دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۶*

آرمان سینائی
آیسا میاهی‌نیا
سید صالح اعتمادی

نیم‌سال اول ۱۴۰۲-۱۴۰۱

ایمیل/تیمز	a_sinaei@comp.iust.ac.ir aysa_mayahinia@comp.iust.ac.ir
نام شاخه	fb_A6
نام پروژه/پوشه/پول ریکوست	A6
مهلت تحویل	۱۴۰۱/۸/۱۴

*تشکر ویژه از خانم مریم سادات هاشمی که در نیم‌سال اول سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند. همچنین از اساتید حل‌تمرین نیم‌سال اول سال تحصیلی ۹۹-۹۸ سارا کدیری، محمد مهدی عبداله‌پور، مهدی مقدمی، مهسا قادران، علیرضا مرادی، پریسا یل‌سوار، غزاله محمودی و محمدجواد میرشکاری که مستند این مجموعه تمرین‌ها را بهبود بخشیدند، متشکرم.

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A6 بسازید. همچنین پروژه تست متناظر آن را ساخته و مطابق راهنمای تمرین یک فایل‌ها را در پوشه متناظر اضافه کرده و تنظیمات مربوط به کپی کردن TestData به پوشه خروجی را در تنظیمات پروژه تست قرار دهید. دقت کنید که پروژه TestCommon فقط یکبار باید در ریشه گیت موجود باشد و نباید در هر تمرین مجدد کپی شود. برای روش ارجاع به این پروژه به تمرین شماره یک مراجعه کنید.

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.

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 System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A6
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(200)]
        public void SolveTest_Q1MoneyChange()
        {
            RunTest(new Q1MoneyChange("TD1"));
        }

        [TestMethod(), Timeout(1000)]
        public void SolveTest_Q2PrimitiveCalculator()
        {
            RunTest(new Q2PrimitiveCalculator("TD2"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q3EditDistance()
        {
            RunTest(new Q3EditDistance("TD3"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q4LCSOfTwo()
        {
            RunTest(new Q4LCSOfTwo("TD4"));
        }

        [TestMethod(), Timeout(600)]
        public void SolveTest_Q5LCSOfThree()
        {
            RunTest(new Q5LCSOfThree("TD5"));
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A6", p.Process, p.TestDataName, p.Verifier);
        }

    }
}
```

۱ Money Change

در تمرین سری چهارم شما این سوال را حل کردید. همانطور که می دانید، استراتژی حریصانه برای حل این مسئله همیشه جواب درست نخواهد داد. برای مثال اگر سکه های ما ۱و ۳ و ۴ باشد و بخواهیم ۶ سنت را به وسیله ی کمترین تعداد از این سکه ها بسازیم، در این صورت با استفاده از الگوریتم حریصانه جواب $4+1+1$ و با استفاده از Dynamic Programming جواب $3+3$ خواهد بود. بنابراین یک بار دیگر مسئله ی Money Change را بااستفاده از Dynamic Programming با سکه های ۱ و ۳ و ۴ حل کنید.

ورودی نمونه	خروجی نمونه
6	2

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس MoneyChange قرار دارد، بنویسید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TestCommon;

namespace A6
{
    4 references
    public class Q1MoneyChange: Processor
    {
        private static readonly int[] COINS = new int[] {1, 3, 4};

        2 references | 1/1 passing
        public Q1MoneyChange(string testDataName) : base(testDataName) { }

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

        1 reference
        public long Solve(long n)
        {
            //Write your code here
            throw new NotImplementedException();
        }
    }
}
```

۲ Primitive Calculator

فرض کنید شما یک ماشین حساب ابتدایی دارید که تنها عمل های زیر را برای یک عدد مانند x انجام می دهد:

۱. ضرب عدد x در عدد ۲
۲. ضرب عدد x در عدد ۳
۳. جمع عدد x با عدد ۱

الگوریتمی با استفاده از Dynamic Programming بنویسید که با استفاده از ۳ عمل بالا و شروع از عدد یک، عدد مثبت و صحیح n را بدست آورید.

ورودی نمونه	خروجی نمونه
28	1 3 9 27 28

ورودی نمونه	خروجی نمونه
22	1 3 9 10 11 22

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس PrimitiveCalculator قرار دارد، بنویسید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;
namespace A6
{
    4 references
    public class Q2PrimitiveCalculator : Processor
    {
        2 references | 1/1 passing
        public Q2PrimitiveCalculator(string testDataName) : base(testDataName) { }
        11 references
        public override string Process(string inStr)
            => TestTools.Process(inStr, (Func<long, long[]>) Solve);

        1 reference
        public long[] Solve(long n)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۳ Edit Distance

فرض کنید که شما دو رشته یا string دارید که می خواهید با استفاده از سه عمل زیر string دوم را با کمترین تعداد از عملگر ها به string اول تبدیل کنید. عملگر هایی که می توانید انجام دهیدبه صورت زیر است:

۱.درج کردن یا insertion که به معنی آن است که یک حرف را در string دوم قرار دهید.
۲. پاک کردن یا deletion که به معنی آن است که یک حرف را از string دوم حذف کنید.
۳. جایگزینی یا substitution که به معنی آن است که یک حرف از string دوم را جایگزین کنید.

الگوریتمی با استفاده از Dynamic Programming بنویسید که با استفاده از ۳ عمل بالا، کمترین تعداد عمل برای تبدیل string دوم به string اول را بدست آورد.

ورودی نمونه	خروجی نمونه
short ports	3

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس EditDistance قرار دارد، بنویسید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestCommon;

namespace A6
{
    4 references
    public class Q3EditDistance : Processor
    {
        2 references | 1/1 passing
        public Q3EditDistance(string testDataName) : base(testDataName) { }
        11 references
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<string, string, long>)Solve);

        1 reference
        public long Solve(string str1, string str2)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۴ Longest Common Subsequence of Two Sequences

فرض کنید که دو Sequence داریم شما باید با Dynamic Programming طول بلندترین SubSequence مشترک این دو را پیدا کنید.

ورودی نمونه	خروجی نمونه
19 0 17 13 6 16 19 0 13 10 18 1 3	3

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس LCSOfTwo قرار دارد، بنویسید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestCommon;

namespace A6
{
    4 references
    public class Q4LCSOfTwo : Processor
    {
        2 references | ✓ 1/1 passing
        public Q4LCSOfTwo(string testDataName) : base(testDataName) { }

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

        1 reference
        public long Solve(long[] seq1, long[] seq2)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۵ Longest Common Subsequence of Three Sequences

فرض کنید که سه Sequence داریم شما باید با Dynamic Programming طول بلندترین SubSequence مشترک این سه را پیدا کنید.

ورودی نمونه	خروجی نمونه
5 4 11 8 16 2 10 2 17 6 1 8 14 4 15 3 13 16 5 2 6 17 3 14 1 15 9 0 11	1

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس LCSOfThree قرار دارد، بنویسید.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestCommon;

namespace A6
{
    4 references
    public class Q5LCSOfThree: Processor
    {
        2 references | 1/1 passing
        public Q5LCSOfThree(string testDataName) : base(testDataName) { }

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

        1 reference
        public long Solve(long[] seq1, long[] seq2, long[] seq3)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```