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»

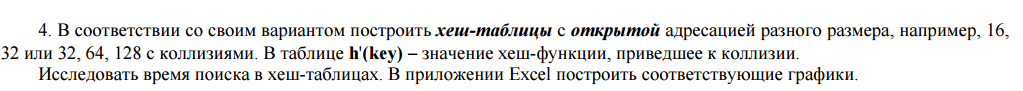
Выполнил:

Студент 1 курса 6 группы

Романов Игорь Вячеславович

Преподаватель: асс. Андронова М.В.

2024, Минск





**Hash.h**

using namespace std;

#define HASHDEL (void\*) -1 // Макрос для определения значения удаленного элемента

// Структура объекта

struct Object {

void\*\* data; // Указатель на массив указателей на данные

int size; // Размер массива

int N; // Количество элементов в массиве

int(\*getKey)(void\*); // Указатель на функцию получения ключа из данных

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

bool insert(void\*); // Функция вставки элемента в массив

int searchInd(int key); // Функция поиска индекса элемента по ключу

void\* search(int key); // Функция поиска элемента по ключу

void\* deleteByKey(int key); // Функция удаления элемента по ключу

bool deleteByValue(void\*); // Функция удаления элемента по значению

void scan(void(\*f)(void\*)); // Функция сканирования элементов с применением функции к каждому

};

static void\* DEL = (void\*)HASHDEL; // Статическая переменная для обозначения удаленного элемента

Object create(int size, int(\*getkey)(void\*)); // Функция создания объекта

#undef HASHDEL // Удаление макроса для значения удаленного элемента

**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;

}

//Получение следующего хэша в таблице

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

{

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

}

//Создание хэш-таблицы с введенным размером

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

{

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

}

//Создание хэш-таблицы

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) //Заполнение таблицы пустыми ячейками согласно размеру

data[i] = NULL;

}

//Вставка ключ-элемента в таблицу

bool Object::insert(void\* d)

{

bool b = false;

if ((double)N / (double)size >= 0.5) {

int newSize = size \* 2; //Увеличение размера новой таблицы в 2 раза

void\*\* newData = new void\*[newSize]; //Выдеделение динамической памяти в соответствии с размером

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

newData[i] = NULL; //Заполнение новой таблицы пустыми ячейками

}

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

if (data[i] != NULL && data[i] != DEL) {

int k = getKey(data[i]); //Получаем ключ нового элемента

int index = k % newSize; //Получаем индекс элемента в новой таблице

//Поиск пустой ячейки для вставки элемента

while (newData[index] != NULL) {

index = (index + 1) % newSize;

}

newData[index] = data[i]; //Вставляем элемент

}

}

delete[] data; //Освобождение памяти старой таблицы

data = newData; //Перенаправляем указатель старой таблицы на новый

size = newSize; //Обновляем размер таблицы

cout << "Размер таблицы обновлен" << endl;

}

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0); //Используем хэш-функцию для преборазования ключа

i != size && !b; j = Next\_hash(j, size, ++i))

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

data[j] = d;

N++;

b = true;

}

return b;

}

//Поиска ключа

int Object::searchInd(int 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;

}

//Поиск элемента

void\* Object::search(int key)

{

int t = searchInd(key);

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

}

//Удаление элемента из таблицы по заданному ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//Удаление по значению

bool Object::deleteByValue(void\* d)

{

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

}

//Проверка, является ячейка заполненной, пустой или удаленной

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>

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k; mas = z;

} AAA() {}

};

//Создание ключа

int key(void\* d)

{

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

return f->key;

}

//Вывод ключа

void AAA\_print(void\* d)

{

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

}

//Главная фукнция с меню

int main()

{

setlocale(LC\_ALL, "rus");

int siz = 10, choice, k;

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

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; //Добавляем ключ в таблицу

cout << "введите строку" << endl;

cin >> str;

a->mas = str; //Добавляем строку по клч

if (H.N == H.size) //Проверяем, заполнена ли таблица

cout << "Таблица заполнена" << endl;

else //Иначе вставляем в нее ключ-элемент

H.insert(a);

