دانشکده مهندسی کامپیوتر

طراحی و تحلیل الگوریتم‌ها

تمرین ۶*

اساتید حل تمرین: زهرا حسینی، سهراب نمازی
تهیه و تنظیم مستند: مریم سادات هاشمی

استاد درس: سید صالح اعتمادی

نیم‌سال دوم ۱۴۰۰-۱۳۹۹

تلگرام	@arrhhaz @Sohlaub
نام شاخه	fb_A6
نام پروژه/پوشه/پول ریکوست	A6
مهلت تحویل	۱۳۹۹/۲/۱۱

*تشکر ویژه از اساتید حل‌تمرین مریم سادات هاشمی، بنفشه کریمیان، مهسا سادات رضوی، امیر خاکپور، سهیل رستگار و علی آل‌یاسین که در نیم‌سال دوم سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند.

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A6 بسازید.

۲. کلاس هر سوال را به پروژه‌ی خود اضافه کنید و در قسمت مربوطه کد خود را بنویسید. هر کلاس شامل دو متد اصلی است:

- متد اول: تابع Solve است که شما باید الگوریتم خود را برای حل سوال در این متد پیاده سازی کنید.
- متد دوم: تابع Process است که مانند تمرین های قبلی در TestCommon پیاده سازی شده است. بنابراین با خیال راحت سوال را حل کنید و نگران تابع Process نباشید! زیرا تمامی پیاده سازی ها برای شما انجام شده است و نیازی نیست که شما کدی برای آن بزنید.

۳. اگر برای حل سوالی نیاز به تابع های کمکی دارید؛ می توانید در کلاس مربوط به همان سوال تابع تان را اضافه کنید.

اکنون که پیاده سازی شما به پایان رسیده است، نوبت به تست برنامه می رسد. مراحل زیر را انجام دهید.

۱. یک UnitTest برای پروژه‌ی خود بسازید.

۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژه‌ی تست خود اضافه کنید.

۳. فایل GradedTests.cs را به پروژه‌ی تستی که ساخته اید اضافه کنید.

توجه:
برای اینکه تست شما از بهینه سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست ها را روی بیلد Release امتحان کنید، درغیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using A6;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A6.Tests
{
    [TestClass()]
    [DeploymentItem("TestData", "A6_TestData")]
    public class GradedTests
    {
        [TestMethod(), Timeout(500)]
        public void SolveTest_Q1ConstructBWT()
        {
            RunTest(new Q1ConstructBWT("TD1"));
        }

        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q2ReconstructStringFromBWT()
        {
            RunTest(new Q2ReconstructStringFromBWT("TD2"));
        }

        [TestMethod(), Timeout(500)]
        public void SolveTest_Q3MatchingAgainCompressedString()
        {
            RunTest(new Q3MatchingAgainCompressedString("TD3"));
        }

        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q4ConstructSuffixArray()
        {
            RunTest(new Q4ConstructSuffixArray("TD4"));
        }
        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A6", p.Process, p.TestDataName, p.Verifier,
                VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                excludedTestCases: p.ExcludedTestCases);
        }
    }
}
```

۱ بدست آوردن تبدیل Burrows Wheeler برای یک رشته [۱]

تبدیل Burrows Wheeler یک متن، نمادهای متن را به گونه‌ای تغییر می‌دهد تا به راحتی قابل فشرده‌سازی شود. همچنین، این تبدیل برگشت‌پذیر نیز هست. یعنی از روی تبدیل BW یک متن می‌توانیم متن اصلی را بدست آوریم. کاربرد این تبدیل علاوه بر فشرده سازی متن، در حل کردن الگوریتم تطبیق چندگانه الگوها و ... است.

بدست آوردن تبدیل Burrows Wheeler برای یک رشته به صورت مقابل تعریف می‌شود: اول تمام چرخش‌های متناوب ممکن از متن را تشکیل دهید. یک چرخش متناوب از جداکردن پسوندی از رشته از انتهای رشته و اضافه کردن آن به ابتدای رشته بدست می آید. سپس این رشته‌ها را به ترتیب الفبایی مرتب کنید تا یک ماتریس M با سایز |TEXT| × |TEXT| تشکیل شود. BWT برای رشته ورودی برابر با ستون آخر ماتریس M است.

در این سوال شما باید الگوریتمی بنویسید که BWT برای یک رشته ورودی که با حروف T G C A ساخته شده است و با نماد $ پایان می‌یابد را برگرداند. سایز رشته ورودی بین ۱ تا ۱۰۰۰ کاراکتر است.

توجه:
برای دیباگ و تست کردن جواب‌های خود می‌توانید از این لینک استفاده کنید.

ورودی نمونه	خروجی نمونه
AA$	AA$

$$M(Text) = \begin{bmatrix} \$ & A & A \\ A & \$ & A \\ A & A & \$ \end{bmatrix}$$

شکل ۱: نمونه اول

[۱] Construct the Burrows–Wheeler Transform of a String

ورودی نمونه	خروجی نمونه
ACACACAC$	CCCC$AAAA

$$M(Text) = \begin{bmatrix} \$ & A & C & A & C & A & C & A & C \\ A & C & \$ & A & C & A & C & A & C \\ A & C & A & C & \$ & A & C & A & C \\ A & C & A & C & A & C & \$ & A & C \\ A & C & A & C & A & C & A & C & \$ \\ C & \$ & A & C & A & C & A & C & A \\ C & A & C & \$ & A & C & A & C & A \\ C & A & C & A & C & \$ & A & C & A \\ C & A & C & A & C & A & C & \$ & A \end{bmatrix}$$

شکل ۲: نمونه دوم

ورودی نمونه	خروجی نمونه
AGACATA$	ATG$CAAA

$$M(Text) = \begin{bmatrix} \$ & A & G & A & C & A & T & A \\ A & \$ & A & G & A & C & A & T \\ A & C & A & T & A & \$ & A & G \\ A & G & A & C & A & T & A & \$ \\ A & T & A & \$ & A & G & A & C \\ C & A & T & A & \$ & A & G & A \\ G & A & C & A & T & A & \$ & A \\ T & A & \$ & A & G & A & C & A \end{bmatrix}$$

شکل ۳: نمونه سوم

