String Processing: Effective method to check string contains in Java

There are many applications (e.g. information retrieval, Natural Language processing) that require searching a large string for words, terms, or statements. The link below describe several methods to do that using C#.

In this assignment, you are expected to use a large input text file (of more than 3000 words). You will evaluate 3 different method, based on performance. All three methods should have the same signature

Public int NumberOfOccurrences (string inputfile, string word)

The output will be the number of time the word occur in the input file. Evaluate the three methods you selected based on performance. Test your code with 5 different test cases.

http://cc.davelozinski.com/c-sharp/fastest-way-to-check-if-a-string-occurs-within-a-string

C# .NET: FASTEST WAY TO CHECK IF A STRING OCCURS WITHIN A STRING

AUGUST 17, 2013 DAVID LOZINSKI 12 COMMENTS

- 2
-
-
-
-
-
-
-

Using C# .Net: Fastest Way to check if a string occurs within a string.

This will examine many techniques to determine in C# .Net: Fastest Way to check if a string occurs within a string.

The Background:

How many of us C# programmers have had to check if a string is contained within another string? A simple match. We don't care how many times it may exist, we only want to know if it does.

There are numerous native C# methods for doing this: String.Contains(), String.IndexOf(), through Regex regular expressions, and similar options for those programmers obsessed with LINQ.

So that's when this curious consultant started wondering… what is the fastest way to test and see if a string is contained within another string?

The Set Up:

I wrote a C# Console application to test 16 different search techniques: 8 single threaded loop; 8 multi-threaded parallel.for loop.

The code is written in Visual Studio 2012 targeting .Net Framework version 4.5 x64. The source code is available at the end of this blog so you can benchmark it on your own system if you wish.

In a nutshell, the code does the following:

1) Creates an array of random strings that will serve as the strings to be searched using the System.Web.Security.Membership.GeneratePassword method. We'll call these "A".

2) Creates an array of random strings that will serve as the strings being searched for using the System.Web.Security.Membership.GeneratePassword method. We'll call these "B".

3) Executes the techniques to see if any of the search strings occur in the strings to be searched. The techniques are as follows:

#	Technique	Code Snippet

T1	A common counting method	`for (int x = 0; x < ss.Length; x++)`
		`  for (int y = 0; y < sf.Length; y++`
		`  c[y] += ((ss[x].Length - ss[x].Repla`
		`String.Empty).Length) / sf[y].Length > 0 ? 1 : 0);`

T2	splitting each string in A by string B and counting the results.	`for (int x = 0; x < ss.Length; x++)`
		`  for (int y = 0; y < sf.Length; y++)`
		`  c[y] += (ss[x].Split(new string[] {`
		`StringSplitOptions.None).Count() - 1 > 0 ? 1 : 0);`

T3	C#'s String.Contains() method.	`for (int x = 0; x < ss.Length; x++)`
		`  for (int y = 0; y < sf.Length; y++)`
		`  c[y] += (ss[x].Contains(sf[y]) == tr`

T4	C#'s String.IndexOf() method	`for (int x = 0; x < ss.Length; x++)`
		`  for (int y = 0; y < sf.Length; y++)`
		`  c[y] += (ss[x].IndexOf(sf[y]) >= 0 ?`

T5	Using Contains() on a LINQ query	`for (int y = 0; y < sf.Length; y++)` `  c[y] += ss.Where(o => o.Contains(sf[y])).Co`
T6	Using IndexOf() on a LINQ query	`for (int y = 0; y < sf.Length; y++)` `  c[y] += ss.Where(o => o.IndexOf(sf[y]) > -1`

T7	C#'s Regex.IsMatch() method	`for (int x = 0; x < ss.Length; x++)`
		`  for (int y = 0; y < sf.Length; y++)`
		`  c[y] += Regex.IsMatch(ss[x], Regex.E`
		`0;`

T8	AsParallel() with Regex	`for (int x = 0; x < ss.Length; x++)`
		`  total += sf.AsParallel().Sum(s => Regex.IsM`
		`Regex.Escape(s)) ? 1 : 0);`

