Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 13

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы c открытой адресацией»

Выполнил:

Кулешов Артём

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

4. В соответствии со своим вариантом построить ***хеш-таблицы*** с ***открытой*** адресацией разного размера, например, 16, 32 или 32, 64, 128 с коллизиями. В таблице **h**'**(key)** −значение хеш-функции, приведшее к коллизии.

Исследовать время поиска в хеш-таблицах. В приложении Excel построить соответствующие графики.

Вариант №5

Hash.h

#pragma once

#define HASHDEL (void\*) -1

struct Object

{

void\*\* data;

Object(int, int(\*)(void\*));

int size; int N;

int(\*getKey)(void\*);

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, int(\*getkey)(void\*));

#undef HASHDEL

Hash.cpp

#include "Hash.h"

#include <iostream>

int HashFunction(int key, int size, int p) //Хеш-функция

{

double key2 = 5 \* ((0.6180339887499 \* key) - int((0.6180339887499 \* key)));

return (p + key) % size;

}

//-------------------------------

int Next\_hash(int hash, int size, int p)

{

return (hash + 5 \* p + 3 \* p \* p) % size;

}

//-------------------------------

Object create(int size, int(\*getkey)(void\*))

{

return \*(new Object(size, getkey));

}

//-------------------------------

Object::Object(int size, int(\*getkey)(void\*))

{

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

//-------------------------------

bool Object::insert(void\* d)

{

bool b = false; // Флаг, указывающий, был ли элемент успешно вставлен

if (N != size) { // Проверка, что таблица не заполнена

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0);

i != size && !b; i++) { // Цикл по всем слотам таблицы

j = (j + 1) % size; // Переходим к следующему слоту (линейный алгоритм)

if (data[j] == NULL || data[j] == DEL) { // Если слот пуст или удален

data[j] = d; // Вставляем элемент

N++; // Увеличиваем количество элементов в таблице

b = true; // Устанавливаем флаг успешной вставки

}

}

}

return b; // Возвращаем успешность вставки

}

//-------------------------------

int Object::searchInd(int key)

{

int t = -1; // Индекс элемента, если найден, иначе -1

bool b = false; // Флаг, указывающий, найден ли элемент

if (N != 0) { // Проверка, что таблица не пуста

for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i != size && !b; i++) {

// Цикл по всем слотам таблицы для поиска элемента

if (getKey(data[j]) == key) { // Если ключ элемента совпадает с искомым

t = j; // Сохраняем индекс элемента

b = true; // Устанавливаем флаг успешного поиска

}

j = (j + 1) % size; // Переходим к следующему слоту (линейный алгоритм)

}

}

return t; // Возвращаем индекс найденного элемента или -1

}

//-------------------------------

void\* Object::search(int key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL);

}

//-------------------------------

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//-------------------------------

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL);

}

//-------------------------------

void Object::scan(void(\*f)(void\*))

{

for (int i = 0; i < this->size; i++)

{

std::cout << " Элемент: " << i;

if ((this->data)[i] == NULL)

std::cout << " пусто" << std::endl;

else

if ((this->data)[i] == DEL)

std::cout << " удален" << std::endl;

else

f((this->data)[i]);

}

}

Main.cpp

#include "Hash.h"

#include <iostream>

#include <chrono>

using namespace std;

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k; mas = z;

} AAA() {}

};

//-------------------------------

int key(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

//-------------------------------

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;

}

//-------------------------------

int main()

{

setlocale(LC\_ALL, "rus");

int siz = 10, choice, k;

cout << "Введите размер хеш-таблицы: ";

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl; cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: H.scan(AAA\_print); break;

case 2: { AAA\* a = new AAA;

char\* str = new char[20];

cout << "введите ключ" << endl;

cin >> k;

a->key = k;

cout << "введите строку" << endl;

cin >> str;

a->mas = str;

if (H.N == H.size)

cout << "Таблица заполнена" << endl;

else

H.insert(a);

} break;

case 3: { cout << "введите ключ для удаления" << endl;

cin >> k;

H.deleteByKey(k);

} break;

case 4: {

double time1, time2;

time1 = clock();

cout << "введите ключ для поиска" << endl;

cin >> k;

if (H.search(k) == NULL)

cout << "Элемент не найден" << endl;

else

AAA\_print(H.search(k));

time2 = clock();

cout << endl << "Время поиска:" << (time2 - time1) / 1000 << endl;

} break;

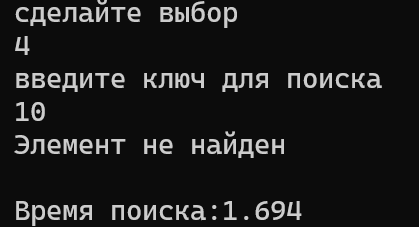
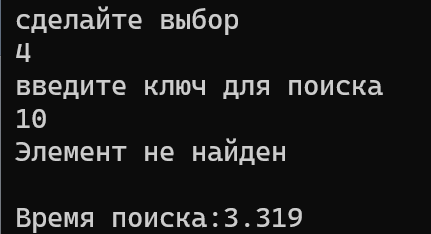
}

