Описание основных структур

В данном разделе приведены описания, структуры и краткие описания базовых методов для основных элементов библиотеки.

2.3.1 Структура для работы с файлами

Поскольку библиотека mrjob позволяет использовать как локальные файлы, так и файлы из DFS, то для удобства дальнейшей разработки необходим общий интерфейс, который нивелирует различия при обращении к файлам.

На рисунке 2.3.1.1 приведена UML-диаграмма данной структуры.

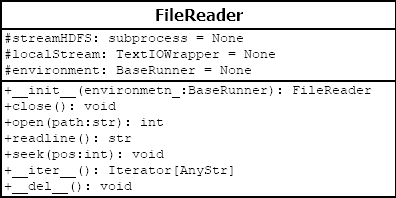


Рисунок 2.3.1.1 – UML-диаграмма класса FileReader

Структура состоит из трех полей, два из которых (streamHDFS и localStream) необходимы для работы с файлами. Их разделение обусловлено различиями в интерфейсах. Поле environmetn является ссылкой на внешнюю структуру, содержащую информацию:

* о пути до директории DFS;
* о префиксе данной DFS;
* о необходимой кодировке информации.

Говоря о методах этого класса, можно отметить их тривиальность:

* \_\_init\_\_(…) – создает объект класса и сохраняет переданную ему ссылку;
* \_\_dell\_­\_() – вызывает функцию close() для корректного закрытия файлов;
* \_\_iter\_\_() – возвращает итератор для чтения файла;
* close() – закрывает текущий файл, используя свои методы для DFS и локального файла;
* open(…) – открывает файл по пути path, используя соответствующие методы;
* seek(…) – устанавливает позицию для начала чтения;
* readline() – возвращает строку из открытого файла.

2.3.2 Структура для работы с CSV-форматом

Для удобства работы с табличным форматом (CSV) был спроектирован класс-обертка Table, использующий стандартную библиотеку csv. Диаграмма данной структуры приведена на рисунке 2.3.2.1.

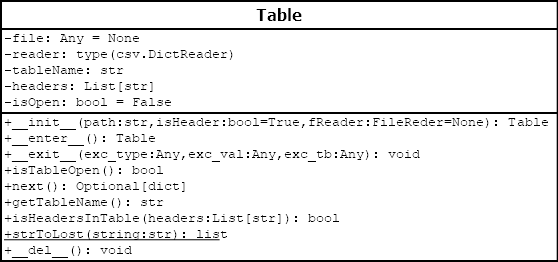


Рисунок 2.3.2.1 – UML-диаграмма класса Table

Данная структура содержит в себе пять полей:

* file – текстовый поток для табличного файла;
* reader – класс, предназначенный для парсинга CSV файлов;
* tableName – хранит в себе название таблицы;
* headers – содержит в себе список полей таблицы (если таковые присутствуют);
* isOpen – флаг, содержащий в себе состояние таблицы.

Функционал имеющихся методов схож с их названием:

* \_\_init\_\_(…) – создает объект класса, текстовый поток (file), парсер (reader), определяет структуру таблицы (headers) и ее название;
* \_\_del\_\_() и \_\_exit\_\_(…) – схожи по функционалу и предполагают закрытие текстового потока;
* \_\_enter\_\_() – возвращает ссылку на объект;
* isTableOpen() – возвращает состояние таблицы;
* isHeadersInTable(…) – проверяет соответствие header таблицы и переданного списка полей.
* next() – позволяет получить следующее поле таблицы;
* strToList(…) – вспомогательный метод, преобразовывающий строку, содержащую данные, в список данных.

2.3.3 Структура для работы с графами

На этом этапе алгоритмы для обработки графов не рассматриваются к реализации, но информация, хранящаяся в DFS, может быть различна, а значит, задачи могут предполагать решение через алгоритмы графов. Поэтому для предоставления функционала сторонним разработчикам будут реализованы базовые структуры для обработки соответствующих данных.

Для первичной обработки графов были написаны две структуры:

* Node, которая содержит информацию о конкретном узле;
* Graph, позволяющий считывать Node из файла.

Диаграмма Node приведена на рисунке 2.3.3.1.

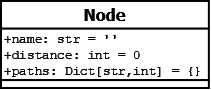


Рисунок 2.3.3.1 – UML-диаграмма класса Node

В структуре содержится следующая информация:

* name – название данного узла;
* distance – расстояние от начального до текущего узла;
* paths – возможные переходы на другие узлы и затраты на перемещение.

Диаграмма Graph приведена на рисунке 2.3.3.2.

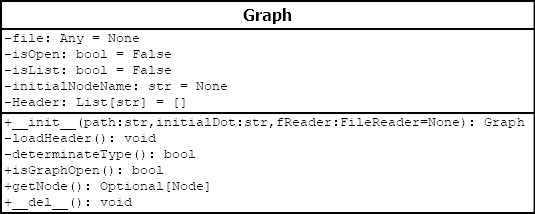


Рисунок 2.3.3.2 – UML-диаграмма класса Graph

Поля, содержащиеся в данной структуре, хранят следующую информацию:

* file – текстовый поток для файла;
* isOpen – флаг, содержащий в себе состояние файла;
* isList – флаг, определяющий формат файла (матрица или список смежностей);
* initalNodeName – поле, хранящее в себе название стартовой ноды;
* Headers – поле, содержащее в себе список нод.

Класс Graph несет в себе следующий функционал:

* \_\_init\_\_(…) – создает объект класса, текстовый поток (file), сохраняет начальную ноду, определяет существующие ноды (Headers) и формат записи переходов;
* \_\_del\_\_() – закрывает текстовый поток;
* loadHeader() – вспомогательный метод, загружающий информацию о нодах графа;
* determinateType() – вспомогательный метод, определяющий формат записи переходов между нодами;
* isGraphOpen() – возвращает состояние текстового потока;
* getNode() – позволяет получить информацию о следующей ноде.

2.3.4 Базовая структура для написания алгоритмов

Для запуска алгоритмов также необходимо разработать структуру, которая будет предусматривать следующие возможности:

* обработка входных параметров программы;
* задание дополнительных опций для алгоритмов, если того требует их логика;
* предпроверка файлов и опций, перед запуском MapReduce задачи.

Диаграмма соответствующего класса представлена на рисунке 2.3.4.1.



Рисунок 2.3.4.1 – UML-диаграмма класса BaseRunner

Вложенный класс parseState содержит необходимые для запуска алгоритма с библиотекой mrjob поля: список входных и выходных файлов, а также префикс для передачи аргументов.

Поля в классе BaseRunner содержат

* информацию о DFS (путь до DFS, префикс DFS, количество полей в записи файла на DFS),
* необходимую кодировку файла,
* список дополнительных опций,
* тип запускаемой задачи.

В случае необходимости первые четыре поля переопределяются через механизм наследования.

На основе выполняемых задач методы можно разделить на шесть:

* protected-методы, которые реализуются разработчиком для добавления параметров (setExtraArgs), проверки входящих файлов (checkFile, checkOption), формата вывода результата (printInfo) и вывода дополнительной информации (zeroPrintInfo);
* методы для формирования входного списка файлов (buildInput, getFileListFromLocal, getFileListFromHDFS, getFileList, getInputPaths, getFileName, getInputFiles);
* методы для формирования входного списка опций (isOption, setOption, optionsParse, buildOptions);
* методы настройки выходного файла (buildOutput) и вывода результата (outprint);
* методы парсинга аргументов (argsParser, cmdArgPars);
* методы запуска MapReduce задач (buildArgs, run).

Алгоритмы обработки табличных данных

3.1.1 Операция Union

В языке SQL операция Union применяется для объединения двух и более таблиц. Условием, при котором объединение допустимо, является одинаковая структура таблиц (одинаковое количество столбцов). Данная операция имеет опцию

* --isHeader, которая хранит информацию о совпадении заголовков таблиц. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – строка таблицы, «значение» – имя таблицы.

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» - строка таблицы, «значение» – игнорируется (None).

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-8а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

Общий алгоритм представлен на рисунке 3.1.1.1.

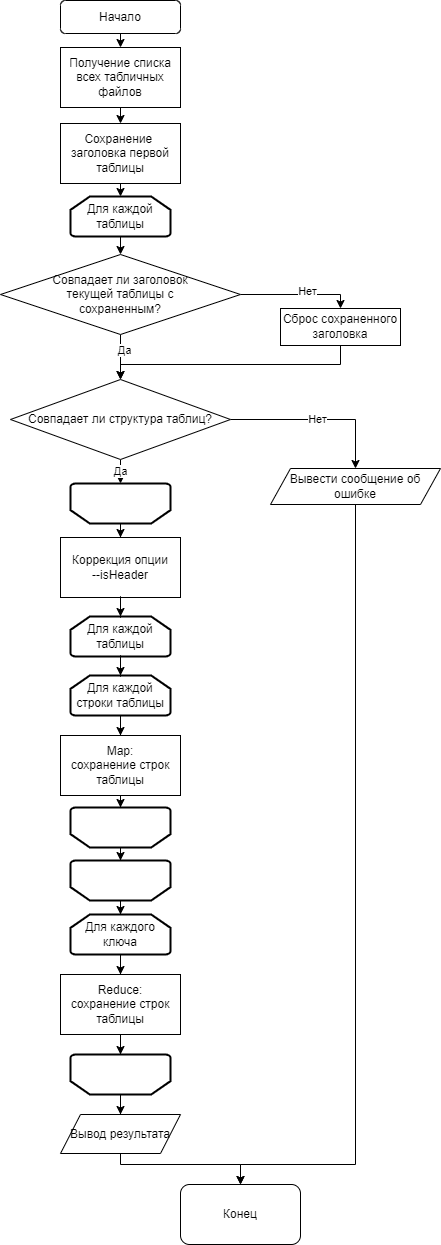


Рисунок 3.1.1.1 – Алгоритм операции Union

**Тестирование**

Для проверки работоспособности были взяты данные FoodData Central за 2020 год.

В качестве теста возьмем несколько строк одной таблицы и разделим их между несколькими файлами, сохранив заголовки таблиц. Тестовые данные представлены на рисунке 3.1.1.2.

