**Описание алгоритма анализа расходования кредитных средств заемщиками Банка**

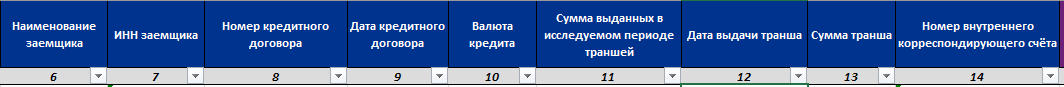
1. ***Общее описание алгоритма:***

Алгоритм собирает из банковских выписок по ссудным и расчетным счетам таблицу движения денежных средств(транжей) от заемщика к конечному получателю

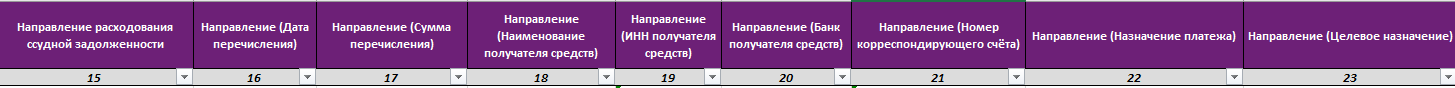
Пример такой выгрузки:

Таблица разделена на три условных блока (не включая нумерацию):

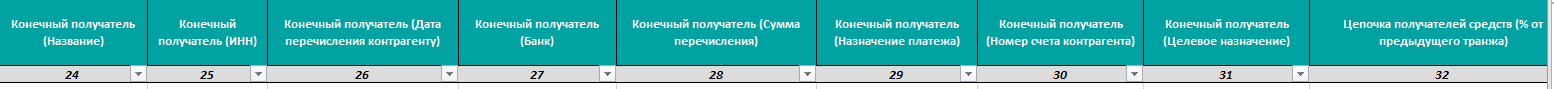
* + 1. Блок, получаемый из данных выписок по ссудным счетам:



* + 1. Блок, получаемый из данных выписок по расчетным счетам (первый получатель заемных средств)



* + 1. Блок, получаемый из данных выписок по расчетным счетам (конечный получатель заемных средств)



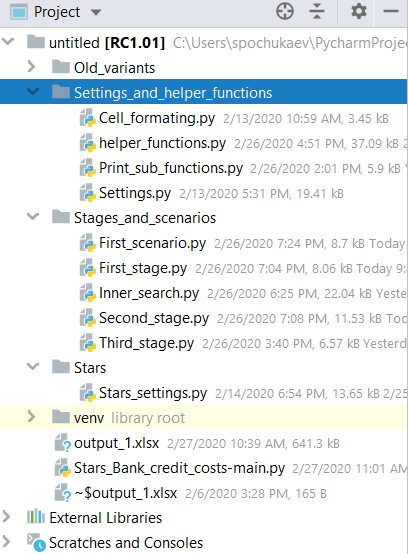
1. ***Общий порядок действий алгоритма:***
2. Входные данные: ИНН и название банка: алгоритм итерируется через цикл for по всем полученным ИНН
3. Алгоритм получает список всех счетов клиента по ИНН и названию банка и фильтрует только ссудные счета.

regex: (^44[1-9])|(^45[0-7]).\*

and

regex: ~ (^\d\d\d15\d\*) (проверка на резервный счет)

1. Алгоритм через цикл for начинает выгрузку всех ссудных счетов клиента (из БД STAR) (далее алгоритм применяется к каждому счету по отдельности)
   1. Формируются шаблоны конечных таблиц (шапка и первая строка с нумерации колонок)
   2. Ссудный счет фильтруется по корсчетам, которые не попадают ни в один из описанных в ТЗ сценариев (более подробное описание пункта ниже)
   3. Заполняется первый блок конечной таблицы на основе отфильтрованной на 4 пункте ссудной выписки (более подробное описание пункта ниже)
   4. По каждой строчке отфильтрованной ссудной выписки идет поиск первого и конечного получателя средств (более подробное описание пункта ниже)
   5. Заполняются второй и третий блоки конечной таблицы (более подробное описание пункта ниже)
   6. Подготовка полученной таблицы к объединению в общую конечную таблицу
2. Объединение всех полученных таблиц в разрезе ИНН и ссудных счетов клиентов в одну общую таблицу
3. Выгрузка таблицы в Excel
4. ***Структура файлов в папках проекта***



