CSCI 3104 **Chen, Hoenigman**
Problem Set 5 **Fall 2017, CU-Boulder**

This assignment contains written and programming components. Programming should be done in Python. Written portions should be type-set similar to other assignments. Submit both your .py and .pdf files to Moodle as a .zip file.

1. (10 pts) Given the balanced binary trees T_1 and T_2, which contain m and n elements respectively, we want to determine whether they have some particular key in common. Assume an adversarial sequence that placed the m and n items into the two trees.
   (a) Suppose our algorithm traverses each node of T_1 using an in-order traversal and checks if the key corresponding to the current node traversed exists in T_2. Express the asymptotic running time of this procedure, in terms of m and n.
   (b) Now suppose our algorithm first allocates a new hash table H_1 of size m (assume H_1 uses a uniform hash function) and then inserts every key in T_1 into H_1 during a traversal of T_1. Then, we traverse the tree T_2 and search for whether the key of each node traversed already exists in H_1. Give the asymptotic running time of this algorithm in the average case. Justify your answer.

2. (45 pts) A good hash function $h(x)$ behaves in practice very close to the uniform hashing assumption analyzed in class, but is a deterministic function. That is, $h(x) = k$ each time x is used as an argument to $h()$. Designing good hash functions is hard, and a bad hash function can cause a hash table to quickly exit the sparse loading regime by overloading some buckets and under loading others. Good hash functions often rely on beautiful and complicated insights from number theory, and have deep connections to pseudorandom number generators and cryptographic functions. In practice, most hash functions are moderate to poor approximations of uniform hashing.

   Consider the following two hash functions. Let U be the universe of strings composed of the characters from the alphabet $\Sigma = [\mathtt{A},\ldots,\mathtt{Z}]$, and let the function $f(x_i)$ return the index of a letter $x_i \in \Sigma$, e.g., $f(\mathtt{A}) = 1$ and $f(\mathtt{Z}) = 26$. Finally, for an m-character string $x \in \Sigma^m$, define $h1(x) = ([\sum_{i=1}^{m} f(x_i)] \mod \ell)$, where ℓ is the number of buckets in the hash table. For the other hash function, let the function $f2(x_i, a_i)$ return $f(x_i) * a_i$, where a_i is a uniform random integer, $a_i \in [\mathtt{0},\ldots,\ell\mathtt{-1}]$, and define $h2(x) = ([\sum_{i=1}^{m} f2(x_i, a_i)] \mod \ell)$. That is, the first hash function sums up the index values of the characters of a string x and maps that value onto one of the ℓ buckets, and the second hash function is a universal hash function.

   (a) There is a txt file on Moodle that contains US Census derived last names:
   Using these names as input strings, first choose a uniformly random 50% of these name strings and then hash them using $h1(x)$ and $h2(x)$.

(b) Produce a histogram showing the corresponding distribution of hash locations for each hash function when $\ell = 5701$. Label the axes of your figure. Brief description what the figure shows about $h1(x)$ and $h2(x)$; justify your results in terms of the behavior of $h(x)$.

Hint: the raw file includes information other than the name strings, which will need to be removed; and, think about how you can count hash locations without building or using a real hash table.

(c) Enumerate at least 4 reasons why $h1(x)$ is a bad hash function relative to the ideal behavior of uniform hashing.

(d) Produce a plot showing (i) the length of the longest chain (were we to use chaining for resolving collisions) as a function of the number n of these strings that we hash into a table with $\ell = 5701$ buckets. Produce another plot showing the number of collisions as a function of ℓ. Choose prime numbers for ℓ and comment on how collisions decrease as ℓ increases.

3. (15 pts) Voldemort is writing a secret message to his lieutenants and wants to prevent it from being understood by mere Muggles. He decides to use Huffman encoding to encode the message. Magically, the symbol frequencies of the message are given by the *Lucas numbers*, a famous sequence of integers discovered by the same person who discovered the *Fibonacci numbers*. The nth Lucas number is defined as $L_n = L_{n-1} + L_{n-2}$ for $n > 1$ with base cases $L_0 = 2$ and $L_1 = 1$.

(a) For an alphabet of $\Sigma = \{a, b, c, d, e, f, g, h\}$ with frequencies given by the first $|\Sigma|$ Lucas numbers, give an optimal Huffman code and the corresponding encoding tree for Voldemort to use.

(b) Generalize your answer to (3a) and give the structure of an optimal code when the frequencies are the first n Lucas numbers.