Bread Buying Problem

Problem description

You live in a remote settlement where bread sellers come through periodically at irregular intervals. Whenever you buy fresh bread, it lasts for 30 days until it becomes too stale to eat.

Your family eats one loaf of bread per day.

You are given a calendar of when the bread sellers will be visiting over the coming days and the price each bread seller will charge per loaf. You currently have 10 fresh loaves, and at the end of the calendar you'll get a bunch of free bread, so you won't need to have any left on hand.

Write a function that tells you how much bread to buy from each of the sellers

Input:

total_days, an integer, the number of days in the calendar until the free bread arrives

sellers, a list of pairs of integers (day, price). Each pair represents one bread seller
The day is how many days from the start until the seller arrives
The price is the price to buy each loaf of bread from this seller, in pennies

Output:

Purchases, a list of integers of the same length as sellers. Each integer is how many
loaves you should buy from each seller

Or None, if there is no solution that does not force your family to eat stale bread at some point
You should output the purchase plan that minimizes the total cost. In case of ties, output the
plan that requires buying from the fewest number of different sellers, and within those choose
the plan that buys more bread earlier
Requirements:
Please write a well formatted, easily readable python function
calculate_purchasing_plan(total_days, sellers) in a file called
your_name.py that solves this problem using only the standard python (i.e. no packages
required).

Please include a comment at the top of the file with your name and email as well as a short
paragraph describing why the algorithm you've provided should work.

Example:

calculate_purchasing_plan(60,[(10,200), (15,100), (35,500), (50,30)]) -> [5,30,5,10]

Analysis

Mathematical model

This problem can be described by **Objective Function**, which is typically used to represent maximizations of profits in business.

A simplified model of the function for our particular case can be represented as:

$$C(p,q)=\sum_{i=0}^{N}(pq)$$

where:

C = the cost function
p = price of bread the seller is offering
q = quantity of bread the User is buying
N = number of purchases the User is making

Now, to calculate the sum, there are a few restrictions that apply (like bread expiration, etc) but we are omitting them to simplify the model.

Therefore, to find the solution to our problem we have to minimize the C(p,q) function.

Possible solutions

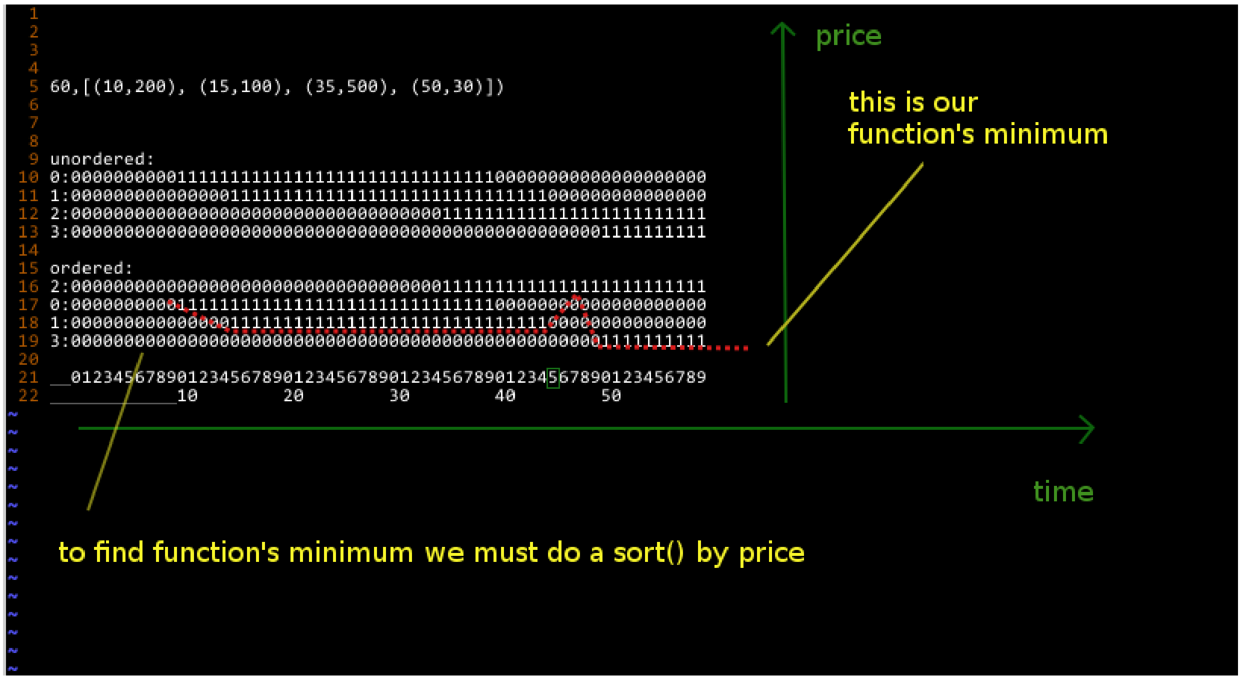
To design the solution to our function minimization problem we can use different techniques:

- Decision trees
- Reinforcement Learning
- Linear programming

If our problem would be bigger (like thousands of providers, thousands or millions of products each with different price, thousands of product categories with an option to be fungible, etc… we would be using some stochastic process like markov decision process, or maybe a montecarlo search or maybe even using machine learning techniques like Reinforcement Learning. But this problem is simplified, so we are going to use linear programming.

Design

To facilitate the search for the function minimum, we are going to create a matrix of bread availability per day, that is going to give us the answer to the question: is there going to be some bread available to eat at a particular day? We will generate this matrix using days as columns and providers as rows. Here is how a sample matrix looks like:



```
60,[(10,200), (15,100), (35,500), (50,30)])



unordered:
0:000000000011111111111111111111111111111100000000000000000000
1:000000000000000111111111111111111111111111111000000000000000
2:000000000000000000000000000000000001111111111111111111111111
3:000000000000000000000000000000000000000000000000001111111111

ordered:
2:000000000000000000000000000000000001111111111111111111111111
0:000000000011111111111111111111111111111100000000000000000000
1:000000000000000111111111111111111111111111111000000000000000
3:000000000000000000000000000000000000000000000000001111111111

__012345678901234567890123456789012345678901234567890123456789
____________10        20        30        40        50
```

price
this is our function's minimum
time
to find function's minimum we must do a sort() by price

Now, the minim values of the Cost Function are clearly seen in the matrix. All that is left to do is to code the algorithm to purchase the bread according to the data in the matrix.

Implementation

Running the program:

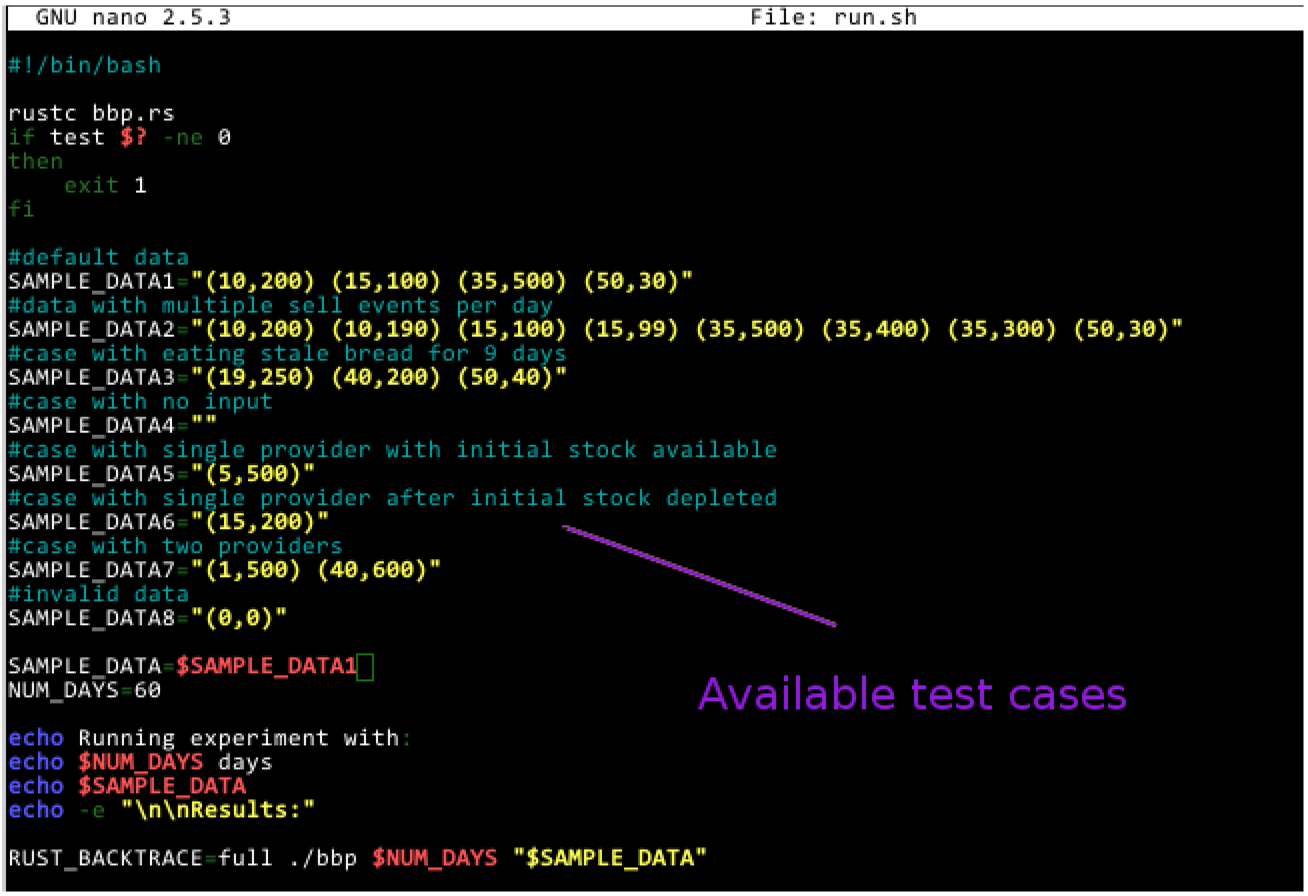
```
nickhash@lap:~/bbp$ ./run.sh
Running experiment with:
60 days
(10,200) (15,100) (35,500) (50,30)


Results:
5,30,5,10
nickhash@lap:~/bbp$
```

Implementation details:

- If stale bread is eaten, we output None for each day, so in these particular cases, the output vector doesn't match input vector. It is done this way because we can't accumulate None values, it wouldn't be clear to the user how long is stale bread eaten.
- Consumption process is executed before purchasing process, so it means, the bread starts expiring from the next day after purchasing it.

Available test cases:



```
  GNU nano 2.5.3                                      File: run.sh

#!/bin/bash

rustc bbp.rs
if test $? -ne 0
then
    exit 1
fi

#default data
SAMPLE_DATA1="(10,200) (15,100) (35,500) (50,30)"
#data with multiple sell events per day
SAMPLE_DATA2="(10,200) (10,190) (15,100) (15,99) (35,500) (35,400) (35,300) (50,30)"
#case with eating stale bread for 9 days
SAMPLE_DATA3="(19,250) (40,200) (50,40)"
#case with no input
SAMPLE_DATA4=""
#case with single provider with initial stock available
SAMPLE_DATA5="(5,500)"
#case with single provider after initial stock depleted
SAMPLE_DATA6="(15,200)"
#case with two providers
SAMPLE_DATA7="(1,500) (40,600)"
#invalid data
SAMPLE_DATA8="(0,0)"

SAMPLE_DATA=$SAMPLE_DATA1
NUM_DAYS=60

echo Running experiment with:
echo $NUM_DAYS days
echo $SAMPLE_DATA
echo -e "\n\nResults:"

RUST_BACKTRACE=full ./bbp $NUM_DAYS "$SAMPLE_DATA"
```