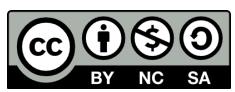
UCT

Probabilistic Algorithms

Klaus Sutner

Carnegie Mellon University

Spring 2022



1 **Some Probabilistic Algorithms**

2 **Probabilistic Primality Testing**

We have an apparently correct abstract definition of randomness, regrettably but necessarily one that excludes computability.

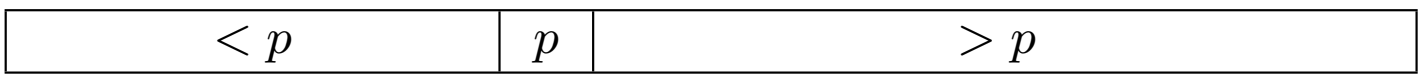
On the other hand, there are methods to get reasonably random bits by iteration (Mersenne twister) or via quantum physics (Quantis card). We can use these pseudo-random bits to speed up algorithms.

It remains to identify complexity classes that correspond to these randomized algorithms, and check how they relate to traditional classes.

For some algorithms, randomness is a convenient analysis tool, but not really required. Typical examples are algorithms requiring the choice of a pivot:

- order statistics
- quicksort

We want to move elements in an array to the left and right of a chosen pivot:



Ideally, we would like to pick p at random so that the expected value of the length of the left sublist is $n/2$.

Here we are given a list $A = a_1, \ldots, a_n$ and some index k, $1 \leq k \leq n$. The problem is to find the k-smallest element in the list:

$$\mathsf{ord}(k, A) = a'_k$$

where $a'_1, a'_2, \ldots, a'_n$ is the sorted version of the list[†] [‡].

Special cases: $k = 1$: min, $k = n$: max, $k = n/2$: median. Obviously, we don't want to sort the whole array.

Here is a trick: think about verification rather than construction. If someone told us that $b = \mathsf{ord}(k, A)$ we could verify that claim in $\mathbb{L}$ space and linear time: just count the smaller guys.

$$b = \mathsf{ord}(k, A) \iff |\{\, i \mid a_i < b \,\}| = k - 1$$

So we want to identify the element b that is larger than exactly $k - 1$ others.

[†] For simplicity, assume all elements are distinct, so there is a unique solution.

[‡] Also, never define a mathematical object in terms of an algorithm.

Insight: This looks vaguely like the partitioning technique in quick sort.

We can adapt pivoting to order statistics. Let's write (B, p, C) for the result of partitioning with pivot p and let m be the position of p after partitioning.

- If $m = k$: return p.
- If $m > k$ repeat with (B, k).
- If $m < k$ repeat with $(C, k - m)$.

If we are out of luck, this will lead to bad splits and linear recursion depth. But if we pick the pivot at random this method is very fast on average.

Lemma

Partitioning solves the order statistics problem in expected linear time.

Choosing the pivot uniformly at random has the effect that the expected length of the sublist generated by a single pivot step is something like $n/2$. This would suggest that the depth of the recursion is $\log n$, and the lengths decrease exponentially, so the expected running time should be linear.

If the pivot is constructed from true random bits, then the use of this type of argument is fully justified. But if we are dealing with pseudo-random bits, there is potentially a problem and we really should verify that the probabilistic arguments are sufficiently robust: they have to hold up (more or less) even if the bits do not pass all sequential tests.

We can get away with murder in this case: instead of picking the pivot at random, sample the list in 3 places (say, first, middle, last) and pick the median. Let X_i be the length of the left sublist generated in partitioning.

We need to calculate the probabilities $p_i = \mathsf{Pr}[X_i = i]$.

$$p_i = \frac{6\, i(n-i-1)}{n(n-1)(n-2)}, \qquad i = 0, \ldots, n-2$$

$$\mathsf{E}[X_n] = \frac{n-1}{2}$$

Actually, these p_i are better clustered around the middle than just $1/n$. But, as a matter of principle, there are bad inputs where the deterministic method fails.

For quite some time it was believed that order statistics was as difficult as sorting and could not be done in less than log-lin time for the worst case (assuming that that only full comparisons between one item and another are possible).

In fact, people tried to show that sorting could be reduced to order statistics.

It was a huge surprise when finally a divide-and-conquer algorithm was found that runs in a linear time.

Unfortunately, the constants are so bad that the algorithm is of no practical importance.

Theorem (Blum, Floyd, Pratt, Rivest, Tarjan 1973)

Order statistics can be handled in linear time.