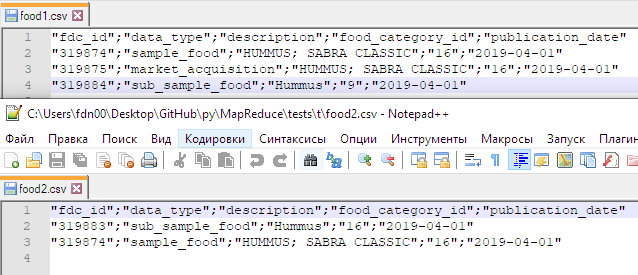


Рисунок 3.1.1.2 – Алгоритм Union, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.1.1.3.

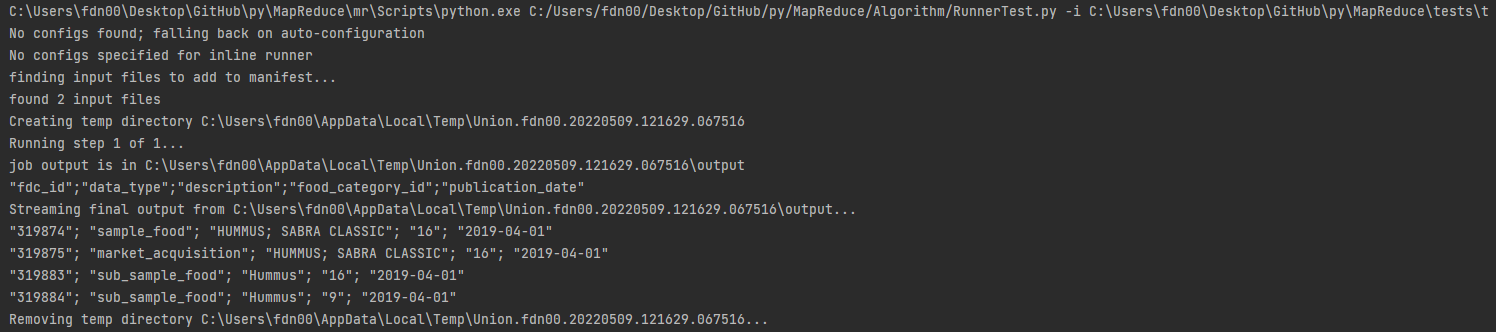


Рисунок 3.1.1.3 – Алгоритм Union, результат

3.1.2 Операция Intersection

В языке SQL операция Intersection используется для получения набора полей таблиц, которые присутствуют во всех таблицах. В нашем случае данная операция имеет следующие опции:

* --countOfTable, в которую заносится число таблиц, участвующих в операции. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно;
* --isHeader, которая хранит информацию о совпадении заголовков таблиц. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.
5. Проверка корректности опции --countOfTable.

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – строка таблицы, «значение» – имя таблицы.

***Функция reduce***

1. Если строка встретилась --countOfTable раз, то результат записывается в следующем формате: «ключ» - строка таблицы, «значение» – игнорируется (None).

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-9а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

5а. Если опция некорректна:

1. Коррекция опция, через подсчет используемых таблиц.

7а. Если запись встретилась несколько раз:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.2.1.

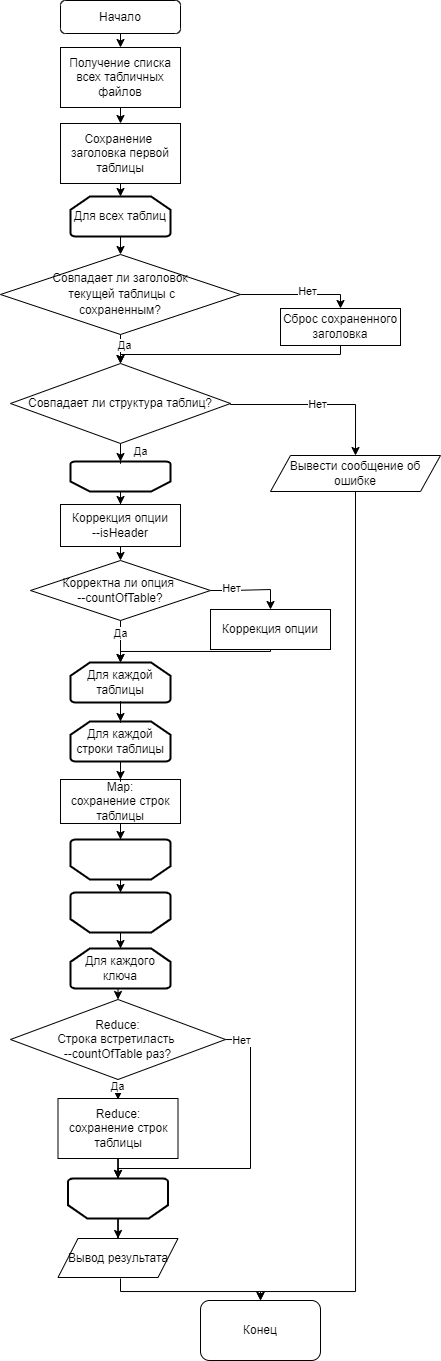


Рисунок 3.1.2.1 – Алгоритм операции Intersection

Для тестирования воспользуемся таблицами, представленными на рисунке 3.1.1.2.

Результат выполнения алгоритма представлен на рисунке 3.1.2.2.

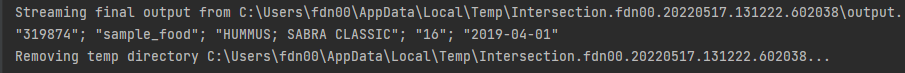


Рисунок 3.1.2.2 – Алгоритм Intersection, результат

3.1.3 Операция Difference

В языке SQL операция Difference используется для получения набора полей из разных таблиц, содержащихся в одной таблице. В реализации содержатся две вариации:

* Difference – сохраняет строку таблицы, если она встречается единственный раз из N`го количества таблиц;
* SymmetricDifference – сохраняет строку таблицы только в том случае, если строка встречается лишь в уменьшаемой таблице.

**SymmetricDifference:**

Данная операции имеет опцию, аналогичную опции Union.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – строка таблицы, «значение» – имя таблицы.

***Функция reduce***

1. Если запись встретилась один раз, то записывается результат в следующем формате: «ключ» – строка таблицы, «значение» – игнорируется (None).

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-8а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

6а. Если запись встретилась несколько раз:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.3.1.

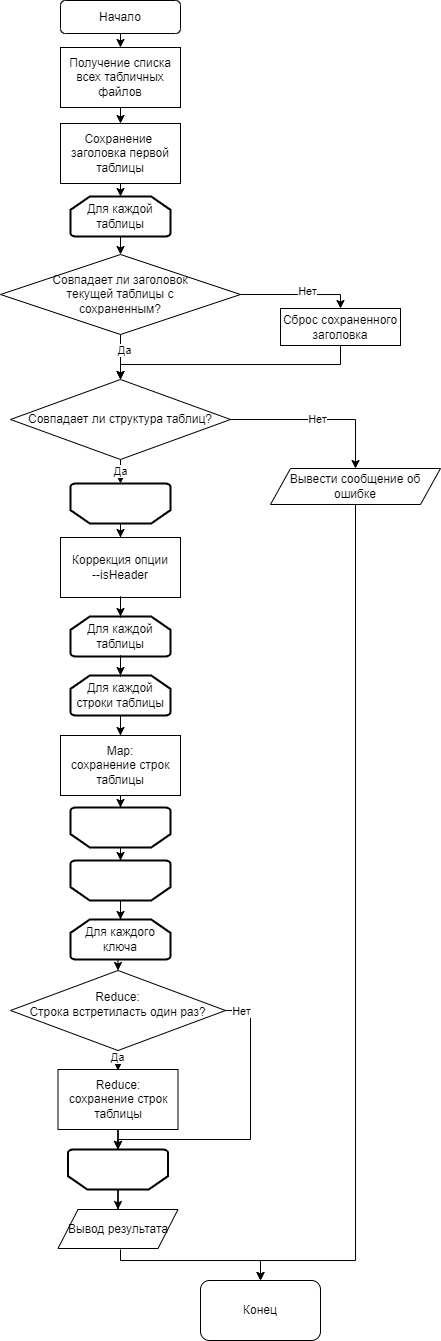


Рисунок 3.1.3.1 – Алгоритм операции SymmetricDifference

Для тестирования воспользуемся таблицами, представленными на рисунке 3.1.1.2.

Результат выполнения алгоритма представлен на рисунке 3.1.3.2.

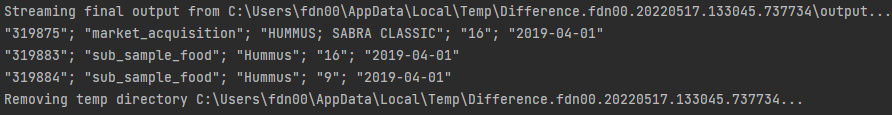


Рисунок 3.1.3.2 – Алгоритм SymmetricDifference, результат

**Difference**

Данная операция имеет следующие опции:

* --isHeader, которая хранит информацию о совпадении заголовков таблиц. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно;
* --reduced, в которую заносится название файла с уменьшаемой таблицей.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.
5. Проверка корректности опции --reduced.

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – строка таблицы, «значение» – имя таблицы;

***Функция reduce***

1. Если запись встретилась только в уменьшаемой таблице, то записывается результат в следующем формате: «ключ» –строка таблицы, «значение» – игнорируется (None)

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-9а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

5-9а. Если таблица, указанная в --reduced, не указана во входных таблицах:

1. Вывод сообщения об ошибке.

7а. Если запись встретилась несколько раз:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.3.3.

