CS/MATH 335: PROBABILITY, COMPUTING, AND GRAPH THEORY
HOMEWORK 6
DUE IN CLASS TUESDAY, NOVEMBER 1, 2016

Taylor Heilman and Matt Iammarino

Partner Problems

Roughly in order of difficulty

(1) Mitzenmacher 5.16

a.) To find the value of p where the expected number of cliques of five vertices in G is equal to 1 we first thought of how many ways could we choose 5 vertices from the total n vertices. $\binom{n}{5}$ is the number of ways we can do this. We then thought of the probability of all 10 necessary edges being present in the clique. p^{10} is the probability of all 10 edges being present. Thus we get the equation $\binom{n}{5}p^{10} = 1$. Solving for p we get:

$\Rightarrow p^{10} = \frac{1}{\binom{n}{5}}$

$\Rightarrow p = (\frac{1}{\binom{n}{5}})^{\frac{1}{10}}$

b.) Following a similar procedure as above, we first thought of the way to get a $K_{3,3}$ graph. We decided that $\binom{n}{3}\binom{n-3}{3}$ is all the possible ways. At first we thought $\binom{n}{6}$ was the correct answer, but we then thought of a $K_{3,3}$ as being two separate groups of 3 vertices, instead of one group of 6 vertices.

We then had p^9 as the probability that the 9 necessary edges are present. Also, $(1-p)^6$ represents the probability that the remaining 6 unnecessary vertices are not present. Thus we have:

$\binom{n}{3}\binom{n-3}{3}p^9(1-p)^6 = 1$

$p^9(1-p)^6 = \frac{1}{\binom{n}{3}\binom{n-3}{3}}$

Without a specific value for n we were unable to simplify anymore.

c.) Originally we thought there were $n!$ different Hamiltonian cycles in a complete graph, but then we realized we over counted because the starting point is arbitrary so we got $(n-1)!$. At this point we thought we were close but had a feeling we were over counting still, we then considered the inverse of a cycle and noticed we were double counting every cycle so we decided that there are $\frac{1}{2}(n-1)!$ distinct Hamiltonian cycles in a complete graph. p^n represents the n necessary edges in the Hamiltonian cycle, hence we get:

$\Rightarrow \frac{1}{2}(n-1)!p^n = 1$

$\Rightarrow (n-1)!p^n = 2$

$\Rightarrow p^n = \frac{2}{(n-1)!}$

$\Rightarrow p = (\frac{2}{(n-1)!})^{\frac{1}{n}}$

(2) Mitzenmacher 5.23

Errors could arise when using fixed-length counters if a bin ends up with a counter above the max number the counter can hold. For example, if 4 bits were allotted, then an error occurs if some bin has a counter that is greater than 16. In our specific problem, we have m counters with b bits, k hash functions, and t insertions. With that being said, to bound the probability of an error occurring we can let X = the value of a single bin's counter and calculate $Pr(X \geq 2^b + 1)$ using Markov's formula:

Notice the $E[$ balls in a bin $] = \frac{kt}{m}$. We got this value because there are k hash functions being used, t insertions and deletions (thought of as balls), and m counters (thought of as bins). The t balls turn into kt balls since each ball gets hashed by k hash functions, hence the number of balls divided by the number of bins is our expected value.

$Pr(X \geq 2^b + 1) \leq \frac{\frac{kt}{m}}{2^b+1}$

Notice however that this is only the probability that one counter is over 2^b, hence we use the union bound the probability of any of the m counters being greater than 2^b. So we get:

$Pr($ any $X \geq 2^b + 1) \leq (\frac{\frac{kt}{m}}{2^b+1})m$

$Pr($ any $X \geq 2^b + 1) \leq \frac{kt}{2^b+1}$

(3) Mitzenmacher 5.24

The method we came up with is the following: Split the array in half to get two equal length arrays, each of length 2^{b-1}. Then, for each index of the arrays take the union of array 1 with array 2. For example, if array 1 has a 0 at the first index and array 2 also has a 0 at the first index the resulting union will be 0. After this step we will have our final array of length 2^{b-1}

which will only have a 0 at some index if array 1 and array 2 both had 0's at that index, otherwise if array 1, array 2, or both arrays have a 1 at some index, the final array will have a 1 at said index.

