DATA STRUCTURE

DAY 9- 05/08/2024

1.hashing using seperate chaining

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 10
 typedef struct Entry {
    char* key;
    int value;
    struct Entry* next;
} Entry;
 typedef struct HashTable {
    Entry* table[TABLE_SIZE];
} HashTable;
 unsigned int hash(char* key) {
    unsigned int hash = 0;
    while (*key) {
        hash = (hash << 5) + *key++;
    }
    return hash % TABLE_SIZE;
}
HashTable* createHashTable() {
    HashTable* hashTable = (HashTable*)malloc(sizeof(HashTable));
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable->table[i] = NULL;
    }
    return hashTable;
```

```
}
 void insert(HashTable* hashTable, char* key, int value) {
    unsigned int index = hash(key);
    Entry* newEntry = (Entry*)malloc(sizeof(Entry));
    newEntry->key = strdup(key);
    newEntry->value = value;
    newEntry->next = hashTable->table[index];
    hashTable->table[index] = newEntry;
}
int search(HashTable* hashTable, char* key) {
    unsigned int index = hash(key);
    Entry* entry = hashTable->table[index];
    while (entry != NULL) {
        if (strcmp(entry->key, key) == 0) {
            return entry->value;
        }
        entry = entry->next;
    }
    return -1;
}
 void delete(HashTable* hashTable, char* key) {
    unsigned int index = hash(key);
    Entry* entry = hashTable->table[index];
    Entry* prev = NULL;
    while (entry != NULL) {
        if (strcmp(entry->key, key) == 0) {
            if (prev == NULL) {
                hashTable->table[index] = entry->next;
            } else {
```

```
                prev->next = entry->next;
            }
            free(entry->key);
            free(entry);
            return;
        }
        prev = entry;
        entry = entry->next;
    }
}
 void freeHashTable(HashTable* hashTable) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Entry* entry = hashTable->table[i];
        while (entry != NULL) {
            Entry* temp = entry;
            entry = entry->next;
            free(temp->key);
            free(temp);
        }
    }
    free(hashTable);
}
 int main() {
    HashTable* hashTable = createHashTable();
    insert(hashTable, "name", 1);
    insert(hashTable, "age", 30);
    insert(hashTable, "height", 175);
    printf("Name: %d\n", search(hashTable, "name"));
    printf("Age: %d\n", search(hashTable, "age"));
```

```
    printf("Height: %d\n", search(hashTable, "height"));

    delete(hashTable, "age");
    printf("Age after deletion: %d\n", search(hashTable, "age"));
    freeHashTable(hashTable);
    return 0;
}
```

OUTPUT:

```
Name: 1
Age: 30
Height: 175
Age after deletion: -1
```

2.hashing using linear probling

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TABLE_SIZE 10
 typedef struct {
    char* key;
    int value;
    int occupied;
} HashTableEntry;
 typedef struct {
    HashTableEntry* table[TABLE_SIZE];
} HashTable;
 unsigned int hash(char* key) {
```

```
    unsigned int hash = 0;
    while (*key) {
        hash = (hash << 5) + *key++;
    }
    return hash % TABLE_SIZE;
}
HashTable* createHashTable() {
    HashTable* hashTable = (HashTable*)malloc(sizeof(HashTable));
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable->table[i] = (HashTableEntry*)malloc(sizeof(HashTableEntry));
        hashTable->table[i]->key = NULL;
        hashTable->table[i]->occupied = 0;
    }
    return hashTable;
}
 void insert(HashTable* hashTable, char* key, int value) {
    unsigned int index = hash(key);
    unsigned int originalIndex = index;
    while (hashTable->table[index]->occupied && hashTable->table[index]->key &&
strcmp(hashTable->table[index]->key, key) != 0) {
        index = (index + 1) % TABLE_SIZE;
        if (index == originalIndex) {
            return;
        }
    }
     if (hashTable->table[index]->key) {
        free(hashTable->table[index]->key);
    } else {
        hashTable->table[index]->key = (char*)malloc(strlen(key) + 1);
```

```
    }
    strcpy(hashTable->table[index]->key, key);
    hashTable->table[index]->value = value;
    hashTable->table[index]->occupied = 1;
}
 int search(HashTable* hashTable, char* key) {
    unsigned int index = hash(key);
    unsigned int originalIndex = index;
    while (hashTable->table[index]->occupied) {
        if (hashTable->table[index]->key && strcmp(hashTable->table[index]->key, key) == 0)
{
            return hashTable->table[index]->value;
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == originalIndex) {
            break;
        }
    }
    return -1;
}
 void delete(HashTable* hashTable, char* key) {
    unsigned int index = hash(key);
    unsigned int originalIndex = index;
    while (hashTable->table[index]->occupied) {
        if (hashTable->table[index]->key && strcmp(hashTable->table[index]->key, key) == 0)
{
            free(hashTable->table[index]->key);
            hashTable->table[index]->key = NULL;
            hashTable->table[index]->occupied = 0;
            return;
```

```
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == originalIndex) {
            break;
        }
    }
}
 void freeHashTable(HashTable* hashTable) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable->table[i]->key) {
            free(hashTable->table[i]->key);
        }
        free(hashTable->table[i]);
    }
    free(hashTable);
}
int main() {
    HashTable* hashTable = createHashTable();
    insert(hashTable, "name", 1);
    insert(hashTable, "age", 30);
    insert(hashTable, "height", 175);
    printf("Name: %d\n", search(hashTable, "name"));
    printf("Age: %d\n", search(hashTable, "age"));
    printf("Height: %d\n", search(hashTable, "height"));
    delete(hashTable, "age");
    printf("Age after deletion: %d\n", search(hashTable, "age"));
    freeHashTable(hashTable);
    return 0;
}
```

OUTPUT:

Name: 1

Age: 30

Height: 175

Age after deletion: -1