**Problem: Deciphering Code**

**Details:**

**To Run Strategy(1)**

bash Run.sh <filename to be tested> <k size>

**To Compile And Run Strategy(2)**

g++ --std=c++11 DecipheringCode.cpp -o DecipheringCode

./DecipheringCode <filename to be tested> <k size>

**Algorithm:**

I solved the problem in three major steps:

Step 1: Strip off all the **non-alphabetical** characters.

Step 2: Form unique tuples of given size k and fetch their distribution.

Step 3: Sort the distribution of k tuples in descending order.

**Design:**

**Strategy 1 Design (Unix tools and Python)**

I chose a combination of Unix tools and Python in this strategy. Unix tools made it easier to strip off all the non-alphabetical characters in a one line code and to sort the distribution of tuples in descending order. Python was useful to extract the tuples of given size k from the data. There was no waste of time declaring the types of arguments or variables which made this strategy simplest and shortest way of solving the problem.

**Strategy 2 Design (C++)**

In this design, I chose C++ as coding language to make the problem little challenging with the idea in mind that this design would improve the speed of the process to get results, if compared with strategy 1. The main motive to solve same problem with different coding languages was to do the impact analysis of the programming languages on the run time, even though the input and results are same.

In C++, I used two types of Standard Template Library (STL), **map** and **set**.

* A m**ap** is an associative container that is used to store the elements in pair. Hence I used map to store pairs consisting of unique tuples of given size k (as keys) and their distribution(as values).
* **Set** was used to sort tuples by their distribution (values) in descending order. Since map is already in the **sorted order** of keys, we cannot sort map by values. Hence I used “set” for that purpose.

**Validation of correctness:**

Results came out to be correct.

* I tested the Captain Kid example with both the codes and the results were same. Also, I manually calculated the number of ‘THE’ possible in Captain Kid with k=3. All the results showed ‘THE’ distribution as 8. That is how I checked the correctness.
* Also, manually I crafted a data file with two lines as follows:

**AZkLaonnn1008dnnnn1888bbbbsj**

**AkLaonnn1008dnnnn1888bbbbsjaz**

* I tested this with k=3
* I checked if my result file has the tuples **‘azk’** and **‘jaz’** which are unique tuples and signifies that correct start and end is taken by the program and all the alphabets are converted to the lower case.

**Further tests and comparisons:**

* First, I tested both the codes with Captain kid example for k=3 and compared the result files using “vimdiff ResultFromStrategy1.txt ResultFromStrategy2.txt”. Results had no difference. But since Captain kid example data is very small, so it’s difficult to measure the difference in run time from both the strategies.
* Second, I tested both the codes with a little larger file of size ~4.8MB. Using this dataset I first matched the results from both the strategies and no difference was observed. Next I analyzed the run time for both the strategies and certainly a difference was observed. I checked the wall clock time for running each strategy via code. The table below gives a detailed analysis on the run time.

**Data used :maori\_utf8.txt (~4.8MB) (Data file maori and results are attached in the mail)**

**Machine used: hopper**

**With maori\_utf8.txt (~4.8MB) (Data file maori and results are attached in the mail)**

|  |  |  |  |
| --- | --- | --- | --- |
| K size | Time taken by Strategy1  (Unix tools and Python) | Time take by Strategy 2  (C++) | Difference in time |
| 3 | 14 sec | 3.43 sec | 10.57 sec |
| 5 | 17 sec | 5.6 sec | 11.4 sec |
| 9 | 35 sec | 12.37 sec | 22.63 sec |
| 10 | 40 sec | 16.41 sec | 23.59 sec |

**Conclusion**

* The results above clearly shows that choice of coding language certainly makes a difference in the run time. Hence the choice of coding language should be made after gaining an understanding of the target data.
* If our target is to work with smaller datasets, language choice doesn’t matter so much. The only concern is correctness of results in that case. But if the target is big data like genomic data or anything of that sort, language choice definitely matters.
* Along with correct results, the speed of an algorithm is also equally important. Hence the final lesson was that there are ample ways to solve a problem using different coding techniques; But it is important to understand the target data before making a choice of coding language to implement an algorithm.