Домашнее задание № 3

Алгоритмы и структуры данных 2020

Фильтры

Авторы задания: команда курса «Алгоритмы и структуры данных 2020». **Вопросы** по заданию направлять: Терлыч Никита Андреевич naterlych@edu.hse.ru или в канал MS Teams “**Домашнее задание 3 Фильтры**”

Отправка решения

Результатом решения должен быть один файл на языке C++ (.cpp), а также файлы тестов (входные и выходные данные). Именно файл на языке C++ (.cpp) и тесты будут оцениваться. Решение нужно загрузить в LMS в проект “**Домашнее задание 3 - фильтры**” по дисциплине Алгоритмы и структуры данных 2020 уч. год Б 2 курс (код 131940, ОП: М090304ПИНЖ)
При выявлении плагиата будет выставлена оценка 0 баллов

Дедлайн. 12 ноября 2020 г. 23:59 МСК

Пожалуйста, загружайте работы заранее, чтобы не возникало технических препятствий в последний момент. Время на сервере LMS и Вашем устройстве может отличаться. Пожалуйста, настройте доступ в ЛМС (даже если Вы проходите курс по ИУП): архивы с решениями не принимаются по электронной почте.

Что нужно сделать?

1. Задание разбито на 3 уровня сложности. Выберите только один уровень сложности. Выполнение более сложной задачи не гарантирует автоматически более высокую оценку.
2. Решите задание (один файл .cpp).
3. Напишите 5 своих тестов (пары файлов входных и выходных данных) к задаче, пронумеруйте их соответственно. Разместите файлы с входными данными в директории input, а с выходными в директории output.
4. Создайте директорию с решением по шаблону <Фамилия и Имя>_<группа>_<уровень>, например TerlychNikita_161_3. Разместите внутри нее директории input и output с тестами и один файл .cpp с решением задачи. Внимание: номер выбранного Вами уровня задания находится в имени директории (в примере TerlychNikita_161_3 — это уровень 3).
5. Упакуйте корневую директорию (TerlychNikita_161_3) в zip-архив и отправьте в LMS (размер файла должен быть не более 1МБ).

Внимание: не публикуйте свое решение в открытом доступе, например на GitHub либо в других открытых источниках. Это действие будет приравниваться к плагиату и повлечет за собой оценку 0 баллов за домашнее задание (в том числе при обнаружении такового факта уже после выставления оценки).

Мотивация

Одна большая компания пригласила Вас на стажировку в новую команду опытных разработчиков. На первой встрече тимлид сказал, что Вы будете заниматься разработкой видеохостинга, как именно он назывался Вы не расслышали, но, кажется, это было что-то оканчивающееся на “-tube”. Оказывается, ваши коллеги уже успели написать плеер, разработали web-интерфейс приложения и даже развернули прототип для тестирования! Осталась одна проблема - пользователям нужно рекомендовать видео для просмотра.

Ваши коллеги долго работали над алгоритмом рекомендательной системы, который умеет подбирать именно то, что нравится пользователю, но им не хотелось бы выдавать пользователю то, что он уже видел. Всё бы ничего, но разработчики столкнулись с ограничениями по памяти и производительности. Дело в том, что на видеохостинге планируется размещение миллионов видеороликов, а ожидаемое число активных пользователей - десятки миллионов, эти пользователя каждый день смотрят видеоролики и системе необходимо как-то помнить, что они уже смотрели.

Каждое видео представлено через url-ссылку к соответствующему файлу. Ваши коллеги пробовали хранить список просмотренных видео, но поиск по списку оказался очень долгим; пробовали использовать хэш-таблицы, они работали быстро, но занимали слишком много места. Придумать алгоритм, который найдёт компромисс между памятью и временем поручили именно Вам.

“Как же хорошо, что у меня был курс Алгоритмы и структуры данных” - думаете Вы, ведь Вы сразу вспоминаете Фильтр Блума (или Кукушкин Фильтр).

Уточнения: id видео (== url-ссылка) и id пользователя - это последовательности символов, состоящие из латинских букв (заглавных и строчных) и цифр, длиной от 5 до 15 символов.

