Сжатие хранимых даннных методом блочной дедупликации многомерной регрессией

juhnowski@gmail.com

20 июня 2025 г.

1 ПРОБЛЕМАТИКА

Темпы роста объема данных (файлов на ФС) dS_{DB} превышают планы по закупкам вычислительнных мощностей организаций dS_{Plan} из-за медленного, чем ожидалось снижения стоимости оборудования:

$$\frac{dS_{DB}}{dt} \geq \frac{dS_{Plan}}{dt} \tag{1}$$

Применение фуннкции сжатия баз данных $C(S)$ является одним из решений этой проблемы - уменьшением хранимого размера БД S_{DB}:

$$C(S_{DB}) \leq S_{DB} \tag{2}$$

, настолько, что:

$$\frac{dC(S_{DB})}{dt} \leq \frac{dS_{Plan}}{dt} \tag{3}$$

Для хранения баз данных, помимо экономии места, сжатие сокращает количество страниц P_{FS}, на которых размещаются данные, что помогает:

- сократить количество дисковых операций ввода-вывода N_{Disk}

$$\frac{dN_{Disk}(C(S_{DB})}{dt} \leq \frac{dN_{Disk}(S_{DB})}{dt} \tag{4}$$

- повысить производительность

$$\frac{dP_{FS}(C(S_{DB})}{dt} \leq \frac{dP_{FS}(S_{DB})}{dt} \tag{5}$$

Для распределенных баз данных, реплицированных по географическим регионам, также существует острая необходимость в сокращении объема передачи данных S_{Net}, используемого для синхронизации реплик:

$$\frac{dS_{Net}(C(S_{DB})}{dt} \leq \frac{dS_{Net}(S_{DB})}{dt} \tag{6}$$

Наиболее широко используемый подход к сокращению объема данных в операционных СУБД — это сжатие на уровне блоков [1], [2] .

Сейчас СХД, например Huawei OceanStor `https://www.huawei-networks.ru/catalog/oceanstor-dorado-v6` предлагает:

- В тестах с виртуализированными средами (VMware vSphere) коэффициент дедупликации достигает 5:1 — 10:1 , сжатие — 2:1 — 3:1 .
- В тестах с базами данных (например, Oracle, SQL Server) коэффициент дедупликации ниже (около 2:1 – 3:1), но сжатие сохраняется на уровне 2:1 – 3:1 .
- Для неструктурированных данных (например, видеофайлов) коэффициент дедупликации ниже 2:1.

Поэтому, существующими аппаратнными или программно-аппаратнными комплексами проблема не решается.

Конечно, многое зависит от природы храннимых данных, но не все.

В телекоме большую часть данных заимает хранение cdr, расшифрованного трафика для СОРМ.

В других областях, источник больших данных - платформа данных, хранящая структурированные OLTP и OLAP данные.

Все эти данные плохо дедуплицируются на СХД, так как минимальнный размер блока для дедупликации - 512, а скорее всего это 4K или 8K. БД на таких блоках будут вносить много измененний, которые тяжело дедуплицировать средствами СХД.

Дедупликация на уровне ИС, поставляющих данные сопряжена с высокими рисками и высокой трудоемкостью настолько, что не позволяет рассматривать как вариант решениня проблемы.

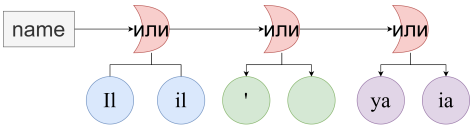
1.1 Эксперимент 1

Рассмотрим нормальзованные данные:

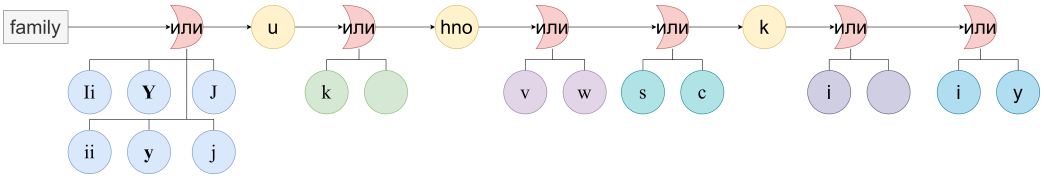
- Варианнты написания имени - `db/initial/name.dat` размер - 50Б
- Варианты написания фамилии - `db/initial/family.dat` размер - 4670Б