} break;

case 3: {

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

cin >> k;

H.deleteByKey(k);

} break;

case 4: {

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

cin >> k;

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

cout << "Элемент не найден" << endl;

else

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

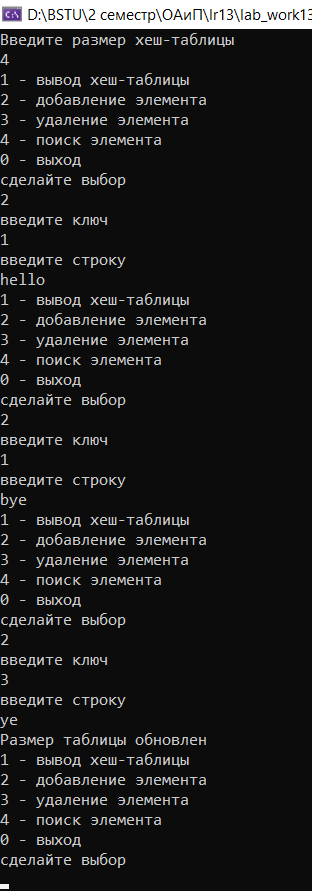
} break;

}

}

return 0;

}



Доп.задания:

Вариант 11



Из основного предыдущего варианта изменена фукнция insert в Hash.cpp

//Вставка ключ-элемента в таблицу

bool Object::insert(void\* d)

{

bool b = false;

if ((double)N / (double)size >= 0.8) {

int newSize = size \* 2; //Увеличение размера новой таблицы в 2 раза

void\*\* newData = new void\*[newSize]; //Выдеделение динамической памяти в соответствии с размером

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

newData[i] = NULL; //Заполнение новой таблицы пустыми ячейками

}

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

if (data[i] != NULL && data[i] != DEL) {

int k = getKey(data[i]); //Получаем ключ нового элемента

int index = k % newSize; //Получаем индекс элемента в новой таблице

//Поиск пустой ячейки для вставки элемента

while (newData[index] != NULL) {

index = (index + 1) % newSize;

}

newData[index] = data[i]; //Вставляем элемент

}

}

delete[] data; //Освобождение памяти старой таблицы

data = newData; //Перенаправляем указатель старой таблицы на новый

size = newSize; //Обновляем размер таблицы

cout << "Размер таблицы обновлен" << endl;

}

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0); //Используем хэш-функцию для преборазования ключа

i != size && !b; j = Next\_hash(j, size, ++i))

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

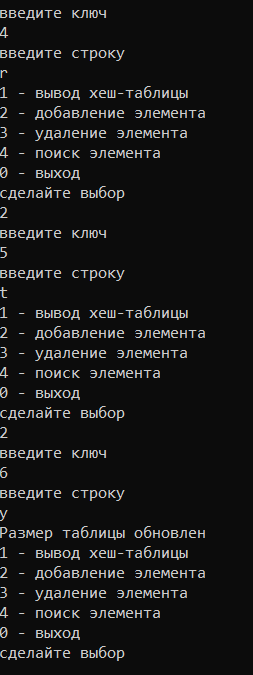
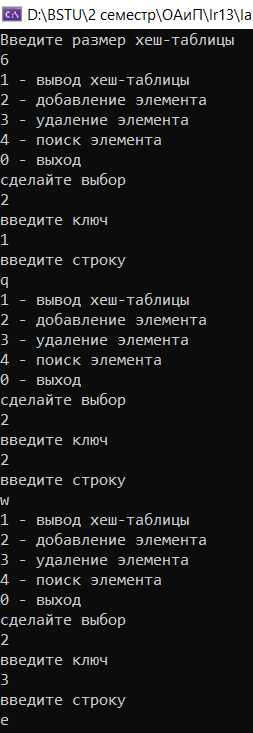
data[j] = d;