Example 2: Matrix Multiplication

Suppose we want to check that $A \cdot B = C$, where A, B, C are $n \times n$ matrices over some ring. In other words, we want to verify the work of some (fancy fast) matrix multiplication algorithm.

One way is to compute the the result with other algorithms and compare. Perfectly fine, but slow.

Alternatively, pick a random 0/1-vector $\boldsymbol{r}$ and compare

$$A \cdot (B \cdot \boldsymbol{r}) \qquad \text{and} \qquad C \cdot \boldsymbol{r}$$

This takes only $O(n^2)$ steps and discovers inequality with probability $1/2$.

We may assume $AB \neq C$.

It suffices to show that for two different vectors $\boldsymbol{x} \neq \boldsymbol{y}$ we have $\mathsf{Pr}[\boldsymbol{x} \cdot \boldsymbol{r} = \boldsymbol{y} \cdot \boldsymbol{r}] \leq 1/2$.

In other words, $\mathsf{Pr}[\boldsymbol{z} \cdot \boldsymbol{r} = \boldsymbol{0}] \leq 1/2$ for any non-zero vector $\boldsymbol{z}$. Wlog assume $z_1 \neq 0$ and write $s = \boldsymbol{z} \cdot \boldsymbol{r} = z_1 r_1 + C$.

$$\mathsf{Pr}[s = 0 \mid C = 0] = \mathsf{Pr}[r_1 = 0] = 1/2$$

$$\mathsf{Pr}[s = 0 \mid C \neq 0] = \mathsf{Pr}[r_1 = 1 \wedge z_1 = -C] \leq 1/2$$

But then $\mathsf{Pr}[\boldsymbol{z} \cdot \boldsymbol{r} = \boldsymbol{0}] \leq \mathsf{Pr}[s = 0] \leq 1/2$.

Note that these bounds are rather rough, in practice we can expect much better performance.

Problem:	**Polynomial Zero Testing (PZT)**
Instance:	A polynomial $P(x_1, \ldots, x_n)$ with integer coefficients.
Question:	Is P identically 0?

Wait, this is totally trivial ...

Yes, but there is a major glitch: the polynomial P need not be given in explicit form as a coefficient vector.

For example, P could be a sum of products $P_1 \cdot P_2 \cdot \ldots \cdot P_r$ where the P_i are polynomials. Of course, we can obtain the explicit form by multiplying out, but that is a potentially exponential operation.

In fact, we should think of the polynomial as given in form of a straight-line program or an arithmetic circuit using operations $\{+, -, \times\}$.

Counting Roots

Let $\mathbb{F}$ be a field of characteristic 0 (such as the rationals, reals, complexes).

Lemma (Schwartz-Zippel 1980)

Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be of degree d and $S \subseteq \mathbb{F}$ a set of cardinality s. If P is not identically zero, then P has at most $d\, s^{n-1}$ roots in S.

If you prefer, you can think of this as a probabilistic assertion:

$$\Pr[P(\boldsymbol{x}) = 0] \leq d/s.$$

Here the arguments $\boldsymbol{x}$ are supposed to be chosen uniformly at random from S.

Note that for the lemma we don't need random bits, but see the algorithm below.

Proof

The proof is by induction on n, the number of variables.

The case $n = 1$ is clear.

For $n > 1$, define d_1 and $P'(x_2, \ldots, x_n)$ to be the degree of x_1 in P and the coefficient, respectively. Hence

$$P(x_1, \ldots, x_n) = x_1^{d_1} P'(x_2, \ldots, x_n) + \text{stuff}$$

Write $\boldsymbol{a} = a_1, a_2, \ldots, a_n \in S^n$ and $\boldsymbol{a}' = a_2, \ldots, a_n$. If $P'(\boldsymbol{a}') \neq 0$, then $P(x_1, \boldsymbol{a}')$ is not zero and has degree d_1. Hence

$$\Pr[P(\boldsymbol{a}) = 0 \mid P'(\boldsymbol{a}') \neq 0] \leq d_1/s$$

But then we have

$$\begin{aligned}
\Pr[P(\boldsymbol{a}) = 0] &= \Pr[P(\boldsymbol{a}) = 0 \wedge P'(\boldsymbol{a}') = 0] + \Pr[P(\boldsymbol{a}) = 0 \wedge P'(\boldsymbol{a}') \neq 0] \\
&\leq \Pr[P'(\boldsymbol{a}') = 0] + \Pr[P(\boldsymbol{a}) = 0 \mid P'(\boldsymbol{a}') \neq 0] \\
&\leq (d - d_1)/s + d_1/s = d/s
\end{aligned}$$

Done.