Те же данные в графовом представлении, используя DSL:

- Варианнты написания имени - `db/graph/name.grd` размер - 29Б



- Варианты написания фамилии - `db/graph/family.grd` размер - 85Б



Сожмем файлы и посмотрим на размер данных после сжатия:

```
./src/utils/gzip/run.bat
```

Сожмем построчно текстовые файлы:

```
./src/utils/gzip_rows/run.bat
```

Размеры файлов приведены в таблице 1

Из результатов видно, что сжатие на уровне блоков (построчно) не решает проблему избыточности между блоками и, следовательно, оставляет значительные возможности для улучшения сжатия, например с помощью графов. В нашем случае, графы построены по байтно, поэтому сжатие не дает никаких результатов. Графы дедуплицируют данные, поэтому мы получаем максимальнный коэффициент сжатия - 55,59.

Дедупликация стала популярной в системах резервного копирования для устранения дублирующегося контента во всем корпусе данных, часто достигая гораздо более высоких коэффициентов сжатия. Поток резервного копирования делится на фрагменты, и в качестве идентификатора каждого фрагмента используется устойчивый к коллизиям хэш (например, SHA-1). Система дедупликации поддерживает глобальный индекс

Представление Даные	name	family	K_{name}	K_{family}
Текст (не сжатые)	50	4670	-	-
Текст (gzip файл)	34	813	1,47	5,74
Текст (gzip построчо)	108	7192	0,46	0,65
Граф (не сжатые)	29	85	1,72	54,94
Граф (gzip файл)	32	84	1,56	55,59
Граф (gzip построчно)	73	196	0,69	23,83

Таблица 1: Размер данных, где K_{name}, K_{family} - коэффициеннты сжатия

всех хэшей и использует его для обнаружения дубликатов. Дедупликация хорошо работает как для основных, так и для резервных наборов данных, которые состоят из больших файлов, которые редко изменяются (а если и изменяются, то изменения редки).

К сожалению, традиционные схемы дедупликации на основе фрагментов не подходят для операционных СУБД, где приложения выполняют запросы на обновление, которые изменяют отдельные записи. Количество дублирующихся данных в отдельной записи, скорее всего, незначительно. Но большие размеры фрагментов (например, 4–8 КБ) являются нормой, чтобы избежать огромных индексов в памяти и большого количества чтений с диска.

Рассмотрим дедупликацию на основе сходства [3] для сжатия отдельных записей OLTP баз данных.

Вместо индексации каждого хеша фрагмента, алгоритм выбирает небольшое подмножество хешей фрагментов для каждой новой записи базы данных, а затем использует этот образец для идентификации похожей записи в базе данных.

Затем он использует дельта-сжатие на уровне байтов для двух записей, чтобы уменьшить как используемое онлайн-хранилище, так и пропускную способность удаленной репликации. dbDedup обеспечивает более высокие коэффициенты сжатия с меньшими накладными расходами памяти, чем дедупликация на основе фрагментов, и хорошо сочетается со сжатием на уровне блоков, как показано на 1.

Авторы объединили несколько методов для достижения этой эффективности:

- двустороннее кодирование для эффективной передачи закодированных новых записей (прямое кодирование) в удаленные реплики,
- сохраняя новые записи с закодированными формами выбранных

