دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۴*

مبین داریوش همدانی
بابک بهکام
سید صالح اعتمادی

نیم‌سال اول ۱۴۰۱-۱۴۰۰

ایمیل/تیمز	m_dariushhamedani@comp.iust.ac.ir babak_behkamkia@comp.iust.ac.ir
نام شاخه	fb_A4
نام پروژه/پوشه/پول ریکوست	A4
مهلت تحویل	۹۹/۸/۱

*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 عبداللهپور، 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 A4 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 TestCommon;

namespace A4.Tests
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(200)]
        public void SolveTest_Q1ChangingMoney()
        {
            RunTest(new Q1ChangingMoney("TD1"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q2MaximizingLoot()
        {
            RunTest(new Q2MaximizingLoot("TD2"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q3MaximizingOnlineAdRevenue()
        {
            RunTest(new Q3MaximizingOnlineAdRevenue("TD3"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q4CollectingSignatures()
        {
            RunTest(new Q4CollectingSignatures("TD4"));
        }

        [TestMethod(), Timeout(500)]
        public void SolveTest_Q5MaximizeNumberOfPrizePlaces()
        {
            RunTest(new Q5MaximizeNumberOfPrizePlaces("TD5"));
        }

        [TestMethod(), Timeout(200)]
        public void SolveTest_Q6MaximizeSalary()
        {
            RunTest(new Q6MaximizeSalary("TD6"));
        }

        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q7MaxSubarraySum()
        {
            RunTest(new Q7MaxSubarraySum("TD7"));
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A4", p.Process, p.TestDataName, p.Verifier, VerifyResultWitho
                excludedTestCases: p.ExcludedTestCases);
        }
    }
```

```
}
```

۱ Money Change

در این سوال، قرار است شما یک الگوریتم حریصانه ابتدایی که توسط صندوقداران در سراسر جهان میلیون ها بار در روز استفاده می شود، را طراحی و پیاده سازی کنید.
فرض کنید سه سکه با مقدار های ۱، ۵ و ۱۰ در اختیار دارید. حداقل تعداد سکه هایی که لازم است تا مقدار پول ورودی را با سکه های مذکور بسازید؛ چقدر است؟ برای حل این سوال الگوریتم حریصانه ای را در تابع Solve کلاس Q1ChangingMoney پیاده سازی کنید. در ورودی تابع به شما مقدار پولی که میخواهید خرد کنید داده شده است.

* محدودیت زمانی : ۲۰۰ میلی ثانیه

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

namespace A4
{
    public class Q1ChangingMoney : Processor
    {
        public Q1ChangingMoney(string testDataName) : base(testDataName)
        {}

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


        public virtual long Solve(long money)
        {
            throw new NotImplementedException();
        }
    }
}
```

۲ Maximum Value of the Loot

یک دزد غنیمت هایی پیدا کرده است که از ظرفیت کیسه اش بیشتر است. به او برای پیدا کردن با ارزش ترین ترکیب از غنیمت ها کمک کنید. فرض کنید که هر کسری از یک غنیمت را می تواند در کیسه خود قرار دهد. برای مثال اگر ظرفیت کیسه ی دزد ۱۰ باشد و یک عنمیت با ارزش ۳۰۰ و ظرفیت ۳۰ وجود داشته باشد؛ دزد می تواند کسری از غنیمت که ظرفیتش ۱۰ و ارزشش ۱۰۰ می باشد را بردارد و در کیسه ی خود بگذارد. الگوریتم خود را در تابع Solve کلاس Q2MaximizingLoot به صورت حریصانه بنویسید. در ورودی تابع، ظرفیت کیسه ی دزد، وزن هر غنیمت و ارزش آن ها داده شده اند.