T9	T1 implemented with a Parallel.For construct	`Parallel.For( .....`
		`  (x, loopState, subtotal) =>`
		`  {`
		`  for (int y = 0; y < sf.Length; y++)`
		`  subtotal += ((ss[x].Length -`
		`String.Empty).Length) / sf[y].Length > 0 ? 1 : 0);`
		`  return subtotal;`
		`  },`
		`  .....`
		`);`

T10	T2 implemented with a Parallel.For construct	`Parallel.For( .....`
		`  (x, loopState, subtotal) =>`
		`  {`
		`  for (int y = 0; y < sf.Length; y++)`

		subtotal += (ss[x].Split(ne StringSplitOptions.None).Count() - 1 > 0 ? 1 : 0); return subtotal; }, ...);

T11	T3 implemented with a Parallel.For construct	Parallel.For(.....
		
		(x, loopState, subtotal) =>
		
		{
		
		for (int y = 0; y < sf.Length; y++)
		
		subtotal += (ss[x].Contains
		
		return subtotal;
		
		},
		
		.....
		
		);

T12	T4 implemented with a Parallel.For construct	Parallel.For(.....
		
		(x, loopState, subtotal) =>
		
		{
		
		for (int y = 0; y < sf.Length; y++)
		
		subtotal += (ss[x].IndexOf(
		
		return subtotal;

		}, ...);
T13	T5 implemented with a Parallel.For construct	Parallel.For(..... (x, loopState, subtotal) => { subtotal += ss.Where(o => o.Contains return subtotal; }, ...);
T14	T6 implemented with a Parallel.For construct	Parallel.For(..... (x, loopState, subtotal) => { subtotal += ss.Where(o => o.IndexOf(return subtotal; }, ...);
T15	T7 implemented with a Parallel.For construct	Parallel.For(.....

		(x, loopState, subtotal) =>
		{
		for (int y = 0; y < sf.Length; y++)
		subtotal += Regex.IsMatch(ss
		== true ? 1 : 0;
		return subtotal;
		},
		.....
		);

T16	T8 implemented with a Parallel.For construct	Parallel.For(..... (x, loopState, subtotal) => { subtotal += sf.AsParallel().Sum(s => Regex.Escape(s)) ? 1 : 0); return subtotal; }, ...);

4) The sum from adding up the number of times a match is found is displayed along with the execution time

5) The arrays are cleared out and deleted being the responsible programmer I am.

The code assumes all tests will happen with no exception testing because I'm just wanting to test the raw speed of finding a string in a string. There may be other techniques, but I think these are the most common. *However, if you have a great alternative solution, please post!*
The exe file was installed and run on an Alienware M17X R3 running Windows 7 64-bit with 16 GB memory on an i7-2820QM processor.

The test was run for each technique over the following number of comparisons:

- Searching for 1 string in 5,000, 25,000, 100,000, and 1,000,000 million strings.
- Searching for 100 strings in 5,000, 25,000, 100,000, and 1,000,000 million strings.
- Searching for 1000 strings in 5,000, 25,000, 100,000, and 1,000,000 million strings.

In regards to the last two bullet points, I do realize though that in most cases, programmers are only going to test if 1 or 2 strings occur in another string. But:

1) We're having fun aren't we?

2) Why not?

Off To The Races!

Before starting, my hypothesis was that I expected the custom counting method to be the winner in both the single and multi-threaded groups since it generally does perform the best whether counting substring occurrences in strings or characters in strings. It's amazing how fast and versatile this method is.

All times are indicated in hours:minutes:seconds.milliseconds format. Yes, some took over an hour! The lower the number, the faster the technique performed.

Green cells indicate the winner(s) for that run.
Yellow cells indicate the second runner(s) up.

