Randomized Algorithms

A randomized algorithm is a deterministic algorithm with the extra ability of making random choices during the computation that are independent of the input values. Using randomization, some worst-case scenarios may be hidden so that it only occurs with a small probability, and so the expected runtime is better than the worst-case runtime. We illustrate this idea by examples.

Example 1. (Quicksort) We consider the following two versions of the well-known Quicksort algorithm.

Deterministic Quicksort
Input: a list \( L = (a_1, \ldots, a_n) \) of integers;
if \( n \leq 1 \) then return \( L \)
else begin
1: let \( i := 1 \);
   let \( L_1 \) be the sublist of \( L \) whose elements are \( < a_i \);
   let \( L_2 \) be the sublist of \( L \) whose elements are \( = a_i \);
   let \( L_3 \) be the sublist of \( L \) whose elements are \( > a_i \);
   recursively Quicksort \( L_1 \) and \( L_3 \);
   return \( L \) as the concatenation of the lists \( L_1, L_2, \) and \( L_3 \)
end.

Randomized Quicksort
{same as deterministic Quicksort, except line 1:}
1: choose a random integer \( i, 1 \leq i \leq n \);

It is easy to see that both algorithms sort list \( L \) correctly. We consider the runtime of the algorithms. Let \( T_d(n) \) denote the maximum number of comparisons (of elements in \( L \)) made by the deterministic Quicksort algorithm on an input list \( L \) of \( n \) integers. It is clear that the worst case occurs when \( L \) is reversely sorted and, hence, \( a_1 \) is always the maximum element in \( L \). Thus, we get the following recurrence inequality:

\[
T_d(n) \geq cn + T_d(n - 1),
\]

for some constant \( c > 0 \). This observation yields \( T_d(n) = \Omega(n^2) \).

Let \( T_r(n) \) be the expected number of comparisons made by the randomized Quicksort algorithm on an input list \( L \) of \( n \) integers. Let \( s \) be the size of \( L_1 \). Since the integer \( i \) is a random number in \( \{1, \ldots, n\} \), the value \( s \) is actually a random variable with the property \( \Pr[s = j] = 1/n \) for all \( j = 0, \ldots, n - 1 \) (assuming that all elements in \( L \) are distinct). Thus, we obtain the following recurrence inequality:

\[
T_r(n) \leq cn + \frac{1}{n} \sum_{j=0}^{n-1} [T_r(j) + T_r(n - 1 - j)],
\]
for some constant $c > 0$. Solving this recurrence inequality, we get $T_r(n) = O(n \log n)$.

Using essentially the same proof, we can show that the deterministic Quicksort algorithm also has an expected runtime $O(n \log n)$, under the uniform distribution over all lists $L$ of $n$ integers. This result, however, is of quite a different nature from the result that $T_r(n) = O(n \log n)$. The expected runtime $O(n \log n)$ of the randomized Quicksort algorithm is the same for any input list $L$, since the probability distribution of the random number $i$ is independent of the input distribution. On the other hand, the expected runtime $O(n \log n)$ of the deterministic Quicksort algorithm is only correct for the uniform distribution on the input domain. In general, it varies when the input distributions change.

In the above, we have seen an example in which the randomized algorithm does not make mistakes and it has an expected runtime lower than the worst-case runtime. We achieved this by changing a deterministic parameter $i$ into a randomized parameter that transforms a worst-case input into an average-case input. (Indeed, all inputs are average-case inputs after the transformation.) In the following, we show another example of the randomized algorithms which run faster than the brute-force deterministic algorithm, but might make mistakes with a small probability. In this example, we avoid the brute-force exhaustive computation by a random search for witnesses (for the answer NO). A technical lemma shows that such witnesses are abundant and so the random search algorithm works efficiently.

**Example 2. (Determinant of a Polynomial Matrix, or DPM).** In this example, we consider matrices of multi-variable integer polynomials. An $n$-variable integer polynomial $Q(x_1, \ldots, x_n)$ is said to be of the standard form if it is written as the sum of terms of the form $c_k x_1^{d_1} x_2^{d_2} \cdots x_k^{d_k}$. For instance, the following polynomial is in the standard form:

$$Q(x_1, \ldots, x_5) = 3x_1^2 x_2 x_4 - 4x_1^3 x_2 x_5 + 3x_3 x_4 x_5^3.$$  

A polynomial $Q$ in the standard form has degree $d$ if the maximum of the total degree of each term is $d$. For instance, the above polynomial $Q$ has degree 7.

