دانشکده مهندسی کامپیوتر

تحلیل و طراحی الگوریتم

تمرین ۳*

بابک بهکام کیا
محمد خاوری
محمدجواد مهدی تبار
سید صالح اعتمادی

نیم‌سال دوم ۱۴۰۱-۱۴۰۰

ایمیل/تیمز	babak_behkamkia@comp.iust.ac.ir mohammad_khavari@comp.iust.ac.ir m_mehditabar@comp.iust.ac.ir
نام شاخه	fb_A3
نام پروژه/پوشه/پول ریکوست	A3
مهلت تحویل	۱۴۰۰/۱۲/۲۱

*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 A3 بسازید.

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.

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

۱. یک UnitTest برای پروژهی خود بسازید.

۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژهی تست خود اضافه کنید.

۳. فایل GradedTests.cs را به پروژهی تستی که ساخته اید اضافه کنید.

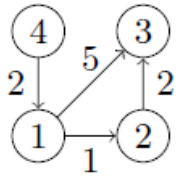
توجه:
برای اینکه تست شما از بهینه سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست ها را روی بیلد Release امتحان کنید، درغیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TestCommon;

namespace A3.Tests
{

    [DeploymentItem("TestData", "A3_TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q1MinCost()
        {
            RunTest(new Q1MinCost("TD1"));
        }

        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q2DetectingAnomalies()
        {
            RunTest(new Q2DetectingAnomalies("TD2"));
        }


        [TestMethod(), Timeout(4000)]
        public void SolveTest_Q3ExchangingMoney()
        {
            RunTest(new Q3ExchangingMoney("TD3"));
        }


        [TestMethod(), Timeout(30000)]
        public void SolveTest_Q4FriendSuggestion()
        {
            RunTest(new Q4FriendSuggestion("TD4"));
        }


        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A3", p.Process, p.TestDataName, p.Verifier,
                VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                excludedTestCases: p.ExcludedTestCases);
        }

    }
}
```

۱ محاسبه حداقل هزینه یک پرواز [۱]

دراین مسئله، ماموریت شما حداقل کردن هزینه‌ی پرواز است. برای این کار باید یک گراف جهت‌دار بسازید که در آن وزن یال بین دو گره(شهر) متناظر با هزینه پرواز بین آن دو شهر است.
اکنون در یک گراف جهت‌دار با یال‌هایی با وزن مثبت و با n راس و m یال، وزن کوتاه‌ترین مسیر بین u و v را پیدا کنید. (در واقع حداقل وزن کل یک مسیر از u به v)
خط اول فایل ورودی، تعداد راس‌های گراف(یعنی n) را مشخص می‌کند. هر یک از خطوط بعدی، شامل دو راس است که بدین معنی است که از راس اول به راس دوم یال وجود دارد. در خط آخر هم راس‌های u و v قرار دارد که الگوریتم شما باید حداقل وزن یک مسیر از u به v را پیدا کند یا اگر مسیری وجود نداشت ۱- چاپ کند.

ورودی نمونه	خروجی نمونه
4 1 2 1 4 1 2 2 3 2 1 3 5 1 3	3

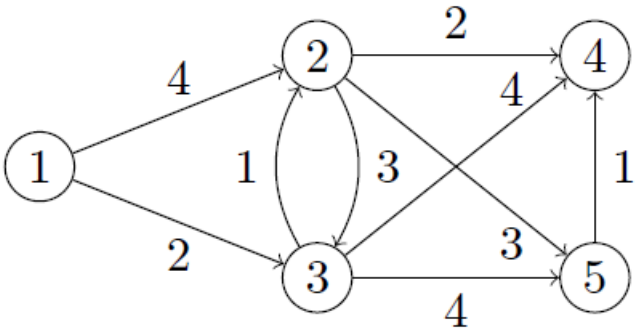


شکل ۱: گراف نمونه اول

در شکل ۱ کوتاه‌ترین مسیر از راس ۱ به راس ۳ (۱ ← ۲ ← ۳) با وزن ۳ است.

ورودی نمونه	خروجی نمونه
5 1 2 4 1 3 2 2 3 2 3 2 1 2 4 2 3 5 4 5 4 1 2 5 3 3 4 4 1 5	6

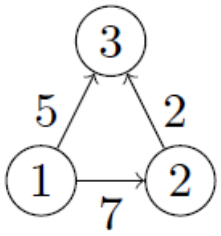
[۱] Computing the Minimum Cost of a Flight



شکل ۲: گراف نمونه دوم

در شکل ۱ دو مسیر از راس ۱ به راس ۵ با وزن ۶ وجود دارد: ۱ ← ۳ ← ۵ و ۱ ← ۳ ← ۲ ← ۵

ورودی نمونه	خروجی نمونه
3 1 2 7 1 3 5 2 3 2 3 2	-1



شکل ۳: گراف نمونه سوم

در شکل ۱ از راس ۳ به راس ۲ هیچ مسیری وجود ندارد.

