# ВШЭ АиСД 2021. Хеширование, хеш-таблицы, CHM

4 дек 2021, 18:37:46
старт: 22 ноя 2021, 11:00:00
финиш: 2 дек 2021, 03:00:00
длительность: 9д. 16ч.
начало: 22 ноя 2021, 11:00:00
конец: 2 дек 2021, 03:00:00

## A. Хеш-таблица (0.5)

| | |
|---|---|
| Ограничение времени | 1.8 секунд |
| Ограничение памяти | 300.0 Мб |
| Ввод | стандартный ввод |
| Вывод | стандартный вывод |

Вам предстоит реализовать класс **хеш-таблицы**, который поддерживает хранение объектов, заданных парами **(Ключ, Значение)**. Хеширование объектов выполняется по полю **Ключ**, то есть при вставке объекта в хеш-таблицу вычисляется значение используемой хеш-функции от поля **Ключ**. Разрешение коллизий выполняется с помощью метода цепочек.

Создаваемый шаблонный класс `HashTable` параметризуется:

1. Типом используемых ключей `KeyType`
2. Типом значения `ValueType`
3. Типом используемого хешера `Func`. По умолчанию используется объект стандартной библиотеки `std::hash<KeyType>`.

В случае если таблица заполнена более, чем на некоторое отсечное значение (коэффициент наполненности), выполняется перехеширование: удвоение емкости таблицы и перенос текущих объектов в новую таблицу.

Скажем пару слов про «хешер». Это некоторый тип, который по ключу типа `KeyType` умеет выдавать значение типа `size_t`, которое можно получить, используя функциональный вызов.

Например:
```
// пусть hasher имеет тип Func
KeyType key = ...; // какой-то ключ
size_t num = hasher(key);
```

Благодаря использованию шаблонов, тип `Func` может быть чем угодно — функцией, лямбдой или же классом, для которого перегружен оператор вызова (). Это позволяет пользователю вашего класса выбирать наиболее предпочтительный для него вариант хеш-функции. Если же пользователя устраивает стандартный вариант, то используется стандартный тип `std::hash<KeyType>`.

Обратите внимание, что сам по себе хешер всего лишь дает вам возможность получить для любого объекта некоторое число (и предоставляется пользователем класса, поскольку вы заранее не знаете, с какими типами будет использоваться ваша таблица), а вот как его использовать для организации быстрой хеш-таблицы — уже ваша забота. Однако же если хешер, например, возвращает для всех ключей число $0$, то ясно, что это проблема пользователя, а от вас мало что зависит (тем не менее, ваш класс должен по-прежнему корректно работать в таких ситуациях, это должно отражаться только на времени работы). Поэтому вы можете считать, что хешер распределяет ключи по диапазону `size_t` достаточно равномерно (в предположении, что ключи случайны) — в частности, это верно для умолчательного варианта `std::hash<KeyType>`.

Ваш класс должен содержать следующие конструкторы и методы:

1. Конструктор по умолчанию. Размер таблицы по умолчанию принимается равным **100**. Коэффициент наполненности по умолчанию принимается равным **0.5**.
2. Конструктор с одним параметром - типом используемой хеш-функции.
3. Конструктор, который позволяет задавать максимальный размер таблицы и коэффициент наполненности, а также тип используемой хеш-функции. Последний параметр - тип используемой функции - может быть опущен. Коэффициент наполненности принимает значения от $0$ до $1$, где $1$ включительно, а $0$ нет. В случае передачи в конструктор неверного значения коэффициента наполненности, следует принять его равным дефолтному значению - **0.5**.
4. Деструктор.
5. Константный метод `size`, возвращающий количество элементов в таблице.
6. Константный метод `capacity`, возвращающий текущее значение емкости таблицы.
7. Метод `insert`, который вставляет пару (ключ, значение) в хеш-таблицу. Принимает на вход два параметра (ключ, значение), вычисляет значение хеш-функции от поля **Ключ** (с поправкой на размер таблицы) и добавляет запись об этом объекте в таблицу. В случае коллизии, объект добавляется в конец соответствующей хешу цепочки. В случае, если объект с таким полем **ключ** уже

имеется, то происходит обновление поля **Значение**. При превышении коэффициента наполненности происходит перехеширование. Тип возвращаемого значения void.

8. Метод `erase`, который принимает ключ и удаляет соответствующую пару (ключ, значение) из таблицы. В случае, если запись с таким ключом не найдена, то метод ничего не меняет.

9. Метод `find`, возвращающий указатель на значение, соответствующее переданному ключу хеш-таблицы, если ключ в ней присутствует, и nullptr в противном случае.

10. Оператор индексации (operator []), который возвращает запись хеш-таблицы по значению переданного хеш-кода (односвязный список элементов) по ссылке. Если переданный хеш-код находится вне таблицы, то генерируется исключение `std::out_of_range`. Если переданный хеш-код указывает на пустую ячейку, то генерируется исключение `std::runtime_error`.

11. Метод `at`, который возвращает запись хеш-таблицы по значению переданного хеш-кода (односвязный список элементов) по значению. Если переданный хеш-код находится вне таблицы, то генерируется исключение `std::out_of_range`. Если переданный хеш-код указывает на пустую ячейку, то генерируется исключение `std::runtime_error`.

## Формат ввода

Всего будет не более $10^6$ операций с таблицей.

## Примечания

Вы должны прислать код, содержащий определение вашего класса.

Для удобства рекомендуется использовать публичный интерфейс из данного примера: ссылка.

Набрать здесь | Отправить файл

```
#include <iostream>

template <class KeyType, class ValueType>
struct Node {
    KeyType key_;
    ValueType value_;
    Node *next_;

    Node(KeyType, ValueType);
};

template <class KeyType, class ValueType>
Node<KeyType, ValueType>::Node(KeyType key_type, ValueType value_type) {
    key_ = key_type;
    value_ = value_type;
    next_ = nullptr;
}

// Хеш-таблица
template <class KeyType, class ValueType, class Func = std::hash<KeyType>>
class HashTable {
private:
    size_t size_;
    int count_elements_;
    double capacity_fill_;
    Func func_;
    Node<KeyType, ValueType> **nodes_;

public:
    HashTable();

    explicit HashTable(Func);

    HashTable(size_t, double, Func = std::hash<KeyType>());

    ~HashTable();

```

Отправить

Следующая