There are two groups of problems: recommended and required. Please turn in just the required problems.

**Required problems:**

1. Suppose that \( r = \frac{p}{q} \), where \( p \) and \( q \) are integers, and that \( 0 < r < \frac{1}{k} \) for some integer \( k \geq 2 \).

Suppose we are maintaining a set of keys in a hash table, using linear probing. Keys may be added but are never deleted. Initially, the table size is \( q \) and there are \( p \) keys in the table (so the load factor is initially \( r \)). As we add each key:

- If the load factor will remain less than \( k \cdot r \) after we add the key, we simply add it.
- If adding the key would make the load factor be \( k \cdot r \) or more, we allocate a new table \( k \) times as large as the original table (so that after the new key is added, the load factor is \( r \)).

Show that the amortized cost per add operation is \( O(1) \). Assume:

- the cost of adding an item, not counting the cost of re-allocating the table, is \( O(1) \).
- the cost of re-allocating the table is proportional to size of the new table.

Suggestion: Choose your potential function carefully!!

2. In class we described a way of preventing infinite loops in the set algorithm of cuckoo hashing, by counting steps and stopping when the count gets too large. This method has a problem: you need to know the constant factor to use in the \( \Theta(\log n) \) bound on the number of steps. Describe an alternative method of preventing infinite loops, without counting, by detecting a loop whenever it happens. Ideally, your solution should

- Use a constant amount of additional memory
- Only perform a constant amount of additional work per step of the set algorithm
- Not use a step counter
- Detect any loop within a number of steps proportional to the number of keys involved in the loop
- Only report that there is a loop when the set algorithm really has an infinite loop

(Hint: consider what happens to the key given as the original argument to the set algorithm during a loop.)
3. Suppose you have two Bloom filters $F_A$ and $F_B$, each with the same number of cells and the same hash functions, representing the two sets $A$ and $B$. Let $F_C = F_A \& F_B$ be the Bloom filter formed by computing the bitwise Boolean and of $F_A$ and $F_B$.

   (a) $F_C$ may not always be the same as the Bloom filter that would be constructed by adding the elements of the set $A \cap B$ one at a time. Explain why not.

   (b) Does $F_C$ correctly represent the set $A \cap B$, in the sense that it always gives a positive answer for membership queries of all elements in this set? Explain why or why not.

4. Prove that tabulation hashing is not 4-independent by finding a set of four keys for which all 4-tuples of indexes are not equally likely.

Recommended problems: These are not to be turned in.

5. (Note: the purpose of this problem is to make sure that you understand the mechanics of the various hash-based structures listed below. If you are convinced you understand how they work you can skip this problem, but be aware that you might get some version of this problem on the midterm and/or the final exam.)

   Choose a small prime number $p$. Generate some random integers $k$ in the range $[0, p^3]$. For each of the following hash-based structures, insert the numbers you generated. Use the hash functions $h_1(x) = x \mod p$, $h_2(x) = x/p \mod p$, $h_3(x) = x/p^2 \mod p$. (If you only need one hash function use $h_1$, if you only need two hash functions use $h_1$ and $h_2$.)

   (a) Hash table using chaining
   (b) Hash table using linear probing
   (c) Hash table using quadratic probing
   (d) Hash table using double hashing
   (e) Cuckoo hashing structure
   (f) Bloom filter
   (g) Cuckoo filter

6. In a cuckoo hashing data structure with two hash functions $g(x)$ and $h(x)$, the hash table may still operate successfully if two elements $x$ and $y$ are inserted with $g(x) = g(y)$ and $h(x) = h(y)$. However, it will fail if it is given three elements $x$, $y$, and $z$ with $g(x) = g(y) = g(z)$ and $h(x) = h(y) = h(z)$.

   (a) If a cuckoo hashing data structure has $N$ slots in each table and the hash functions are uniformly random, what is the probability that some particular triple $(x, y, z)$ collides in this way?

   (b) If $n$ elements are inserted into the structure, what is the expected number of triples that collide in this way?

   (c) Use Markov’s inequality (http://en.wikipedia.org/wiki/Markov%27s_inequality) together with your answer to part (b) to get a bound on the probability that there is at least one triple collision.

It is fine to solve the following two problems either mathematically (i.e., by finding a closed-form solution) or computationally (i.e., by writing a computer program that finds the solution.)

7. Suppose that we want to store a set $S$ of $n = 25$ elements, drawn from a universe of $U = 10000$ possible keys, in a Bloom filter of exactly $N = 120$ cells. Suppose that we care only about the accuracy of the Bloom filter and not its speed. For this problem size, what is the best choice of the number of cells per key (the parameter $k$ in the lecture)? (In other words, what value of $k$ gives the smallest possible probability that a key not in $S$ is a false positive?) What is the probability of a false positive for this choice of $k$?
8. Note: This problem extends the previous problem.

Suppose that we want to store a large set in a Bloom filter. As in the earlier problem, suppose we care only about the accuracy of the Bloom filter and not its speed. Suppose that we want the probability of a false positive to be no larger than 1%. What is the largest ratio \( n/N \) that we can have and still achieve this (by choosing \( k \) appropriately)? If the ratio \( n/N \) is near this maximum value, what is the appropriate choice for \( k \)?