```
using System;
using TestCommon;
// using Priority_Queue;

namespace A3
{
    public class Q1MinCost : Processor
    {
        public Q1MinCost(string testDataName) : base(testDataName) { }

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


        public long Solve(long nodeCount, long[][] edges, long startNode, long endNode)
        {
```

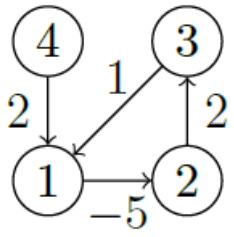
```
            throw new NotImplementedException();
        }
    }
}
```

۲ تشخیص ناهنجاری‌ها در نرخ ارز [۲]

در این سوال به شما لیستی از ارزهای c_1،c_2،....،c_n به همراه لیستی از نرخ‌های مبادله داده می‌شود به طوری که r_{ij} تعداد واحد ارز c_j است که به ازای هر واحد c_i می‌توان دریافت کرد. شما باید بررسی کنید که آیا امکان دارد با شروع از یک واحد از یک ارز مشخص، دنباله‌ای از تبدیل‌هارا انجام داد و به بیش از یک واحد از ارز اولیه رسید یا خیر. برای این منظور شما باید گرافی را بسازید که راس‌ها، همان ارزهای موجود c_1،c_2،....،c_n باشند و وزن یک یال از c_i به c_j برابر با $-log_{r_{ij}}$ باشد. سپس تنها کافی است بررسی کنید که آیا یک دور منفی در این گراف وجود دارد یا خیر.

خط اول فایل ورودی، تعداد راس‌های گراف(یعنی n) را مشخص می‌کند. هر یک از خطوط بعدی، شامل دو راس و یک وزن است که نشان می‌دهد که یالی از راس اول به راس دوم با وزن مشخص شده وجود دارد. اگر در گراف دوری با وزن منفی وجود داشت در خروجی ۱ و در غیر این صورت ۰ چاپ شود.

ورودی نمونه	خروجی نمونه
4 1 2 -5 4 1 2 2 3 2 3 1 1	1



شکل ۴: گراف نمونه اول

وزن دور ۱ ← ۲ ← ۳ در گراف ۲ برابر با ۲- است که منفی است.

```
using System;
using System.Collections.Generic;
using TestCommon;
namespace A3
{
    public class Q2DetectingAnomalies : Processor
    {
        public Q2DetectingAnomalies(string testDataName) : base(testDataName) { }
```

[۲] Detecting Anomalies in Currency Exchange Rates

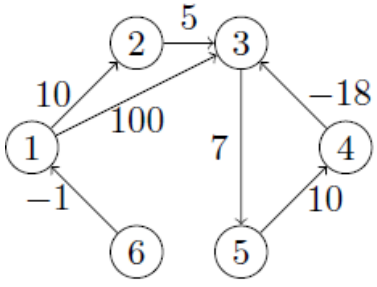
```
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long, long[][], long>)Solve);


        public long Solve(long nodeCount, long[][] edges)
        {
            throw new NotImplementedException();
        }
    }
}
```

۳ تبادل بهینه پول [۳]

حال شما باید یک راه بهینه را برای تبدیل ارز c_i به تمام ارزهای دیگر محاسبه کنید. با دریافت کردن یک گراف جهت‌دار با وزن‌هایی که ممکن است منفی باشند و n راس و m یال و همین‌طور راس s، طول کوتاه‌ترین مسیرها را از s به تمامی راس‌های گراف محاسبه کنید.

خط اول ورودی، تعداد راس‌های گراف را مشخص می‌کند. هر یک از خطوط بعدی، شامل دو راس و یک وزن است که نشان می‌دهد که یالی از راس اول به راس دوم با وزن مشخص شده وجود دارد. در پایان شماره‌ی راس s می‌آید. در خروجی به ازای هر یک از راس‌های گراف به شیوه‌ی زیر عمل کنید: "*" چاپ شود، اگر مسیری از s به u وجود ندارد. "_" چاپ شود، اگر مسیری از s به u وجود دارد، اما کوتاه‌ترین مسیر وجود ندارد (فاصله این دو راس ∞-است). در غیر این‌صورت، طول کوتاه‌ترین مسیر چاپ شود.

ورودی نمونه	خروجی نمونه
6 7 1 2 10 2 3 5 1 3 100 3 5 7 5 4 10 4 3 -18 6 1 -1 1	0 10 - - - *

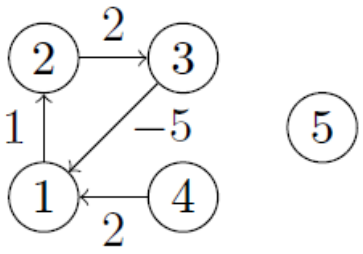


شکل ۵: عنوان شکل اول

[۳] Exchanging Money Optimally

توضیح مربوط به شکل ۳ اینجا آمده.

ورودی نمونه	خروجی نمونه
5 4 1 2 1 4 1 2 2 3 2 3 1 -5 4	- - - 0 *



شکل ۶: عنوان شکل اول

اولین خط خروجی نشان می‌دهد که فاصله‌ی ۱ تا ۱ برابر ۰ است. خط دوم نشان می‌دهد که فاصله‌ی ۱ تا ۲ برابر ۱۰ می‌باشد (مسیر متناظر ۱ -> ۲ است). سه خط بعدی نشان می‌دهند که فاصله ۱ تا راس‌های ۳،۴ و ۵ برابر با ∞- است. در واقع ابتدا می‌توان به راس ۳ توسط یال‌های ۱ -> ۲ -> ۳ رسید و سپس طول یک مسیر را به دلخواه با پیمودن متعدد دور ۳ -> ۵ -> ۴ که وزن منفی دارد، کوچک کرد. خط آخر نیز نشان می‌دهد که مسیری از ۱ به ۶ در این گراف وجود ندارد.

