231501058 CS23231 – D a t a S t r u c t u r e s

**Ex. No.: 16**

**Hashing**

**Date: 31/5/24**

**Write a C prog ram to create a hash table and perform collision resolution using the following techniques .**

1. **Open addressing**
2. **Closed Addressin g**
3. **Rehashin g**

**Algorithm:**

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct HashTable {

int size;

int count;

int\* keys;

int\* values;

bool\* isOccupied;

} HashTable;

HashTable\* createTable(int size) {

HashTable\* newTable = (HashTable\*)malloc(sizeof(HashTable));

newTable->size = size;

newTable->count = 0;

newTable->keys = (int\*)malloc(sizeof(int) \* size); newTable->values = (int\*)malloc(sizeof(int) \* size); newTable->isOccupied = (bool\*)malloc(sizeof(bool) \* size); for (int i = 0; i < size; i++) {

newTable->isOccupied[i] = false;

}

return newTable;

}

int hashFunction(int key, int size) {

return key % size;

}

void rehash(HashTable\* hashTable);

void insert(HashTable\* hashTable, int key, int value) {

if ((float)hashTable->count / hashTable->size >= 0.75) { rehash(hashTable);

}



**Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College**

. 54

231501058 CS23231 – D a t a S t r u c t u r e s

int hashIndex = hashFunction(key, hashTable->size); int originalIndex = hashIndex; int i = 1;

while (hashTable->isOccupied[hashIndex]) { if (hashTable->keys[hashIndex] == key) {

hashTable->values[hashIndex] = value;

return;

}

hashIndex = (originalIndex + i) % hashTable->size; i++;

}

hashTable->keys[hashIndex] = key;

hashTable->values[hashIndex] = value;

hashTable->isOccupied[hashIndex] = true;

hashTable->count++;

}

void rehash(HashTable\* hashTable) {

int oldSize = hashTable->size;

int\* oldKeys = hashTable->keys;

int\* oldValues = hashTable->values;

bool\* oldIsOccupied = hashTable->isOccupied;

int newSize = oldSize \* 2;

hashTable->keys = (int\*)malloc(sizeof(int) \* newSize);

hashTable->values = (int\*)malloc(sizeof(int) \* newSize);

hashTable->isOccupied = (bool\*)malloc(sizeof(bool) \* newSize);

hashTable->size = newSize;

hashTable->count = 0;

for (int i = 0; i < newSize; i++) {

hashTable->isOccupied[i] = false;

}

for (int i = 0; i < oldSize; i++) {

if (oldIsOccupied[i]) {

insert(hashTable, oldKeys[i], oldValues[i]);

}

}

free(oldKeys);

free(oldValues);

free(oldIsOccupied);

}

int search(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); int originalIndex = hashIndex;



**Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College**

. 55

231501058 CS23231 – D a t a S t r u c t u r e s

int i = 1;

while (hashTable->isOccupied[hashIndex]) { if (hashTable->keys[hashIndex] == key) {

return hashTable->values[hashIndex];

}

hashIndex = (originalIndex + i) % hashTable->size; i++;

if (hashIndex == originalIndex) {

break;

}

}

return -1;

}

void delete(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); int originalIndex = hashIndex; int i = 1;

while (hashTable->isOccupied[hashIndex]) { if (hashTable->keys[hashIndex] == key) {

hashTable->isOccupied[hashIndex] = false;

hashTable->count--;

return;

}

hashIndex = (originalIndex + i) % hashTable->size; i++;

if (hashIndex == originalIndex) {

break;

}

}

}

void freeTable(HashTable\* hashTable) {

free(hashTable->keys);

free(hashTable->values);

free(hashTable->isOccupied);

free(hashTable);

|  |  |
| --- | --- |
| } |  |
| int main() { |  |
| HashTable\* hashTable | = createTable(5); |
| insert(hashTable, 1, | 10); |
| insert(hashTable, 2, | 20); |
| insert(hashTable, 3, | 30); |
| insert(hashTable, 4, | 40); |
| insert(hashTable, 5, | 50); |
| insert(hashTable, 6, | 60); |

printf("Value for key 1: %d\n", search(hashTable, 1));



|  |  |
| --- | --- |
| **Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College** | . 56 |