	Searching for 1 string in each of		
	5,000	25,000	100,000
T1: Common Counting	0.00	0.00	0.010000
T2: Count Split String	0.00	0.01	0.030000
T3: String.Contains()	0.00	0.00	0.010000
T4: String.IndexOf()	0.00	0.00	0.020000
T5: Linq.Contains()	0.00	0.00	0.010000
T6: Linq.IndexOf()	0.00	0.0100001	0.010000
T7: Regex.IsMatch()	0.01	0.01	0.050000
T8: AsParallel() with Regex	0.1090063	0.5928010	2.427117
T9: T1 using Parallel.For	0.04	0.00	0.00
T10: T2 using Parallel.For	0.00	0.01	0.010000
T11: T3 using Parallel.For	0.00	0.00	0.010000
T12: T4 using Parallel.For	0.00	0.00	0.00
T13: T5 using Parallel.For	0.00	0.00	0.010000
T14: T6 using Parallel.For	0.00	0.00	0.010000
T15: T7 using Parallel.For	0.00	0.02	0.060000
T16: T8 using Parallel.For	9.5066852	0.9984017	4.120963

	Searching for 100 strings in each of		
	5,000	25,000	100,000
T1: Common Counting	0.0310018	00.1500003	00.611035
T2: Count Split String	0.0890051	00.4200005	01.709051
T3: String.Contains()	0.0400023	00.2000003	00.790001
T4: String.IndexOf()	0.0580033	00.2800004	01.100001
T5: Linq.Contains()	0.0400023	00.2000003	00.800001
T6: Linq.IndexOf()	0.0560032	00.2700004	01.120001
T7: Regex.IsMatch()	1.9920827	10.0042932	39.981982

T8: AsParallel() with Regex	2.6716222	13.2043531	47.694391
T9: T1 using Parallel.For	0.0100000	0.0300000	00.132007
T10: T2 using Parallel.For	0.0400001	0.1700014	00.967055
T11: T3 using Parallel.For	0.0100000	0.0610007	00.209011
T12: T4 using Parallel.For	0.0100000	0.0720041	00.289016
T13: T5 using Parallel.For	0.0100000	0.0580033	00.223012
T14: T6 using Parallel.For	0.0200000	0.0830025	00.326018
T15: T7 using Parallel.For	1.5900023	7.9991942	56.931308
T16: T8 using Parallel.For	1:28.8255242	7:37.3664312	35.970092

	Searching for 1,000 strings in each of		
	5,000	**25,000**	**100,000**
T1: Common Counting	00.3400005	0:01.4500020	0:05.84602
T2: Count Split String	00.9260396	0:04.1800059	0:16.95956
T3: String.Contains()	00.4140236	0:02.0341091	0:08.01008
T4: String.IndexOf()	00.5780331	0:02.8061605	0:11.07824
T5: Linq.Contains()	00.4080233	0:01.9950192	0:08.18427
T6: Linq.IndexOf()	00.5830334	0:02.7400039	0:11.29129
T7: Regex.IsMatch()	20.3724376	1:40.5314815	6:28.70749
T8: AsParallel() with Regex	16.1745050	1:21.2604800	5:29.54540
T9: T1 using Parallel.For	00.0770011	0:00.3400194	0:01.34307
T10: T2 using Parallel.For	00.3550013	0:01.7851021	0:06.79019
T11: T3 using Parallel.For	00.1160039	0:00.5270302	0:02.12611
T12: T4 using Parallel.For	00.1510086	0:00.7440425	0:02.98517
T13: T5 using Parallel.For	00.1140065	0:00.5470291	0:02.25212
T14: T6 using Parallel.For	00.1560067	0:00.7890429	0:03.13917
T15: T7 using Parallel.For	18.2395857	2:57.7773771	6:24.33848

T16: T8 using Parallel.For	14.9139029	1:15.3743422	4:58.98882

The Results:

It's quite obvious technique T1 implementing a common counting method won by running faster *every single time* than any of the native C# methods, Linq queries, or Regex matching. As the old saying goes, "green across the board".

To my surprise, String.Contains() consistently ran second fastest, and stood out from the rest of the pack of contenders as we increased the number of strings we were searching for.

I'm also surprised the String.IndexOf() ran so slow in comparison. Definitely a method to avoid using except when necessary.

