Computational Thinking in a Liberal Arts Cryptology Course

Marcus Schaefer
Department of Computer Science
DePaul University
Chicago, Illinois 60604, USA
mschaefer@cdm.depaul.edu

December 10, 2008

Abstract

We describe aspects of computational thinking as covered in the course *Codes and Ciphers (CSC 233)*.

1 Cryptology and Computational Thinking

Cryptology consists of cryptography and cryptanalysis, or, in other words, the making and the breaking of ciphers. History suggests that one cannot successfully do one without the other: too many ciphers were designed by enthusiasts without any knowledge of cryptanalysis, and the consequences were typically dire. In our *Codes and Ciphers* courses, the breaking of ciphers is therefore a central part of discussion and student activity. Historically too, the breaking of ciphers has typically been the motor of advances in cryptography, including Al-Kindi's description in 900 AD of how to use frequency analysis to break substitution ciphers in Arabic and the work of Bletchley Park on breaking the German Enigma cipher during World War II. Throughout history, the cryptographic process has become more algorithmic, as ciphers became more mechanical, mathematical, and, eventually, digital. But even before the mechanization of cryptography in the 19th century (not accidentally coinciding with the advent of computing machines, both analogue and digital), a cryptographer's work was best described in algorithmic terms, and in *Codes and Ciphers* the students will see evidence of that when breaking many classical ciphers.

The following is not a complete account of the topics covered in *Codes and Ciphers*; it concentrates on some of the relevant examples, and it recasts them in a more systematic fashion to emphasize the computational thinking lessons. For the purposes of this document, computational thinking is understood (narrowly) as the way of thinking involved in effective problem solving. No attempt is made to explain the ciphers discussed in detail; Simon Singh's *The Code Book* is a good, popular introduction; it is also, currently, the textbook for the course.

2 Substitution and Frequency Analysis

2.1 The Shift Cipher

One of the oldest ciphers in use is the shift cipher: take each letter in a piece of text and replace it with the letter that occurs k positions later in the alphabet (after letter "z" start over again with "a"). There are several ways of breaking this (highly insecure) cipher, two of which are covered in class and that illustrate different aspects of computational thinking.

Shift Cipher Exercises

Pen and Pencil Solutions

- ◇ Perform shift cipher encryptions and decryptions.
- ◇ Break a shift cipher with pencil and paper by trying all possible keys. Why is this possible?
- ◇ How can that process be simplified (for pencil and paper solution)?
- ◇ Are there any simple tools you could build using pencil/paper/scissor that help in breaking the shift cipher?

Automated Solutions

- ◇ What do plain- and ciphertext of a shift cipher have in common? What does not change when making the shift substitution? *Hint:* look at frequencies.
- ◇ How can we exploit this observation to break the shift cipher?
- ◇ Analyze this solution: what tools can it be performed by and how efficient is it?
- ◇ Could this solution be used for other ciphers? What property do they need to have?

Discussion

(*i*) *Try all possible keys.* The realization that the number of possibilities is small and can be explored exhaustively, leads to the idea of *exhaustive search*, which is a core strategy in computational problem solving. Modern chess programs, for example, at their core still follow this strategy, of course in a much more sophisticated way. However, even in the easy case of the shift cipher it is possible for the students to come up with ways to improve the actual work performed (when breaking the cipher by hand), emphasizing another goal of computational thinking: reducing the amount of *computational resources* required (what is cheap, what is expensive, what is at hand? Pencil, paper, desktop computer, ...).

(*ii*) *Paper strips.* One way to mechanize breaking a shift cipher is to create vertical strips of paper listing the letters "a" through "z" in order; aligning the strips so they spell the ciphertext, one can then look through the remaining rows to find the plaintext. To make this work well, each strip needs to list the alphabet twice (simulating the wraparound). Paper strips like this were used to break ciphers in the 19th century (though not the shift cipher in particular).