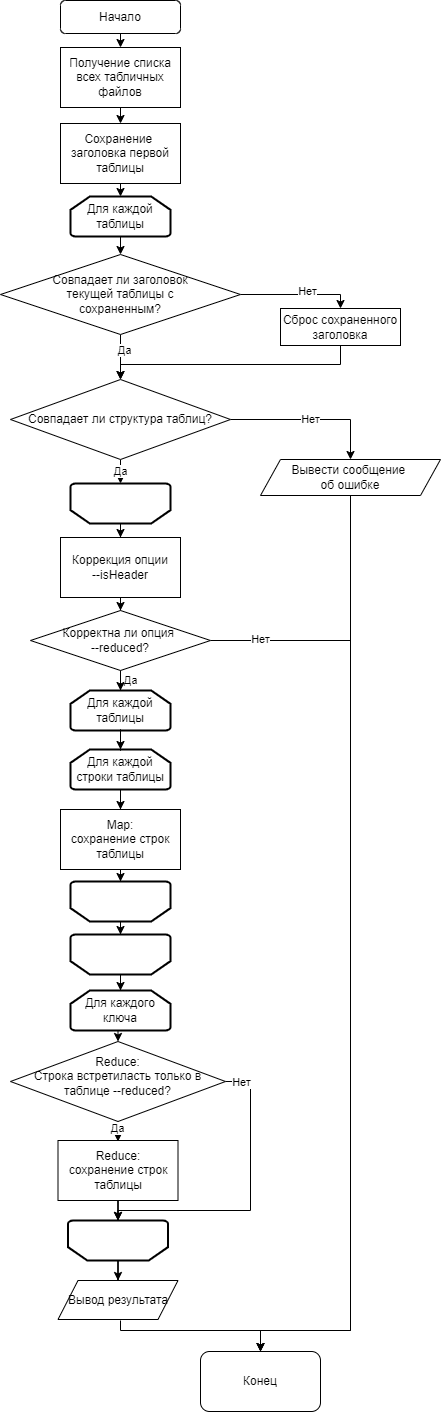


Рисунок 3.1.3.3 – Алгоритм операции Difference

Для тестирования воспользуемся таблицами, представленными на рисунке 3.1.1.2.

Входная опция выглядит следующим образом:

--reduced=food1.csv

Результат выполнения алгоритма представлен на рисунке 3.1.3.4.

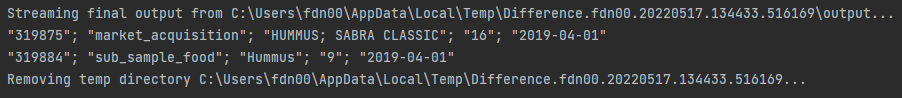


Рисунок 3.1.3.4 – Алгоритм Difference, результат

3.1.4 Операция Select

В языке SQL операция Select используется для получения определенной выборки из БД. В реализации данная операция имеет следующие опции:

* --isHeader, которая хранит информацию о совпадении заголовков таблиц. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно;
* --select, в которой указываются поля таблицы, необходимые для выборки. По умолчанию имеет значение «[\*]» для отображения всех столбцов. Для указания необходимых полей используется следующая запись:

--select=[field1,field2,..,fieldN];

* --where, которая задает ограничения на таблицы. По умолчанию имеет значение «[]», что не накладывает никаких ограничений. Для задания ограничений используется следующая запись:

--where=[field1=value1,field2<value2,.., fieldN =valueN].

Поскольку данная работа не подразумевает полной имплементации функционала СУБД, то агрегация и группировка будут рассмотрены ниже.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.
5. Проверка на единый заголовок таблиц.
6. Проверка наличия полей из опций --select и --where в заголовке таблицы.
7. Проверка знаков условия из опции --where.

***Функция map***

1. Для каждой строки таблицы осуществляется проверка соответствующих полей на соответствие условиям --where.
2. Для каждой строки таблицы сохраняются следующие значения: «ключ» – поля таблицы, указанные в опции --select, «значение» – None.

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-11а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

5-11а. Если заголовки таблиц различны:

1. Вывод сообщения об ошибке.

6-11а. Если полей из опций нет в заголовке таблиц:

1. Вывод сообщения об ошибке.

7-11а. Если используются недопустимые символы:

1. Вывод сообщения об ошибке.

8-11а. Если запись встретилась несколько раз:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.4.1.

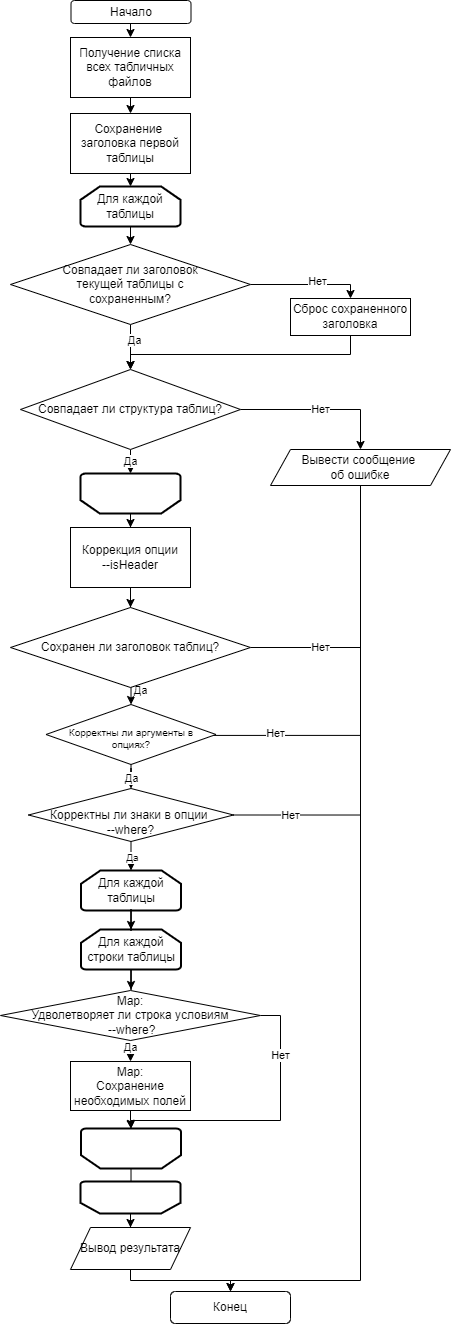


Рисунок 3.1.4.1 – Алгоритм операции Select

Для тестирования воспользуемся таблицами, представленными на рисунке 3.1.1.2. Дополнительные опции запишем следующим образом:

--select=[fdc\_id,data\_type] --where=[food\_category\_id=16]

Результат выполнения алгоритма представлен на рисунке 3.1.4.2.

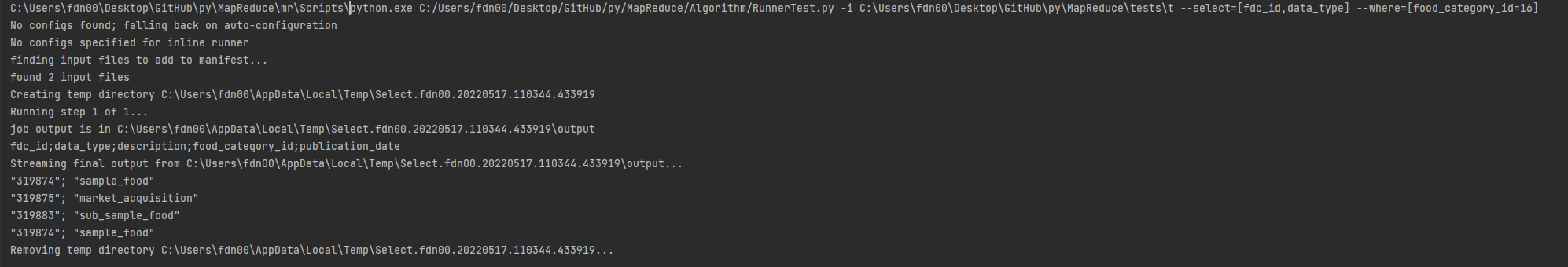


Рисунок 3.1.4.2 – Алгоритм Select, результат

3.1.5 Операция Projection

Операция Projection является модификацией алгоритма Select, исключающей из итога повторяющиеся записи. Дополнительные опции используются аналогично.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.
5. Проверка на единый заголовок таблиц.
6. Проверка наличия полей из опций --select и --where в заголовке таблицы.
7. Проверка знаков условия из опции --where.

***Функция map***

1. Для каждой строки таблицы осуществляется проверка соответствующих полей на соответствие условиям --where.
2. Для каждой строки таблицы сохраняются следующие значения: «ключ» – поля таблицы, указанные в опции --select, «значение» – None;

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – полученная строка таблицы, «значение» – игнорируется (None).

***Вывод***

1. Вывод заголовка таблицы.
2. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-11а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

5-11а. Если заголовки таблиц различны:

1. Вывод сообщения об ошибке.

6-11а. Если полей из опций нет в заголовке таблиц:

1. Вывод сообщения об ошибке.

7-11а. Если используются недопустимые символы:

1. Вывод сообщения об ошибке.

8-11а. Если запись встретилась несколько раз:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.5.1.

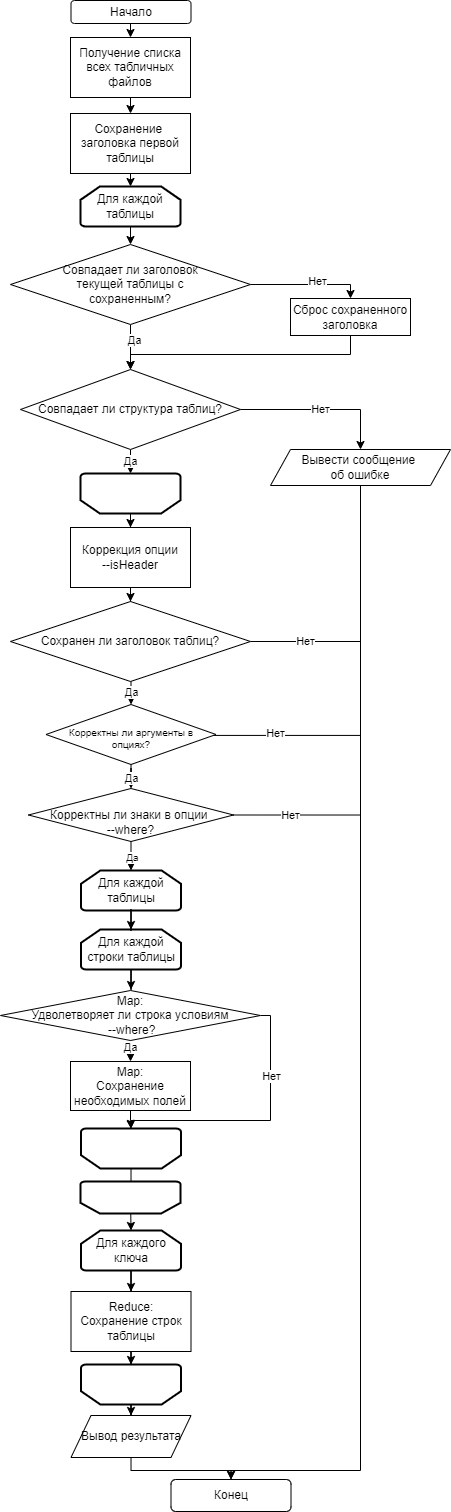


Рисунок 3.1.5.1 – Алгоритм операции Projection

**Тестирование**

Для тестирования воспользуемся таблицами, представленными на рисунке 3.1.1.2. Дополнительные опции останутся аналогичными прошлому запуску.

