CS 170 HW 12

Due on 2018-04-23, at 11:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★★★★) One-Sided Error and Las Vegas Algorithms

An RP algorithm is a randomized algorithm that runs in polynomial time and always gives the correct answer when the correct answer is 'NO', but only gives the correct answer with probability greater than 1/2 when the correct answer is 'YES'.

(a) Prove that every problem in RP is in NP (i.e., show that $RP \subseteq NP$).

(b) In the last homework, we saw an example of a *Las Vegas Algorithm*: a random algorithm which always gives the right solution, but whose runtime is random. A ZPP algorithm is a Las Vegas algorithm which runs in expected polynomial time (ZPP stands for Zero-Error Probabilistic Polytime). Prove that if a problem has a ZPP algorithm, then it has an RP algorithm.

3 (★★★★★) Polynomial Identity Testing

Suppose we are given a polynomial $p(x_1, \ldots, x_k)$ over the reals such that p is not in any normal form. For example, we might be given the polynomial:

$$p(x_1, x_2, x_3) = (3x_1 + 2x_2)(2x_3 - 2_x1)(x_1 + x_2 + 3x_3) + x_2$$

We want to test if p is equal to 0, meaning that $p(a_1, a_2, \ldots, a_k) = 0$ on all real inputs $a_1, a_2, \ldots a_k$. This suggests a simple algorithm to check if p equals 0: test p on some $a_1, \ldots, a_k$ and say p equals 0 if and only if $p(a_1, \ldots, a_k) = 0$. However, this algorithm will fail if $p(a_1, \ldots, a_k) = 0$ and p does not equal 0. We will show through the following questions that if the degree of p is d, then for any finite set of reals S, by randomly choosing $a_1, \ldots, a_k \in S$, we get $Pr[p(a_1, \ldots, a_k) = 0] \leq \frac{d}{|S|}$. So by choosing a large enough subset, S, we are very likely to correctly determine whether p is equal to 0 or not. Note that the degree of, say, $p(x, y) = x^2y^2 + x^3$ is 4, since the monomial x^2y^2 has degree $4 = 2 + 2$. You may assume here that all basic operations on the reals take constant time.

(a) First let's see why a probabilistic algorithm might be necessary. Give a naive deterministic algorithm to test if a given polynomial p is equal to 0. What is the worst-case runtime of your algorithm?

(b) Show that if p is univariate and degree d, then $Pr[p(a_1) = 0] \leq \frac{d}{|S|}$

(c) Show by induction on k that for all polynomials p on k variables:

$$Pr[p(a_1, \ldots, a_k) = 0] \leq \frac{d}{|S|}$$

Hint: Write $p(x_1, \ldots, x_n) = \sum_{i=0}^{d} p_i(x_1, \ldots, x_{n-1}) x_n^i$. Consider the largest i such that the polynomial p_i is not equal to 0, and check $Pr[p_i(x_1, \ldots, x_{n-1}) = 0]$. Then use the fact that $Pr[A] = Pr[A \cap B] + Pr[A \cap \bar{B}] = Pr[A|B]Pr[B] + Pr[A|\bar{B}]Pr[\bar{B}] \leq Pr[B] + Pr[A|\bar{B}]$ for events A and B (and complement event $\bar{B}$).

(d) Use the bound you found in the last part to show a way to check if two polynomials p and q are identical (i.e., yield the same outputs on all inputs).

4 (★★★) Universal Hashing

Let $[m]$ denote the set $\{0, 1, ..., m-1\}$. For each of the following families of hash functions, say whether or not it is universal, and determine how many random bits are needed to choose a function from the family.

(a) $H = \{h_{a_1,a_2} : a_1, a_2 \in [m]\}$, where m is a fixed prime and

$$h_{a_1,a_2}(x_1, x_2) = a_1 x_1 + a_2 x_2 \mod m$$

Notice that each of these functions has signature $h_{a_1,a_2} : [m]^2 \to [m]$, that is, it maps a pair of integers in $[m]$ to a single integer in $[m]$.

(b) H is as before, except that now $m = 2^k$ is some fixed power of 2.

(c) H is the set of all functions $f : [m] \to [m-1]$.

5 (★★★) Streaming Algorithms

In this problem, we assume we are given a stream of integers x_1,x_2, ..., and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to store after each new integer.

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that using $log(N)$ bits of memory, we can compute whether the sum of all integers seen so far is divisible by some fixed number N.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

(d) Assume N is not prime, and we are given the prime factorization of N as $p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible. Represent the number of bits you use in terms of the prime factorization of N.