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»

 Выполнила:

Студентка1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: асс. Андронова М.В.

2023, Минск

Вариант 1



Main.cpp

#include "Hash.h"

#include <iostream>

#include <Windows.h>

using namespace std;

// Структура AAA для представления элементов хеш-таблицы.

struct AAA

{

int key; // Ключ элемента.

char\* mas; // Массив символов.

// Конструктор с параметрами.

AAA(int k, char\* z)

{

key = k; // Присвоение ключа.

mas = z; // Присвоение массива символов.

}

// Пустой конструктор (по умолчанию).

AAA() {}

};

// Функция для получения ключа элемента AAA.

int key(void\* d)

{

AAA\* f = (AAA\*)d; // Приведение указателя к типу AAA.

return f->key; // Возвращение ключа элемента.

}

// Функция для вывода элемента AAA на экран.

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl; // Вывод ключа и массива символов элемента AAA.

}

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

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

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;

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: {

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;

}

Hash.cpp

#include "Hash.h"

#include <iostream>

// Функция вычисления хеша с использованием мультипликативной хеш-функции.

int HashFunction(int key, int size, int p, int i) {

const double A = 0.6180339887499; // Константа A для мультипликативной хеш-функции

double hashValue = key \* A; // Вычисление промежуточного значения хеша

hashValue -= int(hashValue); // Получение дробной части значения хеша

return (p + int(size \* hashValue) + i +i\*i) % size; // Использование линейной функции для учета коллизии

}

// Функция для поиска индекса элемента в хеш-таблице.

int Object::searchInd(int key)

{

int t = -1; // Инициализация индекса элемента.

bool b = false; // Флаг для обозначения успешного поиска.

if (N != 0) // Проверка наличия элементов в таблице.

{

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

if (data[j] != DEL) {// Проверка на удаленность ячейки.

if (getKey(data[j]) == key) { // Проверка на соответствие ключа элемента.

t = j; // Сохранение индекса элемента.

b = true; // Отметка успешного поиска.

}

}

}

}

return t; // Возвращение указателя элемента (или -1, если не найден).

}

// Функция для вычисления следующего значения хеша.

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

{

return (hash + 5 \* p + 3 \* p \* p) % size; // Вычисление следующего значения хеша с учетом параметра p.

}

// Функция для вставки элемента в хеш-таблицу.

bool Object::insert(void\* d)

{

bool b = false; // Флаг для обозначения успешной вставки.

if (N != size) // Проверка на переполнение таблицы.

{

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0, i); i != size && !b; j = Next\_hash(j, size, ++i)) { // Цикл поиска места для вставки.

if (data[j] == NULL || data[j] == DEL) { // Проверка на пустоту или удаленность ячейки.

data[j] = d; // Вставка элемента.

N++; // Увеличение количества элементов в таблице.

b = true; // Отметка успешной вставки.

}

}

}

return b; // Возвращение результата вставки.

}

// Функция для создания объекта хеш-таблицы.

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

{

return \*(new Object(size, getkey)); // Создание и возвращение нового объекта хеш-таблицы.

}

// Конструктор объекта хеш-таблицы.

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

{

N = 0; // Инициализация количества элементов в таблице.

this->size = size; // Задание размера таблицы.

this->getKey = getkey; // Установка функции для получения ключа элемента.

this->data = new void\* [size]; // Выделение памяти под массив указателей на элементы.

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

data[i] = NULL; // Инициализация всех элементов массива как NULL.

}

// Функция для поиска элемента в хеш-таблице по ключу.

void\* Object::search(int key)

{

int t = searchInd(key); // Поиск индекса элемента.

return (t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента (или NULL, если не найден).

}

// Функция для удаления элемента из хеш-таблицы по ключу.

void\* Object::deleteByKey(int key)

{

int i = searchInd(key); // Поиск индекса элемента.

void\* t = data[i]; // Сохранение удаляемого элемента.

if (t != NULL) // Проверка на наличие элемента для удаления.

{

data[i] = DEL; // Отметка ячейки как удаленной.

N--; // Уменьшение количества элементов в таблице.

}

return t; // Возвращение удаленного элемента (или NULL, если не найден).

}

// Функция для удаления элемента из хеш-таблицы по значению.

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу, полученному из значения.

}

// Функция для сканирования содержимого хеш-таблицы и применения к каждому элементу названной функции.

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

{

for (int i = 0; i < this->size; i++) // Цикл сканирования всех ячеек таблицы.

{

std::cout << "Элемент" << i << '\t'; // Вывод номера ячейки.

if ((this->data)[i] == NULL) // Проверка на пустоту ячейки.

std::cout << "пусто" << std::endl; // Вывод сообщения о пустой ячейке.

else

if ((this->data)[i] == DEL) // Проверка на удаленность ячейки.

std::cout << "удален" << std::endl; // Вывод сообщения об удаленной ячейке.

else

f((this->data)[i]); // Применение функции к элементу.

}

}

Hash.h

#pragma once

// Макрос для обозначения удаленного элемента в хеш-таблице.