231501058

CS23231 – D a t a S t r u c t u r e s

printf("Value for key 2: %d\n", search(hashTable, 2));

printf("Value for key 3: %d\n", search(hashTable, 3));

printf("Value for key 4: %d\n", search(hashTable, 4));

printf("Value for key 5: %d\n", search(hashTable, 5));

printf("Value for key 6: %d\n", search(hashTable, 6));

delete(hashTable, 3);

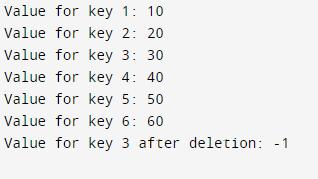
printf("Value for key 3 after deletion: %d\n", search(hashTable, 3));

freeTable(hashTable);

return 0;

}

**OUTPUT**



2.CLOSED ADDRESSING

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

int key;

int value;

struct Node\* next;

} Node;

typedef struct HashTable {

int size;

Node\*\* table;

} HashTable;

Node\* createNode(int key, int value) {

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

newNode->key = key;

newNode->value = value;

newNode->next = NULL;

return newNode;

}

HashTable\* createTable(int size) {

HashTable\* newTable = (HashTable\*)malloc(sizeof(HashTable)); newTable->size = size;



|  |  |
| --- | --- |
| **Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College** | . 57 |

231501058 CS23231 – D a t a S t r u c t u r e s

newTable->table = (Node\*\*)malloc(sizeof(Node\*) \* size); for (int i = 0; i < size; i++) {

newTable->table[i] = NULL;

}

return newTable;

}

int hashFunction(int key, int size) {

return key % size;

}

void insert(HashTable\* hashTable, int key, int value) { int hashIndex = hashFunction(key, hashTable->size); Node\* newNode = createNode(key, value); newNode->next = hashTable->table[hashIndex]; hashTable->table[hashIndex] = newNode;

}

int search(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); Node\* current = hashTable->table[hashIndex]; while (current != NULL) {

if (current->key == key) {

return current->value;

}

current = current->next;

}

return -1;

}

void delete(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); Node\* current = hashTable->table[hashIndex]; Node\* prev = NULL;

while (current != NULL && current->key != key) { prev = current;

current = current->next;

}

if (current == NULL) {

return;

}

if (prev == NULL) {

hashTable->table[hashIndex] = current->next;

} else {

prev->next = current->next;

}

free(current);

|  |  |
| --- | --- |
| } |  |
| void freeTable(HashTable\* hashTable) { |  |
| **Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College** | . 58 |



231501058 CS23231 – D a t a S t r u c t u r e s

for (int i = 0; i < hashTable->size; i++) { Node\* current = hashTable->table[i]; while (current != NULL) {

Node\* temp = current;

current = current->next;

free(temp);

}

}

free(hashTable->table);

free(hashTable);

}

int main() {

HashTable\* hashTable = createTable(10);

insert(hashTable, 1, 10);

insert(hashTable, 2, 20);

insert(hashTable, 12, 30);

printf("Value for key 1: %d\n", search(hashTable, 1));

printf("Value for key 2: %d\n", search(hashTable, 2));

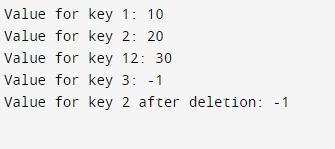
printf("Value for key 12: %d\n", search(hashTable, 12)); printf("Value for key 3: %d\n", search(hashTable, 3)); delete(hashTable, 2);

printf("Value for key 2 after deletion: %d\n", search(hashTable, 2)); freeTable(hashTable);

return 0;

}

**OUTPUT**



**C) REHASHING**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int key;

int value;

struct Node\* next;

} Node;

typedef struct HashTable {



**Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College**

. 59

231501058 CS23231 – D a t a S t r u c t u r e s

int size;

int count; // Number of elements in the table Node\*\* table;

} HashTable;

