Problem Set 1 Solutions

Point totals are in the margin; the maximum total number of points was 52. You should use these solutions as a rough guide to the level of detail expected in your own solutions.

1. Testing commutativity in groups

(a) Note first that some generator g_i must lie outside H, since H is a proper subgroup. So let i be the minimum index such that $g_i \notin H$. Then we can write the random product as $h = a \circ g_i^{b_i} \circ c$, where $a \in H$. Now I claim that, for any choice of a and c, we cannot have $h \in H$ for both $b_i = 0$ and $b_i = 1$. For if $c \in H$ then $h = a \circ g_i \circ c \notin H$, and if $c \notin H$ then $h = a \circ c \notin H$. Hence $\Pr[h \in H] \leq \frac{1}{2}$. *5pts*

Common Issues: Many people claimed that if $g_i \notin H$, then any term with g_i^1 must not be in H. This is clearly false, since for any subgroup H, $1 \in H$, and $g \circ g^{-1} = 1 \in H$, even if $g \notin H$. Some students attempted a proof via an induction on k. This typically did not produce a rigorous solution as either the induction hypothesis was not well formulated or there were issues in the induction step because of a change in the underlying probability space.

(b) When G is not abelian, its center Z is a proper subgroup. Hence by part (a), $\Pr[h \notin Z] \geq \frac{1}{2}$. Assuming $h \notin Z$, we have that $C(h)$ is a proper subgroup, so again by part (a) $\Pr[h' \notin C(h)] \geq \frac{1}{2}$. Putting these together we see that $\Pr[h \circ h' \neq h' \circ h] = \Pr[h' \notin C(h) | h \notin Z] \Pr[h \notin Z] \geq \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$. *5pts*

Common Issues: Some people simply wrote $\Pr[h \notin Z] \Pr[h' \notin C(h)] \geq 1/4$. This is not valid, since if $h \in Z$ then $\Pr[h' \notin C(h)] = 0$, because in that case $C(h)$ is not a proper subgroup. One must instead consider the conditional probability $\Pr[h' \notin C(h) | h \notin Z]$.

2. Perfect matchings in non-bipartite graphs

(a) Since $\det(A_G) = \sum_\sigma \operatorname{sgn}(\sigma) \prod_{i=1}^n a_{i\sigma(i)}$, we see that the non-zero terms in the determinant correspond precisely to *directed cycle covers* in G, i.e., collections of cycles (using oriented edges of G) that cover each vertex of G exactly once. The directed cycle corresponding to a particular permutation σ is just the "obvious picture" of the cycles of σ (i.e., the first cycle is $1 \to \sigma(1) \to \sigma(\sigma(1)) \to \ldots \to 1$, and so on). *6pts*

Suppose first that G contains a perfect matching $\{(u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)\}$ (where $m = \frac{n}{2}$). Then the cycle cover consisting of the m trivial cycles $u_i \to v_i \to u_i$, each of length two, contributes a non-zero term to the determinant. Moreover, this term cannot be canceled by any other term since any other cycle cover must include a variable that does not appear in this term. Hence $\det(A_G) \not\equiv 0$.

For the converse, suppose that $\det(A_G) \not\equiv 0$. We claim that the only non-zero contributions to $\det(A_G)$ must come from cycle covers all of whose cycles are of *even* length. Note that this will imply that G contains a perfect matching, since it means that such an even cycle cover must exist, and if we choose from each even-length cycle one of the two sets of alternate edges we get a perfect matching.

To prove the above claim, let σ be any permutation containing an odd cycle. Also, let σ' be the same permutation except that the direction of one of the odd cycles (for definiteness, say the one containing the lowest-numbered vertex) is reversed. Note that $\operatorname{sgn}(\sigma) = \operatorname{sgn}(\sigma')$. However, the product of the entries a_{ij} around the cycle has *opposite* signs in σ and σ'; this is because $a_{ij} = -a_{ji}$ by definition of the Tutte matrix, so there will be an odd number of sign flips around the cycle. Hence the contributions of all permutations with odd cycles cancel in pairs, and we are done.

Common Issues: Some people claimed that all cycle covers except for those that consist of a single perfect matching (i.e., cycles of length 2) cancel out. This is false since reversing the edge directions in any *even cycle will not flip the sign of the corresponding monomial. Thus, all even cycle covers survive in the determinant. Some people only proved one side of the statement and did not prove the reverse implication: G has a perfect matching $\Rightarrow \det(A_G) \not\equiv 0$.*

(b) Essentially the same algorithm works, where now the random matrix B is constructed from the modified Tutte matrix A_G as defined in part (a). Clearly we can still apply the isolation lemma to deduce that the minimum weight perfect matching (if one exists) is unique with high probability. Now note from our discussion in part (a) that this matching (via its associated cycle cover) contributes $\pm 2^{2w}$ to the determinant, where w is the weight of the matching. Moreover, any other non-zero contribution to the determinant is a multiple of 2^{2w+1} because the associated cycle cover consists of a pair of perfect matchings at least one of which has weight greater than w. Hence we can again identify $2w$ as the largest power of two that divides $\det(B)$, and we can check if edge $\{i, j\}$ belongs to the minimum weight matching by examining $\det(\hat{B}_{ij})\frac{2^{2w_{ij}}}{2^{2w}}$ as before, where now $\hat{B}_{ij}$ is obtained by removing *both* rows i, j and columns i, j from B. *4pts*

Common Issues: Many people did not explain how the algorithm needs to be modified or what needs to be checked. They just said that the same algorithm works and asserted without justification that all the relevant claims still hold.

3. Identity testing for circuits

(a) We prove slightly more generally that the polynomial computed at any node of C has degree at most 2^{s-1}. This claim is trivial in the base case, $s = 1$, since the circuit then computes either a constant or a single variable. For $s > 1$, let v be the output node of C; the claim is trivial if v is a source, so let v_1, v_2 be the two nodes with edges into v. Consider the circuit C' of size $s - 1$ obtained by removing v from C. By induction, the degrees of the polynomials computed at v_1, v_2 in C' are each at most 2^{s-2}, so the degree of the polynomial at v in C is at most $2^{s-2} + 2^{s-2} = 2^{s-1}$. *5pts*

A simple example showing tightness is the circuit with s nodes in which the single source node is labeled with variable x, the nodes are all labeled $\times$ and connected in a linear sequence, and each node has two edges to the next node in the sequence. The final node in the sequence is the output node. This circuit computes the polynomial $x^{2^{s-1}}$. (Note that this circuit implements "repeated squaring.")

Common Issues: Some people attempted to perform the induction by starting with a circuit of size s, adding a node to get a circuit of size $s + 1$, then heuristically arguing about the worst-case way in which this node could be added. This strategy rarely produced a rigorous proof. Some people did not state the induction hypothesis clearly: while their proof used the stronger inductive claim that each node has degree at most 2^{s-1}, the hypothesis only stated this for the final (output) node.

(b) The obvious idea is to compare the polynomials f_C, $f_{C'}$ using the Schwartz-Zippel algorithm. However, we need to take some care because (by part (a)) the degrees of the polynomials may be exponentially large in the sizes of C, C'. By Schwartz-Zippel, our algorithm will have small error probability provided we evaluate f_C, $f_{C'}$ at randomly chosen inputs $x_i = r_i$, where the r_i are chosen independently and u.a.r. from the set $\{1, 2, \ldots, 2^s\}$, where s is the size of the larger of C, C'. Each evaluation requires at most s arithmetic operations, since we can topologically sort the circuit and then compute the output of each node in that order. However, the resulting computations may produce numbers with as many as $s2^{s-1}$ bits, so the cost of these operations is exponential in s. We can get around this problem using fingerprinting: i.e., rather than evaluating each node exactly, we instead evaluate its fingerprint modulo a random prime p, and then at the end compare the fingerprints of the two circuits. As discussed in lecture 3, when comparing $m = s2^{s-1}$-bit numbers, fingerprints of only $O(\log m) = O(s)$ bits suffice to guarantee small error probability. Hence all arithmetic can be performed in time polynomial in s, as required. *5pts*

