Amazing Algorithms

Eric Roberts
CSCI 121
November 29, 2018

The Intellectual Excitement of Algorithms

- Algorithms can be as elegant and intellectually challenging as any other field of study. At Stanford, I taught a seminar for many years with the title "The Intellectual Excitement of Computer Science" to disabuse students of the notion that computer science was all about programming.
- In this lecture, I will present three of my favorite algorithms:
  1. **Public-key encryption**, which allows secure communication even when the encryption keys are not kept secret.
  2. **The median-of-medians algorithm**, which finds the median of a distribution in $O(N)$ time.
  3. **Boyer-Moore string search**, which finds a pattern string in a longer one using a strategy that runs faster as the pattern string gets longer.

Public-Key Encryption

- In 1999, shortly before he arrived at Stanford for an eight-year stay, Harvard Law professor Larry Lessig published *Code and Other Laws of Cyberspace*.
- In his book, Lessig argues—with at least a bit of hyperbole—that cryptography is the most revolutionary development of the last millennium.

Here is something that will sound very extreme but is at most, I think, a slight exaggeration: encryption technologies are the most important breakthrough in the last thousand years. No other technological discovery ... will have a more significant impact on our social and political life. Cryptography will change everything.

Idealized Model of Encryption

The Search for a Perfect Code

- The problem of secure encryption illustrated on the previous slide does have a solution, at least in theory. One strategy is to have the sender and receiver share a codebook with the following properties:
  - The codebook is so large that none of it is ever reused.
  - The mapping of plaintext letters is random in the sense that past transformations convey no information about later ones.
- Because the individual entries in the codebook are never reused, this approach is generally called a **one-time pad**. It is also called a **Vernam cipher** after Gilbert Vernam, an engineer at AT&T Bell Labs who patented the technique in 1917.

Use of the One-Time Pad

- The one-time pad strategy has been used in practice on several occasions.
  - The Soviet Union used a one-time pad for its communications in World War II. Operators, however, occasionally reused pages from the codebook, which enabled British and American codebreakers to decrypt some messages.
  - When he was killed in Bolivia in 1968, Che Guevara was carrying a one-time pad to communicate with Fidel Castro.
- In practice, the one-time pad is problematic because of the size of the codebook and the difficulty of distributing it securely.
Solving the Key-Distribution Problem

• The problem of distributing encryption keys—no matter whether they are complete codebooks used in one-time pads or shorter keys used in other encryption strategies—is hard only if the keys must be kept secret.
• The idea that keys can be made public at first seems crazy. After all, if a key is public than anyone can use it.
• That fact, however, causes problems only if knowing the public key makes it possible for an eavesdropper to read messages to which they should not have access. If knowing the public key lets anyone encrypt a message but allows only the intended recipient to decrypt that message, the problem goes away.
• A coding strategy that allows encryption keys to be shared but protects decryption keys is called public-key encryption.

GCHQ Invents Public-Key Encryption

• Public-key cryptography was invented in England by the Government Communications Headquarters (GCHQ), which succeeded the Government Code and Cipher School (GCCS) that broke the Enigma code in World War II.
• The basic ideas of public-key encryption were developed in the early 1970s by the GCHQ cryptographers James Ellis, Clifford Cocks, and Malcolm Williamson. Their work went well beyond the fundamental concepts to anticipate many of the practical cryptographic protocols that were subsequently rediscovered in the United States.
• Unfortunately, none of the later researchers knew about the British cryptographic work. At GCHQ, everything relating to public-key cryptography remained classified until late in the 1990s, long after this technology was commercialized.

James Ellis on “Non-Secret Encryption”

THE POSSIBILITY OF SECURE NON-SECRET DIGITAL ENCRYPTION
J. H. Ellis, January 1970

Introduction
1. It is generally regarded as self-evident, that, in order to prevent an interceptor from understanding a message which is intelligible to the authorised recipient, it is necessary to have some initial information known to the sender and to the recipient but kept secret from the interceptor. This information can take many forms, such as the method of encipherment itself, the construction of a cipher machine, a key setting or a one-time tape. All these methods require that there is a route by which this secret information can be sent without fear of interception. Only then can the cipher text be sent safely in a non-secret manner, and large quantities of cipher text of high security thus tend to need the parallel transmission of smaller, but still substantial quantities of secret information.

