

Question & Answer

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

Routine for inserting a node into TRIE tree .

```
void insert(struct trienode* root, char *key)
{
    struct trienode *curr;
    int i, index;

    curr = root;
    for(i=0;key[i]!='\0';i++)
    { index=key[i];
        if(curr->child[index]==NULL)
            curr->child[index]=getnode();
        curr=curr->child[index];
    }
    curr->endofword=1;
}
```

Routine for searching a node into TRIE tree .

```
int search(struct trienode * root, char *key)
{
    int i, index;
    struct trienode *curr;
    curr=root;
    for(i=0;key[i]!='\0';i++)
    { index=key[i];
        if(curr->child[index]==NULL)
            return 0;
        curr=curr->child[index];
    }
    if(curr->endofword==1)
        return 1;
    return 0;
}
```

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 *i*th *key* and record respectively. *Table[i].flag* equals FALSE. If the *i*th table position is empty and TRUE, if it is preoccupied. 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. If no such record exists. the **function** returns -1. Assume a hashing routine *h(key)*, and a rehashing routine *rh(index)* that both produce integers in the range of 0 to tablesize-1.

```

void insertHash(int key)
{
    // if hash table is full
    if (isFull())
        return;

    // get index from first hash
    int index = hash1(key);

    // if collision occurs
    if (hashTable[index] != -1) {
        // get index2 from second hash
        int index2 = hash2(key);
        int i = 1;
        while (1) {
            // get newIndex
            int newIndex = (index + i * index2) % TABLE_SIZE;

            // if no collision occurs, store
            // the key
            if (hashTable[newIndex] == -1) {
                hashTable[newIndex] = key;
                break;
            }
            i++;
        }
    }

    // if no collision occurs
    else
        hashTable[index] = key;
    curr_size++;
}

```