Teaching NP completeness in secondary schools

Seungyon Kim and Seongbin Park*

Korea University, Seoul, Korea
{seungyon_kim, hyperspace}@korea.ac.kr

Abstract. In this paper, we show how the theory of NP completeness can be introduced to students in secondary schools. The motivation of this research is that although there are difficult issues that require technical backgrounds, students are already familiar with demanding computational problems through games such as Sudoku or Tetris. Our intention is to bring together important concepts in the theory of NP completeness in such a way that students in secondary schools can easily understand them. This is part of our ongoing research about how to teach fundamental issues in Computer Science in secondary schools. We discuss what needs to be taught in which sequence in order to introduce ideas behind NP completeness to students without technical backgrounds.

1 Introduction

Informatics education in secondary schools typically includes programming as a way to solve computational problems. A programming course can consist of lectures on object-oriented languages such as Java or procedural languages such as C. Once students learn how to program, they may ask whether all computational problems are solvable or whether there exist more efficient solutions (or programs) for given problems than their own programs. These are issues of computability or computational complexity and it is difficult to teach these issues in secondary school level informatics classes because there are many technical terms that students need to understand beforehand.

In this paper, we propose a way by which NP completeness and other relevant issues can be taught to secondary school students who understand what algorithms are. We also discuss why teaching NP completeness is important in secondary school informatics classes. The motivation of this research is as follows. First of all, many students are familiar with games such as Tetris or Sudoku, but few students are aware of the fact that they are NP-complete problems [1, 2]. If they cannot find correct solutions for Sudoku and they do not know of the existence of NP-complete problems, they may just think the reason is that it is difficult to solve. In other words, instead of seeing the structural properties of the problem, they may simply think that it is difficult to solve. However, if we introduce the idea of NP completeness to students, they can see the problems from a general perspective and can understand why some specific instances of Tetris or Sudoku can be hard to solve. Secondly, students who can solve quadratic equations of the form $ax^2 + bx + c = d$ can easily solve linear equations of the form $ex + d = f$, but they may not know that similar tranformations (or reductions) exist among different

* Corresponding author

computational problems as well. If students understand reductions used in NP completeness, they can classify problems that they know and possibly learn how to analyze an arbitrary computational problem.

2 What to teach

In this section, we explain how we can introduce the concept of NP completeness to secondary school students. To this end, we identify candidate topics that most students at secondary schools are familiar with. These can serve as starting points to learn the concept of NP completeness.

Typical topics that secondary school students learn include graphs of inequalities, number sequences, quadratic equations etc. When students learn these topics, they study how to solve different types of problems, but often they are not aware of whether they deal with problem instances or the problem itself. For example, when students learn quadratic equations, they study what quadratic equations are, how solutions for these equations could be found, and they work with some problem instances so that they feel they understand how to solve arbitrary quadratic equations. However, since NP completeness is a property of decision problems and it is important to understand the relationship among the set of problems which are NP-complete, we need to teach students how to relate one problem to another. This requires a clear understanding of the difference between a computational problem and its instances because a transformation between two computational problems can be defined in terms of their instances.

Once students understand the difference between a problem and a problem instance, they can understand the role of an algorithm precisely; i.e., an algorithm is a well-defined computational procedure [3] which takes the input of a problem (i.e. a problem instance) and returns a correct output in a finite number of steps. At this point we can explain why a computational problem that asks whether a program, downloaded by a student from the Internet, is a computer virus or not, is not computable [4] by mentioning that there is no consistent mechanism (i.e. algorithm) which can always give a correct answer for an aribitrary problem instance. We can even introduce a 3 CNF Satisfiability problem which exhibits a phase transition as the ratio of the number of clauses to the number of variables varies [5]. Since students are already familiar with the notion of phase transition from chemistry class, this phenomenon may sound exciting and they may be motivated.

To introduce the concept of reductions used in NP completeness, we can point out that two computational problems that look similar on the surface may have different structural properties. This can be explained using Hamiltonian cycle problem and Eulerian cycle problem [6]. We can mention that their structures are different because they ask to satisfy different constraints; i.e., one asks to visit every vertex once and the other asks to visit every edge once. This can naturally lead to a dicussion about a class of problems that share certain common denominators in terms of their structures. For example, we can use a diagram about the web of reduction [7] which shows how different problems can be connected in terms of a special relation, called reducibility. From the figure, students can see some connection between the Satisfiabilility problem and the Hamiltonian cycle problem. Though it may be difficult to understand how the reduction

really works, students can at least realize that constraints existing in the Satisfiability problem are structurally similar to those in the Hamiltonian cycle problem. As a simple example of reduction, we can explain the Clique problem and show that a satisfying assignment for the Satisfiabilility problem can get us the information about a clique in a graph that is created based on the constraints in the Satisfiabilility problem.

To introduce properties shared by problems in NP, we can first explain problems that belong to a class called P. This can be done by recalling that the running time of an algorithm is typically given as a function of the size of input for the problem under consideration [8]. Students are familar with how to count the number of steps of an algorithm that can be represented as a flowchart. We then point out that certain decision problems admit of efficient solutions which means they can be solvable in time proportional to some polynomial function of input size. We can also mention that there are decision problems whose solutions can be efficiently verified once we are given candidate solutions, but it is not proved yet whether they also admit efficient solutions. At this point, we can define class P as the set of decision problems which admit of efficient solutions and class NP as the set of decision problems which admit of efficient verification.

Once students understand properties of problems in P and NP, we can continue explaining a different way to define NP. That is, we explain that a decision problem belongs to NP if there is a nondeterministic algorithm which solves the problem efficiently [9]. But in order to understand this definition, students need to understand what a nondeterministic algorithm is. We can explain the concept by showing how a two-phase algorithm that consists of guessing and verifying solves a graph coloring problem [10]. We can also point out that an algorithm which we talk about in general is essentially a special type of nondeterministic algorithm that does not contain the first guessing step. In addition, we can explain that it is the reason why P is included in NP; i.e., a deterministic algorithm is a special type of a nondeterministic algorithm.

3 Conclusions

NP completeness is an important topic in the theory of computations and other areas in computer science. Understanding NP completeness is difficult because it requires a clear understanding of various terminologies and mathematical backgrounds. In this paper, we argue that the basic idea of NP completeness can be taught at seconday school. Our expectation is that if students learn the ideas behind NP completeness, they can broaden their horizons with regard to the set of computable problems and problem solving techniques.

References

1. Breukelaar, R., Hoogeboom, H. J., Kosters, W. A.: Tetris is hard, made easy, Technical report, Leiden Institute of Advanced Computer Science, Universiteit Leiden (2003)
2. Aaronson, L.: Sudoku Science, IEEE Spectrum (2005)
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to Algorithms, second edition, The MIT Press

4. Lowther, J. L., Shene, C. K.: Toward an Intuitive and Interesting Theory Course: The First Step of a Road Map, Journal of Computing Sciences in Colleges (2004)
5. Hayes, B.: Can't Get No Satisfaction, American Scientist (1997)
6. Hayes, B.: Accidental Algorithms, American Scientist (2008)
7. Arora, S., Barak, B.: Computational Complexity: A modern approach, Cambridge University Press (2009)
8. Hromkovič, J.: Algorithmic Adventures From Knowledge to Magic, Springer Berlin Heidelberg (2009)
9. http://en.wikipedia.org/wiki/NP_(complexity) (last checked: 1/31/2013)
10. Basse, S., Van Gelder, A.: Computer Algorithms, Addison Wesley (2000)