Результат выполнения алгоритма представлен на рисунке 3.1.5.2.

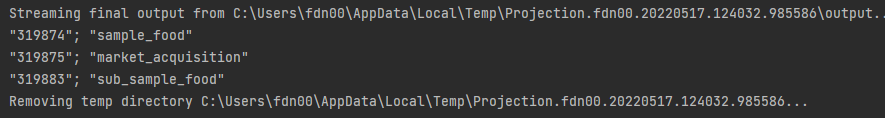


Рисунок 3.1.5.2 – Алгоритм Projection, результат

3.1.6 Операция Group/Aggregation

В SQL операция Group by позволяет сгруппировать строки так, чтобы к ним можно было применять агрегатные функции.

В реализации библиотеки представлены все стандартные функции агрегирования: СOUNT, MIN, MAX, AVG, RANGE и SUM.

Операция Group by имеет следующие опции:

* --isHeader, которая хранит информацию о совпадении заголовков таблиц. Опция является не обязательной, т.к. в процессе выполнения высчитывается программно;
* --groupBy, в которой указываются поля таблицы, по которым будет происходить группировка. По умолчанию имеет значение «[]» для отсутствия группировки. Для указания необходимых используется полей следующая запись:

--groupBy =[field1,field2,..,fieldN];

* --aggregationBy, в которой указываются поля таблицы, к которым будет применяться функция агрегации. Для указания необходимых полей используется следующая запись: -- aggregationBy =[field1,field2,..,fieldN];
* --aggregationFunction, в которой указывается одна из уже описанных функций агрегирования;
* --printType, указывающая тип вывода: вывод всей таблицы (Full) или только сгруппированное значение с результатом агрегации (Grouped).

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Получение и сохранение заголовка первой таблицы.
3. Проверка на одинаковые структуры таблиц и соответствие заголовков.
4. Коррекция опции --isHeader.
5. Проверка на единый заголовок таблиц.
6. Проверка корректности функции агрегирования, полей для группировки (есть ли такие в заголовке) и полей для агрегирования (есть ли такие в заголовке).

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – поля для группировки, «значение» – строка таблицы.

***Функция reduce***

1. К каждым переданным записям применяется функции агрегирования.
2. Записывается результат в следующем формате: «ключ» – полученная строка таблицы/группируемые поля, «значение» – результат агрегации.

***Вывод***

1. Вывод результата работы алгоритма.

**Расширения**

3а. Если заголовки таблиц не совпадают:

1. Отбрасывание заголовков.

3-10а. Если структуры таблиц различны:

1. Вывод сообщения об ошибке.

5-10а. Если заголовки таблиц различны:

1. Вывод сообщения об ошибке.

6-10а. Если одна из опций некорректна

1. Вывод сообщения об ошибке.

Общий алгоритм представлен на рисунке 3.1.6.1.

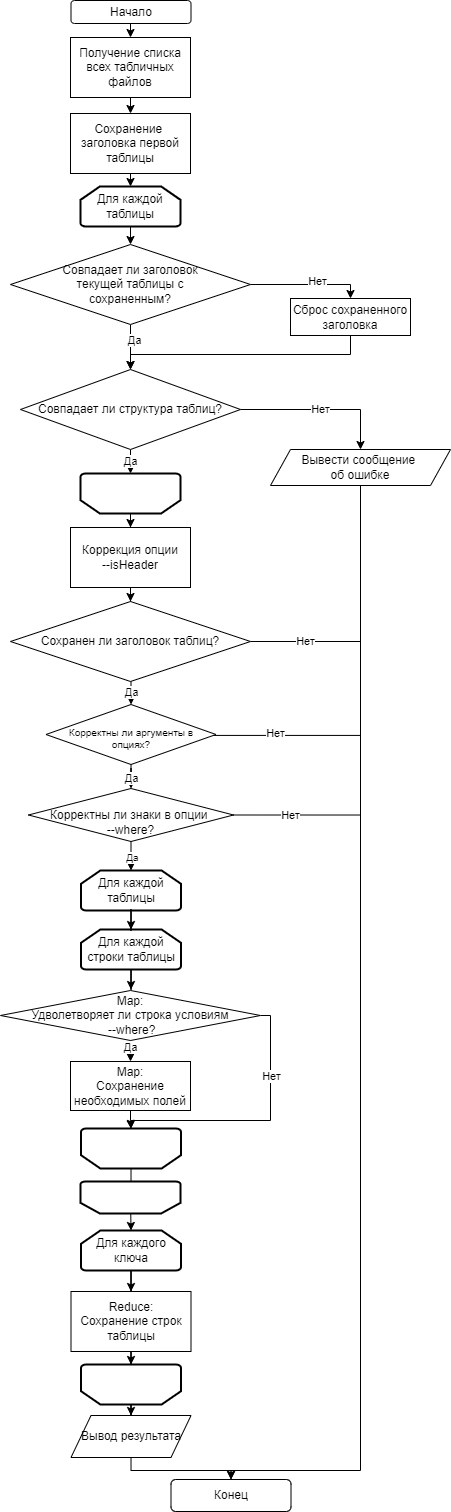


Рисунок 3.1.6.1 – Алгоритм операции GroupByAndAggregation

**Тестирование**

В качестве теста возьмем несколько строк одной таблицы и разделим их между несколькими файлами, сохранив заголовки таблиц. Тестовые данные представлены на рисунке 3.1.6.2.

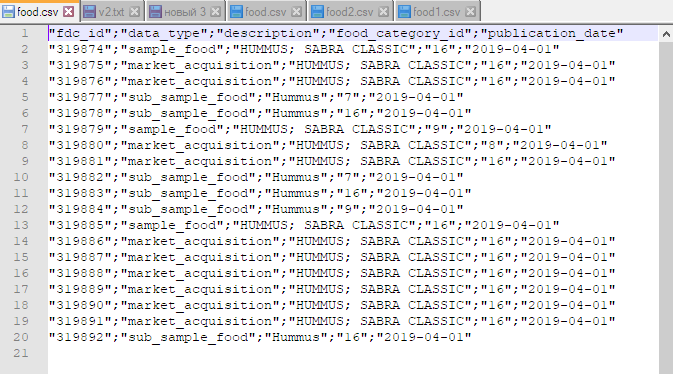


Рисунок 3.1.6.2 – Алгоритм GroupByAndAggregation, тестовые данные

Для запуска теста были выбраны опции, представленные на рисунке 3.1.6.3.



Рисунок 3.1.6.3 – Алгоритм GroupByAndAggregation, опции

Результаты выполнения алгоритма представлены на рисунке 3.1.6.4 и 3.1.6.5.

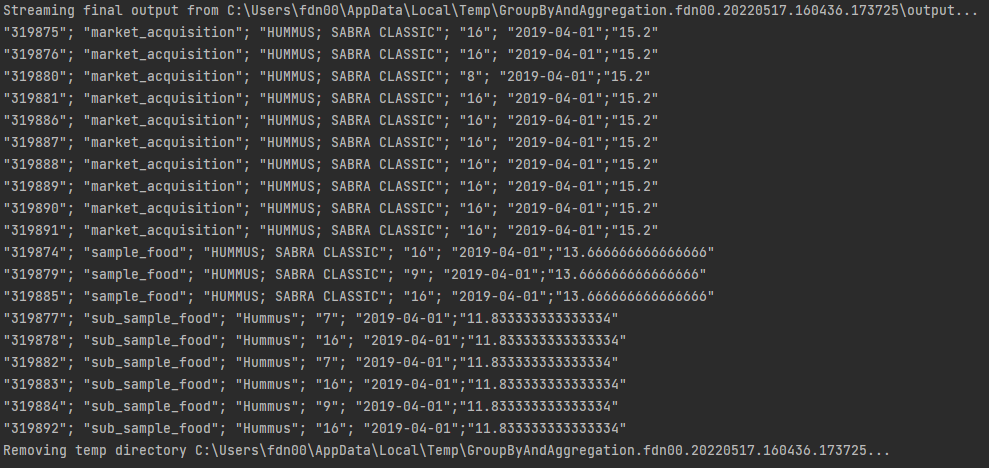


Рисунок 3.1.6.4 – Алгоритм GroupByAndAggregation, полный результат

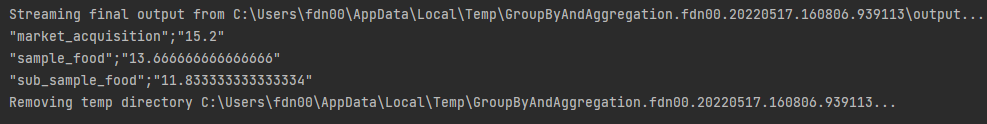


Рисунок 3.1.6.5 – Алгоритм GroupByAndAggregation, сгруппированный результат

3.1.7 Операция Join

В языке SQL операция Join применяется для объединения двух и более таблиц, соединяя их по какому-либо полю таблиц. В основу реализации был положен INNER JOIN.

Данная операция имеет следующую опцию:

* --keys, в которой указываются поля таблиц, по которым будет происходить их соединение (одно поле на таблицу). Для указания необходимых полей используется следующая запись: -- keys=[field1,field2,..,fieldN].

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Получение списка таблиц.
2. Проверка соответствия количества ключей (--keys) и количества таблиц.
3. Проверка наличия ключа в соответствующей таблице.

***Функция map***

1. Для каждой строки таблицы передаются следующие значения в функцию reduce: «ключ» – поле таблицы, по которому будет происходить совмещение, «значение» – имя и строка (за исключением поля, которое используется в качестве ключа) таблицы.

***Функция reduce***

1. Записывается результат, если таблицы были объединены по всем ключам, в следующем формате: «ключ» – текущий ключ, «значение» – результат объединения.

***Вывод***

1. Вывод результата работы алгоритма.

