table represent the player id numbers).

(a) 2 courts

(b) 3 rounds

(c) TABLE:

[Round 1] (1, 2), (3, 4)
[Round 2] (1, 3), (2, 4)
[Round 3] (1, 4), (2, 3)

For an odd number of players, there is a simple algorithm illustrated in Fig. 6(a) and Fig. 6(b). Fig. 6(a) sets up the problem. Fig. 6(b) illustrates the solution.

First, construct two sequences of player id numbers, one in increasing order (shown in blue) and the other in decreasing order (shown in red in Fig. 6(a)). Line them up in parallel. After each round, shift and wrap around the second sequence to the right, as indicated by player **5** in **bold** red.

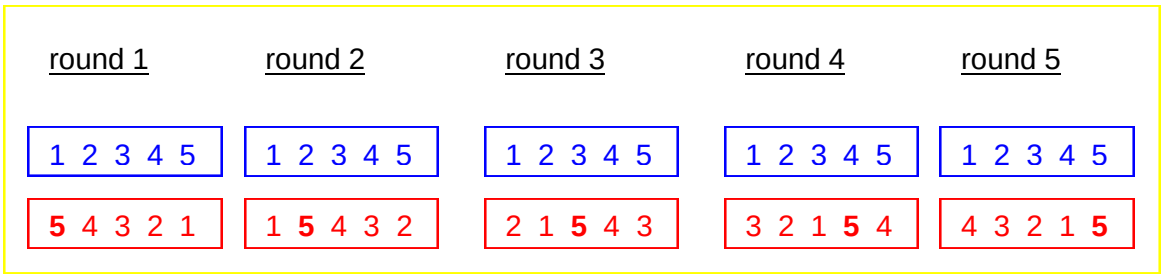


Figure 6(a): Visual aid for problem 7

In each round, proceed from left to right, and find a pair in which a player is paired with itself. That player will sit out in that round. The next two pairs immediately following that pair will have matches in that round. Fig. 6(b) illustrates the solution. For example, in round 1, player *3*, which is paired with itself and has a star above it, sits out. The next two pairs, *(4, 2)* and *(5, 1)*, have matches in round 1. It may be necessary to wrap around. Consider round 3. Player *4* sits out. The next two pairs are *(5, 3)* and, as we wrap around, *(1, 2)*.

When we are done with all five rounds in this manner, it is guaranteed that there are no duplicate matches. It is guaranteed that there are no matches that are missed.

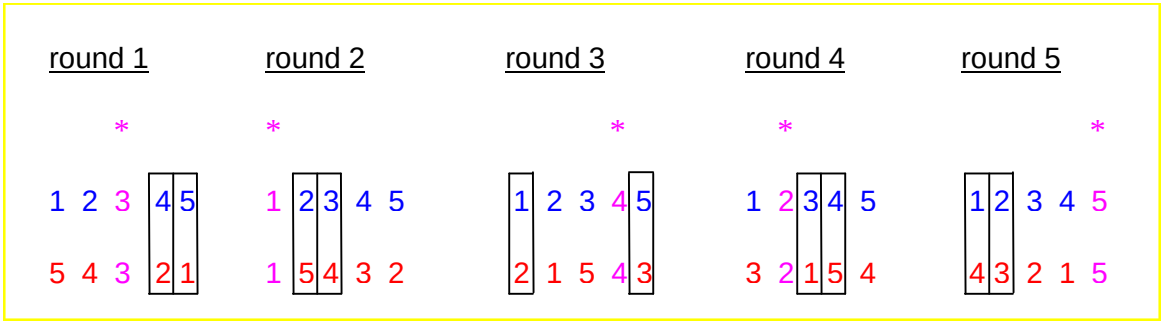


Figure 6(b): Visual aid illustrating a solution for problem 7

The above algorithm has shortcomings. First, it doesn't work for an even number of players. Second, it is not immediately clear why it is guaranteed to produce the correct table, without duplicate or missing matches.

An Exposition Using a Graph

We present an alternative algorithm. It has the following properties. First, it works for both even and odd number of players. Second, the solution for the case of even number of players is closely related to the case of odd number of players. Third, for the case of odd number of players, it turns out to be equivalent to the above algorithm, when the rounds of the above algorithm are ordered differently. Most importantly, however, the merit of our exposition of this algorithm is that, it shows, clearly and immediately, why it produces a correct table without duplicate or missing matches.

Instead of presenting the algorithm, we engage our students in a guided thinking process, in which they take the following steps.