James Ellis on “Non-Secret Encryption”

2. This report demonstrates that this secret information is not theoretically necessary and that, in principle, secure messages can be sent even though the method of encipherment and all transmissions between the authorized communicators are known to the interceptor. This is what is meant by “non-secret encryption”. It must be emphasised however that this demonstration has only the status of an existence theorem. It shows only that such a system is theoretically possible, and not that a practical form exists. The demonstration consists of showing that a particular, but unfortunately as yet highly impractical, system has the desired properties. This is followed by an heuristic discussion which attempts to establish the necessary properties of a system and indicate the likely form of a practical solution.

Stanford RedisCOVERS the Idea

• The first unclassified announcement of public-key encryption appeared in 1976 in a paper by Stanford graduate student Whit Diffie and his advisor Martin Hellman. The two appear in the picture to the right along with another early collaborator, Ralph Merkle.
• The Stanford researchers not only laid out the basic structure of public-key encryption but also proposed an implementation based on a variant of the subset-sum problem, which is an NP-complete problem.
• Unfortunately, the specific variant of the problem they chose resulted in an encryption system that was unexpectedly easy to solve.

Public-Key Encryption

1. Bob generates a pair of keys labeled $D_B$ and $E_B$.
2. Bob publishes $E_B$ as his public key.
3. Bob keeps $D_B$ hidden as his private key.
4. Alice uses $E_B$ to send a secret message to Bob.
5. Only someone with $D_B$ can decipher the message.
The RSA Algorithm

- In 1977, a team consisting of Ron Rivest, Adi Shamir, and Len Adleman—all then at MIT—developed a practical implementation of public-key encryption, which they called RSA after their initials.
- The RSA scheme relies on the difficulty of factoring large numbers. As long as the number is large enough—typically many hundreds of digits—revealing the product of two prime numbers \( p \) and \( q \) does not allow an eavesdropper to recover the original values.

Key Generation in RSA

- As in any public-key encryption scheme, RSA requires each potential recipient to generate two keys, a public key that allows anyone to send an encrypted message and a private key that ensures that only the recipient can decrypt that message.
- Generating an RSA key pair requires the following steps:
  1. Choose two prime numbers \( p \) and \( q \).
  2. Define the variable \( n \) as \( pq \) and the variable \( t \) as \((p - 1)(q - 1)\).
  3. Choose an integer \( d \) \(< n \) so that \( d \) is relatively prime to \( t \). Two integers are relatively prime if they share no common factors.
  4. Set \( e \) to be the modular inverse of \( d \) with respect to \( t \), which is the integer for which the product \( de \) divided by \( t \) leaves a remainder of 1. The fact that \( d \) and \( t \) are relatively prime means that this value is unique.
  5. Release \( n \) and \( e \) as the public key and use \( d \) as the private key.

Encryption and Decryption in RSA

- Given the public-key parameters \( n \) and \( e \), the sender creates an encrypted message \( c \) using the following steps:
  1. Convert the message into a binary integer \( m \) by using the internal character codes. If the message length exceeds the size of \( n \), the sender must break it into smaller pieces and encrypt each one individually.
  2. The sender then computes the ciphertext \( c \) as follows:
    \[ c = m^e \mod n \]
  3. The recipient restores the original message by calculating
    \[ m' = c^d \mod n \]

A Tiny Example (Key Generation)

- Choose two prime numbers \( p \) and \( q \):
  \( p = 11 \)
  \( q = 23 \)
- Define the variable \( n \) as \( pq \) and the variable \( t \) as \((p - 1)(q - 1)\):
  \( n = 253 \)
  \( t = 220 \)
- Choose an integer \( d \) \(< n \) so that \( d \) is relatively prime to \( t \):
  \( d = 17 \)
- Set \( e \) to be the modular inverse of \( d \) with respect to \( t \):
  \( e = 13 \)  \( (17 \times 13 = 221; 221 \mod 220 = 1) \)
- Release \( n \) and \( e \) as the public key and use \( d \) as the private key.

