**Hash Table with Chaining**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 10

typedef struct Node {

char \*key;

char \*value;

struct Node \*next;

} Node;

typedef struct HashTable {

Node \*table[TABLE\_SIZE];

} HashTable;

unsigned int hash\_function(const char \*key) {

unsigned int hash = 0;

while (\*key) {

hash = (hash << 5) + \*key++;

}

return hash % TABLE\_SIZE;

}

HashTable\* create\_table() {

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

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

ht->table[i] = NULL;

}

return ht;

}

void insert(HashTable \*ht, const char \*key, const char \*value) {

unsigned int index = hash\_function(key);

Node \*new\_node = malloc(sizeof(Node));

new\_node->key = strdup(key);

new\_node->value = strdup(value);

new\_node->next = ht->table[index];

ht->table[index] = new\_node;

}

char\* search(HashTable \*ht, const char \*key) {

unsigned int index = hash\_function(key);

Node \*node = ht->table[index];

while (node) {

if (strcmp(node->key, key) == 0) {

return node->value;

}

node = node->next;

}

return NULL;

}

void delete(HashTable \*ht, const char \*key) {

unsigned int index = hash\_function(key);

Node \*node = ht->table[index];

Node \*prev = NULL;

while (node) {

if (strcmp(node->key, key) == 0) {

if (prev) {

prev->next = node->next;

} else {

ht->table[index] = node->next;

}

free(node->key);

free(node->value);

free(node);

return;

}

prev = node;

node = node->next;

}

}

void free\_table(HashTable \*ht) {

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

Node \*node = ht->table[i];

while (node) {

Node \*temp = node;

node = node->next;

free(temp->key);

free(temp->value);

free(temp);

}

}

free(ht);

}

int main() {

HashTable \*ht = create\_table();

insert(ht, "key1", "value1");

insert(ht, "key2", "value2");

insert(ht, "key3", "value3");

printf("Search key2: %s\n", search(ht, "key2"));

printf("Search key3: %s\n", search(ht, "key3"));

delete(ht, "key2");

printf("Search key2 after deletion: %s\n", search(ht, "key2"));

free\_table(ht);

return 0;

}

Hash Table with Linear Probing

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 10

typedef struct {

char \*key;

char \*value;

} HashEntry;

typedef struct {

HashEntry \*\*table;

int size;

} HashTable;

unsigned int hash\_function(const char \*key) {

unsigned int hash = 0;

while (\*key) {

hash = (hash << 5) + \*key++;

}

return hash % TABLE\_SIZE;

}

HashTable\* create\_table() {

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

ht->size = TABLE\_SIZE;

ht->table = malloc(sizeof(HashEntry\*) \* ht->size);

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

ht->table[i] = NULL;

}

return ht;

}

void insert(HashTable \*ht, const char \*key, const char \*value) {

unsigned int index = hash\_function(key);

while (ht->table[index] != NULL && strcmp(ht->table[index]->key, key) != 0) {

index = (index + 1) % ht->size;

}

if (ht->table[index] != NULL) {

free(ht->table[index]->value);

ht->table[index]->value = strdup(value);

} else {

ht->table[index] = malloc(sizeof(HashEntry));

ht->table[index]->key = strdup(key);

ht->table[index]->value = strdup(value);

}

}

char\* search(HashTable \*ht, const char \*key) {

unsigned int index = hash\_function(key);

while (ht->table[index] != NULL) {

if (strcmp(ht->table[index]->key, key) == 0) {

return ht->table[index]->value;

}

index = (index + 1) % ht->size;

}

return NULL;

}

void delete(HashTable \*ht, const char \*key) {

unsigned int index = hash\_function(key);

while (ht->table[index] != NULL) {

if (strcmp(ht->table[index]->key, key) == 0) {

free(ht->table[index]->key);

free(ht->table[index]->value);

free(ht->table[index]);

ht->table[index] = NULL;

return;

}

index = (index + 1) % ht->size;

}

}

void free\_table(HashTable \*ht) {

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

if (ht->table[i] != NULL) {

free(ht->table[i]->key);

free(ht->table[i]->value);

free(ht->table[i]);

}

}

free(ht->table);

free(ht);

}

int main() {

HashTable \*ht = create\_table();

insert(ht, "key1", "value1");

insert(ht, "key2", "value2");

insert(ht, "key3", "value3");

printf("Search key2: %s\n", search(ht, "key2"));

printf("Search key3: %s\n", search(ht, "key3"));

delete(ht, "key2");

printf("Search key2 after deletion: %s\n", search(ht, "key2"));

free\_table(ht);

return 0;

}