1 April 2019

Random Walk Algorithms: Lecture 1

David A. Meyer

In which the topic of these notes is specified, namely random walk algorithms, and the notion of algorithm is introduced etymologically and then by the example of deterministic primality testing algorithms and their complexities.

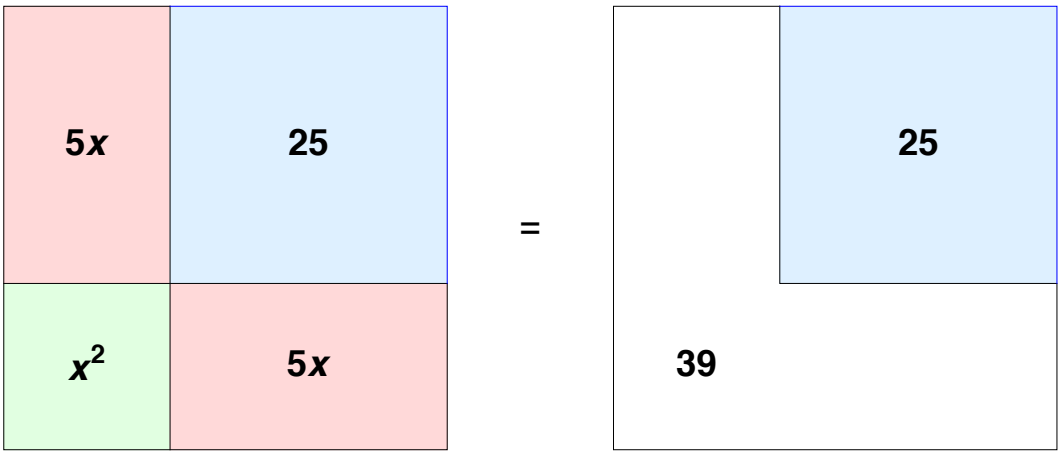
Introduction

There are multiple reasons for studying random walk algorithms. Several practically useful algorithms are random walk algorithms, including Google's PageRank algorithm and Markov Chain Monte Carlo algorithms. Furthermore, they have conceptually important connections with physics, specifically the heat equation. Finally, an understanding of random walk algorithms provides a foundation for studying quantum walk algorithms.

Motivated by these reasons, the plan for these notes is to introduce random walks, explain their connection to physics, and then describe several random walk algorithms in detail. Then we will introduce quantum walks, explain their connection to physics, and finally describe at least one quantum walk algorithm.

Algorithms

Algorithms are well-defined computational processes, often for solving problems [1]. The word "algorithm" comes from the *nisba* (derivational adjective) [2] in the name of the Persian scholar Muḥammad ibn Mūsā al-Ḵwārizmī [3]. His text *Al-kitāb al-mukhtaṣar fīḥisāb al-jabr wal-muqābala*, *Compendium on Calculation by Completing and Equating*, was perhaps the first algebra textbook, and included algorithms for solving quadratic equations [1,4]. For example, consider the problem "a Square and ten Roots are equal to thirty-nine *Dirhems*" [5, p.13], which he solves by completing the square as in this picture:



In modern notation, we would implement this algorithm as [4]:

$$
\begin{aligned}
& x^2 + 10x = 39 \\
\Rightarrow\ & x^2 + 2 \cdot 5x = 39 \\
\Rightarrow\ & x^2 + 2 \cdot 5x + 5^2 = 39 + 5^2 \\
\Rightarrow\ & (x+5)^2 = 64 \\
\Rightarrow\ & x + 5 = 8 \qquad \text{(negative numbers were unknown in the } 9^{\text{th}} \text{ century!)} \\
\Rightarrow\ & x = 3
\end{aligned}
$$

To give another example of an algorithm, consider how you would decide if a number is prime. We all learn in school that a straightforward way to do this is to check whether it is divisible by smaller numbers, and that we don't need to check any that are larger than its square root, since if one divisor is larger than the square root, then the quotient, and hence another divisor, must be smaller than the square root. Let us write this out carefully as an algorithm:

input: $N \in \mathbb{N}$.
output: True if N is prime; False if not.

