**Waleed Afzal**

**23P-0566**

**3- D**

**DSA Assignment #05**

**Ques2:**

#include <iostream>

#include <vector>

using namespace std;

class Library {

vector<int> table;

int size, capacity;

int primaryHash(int key) {

int hash = key % capacity;

if (hash < 0)

return (hash + capacity);

else

return hash;

}

int secondaryHash(int key) {

int hash = 1 + (key % (capacity - 1));

if (hash < 0)

return (hash + capacity);

else

return hash;

}

public:

Library(int c) {

size = 0;

capacity = c;

table.resize(c, -1);

}

bool insertLinearHash(int key) {

int index = primaryHash(key);

int start = index;

while (true) {

if (table[start] == key)

return false;

if (table[start] == -1 || table[start] == -2) {

table[start] = key;

size++;

return true;

}

start = (start + 1) % capacity;

if (start == index)

return false;

}

}

bool insertDoubleHash(int key) {

int index = primaryHash(key);

int start = index;

int i = 0;

while (true) {

if (i == capacity)

return false;

if (table[start] == key)

return false;

if (table[start] == -1 || table[start] == -2) {

table[start] = key;

size++;

return true;

}

start = (index + i \* secondaryHash(key)) % capacity;

i++;

}

}

void display() {

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

if (table[i] == -1)

cout << i << ": Empty\n";

else

cout << i << ": " << table[i] << "\n";

}

}

void rehashing() {

vector<int> oldTable = table;

capacity \*= 2;

table.clear();

table.resize(capacity, -1);

for (int key : oldTable) {

if (key != -1 && key != -2)

insertDoubleHash(key);

}

}

double loadFactor() {

return (double)size / capacity; // Explicit Type Casting from int to double

}

};

int main() {

int fixSize = 15;

Library hashTable(fixSize);

vector<int> ISBNs = { 17, 26, 15, 9, 11, 43, 75, 19, 35, 45, 55, 9, 10, 21, 61, 23 };

cout << "Using Linear Probing:\n";

int i = 0;

for (i; i < fixSize; i++) {

if (hashTable.loadFactor() < 0.7) {

hashTable.insertLinearHash(ISBNs[i]);

continue;

}

break;

}

hashTable.display();

cout << "\nTable is 70% Full. Rehashing...";

hashTable.rehashing();

cout << "\nRehashing Successfull!\n";

hashTable.display();

cout << "\nUsing Double Hashing:\n";

for (i; i < fixSize; i++) {

hashTable.insertDoubleHash(ISBNs[i]);

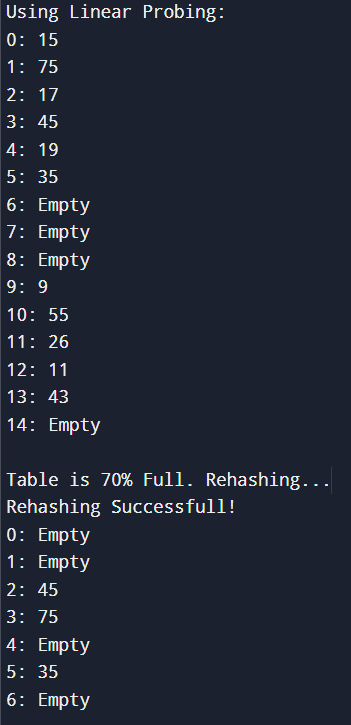
}

hashTable.display();

system("pause");

return 0;

}



A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

asd

**Ques3:**

#include <iostream>

#include <vector>

using namespace std;

struct HashNode {

int SKU;

HashNode\* next;

HashNode() : SKU(-1), next(NULL) {}

HashNode(int sku) : SKU(sku), next(NULL) {}

};

class CatalogChaining {

int size, capacity;

vector<HashNode\*> table;

double loadFactor() const { // Explicity Type Casting

return ((double)size / capacity);

}

int primaryHash(int key) {

int hash = key % capacity;

if (hash < 0)

return (hash + capacity);

else

return hash;

}

void rehash() {

int oldCapacity = capacity;

vector<HashNode\*> oldTable = table;

capacity \*= 2;

size = 0;

table.clear();

table.resize(capacity, NULL);

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

if (oldTable[i] != NULL) {

HashNode\* current = oldTable[i];

while (current != NULL) {

insert(current->SKU);

current = current->next;

}

}

}

// Clean Old Table Memory

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

HashNode\* current = oldTable[i];

while (current != NULL) {

HashNode\* temp = current;

current = current->next;

delete temp;

}

}

}

public:

CatalogChaining(int c) {

capacity = c;

size = 0;

table.resize(capacity, NULL);

}

bool insertChain(int key) {

int index = primaryHash(key);

HashNode\* newNode = new HashNode(key);

if (table[index] == NULL) {

table[index] = newNode;

size++;

return true;

}

HashNode\* current = table[index];

if(current->SKU==key){

return false;

}

while (current->next != NULL) {

if (current->SKU == key)

return false;

current = current->next;

}

current->next = newNode;

size++;

return true;

}

bool insert(int key) {

if (loadFactor() > 0.70) {

rehash();

cout << "\nRehashing Successful!\n";

}

return insertChain(key);

}

bool search(int key, int& returnIndex) {

int index = primaryHash(key);

HashNode\* current = table[index];

returnIndex = index;

while (current != NULL) {

if (current->SKU == key)

return true;

current = current->next;

}

return false;

}

void display() {

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

cout << i << ": ";

HashNode\* current = table[i];

while (current != NULL) {

cout << current->SKU << " -> ";

current = current->next;

}

cout << "NULL\n";

}

}

~CatalogChaining() {

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

HashNode\* current = table[i];

while (current != NULL) {

HashNode\* temp = current;

current = current->next;

delete temp;

}

}

}

};

class CatalogBucketing {

vector<vector<int>> table;

int numBuckets;

int numElements;

double loadFactor() const {

return ((double)numElements / numBuckets);

}

int primaryHash(int key) {

int hash = key % numBuckets;

if (hash < 0)

return (hash + numBuckets);

else

return hash;

}

void rehash() {

vector<vector<int>> oldTable = table;

numBuckets \*= 2;

table.clear();

table.resize(numBuckets);

for (int i = 0; i < oldTable.size(); i++) {

for (int sku : oldTable[i]) {

insert(sku);

}

}

}

public:

CatalogBucketing() : numBuckets(3), numElements(0) {

table.resize(numBuckets);

}

void insert(int key) {

int index = primaryHash(key);

// Check for duplicate values in the bucket

for (int sku : table[index]) {

if (sku == key) {

return;

}

}

table[index].push\_back(key);

numElements++;

if (loadFactor() > 0.7) {

rehash();

}

}

void display() {

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

cout << "Bucket " << i << ": ";

for (int sku : table[i]) {

cout << sku << " ";

}

cout << endl;

}

}

};

int main() {

CatalogChaining cc(15);

vector<int> SKUs = { 17, 26, 15, 9, 11, 43, 75, 19, 35, 45, 55, 9, 10, 21, 61, 23 };

for (int p : SKUs) {

cc.insert(p);

}

CatalogBucketing cb;

cout << "\nChaining Hash Table:\n";

cc.display();

for (int p : SKUs) {

cb.insert(p);

}

cout << "\nBucketing Hash Table:\n";

cb.display();

system("pause");

return 0;

}

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generated**