The Linq.Contains() technique isn't a bad alternative to using String.Contains(). Personally, I'm not crazy about Linq and avoid it like the plague, but I have to give it a fair assessment in these tests.

Regex is a powerful tool, but as one can see from the results, if you know you're going to perform checks 100 times or more for a simple string rather than a complicated pattern, Regex should not be used.

In Summary:

On my system, unless someone spots a flaw in my test code, Technique 1 with the common counting method was the winner. Any application that needs micro optimization or where speed is of the essence, nothing else should be considered. You just need to decide when to use the single-threaded or multi-threaded technique depending on your application's requirements.

Here's the basis of that code. If it returns a value > 0, the substring was found:

```
return (strToSearch.Length - strToSearch.Replace(strKeyToLookFor,
String.Empty).Length) / strKeyToLookFor.Length;
```

Otherwise, if you don’t mind code running ever so slightly slower and want easy code readability, C#’s native String.Contains() method is the way to go. It’s a simple, elegant, one-liner, and performs relatively well in both single and multi-threaded renditions.

Unless you need the actual starting index in the string where the match is found, then when it comes to speed, everything else shouldn’t be considered.

The Code:

```
using System;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Threading;



namespace TestApplication

{

    class Program

    {

        static void Main(string[] args)

        {

            DateTime end;

            DateTime start = DateTime.Now;



            Console.WriteLine("### Overall Start Time: " + start.ToLongTimeString());
```

```
Console.WriteLine();


TestFastestWayToSeeIfAStringOccursInAString(5000, 1);
TestFastestWayToSeeIfAStringOccursInAString(25000, 1);
TestFastestWayToSeeIfAStringOccursInAString(100000, 1);
TestFastestWayToSeeIfAStringOccursInAString(1000000, 1);
TestFastestWayToSeeIfAStringOccursInAString(5000, 100);
TestFastestWayToSeeIfAStringOccursInAString(25000, 100);
TestFastestWayToSeeIfAStringOccursInAString(100000, 100);
TestFastestWayToSeeIfAStringOccursInAString(1000000, 100);
TestFastestWayToSeeIfAStringOccursInAString(5000, 1000);
TestFastestWayToSeeIfAStringOccursInAString(25000, 1000);
TestFastestWayToSeeIfAStringOccursInAString(100000, 1000);
TestFastestWayToSeeIfAStringOccursInAString(1000000, 1000);


end = DateTime.Now;
Console.WriteLine();
Console.WriteLine("### Overall End Time: " + end.ToLongTimeString());
Console.WriteLine("### Overall Run Time: " + (end - start));


Console.WriteLine();
Console.WriteLine("Hit Enter to Exit");
Console.ReadLine();
```

```
        }



        //#################################################################



        //what is the fastest way to see if a string occurs in a string?



        static void TestFastestWayToSeeIfAStringOccursInAString(int NumberOfStringsToGenerate,
int NumberOfSearchCharsToGenerate)

        {

            Console.WriteLine("######## " +
System.Reflection.MethodBase.GetCurrentMethod().Name);

            Console.WriteLine("Number of Random Strings that will be generated: " +
NumberOfStringsToGenerate.ToString("#,##0"));

            Console.WriteLine("Number of Search Strings that will be generated: " +
NumberOfSearchCharsToGenerate.ToString("#,##0"));

            Console.WriteLine();



            object lockObject = new object();

            int total = 0;

            DateTime end = DateTime.Now;

            DateTime start = DateTime.Now;

            //the strings to search

            string[] ss = new string[NumberOfStringsToGenerate];

            //the chars/strings to look for. We use both because we're testing some string
methods too.

            string[] sf = new string[NumberOfSearchCharsToGenerate];

            //the count of each substring finding
```

```
            int[] c = new int[sf.Length];



            //Generate the string arrays

            int z = 10;



            //yes I realize that most real world applications people are going to search strings

