Те, кто может что-то представить, могут сотворить невозможное.

Алан Тьюринг

Алгоритмы и структуры данных–2

2024–2025 учебный год

SET 5. Домашняя работа

Хеширование и вероятностные структуры данных

январь–февраль

27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28		
пн	вт	ср	чт	пт	сб	вс

создано командой преданных своему делу энтузиастов

Немного инструкций

Задания в рамках домашней работы SET 5 подразделяются на два блока:

1. *Блок A* «Аналитические задания» — задачи, связанные с анализом и применением различных способов хеширования.

   > Решения заданий Блока A оформляются в письменном виде в любом удобном формате (LaTeX, текстовый документ, скан, изображение и др.) и загружаются для оценки в соответствующую форму раздела `SET 5. Домашняя работа` на странице курса в `LMS` по ссылке: `https://edu.hse.ru/course/view.php?id=224738`.

2. *Блок P* «Задания на разработку» — задачи, связанные с реализацией алгоритмов и структур данных, основанных на использовании хеширования.

   > Решения заданий Блока P загружаются в систему `CODEFORCES` и проходят автоматизированное тестирование. Для загрузки нужно перейти на `https://dsahse.contest.codeforces.com` и выбрать соответствующее соревнование. Доступ к соревнованию предоставлен по тем же учетным данным, что и к системе Яндекс.Контест.

Домашняя работа SET 5 содержит 3 обязательных задач в Блоке A и 3 обязательных задачи в Блоке P. Баллы, которые можно получить за решение задач, распределены следующим образом:

Блок A				**Блок P**		
A1	A2	A3b	A4	P1	P2	P3
8	12	10	7	15	12	8

Задачи, помеченные “b” не являются обязательными — баллы за их решение относятся к *бонусным*.

Важные даты

1. Домашняя работа SET 4 открыта с **17:50 27 января 2025 г**.
2. Прием решений завершается в **02:00 10 февраля 2025 г**.
3. Защита решения бонусной задачи A3b принимается до **21 февраля 2025 г**.

Содержание

Успехов!

Задача A1. Анализ линейного пробирования

В хеш-таблице с открытой адресацией разрешение коллизий производится с помощью *линейного пробирования*. При удалении объекта из хеш-таблицы свободная ячейка получает значение `ERASED`, отличное от `NULL`, которое обозначает *пустое* значение.

Ниже приведены алгоритмы вставки, удаления и поиска, где `M` обозначает размер хеш-таблицы:

```
INSERT(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key) return
        ind = (ind + 1) mod M

    table[ind] = key
```

```
DELETE(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key)
            table[ind] = ERASED
            return
        ind = (ind + 1) mod M
```

```
SEARCH(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key)
            return true
        ind = (ind + 1) mod M
    return false
```

Система оценки

1. 5 баллов Приведенные выше алгоритмы вставки, удаления и поиска ключа имеют проблему, которая приводит к *долгому* выполнению некоторой(-ых) последовательности(-ей) этих операций.
   - Найдите такую(-ие) последовательность(-и) операций вставки, удаления и поиска.
   - Охарактеризуйте соответствующее состояние хеш-таблицы. Приведите примеры.
2. 3 балла Предложите доработки (*кроме* перехеширования) исходных алгоритмов вставки, удаления и поиска, которые помогут исправить обнаруженную вами проблему.

Задача A2. Кубическое пробирование

Хеш-таблицы с открытой адресацией используют различные методы пробирования для разрешения коллизий, к основным из которых можно отнести:

1. *Линейное* пробирование, при котором последовательно проверяются ячейки хеш-таблицы с индексами `hash(key)`, `hash(key)` + 1, `hash(key)` + 2, ...
2. *Квадратичное* пробирование `hash(key, i)` = `hash(key)` + $c_1 \cdot i + c_2 \cdot i^2$, при котором:
   - в простом варианте при $c_1 = c_2 = 1$ последовательно проверяются ячейки `hash(key)`, `hash(key)`+1, `hash(key)`+2, `hash(key)`+6, ...
   - для хеш-таблицы размера $M = 2^m$ при $c_1 = c_2 = 0.5$ последовательно проверяются ячейки `hash(key)`, `hash(key)`+1, `hash(key)`+3, `hash(key)`+6, ...

Мы решили пойти дальше и рассмотреть кубическое пробирование, при котором проверка ячеек в хеш-таблице выполняются по следующему правилу: `hash(key, i)` = `hash(key)` + $c_1 \cdot i + c_2 \cdot i^2 + c_3 \cdot i^3$.

