Assignment no 4

Muhammad Shaheer

Q1: Assignment#04 - Hashing

• Implement Rehashing in C++

• Create a class named HashTable

• While creating the object of the class you should be able

to:

• Fix the size of the hash table

• Select a range of numbers to be stored in the table (your table

should reject numbers outside of that range)

• Create a function for insertion. That should:

• be able to handle collisions by linear probing

• create another table double the size of the original table if the

original table is 70% full (as in rehasing)

• Create a function for searching which should be able to

handle all the scenarios as discussed in the class

Code: #include <iostream>

#include <list>

using namespace std;

class HashTable {

private:

    int rangeMin;

    int rangeMax;

    int capacity;

    list<int>\* table;

public:

    HashTable(int V);

    bool insert(int key);

    void deleteItem(int key);

    void rehash();

    bool search(int key);

    int checkPrime(int n);

    int getprime(int n);

    int hashFunction(int key);

    void displayHash();

};

HashTable::HashTable(int c) {

    int size = getprime(c);

    this->capacity = size;

    table = new list<int>[capacity];

}

bool HashTable::insert(int key) {

    if (key < rangeMin || key > rangeMax) {

        cout << "Key " << key << " is outside the range [" << rangeMin << ", " << rangeMax << "]." << endl;

        return false;

    }

    if (table[hashFunction(key)].size() >= 0.7 \* capacity) {

        rehash();

    }

    int index = hashFunction(key);

    list<int>& chain = table[index];

    for (int element : chain) {

        if (element == key) {

            cout << "Key " << key << " already exists in the table." << endl;

            return false;

        }

    }

    chain.push\_back(key);

    cout << "Key " << key << " inserted into the table." << endl;

    return true;

}

void HashTable::deleteItem(int key) {

    int index = hashFunction(key);

    list<int>& chain = table[index];

    list<int>::iterator iter = chain.begin();

    while (iter != chain.end()) {

        if (\*iter == key) {

            chain.erase(iter);

            cout << "Key " << key << " deleted from the table." << endl;

            return;

        }

        ++iter;

    }

    cout << "Key " << key << " not found in the table." << endl;

}

void HashTable::rehash() {

    int newCapacity = capacity \* 2;

    list<int>\* newTable = new list<int>[newCapacity];

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

        list<int>& chain = table[i];

        for (int element : chain) {

            int newIndex = hashFunction(element);

            newTable[newIndex].push\_back(element);

        }

    }

    delete[] table;

    table = newTable;

    capacity = newCapacity;

}

bool HashTable::search(int key) {

    int index = hashFunction(key);

    list<int>& chain = table[index];

    for (int element : chain) {

        if (element == key) {

            cout << "Key " << key << " found in the table." << endl;

            return true;

        }

    }

    cout << "Key " << key << " not found in the table." << endl;

    return false;

}

int HashTable::checkPrime(int n) {

    if (n <= 1) {

        return 0;

    }

    for (int i = 2; i \* i <= n; i++) {

        if (n % i == 0) {

            return 0;

        }

    }

    return 1;

}

int HashTable::getprime(int n) {

    if (n % 2 == 0) {

        n++;

    }

    while (!checkPrime(n)) {

        n += 2;

    }

    return n;

}

int HashTable::hashFunction(int key) {

    return (key % capacity);

}

void HashTable::displayHash() {

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

        cout << "table[" << i << "]";

        for (int element : table[i]) {

            cout << "-->" << element;

        }

        cout << endl;

    }

}

int main() {

    int key[] = {231, 321, 212, 321, 433, 262};

    int size = sizeof(key) / sizeof(key[0]);

    HashTable h(size);

    h.insert(123);

    h.insert(432);

    h.insert(523);

    h.insert(43);

    h.insert(423);

    h.insert(111);

    h.insert(456);

    h.insert(789);

    h.insert(321);

    h.insert(654);

    h.search(43);

    h.search(12);

    h.search(789);

    h.deleteItem(43);

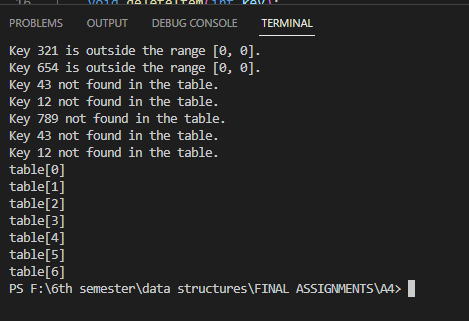
    h.deleteItem(12);

    h.displayHash();

    return 0;

}

Output:



The End\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_