CS 5150/6150: Assignment 5
Due: Nov 6, 2011

This assignment has 5 questions, for a total of 100 points and 0 bonus points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question 1: Biased and unbiased coins . . . . . **[20]**

Solve Question 6 from `http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/09-nutsbolts.pdf`. The breakdown of points per subpart is $2 + 6 + 6 + 6$

Question 2: Randomized Minimum . . . . . **[20]**

Solve Question 8 from `http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/09-nutsbolts.pdf`. The breakdown of points per subpart is $4 + 8 + 8$. Note that in part (b), the sentence should instead read ``...k^{th} iteration of the for loop''.

Question 3: Contention Resolution . . . . . **[20]**

This question is Q3 from Chapter 13 of Algorithm Design.

Suppose we have a graph $G = (V, E)$ where each vertex represents a process and each edge represents a potential conflict between processes. We want to run as many processes in parallel as possible, but we cannot run two processes that conflict. Finding a largest ``non-conflicting'' set of processes is the MAXIMUM INDEPENDENT SET problem, which we WON'T try to solve here.

What we want to produce is a heuristic that generates a reasonably large independent set of processes S that can run in parallel, and we want to make this ``distributed'' so that each node can decide for itself whether it participates in the desired non-conflicting set S.

(a) [**10**] Here's the first protocol:

> Each process i independently picks a random value x_i that is set to 1 with probability 1/2 and 0 with probability 1/2. If $x_i = 1$ and each of i's neighbors choose the value 0, then i enters S

Show that the set S resulting from each node running this protocol is conflict free, and give a formula for the expected size of S in terms of n (number of processes/vertices) and d (number of conflicts per process/degree of vertex). You may assume that all vertices have exactly the same degree d.

(b) [**10**] But why choose a probability of 1/2 ? Consider the following alternate protocol:

> Each process i independently picks a random value x_i that is set to 1 with probability p and 0 with probability $1 - p$. If $x_i = 1$ and each of i's neighbors choose the value 0, then i enters S

Determine the value of p that maximizes the size of S (again expressed in terms of n, d). Also provide the value of S obtained by using this value of p.

Question 4: Discrepancy . . . . . **[10]**

When we toss a fair coin, we expect that we get roughly half-and-half Hs and Ts. Of course, this might not happen in general: the question is, how bad can the difference get ?

Consider a sequence of $2n$ coin tosses, and let X_H be the number of heads and X_T be the number of Ts in the resulting sequence. Obviously, $EX_H = EX_T = n$, and therefore $E(X_H - X_T) = 0$. Show that for any $\epsilon > 0$, there exists a constant c such that

$$Pr[X_H - X_T > c\sqrt{n}] \leqslant \epsilon$$

Note the quantification: in general, c will be a function of ϵ.

HINT: Use a Chebyshev bound.

Question 5: Hashing: An experimental study . . . . . **[30]**

We've seen many different kinds of hashing techniques:

Chaining Hash items into a table and use an overflow linked list to manage collisions

Open addressing Fix the size of the hash table, and if an item is hashed to a full cell, find a new cell either by linear scanning or repeated re-hashing

Cuckoo Hashing Use two hash tables, each with a different hash function, and rehash items if collisions occur.

The first method allows tables to grow arbitrarily, but with an increase in access time. The second method keeps a fixed table size, but insert and lookups might take longer and longer. The third method guarantees constant access time, but rehashings might take longer, and eventually lead to a complete rebuild.

In this question, you will experiment with the different hashing strategies, attempting to determine how they behave for different load profiles.

(a) [**10**] We will assume that you're hashing 32-bit integers ($w = 32$) into a table of size $m = 2^M$. The hash function you will use picks a random odd $a < 2^w$ and then computes

$$h_a(x) = \left\lfloor \frac{(a \cdot x) \mod 2^w}{2^{w-M}} \right\rfloor$$

which can be written in shift notation as

$$h_a(x) = (a * x) \gg (w - M)$$

Fix values of n and M. Choose an odd random number $a < 2^w$. Generate n random 32-bit integers and hash them into a table of size $m = 2^M$ using chaining for overflow. Let ℓ_i be the number of items that hash to bucket i. Compute the average $\mu = \sum_{0 \leqslant i < m} \ell_i / m$, maximum value $\mathsf{max} = \mathsf{max}_{0 \leqslant i < m} \ell_i$ and variance $\sigma^2 = \sum_{0 \leqslant i < m} (\ell_i - \mu)^2 / m$ of the ℓ_i. Repeat this two more times, for different values of a. Now you have three values for each of μ, max and σ^2. Take the middle (median) value for each group of three. This is your result $\mu_*, \mathsf{max}_*, \sigma^2_*$ for a single choice of n and M.

Now for values of M in the set $\{10, 12, 14\}$ and values of n of the form $\alpha 2^M$, where $\alpha = 1, 10, 100, 1000$, plot the results $\mu_*, \mathsf{max}_*, \sigma^2_*$ as a function of n.

What do you observe ? Present both the plots and your analysis. You should have one plot for each value of M, on which the x-axis is labelled with n, and the y-axis plots the three parameters as three different series.

(b) [**10**] We now turn to open addressing schemes. Here, the key parameter is the *load factor* defined as $s = n/m$. Define $h_d(x, i)$ to be

$$h_d(x, i) = h_a(x) + i h_b(x)$$

where $h_a(x)$ is chosen as above, and $h_b(x)$ is constructed by choosing a *second* random value $b < 2^w$ and generating the hash function as we generated h_a.

For a fixed value of n and M, choose random numbers a and b as before, and generate n random 32-bit integers, hashing them into a table of size $m = 2^M$ using double hashing (i.e if $h_d(x, 0)$ is nonempty, you try $h_d(x, 1), h_d(x, 2)$ and so on till you find an empty spot). When inserting x_j, record the number of trials t_j needed to find an empty spot for x_j. If you're unable to find one, set $t_j = m$ and increment a `fail` counter. As above, compute the average, max and variance of the values t_j (note that you have n values of t_j, so your averaging and variance computations must use n instead of m), and also store the `fail` counter. As before, repeat this for three different random choices of the pair (a, b) and take the median values of the four parameters.

Now let M take the values $10, 12, 14$, and let n take the values $s2^M$, where $s = \{0.1, 0.2, 0.5, 0.7, 0.9\}$. Plot the four parameters as a function of s (not n) for each choice of M (one chart for each value of M). Present the plots and your analysis. What do you conclude ?

(c) [**10**] Repeat the above experimental scenario, where now you use cuckoo hashing with the two hash functions $h_a(x)$ and $h_b(x)$. In other words, maintain *two* tables, each of size $m/2 = 2^{M-1}$, and choose a, b to be random odd numbers that are at most 2^{M-1}. Again, let t_j be the number of probes needed to insert element j. Plot charts as above. Present your observations.

What do you conclude about the two open addressing schemes, as well as how they compare to chaining ?

Note: Unlike in previous assignments, this question is platform and language independent. You will not be expected to submit your source code. All you will be expected to submit are the plots described in the question.

Note that the standard random generators provided with most languages don't always yield high quality randomness. You might consider using the Mersenne Twister (which has ready-to-use code in different languages at `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/eversions.html`: I'll leave the final decision to you.