#define HASHDEL (void\*) -1

// Структура для представления хеш-таблицы.

struct Object

{

void\*\* data; // Массив указателей на элементы хеш-таблицы.

int size; // Размер хеш-таблицы.

int N; // Количество элементов в хеш-таблице.

int(\*getKey)(void\*); // Указатель на функцию для получения ключа элемента.

// Конструктор с параметрами.

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

// Методы для работы с хеш-таблицей.

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

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

};

// Указатель на удаленный элемент.

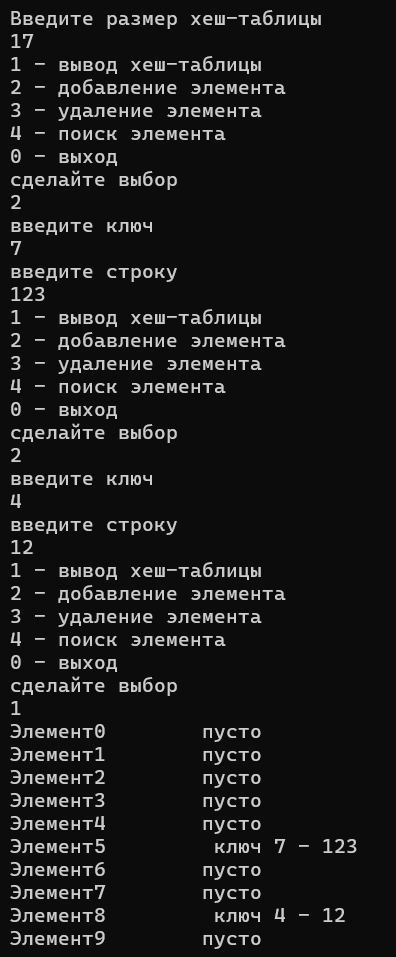
static void\* DEL = (void\*)HASHDEL;

// Функция для создания объекта хеш-таблицы.

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

// Отмена определения макроса HASHDEL.

#undef HASHDEL



Дополнительные варианты



#include <iostream>

#include <Windows.h>

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) {

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

}

struct Object {

void\*\* data;

int size;

int N;

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

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

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

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

};

// Функция вычисления хеша с использованием мультипликативной хеш-функции.

int HashFunction(int key, int size, int p, int i) {

const double A = 0.6180339887499; // Константа A для мультипликативной хеш-функции

double hashValue = key \* A; // Вычисление промежуточного значения хеша

hashValue -= int(hashValue); // Получение дробной части значения хеша

return (p + int(size \* hashValue) + i) % size; // Использование линейной функции для учета коллизии

}

static void\* DEL = (void\*)-1;

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

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; // Инициализация флага успешности вставки

// Проверка, заполнена ли таблица на 80% или больше

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

// Если таблица заполнена на 80% или больше, производим ее расширение

// Вычисление нового размера таблицы (удвоение текущего размера)

int newSize = size \* 2;

// Выделение памяти под новый массив указателей на элементы для новой таблицы

void\*\* newData = new void\* [newSize];

// Инициализация каждого элемента новой таблицы значением NULL

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

newData[i] = NULL;

// Копирование элементов из текущей таблицы в новую

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

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

// Получение ключа текущего элемента

int k = key(data[i]);

// Вычисление индекса нового элемента в новой таблице

int index = k % newSize;

// Поиск пустого слота в новой таблице для вставки элемента

while (newData[index] != NULL)

index = (index + 1) % newSize;

// Вставка элемента в новую таблицу

newData[index] = data[i];

}

}

// Освобождение памяти, занятой старой таблицей

delete[] data;

// Обновление указателя на данные для сохранения ссылки на новую таблицу

data = newData;

// Обновление размера таблицы

size = newSize;

}

// Попытка вставки нового элемента в текущую таблицу

// Цикл проходит по слотам таблицы, начиная с индекса, вычисленного по хеш-функции

for (int i = 0, t = getKey(d), j = t % size; i != size && !b; j = (j + 1) % size) {

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

// Если текущий слот пуст или удален, вставляем элемент

data[j] = d;

// Увеличиваем счетчик количества элементов в таблице

N++;

// Устанавливаем флаг успешной вставки

b = true;

}

}

// Возвращаем результат попытки вставки (успешно или нет)

return b;

}

// Функция для вычисления следующего значения хеша.

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

{

return (hash + 5 \* p + 3 \* p \* p) % size; // Вычисление следующего значения хеша с учетом параметра p.

}

// Функция для создания объекта хеш-таблицы.

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

{

return \*(new Object(size, getkey)); // Создание и возвращение нового объекта хеш-таблицы.

}

// Функция для поиска индекса элемента в хеш-таблице.

int Object::searchInd(int key)

{

int t = -1; // Инициализация индекса элемента.

bool b = false; // Флаг для обозначения успешного поиска.

if (N != 0) // Проверка наличия элементов в таблице.

{

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

{

if (data[j] != DEL) // Проверка на удаленность ячейки.

{

if (getKey(data[j]) == key) // Проверка на соответствие ключа элемента.

{

t = j; // Сохранение индекса элемента.

b = true; // Отметка успешного поиска.

