Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ОТЧЕТ
по дисциплине Алгоритмы и структуры данных
Практическая работа №4 — Хэш-таблицы

Преподаватель _____ Матковский И. В.
подпись, дата инициалы, фамилия

Студент КИ19-07Б, 031941597 _____ Горбацевич А. А.
номер группы, зачётной книжки подпись, дата инициалы, фамилия

Красноярск 2020

Содержание

1. Задание на работу

1.1 Разработать программу, демонстрирующую основные этапы работы с хэш-таблицами — формирование хэша, получение данных, разрешение коллизий.

2. Задание на вариант

2.1 Хэш-таблица с методом хэширование MD5 (взят Murmur Hash) и решением коллизий методом открытой адресации.

3. Исходный код программы

```cpp
// dsaa_04.cpp
// Горбацевич Андрей
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <cassert>
#include <algorithm>
#include <unordered_set>


typedef int num_type;
typedef double long_type;


class PseudoMurmur {
private:
    static inline uint32_t avalanche(uint32_t val) {
        val *= 0xcc9e2d51;
        val = (val << 15u) | (val >> 17u);
        val *= 0x1b873593;
        return val;
    }

public:
    static uint32_t hash_32(const uint32_t &value) {
        uint32_t h = 0xdeadbeef;
        h ^= PseudoMurmur::avalanche(value);
        h = (h << 13u) | (h >> 19u);
        h *= 5;
        h += 0xe6546b64;
        h ^= 4u;
        h ^= h >> 16u;
        h *= 0x85ebca6b;
        h ^= h >> 13u;
        h *= 0xc2b2ae35;
        h ^= h >> 16u;
        return h;
    }
};


struct HashableNode {
    num_type value = 0;
    bool used = false;

    HashableNode() = default;
    explicit HashableNode(num_type value) : value(value), used(true) {}

    uint32_t hash() const {  // NOLINT(modernize-use-nodiscard)
        return PseudoMurmur::hash_32(this->value);
    }

    bool operator==(const HashableNode &other) const {
        return this->used == other.used && this->value == other.value;
```

```
    }
};


class IntHashtable {
private:
    std::vector<std::unordered_set<num_type>> _data;

    uint32_t _find_real_pos(HashableNode value) {
        auto pos = value.hash() % _data.capacity();

        auto set = _data[pos];
        auto it_pos = set.find(value.value);

        return it_pos == set.end()? -1 : *it_pos;
    }
public:
    IntHashtable() {
        this->_data = std::vector<std::unordered_set<num_type>>(23);
    }

    ~IntHashtable() = default;

    void put(num_type value) {
        auto ppos = HashableNode(value).hash() % _data.capacity();
        this->_data[ppos].insert(value);
    }

    num_type find(num_type value) {
        return this->_find_real_pos(HashableNode(value));
    }

    void remove(num_type value) {
        auto ppos = HashableNode(value).hash() % _data.capacity();
        auto pos = this->find(value);
        if (pos != -1) {
            this->_data[ppos].erase(value);
        }
    }

    [[maybe_unused]] uint32_t capacity() const {  // NOLINT(modernize-use-nodiscard)
        return this->_data.capacity();
    }

    uint32_t size() const {  // NOLINT(modernize-use-nodiscard)
        int real_size = 0;
        for (const auto &ref : this->_data) {
            real_size += ref.size();
        }
        return real_size;
    }
};


inline void time_passed(std::chrono::system_clock::time_point start, long_type &holder) {
    auto stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
    holder = duration.count();
```

```
}

template<typename T>
T input(const std::string &message="") {
    if (message.length() > 0) {
        std::cout << message << " ";
    }
    std::cout << ">>>";

    T out;
    std::cin >> out;
    return out;
}


void test_insertion(IntHashtable &hashtable);
void test_search(IntHashtable &hashtable);
void test_deletion(IntHashtable &hashtable);


int main() {
    IntHashtable el;

    test_insertion(el);
    test_search(el);
    test_deletion(el);
    test_search(el);
    test_deletion(el);
    test_search(el);

    return 0;
}

void test_insertion(IntHashtable &hashtable) {
    std::vector<num_type> in_nums;
    int is_manual = input<int>("Manual insertion? [1 = yes; other = no]");
    if (is_manual == 1) {  // users input can be O(inf), so we can read unput to vector and then use it
        int N = input<int>("How many numbers you wish to input?");
        for (int i = 0; i < N; i++) {
            int cv = input<int>("num");
            in_nums.push_back(cv);
        }
    }
    else {  // reading from disk can also take a lot of time, so the same approach
        std::ifstream ifs("tests/snba1.txt");
        int N;
        ifs >> N;
        for (int i = 0; i < N; i++) {
            int cv;
            ifs >> cv;
            in_nums.push_back(cv);
        }
    }

    auto start = std::chrono::high_resolution_clock::now();
    long_type end;

    for (const auto &ref_num : in_nums) {
```

```
        hashtable.put(ref_num);
    }

    time_passed(start, end);
    printf("Insertion of %d items taken %.0f microseconds\n", in_nums.size(), end);
}

void test_search(IntHashtable &hashtable) {
    std::vector<num_type> in_nums;
    int is_manual = input<int>("Manual search? [1 = yes; other = no]");
    if (is_manual == 1) {
        int N = input<int>("How many numbers you wish to search?");
        for (int i = 0; i < N; i++) {
            int cv = input<int>("num");
            in_nums.push_back(cv);
        }
    }
    else {
        // + + + + - - + + -
        in_nums = { 140, 282, 26306, 72218, 4567, 69069, 45670, 69068, 45679 };
    }

    auto start = std::chrono::high_resolution_clock::now();
    long_type end;

    for (const auto &ref_num : in_nums) {
        printf("%d - %s found\n", ref_num, hashtable.find(ref_num) == -1? "not" : "");
    }

    time_passed(start, end);
    printf("Search of %d items taken %.0f microseconds\n", in_nums.size(), end);
}

void test_deletion(IntHashtable &hashtable) {
    std::vector<num_type> in_nums;
    int is_manual = input<int>("Manual deletion? [1 = yes; other = no]");
    if (is_manual == 1) {
        int N = input<int>("How many numbers you wish to delete?");
        for (int i = 0; i < N; i++) {
            int cv = input<int>("num");
            in_nums.push_back(cv);
        }
    }
    else {
        // + + + + - - + + -
        in_nums = { 140, 282, 26306, 72218, 4567, 69069, 45670, 69068, 45679 };
    }

    auto start = std::chrono::high_resolution_clock::now();
    long_type end;


    for (const auto &ref_num : in_nums) {
        hashtable.remove(ref_num);
    }

    time_passed(start, end);
    printf("Deletion of %d items taken %.0f microseconds\n", in_nums.size(), end);
```

```
    for (const auto &ref_num : in_nums) {
        assert(hashtable.find(ref_num) == -1);
    }
}
```

