HASHING STUDY QUESTIONS
Fundamental Algorithms, Fall 2004, Professor Yap

Due: Do not submit.

INSTRUCTIONS:

- This Study Question is in lieu of Homework 6, and focuses on hashing (Lecture XI). This should be your priority in studying Lecture XI.

---

1. Coalesced Hashing
   As usual, we have a hash table $T[0..m-1]$ with m slots. We assume a hash function $h : U \to \mathbb{Z}_m = \{0, \ldots, m-1\}$. We want to implement coalesced hashing, but a little differently than described in the Lecture Notes. The approach described here is more[1] straight forward because we will *explicitly* introduce a variable to remember the "state" of each slot. As usual, we assume that the keys stored in the hash table are unique. The i-th slot $T[i]$ has four fields:

   (a) $T[i]$.`Key` which stores a key (element of U).

   (b) $T[i]$.`next` which stores an element of $\mathbb{Z}_m$.

   (c) $T[i].State$ stores a value in $\{-2, -1, 0, 1, 2\}$ where $State = 0$ indicates the ORIGINAL state, $|State| = 1$ indicates OCCUPIED, $|State| = 2$ indicates DELETED. Initially, $State = 0$ but once a slot has been used, it never revert to this state again. Moreover, if $State > 0$, indicates this slot as EOC (=END_OF_CHAIN); $State < 0$ means it is not EOC.

   (d) $T[i]$.`Data` which stores associated data. This is important in practice, but as usual, we ignore this field in our algorithms.

   (i) Describe scenarios where we need to use all the 5 possible state values in our data structure.
   (ii) Consider the operation FINDKEY(k) which returns $i \in \mathbb{Z}_m$ where $T[i]$ is a slot in the chain that begins at slot $T[h(k)]$. Moreover, one of three properties hold: (a) $|T[i].State| = 1$ and $T[i].Key = k$. (b) k is not in the current hash table, and $|T[i].State| \neq 1$. (c) k is not in the current hash table, and $T[i].State = 1$ (thus $T[i]$ is OCCUPIED and is EOC). Implement this algorithm.
   (iii) Consider the operation INSERT(k) which inserts k into the table if the table is not already full and does not contain k. Implement INSERT(k) with the help of FINDKEY(k). Assume a global integer variable N which remembers the number of keys currently in the hash table. If $N = m$, INSERT(k) returns an ERROR condition. Otherwise, it returns the $i \in \mathbb{Z}_m$ where k is stored. It is important to ensure that you do not create cycles during INSERTION.
   (iv) Implement DELETION(k), with the obvious meaning: if k is in the table, it will be deleted.
   (v) We want to prove a fundamental property of your solution in parts (iii) and (iv). SHOW that a sequence of INSERTION and DELETION operations does not introduce a cycle in our linked lists. Assume that we start from an empty hash table where $T.State[i] = 0$ for all i.

2. Universal Hashing
   The key result about how to use Universal Hash Functions is represented by Theorem 5 (p. 12, Lecture XI). Through this exercise, we want you to be familiar with a particular class of universal hash sets, namely the ones described in Lecture XI in §5 (p. 15-16). By a "finite field" F, you may assume that we mean[2] a set of the form $F = \mathbb{Z}_q = \{0, \ldots, q-1\}$ where q is a prime number. The four arithmetic operations in F are just the usual ones, but always modulo q. The most important thing you need to know about F is that the operation of *inverse* is defined. That is, for each $x \in F$, if $x \neq 0$ then there is a unique element $y \in F$ such that $xy = 1$. We call y the *inverse* of x and denote it by x^{-1}. The Example and Solution on page 16 should be mastered.
   (i) What is the simplest example of f finite field?

[1] In the lecture notes, we used special values of `next` to encode this state information. Actually, we will also need a special value of `Key` (say `Key` = 0) to help us in this encoding – hence the write up in the Lecture Notes is buggy.

[2] There are other finite fields besides these, namely those with $|F|$ a power of prime. But the arithmetic here is more complicated, and you need not know about them.

(ii) In the finite field $\mathbb{Z}_{13}$, find the inverses of $x = 1, 2, 3, 4, 5, 6$. REMARK: there is an algorithm based on Euclid's algorithm for computing inverses. But you just need to find inverses by brute force search.
(iii) As a compiler designer, you want to construct a hash table to store all the user-defined variables that might be encountered in a compiled program. Assume each variable name (i.e., *key* for hashing) comes from the set $U = \Sigma^{30}$ where $\Sigma = \{\sqcup, a, b, \ldots, z, 0, 1, \ldots, 9\}$. So $|\Sigma| = 37$ and each key has length exactly 30. (NOTE: if a key has length less than 30, we assume you pad it with $\sqcup$ until it is 30.) You want to create a hash table $T[0..m-1]$ where $1000 < m < 2000$, and resolve collision using separate chaining. Show how to choose a hash function so that the expected number of keys that collide with any given key is at most 1.

HINT: First, choose a universal hash set $H \subseteq [U \to \mathbb{Z}_m]$ for an appropriate m. Remember that there are lots of primes[3] but for our purposes perhaps it is enough to know that smallest prime larger than $37^2 = 1369$ is $q = 1373$.
(iv) For a program with at most 1000 variable names, what is an upper bound on the expected length of any chain in your hash table in part (ii)?

[3] You can easily look up some standard mathematical table for primes up to 2000.