A. Observe the case of three players and note a regularity.

B. For the case of four players, find a way to produce a solution based on the solution for the case of three players.

C. For the case of five players, note the same regularity as the one observed in the case of three players, and generalize it to *any* odd number of players.

D. For *any* even number n of players, make a generalization that a solution can be found based on the solution for $n - 1$ players.

Thinking Step A

First, students are encouraged to find a regularity by making an observation. They are encouraged to have a discussion in groups. The regularity is that, for three players, only two players can have a match in a round, as illustrated in Fig. 7. In other words, in each round there is always a player who sits out. Therefore, only one court is needed. The number of rounds is equal to the number of players, for the same reason.

In each round, we will systematically let the id of the player who sits out be equal to the round number (Fig. 7).

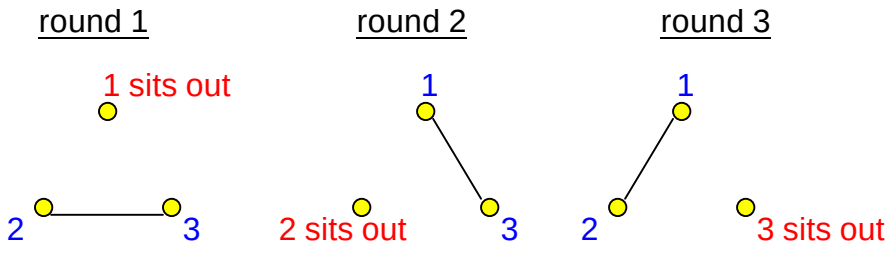


Figure 7: In round k, the player with id number k sits out.

Thinking Step B

For four players, think about the problem as follows. To the case of three players, a fourth player is added. Since, in each round, there is a player who sits out, the fourth player has only to play with him/her, as illustrated in Fig. 8. Now one more court is needed than the case of three players. However, the number of rounds required remains three.

Clearly, the solution for the case of four players can be obtained from the solution for the case of three players. The solutions for the two cases are related.

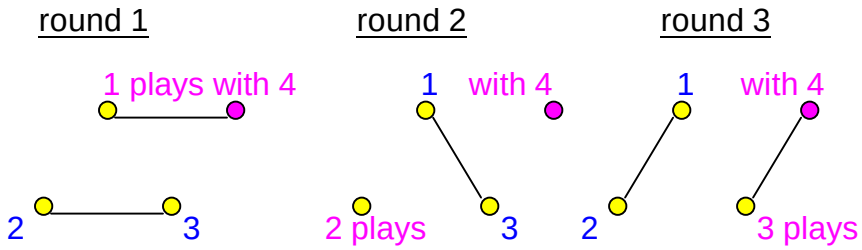


Figure 8: The fourth player plays with the player who would sit out in each round in the case of three players.

Thinking Step C

Before moving on to the case of five players, we need to familiarize students with some basic graph theory.

> A league of *n* players can be represented with a graph of *n* nodes. The nodes represent players. An edge between two nodes represents a match. A complete graph is a graph in which there is an edge between each possible pair of nodes. All the required matches of a league can be represented by the edges of a complete graph (even though the complete graph does not present a schedule we want). A complete graph of *n* nodes has $n(n-1)/2$ edges.

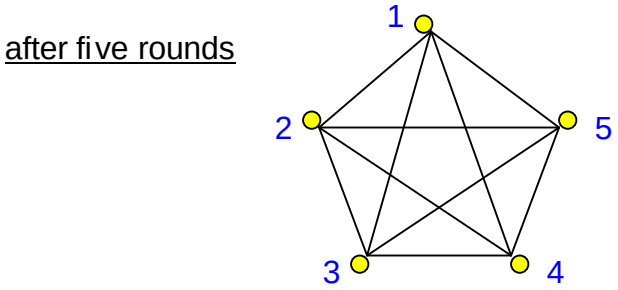


Figure 9: A complete graph constructed when

all the matches of five rounds are represented by edges.

Constructing a correct match schedule, therefore, amounts to systematically constructing a complete graph, without duplicate or missing edges (Fig. 9 for five players).

For the case of five players, the same regularity as the one observed in the case of three players is observed. That is, the players must take turn sitting out, with exactly one player sitting out in each round. It can easily be generalized to *any* odd number of players. As for the case of three players, we will *systematically* let the id of the player who sits out be equal to the round number.