**Главным файлом всего проекта является Stars\_Bank\_credit\_costs-main.py**

Результат работы алгоритма сохраняется в **output\_1.xslx** (если файла в директории нет, то алгоритм его создаст, если есть, то заменит старый файл на новый (но файл не должен быть открыт в excel)

В проекте есть четыре папки, в которых лежат .py файлы с функциями для работы главного алгоритма:

* 1. Old\_variants – хранит в себе старые наработки (не используется в основной программе)
  2. Settings\_and\_helper\_functions – хранит в себе вспомогательные файлы:
     + **Cell\_formating.py** - хранит в себе форматы ячеек для выгрузки таблицы в Excel
     + **helper\_functions.py** – хранит в себе основные алгоритмы, которые используются на протяжении всех этапов алгоритма
     + **print\_sub\_functions.py** – хранит в себе функции Print, которые регулируются атрибутом debug, который задается в Stars\_Bank\_credit\_costs-main.py
     + **Settings\_py** - хранит в себе шаблоны конечных таблиц и коды валют
  3. Stages\_and\_scenarios – хранит в себе этапы основного алгоритма
* **First\_scenario.py** – описывает порядок действия для этапа 6 этапа алгоритма (работа с расчетными счетами)
* **First\_stage.py** – описывает порядок действий для 4 и 5 этапа алгоритма (работа с ссудными счетам)
* **Inner\_search.py** – хранит в себе функции рекурсии, по которым происходит поиск конечного получателя средств
* **Second\_stage.py** – описывает общий порядок действий на 6 этапе алгоритма (заполнение первого получателя заемных средств)
* **Third\_stage.py** - описывает общий порядок действий на 6 этапе алгоритма (заполнение конечного получателя заемных средств)

1. ***Описание файла First\_stage.py***

***Задача файла отфильтровать ссудный счет по полученным транжам, сгруппировать полученные данные по дням и записать полученную данные в конечную таблицу***

В файле представлены три функции:

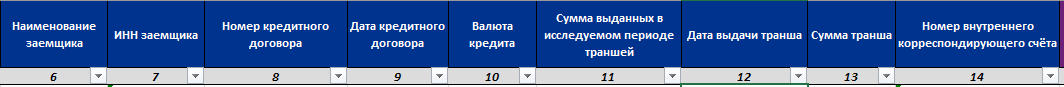
1. **filter\_unsuitable\_rows**

Функция фильтрует ссудный счет по заданным в функции критериям:

*# Сценарий 1: перевод с ссудного счета на расчетный счет (.str.contains(r'^40[1-8]\d\*')))  
# Сценарий 2: перевод с ссудного счета на кассу (.str.contains(r'^202\d\*')))  
# Сценарий 3: перевод с ссудного счета на счет в другом банке (.isna()))  
# pd.notna(input\_file['Дата операции'] нужна для фильтрации входящих и исходящих остатков*

1. **fill\_rows\_based\_on\_credit\_account**

Функция заполняет первую блок конечной таблицы на основе данных отфильтрованной выписки по ссудному счету



1. **group\_by\_credit\_rows\_based\_on\_date**

Группирует полученную от функции fill\_rows\_based\_on\_credit\_account таблицу

по дням выдачи транжа:

Для каждой группы берет первую строчку и заменяет в этой строчке сумму транжа на сумму всех транжей за день

