Problem Statement: Calculate read coverage at positions of interest ("loci").

Design approach:

The data can be segregated by using the length of string as a factor to product multiple hash map. The current data has reads from length of 5 to 9 characters (9: 1840415, 8: 138523, 7: 1477, 6: 163, 5: 6 values). We can generate multiple hash maps of reads data stored separately and reducing the hashing dataset. The reads can have multiple overlapping data, so we can keep a counter to keep count of occurrences. The reads data value is the best use case for the key and value can be the counter. The loci data is read and searched in hash map to retrieve the read coverage/count of position of interest. As the loci data will exponentially increase the runtime will not be greatly affected as the value is fetched from hash map.

Customizable Variables/Assumptions:

1) Location of read and loci file. As a default allocation, it assumes the operating system is linux and data is stored locally to the folder containing python file.
2) Output file creates a new file lociOP.csv to store output.

Function definitions:

1) load_data(): Loads initial data into python workspace and perform string manipulation to convert data into an array for easy access.
2) preprocessing_data(): The function converts data into hash map and uses add_ele_to_hashmap() function to run data addition to hash map and maintain the value counter.
3) search_loci_reads(): The function performs search of loci read values in the reads hash map and saves the values in loci_data_values[] list. It uses assign_value_loci() function to reduce code repetition and assign values to the loci_data_values[] array.
4) file_write_loci(): The function performs file write process to add corresponding values of loci and their respective count to the loci file.

Procedure:

1) Main function initializes the UnittestLociValues class to perform evaluation. Instantiation of the class includes the following:
    a. The value of sanity check array is instantiated.
    b. It initializes FindLociReadsMatch class with data location variables as input and assigns it to search_loci local object.
    c. Local object search_loci executes load_data() function.
    d. Local object search_loci executes preprocessing_data() function.
    e. Local object search_loci executes search_loci_reads() function.
    f. Local object search_loci executes file_write_loci() function.
2) Main function initializes runtest_loci_values() function.
    a. Result list matrix is created.
    b. Result of search_loci output is compared with sanity check to create correct (True) and incorrect (False) values representation of unit test.
    c. Data is represented as True (correct output) and False (incorrect output) variables.

Object flow diagram: Can be found in ObjectFlowDiagram.png file.

Time complexity:

R= number of lines in reads files

K= range of different numbers

L= number of lines in loci files

Preprocessing: O(K) for every element

Loci Search: O(1) for every element

Profiling:

The profiling has been performed on a windows system using cProfile (python -m cProfile bl_prog.py) with the following results:

```
287606879 function calls (287606615 primitive calls) in 187.320 seconds
```

Functions with maximum run time:

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)

1980588    1.365    0.000    1.365    0.000 {method 'split' of 'str' objects}

1980584  141.156    0.000  181.543    0.000 bl_prog.py:59(add_ele_to_hashmap)

1    0.245    0.245    5.917    5.917 bl_prog.py:27(load_data)

2    0.129    0.064    0.129    0.064 {method 'splitlines' of 'str' objects}

281649879   40.387    0.000   40.387    0.000 {method 'get' of 'dict' objects}

1982029/1981996    0.187    0.000    0.187    0.000 {built-in method builtins.len}
```