

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ**

**KING MONGKUT’S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK**

**ASSIGNMENT 3 Hashing**

**เสนอ**

**อาจารย์ประดิษฐ์ พิทักษ์เสถียรกุล**

**จัดทำโดย**

**นายวรศิษฏ์ ภู่สุวรรณ์**

**ITI-2RB รหัส 6206021421237**

**รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm**

**(รหัสรายวิชา 060223119)**

**ภาคการศึกษาที่ 1 ปีการศึกษา 2563**

**สาขา เทคโนโลยีสารสนเทศ ภาควิชา เทคโนโลยีสารสนเทศ**

**คณะเทคโนโลยีและการจัดการอุตสาหกรรม**

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ วิทยาเขตปราจีนบุรี**

Code ไฟล์ hash.h

//////////////Header////////////////////

#include<iomanip>

#include<iostream>

#include<string>

#include<vector>

using namespace std;

template <class HashedObj>

class HashTable

{

public:

explicit HashTable(const HashedObj & notFound, int size = 101);

HashTable(const HashTable & rhs) : currentSize(rhs.currentSize), ITEM\_NOT\_FOUND(rhs.ITEM\_NOT\_FOUND), array(rhs.array) {}

const HashedObj & find(const HashedObj & x) const;

void makeEmpty();

void insert(const HashedObj & x);

void remove(const HashedObj & x);

void print();

void create(int size);

const HashedObj & findarray(const HashedObj & x) const;

const HashTable & operator=(const HashTable & rhs);

enum EntryType { ACTIVE, EMPTY, DELETED };

private:

struct HashEntry

{

HashedObj element;

EntryType info;

HashEntry(const HashedObj & e = HashedObj(), EntryType i = EMPTY) : element(e), info(i) {}

};

vector<HashEntry> array;

int currentSize;

const HashedObj ITEM\_NOT\_FOUND;

int nextPrime(int size);

int previousPrime(int size) const;

bool isActive(int currentPos) const;

int findPos(const HashedObj & x) const;

void rehash();

int hash(const string & key, int tableSize) const;

int hash(int key, int tableSize) const;

void print(int tableSize) const;

};

////////////////////////////////////create////////////////////////////////////

template <class HashedObj>

void HashTable<HashedObj>::create(int size)

{

array.resize(size);

makeEmpty();

}

//////////////////////////////////print//////////////////////////////////////////

template <class HashedObj>

void HashTable<HashedObj>::print()

{

print(array.size());

}

template <class HashedObj>

void HashTable<HashedObj>::print(int tableSize) const

{

cout << "\t------------------------\n\t" << setw(6) << "[ADDR][" << setw(7) << "ELEMENT][" << setw(9) << "INFO]\n";

for (int i = 0; i<tableSize; i++)

{

string info = array[i].info == 0 ? "ACTIVE" : array[i].info == 1 ? "EMPTY" : "DELETED";

cout << "\t------------------------\n\t" << "[" << setw(4) << i << "][" << setw(7) << array[i].element << "][" << setw(7) << info << "]\n";

}

cout << "\t------------------------\n";

}

//////////////////////////////////HashTable//////////////////////////////////////////

template <class HashedObj>

HashTable<HashedObj>::HashTable(const HashedObj & notFound, int size) : ITEM\_NOT\_FOUND(notFound), array(nextPrime(size))

{

makeEmpty();

}

//////////////////////////////////makeEmpty//////////////////////////////////////////

template<class HashedObj>

void HashTable<HashedObj>::makeEmpty()

{

currentSize = 0;

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

array[i].info = EMPTY;

}

//////////////////////////////////find//////////////////////////////////////////

template<class HashedObj>

const HashedObj & HashTable<HashedObj>::find(const HashedObj & x) const

{

int currentPos = findPos(x);

return isActive(currentPos) ? array[currentPos].element : ITEM\_NOT\_FOUND;

}

//////////////////////////////////findarray//////////////////////////////////////////

template<class HashedObj>

const HashedObj & HashTable<HashedObj>::findarray(const HashedObj & x) const

{

int currentPos = findPos(x);

int index = isActive(currentPos) ? currentPos : ITEM\_NOT\_FOUND;

if (index != ITEM\_NOT\_FOUND) cout << "\n\t!!! ADDRESS OF " << x << " IN ARRAY INDEX IS " << index << " !!!\n\t!!! AND ADDRESS OF MEMORY IS " << &array[index] << " !!!\n\n";

return index;

}

//////////////////////////////////findPos//////////////////////////////////////////

template<class HashedObj>

int HashTable<HashedObj> ::findPos(const HashedObj & x) const

{

int collisionNum = 0;

int currentPos = hash(x, array.size());

int R = previousPrime(array.size());

int temp = currentPos;

while (array[temp].info != EMPTY&&array[temp].element != x)

{

temp = (currentPos + (++collisionNum \* (R - (x % R)))) % array.size();

}

return temp;

}

//////////////////////////////////isActive//////////////////////////////////////////

template<class HashedObj>

bool HashTable<HashedObj>::isActive(int currentPos) const

{

return array[currentPos].info == ACTIVE;

}

//////////////////////////////////insert//////////////////////////////////////////

template<class HashedObj>

void HashTable<HashedObj>::insert(const HashedObj & x)

{

int currentPos = findPos(x);

if (isActive(currentPos))

return;

array[currentPos] = HashEntry(x, ACTIVE);

if (++currentSize>array.size()\*0.7)

rehash();

}

///////////////////////////remove//////////////////////////////////////////////

template<class HashedObj>

void HashTable<HashedObj>::remove(const HashedObj & x)

{

int currentPos = findPos(x);

if (isActive(currentPos))

array[currentPos].info = DELETED;

cout << "\t" << endl;

