Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Лабораторная работа № 5

«ОБРАБОТКА ДАННЫХ НА ОСНОВЕ МЕТОДА

ЗАКРЫТОГО ХЕШИРОВАНИЯ»

Вариант 3

Выполнил: студент гр. 281073

Буйновский М.В.

Проверил: Потоцкий Д.С.

Минск 2023

**Цель работы:** научиться выполнять операции с данными, используя методы закрытого хеширования.

**Задание:** ввести массив из *n* целых чисел из заданного диапазона. Создать хеш-таблицу из *М* элементов. Осуществить поиск элемента в хеш-таблице. Вывести на экран исходный массив, хеш-таблицу и результат поиска. Задание: n = 15, диапазон значений 12000-34000, M = 20, Схема хеширования — с открытой адресацией.

**Код программы:**

**hash\_map.h**

#include <iomanip>

#include <iostream>

#include <stdexcept>

#include <string>

#include <vector>

template <typename T = int> class HashMap

{

public:

typedef unsigned key\_t;

private:

struct HashEntry

{

HashMap::key\_t key = (HashMap::key\_t)-1;

T value = (T)0;

};

unsigned capacity;

std::vector<HashEntry> map;

unsigned hash(key\_t key)

{

return 10 + (key % 5);

}

void showMap(HashMap::key\_t hash, std::string label, unsigned ansi\_color)

{

std::cout << " map["

<< "\033[36m" << std::setw(2) << hash << "\033[0m"

<< "] - "

<< "\033[" << ansi\_color << "m" << label << "\033[0m" << std::endl;

}

public:

HashMap(HashMap::key\_t capacity) : capacity(capacity), map(std::vector<HashEntry>(capacity))

{

}

void show()

{

std::cout << "Hash table: " << std::endl;

for (HashMap::key\_t i = 0; i < map.size(); i++)

{

if (this->map[i].key != (HashMap::key\_t)-1)

{

this->showMap(

i,

"Key: \033[36m" + std::to\_string(this->map[i].key) + "\033[0m" + "; Value: " + "\033[33m"

+ std::to\_string(this->map[i].value),

0

);

}

else

{

this->showMap(i, (std::string) "Empty", 32);

}

}

}

void insert(HashMap::key\_t key, T value)

{

HashMap::key\_t hash = this->hash(key);

HashMap::key\_t initial = hash;

std::cout << "Inserting '"

<< "\033[33m" << value << "\033[0m"

<< "' with key '"

<< "\033[36m" << key << "\033[0m"

<< "':" << std::endl;

// Find empty cell

if (this->map[hash].key != (HashMap::key\_t)-1)

{

do

{

if (this->map[hash].key != (HashMap::key\_t)-1)

{

this->showMap(hash, "FILLED", 31);

}

hash = map.size() - 1 == hash ? 0 : hash + 1;

} while (initial != hash && this->map[hash].key != (HashMap::key\_t)-1);

}

if (initial == hash && this->map[hash].key != (HashMap::key\_t)-1)

{

std::cout << " "

<< "\033[31m"

<< " Hash map is filled. Skipping."

<< "\033[0m" << std::endl;

}

else

{

this->showMap(hash, "EMPTY \033[0m - filling", 32);

this->map[hash] = HashMap::HashEntry{key, value};

}

}

T \*find(HashMap::key\_t key)

{

HashMap::key\_t hash = this->hash(key);

HashMap::key\_t initial = hash;

do

{

if (this->map[hash].key == key)

{

this->showMap(hash, "HIT", 32);

return &(this->map[hash].value);

}

this->showMap(hash, "MISS", 31);

hash = map.size() - 1 == hash ? 0 : hash + 1;

} while (initial != hash);

return nullptr;

}

};

**main.cpp**

#include <chrono>

#include <iomanip>

#include <iostream>

#include <random>

#include <stdlib.h>

#include "hash\_map.h"

template <typename T = unsigned> void showVector(std::vector<T> vec)

{

std::cout << "Input array ("

<< "\033[33m" << vec.size() << "\033[0m"

<< "):" << std::endl;

for (size\_t i = 0; i < vec.size(); i++)

{

std::cout << " array["

<< "\033[36m" << std::setw(2) << i << "\033[0m"

<< "] - "

<< "\033[33m" << vec[i] << "\033[0m" << std::endl;

}

std::cout << std::endl;

}

int main()

{

const int M = 20;

const int n = 15;

const int min = 12000;

const int max = 34000;