Это нужно для того, чтобы алгоритм мог находить случаи, когда контрагент получил, например, два кредита по 2.5 млн за один день и перевел их одним платежем на 5 млн другому контрагенту(описание непосредственной работы алгоритма, ответственного за поиск получателей средств будет описан ниже)

1. ***Описание файла Second\_stage.py***

***Задача файла определить направление движения ссудных средств и записать полученную данные в конечную таблицу***

В файле представлены три функции:

1. **fill\_rows\_based\_on\_payment\_account**

Определяет направление движения транжа на основе анализа корсчета из выписки по ссудному счету и сопоставляет со сценариями (полученный результат записывает в dict, который хранится на этапе 6 в виде отдельного столбца конечной таблицы и на 7 этапе преобразовывается в строчку функцией fill\_rows\_from\_dict):

* **Сценарий 1: перевод с ссудного счета на расчетный счет**:

1. запускает алгоритмы из файла first\_scenario.py
2. Создает dict (в зависимости от количества получателей денежных средств, количество строк может меняться, поэтому используются отдельные функции для инициализации первой строки и добавления последующих строк)

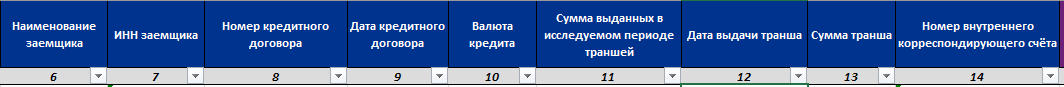
* **Сценарий 2: перевод с ссудного счета на кассу** (всегда одна строка в конечном отчете
* **Сценарий 3: перевод с ссудного счета на счет в другом банк** (всегда одна строка в конечном отчете)

1. **fill\_rows\_from\_dict**

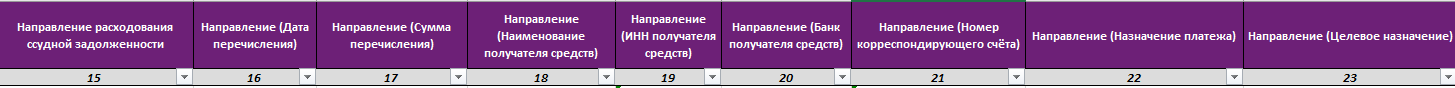
Переводит данные о первом получателе и конечном получателе средств из dict в колонки конечной таблицы.

Основная логика и смысл этой и последующих заполнений колонок конечной таблицы(блок 2 и 3) из dict состоит в том, что изначально мы не знаем, сколько мы получим строчек (первых получателей денежных средств от транжа), поэтому первоначально мы храним полученную таблицу в виде одного объекта (в данном случае dict), потом создается отдельная таблица из dict c сохраненными айдишниками транжей и полученная таблица соединяется с 1 первым блоком конечной таблицы (далее на примере показано соединение первого и второго блока, но в самом алгоритме соединяются блок 1 и совместная таблица блока 2 и 3 (которые были соединены аналогичным образом)):

Блок 1



Блок 2



* 1. Результат работы функции **fill\_rows\_based\_on\_payment\_account**

|  |  |
| --- | --- |
| **Шаблон конечной таблицы (заполнен только Блок 1)** | **dict\_column** |
| Транж 1 | dict с информацией о первом получателе средств (3 строчки) |
| Транж 2 | dict с информацией о первом получателе средств (1 строчки) |
| Транж 3 | dict с информацией о первом получателе средств (2 строчки) |
| Транж N | dict с информацией о первом получателе средств (2 строчки) |

* 1. Распаковываем dict\_column

|  |  |
| --- | --- |
| **Транж (только id)** | **Блок 2** |
| Транж 1 | Информация о получателе средств |
| Транж 1 | Информация о получателе средств |
| Транж 1 | Информация о получателе средств |
| Транж 2 | Информация о получателе средств |
| Транж 2 | Информация о получателе средств |
| Транж N | Информация о получателе средств |
| Транж N | Информация о получателе средств |