            //that are "human readable" and not random like I generate below. But this is a test

            //and I'm not about to type of millions of different strings for testing. 

            //strings to be searched. Completely random. Using generate password method to come
up with all sorts of mixtures.

            Console.WriteLine("Generating strings to search.");

            for (int x = 0; x < ss.Length; x++)

            {

                ss[x] = System.Web.Security.Membership.GeneratePassword(z, x % 5);

                z += 1;

                if (z > 25)

                    z = 10;

            }



            //strings to search for

            Console.WriteLine("Generating strings to search for.");

            z = 2;

            for (int x = 0; x < sf.Length; x++)

            {

                sf[x] = System.Web.Security.Membership.GeneratePassword(z, x % 2);
```

```
                z += 1;

                if (z > 8)

                    z = 2;

            }

            Console.WriteLine("############################################################");

            Console.WriteLine();



            //return (strToSearch.Length - strToSearch.Replace(strKeyToLookFor,
String.Empty).Length) / strKeyToLookFor.Length;

            Console.WriteLine("Starting method: "custom string method" ");

            start = DateTime.Now;

            for (int x = 0; x < ss.Length; x++)

            {

                for (int y = 0; y < sf.Length; y++)

                {

                    c[y] += ((ss[x].Length - ss[x].Replace(sf[y], String.Empty).Length) /
sf[y].Length > 0 ? 1 : 0);

                }

            }

            end = DateTime.Now;

            Console.WriteLine("Finished at: " + end.ToLongTimeString());

            Console.WriteLine("Time: " + (end - start));

            total = 0;

            for (int x = 0; x < c.Length; x++)

            {
```

```
            total += c[x];
        }
        Console.WriteLine("Total finds: " + total + Environment.NewLine);
        Console.WriteLine();
        Console.WriteLine("############################################################");
        Console.WriteLine();

        Array.Clear(c, 0, c.Length);
        Console.WriteLine("Starting method: "count split string on string" ");
        start = DateTime.Now;
        for (int x = 0; x < ss.Length; x++)
        {
            for (int y = 0; y < sf.Length; y++)
            {
                c[y] += (ss[x].Split(new string[] { sf[y] }, StringSplitOptions.None).Count()
- 1 > 0 ? 1 : 0);
            }
        }
        end = DateTime.Now;
        Console.WriteLine("Finished at: " + end.ToLongTimeString());
        Console.WriteLine("Time: " + (end - start));
        total = 0;
        for (int x = 0; x < c.Length; x++)
        {
            total += c[x];
```

```
}
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();


Array.Clear(c, 0, c.Length);
Console.WriteLine("Starting method: "String.Contains" ");
start = DateTime.Now;
for (int x = 0; x < ss.Length; x++)
{
    for (int y = 0; y < sf.Length; y++)
    {
        c[y] += (ss[x].Contains(sf[y]) == true ? 1 : 0);
    }
}
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
total = 0;
for (int x = 0; x < c.Length; x++)
{
    total += c[x];
}
```

```
Console.WriteLine("Total finds: " + total + Environment.NewLine);

Console.WriteLine();

Console.WriteLine("############################################################");

Console.WriteLine();



Array.Clear(c, 0, c.Length);

Console.WriteLine("Starting method: "String.indexOf" ");

start = DateTime.Now;

for (int x = 0; x < ss.Length; x++)

{

    for (int y = 0; y < sf.Length; y++)

    {

        c[y] += (ss[x].IndexOf(sf[y]) >= 0 ? 1 : 0);

    }

}

end = DateTime.Now;

Console.WriteLine("Finished at: " + end.ToLongTimeString());

Console.WriteLine("Time: " + (end - start));

total = 0;

for (int x = 0; x < c.Length; x++)

{

    total += c[x];

}

Console.WriteLine("Total finds: " + total + Environment.NewLine);
```

```
Console.WriteLine();

Console.WriteLine("############################################################");

Console.WriteLine();



Array.Clear(c, 0, c.Length);

