***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №1**

**По дисциплине АОИС за III семестр**

**Тема: «Применение хэширование для ассоциативного поиска по ключам»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Михно Е. В.

Брест 2018

**Цель работы:** Изучить основные принципы хэширования, построить программную модель, обеспечивающую формирование хэш-таблиц и ассоциативный поиск по ключам.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | XOR без сдвига + метод квадрата | метод цепочек | 20 | 16 |

Код программы:

#include "stdafx.h"

#include <Windows.h>

#include <conio.h>

#include <string>

#include <iostream>

#include <iomanip>

using namespace std;

class Cell

{

private:

int empty;

string key;

string firstName;

string lastName;

Cell \* next;

public:

Cell();

Cell(string key, string firstName, string lastName);

Cell(string key);

Cell(Cell& obj);

string getKey();

string getFName();

string getLName();

Cell\* getNext();

void setKey(string newKey);

void setFName(string newFName);

void setLName(string newLName);

void setNext(Cell\* newNext);

void showInfo();

bool isEmpty();

};

class HashTable

{

private:

int numOfCells;

Cell \* cArr; // array of cells

int toInt(string key); // key to int

int toHashCode(int key); // key to hash code

void addToBucket(Cell c, int bucket); // to add the cell to the bucket

Cell \* getFromBucket(string key, int bucket); // to get the cell from the bucket

public:

HashTable(int numOfCells);

Cell operator[](string key); // return the cell by given id

void add(Cell c);

void add(string key, string firstName, string lastName);

friend void showHashTable(HashTable ht);

};

void showHashTable(HashTable ht);

int main()

{

const int n = 20;

HashTable ht(16);

Cell test[n];

ht.add(\*(new Cell("qweras", "1", "")));

ht.add(\*(new Cell("tyuidf", "2", "")));

ht.add(\*(new Cell("op[bv]", "3", "")));

ht.add(\*(new Cell("asdfdsagf", "4", "")));

ht.add(\*(new Cell("ghcak", "5", "")));

ht.add(\*(new Cell(";'dfal", "6", "")));

ht.add(\*(new Cell("zxcacv", "7", "")));

ht.add(\*(new Cell("bncgam,", "8", "")));

ht.add(\*(new Cell("./ddswq`", "9", "")));

ht.add(\*(new Cell("qadfaz", "10", "")));

ht.add(\*(new Cell("wsvbvx", "11", "")));

ht.add(\*(new Cell("eddhvc", "12", "")));

ht.add(\*(new Cell("rfasjv", "13", "")));

ht.add(\*(new Cell("tgsdwb", "14", "")));

ht.add(\*(new Cell("yhnbdh", "15", "")));

ht.add(\*(new Cell("ujsajm", "16", "")));

ht.add(\*(new Cell("iksdgr,", "17", "")));

ht.add(\*(new Cell("ohbadgjl.", "18", "")));

ht.add(\*(new Cell("pwwfd;/", "19", "")));

ht.add(\*(new Cell("['a151", "20", "")));

showHashTable(ht);

system("pause");

return 0;

}

void showHashTable(HashTable ht)

{

int cw = 12;

cout << setw(cw) << "Hash" << setw(cw) << "Key"

<< setw(cw) << "First name" << setw(cw) << "Last name" << endl;

for (int i = 0; i < ht.numOfCells; i++)

{

Cell \* cur = ht.cArr + i;

for (Cell \* cur = &ht.cArr[i]; cur != NULL; cur = cur->getNext())

{

cout << setw(cw) << i << setw(cw) << cur->getKey()

<< setw(cw) << cur->getFName().c\_str() << setw(cw)

<< cur->getLName().c\_str() << endl;

}

}

}

Cell::Cell()

{

this->empty = true;

this->setNext(NULL);

}

Cell::Cell(string key, string firstName, string lastName)

{

this->empty = false;

this->setKey(key);

this->setFName(firstName);

this->setLName(lastName);

this->setNext(NULL);

}

Cell::Cell(string key)

{

this->empty = false;

this->setKey(key);

this->setFName("empty");

this->setLName("empty");

this->setNext(NULL);

}

Cell::Cell(Cell& obj)

{

this->empty = false;

this->setKey(obj.getKey());

this->setFName(obj.getFName());

this->setLName(obj.getLName());

this->setNext(obj.getNext());

}

string Cell::getKey()

{

return key;

}

string Cell::getFName()

{

return firstName;

}

string Cell::getLName()

{

return lastName;

}

Cell\* Cell::getNext()

{

return next;

}

void Cell::setKey(string newKey)

