CS174 **Midterm 2 Solutions** **Spring 99**

1. (20 points) Give brief answers to the following:

   (a) (4 points) In the byzantine agreement algorithm, do all good processors set their output bits permanently on the same round?

   No, if bad processors send different outputs to different good processors, some processors will have $tally > G$ while others will have $tally \leq G$. The processors with $tally > G$ will halt in that round, while the others will run for one more round.

   (b) (4 points) The data-punctuated token trees from class don't work on a string of the same character repeated many times. How would you fix that?

   Method 1: use run-length coding. Replace the subsequence $aaaa\cdots$ with $\#Na$ where N is an integer, and $\#$ is a special character. Method 2: use redundant encoding, where there are more than one representation for each char. e.g. in ASCII, the 8th bit isnt used, so we can set it to one or zero. Then replace $aaaa\cdots$ with $a_0a_1a_1\cdots$ where a_0 has the 8th bit set to 0 and a_1 has the 8th bit set to 1. The sequence of a_0s and a_1s should be pseudo-random and repeatable.

   (c) (4 points) In the fingerprint algorithms from class, the sequence of characters is converted to a polynomial, evaluated, and then reduced modulo a prime. Why is a prime used rather than a non-prime modulus?

   The integers mod p form a field. That guarantees that a polynomial $P(x)$ of degree d has $\leq d$ roots. If p is not prime, there may be more than d roots for $P(x)$.

   (d) (4 points) What is the minimum number of edges in a spanning forest on n vertices?

   A spanning forest must touch every vertex in the graph. Each edge in the forest can touch two vertices, so the minimum number of edges is $n/2$.

   (e) (4 points) In choice coordination with two processors, do all processors halt in the same round?

   No, One processor that writes a $\sqrt{}$ halts immediately, while the other runs for one more round.

2. (20 points) Suppose we run the randomized routing algorithm from class on a hypercube with $N = 2^n$ processors. We use bit-fixing, so that at the i^{th} step, the i^{th} bit of a packet's address $a_1a_2\cdots a_n$ is set to the destination address.

   (a) (4 points) What is the expected number of collisions during the second time step? (note: there are no collisions during the first time step, but more than one packet may arrive at the same processor in the first time step)

   The answer is $N/8$. At the start of the first time step, every packet has a unique address. In order to have a collision, two packets must arrive at the same vertex in the first time step *and* try to leave via the same edge in the second time step.

The only packets that can arrive at the same processor during step 1 are pairs v_i, v_j whose starting addresses i and j differ only in the first bit. If the first bit of their final destinations agree, they will move to the same processor. The probability that this happens is $1/2$. If this pair of packets also has the same second bit of destination address, then a collision occurs. The probability that this happens given that they are at the same processor is $1/2$. The expected number of collisions in the second time step is $(N/2) \times (1/2) \times (1/2) = N/8$, which is the number of pairs which could collide $(N/2)$ times the probability $(1/2)$ that a pair moves to the same processor at step 1, times the probability that the pair tries to leave via the same edge $(1/2)$ at step 2.

(b) (4 points) What is the probability that two randomly chosen routes share at least one edge? Give an upper bound.

This is the expected value of the random variable H_{ij} defined in lecture. By that analysis, the expected value of the sum of H_{ij} over all j, which is NH_{ij}, was $\leq n/2$. Therefore $E[H_{ij}] = Pr[H_{ij} = 1] \leq n/(2N)$ which is the answer we want.

(c) (8 points) Let X_{ij} be a random variable that counts the number of edges shared by routes of packets v_i and v_j *given* that these two routes share an edge, so $X_{ij} > 0$. What is the distribution of X_{ij}? You can assume the destination addresses for v_i and v_j are random bit strings.

This is a geometric random variable. Given that two packets arrive at the same processor and then leave along the same edge, their subsequent routes are given by the bits of their destinations, which are assumed to be random bit strings. The probability that they agree on each additional edge is 1/2. So the distribution is given by $Pr[X_{ij} = k] = 1/2^k$ for $k \geq 1$.

(d) (4 points) What is $E[X_{ij}]$ for X_{ij} as defined in the last question?

The expected value is 3. For a normal geometric random variable, the expected value is $1/p$. Here $p = 1/2$, but the random variable X_{ij} is at least 1, so strictly speaking, it is 1 + a geometric random variable. Its expected value is $1 + 1/p = 1 + 2 = 3$.

3. (20 points) Consider the caching problem where the cache holds k items. Assume memory items are identified with positive integers. Let the following sequence of N requests occur: $(1, 2, 3, \ldots, 2k, 1, 2, 3, \ldots, 2k, \ldots)$, that is, the sequence $(1, \ldots, 2k)$ repeated until there are N requests (assume $N >> k$).

(a) (15 points) How many misses will the optimal offline algorithm MIN make on this sequence as a function of N?

It makes $\approx N/2$. For the first k requests, $1, \ldots, k$ the items simply fill the cache. Then each request from $k + 1, \ldots, 2k$ causes eviction of the page that was added one step earlier, which has its next request the furthest in the future. After this, the cache contents is $1, \ldots, k - 1, 2k$. Then the next $k - 1$ requests cause no misses. The requests $k, \ldots, 2k$ each cause a miss and successively occupy the last cache location. This process repeats for ever. During each cycle we have $k - 1$

hits and $k+1$ misses. The number of cycles is $N/(2k)$, so the total number of misses is $N(k+1)/2k \approx N/2$.

(b) (5 points) Is there a sequence of requests for $2k$ distinct memory items that would produce more misses for MIN?

This is the worst possible sequence, so there is no other sequence that produces more misses.

4. (20 points) In the algorithm for computing minimum cuts from class, we used contraction to simplify the graph. We showed that the probability that a randomly-selected edge lies in the minimum cut is $\leq 2/n$.

(a) (8 points) Suppose we pick $n/2$ edges independently and at random without contracting any of them. What is the probability that none of them is in the min-cut? Assume n is very large, and simplify your answer.

Since we are taking $n/2$ independent samples, and each has probability $(1-2/n)$ of being "good" (not in the cutset), our overall probability of success is

$$(1-2/n)^{n/2} \approx e^{-1}$$

(b) (6 points) Suppose we now contract all of those edges. Give lower and upper bounds on the number of vertices in the contracted graph.

Well, we might pick the same edge every time (or there may be only one edge in the graph), so the upper bound on the number of vertices after contraction is $n-1$. For the lower bound, we suppose that every edge reduces the number of vertices by one. Then we would have $n/2$ vertices after contraction.

(c) (6 points) What is the expected number of vertices in the contracted graph after (b) above? Hint: the MIN-CUT algorithm from lecture samples edges randomly from a contracted graph. That's the same as sampling as in part (a) above, and discarding redundant edges, i.e. edges that join vertices that have already been contracted. So you can assume that edges were sampled from the contracted graph. Then use the formula derived in class (or derive it yourself) for the probability of avoiding the min cut as a function of the number of vertices t remaining in the contracted graph.

The observation here is that there are two different but equivalent methods for sampling edges, one with repeats and redundant edges and one without. For each method, we can compute the exact probability of never picking an edge in the cutset. We can equate the two probabilities and determine how many edges were contracted in the method from class. That tells us how many vertices remain in the graph. From part (a), the probability is e^{-1}. This must equal the probability t^2/n^2 as a function of the number of vertices t after contraction for the method from lecture. So

$$t^2/n^2 = e^{-1}$$

and therefore $t = ne^{-0.5} = 0.6065n$ is the expected number of vertices after contraction.