2024 SoCal ICPC Letter Distance

Mark E. Lehr Ph.D.

November 21st, 2024

1 Preface

The International Collegiate Programming Contest (ICPC) is an annual, multi-tiered programming competition for college students, organized by the ICPC Foundation. Teams of three students, representing their universities, compete to solve algorithmic problems within a limited time, fostering collaboration, creativity, and problem-solving skills. The contest is considered the "Olympics of programming" and is the oldest, largest, and most prestigious programming competition in the world.

Not one Community College team solved the Letter Distance Problem during the 2024 Southern California Regional Intercollegiate Programming Competition. This includes many Universities as well since the problem was only solved by about 25 percent of the combined teams. The concepts utilized to solve the problem are covered in courses taught at Riverside City College which are associated with Discrete Math CSC 7 and Data Structures CSC 17c. Implementing the solution in code can easily be accomplished after completion of CSC 5 C++ Programming Concepts and more competently with CSC 17a C++ ADT and Objects. Our students have the concepts typically by the end of their sophomore academic year when they transfer, but lack this knowledge prior to the competition since it is held in the Fall which would be the beginning of their sophomore year.

2 Problem Description [1]

You are given a string $a_1, a_2 \dots a_n$ consisting of n upper-case letters of the English alphabet (A–Z). You want to calculate the following sum:

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j-i)^2 [a_i \neq a_j] \tag{1}$$

$$Boolean->[a_i \neq a_j] = \begin{cases} 1 & \text{condition is true,} \\ 0 & \text{condition is false.} \end{cases} \tag{2}$$

The input to your program is a single line of at least 1 and not more than 10^6 upper-case English letters. Print one line with the value of the above sum. Because the value of the sum can be very large, print the value modulo (10^9+7)

Table 1 shows the sample Inputs/Outputs which are meant to test the code at a rudimentary level. They do not test the boundary conditions. For instance, the following table shows the answers for 3 possible letter sequences. What is not exercised with these samples are the modular arithmetic that will be required to bound the answer or the efficiency of the implementation needed to pass the ICPC judge.

Table 1: Test Cases for Letter Distance

Sample	Sample Input	Sample Output
1	ICPC	16
2	SOCAL	50
3	ZZZZZZ	0

2.1 Analysis of Data Types

These are some salient points with respect to code implementation in C++ and the limitations of the data types. Another language such as Python does not have limitations with respect to integer data types. But, then again, Python is much slower in terms of execution than C++

1. If n is 10^6, then the double sum represents a triangular matrix with contents 10^6 equating to a sum $\approx ((10^6)^2)^2 = 10^{24}$. This explains the modular arithmetic requirement. Not even a Long Long integer type 10^{18} will work. However, to keep the $((j-i)^2)$ within bounds a Long Long is required given the size of the modulo.

2. The input data only requires a char array. The largest static char array is about $5x10^5$ So, it is required that a dynamic array or vector is implemented to hold the number of characters.

2.2 Analysis of Timing

These are some salient points with respect to code run time as well.

1. This algorithm requires a double sum/nested loop. The asymptotic analysis of this is $O(n^2)$. Unfortunately, an equivalent sorting problem would require about an hour of computer resource time. The contest only allows seconds of compilation and run time.

2. In order to reduce the time complexity, requires a completely different approach and reformulation of the problem. This would entail the solution of the complement and partition of the character set.

3. There are formulas that can be used to solve the sum, or if you do not want to derive, then the approach would be to use a single sum transformation.

4. And finally, given that the complment only requires like characters, n can be split to n/26. A speed up of 26 times is possible $26 * O((n/26)^2) \propto n^2/26$. It is still asymptotic $O(n^2)$ but the scaling coefficient has been reduced by 26 or an order of magnitude which might fit the timing constraint.

2.3 Approach

The solution can be implemented with a simple reformulation of the problem in the following way.

2.3.1 The Indices

The first transformation will be directly related to the array implementation on a computer. We can see that an array of size n utilizes indices of [0,n-1]. In addition, when j=i the measure is 0.