□

The set $S \subseteq \mathbb{F}$ here could be anything. For example, over $\mathbb{Q}$ we might choose $S = \{0, 1, 2, \ldots, s-1\}$.

The main application of the lemma is to give a probabilistic algorithm to check whether a polynomial is identically zero.

Suppose P is not identically zero and has degree d. Choose a point $\boldsymbol{a} \in S^n$ uniformly at random and evaluate $P(\boldsymbol{a})$. Then

$$\Pr[P(\boldsymbol{a}) = 0] \leq d/s$$

So by selecting S to have cardinality $2d$ the error probability is $1/2$.

Note that the number of variables plays no role in the error bound. To lower the error bound, we can amplify truth by repeating the basic step, relying on independence: the choices in each run must be independent.

Alternatively, we can increase s.

So What?

This result is really quite amazing. It is always a good idea to try to figure out how one might derandomize a probabilistic method.

First consider univariate polynomials. To check deterministically, we would need to evaluate at $d + 1$ points. Not too bad.

But for multivariate polynomials we need $(d + 1)^n$ test points, a rather unwieldy method.

Some combinatorial problems can be translated relatively easily into PZT.

For example, suppose $G = \langle [n], E \rangle$ is an undirected graph and define its Tutte matrix as the skew-symmetric matrix

$$A(i,j) = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j, \\ -x_{ij} & \text{if } (i,j) \in E \text{ and } i > j, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem (Tutte 1947)

G has a perfect matching iff its Tutte matrix has non-zero determinant.

$$
\begin{pmatrix}
0 & 0 & x_{1\,3} & x_{1\,4} & 0 & x_{1\,6} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & x_{2\,4} & x_{2\,5} & 0 & x_{2\,7} & 0 & 0 & 0 \\
-x_{3\,1} & 0 & 0 & 0 & x_{3\,5} & 0 & 0 & x_{3\,8} & 0 & 0 \\
-x_{4\,1} & -x_{4\,2} & 0 & 0 & 0 & 0 & 0 & 0 & x_{4\,9} & 0 \\
0 & -x_{5\,2} & -x_{5\,3} & 0 & 0 & 0 & 0 & 0 & 0 & x_{5\,10} \\
-x_{6\,1} & 0 & 0 & 0 & 0 & 0 & x_{6\,7} & 0 & 0 & x_{6\,10} \\
0 & -x_{7\,2} & 0 & 0 & 0 & -x_{7\,6} & 0 & x_{7\,8} & 0 & 0 \\
0 & 0 & -x_{8\,3} & 0 & 0 & 0 & -x_{8\,7} & 0 & x_{8\,9} & 0 \\
0 & 0 & 0 & -x_{9\,4} & 0 & 0 & 0 & -x_{9\,8} & 0 & x_{9\,10} \\
0 & 0 & 0 & 0 & -x_{10\,5} & -x_{10\,6} & 0 & 0 & -x_{10\,9} & 0
\end{pmatrix}
$$

Expanded, the determinant has 60 monomials of length 10 each.

The idea behind the proof is that in the standard Leibniz representation of the determinant

$$|M| = \sum_{\pi} \mathsf{sign}(\pi) \prod_{i} M_{i,\pi(i)}$$

perfect matchings π contribute a term, but nothing else does.

So all we need to do is compute the determinant, using, say, Gaussian elimination, and we're done.

Right?

Wrong!

The determinant of this matrix is an integer polynomial in n^2 variables x_{ij}. Worse, the Gaussian elimination takes place over a field of rational functions $\mathbb{Q}(x_{11}, \ldots, x_{nn})$.

Computing the determinant requires a cubic number of ring operations, but the ring elements have potentially exponential size.

But we can use Schwartz's lemma to determine whether the determinant is zero in polynomial time, with small error probability: we can compute determinants with numerical entries in polynomial time, avoiding the exponential description from above.

This result holds for general graphs, not just bipartite ones. The proof is significantly harder in the general case. In the end, one obtains an $O(n^2e)$ algorithm, but it's tricky.

J. Edmonds

Paths, Trees, and Flowers

Canad. J. Math. 17(1965)449–467

Edmonds was also arguably the first person to give a coherent exposition of the problem of feasible computation (albeit a whole decade after Gödel discovered the topic).

Note that the use of randomness to speed up computations leads to different kinds of algorithms.