4. Теоретические оценки временной сложности алгоритмов

4.1 Вставка: O(1) (для защиты)

4.2 Поиск: O(n) (для защиты)

4.3 Удаление: O(n) (для защиты)

5. Экспериментальные оценки временной и пространственной сложности программы

Размер входного набора данных	Вставка, микросекунды	Поиск, 9 элементов, микросекунды	Удаление, 9 элементов, микросекунды
500	0	11942	0
1000	996	10985	0
2000	997	16956	995
2500	1022	11965	997
5000	1990	10970	992
7500	3990	12961	996
10000	4989	16955	1995
20000	14962	14958	1993
25000	14952	17963	3990
50000	28918	15960	6980

Приложение А
Результаты работы программы

```
C:\Users\Admin\CLionProjects\instp_01\cmake-build-debug\instp_01.exe
Manual insertion? [1 = yes; other = no] >>>2
Insertion of 50000 items taken 17952 microseconds
Manual search? [1 = yes; other = no] >>>2
140 -  found
282 -  found
26306 -  found
72218 -  found
4567 - not found
69069 - not found
45670 -  found
69068 -  found
45679 - not found
Search of 9 items taken 14958 microseconds
Manual deletion? [1 = yes; other = no] >>>2
Deletion of 9 items taken 8974 microseconds
Manual search? [1 = yes; other = no] >>>2
140 - not found
282 - not found
26306 - not found
72218 - not found
4567 - not found
69069 - not found
45670 - not found
69068 - not found
45679 - not found
Search of 9 items taken 15956 microseconds
Manual deletion? [1 = yes; other = no] >>>2
Deletion of 9 items taken 5035 microseconds
Manual search? [1 = yes; other = no] >>>2
140 - not found
282 - not found
26306 - not found
72218 - not found
4567 - not found
69069 - not found
45670 - not found
69068 - not found
45679 - not found
Search of 9 items taken 12969 microseconds

Process finished with exit code 0
```

Рисунок 1: результат работы программы