prime $\leftarrow$ True
if $N = 1$, *prime* $\leftarrow$ False
else $d = 2$;
    while $d \leq \sqrt{N}$,
        if $d|N$, *prime* $\leftarrow$ False
        else $d \leftarrow d + 1$
return *prime*

An important consideration for any algorithm is how efficient it is—how long, or how many steps, does it take to complete. (An even more important consideration, of course, is whether it returns the correct answer! This algorithm does, if we agree that 1 is not a prime number, more about which later.) Since this algorithm checks divisibility by d, for each integer in $\{2, 3, 4, \ldots, \sqrt{N}\}$, it does $\lfloor \sqrt{N} \rfloor - 1$ divisions of N. And if you think about the long division algorithm you learned in elementary school, you will realize that each digit in the quotient (of which there are about the same number as in N) must be multiplied by the divisor d, which is of size $\sqrt{N}/2$ on average. Since N has $n = \log N$ digits and $\sqrt{N}/2$ has about $n/2$ digits, the total running time (counting only the multiplications) for large N is $O\big(\sqrt{N}(\log N)^2\big)$. Here we are using "big-O" notation:

DEFINITION. $f(x) = O\big(g(x)\big)$ as $x \to \infty$ iff there exist $k, M > 0$ such that for all $x \geq M$, $|f(x)| \leq kg(x)$. Also, $f(x) = \tilde{O}\big(g(x)\big)$ if for some $c > 0$, $f(x) = O\big(g(x) \log^c x\big)$; this is called "soft-$O$" notation.

Of course, it is easy to improve this algorithm. The simplest thing to do is to replace the

check at the beginning of the while loop with

$$\text{while } d \leq \sqrt{N} \text{ and } \textit{prime}$$

so that as soon as N is known not to be prime the algorithm terminates. But of course this only speeds up the algorithm when N is not prime; when it is prime all the possible factors will be checked. So while this reduces the *average case complexity*, it does not reduce the *worst case complexity*.

Long division, however, is suboptimal. An asymptotically better option, for example, is Newton-Raphson division [6], which has the complexity of multiplication of two n digit numbers. This was conjectured by Arnold Schönhage and Volker Strassen in 1971 to be $O(n \log n)$ [7]; as of March 2019, this may have been proved [8]. If so, the complexity of this primality testing algorithm would be reduced to $O(\sqrt{N} \log N \log\log N)$.

In fact, a polynomial (in $n = \log N$) time class of primality testing algorithms was discovered in 2002 by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena [9]. Their original algorithm had complexity $\tilde{O}(n^{12})$; by the time it was published it had been improved to $\tilde{O}(n^{15/2})$. Subsequently, Hendrik Lenstra and Carl Pomerance improved this to $\tilde{O}(n^6)$ [10].

All of these algorithms are *deterministic*; they have no probabilistic components. For primality testing, and other problems, random algorithms can be superior for practical purposes. Random algorithms include the random walk algorithms on which we will focus, so in the next few lectures we will review basic ideas from probability theory.

References

[1] D. E. Knuth, "Algorithms in modern mathematics and computer science", Stanford Department of Computer Science Report No. STAN-CS-80-786 (1980).

[2] A. F. L. Beeston, *Arabic Nomenclature: A summary guide for beginners* (Oxford: University Press 1971).

[3] Oxford English Dictionary, "algorism, *n.*".

[4] P. Maher, "From al-jabr to algebra", *Mathematics in School* **27** (1998) 14-15.

[5] F. Rosen, editor and translator, *The Algebra of Mohammad ben Musa* (London: Oriental Translation Fund 1831).

[6] M. J. Flynn, "On division by functional iteration", *IEEE Transactions on Computers* **C-19** (1970) 702-706.

[7] A. Schönhage and V. Strassen, "*Schnelle Multiplikation großer Zahlen*", *Computing* **7** (1971) 281–292.

[8] D. Harvey and J. Van Der Hoeven, "Integer multiplication in time $O(n \log n)$", hal-02070778 (2019).

[9] M. Agrawal, N. Kayal and N. Saxena, "PRIMES is in P", *Annals of Mathematics* **160** (2004) 781–793.

[10] H. W. Lenstra, Jr. and C. B. Pomerance, "Primality testing with Gaussian periods", *Journal of the European Mathematical Society* (2019) doi: 10.4171/JEMS/861.