Оцените, будет ли кубическое пробирование выполнять распределение ключей по хеш-таблице лучше (более равномерно), чем квадратичное, с точки зрения образования кластеров и возникновения коллизий. Подкрепите свои рассуждения *программными экспериментами* с хеш-таблицами различных размеров, а также приложите код. *Ограничений на используемые языки программирования в этом задании нет.*

Система оценки

1. 6 баллов Общий анализ, сравнение и обоснование достоинств, а также недостатков кубического пробирования.
2. 6 баллов Реализация и анализ программных экспериментов по применению кубического пробирования для разрешения коллизий.

Задача A3b. Взломщик!

Для хеширования строковых ключей, которые могут содержать строчные/прописные латинские буквы и цифры, используется следующая полиномиальная хеш-функция, значение которой определяется числовым параметром `p`:

```
size_t hash(std::string key) {
    const int p = ???;
    long long h = 0, p_pow = 1;
    for (size_t i = 0; i < s.length(); ++i) {
        h += (s[i] - 'a' + 1) * p_pow;
        p_pow *= p;
    }
    return h;
}
```

Для того, чтобы взломать хеш-функцию, требуется найти такой набор строк, который вызывает *коллизии* — одинаковые значения хеш-функции.

Система оценки

1. 7 баллов Одним из способов подбора строк, вызывающих коллизии, является поиск так называемых нейтральных элементов — строк, значение хеш-функции которых *обращается в 0*. Разработайте и обоснуйте алгоритм поиска строк, состоящих из двух символов, которые будут являться нейтральными элементами. Представьте обоснование и реализацию алгоритма. *Ограничений на используемые языки программирования в этом задании нет.*

2. 3 балла Найдите нейтральные элементы для всех значений параметра $\texttt{p} \leqslant 31$.

Задача A4. Совместимость фильтров Блума

Предположим, что по двум множествам объектов A и B создано два фильтра Блума одинакового размера — $F(A)$ и $F(B)$ соответственно. Путем выполнения поэлементного побитового И над фильтрами $F(A)$ и $F(B)$ был получен третий фильтр, который обозначен $F(AB)$.

В рамках задания фильтр Блума можно рассматривать в виде *одного* битового вектора.

Система оценки

Ответьте на следующие вопросы:

1. 3 балла Верно ли, что $F(AB)$ будет выдавать положительные ответы о принадлежности объектов из множества $A \cap B$? Почему (нет)?

2. 4 балла Верно ли, что $F(AB)$ будет в точности соответствовать другому фильтру, который будет получен в результате последовательной вставки объектов из множества $A \cap B$? Почему (нет)? *Хеш-функции не меняются.*

Задача P1. Хеш-таблица с закрытой адресацией

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Однажды Ёжик решил пригласить своих друзей на вечернее чаепитие, но, чтобы сделать вечер особенным, он придумал приготовить разные сорта варенья. В сумерках Ёжик отправился в лесное хранилище, где хранятся баночки с вареньем. У каждого из друзей Ёжика свои предпочтения, и Ёжику нужно быть уверенным, что он взял собой на чаепитие нужное количество баночек с разными вкусами.

В помощь Ёжику разработайте хеш-таблицу для систематизации лесного хранилища варенья.

Хеширование объектов-баночек варенья 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 функциональный объект стандартной библиотеки `std::hash<KeyType>`.

В случае если таблица заполнена более, чем на некоторое отсечное значение (коэффициент заполненности), выполняется перехеширование: удвоение емкости таблицы и перенос текущих объектов в новую таблицу. Коэффициент заполненности рассчитывается как отношение количества записей в хеш-таблице к количеству всех ячеек хеш-таблицы.

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 варианта по умолчанию: 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 (`size / capacity > coeff`) 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 ссылку на 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`.

Ваш класс должен состоять из узлов `Node` (тип элементов параметризуется — `KeyType`, `ValueType`) — односвязный список, содержащий объекты с одним хеш-кодом.

Структура `Node` должна содержать следующее:

1. Поле `key` — хранит ключ текущего узла (тип элемента — `KeyType`).
2. Поле `value` — хранит значение текущего узла (тип элемента — `ValueType`).
3. Поле `next` — хранит указатель на следующий узел с тем же хеш-кодом.
4. Конструктор, принимающий на вход два значения: key (тип — `KeyType`), value (тип — `ValueType`)

Формат выходных данных

Вы должны прислать код, содержащий определение вашего класса.