* محدودیت زمانی : ۲۰۰ میلی ثانیه

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

```
using System.Text;
using TestCommon;

namespace A4
{
    public class Q2MaximizingLoot : Processor
    {
        public Q2MaximizingLoot(string testDataName) : base(testDataName)
        {}

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


        public virtual long Solve(long capacity, long[] weights, long[] values)
        {
            throw new NotImplementedException();
        }


        public override Action<string, string> Verifier { get; set; } =
            TestTools.ApproximateLongVerifier;

    }
}
```

۳ Maximum Advertisement Revenue

شما n عدد آگهی برای قرار دادن در یک صفحه اینترنتی محبوب را دارید. برای هر آگهی، شما می دانید چه مقدار هزینه قرار است آگهی دهنده به ازای یک کلیک در این آگهی پرداخت کند. شما در صفحه خود n تا Slot را تنظیم کرده اید و تعداد کلیک های مورد انتظار را برای هر Slot در هر روز تخمین زده اید. شما باید الگوریتمی طراحی کنید که آگهی ها را میان Slot ها به گونه ای توزیع کند که درآمد کل به حداکثر برسد. دقت کنید که در خط اول هر TestCase تعداد Slot ها یا آگهی ها می باشد. سپس در خط های بعدی عدد اول هزینه ی یک کلیک بر روی آگهی i ام است و عدد دوم میانگین تعداد کلیک هایی در روز است که بر روی $slot_i$ ام انجام می شود. بنابراین شما هزینه ی کلیک بر روری یک آگهی را در یک آرایه مانند a_i و میانگین تعداد کلیک ها در روز را در آرایه دیگری مانند b_i بریزید. سپس این دو آرایه را به همراه تعداد Slot ها یعنی n به عنوان ورودی به تابع الگوریتم خود بدهید.

* محدودیت زمانی : ۲۰۰ میلی ثانیه

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

namespace A4
{
    public class Q3MaximizingOnlineAdRevenue : Processor
    {
        public Q3MaximizingOnlineAdRevenue(string testDataName) : base(testDataName)
```

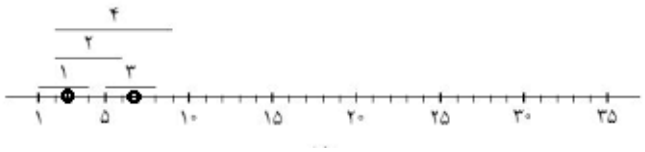
```
        {}

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


        public virtual long Solve(long slotCount, long[] adRevenue, long[] averageDailyClick)
        {
            throw new NotImplementedException();
        }
    }
}
```

۴ Collecting Signatures

شما مسئول جمع آوری امضا از همه مستاجران یک ساختمان خاص هستید. برای هر مستاجر، شما می دانیدکه در چه بازه ی زمانی در خانه است. شما باید تمام امضا ها را تا جایی که ممکن است با کمترین تعداد مراجعه به ساختمان، جمع آوری کنید. مدل ریاضی برای این مشکل به صورت شکل زیر است. شما مجموعه ای از بازه ها را در یک خط قرار داده اید و هدف شما این است که کمترین تعداد نقطه ها را بر روی خط پیدا کنید که هر بازه حداقل شامل یک نقطه باشد. دقت کنید که در خط اول هر TestCase تعداد بازه ها می باشد و در خط های بعدی، به ترتیب عدد اول شروع بازه و عدد دوم پایان بازه می باشد. بنابراین شما شروع بازه ها را در یک آرایه مانند a_i و پایان بازه ها را در آرایه دیگری مانند b_i بریزید. سپس این دو آرایه را به همراه تعداد بازه ها یعنی n به عنوان ورودی به تابع الگوریتم خود بدهید.




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

namespace A4
{
    public class Q4CollectingSignatures : Processor
    {
        public Q4CollectingSignatures(string testDataName) : base(testDataName)
        {}

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


        public virtual long Solve(long tenantCount, long[] startTimes, long[] endTimes)
```

```
        {
            throw new NotImplementedException();
        }
    }
}
```

۵ Maximum Number of Prizes

فرض کنید قرار است شما یک رقابت هیجان انگیز را برای کودکان سازماندهی کنید. یک صندوق جایزه دارید که n آب نبات در آن وجود دارد. شما باید این آبنبات ها را به بچه هایی که در مکان های یک تا k در مسابقه قرار گرفته اند، به عنوان جایزه بدهید؛ به گونه ای که رتبه های بالاتر تعداد بیشتری آبنبات بگیرند. شما باید k را به گونه ای پیدا کنید که تا جای ممکن تعداد بیشتری از بچه ها جایزه بگیرند و خوشحال شوند.در واقع مدل ریاضی این سوال به این صورت است که یک عدد صحیح مثبت n را به صورت جمعی از یک سری اعداد صحیح مثبت که دو به دو نیز متمایزهستند؛ دربیاوریم به گونه ای که تعداد این اعداد بیشترین مقدار ممکن باشد.

