Department of CSE, PES University, Bangalore - 560085

Subject Title: Data Structures and its Applications
Subject Code: UE19CS202
Unit-5

Question & Answer

7.3.12. Show how to implement a trie in external storage. Writc a C search-and-insert routine for a trie Tree.

7.4.2. Write a C function *search ,ahk, key* that search(table, key) that searches for a record with key *key.* The function accepts an integer key and a table declared by
struct record
KEY TYPE k;
RECTYPE r;
int flag;
} array[TABLESIZE);
Table[i] .k and table[i*].r* are the ith *key* and record respectively. Table[i].flag equals FALSE. I*f* the ith table position is empty and *TRUE, if it is pre*occupied. The routine returns an integer, in the range of 0 to table-1. If a record is present in the table. Otherwise, the function returns -1. It no such record exists. the **function** returns - *I* . Assume a hashing routine *h(key),* and a rehashing routine *rh(index)* that both produce integers in ,the range of 0 to tablesize-1.

7.4.2. Write a e function *sinsert(rable. key, rec)* to search and insert into a hash table as in Exercise 7.4.1.

7.4.3. Develop a mechanism for detecting when all possible rehash positions of a given key have been searched. Incorporate this method into the C routines *search* and *sinserl* of the previous exercises.

7.4.4. Consider a double hahing method using primary hash function *hi (key)* and rehash function *th(i) tablesize % (i + h2(key), tablesize).* Assume that *h2(key)* is relatively prime to *tablesize.* for any key *key.* Develop a search algorithm and an algorithm to insert a record whose key is known not to exist in the table so that the keys at sucessive rehashes of a single key are in ascending order. The insertion algorithm may rearrange records previously inserted into the table. Can you extend these algorithms to a search and insertion algorithm?

7.4.5 Suppose that a key is equally likely to be any integer between *a* and *b. Suppose* the midsquare hash method is used to produce an integer between 0 and *2* 1 *.* Is the result equally likely to be any integer within that range? Why?

7.4.6, Given a hash function *h(key),* write a C simulation program to determine each of the following quantities after 0.8 *tablesize* random keys have been generated. The keys should be. random integers.

1. the percentage of integers between 0 and *tablesize - I* that do not equal *h(key)* for some generated key
2. the percentage of integers between 0 and *tablesize - I* that equal *h(key)* for more than one generated key
3. the maximum number of keys that hash into a single value between 0 and *tablesize - 1*
4. the average number of keys that hash into values between 0 and *zablesize - 1,* not including those values into which no key hashes

Run the program to test the uniformity of each of the following hash functions.

(a) h(key) = key % tablesize for *zablesize* a prime

(b) h(key) = key % *Zablesize* for *tablesize* a power of 2

(c) The folding method using *exclusive or* to produce five-bit indices, where *tab!esize 32*

(d) The mid-square method using decimal arithmetic to produce four-digit indexes, where *tab!esize 10,000*

7.4.7. If a hash table Contains *uzblesize* positions, and *n* records currently occupy the table, the *Ioadfactor* is defined as *nifahiesize.* Show that if a hash function uniformly distributes keys over the *tablesize* positions of the table and if *if* is the load factor of the table. (a - I) . !*fl2* of then keys in the table collided upon insertion with a previously entered key.

7.4.8. Assume that *n* random positions of a *tablexize-element* hash table are occupied, using hash and rehash functions that are equally likely to produce any index in the table. Show that the average number of comparisons needed to insert a new element is *(rablesize + I)I(tablesize - a + I).* Explain why linear probing does not Satisfy this condition. -