تمرین اول درس تحلیل و طراحی الگوریتم

مریم سادات هاشمی
سید صالح اعتمادی

دانشگاه علم و صنعت ـ نیمسال دوم ۹۸ـ۹۷

لطفا به نکات زیر توجه کنید:

- مهلت ارسال این تمرین شنبه ۴ اسفند 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.
- این تمرین باید در پروژه جدید در visualstudio.com به نام $AD97982$ ایجاد شود.
- کاربر sauleh@gmail.com باید در این پروژه به عنوان ادمین اضافه شده باشد.
- برای این پروژه باید Policy Build تعریف شده و جزو شرایط شاخه مستر اضافه شود.
- دانشجویانی که در درس ساختمان داده این تمرین را به طور کامل خودشان انجام دادند میتوانند تمرین قبلی را به همراه TestCommon در شاخه $A1$ در ریپازیتوری جدید اضافه کنند و پول ریکوست به همین نام درست کرده به شاخه مستر ببرند.
- دانشجویانی که ترم پیش این تمرین را خودشان انجام ندادند لازم است تمرین را به طور کامل انجام دهند. TestCommon ضمیمه شده است که باید در ریشه ریپاسیتوری در شاخه Common باز شود.
- نام شاخه، پوشه و پول ریکوست همگی دقیقا "$A1$" 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 ۱ مشکلی داشتید،لطفا به این لینک 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 $A1$ بسازید.

۲. کلاس 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 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 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.

دقت کنید که TestCommon تغییر یافته است. بنابراین شما باید نسخه ی جدید آن را با

دستور **git Pull** دریافت کنید .

در این سری از تمرین، علاوه برفایل ها با پسوند **txt** که تست کیس های سوالات بودند و شما از آن ها برای تست کدتان استفاده می کردید، فایل هایی با پسوند **webgraphviz** نیز در پوشه **TestDat** وجود دارد. شما با استفاده از این فایل ها می توانید گراف های کوچکتر از ۱۰۰ گره را به صورت تصویری در این سایت مشاهده کنید.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using A12;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A12.Tests
{
    [TestClass()]
    – references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class GradedTests
    {
        [TestMethod(), Timeout(30000)]
        [DeploymentItem("TestData", "A12_TestData")]
        – references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public void SolveTest()
        {
            Processor[] problems = new Processor[] {
                new Q1MazeExit("TD1"),
                new Q2AddExitToMaze("TD2"),
                new Q3Acyclic("TD3"),
                new Q4OrderOfCourse("TD4"),
                new Q5StronglyConnected("TD5")
            };

            foreach (var p in problems)
            {
                TestTools.RunLocalTest("A12", p.Process, p.TestDataName, p.Verifier);
            }
        }
    }
```

۱ Finding an Exit from a Maze

Maze یک شبکه مستطیل شکل از سلول ها و خانه هاست. در این سوال شما باید بررسی کنید که با شروع از یک نقطه ی خاص از maze آیا مسیری برای خروج از آن وجود دارد یا خیر. برای این منظور، می توانید maze را یک گراف غیر جهت دار در نظر بگیرید که خانه های maze رأس این گراف هستند. اگر دو خانه با هم مجاور باشند و هیچ دیواری بین آن ها نباشند، با یک یال غیر جهت دار به هم متصل می شوند. بنابراین، برای بررسی این که آیا بین دو خانه داده شده در maze مسیری وجود دارد یا خیر، کافی است، چک کنیم که بین دو رأس متناظر در گراف مسیر وجود دارد یا خیر.

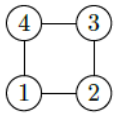
خط اول فایل ورودی، تعداد گره های گراف(یعنی n) را مشخص می کند. هر یک از n خط بعدی، شامل دو گره است که بدین معنی است که این دو گره توسط یک یال بهم متصل شدند. در خط آخر هم گره های u و v قرار دارد که الگوریتم شما باید چک کند آیا در گراف داده شده مسیری از گره ی u به گره ی v وجود دارد یا خیر. اگر مسیری وجود داشت خروجی الگوریتم یک می باشد و در غیر این صورت صفر.

ورودی:

```
4
1 2
3 2
4 3
1 4
1 4
```

خروجی:

```
1
```



همانطور که در شکل بالا معلوم است دو مسیر از گره ۱ به گره ۴ وجود دارد: ۱ـ۴ و ۱ـ۲ـ۳ـ۴.