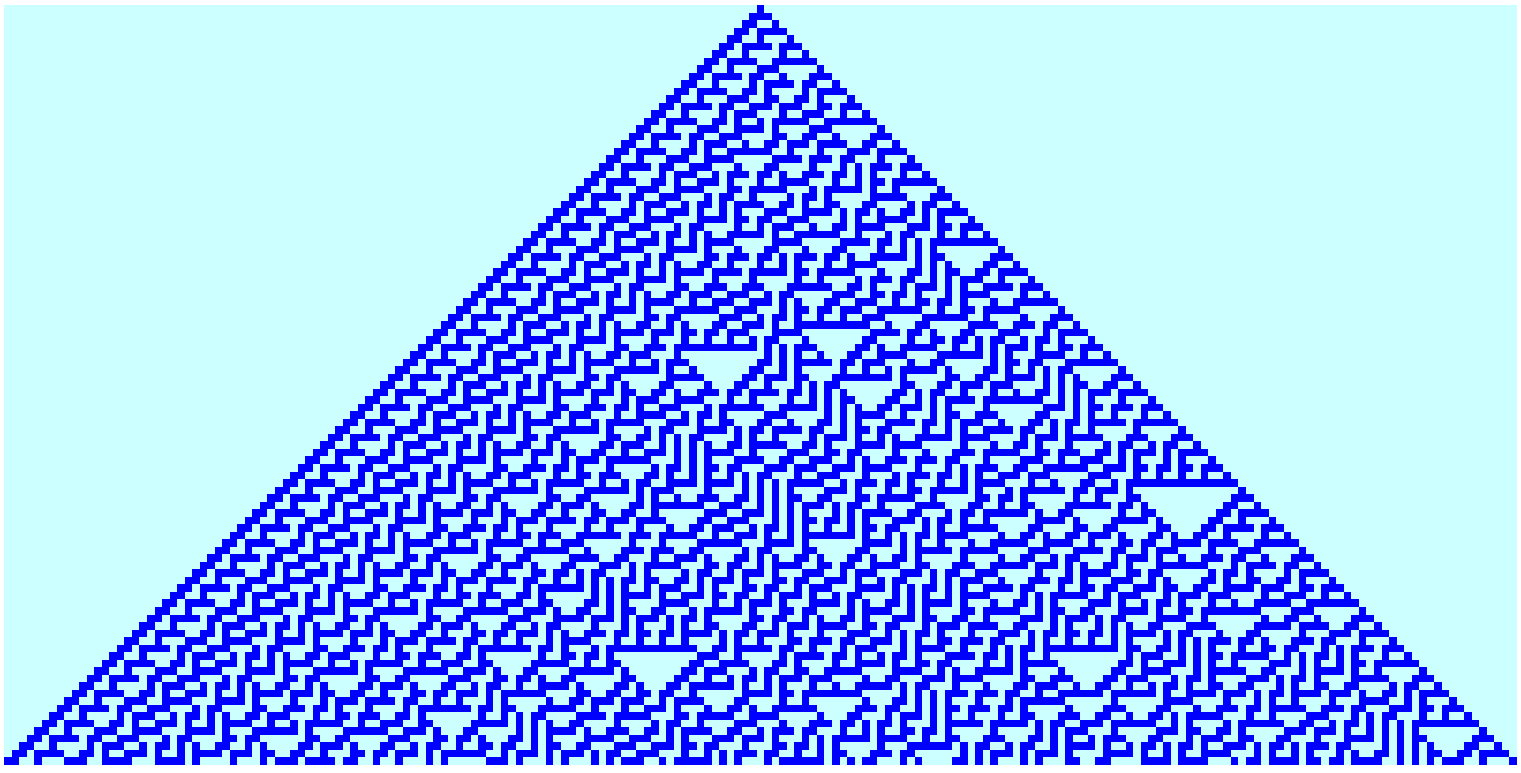
- Monte Carlo:
  may return false answers, but only with small probability; is always fast.
- Las Vegas algorithms:
  always correct, but may be slow with small probability.

And there may be no false positives or no false negatives; see below.

MC algorithms are not to be confused with Monte Carlo simulations, nowadays a staple of much high-end research: estimate the possible outcomes of an uncertain event (produced by a deterministic system that is so complicated that anything resembling an analytical solution fails to exist.

Pioneered by S. Ulam and von Neumann. Used everywhere: engineering, operations research, physics, chemistry, finance, statistics.

Instead of trying to derive properties from an explicit mathematical model, simulate the system many times using random initial conditions. Observe the outcomes, and use the LoLN to figure out behavior.



These simulations often require large numbers of random bits, so the standard method to obtain these bits is to cheat and use a PRNG.

As long as the generator is not obviously broken, that should be good enough. After all, we are dealing with nature, not a malicious opponent.

