Practical-09

Implement a hash table data structure using different hash function and collision resolution techniques such as chaining and open addressing.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 10

// Structure for Chaining (Linked List) method

struct Node {

int key;

struct Node\* next;

};

// Structure for Open Addressing method

struct HashTable {

int \*table;

int \*status; // to track the state of each slot: 0 = empty, 1 = occupied, 2 = deleted

};

// Function to create a new node for chaining

struct Node\* createNode(int key) {

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

newNode->key = key;

newNode->next = NULL;

return newNode;

}

// Hash function 1 (Simple modulo based)

int hashFunction1(int key) {

return key % TABLE\_SIZE;

}

// Hash function 2 (Multiplicative method)

int hashFunction2(int key) {

double A = 0.6180339887; // A constant (fractional part of golden ratio)

double temp = key \* A;

return (int)(TABLE\_SIZE \* (temp - (int)temp));

}

// Chaining Method: Insert key into hash table

void insertChaining(struct Node\* hashTable[], int key) {

int index = hashFunction1(key);

struct Node\* newNode = createNode(key);

if (hashTable[index] == NULL) {

hashTable[index] = newNode;

} else {

newNode->next = hashTable[index];

hashTable[index] = newNode;

}

}

// Chaining Method: Search for a key in hash table

int searchChaining(struct Node\* hashTable[], int key) {

int index = hashFunction1(key);

struct Node\* temp = hashTable[index];

while (temp) {

if (temp->key == key)

return 1;

temp = temp->next;

}

return 0;

}

// Open Addressing (Linear Probing) Method: Initialize hash table

void initHashTable(struct HashTable\* ht) {

ht->table = (int\*)malloc(sizeof(int) \* TABLE\_SIZE);

ht->status = (int\*)malloc(sizeof(int) \* TABLE\_SIZE);

for (int i = 0; i < TABLE\_SIZE; i++) {

ht->table[i] = -1; // -1 means empty slot

ht->status[i] = 0; // 0 means empty

}

}

// Open Addressing Method: Insert key into hash table (Linear Probing)

void insertOpenAddressing(struct HashTable\* ht, int key) {

int index = hashFunction1(key);

int i = 0;

while (i < TABLE\_SIZE) {

int probeIndex = (index + i) % TABLE\_SIZE;

if (ht->status[probeIndex] == 0 || ht->status[probeIndex] == 2) { // empty or deleted slot

ht->table[probeIndex] = key;

ht->status[probeIndex] = 1; // Mark slot as occupied

return;

}

i++;

}

printf("Hash table is full\n");

}

// Open Addressing Method: Search for key in hash table (Linear Probing)

int searchOpenAddressing(struct HashTable\* ht, int key) {

int index = hashFunction1(key);

int i = 0;

while (i < TABLE\_SIZE) {

int probeIndex = (index + i) % TABLE\_SIZE;

if (ht->status[probeIndex] == 0) {

return 0; // Key not found

}

if (ht->status[probeIndex] == 1 && ht->table[probeIndex] == key) {

return 1; // Key found

}

i++;

}

return 0;

}

// Function to display hash table contents (Chaining)

void displayChaining(struct Node\* hashTable[]) {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] == NULL) {

printf("Index %d: Empty\n", i);

} else {

printf("Index %d: ", i);

struct Node\* temp = hashTable[i];

while (temp) {

printf("%d -> ", temp->key);

temp = temp->next;

}

printf("NULL\n");

}

}

}

// Function to display hash table contents (Open Addressing)

void displayOpenAddressing(struct HashTable\* ht) {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (ht->status[i] == 0) {

printf("Index %d: Empty\n", i);

} else if (ht->status[i] == 1) {

printf("Index %d: %d\n", i, ht->table[i]);

} else {

printf("Index %d: Deleted\n", i);

}

}

}

int main() {

// Chaining Implementation

struct Node\* hashTableChaining[TABLE\_SIZE] = {0}; // Initialize hash table with NULL

insertChaining(hashTableChaining, 10);

insertChaining(hashTableChaining, 20);

insertChaining(hashTableChaining, 30);

insertChaining(hashTableChaining, 5);

printf("Chaining Method Hash Table:\n");

displayChaining(hashTableChaining);

// Search in Chaining

printf("Searching for key 25: %s\n", searchChaining(hashTableChaining, 25) ? "Found" : "Not Found");

// Open Addressing (Linear Probing) Implementation

struct HashTable ht;

initHashTable(&ht);

insertOpenAddressing(&ht, 10);

insertOpenAddressing(&ht, 20);

insertOpenAddressing(&ht, 30);

insertOpenAddressing(&ht, 5);

printf("\nOpen Addressing Hash Table:\n");

displayOpenAddressing(&ht);

// Search in Open Addressing

printf("Searching for key 25: %s\n", searchOpenAddressing(&ht, 25) ? "Found" : "Not Found");

return 0;

}

Output:

