Solution by Khongorzul Khenchbish
Time spent: 3 hours
Language used: python3

Problem Description:

We have a dataset that includes a stream of data about the color of cars passing an individual point on a road. Cars continuously pass the point with one unit interval between them. For any pair of colors (A, B), we wish to know the shortest distance between all pairs of cars of colors A and B in the data set.

You will receive as input a space separated string of car colors, you can assume that the string contains only the characters a-z and spaces. You can also assume that the data is all available at the start of your program execution and can be read once and interrogated many times. The sequence of observations could be very long (but you can assume short enough to fit into memory). We have provided a variety of sample inputs and outputs at the end of the document for you to use whilst testing.

There are lots of complications outside the scope of this exercise so please comment on any assumptions in your code. When submitting if there are any points that could affect accuracy / speed that might need further consideration in future work please list what these. If you have any potential solutions to these then a brief description would also be good.

Example Input:

Input: red red blue red
Parameters: red, blue
Output: 1

Input: red yellow blue
Parameters: red, blue
Output: 2

Input: yellow yellow blue yellow yellow red
Parameters: red, blue
Output: 3

Edge cases:

1. parameters is empty OR length is one instead of pair.
2. colors array is empty OR colors array has only one element.
3. colors array has more than one elements but:
    - none of the colors in parameters are present in the colors data set.

- when only one of the colors in the parameters is present in the colors data set.
- if two are the same color such as [blue, blue] in parameters then the number of blue cars in the dataset should be more than one.

Solution 1. Naive approach using Brute force

The idea of pairs in parameters is used. The outer loop selects the first color of the pair and the inner loop traverses the array in search for the other pair and takes the minimum distance between them using the index position.

Time complexity: $O(n^2)$ because of the nested loop.
Space complexity: $O(1)$ no additional space used.

Pseudo code:

```
int shortest_distance(param)

str[] colors = read input
if(param.len != 2) then
    raise error "parameters should be pairs"
if(colors.len < 2) then
    raise error "data stream should have more than 2 records"

int shortestDist := INT_MAX
for i := 0 in colors.len
    for j = i + 1 in colors.len
        if param[0] = colors[i] and param[1] = colors[j] OR
           param[1] = colors[i] and param[0] = colors[j] then
            shortestDist := min(shortestDist, j-i)

if shortestDist < INT_MAX then
    return shortestDist
else
    raise error "No matching pairs found"
```

Solution 2. Improved solution using pointers

Instead of using a nested loop, we could traverse the array once using two pointers.
We will need $previous$ pointer to save position of the first found color and $current$ pointer to keep track of the rest of the cars. We start the traversal with the current pointer traversing from

the left to the right and we stop(not exiting the loop) when encountering any of the pair colors in parameters.

The initial value of the $previous$ index is -1 since none of the pairs are in the dataset yet. As the first time it finds any color of the pairs inside the datastream it will assign the $current$ value to $previous$ and from here the value of the previous index will always be different than -1.

On the next iterations, when it finds the next color from the pairs, it updates the shortestDistance if the color at the current index is different from the color at the previous index(Here we will have the previous index already). In other words, if the car colors are different colors from the pairs, we should measure the unit intervals between them. Else if the value at $current$ index is similar to value at $previous$ index, we move the $previous$ index to $current$, because it is closer. In other words, when we find similar colored cars, we should keep the current car as the latest and the closest one.

At the end of the loop, shortestDistance should be updated if there were at least one pair in the datastream of car colors.

Time Complexity: O(n). Only one traversal.
Space Complexity: O(1). no extra space is needed.

Python3 code:

```
import sys

def shortest_distance(params):

   is_empty(params, "parameters")
   pairs = params.split(", ")
   validate_length(pairs, "parameters")

   inp = input()
   is_empty(inp, "car data ")
   colors = inp.split(" ")
   validate_length(colors, "car data ")

   # previous index
   previous=-1
   # shortest distance
   shortestDist = sys.maxsize;

   for current in range(0, len(colors)):
       if(colors[current] ==pairs[0] or colors[current] == pairs[1]):
           # If the element at current index matches with
```

```
            # the element at previous index, then update the shortest distance
            if(previous != -1 and colors[current] != colors[previous]):
                shortestDist = min(shortestDist,current-previous)

            #update the previous index
            previous=current

    if(shortestDist == sys.maxsize):
        # distance has not found
        return -1
    return shortestDist

def validate_length(list, message):
 if len(list) < 2:
    raise ValueError(message + " should have length more than one")

def is_empty(string, message):
 if(string==""):
    raise ValueError(message + " should be non empty")
```