Задание

Настало время написать модуль для системы рекомендаций видео. Ваш модуль должен уметь реагировать на действия системы и хранить просмотренные видео пользователя в Фильтре Блума, далее представлен полный список таких действий:

- videos <X> - всегда первая команда, встречающаяся ровно один раз, включающая актуальную информацию о количестве видео в системе, 0 < X < 10^7;
- watch <id пользователя> <id видео> - просмотр пользователем видео, каждое такое видео нужно пометить в фильтре для соответствующего пользователя;
- check <id пользователя> <id видео> - проверка, смотрел ли пользователь данное видео, на каждый такой вопрос нужно ответить “Probably” или “No”.

Входные данные содержат список команд - **N** строк (0 < **N** < 10^6), каждая из которых содержит одну из команд, описанных выше.

Выходные данные:
На все команды, **кроме check**, нужно ответить **Ok**.
На команду **check** нужно ответить **Probably**, если пользователь вероятно смотрел видео, или **No**, если пользователь точно не смотрел видео.

Пример

Входные данные	Выходные
videos 6 watch user1 video1 watch user1 video1 watch user1 video2 watch user2 video1 check user1 video1 check user1 video2 check user1 video3 check user2 video2 check user3 video4	Ok Ok Ok Ok Ok Probably Probably Probably No No

Пояснение к примеру.
На хостинге 6 видео (*это число N из параметров фильтра*).
Пользователь *user1* два раза посмотрел видео *video1* и один раз видео video2.
Затем пользователь *user2* посмотрел видео *video1*.
В конце следует череда проверок для фильтра блума от рекомендательной системы, на которые возвращается ответ на вопрос “Смотрел ли указанный пользователь данное видео?”

Уровни оценивания

Уровень 1. Максимум 5 баллов из 10

Для прохождения первого уровня достаточно скачать существующую библиотеку, реализующую Фильтр Блума, и с её помощью решить задачу.
Пример библиотеки: https://www.partow.net/programming/bloomfilter/index.html

Уровень 2. Максимум 8 баллов из 10:
7 баллов за фильтр + 1 балл за написание своих хэш-функций

На втором уровне требуется реализовать свой собственный Фильтр Блума.
Для этого нужно будет использовать следующие параметры:

Параметр	Описание	Формула	Значение
n	Количество элементов, которые необходимо хранить	-	из теста
fpr	Доля ошибки, когда фильтр для элемента выдаёт “Probably”, но этого элемента в множестве нет	e^(-(m/n) * (ln(2)^2))	0.01
m	Количество ячеек в битмапе фильтра	-(n * ln(fpr)) / (ln(2)^2)	round(n * 9.585)
k	Количество хэш-функций фильтра	(m/n) * ln(2)	3

Используйте значения параметров, указанные в столбце “Значение” таблицы.
Про оптимальные параметры можно также почитать здесь:
https://www.di-mgt.com.au/bloom-filter.html

Таким образом, Вам понадобится k == 3 различные хэш-функции. Можно сделать вот так:

1. hash из стандартной библиотеки C++:
   https://en.cppreference.com/w/cpp/utility/hash
2. Написать свою функцию для получения хэша строки, например, как здесь: https://e-maxx.ru/algo/string_hashes
3. Написать ещё одну (можно модифицировать предыдущую)

Также допустимо использование любых других библиотек, реализующих хэш-функции, которые вы найдёте на просторах Интернет.

Уровень 3. Максимум 10 баллов из 10

Требуется решить задачу используя Кукушкин Фильтр (написать свою собственную реализацию).
Статья с описанием фильтра прикреплена в архиве (“**cuckoo-conext2014.pdf**”), библиографическое название в списке литературы.

