Анализ задания

Перед тем, как писать программный код, необходимо проанализировать данные, их структурировать и на их основе построить архитектуру работы приложения.

На входе нам доступно:

1. Список наименований видов товара
2. Список самих товаров
3. Высказывания экспертов кассиров

Сами высказывания являются абстрактными значениями «часто», «редко» и т. д. и им присваиваются значения весов для дальнейшей генерации чеков.

Как следствие у нас есть 4 набора данных, которые между собой тесно связаны. Для простоты работы с этим набором данных и организации связей между этими наборами проще всего предоставить их в виде реляционной базы данных, которая будет состоять из 4 таблиц:

1) products.csv

Таблица всех продуктов

StatementId -> statements.csv[RowId]

ProductTypeId -> product\_types.csv[RowId]

RowId – id строки

ProductTypeId – id типа продукта (хлеб, молоко…)

ProductName – имя продукта

StatementId – id высказывания (берут часто, редко, чаще всего…)

Price – цена товара

| RowId | ProductTypeId | ProductName | StatementId | Price |

| ----- | :-----------: | :----------:| :---------: | ------:|

| int | int | string | int | double |

2) statements.csv

Таблица всех утверждений кассиров и их вероятности встречи, т.е. если о товаре сказали, что его берут «часто», то в таблице есть вероятность выпадения данного продукта

В сумме они должны давать 1

RowId – id строки

Title – высказывание

Probability – вероятность этого высказывания

| RowId | title | Probability |

| ----- | :----: | ----------: |

| int | string | double |

3) product\_types.csv

Таблица типов товаров

| RowId | title |

| ----- | -----: |

| int | string |

4) shopping.csv

Список высказываний, какие продукты обычно с чем-то берут

-1 означает, что берут "еще что-то"

Все элементы идут через точку с запятой

listProductTypeIds -> product\_types.csv[RowId][]

| listProductTypeIds |

| ------------------ |

| int[] |

Так как по заданию в итоге должна генерироваться CSV таблица, то и исходные данные было принято хранить в формате csv.

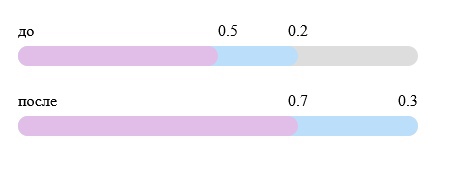
Следующим этапом является анализ самого задания. По заданию нам нужно генерировать список корзин на основании высказываний экспертов-кассиров. Эти эксперты говорят, какие типы товаров обычно с чем берут + по каждому типу говорят, с какой вероятностью берут какой продукт.

Для начала надо определиться с тем, как будет выбираться продукт по его типу. У каждого продукта есть утверждение, как часто оно берется. Эти утверждения мы преобразуем в вероятности, которые хранятся в таблице statements.csv. У нас есть генератор, который может вернуть случайное число от 0 до 1, поэтому оптимальным решением будет разместить все вероятности продуктов данного типа на одной числовой шкале от 0 до 1 и дальше при генерации смотреть, на какой участок упал генератор. Но есть ситуации, когда сумма высказываний не равна 1. К примеру, у нас есть 2 продукта и по высказываниям у одного вероятность – 0.5, а у другого – 0.2. Для того, чтоб данные значения привести к суммарной единице, мы должны просуммировать все значения и перевернуть дробь и после этого все значения умножить на данное значение. Т. е.:

Offset = Svalue-1

Valuei = valuei \* offset

Пример работы показан на рисунке ниже



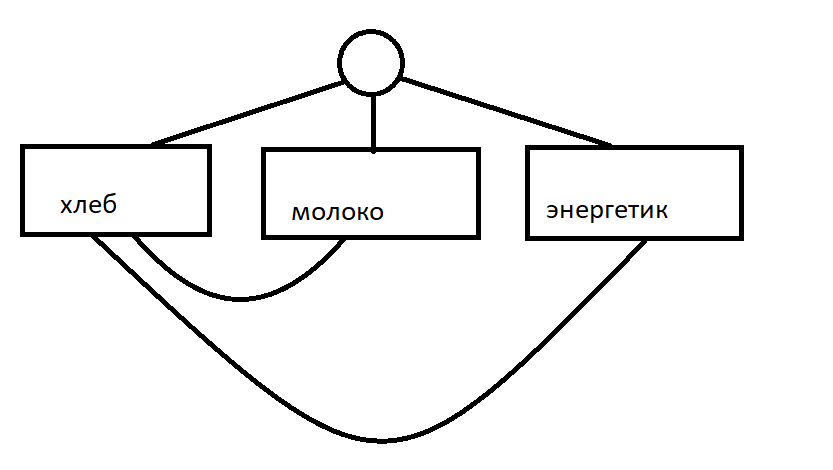
С тем как будем выбирать продукт данного типа мы определились. Дальше необходимо решить, как будет генерироваться корзина. Но для начала нам нужно построить дерево вариантов.

Для хранения данных было принято решение использовать двусвязный список, в котором у нас на старте есть доступ к любому элементу списка. Эта структура была выбрана из-за того, что если у нас есть утверждение, что с молоком обычно берут хлеб, то это также означает, что, если клиент взял хлеб, то есть большая вероятность того, что он возьмет и молоко. А доступ на старте к каждому из узлов нужно, чтоб начать генерацию корзины со случайного продукта.

Также нужно учесть, что, если есть 2 возможных набора возможной корзины, которые между собой связаны, то есть шанс, что они будут объединены. К примеру ситуация: у нас есть 2 утверждения:

1. С хлебом часто берут молоко
2. С энергетиком часто берут хлеб

Это означает, что, если покупатель берет есть шанс, что в корзине с определенной вероятностью будет как хлеб, так и энергетик. Эта связь показана на рисунке снизу



Для таких ситуаций программа должна уметь рекурсивно формировать корзину, но при этом должен быть элемент случайности, чтоб в каждой корзине не было одних и тек же типов продуктов.

Теперь мы в нашем двусвязном списке можем выбрать любой продукт, как точку отправки и рекурсивно сформировать корзину.

Реализация

Исходный код проекта лежит на GitHub:

<https://github.com/lazaryan/algorithms_and_data_structures/tree/master/lab_1>

В папке database находятся примеры исходных данных, по которым программа строит итоговую CSV таблицу.

В качестве языка программирования был выбран C++. Сборщик проекта – Cmake.

Для начала необходимо было реализовать классы для работы с CSV. Для этого сначала были описаны абстрактные классы с чистыми виртуальными функциями для документирования интерфейса.

После этого шла реализация этих классов. Класс для записи данных общий, ибо перед записью можно спокойно преобразовать данные в нужный формат. А вот с чтением другая история. Необходимо было сделать класс, который будет считывать строки с CSV файла и сразу их преобразовывать в структуру нужного типа. Для этого был реализован базовый класс, который имеет 1 виртуальную функцию – функцию, которая отвечает за преобразование строки в структуру. Она будет реализована в дочерних классах под каждую таблицу.

Дальше для каждой базы данных был создан класс, который наследуется от CSV::CSVReader и реализцет 2 обязательных метода: чтение строки и поиск структуры по id. Пример одной из таких структур:

Следующим этапом является реализация рандоайзера. Он также является общим, т.к. нам нужен рандомайзер для продуктов, для типов продуктов и т.д. Для работы класса был выбран рандомайзер mersen

На этом этапе наша программа умеет получать данные из БД и выдавать случайные значения. Следующим этапом является класс, который сможет генерировать случайный продукт для конкретного типа.

Осталось 2 последних шага. Сформировать дерево зависимостей типов и на основе него написать генератор таблицы. Это все реализовано в классе Tree.

Осталось только передать в Tree то, откуда тянуть все данные. Сгенерировать таблицу и записать ее в out.csv файл. Это все уже было реализовано в index.cpp файле

В результате генерируется таблица, где колонка – это каждый товар, а строка – чек. Последняя колонка – цена чека

Изображение выглядит как стол

Автоматически созданное описание

А в консоли пишется общая статистика по сгенерированным данным

Изображение выглядит как текст

Автоматически созданное описание