МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра системного проектування

**ЗВІТ**про виконання лабораторної роботи №2

«Дослідження структури даних хеш-таблиця»

з дисципліни «Алгоритми та структури даних»

Виконав:

cтудент І курсу

групи ДА-02

Гринчишин Д.Б.

Перевірили:

..

Київ – 2020

**Мета**. Ознайомитись і дослідити структуру даних хеш-таблиця. Набути навичок реалізації хеш-таблиці за методом ланцюжків мовою програмування C++, познайомитись з використанням STL контейнерів на прикладі unordered\_map та порівняти власну реалізацію з готовим бібліотечним рішенням.

**Варіант завдань – 2**

Код завдань 1 - 4:

common.h :

#include<iostream>

#include <ctime>

#include <cmath>

#include <string>

#include <cstdlib>

#include <unordered\_map>

using namespace std;

long long generateRandLong(int len = 12) {

long long num = (rand() % 9 + 1) \* pow(10, len - 1);

for (long long rank = pow(10, len - 2); rank != 0; rank /= 10) num += rand() % 10 \* rank;

return num;

}

struct Data{

int spec;

int year;

float mean;

Data(){

spec = rand() % 100;

year = rand() % 2021;

mean = (float)rand() / 32767 \* 100;

}

};

string generateRandString(int len = 12) {

string key(len, (char)0);

for (size\_t i = 0; i < len; i++) key[i] = (rand() % 1000);

return key;

}

template<typename table> bool testHashTable(){

const int iters = 500000;

const int keysAmount = iters \* 1;

// generate random keys:

long long\* keys = new long long[keysAmount];

long long\* keysToInsert = new long long[iters];

long long\* keysToErase = new long long[iters];

long long\* keysToFind = new long long[iters];

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

{

keys[i] = generateRandLong();

}

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

{

keysToInsert[i] = keys[generateRandLong() % keysAmount];

keysToErase[i] = keys[generateRandLong() % keysAmount];

keysToFind[i] = keys[generateRandLong() % keysAmount];

}

// test my HashTable:

table hashTable;

clock\_t myStart = clock();

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

{

hashTable.insert(keysToInsert[i], Data());

}

int myInsertSize = hashTable.size();

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

{

hashTable.erase(keysToErase[i]);

}

int myEraseSize = hashTable.size();

int myFoundAmount = 0;

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

{

if (hashTable.find(keysToFind[i]) != NULL)

{

myFoundAmount++;

}

}

clock\_t myEnd = clock();

float myTime = (float(myEnd - myStart)) / CLOCKS\_PER\_SEC;

// test STL hash table:

unordered\_map<long long, Data> unorderedMap;

clock\_t stlStart = clock();

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

{

unorderedMap.insert({keysToInsert[i], Data()});

}

int stlInsertSize = unorderedMap.size();

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

{

unorderedMap.erase(keysToErase[i]);

}

int stlEraseSize = unorderedMap.size();

int stlFoundAmount = 0;

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

{

if (unorderedMap.find(keysToFind[i]) != unorderedMap.end())

{

stlFoundAmount++;

}

}

clock\_t stlEnd = clock();

float stlTime = (float(stlEnd - stlStart)) / CLOCKS\_PER\_SEC;

cout << "My HashTable:" << endl;

cout << "Time: " << myTime << ", size: " << myInsertSize << " - " << myEraseSize << ", found amount: " << myFoundAmount << endl;

cout << "STL unordered\_map:" << endl;

cout << "Time: " << stlTime << ", size: " << stlInsertSize << " - " << stlEraseSize << ", found amount: " << stlFoundAmount << endl << endl;

delete [] keys;

delete [] keysToInsert;

delete [] keysToErase;

delete [] keysToFind;

if (myInsertSize == stlInsertSize && myEraseSize == stlEraseSize && myFoundAmount == stlFoundAmount)

{

cout << "The lab is completed" << endl;

return true;

}

cerr << ":(" << endl;

return false;

}

template<typename table> bool testHashTable\_s(){

const int iters = 500000;

const int keysAmount = iters \* 1;

// generate random keys:

string\* keys = new string[keysAmount];

string\* keysToInsert = new string[iters];

string\* keysToErase = new string[iters];

string\* keysToFind = new string[iters];

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

{

keys[i] = generateRandString();

}

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

{

keysToInsert[i] = keys[generateRandLong() % keysAmount];

keysToErase[i] = keys[generateRandLong() % keysAmount];

keysToFind[i] = keys[generateRandLong() % keysAmount];

}

// test my HashTable:

table hashTable;

clock\_t myStart = clock();

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

{

hashTable.insert(keysToInsert[i], Data());

}

int myInsertSize = hashTable.size();

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

{

hashTable.erase(keysToErase[i]);

}

int myEraseSize = hashTable.size();

int myFoundAmount = 0;

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

{

if (hashTable.find(keysToFind[i]) != NULL)

{

myFoundAmount++;