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j-i)^2[a_i \neq a_j] = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i \neq a_j] \tag{3}$$

This form is exactly the same sequence used to sort data with $O(n^2)$ algorithms.

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}[a_i > a_j]swap(a_i, a_j)$$

Given the equivalence it is easy to see the difficulty with implementation. This problem for the maximum array size will lead to a timeout indication.

2.3.2 The Complement

Now let's formulate a set theory equivalence.

$$\eta(A^{\neq}) = \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i)^2[a_i \neq a_j] = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i \neq a_j] \tag{4}$$

$$\eta(\Omega) = \eta(A^{\neq}) + \eta(A^{=}) \tag{5}$$

$$\eta(\Omega) = \eta([a_i \neq a_j]) + \eta([a_i = a_j]) \tag{6}$$

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2 = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i \neq a_j] + \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i = a_j] \tag{7}$$

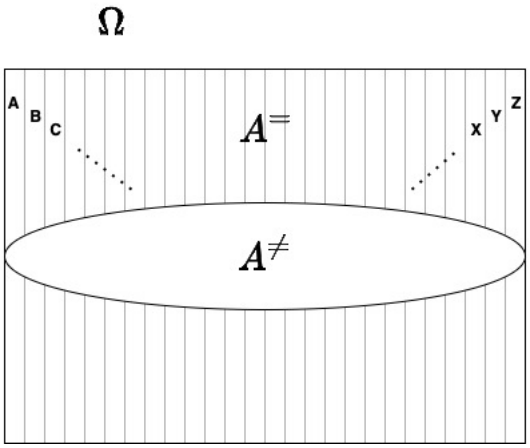

Figure 1: The Sample Universe of Letter Distance

So, of course restate with the complement

$$\eta(A^{\neq}) = \eta(\Omega) - \eta(A^{=}) \tag{8}$$

$$\eta([a_i \neq a_j]) = \eta(\Omega) - \eta([a_i = a_j]) \tag{9}$$

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i \neq a_j] = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2 - \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i = a_j] \tag{10}$$

We have shown the equivalence now solve the reformulated problem. The question to answer is why would this be better than the original problem? $\eta(\Omega)$ can be solved with a single sum or closed form solution and the $\eta(a_i = a_j)$ can be partitioned to speed the solution by a factor of 26 representing the capital letters from [A-Z].

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2 - \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2[a_i = a_j] \tag{11}$$

2.3.3 Single Sum Equivalence

Determine a single sum or simple closed form solution to the following:

$$\eta(\Omega) = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(j-i)^2 \tag{12}$$

By examining the double sum, we can make a simple transformation to a single sum by inspection,

$$\eta(\Omega) = \begin{cases} 1^2 + 2^2 + 3^2 + \cdots + (n-3)^2 + (n-2)^2 + (n-1)^2 + \\ 1^2 + 2^2 + 3^2 + \cdots + (n-3)^2 + (n-2)^2 + \\ 1^2 + 2^2 + 3^2 + \cdots + (n-3)^2 + \\ \vdots \\ 1^2 + 2^2 + 3^2 + \\ 1^2 + 2^2 + \\ 1^2 \end{cases} \tag{13}$$

which gives the following.

$$\eta(\Omega) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (j-i)^2 = \sum_{i=1}^{n-1} (n-i) * i^2 \tag{14}$$

However, we will implement with the index starting at 0 since the sum will be the same and we will hash and store the data into the parsed arrays with the same loop.

$$\eta(\Omega) = \sum_{i=0}^{n-1} ((n-i) * i^2 \wedge A^{=}[a[i]] = i) \tag{15}$$

2.3.4 Closed Form Solution

If we want a closed form solution then using

$$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6 \tag{16}$$

and

$$\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4 \tag{17}$$

Which gives the following.

$$\eta(\Omega) = n^2(n^2-1)/12 \tag{18}$$

And to cover the repetitive significance of a double sum simplified to a single sum to a closed form solution.

$$\eta(\Omega) = \sum_{i=1}^{n} \sum_{j=i}^{n} (j-i)^2 = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (j-i)^2 = \sum_{i=1}^{n-1} (n-i) * i^2 = n^2(n^2-1)/12 \tag{19}$$

2.3.5 Modular Arithmetic

Given the Quotient-Remainder Theorem let's restate and define $n \in \mathbb{Z}$ and d $\in \mathbb{Z}^+$ then there exists unique integers q and r such that

$$n = dq + r \quad \text{and} \quad 0 \leq r < d \tag{20}$$

To Provide a name to each term:

1. n is the dividend
2. d is the divisor
3. q is the quotient and
4. r is the remainder

Now, let's define how the mod operator as represented in Computer Science code using a language like C++ appears. Given the above Theorem and definitions

$$n/d = q \quad \text{and} \quad n\%d = r \tag{21}$$

or in words, n divides d = q and n mod d = r.

The four basic rules we can apply to the Letter Distance problem for the mod operator would be

$$Rules = \begin{cases} (a+b)\%m = (a\%m + b\%m)\%m \\ (a-b)\%m = (a\%m - b\%m)\%m \\ (a*b)\%m = (a\%m * b\%m)\%m \\ ab\%m = 1 \quad \text{implying} \quad b = a^{-1} \end{cases} \tag{22}$$

For instance, let's apply the inverse rule to closed form solution where $m = (10^9 + 7)$

$$\eta(\Omega) = (n^2(n^2-1)/12) \tag{23}$$

$$\eta(\Omega)\%m = (n^2(n^2-1)/12)\%m \tag{24}$$

$$\eta(\Omega)\%m = (83,333,334n^2(n^2-1))\%m \tag{25}$$

since $(83,333,334 * 12)\%m = 1$ implying $12^{-1} = 83,333,334$ under %m. And of course to keep it bounded the full implementation would be

$$\eta(\Omega)\%m = (83,333,334(n^2(n^2-1)\%m)\%m)\%m \tag{26}$$

However, due to time pressure which the competitors are under, it is probably best to simply calculate the single sum.

3 Conclusion [2]

This problem used coding concepts associated with data types, data ranges, and mathematical concepts. The results of the competition show that there were 126 attempts at solution but only 23 successful teams. It took 93 hours for 23 successes or averaging about 4 hours per solution but the fastest solution was done in 18 minutes with only 1 try.

References

[1] Southern California Region. Southern California Regional Problem Set problem 9. http://socalcontest.org/history/2024/SCICPC-2024-2025-ProblemSet.pdf. Accessed: 2024-11-28.

[2] SoCalRegion. Southern California Regional Preliminary Results problem 9. http://socalcontest.org/history/2024/SCICPC-2024-2025PreliminaryResults.html. Accessed: 2024-11-28.

4 Code

```
/*
 * File:     main.cpp
 * Author:   Dr. Mark E. Lehr
 * Date:     Created on November 17th, 2024, 7:01 PM
 * Problem:  2024 Southern California ICPC November 16th @RCC MLK-CIS Bldg
 *           Problem 9 Letter Distance
 * Purpose:  Turn double sum long long into single and sub-sums
 *           sum(0,n-2)(sum(i+1,n-1)(j-i)^2 [0-1] =
 *                      sum(0,n-2)(n-i)*i*i - subArray
 */

//System Libraries
#include <iostream>  //I/O Library
#include <cstdlib>   //Rand Library
#include <ctime>     //Time Library
#include <list>      //List STL Library
#include <vector>    //Vector STL Library
using namespace std;

//Function Prototypes
unsigned char *fillAry(int);                    //Dynamically fill a char array
void prntAry(unsigned char *,int,int);          //Print the array
long long int dSum(unsigned char *,int,int);    //Double Sum Calculation
```

```
long long int sSumLL(unsigned char *,int,int);   //Single sum with Linked Lists
long long int sSumV(unsigned char *,int,int);    //Single sum with Vectors
long long int smalldSum(int *,int,int);          //Small Partitioned Double Sum
long long int smalldSum(vector<int> &,int);      //Small Partitioned Double Sum
int inverse(int,int);                            //Modular Inverse inefficient
bool sSum(int,int);                              //Single sum to formula

//Program Entrance/Execution
int main(int argc, char** argv) {
    //Set the seed once for pseudo-random with current time
    srand(static_cast<unsigned long long int>(time(0)));

    //Initialize the array with random characters or use the 3 sample cases
    //in comments.  Just un-comment, run, then comment out again.
    int size=100000,perLine=10,modIt=1000000000+7;
    unsigned char *array=fillAry(size);

    /*
    size=4;array[0]='I';array[1]='C';array[2]='P';array[3]='C';//16
    size=5;array[0]='S';array[1]='O';array[2]='C';array[3]='A';array[4]='L';//50
    size=5;array[0]='Z';array[1]='Z';array[2]='Z';array[3]='Z';array[4]='Z';//0
    */

    //Runs commented with output, or runs with timing as the investigation
    //Prlong long int the array if sample or test case
    //cout<<"The Original Array"<<endl;
    //prntAry(array,size,perLine);

    //Does a single sum = the closed form
    sSum(size,modIt);

    //Test the time for the partition using an STL linked list
    int beg=time(0);
    long long int ssum=sSumLL(array,size,modIt);
    if(ssum<0)ssum+=modIt;
    cout<<"The single for-loop sum and Partitioned List = "<<ssum<<endl;
    int end=time(0);
    cout<<"Time for single for-loop sum Partition List = "<<end-beg<<endl;

    //Test the time for the partition using an STL Vector
    beg=time(0);
    ssum=sSumV(array,size,modIt);
    if(ssum<0)ssum+=modIt;
    cout<<"The single for-loop sum and Partitioned Vector = "<<ssum<<endl;
    end=time(0);
    cout<<"Time for single for-loop sum Partition Vector = "<<end-beg<<endl;
```

```
    //Test the time for a nested for-loop, non-partitioned implementation
    beg=time(0);
    cout<<"The double for-loop sum = "<<dSum(array,size,modIt)<<endl;
    end=time(0);
    cout<<"Time for double for-loop sum = "<<end-beg<<endl;

    //Clean up the dynamic char array
    delete []array;

    //Exit state right
    return 0;
}

bool sSum(int n,int modIt){
    //Declaration of intermediate integer values
    long long int sum=0,nmi,ii,summ,nn;

    //Single Sum Formula sum((n-i)*i*i) over i from 0 to n-1
    for(int i=0;i<n;i++){
        nmi=(n-i)%modIt;
        ii=i%modIt;
        sum+=((nmi*(ii*ii)%modIt)%modIt);
    }
    sum%=modIt;

    //Closed form solution n^2*(n^2-1)/12
    summ=n;
    summ*=n;
    summ%=modIt;//n*n
    nn=n;
    nn*=n;
    nn-=1;
    nn%=modIt;//n*n-1
    nn*=inverse(12,modIt);//(n*n-1)*12^-1
    nn%=modIt;
    summ*=nn;//n*n*(n*n-1)/12
    summ%=modIt;
    cout<<"Single sum vs. Closed form -> "<<sum<<" = "<<summ<<endl;

    return sum==summ;
}

int inverse(int divisor,int modIt){
    //Finding the Inverse of 12 mod 10^9+7
    long long int z;
```

```
    for(int i=1;i<modIt;i++){
        z=divisor*i;
        if(z%modIt==1){
            return i;
        }
    }
    return 1;
}

//Single sum with Double Sum partitioned with Vectors
long long int sSumV(unsigned char *a,int n,int modIt){
    //Create 26 sub-arrays for each letter with a list
    long long int sum=0,nmi,ii;
    vector<int> *vchar=new vector<int>[26];
    //Simultaneously calculate the Double Sum with a Single Sum and
    //hash data into a vector
    for(int i=0;i<n;i++){
        nmi= (n-i)%modIt;
        ii=i%modIt;
        sum+=((nmi*(ii*ii)%modIt)%modIt);
        vchar[a[i]-'A'].push_back(i);
    }
    sum%=modIt;
    //For each sub-array take fill an array
    for(int i=0;i<26;i++){
        if(vchar[i].size()<=1)continue;
        //Take the sub-array sum
        sum-=smalldSum(vchar[i],modIt);
        sum%=modIt;
    }
    //Clean up and return
    delete []vchar;
    return sum;
}

//Single sum with Double Sum partitioned with Linked List
long long int sSumLL(unsigned char *a,int n,int modIt){
    //Create 26 sub-arrays for each letter with a list
    long long int sum=0,nmi,ii;
    list<int> *lchar=new list<int>[26];
    //Simultaneously calculate the Double Sum with a Single Sum
    //and fill the list
    for(int i=0;i<n;i++){
        nmi= (n-i)%modIt;
        ii=i%modIt;
        sum+=((nmi*(ii*ii)%modIt)%modIt);
```

```
        lchar[a[i]-'A'].push_front(i);
    }
    sum%=modIt;
    //For each sub-array take fill an array
    for(int i=0;i<26;i++){
        int size=lchar[i].size();
        if(size<=1)continue;
        int *aa=new int[size];
        for(int ii=0;ii<size;ii++){
            aa[ii]=lchar[i].front();
            lchar[i].pop_front();
        }
        //Take the sub-array sum and subtract
        sum-=smalldSum(aa,size,modIt);
        sum%=modIt;
        delete []aa;
    }
    //Clean up and return
    delete []lchar;
    return sum;
}

//Using partitioned array positions for the double sum
long long int smalldSum(vector<int> &a,int modIt){
    //Find the double sum of the smaller sequence
    long long int sum=0,diff;
    for(int i=0;i<a.size()-1;i++){
        for(int j=i+1;j<a.size();j++){
            diff=(a[j]-a[i])%modIt;
            sum+=((diff*diff)%modIt);
            sum%=modIt;
        }
    }
    return sum;
}

//Using partitioned array positions for the double sum
long long int smalldSum(int *a,int n,int modIt){
    //Find the double sum of the smaller sequence
    long long int sum=0,diff;
    for(int i=0;i<n-1;i++){
        for(int j=i+1;j<n;j++){
            diff=(a[j]-a[i])%modIt;
            sum+=((diff*diff)%modIt);
            sum%=modIt;
        }
```

```
    }
    return sum;
}

//Using the original double sum formulation with mod to prevent overflow
long long int dSum(unsigned char *a,int n,int modIt){
    //Find the double sum of the larger array
    long long int sum=0,diff;
    for(int i=0;i<n-1;i++){
        for(int j=i+1;j<n;j++){
            if(a[i]!=a[j]){
                diff=(j-i)%modIt;
                sum+=((diff*diff)%modIt);
                sum%=modIt;
            }
        }
    }
    return sum;
}

//Print char array with perLine output
void prntAry(unsigned char *a,int n,int perLine){
    cout<<endl;
    for(int i=0;i<n;i++){
        cout<<a[i]<<" ";
        if(i%perLine==(perLine-1))cout<<endl;
    }
    cout<<endl;
}

//Simple random fill of char array with capital letters [A-Z] for testing
unsigned char *fillAry(int n){
    unsigned char *a=new unsigned char[n];
    for(int i=0;i<n;i++){
        a[i]=rand()%26+'A';//[A-Z] Random Fill
    }
    return a;
}

/*
//Implementation for n<2^8 = 256
//Using the original double sum formulation without mod operator
unsigned int dSum(unsigned char *a,int n,int modIt){
    //Find the double sum of the larger array
    unsigned int sum=0;
    for(int i=0;i<n-1;i++){
```

```
        for(int j=i+1;j<n;j++){
            if(a[i]!=a[j]){sum+=((j-i)*(j-i));
            }
        }
    }
    return sum;
}

//Implementation for n<2^16 = 64,536
//Using the original double sum formulation without mod operator
unsigned long long int dSum(unsigned char *a,int n,int modIt){
    //Find the double sum of the larger array
    unsigned long long int sum=0;
    for(int i=0;i<n-1;i++){
        for(int j=i+1;j<n;j++){
            if(a[i]!=a[j]){sum+=((j-i)*(j-i));
            }
        }
    }
    return sum;
}
*/
```