```
using System;
using TestCommon;

namespace A6
{
    public class Q1ConstructBWT : Processor
    {
        public Q1ConstructBWT(string testDataName)
        : base(testDataName) { }

        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<String, String>)Solve);

        /// <summary>
        /// Construct the Burrows-Wheeler transform of a string
        /// </summary>
        /// <param name="text"> A string Text ending with a "$" symbol </param>
        /// <returns> BWT(Text) </returns>
        public string Solve(string text)
        {
            throw new NotImplementedException();
        }
    }
}
```

۲ بدست آوردن رشته از روی BWT [۲]

در سوال قبل با چگونه بدست آوردن BWT برای یک رشته آشنا شدید و دیدیم این عمل برای فشرده‌سازی متون انجام می‌شود. برای تکمیل فرآیند فشرده‌سازی لازم است تا بتوانیم برعکس این فرآیند را نیز انجام دهیم. یعنی بتوانیم از روی BWT یک رشته، رشته اصلی را بدست آوریم. در این سوال باید الگوریتمی بنویسید که رشته اصلی را از روی BWT رشته بدست آورد. ورودی الگوریتم یک رشته شامل یک نماد $ است که با حروف G T C A ساخته شده است و سایز رشته ورودی بین ۱ تا ۱۰۰۰۰۰۰ کاراکتر است. خروجی الگوریتم آن رشته اصلی است که تبدیل BW آن در ورودی داده شده است.

ورودی نمونه	خروجی نمونه
AC$A	ACA$

$$M(Text) = \begin{bmatrix} \$ & A & C & A \\ A & \$ & A & C \\ A & C & A & \$ \\ C & A & \$ & A \end{bmatrix}$$

شکل ۴: نمونه اول

ورودی نمونه	خروجی نمونه
AGGGAA$	GAGAGA$

$$M(Text) = \begin{bmatrix} \$ & G & A & G & A & G & A \\ A & \$ & G & A & G & A & G \\ A & G & A & \$ & G & A & G \\ A & G & A & G & A & \$ & G \\ G & A & \$ & G & A & G & A \\ G & A & G & A & \$ & G & A \\ G & A & G & A & G & A & \$ \end{bmatrix}$$

شکل ۵: نمونه دوم

[۲] Reconstruct a String from its Burrows–Wheeler Transform

```
using System;
using TestCommon;

namespace A6
{
    public class Q2ReconstructStringFromBWT : Processor
    {
        public Q2ReconstructStringFromBWT(string testDataName)
        : base(testDataName) { }

        public override string Process(string inStr) =>
        TestTools.Process(inStr, (Func<String, String>)Solve);

        /// <summary>
        /// Reconstruct a string from its Burrows-Wheeler transform
        /// </summary>
        /// <param name="bwt"> A string Transform with a single "$" sign </param>
        /// <returns> The string Text such that BWT(Text) = Transform.
        /// (There exists a unique such string.) </returns>
        public string Solve(string bwt)
        {
            throw new NotImplementedException();
        }
    }
}
```

۳ تطبیق الگوها با استفاده از رشته‌های فشرده شده [۳]

یکی دیگر از قابلیت‌های جالب BWT ، این است که به ما این قابلیت را می‌دهد که بدون decompress کردن رشته بتوانیم مسئله Pattern Matching را حل کنیم. بدین ترتیب حافظه کمتری مصرف می‌شود.

الگوریتم BWMatching تعداد تکرار شدن الگوها را در متن می شمارد. اطلاعاتی که این الگوریتم در اختیار دارد تنها ستون اول ماتریس، ستون آخر (یعنی BWT) و یک mapping از ابتدا به انتها است.

در این سوال شما باید الگوریتمی بنویسید که بتواند الگوها را در یک متن فشرده شده پیدا کند.

در خط اول فایل ورودی یک رشته که فرم BWT متن است وجود دارد. این رشته با نمادهای A T C G ساخته شده است و طول آن بین ۱ تا ۱۰۰۰ حرف است و در آن نماد $ نیز وجود دارد. در خط بعدی یک عدد صحیح بین ۱ تا ۵۰۰۰ وجود دارد که نشان‌دهنده ی تعداد الگوهاست. در n خط بعدی الگوهایی که باید در رشته ورودی یافت شوند وجود دارد.

شما باید در فایل خروجی لیستی از اعداد صحیح که هر یک نشان‌دهنده‌ی تعداد تکرار الگوی متناظرش در متن است را برگردانید.

ورودی نمونه	خروجی نمونه
AGGGAA$ 1 GA	3

در این مثال متن GAGAGA$ بوده که الگوی GA سه بار در آن تکرار شده است.

ورودی نمونه	خروجی نمونه
ATT$AA 2 ATA A	2 3

در این مثال متن ATATA$ بوده که شامل ۲ الگوی ATA و سه الگوی A است.

ورودی نمونه	خروجی نمونه
AT$TCTATG 2 TCT TATG	0 0

در این مثال متن ATCGTTTA بوده که شامل هیچ یک از الگوهای داده شده در ورودی نیست.

[۳] Matching Against a Compressed String