Think again. In 1992 Ferrenberg et.al. showed that a heavily used method to study phase transitions (Wolff algorithm) produces wrong results when used with standard high quality PRNGs.

The generators pass all kinds of statistical tests, but are not random enough for MC simulations.

1 Some Probabilistic Algorithms

2 **Probabilistic Primality Testing**

There are several areas where randomness is utterly critical:

- Simulations (Monte Carlo simulations)
- **Primality Testing**
- Cryptography

We know (since 2002) that primality testing can be fully derandomized while maintaining a polynomial time algorithm. But, the constants are so bad that no actual implementation uses the algorithm.

The Solovay-Strassen test (invented 1974, published 1977) was arguably the starting point of a huge amount of work in randomized algorithms.

The multiplicative subgroup of $\mathbb{Z}_m$ is

$$\mathbb{Z}_m^\star = \{\, x \in \mathbb{Z}_m \mid \gcd(x, m) = 1 \,\}$$

Definition (Quadratic Residues)

$a \in \mathbb{Z}_m^\star$ is a quadratic residue modulo m if $x^2 = a \pmod m$ has a solution over $\mathbb{Z}_m^\star$, and a quadratic non-residue otherwise.

Thus $\mathbb{Z}_m^\star$ can be partitioned into QR_m and QNR_m. For example

$$\mathsf{QR}_{17} = \{1, 2, 4, 8, 9, 13, 15, 16\}$$

$$\mathsf{QNR}_{17} = \{3, 5, 6, 7, 10, 11, 12, 14\}$$

Note that half/half split.

Another Example

Here are the solutions to $x^2 = a \pmod{m}$ for $m = 20$ (ignoring coprimality).

0	$0, 10$
1	$1, 9, 11, 19$
4	$2, 8, 12, 18$
5	$5, 15$
9	$3, 7, 13, 17$
16	$4, 6, 14, 16$

For a tiny modulus this can obviously be done by brute-force computation, just compute all squares mod m.

Question: Can we check efficiently whether a number is a quadratic residue?

Legendre Symbol

Suppose p is an odd prime and $z \in Z_p^*$. The Legendre symbol $\mathrm{LS}(z, p)$ is defined by:

$$\mathrm{LS}(z,p) = \begin{cases} +1 & \text{if } z \text{ is a quadratic residue,} \\ -1 & \text{otherwise.} \end{cases}$$

Proposition

$\mathrm{LS}(z,p) = z^{(p-1)/2} \pmod{p}$.

Write $\mathrm{M}(k)$ for the number of steps needed in multiplying two k-bit numbers.

Then the proposition provides an $O(\lg p\, \mathrm{M}(\lg p))$ algorithm to compute $\mathrm{LS}(z,p)$: use a fast exponentiation algorithm modulo p.

Unfortunately, we need p to be an odd prime here (though we can handle $p = 2$ pretty well).

Suppose $q = p_1 \cdot \ldots \cdot p_r$ where the p_i are odd primes, not necessarily distinct, z relatively prime to q. The Jacobi symbol $\left(\frac{z}{q}\right)$ is defined by:

$$\left(\frac{z}{q}\right) = \mathsf{LS}\,(z, p_1) \cdot \mathsf{LS}\,(z, p_2) \cdot \ldots \cdot \mathsf{LS}\,(z, p_r) \in \{-1, +1\}.$$

If we knew the p_i (i.e., the prime decomposition of q), then we could easily compute the Jacobi symbol according to its definition.

But what if all we know is q?

The following properties of the Jacobi symbol will help in finding an algorithm. The proof is quite elementary.

$$\left(\frac{z}{q}\right) = \left(\frac{z \bmod q}{q}\right)$$

$$\left(\frac{z_1 \cdot z_2}{q}\right) = \left(\frac{z_1}{q}\right) \cdot \left(\frac{z_2}{q}\right)$$

$$\left(\frac{1}{q}\right) = 1$$

$$\left(\frac{-1}{q}\right) = \begin{cases} -1 & \text{if } q = 3 \pmod 4, \\ +1 & \text{otherwise.} \end{cases}$$

$$\left(\frac{2}{q}\right) = \begin{cases} -1 & \text{if } q = 3, 5 \pmod 8, \\ +1 & \text{otherwise.} \end{cases}$$

We need one more ingredient for the algorithm. This is one of Gauss's favorite results, he gave several proofs for it.

