Homework # 2 **Due April 7, 2014**

Do Exercises E6 on page 409 and E1 on page 415.

Exercises 9.6

E1. Prove by mathematical induction that $1+3+5+\cdots+(2i-1)=i^2$ for all integers $i>0$.

E2. Write a C++ function to insert an entry into a hash table with open addressing using linear probing.

E3. Write a C++ function to retrieve an entry from a hash table with open addressing using **(a)** linear probing; **(b)** quadratic probing.

E4. In a student project for which the keys were integers, one student thought that he could mix the keys well by using a trigonometric function, which had to be converted to an integer index, so he defined his hash function as

(**int**) sin(n).

What was wrong with this choice? He then decided to replace the function sin(n) by exp(n). Criticize this choice.

E5. Devise a simple, easy to calculate hash function for mapping three-letter words to integers between 0 and $n-1$, inclusive. Find the values of your function on the words

PAL LAP PAM MAP PAT PET SET SAT TAT BAT

for $n = 11, 13, 17, 19$. Try for as few collisions as possible.

E6. Suppose that a hash table contains hash_size = 13 entries indexed from 0 through 12 and that the following keys are to be mapped into the table:

10 100 32 45 58 126 3 29 200 400 0

(a) Determine the hash addresses and find how many collisions occur when these keys are reduced by applying the operation % hash_size.

(b) Determine the hash addresses and find how many collisions occur when these keys are first folded by adding their digits together (in ordinary decimal representation) and then applying % hash_size.

perfect hash functions

(c) Find a hash function that will produce no collisions for these keys. (A hash function that has no collisions for a fixed set of keys is called ***perfect***.)

(d) Repeat the previous parts of this exercise for hash_size = 11. (A hash function that produces no collision for a fixed set of keys that completely fill the hash table is called ***minimal perfect***.)

E7. Another method for resolving collisions with open addressing is to keep a separate array called the ***overflow table***, into which are put all entries that collide with an occupied location. They can either be inserted with another hash function or simply inserted in order, with sequential search used for retrieval. Discuss the advantages and disadvantages of this method.

E8. Write the following functions for processing a chained hash table, using the function sequential_search( ) of Section 7.2 and the list-processing operations of Section 6.1 to implement the operations.

Load factor	0.1	0.5	0.8	0.9	0.99	2.0
Successful search, average number of probes:						
Chaining	1.04	1.2	1.4	1.4	1.5	2.0
Open, quadratic probes	1.04	1.5	2.1	2.7	5.2	—
Open, linear probes	1.05	1.6	3.4	6.2	21.3	—
Unsuccessful search, average number of probes:						
Chaining	0.10	0.50	0.80	0.90	0.99	2.00
Open, quadratic probes	1.13	2.2	5.2	11.9	126.	—
Open, linear probes	1.13	2.7	15.4	59.8	430.	—

Figure 9.16. Empirical comparison of hashing methods

conclusions

In comparison with other methods of information retrieval, the important thing to note about all these numbers is that they depend only on the load factor, not on the absolute number of entries in the table. Retrieval from a hash table with 20,000 entries in 40,000 possible positions is no slower, on average, than is retrieval from a table with 20 entries in 40 possible positions. With sequential search, a list 1000 times the size will take 1000 times as long to search. With binary search, this ratio is reduced to 10 (more precisely, to lg 1000), but still the time needed increases with the size, which it does not with hashing.

Highlights

We can summarize these observations for retrieval from n entries as follows:

- Sequential search is $\Theta(n)$.
- Binary search is $\Theta(\log n)$.
- Hash-table retrieval is $\Theta(1)$.

Finally, we should emphasize the importance of devising a good hash function, one that executes quickly and maximizes the spread of keys. If the hash function is poor, the performance of hashing can degenerate to that of sequential search.

Exercises 9.7

E1. Suppose that each entry in a hash table occupies s words of storage (exclusive of the pointer member needed if chaining is used), where we take one *word* as the amount of space needed for a pointer. Also suppose that there are n occupied entries in the hash table, and the hash table has a total of t possible positions (t is the same as hash_size), including occupied and empty positions.

(a) If open addressing is used, determine how many words of storage will be required for the hash table.

(b) If chaining is used, then each node will require $s + 1$ words, including the pointer member. How many words will be used altogether for the n nodes?

(c) If chaining is used, how many words will be used for the hash table itself? (Recall that with chaining the hash table itself contains only pointers requiring one word each.)

(d) Add your answers to the two previous parts to find the total storage requirement for chaining.

(e) If s is small (that is, the entries have a small size), then open addressing requires less total memory for a given load factor $\lambda = n/t$, but for large s (large entries), chaining requires less space altogether. Find the break-even value for s, at which both methods use the same total storage. Your answer will be a formula for s that depends on the load factor λ, but it should not involve the numbers t or n directly.

(f) Evaluate and graph the results of your formula for values of λ ranging from 0.05 to 0.95.

E2. One reason why the answer to the birthday problem is surprising is that it differs from the answers to apparently related questions. For the following, suppose that there are n people in the room, and disregard leap years.

(a) What is the probability that someone in the room will have a birthday on a random date drawn from a hat?

(b) What is the probability that at least two people in the room will have that same random birthday?

(c) If we choose one person and find that person's birthday, what is the probability that someone else in the room will share the birthday?

ordered hash table

E3. In a chained hash table, suppose that it makes sense to speak of an order for the keys, and suppose that the nodes in each chain are kept in order by key. Then a search can be terminated as soon as it passes the place where the key should be, if present. How many fewer probes will be done, on average, in an unsuccessful search? In a successful search? How many probes are needed, on average, to insert a new node in the right place? Compare your answers with the corresponding numbers derived in the text for the case of unordered chains.

E4. In our discussion of chaining, the hash table itself contained only lists, one for each of the chains. One variant method is to place the first actual entry of each chain in the hash table itself. (An empty position is indicated by an impossible key, as with open addressing.) With a given load factor, calculate the effect on space of this method, as a function of the number of words (except links) in each entry. (A link takes one word.)