(*iii*) *Frequency analysis.* Exhaustive search makes little use of the structure of the shift cipher; a shift cipher is a substitution cipher and as such does not change the frequency distribution of the letters, it just disguises it by renaming the letters. This observation, with a little bit of statistical mathematics, can be used to automate the exhaustive breaking of the shift cipher, that is, the process can be done by a computer (or simulated by hand) without any human involvement. For the students to understand and recreate this process, they need to understand one of the most basic principles in computational thinking: when trying to understand a process (be it a shift cipher, a sorting algorithm, or some other more complex procedure) it is difficult to argue about what *changes*; indeed, it is necessary to understand what a particular process does *not* change. In the case of the shift cipher, one such property is the frequency profile; an invariant property of a process is known as a *loop invariant* in computer science. This is a very simple example of the analysis of change through invariant characteristics, but one that students can easily experiment with visually (the excel spreadsheet used for this cipher visualizes the invariant).

The ideas sketched above lead to a complete and satisfactory solution of the shift cipher; in class these ideas are developed by breaking examples of texts encoded by shift ciphers by hand, so the students have to develop these ideas themselves by *abstracting* from the particular examples they see.

2.2 The Substitution Cipher

The substitution cipher is a generalization of the shift cipher in which each letter is replaced by some other (unique) letter of the alphabet.

Substitution Cipher Exercises

- ◇ Perform some substitution cipher encryptions and decryptions.
- ◇ Can we try all possible keys to break the cipher? How many keys are there?
- ◇ How long would it take to try all keys by hand? On a computer? How would we recognize English plaintext?
- ◇ Does the frequency approach used for the shift cipher work?
- ◇ Does the frequency approach lead to a mechanizable solution of substitution ciphers? How could it be extended to work?
- ◇ What information about English words would be helpful in solving a substitution cipher by hand?

Discussion

(*i*) *Frequency analysis.* Some (overly optimistic) texts claim that substitution ciphers are easily solved by frequency analysis; when trying this by hand, the students will quickly find out that this is at best a half-truth. The loop invariant approach that worked very well for the shift cipher and which, at least in theory, applies to all substitution ciphers, fails to lead to good results. Students realize that an approach

can be *heuristic*, i.e. only lead to partial or suboptimal solutions which might not even be fully correct. The question then is how to improve results.

(*ii*) *Trial and Error.* A form of exhaustive search, trial and error quickly becomes painful for substitution ciphers; students realize that additional understanding of the syntactic nature of English text is required before this approach is meaningful. Indeed, using a combination of frequency cliques, bigrams, and other information about English words, leads to a collection of rules which together with frequency analysis can be used to solve substitution ciphers in which word separations have been maintained. The process is no longer automatic and requires trial and error, but that trial and error is guided by rules based on the behavior of letters in English words.

(*iii*) *Frequency analysis again.* Simple frequency analysis can be complemented by information about the frequency of bigrams and trigrams (this is particularly useful for ciphertexts in which word separations have been eliminated).

(*iv*) *Trial and Error again.* On a computer it is quite feasible to try all possible keys one by one, reducing the problem to how we would recognize English plaintext automatically. For that we can adapt the method of recognizing plaintext in the shift cipher.

The are several computational thinking lessons here: extending a solution of another problem (frequency analysis to solve a shift cipher) to the current problem is a form of reasoning by analogy; one tries to reduce a problem to the solution of a problem one already knows. However, this simple-minded approach fails in this particular case; if complemented with additional rules, however, it can be made to work reasonably well. The students learn that problem-solving itself is an iterative process and that there is value in heuristic solutions when exact solutions are hard to come by. Using a modern computer we can perform an exhaustive search of the keyspace, and recognize English plaintext since that problem reduces to the problem of recognizing plaintext for the shift cipher.

3 The Vigenére Cipher and Reductionism

The Vigenére cipher is a Renaissance cipher that is still known as the "unbreakable cipher", even though it is far from unbreakable and this fact was known in the Renaissance. The attacks on the Vigenére cipher became increasingly more sophisticated, and, with Friedman's statistical work at the beginning of the 20-th century, reached maturity. The Vigenére cipher works by taking a keyword and adding that keyword to the plaintext; in general, the plaintext will be longer than the keyword, so we repeat the keyword as often as necessary.