Рекомендуется использовать публичный интерфейс, доступный по ссылке `https://disk.yandex.ru/d/wU3C6JqNzMQesw`. Подключать этот заголовочный файл в отправляемом решении **не нужно**.

Можно добавлять свои приватные поля или методы в класс/структуру.

Система оценки

Всего будет не более $5 \cdot 10^6$ операций с таблицей.

Функция/Test	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Default Constructor	+		+	+	+	+	+	+	+	+	+	+	+	+
Constructor with Function		+									+			
Constructor with Parameters														
Destructor	+	+	+			+								
Size	+	+	+			+								
Capacity	+	+	+			+								
Insert	+	+	+			+								
Erase	+		+											
Find							+	+						
Operator []							+					+		
At										+				+

Группа	**Баллы**	**Необходимые группы**	**Дополнительная информация**	**Информация о проверке**
1	2	–	–	первая ошибка
2	7	1	–	первая ошибка
3	3	1–2	–	первая ошибка
4	3	1–3	доп. проверки	первая ошибка

Задача P2. Реализация фильтра ~~Блума~~ Рика

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Как известно, Рик и Морти очень много путешествуют. Настолько много, что уже начали забывать, в каких мирах они были, а в каких нет. Чтобы решить эту проблему, Рик попросил Морти сделать базу данных. База данных будет храниться в виде множества файлов на диске. Поиск в файлах названия мира, в котором герои уже были, может занимать довольно долго, поэтому решено оптимизировать его с помощью фильтра Блума.

СУБД и запись на диск Морти уже реализовал и просит Вас написать фильтр Блума. Помогите Морти, иначе Рик не возьмёт его на следующее приключение!

Для решения этой задачи вам предстоит реализовать Фильтр Блума, который работает со строковыми объектами. Ниже содержатся требования, которые предоставил Морти:

Фильтр должен поддерживать:

1. Добавление строки в множество.
2. Тест принадлежности строки к множеству объектов.

Помимо этого вам требуется добавить поддержку расчета доли ложно-положительных срабатываний, то есть значения, которое показывает отношение количества тестов принадлежности, которые дали ложно-положительный результат, к общему числу поступивших тестов принадлежности.

Реализуемый вами класс `BloomFilter` должен содержать следующие методы и конструкторы:

1. Конструктор, параметризуемый числом хеш-функций фильтра и числом ячеек фильтра.
2. Деструктор.
3. Метод `add`, который вставляет информацию о строке, с использованием хэш-функций, в множество. Принимает на вход значение строкового типа.
4. Метод `verify`, который проверяет, существует ли строка в Фильтре Блума.

   Принимает на вход значение типа `std::string`, возвращает значение типа `bool`.

В случае, если Фильтр Блума показывает, что строка в нём находится, но при этом она не добавлялась — необходимо инкрементировать счетчик ложно-положительных значений на единицу.

5. Метод `getFPRate`, который возвращает отношение количества ложно-положительных срабатываний к сумме всех запросов к функции `verify`.

   Тип возвращаемого значения — `double`.

6. Константный метод `numberOfHashFunctions`, который возвращает количество хеш-функций для данного фильтра.

7. Константный метод `numberOfBits`, который возвращает количество ячеек в данном фильтре.

Поскольку фильтр содержит информацию о количестве используемых хеш-функций, применяется следующий вариант получения k хеш-функций:

- используется объект стандартной библиотеки `std::hash`;
- при подсчете хеша для n-го хешера, где $n > 0$, добавляется некоторое подобие «соли»;
- в качестве «соли» выступает номер хеш-функции, приведенный к строке и добавленный в конец исходной строки.

Таким образом, например, вычисление 4 хеш-функций от строки «`abcd`» будет выглядеть следующим образом:

- `hash_0("abcd") = std::hash<std::string>{}("abcd");`
- `hash_1("abcd") = std::hash<std::string>{}("abcd1");`
- `hash_2("abcd") = std::hash<std::string>{}("abcd2");`
- `hash_3("abcd") = std::hash<std::string>{}("abcd3");`

В рамках задачи фильтр рассматривается в виде одного битового вектора.

Формат входных данных

Вам нужно загрузить определение класса `BloomFilter`.

Загрузка неполного определения/реализации класса не гарантирует полное прохождение тестов в рамках группы.

Формат выходных данных

Это задача-грейдер.