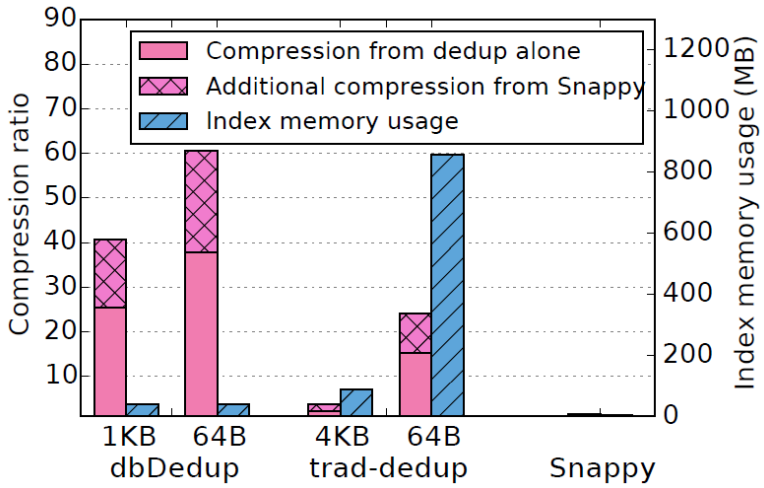


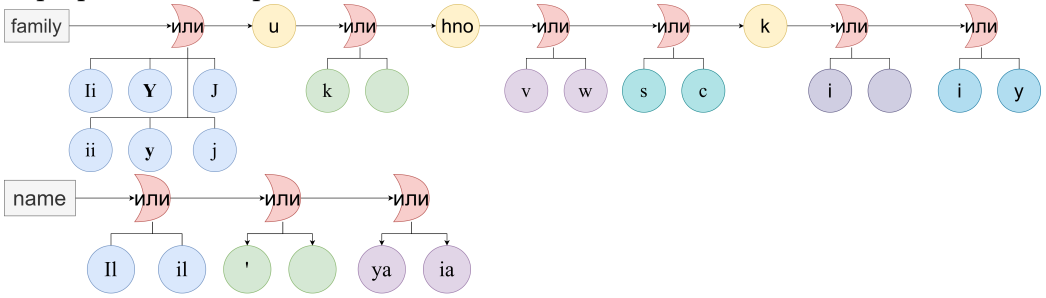
Рис. 1: Коэффициент сжатия и использование памяти индекса для данных Википедии, хранящихся в пяти конфигурациях MongoDB: с dbDedup (размер фрагмента 1 КБ и 64 Б), с традиционной дедупликацией (4 КБ и 64 Б) и с Snappy (сжатие на уровне блоков). dbDedup обеспечивает более высокую степень сжатия и меньшие накладные расходы на память индекса, чем традиционная дедупликация. Snappy обеспечивает такое же сжатие 1,6 для данных после дедупликации или исходных данных. [4]

исходных записей (обратное кодирование).

2 ПРЕДЛОЖЕНИЕ

Предлагается модифицировать алгоритм блочной дедупликации данных с последующим сжатием, описаный в [4]

Но внести изменения - с помощью многомерной регресиии сворачивать различия похожих блоков в графы и хранить дельты в виде сжатых графовых векторов.



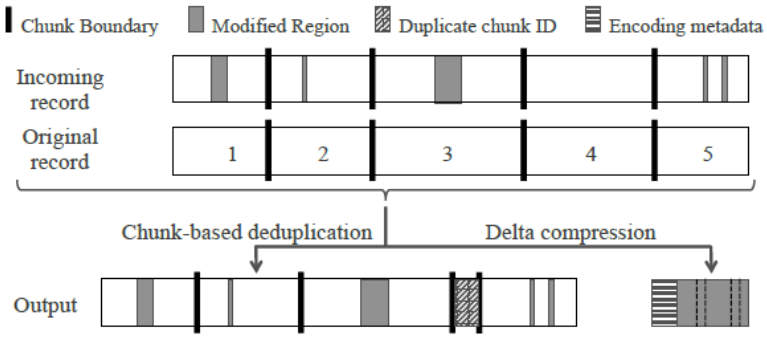
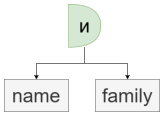


Figure 2: Comparison between chunk-based deduplication and similarity-based deduplication using delta compression for typical database workloads with small and dispersed modifications.

Рис. 2: Алгоритм блочной дедупликации, описаный в [4]



Данный подход позволит сжимать данные с коэффициенттом более 50, что существенно сэкономит потребность в закупке нового оборудования.

3 ЗАКЛЮЧЕНИЕ

Использование ИИ в задачах, казавшихся достигших предела технических возможностей, приносит качественнные улучшения. Например, RoCE дало не только увеличение скорости передачи до 600GB, но еще и packetless network. Такой же эффект ожидается и от внедрения ИИ в алгоритмы дедупликации.

Для Заказчика это будет иметь коммерческий эффект в виде сокращения стоимости владениня хранилищ даных в десятки раз (до 60) и синжениня планов на закупки нового дорогостоящего оборудования - СХД, сетевая инфраструктура.

