CMPUT 675: Randomized Algorithms **Fall 2005**

Lecture 1 Introduction : Sep 8, 2005

Lecturer: Mohammad R. Salavatipour *Scribe: Mohammad R. Salavatipour*

1.1 Introduction

What is a randomized algorithm? Put it simply, any algorithm that makes random choices during its execution. We assume these algorithm have access to a source of fair random bits. Why do we study and use randomized algorithms? There are several factors, some of the more important ones are:

Speed: Randomized algorithms are usually fast, sometimes much faster than any deterministic algorithm for the same problem. Even if the asymptotic running time is the same as the deterministic one it can be faster in practice. For example QuickSort, though has worse-case running time worse than that of MergeSort or Heapsort, it is the fastest algorithm in practice. The disadvantage of some randomized algorithms (like QuickSort) is that sometimes the running time is more than usual. Also, some randomized algorithms produce a solution that is correct with high (but smaller than 1) probability, so there is a chance that the solution is not correct.

Simplicity: Even if the known randomized algorithm is not faster than the deterministic one, it is often simpler.

Only thing we can do: In some situations, randomized algorithms are the only algorithms we know. For example, in some protocols or online algorithms (in the presence of an adversary) randomness helps to defeat the adversary, whereas any deterministic algorithm can be defeated by an adversary. Or, to check if a multivariable polynomial is identical to zero we only know randomized algorithms. Note that the description of the polynomial might be implicit, e.g. the determinant of a matrix whose entries contain different variables.

Randomized algorithms should not be confused with probabilistic analysis of algorithms and average case analysis: in the average case analysis we assume some probability distribution (e.g. uniform) on input and then analyze the running time based on the assumption that the input is drawn from that probability distribution (e.g. average case analysis of QuickSort). However, when we analyze randomized algorithms we do not assume a particular (random) distribution on input. The analysis must hold for all inputs.

1.2 An Example: Comparing strings with communication costs

Suppose that Alice and Bob each have an n-bit binary string, call them a (for Alice) and b (for Bob). They want to find out if these two strings are equal. There is a communication link between them over which they can send bits. It is enough if one of them figures out whether $a = b$ or not. However, there is a cost associated with each bit transmission. Clearly, any deterministic algorithm needs to use n bits of communication (Why?). Here we give a randomized algorithms which uses only $O(\log n)$ bits and has a very high success probability. The idea behind the solution is general: instead of comparing two objects from a large universe, map them into two objects from a smaller universe and then compare those two. These two smaller objects behave as the fingerprint of the two larger ones.

We will fix the number t later. Alice picks a prime number $p \leq t$ uniformly at random and then sends both p and $a \bmod p$ to Bob. Bob computes $b \bmod p$ and if it is equal to $a \bmod p$ then says they are equal. Otherwise he concludes that they are not equal.

Clearly if $a = b$ then this protocol gives the correct answer (since $a \bmod p = b \bmod p$). However, if $a \neq b$ there is a chance that this algorithm give a wrong answer. We upper bound this probability (and therefore find an upper bound on the error probability of the algorithm).

Lemma 1.1 *With $t = c \cdot n \ln n$ for some large constant c:* $\Pr[a \bmod p = b \bmod p] \in O(\frac{1}{c})$.

Proof: Since $|a - b| \leq 2^n$ and since every prime is at least 2, there are at most n primes that divide $|a - b|$. This is an upper bound on the number of "dangerous" primes, i.e. those that if we select the algorithm will fail. The Prime Number Theorem says that the number of primes up to x is about $\frac{x}{\ln x}$. Therefore, there are $\frac{t}{\log t}$ primes that we can choose, out of which at most n are dangerous. Thus:

$$\begin{aligned} \Pr[failure] = \Pr[a \bmod p = b \bmod p] &= \Pr[p \text{ divides } |a-b|] \\ &\leq \frac{n}{\frac{t}{\ln t}} \\ &= \frac{n[\ln n + \ln\ln n + \ln c]}{c \cdot n \ln n} \in O(\frac{1}{c}) \end{aligned}$$

■

Therefore, the algorithm succeeds with probability at least $1 - \epsilon$ where ϵ is made arbitrary close to 0 by making c large enough.

1.3 Paradigms for Randomized Algorithms

Avoiding adversarial input (random reordering): For deterministic algorithms, it is not difficult to produce inputs that cause the algorithm to perform poorly. Randomness can be used to avoid these situations . For example, compare the (deterministic) quicksort (where the pivot is always the first element of the list) with randomized quicksort. Note that adversarial input could be a well-structured input. For example, a sorted list causes the deterministic quicksort algorithm to perform badly. The importance of foiling adversaries becomes more vivid in online algorithms.

Fingerprinting and hashing: We use short fingerprints generated randomly for larger objects. Then we only need to compare smaller objects (fingerprints). This is the idea behind hashing too where a small set S of elements drawn from a large universe is mapped into a small set from a small universe.

Random sampling: Used in graph algorithms, approximate counting, median finding, and several other applications. The idea is that a random "typical" sample from a large population is a representation of the whole population.

Load balancing: Randomness can be used in problems where we want to spread the load among a common resources, such as communication links.

Rapidly mixing Markov chains: Is typically used to count the number of combinatorial objects with certain properties (e.g. matchings, colorings). There are approximate algorithms based on random sampling.

However, generating a random sample is not always easy. In this technique, we show that starting from an arbitrary sample and then doing a suitable random walk around this sample in the space takes us to a random sample with high probability in a short amount of time.

Symmetry breaking: Randomness can be quite helpful to keep everybody from doing the same thing. For example, protocol in Ethernet card uses randomness to access the shared medium. Randomness is also used to avoid deadlock in distributed systems.

Probabilistic Method and existential proofs: The probabilistic method is a powerful technique in combinatorics. The basic probabilistic method can be described as follows: in order to prove the existence of a combinatorial structure or object with certain properties, we construct an appropriate probability space and show that a randomly chosen element in this space has the desired properties with positive probability. In most applications, this probability is not only positive, but is actually high and frequently tends to 1 as the parameters of the problem tend to infinity. This yields a randomized algorithm for constructing an object with the desired properties. Sometimes we need to use more complicated techniques to turn these existential results into randomized algorithms.