تمرین ۸ درس طراحی الگوریتم

سید علی آل یاسین
سید صالح اعتمادی
دانشگاه علم و صنعت - نیمسال دوم ۹۸-۹۷

لطفا به نکات زیر توجه کنید:

- مهلت ارسال این تمرین شنبه ۱۴ اردیبهشت 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.
- در صورتی که به اطلاعات بیشتری نیاز دارید می توانید با ایدی تلگرام $@Ali\_98\_i$ در ارتباط باشید.
- همچنین اگر در حل تمرین شماره ۸ مشکلی داشتید به آیدی بالا در تلگرام پیام بدهید.
- برای مشاهده ی بازه تغییر هر یک از متغیر ها میتوانید به مستند انگلیسی تمرین مراجعه کنید

موفق باشید.

توضیحات کلی تمرین

تمرین این هفته ی شما، ۳ 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 $A8$ 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. برای این تمرین مانند $A7$ برای هر سوال تست جداگانه با زمان جداگانه در نظر گرفته شده. بعد از حل کردن هر تمرین Assert.Inconclusive را از ابتدای تست حذف کرده و آن را اجرا کنید. چنانچه سوالی را حل نکردید از Assert.Inconclusive استفاده کنید.

```
[DeploymentItem("TestData", "A8_TestData")]
[TestClass()]
public class GradedTests
{
    [TestMethod(), Timeout(2000)]
    public void SolveTest_Q1Evaquating()
    {
        Assert.Inconclusive("A8.Q1 Not Solved");
        RunTest(new Q1Evaquating("TD1"));
    }

    [TestMethod(), Timeout(2000)]
    public void SolveTest_Q2Airlines()
    {
        Assert.Inconclusive("A8.Q2 Not Solved");
        RunTest(new Q2Airlines("TD2"));
    }

    [TestMethod(), Timeout(2000)]
    public void SolveTest_Q3Stocks()
    {
        Assert.Inconclusive("A8.Q3 Not Solved");
        RunTest(new Q3Stocks("TD3"));
    }
```

۱ Evacuating People

در این سوال شما باید مقدار Max Flow برای گرافی که در ورودی دریاف میکنید را محاسبه کنید. و خلاصه داستان سوال به این صورت است که در اثر وقوع یک تورنادو سکنه شهر اول باید به سمت پایتخت تخلیه بشوند.

در خط اول ورودی شما دو عدد n و m را دریافت میکنید که عدد اول نشان دهنده تعداد راس های گراف است و عدد دوم تعداد یال های گراف را توضیح میدهد. در هر یک از m خط بعدی شما به ترتیب سه عدد u و v و c را دریافت میکنید که هر خط نشان دهنده یک یال جهت دار است که از راس u به راس v با ظرفیت c میرود. شما باید مقدار Max Flow را از شهر ۱ به شهر n حساب کنید.

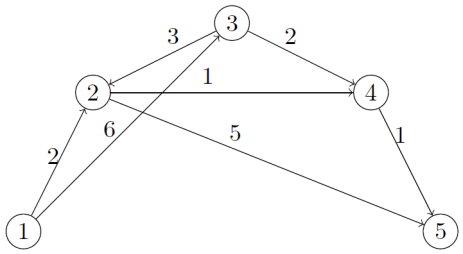
توجه کنید که امکان دارد چند یال ازراس u به v داشته باشیم و همچنین امکان دارد که هم یالی از u به v داشته باشیم و در یالی جداگانه از راس v به u را هم داشته باشیم. برای مشاهده ی محدودیت های ورودی میتوانید به منبع انگلیسی سوال مراجعه کنید.

ورودی:

```
5 7
1 2 2
2 5 5
1 3 6
3 4 2
4 5 1
3 2 3
2 4 1
```

خروجی:

```
6
```



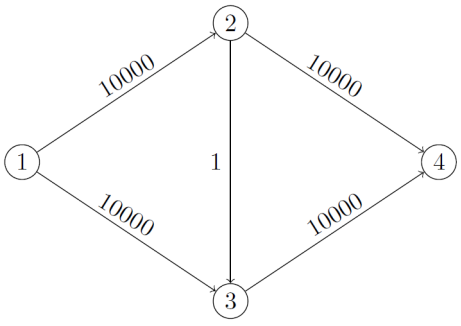
در مثال بالا جریان دو واحد از مسیر ۱-۲-۵ ، سه واحد از مسیر ۱-۳-۲-۵ و یک واحد از مسیر ۱-۳-۴-۵ میتوانیم داشته باشیم.

