Solution: Sherlock and Anagrams (HackerRank)

Phil Mayer

June 16, 2022

1 Problem

HackerRank provides an Interview Preparation Kit containing a number of problems spanning programming topics like data structures, sorting, and searching. After taking some time away from HackerRank, I decided to pick up a question on dictionaries and hash maps. The "Sherlock and Anagrams" question begins by defining anagram within the context of the exercise.

Definition 1.1. *S_1 is an **anagram** of S_2 if the letters of S_1 can be rearranged to form S_2.*

The challenge is to implement a function which accepts an input string and returns the number of a pairs of substrings which are anagrams of each other. For this challenge, a substring does not include the original string. I chose the C# programming language because I'm a little rusty with it.

2 Approach

After some thinking about the problem, I decided that my strategy would be to:

1. Compute all substrings of the input s, except for s itself.
2. For each substring, sort its characters alphabetically. For example, the substring baa would be reordered to aab.
3. Count the frequency $f(i)$ of each sorted substring i within s. For example, if s contains substrings baa and aba, both would be reordered to aab by alphabetical character sorting. The sorted substring aab would have frequency 2.
4. Take all sorted substrings with frequency $f(i) \geq 2$; these are the substrings for which at least one pair can be assembled. The number of possible pairs of the underlying, unsorted substrings is given by $\binom{f(i)}{2}$. Aggregate this value for all sorted substrings.

As a sanity check, my first concern was to count the number of expected substrings of a given string. Counting the number of substrings required a bit of algebra. As it turns out, a string of length n has n substrings of length 1, $n-1$ substrings of length 2, and so on: this is a summation of i for $1 \leq i \leq n$. The value for this series is given by the following formula (see "Proofs" section below):

$$\sum_{i=1}^{n} i = \frac{(n)(n+1)}{2}$$

Since the challenge does not consider s to be a substring of itself, the number of substrings is actually one less than this value.

My justification for the fourth point is given by combinatorics. Since I need to count the number of possible ways to select $k = 2$ substrings (disregarding order) from $n = f(i)$ possible substrings of s, I needed to essentially calculate n choose k, given by $\binom{n}{k}$. More on this later.

3 Solution

My solution begins by computing all substrings of s except for s itself. In the loops below, i represents the starting position and j represents the substring length. I guard against possibly adding s to the list of substrings by adding a condition in the inner loop.

```
var numSubstrings = (s.Length * (s.Length + 1)) / 2 - 1;
var substrings = new List<string>(numSubstrings);
for (var i = 0; i < s.Length; i++) {
    for (var j = 1; i + j <= s.Length && !(i == 0 && j == s.Length); j++) {
        substrings.Add(s.Substring(i, j));
    }
}
```

Since the challenge is intended to test the user's ability to use dictionaries and hash maps, I then used a C# SortedDictionary to count the frequency of each substring within s. I used LINQ here to sort the characters in each substring.

```
var repeatFrequency = new SortedDictionary<string, int>();
foreach (var substring in substrings) {
    var sortedCharacters = new string(substring.OrderBy(c => c).ToArray());
    if (repeatFrequency.ContainsKey(sortedCharacters)) {
        repeatFrequency[sortedCharacters]++;
    } else {
        repeatFrequency[sortedCharacters] = 1;
    }
}
```

Finally, to count the number of anagram pairs within s, I wrote the following. Note that possible refactors are discussed in the next section.

```
var numAnagramPairs = 0;
foreach(var frequency in repeatFrequency.Values) {
    if (frequency > 1) {
        numAnagramPairs += getNumPairs(frequency);
    }
}
```

The function to count the number of pairs (i.e. n choose 2) is defined below.

```
private static int getNumPairs(int n)
{
    if (n < 2) {
        return 0;
    }

    return n * (n - 1) / 2;
}
```

For my first submission, I implemented a *choose* function which calculated $\binom{n}{k}$ by the well-known formula $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. After encountering runtime errors for a couple of the HackerRank test cases, I figured that a brute-force approach (implementing some $factorial$ function) would not be suitable to pass all test cases. Fortunately, it turns out that for $n > 1$:

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \\ \frac{n\binom{n-1}{k-1}}{k} & k > 0 \end{cases}$$

Now since we start with $k = 2$, we arrive at the simplified form used in the code above, since $\binom{n-1}{1} = n - 1$.

$$\binom{n}{2} = \frac{n\binom{n-1}{1}}{2} = \frac{n(n-1)}{2}$$

4 Possible Improvements

I chose to implement my solution in C# because I haven't worked with the language regularly in several years. After my first iteration, I realized I could use LINQ to write my calculation for $numAnagramPairs$ more expressively. In fact, the calculation for $numAnagramPairs$ can be simplified to the expression:

```
repeatFrequency.Values
    .Where(n => n > 1)
    .Aggregate(0, (acc, n) => acc + (n * (n - 1) / 2))
```

Coming from mostly writing JavaScript for the past four years, this more functional solution captures my original intent in a less imperative style. I'm sure there are other refactors I could make to modernize my C# code. Perhaps more importantly, I'm also curious to learn about other clever mathematics/combinatorics to apply here for a more efficient solution.

5 Proofs

Since I'm also rusty on my math, let's do a few quick proofs.

Theorem 5.1. *Let $n \in \mathbb{N}$ and $n > 0$. Then* $\sum_{i=1}^{n} i = \frac{(n)(n+1)}{2}$.

Proof. For the base-case $n = 1$:

$$\sum_{i=1}^{1} i = \frac{(1)(1+1)}{2} = \frac{2}{2} = 1$$

Now assume the theorem is true for $n = k$. If we sum up to $k + 1$, we see that:

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) \\
&= \frac{k(k+1)}{2} + (k+1) \\
&= \frac{k(k+1)}{2} + \frac{2(k+1)}{2} \\
\sum_{i=1}^{k+1} i &= \frac{(k+1)(k+2)}{2}
\end{aligned}$$

$\square$

Theorem 5.2. *Let $n, k \in \mathbb{N}$ where $1 < k \leq n$. Then* $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$.

Proof. We already know that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. By the definition of factorial:

$$\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\
&= \frac{n(n-1)!}{k(k-1)!(n-k)!} \\
&= \frac{n}{k} \cdot \frac{(n-1)!}{(k-1)!(n-1-k+1)!} \\
&= \frac{n}{k} \cdot \frac{(n-1)!}{(k-1)!([n-1]-[k-1])!} \\
\binom{n}{k} &= \frac{n}{k} \cdot \binom{n-1}{k-1}
\end{aligned}$$

$\square$