// Some random C++ bullshit

std::random\_device rd;

// seed value is designed specifically to make initialization

// parameters of std::mt19937 (instance of std::mersenne\_twister\_engine<>)

// different across executions of application

std::mt19937::result\_type seed

= rd()

^ ((std::mt19937::result\_type

)std::chrono::duration\_cast<std::chrono::seconds>(std::chrono::system\_clock::now().time\_since\_epoch())

.count()

+ (std::mt19937::result\_type)std::chrono::duration\_cast<std::chrono::microseconds>(

std::chrono::high\_resolution\_clock::now().time\_since\_epoch()

)

.count());

std::mt19937 gen(seed);

std::uniform\_int\_distribution<unsigned> distribution(min, max);

std::vector<unsigned> vec;

HashMap<unsigned> map(M);

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

{

vec.push\_back(distribution(gen));

map.insert(i + 1, vec.back());

std::cout << std::endl;

}

showVector(vec);

map.show();

std::cout << std::endl;

while (true)

{

int searchKey;

std::cout << "Search for value with key ("

<< "\033[35m"

<< "use ctrl + c to exit"

<< "\033[0m"

<< ") - "

<< "\033[33m";

std::cin >> searchKey;

std::cout << "\033[0m";

unsigned \*value = map.find(searchKey);

if (value == nullptr)

{

std::cout << std::endl

<< "\033[31m"

<< " Key is not present in the hash map."

<< "\033[0m" << std::endl

<< std::endl;

}

else

{

std::cout << std::endl

<< " Found: "

<< "\033[33m" << \*value << "\033[0m" << std::endl

<< std::endl;

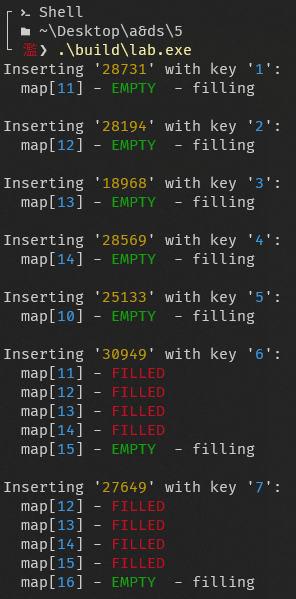
}

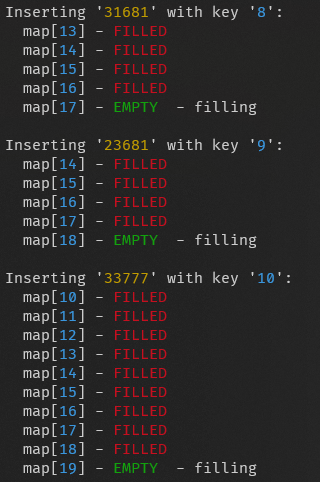
}

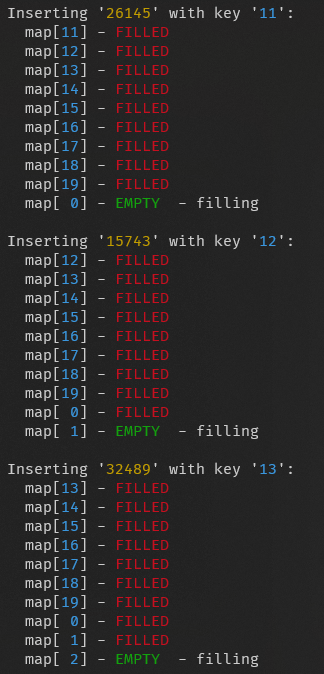
return EXIT\_SUCCESS;

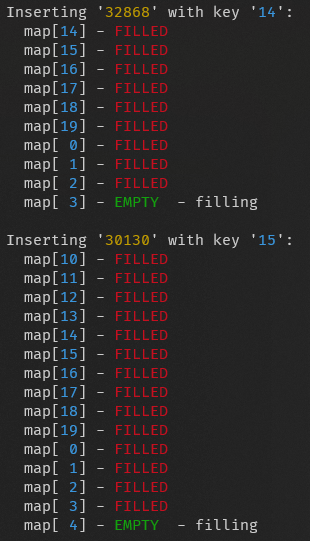
}

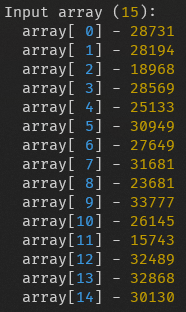
**Скриншот работы программы, демонстрация результатов:**

