CSCI 6220/4030: Practice Exam

Provide mathematically convincing and clearly written arguments for the following problems.

1. Company Z has a complicated internal network of n machines, and would like to constantly monitor their network traffic in order to know how efficiently messages are routed through their network.

   In particular, they want to know which machines are involved in routing UDP packets from machine i to machine j. Unlike TCP, UDP does not guarantee the delivery of all packets. Assume that each packet sent from i to j follows the same route of length ℓ, each message consists of q packets, and UDP delivers each packet with probability p. Each machine has a unique MAC address that is 48 bits long.

   One approach to this problem is to label each packet with the MAC address of each machine it passes through. What is the probability that at the end of sending m packets from i to j, machine j knows all the machines along the route? How many packets must we send before the probability of failure is less than δ?

   Another programmer suggests instead that each time a message passes through a machine, we label it with that machine's MAC address with a probability p_{label}. Thus each packet identifies some random subset of the machines it has passed through on its way to j. How many packets suffice before we can be sure that with probability at least $1 - \delta$, machine j knows all the machines along the route (use a union bound)?

   What are the expected communication overheads in bits (due to storing the addresses) of using these algorithms to achieve a failure rate of δ, if we only consider the cost of labeling the packets that are successfully delivered?

2. Suppose we are given n records, $R_1, \ldots, R_n$. The records are kept in some order. The cost of accessing the jth record in the order is j. Thus, if we had the four records ordered as R_2, R_4, R_3, R_1, then the cost of accessing R_4 would be 2 and the cost of accessing R_1 would be 4. Suppose further that, at each step, record R_j is accessed with probability p_j, with each step being independent of the other steps. If we knew the values of p_j in advance, we would keep the R_j in decreasing order with respect to p_j.

   Since we don't know the p_j in advance, we will use the 'move to front' heuristic: each time we access a record we move it to the front of the list. For example, if the order was R_2, R_4, R_3, R_1 before R_3 was accessed, then the order at the next step would be R_3, R_2, R_4, R_1. Argue that the order of the records can be thought of as the state of an ergodic Markov Chain (assume all p_j are strictly positive). Give the stationary distribution of this chain.

3. Consider that you have a huge collection of documents, say the entire New York Times archive, and you want to compute the word-word coccurence statistics on this corpus. Here we count the two words 'a' and 'b' as co-occurring whenever we see 'a b' or 'b a'.

   Let there be n unique words in the corpus, and m pairs of words that co-occur. We can assume that n is known, but we have no idea what m is a priori. Since n may be on the order of millions, due to the presence of a large number of proper nouns, the naive approach of constructing an $n \times n$ matrix of zeros and simply incrementing the relevant entries every time we see a pair of words co-occurring will not work: we simply do not have enough memory (terabytes) to store such a large matrix on one machine.

   Note, however, that m is likely to be much smaller than $\binom{n}{2}$ (e.g. "mountebank" and "paracetemol" are unlikely to co-occur), so we can expect that we may only see a few billions of word-word pairs, and think of our problem as that of efficiently constructing a sparse matrix on the scale of gigabytes. In particular we can expect it to fit in memory on a high-end desktop system.

To build this sparse matrix efficiently, we will construct a hash table T indexed by pairs of (w_1, w_2) and stream over the corpus incrementing $T[h[(w_1, w_2)]]$ each time we see words w_1 and w_2 co-occurring. Once T is constructed, we can easily convert it into a sparse matrix representation of the word-word co-occurrence matrix (if we store (w_1, w_2) in T along with their co-occurrence count, but we'll ignore this step for simplicity).

Assume n is known. The relevant questions here are: how many entries should T have, what hashing should we use to ensure that we do not have collisions, and what is the cost of constructing T?

First, we need to estimate m so that we know how much space to allocate for T. Give a streaming algorithm that runs over the corpus and calculates an over-estimate of m. This algorithm should guarantee that with probability at least $1 - \delta$, the returned estimate is guaranteed to be an overestimate of m, and is at most $9m$. How much space does this algorithm require?

Second, we need to hash in a way that avoids collisions, so that each bucket corresponds to a unique word-pair. In an ideal world, we would use ideal hash functions and open-addressing, but this is not feasible. Instead, describe a universal hash family that maps pairs of words (w_1, w_2) to one of $18m$ buckets and is computable very quickly (since we will be using it on every pair of words). Now pretend that this is an ideal random hash family that can be used for open-addressing, and using the conclusions of problem 4 on HW4, give an upper bound on the expected time that it takes to locate and update the correct bucket for a given pair of words (w_1, w_2). In the context of problem 4 on HW4, explain why we chose to use $18m$ hash buckets.

What is the best upper bound that you can give on the expected time cost of streaming over the corpus and updating the entries of T to capture the word-word co-occurrence statistics? Compare this to the time cost of streaming over the corpus and updating the entries of a matrix directly to capture the word-word co-occurrence statistics. Similarly, compare the space costs of the two algorithms.