}

}

clock\_t myEnd = clock();

float myTime = (float(myEnd - myStart)) / CLOCKS\_PER\_SEC;

// test STL hash table:

unordered\_map<string, Data> unorderedMap;

clock\_t stlStart = clock();

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

{

unorderedMap.insert({keysToInsert[i], Data()});

}

int stlInsertSize = unorderedMap.size();

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

{

unorderedMap.erase(keysToErase[i]);

}

int stlEraseSize = unorderedMap.size();

int stlFoundAmount = 0;

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

{

if (unorderedMap.find(keysToFind[i]) != unorderedMap.end())

{

stlFoundAmount++;

}

}

clock\_t stlEnd = clock();

float stlTime = (float(stlEnd - stlStart)) / CLOCKS\_PER\_SEC;

cout << "My HashTable:" << endl;

cout << "Time: " << myTime << ", size: " << myInsertSize << " - " << myEraseSize << ", found amount: " << myFoundAmount << endl;

cout << "STL unordered\_map:" << endl;

cout << "Time: " << stlTime << ", size: " << stlInsertSize << " - " << stlEraseSize << ", found amount: " << stlFoundAmount << endl << endl;

delete [] keys;

delete [] keysToInsert;

delete [] keysToErase;

delete [] keysToFind;

if (myInsertSize == stlInsertSize && myEraseSize == stlEraseSize && myFoundAmount == stlFoundAmount)

{

cout << "The lab is completed" << endl;

return true;

}

cerr << ":(" << endl;

return false;

}

chain\_long\_long.cpp :

#include <iostream>

#include <ctime>

#include <cmath>

#include <string>

#include <cstdlib>

#include <unordered\_map>

#include "common.h"

using namespace std;

struct HashNode{ long long key; Data diplom; };

struct Node { HashNode data; Node\* next = NULL; Node\* prev = NULL; };

struct LinkedList {

Node\* head = NULL;

int size = 0;

bool insert(HashNode new\_elem) {

Node\* node = get\_by\_key(new\_elem.key);

if (node == NULL) {

head = new Node{new\_elem, head, NULL};

if (size > 0) head->next->prev = head;

size++;

return true;

}

else \*node = {new\_elem, node->next, node->prev};

return false;

}

bool pop(long long key) {

bool deleted = false;

if(size > 0) {

for (Node\* node = head; node != NULL;node = node->next) {

if(node->data.key == key){

if(node->prev != NULL) node->prev->next = node->next;

else head = node->next;

if(node->next != NULL) node->next->prev = node->prev;

delete node;

deleted = true;

break;

}

}

if (deleted){

size--;

if(size == 0) head = NULL;

}

}

return deleted;

}

Node\* get\_by\_key(long long key) {

for(Node\* node = head; node != NULL; node = node->next){

if(node->data.key == key) return node;

}

return NULL;

}

};

struct HashTable {

int M = 8;

int Size = 0;

float loadFactor = 8.5;

LinkedList\* bucketsArray = new LinkedList[M];

int hash(long long key){ return key % M;}//(((rand() % 100) \* key + rand() % 100) % 9149658775000477) % M; }

int size(){ return Size; }

void erase(long long key) { Size -= bucketsArray[hash(key)].pop(key) ? 1 : 0; }

void insert(long long key, Data value) {

if((float)(Size + 1) / M >= loadFactor) {

M \*= 2;

LinkedList\* newBucketsArray = new LinkedList[M];

for (size\_t i = 0; i < M / 2; i++) {

for (Node\* node = bucketsArray[i].head; node != NULL; node = node->next)

newBucketsArray[hash(node->data.key)].insert(node->data);

}

delete [] bucketsArray;

bucketsArray = newBucketsArray;

}

Size += bucketsArray[hash(key)].insert(HashNode{key, value}) ? 1 : 0;

}

Data\* find(long long key) {

Node\* linkedListNode = bucketsArray[hash(key)].get\_by\_key(key);

if (linkedListNode != NULL) return &linkedListNode->data.diplom;

return NULL;

}

}hashTable;

int main(int argc, char\*\* argv){

srand(time(0));

testHashTable<HashTable>();

return 0;

}

Дод. Завдання 1:

chain\_string.cpp :

#include <iostream>

#include <ctime>

#include <cmath>

#include <string>

#include <cstdlib>

#include <unordered\_map>

#include <Windows.h>

#include "common.h"

using namespace std;

struct HashNode{ string key; Data diplom; };

struct Node { HashNode data; Node\* next = NULL; Node\* prev = NULL; };

struct LinkedList {

Node\* head = NULL;

int size = 0;

bool insert(HashNode new\_elem) {

Node\* node = get\_by\_key(new\_elem.key);

if (node == NULL) {

head = new Node{new\_elem, head, NULL};

if (size > 0) head->next->prev = head;

size++;

return true;

}

else \*node = {new\_elem, node->next, node->prev};

return false;