```
using System;
using System.Collections.Generic;
using TestCommon;
namespace A3
{
    public class Q3ExchangingMoney : Processor
    {
        public Q3ExchangingMoney(string testDataName) : base(testDataName) { }

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


        public string[] Solve(long nodeCount, long[][] edges, long startNode)
        {
            throw new NotImplementedException();
        }
    }
}
```

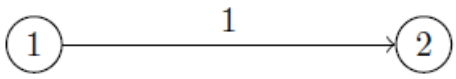
۴ پیشنهاد دوست [۴]

شبکه‌های اجتماعی در ارتباط بین افراد نقش موثری دارد. بنابراین پیشنهادات دوست یکی از مهم‌ترین ویژگی‌های شبکه‌ی اجتماعی مثل فیسبوک است. یکی از مهم‌ترین ورودی‌های الگوریتم پیشنهاد دوست، فاصله فعلی بین شما و شخص پیشنهادشده در نمودار ارتباطات دوستان است. وظیفه شما این است که الگوریتم بهینه‌ای را برای بدست آوردن این فاصله طراحی کنید.

خط اول فایل ورودی شامل دو عدد صحیح n و m به ترتیب تعداد گره‌ها و لبه‌ها در شبکه است. گره‌ها از ۱ تا n هستند. هر کدام از خطوط بعدی شامل سه عدد صحیح u ، v و l که نشان‌دهنده‌ی یک لبه به طول l از شماره گره u به شماره گره v است. (توجه داشته باشید که بعضی از شبکه‌های اجتماعی توسط گراف‌های جهتدار نشان داده می‌شوند، در حالی که برخی دیگر توسط گراف‌های غیر جهتدار نشان داده می‌شوند. به عنوان مثال، توییتر یک گراف جهتدار است (به این معنی است که u دنبال کننده v است)، در حالی که فیسبوک گراف غیر جهتدار است (به این معنی است که u و v دوست هستند).

در فایل ورودی بعد از لبه‌های گراف، خط بعدی شامل عدد صحیح q است که تعداد کوئری‌ها را مشخص می‌کند. هر یک از خطوط بعدی نشان‌دهنده‌ی کوئری‌ها هستند که شامل دو گره u و v هست که شما باید فاصله‌ی این دو را بدست آورید و اگر هیچ مسیری بین این دو گره وجود نداشت؛ عدد ۱- را برگردانید.

ورودی نمونه	خروجی نمونه
2 1 1 2 1 4 1 1 2 2 1 2 2 1	0 0 1 -1

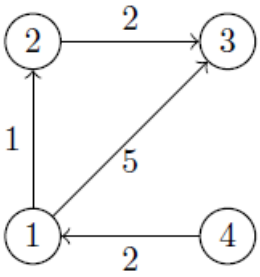


شکل ۷: گراف نمونه اول

توضیح مربوط به شکل ۴ اینجا آمده.

ورودی نمونه	خروجی نمونه
4 4 1 2 1 4 1 2 2 3 2 1 3 5 1 1 3	3

[۴] Friend Suggestion



شکل ۸: گراف نمونه دوم

توضیح مربوط به شکل ۴ اینجا آمده.

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

namespace A3
{
    public class Q4FriendSuggestion : Processor
    {
        public Q4FriendSuggestion(string testDataName) : base(testDataName) { }

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

        public long[] Solve(long nodeCount, long edgeCount,
                            long[][] edges, long queriesCount,
                            long[][] queries)
        {
            throw new NotImplementedException();
        }
    }
}
```