Theorem (C. F. Gauss 1798)

Let p, q be two odd primes. Then we have

$$\left(\frac{p}{q}\right) = \begin{cases} -\left(\frac{q}{p}\right) & \text{if } p = q = 3 \pmod 4, \\ +\left(\frac{q}{p}\right) & \text{otherwise.} \end{cases}$$

The theorem together with the proposition makes it possible to apply mods to cut down the size of the numbers (much like in the computation of the GCD).

We may safely assume $0 < p < q$.

```
sign = 1;
while( p > 0 ) {
    while( p even ) {          // eliminating even part
        p /= 2;
        if( q mod 8 == 3, 5 )
            sign = -sign;
    }
    swap p and q;              // quadratic reciprocity
    if( p, q == 3  mod 4 )
        sign = -sign;
    p = p mod q;
}
if( q == 1 )
    return  sign
else
    return 0;                  // coprimality broken
```

First note that the algorithm maintains the invariant: $0 < p < q$, both odd.

If both numbers are coprime, they will stay so during the execution of the loop (we return 0 if it turns out that coprimality is violated). Correctness now follows from the mentioned properties of the Jacobi symbol.

For running time, note that all the numbers have at most k bits where k is the length of p and q. Hence, all the arithmetic operations are polynomial time. The same argument that shows that Euclidean algorithm is polynomial time can be used to show that the loop executes no more than $2k$ times.

With a little more effort one can show that the algorithm runs in time $O(\log p \ \log q)$.

Jacobi Example

The following table shows the computation for $p = 117$ and $q = 271$.

Result: -1.

p	q	sign
117	271	$+$
271	117	$+$
117	37	$+$
6	37	$+$
3	37	$-$
37	3	$-$
1	3	$-$

The Jacobi symbol is an extension of the Legendre symbol in the sense that whenever q is prime we have $\left(\frac{p}{q}\right) = \mathsf{LS}\,(p, q)$.

Note, however, that $\left(\frac{p}{q}\right) = +1$ no longer implies that p is a QR modulo q. For example, $\left(\frac{2}{9}\right) = \left(\frac{2}{3}\right)^2 = 1$ but $2 \in \mathsf{QNR}_9$.

Hence we now have two different ways to compute the Legendre symbol for q prime. This is the basic idea behind the Solovay-Strassen algorithm: pick z uniformly at random and compute $\left(\frac{z}{p}\right)$ in two ways, yielding LS_1 and LS_2.

If $LS_1 \neq LS_2$ then p is not prime; if $LS_1 = LS_2$ then p is prime with a certain high probability.

Solovay-Strassen Primality Testing Algorithm
Input: n

pick z uniformly at random, $1 < z < n$

if $\gcd(z, n) > 1$
then return NO

$LS_1 = z^{(n-1)/2} \pmod{n}$

$LS_2 = \left(\frac{z}{n}\right)$

if $LS_1 \neq LS_2$
then return NO
else return YES?

An odd composite number n such that $z^{(n-1)/2} = \left(\frac{z}{n}\right) \pmod{n}$ is called a Euler-Jacobi pseudoprime (in base z).

Unfortunately, these things exist, though they are relatively rare: for $n \leq 10000$ and $z = 2$ we get

$$561, 1105, 1729, 1905, 2047, 2465, 3277, 4033, 4681, 6601, 8321, 8481$$

For base $z = 3$ we get

$$121, 703, 1729, 1891, 2821, 3281, 7381, 8401, 8911$$

By choosing the base at random (think: a generic number, without special properties), we avoid these problems.

Theorem (Solovay-Strassen)

If n is prime, the Solovay-Strassen test returns YES?; otherwise, the test returns NO with probability at least $1/2$.
The running time is $O(\log n\, \mathsf{M}(\log n))$.

Proof.

Suppose n is composite and consider the set S of bad choices for our algorithm where we get a false positive:

$$S = \left\{ z \in \mathbb{Z}_n^\star \;\middle|\; z^{(n-1)/2} = \left(\frac{z}{n}\right) \pmod{n} \right\}.$$

Claim: $|S| \leq |\mathbb{Z}_n^\star|/2$.

First note the S is a subgroup because of the multiplicativity property of the Jacobi symbol, so $|S|$ divides $|\mathbb{Z}_n^\star|$.

It remains to show that S is proper. Assume otherwise, so that in particular $z^{n-1} = 1 \pmod{n}$ for all $z \in \mathbb{Z}_n^\star$.

Consider the prime factor p^e of n with maximal exponent e and set $m = n/p^e$.

Pick a generator g for $\mathbb{Z}_{p^e}^\star$ (which is known to exist).