**Расширения**

3-6а. Если количество ключей не соответствует количеству таблиц:

1. Вывод сообщения об ошибке.

3-6а. Если ключа не оказалось в таблице:

1. Вывод сообщения об ошибке.

5а. Если объединение произошло не по всем ключам:

1. Игнорирование записи.

Общий алгоритм представлен на рисунке 3.1.7.1.

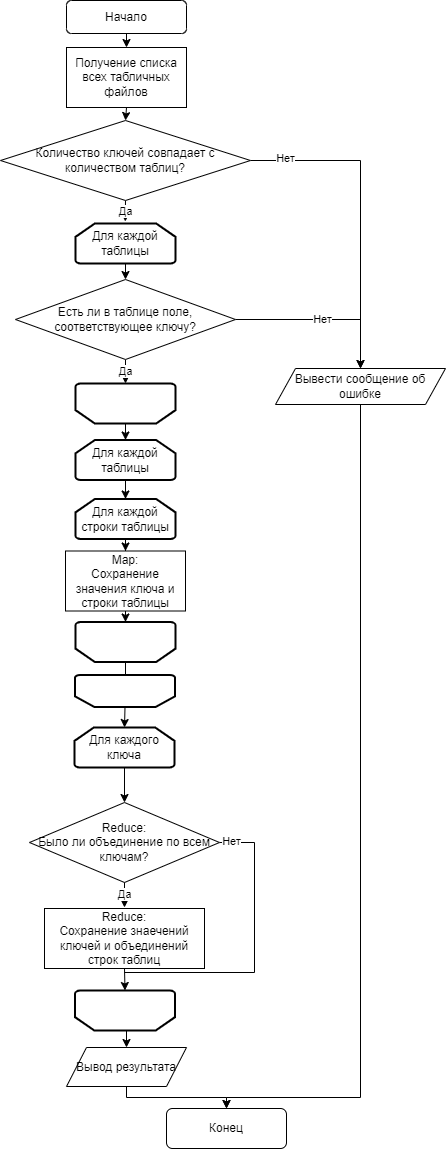


Рисунок 3.1.7.1 – Алгоритм операции Join

**Тестирование**

Для запуска теста были выбраны следующие опции:

--keys=[food\_category\_id,id]

Для тестирования возьмем таблицы, представленные на рисунке 3.1.7.2.

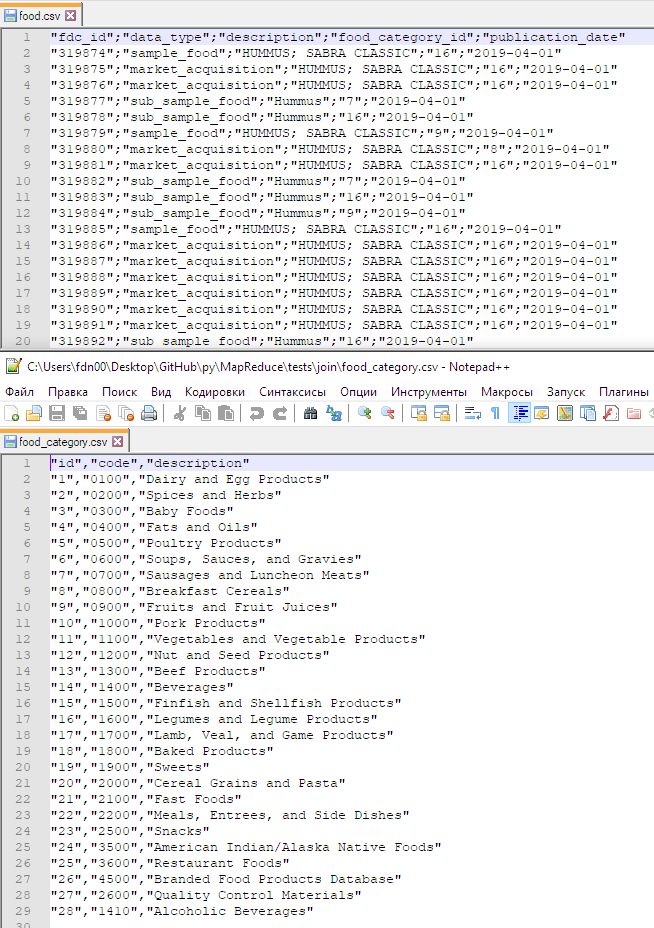


Рисунок 3.1.1.2 – Алгоритм Join, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.1.7.3.

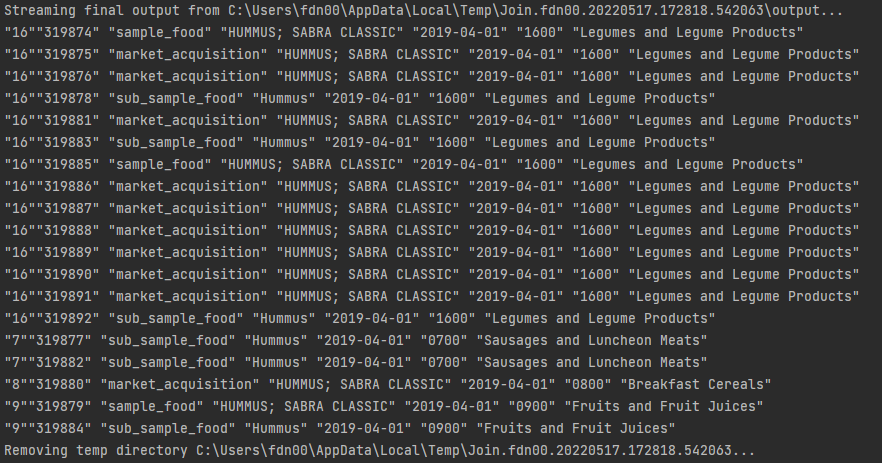


Рисунок 3.1.7.3 – Алгоритм Join, результат

3.2 Алгоритмы обработки текстовых данных

3.2.1 Алгоритм WordCount

Алгоритм подсчета количества слов является одним из самых простых в реализации, но может стать составной частью более сложных алгоритмов, к примеру, алгоритма TF-IDF.

Данный алгоритм реализован в трех вариациях:

* Стандартная реализация, без использования оперативной памяти;
* С использованием оперативной памяти без контроля;
* С использованием оперативной памяти с контролем.

**Стандартная реализация**

Стандартная реализация заключается в использовании только функций Map, Combine, Reduce.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

***Функция map***

1. Файл разбивается на отдельные термы.
2. Для каждого слова файла передаются следующие значения в функцию reduce/combine: «ключ» – терм в нижнем регистре, «значение» – единица.

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – терм, «значение» – количество данного терма (сумма единиц).

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – терм, «значение» – количество данного терма (сумма единиц).

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.1.1.

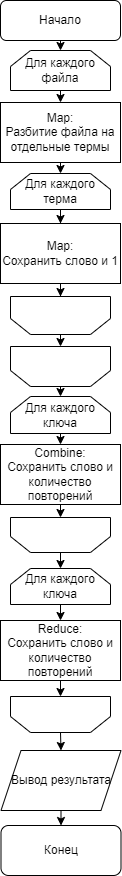


Рисунок 3.2.1.1 – Алгоритм операции WordCounter

**Реализация с использованием Heap**

Оптимизация стандартной реализации, при которой пропускается этап Combine за счет использования оперативной памяти что, соответственно уменьшает количество операций с DFS.

Недостатком реализации является то, что разработчику необходимо быть уверенным, что имеющейся памяти достаточно для операции.

**Основной успешный сценарий**

***Функция map***

1. Выделение памяти на каждом Mapper`е под ассоциативный массив.
2. Файл разбивается на отдельные термы.
3. Каждый терм, в нижнем регистре, заносится в данный массив, инкрементируя значение в массиве при каждой встрече.
4. Передача в функцию reduce следующих данных: «ключ» – терм в нижнем регистре, «значение» – количество терма в документе.

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – терм, «значение» – количество данного терма.

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.1.2.

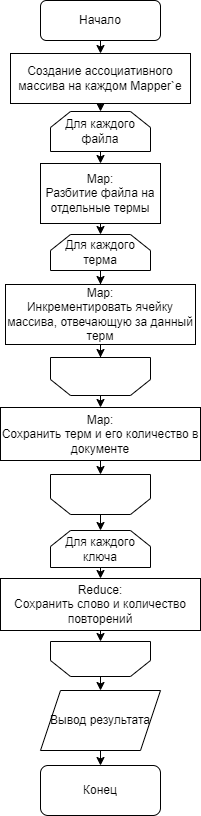


Рисунок 3.2.1.2 – Алгоритм операции WordCounterMemUndetect

**Реализация с контролем использования Heap**

От предыдущей вариации реализация отличается лишь тем, что возобновляет использование Combine и на функции Map регулирует использование памяти, периодически передавая данные в функцию combine.

**Основной успешный сценарий**

***Функция map***

1. Выделение памяти на каждом Mapper`е памяти под ассоциативный массив.
2. Файл разбивается на отдельные термы.
3. Каждый терм, в нижнем регистре, заносится в данный массив, инкрементируя значение в массиве при каждой встрече.
4. Передача в функцию reduce/combine следующих данных: «ключ» – терм в нижнем регистре, «значение» – количество терма в документе.

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – терм, «значение» – количество данного терма (сумма значений, переданных с map).

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – терм, «значение» – количество данного терма.

***Вывод***

1. Вывод результата работы алгоритма.

**Расширения**

3а. Если при инкрементировании/добавлении элемента памяти недостаточно:

1. Выгрузить данные в функцию combine;
2. Очистить ассоциативный массив;
3. Записать элемент в массив.

Общий алгоритм представлен на рисунке 3.2.1.3.

