تمرین ۶ درس ساختمان داده

مریم سادات هاشمی
سید صالح اعتمادی

دانشگاه علم و صنعت ۹۸_۹۷

لطفا به نکات زیر توجه کنید:

- مهلت ارسال این تمرین شنبه ۱۲ آبان ماه ساعت ۱۱:۵۹ ب.ظ است.
- این تمرین شامل سوال های برنامه نویسی می باشد، بنابراین توجه کنید که حتماً موارد خواسته شده را رعایت کنید. .
- نام شاخه، پوشه و پول ریکوست همگی دقیقا "A۶" باشد.
- در صورتی که به اطلاعات بیشتری نیاز دارید می توانید با ایدی تلگرام @maryam_sadat_hashemi در ارتباط باشید.
- اگر در حل تمرین شماره ی ۵ مشکلی داشته اید،لطفا به `https://calendly.com/hashemi-maryam-sadat` مراجعه کنید و زمانی را برای رفع اشکال تنظیم کنید.

موفق باشید.

توضیحات کلی تمرین

تمرین این هفته ی شما، ۵ سوال دارد که باید به همه ی این سوال ها پاسخ دهید.
برای حل این سری از تمرین ها مراحل زیر را انجام دهید:

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A۶ بسازید.

۲.در تمرین های قبل، شما تمامی تابع ها را در فایل program.cs پیاده سازی می کردید که کد شما را طولانی می کرد. برای رفع این مشکل و راحتی شما، برای هر سوال یک کلاس درست شده است.بنابراین شما کافی است که کلاس های هر سوال را به پروژه ی 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 آن سوال در این متد پیاده سازی کنید.

متد دوم: تابع process است که مانند تمرین های قبلی در TestCommen ساخته شده است. بنابراین با خیال راحت سوال خود 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 به راحتی در کلاس مربوط به همان سوال بنویسید.

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

۱. یک UnitTest برای پروژه ی خود بسازید.
۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژه ی تست خود اضافه کنید.
۳. فایل GradedTests.cs را به پروژه ی تستی که ساخته اید اضافه کنید. توجه کنید که مانند تمرین های قبل، لازم نیست که برای هر سوال TestMethod بنویسید. تمامی آنچه که برای تست هر سوالتان نیاز دارید از قبل در این فایل برای شما پیاده سازی شده است.

```
using A6;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A6.Tests
{
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(1000)]
        [DeploymentItem("TestData","A6_TestData")]
        public void SolveTest()
        {
            Processor[] problems = new Processor[] {
                new MoneyChange("TD1"),
                new PrimitiveCalculator("TD2"),
                new EditDistance("TD3"),
                new LCSOfTwo("TD4"),
                new LCSOfThree("TD5")
            };

            foreach(var p in problems)
            {
                TestTools.RunLocalTest("A6", p.Process, p.TestDataName);
            }
        }
    }
```

دقت کنید که تغییرTestCommon یافته است. بنابراین شما باید نسخه ی جدید این آن را با دستورgit Pull دریافت کنید .

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.

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.Text;
using System.Threading.Tasks;
using TestCommon;

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

        public MoneyChange(string testDataName) : base(testDataName) { }

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

        public long Solve(long n)
        {
            //Write your code here
            return 0;
        }
    }
}
```

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.

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

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

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

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

        public long[] Solve(long n)
        {
            //Write your code here
            return new long[] { 0 };
        }
    }
}
```

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.

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 System.Threading.Tasks;
using TestCommon;

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

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

        public long Solve(string str1, string str2)
        {
            //Write your code here
            return 0;
        }

    }
}
```

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.
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 System.Threading.Tasks;
using TestCommon;

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

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

        public long Solve(long[] seq1, long[] seq2)
        {
            //Write your code here
            return 0;
        }
    }
}
```

Longest Common Subsequence of Three Sequences ۵

فرض کنید که سه Sequence داریم شما باید با Dynamic Programming طول بلندترین SubSequence مشترک این سه را پیدا کنید.
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 System.Threading.Tasks;
using TestCommon;

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

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

        public long Solve(long[] seq1, long[] seq2, long[] seq3)
        {
            //Write your code here
            return 0;
        }
    }
}
```