Common Issues: Some people either did not use fingerprinting at all, or attempted to evaluate the circuits modulo a fixed (i.e., not randomly chosen) prime p. The former error allows the numbers computed by the circuit to get doubly exponentially large, thus requiring exponentially many bits. The latter does not work because two polynomials $f(x)$, $g(x)$ may be equal mod p even when they are not actually equal (e.g., if $f(x) = 0$ and $g(x) = px$). Doing the computations modulo a random *prime p fixes this problem because the fingerprinting analysis shows that two unequal polynomials are unlikely to be equal modulo a random p.*

4. A certificate for primality

(a) Note first that condition (i) implies that $\gcd(a, n) = 1$. (This follows from the fact that it is equivalent to the integer equation $da - cn = 1$ for positive integers $d = a^{n-2}$ and c, which is impossible if the two terms on the lhs share a non-trivial factor.) Hence $a \in \mathbb{Z}_n^*$. Now condition (i) also implies that the order of a in $\mathbb{Z}_n^*$ must divide $n - 1$. Hence, if the order of a is strictly less than $n - 1$ then it must also divide at least one of the $(n - 1)/p_i$. But condition (ii) rules this out, so we conclude that the order of a is in fact $n - 1$. This implies that $\mathbb{Z}_n^*$ contains $n - 1$ elements, so n is prime. *4pts*

Common Issues: Some people made the minor mistake of not showing that $\gcd(a, n) = 1$ *so that* $a \in \mathbb{Z}_n^*$.

(b) The obvious choice of certificate is just the number a as in part (a). Note that, by part (a), any witness for n proves that n is prime; also, every prime n does have a witness, namely a generator for the cyclic group $\mathbb{Z}_n^*$. However, checking condition (ii) for a requires the prime factorization of $n - 1$, which we cannot compute in polynomial time. Hence we have to include in our certificate the factorization $n-1 = \prod_i p_i^{\alpha_i}$, which does allow us to check both conditions (i) and (ii) in polynomial time. However, we now also have to check the prime factorization; we can easily verify the factorization itself just by multiplying the factors, but we need to check also that the p_i are primes! This can be done by a recursive application of the same idea: i.e., we include in our certificate a separate certificate for primality of each of the p_i's. The base case is $n = 2$ (the only leaf in the recursion tree), which we automatically recognize as prime. *4pts*

To bound the size of this certificate, let $f(n)$ denote the number of *numbers* appearing in the certificate for n. Then $f(n)$ satisfies the recurrence

$$f(n) = \sum_{i=1}^{k} f(p_i) + 1,$$

where the $+1$ counts the actual witness a. The base case is $f(2) = 1$. An easy computation (using the facts that $\sum_i \log(p_i) = \log(n - 1)$ and that $k \geq 2$ except when $n = 3$) confirms that $f(n) \leq 2\log_2 n - 1$ solves this recurrence. Since each number appearing in the witness has $\log n$ bits, the size of the witness is $O(\log^2 n)$. Verifying the witness involves $O(\log n)$ multiplications at each node of the recursion tree, for a total of $O(\log^2 n)$ multiplications of $\log n$-bit numbers, which takes time polynomial in $\log n$.

Common Issues: Some people forgot to analyze the size of the certificate and/or the time required to check it. Some people chose a random a *in the certificate instead of one which satisfied the condition in the problem. Clearly a random choice of* a *will not satisfy the conditions and hence will not constitute a certificate.*

(c) There is no guarantee that the certificates are dense, so the probability that a randomly chosen a is a certificate may be very small. *2pts*

Common Issues: Many people failed to identify this simple requirement of certificates.