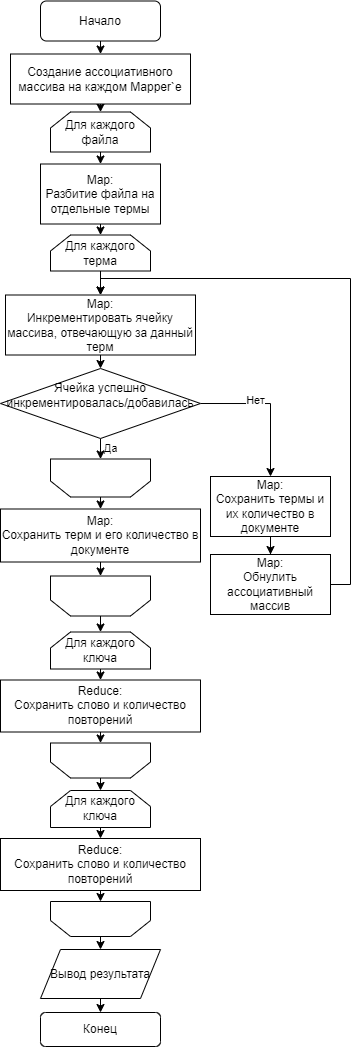


Рисунок 3.2.1.3 – Алгоритм операции WordCounterMemDetect

**Тестирование**

Поскольку данные алгоритмы принципиально не отличаются итоговым результатом, то будет представлено тестирование последнего алгоритма.

Текст для тестирования, представлен на рисунке 3.2.1.4.

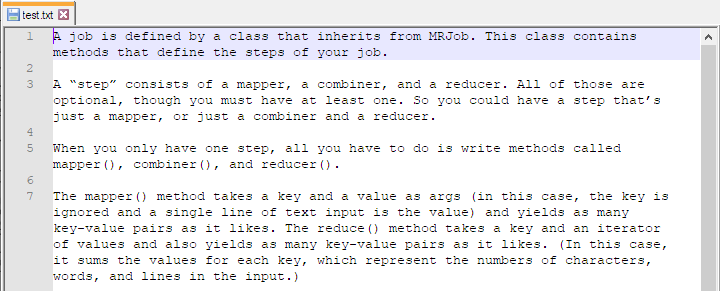


Рисунок 3.2.1.4 – Алгоритм WordCounter, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.2.1.5.

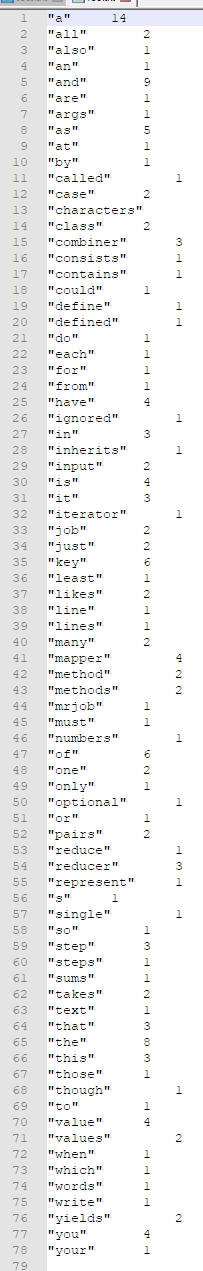


Рисунок 3.2.1.5 – Алгоритм WordCounter, результат

3.2.2 Алгоритм TF-IDF

Алгоритм TF-IDF используется для вычисления «важности» слова в документе или их наборе. «Важность» слова зависит от частоты упоминаемого слова в одном документе и обратно пропорциональна частоте в остальных документах коллекции. Формула для вычисления имеет следующий вид (3.2.2.1):

где – число вхождений слова t документ,

– общее число слов в документе,

– общее число документов в коллекции,

– число документов, в которых встречается слово (когда ).

Данный алгоритм имеет следующую необязательную опцию:

* --countOfFiles, в которой указывается количество файлов, участвующих в вычислении. Во время работы алгоритма значение опции высчитывается программно.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

1. Расчет опции –countOfFiles.

***Функция map***

1. Файл разбивается на отдельные термы.
2. Для каждого слова файла передаются в функцию reduce/combine следующие значения: «ключ» – название документа, «значение» – пара «терм – единица».

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – название документа, «значение» – пара «терм – количества терма в документе».

***Функция reduce***

1. Для каждого терма рассчитывается количество его повторений.
2. Записывается результат в следующем формате: «ключ» – пара «название документа – текущий терм», «значение» – tf для данного терма.

***Функция map***

1. Для каждой записи передаются следующие значения в функцию reduce: «ключ» – терм, «значение» – кортеж «название документа – tf данного терма – единица».

***Функция reduce***

1. Подсчет количества документов, в которых встречается данный терм.
2. Записывается результат в следующем формате: «ключ» – пара «текущий терм – название документа», «значение» – пара «tf данного терма – количество документов, в которых встретился данный терм».

***Функция map***

1. Расчет TF-IDF для каждого терма.
2. Записывается результат в следующем формате: «ключ» – терм, «значение» – TF-IDF данного терма.

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.2.1.

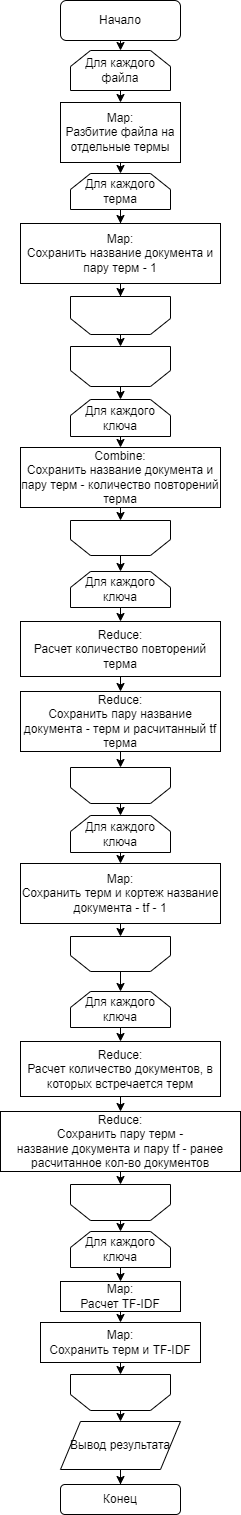


Рисунок 3.2.2.1 – Алгоритм операции TF-IDF

Исходные данные для тестирования, представлены на рисунке 3.2.2.2.

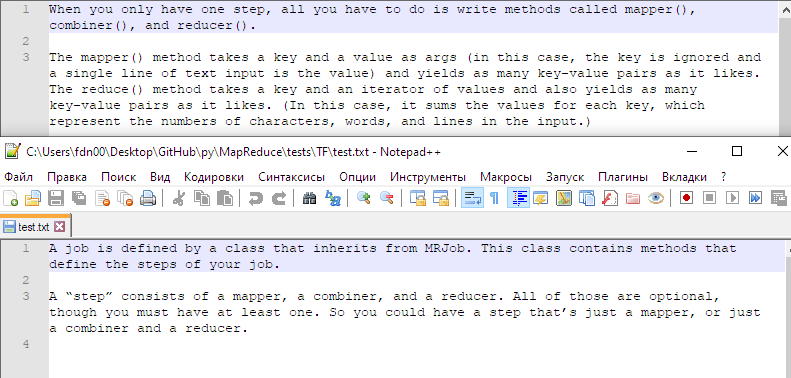


Рисунок 3.2.2.2 – Алгоритм TF-IDF, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.2.2.3.

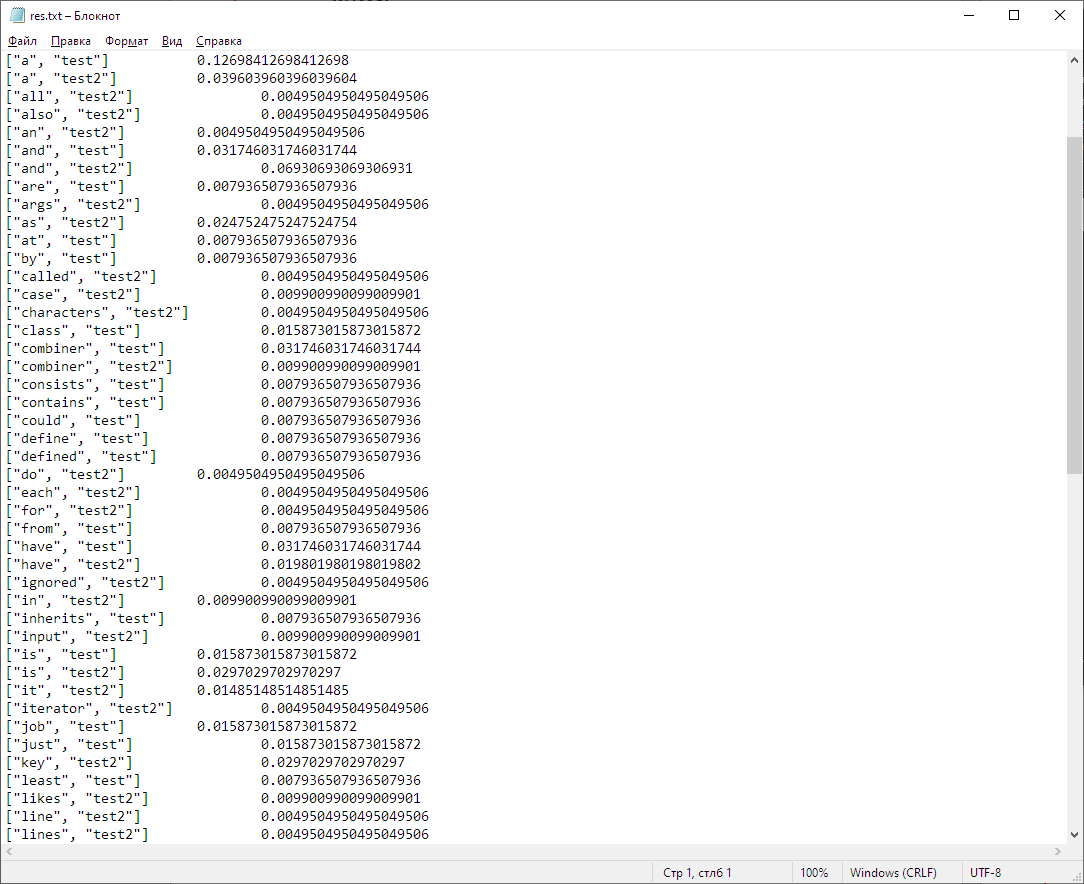


Рисунок 3.2.2.3 – Алгоритм TF-IDF, результат

3.2.3 Алгоритм Cross-correlation

Алгоритм Cross-Correlation применяется для подсчета возможных пар термов (или объектов) в том случае, если они встречаются вместе. Чаще применяется в анализе текста.

