The Ration Predicament

In this story you are the official that is responsible to allocate provisions to various ration shops across the country. To keep the challenge simple, let's assume that the ration shop only deals with one kind of provision, viz. Rice.

You have control over a godown where you pack rice in various sizes(size variants), namely 500g, 1kg, 2kg and 5kg. Each ration shop that you service has only stipulated storage size which cannot be exceeded. For. e.g. a shop can only hold 100 units of 500g, 200 units of 1kg, 200 units of 2kg and 200 units of 5kg at a point in time.

Logistics Constraint: To simplify and optimise logistics, you ship each variants only in minimum numbers, i.e. 500g has a minimum quantity of 50 units with further top ups of 20 units. The 1kg and 2kg variant has a minimum allocation constraint of 50 units with further top ups of 10 units. The 5kg variant has a minimum allocation constraint of 30 units with further top up of 10 units.

Here are some samples

Variants	Acceptable allocations	Invalid Allocations
500g	50,70,90….	10,20,30,40,60,80,100…
1kg	50,60,70,80…	10,20,30,40
2kg	50,60,70,80…	10,20,30,40
5kg	30,40,50,60…	10,20

People can buy from the ration shop in quantities that they like as many times they want in a month, but have a monthly budget of 5kg/person/family. As reward for you efforts you are paid a fixed commission on every purchase bill that the customer makes (irrespective of the quantity purchased)

The Challenge

Given one store, can you perform an allocation (for one day) given you have the following at the start of every day

1. Current capacity available by size. E.g. the store can hold a further 75 units of 500g rice, 90 units of 1kg, 87 units of 2kg, 35 units of 5kg.
2. Sales forecast for the day in kgs. E.g. today 100kgs of rice would sell in the store.
3. Logistics constraints - described in the section above.

Can you write a program that can perform smart allocation that plays with the logistics and store size constraint while also maximises the revenue obtained via commission. When there is a residue to allocation, suggest the best option of either allocating less than the forecast or send more than the forecasted quantity.

Hint: Revenues obtained via commissions would be high with more number of transactions. So the smaller size variants have to be filled on priority as given the size and restriction on monthly rations, larger transactions means possible lower number of transactions.

Submission

Use the programming language of your choice and submit your implementation source code that can be compile and run on Linux. Include instructions to build and execute with the submission

Function signature: `fun allocate(quantity_in_kgs, sizes, logistics_constraints, topups, available_capacity)`

Output format: A Hash/Map with the following detail

* The reminder in kgs that was not allocated. If the reminder is negative, that suggests that the system has over allocated.
* Variant as a list, order is important as order defines the association in the other keys
* UOMs: the logistic constraint in units of variant (note that this is not in kgs but units), maps to variant by order
* Allocations as a list: Allocated units, maps to variant by order.
* Post Alloc Capacity: The pending capacity in the store post allocation, maps to variant by order.

Sample input/output

There can be multiple allocations to a given case, in case of which the one making sure of the least reminder and max lower variant allocation is considered the best. The below are for reference, incase you can do better, make sure to include your comments. The allocation would be tested for other use cases on submission so try to cover more than just whats below in your tests/development.

Case 1:

```
< allocate(123, [0.5, 1, 2, 5], [50,50,50,30], [10,10,10,10], [63, 125, 40, 20])
>> {“reminder":3,"variant":[0.5,1,2,5],"uoms":[50,50,50,30],"allocations":[60,90,0,0],"post_alloc_capacity":[3,35,40,20]}
```

Case 2:

```
< allocate(146, [0.5, 1, 2, 5], [50,50,50,20], [10,10,10,10], [103, 15, 27, 20])
>> {“reminder”:1,”variant”:[0.5,1,2,5],”uoms”:[50,50,50,20],”allocations":[90,0,0,20],"post_alloc_capacity":[13,15,27,0]}
```

Case 3:

```
< allocate(289, [0.5, 1, 2, 5], [50,50,50,20], [10,10,10,10], [60, 65, 55, 20])
>> {"reminder":-1,"variant":[0.5,1,2,5],"uoms":[50,50,50,20],"allocations":[60,60,50,20],"post_alloc_capacity":[0,5,5,0]}
```

Case 4:

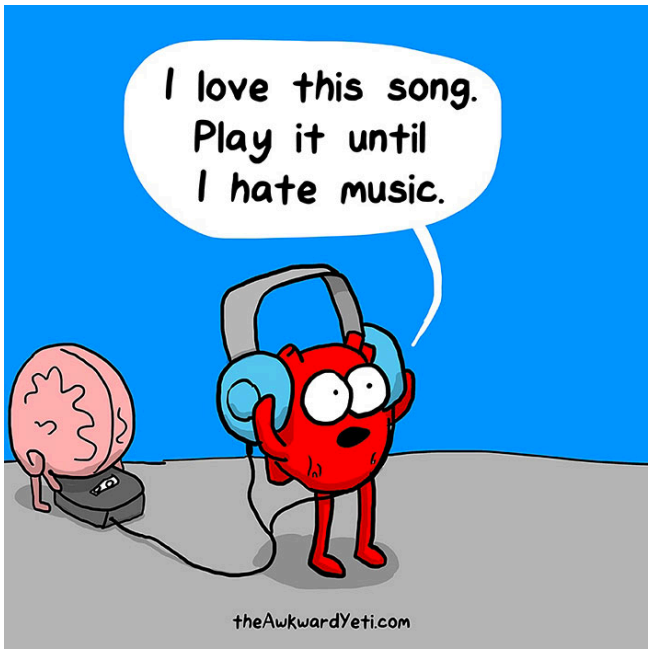
```
< allocate(457, [0.5, 1, 2, 5], [50,50,50,20], [10,10,10,10], [60, 65, 55, 20])
>> {"reminder":167,"variant":[0.5,1,2,5],"uoms":[50,50,50,20],"allocations":[60,60,50,20],"post_alloc_capacity":[0,5,5,0]}
```

What are we looking for?

Clean and simple code that self documents itself. No out of box algorithms or libraries should be used.

Bonus:

* Lower number or no loops
* Code that is test covered. The more the number of unique cases you can find, higher your bonus score
* Automation of Build, Test and Packaging.



Happy Coding and happy Allocating!!
- Team Delium