4 ПРИЛОЖЕНИЕ

4.1 db/initial/name.dat - 50Б

```
Ilia
Ilya
```

```
Il'ya
Il'ia
ilia
ilya
il'ya
il'ia
```

4.2 db/initial/family.dat - 7192Б

```
Juhnovckiy
Juchnovckiy
Jukhnovckiy
Juhnovskiy
Juchnovsckiy
Jukhnovskiy
Juhnowckiy
Juchnwvckiy
Jukhnowckiy
Juhnowskiy
Juchnowsckiy
Jukhnowskiy
Juhnovcki
Juchnovcki
Jukhnovcki
Juhnovski
Juchnovscki
Jukhnovski
Juhnowcki
Juchnwvcki
Jukhnowcki
Juhnowski
Juchnowscki
Jukhnowski
Juhnovcky
Juchnovcky
Jukhnovcky
Juhnovsky
Juchnovscky
Jukhnovsky
Juhnowcky
Juchnwvcky
```

Jukhnowcky
Juhnowsky
Juchnowscky
Jukhnowsky
Juhnovckyi
Juchnovckyi
Jukhnovckyi
Juhnovskyi
Juchnovsckyi
Jukhnovskyi
Juhnowckyi
Juchnwvckyi
Jukhnowckyi
Juhnowskyi
Juchnowsckyi
Jukhnowskyi
Juhnovckyii
Juchnovckyii
Jukhnovckyii
Juhnovskyii
Juchnovsckyii
Jukhnovskyii
Juhnowckyii
Juchnwvckyii
Jukhnowckyii
Juhnowskyii
Juchnowsckyii
Jukhnowskyii
Yuhnovckiy
Yuchnovckiy
Yukhnovckiy
Yuhnovskiy
Yuchnovsckiy
Yukhnovskiy
Yuhnowckiy
Yuchnwvckiy
Yukhnowckiy
Yuhnowskiy
Yuchnowsckiy
Yukhnowskiy
Yuhnovcki

Yuchnovcki
Yukhnovcki
Yuhnovski
Yuchnovscki
Yukhnovski
Yuhnowcki
Yuchnwvcki
Yukhnowcki
Yuhnowski
Yuchnowscki
Yukhnowski
Yuhnovcky
Yuchnovcky
Yukhnovcky
Yuhnovsky
Yuchnovscky
Yukhnovsky
Yuhnowcky
Yuchnwvcky
Yukhnowcky
Yuhnowsky
Yuchnowscky
Yukhnowsky
Yuhnovckyi
Yuchnovckyi
Yukhnovckyi
Yuhnovskyi
Yuchnovsckyi
Yukhnovskyi
Yuhnowckyi
Yuchnwvckyi
YYukhnowckyi
Yuhnowskyi
Yuchnowsckyi
Yukhnowskyi
Yuhnovckyii
Yuchnovckyii
Yukhnovckyii
Yuhnovskyii
Yuchnovsckyii
Yukhnovskyii

Yuhnowckyii
Yuchnwvckyii
Yukhnowckyii
Yuhnowskyii
Yuchnowsckyii
Yukhnowskyii
Iiuhnovckiy
Iiuchnovckiy
Iiukhnovckiy
Iiuhnovskiy
Iiuchnovsckiy
Iiukhnovskiy
Iiuhnowckiy
Iiuchnwvckiy
Iiukhnowckiy
Iiuhnowskiy
Iiuchnowsckiy
Iiukhnowskiy
Iiuhnovcki
Iiuchnovcki
Iiukhnovcki
Iiuhnovski
Iiuchnovscki
Iiukhnovski
Iiuhnowcki
Iiuchnwvcki
Iiukhnowcki
Iiuhnowski
Iiuchnowscki
Iiukhnowski
Iiuhnovcky
Iiuchnovcky
Iiukhnovcky
Iiuhnovsky
Iiuchnovscky
Iiukhnovsky
Iiuhnowcky
Iiuchnwvcky
Iiukhnowcky
Iiuhnowsky
Iiuchnowscky