A Tiny Example (Encryption)

- Convert the message into an integer \( m \):
  \( m = 'A' = 65 \)
- Create the ciphertext \( c \) by calculating \( m^e \mod n \):
  \( c = 76 \)
- Validate the encryption by calculating \( c^d \mod n \):
  \( c^d = 369720589101871337890625 \)
  \( c' = 76 \)

Fast String Search

- A common operation that comes up often in word-processing applications is searching for one string in a longer string.
- Although string searches are very easy to implement, it can be hard to make them run efficiently.
- Traditional string searches typically run more slowly as the string you’re searching for gets longer, which appears to make intuitive sense.
- In 1977, Robert Boyer and J Strother Moore at the University of Texas at Austin developed an algorithm that runs faster for longer search strings.
- The key insight behind Boyer-Moore search is that it is far more efficient to check for matches starting at the end of the search string rather than at the beginning.
Traditional String Search

- The most straightforward string algorithm aligns the search string against the beginning of the text.
- If the characters match, the algorithm goes on to check each subsequent character. If it matches all the way through, the search is over.
- If the characters ever fail to match, the algorithm shifts the search pattern ahead one character position and tries again.

Boyer-Moore String Search

- The Boyer-Moore algorithm works from the end of the search string using a "bad-character table" that advances to the next possible match position.
- If the last characters match, it goes through the search string backwards, checking for a match.

Finding Medians in Linear Time

- The median of a distribution is the element at the halfway point in a sorted list of values, with just as many elements larger than the median as elements smaller than the median.
- It is easy to compute the median by sorting an an array and then selecting the middle element, as shown in the following Python code:

```
def median(array):
    return sorted(array)[len(array) // 2]
```
- The problem with this approach is that sorting algorithms require \( O(N \log N) \) time.
- Before Manuel Blum, Robert Floyd, Ron Pratt, Ron Rivest, and Bob Tarjan found an \( O(N) \) solution in 1973, most computer scientists believed that \( O(N \log N) \) was minimal.

Bob Tarjan

- I still remember hearing Bob Tarjan give a talk at Harvard about this result in 1974 or 1975 when he was an Assistant Professor at Stanford.
- Bob’s talk was absolutely stunning. He described a series of results—each of which was absolutely amazing—as if they were simply mundane discoveries of the sort he would make every day.
- Bob Tarjan has been Professor of Computer Science at Princeton since 1985 and won the ACM Turing Award in 1986.
- The linear-time median finding algorithm is animated on the next slide.

Linear-Time Median Algorithm

1. Start with an array of values.
2. Divide the elements into groups of five.
3. Sort each of the groups.
4. Highlight the medians in each group.
5. Use recursion to find the median of the medians.
6. Eliminate the upper left and lower right quadrants.
7. Use recursion to find the median of the remaining elements.

```
27 18 28 28 28 45 45 53 60 28 71 71 51 28 49 77 57 74 70 93 69 99
```

```
18      23     28     26     24
18      45     35     59     69
27      45     60     57     70
28      53     71     62     93
28      90     74     27     99
```

Why this Algorithm Runs in Linear Time

- A formal proof that this algorithm uses linear time is beyond the scope of CSCI 121, but I can offer a general argument.
- At each level, the algorithm requires the following steps:
  - Divide the input into groups of five, creating \( N/5 \) groups.
  - Sort each group. Since the size is always five, the sorting time does not depend on \( N \) and is therefore \( O(1) \) for each subarray.
  - Use a recursive call to find the median of the medians.
  - Use a second recursive call to find the actual median after eliminating values that are known to be too small or too large.
  - This step finds the median of no more than \( 7N/10 \) values.
- The total time is then given by the recurrence relation
  \[
  T(n) = n + T(n/5) + T(7n/10)
  \]
- In CSCI 382, you will prove that this formula is \( O(n) \).