Console.WriteLine("Starting method: "linq contains usage" ");

start = DateTime.Now;

for (int y = 0; y < sf.Length; y++)

{

    c[y] += ss.Where(o => o.Contains(sf[y])).Count();

}

end = DateTime.Now;

Console.WriteLine("Finished at: " + end.ToLongTimeString());

Console.WriteLine("Time: " + (end - start));

total = 0;

for (int x = 0; x < c.Length; x++)

{

    total += c[x];

}

Console.WriteLine("Total finds: " + total + Environment.NewLine);

Console.WriteLine();

Console.WriteLine("############################################################");

Console.WriteLine();
```

```
Array.Clear(c, 0, c.Length);

Console.WriteLine("Starting method: "linq with IndexOf usage" ");

start = DateTime.Now;

for (int y = 0; y < sf.Length; y++)

{

    c[y] += ss.Where(o => o.IndexOf(sf[y]) > -1).Count();

}

end = DateTime.Now;

Console.WriteLine("Finished at: " + end.ToLongTimeString());

Console.WriteLine("Time: " + (end - start));

total = 0;

for (int x = 0; x < c.Length; x++)

{

    total += c[x];

}

Console.WriteLine("Total finds: " + total + Environment.NewLine);

Console.WriteLine();

Console.WriteLine("############################################################");

Console.WriteLine();



Array.Clear(c, 0, c.Length);

Console.WriteLine("Starting method: "Regex IsMatch uncompiled" ");

start = DateTime.Now;

for (int x = 0; x < ss.Length; x++)
```

```
{
    for (int y = 0; y < sf.Length; y++)
    {
        c[y] += Regex.IsMatch(ss[x], Regex.Escape(sf[y])) == true ? 1 : 0;
    }
}
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
total = 0;
for (int x = 0; x < c.Length; x++)
{
    total += c[x];
}
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();

total = 0;
Console.WriteLine("Starting method: "Single AsParallel() Regex" ");
start = DateTime.Now;
for (int x = 0; x < ss.Length; x++)
{
```

```
                total += sf.AsParallel().Sum(s => Regex.IsMatch(ss[x], Regex.Escape(s)) ? 1 :
0);

            }

            end = DateTime.Now;

            Console.WriteLine("Finished at: " + end.ToLongTimeString());

            Console.WriteLine("Time: " + (end - start));

            Console.WriteLine("Total finds: " + total + Environment.NewLine);

            Console.WriteLine();

            Console.WriteLine("############################################################");

            Console.WriteLine();



            total = 0;

            Console.WriteLine("Starting method: "Parallel For Custom Counting" ");

            start = DateTime.Now;

            Parallel.For(0, ss.Length,

                () => 0,

                (x, loopState, subtotal) =>

                {

                    for (int y = 0; y < sf.Length; y++)

                    {

                        subtotal += ((ss[x].Length - ss[x].Replace(sf[y], String.Empty).Length) /
sf[y].Length > 0 ? 1 : 0);

                    }

                    return subtotal;

                },
```

```
    (s) =>
    {
        lock (lockObject)
        {
            total += s;
        }
    }
);
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();

total = 0;
Console.WriteLine("Starting method: "Parallel For Split String" ");
start = DateTime.Now;
Parallel.For(0, ss.Length,
    () => 0,
    (x, loopState, subtotal) =>
    {
        for (int y = 0; y < sf.Length; y++)
```

```
                    {

                        subtotal += (ss[x].Split(new string[] { sf[y] },
StringSplitOptions.None).Count() - 1 > 0 ? 1 : 0);

                    }

                    return subtotal;

                },

                (s) =>

                {

                    lock (lockObject)

                    {

                        total += s;

                    }

                }

            );

            end = DateTime.Now;

            Console.WriteLine("Finished at: " + end.ToLongTimeString());

            Console.WriteLine("Time: " + (end - start));

            Console.WriteLine("Total finds: " + total + Environment.NewLine);

            Console.WriteLine();

            Console.WriteLine("############################################################");

            Console.WriteLine();