Iiukhnowsky
Iiuhnovckyi
Iiuchnovckyi
Iiukhnovckyi
Iiuhnovskyi
Iiuchnovsckyi
Iiukhnovskyi
Iiuhnowckyi
Iiuchnwvckyi
IiYukhnowckyi
Iiuhnowskyi
Iiuchnowsckyi
Iiukhnowskyi
Iiuhnovckyii
Iiuchnovckyii
Iiukhnovckyii
Iiuhnovskyii
Iiuchnovsckyii
Iiukhnovskyii
Iiuhnowckyii
Iiuchnwvckyii
Iiukhnowckyii
Iiuhnowskyii
Iiuchnowsckyii
Iiukhnowskyii
juhnovckiy
juchnovckiy
jukhnovckiy
juhnovskiy
juchnovsckiy
jukhnovskiy
juhnowckiy
juchnwvckiy
jukhnowckiy
juhnowskiy
juchnowsckiy
jukhnowskiy
juhnovcki
juchnovcki
jukhnovcki
juhnovski

juchnovscki
jukhnovski
juhnowcki
juchnwvcki
jukhnowcki
juhnowski
juchnowscki
jukhnowski
juhnovcky
juchnovcky
jukhnovcky
juhnovsky
juchnovscky
jukhnovsky
juhnowcky
juchnwvcky
jukhnowcky
juhnowsky
juchnowscky
jukhnowsky
juhnovckyi
juchnovckyi
jukhnovckyi
juhnovskyi
juchnovsckyi
jukhnovskyi
juhnowckyi
juchnwvckyi
jukhnowckyi
juhnowskyi
juchnowsckyi
jukhnowskyi
juhnovckyii
juchnovckyii
jukhnovckyii
juhnovskyii
juchnovsckyii
jukhnovskyii
juhnowckyii
juchnwvckyii
jukhnowckyii

juhnowskyii
juchnowsckyii
jukhnowskyii
yuhnovckiy
yuchnovckiy
yukhnovckiy
yuhnovskiy
yuchnovsckiy
yukhnovskiy
yuhnowckiy
yuchnwvckiy
yukhnowckiy
yuhnowskiy
yuchnowsckiy
yukhnowskiy
yuhnovcki
yuchnovcki
yukhnovcki
yuhnovski
yuchnovscki
yukhnovski
yuhnowcki
yuchnwvcki
yukhnowcki
yuhnowski
yuchnowscki
yukhnowski
yuhnovcky
yuchnovcky
yukhnovcky
yuhnovsky
yuchnovscky
yukhnovsky
yuhnowcky
yuchnwvcky
yukhnowcky
yuhnowsky
yuchnowscky
yukhnowsky
yuhnovckyi
yuchnovckyi

yukhnovckyi
yuhnovskyi
yuchnovsckyi
yukhnovskyi
yuhnowckyi
yuchnwvckyi
yyukhnowckyi
yuhnowskyi
yuchnowsckyi
yukhnowskyi
yuhnovckyii
yuchnovckyii
yukhnovckyii
yuhnovskyii
yuchnovsckyii
yukhnovskyii
yuhnowckyii
yuchnwvckyii
yukhnowckyii
yuhnowskyii
yuchnowsckyii
yukhnowskyii
iiuhnovckiy
iiuchnovckiy
iiukhnovckiy
iiuhnovskiy
iiuchnovsckiy
iiukhnovskiy
iiuhnowckiy
iiuchnwvckiy
iiukhnowckiy
iiuhnowskiy
iiuchnowsckiy
iiukhnowskiy
iiuhnovcki
iiuchnovcki
iiukhnovcki
iiuhnovski
iiuchnovscki
iiukhnovski
iiuhnowcki

iiuchnwvcki
iiukhnowcki
iiuhnowski
iiuchnowscki
iiukhnowski
iiuhnovcky
iiuchnovcky
iiukhnovcky
iiuhnovsky
iiuchnovscky
iiukhnovsky
iiuhnowcky
iiuchnwvcky
iiukhnowcky
iiuhnowsky
iiuchnowscky
iiukhnowsky
iiuhnovckyi
iiuchnovckyi
iiukhnovckyi
iiuhnovskyi
iiuchnovsckyi
iiukhnovskyi
iiuhnowckyi
iiuchnwvckyi
iiyukhnowckyi
iiuhnowskyi
iiuchnowsckyi
iiukhnowskyi
iiuhnovckyii
iiuchnovckyii
iiukhnovckyii
iiuhnovskyii
iiuchnovsckyii
iiukhnovskyii
iiuhnowckyii
iiuchnwvckyii
iiukhnowckyii
iiuhnowskyii
iiuchnowsckyii
iiukhnowskyii