Now, the tricky part is how to *systematically* pair the other players in each round. The process of pairing players should not be arbitrary for each round but must be systematically identical for all rounds. In addition, the algorithm for the case of five players should be designed in such a way that it works for *any* odd number of players. Students are encouraged to think about it, constructing visual aids using a graph with five nodes.

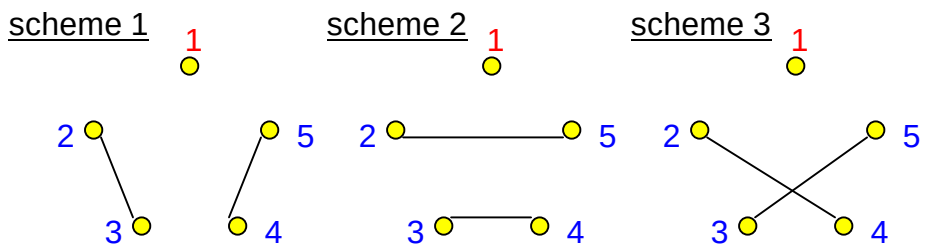


Figure 10: Three different schemes to pair players *2*, *3*, *4*, and *5* systematically. The scheme chosen should works for *any* odd number of players.

Thinking Step D

For any even number *n* of players, students make the following generalization: a solution can be found based on the solution for *n – 1* players.

The Tricky Part Revisited

After students have given enough thought to the tricky part mentioned in the Thinking Step C, they are sure to come up with a systematic way of pairing players that works for the case of five players. That is because there are only three systematic ways to pair players, as illustrated in the above Fig. 10, and students realize that one and only one of them works, as illustrated in Fig. 11.

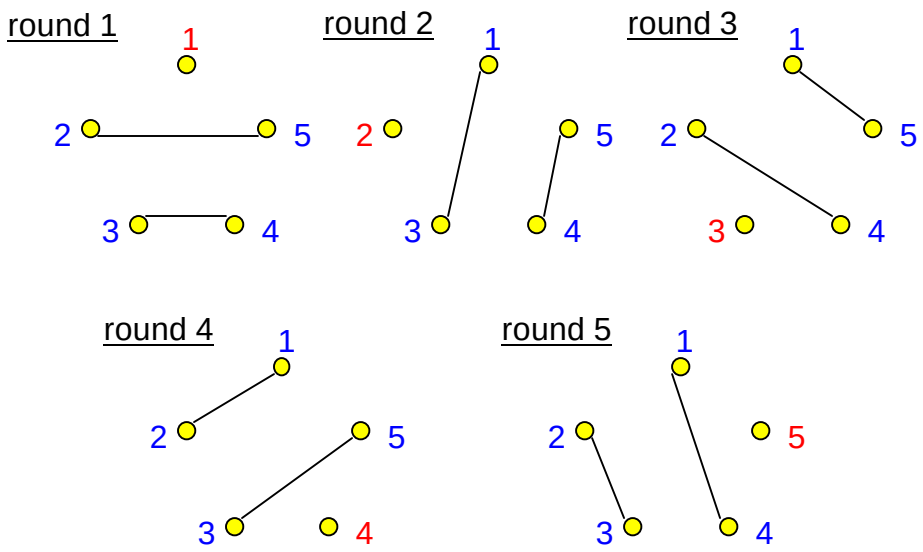


Figure 11: A systematic scheme to pair players for the case of five players

Generalization to Any Odd n > 5

Once students are convinced about the systematic way for the case of five players, they easily generalize it to any odd number greater than five and produce computer programs based on the above algorithm. (When they see their programs work, they tend to think that *obviously* their programs are correct, but we warn them, with the words of Bertrand Russell, "Obviousness is always the enemy of correctness.")

We leave it to the reader to make the necessary generalization for her/himself.

We believe that the above exposition revealed the common properties shared by the scheduling problem and the systematic production of edges of a complete graph with an odd number of nodes. Without proof, we assert that it helped our students have a better understanding of the scheduling problem and also helped them in the process of constructing a computer algorithm. In addition, we believe that, by engaging our students in a guided thinking process, we stimulated their thinking skills.

Suffix Trees

To illustrate the importance of *good* expositions in computer science, we briefly discuss the history of algorithms for constructing a data structure called *suffix trees*. According to Gusfield [7], there is "no other single data structure … that allows efficient solutions to such a wide range of complex string problems." The first linear-time algorithm was given by Weiner in 1973 [11] (and it was followed by McCreight's linear-time algorithm in 1976 [8]). Weiner's algorithm was so fascinating that Knuth called it "the algorithm of

1973." Unfortunately, however, it earned a reputation of being extremely difficult to understand. For the next twenty years, the lack of good expositions kept suffix trees from being taught in mainstream computer science education [7].