ورودی:

```
4 5
1 2 10000
1 3 10000
2 3 1
3 4 10000
2 4 10000
```

خروجی:

```
20000
```



در مثال بالا جریان ۱۰۰۰۰ واحد از مسیر ۱-۲-۴ و ۱۰۰۰۰ واحد از مسیر ۱-۳-۴ میتوانیم داشته باشیم.
توجه کنید که در مثال بالا اگر تنها الگوریتم Ford-Fulcerson را پیاده سازی کنید کافی نیست و لازم است آن را ارتقا بدید.

```
public class Q1Evaquating : Processor
{
    public Q1Evaquating(string testDataName) : base(testDataName)
    {
        //this.ExcludeTestCaseRangeInclusive(1, 1); // Uncomment if you need to exclude any tests
    }

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

    public virtual long Solve(long nodeCount, long edgeCount, long[][] edges)
    {
        return 0;
    }
}
```

۲ Assigning Airline Crews to Flights

در این سوال شما وظیفه دارید که maximum-matching را در گرافی دو بخشی که در ورودی به شما داده میشود پیدا کنید.

در خط اول ورودی دو عدد n ، m را دریافت میکنید که به ترتیب تعداد راس های در هر یک از بخش های گراف ما هستند. سپس در n خط بعدی در هر یک از خطوط m عدد دریافت میکنید که اگر درایه j ام خط i ام برابر یک بود نشان میدهد که راس i از بخش اول به راس j از بخش دوم یال دارد. درواقع شما ماتریس مجاورت این گراف دو بخشی را دریافت میکنید.

در تنها خط خروجی شما باید n خط چاپ شود که عدد i ام نشان میدهد که راس i از بخش اول به کدام راس match شده است. در صورتی که راس i در maximum-matching شما نبود عدد i ام خروجی شما باید ۱- باشد. توجه کنید که این سوال برای هر تست ممکن است چند جواب داشته باشد و شما کافیست که تنها یکی از جواب هارا چاپ کنید.

ورودی:

```
3 4
1 1 0 1
0 1 0 0
0 0 0 0
```

خروجی:

```
1 2 -1
```

در مثال بالا راس سوم از بخش اول به هیچ راسی از بخش دوم match نشده است

ورودی:

```
2 2
1 1
1 0
```

خروجی:

2 1

در مثال بالا یک matching کامل رخ داده است ، یعنی هیچ راسی نیست که match نشده باشد.

```
public class Q2Airlines : Processor
{
    public Q2Airlines(string testDataName) : base(testDataName)
    {
    }

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

    public virtual long[] Solve(long flightCount, long crewCount, long[][] info)
    {
        return new long[] { 0 };
    }
}
```

۳ Stocks Charts

وقتی که شما وارد این فصل (Algorithms Advanced) میشوید باید انتظار سوال های سخت رو هم داشته باشید. این سوال سخت ترین تمرین این بخش هست که در آن شما با پیاده سازی یک الگوریتم باید حداقل تعداد صفحات مختصات را پیدا کنید که به وسیله آن ها بتوان همه ی نمودار هایی که در ورودی تحویل داده میشود را نمایش داد به صورتی که هیچ دو نموداری که در یک صفحه قرار میگیرند با یکدیگر تداخل نداشته باشند.

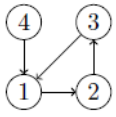
در خط اول ورودی شما به ترتیب اعداد n و k را دریافت میکنید . در n خط بعدی شما در هر خط k عدد دریافت میکنید که این اعداد نشان دهنده قیمت n سهام در k نقطه زمانی است . با اعداد هر خط یک نمودار خطی تشکیل میدهیم و میخواهیم این نمودار ها را در صفحاتی کنار هم قرار بدیم که در یک صفحه هیچ دو نموداری با یکدیگر برخورد نداشته باشند. شما باید تعداد صفحات لازم را پیدا کنید و عدد آن را در تنها خط خروجی چاپ کنید.

ورودی:

```
3 4
1 2 3 4
2 3 4 6
6 5 4 3
```

خروجی:

```
2
```



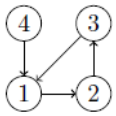
نمودار های اول و دوم با هم تداخل ندارند پس میتوانند در یک صفحه قرار بگیرند ولی نمودار سوم که با هردو نمودار قبلی برخورد دارد باید در یک صفحه جداگانه قرار بگیرد.

ورودی:

```
3 3
5 5 5
4 4 6
4 5 5
```

خروجی:

```
3
```



در این مثال برای هر نمودار احتیاج به یک صفحه جدا داریم . نمودار اول در نقطه ۲ با نمودار سوم و بین نقاط۲و ۳ با نمودار دوم برخورد دارد. نمودار دوم و سوم هم در نقطه ی اول و بین نقاط ۲ و ۳ با هم برخورد دارند.

```
public class Q3Stocks : Processor
{
    public Q3Stocks(string testDataName) : base(testDataName)
    { }

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

    public virtual long Solve(long stockCount, long pointCount, long[][] matrix)
    {
        return 0;
    }
}
```