Данный алгоритм реализован в трех вариациях:

* Стандартная реализация, без использования оперативной памяти;
* С использованием оперативной памяти без контроля;
* С использованием оперативной памяти с контролем.

**Стандартная реализация**

Стандартная реализация заключается в использовании только функций Map, Combine, Reduce.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

***Функция map***

1. Файл разбивается на отдельные термы.
2. Составляются различные пары термов.
3. Для каждого слова файла передаются следующие значения в функцию reduce/combine: «ключ» – пара термов, «значение» – единица.

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – пара термов, «значение» – количество встреченных пар (сумма единиц).

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – пара термов, «значение» – количество встреченных пар.

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.3.1.



Рисунок 3.2.3.1 – Алгоритм операции CrossCorrelation

**Реализация с использованием Heap:**

Оптимизация стандартной реализации, при которой пропускается этап Combine за счет использования оперативной памяти, уменьшает количество операций с DFS.

Недостатком реализации является то, что разработчику необходимо быть уверенным, что имеющейся памяти достаточно для операции.

**Основной успешный сценарий**

***Функция map***

1. Файл разбивается на сортированный массив отдельных терм.
2. Для каждого терма создается ассоциативный массив.
3. Количество повторений пар с участием текущего терма записываются в ассоциативный массив.
4. Для каждого терма сохраняются в функцию reduce следующие значения: «ключ» – терм, «значение» – ассоциативный массив этого терма.

***Функция reduce***

1. Подсчитывается количество повторений пар из разных документов.
2. Записывается результат в следующем формате: «ключ» – пара термов, «значение» – количество встреченных пар.

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.3.2.



Рисунок 3.2.3.2 – Алгоритм операции CrossCorrelationMemUnDetect

**Реализация с контролем использования Heap**

От предыдущей вариации реализация отличается лишь тем, что возобновляет использование Combine и на функции Map регулирует использование памяти, периодически передавая данные в функцию combine.

**Основной успешный сценарий**

***Функция map***

1. Файл разбивается на сортированный массив отдельных терм.
2. Для каждого терма создается ассоциативный массив.
3. Количество повторений пар с участием текущего терма записываются в ассоциативный массив.
4. Для каждого терма сохраняются в функцию reduce следующие значения: «ключ» – терм, «значение» – ассоциативный массив этого терма.

***Функция combine***

1. Подсчет количества пар термов в документе с использованием ассоциативного массива.
2. Для каждого терма сохраняются в функцию reduce следующие значения: «ключ» – терм, «значение» – ассоциативный массив этого терма.

***Функция reduce***

1. Подсчет количества повторений пар из разных документов.
2. Записывается результат в следующем формате: «ключ» – пара термов, «значение» – количество встреченных пар.

***Вывод***

1. Вывод результата работы алгоритма.

**Расширения**

3а. Если при инкрементировании/добавлении элемента памяти недостаточно:

1. Выгрузить данные в функцию combine;
2. Очистить ассоциативный массив;
3. Записать элемент в массив.

Общий алгоритм представлен на рисунке 3.2.3.3.

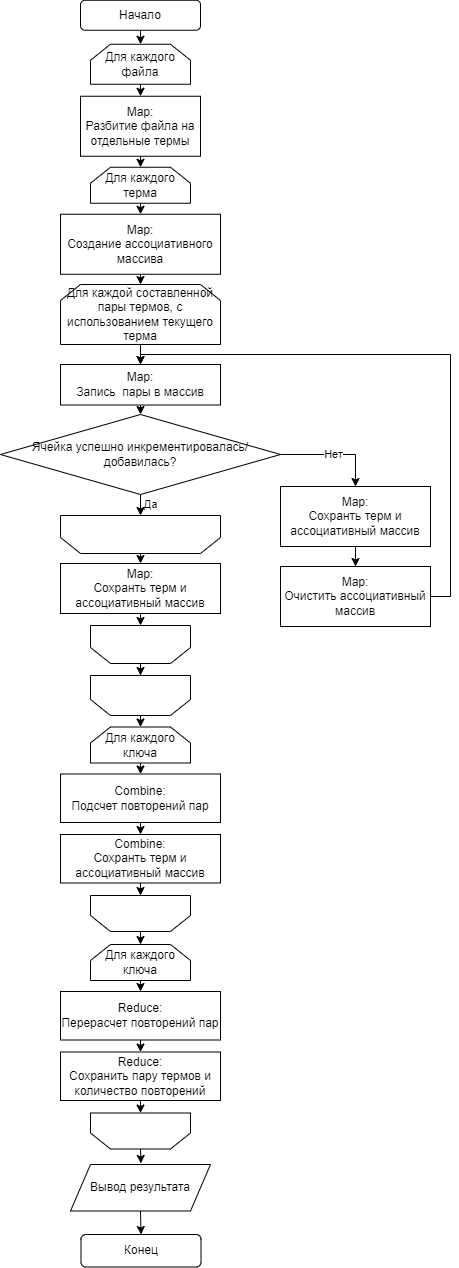


Рисунок 3.2.3.3 – Алгоритм операции CrossCorrelationMemDetect

**Тестирование**

Поскольку данные алгоритмы принципиально не отличаются итоговым результатом, то будет представлено тестирование последнего алгоритма.

Текст для тестирования, представлен на рисунке 3.2.3.4.

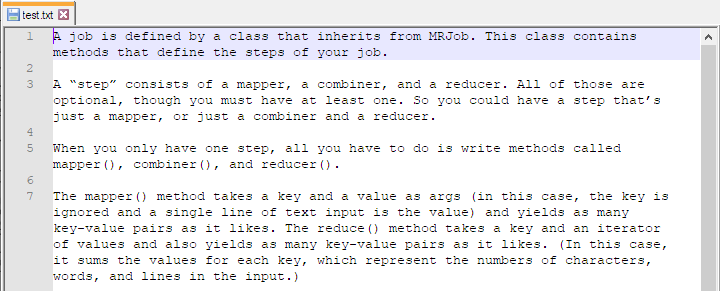


Рисунок 3.2.3.4 – Алгоритм CrossCorrelationMemDetect, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.2.3.5.

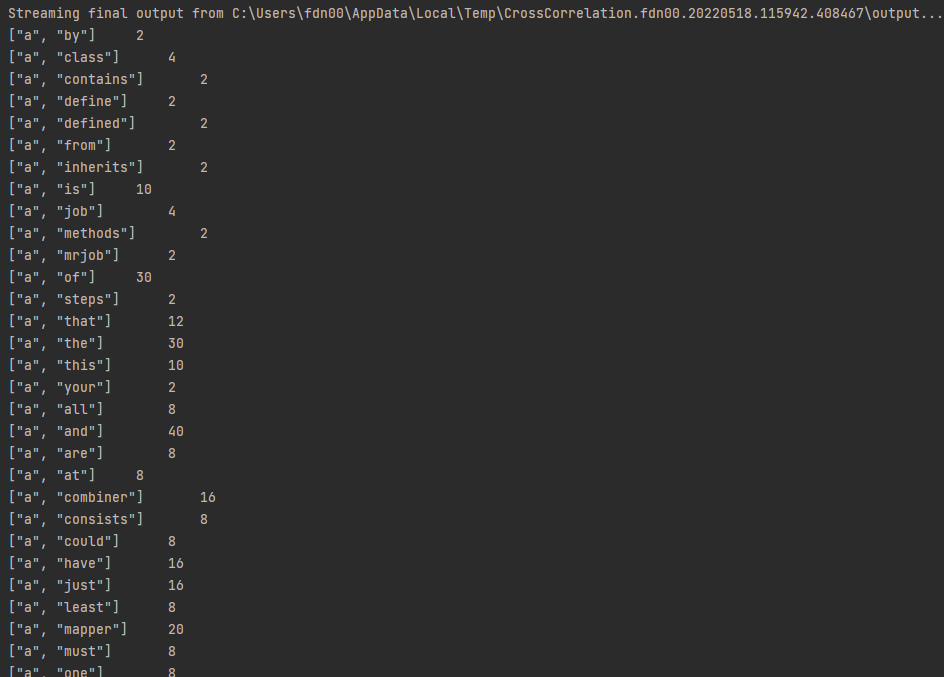


Рисунок 3.2.3.5 – Алгоритм CrossCorrelationMemDetect, результат

3.2.4 Алгоритм Distinct Values

Алгоритм Distinct Values позволяет найти общее число уникальных записей F, для каждого подмножества записей, для каждого значения в каждой категории G. Пример работы подобного алгоритма представлен на рисунке 3.2.4.1.

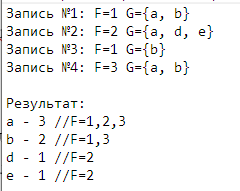


Рисунок 3.2.4.1 – Пример работы алгоритма Distinct Values

Данный алгоритм реализован в двух вариациях:

* Стандартная реализация, без использования оперативной памяти;
* С использованием оперативной памяти без контроля.

**Стандартная реализация**

Стандартная реализация заключается в использовании только функций Map, Combine, Reduce.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

***Функция map***

1. Строки файла разбиваются на запись F и значения категорий G.
2. Для каждой строки передаются следующие значения в функцию reduce/combine: «ключ» – пара значение категорий G-F, «значение» – игнорируется (None).

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – пара значение категории G-F, «значение» – игнорируется (None).

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – пара значение категории G-F, «значение» – игнорируется (None).

***Функция map***

1. Для каждой записи передаются следующие значения в функцию reduce/combine: «ключ» – значение категории G, «значение» – единица.

***Функция combine***

1. Записывается результат в следующем формате: «ключ» – значение G, «значение» – количество встреченных значений G (сумма единиц).

***Функция reduce***

1. Записывается результат в следующем формате: «ключ» – значение G, «значение» – количество встреченных значений G.

***Вывод:***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.4.2.



Рисунок 3.2.4.2 – Алгоритм операции DistinctValue

**Реализация с использованием Heap**

Оптимизация стандартной реализации за счет использования оперативной памяти уменьшает количество операций с DFS.

Недостатком реализации является то, что разработчику необходимо быть уверенным, что имеющейся памяти достаточно для операции.

***Функция map***

1. Строки файла разбиваются на запись F и значения категорий G.
2. Для каждой строки передаются в функцию reduce/combine следующие значения: «ключ» – значение F, «значение» – каждое из значений G.