}

/////////////////////////////rehash///////////////////////////////////////

template <class HashedObj>

void HashTable<HashedObj>::rehash()

{

vector<HashEntry> oldArray = array;

array.resize(nextPrime(2 \* oldArray.size()));

for (int j = 0; j<array.size(); j++)

array[j].info = EMPTY;

currentSize = 0;

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

if (oldArray[i].info == ACTIVE)

insert(oldArray[i].element);

}

///////////////////////////////nextPrime///////////////////////////

template <class HashedObj>

int HashTable<HashedObj>::nextPrime(int size)

{

int i = 2;

while (size%i != 0) i++;

if (i == size) return i;

else{

i = size + 1;

bool check = false;

while (true){

for (int j = 2; j <= i; j++){

if (i%j == 0 && j<i) break;

else if (i%j == 0 && j == i) check = true;

}

if (check) break;

i++;

}

return i;

}

}

/////////////////////////////////previousPrime///////////////////////////

template <class HashedObj>

int HashTable<HashedObj>::previousPrime(int size) const

{

int R = size;

bool flag = true;

while (flag){

R--;

if (R % 2 != 0){

for (int i = R - 1; i >= 2; i--){

if (R%i == 0){

break;

}

if (i == 2 && R%i != 0){

flag = false;

}

}

}

}

return R;

}

///////////////////////////////hash/////////////////////////////

template<class HashedObj>

int HashTable<HashedObj>::hash(const string & key, int tableSize) const

{

int hashVal = 0;

for (int i = 0; i<key.length(); i++)

hashVal += key[i];

return hashVal % tableSize;

}

template<class HashedObj>

int HashTable<HashedObj>::hash(int key, int tableSize) const

{

if (key < 0) key = -key;

return key % tableSize;

}

Code ไฟล์ Hash.cpp

#include "hash.h"

#include<iostream>

#include<string>

int nextPrime(int size);

using namespace std;

int main(){

HashTable<int> a(-999);

string choice;

bool exit = true;

bool create = false;

do{

system("cls");

cout << "\t##### Assignment Double Hashing #####\n\n";

cout << "\t1.) CREATE HASH TABLE\n\t2.) INSERT\n\t3.) PRINT\n\t4.) REMOVE\n\t5.) FIND\n\t6.) QUIT\n\n\tEnter Choice : ";

cin >> choice;

switch (choice[0])

{

int x;

case '1':

system("cls");

if (!create){

int cell;

cout << "\t!!!!! CREATE MODE !!!!!\n\n\tEnter TableSize : ";

cin >> cell;

if(cell>30){

cout << endl;

cout<<"\t\*\*You can't create\*\*";

cout << endl;

cout<<"\t\*\*Please Try Again\*\*";

cout << endl;

}

else {

cout << "\n\tNEXT PRIME OF " << cell << " IS " << nextPrime(cell) << "\n\tSIZE OF HASH TABLE IS " << nextPrime(cell) << endl << endl;

a.create(nextPrime(cell));

create = !create;

}

system("pause");

}

else

{

cout << "\n!!!CREATED ALL MODE AVAILABLE!!!\n\n";

system("pause");

}

break;

case '2':

system("cls");

if (create)

{

cout << "\t!!!!! INSERT MODE !!!!!\n\n\tEnter Element to Insert : ";

cin >> x;

if (a.find(x) == -999){

a.insert(x);

cout << "\n\t!!! INSERT ELEMENT SUCCESSFULL !!!\n";

}

else cout << "\n\t!!! HASH TABLE HAVE SAME ELEMEMT PLEASE TRY AGAIN !!!\n";

cout << endl;

system("pause");

}

else

{

cout << "\n!!!MUST CREATE TABLE BEFORE INSERT ELEMENT!!!\n\n";

system("pause");

}

break;

case '3':

system("cls");

if (create)

{

cout << "\t!!!!! PRINT MODE !!!!!\n\n";

a.print();

cout << endl;

system("pause");

}

else

{

cout << "\n!!!MUST CREATE TABLE BEFORE PRINT HASH TABLE!!!\n\n";

system("pause");

}

break;

case '4':

system("cls");

if (create)

{

cout << "\t!!!!! REMOVE MODE !!!!!\n\n\tEnter Element to Remove : ";

cin >> x;

if (a.find(x) != -999){

a.remove(x);

cout << "\n\t!!! REMOVE ELEMENT SUCCESSFULL !!!\n";

}

else cout << "\n\t!!! HASH TABLE HAVEN'T ELEMEMT PLEASE TRY AGAIN !!!\n";

cout << endl;

system("pause");

}

else

{

cout << "\n!!!MUST CREATE TABLE BEFORE REMOVE ELEMENT!!!\n\n";

system("pause");

}

break;

case '5':

system("cls");

if (create)

{

cout << "\t!!!!! FIND MODE !!!!!\n\n\tEnter Element to Find : ";

cin >> x;

if (a.findarray(x) == -999)

cout << "\n\t!!! FIND ELEMENT NOT FOUND !!!\n";

cout << endl;

system("pause");

}

else

{

cout << "\n!!!MUST CREATE TABLE BEFORE FIND ELEMENT!!!\n\n";

system("pause");

}

break;

case '6':

exit = false;

break;

default: break;

}

} while (exit);

return 0;

}

int nextPrime(int size)

{

int i = 2;

if (size == 1) return 1;

while (size%i != 0) i++;

if (i == size) return i;

else{

i = size + 1;

bool check = false;

while (true){

for (int j = 2; j <= i; j++){

if (i%j == 0 && j<i) break;

else if (i%j == 0 && j == i) check = true;

}

if (check) break;

i++;

}

return i;

}

}