Задача. Архиватор логов

Неотъемлемая часть любого промышленного приложения — это его логи. Туда попадают записи о всех важных событиях в жизни приложения, например ошибках, скорости и результатах выполнения пользовательских запросов. Зачастую чтение логов — единственный способ разобраться с внезапно возникшей в продакшене неведомой проблемой, или посчитать какую-то статистику о работе приложения.

За день приложение под хорошей нагрузкой может генерировать десятки гигабайт логов. Все эти логи хочется хранить вечно, а значит, их надо сжимать.

Обычно для этих целей используются алгоритмы сжатия общего назначения — например, gzip. Но можно ли сжимать лучше? Наверное да — ведь логи имеют специфичный формат записей и множество повторяющихся элементов. Ваша задача — продемонстрировать, что это возможно.

Задача

Разработать приложение для архивации логов в заданном формате (см. конец этого документа). Это должно быть консольное приложение, имеющее два режима работы: сжатие и разжатие.

При запуске с двумя аргументами приложение должно запускаться в режиме сжатия: dotnet LogPacker.dll <input_file> <output_file> Первый аргумент задает путь до исходного лог-файла. Второй аргумент задает путь до файла, куда вы запишете сжатые данные.

При запуске с тремя аргументами, где первый равен -d, приложение должно запускаться в режиме разжатия: dotnet Kontur.LogPacker.dll -d <input_file> <output_file> Здесь второй аргумент задает путь до сжатого файла, сформированного вашим приложением, а третий аргумент — путь до разжатого файла.

Требования и ограничения

Внимание! Чтобы соблюсти все базовые требования к решению, рекомендуем воспользоваться
шаблоном: https://github.com/DQKrait/Kontur.LogPacker/tree/master Там также есть набор простейших тестов и утилита для запаковки решения в архив перед отправкой на проверку.

Внимание! Для выполнения задания вам понадобится .NET Core SDK 2.1 и среда разработки Visual Studio 2017 или JetBrains Rider. Пробную версию JetBrains Rider можно скачать тут: https://www.jetbrains.com/rider/

Функциональные требования: После цикла сжатие-разжатие файл должен быть побайтово равен исходному. Алгоритм должен корректно работать на логах, где строки имеют другой формат, а также на данных, вообще не являющихся логами (в общем случае это просто случайные бинарные данные). Требований по качеству сжатия в таком случае нет, но требования по скорости сжатия сохраняются. Качество сжатия: на произвольных лог-файлах заданного формата размер сжатых данных должен быть меньше, чем при использовании gzip (здесь и далее имеется в

виду стандартная реализация из .NET Core: GZipStream с CompressionLevel.Optimal). Ваш архиватор должен выигрывать хотя бы 1 процентный пункт (если gzip сжимает файл до 25% исходного размера, у вас должно получаться 24% или ещё меньше). Скорость сжатия: на любых данных время работы вашего алгоритма не должно превышать время работы gzip более чем в два раза. Аналогично со скоростью разжатия. Потребление оперативной памяти: объем потребляемой оперативной памяти не должен зависеть от размера входных данных и на любых входных данных не должен выходить за разумные пределы (100-200 МБ).
Загрузка процессора: архивация логов не должна мешать другим работающим на машине приложениям. Поэтому ваш архиватор должен работать в один поток. Решение может создавать временные файлы, но их суммарный размер не должен превышать размер входного файла, и они должны удаляться до завершения процесса архиватора. Все временные файлы должны находиться в рабочей директории приложения (т. е. если вы будете указывать только имя файла, без директории, всё сработает как надо).

Требования к оформлению: Решение должно представлять собой консольное приложение на .NET Core 2.1. Нельзя использовать внешние библиотеки и приложения, как из nuget, так и подключаемые вручную. Решение должно быть запаковано в zip-архив с фиксированной структурой папок. Чтобы не ошибиться, рекомендуем воспользоваться утилитой Kontur.LogPacker.SubmitHelper из проекта с шаблоном.

Проверка решений

Все решения будут проходить следующие этапы проверки: Тесты, аналогичные тестам из шаблона решения. Если какой-то из этих тестов не пройдет, решение не будет рассматриваться дальше. Замеры степени сжатия и скорости на различных логах от реальных приложений. Код-ревью и окончательный отбор. На третий этап решения будут попадать в порядке результатов второго этапа, от лучших к худшим. Это означает, что чем лучше ваше решение относительно других, тем раньше оно будет рассмотрено и тем больше вероятность попасть в квоту мест на стажировку. Если будут сомнения, какое из одинаково хорошо написанных решений выбрать (а такая ситуация обязательно возникнет), будет выбрано более оптимальное по качеству сжатия.

Как пройти код-ревью?

Позаботьтесь о чистоте кода. Перед отправкой лучше свежим взглядом окинуть свой код и убедиться, что там всё красиво: четкая архитектура, понятные названия, нет неиспользуемого/закомментированного кода, и так далее.

Если в вашем решении применяются сложные алгоритмы, их код тоже должен быть читаемым: чтобы непосвященный человек глядя на него смог понять, как это работает.

Помните, что подключать внешние библиотеки нельзя. Если вам захочется использовать алгоритм сжатия общего назначения, не входящий в состав .NET Core 2.1, придется реализовать его самостоятельно. Качество кода здесь не должно быть хуже, чем в остальных частях решения.

Формат логов

Вот маленький фрагмент лог-файла в нужном формате:

2018-11-13 00:02:40,574 76 INFO Configured ThreadPool: 1024/32767 workers, 1024/32767 IOCP (min/max). 2018-11-13 00:02:41,315 816 INFO [BackgroundDaemon] Starting iteration #15434. 2018-11-13 00:02:41,330 831 INFO [0de1db] Started processing user request. Id = 0de1dbfb-fddb-40b5-b34b-7a7beb43a33f. Size = 2.83 KB. 2018-11-13 00:02:41,344 845 INFO [0de1db] Doing some complicated stuff.. Random numbers are: 1192743271, 187325574, 168764164. 2018-11-13 00:02:41,345 846 INFO [3a61cc] Started processing user request. Id = 3a61cc3f-2ee9-4738-8283-efc073235ec8. Size = 67.46 KB. 2018-11-13 00:02:41,354 855 INFO [3a61cc] Doing some complicated stuff.. Random numbers are: 905206199, 2002603058, 533352593. 2018-11-13 00:02:41,373 875 INFO [0de1db] Finished processing user request with id = 0de1dbfb-fddb-40b5-b34b-7a7beb43a33f in 00:00:00.0442952. ... (файл целиком можно найти в репозитории с шаблоном решения)

Число перед уровнем логирования может быть большим, тогда строки будут выглядеть так: 2018-12-10 00:00:03,245 3605942787 INFO ...

Формальное описание формата корректной строки лога: [дата в формате 'yyyy-MM-dd HH:mm:ss,fff'][пробел][число из полуинтервала [0, ulong.MaxValue), дополненное пробелами справа до 6 символов][пробел][строка, дополненная пробелами справа до 5 символов][пробел][строка]

- дополнение пробелами справа работает аналогично методу string.PadRight()

Логи всегда пишутся в кодировке utf-8.

Ваш алгоритм должен хорошо работать на логах именно такого формата. Если возникнут вопросы по поводу формата (или какие-нибудь ещё вопросы), их можно задать в чате. В чате также опубликован и обновляется список популярных ошибок в решениях. Загляните туда перед отправкой!