6. Common Substring Counting

Let, m and n be the lengths of two input strings s1 and s2.

Task 1: Find unique substrings of s1 and s2.
For s1, it contains 'm' characters.
There are (m) substrings of different lengths which start from s1[0]
There are (m-1) substrings of different lengths which start from s1[1]
In general, there are (m-i) substrings of different lengths which start from s1[i]
So, for s1[m-1] there is only one possible substring starting from it i.e. the last character s1[m-1] itself.

Thus, total number of substrings $= m + (m-1) + (m-2) + \cdots + 1$
$= \frac{m(m+1)}{2}$

e.g. abcd (m=4)

substrings starting with a: {a, ab, abc, abcd}	count = 4
substrings starting with b: {b, bc, bcd}	count = 3
substrings starting with c: {c, d}	count = 2
substrings starting with d: {d}	count = 1

Total = 4(4+1)/2 = 10

But, this also contains some repeated substrings

e.g. aba (m=3)

substrings starting with a: {**a**, ab, aba}	count = 3
substrings starting with b: {b, ba}	count = 2
substrings starting with c: {**a**}	count = 1

Total = 3(3+1)/2 = 6
(Here, 'a' is repeated. So, we need to eliminate that)

For this purpose, a set is defined for each of s1 and s2, where a substring will be added only if its unique. So, here, for the first time; 'a' will be inserted into the set. When for the second time 'a' comes, it will not be added into the set.

```
for (int i=1;i<m+1;i++){
        for (int j=0;j<m-i+1;j++){
                if (! sub1.count (s1.substr(j,i))){
                        count1++;
                        sub1.insert(s1.substr(j,i));
                }
        }
}
```

The if statement will:
execute (m) times if i = 1
execute (m-1) times if i = 2
execute (m-i+1) times for i (in general)
and substr( ) generally takes linear time to run.
Total # of executions of 'if' statement = $(m + (m-1) + (m-2)+..+1) * m$

$$= \frac{m(m+1)}{2} * m$$

$$= \frac{(m^3+m^2)}{2}$$

Task 2: Find common substrings of s1 and s2

Each s1 and s2 has its own set of unique substrings. We just have to count the number of substrings which are common to both s1 and s2.

```
int common_count = 0;
set<string>::iterator i;
for(i = sub1.begin( );i!=sub1.end( );i++){
        if(sub2.count(*i)){
                common_count++;
        }
}
```

The for loop will run for 'number of unique substrings' time. So, for worst case, there is no string which is repeated. That is, every generated substring is unique.
count( ) function of set takes logarithmic time to find out the number of occurrences of an element in a set.
Thus, this above snippet will run for mlog(m) times.

Worst Case Running Time

To calculate the time complexity, let c1 and c2 be the cost of execution for "if statement", "set::count( )" respectively.

$$T(m) = c1 * \frac{(m^3 + m^2)}{2} + c2 * mlog(m)$$

This equation is in form of $am^3 + bm^2 + c$ with the m^3 term, which dominates all the other terms. Therefore, eliminating constants and other small terms, the best case running time of selection sort in asymptotic notation is: **O (m^3)**

(m or n, whichever is greater, will dominate the other.)