            total = 0;

            Console.WriteLine("Starting method: "Parallel For String.Contains()" ");

            start = DateTime.Now;
```

```
Parallel.For(0, ss.Length,
    () => 0,
    (x, loopState, subtotal) =>
    {
        for (int y = 0; y < sf.Length; y++)
        {
            subtotal += (ss[x].Contains(sf[y]) == true ? 1 : 0);
        }
        return subtotal;
    },
    (s) =>
    {
        lock (lockObject)
        {
            total += s;
        }
    }
);
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
```

```
Console.WriteLine();



total = 0;

Console.WriteLine("Starting method: "Parallel For String.IndexOf()" ");

start = DateTime.Now;

Parallel.For(0, ss.Length,

    () => 0,

    (x, loopState, subtotal) =>

    {

        for (int y = 0; y < sf.Length; y++)

        {

            subtotal += (ss[x].IndexOf(sf[y]) >= 0 ? 1 : 0);

        }

        return subtotal;

    },

    (s) =>

    {

        lock (lockObject)

        {

            total += s;

        }

    }

);

end = DateTime.Now;
```

```
Console.WriteLine("Finished at: " + end.ToLongTimeString());

Console.WriteLine("Time: " + (end - start));

Console.WriteLine("Total finds: " + total + Environment.NewLine);

Console.WriteLine();

Console.WriteLine("############################################################");

Console.WriteLine();



total = 0;

Console.WriteLine("Starting method: "Parallel For Linq.Contains()" ");

start = DateTime.Now;

Parallel.For(0, sf.Length,

    () => 0,

    (x, loopState, subtotal) =>

    {

        subtotal += ss.Where(o => o.Contains(sf[x])).Count();

        return subtotal;

    },

    (s) =>

    {

        lock (lockObject)

        {

            total += s;

        }

    }
```

```
);
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();

total = 0;
Console.WriteLine("Starting method: "Parallel For Linq.IndexOf()" ");
start = DateTime.Now;
Parallel.For(0, sf.Length,
    () => 0,
    (x, loopState, subtotal) =>
    {
        subtotal += ss.Where(o => o.IndexOf(sf[x]) > -1).Count();
        return subtotal;
    },
    (s) =>
    {
        lock (lockObject)
        {
            total += s;
```

```
        }
    }
);
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();

total = 0;
Console.WriteLine("Starting method: "Parallel For Regex.IsMatch()" ");
start = DateTime.Now;
Parallel.For(0, ss.Length,
    () => 0,
    (x, loopState, subtotal) =>
    {
        for (int y = 0; y < sf.Length; y++)
        {
            subtotal += Regex.IsMatch(ss[x], Regex.Escape(sf[y])) == true ? 1 : 0;
        }
        return subtotal;
    },
```

```
            (s) =>
            {
                lock (lockObject)
                {
                    total += s;
                }
            }
        );
        end = DateTime.Now;
        Console.WriteLine("Finished at: " + end.ToLongTimeString());
        Console.WriteLine("Time: " + (end - start));
        Console.WriteLine("Total finds: " + total + Environment.NewLine);
        Console.WriteLine();
        Console.WriteLine("############################################################");
        Console.WriteLine();


        total = 0;
        Console.WriteLine("Starting method: "Parallel For AsParallel() Regex" ");
        start = DateTime.Now;
        Parallel.For(0, ss.Length,
            () => 0,
            (x, loopState, subtotal) =>
                {
                    subtotal += sf.AsParallel().Sum(s => Regex.IsMatch(ss[x],
Regex.Escape(s)) ? 1 : 0);
```

```
            return subtotal;
        },
    (s) =>
        {
            lock (lockObject)
            {
                total += s;
            }
        }
);
end = DateTime.Now;
Console.WriteLine("Finished at: " + end.ToLongTimeString());
Console.WriteLine("Time: " + (end - start));
Console.WriteLine("Total finds: " + total + Environment.NewLine);
Console.WriteLine();
Console.WriteLine("############################################################");
Console.WriteLine();

Array.Clear(ss, 0, ss.Length);
ss = null;
Array.Clear(sf, 0, sf.Length);
sf = null;
Array.Clear(c, 0, c.Length);
c = null;
```

```
            GC.Collect();



        }

    }


}
```