Vigenére Cipher Exercises

- ◇ Perform some Vigenére cipher encryptions and decryptions.
- ◇ Can we try all possible keys to break the cipher? How many keys are there? How many keys are there if we assume a bound on the length of the keyword?
- ◇ Assuming we have keywords of length at most ℓ how long would it take to break the Vigenére cipher by hand? On a computer? How long must the keyword be for the

Vigenére cipher to be more secure against a brute force attack than a substitution cipher?

- ◇ Is the Vigenére cipher a substitution cipher? How does it relate to substitution and shift ciphers?
- ◇ Suppose we knew the length of the keyword, would that help in breaking the cipher?
- ◇ Encrypt some plaintext with very short keywords; does the ciphertext show any structure that gives you a hint on how to determine the length of the keyword? *Hint:* look for repeated letter sequences. What do they mean?
- ◇ What is the relationship between the index of coincidence and the length of the keyword? Why? *Hint:* remember how we broke the shift cipher automatically.
- ◇ What is the size of the keyspace in the Yale key system?
- ◇ How many keys are required to break the system using Blaize's idea? What is the similarity to breaking the Vigenére cipher?

Discussion

(*i*) *Guessing keywords.* A traditional, and often successful method. Like password guessing, it can serve as a reminder to the student that an exhaustive search need not be truly exhaustive, it can concentrate on the most likely candidates of the search space first (as in a dictionary attack); a crude, but effective heuristic. For the Vigenére cipher the keyspace is infinite in principle, but in practice one can assume that keywords have bounded length.

(*ii*) *Shift ciphers.* As the students play with the system, they will quickly notice that the Vigenére cipher is nothing but a series of ℓ parallel shift ciphers, where ℓ is the length of the keyword. However, shift ciphers are easy to break automatically, as we saw earlier, so if we knew ℓ, we could break the problem into ℓ shift cipher problems, solve them separately (automatically), and combine the solutions. We have modularized the problem, by splitting it into two orthogonal problems: finding the keylength (which we do not yet know how to do) and breaking shift ciphers (which we do know how to do). Breaking problems into orthogonal components like this (i.e. components that can be handled independently), is an important technique in computational thinking, related to divide & conquer (which also occurs here when we split the Vigenére cipher into several shift ciphers). The third computational technique illustrated here is *reduction to a known problem*: we reduce the solution of the Vigenére cipher to the solution of several shift ciphers; to do this we need to ignore, for the moment, that we do not yet know how to solve the first problem: how to find the keylength. One of the advantages of modularization is that we can deal with this issue separately.

(*iii*) *Finding the keylength.* There have been many methods to determine the keylength of a Vigenére cipher, from Porta's observation on repeated letters, the Kasiski test (looking for repeated subwords in the ciphertext), and Friedman's kappa test (index of coincidence). There are more, but these are the three we discuss in class; Porta's (weak) heuristic naturally leads the way to the Kasiski test as a powerful generalization (abstracting from the irrelevant repeated letters); the Kasiski test was invented

at a time when mechanization of society was in full swing, and, not surprisingly, it is a process that lends itself to automation (and it was automated in WWI and before). Looking for automatable solutions is, of course, a trademark, of computational thinking. Finally, Friedman's kappa test is slightly more sophisticated needing some statistical argument, but the students can easily understand it and see that in terms of automation it is superior to Kasiski (which was easy to mechanize, but is more painful, in terms of computational resources, to implement on a computer); it also reuses ideas we saw when breaking the shift cipher automatically. An important point here is that the students get to appreciate that different solutions for the same problem are possible that have different properties in terms of computational resources and that their appropriateness varies by application (trenches in WWI are different from a computer desktop in 2008). This is another important aspect of *modularization*: as long as the interface remains the same (we determine the keylength) we can replace the module with another module performing the same task, but possibly having other properties that are better suited to the environment in which we have to solve the problem (do we need a quick solution, do we have limited memory resources, and so on).