```
using System;
using TestCommon;

namespace A12
{
    3 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class Q1MazeExit : Processor
    {
        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public Q1MazeExit(string testDataName) : base(testDataName) { }

        10 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long, long, long>)Solve);

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public long Solve(long nodeCount, long[][] edges, long StartNode, long EndNode)
        {
            // Your code here
            return 0;
        }
    }
}
```

۲ Adding Exits to a Maze

فرضیات سوال قبل را در نظر بگیرید. در این سوال شما باید مطمئن شوید که در یک maze هیچ منطقه ی مرده ای وجود ندارد. یعنی حداقل یک مسیر خروج از هر خانه وجود دارد که ما را به بیرون از Maze می رساند. برای این کار، اجزای متصل در گراف غیر جهت دار را پیدا کنید و اطمینان حاصل کنید که در هر جز متصل، یک راه خروج به بیرون از maze وجود دارد. در این سوال یک گراف با n گره به شما داده می شود و شما باید تعداد اجزای متصل این گراف را پیدا کنید.

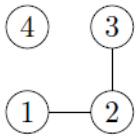
خط اول فایل ورودی، تعداد گره های گراف(یعنی n) را مشخص می کند. هر یک از n خط بعدی، شامل دو گره است که بدین معنی است که این دو گره توسط یک یال بهم متصل شدند. در فایل خروجی هم یک عدد که نشان دهنده ی تعداد اجزای متصل گراف است، می باشد.

ورودی:

```
4
1 2
3 2
```

خروجی:

```
2
```



همان طور که در شکل بالا مشخص است، تعداد اجزای متصل در این گراف دو تا می باشد.

```
using System;
using TestCommon;

namespace A12
{
    3 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class Q2AddExitToMaze : Processor
    {
        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public Q2AddExitToMaze(string testDataName) : base(testDataName) { }

        10 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long>)Solve);

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public long Solve(long nodeCount, long[][] edges)
        {
            // Your code here
            return 0;
        }
    }
}
```

۳ Checking Consistency of CS Curriculum

فرض کنید که برنامه درسی رشته علوم کامپیوتر را به شما داده اند که لیستی از درس های این رشته است که پیش نیاز هر درس را هم مشخص می کند. وظیفه ی شما این است که در این لیست چک کنید که وابستگی درس ها بهم ایجاد دور یا cycle نمی کند. بدین منظور، یک گراف جهت دار در نظر بگیرید که گره های این گراف درس ها هستند و یال ها وابستگی درس ها به یکدیگر را نشان می دهند. یعنی اگر درس u پیش نیاز درس v باشد، از راس u به راس v یک یال جهتدار در گراف متناظر وجود دارد. پس از ساخت چنین گرافی شما کافی است چک کنید که در این گراف دور وجود نداشته باشد.

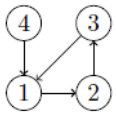
خط اول ورودی، تعداد گره های گراف را مشخص می کند و در هر یک از خط های بعدی، دو راس وجود دارد که نشان دهنده ی وجود یک یال جهتدار از راس اول به راس دوم می باشد. اگر در گراف دور وجود داشته باشد، خروجی ۱ و در غیر این صورت صفر می باشد.

ورودی:

```
4
1 2
4 1
2 3
3 1
```

خروجی:

```
1
```



همانطور که در شکل بالا معلوم است، دور ۳۱۲۳ در گراف وجود دارد. بنابراین خروجی یک خواهد بود.

```
using System;
using TestCommon;

namespace A12
{
    3 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class Q3Acyclic : Processor
    {
        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public Q3Acyclic(string testDataName) : base(testDataName) { }

        10 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long>)Solve);

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public long Solve(long nodeCount, long[][] edges)
        {
            // Your code here
            return 0;
        }
    }
}
```

۴ Determining an Order of Courses

حال که مطمئن شدید که در برنامه ی درسی علوم کامپیوتر، وابستگی های چرخه ای وجود ندارد، وظیفه ی شما این است که ترتیبی را برای اخذ درس ها، با رعایت پیش نیاز ها پیشنهاد دهید که تمامی درس ها در این ترتیب وجود داشته باشد. برای این کار کافی است همان گراف جهتدار سوال قبل را در نظر بگیرید و یکی از توپولوژی های این گراف را پیدا کنید.

خط اول ورودی، تعداد گره های گراف را مشخص می کند و در هر یک از خط های بعدی، دو راس وجود دارد که نشان دهنده ی وجود یک یال جهتدار از راس اول به راس دوم می باشد. تمامی گراف های تست کیس ها یک گراف جهت دار بدون دور هستند(DAG). در فایل خروجی هم، توپولوژی گراف داده شده، می باشد.

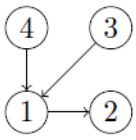
برای این سوال جواب ارائه شده فقط یک راه حل معتبر است. کد شما با متدی که داخل کلاس Q۴ پیاده سازی شده راست آزمایی می شود. لطفا این کد را تغییر ندهید

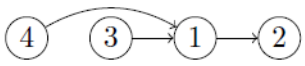
ورودی:

```
4
1 2
4 1
3 1
```

خروجی:

```
4 3 1 2
```



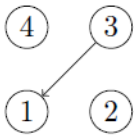


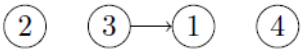
ورودی:

```
4
3 1
```

خروجی:

```
2 3 1 4
```





```
using System;
using System.IO;
using System.Linq;
using TestCommon;

namespace A12
{
    3 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class Q4OrderOfCourse: Processor
    {
        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public Q4OrderOfCourse(string testDataName) : base(testDataName) { }

        10 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long[]>)Solve);

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public long[] Solve(long nodeCount, long[][] edges)
        {
            // Your code here
            return new long[] { };
        }

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override Action<string, string> Verifier { get; set; } = TopSortVerifier;

        /// <summary>
        /// کد شما با متد زیر راست آزمایی میشود
        /// این کد نباید تغییر کند
        /// داده آزمایشی فقط یک جواب درست است
        /// تنها جواب درست نیست
        /// </summary>
```

```
        public static void TopSortVerifier(string inFileName, string strResult)
        {
            long[] topOrder = strResult.Split(TestTools.IgnoreChars)
                .Select(x => long.Parse(x)).ToArray();

            long count;
            long[][] edges;
            TestTools.ParseGraph(File.ReadAllText(inFileName), out count, out edges);

            // Build an array for looking up the position of each node in topological order
            // for example if topological order is 2 3 4 1, topOrderPositions[2] = 0,
            // because 2 is first in topological order.
            long[] topOrderPositions = new long[count];
            for (int i = 0; i < topOrder.Length; i++)
                topOrderPositions[topOrder[i] - 1] = i;
            // Top Order nodes is 1 based (not zero based).

            // Make sure all direct depedencies (edges) of the graph are met:
            //   For all directed edges u -> v, u appears before v in the list
            foreach (var edge in edges)
                if (topOrderPositions[edge[0] - 1] >= topOrderPositions[edge[1] - 1])
                    throw new InvalidDataException(
                        $"{Path.GetFileName(inFileName)}: " +
                        $"Edge dependency violoation: {edge[0]}->{edge[1]}");

        }
    }
}
```

۵ Checking Intersection in a City is Reachable

فرض کنید که اداره پلیس یک شهر همه خیابان ها را یک طرفه کرده است. شما باید بررسی کنید که آیا هنوز هم می توان به طور قانونی از هر تقاطع به هر تقاطع دیگر رفت و رانندگی کرد. بدین منظور، شما یک گراف جهت دار ایجاد کنید که رأسها تقاطع ها باشند و هر زمان که یک خیابان (یک طرفه) از u تا v در شهر وجود داشته باشد یک یال بین راس های u و v باشد. سپس، کافی است که بررسی کنید که آیا تمام رأس های گراف در یک strongly connected component هستند یا خیر.

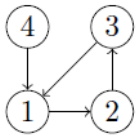
خط اول ورودی، تعداد گره های گراف را مشخص می کند و در هر یک از خط های بعدی، دو راس وجود دارد که نشان دهنده ی وجود یک یال جهتدار از راس اول به راس دوم می باشد. در فایل خروجی هم، تعداد strongly connected component در گراف داده شده، می باشد.

ورودی:

```
4
1 2
4 1
2 3
3 1
```

خروجی:

```
2
```



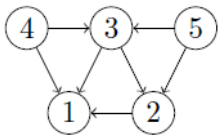
همانطور که در شکل بالا مشاهده می کنید، این گراف دو تا strongly connected component می باشد: (۱و۲و۳) ـ (۴)

ورودی:

```
5
2 1
3 2
3 1
4 3
4 1
5 2
5 3
```

خروجی:

```
5
```



همانطور که در شکل بالا مشاهده می کنید، این گراف ۵ تا strongly connected component می باشد: (۱) ـ (۲) ـ (۳) ـ (۴) ـ (۵)

```
using System;
using TestCommon;

namespace A12
{
    3 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
    public class Q5StronglyConnected: Processor
    {
        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public Q5StronglyConnected(string testDataName) : base(testDataName) { }

        10 references | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long>)Solve);

        1 reference | Sauleh Eetemadi, 5 hours ago | 1 author, 1 change
        public long Solve(long nodeCount, long[][] edges)
        {
            // Your code here
            return 0;
        }
    }
}
```