Node\* createNode(int key, int value) {

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

newNode->key = key;

newNode->value = value;

newNode->next = NULL;

return newNode;

}

HashTable\* createTable(int size) {

HashTable\* newTable = (HashTable\*)malloc(sizeof(HashTable));

newTable->size = size;

newTable->count = 0;

newTable->table = (Node\*\*)malloc(sizeof(Node\*) \* size); for (int i = 0; i < size; i++) {

newTable->table[i] = NULL;

}

return newTable;

}

int hashFunction(int key, int size) {

return key % size;

}

void insert(HashTable\* hashTable, int key, int value); void rehash(HashTable\* hashTable) {

int oldSize = hashTable->size;

Node\*\* oldTable = hashTable->table;

int newSize = oldSize \* 2;

hashTable->table = (Node\*\*)malloc(sizeof(Node\*) \* newSize);

hashTable->size = newSize;

hashTable->count = 0;

for (int i = 0; i < newSize; i++) {

hashTable->table[i] = NULL;

|  |  |
| --- | --- |
| } |  |
| for (int i = 0; i < oldSize; i++) { |  |
| Node\* current = oldTable[i]; |  |
| while (current != NULL) { |  |
| insert(hashTable, current->key, current->value); |  |
| Node\* temp = current; |  |
| current = current->next; |  |
| free(temp); |  |
| } |  |
| } |  |
| free(oldTable); |  |
| } |  |
| **Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College** | . 60 |



231501058

CS23231 – D a t a S t r u c t u r e s

void insert(HashTable\* hashTable, int key, int value) {

if ((float)hashTable->count / hashTable->size >= 0.75) {

rehash(hashTable);

}

int hashIndex = hashFunction(key, hashTable->size); Node\* newNode = createNode(key, value); newNode->next = hashTable->table[hashIndex]; hashTable->table[hashIndex] = newNode; hashTable->count++;

}

int search(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); Node\* current = hashTable->table[hashIndex]; while (current != NULL) {

if (current->key == key) {

return current->value;

}

current = current->next;

}

return -1;

}

void delete(HashTable\* hashTable, int key) {

int hashIndex = hashFunction(key, hashTable->size); Node\* current = hashTable->table[hashIndex]; Node\* prev = NULL;

while (current != NULL && current->key != key) { prev = current;

current = current->next;

}

if (current == NULL) {

return;

}

if (prev == NULL) {

hashTable->table[hashIndex] = current->next;

} else {

prev->next = current->next;

}

free(current);

hashTable->count--;

}

void freeTable(HashTable\* hashTable) {

for (int i = 0; i < hashTable->size; i++) {

Node\* current = hashTable->table[i];

while (current != NULL) {

Node\* temp = current;

current = current->next;



|  |  |
| --- | --- |
| **Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College** | . 61 |

231501058 CS23231 – D a t a S t r u c t u r e s

free(temp);

}

}

free(hashTable->table);

free(hashTable);

}

int main() {

HashTable\* hashTable = createTable(5);

insert(hashTable, 1, 10);

insert(hashTable, 2, 20);

insert(hashTable, 3, 30);

insert(hashTable, 4, 40);

insert(hashTable, 5, 50);

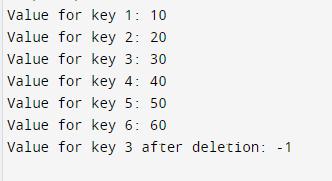
insert(hashTable, 6, 60); // This should trigger rehashing printf("Value for key 1: %d\n", search(hashTable, 1)); printf("Value for key 2: %d\n", search(hashTable, 2)); printf("Value for key 3: %d\n", search(hashTable, 3)); printf("Value for key 4: %d\n", search(hashTable, 4)); printf("Value for key 5: %d\n", search(hashTable, 5)); printf("Value for key 6: %d\n", search(hashTable, 6)); delete(hashTable, 3);

printf("Value for key 3 after deletion: %d\n", search(hashTable, 3)); freeTable(hashTable);

return 0;

}

**OUTPUT**



**Dept of Artificial Intelligence and Machine Learning** | **Rajalakshmi Engineering College**

. 62