By Chinese Remainder, there is an element $a \in \mathbb{Z}_n^\star$ such that $a = g \pmod{p^e}$ and $a = 1 \pmod{m}$. From our assumptions, $a^{n-1} = 1 \pmod{n}$, and therefore $g^{n-1} = 1 \pmod{p^e}$, so that $\mathrm{ord}(g; \mathbb{Z}_{p^e}^\star) \mid n-1$.

On the other hand, $\mathrm{ord}(g; \mathbb{Z}_{p^e}^\star) = p^{e-1}(p-1)$.

If $e > 1$ then we have $p^{e-1}(p-1) \mid n-1$, a contradiction.

If $e = 1$ then n must be square-free and contains at least two prime factors. Clearly $\left(\frac{a}{p}\right) = -1$ since g is a generator, and $\left(\frac{a}{q}\right) = 1$ for all the other prime factors of n. Hence $a^{(n-1)/2} = \left(\frac{a}{n}\right) = -1 \pmod{n}$. But that contradicts $a = 1 \pmod{m}$.

It follows that if n is composite then the probability of the Solovay-Strassen algorithm returning NO is at least $1/2$.

□

Of course, $|S|$ may be smaller. E.g., for $n = 703$ we have $|S| = 162$ whereas $|\mathbb{Z}_n^\star| = 648$. For $n = 1905$ we have $|S| = 56$ whereas $|\mathbb{Z}_n^\star| = 1008$.

Alas, for $n = 2465$ our bound is tight: $|S| = 896$ where $|\mathbb{Z}_n^\star| = 1792$.

This algorithm expands on the idea of using Fermat's Little Theorem to reject non-primes: if we can find an a such that

$$a^{n-1} \neq 1 \bmod n$$

then we know that n is not prime.

For example, a Fermat number is a number of the form $F_n = 2^{2^n} + 1$. Fermat claimed that these numbers are all prime. This is true for the first 4:

$$F_n = 3, 5, 17, 65537$$

But how about $F_4 = 4294967297$? It turns out that $2^{F_4-1} = 1 \bmod F_4$ but $3^{F_4-1} = 3029026160 \bmod F_4$.

Since exponentiation modulo some fixed number is easy, this provides a cheap proof of non-primality.

The Problem: which base a should we pick?

If we had some superb algebraic theory of primality, there might be some powerful theorem that tells us how to choose a. Alas, no such luck. In the absence of any good ideas, a random choice might work.

- pick a in $[2, n-2]$ at random
- if $\gcd(a, n) \neq 1$ return NO
- compute $b = a^{n-1} \bmod n$
- If $b = 1$ return YES? else return NO

No false negatives, but potentially false positives.

What Could Go Wrong?

Problems arise when n is composite but we happen to choose a in the "bad" set

$$C_n = \{\, a \in \mathbb{Z}_n^\star \mid a^{n-1} = 1 \pmod{n} \,\}.$$

To get a performance guarantee, we need to bound the size of C_n.

Clearly, C_n is a subgroup of $\mathbb{Z}_n^\star$, and Fermat's little theorem says that n prime implies $\mathbb{Z}_n^\star = C_n$. By Lagrange's theorem, a proper subgroup has size at most $|C_n| \leq |\mathbb{Z}_n^\star|/2$.

So far, this is analogous to the Solovay-Strassen argument.

Unfortunately, it may happen that $\mathbb{Z}_n^\star = C_n$ but n fails to be prime.

These numbers are called Carmichael numbers:

$$561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841,$$
$$29341, 41041, 46657, 52633, 62745, 63973, 75361, \ldots$$

Alas, there are infinitely many such numbers.

Moreover, the $\gcd$ test in the algorithm is of no help either: consider $n = pq$; then $\phi(n)/n = 1 - (p + q - 1)/pq$, so the likelihood of picking an element in $\mathbb{Z}_n^\star$ is not bounded away from 1.

Proposition

If n is prime or Carmichael, the Fermat test returns YES?. Otherwise, the algorithm returns NO with probability at least $1/2$. The running time is $O(\lg n \, \mathsf{M}(\lg n))$.

So we are almost there.

Of course, this is not good enough for, say, cryptography.

For a positive integer x write $\nu_2(x)$ for the largest e such that $2^e \mid x$. Write $\text{odd}(x)$ for $x2^{-\nu_2(x)}$, the odd part of x.

Suppose $n \geq 5$ is odd, let $k = \nu_2(n-1)$ and $m = \text{odd}(n-1)$. If n is indeed prime, then $\mathbb{Z}_n^\star$ is a cyclic group of order $n-1$. Hence, for any $1 \leq a < n$, $\text{ord}(a) = m_0\, 2^i$ for some $m_0 \mid m$ and $0 \leq i \leq k$. But then

$$a^{m2^i} = 1 \pmod{n},$$

so we can search for the appropriate i.