***Функция combine***

1. Для каждого Combiner`а создается ассоциативный массив.
2. В ассоциативный массив сохраняется сет значений G, для каждого F.
3. Для каждой записи массива записывается результат в следующем формате: «ключ» – ключ массива, «значение» – каждое значение из данной ячейки.

***Функция reduce***

1. Для каждого Reducer`а создается ассоциативный массив.
2. В ассоциативный массив сохраняется количество каждого из значений G во всех наборах G.
3. Для каждой записи массива записывается результат в следующем формате: «ключ» – значение из набора G, «значение» – количество повторений набора.

***Функция reduce***

1. Для каждой записи массива записывается результат в следующем формате: «ключ» – значение из набора G, «значение» – количество повторений набора.

***Вывод:***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.4.3.

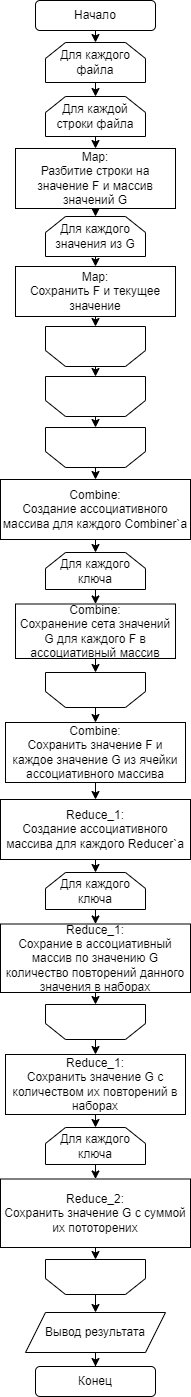


Рисунок 3.2.4.3 – Алгоритм операции DistinctValueMemUnDetect

Поскольку данные алгоритмы принципиально не отличаются итоговым результатом, то будет представлено тестирование последнего алгоритма.

Текст для тестирования, представлен на рисунке 3.2.4.4.

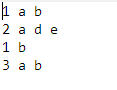


Рисунок 3.2.4.4 – Алгоритм DistinctValueMemUnDetect, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.2.4.5.

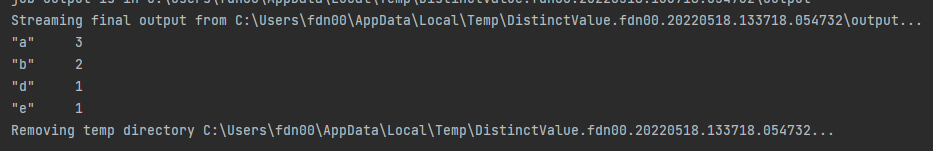


Рисунок 3.2.4.5 – Алгоритм DistinctValueMemUnDetect, результат

3.2.5 Алгоритм поиска среднего значения

Алгоритм поиска среднего значения один из базовых алгоритмов для MapReduce и может являться составной частью для более сложных алгоритмов.

Данный алгоритм реализован в трех вариациях:

* Стандартная реализация, без использования оперативной памяти;
* С использованием оперативной памяти без контроля;
* С использованием оперативной памяти с контролем.

**Стандартная реализация**

Стандартная реализация заключается в использовании только функций Map, Combine, Reduce.

Используя парадигму MapReduce, алгоритм может быть описан следующим образом.

**Основной успешный сценарий**

***Функция map***

1. Строки файла разбиваются на терм и его значения.
2. Для каждого значения терма передаются следующие значения в функцию reduce/combine: «ключ» – терм, «значение» – пара значение-единица;

***Функция combine***

1. Пересчитывает количество значений терма и сумму для данного терма.
2. Записывается результат в следующем формате: «ключ» – терм, «значение» – пара «сумма терма-количество значений».

***Функция reduce***

1. Пересчитывается количество значений терма и сумма для данного терма.
2. Считается среднее значение для терма.
3. Записывается результат в следующем формате: «ключ» – терм, «значение» – среднее значение.

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.5.1.



Рисунок 3.2.5.1 – Алгоритм операции MeanCount

**Реализация с использованием Heap**

Оптимизация стандартной реализации, при которой пропускается этап Combine, за счет использования оперативной памяти уменьшает количество операций с DFS.

Недостатком реализации является то, что разработчику необходимо быть уверенным, что имеющейся памяти достаточно для операции.

**Основной успешный сценарий**

***Функция map***

1. Создание двух ассоциативных массивов для записи количества значений и их суммы.
2. Строки файла разбиваются на терм и его значения.
3. Для каждого терма увеличиваются соответствующие ячейки массивов.
4. Для каждого терма сохраняются следующие значения в функцию reduce: «ключ» – терм, «значение» – пара «сумма терма-количество значений».

***Функция reduce***

1. Пересчитывается количество значений терма и сумма для данного терма.
2. Считается среднее значение для терма.
3. Записывается результат в следующем формате: «ключ» – терм, «значение» – среднее значение;

***Вывод***

1. Вывод результата работы алгоритма.

Общий алгоритм представлен на рисунке 3.2.5.2.

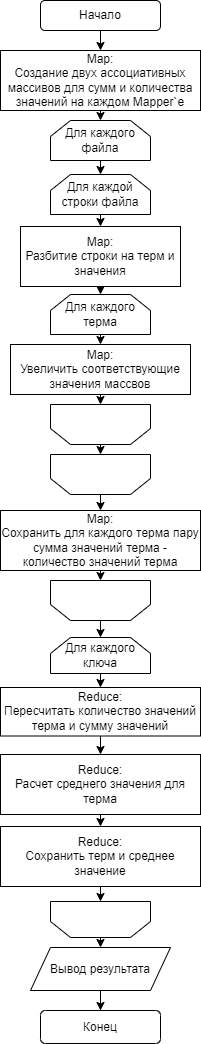


Рисунок 3.2.5.2 – Алгоритм операции MeanCountMemUnDetect

**Реализация с контролем использования Heap**

От предыдущей вариации отличается лишь тем, что возобновляет использование Combine и на функции Map регулирует использование памяти, периодически передавая данные в функцию combine.

**Основной успешный сценарий**

***Функция map***

1. Создание двух ассоциативных массивов для записи количества значений и их суммы.
2. Строки файла разбиваются на терм и его значения.
3. Для каждого терма увеличиваются значения в соответствующих ячейках массивов.
4. Для каждого терма сохраняются в функцию reduce следующие значения: «ключ» – терм, «значение» – пара «сумма терма-количество значений».

***Функция combine***

1. Пересчитывает количество значений терма и сумму для данного терма.
2. Записывается результат в следующем формате: «ключ» – терм, «значение» – пара «сумма терма-количество значений».

***Функция reduce***

1. Пересчитывается количество значений терма и сумма для данного терма.
2. Считается среднее значение для терма.
3. Записывается результат в следующем формате: «ключ» – терм, «значение» – среднее значение.

***Вывод***

1. Вывод результата работы алгоритма.

**Расширения**

3а. Если при инкрементировании/добавлении элемента памяти недостаточно:

1. Выгрузить данные в функцию combine;
2. Очистить ассоциативные массивы;
3. Записать элемент в массивы.

Общий алгоритм представлен на рисунке 3.2.5.3.

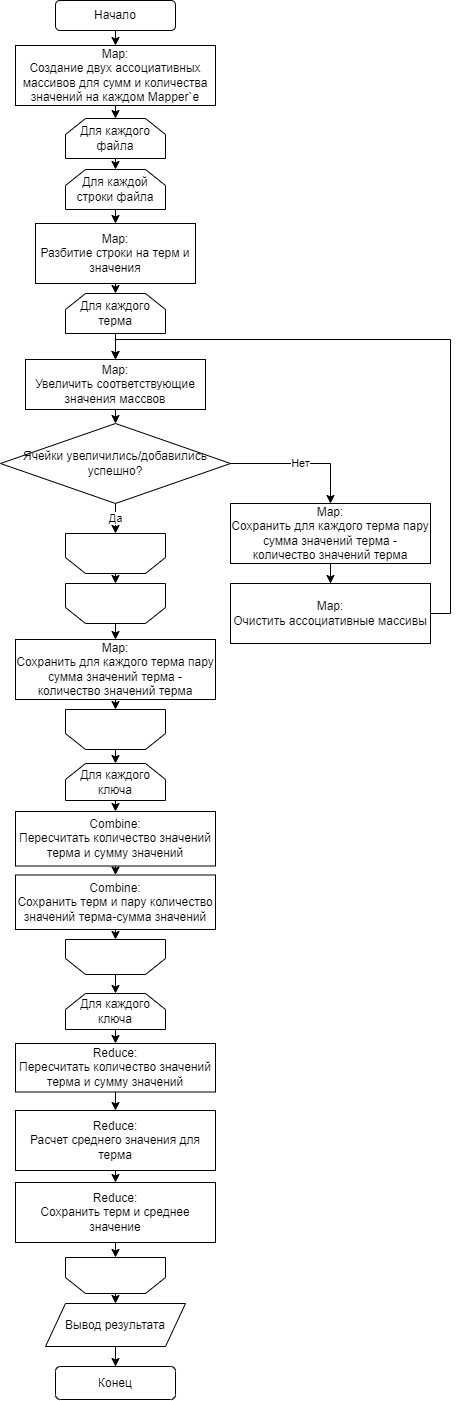


Рисунок 3.2.5.3 – Алгоритм операции MeanCountMemDetect

**Тестирование**

Поскольку данные алгоритмы принципиально не отличаются итоговым результатом, то будет представлено тестирование последнего алгоритма.

Текст для тестирования, представлен на рисунке 3.2.5.4.

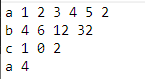


Рисунок 3.2.5.4 – Алгоритм MeanCountMemDetect, тестовые данные

Результат выполнения алгоритма представлен на рисунке 3.2.5.5.

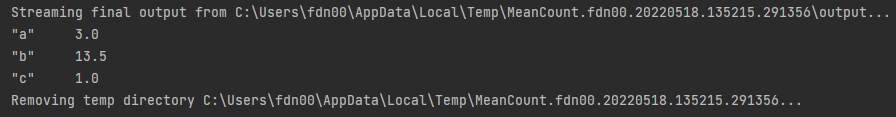


Рисунок 3.2.5.5 – Алгоритм CrossCorrelationMemDetect, результат