{

this->empty = false;

this->key = newKey;

}

void Cell::setFName(string newFName)

{

this->empty = false;

this->firstName = newFName;

}

void Cell::setLName(string newLName)

{

this->empty = false;

this->lastName = newLName;

}

void Cell::setNext(Cell\* newNext)

{

this->next = newNext;

}

void Cell::showInfo()

{

int cw = 13;

cout << setw(cw) << this->getKey().c\_str() << setw(cw) << this->getFName().c\_str()

<< setw(cw) << this->getLName().c\_str() << endl;

}

bool Cell::isEmpty()

{

return this->empty;

}

HashTable::HashTable(int numOfCells)

{

this->numOfCells = numOfCells;

this->cArr = new Cell[numOfCells];

}

Cell HashTable::operator[](string key)

{

int intKey = this->toInt(key);

int hashCode = this->toHashCode(intKey);

if (this->getFromBucket(key, hashCode) == NULL)

{

cout << "There is no such a cell!" << endl;

Cell empty;

return empty;

}

return \*(this->getFromBucket(key, hashCode));

}

void HashTable::add(Cell c)

{

int intKey = this->toInt(c.getKey());

int hashCode = this->toHashCode(intKey);

this->addToBucket(c, hashCode);

}

void HashTable::add(string key, string firstName, string lastName)

{

Cell c(key, firstName, lastName);

this->add(c);

}

int HashTable::toInt(string key)

{

int num = 0;

int size = key.size();

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

{

num ^= (int)((key.c\_str())[i]);

}

return num;

}

int HashTable::toHashCode(int intKey)

{

int sqKey = intKey \* intKey;

int lengthOfHash = 0;

int lengthOfSqKey = 0;

int temp = 0;

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

{

temp += (int)pow(2.0, (double)i);

if (i >= lengthOfHash)

break;

}

while (this->numOfCells / temp)

{

lengthOfHash++;

temp = 0;

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

{

temp += (int)pow(2.0, (double)i);

if (i >= lengthOfHash)

break;

}

}

temp = 0;

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

{

temp += (int)pow(2.0, (double)i);

if (i >= lengthOfSqKey)

break;

}

while (sqKey / temp)

{

lengthOfSqKey++;

temp = 0;

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

{

temp += (int)pow(2.0, (double)i);

if (i >= lengthOfSqKey)

break;

}

}

int indent;

if (lengthOfSqKey > lengthOfHash)

indent = (lengthOfSqKey - lengthOfHash) / 2;

else

indent = 0;

sqKey >>= indent;

temp = 0;

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

{

temp += (int)pow(2.0, (double)i);

}

sqKey &= temp;

return sqKey;

}

void HashTable::addToBucket(Cell c, int bucket)

{

Cell \* cur = &(this->cArr[bucket]);

if (cur->isEmpty())

{

this->cArr[bucket] = c;

return;

}

while (cur->getNext() != NULL)

cur = cur->getNext();

Cell \*newC = new Cell;

\*newC = c;

cur->setNext(newC);

}

Cell \* HashTable::getFromBucket(string key, int bucket)

{

Cell \* cur = &(this->cArr[bucket]);

try

{

if (cur == NULL)

{

throw 0;

}

while (strcmp(cur->getKey().c\_str(), key.c\_str()))

{

cur = cur->getNext();

if (cur == NULL)

{

throw 0;

}

}

return cur;

}

catch (...)

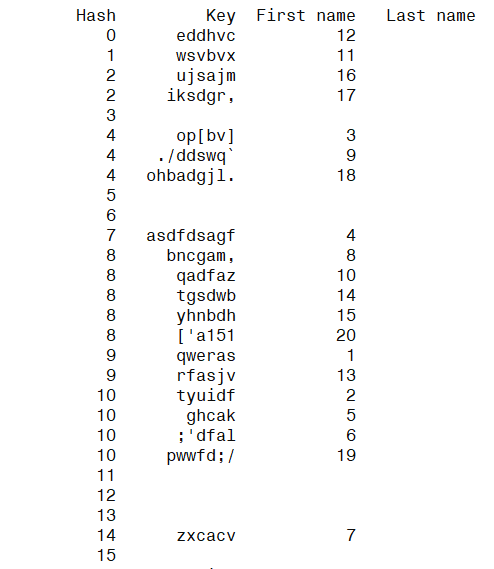
{

return NULL;

}

}

Результат выполнения:



Вывод: Изучили основные принципы хэширования, построили программную модель, обеспечивающую формирование хэш-таблиц и ассоциативный поиск по ключам.