Algorithm

let $n - 1 = 2^k \cdot m$
pick a at random, $2 \leq a \leq n - 2$
if $\gcd(a, n) > 1$ **then** return NO

compute $b = a^m \bmod n$
if $b = \pm 1$ **then** return YES?

for $i = 1, \ldots, k - 1$ **do**
    compute $b = b^2 \bmod n$ // $b = a^{m2^i}$
    if $b = -1$ **then** return YES?
    if $b = +1$ **then** return NO
return NO

A clever way of searching for an exponent i as above.

Theorem

If n is prime, the Miller-Rabin test returns YES?. If n is composite, the algorithm returns NO with probability at least $1/2$.

The running time is $O(\lg n \, \mathsf{M}(\lg n))$.

Incidentally, the first version of the algorithm due to Miller was not randomized, it used the (extended) Riemann Hypothesis to show that the search can be handled in polynomial time.

Rabin realized that one can use randomization to obtain a highly efficient algorithm, without RH.

Batten down the hatches.

It is clear that n prime results in output YES?

So let $n \geq 5$ be odd and composite. Define

$$B = \{\, a \in \mathbb{Z}_n^\star \mid a^m = 1 \pmod{n} \vee a^{m2^i} = -1 \pmod{n},\, i < k \,\}$$

to be the set of "bad" choices for Miller-Rabin that make n appear prime.

Note that $B \subseteq C_n$, the bad set from the Carmichael section, so if n fails to be a Carmichael number we have $|B| \leq \phi(n)/2 < n/2$.

Assume then that n is Carmichael. It can be shown that n must be square-free, so that $n = p_1 \cdots p_r$.

It is convenient to inflate and simplify B a bit. To this end let

$$\ell = \min\Big(\nu_2(p_i - 1) \mid i \in [r] \Big)$$

whence $2^\ell \mid p_i - 1$ for all $i \in [r]$.

Define $e = m2^{\ell-1}$ and let

$$C^{\pm} = \{\, a \in \mathbb{Z}_n^\star \mid a^{m2^{\ell-1}} = \pm 1 \pmod{n} \,\}$$

Note that $C^{\pm}$ is a subgroup of $\mathbb{Z}_n^\star$.

We claim that $B \subseteq C^{\pm}$.

This is clear for $a^m = 1 \pmod{n}$, so consider $a^{m2^i} = -1 \pmod{n}$ for some $i < k$. We have $a^{m2^i} = -1 \pmod{p}$ for all $p \mid n$ and thus $\nu_2(\mathsf{ord}(a, \mathbb{Z}_p^\star)) = i + 1$. Since p is prime, $2^{i+1} \mid p - 1$ and so by definition $\ell \geq i + 1$. Hence $a \in C^{\pm}$.

By the Chinese Remainder Theorem we have

$$\mathbb{Z}_n^\star \simeq \mathbb{Z}_{p_1}^\star \times \cdots \times \mathbb{Z}_{p_r}^\star,$$

so it suffices to count the number of solutions of the modular equation $x^e = 1 \pmod{p}$, for all $p \mid n$, where $e = m2^{\ell-1}$.

But p is prime, so we have

$$\begin{aligned} \#\text{sol } x^e = 1 \pmod p &= \gcd(\phi(p), e) \\ &= \gcd(p-1, m2^{\ell-1}) \\ &= 2^{\ell-1} \gcd(p-1, m) \end{aligned}$$

It follows that the $+$ component of $C^{\pm}$ has size

$$\prod_{p|n} 2^{\ell-1} \gcd(p-1, m)$$

A similar argument shows that the minus component has the same size, so the total cardinality is

$$|C^{\pm}| = 2 \prod_{p|n} 2^{\ell-1} \gcd(p-1, m)$$

But then

$$|C^{\pm}|/\phi(n) = 2\prod_{p|n} 2^{\ell-1}\ \gcd(p-1,m)/(p-1)$$

$$= 2\prod_{p|n} 1/(2\alpha_p) \leq 1/2$$

Done.

□

Here is a hack I found online that is supposed to improve performance of Miller-Rabin.

- Choose the first value of a at random, but small.
- Run Miller-Rabin with that a.
- Increment a and repeat a suitable number of times.

So the point is to avoid computation of future random numbers to speed things up. Also note the weird "a small" idea.

This kind of skulduggery is rather hard to analyze.