У Вашего Кукушкиного Фильтра будут следующие параметры:

Arg	Описание	Вывод значения	Значение
n	Количество элементов, которые необходимо хранить	Задается по условию	из теста
fpr	Доля ошибки, когда фильтр для элемента выдаёт “Probably”, но этого элемента в множестве нет	Задается в соответствии с Вашими требованиями	0.06
m	Количество групп элементов в фильтре	m = (1 + fpr) * n	1.06 * n
b	Количество элементов в одной группе фильтра	Эксперименты показали, что a = 0.95 при b = 4 это оптимально	**4**
a	Загруженность таблицы		0.95
f	Длина “fingerprint” в битах	f >= log2(2b/fpr)	**7 бит**

Итак, у каждого элемента **2** потенциальные группы, в одной группе **b == 4** записи, а одна запись в группе это **f == 7** бит.

Этот фильтр требует всего две хэш функции. Одна будет использована для вычисления *fingerprint* (хэша-последовательности из f бит) от элемента, вторая будет применяться для вычисления хэша и *fingerprint’ов* и элементов. Эти хэш-функции можно получить следующим образом:

- hash() из стандартной библиотеки C++: https://en.cppreference.com/w/cpp/utility/hash;
- написать свою функцию для получения хэша от байта/массива байт, например, как здесь: https://e-maxx.ru/algo/string_hashes;
- найти библиотеки, реализующие хэш-функции, на просторах Интернет.

Помните, что *fingerprint* это последовательность (из семи) битов, которую можно получить посредством хэш-функции. Пример: https://github.com/efficient/cuckoofilter/blob/master/src/cuckoofilter.h

Как будет оцениваться работа?

Решения будут проверяться на наборах тестов, каждый тест — это два файла, один для входных данных, другой для выходных. Таким образом, ввод и вывод данных происходит через файлы.

Будут оценены: корректность кода, составленные тесты (граничные условия, сложные случаи и т.п.), декомпозиция, аккуратность кода, дополнительные условия для каждого уровня сложности (напр., поощряется качественная реализация собственных хэш-функций). Возможны устные собеседования при сомнении в самостоятельности выполнения работы.

В приложенных к условию задачи директориях расположено по 5 тестов к каждой вариации алгоритма, на которых можно проверить работоспособность решения, однако стоит учитывать, что прохождение приложенных тестов не гарантирует полное отсутствие ошибок в решении (решения будут также проверяться и на других, расширенных тестах). Файлы входных данных именуются по принципу 'test<номер теста>.txt', файлы выходных данных (или ответов) именуются 'answer<номер теста>.txt'. Названия файлов для входных данных и выходных данных должны быть указаны через командную строку. Кроме того, в параметрах командной строки могут указываться не только имена файлов и директорий, но и относительные/абсолютные пути к ним. Например, так будет выглядеть тестирование программы TerlychNikita_161_3.cpp:

```
cl TerlychNikita_161_3.cpp
TerlychNikita_161_3 test1.txt answer1.txt
```

либо

```
TerlychNikita_161_3 path/to/test1.txt path/to/answer1.txt
```

где

1-ый параметр – путь к входному файлу,

2-ий параметр – путь к выходному файлу.

(пути могут быть абсолютными и относительными)

Многие примеры кода на C++, в том числе чтение файлов и использования аргументов командной строки, можно найти в репозитории семинарских занятий: https://github.com/b1nd/cpp

Ограничения

При реализации собственных фильтров запрещается использовать реализованные за Вас контейнеры (std::vector, std::bitset, std::array и прочие) для создания bitmaps. Вам необходимо создать свой контейнер для побитовой работы с bitmaps фильтра.

Список Литературы

1. **Готовая библиотека для Фильтра Блума**
   https://www.partow.net/programming/bloomfilter/index.html
2. **Описание параметров Фильтра Блума**
   https://www.di-mgt.com.au/bloom-filter.html
3. **Калькулятор параметров Фильтра Блума**
   https://www.di-mgt.com.au/bloom-calculator.html
4. **std::hash()** https://en.cppreference.com/w/cpp/utility/hash
5. **Написание string hash** https://e-maxx.ru/algo/string_hashes
6. **Семинарский репозиторий с примерами кода**
   https://github.com/b1nd/cpp
7. **Статья по Кукушкиному Фильтру**
   Fan B. et al. Cuckoo filter: Practically better than bloom //Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. – 2014. – С. 75-88.