* محدودیت زمانی : **۵۰۰ میلی ثانیه**

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

namespace A4
{
    public class Q5MaximizeNumberOfPrizePlaces : Processor
    {
        public Q5MaximizeNumberOfPrizePlaces(string testDataName) : base(testDataName)
        {}

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


        public virtual long[] Solve(long n)
        {
            throw new NotImplementedException();
        }
    }
}
```

۶ Maximum Salary

فرض کنید برای استخدام در یک مصاحبه شرکت کرده اید و به عنوان آخرین سوال مصاحبه، رئیس شما به شما چند قطعه کاغذ با اعداد روی آن می دهد و از شما می خواهد بزرگترین شماره را از این اعداد بنویسید. پاسخ شما همان حقوق شما خواهد بود، بنابراین شما بسیار علاقه مند به حداکثر رساندن این عدد هستید. چطور می توانید این کار

را بکنید؟ در ویدئو های درس، الگوریتم زیر را برای ساختن بزرگترین عدد از شماره های تک رقمی داده شده در نظر گرفتیم.

```
LargestNumber(Digits):
answer ← empty string
while Digits is not empty:
  maxDigit ← −∞
  for digit in Digits:
    if digit ≥ maxDigit:
      maxDigit ← digit
  append maxDigit to answer
  remove maxDigit from Digits
return answer
```

متاسفانه این الگوریتم تنها در صورتی کار می کند که ورودی از اعداد تک رقمی باشد. به عنوان مثال، برای یک ورودی متشکل از دو عدد صحیح ۲۳ و ۳ (۲۳ عدد یک رقمی نیست!) الگوریتم عدد ۲۳۳ را برمی گرداند، در حالی که بزرگترین عدد در واقع ۳۲۳ است. به عبارت دیگر، استفاده از بزرگترین عدد از ورودی به عنوان شماره اول همیشه ما را به جواب درست نمی رساند. هدف شما در این مشکل این است که الگوریتم بالا را بهینه سازی کنید تا کارایی آن نه تنها برای عدد تک رقمی، بلکه برای هر عدد صحیح دلخواه مثبت باشد.

* محدودیت زمانی : **۲۰۰ میلی ثانیه**

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

namespace A4
{
    public class Q6MaximizeSalary : Processor
    {
        public Q6MaximizeSalary(string testDataName) : base(testDataName)
        {}

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


        public virtual string Solve(long n, long[] numbers)
        {
            throw new NotImplementedException();
        }
    }
}
```

۷ MaxSubarraySum (امتیازی)

برنامه ای بنویسید که بیشترین مجموع دنباله ای از خانه های مجاور در یک آرایه را پیدا کند.

نمونه خروجی:	نمونه ورودی:
12	8 -2, 5, 6, -2, -3, 1, 5, -6
7	8 -2, -3, 4, -1, -2, 1, 5, -3

توضیح:

- مثال ۱:

$$۵ + ۱ + (-۳) + (-۲) + ۶ + ۵ = ۱۲$$

- مثال ۲:

$$۵ + ۱ + (۲) - + (-۱) + ۴ = ۷$$

* محدودیت زمانی : ۲۰۰۰ میلی ثانیه

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

namespace A4
{
    public class Q7MaxSubarraySum : Processor
    {
        public Q7MaxSubarraySum(string testDataName) : base(testDataName)
        { }

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


        public virtual long Solve(long n, long[] numbers)
        {
            throw new NotImplementedException();
        }
    }
}
```