(*iv*) *Pushing the envelope.* The weakness of the Vigenére cipher lies in repeating the keyword; students are naturally led to the question of what happens if the keyword has the same length as the plaintext? Studying this option, one observes that this would still leave too much structure in the ciphertext that can be exploited for a break (though we need different methods from the ones mentioned above; this is actually a slightly difficult problem, so in class we move right on). What happens though, if we chose the keyword at random to destroy this structure? The students have learnt to parameterize the system; at this point the students realize that a system can be parameterized, that is, we can make some aspect of the system variable and change it to improve the system. This is an essential step in understanding how a system works and why it works or fails. Moreover it leads the path to improved systems, in this case the letter one-time pad, that was indeed invented between WWI and WWII used in several forms. A thorough investigation of this system is beyond this class, but it immediately leads to Shannon's theory of information (and he, in turn, was led by considerations like this) which is the foundation of modern computer science. There are two lessons here, one that computational thinking requires theorizing: a well-understood system has mathematical underpinnings; and, secondly, something once understood can become a building block (if appropriately adjusted) in a new endeavor. Indeed, bright students at this point might realize that there is a much stronger form of the Vigenére cipher possible if instead of combining the keyword with shift ciphers one combines it with substitution ciphers; such ciphers are known and easily breakable today, but, in the Renaissance, would have offered the security that the Vigenére cipher did not; however, the two ideas were put together rather late, and this version of the Vigenére cipher never became very popular as a paper and pen cipher; it did, however, eventually lead to rotor ciphers and the Enigma (which is covered in the next section).

(v) *Application.* Computational problem-solving has two aspects: modeling and algorithmic problem-solving; the modeling part is less obviously computational than the algorithmic part, but it is one of the most powerful tricks of computational thinking: recasting one problem as another problem that has already been solved; prime examples in computer science include linear programming and network flows; computational complexity theory is based on the notion of reduction between problems. To be able to model a problem successfully, one must be able to see structural connections between problems that might not be apparent at the surface; one such example which ties in with the Vigenére cipher is Matt Blaize's ingenious break into the Yale key system; Blaize showed that it is to break the Yale key system if there is a master key and one has access to a single lock (in that case one can easily reconstruct the master key). Blaize's solution is structurally the same as the break into the Vigenére cipher: the problem is broken into a small set of independent problems that can be solved in parallel, which leads to a significant reduction in the size of the search space. The analogy between the two problem solutions drives home the importance of understanding a problem structurally, at an abstract level.

At this point students have seen the brute-force approach to breaking a cipher repeatedly, and they realize that the size of the keyspace is a first, rough, measure of complexity for breaking a cipher. Also, all examples so far have illustrated that it can be a very misleading measure: the size of the keyspace might be inflated. The core concept emphasize by the breaking of the Vigenére cipher, however, is *modularization*: the weakness of the cipher lies in the fact that its complexity is illusionary, it can easily be separated into two independent parts that can easily be tackled separately.

4 Enigma and the Mechanization of Cryptology

Breaking the enigma code was one of the major intelligence triumphs of WWII; the first breaks into the German enigma system were found by the Polish cipher bureau; later Bletchley Park completed this work. Today a desktop computer can break an enigma cipher in a matter of minutes (given enough ciphertext), but this, of course, was not the case in the 1940s. The Polish contribution to breaking the enigma again illustrates the importance of finding an *invariant*; the enigma cipher is a progressive cipher: a substitution cipher in which the substitution changes for each encrypted letter; what Rejewski, the head of the Polish cipher bureau realized, was that from the ciphertext one could abstract information that allowed one to identify the rotor settings of the enigma machine (if one had sufficiently many ciphertexts). To make this method operational, this information had to be collected and indexed in a large catalogue; this method was barely practical and Bletchley Park improved on it: as in the case of the Vigenére cipher (and the Yale key system), Alan Turing realized that one could separate out the effects of the rotor settings from the effects of the steckerboard (an added, fixed, substitution cipher); modularizing the problem thus split the problem into two: finding the rotor settings (which, with a guided search, was amenable to automated search by the bombes, special purpose computers performing an exhaustive search of the keyspace), and the solution of a simple (restricted) substitution cipher. Not coincidentally, the work on the enigma also led Turing to work on and help design one of the

first general purpose computers (the Colossus), as a more general tool for breaking ciphers. Interestingly this work extends Turing's theoretical work in the foundations of computability: he was the first to rigorously model what a computer is and ask (and answer) the question of what is, in principle, computable with a computer. Breaking the enigma was one of the pathways into the modern computer. Turing did not foresee the refinements his work led to: distinguishing computable problems into feasible and infeasible problems, but he laid the foundation for this program that turned into modern theoretical computer science and the study of resource-bounded computation. The importance of studying resource bounded computation is discussed in the next section.

