Learning Guide Unit 7

Site: University of the People
Course: CS 3304-01 Analysis of Algorithms - AY2024-T5
Book: Learning Guide Unit 7

Printed by: Mejbaul Mubin
Date: Saturday, 29 June 2024, 6:44 AM

Description

Learning Guide Unit 7

Table of contents

Overview

Unit 7: Limits to Computation (Part 1)

Topics:

- Hard Problems
- P, NP and Co-NP Classes
- Reductions

Learning Objectives:

By the end of this Unit, students will be able to:

1. Recognize and be able to define the differences between the P, NP, and Co-NP classes
2. Recognize and be able to describe reductions and how they are used in determining if a problem is NP hard or NP complete
3. Define what a 'hard' problem is from the perspective of algorithm analysis
4. Identify problems that are NP complete

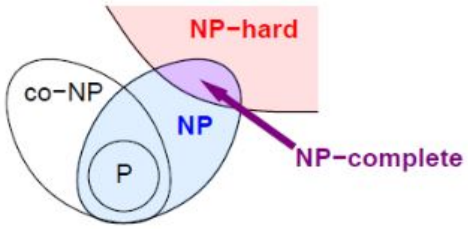
Tasks:

- Read the Learning Guide and Reading Assignments
- Participate in the Discussion Assignment (post, comment, and rate in the Discussion Forum)
- Make entries to the Learning Journal
- Take the Self-Quiz

Introduction

Unit seven introduces important and advanced concepts in algorithm analysis. In unit seven we get a definition of problems that are 'hard' to solve and those that are considered to be impossible.

Problem space
The best way to begin looking at the theory of NP completeness is to define some terms. First of all think of all of the problems that are computable as being a part of the problem space. All of these problems form a set of problems within this larger set of problems there are subsets. One of the subsets includes all of the problems that can be solved in Polynomial time. We will call this set to be P.



More of what we *think* the world looks like.

P is the set of problems that can be solved in polynomial time. You will notice that the entire set of P is actually contained within NP. A simple way of thinking about this is that NP is the set of Problems … some of which can be solved in Polynomial time. The distinction between NP and co-NP comes from the original use of NP and P nomenclature as a way of defining the complexity of decision problems. NP was used to indicate where a YES answer could be found and co-NP was the opposite indicating that a NO answer could be found.

The definition that we will use for the moment is to consider NP as the set of all problems that we will need to solve and the subset of P are those problems that can be solved within Polynomial time.

We know that not all problems can be solved in polynomial time and we further understand that some problems can be difficult if not impossible solve as the algorithms are inefficient and in some cases require exponential time. As the size of n (or the size of the input to problem) becomes large the amount of time required to solve the problem becomes exponentially larger.

As the size of n approaches infinity, there may be cases that are impossible to solve. We can clearly see that there are some problems that could easily be construed as 'hard' problems to solve because of the time required to do so.

Defining what is hard is a bit tricky. So called NP-hard problems are considered to be hard because no one has yet to find an easy solution. We explored the idea of reductions a little bit in the course and we know that a reduction is simply a way of taking one problem and attempting to solve it with another known algorithm.

Many of you will recall the concept of the Turning machine which is a simplified concept of a computer. The Turing machine had the concept of a ribbon upon which there was data and that machine could only read or write one bit of data to or from the tape and of course it could move the tape left or right. That is a pretty simple instruction set. Researchers have found this to be the ultimate test of computability because so far there hasn't been any algorithm or computing problem that couldn't be reduced to the level of the simplistic turing machine. Using the Turing machine to determine the 'computability' of any kind of problem is an example of a reduction.

The problem with NP-hard problems is that if a solution is found for ONE of these NP-hard problems then that solution would solve all NP-hard problems because any problem could be reduced to the solution that was found. The set of NP-hard problems is an entire set of problems.

The formal definition of NP is the set of all Non Deterministic Problems. Something that is non-deterministic is something that given some input could come up with different answers every time. Perhaps one way of describing the set of NP problems is that they MIGHT or COULD be solved in polynomial time it doesn't, however, imply or ensure that they will be.

This gets us to NP-Complete. NP-Complete is the set of NP problems (remember non-deterministic so it a bit of a crap shoot as to whether you can get the answer in polynomial time) that are in the subset formed by the intersection of NP-Hard and NP.

Reductions

A reduction is an approach that allows us to solve one problem in terms of another. One way that this can be used is in asymptotic analysis. If we know the asymptotic complexity of one problem but not another, the ability to redefine the problem in terms of the complexity known for another gives us the asymptotic complexity of the problem.

The example of this used in the text is a good one. Imagine if you would that you have two lists of numbers. Both lists have the same number of items in them and you wanted to pair the items. By pairing we mean that the smallest item in the first list must be paired with the smallest item in the second list, the next smallest in the first list would then need to be paired with the next smallest in the second list and so on.

First List	Second List
1	6
5	3
3	8
9	9
2	2
8	1

This is a pretty complicated process, but if we can 'TRANSFORM' it into another better known algorithm then we could determine its complexity. In this case if we simply sorted both lists, then the pairing is easy.

First List	Second List
1	1
2	2
3	3
5	6
8	8
9	9

By transforming the unknown problem pairing into known problem sorting, we now have a good idea of what the complexity of the pairing problem is. This concept of transforming an unknown problem into something that is known is the basis of reductions.

Hard Problems

For many of us, all of the problems that we are encountering are 'hard'. They may be difficult to understand or it might be difficult for us to develop an algorithm to solve them. However from the perspective of computer science, the definition of a 'hard' problem is one that is expensive in terms of running time with even the best algorithms.