}

bool pop(string key) {

bool deleted = false;

if(size > 0) {

for (Node\* node = head; node != NULL; node = node->next) {

if(node->data.key.compare(key) == 0){

if(node->prev != NULL) node->prev->next = node->next;

else head = node->next;

if(node->next != NULL) node->next->prev = node->prev;

delete node;

deleted = true;

break;

}

}

if (deleted){

size--;

if(size == 0) head = NULL;

}

}

return deleted;

}

Node\* get\_by\_key(string key) {

for(Node\* node = head; node != NULL; node = node->next){

if(node->data.key.compare(key) == 0) return node;

}

return NULL;

}

};

struct HashTable {

int M = 8;

int Size = 0;

float loadFactor = 8.5;

LinkedList\* bucketsArray = new LinkedList[M];

int hash(string key) {

unsigned int res = 37;

for(int i = 0; i < key.length(); i++) res = 54059 \* res ^ ((int)key[i] \* 76963);

return res % M;

}

int size(){ return Size; }

void erase(string key) { Size -= bucketsArray[hash(key)].pop(key) ? 1 : 0; }

void insert(string key, Data value) {

if((float)(Size + 1) / M >= loadFactor) {

M \*= 2;

LinkedList\* newBucketsArray = new LinkedList[M];

for (size\_t i = 0; i < M / 2; i++) {

for (Node\* node = bucketsArray[i].head; node != NULL; node = node->next)

newBucketsArray[hash(node->data.key)].insert(node->data);

}

delete [] bucketsArray;

bucketsArray = newBucketsArray;

}

Size += bucketsArray[hash(key)].insert(HashNode{key, value}) ? 1 : 0;

}

Data\* find(string key) {

Node\* linkedListNode = bucketsArray[hash(key)].get\_by\_key(key);

if (linkedListNode != NULL) return &linkedListNode->data.diplom;

return NULL;

}

}hashTable;

int main(int argc, char\*\* argv){

srand(time(0));

testHashTable\_s<HashTable>();

return 0;

}

Дод. Завдання 2 :

adresation.cpp :

#include <iostream>

#include <ctime>

#include <cmath>

#include <string>

#include <cstdlib>

#include <unordered\_map>

#include "common.h"

using namespace std;

struct HashNode{bool deleted = true; bool deleted\_for\_search = true; long long key = 0; Data diplom;};

struct HashTable {

int M = 8;

int Size = 0;

HashNode\* bucketsArray = new HashNode[M];

float loadFactor = 1;

int hash(long long key){ return key % M;}//(((rand() % 100) \* key + rand() % 100) % 9149658775000477) % M; }

int size(){ return Size; }

void erase(long long key) {

int index, iter = 0;

for(index = hash(key); iter < M && !bucketsArray[index].deleted\_for\_search; index = (index + 1) % M){

if(bucketsArray[index].key == key && !bucketsArray[index].deleted){

bucketsArray[index].deleted = true;

Size --;

break;

}

iter ++ ;

}

}

void insert(long long key, Data value) {

int index;

if((float)Size / M >= loadFactor) {

M \*= 2;

HashNode\* newBucketsArray = new HashNode[M];

for (size\_t i = 0; i < M / 2; i++) {

if(!bucketsArray[i].deleted){

for(index = hash(bucketsArray[i].key); !newBucketsArray[index].deleted; index = (index + 1) % M){}

newBucketsArray[index] = bucketsArray[i];

}

}

delete [] bucketsArray;

bucketsArray = newBucketsArray;

}

bool temp = false;

for(index = hash(key); !bucketsArray[index].deleted; index = (index + 1) % M){

temp = bucketsArray[index].key == key;

if(temp) break;

}

bucketsArray[index] = {false, false, key, value};

Size += temp ? 0 : 1;

}

Data\* find(long long key) {

int index, iter = 0;

for(index = hash(key); iter < M && !bucketsArray[index].deleted\_for\_search; index = (index + 1) % M){

if(bucketsArray[index].key == key && !bucketsArray[index].deleted) return &bucketsArray[index].diplom;

iter ++ ;

}

return NULL;

}

}hashTable;

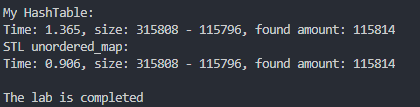
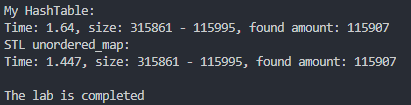
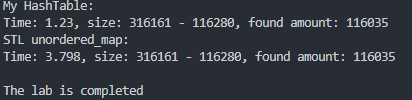
int main(int argc, char\*\* argv){

srand(time(0));

testHashTable<HashTable>();

return 0;

}

**Результат виконання програми:  
  **

**Висновок:**

В ході виконання лабораторної роботи було реалізовано структури даних хеш таблиця на методі ланцюжків і з чисельними та рядковими ключами, хеш-таблиця на методі відкритої адресації і всі необхідні для роботи з ними функції. Було заміряно час виконання цих алгоритмів та порівняно з готовим рішенням STL. Моя реалізація в середньому працює швидше за вбудовану, оскільки є менш універсальною.