CALCUMET

консольное приложение для расчета потребности листового металла

ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ

1. ВВЕДЕНИЕ

Для демонстрации работы приложения случайным образом сгенерирована номенклатура, состоящая из 45 артикулов металлоизделий: пластины, ребра, уголки — моноизделия, по 10 артикулов; 10 кронштейнов и 5 балок — сборные изделия, состоящие из 2 или 3 частично пересекающихся условных "деталей". Таким образом, мы имеем отношение "многие ко многим": есть изделия, состоящие из нескольких деталей, а деталь, в свою очередь, может входить в состав нескольких сборных изделий. С точки зрения выбранной модели учета все изделия формально сборные — с той лишь разницей, что сборное изделие ссылается на несколько деталей, а моноизделие и деталь ссылаются сами на себя, являясь комплектами размерности один. Артикулы есть у всех типов изделий, детали отдельно продаже не подлежат, моноизделия в состав сборных не входят.

2. СПРАВОЧНИКИ ДАННЫХ

Работа с приложением начинается с заполнения трех справочников в файле *loaders.xlsx*:

1. Вкладка *feedstock* — справочник типов листового металла. Введите толщину, ширину и длину листа — столбцы *d*, *W*, и *L* соответственно; *rho* (рус. "ро") — плотность марки стали — и присвойте данной комбинации параметров уникальный идентификатор в столбце *sheet*.
2. Вкладка *preforms* — справочник заготовок. Введите наименование детали или моноизделия (сборных изделий здесь быть не может) в столбец *elem*; в *a* и *b* — ширину и длину заготовки, порядок значения не имеет. Под *заготовкой* в данном тексте понимается прямоугольный участок листа металла, описанный вокруг вырезаемого изделия в развернутом виде. В столбец *sheet* внесите соответствующий тип стального листа.

3. Вкладка *contents* — справочник составов изделий. Столбец *item* — сборное изделие, построчно расписанное на составляющие детали (столбец *elem*) и соответствующие количества (столбец *pcs*), деталь или моноизделие, ссылающееся само на себя и количество 1.

Во всех трех справочниках первый столбец — *date*. Он предназначен для обработчика и не должен заполняться пользователем. Далее из консоли будет отправлена команда на создание json-дампов соответствующих справочников, и если валидация данных в строке прошла успешно, в поле *date* записывается дата и время обработки строки. Маркером того, что строка нуждается в обработке, служит пустое поле *date*. Поэтому заполнять его пользователь не должен. Если валидация данных строки вернула ошибку, в поле *date* записывается соответствующий тип ошибки. Исправьте данные, очистите поле *date* от типа ошибки, запустите обработку повторно — ошибка исправлена, дата обработки записана. У валидных же строк стирать дату, инициируя тем самым их перезапись, не нужно.

Останавливаться подробно на типах ошибок в рамках данной инструкции мы не будем. Заинтересованный читатель может либо самостоятельно прочитать код, либо задать вопросы автору. Отметим лишь, что валидация интуитивно понятна и соответствует логической природе и допустимым значениям вводимых данных, а количество 0 во вкладке *contents* работает как удаление компонента из состава. Вкладки загрузчика *loader.xlsx* служат и для заведения (начальной инициализации) дампов, и для их редактирования в случае любого обновления данных.

Обратите внимание, что в случае правки таблицы-справочника необходимо обновлять соответствующий дамп командой из консоли — код работает с json, а *loader.xlsx* служит инструментом подачи объемных табличных данных. Новая строка в *contents* уже существующего *item* редактирует количество соответствующего компонента *elem* (или добавляет новый компонент в состав), не влияя на другие компоненты, если они существуют. Другими словами, в составах *contents* ключами для обновления служат только компоненты, а обновление целого, т.е. сборного изделия, работает через обновление компонент: сборное изделие, в котором нет ни одного компонента, удалит себя само. И наоборот: новое сборное изделие заводится путем добавления хотя бы одного ненулевого компонента.

В дампах заготовок и типов металла обновление по введенному ранее ключу — компоненту *elem* или типу листа *sheet* — работает иначе: происходит полное обновление записи согласно стандартной логике работы ключей в словарях python. Если в *feedstock* или *preforms* сделать несколько разных записей под одним ключом, обновить дамп, то актуальной для данного ключа будет последняя запись. Обновление данных происходит не через поиск и редактирование старой записи, а перезаписью поверх. Таким образом, вкладки-справочники работают не только как инструмент загрузки данных и индикации невалидных записей, но и как лог, хронологически упорядочивающий историю ваших технологических изменений. Дамп же при этом хранит только актуальное состояние и используется кодом для расчетов.

3. ДАМПЫ В JSON

Справочники в *loader* заполнены, перейдите в командной строке в папку с проектом (*calcumet*) и после команды **`update`** перечислите через пробел вкладки *loader*, дампы которых требуется сделать.