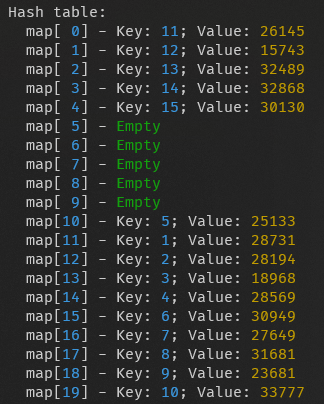
****

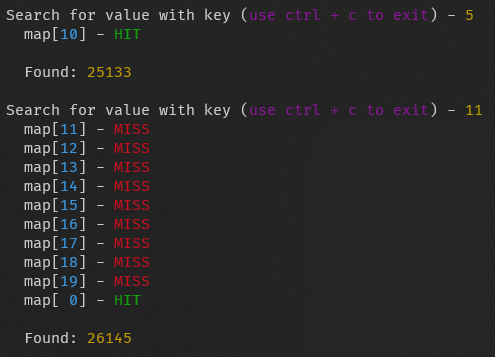
****

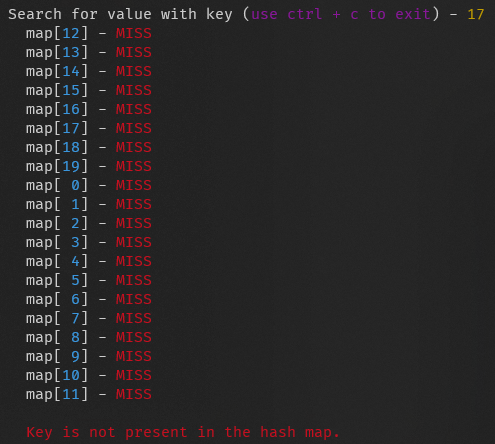
****

****

****

****

****

****

**Контрольные вопросы:**

**1. В чем отличия между открытым и закрытым хешированием данных?**

Открытое и закрытое хеширование данных отличаются в способе обработки коллизий, то есть ситуаций, когда двум разным ключам соответствует одно и то же значение хеша.

* В открытом хешировании, если происходит коллизия, элементы с одинаковым значением хеша добавляются в структуру данных (обычно в виде связанных списков), создавая цепочки. При поиске элемента с использованием хеша, нужно последовательно просмотреть цепочку, чтобы найти конкретный элемент.
* В закрытом хешировании, если происходит коллизия, другой свободный слот в хеш-таблице ищется с помощью определенного алгоритма. Элементы с одинаковым хешем могут быть размещены в разных слотах сегментов хеш-таблицы. Методы закрытого хеширования включают линейное и квадратичное опробование, где вычисляется новый слот на основе начального хеша и номера попытки.

**2. Объясните суть коллизий при закрытом хешировании данных.**

Коллизии в закрытом хешировании данных возникают, когда двум разным ключам соответствует один и тот же слот в хеш-таблице. Это означает, что два элемента имеют одинаковое значение хеша. Коллизии могут привести к перезаписи элементов или сложности поиска элементов в хеш-таблице, особенно если используется линейное или квадратичное опробование для разрешения коллизий.

**3. Перечислите способы борьбы с коллизиями.**

Способы борьбы с коллизиями в закрытом хешировании включают:

* Метод цепочек: при коллизии элементы с одинаковым хешем добавляются в список (цепочку) в одном слоте хеш-таблицы. Это позволяет хранить несколько элементов с одним хешем в одном слоте.
* Метод открытой адресации: при коллизии новый слот в хеш-таблице выбирается с использованием определенного алгоритма, который перебирает другие слоты, пока не будет найден свободный слот. Это позволяет разместить элементы с одинаковым хешем в разных слотах.
* Линейное опробование: при коллизии новый слот выбирается путем последовательного перебора слотов с фиксированным шагом, обычно равным единице.
* Квадратичное опробование: при коллизии новый слот выбирается путем перебора слотов с использованием квадратичной функции, зависящей от номера попытки. Это позволяет более равномерно распределить элементы в хеш-таблице и избежать слишком длинных цепочек.

**Вывод:** в результате выполнения лабораторной работы освоили использование закрытого хеширования для обработки данных и изучили основные принципы работы алгоритмов хеширования. Также рассмотрели понятие коллизий и способы их предотвращения.