Tags: Complexity Theory *Written By: Nimitt*

Randomized Complexity Classes

This Daily Computer introduces randomized algorithms: algorithms that use randomness during their execution, and randomized complexity classes: **RP**, **BPP** and **ZPP**.

Randomized Algorithms. A randomized algorithm is an algorithm that uses randomness during execution. Hence, on the same input, the algorithm may behave differently across different runs. Formally, a randomized algorithm can be viewed as a collection of deterministic algorithms $(A_r)_r$ indexed by random strings $r \in \{0,1\}^s$. On input x, the algorithm samples r uniformly at random and executes $A_r(x)$. Consequently, the running time and/or output become random variables. We can divide these algorithms in two categories:
Las Vegas Algorithms. A Las Vegas algorithm always produces the correct output, but its running time is a random variable. Example: randomized QuickSort.
Monte Carlo Algorithms. A Monte Carlo algorithm runs within bounded time, but its output may be incorrect with small probability. Example: the Miller–Rabin primality test.

Zero-error Probabilistic Polynomial **ZPP** is the class of decision problems for which there exists a randomized algorithm which
(i) always returns correct output, and
(ii) has polynomial expected running time.

Randomized Polynomial. **RP** is the class of decision problems for which there exists a polynomial-time randomized algorithm satisfying:
(i) If $x \notin L$ (NO), the algorithm always rejects.
(ii) If $x \in L$ (YES), the algorithm accepts with probability at least $1/2$.

Thus, **RP** algorithms have bounded *one-sided error*. The success probability can be set to arbitrary constant greater than $1/2$ since it can be amplified by repetition. Running the algorithm independently k times and accepting if any run accepts gives failure probability at most 2^{-k}.

Bounded-error Probabilistic Polynomial. **BPP** is the class of decision problems for which there exists a polynomial-time randomized algorithm satisfying:
(i) If $x \notin L$ (NO), the algorithm rejects with probability at least $2/3$.
(ii) If $x \in L$ (YES), the algorithm accepts with probability at least $2/3$.

Hence, the **BPP** algorithms have a bounded *two-sided error*. The constants $2/3$ and $1/3$ are arbitrary. Similar to RP, by repeating the algorithm independently and taking a majority vote, the error probability can be reduced exponentially using Hoeffding inequality. Running a **BPP** algorithm independently k times and returning the majority answer yields error probability

$$\Pr[\text{majority is wrong}] \leq e^{-\Omega(k)}.$$

Thus, randomized algorithms can achieve extremely small error probabilities while still remaining polynomial-time.

Theorem 1. $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.

Proof. Suppose $L \in \mathbf{RP} \cap \mathbf{coRP}$. Then there exists an RP algorithm that correctly recognizes YES instances (let A_1) and a coRP algorithm that correctly recognizes NO instances (let A_2). Consider the algorithm that runs both algorithms repeatedly $A_1, A_2, A_1, A_2, \ldots$ until one gives a definitive output. This yields a correct answer in an expected constant number of runs, giving an algorithm with expected polynomial running time for L, hence $L \in \mathbf{ZPP}$.

Conversely, if $L \in \mathbf{ZPP}$, then there exists a randomized algorithm with expected running time $p(n)$ (let A). Consider the algorithm that runs A for $10p(n)$ steps, if the algorithm has not terminated, output NO. The algorithm only makes an error on YES instances. Moreover, the probability that running time exceeds $10p(n)$ is at most $1/10$ (using Markov inequality), giving an **RP** algorithm for L. Similarly, returning YES if not terminated after $10p(n)$ steps gives a **coRP** algorithm. □

Containments. Some fundamental containments are:

$$\mathbf{P} \overset{1}{\subseteq} \mathbf{RP} \overset{2}{\subseteq} \mathbf{BPP} \overset{3}{\subseteq} \mathbf{PSPACE}.$$

1. A polynomial algorithm is a randomized algorithm without using any randomness ($s = 0$).
2. For NO instances an **RP** algorithm rejects with probability ($1 > 2/3$). For YES instances, probability can be amplified to $> 2/3$ by repetition.
3. A **BPP** algorithm uses polynomially many random bits, so for an input x we can enumerate all random strings r and simulate the deterministic computation $A_r(x)$. Using polynomial space, we count how many random strings lead to acceptance and accept iff a majority of them do; hence $\mathbf{BPP} \subseteq \mathbf{PSPACE}$.

Remark. It is unknown whether any of these containments are strict.

RP vs NP. **RP** is closely related to **NP**. In fact,

$$\mathbf{RP} \subseteq \mathbf{NP}.$$

Indeed, since an **RP** algorithm accepts with positive probability, then there exists some random string causing acceptance, which acts as a certificate.

Remark. However, whether $\mathbf{NP} \subseteq \mathbf{BPP}$ is one of the major open problems in complexity theory.