1. Базовые операции над структурами данных

Общее условие:

Задан набор символов и число n. Опишите функцию, которая возвращает список всех строк длины n, состоящих из этих символов и не содержащих двух одинаковых символов, идущих подряд.

Пример:

Для символов 'a', 'b', 'c' и n=2 результат должен быть ("ab" "ac" "ba" "bc" "ca" "cb") с точностью до перестановки.

1.1. Решите задачу с помощью элементарных операций над последовательностями и рекурсии

1.2. Перепишите программу 1.1. так, чтобы все рекурсивные вызовы были хвостовыми

1.3. Определить функции my-map и my-filter, аналогичные map (для одного списка) и filter, выразив их через reduce и базовые операции над списками (cons, first, concat и т.п.)

1.4. Решите задачу с помощью элементарных операций над последовательностями и функционалов map/reduce/filter

2. Решето Эратосфена

2.1. Напишите функцию, которая ищет **n**-ое простое число с помощью решета Эратосфена

2.2. Реализуйте бесконечную последовательность простых чисел

3. Численное интегрирование

Общее условие:

Реализовать функцию (оператор), принимающую аргументом функцию от одной переменной **f** и возвращающую функцию одной переменной, вычисляющую (численно) выражение:

$$\int_0^x f(t)dt$$

Можно использовать метод трапеций с постоянным шагом.

При оптимизации исходить из того, что полученная первообразная будет использоваться для построения графика (т.е. вызываться многократно в разных точках)

3.1. Оптимизируйте функцию с помощью мемоизации

3.2. Оптимизируйте функцию с помощью бесконечной последовательности частичных решений

4. ДНФ

4.1. По аналогии с задачей дифференцирования реализовать представление символьных булевых выражений с операциями конъюнкции, дизъюнкции отрицания, импликации. Выражения могут включать как булевы константы, так и переменные.
Реализовать подстановку значения переменной в выражение с его приведением к ДНФ.
Обеспечить расширяемость для новых операций (исключающее или, стрелка Пирса и пр.)

Код должен быть покрыт тестами, API документирован.

5. Параллельная обработка последовательностей

5.1. Реализуйте параллельный вариант **filter** (не обязательно ленивый) с помощью **future**. Количество одновременно исполняемых объектов **future** задается константой (исходя, например, из количества ядер), каждый объект отвечает за вычисление своего блока исходных данных. Разделение на блоки следует проводить «вручную», без использования готовых функций вроде **partition** (для разделения последовательности следует использовать **take** и **drop**). Продемонстрируйте прирост производительности в сравнении с обычным фильтром.

5.2.* Реализуйте ленивый параллельный **filter**, который должен работать в том числе с бесконечными потоками. Примените для задачи 2.2., покажите прирост производительности.

6. **Задача об обедающих философах.**

Реализуйте задачу об обедающих философах, используя STM в Clojure. Каждая вилка представляется ссылкой (ref), содержащей счетчик успешных использований. Каждый философ представляется потоком исполнения (thread), периоды размышления и поглощения спагетти – задержками.

Количество философов, количество «подходов» к тарелке, длину периодов размышления и поглощения спагетти задайте константами.

Проведите эксперименты с четным и нечетным количеством философов. Посчитайте количество лишних рестартов транзакций (используйте atom). Насколько решение отличается от оптимального по времени?

7.* **Бронирование авиабилетов.**

Задан набор аэропортов, авиамаршрутов (каждый характеризуется начальным и конечным аэропортом), количество билетов на каждый маршрут и их цены. На основе этих данных реализуйте программу, имитирующую одновременное бронирование авиабилетов.

В общем случае для перелета между двумя точками могут потребоваться пересадки. При этом клиента интересует бронирование на все промежуточные перелеты одновременно, либо нотификация, что это невозможно по причине отсутствия свободных билетов. При бронировании следует отдавать предпочтение той последовательности перелетов, которая дешевле (даже если количество перелетов при этом больше).

Дополните предоставленный код, используйте STM для атомарности бронирования. Настройте задержки таким образом, чтобы все клиенты (представленные потоками исполнения) смогли забронировать билеты.