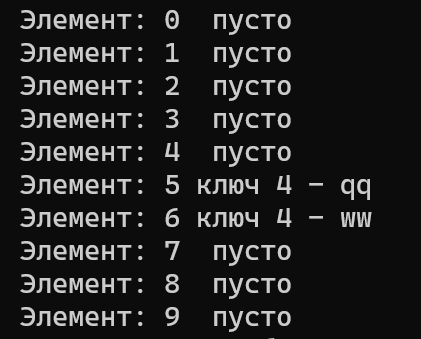
}

return 0;

}

Время поиска (размер 16): Время поиска (размер 128):



Вариант №3

Изменить функцию вычисления хеш на мультипликативную функцию, которая строится на основе формулы: **H(key) = [hashTableSize(key ∙ A mod 1)]**, где **A = (sqrt(5) - 1) / 2 = 0,6180339887499**.

Вариант №8

Изменить проект с тем, чтобы использовался аддитивный метод хеширования (ключи должны быть строковыми данными).

Вариант №15

Изменить функцию вычисления хеш для решения коллизии на функцию, которая строится на основе формулы: **h(key, i) = (h1(key) + i∙h2(key))⋅mod hashTableSize**, где **h1 (key) = key⋅mod hashTableSize** , **h2(key) = 1 + (key⋅mod hashTableSize )**.

Hash.h

#pragma once

#define HASHDEL (void\*) -1

struct Object

{

void\*\* data;

Object(int, char\*(\*)(void\*));

int size;  int N;

char\*(\*getKey)(void\*);

bool insert(void\*);

int searchInd(char\* key);

void\* search(char\* key);

void\* deleteByKey(char\* key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, char\*(\*getkey)(void\*));

#undef HASHDEL

Hash.cpp

#include "Hash.h"

#include <iostream>

#include <string>

int HashFunction(char\* key, int size, int p)    //Хеш-функция

{

int k = atoi(key);//Вариант 8

//return size\*( (0.6180339887499 \* k) - (int)(0.6180339887499 \* k) ); //Вариант 3

//return (p + k) % size;//Вариант 3

return (k%size) +p\*(1+k%size);//Вариант 15

}

//-------------------------------

int Next\_hash(int hash, int size, int p)

{

return (hash + 5 \* p + 3 \* p \* p) % size;

}

//-------------------------------

Object create(int size, char\*(\*getkey)(void\*))

{

return \*(new Object(size, getkey));

}

//-------------------------------

Object::Object(int size, char\*(\*getkey)(void\*))

{

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

//-------------------------------

bool Object::insert(void\* d)

{

bool b = false;

int i=0, j=0;

char\* t;

if (N != size)

for (i = 0,t=getKey(d), j = HashFunction(t, size, 0);i != size && !b; j = Next\_hash(j, size, ++i))

if (data[j] == NULL || data[j] == DEL)

{

data[j] = d;

N++;

b = true;

}

return b;

}

//-------------------------------

int Object::searchInd(char\* key)

{

int t = -1;

bool b = false;

if (N != 0)

for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i != size && !b; j = HashFunction(key, size, ++i))

if (data[j] != DEL)

if (getKey(data[j]) == key)

{

t = j; b = true;

}

return t;

}

//-------------------------------

void\* Object::search(char\* key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL);

}

//-------------------------------

void\* Object::deleteByKey(char\* key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//-------------------------------

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL);

}

//-------------------------------

void Object::scan(void(\*f)(void\*))

{

for (int i = 0; i < this->size; i++)

{

std::cout << " Элемент" << i;

if ((this->data)[i] == NULL)

std::cout << "  пусто" << std::endl;

else

if ((this->data)[i] == DEL)

std::cout << "  удален" << std::endl;

else

f((this->data)[i]);

}

}

Main.cpp

#include "Hash.h"

#include <iostream>

#include <chrono>

using namespace std;

struct AAA

{

char\* key;

//int key;

char\* mas;

AAA(char\* k, char\* z)

{

key = k;  mas = z;

} AAA() {}

};

//-------------------------------

char\* key(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

//-------------------------------

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;

}

//-------------------------------

int main()

{

      setlocale(LC\_ALL, "rus");

int siz = 10, choice;

cout << "Введите размер хеш-таблицы: ";

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl;   cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: H.scan(AAA\_print);  break;

case 2: {

AAA\* a = new AAA;

char\* str = new char[20];

char\* k = new char[20];

cout << "введите ключ" << endl;

cin >> k;

cout << "введите строку" << endl;

cin >> str;

a->key = k;

a->mas =str;

if (H.N == H.size)

cout << "Таблица заполнена" << endl;

else H.insert(a);

} break;

case 3: {  cout << "введите ключ для удаления" << endl;

char\* k = new char[20];

cin >> k;

H.deleteByKey(k);

}  break;

case 4: {

char\* k = new char[20];

double time1,time2;

time1 = clock();

cout << "введите ключ для поиска" << endl;

cin >> k;

if (H.search(k) == NULL)

cout << "Элемент не найден" << endl;

else

AAA\_print(H.search(k));

time2 = clock();

cout << endl << "Time of search:" << (time2 - time1) / 1000<<endl;

}  break;

}

}

return 0;

}