The problem DPM is the following: Given an $m \times m$ matrix $Q = [Q_{i,j}]$ of $n$-variable integer polynomials, and an $n$-variable integer polynomial $Q_0$, with $Q_0$ and all $Q_{i,j}$ in $Q$ written in the standard form, determine whether $\det(Q) = Q_0$, where $\det(Q)$ denotes the determinant of $Q$. Even for small sizes of $m$ and $n$, the brute-force evaluation of the determinant of $Q$ appears formidable. On the other hand, the determinant of an $m \times m$ matrix over integers in $\{-k, -k+1, \ldots, k-1, k\}$ can be computed in time polynomial in $(m + \log k)$. (In fact, as discussed in Chapter 6, this problem is solvable in $NC^2$.) In the following we present a randomized algorithm for DPM, based on the algorithm for evaluating determinants of integer matrices.\(^1\)

---

1In the rest of this example, $|u|$ denotes the absolute value of an integer $u$; do not confuse
Randomized Algorithm for DPM

Input: $Q = [Q_{i,j}]$, $Q_0$, $\epsilon$;
let $d' := \max\{\text{degree of } Q_{i,j} : 1 \leq i, j \leq m\}$;
let $d_0 := \text{degree of } Q_0$;
let $d := \max\{md', d_0\}$;
let $k := \lceil -\log \epsilon \rceil$;
for $i := 1$ to $k$ do
   choose $n$ random integers $u_1, \ldots, u_n$ in the range $\{-d, \ldots, d\}$;
   if $\det(Q(u_1, \ldots, u_n)) \neq Q_0(u_1, \ldots, u_n)$ then output \texttt{NO} and halt;
end
output \texttt{YES} and halt.

Suppose the absolute values of the coefficients of the polynomials $Q_{i,j}$ and $Q_0$ are bounded by $r$. Then, the absolute value of each $Q_{i,j}(u_1, \ldots, u_n)$ is bounded by $d'^{(n+d)} r$, and so $\det(Q(u_1, \ldots, u_n))$ can be computed in time polynomial in $(m + n + d + \log r)$. Thus this algorithm always halts in polynomial time. It, however, might make mistakes. We claim nevertheless that the error probability is smaller than $\epsilon$:

1. If $\det(Q) = Q_0$, then the above algorithm always outputs \texttt{YES}.
2. If $\det(Q) \neq Q_0$, then the above algorithm outputs \texttt{NO} with probability $\geq 1 - \epsilon$.

The claim (1) is obvious since the algorithm outputs \texttt{NO} only when a witness $(u_1, \ldots, u_n)$ is found such that $\det(Q(u_1, \ldots, u_n)) \neq Q_0(u_1, \ldots, u_n)$. The claim (2) follows from the following lemma. Let $A_{n,m} = \{(u_1, \ldots, u_n) : |u_i| \leq m, 1 \leq i \leq n\}.$

**Lemma.** Let $Q$ be an $n$-variable integer polynomial of degree $d$ that is not identical to zero. Let $m \geq 0$. Then, $Q$ has at most $d \cdot (2m + 1)^{n-1}$ zeros $(u_1, \ldots, u_n)$ in $A_{n,m}$.

**Proof.** The case $n = 1$ is trivial since a 1-variable nonzero polynomial $Q$ of degree $d$ has at most $d$ zeros. We show the general case $n > 1$ by induction. Write $Q$ as a polynomial of a single variable $x_1$, and let $d_1$ be the degree of $Q$ in $x_1$. Let $Q_1$ be the coefficient of $x_1^{d_1}$ in $Q$. Then $Q_1$ is an $(n-1)$-variable polynomial of degree $\leq d - d_1$.

By induction, since $Q_1$ is not identical to zero, it has at most $(d-d_1)(2m+1)^{n-2}$ zeros in $A_{n-1,m}$. For any $(v_2, \ldots, v_n)$ in $A_{n-1,m}$, if $(v_2, \ldots, v_n)$ is not a zero of $Q_1$, then $Q(x_1, v_2, \ldots, v_n)$ is a 1-variable polynomial not identical to zero and, hence, has at most $d_1$ zeros $u_1$, with $|u_1| \leq m$. If $(v_2, \ldots, v_n)$ is a zero of $Q_1$, then $Q(x_1, v_2, \ldots, v_n)$ might be identical to zero and might have $2m + 1$ zeros $u_1$, with $|u_1| \leq m$. Together, the polynomial $Q$
has at most
\[d_1 \cdot (2m + 1)^{n-1} + (2m + 1) \cdot (d - d_1) \cdot (2m + 1)^{n-2} = d \cdot (2m + 1)^{n-1}\]
zeros in \(A_{n,m}\).

The above lemma shows that if \(\det(Q) \neq Q_0\) then the probability that a random \((u_1, \ldots, u_n)\) in \(A_{n,d}\) satisfies \(\det(Q(u_1, \ldots, u_n)) = Q_0(u_1, \ldots, u_n)\) is less than \(1/2\). Thus, the probability that the algorithm fails to find a witness in \(A_{n,d}\) for \(k\) times is \(\leq 2^{-k} \leq \epsilon\).