1. **Соединяем таблицы из первого и второго шага:**

|  |  |
| --- | --- |
| **Шаблон конечной таблицы (заполнен только Блок 1)** | **Блок 2** |
| Транж 1 | Информация о получателе средств |
| Транж 1 | Информация о получателе средств |
| Транж 1 | Информация о получателе средств |
| Транж 2 | Информация о получателе средств |
| Транж 2 | Информация о получателе средств |
| Транж N | Информация о получателе средств |
| Транж N | Информация о получателе средств |

Таким образом, первый блок изменяется в зависимости от второго блока (дублирует строчки, где получателей средств несколько)

Аналогичный алгоритм используется в случае добавления блока 3 к блоку 2 (Third\_stage.py)

1. **replace\_values\_from\_merge\_columns**

функция переносит данные из блока 2 в шаблон конечной таблицы, потом удаляет блок 2 и подготавливает нумерацию

1. ***Описание файла First\_scenario.py***

Файл представляет собой алгоритм поиска получателей средств.

1. На входе алгоритм получает данные об анализируемом транже (отдельной строчки в отфильтрованной выписки ссудного счета, полученной ранее)
2. Загружается расчетный корсчет (или если он уже ранее был загружен, то забирает выписку по счету из accounts\_dict)
3. Выписка фильтруется по дате получения транжа:

pandas\_file ['Дата операции'] >= cash\_flow['Дата выдачи транша']

1. Определяется временное окно (первоначально только дата выдачи транжа) и выписка по расчетному счету фильтруется по этой дате
2. Дальнейшие шаги повторяются пока функция не вернет какой либо результат ( While True)
   1. Идет проверка, если сумма по дебету больше либо примерно равна сумме транжа (99%), то идет переход на следующий шаг (смысл в том, что мы ограничиваем поиск днями, близкими к дню выдачи транжа):
      1. Если условие неудовлетворено, то временное окно увеличивается на один день и делает повторная проверка шага 5 и так пока условие не будет выполнено
      2. Если временное окно перешло 30 дней, то считается, что определить направление расходования не представляется возможным и возвращается пустая строчка
   2. Делается проверка на идеальное совпадение (ищется сумма по дебету равная сумме транжа (в таком случае считаем, что именно это расходование соответствует предоставленному транжу и отдаем эту строчку расходования)
   3. Если найти идеальное совпадение не удалось, то используется алгоритм knapsack(смысл алгоритма в подборе расходований с целью получения максимально приближенной суммы расходований к сумме транжа) :
3. Алгоритм передаются все отфильтрованные расходования (суммы по дебету) и сумма транжа
4. Далее алгоритм ищет наиболее приближенную к сумме транжа (но не превышая ее) возможную сумму из отфильтрованных по дебету отдельных расходований

<https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%80%D1%8E%D0%BA%D0%B7%D0%B0%D0%BA%D0%B5>

1. Далее алгоритм возвращает расходования из которых данная приближенная сумма может быть получена
   1. Если в результате работы knapsack алгоритма полученная сумма элементов меньше 75%, то используется другой алгоритм (knapsack алгоритм не может определить расходования, которые превышают сумму транжа (например, транж был на 1 млн, а расходование на 1.1 млн, больше расходований за анализируемый день не было и предполагается, что транж ушел на это расходование)
2. Алгоритм откидывает наименьшие элементы последовательности, пока не останется комбинация элементов, которая в сумме будет давать < 130% от суммы транжа или пока не останется один элемент

**Аналогичный алгоритм фильтрации расчетного счета используется в Inner\_search.py**

1. ***Описание файла Third\_stage.py***

Задача файла в соединении блока 2 и блока 3 (порядок соединения описан в Описание файла Second\_stage.py)

В файле запускается процесс рекурсии из ***Inner\_search.py***

Также создает поле комментарии, из которого далее формируется цепочка получателей средств

1. ***Описание файла Inner\_search.py***

***Задача файла рекурсией получить конечного получателя средств***

Описание функций файла:

* 1. **inner\_search\_end\_receiver**

основная функция рекурсии, порядок ее работы будет описан ниже

* 1. **upper\_concat**

функция соединяющая результаты на обратном шаге рекурсии

* 1. **inner\_analyze\_first\_scenario**

Аналог функции из First\_scenario.py (единственное отличие, что этапе 2 также отфильтровываются строки, которые ранее уже были использованы в рекурсии - all\_operations\_id)

* 1. **add\_operations\_to\_all\_operations\_id**

Добавляет исследуемого контрагента и все операции, которые участвовали в рекурсии

Нужна для предотвращения возможности закольцовывания рекурсии (когда рекурсия переходит по одним и тем же контрагентам0

* 1. **calculate\_percentages**

Рассчитывает проценты анализируемого расходования от предыдущего расходования (предыдущего получателя заемных средств)

Нужна для формирования цепочки получателя средств.

**Описание рекурсии:**

1. На входе функция получает данные о расходовании первого получателя денежных средств (далее будет рассмотрен случай, когда деньги первого получателя далее были переведены на расчетный счет внутри банка(счет начинается на 407), иначе выдается пустая строчка)
2. Делается выгрузка по расчетному счету следующего получателя (получателей денежных средств) и делается анализ аналогично описанному анализу в First\_scenario.py

Получаем отфильтрованную выписку по расчетному счету (пример таблицы на очередном этапе рекурсии):

|  |  |  |  |
| --- | --- | --- | --- |
| Исследуемый контрагент | Дата расходования | Сумма расходования | Корсчет (куда ушли деньги) |
| Ромашка -1 | 12.02.2017 | 10000 | Расчетный счет |
| Ромашка -1 | 20.02.2017 | 1000 | Касса |
| Ромашка -2 | 21.02.2017 | 20000 | Расчетный счет |
| Ромашка -3 | 23.02.2017 | 30000 | Расчетный счет |

1. Для каждой строчки полученной таблицы, в случае если в строчке корсчет не является Расчетным счетом, то выдается конечная строчка (конечный получатель средств, которая представляет собой информацию в виде dict, необходимую для формирования блока 3 конечной таблицы
2. В случае если корсчет является расчетным счетом, то происходит рекурсия и возвращается таблица, аналогичная примеру из этапа 1: полученная таблица добавляется в отдельном столбце(dict\_column) первоначальной таблицы

То есть, в ходе рекурсии получается матрешка из таблиц, каждая из которых хранит в себе информацию о получателе средств и в колонке dict\_column хранится информация в dict о следующем получателе средств, которая хранит в себе информацию о получателе средств и колонку dict\_column со следующим получателем средств и т.д., пока деньги либо не уйдут через кассу, либо в другой банк

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исследуемый контрагент | Дата расходования | Сумма расходования | Корсчет (куда ушли деньги) | Dict\_column |
| Ромашка -1 | 12.02.2017 | 10000 | Расчетный счет | Таблица с данными о следующем получателе |
| Ромашка -1 | 20.02.2017 | 1000 | Касса | Данные о конечном получателе |
| Ромашка -2 | 21.02.2017 | 20000 | Расчетный счет | Таблица с данными о следующем получателе |
| Ромашка -3 | 23.02.2017 | 30000 | Расчетный счет | Таблица с данными о следующем получателе |

1. Когда с помощью рекурсии все полученные таблицы достигли своих конечных получателей, то на обратном ходе рекурсии функция upper\_concat начинает соединять данные внутри таблицы, аналогично с функцией fill\_rows\_from\_dict

То есть на первом шаге обратной рекурсии данные о конечном получателе средств соединятся с данными о предыдущем получателе денежных средств, потом полученная таблица будет соединена с данными о втором с конца получателе денежных средств и т.д. пока все не будет объединено в одну таблицу

Также в ходе обратной рекурсии происходит формирование обратной цепочки контрагентов – получателей денежных средств