In the following paragraphs, we briefly describe Gusfield's stepwise exposition (1997) [7] of Ukkonen's algorithm (1995) [10] for constructing suffix trees. Since we will focus on how Gusfield presents his exposition in a stepwise manner, those who are interested in the contents of the exposition or Ukkonen's algorithm itself are referred to [7, 10].

First, Gusfield presents a naïve implementation that runs in $O(m^3)$ time, where m is the length of the text in which a pattern of length n is searched for. It is even worse than a known naïve algorithm that runs in $O(m^2)$ time!

Next, several improvements are made. They include improvement on the use of space. The first of such efforts is to introduce what are called *suffix links*, even though the running time after their introduction is still the same $O(m^3)$ time. However, it has imposed sufficient structure on the process of creating suffix trees that, by using three "tricks", we can make the algorithm run in linear time.

The first trick is called "skip/count" trick. After it is introduced, the running time is reduced to $O(m^2)$ time. Before the second trick is introduced, a simple tweak reduces the space requirement to linear space. This is important because, when the space requirement is greater than $O(m)$, it is not possible for us to achieve $O(m)$ time.

The second trick, working on a "show stopper" case, allows execution of a phase to end with no further action when one of the extension rules (rule 3) applies. The third trick is based on the observation: "Once a leaf, always a leaf." It achieves further reduction of explicit work by allowing it to be done implicitly. With the second and third tricks in action, the running time is reduced to $O(m)$.

At the end, Gusfield uses a simple cartoon to illustrate why the algorithm runs in linear time (Fig. 6.9, p. 106 of [7]). The point is that "In any two consecutive phases, there is at most one place where the same explicit extension is executed in both phases." Therefore, the total work of all phases combined is linear.

After having gone through Gusfield's exposition, and especially after having a look at the simple cartoon at the end, one may think that it is more complex than necessary to begin with an implementation whose running time is even worse than a known naïve algorithm and then to introduce all the tricks to reduce it to linear time. However, the stepwise exposition seems to have gained a reputation as a *good* exposition from a pedagogical point of view. We hope that it helps suffix trees to be more easily understandable and more accessible in the science community and to be used for the wide range of applications.

Conclusion

In this paper, we tried to illustrate the importance of providing students with *good* expositions. Without proof, we made the following assertions. Good expositions, typically with visual aids, were effective when we were teaching simple mathematical concepts and formulas to children. We also found them effective when we tried to engage students in our introductory programming classes in a guided thinking process. We believe that good expositions can be used to enhance the thinking skills of students, young and old.

In addition, critical reading and writing of expositions can be a learning tool for students to enhance thinking skills. These days numerous expositions of the same topic can be found on the Internet by keyword search. It may be a good mental exercise for students to compare them and write up a critique.

Students can also be motivated to write their own expositions and to post them on the Internet. They can be motivated to add visual aids or animation to simulate execution of algorithms, as noted in many of the expositions of various topics found on the Internet.

References

1. Anno, M. (1987). *Anno's Math Games*. New York: Philomel Books.

2. Anno, M. and M. Anno. (1999). *Anno's Mysterious Multiplying Jar*. New York: Penguin Putnam Books.

3. Bell, E, T. (1965). *Men of Mathematics*. New York: Simon & Schuster.

4. Boyer, C, B. (1968). *A History of Mathematics*. Princeton: Princeton University Press.

5. Cooper, D. and M. Clancy (1985). *Oh! Pascal!* New York: W. W. Norton & Company.

6. Eves, H. (1983). *An Introduction to the History of Mathematics*. New York: Saunders College Publishing.

7. Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. New York: Cambridge University Press.

8. McCreight, E, M. (1976). A space-economical suffix tree construction algorithm. *J. ACM,* 23:262-72.

9. Polya, G. (1945). *How to Solve It*. Princeton: Princeton University Press.

10. Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica,* 14:249-60.

11. Weiner, P. (1998). Linear pattern matching algorithms. *Proc. Of the 14th IEEE Symp. on Switching and Automata Theory*, pp. 1 – 11.

Acknowledgements

We would like to thank Robert Raymond for referring us to the works of Masaichiro and Mitsumasa Anno [1, 2] and encouraging us to work with children.