5. Generating random factored integers

(a) The only way this experiment can generate r is if coins $c_n, c_{n-1}, \ldots, c_{r+1}$ come up Tails and c_r comes up Heads. Since the coin tosses are independent, the probability of this is *2pts*

$$\frac{1}{r} \times \prod_{i=r+1}^{n} \left(1 - \frac{1}{i}\right) = \frac{n-1}{n} \times \frac{n-2}{n-1} \times \cdots \times \frac{r}{r+1} \times \frac{1}{r} = \frac{1}{n}.$$

(b) Imagine picking each s_i using the method of part (a). Then we can view the entire process as tossing a sequence of coins as in part (a), with the twist that each time a coin comes up Heads we re-toss that same coin. The probability of generating a particular sequence (s_i) is then clearly given by the expression in the question. *2pts*

(c) From the form of the distribution in part (b), and its interpretation as a sequence of coin tosses, we see that the probability that any given integer j occurs exactly k times in the sequence (s_i) is $(1 - 1/j)(1/j)^k$, and that these events are mutually independent over j. Now consider any integer r that is generated in step 2 of the algorithm. Denote by P the set of all primes $p \le n$, and by Q the subset of primes that appear in the prime factorization of r, so that $r = \prod_{q \in Q} q^{m_q}$. The probability that r is generated in step 2 is equal to the probability that each prime $q \in Q$ occurs exactly m_q times in the sequence, and no prime in $P \setminus Q$ occurs; by independence, this is just *2pts*

$$\prod_{q \in Q} \Big(\frac{1}{q}\Big)^{m_q} \prod_{p \in P} \Big(1 - \frac{1}{p}\Big) = \frac{1}{r}\alpha_n.$$

The probability that r is actually output in step 3 is therefore $(\alpha_n/r) \times (r/n) = \alpha_n/n$, as required. [NOTE: The role of step 3 is just to equalize the probabilities over r.]

Common Issues: Some people claimed that $m_j = 0$ for all composite j whenever the algorithm outputs some r. This is false because the algorithm simply ignores composites occurring in the sequence (s_i), and so may output an r no matter what the value of m_j is for composites j.

(d) By part (c), the probability that each r in the range $1 \le r \le n$ is output is α_n/n. Hence the probability that some r is output (the success probability on a single trial) is α_n. The expected number of trials until the first success is therefore $\alpha_n^{-1} \sim 1.8 \ln n$. *2pts*

(e) For each j, $1 \le j \le n$, the probability that a primality test on j is performed is equal to the probability that $m_j \ge 1$. By the discussion in part (c), this is just $1/j$. Hence the expected number of primality tests performed is *2pts*

$$\sum_{j=1}^{n} \frac{1}{j} = H_n,$$

as required.

(f) The total number of primality tests performed is $X_1 + X_2 + \ldots + X_T$, where X_i is the number of tests performed in the ith trial and T is the total number of trials needed until success is achieved. Since T is a stopping time for the X_i (in the sense that the event $T = t$ depends only on the outcomes of the first t trials), and the X_i are iid, we have by Wald's equation *2pts*

$$\mathrm{E}(X_1 + X_2 + \ldots + X_T) = \mathrm{E}(X_1)\mathrm{E}(T) = O((\log n)^2),$$

using parts (d) and (e).

An equally valid alternative argument uses conditional expectations in a careful way (which actually amounts to a proof of Wald's equation in this case). First note that we can write $\sum_{i=1}^{T} X_i$ as $\sum_{i=1}^{\infty} X_i I_{\{T \ge i\}}$, where $I_{\{T \ge i\}}$ is the indicator r.v. of the event that $T \ge i$. Taking expectations:

$$\mathrm{E}\Big(\sum_{i=1}^{\infty} X_i I_{\{T \ge i\}}\Big) = \sum_{i=1}^{\infty} \mathrm{E}(X_i I_{\{T \ge i\}}) = \sum_{i=1}^{\infty} \mathrm{E}(X_i)\Pr[T \ge i] = \mathrm{E}(X_1)\sum_{i=1}^{\infty} \Pr[T \ge i] = \mathrm{E}(X_1)\mathrm{E}(T).$$

The key step here is the second equality, which follows from the fact that X_i is independent of the event $T \ge i$ (since this event depends only on the outcomes of the first $i - 1$ trials).

Common Issues: Some people gave no argument or an incorrect argument for why it is valid to multiply $\mathrm{E}(X_1)$ by $\mathrm{E}(T)$ to obtain the expected total number of primality tests.