Задание №1

Задача:

Разработать приложение для конвертации между единицами измерения расстояния с поддержкой метрической и имперской систем мер. Соотношения для конвертации вы можете взять из таблицы. По умолчанию, приложение должно распознавать метры (“m”), сантиметры (“cm”), дюймы (“in”) и футы (“ft”), и поддерживать конвертацию между любыми единицами.

Также необходимо реализовать возможность расширять список поддерживаемых единиц путем задания правил конвертации посредством JSON файла. Формат JSON файла - на ваше усмотрение. В качестве примера, расширьте ваше приложение добавив в файл значения для миллиметров (“mm”), ярдов (“yd”), и километров (“km”).

Входящие параметры:

Объект в JSON формате, содержащий расстояние заданное для конвертации (*distance*) со значением (*value*) и шкалой (*unit*), а также обозначение единицы для шкалы, в которую должна быть произведена конвертация (*convert_to*). Например:

{*distance*: {*unit*: m, *value*: 0.5}, *convert_to*: ft}

Выходные данные:

Объект в JSON формате, содержащий полученное значение расстояния, округленное до сотых, а также обозначение соответствующей единицы измерения, например:

{*unit*: ft, *value*: 1.64}

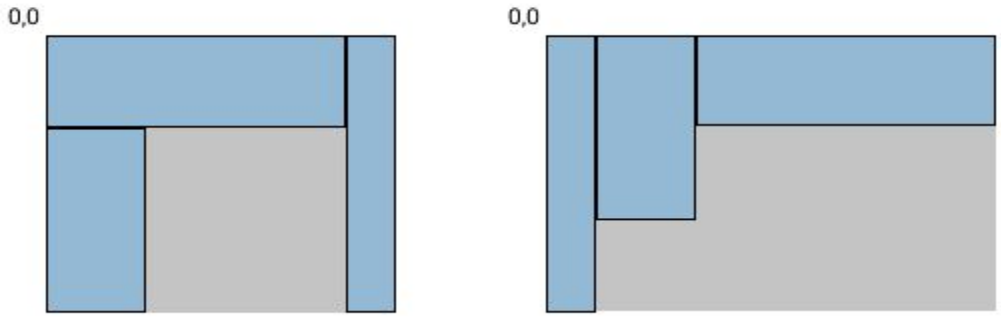
Задание №2

Задача:

Необходимо создать приложение, позволяющее вычислить оптимальный способ размещения плоских прямоугольных объектов произвольного размера внутри плоской **прямоугольной** области, так чтобы эта прямоугольная область имела минимально

возможную площадь, и при условии что объекты не могут **накладываться** друг на друга, или **вращаться**.

На рисунке ниже представлен пример оптимального (слева), и произвольного (справа) расположения трех прямоугольных объектов внутри области:



Входящие параметры:

Массив в JSON формате, где указаны целочисленные значения размеров (*width* и *height*) прямоугольных объектов, которые требуется разместить внутри области. Например:

[{*width*: 10, *height*: 20}, {*width*: 30, *height*: 10}, {*width*: 5, *height*: 30}]

Выходные данные:

Объект в JSON формате, содержащий размеры области, требуемые для размещения всех исходных прямоугольников (*area_width* и *area_height*), а также список с координатами левого верхнего угла каждого из прямоугольников и его размеры. Например:

```
{
  area_width: 35,
  area_height: 30,
  rectangles: [
    {position: {x: 0, y: 10}, size: {width: 10, height: 20}},
    {position: {x: 0, y: 0}, size: {width: 30, height: 10}},
    {position: {x: 30, y: 0}, size: {width: 5, height: 30}}
  ]
}
```

Или, альтернативно, графическое представление расположения элементов на плоскости.

Задание №3

Задача:

Необходимо разработать алгоритм, позволяющий определить порядок задействования топливных капсул ионных двигателей спутника для совершения заранее заданного ряда маневров. Капсулы имеют 5 разновидностей, и разработаны для получения прироста скорости в **2, 4, 6, 8 или 10** м/с. Каждый маневр требует получить прирост скорости **от 1 до 12** м/с. Для совершения одного маневра спутник может одновременно использовать два двигателя:

- первый - позволяет получить прирост скорости **равный значению** используемой капсулы. Например, для капсулы 4 м/с, прирост будет ровно 4 м/с.
- второй - позволяет получить прирост скорости **равный половине значения** капсулы. Например, для капсулы 4 м/с, прирост будет ровно 2 м/с.

Для совершения одного маневра запускать каждый двигатель можно максимум **один раз**. Также, для одного маневра, допускается суммарный прирост скорости **меньше** требуемого (например если запаса капсул недостаточно), но **превышение заданного приращения скорости запрещено**. Капсулы **невозможно использовать повторно**.

Алгоритм должен определять такой порядок использования капсул, при котором сумма приращений скорости по всем маневрам, и при соблюдении всех условий, **будет максимальной**, задавая таким образом наиболее точную траекторию. Количество маневров, допустимое приращение скорости на каждом из них, а также доступный набор капсул может быть произвольным.

Мы рекомендуем решать эту задачу используя **генетический алгоритм** (возможно, с определенными модификациями).

Входящие параметры:

JSON объект, содержащий массив произвольной длины с целыми положительными приращениями скорости, которые требуется достичь на каждом из маневров (*corrections*); и массив произвольной длины содержащий список доступных топливных капсул (*cells*), например:

{ *corrections*: [1, 12, 7, 1], *cells*: [8, 4, 6, 2, 2] }

Выходные данные:

JSON объект, содержащий последовательность использования капсул для первого двигателя (*main_thruster*); последовательность использования капсул для второго двигателя (*secondary_thruster*); и итоговое полученное приращение скорости (*delta_velocity*). Например:

{ *main_thruster*: [0, 8, 6, 0]*, secondary_thruster*: [2, 4, 2, 0]*, delta_velocity:* 18}

Примечания к выполнению заданий

Для задач 2-3 следует дополнительно добавить описание выбранного вами алгоритма, а также, коротко, ваши рассуждения, обосновывающие решение.

Также, во время написания программ, обратите внимание на следующее:

- код тестовых приложений необходимо разбить по логическим блокам, так чтобы он был компактным, легко читаемым, и не содержал повторений
- приложения должны корректно реагировать на широкий спектр возможных входных значений, и обрабатывать исключительные ситуации
- все задачи должны быть решены наиболее оптимальным образом, с наименьшим использованием ресурсов памяти и процессора

Выполненное задание (исходный код) присылайте на email hr@sysgears.com, в качестве темы письма укажите: “Выполненные задания. [Имя Фамилия]”.

Дополнительно к письму необходимо прикрепить резюме.