}

}

}

}

return t; // Возвращение указателя элемента (или -1, если не найден).

}

// Функция для поиска элемента в хеш-таблице по ключу.

void\* Object::search(int key)

{

int t = searchInd(key); // Поиск индекса элемента.

return (t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента (или NULL, если не найден).

}

// Функция для удаления элемента из хеш-таблицы по ключу.

void\* Object::deleteByKey(int key)

{

int i = searchInd(key); // Поиск индекса элемента.

void\* t = data[i]; // Сохранение удаляемого элемента.

if (t != NULL) // Проверка на наличие элемента для удаления.

{

data[i] = DEL; // Отметка ячейки как удаленной.

N--; // Уменьшение количества элементов в таблице.

}

return t; // Возвращение удаленного элемента (или NULL, если не найден).

}

// Функция для удаления элемента из хеш-таблицы по значению.

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу, полученному из значения.

}

// Функция для сканирования содержимого хеш-таблицы и применения к каждому элементу названной функции.

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

{

for (int i = 0; i < this->size; i++) // Цикл сканирования всех ячеек таблицы.

{

std::cout << "Элемент" << i << '\t'; // Вывод номера ячейки.

if ((this->data)[i] == NULL) // Проверка на пустоту ячейки.

std::cout << "пусто" << std::endl; // Вывод сообщения о пустой ячейке.

else

if ((this->data)[i] == DEL) // Проверка на удаленность ячейки.

std::cout << "удален" << std::endl; // Вывод сообщения об удаленной ячейке.

else

f((this->data)[i]); // Применение функции к элементу.

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int siz = 10, choice, k;

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

std::cin >> siz;

Object H = create(siz, key);

for (;;) {

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

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

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

std::cout << "4 - поиск элемента" << std::endl;

std::cout << "0 - выход" << std::endl;

std::cout << "сделайте выбор" << std::endl;

std::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];

std::cout << "введите ключ" << std::endl;

std::cin >> k;

a->key = k;

std::cout << "введите строку" << std::endl;

std::cin >> str;

a->mas = str;

if (H.N == H.size)

std::cout << "Таблица заполнена" << std::endl;

else

H.insert(a);

}

break;

case 3: {

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

std::cin >> k;

H.deleteByKey(k);

}

break;

case 4: {

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

std::cin >> k;

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

std::cout << "Элемент не найден" << std::endl;

else

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

}

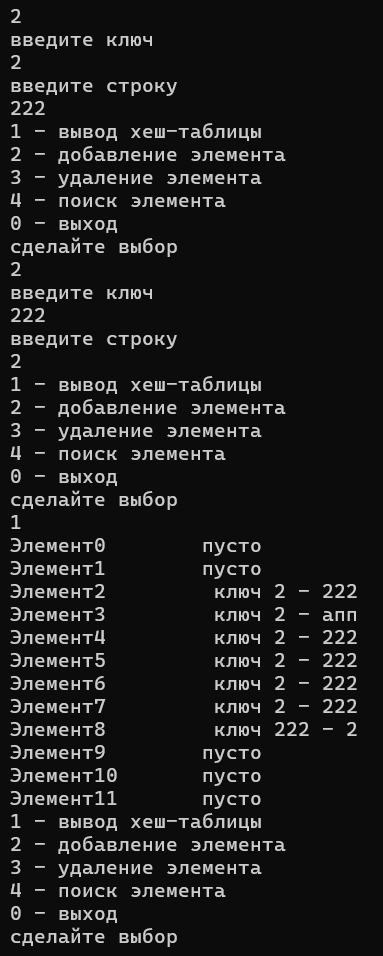
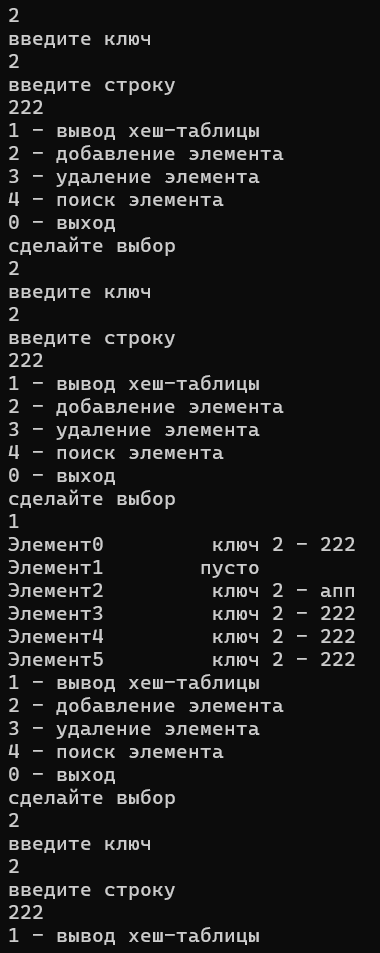
break;

}

}

return 0;

}





Тот же код только if ((double)N / size >= 0.8) меняется на if ((double)N / size >= 0.5)



Тот же код только if ((double)N / size >= 0.8) меняется на if ((double)N / size >= 0.7)