4.3 db/graph/name.grd - 29Б

```
2|3&4|&5|6
Il
Il
,
ya
ia
```

4.4 db/graph/family.grd - 85Б

```
2|-7&8&k|&10&11|12&13|14&15&16|&16|6
Ii
Y
J
ii
y
j
u
k
hno
v
w
s
c
k
i
```

4.5 src/utils/gzip/Compress.java

```
import java.io.ByteArrayOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.zip.Deflater;

public class Compress {
    public static void main(String[] args) {
        System.out.println(args[0]);
        byte[] input;
```

```
        try {
            input = Files.readAllBytes(Paths.get(args[0]));
            byte[] compressed = deflate(input); // используем тот же метод defla

            try (FileOutputStream stream = new FileOutputStream(args[0]+".gzip")
                stream.write(compressed);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Метод для сжатия байтового массива с использованием Deflater
     */
    private static byte[] deflate(byte[] data) {
        Deflater deflater = new Deflater();
        deflater.setInput(data);
        deflater.finish();

        ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.leng
        byte[] buffer = new byte[1024];
        while (!deflater.finished()) {
            int count = deflater.deflate(buffer);
            outputStream.write(buffer, 0, count);
        }
        try {
            outputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return outputStream.toByteArray();
    }
}
```

4.6 src/utils/gzip/run.bat

```
javac Compress.java
jar -cf Compress.jar Compress.class
java -cp Compress.jar Compress "..\..\..\db\graph\family.grd"
java -cp Compress.jar Compress "..\..\..\db\graph\name.grd"
```

```
java -cp Compress.jar Compress "..\..\..\db\initial\family.dat"
java -cp Compress.jar Compress "..\..\..\db\initial\name.dat"
```

4.7 CompressRows.java

```
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.zip.Deflater;

public class CompressRows {
    public static void main(String[] args) {

       // Используем BufferedReader для чтения файла построчно
        try (BufferedReader reader = new BufferedReader(
                new InputStreamReader(new FileInputStream(args[0]), StandardChar
             FileOutputStream stream = new FileOutputStream(args[0]+".rgz");
            ) {
            String line;

            // Читаем файл построчно
            while ((line = reader.readLine()) != null) {
                // Преобразуем строку в массив байтов
                byte[] lineBytes = line.getBytes(StandardCharsets.UTF_8);
                byte[] compressed = deflate(lineBytes); // используем тот же мет
                stream.write(compressed);
                stream.write('\n');
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Метод для сжатия байтового массива с использованием Deflater
     */
```

```
    private static byte[] deflate(byte[] data) {
        Deflater deflater = new Deflater();
        deflater.setInput(data);
        deflater.finish();

        ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.leng
        byte[] buffer = new byte[1024];
        while (!deflater.finished()) {
            int count = deflater.deflate(buffer);
            outputStream.write(buffer, 0, count);
        }
        try {
            outputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return outputStream.toByteArray();
    }
}
```

4.8 src/utils/gzip-rows/run.bat

```
javac CompressRows.java
jar -cf CompressRows.jar CompressRows.class
java -cp CompressRows.jar CompressRows "..\..\..\db\graph\family.grd"
java -cp CompressRows.jar CompressRows "..\..\..\db\graph\name.grd"
java -cp CompressRows.jar CompressRows "..\..\..\db\initial\family.dat"
java -cp CompressRows.jar CompressRows "..\..\..\db\initial\name.dat"
```

Список литературы

[1] G. V. Cormack. Data compression on a database system. Communications of the ACM, 28(12):1336-1342, 1985.

[2] B. Iyer and D. Wilhite. Data compression support in databases. 1994.

[3] L. Xu, A. Pavlo, S. Sengupa, J. Li, and G. R. Ganger. Reducing replication bandwidth for distributed document databases. In SoCC, pages 222–235, 2015.

[4] Xu, L., Pavlo, A., Sengupta, S., and Ganger, G. R.(2017). Online Deduplication for Databases. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD 17) (pp.1355-1368). ACM Digital Library