In unit six we had an example of a 'brute force' algorithm for solving the knapsack problem that was 'hard'. The brute force algorithm required exponentially greater iterations to solve as the number of items that the thief could select from increased. In our example with only 16 items in the store, the algorithm required 65,536 combinations of knapsack's to find the optimal solution. A store that had 32 items would require that 4,294,967,296 combinations of knapsack's be examined to find the most optimal combination. Just imagine what it would take if the store had 1,000 or 10,000 items!

The knapsack problem is not really a 'Hard' problem because as we learned there are alternate and better algorithms that can be used to solve this problem that are not as expensive as the 'brute force' approach, but this example does provide an understanding of what hard means in computer science.

NP Completeness

NP completeness or the theory of NP completeness is an approach to 'hard' problems.

The most notable characteristic of NP-complete problems is that no fast solution to them is known; that is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows.

As a result, the time required to solve even moderately large versions of many of these problems quickly becomes so great that it isn't practical to solve them using current technology solutions.

Determining whether or not it is even possible to solve these problems quickly is one of the principal unsolved problems in Computer Science today.

While a method for computing the solutions to NP-complete problems using a reasonable amount of time remains undiscovered, computer scientists and programmers still frequently encounter NP-complete problems. It is important for programmers to be able to recognize an NP-complete problem so that they don't unknowingly waste time trying to solve a problem which so far has eluded generations of computer scientists.

The following url provides a humorous look at NP-complete problems. http://xkcd.com/399/

One approach that is taken to come up with a solution to NP-complete problems is to use techniques of approximation. If you recall the knapsack problem from unit 6, we had an example of a brute force approach that would look at every possible combination of items to put into a knapsack to get the 'optimal' contents which maximized the value of the items the thief would steal. In NP-complete problems this approach to finding an optimal solution is not practical so another approach might be to make an educated guess at the best solution and then test it to determine if in fact that it was the correct one.

Reading Assignment

Topic 1: Defining P, NP and Co-NP
Read “lecture 21: NP-Hard Problems”, sections 21.1 through 21.3. Download the pdf.

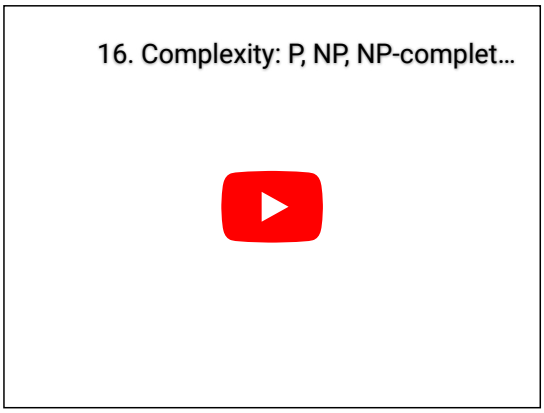
Topic 2: The Theory of NP-Complete
Chapter 17: Limits to Computation Section 17.1 in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.
Read “lecture 21: NP-Hard Problems”. Download the pdf.

Topic 3: Reductions and SAT
Chapter 17: Limits to Computation Section 17.1 in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.

Read “lecture 21: NP-Hard Problems”, sections 21.1 through 21.3. Download the pdf.

Chapter 8 NP Complete Problems in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap8.pdf

Reductions and NP-Completeness lecture notes available at



Alternate link: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-videos/lecture-16-complexity-p-np-np-completeness-reductions/

Supplemental Materials
Lecture notes presentation on NP-Complete problems https://courses.engr.illinois.edu/cs473/sp2010/notes/21-nphard.pdf

Discussion Assignment

Describe in your own words what a NP complete problem is and why a NP complete program is considered to be ‘hard.’

Include one or two examples to explain your thought process to show what is occurring and how the methodology works. Use APA citations and references for any sources used.

Learning Journal

The Learning Journal is a tool for self-reflection on the learning process. In addition to completing directed tasks, you should use the Learning Journal to document your activities, record problems you may have encountered and to draft answers for Discussion Forums and Assignments. The Learning Journal should be updated regularly (on a weekly basis), as the learning journals will be assessed by your instructor as part of your Final Grade.

Your learning journal entry must be a reflective statement that considers the following questions:

- Describe what you did. This does not mean that you copy and paste from what you have posted or the assignments you have prepared. You need to describe what you did and how you did it.
- Describe your reactions to what you did
- Describe any feedback you received or any specific interactions you had. Discuss how they were helpful
- Describe your feelings and attitudes
- Describe what you learned

Another set of questions to consider in your learning journal statement include:

- What surprised me or caused me to wonder?
- What happened that felt particularly challenging? Why was it challenging to me?
- What skills and knowledge do I recognize that I am gaining?
- What am I realizing about myself as a learner?
- In what ways am I able to apply the ideas and concepts gained to my own experience?

Your Learning Journal should be a minimum of 500 words

Self-Quiz

The Self-Quiz gives you an opportunity to self-assess your knowledge of what you have learned so far.

The results of the Self-Quiz do not count towards your final grade, but the quiz is an important part of the University's learning process and it is expected that you will take it to ensure understanding of the materials presented. Reviewing and analyzing your results will help you perform better on future Graded Quizzes and the Final Exam.

Please access the Self-Quiz on the main course homepage; it will be listed inside the Unit.

Checklist

- [ ] Read the Learning Guide and Reading Assignments
- [ ] Participate in the Discussion Assignment (post, comment, and rate in the Discussion Forum)
- [ ] Make entries to the Learning Journal
- [ ] Take the Self-Quiz