Analyzing this solution we calculated the Probability of having a 0 or 1 in the final array. When throwing one ball the $Pr($ bit is 1$) = \frac{2}{m}$ because in the m bins there are two different indices such that if at least one of the indices = 1, the final array will have a 1 as the bit. The $Pr($ bit is 0 $) = 1 - \frac{2}{m}$ because if a bit is not 1 it must be equal to 0. Taking these probabilities we then calculated the probability that a bit is 0 after the words have been hashed. We defined n to be the number of words being hashed and k hash functions which are being used in the bloom filter. Hence, we get :

$Pr($ bit is 0 $) = (1 - \frac{2}{m})^{kn}$

To find the probability of a false positive we do $(1 - (1 - \frac{2}{m})^{kn})^k$. We take 1 minus the probability that the bit is 0, which represents the probability that a bit is 1, then we put it to the k because each item that is hashed by the k hash functions must hash to a 1.

In the original bloom filter problem, the optimal result is achieved when each bit of the bloom filter is 0 with probability $\frac{1}{2}$. From this we can say that our idea is not optimal because our bloom filter has a higher probability of a bit being 1 than a bit being 0. Notice, the only way a bit is 0 is if both indices which are unioned are 0. On the other hand, there are three different cases which results in a bit being 1. The three cases of a bit being 1 can be visualized as follows: 10 = 1, 11 = 1, 01 = 1 The left side of the equation represent the two indices being unioned. So since there is a higher probability that a bit equals 1, this is not optimal.

(4) Mitzenmacher 5.25

From the problem it's given that we have n users, each of which get inputted into the hash function which outputs a b-bits hash value. To find the lower bound we followed section 5.5.4 from the book. The probability that two hash values collide is $1 - (1 - \frac{1}{2^b})^{n-1} \leq \frac{n-1}{2^b}$. Notice that $\frac{1}{2^b}$ is the probability of hashing to one specific value, hence $1 - \frac{1}{2^b}$ is the probability of not hashing to one specific value. Then raising this value to the $n - 1$ gives us $(1 - \frac{1}{2^b})^{n-1}$ which represents the probability that any of the other users do not hash to a specific value, hence doing $1 - (1 - \frac{1}{2^b})^{n-1}$ is the probability that any of the other uses do hash to a specific value. Using this information we said $p \geq 1 - (\frac{n-1}{2^b})$ as the probability that all other users do not obtain the same hash value as the leader.

To find the upper bound we used section 5.5.2. From the book we have that the probability that 2 users hash to the same value is $1 - \frac{1}{2^b}$, which follows what we said above. If we have m users and hash values outputting b-bit

hash values, the probability that a collision occurs is $1-(1-\frac{1}{2^b})^m \geq 1-e^{\frac{-m}{2^b}}$. Using this information, we can substitute $n-1$ for m since we only care that the other $n-1$ users do not obtain the same hash value as the leader, getting:

$p \leq 1 - e^{\frac{-n-1}{2^b}}$

Thus we get, $1 - (\frac{n-1}{2^b}) \leq p \leq 1 - e^{\frac{-n-1}{2^b}}$

Solving for b we get:

$\Rightarrow p \geq 1 - \frac{n-1}{2^b}$

$\Rightarrow \frac{n-1}{2^b} \geq (1-p)$

$\Rightarrow \frac{n-1}{1-p} \geq 2^b$

$\Rightarrow \log_2(\frac{n-1}{1-p}) \geq b$

Similarly we get:

$\Rightarrow p \leq 1 - e^{\frac{-n-1}{2^b}}$

$\Rightarrow e^{\frac{-n-1}{2^b}} \leq 1-p$

$\Rightarrow \frac{-n-1}{2^b} \leq \ln(1-p)$

$\Rightarrow -n-1 \leq \ln(1-p)2^b$

$\Rightarrow \frac{-n-1}{\ln(1-p)} \leq 2^b$

$\Rightarrow log(\frac{-n-1}{\ln(1-p)}) \leq b$

Thus by choosing a b such that $log(\frac{-n-1}{\ln(1-p)}) \leq b \leq \log_2(\frac{n-1}{1-p})$, the leader will be successfully chosen with probability p.

(5) Problem 5

Emailed Python program to you.

(6) Bonus problem

Emailed Python program to you.