N++;

b = true;

}

return b;

}

Вариант 7



Изменены функции в Hash.cpp

#include "Hash.h"

#include <iostream>

// Произвольные простые числа, отличные от 0

const int a = 7;

const int b = 13;

const int p = 11;

int HashFunction(int key, int size) // Универсальная фукнция

{

return ((a \* key + b) % p) % size;

}

//Получение следующего хэша в таблице

int Next\_hash(int hash, int size)

{

return (hash + 1) % size;

}

//Создание хэш-таблицы с введенным размером

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

{

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

}

//Создание хэш-таблицы

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) //Заполнение таблицы пустыми ячейками согласно размеру

data[i] = NULL;

}

//Вставка ключ-элемента в таблицу

bool Object::insert(void\* d)

{

bool b = false;

if (N != size)

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

i != size && !b; j = Next\_hash(j, size))

if (data[j] == NULL || data[j] == DEL) //Если ячейка не пустая и не удалена

{

data[j] = d; //Вставляем элемент

N++; //Увеличиваем количество заполненных ячеек

b = true;

}

return b;

}

//Поиска ключа

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

if (N != 0)

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

!= size && !b; j = HashFunction(key, size))

if (data[j] != DEL)

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

{

t = j; b = true;

}

return t;

}

//Поиск элемента

void\* Object::search(int key)

{

int t = searchInd(key);

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

}

//Удаление элемента из таблицы по заданному ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//Удаление по значению

bool Object::deleteByValue(void\* d)

{

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

}

//Проверка, является ячейка заполненной, пустой или удаленной

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)

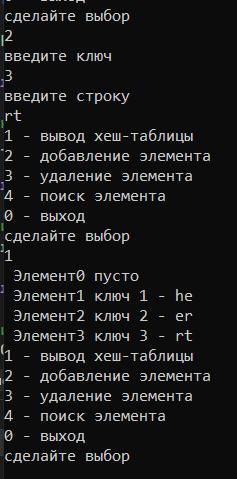
std::cout << " удален" << std::endl;

else

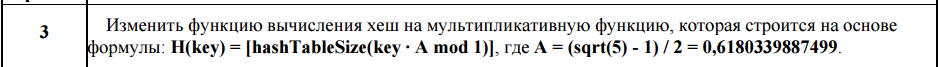
f((this->data)[i]);

}

}



Вариант 3



Изменены фукнции в Hash.cpp

#include "Hash.h"

#include <iostream>

int HashFunction(int key, int size) // Мультипликативная хеш-функция

{

double A = (sqrt(5) - 1) / 2; // Число Кнута

double hashValue = size \* (key \* A - int(key \* A)); // Вычисляем значение хеша

return static\_cast<int>(hashValue); // Возвращаем значение хеша как целое число

}

//Функция вычисления следующего хэша

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

{

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

}

//Создание хэш-таблицы с введенным размером

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

{

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

}

//Создание хэш-таблицы

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) //Заполнение таблицы пустыми ячейками согласно размеру

data[i] = NULL;

}

//Вставка ключ-элемента в таблицу

bool Object::insert(void\* d)

{

bool b = false;

if (N != size)

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

i != size && !b; j = Next\_hash(j, size, ++i))

if (data[j] == NULL || data[j] == DEL) //Если ячейка не пустая и не удалена

{

data[j] = d; //Вставляем элемент

N++; //Увеличиваем количество заполненных ячеек

b = true;

}

return b;

}

//Поиска ключа

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

if (N != 0)

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

!= size && !b; j = HashFunction(key, size))

if (data[j] != DEL)

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

{

t = j; b = true;

}

return t;

}

//Поиск элемента

void\* Object::search(int key)

{

int t = searchInd(key);

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

}

//Удаление элемента из таблицы по заданному ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//Удаление по значению

bool Object::deleteByValue(void\* d)

{

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

}

//Проверка, является ячейка заполненной, пустой или удаленной

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]);

}

}