Допустимые вкладки только *feedstock*, *preforms* и *contents*, поэтому недопустимая, *non-existing-tab*, вернет сообщение *Wrong tab: {имя_вкладки}* без возбуждения какого-либо исключения, а допустимая, *contents* в данном случае, отработает корректно. Повторим процедуру для оставшихся двух вкладок. Результат: preforms после обновления, feedstock после обновления.

В папке проекта создана папка *json* с 4 дампами: по описанным выше вкладкам и *subitems.json*, производный от *contents*. Его создание и обновление инициируется созданием и обновлением *contents*, поэтому работать с ним отдельно не нужно. Обновление дампов производится аналогично: в конец соответствующего справочника *loader* добавляются строки, дампы перезаписываются командой **`update`**, успешная обработка строки маркируется временем операции.

4. РАСЧЕТ ПОТРЕБНОСТИ ЛИСТОВОГО МЕТАЛЛА

Список изделий, планируемых к производству (сборные и моноизделия), вносятся во вкладку *demand*: артикул и соответствующее количество в штуках. Эти данные определяются пользователем самостоятельно, для данного инструмента они входящие. Назначение столбца *date*, а также проверка введенных количеств аналогичны описанным выше для справочников: пустая ячейка в *date* как приглашение к обработке, построчная валидация данных, дата и время в случае успеха. Валидация включает проверку на вхождение артикула в дамп *contents*. Обнаруженные дефективные строки маркируются типом ошибки, но это не препятствует обсчету валидных. Если такое поведение не подходит, исправьте ошибки и очистите *date*, чтобы заново обсчитать все строки совместно. Например, если хотя бы для части заказа требуется сгруппированный обсчет, о котором ниже.

Для демонстрации случайным образом сгенерирован заказ и записан во вкладку *demand*. Далее в командной строке пишется команда **`go`** и опциональный аргумент **`-g`** (**`--grouped`**) — список через пробел слов-маркеров, которые отбирают из заказа наименования для сгруппированного обсчета потребности в листах: go -g ребро. Если обработка запущена ошибочно, т.е. если строк с пустой датой нет, или ни одна строка не прошла валидацию, в консоль выводится сообщение *No demand to process or incorrect values in demand*.

В данном примере группируются ребра. Это означает, что заготовки разных ребер, при условии общности типа металла, могут быть размещены на одном листе металла при переходе от одного вида ребра к другому. См. строки со стратегией *grouped* в столбце *mode* и соответствующее рассчитанное число листов металла. Арифметически данная стратегия расчета реализована следующим образом.

При допустимости группировки на одном листе заготовок разных артикулов мы хотим израсходовать минимальное *целое* количество листов данного типа так, чтобы изготовить ребра в количествах *не меньших*, чем в заказе. Для этого сначала считаются количества листов поартикульно (нецелые числа), они складываются, давая суммарное количество листов данного типа, необходимое для группируемых изделий. Эта сумма (также нецелое в общем случае число) округляется до целого в большую сторону, а "начисленная" при этом разница распределяется между группируемыми артикулами так, что в первую очередь прибавляется к тому, который ближе других снизу к целому числу. Такой подход способствует тому, что будет больше целочисленных слагаемых при данной фиксированной целочисленной сумме.

Таким образом, в данном примере для ребер, изготавливаемых из листов типа *type1*, первых ребер (см. построчно) рекомендуется изготовить +10 шт. к минимуму, затратив ровно 5 листов, вторых +5 шт. к минимуму, затратив 14,7 листов. Остальные артикулы ребер изготовить ровно в минимально необходимом количестве. Совокупно при данной группирующей стратегии будет израсходовано 54 целых листа.

Если список *grouped* пуст, то все входящие в заказ артикулы обсчитываются по стратегии *single*: изделия изготавливаются кратно своему листу в количестве не меньшем заказа. Фактически стратегия *single* это вырожденный случай стратегии *grouped*, если в *grouped* брать не минимально необходимое целое количество листов, а добавлять по листу до момента, когда каждое слагаемое станет целым.

Обратите внимание, что фильтрация по словам, входящим в список *grouped*, осуществляется поиском по подстроке: можно фильтровать по любому фрагменту наименования, например, не просто по типу изделий, а по артикульной серии, по любому текстовому маркеру, если это осмысленно в рамках технологического процесса. При этом маркер, записанный в обоих регистрах, ищется в оригинальном наименовании, записанный в нижнем — в наименовании вне зависимости от регистра.

Таблица с результатами расчета, которую вы видите на предыдущем изображении из консоли, также записывается во вкладку *report:* фрагмент таблицы.
Дата расчета соответствует дате обработки заказа во вкладке demand. Если деталь входит в состав нескольких сборных изделий, все они перечисляются через пробел в столбце *items*.