```
using System;
using TestCommon;

namespace A6
{
    public class Q3MatchingAgainCompressedString : Processor
    {
        public Q3MatchingAgainCompressedString(string testDataName)
        : base(testDataName) { }

        public override string Process(string inStr) =>
        TestTools.Process(inStr, (Func<String, long, String[], long[]>)Solve);

        /// <summary>
        /// Implement BetterBWMatching algorithm
        /// </summary>
        /// <param name="text"> A string BWT(Text) </param>
        /// <param name="n"> Number of patterns </param>
        /// <param name="patterns"> Collection of n strings Patterns </param>
        /// <returns> A list of integers, where the i-th integer corresponds
        /// to the number of substring matches of the i-th member of Patterns
        /// in Text. </returns>
        public long[] Solve(string text, long n, String[] patterns)
        {
            throw new NotImplementedException();
        }
    }
}
```

۴ تشکیل Suffix Array برای یک رشته [۴]

در تمرین قبل با Suffix هاTree آشنا شدیم و دیدیم در عمل حافظه زیادی اشغال می‌کنند. آرایه پیشوندی یک راه‌حل جایگزین برای درخت پیشوندی است که از نظر حافظه بهینه عمل می‌کند. به عنوان مثال برای ذخیره درخت پسوندی ژنوم انسان نیاز به ۶۰ گیگ حافظه داریم درحالی که آرایه پسوندی تنها ۱۲ گیگ حافظه اشغال می‌کند. برای ساختن Suffix Array یک رشته به این صورت عمل می‌کنیم که تمامی پسوندهای رشته ورودی را به ترتیب الفبایی (با فرض $ بعنوان اولین نماد الفبا) مرتب می‌کنیم. آرایه پسوندی، لیستی از ایندکس اولین نماد از این پسوندهای مرتب شده در رشته ورودی است. در این سوال باید آرایه پسوندی برای یک رشته ورودی ساخته شده با حروف G T C A که با نماد $ تمام می‌شود را تشکیل دهید.

توجه:

برای دیباگ و تست کردن جواب‌های خود می‌توانید از این لینک استفاده کنید.

ورودی نمونه	خروجی نمونه
GAC$	3 1 2 0

Sorted suffixes:

```
3   $
1   AC$
2   C$
0   GAC$
```

شکل ۶: نمونه اول

[۴] Construct the Suffix Array of a String

ورودی نمونه	خروجی نمونه
GAGAGAGA$	8 7 5 3 1 6 4 2 0

Sorted suffixes:

```
8   $
7   A$
5   AGA$
3   AGAGA$
1   AGAGAGA$
6   GA$
4   GAGA$
2   GAGAGA$
0   GAGAGAGA$
```

شکل ۷: نمونه دوم

ورودی نمونه	خروجی نمونه
AACGATAGCGGTAGA$	15 14 0 1 12 6 4 2 8 13 3 7 9 10 11 5

Sorted suffixes:

```
15  $
14  A$
0   AACGATAGCGGTAGA$
1   ACGATAGCGGTAGA$
12  AGA$
6   AGCGGTAGA$
4   ATAGCGGTAGA$
2   CGATAGCGGTAGA$
8   CGGTAGA$
13  GA$
3   GATAGCGGTAGA$
7   GCGGTAGA$
9   GGTAGA$
10  GTAGA$
11  TAGA$
5   TAGCGGTAGA$
```

شکل ۸: نمونه سوم

```
using System;
using TestCommon;

namespace A6
{
    public class Q4ConstructSuffixArray : Processor
    {
        public Q4ConstructSuffixArray(string testDataName)
        : base(testDataName) { }

        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<String, long[]>)Solve);

        /// <summary>
        /// Construct the suffix array of a string
        /// </summary>
        /// <param name="text"> A string Text ending with a "$" symbol </param>
        /// <returns> SuffixArray(Text), that is, the list of starting positions
        /// (0-based) of sorted suffixes separated by spaces </returns>
        public long[] Solve(string text)
        {
            throw new NotImplementedException();
        }
    }
}
```