CHAPTER 5 – PROBLEMS

Problem 1. Imagine an application program that behaves like an English dictionary. The user types a word and the program provides the word's defi nition. Thus, the dictionary needs only a retrieval operation. Which implementation of the ADT dictionary would be most effi cient as an English dictionary?

Problem 2. Suppose that a dynamic set S is represented by a direct-address table T of length m. Describe a procedure that finds the maximum element of S. What is the worst-case performance of your procedure?

Problem 3. Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\}: k \neq l \text{ and } h(k) = h(l)\}$?

Problem 4. Demonstrate what happens when we insert the keys 5; 28; 19; 15; 20; 33; 12; 17; 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \bmod 9$.

Problem 5. Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

Problem 6. Given a hash table T, $m = 11$, the hash function is $h(k) = k \bmod m$. Demonstrate what happens when we insert the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table with collisions resolved by:

a. Chaining
b. Linear probing
c. Quadratic probing
d. Double hashing (where $h'(k) = 1 + (k \bmod (m - 1))$)

Problem 7. The success of a hash-table implementation of the ADT dictionary is related to the choice of a good hash function. A good hash function is one that is easy to compute and will evenly distribute the possible data. Comment on the appropriateness of the following hash functions. What patterns would hash to the same location?

a. The hash table has size 2,048. The search keys are English words. The hash function is
   $h(key)$ = (Sum of positions in alphabet of key's letters) mod 2048
b. The hash table has size 2,048. The keys are strings that begin with a letter. The hash function is
   $h(key)$ = (position in alphabet of fi rst letters key) mod 2048
   Thus, "BUT" maps to 2. How appropriate is this hash function if the strings are random? What if the strings are English words?

c. The hash table is 10,000 entries long. The search keys are integers in the range 0 through 9999. The hash function is:

   $h(key)$ = (key* random) truncated to an integer

   where random represents a sophisticated random-number generator that returns a real value between 0 and 1.