К вопросу о том, как число заготовок в штуках пересчитывается в листы.
Дамп заготовок *preforms*, как мы помним, хранит размеры заготовки и тип листа для данного моноизделия или детали. Число заготовок из одного листа, или *pps* (*preforms per sheet*), рассчитывается как максимально возможное число заготовок данных размеров из листа данного размера при учете обеих ориентаций заготовки относительно листа: если б*о*льшая сторона заготовки параллельна б*о*льшей стороне листа и если б*о*льшая сторона заготовки параллельна меньшей стороне листа: *pps = max((length // a) * (width // b), (length // b) * (width // a))*, где *width* и *length* — ширина и длина листа соответственно, *a* и *b* — стороны заготовки, // — целочисленное деление.

Если заготовка мала относительно листа, то, как правило, *pps* при разных ориентациях отличается незначительно — на величину, сопоставимую с допустимым браком. Поэтому возможное изменение оптимальной ориентации заготовок при комбинировании на одном листе данным калькулятором не учитывается. В пользу допустимости такого решения также то, что массивные заготовки разных артикулов, для которых ориентация имеет значение, никогда не группируются на одном листе.

Справочные размеры заготовок могут быть установлены технологом исходя из той стратегии, которая кажется более предпочтительной: строго впритык или с запасом, чтобы быть уверенным, что данное для артикула *pps* гарантированно выйдет с листа. Обратите внимание на строку 33 в расчете, на существенный разрыв между рекомендацией и потребностью: 930 шт. против 748. Дело в том, что *pps* данной детали составляет 186 шт./лист. Из четырех листов получится 4 * 186 = 744 заготовки, а минимально необходимое для данного по заказу количества кронштейнов и балок — 748. Поэтому калькулятор, руководствуясь стратегией *кратно листу и меньше заказа изготовить нельзя*, округляет до 5 листов, что составляет уже 930 шт. Однако если технолог при задании справочных размеров заготовки (или при задании *pps* как константы для данного артикула — это имеет тот же смысл в этой задаче) брал значения с некоторым запасом, то, разумеется, при формировании окончательного задания для производства разумнее выбирать количество листов 4. Этот пример — иллюстрация того, что качество расчета калькулятора, качество его рекомендации, значительно зависит от качества заведенных в начале работы справочников. Калькулятор "полагает", что выхода больше *pps* с листа быть не может. Выбор стратегии задания справочных параметров и, соответственно, интерпретация результатов расчета остаются за профессиональным решением технолога.

5. ПОИСК НАИМЕНОВАНИЙ И ВЫВОД СПРАВОЧНОЙ ИНФОРМАЦИИ

Поиск наименований по номенклатуре и вывод результатов поиска осуществляется командой **`call`**, после которой через пробел указываются маркеры-подстроки. Аналогично списку *grouped* фильтровать можно по любому фрагменту наименования. Интерпретация регистров также аналогична описанной выше.

В качестве примера найдем в тестовой номенклатуре все наименования, которые содержат подстроки "ma" или "55". Результат поиска — одна или две таблицы. В первой — наименования *contents*, содержащие искомые подстроки и соответствующие параметры, подтянутые из *preforms*. *pps* рассчитывается динамически. Компоненты сборных наименований имеют общий номер как относящиеся к одному сборному изделию.

Если в первую таблицу отобралась деталь, нам может быть интересно увидеть все сборные изделия, в которых она присутствует. Для вывода этой информации служит вторая таблица отчета: начало, конец. Первая цифра в нумерации соответствует номеру детали в первой таблице, вторая — сквозной номер сборного изделия, содержащего данную деталь. Стрелка указывает на положение детали в составе сборного изделия.

Если указанные подстроки в номенклатуре не встречаются, в консоль выводится сообщение *Your call did not match any items*.

Параллельно с консолью результат работы **`call`**, если он есть, записывается в папку *calls* в корневой папке проекта, в спред, именуемый датой и временем обработки запроса и списком маркеров-подстрок.

Парсер командной строки снабжен справкой:

```
PS D:\YandexDisk\courses\Projects\calcumet> python calcumet.py -h
usage: calcumet.py [-h] [-g [GROUPED ...]] {go,call,update} [targets ...]

CALCUMET. Sheet metal calculator

positional arguments:
  {go,call,update}      Action to perform.

                        go:     Calculate number of metal sheets required to fulfill given items demand.
                                May be followed by optional GROUPED parameter.

                        call:   Search for items containing any substring from tester list and represent
                                items' parameters.

                        update: Instantiate/update json dump and loader's tab with respective processor func.

  targets               0+ list containing tester for CALL or dump(s) for UPDATE.

options:
  -h, --help            show this help message and exit
  -g [GROUPED ...], --grouped [GROUPED ...]
                        0+ list of substrings filtering items that may share a metal sheet.

Meredelin Evgeny, meredelin@pm.me, 2022
```