5 Public-Key Cryptography and the Mathematization of Cryptology

For much of its history, cryptology has been an arcane art, and it still is. However, starting in the 20th century, there has been a notable trend towards mathematical methods in both the making and the breaking of ciphers. One of the earliest examples (only quickly discussed in class) is the Hill cipher based on matrix multiplication; modern cryptography is based on modular arithmetic, elliptic curves, and many other mathematical concepts.

Public-key cryptography is a strong departure from classical cryptography: in public-key cryptography two parties (typically called Alice and Bob) can communicate securely over a public channel, even if they have never met before or exchanged any secret information: they can establish secure communication between them while other parties are listening (the other person is always known as Eve, the eavesdropper). Intuitively, this seems impossible: how could you exchange a secret with somebody you don't know over the phone, say, in the presence of other people listening in on your communication? However, public-key cryptography shows that this is indeed possible, and the mathematics involved for the easier systems is not very difficult and can be understood by the students.

Public-Key Cryptography Exercises

- ◇ Do an example of a Diffie/Hellman key exchange. How does this make public-key cryptography possible?
- ◇ Do an example of the three-way protocol. Why can this system not be broken using logarithms? What about discrete logarithms?
- ◇ How can we calculate the average salary of everybody in the room without anybody learning anybody else's salary? *Hint:* the problem can be solved using a trusted third party (Trent), but what if we do not even allow that? We can assume that everybody is honest, but we do not want to share our salary information.
- ◇ Do an example of bit-commitment using the hardness of the discrete log problem.
- ◇ How can we use bit-commitment to convince somebody that we have found a legal coloring of a graph?

Discussion

- *Key exchange.* The Diffie/Hellman protocol allows Alice and Bob to exchange a secret

key; they can then use that secret key to use any traditional cipher to communicate secretly. So while Diffie/Hellman is not a public-key cryptosystem, it fulfills a similar function: it makes secret communication over an open channel possible without requiring a shared secret at the start. This is a classical instance of reducing one problem (secure communication) to another (key exchange).

- *Computational limitations.* All modern cryptosystems are based on the assumption that certain mathematical problems are intractable, that is, hard to solve by a computer. While traditional systems use estimates on the size of the keyspace as a rough measure of complexity, the hardness of breaking modern systems can often be tied more closely to mathematical problems. For example, Diffie/Hellman is based on the assumption that a variant of the discrete logarithm problem is hard, amd RSA is based on the hardness of factoring large numbers. So in modern cryptography computation is more than a tool, it has become a meta-tool: the very fact that some task is computationally hard, can be exploited to build a feasible solution to another problem (public-key cryptography), and indeed, there are approaches to cryptography that link the theory of computation to cryptography (an advanced topic not explored in this class). However, the students do understand that for certain computational tasks, such as cryptography, understanding the nature of computation itself becomes important. One consequence, and an important one as far as thinking computationally is concerned, is that when designing a system, such as a cryptosystem, that it can be useful to make assumptions on the computational power of an opponent, and thereby make possible solutions to problems that are otherwise unsolvable.

- *Randomness.* Randomness has been a resource for millennia: simple games are not possible without a die or some other source of random choice. However, randomness, being random, at first glance, does not seem to be a useful resource in the solution of practical problems. The opposite turns out to be true: there are areas of computer science, such as computational geometry, where many basic algorithms require randomness to become practical, and modern cryptography has also been able to channel the power of randomness. There are problems which are, in principle, not solvable in a deterministic computational model, however, allowing randomness suddenly yields solutions. Zero-knowledge protocols belong to this area. Slightly simplified, one can convince another person that one holds a particular piece of information without giving away anything at all about the information itself (how do you convince somebody that you were able to solve the Times crossword puzzle without even giving them a single letter of your solution?). We touch on these ideas in the class. Again, students come to understand, that making the nature of computation (deterministic versus randomized) the object of study was a central step in making possible solutions to problems that had been unsolvable before.