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 definition. Thus, the dictionary needs only a retrieval operation. Which implementation of the ADT dictionary would be most efficient as an English dictionary?

Hash Table because it offer O(1) time retrieval time. When the user types a word, the program can hash that word to find its position in the hash table easily.

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?

1. Start from the last element of T (index m-1)
2. Move backward to the first element of T (index 0), searching for the first non-empty slot. That is the maximum element of S

The worst case happens when you have to scan through the whole table T (i.e., only T[0] has value).

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 $\{\{k,l\}: k \neq l \text{ and } h(k) = h(l)\}$?

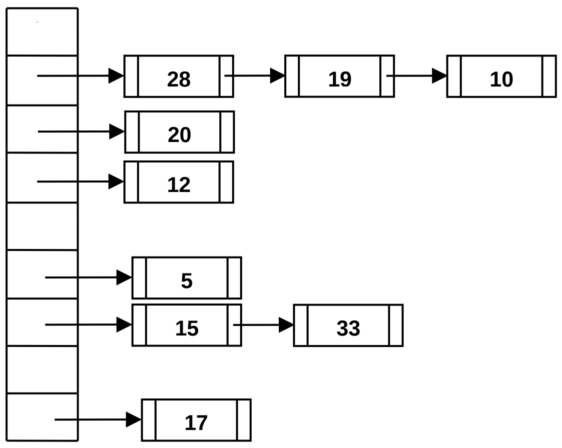
For each pair of key k, l, where $k \neq l$, define the indicator random variable $X_{lk} = I\{h(k) = h(l)\}$. Since we assume simple uniform hashing, we have

$$Pr(X_{lk} = 1) = \Pr\{h(k) = h(l)\} = 1/m$$

Define the random variable Y to be the total number of collisions, $Y = \sum_{k \neq l} X_{kl}$. The expected number of collisions is

$$E[Y] = E\left[\sum_{k \neq l} X_{kl}\right] = \sum_{k \neq l} E[X_{kl}] = \sum_{k \neq l} \frac{1}{m} = \binom{2}{n}\frac{1}{m} = \frac{n(n-1)}{2m}$$

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 $h(k) = k \bmod 9$.

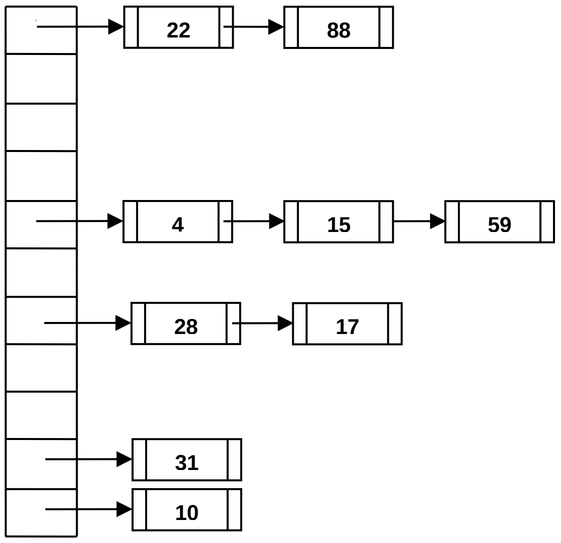


Problem 5. Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

- ✓ *Successful searches*: is identical to the original running time, $O(1+\alpha)$.
- ✓ *Unsuccessful searches*: is better than the original running time because we can stop searching if we find a larger value. If the probability of one element's value falling between two consecutive elements in the hash slot is uniformly distributed, the running time for unsuccessful searches is a half of the original running time.
- ✓ *Insertions*: $O(1+\alpha)$, compared to the original running time of $O(1)$. This is because we need to find the right location to insert the element so that the list remains sorted.
- ✓ *Deletions*: is identical to the original running time, $O(1+\alpha)$

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

22
88
4
15
28
17
59
31
10

c. Quadratic probing
Suppose $c_1 = 1, c_2 = 2$

22
88
4
17
28
15
59
31
10

d. Double hashing (where $h'(k) = 1 + (k \bmod (m - 1))$)

22
59
17
4
15
28
88
31
10

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

$$h(key) = \text{(Sum of positions in alphabet of key's letters) mod 2048}$$

This hash function is easy to implement, but it will not evenly distribute the data through the hash table. Words with similar letters would likely hash to the same position. For example, "CARE", "RACE", "ACER" would have the same hash value.

b. The hash table has size 2,048. The keys are strings that begin with a letter. The hash function is

$$h(key) = \text{(position in alphabet of first letters key) mod 2048}$$

Thus, "BUT" maps to 2. How appropriate is this hash function if the strings are random? What if the strings are English words?

This hash function only considers the first letter of a string. Hence, there are only 26 positions used and there would be a lot of collisions among strings.

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

$$h(key) = \text{(key* random) truncated to an integer}$$

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

If the random-number generator provides a good distribution of random values, this hash function could potentially distribute the keys evenly. Otherwise, the quality of the random number generator would affect the distribution of keys.