Вашей целью будет загрузить реализацию функций класса `BloomFilter` шаблоны всех функций даны. **Не требуется загружать что-то иное, кроме реализации предоставленных функций класса.**

Например, вы не можете добавить свои поля, как-то модифицировать файл `bloom_filter.h`. Тестироваться код будет с исходной версией этого файла, которая вам предоставлена.

Система оценки

Функция/Test	1	2	3	4	5	6	7	8	9	10
`default constructor`	+	+	+	+	+	+	+	+	+	+
`destructor`	+	+	+	+	+		+	+	+	+
`add`		+	+	+		+	+	+	+	+
`verify`		+	+	+		+	+	+	+	+
`getFPRate`					+	+	+	+	+	+
`numberOfHashFunctions`	+									
`numberOfBits`	+									

Группа	Баллы	Тесты	Дополнительная информация	Необходимые группы	Информация о проверке
1	2	1—2	–	–	первая ошибка
2	6	3—6	–	1	первая ошибка
3	2	7—10	–	1–2	первая ошибка
4	2	11—12	дополнительные проверки	1–3	первая ошибка

Замечание

- Используйте предложенный шаблон кода, который доступен по ссылке: `https://gist.github.com/otter18/8c55e6cc58a67471f92444c3a2b10189`.
- Посмотреть исходный код заголовочного файла можно по ссылке `https://gist.github.com/otter18/8c55e6cc58a67471f92444c3a2b10189`.
- Не разрешается добавлять свои приватные поля и функции в реализацию класса, это приведёт к ошибке компиляции.

Задача P3. Анаграммы-2

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Недавно Человек-Невидимка от нечего делать прогуливался по крышам домов и случайно подслушал интересный разговор, доносящийся из открытого окна последнего этажа. Разговаривали два человека, одного из которых звали «Нолик», а второго - «Симка». «Странные имена», - подумал Человек-Невидимка. Но для него это было неважно, намного интереснее была тема разговора - это было что-то, связанное с программированием, а он никогда не мог пройти мимо такого соблазна.

Внимательно все послушав, Человек-Невидимка понял, что суть задачи, которую обсуждали эти два странных человека, состоит в следующем: по данному массиву-шаблону и массиву-тексту надо было понять, существует ли такой подотрезок текста, совпадающий с массивом-шаблоном как анаграмма. Под анаграммами в данном случае понимались два слова, в которых можно как-то переставить буквы, чтобы они стали одинаковыми. Оценив задачу, Человек-Невидимка понял, что она для него слишком простая, поэтому он решил усложнить ее. После некоторых раздумий, ему в голову пришла следующая ее модификация: по данным двум массивам требовалось найти такое максимальное число k, что в первом и втором массивах существуют подотрезки длиной k, совпадающие как анаграммы. Но эта задача уже оказалась Человеку-Невидимке не по силам, поэтому он попросил у вас помощи в решении этой задачи.

Формат входных данных

В первой строке входного файла дано число n ($1 \leqslant n \leqslant 5 \cdot 10^3$) – длина первого массива.

Во второй строке через пробел заданы n чисел a_i ($1 \leqslant a_i \leqslant 10^6$) – первый массив.

В третьей строке входного файла дано число m ($1 \leqslant m \leqslant 5 \cdot 10^3$) – длина второго массива.

В четвертой строке через пробел заданы m чисел b_i ($1 \leqslant b_i \leqslant 10^6$) – второй массив.

Формат выходных данных

В единственной строке выведите три неотрицательных числа k, i, j – максимальная длина подотрезков, совпадающих как анаграммы, а также начало отрезка в первом массиве и во втором соответственно. Если максимальная длина подотрезка равна 0, следующие два числа в выходном файле должны равняться -1.

Система оценки

Подзадача	Баллы	Дополнительные ограничения	Необходимые подзадачи	Информация о проверке
0	0	тесты из условия	–	полная
1	1	$1 \leqslant n, m \leqslant 10$	0	первая ошибка
2	1	$10 \leqslant n, m \leqslant 100$	1	первая ошибка
3	2	$100 \leqslant n, m \leqslant 500$	1–2	первая ошибка
4	2	$500 \leqslant n, m \leqslant 1000$	1–3	первая ошибка
5	2	$1000 \leqslant n, m \leqslant 5000$	1–4	первая ошибка

Примеры

стандартный ввод	стандартный вывод
6 1 5 9 8 7 5 6 5 7 8 4 5 1	3 4 1
6 5 4 3 2 1 9 6 3 4 5 7 6 5	3 1 1